Chapter 3

Pursuing Low-Dimensional Distributions via Denoising

"*Information is the resolution of uncertainty.*"

—Claude Shannon, 1948

In Chapter 2, we showed how to learn simple classes of distributions whose supports are assumed to be either a single low-dimensional subspace, a mixture of such subspaces, or low-rank Gaussians. For simplicity, the different (hidden) linear or Gaussian modes are taken to be orthogonal or independent,[1] as illustrated in Figure 2.1. For these special distributions, simple and effective learning algorithms with correctness and efficiency guarantees can be derived, and the geometric and statistical interpretations of the associated algorithmic operations are clear.

In practice, both linearity and independence are idealistic assumptions that distributions of real-world high-dimensional data *rarely* satisfy. The only assumption we can make is that the intrinsic dimension of the distribution is very low compared with the dimension of the ambient space in which the data are embedded. Hence, in this chapter we show how to learn a more general class of low-dimensional distributions in a high-dimensional space that is *not* necessarily (piecewise) linear.

Real-world distributions typically contain multiple components or modes, for example, corresponding to different object classes in images. These modes may not be statistically independent and can even have different intrinsic dimensions. Therefore, we generally assume that the data are distributed on a mixture of low-dimensional nonlinear submanifolds embedded in a high-dimensional space. Figure 3.1 illustrates such a distribution. Another constraint is that in real-world situations we typically have access only to a finite number of samples

[1] Or can be reduced to such idealistic cases under benign conditions.

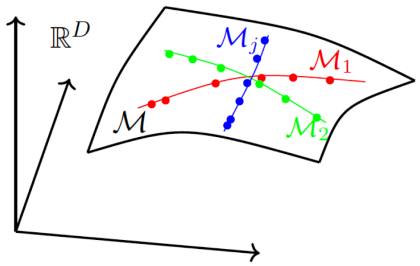

Figure 3.1: Data $\boldsymbol{X}$ are distributed on a mixture of low-dimensional submanifolds $\cup_j \mathcal{M}_j$ in a very high-dimensional ambient space, say $\mathbb{R}^D$. This is a much more general case than the special cases studied in Chapter 2, where the submanifolds were assumed to be piecewise linear and thus admitted parametric forms and analytical solutions.

from the data distribution, rather than prior knowledge of the exact distribution itself.

To learn such a distribution under these conditions, we must address three fundamental questions:

- What is a general approach to learning and representing a low-dimensional distribution in a high-dimensional space?
- How do we measure the complexity of the distribution so that we can exploit its low dimensionality effectively for learning?
- How do we make the learning process computationally tractable and scalable, especially when the ambient dimension is high and the sample size is large?

As we will see, the fundamental idea of *compression*, or *dimension reduction*, which we have already shown to be effective in the linear/independent case, remains a guiding principle for developing computational models and methods for general nonlinear low-dimensional distributions.

Due to its theoretical and practical importance, we examine in greater depth how this compression-based framework materializes when the distribution of interest can be well modeled or approximated by a mixture of low-dimensional subspaces or low-rank Gaussians. As later chapters show, these seemingly restrictive model classes serve as basic building blocks for algorithms that perform compression on general distributions.

3.1 Entropy Minimization and Compression

3.1.1 Entropy and Coding Rate

In Chapter 1 we mentioned that the goal of learning is to find the simplest way to generate a given set of data. Conceptually, Kolmogorov complexity was intended to provide such a measure of complexity, but it is not computable and is not associated with any implementable scheme that can actually reproduce the data. Hence, we need an alternative, computable, and realizable measure of complexity. This leads us to the notion of *entropy*, introduced by Shannon in 1948 [Sha48].

To illustrate the constructive nature of entropy, let us start with the simplest case. Suppose we have a discrete random variable that takes N distinct values, or *tokens*, $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ with equal probability $1/N$. We can encode each token $\boldsymbol{x}_i$ using the $\log_2 N$-bit binary representation of i. This coding scheme can be generalized to encoding arbitrary discrete distributions [CT91]: given a distribution p such that $\sum_{i=1}^N p(\boldsymbol{x}_i) = 1$, one can assign each token $\boldsymbol{x}_i$ with probability $p(\boldsymbol{x}_i)$ a binary code of length $\log_2[1/p(\boldsymbol{x}_i)] = -\log_2 p(\boldsymbol{x}_i)$ bits. Hence the average number of bits, or *the coding rate*, needed to encode any sample from the distribution $p(\cdot)$ is given by the expression[2]:

$$H(p) \doteq \mathbb{E}_{\boldsymbol{x}\sim p}[-\log p(\boldsymbol{x})] = -\sum_{i=1}^{N} p(\boldsymbol{x}_i)\log p(\boldsymbol{x}_i). \tag{3.1.1}$$

This is known as the *entropy* of the (discrete) distribution $p(\cdot)$. Note that this entropy is always nonnegative and it is zero if and only if $p(\boldsymbol{x}_i) = 1$ for some $\boldsymbol{x}_i$ with $i \in [N]$.[3] For a random variable $\boldsymbol{x}$, we will often write $H(\boldsymbol{x})$ to mean the entropy of the (marginal) distribution of $\boldsymbol{x}$.

3.1.2 Differential Entropy

When the random variable $\boldsymbol{x} \in \mathbb{R}^D$ is continuous and has a probability density p, one may view the limit of the above sum (3.1.1) as related to an integral:

$$h(p) \doteq \mathbb{E}_{\boldsymbol{x}\sim p}[-\log p(\boldsymbol{x})] = -\int_{\mathbb{R}^D} p(\boldsymbol{\xi})\log p(\boldsymbol{\xi})\mathrm{d}\boldsymbol{\xi}. \tag{3.1.2}$$

More precisely, given a continuous probability distribution p on $\mathbb{R}^D$, we may discretize $\mathbb{R}^D$ into a union of disjoint cubes C_i^ϵ of side length $\epsilon^{1/D}$, each (say) centered at $\boldsymbol{x}_i$, and define the discretized distribution p^ϵ by $p^\epsilon(\boldsymbol{x}_i) = \mathbb{P}_{\boldsymbol{x}\sim p}[\boldsymbol{x} \in C_i^\epsilon]$. Then one can show that $H(p^\epsilon) + \log\epsilon \approx h(p)$. By taking ϵ to be small, we observe that the differential entropy $h(p)$ can be negative. Interested readers may refer to [CT91] for a more detailed explanation. Similar to the discrete entropy,

[2] By the convention of information theory [CT91], the log here has base 2. Hence, entropy is measured in (binary) bits.

[3] Here we use the fact $\lim_{p\to 0} p\log p = 0$.

we will often write $h(\boldsymbol{x})$ to mean the entropy of the (marginal) distribution of $\boldsymbol{x}$.

Example 3.1 (Entropy of Gaussian distributions). Through direct calculation, one can show that the entropy of a Gaussian random variable $x \sim \mathcal{N}(\mu, \sigma^2)$ is

$$h(x) = \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2}. \tag{3.1.3}$$

The entropy of a multivariate Gaussian random variable $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ in $\mathbb{R}^D$ is

$$h(\boldsymbol{x}) = \frac{D}{2}(1 + \log(2\pi)) + \frac{1}{2}\operatorname{logdet}(\boldsymbol{\Sigma}). \tag{3.1.4}$$

■

Similar to the entropy of a discrete distribution, we would like the differential entropy to be associated with the coding rate of some realizable coding scheme. For example, as mentioned above, if we discretize the domain of the distribution with a grid of size $\epsilon > 0$ with centers $\boldsymbol{x}_i$, one possible coding scheme is to map each $\boldsymbol{x}$ to its nearest center $\boldsymbol{x}_i$, and the entropy of the resulting discrete distribution approximates the differential entropy [CT91].

There are some caveats associated with the definition of differential entropy. For a distribution in a high-dimensional space, when its support becomes degenerate (low-dimensional), its differential entropy diverges to $-\infty$. This fact is proved in Theorem B.1, but even in the simple explicit case of Gaussian distributions (3.1.4), when the covariance $\boldsymbol{\Sigma}$ is singular, we can see that $\operatorname{logdet}(\boldsymbol{\Sigma}) = -\infty$ so we have $h(\boldsymbol{x}) = -\infty$ (note that by Theorem B.1, Gaussian distributions have the largest entropy among all distributions with the same mean and covariance). In such a situation, it is not obvious how to properly quantize or encode such a distribution. Nevertheless, degenerate (Gaussian) distributions are precisely the simplest possible, and arguably the most important, instances of low-dimensional distributions in a high-dimensional space. In the next chapter, we will discuss a complete resolution to this seeming difficulty with degeneracy.

3.1.3 Minimizing Coding Rate

The learning problem entails recovering a (potentially continuous) distribution $p(\boldsymbol{x})$ from samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ drawn from it. For ease of exposition, we write $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{D \times N}$. Because the distributions of interest are (nearly) low-dimensional, we expect their (differential) entropy to be very small. However, unlike the situations studied in the previous chapter, we do not know the family of (analytical) low-dimensional models to which $p(\boldsymbol{x})$ belongs. Thus, checking whether the entropy is small seems to be the only guideline we can rely on to identify and model the distribution.

Now, given the samples alone without knowing what $p(\boldsymbol{x})$ is, in theory they could be interpreted as samples from any generic distribution. In particular, they could be interpreted as any of the following:

1. as samples from the empirical distribution $p^{\boldsymbol{X}}$ itself, which assigns probability $1/N$ to each of the N samples $\boldsymbol{x}_i, i \in [N]$;

2. as samples from a standard normal distribution $\boldsymbol{x}^n \sim p^n \doteq \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ with variance σ^2 large enough (say larger than the squared sample norms);

3. as samples from a normal distribution $\boldsymbol{x}^e \sim p^e \doteq \mathcal{N}(\boldsymbol{0}, \hat{\boldsymbol{\Sigma}})$ with covariance $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top$ being the empirical covariance of the samples;

4. as samples from a distribution $\hat{\boldsymbol{x}} \sim \hat{q}(\boldsymbol{x})$ that closely approximates the ground-truth distribution p.

The question is: which interpretation is better, and in what sense? Suppose that you believe these data $\boldsymbol{X}$ are drawn from a particular distribution $q(\boldsymbol{x})$, which may be one of the above. Then we could encode the data points with the optimal codebook for the distribution $q(\boldsymbol{x})$. Then, as the number of samples N becomes large, the required average coding length (or coding rate) is given by the so-called *cross-entropy*:

$$\mathrm{CE}(p \parallel q) \doteq \mathbb{E}_{\boldsymbol{x}\sim p}[-\log q(\boldsymbol{x})]. \tag{3.1.5}$$

If we have identified the correct distribution $p(\boldsymbol{x})$, the coding rate is given by the entropy $h(p) = \mathbb{E}_{\boldsymbol{x}\sim p}[-\log p(\boldsymbol{x})]$. It turns out that the above coding length $\mathrm{CE}(p \parallel q)$ is always greater than or equal to the entropy $h(\boldsymbol{x})$ unless $p = q$.

Hence, given a set of sampled data $\boldsymbol{X}$, to determine which distribution best characterizes the data among p^n, p^e, and $\hat{q}$, we may compare their coding rates for $\boldsymbol{X}$ and see which yields the lowest rate. We know from the above that the (theoretically achievable) coding rate for a distribution is closely related to its entropy. In general,

$$h(p^n) > h(p^e) > h(\hat{q}). \tag{3.1.6}$$

If the data $\boldsymbol{X}$ were encoded by the codebook associated with each of these distributions, the coding rate for $\boldsymbol{X}$ would therefore decrease in the same order:

$$\mathrm{CE}(p \parallel p^n) > \mathrm{CE}(p \parallel p^e) > \mathrm{CE}(p \parallel \hat{q}). \tag{3.1.7}$$

This observation gives a general guideline for pursuing a distribution $p(\boldsymbol{x})$ that has low-dimensional structure. It suggests two possible approaches:

A1. Starting with a general distribution (say a normal distribution) with high entropy, gradually transform the distribution toward the (empirical) distribution of the data by reducing entropy;

A2. Among a large family of (parametric or non-parametric) distributions with explicit coding schemes that encode the given data, progressively search for better coding schemes that give lower coding rates.

Conceptually, both approaches A1 and A2 try to do the same thing. For A1, we need to ensure such a path of transformation exists and is computable. For A2, it is necessary that the chosen family is rich enough and can closely

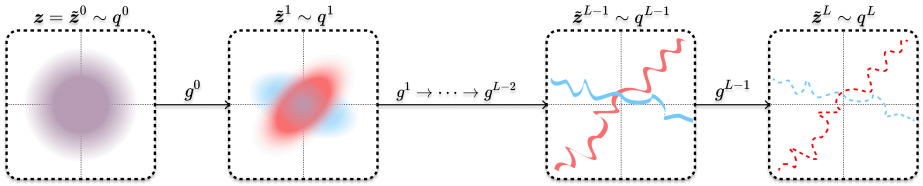

Figure 3.2: Illustration of an iterative denoising process that, starting from an isotropic Gaussian distribution, converges to an arbitrary data distribution.

approximate (or contain) the ground-truth distribution. For either approach, we need to ensure that solutions with lower entropy or better coding rates can be efficiently computed and converge to the desired distribution quickly.[4] We will explore both approaches in the remaining sections of this chapter.

3.2 Compression via Denoising

In this section we describe a *natural* and *computationally tractable* way to learn a distribution $p_{\boldsymbol{x}}$ by learning a parametric encoding that minimizes the entropy or coding rate of the representation, then using this encoding to transform high-entropy samples from a standard Gaussian into low-entropy samples from the target distribution, as illustrated in Figure 3.2. This methodology combines approaches A1 and A2 above to learn and sample from the distribution.

3.2.1 Diffusion and Denoising Processes

We first want a procedure that reduces the entropy of a very noisy random variable, producing a lower-entropy sample from the data distribution. Here we describe one potential approach, perhaps the most natural. First, we devise a way to *gradually increase* the entropy of samples already drawn from the data distribution. Next, we seek an *approximate inverse* of this process. In general, increasing entropy has no inverse, because information from the original distribution may be destroyed. We therefore restrict ourselves to a special case in which (1) the entropy-increasing operation is simple, computable, and reversible, and (2) we have a (parametric) encoding of the data distribution, as alluded to above. As we will see, these two conditions make our approach feasible.

Diffusion Processes

We will increase the entropy in arguably the simplest possible way, i.e., *adding isotropic Gaussian noise*. More precisely, given the random variable $\boldsymbol{x}$, we can consider the *stochastic process* $(\boldsymbol{x}_t)_{t\in[0,T]}$ that adds gradual noise to it, i.e.,

$$\boldsymbol{x}_t \doteq \boldsymbol{x} + t\boldsymbol{g}, \qquad \forall t \in [0, T], \tag{3.2.1}$$

[4] Say the distribution of real-world data such as images and texts.

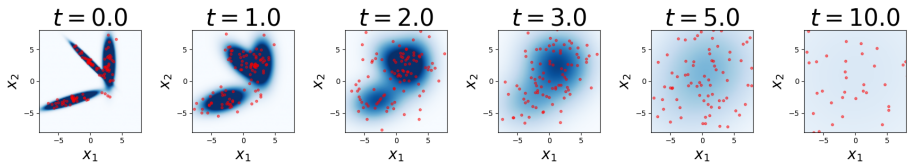

Figure 3.3: **Diffusing a mixture of Gaussians $\boldsymbol{x}$.** From left to right, we observe the evolution of the density p_t of $\boldsymbol{x}_t$ as t grows from 0 to 10, along with some representative samples (red). The plane is colored by the probability density p_t; high-density regions are colored darker blue. We observe that the probability mass becomes less concentrated as t increases, signaling that entropy increases.

where $T \in [0, \infty)$ is a time horizon and $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is drawn independently of $\boldsymbol{x}$. This process is an example of a *diffusion process*, so named because it spreads (i.e., *diffuses*) the probability mass over all of $\mathbb{R}^D$ as time goes on, increasing the entropy. This intuition is confirmed graphically by Figure 3.3, and rigorously via the following theorem.

Theorem 3.1 (Simplified version of Theorem B.2)**.** *Suppose that $(\boldsymbol{x}_t)_{t\in[0,T]}$ follows the model (3.2.1). For any $t \in (0, T]$, the random variable $\boldsymbol{x}_t$ has a probability density p_t and differential entropy $h(\boldsymbol{x}_t) > -\infty$. Moreover, under certain technical conditions on $\boldsymbol{x}$,*

$$\frac{\mathrm{d}}{\mathrm{d}t} h(\boldsymbol{x}_t) > 0, \qquad \forall t \in (0, T], \tag{3.2.2}$$

i.e., the entropy of the distribution of $\boldsymbol{x}_t$ increases over time t.

The proof is elementary, but it is rather long, so we postpone it to Section B.2.1. The main as-yet unstated implication of this result is that $h(\boldsymbol{x}_t) > h(\boldsymbol{x}_s)$ for every $t > s \geq 0$. To see this, note that if $s = 0$ and $h(\boldsymbol{x}) = h(\boldsymbol{x}_0) = -\infty$ then $h(\boldsymbol{x}_t) > -\infty$ for all $t > 0$, so the conclusion is true; otherwise $h(\boldsymbol{x}_t) = h(\boldsymbol{x}_s) + \int_s^t \left[\frac{\mathrm{d}}{\mathrm{d}u} h(\boldsymbol{x}_u)\right] \mathrm{d}u > h(\boldsymbol{x}_s)$ by the fundamental theorem of calculus, so in both cases $h(\boldsymbol{x}_t) > h(\boldsymbol{x}_s)$ for every $t > s \geq 0$.

Example 3.2 (Adding noise to a mixture of Gaussians). In this example, we consider adding noise to a mixture of K Gaussians. Choose K means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K \in \mathbb{R}^D$, K covariance matrices $\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K \in \mathsf{PSD}(D)$, and probabilities $\pi_1, \ldots, \pi_K \in [0, 1]$ such that $\sum_{k=1}^K \pi_k = 1$. Then $\boldsymbol{x}$ is sampled as follows.

- First, an index (or *label*) $y \in [K]$ is sampled s.t. $y = k$ with probability π_k.
- Second, $\boldsymbol{x}$ is sampled from the Gaussian $\mathcal{N}(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)$; i.e., if $y = k$ then $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Thus $\boldsymbol{x}$ has the Gaussian mixture distribution

$$\boldsymbol{x} \sim \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{3.2.3}$$

Gaussian mixture models are relevant because they can approximate many distributions: given any distribution for $\boldsymbol{x}$, there exists a Gaussian mixture that approximates it arbitrarily well. For this reason, Gaussian mixture models are a popular, analytically tractable surrogate for multi-modal and complex data distributions. In this book we use them extensively as a "model problem" to understand and develop almost every concept. For now, we represent the data $\boldsymbol{x}$ by a Gaussian mixture. After adding noise, the distribution of $\boldsymbol{x}_t$ is again a Gaussian mixture:

$$\boldsymbol{x}_t = \boldsymbol{x} + t\boldsymbol{g} \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k + t^2\boldsymbol{I}). \tag{3.2.4}$$

To see by example how the entropy evolves, suppose $K = 1$, so $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. By (3.1.4), the entropy of $\boldsymbol{x}_t$ is

$$h(\boldsymbol{x}_t) = \frac{D}{2}(1 + \log(2\pi)) + \frac{1}{2}\operatorname{logdet}(\boldsymbol{\Sigma} + t^2\boldsymbol{I}). \tag{3.2.5}$$

Differentiating with respect to time (see Chapter A) gives

$$\frac{\mathrm{d}}{\mathrm{d}t}h(\boldsymbol{x}_t) = t\operatorname{tr}((\boldsymbol{\Sigma} + t^2\boldsymbol{I})^{-1}) = \sum_{i=1}^{D} \frac{t}{\lambda_i(\boldsymbol{\Sigma}) + t^2} > 0, \tag{3.2.6}$$

where $\lambda_i(\boldsymbol{\Sigma})$ are the (non-negative) eigenvalues of $\boldsymbol{\Sigma}$. Because this derivative is positive, the entropy $h(\boldsymbol{x}_t)$ of the noised distribution increases over time. ■

Denoising Processes

The inverse operation to adding noise is known as *denoising*. It is a classical and well-studied topic in signal processing and system theory, instantiated (for example) as the Wiener filter and the Kalman filter (see Section 1.3.1). Several problems discussed in Chapter 2, such as PCA, ICA, and Dictionary Learning, are specific instances of the denoising problem. For a fixed t and the additive Gaussian noise model (3.2.1), the denoising problem can be formulated as attempting to learn a function $\bar{\boldsymbol{x}}^\star(t, \cdot)$ that forms the best possible approximation (in expectation) of the true random variable $\boldsymbol{x}$, given both t and $\boldsymbol{x}_t$:

$$\bar{\boldsymbol{x}}^\star(t, \cdot) \in \arg\min_{\bar{\boldsymbol{x}}(t,\cdot)} \mathbb{E}\,\|\boldsymbol{x} - \bar{\boldsymbol{x}}(t, \boldsymbol{x}_t)\|_2^2. \tag{3.2.7}$$

The solution to this problem, when optimizing $\bar{\boldsymbol{x}}(t, \cdot)$ over all possible (square-integrable) functions, is the so-called *Bayes optimal denoiser*, and turns out to be the conditional expectation:

$$\bar{\boldsymbol{x}}^\star(t, \boldsymbol{\xi}) \doteq \mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t = \boldsymbol{\xi}]. \tag{3.2.8}$$

This expression justifies the notation $\bar{\boldsymbol{x}}$, which computes a conditional expectation (i.e., conditional mean or conditional average). In short, it attempts

to remove the noise from the noisy input, outputting the best possible guess (in expectation and with respect to the ℓ^2 distance) of the denoised original random variable.

Example 3.3 (Denoising Gaussian Noise from a Mixture of Gaussians). In this example we compute the Bayes optimal denoiser for an important class of distributions, the Gaussian mixture model in Example 3.2. Let us define $\varphi(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ as the probability density of $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ evaluated at $\boldsymbol{x}$. In this notation, the density of $\boldsymbol{x}_t$ is

$$p_t(\boldsymbol{x}_t) = \sum_{k=1}^{K} \pi_k \varphi(\boldsymbol{x}_t; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k + t^2 \boldsymbol{I}). \tag{3.2.9}$$

Conditioned on y, the variables are jointly Gaussian: if we write $\boldsymbol{x} = \boldsymbol{\mu}_y + \boldsymbol{\Sigma}_y^{1/2} \boldsymbol{u}$ where $(\cdot)^{1/2}$ is the matrix square root and $\boldsymbol{u} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ independently of y (and $\boldsymbol{g}$), then we have

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x}_t \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu}_y \\ \boldsymbol{\mu}_y \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Sigma}_y^{1/2} & \boldsymbol{0} \\ \boldsymbol{\Sigma}_y^{1/2} & t\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{g} \end{bmatrix}. \tag{3.2.10}$$

This shows that $\boldsymbol{x}$ and $\boldsymbol{x}_t$ are jointly Gaussian (conditioned on y) as claimed. Thus we can write

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x}_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_y \\ \boldsymbol{\mu}_y \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_y & \boldsymbol{\Sigma}_y \\ \boldsymbol{\Sigma}_y & \boldsymbol{\Sigma}_y + t^2 \boldsymbol{I} \end{bmatrix} \right). \tag{3.2.11}$$

Thus the conditional expectation of $\boldsymbol{x}$ given $\boldsymbol{x}_t$ (i.e., the Bayes optimal denoiser conditioned on y) is famously (see also Exercise 3.2)

$$\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t, y] = \boldsymbol{\mu}_y + \boldsymbol{\Sigma}_y (\boldsymbol{\Sigma}_y + t^2 \boldsymbol{I})^{-1} (\boldsymbol{x}_t - \boldsymbol{\mu}_y). \tag{3.2.12}$$

To find the overall Bayes optimal denoiser, we use the law of iterated expectation, obtaining

$$\begin{aligned} \bar{\boldsymbol{x}}^\star(t, \boldsymbol{x}_t) &= \mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t] && (3.2.13) \\ &= \mathbb{E}[\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t, y] \mid \boldsymbol{x}_t] && (3.2.14) \\ &= \sum_{k=1}^{K} \mathbb{P}[y = k \mid \boldsymbol{x}_t] \, \mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t, y = k]. && (3.2.15) \end{aligned}$$

The probability can be dealt with as follows. Let $p_{t|y}$ be the probability density of $\boldsymbol{x}_t$ conditioned on the value of y. Then

$$\begin{aligned} \mathbb{P}[y = k \mid \boldsymbol{x}_t] &= \frac{p_{t|y}(\boldsymbol{x}_t \mid k) \pi_k}{p_t(\boldsymbol{x}_t)} && (3.2.16) \\ &= \frac{\pi_k \varphi(\boldsymbol{x}_t; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k + t^2 \boldsymbol{I})}{\sum_{i=1}^{K} \pi_i \varphi(\boldsymbol{x}_t; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i + t^2 \boldsymbol{I})}. && (3.2.17) \end{aligned}$$

On the other hand, the conditional expectation is as described before:

$$\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t, y = k] = \boldsymbol{\mu}_k + \boldsymbol{\Sigma}_k(\boldsymbol{\Sigma}_k + t^2\boldsymbol{I})^{-1}(\boldsymbol{x}_t - \boldsymbol{\mu}_k). \tag{3.2.18}$$

So putting this all together, the true Bayes optimal denoiser is

$$\bar{\boldsymbol{x}}^\star(t, \boldsymbol{x}_t) = \sum_{k=1}^{K} \left\{ \frac{\pi_k \varphi(\boldsymbol{x}_t; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k + t^2\boldsymbol{I})}{\sum_{i=1}^{K} \pi_i \varphi(\boldsymbol{x}_t; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i + t^2\boldsymbol{I})} \right. \tag{3.2.19}$$

$$\left. \cdot \left(\boldsymbol{\mu}_k + \boldsymbol{\Sigma}_k(\boldsymbol{\Sigma}_k + t^2\boldsymbol{I})^{-1}(\boldsymbol{x}_t - \boldsymbol{\mu}_k)\right) \right\}. \tag{3.2.20}$$

This example is particularly important, and several special cases will give us great conceptual insight later. For now, let us attempt to extract some geometric intuition from the functional form of the optimal denoiser (3.2.19).

To understand (3.2.19) intuitively, let us first set $K = 1$ (i.e., one Gaussian) such that $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Let us then diagonalize $\boldsymbol{\Sigma} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top$. Then the Bayes optimal denoiser is

$$\bar{\boldsymbol{x}}^\star(t, \boldsymbol{x}_t) = \boldsymbol{\mu} + \boldsymbol{\Sigma}(\boldsymbol{\Sigma} + t^2\boldsymbol{I})^{-1}(\boldsymbol{x}_t - \boldsymbol{\mu}) \tag{3.2.21}$$

$$= \boldsymbol{\mu} + \boldsymbol{V} \begin{bmatrix} \lambda_1/(\lambda_1 + t^2) & & \\ & \ddots & \\ & & \lambda_D/(\lambda_D + t^2) \end{bmatrix} \boldsymbol{V}^\top(\boldsymbol{x}_t - \boldsymbol{\mu}), \tag{3.2.22}$$

where $\lambda_1, \ldots, \lambda_D$ are the eigenvalues of $\boldsymbol{\Sigma}$. We can observe that this denoiser has three steps:

- Translate the input $\boldsymbol{x}_t$ by $\boldsymbol{\mu}$.
- Contract the (translated) input $\boldsymbol{x}_t - \boldsymbol{\mu}$ in each eigenvector direction by a quantity $\lambda_i/(\lambda_i + t^2)$. If the translated input is low-rank and some eigenvalues of $\boldsymbol{\Sigma}$ are zero, these directions get immediately contracted to 0 by the denoiser, ensuring that the output of the contraction is similarly low-rank.
- Translate the output back by $\boldsymbol{\mu}$.

This is the geometric interpretation of the denoiser of a *single* Gaussian. The overall denoiser of the Gaussian mixture model (3.2.19) uses K such denoisers, weighting their output by the posterior probabilities $\mathbb{P}[y = k \mid \boldsymbol{x}_t]$. If the means of the Gaussians are well separated, these posterior probabilities are very close to 0 or 1 near each mean or cluster. In this regime, the overall denoiser (3.2.19) has the same geometric interpretation as the above single Gaussian denoiser. ■

Intuitively, and as we can see from Example 3.3, the Bayes-optimal denoiser $\bar{\boldsymbol{x}}^\star(t, \cdot)$ should move its input $\boldsymbol{x}_t$ toward the modes of the distribution of $\boldsymbol{x}$. It turns out that we can quantify this by showing that the Bayes-optimal denoiser *takes a gradient-ascent step* on the (log-)density of $\boldsymbol{x}_t$, which (recall) we denoted

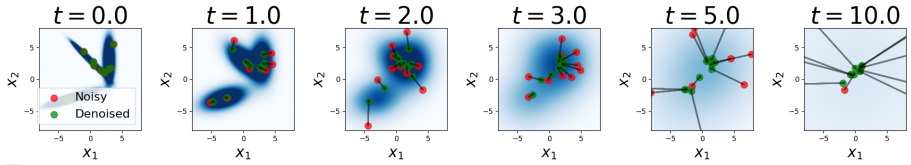

Figure 3.4: **Bayes-optimal denoiser and score of a Gaussian mixture model.** In the same setting as Figure 3.3, we demonstrate the effect of the Bayes-optimal denoiser $\bar{\boldsymbol{x}}^\star$ by plotting $\boldsymbol{x}_t$ (red) and $\bar{\boldsymbol{x}}^\star(t, \boldsymbol{x}_t)$ (green) for some choice of t and $\boldsymbol{x}_t$. By Tweedie's formula Theorem 3.2, the residual between them is proportional to the so-called (Hyvärinen) score $\nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t)$. We can see that the score points toward the modes of the distribution of $\boldsymbol{x}_t$.

p_t. That is, following the denoiser means moving from the input iterate to a region of higher probability within this (perturbed) distribution. For small t, the perturbation is small, so our initial intuition is therefore almost exactly right. The picture is visualized in Figure 3.4 and rigorously formulated as Tweedie's formula [Rob56].

Theorem 3.2 (Tweedie's Formula)**.** *Suppose that* $(\boldsymbol{x}_t)_{t\in[0,T]}$ *obeys* (3.2.1)*. Let* p_t *be the density of* $\boldsymbol{x}_t$ *(as previously declared). Then*

$$\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t] = \boldsymbol{x}_t + t^2 \nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t). \tag{3.2.23}$$

Proof. For the proof, suppose that $\boldsymbol{x}$ has a density,[5] and call this density p. Let $p_{0|t}$ and $p_{t|0}$ be the conditional densities of $\boldsymbol{x} = \boldsymbol{x}_0$ given $\boldsymbol{x}_t$ and $\boldsymbol{x}_t$ given $\boldsymbol{x}$ respectively. Let $\varphi(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ be the density of $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ evaluated at $\boldsymbol{x}$, so that $p_{t|0}(\boldsymbol{x}_t \mid \boldsymbol{x}) = \varphi(\boldsymbol{x}_t; \boldsymbol{x}, t^2\boldsymbol{I})$. Then a simple calculation gives

$$\begin{aligned}
\nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) &= \frac{\nabla_{\boldsymbol{x}_t} p_t(\boldsymbol{x}_t)}{p_t(\boldsymbol{x}_t)} && (3.2.24)\\
&= \frac{1}{p_t(\boldsymbol{x}_t)} \nabla_{\boldsymbol{x}_t} \int_{\mathbb{R}^D} p(\boldsymbol{x}) p_{t|0}(\boldsymbol{x}_t \mid \boldsymbol{x}) \mathrm{d}\boldsymbol{x} && (3.2.25)\\
&= \frac{1}{p_t(\boldsymbol{x}_t)} \nabla_{\boldsymbol{x}_t} \int_{\mathbb{R}^D} p(\boldsymbol{x}) \varphi(\boldsymbol{x}_t; \boldsymbol{x}, t^2\boldsymbol{I}) \mathrm{d}\boldsymbol{x} && (3.2.26)\\
&= \frac{1}{p_t(\boldsymbol{x}_t)} \int_{\mathbb{R}^D} p(\boldsymbol{x}) [\nabla_{\boldsymbol{x}_t} \varphi(\boldsymbol{x}_t; \boldsymbol{x}, t^2\boldsymbol{I})] \mathrm{d}\boldsymbol{x} && (3.2.27)\\
&= \frac{1}{p_t(\boldsymbol{x}_t)} \int_{\mathbb{R}^D} p(\boldsymbol{x}) \varphi(\boldsymbol{x}_t; \boldsymbol{x}, t^2\boldsymbol{I}) \left[-\frac{\boldsymbol{x}_t - \boldsymbol{x}}{t^2} \right] \mathrm{d}\boldsymbol{x} && (3.2.28)\\
&= \frac{1}{t^2 p_t(\boldsymbol{x}_t)} \int_{\mathbb{R}^D} p(\boldsymbol{x}) \varphi(\boldsymbol{x}_t; \boldsymbol{x}, t^2\boldsymbol{I}) [\boldsymbol{x} - \boldsymbol{x}_t] \mathrm{d}\boldsymbol{x} && (3.2.29)\\
&= \frac{1}{t^2 p_t(\boldsymbol{x}_t)} \int_{\mathbb{R}^D} p(\boldsymbol{x}) \varphi(\boldsymbol{x}_t; \boldsymbol{x}, t^2\boldsymbol{I}) \boldsymbol{x} \mathrm{d}\boldsymbol{x} && (3.2.30)
\end{aligned}$$

[5] Note that the theorem is true without this assumption, and it can be proven through a minimal modification of this proof to use probability measures instead of densities.

$$
\begin{aligned}
&\quad - \frac{\boldsymbol{x}_t}{t^2 p_t(\boldsymbol{x}_t)} \int_{\mathbb{R}^D} p(\boldsymbol{x}) \varphi(\boldsymbol{x}_t; \boldsymbol{x}, t^2 \boldsymbol{I}) \mathrm{d}\boldsymbol{x} \\
&= \frac{1}{t^2 p_t(\boldsymbol{x}_t)} \int_{\mathbb{R}^D} p(\boldsymbol{x}) p_{t|0}(\boldsymbol{x}_t \mid \boldsymbol{x}) \boldsymbol{x} \mathrm{d}\boldsymbol{x} - \frac{\boldsymbol{x}_t}{t^2 p_t(\boldsymbol{x}_t)} p_t(\boldsymbol{x}_t) && (3.2.31) \\
&= \frac{1}{t^2 p_t(\boldsymbol{x}_t)} \int_{\mathbb{R}^D} p_t(\boldsymbol{x}_t) p_{0|t}(\boldsymbol{x} \mid \boldsymbol{x}_t) \boldsymbol{x} \mathrm{d}\boldsymbol{x} - \frac{\boldsymbol{x}_t}{t^2 p_t(\boldsymbol{x}_t)} p_t(\boldsymbol{x}_t) && (3.2.32) \\
&= \frac{1}{t^2} \int_{\mathbb{R}^D} p_{0|t}(\boldsymbol{x} \mid \boldsymbol{x}_t) \boldsymbol{x} \mathrm{d}\boldsymbol{x} - \frac{\boldsymbol{x}_t}{t^2} && (3.2.33) \\
&= \frac{1}{t^2} \mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t] - \frac{\boldsymbol{x}_t}{t^2} && (3.2.34) \\
&= \frac{\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t] - \boldsymbol{x}_t}{t^2}. && (3.2.35)
\end{aligned}
$$

Simple rearrangement of the above equality proves the theorem. □

This result develops a connection between denoising and optimization: the Bayes-optimal denoiser takes a single step of gradient ascent on the perturbed data density p_t, and the step size adaptively becomes smaller (i.e., takes more precise steps) as the perturbation to the data distribution grows smaller. The quantity $\nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t)$ is called the *(Hyvärinen) score* and frequently appears in discussions about denoising, etc.; it first appeared in a paper of Aapo Hyvärinen in the context of ICA [Hyv05].

Similar to how one step of gradient descent is almost never sufficient to minimize an objective in practice when initializing far from the optimum, the output of the Bayes-optimal denoiser $\bar{\boldsymbol{x}}^\star(t, \cdot)$ is almost never contained in a high-probability region of the data distribution when t is large, *especially* when the data have low-dimensional structure. We illustrate this point explicitly in the following example.

Example 3.4 (Denoising a two-point mixture). Let x be uniform on the two-point set $\{-1, +1\}$ and let $(x_t)_{t \in [0,T]}$ follow (3.2.1). This is precisely a degenerate Gaussian mixture model with priors equal to $\frac{1}{2}$, means $\{-1, +1\}$, and covariances both equal to 0. For a fixed $t > 0$ we can use the calculation of the Bayes-optimal denoiser in (3.2.19) to obtain (proof as exercise)

$$
\bar{x}^\star(t, x_t) = \frac{\varphi(x_t; +1, t^2) - \varphi(x_t; -1, t^2)}{\varphi(x_t; 1, t^2) + \varphi(x_t; -1, t^2)} = \tanh\Big(\frac{x_t}{t^2}\Big). \tag{3.2.36}
$$

For t near 0, this quantity is near $\{-1, +1\}$ for almost all inputs $\bar{x}^\star(t, x_t)$. However, for t large, this quantity is not necessarily even approximately in the original support of x, which, remember, is $\{-1, +1\}$. In particular, for $x_t \approx 0$ it holds $\bar{x}^\star(t, x_t) \approx 0$, which lies completely in between the two possible points. Thus $\bar{x}^\star$ *will not output "realistic" x*. More rigorously, the distribution of $\bar{x}^\star(t, x_t)$ is very different from the distribution of x. ■

Incremental Denoising

Therefore, to denoise the very noisy sample $\boldsymbol{x}_T$ (where, recall, T is the maximum time), we cannot use the denoiser just once. Instead, we must apply it many times, analogously to gradient descent with decaying step sizes, to converge to a stationary point $\hat{\boldsymbol{x}}$. Namely, we use the denoiser to go from $\boldsymbol{x}_T$ to $\hat{\boldsymbol{x}}_{T-\delta}$, which approximates $\boldsymbol{x}_{T-\delta}$, then from $\hat{\boldsymbol{x}}_{T-\delta}$ to $\hat{\boldsymbol{x}}_{T-2\delta}$, and so on, all the way from $\hat{\boldsymbol{x}}_\delta$ to $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}_0$. Each denoising step makes the action of the denoiser more like a gradient step on the original (log-)density.

More formally, we uniformly discretize $[0, T]$ into $L+1$ timesteps $0 = t_0 < t_1 < \cdots < t_L = T$, i.e.,

$$t_\ell = \frac{\ell}{L} T, \qquad \ell \in \{0, 1, \dots, L\}. \tag{3.2.37}$$

Then for each $\ell \in [L] = \{1, 2, \dots, L\}$, going from $\ell = L$ to $\ell = 1$, we run the iteration

$$\begin{aligned}
\hat{\boldsymbol{x}}_{t_{\ell-1}} &= \mathbb{E}[\boldsymbol{x}_{t_{\ell-1}} \mid \boldsymbol{x}_{t_\ell} = \hat{\boldsymbol{x}}_{t_\ell}] && (3.2.38)\\
&= \mathbb{E}[\boldsymbol{x} + t_{\ell-1}\boldsymbol{g} \mid \boldsymbol{x}_{t_\ell} = \hat{\boldsymbol{x}}_{t_\ell}] && (3.2.39)\\
&= \mathbb{E}\left[\boldsymbol{x} + t_{\ell-1} \cdot \frac{\boldsymbol{x}_{t_\ell} - \boldsymbol{x}}{t_\ell} \;\middle|\; \boldsymbol{x}_{t_\ell} = \hat{\boldsymbol{x}}_{t_\ell}\right] && (3.2.40)\\
&= \frac{t_{\ell-1}}{t_\ell}\hat{\boldsymbol{x}}_{t_\ell} + \left(1 - \frac{t_{\ell-1}}{t_\ell}\right)\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_{t_\ell} = \hat{\boldsymbol{x}}_{t_\ell}] && (3.2.41)\\
&= \left(1 - \frac{1}{\ell}\right)\hat{\boldsymbol{x}}_{t_\ell} + \frac{1}{\ell}\bar{\boldsymbol{x}}^\star(t_\ell, \hat{\boldsymbol{x}}_{t_\ell}). && (3.2.42)
\end{aligned}$$

At the beginning of the iteration, when ℓ is large, we barely trust the denoiser's output and mostly keep the current iterate. This makes sense, as the denoiser can have huge variance (cf. Example 3.4). When ℓ is small, the denoiser "locks on" to the modes of the data distribution, since a denoising step is essentially a gradient step on the true distribution's log-density, and we can trust it not to produce unreasonable samples; thus the denoising step is dominated by the denoiser's output. At $\ell = 1$ we even discard the current iterate and keep only the denoiser's output.

We visualize the convergence process in $\mathbb{R}^3$ in Figure 3.5. While we will develop some rigorous results about convergence later, we now give a closed-form example of the iterative denoising process applied to denoising a Gaussian data distribution.

Example 3.5 (Iterative denoising for a single Gaussian). To understand the above iterative denoising iteration intuitively, we apply it to denoise $\boldsymbol{x}_T$ defined in (3.2.1), where $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Using (3.2.42), we compute

$$\begin{aligned}
\hat{\boldsymbol{x}}_{t_{\ell-1}} - \boldsymbol{\mu} &= \left(1 - \frac{1}{\ell}\right)\hat{\boldsymbol{x}}_{t_\ell} + \frac{1}{\ell}\bar{\boldsymbol{x}}^\star(t_\ell, \hat{\boldsymbol{x}}_{t_\ell}) - \boldsymbol{\mu} && (3.2.43)\\
&= \left(1 - \frac{1}{\ell}\right)(\hat{\boldsymbol{x}}_{t_\ell} - \boldsymbol{\mu}) + \frac{1}{\ell}(\bar{\boldsymbol{x}}^\star(t_\ell, \hat{\boldsymbol{x}}_{t_\ell}) - \boldsymbol{\mu}). && (3.2.44)
\end{aligned}$$

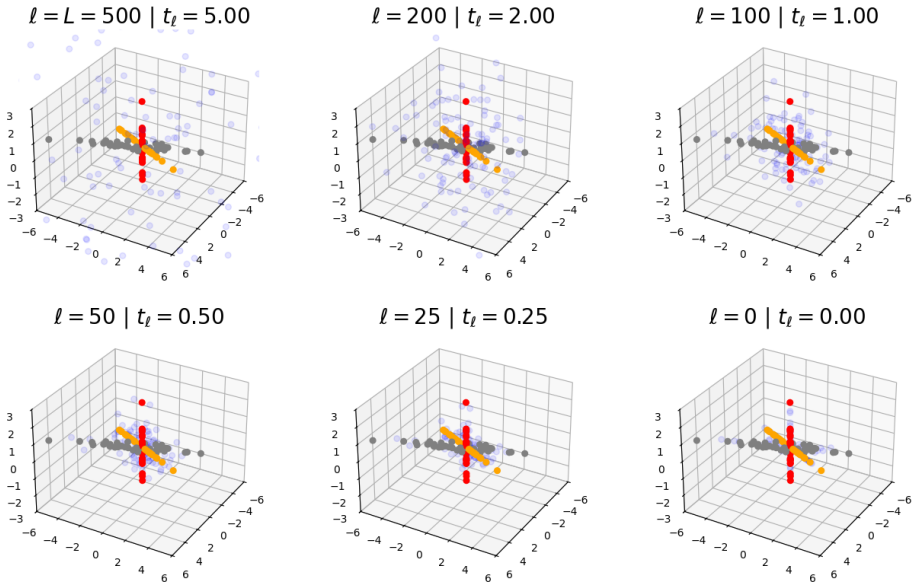

Figure 3.5: **Denoising a low-rank mixture of Gaussians.** Each figure shows samples from the true data distribution (gray, orange, red) and samples undergoing the denoising process (3.2.82) (light blue). At top left, the process has just started and the noise is very large. As the process continues, the noise is pushed further toward the support of the low-rank data distribution. Finally, in the bottom right, the generated samples are perfectly aligned with the support of the data and closely resemble samples drawn from the low-rank Gaussian mixture model.

Then, plugging in the form for $\bar{\boldsymbol{x}}^\star$ that we obtain from (3.2.22) and performing some tedious matrix algebra, we obtain

$$\boldsymbol{x}_{t_{\ell-1}} - \boldsymbol{\mu} = \boldsymbol{V} \begin{bmatrix} 1 - \frac{\ell T^2}{\lambda_1 L^2 + \ell^2 T^2} & & \\ & \ddots & \\ & & 1 - \frac{\ell T^2}{\lambda_D L^2 + \ell^2 T^2} \end{bmatrix} \boldsymbol{V}^\top (\boldsymbol{x}_{t_\ell} - \boldsymbol{\mu}). \quad (3.2.45)$$

Notice that all entries in this diagonal matrix are in $(0, 1)$. Taking the ℓ^2-norm and using a cheap yet tight upper bound, it holds

$$\|\boldsymbol{x}_{t_{\ell-1}} - \boldsymbol{\mu}\|_2 \leq \underbrace{\max_{i \in [D]} \left\{ 1 - \frac{\ell T^2}{\lambda_i L^2 + \ell^2 T^2} \right\}}_{c_{\ell, L}} \cdot \|\boldsymbol{x}_{t_\ell} - \boldsymbol{\mu}\|_2. \quad (3.2.46)$$

At first glance, this inequality seems to imply that the denoising iteration is *very close*, conceptually, to a *contraction mapping* such as power iteration (2.1.17) sending the iterate $\boldsymbol{x}_t$ to the mean $\boldsymbol{\mu}$, since $c_{\ell,L}$ are all in $(0, 1)$ for any ℓ and L.

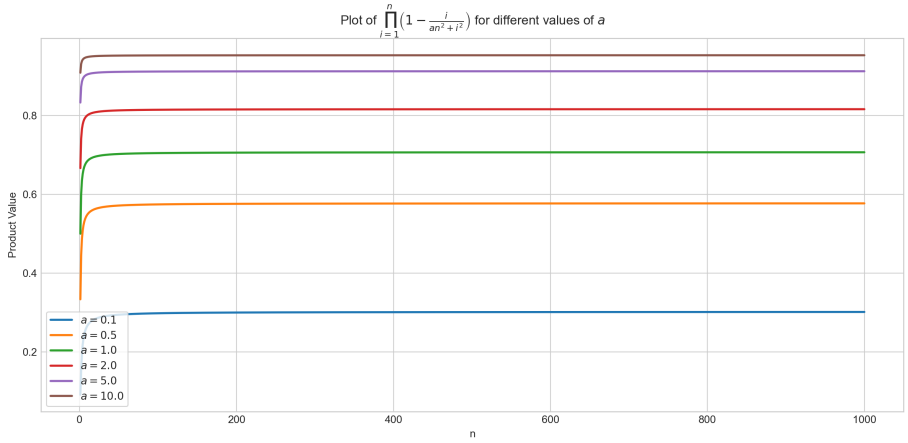

Figure 3.6: **The denoising iteration from Example 3.5 is not a contraction mapping.** Plot of the contraction coefficient products $\prod_{\ell=1}^{L} c_{\ell,L}$ in the case where the data are in $D = 1$, the time horizon $T = 1$, and the data variance $\lambda_1 = a$, for varying values of a. If the denoising iteration were (conceptually similar to) a contraction mapping, these products would go to 0, but instead they converge to non-zero values.

However, this is false. To see why, observe that by iterating (3.2.46) it holds

$$\|\boldsymbol{x}_0 - \boldsymbol{\mu}\|_2 \leq \prod_{\ell=1}^{L} c_{\ell,L} \cdot \|\boldsymbol{x}_T - \boldsymbol{\mu}\|_2. \tag{3.2.47}$$

If each denoising update had similar properties to a contraction mapping, then $\prod_{\ell=1}^{L} c_{\ell,L}$ would go to 0 as $L \to \infty$, and the iterate would converge to $\boldsymbol{\mu}$ extremely fast, rendering the sampler completely useless or vacuous. But in reality the denoising iteration is *actually not a contraction mapping* since $c_{\ell,L}$ changes with both the indices ℓ and L, so that $\prod_{\ell=1}^{L} c_{\ell,L}$ does not go to 0 as $L \to \infty$, as demonstrated in Figure 3.6 (it is also possible to compute the limit analytically). In reality, the sampler ensures that the terminal iterates spread out amongst the support of the data distribution, enabling actual sampling from the data distribution, which is quite unlike the objective of power iteration (for example). ■

Recall that we wanted to build the iterative denoising process to reduce the entropy of the iterates. While we did this in a roundabout way by inverting a process that adds entropy, it is now time to pay the piper and confirm that our iterative denoising process reduces the entropy.

Theorem 3.3 (Simplified Version of Theorem B.3)**.** *Suppose that* $(\boldsymbol{x}_t)_{t\in[0,T]}$ *obeys* (3.2.1)*. Then, under certain technical conditions on* $\boldsymbol{x}$*, for every* $s < t$ *with* $s, t \in (0, T]$*,*

$$h(\mathbb{E}[\boldsymbol{x}_s \mid \boldsymbol{x}_t]) < h(\boldsymbol{x}_t). \tag{3.2.48}$$

The full statement of the theorem and the proof itself require some technicality, so they are postponed to Section B.2.2.

Computing the Denoiser

The last thing we discuss here is that many times, we will *not be able to compute* $\bar{\boldsymbol{x}}^\star(t,\cdot)$ for any t, since we do not have the distribution p_t. But we can try to *learn one from data*. Recall that the denoiser $\bar{\boldsymbol{x}}^\star$ is defined in (3.2.7) as minimizing the mean-squared error $\mathbb{E}\,\|\bar{\boldsymbol{x}}(t,\boldsymbol{x}_t)-\boldsymbol{x}\|_2^2$. We can use this mean-squared error as a loss or objective function to learn the denoiser. For example, we can parameterize $\bar{\boldsymbol{x}}(t,\cdot)$ by a neural network, writing it as $\bar{\boldsymbol{x}}_\theta(t,\cdot)$, and optimize the loss over the parameter space Θ:

$$\min_{\theta\in\Theta}\mathbb{E}\,\|\bar{\boldsymbol{x}}_\theta(t,\boldsymbol{x}_t)-\boldsymbol{x}\|_2^2. \tag{3.2.49}$$

The solution to this optimization problem, implemented via gradient descent or a similar algorithm, will give us a $\bar{\boldsymbol{x}}_{\theta^\star}(t,\cdot)$ which is a good approximation to $\bar{\boldsymbol{x}}^\star(t,\cdot)$ (at least if the training works) and which we will use as our denoiser.

What is a good architecture for this neural network $\bar{\boldsymbol{x}}_{\theta^\star}(t,\cdot)$? To answer this question, we will first attempt to understand the case of data $\boldsymbol{x}$ that are *low-rank*, i.e., having low-rank support. The most canonical way to represent low-rank data is through a *low-rank Gaussian*, and so we start our exploration by assuming that $\boldsymbol{x}$ is drawn (approximately) from a low-rank Gaussian:

$$\boldsymbol{x}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{U}\boldsymbol{U}^\top), \tag{3.2.50}$$

where $\boldsymbol{U}\in\mathsf{O}(D,P)$ is a tall orthogonal matrix with $P\le D$ (using zero-mean for essentially notational convenience). The optimal denoiser under (3.2.50) is (from Example 3.3):

$$\bar{\boldsymbol{x}}^\star(t,\boldsymbol{x}_t)=\boldsymbol{U}\boldsymbol{U}^\top(\boldsymbol{U}\boldsymbol{U}^\top+t^2\boldsymbol{I})^{-1}\boldsymbol{x}_t. \tag{3.2.51}$$

The matrix $\boldsymbol{U}\boldsymbol{U}^\top+t^2\boldsymbol{I}$ is a low-rank perturbation of the full-rank matrix $t^2\boldsymbol{I}$, and thus ripe for simplification via the *Sherman-Morrison-Woodbury identity*, i.e., for matrices $\boldsymbol{A},\boldsymbol{C},\boldsymbol{U},\boldsymbol{V}$ such that $\boldsymbol{A}$ and $\boldsymbol{C}$ are invertible,

$$(\boldsymbol{A}+\boldsymbol{U}\boldsymbol{C}\boldsymbol{V})^{-1}=\boldsymbol{A}^{-1}-\boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{C}^{-1}+\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1}. \tag{3.2.52}$$

We prove this identity in Exercise 3.3. For now we apply this identity with $\boldsymbol{A}=t^2\boldsymbol{I}$, $\boldsymbol{U}=\boldsymbol{U}$, $\boldsymbol{V}=\boldsymbol{U}^\top$, and $\boldsymbol{C}=\boldsymbol{I}$, obtaining

$$(\boldsymbol{U}\boldsymbol{U}^\top+t^2\boldsymbol{I})^{-1}=\frac{1}{t^2}\boldsymbol{I}-\frac{1}{t^4}\boldsymbol{U}\left(\boldsymbol{I}+\frac{1}{t^2}\boldsymbol{U}^\top\boldsymbol{U}\right)^{-1}\boldsymbol{U}^\top \tag{3.2.53}$$

$$=\frac{1}{t^2}\boldsymbol{I}-\frac{1}{t^4\left(1+\frac{1}{t^2}\right)}\boldsymbol{U}\boldsymbol{U}^\top \tag{3.2.54}$$

$$=\frac{1}{t^2}\left(\boldsymbol{I}-\frac{1}{1+t^2}\boldsymbol{U}\boldsymbol{U}^\top\right). \tag{3.2.55}$$

This implies

$$\boldsymbol{U}\boldsymbol{U}^\top(\boldsymbol{U}\boldsymbol{U}^\top + t^2\boldsymbol{I})^{-1} = \frac{1}{t^2}\boldsymbol{U}\boldsymbol{U}^\top\left(\boldsymbol{I} - \frac{1}{1+t^2}\boldsymbol{U}\boldsymbol{U}^\top\right) \tag{3.2.56}$$

$$= \frac{1}{t^2}\left(1 - \frac{1}{1+t^2}\right)\boldsymbol{U}\boldsymbol{U}^\top \tag{3.2.57}$$

$$= \frac{1}{1+t^2}\boldsymbol{U}\boldsymbol{U}^\top. \tag{3.2.58}$$

Our optimal denoiser is a (rescaled) linear projection onto the low-rank subspace supporting our data distribution. This is essentially the same thing as probabilistic PCA from Chapter 2 (cf. Section 2.1.4). We can formalize this intuition by considering a class of subspace-parameterized denoisers:

$$\bar{\boldsymbol{x}}_{\boldsymbol{V}}(t, \boldsymbol{x}_t) = \frac{1}{1+t^2}\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{x}_t, \tag{3.2.59}$$

and showing that the denoising problem $\min_{\boldsymbol{V}\in\mathsf{O}(D,P)} \mathbb{E}\,\|\bar{\boldsymbol{x}}_{\boldsymbol{V}}(t,\boldsymbol{x}_t) - \boldsymbol{x}\|_2^2$ obtains the same solution as probabilistic PCA (Section 2.1.4). Indeed, plugging our subspace denoiser into the training loss (3.2.7), we obtain

$$\min_{\boldsymbol{V}\in\mathsf{O}(D,P)} \mathbb{E}\,\|\bar{\boldsymbol{x}}_{\boldsymbol{V}}(t,\boldsymbol{x}_t) - \boldsymbol{x}\|_2^2 = \mathbb{E}\left\|\frac{1}{1+t^2}\boldsymbol{V}\boldsymbol{V}^\top(\boldsymbol{x} + t\boldsymbol{g}) - \boldsymbol{x}\right\|_2^2, \tag{3.2.60}$$

where the equality is due to (3.2.1). Conditioned on $\boldsymbol{x}$ (thus integrating over $\boldsymbol{g}$), we compute

$$\mathbb{E}_{\boldsymbol{g}}\left\|\frac{1}{1+t^2}\boldsymbol{V}\boldsymbol{V}^\top(\boldsymbol{x} + t\boldsymbol{g}) - \boldsymbol{x}\right\|_2^2 \tag{3.2.61}$$

$$= \left\|\frac{1}{1+t^2}\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{x} - \boldsymbol{x}\right\|_2^2 - \frac{t}{1+t^2}\,\mathbb{E}_{\boldsymbol{g}}\left\langle \boldsymbol{x} - \frac{1}{1+t^2}\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{x}, \boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{g}\right\rangle \tag{3.2.62}$$

$$+ \frac{t^2}{(1+t^2)^2}\,\mathbb{E}_{\boldsymbol{g}}\left\|\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{g}\right\|_2^2 \tag{3.2.63}$$

$$= \left\|\frac{1}{1+t^2}\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{x} - \boldsymbol{x}\right\|_2^2 + \frac{t^2 P}{(1+t^2)^2} \tag{3.2.64}$$

where the second equality follows from $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and the fact that $\boldsymbol{V} \in \mathsf{O}(D,P)$ so that $\mathbb{E}_{\boldsymbol{g}}\,\|\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{g}\|_2^2 = \mathbb{E}_{\boldsymbol{g}}\,\|\boldsymbol{V}^\top\boldsymbol{g}\|_2^2 = P$. Therefore, the denoising problem is equivalent to

$$\min_{\boldsymbol{V}\in\mathsf{O}(D,P)}\left\{\mathbb{E}\left\|\frac{1}{1+t^2}\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{x} - \boldsymbol{x}\right\|_2^2\right. \tag{3.2.65}$$

$$\left. = \mathbb{E}\,\|\boldsymbol{x}\|_2^2 + \left(\frac{1}{(1+t^2)^2} - \frac{2}{1+t^2}\right)\mathbb{E}\,\|\boldsymbol{V}^\top\boldsymbol{x}\|_2^2\right\}, \tag{3.2.66}$$

which in turn is equivalent to the problem

$$\max_{\boldsymbol{V}\in\mathsf{O}(D,P)} \mathbb{E}\,\|\boldsymbol{V}^\top\boldsymbol{x}\|_2^2 \tag{3.2.67}$$

which was shown in Section 2.1.1 to be equivalent to the problem

$$\min_{\boldsymbol{V}\in\mathsf{O}(D,P)} \mathbb{E}\,\|\boldsymbol{x}-\boldsymbol{V}\boldsymbol{V}^\top\boldsymbol{x}\|_2^2, \tag{3.2.68}$$

i.e., the probabilistic PCA formulation (2.1.30).

Thus, if we were to assume our data were drawn from a low-rank Gaussian, our denoising algorithm would essentially consist of many PCA iterations in a row applied to progressively less-noisy data. While this method sounds very simple, it can actually produce complex and relatively powerful denoising behaviors when applied to practical data, as demonstrated by PCANet [CJG+15]. Nevertheless, to obtain a more general denoising operator that may work on complex and multi-modal data distributions, we will consider a broader class of data distributions: the ubiquitous class of *low-rank Gaussian mixture models*, whose denoiser we computed in Example 3.2 and Example 3.3. In the context of denoising, since Gaussian mixture models can approximate a very broad class of distributions arbitrarily well, in practice optimizing among the class of denoisers for Gaussian mixture models can potentially give us something close to the optimal denoiser for the real data distribution.

Formally, we now assume that $\boldsymbol{x}$ is drawn from a low-rank Gaussian mixture model $\frac{1}{K}\sum_{k=1}^K \mathcal{N}(\boldsymbol{0},\boldsymbol{U}_k\boldsymbol{U}_k^\top)$, where $\boldsymbol{U}_k\in\mathsf{O}(D,P)$ are tall orthogonal matrices whose columns span the P-dimensional subspaces that support the K low-rank Gaussian components. We write

$$\boldsymbol{x}\sim\frac{1}{K}\sum_{k=1}^K\mathcal{N}(\boldsymbol{0},\boldsymbol{U}_k\boldsymbol{U}_k^\top). \tag{3.2.69}$$

Then the optimal denoiser under (3.2.1) is (from Example 3.3)

$$\bar{\boldsymbol{x}}^\star(t,\boldsymbol{x}_t)=\sum_{k=1}^K\left\{\frac{\varphi(\boldsymbol{x}_t;\boldsymbol{0},\boldsymbol{U}_k\boldsymbol{U}_k^\top+t^2\boldsymbol{I})}{\sum_{i=1}^K\varphi(\boldsymbol{x}_t;\boldsymbol{0},\boldsymbol{U}_i\boldsymbol{U}_i^\top+t^2\boldsymbol{I})}\right. \tag{3.2.70}$$

$$\left.\cdot\left(\boldsymbol{U}_k\boldsymbol{U}_k^\top(\boldsymbol{U}_k\boldsymbol{U}_k^\top+t^2\boldsymbol{I})^{-1}\boldsymbol{x}_t\right)\right\}. \tag{3.2.71}$$

Applying the same Sherman-Morrison-Woodbury identity as in (3.2.55), we obtain

$$(\boldsymbol{U}_k\boldsymbol{U}_k^\top+t^2\boldsymbol{I})^{-1}=\frac{1}{t^2}\left(\boldsymbol{I}-\frac{1}{1+t^2}\boldsymbol{U}_k\boldsymbol{U}_k^\top\right). \tag{3.2.72}$$

We can then compute the posterior probabilities as follows. Since the $\boldsymbol{U}_k$ are orthogonal, $\det(\boldsymbol{U}_k\boldsymbol{U}_k^\top+t^2\boldsymbol{I})$ is the same for every k. Thus

$$\frac{\varphi(\boldsymbol{x}_t;\boldsymbol{0},\boldsymbol{U}_k\boldsymbol{U}_k^\top+t^2\boldsymbol{I})}{\sum_{i=1}^K\varphi(\boldsymbol{x}_t;\boldsymbol{0},\boldsymbol{U}_i\boldsymbol{U}_i^\top+t^2\boldsymbol{I})} \tag{3.2.73}$$

$$= \frac{\exp\left(-\frac{1}{2}\boldsymbol{x}_t^\top (\boldsymbol{U}_k\boldsymbol{U}_k^\top + t^2\boldsymbol{I})^{-1}\boldsymbol{x}_t\right)}{\sum_{i=1}^K \exp\left(-\frac{1}{2}\boldsymbol{x}_t^\top (\boldsymbol{U}_i\boldsymbol{U}_i^\top + t^2\boldsymbol{I})^{-1}\boldsymbol{x}_t\right)} \tag{3.2.74}$$

$$= \frac{\exp\left(-\frac{1}{2t^2}\boldsymbol{x}_t^\top \left(\boldsymbol{I} - \frac{1}{1+t^2}\boldsymbol{U}_k\boldsymbol{U}_k^\top\right)\boldsymbol{x}_t\right)}{\sum_{i=1}^K \exp\left(-\frac{1}{2t^2}\boldsymbol{x}_t^\top \left(\boldsymbol{I} - \frac{1}{1+t^2}\boldsymbol{U}_i\boldsymbol{U}_i^\top\right)\boldsymbol{x}_t\right)} \tag{3.2.75}$$

$$= \frac{\exp\left(-\frac{1}{2t^2}\|\boldsymbol{x}_t\|_2^2 + \frac{1}{2t^2(1+t^2)}\|\boldsymbol{U}_k^\top\boldsymbol{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(-\frac{1}{2t^2}\|\boldsymbol{x}_t\|_2^2 + \frac{1}{2t^2(1+t^2)}\|\boldsymbol{U}_i^\top\boldsymbol{x}_t\|_2^2\right)} \tag{3.2.76}$$

$$= \frac{\exp\left(-\frac{1}{2t^2}\|\boldsymbol{x}_t\|_2^2\right)\exp\left(\frac{1}{2t^2(1+t^2)}\|\boldsymbol{U}_k^\top\boldsymbol{x}_t\|_2^2\right)}{\exp\left(-\frac{1}{2t^2}\|\boldsymbol{x}_t\|_2^2\right)\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)}\|\boldsymbol{U}_i^\top\boldsymbol{x}_t\|_2^2\right)} \tag{3.2.77}$$

$$= \frac{\exp\left(\frac{1}{2t^2(1+t^2)}\|\boldsymbol{U}_k^\top\boldsymbol{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)}\|\boldsymbol{U}_i^\top\boldsymbol{x}_t\|_2^2\right)}. \tag{3.2.78}$$

This is a softmax operation weighted by the projection of $\boldsymbol{x}_t$ onto each subspace measured by $\|\boldsymbol{U}_i^\top\boldsymbol{x}_t\|_2$ (tempered by the temperature $2t^2(1+t^2)$). Meanwhile, the component denoisers can be written in the same way as (3.2.58) to obtain

$$\boldsymbol{U}_k\boldsymbol{U}_k^\top(\boldsymbol{U}_k\boldsymbol{U}_k^\top + t^2\boldsymbol{I})^{-1}\boldsymbol{x}_t = \frac{1}{1+t^2}\boldsymbol{U}_k\boldsymbol{U}_k^\top\boldsymbol{x}_t. \tag{3.2.79}$$

Putting these together, we have

$$\bar{\boldsymbol{x}}^\star(t,\boldsymbol{x}_t) = \frac{1}{1+t^2}\sum_{k=1}^K \frac{\exp\left(\frac{1}{2t^2(1+t^2)}\|\boldsymbol{U}_k^\top\boldsymbol{x}_t\|_2^2\right)}{\sum_{i=1}^K \exp\left(\frac{1}{2t^2(1+t^2)}\|\boldsymbol{U}_i^\top\boldsymbol{x}_t\|_2^2\right)}\boldsymbol{U}_k\boldsymbol{U}_k^\top\boldsymbol{x}_t, \tag{3.2.80}$$

i.e., a projection of $\boldsymbol{x}_t$ onto each of the K subspaces, weighted by a softmax of a quadratic function of $\boldsymbol{x}_t$. This functional form is similar to an *attention mechanism* in a transformer architecture! As we will see in Chapter 5, this is no coincidence at all; the deep link between denoising and lossy compression (to be covered in Section 4.1) makes transformer denoisers so effective in practice. Thus, our Gaussian mixture model theory motivates the use of transformer-like neural networks for denoising.

Overall, the learned denoiser architecture corresponds to an (implicit parametric) encoding scheme of the given data, since it can be used to denoise/project onto the data distribution. Training a denoiser is equivalent to finding a better coding scheme, and this partially implements approach A2 (i.e., pursuing a distribution by searching for schemes with better coding rates) at the end of Section 3.1.3. In the sequel, we discuss how to implement the other approach.

3.2.2 Sampling a Distribution via Iterative Denoising

Remember that at the end of Section 3.1.3, we discussed two approaches for pursuing a distribution with low-dimensional structure. The first approach (A1)

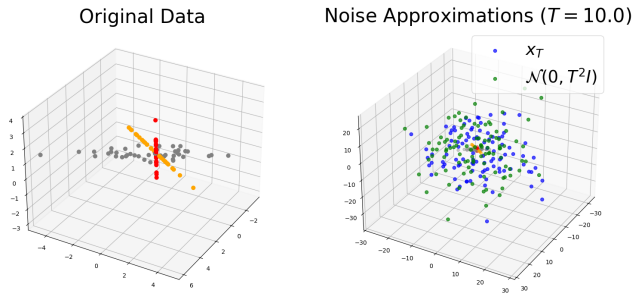

Figure 3.7: **Visualizing $\boldsymbol{x}_T$ versus $\mathcal{N}(\mathbf{0}, T^2\boldsymbol{I})$.** *Left:* A plot of Gaussian mixture model data $\boldsymbol{x}$. *Right:* A plot of $\boldsymbol{x}$ as well as $\boldsymbol{x}_T$ and an independent sample of $\mathcal{N}(\mathbf{0}, T^2\boldsymbol{I})$, for $T = 10$. On the right plot, $\boldsymbol{x}$ is plotted in the same colors as the left; however, samples from $\boldsymbol{x}_T$ and $\mathcal{N}(\mathbf{0}, T^2\boldsymbol{I})$ are both much larger, on average, than samples from $\boldsymbol{x}$, and so it appears much smaller because of the scaling. Despite this large blow-up, we clearly observe the similarities in the distributions of $\boldsymbol{x}_T$ and $\mathcal{N}(\mathbf{0}, T^2\boldsymbol{I})$.

starts with a normal distribution of high entropy and gradually reduces its entropy until it reaches the data distribution. We call this procedure *sampling*, since we generate new samples. It is now time to discuss how to implement this with the toolkit we have built.

We know how to denoise very noisy samples $\boldsymbol{x}_T$ to obtain approximations $\hat{\boldsymbol{x}}$ whose distributions resemble that of the target random variable $\boldsymbol{x}$. Yet the sampling procedure requires us to start from a template distribution that has *no* influence from the distribution of $\boldsymbol{x}$ and to use the denoiser to guide the iterates toward the distribution of $\boldsymbol{x}$. How can we do this? One way is motivated as follows:

$$\frac{\boldsymbol{x}_T}{T} = \frac{\boldsymbol{x} + T\boldsymbol{g}}{T} = \frac{\boldsymbol{x}}{T} + \boldsymbol{g} \xrightarrow{T\to+\infty} \boldsymbol{g} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{3.2.81}$$

Thus, $\boldsymbol{x}_T \approx \mathcal{N}(\mathbf{0}, T^2\boldsymbol{I})$. This approximation is quite good for almost all practical distributions, and is visualized in Figure 3.7.

So, discretizing $[0, T]$ into $0 = t_0 < t_1 < \cdots < t_L = T$ uniformly using $t_\ell = T\ell/L$ (as in the previous section), one possible way to sample from pure noise is:

- Sample $\hat{\boldsymbol{x}}_T \sim \mathcal{N}(\mathbf{0}, T^2\boldsymbol{I})$ (independently of everything else).
- Run the denoising iteration as in Section 3.2.1, i.e.,

$$\hat{\boldsymbol{x}}_{t_{\ell-1}} = \left(1 - \frac{1}{\ell}\right)\hat{\boldsymbol{x}}_{t_\ell} + \frac{1}{\ell}\bar{\boldsymbol{x}}^\star(t_\ell, \hat{\boldsymbol{x}}_{t_\ell}). \tag{3.2.82}$$

- Output $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}_0$.

This is conceptually all there is behind *diffusion models*, which transform noise into data samples in accordance with the first desideratum. However, a few

steps remain before we obtain models that can actually sample from real data distributions like images under practical resource constraints. In the sequel, we introduce and motivate several such steps.

Step 1: Different discretizations. The first step is motivated by the observation that *we do not need to spend so many denoising iterations* at large t. Figure 3.5 shows that the first 200 or 300 iterations out of the 500 iterations of the sampling process are spent contracting the noise toward the data distribution as a whole, before the remaining iterations push the samples toward a subspace. Given a fixed iteration count L, this suggests that we should allocate more timesteps t_ℓ near $t = 0$ than near $t = T$. During sampling (and training), we can therefore use an alternative discretization of $[0, T]$ into $0 \leq t_0 < t_1 < \cdots < t_L \leq T$, such as an *exponential discretization*:

$$t_\ell = C_1(e^{C_2 \ell} - 1), \qquad \forall \ell \in \{0, 1, \ldots, L\} \tag{3.2.83}$$

where $C_1, C_2 > 0$ are constants that can be tuned for optimal performance in practice; theoretical analyses often specify such optimal constants as well. The denoising/sampling iteration then becomes

$$\hat{\boldsymbol{x}}_{t_{\ell-1}} \doteq \frac{t_{\ell-1}}{t_\ell} \hat{\boldsymbol{x}}_{t_\ell} + \left(1 - \frac{t_{\ell-1}}{t_\ell}\right) \bar{\boldsymbol{x}}^\star(t_\ell, \hat{\boldsymbol{x}}_{t_\ell}), \tag{3.2.84}$$

with $\hat{\boldsymbol{x}}_{t_L} \sim \mathcal{N}(\boldsymbol{0}, t_L^2 \boldsymbol{I})$.

Step 2: Different noise models. The second step is to consider slightly different models compared to (3.2.1). The basic motivation for this is as follows. In practice, the noise distribution $\mathcal{N}(\boldsymbol{0}, t_L^2 \boldsymbol{I})$ becomes an increasingly poor estimate of the true covariance in high dimensions, i.e., (3.2.81) becomes an increasingly poor approximation, especially with anisotropic high-dimensional data. The increased distance between $\mathcal{N}(\boldsymbol{0}, t_L^2 \boldsymbol{I})$ and the true distribution of $\boldsymbol{x}_{t_L}$ may cause the denoiser to perform worse in such circumstances. Theoretically, $\boldsymbol{x}_{t_L}$ never converges to any distribution as t_L increases, so this setup is difficult to analyze end-to-end. In this case, our remedy is to *simultaneously add noise and shrink the contribution of* $\boldsymbol{x}$*, such that* $\boldsymbol{x}_T$ *converges as* $T \to \infty$. The rate of added noise is denoted $\sigma \colon [0, T] \to \mathbb{R}_{\geq 0}$, and the rate of shrinkage is denoted $\alpha \colon [0, T] \to \mathbb{R}_{\geq 0}$, such that σ is *increasing* and α is (not strictly) *decreasing*, and

$$\boldsymbol{x}_t \doteq \alpha_t \boldsymbol{x} + \sigma_t \boldsymbol{g}, \qquad \forall t \in [0, T]. \tag{3.2.85}$$

The previous setup has $\alpha_t = 1$ and $\sigma_t = t$, and this is called the *variance-exploding (VE) process*. A popular choice that decreases the contribution of $\boldsymbol{x}$, as we described originally, has $T = 1$ (so that $t \in [0, 1]$), $\alpha_t = \sqrt{1 - t^2}$ and $\sigma_t = t$; this is the *variance-preserving (VP) process*. Note that under the VP process, $\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ exactly, so we can just sample from this standard

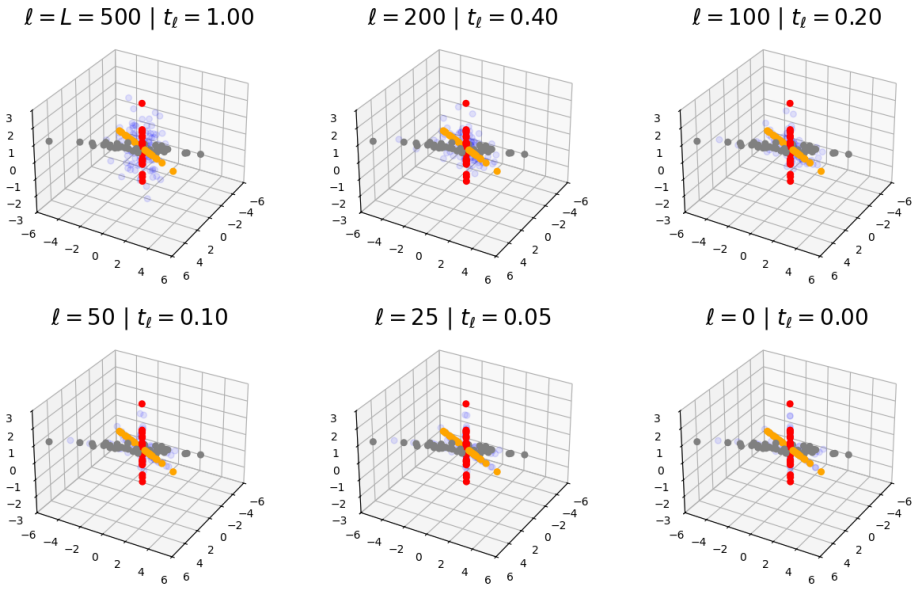

Figure 3.8: **Denoising a mixture of Gaussians using the VP diffusion process.** We use the same figure setup and data distribution as Figure 3.5. Note that compared to Figure 3.5, the noise distribution is much more concentrated around the origin.

distribution and iteratively denoise. As a result, the VP process is much easier to analyze theoretically and more stable empirically.[6]

With this more general setup, Tweedie's formula (3.2.23) becomes

$$\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t] = \frac{1}{\alpha_t}\left(\boldsymbol{x}_t + \sigma_t^2 \nabla \log p_t(\boldsymbol{x}_t)\right). \tag{3.2.86}$$

The denoising iteration (3.2.84) becomes

$$\hat{\boldsymbol{x}}_{t_{\ell-1}} = \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \hat{\boldsymbol{x}}_{t_\ell} + \left(\alpha_{t_{\ell-1}} - \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \alpha_{t_\ell}\right) \bar{\boldsymbol{x}}^\star(t_\ell, \hat{\boldsymbol{x}}_{t_\ell}). \tag{3.2.87}$$

Example 3.6 (Gaussian mixture model). For a Gaussian mixture model (see Example 3.2), its denoiser (3.2.19) becomes

$$\bar{\boldsymbol{x}}^\star(t, \boldsymbol{x}_t) = \sum_{k=1}^{K} \left\{ \frac{\pi_k \varphi(\boldsymbol{x}_t; \alpha_t \boldsymbol{\mu}_k, \alpha_t^2 \boldsymbol{\Sigma}_k + \sigma_t^2 \boldsymbol{I})}{\sum_{i=1}^{K} \pi_i \varphi(\boldsymbol{x}_t; \alpha_t \boldsymbol{\mu}_i, \alpha_t^2 \boldsymbol{\Sigma}_i + \sigma_t^2 \boldsymbol{I})} \right. \tag{3.2.88}$$
$$\left. \cdot \left(\boldsymbol{\mu}_k + \alpha_t \boldsymbol{\Sigma}_k (\alpha_t^2 \boldsymbol{\Sigma}_k + \sigma_t^2 \boldsymbol{I})^{-1} (\boldsymbol{x}_t - \alpha_t \boldsymbol{\mu}_k)\right) \right\}.$$

[6] Why use the whole α, σ setup? As we will see in Exercise 3.5, it encapsulates and unifies many proposed processes, including the recently popular so-called *flow matching* process. Despite this, the VE and VP processes are still the most popular empirically and theoretically (so far), and so we will consider them in this Section.

Figure 3.8 demonstrates iterations of the sampling procedure. ■

Note that the denoising iteration (3.2.87) gives a sampling algorithm called the DDIM ("Denoising Diffusion Implicit Model") sampler [SME20], and is one of the most popular sampling algorithms used in diffusion models today. We summarize it in Algorithm 3.1.

Algorithm 3.1 Sampling using a denoiser.

Input: An ordered list of timesteps $0 \le t_0 < \cdots < t_L \le T$ to use for sampling.
Input: A denoiser $\bar{\boldsymbol{x}}\colon \{t_\ell\}_{\ell=1}^L \times \mathbb{R}^D \to \mathbb{R}^D$.
Input: Scale and noise level functions $\alpha, \sigma\colon \{t_\ell\}_{\ell=0}^L \to \mathbb{R}_{\ge 0}$.
Output: A sample $\hat{\boldsymbol{x}}$, approximately from the distribution of $\boldsymbol{x}$.

function DDIMSAMPLER($\bar{\boldsymbol{x}}, (t_\ell)_{\ell=0}^L$)
  Initialize $\hat{\boldsymbol{x}}_{t_L} \sim$ approximate distribution of $\boldsymbol{x}_{t_L}$
  `# VP` $\implies \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, `VE` $\implies \mathcal{N}(\mathbf{0}, t_L^2 \boldsymbol{I})$.
  for $\ell = L, L-1, \ldots, 1$ **do**
    Compute

$$\hat{\boldsymbol{x}}_{t_{\ell-1}} \doteq \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \hat{\boldsymbol{x}}_{t_\ell} + \left(\alpha_{t_{\ell-1}} - \frac{\sigma_{t_{\ell-1}}}{\sigma_{t_\ell}} \alpha_{t_\ell} \right) \bar{\boldsymbol{x}}(t_\ell, \hat{\boldsymbol{x}}_{t_\ell})$$

  end for
  return $\hat{\boldsymbol{x}}_{t_0}$
end function

Step 3: Optimizing training pipelines. If we use the procedure dictated by Section 3.2.1 to learn a separate denoiser $\bar{\boldsymbol{x}}(t, \cdot)$ for each time t to be used in the sampling algorithm, *we would have to learn L separate denoisers!* This is highly inefficient—the usual case is that we have to train L separate neural networks, taking up L times the training time and storage memory, and then be locked into using these timesteps for sampling forever. Instead, we can *train a single neural network* to denoise across all times t, taking as input the continuous variables $\boldsymbol{x}_t$ and t (instead of just $\boldsymbol{x}_t$ before). Mechanically, our training loss averages over t, i.e., solves the following problem:

$$\min_\theta \mathbb{E} \left\| \bar{\boldsymbol{x}}_\theta(\tau, \boldsymbol{x}_\tau) - \boldsymbol{x} \right\|_2^2, \tag{3.2.89}$$

where $\tau \sim \mathcal{U}([0, T])$, i.e., is drawn uniformly from $[0, T]$. Similar to Step 1, where we used more timesteps closer to $t = 0$ to ensure a better sampling process, we may want to ensure that the denoiser is higher quality closer to $t = 0$, and thereby *weight the loss* so that t near 0 has higher weight.[7] Letting w_t be the

[7] In practice, the distribution of sampled t may also be non-uniform, in order to ensure (in conjunction with the weighting scheme) that the Monte Carlo estimate for the loss has small variance, greatly improving training stability.

weight at time t, the weighted loss would look like

$$\min_\theta \mathbb{E}[w_\tau \|\bar{\boldsymbol{x}}_\theta(\tau, \boldsymbol{x}_\tau) - \boldsymbol{x}\|_2^2]. \tag{3.2.90}$$

One reasonable choice of weight in practice is $w_t = \alpha_t^2/\sigma_t^2$. The precise reason will be covered in the next paragraph, but generally it serves to up-weight the losses corresponding to t near 0 while still remaining reasonably numerically stable. Also, of course, we cannot compute the expectation in practice, so we use the most straightforward Monte Carlo average to estimate it. The series of changes made here have several conceptual and computational benefits: we do not need to train multiple denoisers, we can train on one set of timesteps and sample using a subset (or others entirely), etc. The full pipeline is discussed in Algorithm 3.2. Certainly, this pipeline works in practice; we discuss some results in Sections 8.6 to 8.10.

Step 4: Changing the estimation target. It is common to instead reorient the whole denoising pipeline around *noise predictors*, i.e., estimates of $\mathbb{E}[\boldsymbol{g} \mid \boldsymbol{x}_t]$. In practice, noise predictors are slightly easier to train because their output is (almost) always of comparable size to a Gaussian random variable, so training is more numerically stable, but they may require larger models since they predict a full-dimensional target instead of potentially low-dimensional data [LH25]. By (3.2.85) we have

$$\boldsymbol{x}_t = \alpha_t \mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t] + \sigma_t \mathbb{E}[\boldsymbol{g} \mid \boldsymbol{x}_t] \implies \mathbb{E}[\boldsymbol{g} \mid \boldsymbol{x}_t] = \frac{1}{\sigma_t}\left(\boldsymbol{x}_t - \alpha_t \mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t]\right), \tag{3.2.91}$$

so any predictor for $\boldsymbol{x}$ can be turned into a predictor for $\boldsymbol{g}$ using the above relation, i.e.,

$$\bar{\boldsymbol{g}}(t, \boldsymbol{x}_t) = \frac{1}{\sigma_t}\boldsymbol{x}_t - \frac{\alpha_t}{\sigma_t}\bar{\boldsymbol{x}}(t, \boldsymbol{x}_t), \tag{3.2.92}$$

and vice-versa. Thus a good network for estimating $\bar{\boldsymbol{g}}$ is the same as a good network for estimating $\bar{\boldsymbol{x}}$ *plus a residual connection* (as seen in, e.g., transformers). Their losses are also the same as the denoiser, up to the factor of α_t/σ_t, i.e.,

$$\mathbb{E}[w_\tau \|\boldsymbol{g} - \bar{\boldsymbol{g}}(\tau, \boldsymbol{x}_\tau)\|_2^2] = \mathbb{E}\left[w_\tau \frac{\alpha_\tau^2}{\sigma_\tau^2}\|\boldsymbol{x} - \bar{\boldsymbol{x}}(\tau, \boldsymbol{x}_\tau)\|_2^2\right], \tag{3.2.93}$$

recalling that $\tau \sim \mathcal{U}([0, T])$. Other targets have been proposed for different tasks, e.g., $\mathbb{E}[\boldsymbol{v}_t \mid \boldsymbol{x}_t]$ where $\boldsymbol{v}_t = \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}_t$ (called v-prediction or velocity prediction), etc. This turns out to be equivalent to denoising and noise prediction, but also fundamentally useful for generalizations of diffusion such as flow matching, which we will discuss in Remark 3.3.

Sampling Error Rate of the Denoising Process

We have made many changes to our original platonic noising/denoising process. To assure ourselves that the new process still works in practice, we can compute

Algorithm 3.2 Learning a denoiser from data.

Input: Dataset $\mathcal{D} \subseteq \mathbb{R}^D$.
Input: An ordered list of timesteps $0 \leq t_0 < \cdots < t_L \leq T$ to use for sampling.
Input: A weighting function $w\colon \{t_\ell\}_{\ell=1}^L \to \mathbb{R}_{\geq 0}$.
Input: Scale and noise level functions $\alpha, \sigma\colon \{t_\ell\}_{\ell=0}^L \to \mathbb{R}_{\geq 0}$.
Input: A parameter space Θ and a denoiser architecture $\bar{\boldsymbol{x}}_\theta$.
Input: An optimization algorithm for the parameters.
Input: The number of optimization iterations M.
Input: The number of Monte Carlo draws N per training step.
Output: A trained denoiser $\bar{\boldsymbol{x}}_{\theta^\star}$.

function TrainDenoiser($\mathcal{D}, \Theta$)
  Initialize $\theta^{(1)} \in \Theta$
  for $m \in [M]$ **do**
    for $i \in [N]$ **do**
      `# Draw a sample from the dataset.`
      $\boldsymbol{x}^{(m,i)} \sim \mathcal{D}$
      `# Sample a timestep.`
      $t^{(m,i)} \overset{\text{i.i.d.}}{\sim} \mathcal{U}(\{t_\ell\}_{\ell=1}^L)$
      `# Compute weights and scales.`
      $(\alpha^{(m,i)}, \sigma^{(m,i)}, w^{(m,i)}) = (\alpha_{t^{(m,i)}}, \sigma_{t^{(m,i)}}, w_{t^{(m,i)}})$
      `# Sample a noise vector.`
      $\boldsymbol{g}^{(m,i)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
      `# Compute the noised sample.`
      $\boldsymbol{x}_t^{(m,i)} \doteq \alpha^{(m,i)} \boldsymbol{x}^{(m,i)} + \sigma^{(m,i)} \boldsymbol{g}^{(m,i)}$
    end for
    `# Compute the training loss.`
    $\hat{\mathcal{L}}^{(m)} \doteq \dfrac{1}{N} \displaystyle\sum_{i=1}^{N} w^{(m,i)} \| \boldsymbol{x}^{(m,i)} - \bar{\boldsymbol{x}}_{\theta^{(m)}}(t^{(m,i)}, \boldsymbol{x}_t^{(m,i)}) \|_2^2$
    `# Update the model parameters.`
    $\theta^{(m+1)} \doteq \texttt{OptimUpdate}^{(m)}(\theta^{(m)}, \nabla_{\theta^{(m)}} \hat{\mathcal{L}}^{(m)})$
  end for
  `# Return the trained denoiser.`
  return $\bar{\boldsymbol{x}}_{\theta^{(M+1)}}$
end function

numerical examples (such as Figure 3.8). To assure ourselves that it is theoretically sound, we can prove a *bound on the error rate* for the sampling algorithm, which shows that the error rate is small. We now furnish such a rate from the literature, which shows that the output distribution of the sampler converges in the so-called *total variation (TV) distance* to the true distribution. The TV distance between two random variables $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as:[8]

$$\mathsf{TV}(\boldsymbol{x}, \boldsymbol{y}) \doteq \sup_{A \subseteq \mathbb{R}^d} \left| \mathbb{P}[\boldsymbol{x} \in A] - \mathbb{P}[\boldsymbol{y} \in A] \right|. \tag{3.2.94}$$

If $\boldsymbol{x}$ and $\boldsymbol{y}$ are very close (uniformly), then the supremum will be small. So the TV distance measures the closeness of random variables. (It is indeed a metric, as the name suggests; the proof is an exercise.)

Theorem 3.4 ([LY24] Theorem 1, Simplified)**.** *Suppose that* $\mathbb{E}\,\|\boldsymbol{x}\|_2 < \infty$*. If* $\boldsymbol{x}$ *is denoised according to the VP process with an exponential discretization*[9] *as in* (3.2.83)*, the output* $\hat{\boldsymbol{x}}$ *of Algorithm 3.1 satisfies the total variation bound*

$$\mathsf{TV}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \tilde{\mathcal{O}}\Bigg(\underbrace{\frac{D}{L}}_{\text{discretization error}} + \underbrace{\sqrt{\frac{1}{L}\sum_{\ell=1}^{L} \frac{\alpha_{t_\ell}^2}{\sigma_{t_\ell}^4}\, \mathbb{E}\,\|\bar{\boldsymbol{x}}^\star(t_\ell, \boldsymbol{x}_{t_\ell}) - \bar{\boldsymbol{x}}(t_\ell, \boldsymbol{x}_{t_\ell})\|_2^2}}_{\text{average excess error of the denoiser}} \Bigg) \tag{3.2.95}$$

where $\bar{\boldsymbol{x}}^\star$ *is the Bayes optimal denoiser for* $\boldsymbol{x}$*, and* $\tilde{\mathcal{O}}$ *is a version of the big-*$\mathcal{O}$ *notation that ignores logarithmic factors in* L*.*

The very high-level proof technique is, as discussed earlier, to bound the error at each step, distinguish the error sources (between discretization and denoiser error), and carefully ensure that the errors do not accumulate too much (or even cancel out).

Note that if $L \to \infty$ and we correctly learn the Bayes-optimal denoiser $\bar{\boldsymbol{x}} = \bar{\boldsymbol{x}}^\star$ (so the excess error is 0), then the sampling process in Algorithm 3.1 yields a *perfect (in distribution) inverse* of the noising process, since the error rate in Theorem 3.4 goes to 0,[10] as argued heuristically above.

Remark 3.1. What if the data is low-dimensional, say supported on a *low-rank subspace* of the high-dimensional space $\mathbb{R}^D$? If the data distribution is compactly supported—say if the data is normalized to the unit hypercube, which is often ensured as a preprocessing step for real data such as images—it is possible to

[8] The supremum here is only over *measurable* sets A, not absolutely all sets. As with all other measure-theoretic caveats in this book, feel free to ignore it for practical purposes.

[9] The precise definition is rather lengthy in our notation and only defined up to various absolute constants, so we omit it here for brevity. It is in the original paper [LY24].

[10] Similar results hold for VE processes; the theoretical toolkits used for the proofs differ because of the structure of the underlying stochastic processes.

do better. Namely, the authors of [LY24] also define a measure of *approximate intrinsic dimension* κ using the asymptotics of the so-called covering number, which is extremely similar in intuition (if not in implementation) to the rate-distortion function presented in the next Chapter. Then they show that using a particular small modification of the DDIM sampler in Algorithm 3.1 (i.e., slightly perturbing the update coefficients), the discretization error becomes $\tilde{\mathcal{O}}(\kappa/L)$ instead of $\tilde{\mathcal{O}}(D/L)$ as in Theorem 3.4. Therefore, using this modified algorithm, L does not have to be too large even as D reaches the thousands or millions, since real data have low-dimensional structure. However, in practice we use the DDIM sampler instead, so L should have a mild dependence on D to achieve consistent error rates. The exact choice of L trades off between the computational complexity (e.g., runtime or memory consumption) of sampling and the statistical complexity of learning a denoiser for low-dimensional structures. The value of L is often different at training time (where a larger L allows better coverage of the interval $[0, T]$, which helps the network learn a relationship that generalizes over t) and sampling time (where L being smaller means more efficient sampling). One can even pick the timesteps adaptively at sampling time to optimize this tradeoff [BLZ+22].

Remark 3.2. Various other works define the reverse process as moving backward in the time index t using an explicit difference equation, or differential equation in the limit $L \to \infty$, or forward in time using the transformation $\boldsymbol{y}_t = \boldsymbol{x}_{T-t}$, such that if t increases then $\boldsymbol{y}_t$ becomes closer to $\boldsymbol{x}_0$. Each paper uses its own notation, and it can sometimes be confusing to understand the relationships between different papers' results. In this work we strive to keep consistency: we move forward in time to noise, and backward in time to denoise. If you are reading another work that is not clear on the time index, or trying to implement an algorithm that is similarly unclear, there is one way to disambiguate every time: the sampling process should always have a *positive* coefficient on the denoiser term (or equivalently the score function) when taking a step.

Remark 3.3. The diffusion/denoising processes and associated models provide a way to transport the data distribution to and from a high-entropy (Gaussian or Gaussian-like) distribution. On the other hand, it may be useful to transport the data distribution to and from an arbitrary target distribution, which could be Gaussian, a Gaussian mixture, or even another type of data distribution. If we have this desideratum, we can use the *flow matching* framework to achieve this. Suppose the target random variable is $\boldsymbol{y}$, and our flow is

$$\boldsymbol{x}_t = \alpha_t \boldsymbol{x} + \sigma_t \boldsymbol{y}, \qquad \forall t \in [0, 1]. \tag{3.2.96}$$

such that $\alpha_0 = \sigma_1 = 1$ and $\alpha_1 = \sigma_0 = 0$, so that $\boldsymbol{x}_0 = \boldsymbol{x}$ and $\boldsymbol{x}_1 = \boldsymbol{y}$. Nothing about our strategy changes: we can train a denoiser $\bar{\boldsymbol{x}}_\theta$ to denoise $\boldsymbol{x}_t$ back to $\boldsymbol{x}$ (or equivalently a "noise" predictor to predict $\boldsymbol{y}$), and then apply our iterative denoising algorithms to start from a realization of $\boldsymbol{y}$ to obtain $\boldsymbol{x}$. The only thing missing is Tweedie's formula which connects the score $\nabla \log p_t$ of the denoiser to the denoiser; such a formula does not exist in general.

In practice, inspired by the concept of learning a so-called *transport map* between data pairs $\boldsymbol{x}$ and $\boldsymbol{y}$, practitioners often wish to train a *velocity predictor* $\bar{\boldsymbol{v}}_\theta$ to predict $\boldsymbol{v}_t = \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}_t = \alpha_t'\boldsymbol{x} + \sigma_t'\boldsymbol{y}$ instead of a denoiser $\bar{\boldsymbol{x}}_\theta$, or rather train $\bar{\boldsymbol{v}}_\theta$ to predict $\mathbb{E}[\boldsymbol{v}_t \mid \boldsymbol{x}_t]$. In this case, it is simplest to take the choice of coefficients $\alpha_t = 1-t$ and $\sigma_t = t$ (for $t \in [0,1]$), so that the prediction target is $\boldsymbol{v}_t = \boldsymbol{y} - \boldsymbol{x}$ for each time t. Given the estimator $\bar{\boldsymbol{v}}_\theta$, the standard sampling algorithm follows the field backwards in time, i.e., starting from a realization of $\boldsymbol{y}$, we can iteratively update the iterate as follows:

$$\hat{\boldsymbol{x}}_{t_{\ell-1}} = \hat{\boldsymbol{x}}_{t_\ell} - \bar{\boldsymbol{v}}_\theta(t_\ell, \hat{\boldsymbol{x}}_{t_\ell})\Delta t, \tag{3.2.97}$$

where $\Delta t = t_\ell - t_{\ell-1}$ is the timestep. Note that velocity prediction is algebraically equivalent to denoising (see Exercise 3.6). See Sections 8.6 and 8.9 for practical applications of flow matching.

Remark 3.4. In practice, diffusion usually takes many steps and many function evaluations of the denoiser. It is therefore natural to ask: "can we speed up the sampling process by using a smaller number of steps and function evaluations?" The answer is yes, and such models are often called *consistency models* or, more descriptively, *few-step models*. The core idea is to use a denoiser that takes in two times $\bar{\boldsymbol{x}}_\theta(s,t,\boldsymbol{x}_t)$ and estimates $\mathbb{E}[\boldsymbol{x}_s \mid \boldsymbol{x}_t]$, which in some sense is an easier task, or alternatively to use a velocity predictor $\bar{\boldsymbol{v}}_\theta(s,t,\boldsymbol{x}_t)$ and estimate the average velocity $\mathbb{E}[(\boldsymbol{x}_t - \boldsymbol{x}_s)/(t-s) \mid \boldsymbol{x}_t]$. Such denoisers have additional consistency conditions such as $\bar{\boldsymbol{x}}_\theta(u,t,\bar{\boldsymbol{x}}_\theta(s,u,\boldsymbol{\xi})) = \bar{\boldsymbol{x}}_\theta(s,t,\boldsymbol{\xi})$, which motivate auxiliary training losses. Once such a predictor is learned, the sampling process becomes very simple and efficient, as it only requires a few steps of the form

$$\hat{\boldsymbol{x}}_s = \bar{\boldsymbol{x}}_\theta(s,t,\hat{\boldsymbol{x}}_t) = \hat{\boldsymbol{x}}_t - \bar{\boldsymbol{v}}_\theta(s,t,\hat{\boldsymbol{x}}_t)\cdot(t-s). \tag{3.2.98}$$

Relevant formulations include flow maps [BAV25] and mean flows [GDB+25].

3.3 Memorization, Generalization, and Coding Rates

The calculation presented at the end of Section 3.2.1 seems to suggest (loosely speaking) that in practice, using a transformer-like network is a good choice for learning or approximating a denoiser. This is reasonable, but what is the problem with using any old neural network (such as a multi-layer perceptron (MLP)) and simply scaling it up to infinity? To answer this question, let us consider a constraint we have ignored or glossed over so far in this Chapter: we only have finite data with which to learn the denoiser. Indeed, let us consider the training problem (3.2.90), except this time we assume that our data $\boldsymbol{x}$ are drawn from the empirical distribution $p^{\boldsymbol{X}}$ on N training samples $\boldsymbol{X} = [\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}]$. Because $p^{\boldsymbol{X}}$ is supported on a 0-dimensional subset of the ambient space $\mathbb{R}^D$, we have that $h(p^{\boldsymbol{X}}) = -\infty$, so in some sense $p^{\boldsymbol{X}}$ is the simplest (i.e., lowest-entropy) distribution that could generate our observations $\boldsymbol{X}$. In short, this

assumption arguably *has the least inductive bias* about the data distribution! Many popular commentaries on deep learning and generative models would have you believe that, due to this property, the best way to solve the generative modeling problem is to scale a generic denoiser as large as possible and reap the corresponding benefits. However, this advice is (provably!) wrong in this setting. To understand why, let us note that the empirical distribution $p^{\boldsymbol{X}}$ is actually a mixture of Gaussians with zero covariances:

$$p^{[\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(N)}]} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{N}(\boldsymbol{x}^{(i)}, \mathbf{0}). \tag{3.3.1}$$

As such, we know that the (globally) optimal solution to the denoising problem (3.2.90) with $\boldsymbol{x}$ drawn from the empirical distribution, written out as follows:

$$\min_{\theta\in\Theta}\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[w_\tau\|\boldsymbol{x}^{(i)} - \bar{\boldsymbol{x}}_\theta(\tau, \boldsymbol{x}_\tau^{(i)})\|_2^2] \tag{3.3.2}$$

where $\boldsymbol{x}_t^{(i)}$ is the noised version of the training sample $\boldsymbol{x}^{(i)}$. The optimizer of this loss over all (square-integrable) denoisers $\bar{\boldsymbol{x}}$ is the associated optimal Gaussian denoiser from Example 3.3, i.e.,

$$\bar{\boldsymbol{x}}^\star(t, \boldsymbol{x}_t) = \sum_{i=1}^{N}\frac{\exp(-\|\boldsymbol{x}_t - \alpha_t\boldsymbol{x}^{(i)}\|_2^2/(2\sigma_t^2))}{\sum_{j=1}^{N}\exp(-\|\boldsymbol{x}_t - \alpha_t\boldsymbol{x}^{(j)}\|_2^2/(2\sigma_t^2))}\boldsymbol{x}^{(i)}. \tag{3.3.3}$$

This is a convex combination of the data $\boldsymbol{x}^{(i)}$, and the coefficients get "sharper" (i.e., closer to 0 or 1) as $t \to 0$. In particular, for small t, the denoiser is almost exactly equal to the closest sample in the training set, and the update moves the iterate closer to the closest point, such that at the end of sampling, the iterate $\hat{\boldsymbol{x}}$ is (nearly) equal to a training sample $\boldsymbol{x}^{(i)}$. Let us re-emphasize this point: *if we use the denoiser* (3.3.3) *and a good sampler, our samples will re-generate the training data.* This is confirmed by our theoretical guarantee on sampling Theorem 3.4, which, when instantiated in this context, says that if we take the number of sampling steps to ∞ and use the prescribed timestep schedule, the diffusion model associated with the denoiser (3.3.3) will always sample from the empirical distribution and thus always reproduce a training sample. This phenomenon is a particularly strict instantiation of *memorization* in generative modeling.[11]

Since in practice we only have a fixed finite dataset on which we can train our denoiser, the learning problem we aim to solve in this case is the finite-sample denoising loss (3.3.2) (compared to the infinite-sample denoising loss (3.2.90), for example). As such, *if we have an arbitrarily large model and optimize it as well as possible, it will learn to memorize and re-generate the training set.* In other words, this very large-scale model is not much more interesting than a simple database of the training data.

[11] Memorization is a much broader phenomenon whose exact characterization depends on the domain the model is trained on (e.g., text, images, videos, audio, etc.). Later in this Section we will discuss more heuristic measures of memorization.

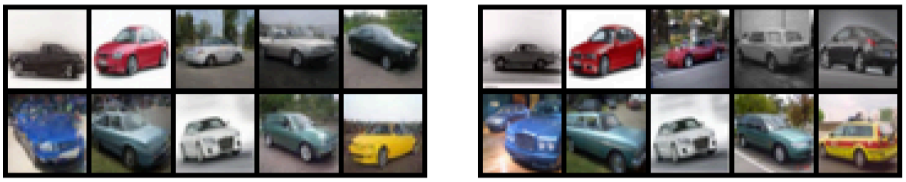

Figure 3.9: **Generated samples (left) from a diffusion model trained on 6,000 samples and (right) their closest points in the training dataset (in the Euclidean norm).** We observe that the generated samples are essentially pixel-perfect equivalents to the most similar training data. Both sets of images are from [YCK+23].

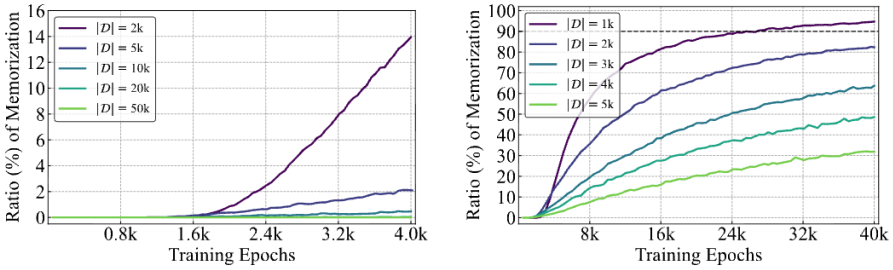

Figure 3.10: **Memorization versus dataset size for a fixed model, on (left) 4,000 training epochs and (right) 40,000 training epochs with smaller datasets**. We observe from the left plot that large enough datasets will never be memorized (e.g., 10,000 to 50,000 samples), whereas small datasets (1,000 to 5,000 samples) will be memorized after enough training. This means that as the dataset becomes larger relative to the model, the memorization ratio at the end of training goes from near-1 to near-0 relatively rapidly, indicating the possible existence of a phase transition from memorization to generalization as the dataset size increases. Figure taken from [GDP+23].

The above argument holds for arbitrarily powerful models that can fit the (true) memorizing denoiser. It is therefore relevant to ask whether it actually bears out in practice: do very large models (relative to the dataset) learn to memorize the training data? It turns out that this is empirically true. One of the first studies on this was [YCK+23]. To understand their results, we define the memorization ratio as the probability of memorization:

$$\text{MemRatio}(\bar{\boldsymbol{x}}) = \mathbb{P}_{\hat{\boldsymbol{x}} \sim \text{DM}(\bar{\boldsymbol{x}})}[\hat{\boldsymbol{x}} \text{ is memorized}], \tag{3.3.4}$$

where $\text{DM}(\bar{\boldsymbol{x}})$ is the distribution induced by the diffusion model that uses denoiser $\bar{\boldsymbol{x}}$, and the sample $\hat{\boldsymbol{x}}$ is memorized using the following criterion:

$$\hat{\boldsymbol{x}} \text{ is memorized} \iff \frac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_{(1)}(\hat{\boldsymbol{x}})\|_2}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_{(2)}(\hat{\boldsymbol{x}})\|_2} < \frac{1}{3}, \tag{3.3.5}$$

where $\boldsymbol{x}_{(i)}(\hat{\boldsymbol{x}})$ is the i^{th} closest point to $\hat{\boldsymbol{x}}$ in the training dataset (in the Euclidean norm), and $1/3$ is a constant that is empirically shown to agree well with human perception of memorization. Notice that this definition exactly corresponds to our earlier intuitive definition of memorization as sampled points each being much closer to a particular training sample than all other training samples. They showed that this kind of memorization occurs in practice when the model is much larger (in terms of the number of parameters) relative to the data; Figure 3.9 demonstrates that in this case, generated samples are essentially pixel-perfect copies of training samples. Slightly later, [GDP+23] performed a careful study to characterize how various factors such as architectures and training time impact memorization, showing among other things that as the dataset size increases relative to a fixed model, the memorization ratio rapidly changes from near-1 to near-0, i.e., the model quickly stops memorizing. This indicates the possibility of a phase transition between memorization and generalization as the model becomes smaller relative to the data. [BPM+25] investigates the existence and theoretical mechanism of this phase transition in synthetic data drawn from a Gaussian mixture model, demonstrating that it exists and providing a partial characterization of the model size at which models will start or stop memorizing. Other theoretical work such as [GVM25; KG24; NZM+24; ZLL+25] also attempts to provide a theoretical characterization of the phase transition in terms of model and dataset size in different settings.

In order to empirically prove the existence of a phase transition between memorization and generalization in real data, [ZZL+24] rethought the definition of memorization using the following reasoning. In the very high-dimensional space of natural images, Euclidean distances between points become much more strict [WM22], in the sense that if two images are not pixel-perfect matches to each other, then their Euclidean distances will concentrate sharply around a fixed value (scaling with the images' dimension). This does not catch the case where certain *concepts* (such as copyrighted cartoon characters) are generated in novel positions, making it very difficult to form effective and practical memorization criteria around the Euclidean distance. To fix this, [ZZL+24] uses a particular function f_{SSCD} (a pre-trained neural network called a Self-Supervised

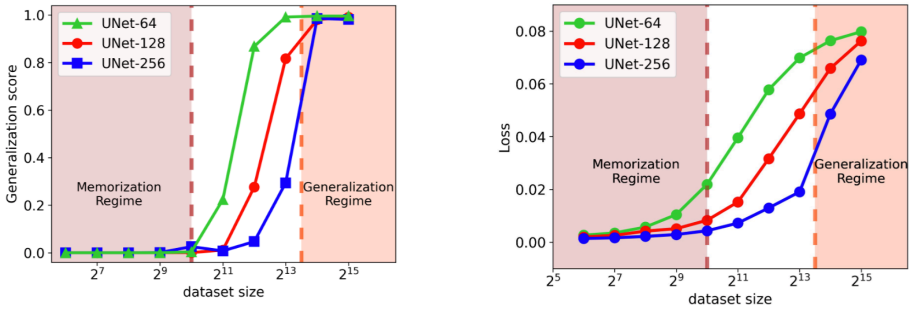

Figure 3.11: **The generalization score (left) and training loss (right) for a fixed model and increasing data**, from [ZZL+24]. The generalization score is 1− the fraction of samples that are memorized, and the different lines correspond to differently-sized denoiser models (UNet-64 is smaller than UNet-128, which in turn is smaller than UNet-256). The figures show that, as the dataset size becomes larger relative to the model size, the loss increases significantly and memorization stops occurring, showing that our theory and intuition apply to real models.

Copy Detector [PRR+22]) to map images into a Euclidean space, then takes the normalized inner product to obtain their (cosine) similarity, such that images that are similar due to this cosine similarity are claimed to be conceptually similar in the sense needed for memorization. Then, [ZZL+24] says that a sample $\hat{\boldsymbol{x}}$ is memorized if its similarity with any training point is $> 3/5$, i.e.,

$$\hat{\boldsymbol{x}} \text{ is memorized} \iff \max_{i\in[N]} \frac{f_{\text{SSCD}}(\hat{\boldsymbol{x}})^\top f_{\text{SSCD}}(\boldsymbol{x}^{(i)})}{\|f_{\text{SSCD}}(\hat{\boldsymbol{x}})\|_2 \|f_{\text{SSCD}}(\boldsymbol{x}^{(i)})\|_2} > \frac{3}{5} \tag{3.3.6}$$

Figure 3.11 shows the outcome of taking many (around 10,000) samples from diffusion models trained on differently-sized training datasets in terms of estimating memorization and computing the training loss. The outcome is in line with the above discussion; when the model is too large relative to the training data and the training loss is too small, memorization occurs and samples reproduce the training data. Thus, our intuition applies in practice: *for generative diffusion models, scaling the model is not all you need!*

So now the question is: what kind of denoiser should we use? The key is to choose a network architecture that can approximate the true denoiser (say, the one corresponding to a low-rank distribution as in (3.2.80)) without approximating the memorizing denoiser (3.3.3). Some work has explored this fine line and why modern diffusion models, which use Transformer- and convolution-based architectures, can memorize and generalize in different regimes [KG24; NZM+24; ZLL+25].

Let us at last return to the initial premise of this Chapter, described in Section 3.1.3: finding the minimal-entropy distribution that models the data is the only way to learn the distribution. The above discussion has hopefully shown that this claim is false. However, a related claim, also argued in Section 3.1.3

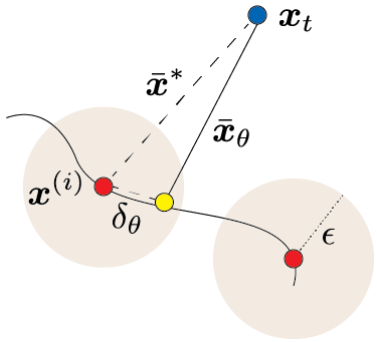

Figure 3.12: **Visualizing the action of a well-trained denoiser** as a perturbation of the empirical denoiser at *small times*, when the memorizing denoiser acts like a projection onto the nearest point in the training set.

because of its equivalence in the situations we had then considered, might still be true: finding the distribution with minimal *coding rate* is a way to learn the distribution. Supposing for the moment that $\boldsymbol{x}$ has a density p, the coding rate (i.e., cross-entropy) of the empirical distribution $p^{\boldsymbol{X}}$ with respect to p is $\mathrm{CE}(p \parallel p^{\boldsymbol{X}}) = \infty$. This coding rate is obviously not minimal in any sense; if we use the same successively better approximations p^n, p^e, and $\hat{q}$ discussed in Section 3.1.3, it holds

$$\infty = \mathrm{CE}(p \parallel p^{\boldsymbol{X}}) > \mathrm{CE}(p \parallel p^n) > \mathrm{CE}(p \parallel p^e) > \mathrm{CE}(p \parallel \hat{q}) \approx h(p). \tag{3.3.7}$$

Minimizing the coding rate therefore seems a sensible way to learn the distribution while avoiding this issue. However, a problem of tractability now arises: how can we devise algorithms that learn the distribution by minimizing the coding rate while avoiding the drawbacks associated with overly minimizing entropy?

One clue is to examine what happens with successfully trained diffusion models that do not memorize. By writing the trained denoiser $\bar{\boldsymbol{x}}_\theta$ as a perturbation of the memorizing denoiser $\bar{\boldsymbol{x}}^\star$, i.e.,

$$\bar{\boldsymbol{x}}_\theta(t, \boldsymbol{x}_t) = \bar{\boldsymbol{x}}^\star(t, \boldsymbol{x}_t) + \boldsymbol{\delta}_\theta(t, \boldsymbol{x}_t) \tag{3.3.8}$$

we can gain graphical intuition about the action of a well-trained denoiser, as in Figure 3.12. At a very high level, the trained denoiser computes the memorizing denoiser plus some perturbation. At a small time t (perhaps even at the last step of denoising), the memorizing denoiser points entirely in the direction of a single training sample (the training sample to be re-generated by the current iterate). The distribution being learned, or denoised against, is just the empirical distribution with some ϵ-balls around each point,[12] where ϵ is an upper bound on the magnitude of the perturbation, visualized in Figure 3.12 by the

[12] More formally, this perturbed distribution has support on (a subset of) a union of ϵ-balls centered at each training sample. This picture is supported by past work, e.g., [SKC+25].

faint orange balls.[13] For increasingly larger well-trained diffusion models, the perturbation radius ϵ shrinks to 0, and the model learns the empirical distribution. On the other hand, ϵ large enough that balls corresponding to different points overlap or are close may imply that the diffusion model generalizes.

Thus, our successful diffusion model can be viewed as minimizing a *lossy coding rate*, i.e., coding rate up to a precision ϵ (so as to be able to disregard the ϵ-balls around each point). This notion of the lossy coding rate turns out to be the right thing to minimize, morally speaking, and will be *extremely* useful. We will tease out its uses in Chapter 4 and throughout the rest of this book.

3.4 Summary and Notes

Key messages. In this Chapter, we have shown that, to pursue a general low-dimensional distribution, a universal computational approach is to start with a generic distribution and compress it toward the distribution through (iterative) noise or entropy reduction. As we discussed in the preceding Chapter, this is even the case for arguably the simplest distributions of a single subspace, such as computing PCA via the power-iteration algorithm. As we will see in future chapters, the fundamental roles of modern deep networks are not to fit arbitrary (noisy) data distributions or functions; instead, they are to realize or approximate those (lossy) compressing or denoising operators. We will derive precise forms of those operators and the associated deep architectures in future chapters.

More extensions and references. The use of denoising and diffusion for sampling has a rich history. The first work clearly about a diffusion model is probably [SWM+15], but before this there are many works on denoising as a computational and statistical problem. The most relevant of these is probably [Hyv05], which explicitly uses the score function to denoise and to perform independent component analysis. The most popular follow-ups are essentially co-occurring: [HJA20; SE19]. Since then, thousands of papers have built on diffusion models; we revisit this topic in Chapter 6.

Many of these works use a different stochastic process than the simple linear combination (3.2.85). In fact, all works listed above emphasize the need to add *independent* Gaussian noise at the beginning of each step of the forward process. Theoretically-minded work uses Brownian motion or stochastic differential equations to formulate the forward process [SSK+21]. However, since linear combinations of Gaussians still result in Gaussians, the *marginal distributions* of such processes still take the form of (3.2.85). Most of our discussion requires only that the marginal distributions are what they are, so our overly simplistic model is actually enough for almost everything. The only time marginal distributions are not enough is when we derive an expression for $\mathbb{E}[\boldsymbol{x}_s \mid \boldsymbol{x}_t]$ in terms

[13] In high dimensions, it is hypothesized that a better model for these perturbations is not being contained by balls but rather by much more anisotropic geometry-adaptive shapes [FPH+25], but the intuition remains similar.

of $\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t]$. Different (noising) processes give different such expressions, which can be used for sampling (and of course there are other ways to derive efficient samplers, such as the ever-popular DDPM sampler). The process in (3.2.85) is a bona fide stochastic process whose "natural" denoising iteration takes the form of the popular DDIM algorithm [SME20]. (Even this equivalence is not trivial; we cite [DGG+25] as justification.)

On top of the theoretical work [LY24] covered in Section 3.2.2, and the lineage of work that it builds on, which studies the *sampling* efficiency of diffusion models when the data has low-dimensional structure, there is a large body of work that studies the *training* efficiency of diffusion models when the data has low-dimensional structure. Specifically, Chen et al. [CHZ+23] and Wang et al. [WZZ+24] characterized the approximation and estimation error of denoisers when the data belongs to a mixture of low-rank Gaussians, showing that the number of training samples required to accurately learn the distribution scales with the intrinsic dimension of the data rather than the ambient dimension. There is considerable *methodological* work that attempts to use the low-dimensional structure of the data to perform various tasks with diffusion models. We highlight three: image editing [CZG+24], watermarking [LZQ24], and unlearning [CZL+25], though this list is inexhaustive.

3.5 Exercises and Extensions

Exercise 3.1. Show that (3.2.8) is the optimal solution of Problem (3.2.7).

Exercise 3.2. Consider random vectors $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{y} \in \mathbb{R}^d$, such that the pair $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{D+d}$ is jointly Gaussian. This means that

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_{\boldsymbol{x}} \\ \boldsymbol{\mu}_{\boldsymbol{y}} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{\boldsymbol{x}} & \boldsymbol{\Sigma}_{\boldsymbol{xy}} \\ \boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top & \boldsymbol{\Sigma}_{\boldsymbol{y}} \end{bmatrix} \right),$$

where the mean and covariance parameters are given by

$$\boldsymbol{\mu}_{\boldsymbol{x}} = \mathbb{E}[\boldsymbol{x}], \quad \boldsymbol{\mu}_{\boldsymbol{y}} = \mathbb{E}[\boldsymbol{y}], \quad \begin{bmatrix} \boldsymbol{\Sigma}_{\boldsymbol{x}} & \boldsymbol{\Sigma}_{\boldsymbol{xy}} \\ \boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top & \boldsymbol{\Sigma}_{\boldsymbol{y}} \end{bmatrix} = \mathbb{E}\left[\begin{bmatrix} \boldsymbol{x} - \mathbb{E}[\boldsymbol{x}] \\ \boldsymbol{y} - \mathbb{E}[\boldsymbol{y}] \end{bmatrix} \begin{bmatrix} \boldsymbol{x} - \mathbb{E}[\boldsymbol{x}] \\ \boldsymbol{y} - \mathbb{E}[\boldsymbol{y}] \end{bmatrix}^\top \right]$$

Assume that $\boldsymbol{\Sigma}_{\boldsymbol{y}}$ is positive definite (hence invertible); then positive semidefiniteness of the covariance matrix is equivalent to the Schur complement condition $\boldsymbol{\Sigma}_{\boldsymbol{x}} - \boldsymbol{\Sigma}_{\boldsymbol{xy}} \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1} \boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top \succeq \mathbf{0}$.

In this exercise, we will prove that the conditional distribution $p_{\boldsymbol{x}|\boldsymbol{y}}$ is Gaussian: namely,

$$p_{\boldsymbol{x}|\boldsymbol{y}} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}} + \boldsymbol{\Sigma}_{\boldsymbol{xy}} \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1} (\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}}), \boldsymbol{\Sigma}_{\boldsymbol{x}} - \boldsymbol{\Sigma}_{\boldsymbol{xy}} \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1} \boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top \right). \tag{3.5.1}$$

A direct path to prove this result manipulates the defining ratio of densities $p_{\boldsymbol{x},\boldsymbol{y}} / p_{\boldsymbol{y}}$. We sketch an algebraically concise argument of this form below.

1. Verify the following matrix identity for the covariance:

$$\begin{bmatrix} \boldsymbol{\Sigma}_{\boldsymbol{x}} & \boldsymbol{\Sigma}_{\boldsymbol{xy}} \\ \boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top & \boldsymbol{\Sigma}_{\boldsymbol{y}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_D & \boldsymbol{\Sigma}_{\boldsymbol{xy}}\boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1} \\ \boldsymbol{0} & \boldsymbol{I}_d \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_{\boldsymbol{x}} - \boldsymbol{\Sigma}_{\boldsymbol{xy}}\boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_{\boldsymbol{y}} \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_D & \boldsymbol{0} \\ \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top & \boldsymbol{I}_d \end{bmatrix}. \tag{3.5.2}$$

   One arrives at this identity by performing two rounds of (block) Gaussian elimination on the covariance matrix.

2. Based on the previous identity, show that

$$\begin{bmatrix} \boldsymbol{\Sigma}_{\boldsymbol{x}} & \boldsymbol{\Sigma}_{\boldsymbol{xy}} \\ \boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top & \boldsymbol{\Sigma}_{\boldsymbol{y}} \end{bmatrix}^{-1} = \boldsymbol{P}^\top \begin{bmatrix} \left(\boldsymbol{\Sigma}_{\boldsymbol{x}} - \boldsymbol{\Sigma}_{\boldsymbol{xy}}\boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top\right)^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1} \end{bmatrix} \boldsymbol{P}, \tag{3.5.3}$$

   where

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{I}_D & \boldsymbol{0} \\ -\boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top & \boldsymbol{I}_d \end{bmatrix}, \tag{3.5.4}$$

   and whenever the relevant inverses are defined.[14] Conclude that

$$\begin{bmatrix} \boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}} \\ \boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{\Sigma}_{\boldsymbol{x}} & \boldsymbol{\Sigma}_{\boldsymbol{xy}} \\ \boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top & \boldsymbol{\Sigma}_{\boldsymbol{y}} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}} \\ \boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}} \end{bmatrix} \tag{3.5.5}$$

$$= \begin{bmatrix} \boldsymbol{x} - \boldsymbol{m} \\ \boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}} \end{bmatrix}^\top \begin{bmatrix} \left(\boldsymbol{\Sigma}_{\boldsymbol{x}} - \boldsymbol{\Sigma}_{\boldsymbol{xy}}\boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{xy}}^\top\right)^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} - \boldsymbol{m} \\ \boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}} \end{bmatrix}, \tag{3.5.6}$$

   where $\boldsymbol{m} = \boldsymbol{\mu}_{\boldsymbol{x}} + \boldsymbol{\Sigma}_{\boldsymbol{xy}}\boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}})$.

3. By dividing $p_{\boldsymbol{x},\boldsymbol{y}}/p_{\boldsymbol{y}}$, prove Equation (3.5.1). (*Hint: Using the previous identities, only minimal algebra should be necessary. For the normalizing constant, use Equation* (3.5.3) *to factor the determinant similarly.*)

Exercise 3.3. Show the Sherman–Morrison–Woodbury identity, i.e., for matrices $\boldsymbol{A}$, $\boldsymbol{C}$, $\boldsymbol{U}$, $\boldsymbol{V}$ such that $\boldsymbol{A}$, $\boldsymbol{C}$, and $\boldsymbol{A} + \boldsymbol{UCV}$ are invertible,

$$(\boldsymbol{A} + \boldsymbol{UCV})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{C}^{-1} + \boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1}. \tag{3.5.7}$$

Exercise 3.4. Rederive the following, assuming $\boldsymbol{x}_t$ follows the generalized noise model (3.2.85).

- Tweedie's formula: (3.2.86).
- The DDIM iteration: (3.2.87).
- The Bayes-optimal denoiser for a Gaussian mixture model: (3.2.88).

Exercise 3.5.

[14] In cases where the Schur complement term is not invertible, the same result holds with its inverse replaced by the Moore-Penrose pseudoinverse. In particular, the conditional distribution (3.5.1) becomes a degenerate Gaussian distribution.

1. Implement the formulae derived in Exercise 3.4, building a sampler for Gaussian mixtures.
2. Reproduce Figure 3.4 and Figure 3.8.
3. We now introduce *Flow Matching (FM)* (see Remark 3.3), as follows:

$$\alpha_t = 1 - t, \qquad \sigma_t = t. \tag{3.5.8}$$

   Implement this process using the same framework, and test it for sampling in high dimensions. Which process gives better or more stable results?

Exercise 3.6. Consider the somewhat familiar interpolation

$$\boldsymbol{x}_t = \alpha_t \boldsymbol{x} + \sigma_t \boldsymbol{y}, \qquad \forall t \in [0, T]. \tag{3.5.9}$$

where $\boldsymbol{y}$ is a (not necessarily Gaussian) random variable. We define the *velocity* as

$$\boldsymbol{v}_t = \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}_t = \alpha_t' \boldsymbol{x} + \sigma_t' \boldsymbol{y}, \qquad \forall t \in [0, T]. \tag{3.5.10}$$

Compute an algebraic relationship writing $\mathbb{E}[\boldsymbol{v}_t \mid \boldsymbol{x}_t]$ in terms of $\mathbb{E}[\boldsymbol{x} \mid \boldsymbol{x}_t]$ and vice versa, showing that velocity prediction is algebraically equivalent to denoising and thus also to noise prediction (see Remark 3.3).