Chapter 6

Consistent and Self-Consistent Representations

> "*Everything should be made as simple as possible, but not any simpler.*"
>
> – Albert Einstein

In the past chapters, we have argued that the fundamental goal of learning is to *identify* a data distribution with low-dimensional support and *transform* it into a compact and structured representation. This entails two related problems:

1. **Distribution learning:** How do we get ahold of the distribution of data $\boldsymbol{X}$, to enable generalizable sampling/generation?
2. **Representation learning:** How do we map the data $\boldsymbol{X}$ to a compact and structured representation $\boldsymbol{Z}$, such that the representation is sufficient to reconstruct $\boldsymbol{X}$?

In Chapters 3 and 4, we have developed a complete computational solution to the distribution learning problem for general data distributions, based on the diffusion-denoising process. In Chapter 4, we have defined a compression-based objective function, the information gain, whose maximization leads to compact and structured representations. And in Chapter 5, we have seen that the layers of popular deep network architectures can be interpreted as incremental optimization steps towards maximizing the information gain. This leaves us poised to present a mathematically-founded solution to the representation learning problem.

However, when we try to achieve a certain objective through optimization, there is no guarantee that the solution $\boldsymbol{Z}$ found in the end by greedy optimization is the correct solution. In fact, even if the optimization process manages

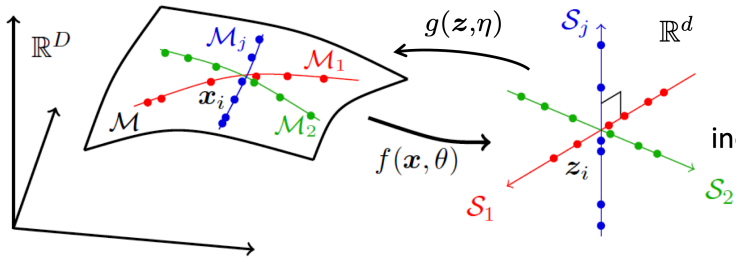


Figure 6.1: Learn a good auto-encoding representation for a general distribution whose supports are on a mixture of low-dimensional submanifolds. Notice that in addition to learning a good representation via a mapping $f(\boldsymbol{x}, \theta)$ in an end-to-end fashion as we did in Chapter 5, we want to make sure that the so-learned representation can be decoded, via an "inverse" mapping $g(\boldsymbol{z}, \eta)$, back to approximate the original data distribution.

to find the globally optimal solution $\boldsymbol{Z}^*$, there is no guarantee that the solution corresponds to a complete representation of the data distribution.[1] Therefore, an outstanding question is: how can we ensure that the learned representation of the data distribution is correct or good enough?

Of course, the only way to verify this is to see whether there exists a decoding map, say g, that can decode the learned representation to reproduce the original data (distribution) well enough:

$$\boldsymbol{X} \xrightarrow{\mathcal{E}=f} \boldsymbol{Z} \xrightarrow{\mathcal{D}=g} \hat{\boldsymbol{X}} \tag{6.0.1}$$

in terms of some measure of similarity:

$$d(\boldsymbol{X}, \hat{\boldsymbol{X}}). \tag{6.0.2}$$

This leads to the concept of *consistent representation*.[2] As we have briefly alluded to in Chapter 1 (see Figure 1.25), *autoencoding*, which integrates the encoding and decoding processes, is a natural framework for learning such a representation. We have studied some important special cases in Chapter 2 where the supports of the distribution are piecewise linear. In this chapter, Section 6.1, we will study how to extend autoencoding to more general classes of distributions whose supports are nonlinear, as illustrated in Figure 6.1, while enforcing the consistency between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$.

Given this classical autoencoding formalism, there is a natural question: *where does distribution learning end and representation learning begin?* As we

[1] This could be due to many reasons: for example, the data available for learning the distribution might not be sufficient, or the formulation of the optimization program fails to consider some additional constraints or conditions.

[2] Note that here the notion of consistency is empirical or inductive in nature. It is different from the notion of logic and deductive consistency in mathematics, also known as the Hilbert's Principle, which we will have more discussions in Chapter 9.

will see, the answer comes down to *efficiency*. It is typically far more effective, as demonstrated by modern practice, to learn a compact representation for the data distribution as a type of 'preparation' of the distribution, and only then to perform distribution learning on the so-learned representation. This is the dominant methodology in practical problems. We will demonstrate the links between the foundations we have developed in Chapters 4 and 5 and the state of the art in Section 6.1.5, where the "representation autoencoder" framework demonstrates how open-loop learned representations (which we discussed in Section 4.3.2, and will review) can be combined with a simple decoder and a $\boldsymbol{Z}$-space diffusion process to enable efficient, simultaneous representation learning and sampling.

More fundamentally, in many practical and natural learning scenarios, it can be difficult or even impossible to compare distributions of the data $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$. We are left with the only option of comparing the learned feature $\boldsymbol{Z}$ with its image $\hat{\boldsymbol{Z}}$ under the encoder f:

$$\boldsymbol{X} \xrightarrow{\mathcal{E}=f} \boldsymbol{Z} \xrightarrow{\mathcal{D}=g} \hat{\boldsymbol{X}} \xrightarrow{\mathcal{E}=f} \hat{\boldsymbol{Z}} \tag{6.0.3}$$

This leads to the notion of a *self-consistent representation*. In Section 6.2, we will study when and how we can learn a consistent representation by enforcing the self-consistency between $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$ only through a closed-loop transcription framework.

Furthermore, in many practical and natural learning scenarios, we normally do not have sufficient samples of the data distribution all available at once. For example, animals and humans develop their visual memory by continuously taking in increments of observations throughout their lives. In Section 6.3, we will study how to extend the closed-loop transcription framework to learn a self-consistent representation in a *continuous learning* setting.

Of course, a fundamental motivation for why we ever want to identify the low-dimensional structures in a data distribution and find a good representation is to make it easy to use the data for various tasks of intelligence, such as classification, completion, and prediction. Therefore, the resulting joint representation $(\boldsymbol{x}, \boldsymbol{z})$ must be structured in such a way that is best suited for these tasks. In the following chapter, we will see how the learned representation can be structured to facilitate conditioned completion or generation tasks.

6.1 Learning Consistent Representations

Here we give a formal definition of consistent representations, which are closely related to the concept of autoencoding.

Definition 6.1 (Consistent Representations)**.** Given data $\boldsymbol{X}$, a *consistent representation* is a pair of functions $(f\colon \mathcal{X} \to \mathcal{Z}, g\colon \mathcal{Z} \to \mathcal{X})$, such that the *features* $\boldsymbol{Z} = f(\boldsymbol{X})$ are compact and structured, and the *autoencoding*

$$\hat{\boldsymbol{X}} \doteq g(\boldsymbol{Z}) = g(f(\boldsymbol{X})) \tag{6.1.1}$$

is *close* to $\boldsymbol{X}$ according to one of the following two measures:

1. We say that it is *sample-wise* consistent if $\boldsymbol{X} \approx \hat{\boldsymbol{X}}$ in a certain norm with high probability.

2. We say that the representation is *distributionally consistent* if $\text{Law}(\boldsymbol{X}) \approx \text{Law}(\hat{\boldsymbol{X}})$.

Astute readers may have noticed that if we do not impose certain requirements on the representation $\boldsymbol{Z}$ sought, the above problem has a trivial solution: one may simply choose the functions f and g to be the identity map! Hence, the true purpose of seeking an autoencoding is to ensure that the $\boldsymbol{Z}$ so obtained is both more compact and more structured than $\boldsymbol{X}$. As we have seen in Chapter 5, it suffices to design the representation to maximize the rate reduction

$$\Delta R_\epsilon(\boldsymbol{Z}). \tag{6.1.2}$$

From the definition of consistent representation, it is required that the representation $\boldsymbol{Z}$ be sufficient to recover the original data distribution $\boldsymbol{X}$ to some degree of accuracy. For sample-wise consistency, a typical choice is to minimize the expected reconstruction error:

$$d(\boldsymbol{X}, \hat{\boldsymbol{X}}) = \mathbb{E}[\|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_2^2]. \tag{6.1.3}$$

For consistency in distribution, a typical choice is to minimize a certain distributional distance such as the KL divergence[3]:

$$d(\boldsymbol{X}, \hat{\boldsymbol{X}}) = \mathcal{D}_{KL}(\boldsymbol{X} \| \hat{\boldsymbol{X}}). \tag{6.1.4}$$

Hence, computation aside, when we seek a good autoencoding for a data distribution $\boldsymbol{X}$, conceptually we try to find an encoder f and decoder g such that

$$\min_{f,g} [-\Delta R_\epsilon(\boldsymbol{Z}) + d(\boldsymbol{X}, \hat{\boldsymbol{X}})]. \tag{6.1.5}$$

Notice that the quantization error ϵ plays the role of the tradeoff parameter in this Lagrangian relaxation.

In the remainder of this section, we will study how to solve such autoencoding problems under different conditions, from simple and ideal cases to increasingly more challenging and realistic conditions. We will culminate with an efficient instantiation of (6.1.5) (in Section 6.1.5) that ensures the representation space $\boldsymbol{Z}$ is compact and structured while also allowing for accurate reconstruction and efficient sampling.

[3] Note that for distributions without common support, which is typical for degenerate distributions, KL divergence may not even be well-defined. In fact, much of the distribution learning literature is trying to address this technical difficulty by replacing or approximating it with something well-defined and efficiently computable.

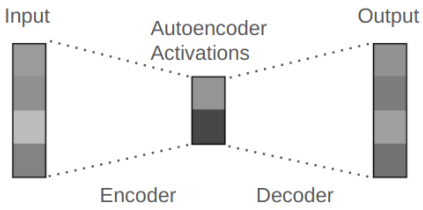


Figure 6.2: Illustration of a typical autoencoder such as PCA, seeking a low-dimensional representation $\boldsymbol{z}$ of high-dimensional data $\boldsymbol{x}$.

6.1.1 Linear Autoencoding via PCA

According to [Bal11], the phrase "autoencoder" was first introduced by Hinton and Rumelhart [RHW86a] so that a deep representation could be learned via back propagation (BP) in a self-supervised fashion—reconstructing the original data is the self-supervising task. However, the very same concept of seeking a compact and consistent representation has its roots in many classic studies. As we have already seen in Chapter 2, the classical PCA, ICA, and sparse dictionary learning all share a similar goal. The only difference is that when the underlying data distribution is simple (linear and independent), the encoding or decoding mappings become easy to represent and learn: they do not need to be deep and often can be computed in closed form or with an explicit algorithm.

It is instructive to see how the notion of consistency we have defined plays out in the simple case of PCA: here, the consistent encoding and decoding mappings are given by a single-layer linear transform:

$$\boldsymbol{X} \xrightarrow{\mathcal{E}=\boldsymbol{U}^\top} \boldsymbol{Z} \xrightarrow{\mathcal{D}=\boldsymbol{U}} \hat{\boldsymbol{X}}, \tag{6.1.6}$$

where $\boldsymbol{U} \in \mathbb{R}^{D\times d}$ typically with $d \ll D$. Hence $\boldsymbol{U}^\top$ represents a projection from a higher-dimensional space $\mathbb{R}^D$ to a lower one $\mathbb{R}^d$, as illustrated in Figure 6.2.

As we saw in Chapter 2, when the distribution of $\boldsymbol{x}$ is indeed supported on a low-dimensional subspace $\boldsymbol{U}_o$, the compactness of the representation $\boldsymbol{z}$ produced by $\mathcal{E}$ is a direct consequence of correctly estimating (and enforcing) the dimension of this subspace. Recall that gradient descent on the reconstruction criterion exactly yields these sample-wise consistent mappings: indeed, the optimal solution to the problem

$$\min_{\boldsymbol{U}^\top\boldsymbol{U}=\boldsymbol{I}} \mathbb{E}_{\boldsymbol{x}}\left[\left\|\boldsymbol{x}-\boldsymbol{U}\boldsymbol{U}^\top\boldsymbol{x}\right\|_2^2\right] \tag{6.1.7}$$

precisely coincides with $\boldsymbol{U}_o$ when the dimension of the representation is sufficiently large. In this case, we obtain sample-wise consistency for free, since this guarantees that $\boldsymbol{U}_o\boldsymbol{U}_o^\top\boldsymbol{x} = \boldsymbol{x}$. Notice that in the case of PCA, the rate reduction term in (6.1.5) becomes void as the regularization on the representation $\boldsymbol{Z}$ sought is explicit: It spans the entire subspace of lower dimension[4].

[4] One may view $\Delta R = 0$ in this case.

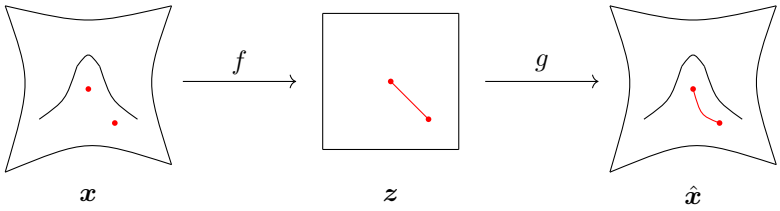


Figure 6.3: A depiction of interpolation through manifold flattening on a manifold in $\mathbb{R}^3$ of dimension $d = 2$. To interpolate two points on the data manifold, map them through the flattening map f to the flattened space, take their convex interpolants, and then map them back to the data manifold through the reconstruction map g.

6.1.2 Nonlinear PCA and Autoencoding

Of course, one should expect that things will no longer be so simple when we deal with more complex distributions whose underlying low-dimensional structures are nonlinear.

Data on a Nonlinear Submanifold. So, to move beyond the linear structure addressed by PCA, we may assume that the data distribution lies on a (smooth) submanifold $\mathcal{M}$. The intrinsic dimension of the submanifold, say d, is typically much lower than the dimension of the ambient space $\mathbb{R}^D$. From this geometric perspective, we typically want to find a nonlinear mapping f such that the resulting manifold $f(\mathcal{M})$ is flattened, as illustrated by the example shown in Figure 6.3. The resulting feature $\boldsymbol{z}$-space is typically more compact (of lower-dimension) than the $\boldsymbol{x}$-space, and the manifold is flat. From the statistical perspective, which is complementary to the geometric perspective but distinct in general, we may also want to ensure that the data distribution on $\mathcal{M}$ is mapped to a sufficiently regular distribution, say a Gaussian or uniform distribution (with very low-dimensional support), in the $\boldsymbol{z}$-space. In general, the problem of learning such an autoencoding mapping for this class of data distributions is known as *nonlinear principal component analysis* (NLPCA).

A Classical Attempt via a Two-Layer Network. As we have seen above, in the case of PCA, a single-layer linear neural network is sufficient. That is no longer the case for NLPCA. In 1991, Kramer [Kra91] proposed to solve NLPCA by using a two-layer neural network to represent the encoder mapping f (or its inverse g) based on the universal representation property of two-layer networks with sigmoid activation:

$$\boldsymbol{z} = \boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}), \tag{6.1.8}$$

where $\sigma(\,\cdot\,)$ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{6.1.9}$$

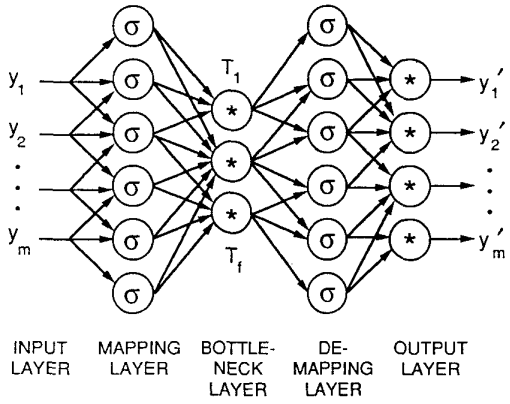


Figure 6.4: Nonlinear PCA by autoassociative neural networks of depth two for both the encoding and decoding mappings, suggested by Kramer [Kra91].

Cybenko [Cyb89] showed that functions of the above form (with enough hidden nodes) can approximate any smooth nonlinear function, say the encoder $f(\cdot)$, to an arbitrary precision. In particular, they can represent the flattening and reconstruction maps for data distributions supported on unions of low-dimensional manifolds, as in Figure 6.3. The overall architecture of the original networks proposed by Kramer is illustrated in Figure 6.4.

Unfortunately, unlike the case of PCA above, there is in general no closed-form learning scheme for the parameters $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{b})$ of these networks. Hence it was proposed to train the network via back propagation with the supervision of reconstruction error:

$$\min_{\boldsymbol{\theta}} \mathbb{E}[\|\boldsymbol{x} - \hat{\boldsymbol{x}}(\boldsymbol{\theta})\|_2^2]. \tag{6.1.10}$$

Compared to the simple case of PCA, we utilize the same reconstruction objective for learning, but a far more complex nonlinear class of models for parameterizing and learning the encoder and decoder. Although universal approximation properties such as Cybenko's suggest that *in principle* learning consistent autoencoders via this framework is possible—because for any random sample of data, given enough parameters, such autoencoding pairs exist—one often finds it highly nontrivial to find them using gradient descent. Moreover, to obtain a sufficiently informative reconstruction objective and model for the distribution of high-dimensional real-world data such as images, the required number of samples and hidden nodes can be huge. In addition, as a measure of the compactness of the learned representation, the (lower) dimension of $\boldsymbol{z}$ for the bottleneck layer is often chosen heuristically.[5]

[5] In the later work [HZ93], Hinton et al. suggested to use the minimum description length (MDL) principle to promote the compactness of the learned coding scheme, in a spirit very similar to the rate distortion measure introduced in this book.

Manifold Flattening via a Deeper Network. Based on the modern practice of deep networks, such classical shallow and wide network architectures are known to be rather difficult to train effectively and efficiently via back propagation (BP), partly due to the vanishing gradient of the sigmoid function. Hence, the modern practice normally suggests further decomposing the nonlinear transform f (or g) into a composition of many more layers of simpler transforms, resulting in a deeper network architecture [HS06], as illustrated in Figure 6.5.

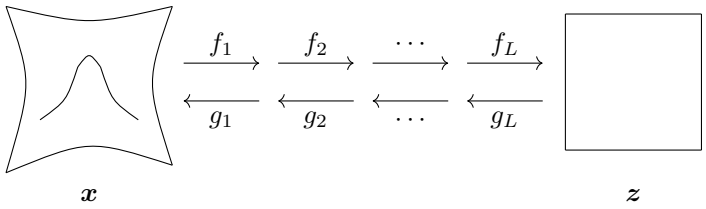


Figure 6.5: A depiction of the construction process of the flattening and reconstruction pair (f, g), where the encoder $f = f_L \circ f_{L-1} \circ \cdots \circ f_1$ is constructed from composing flattening layers, and the decoder $g = g_1 \circ g_2 \circ \cdots \circ g_L$ is composed of inversions of each f_ℓ.

In light of universal approximation theorems such as Cybenko's, one may initially wonder why, conceptually, deeper autoencoders should be preferred over shallow ones. From purely an expressivity perspective, we can understand this phenomenon through a geometric angle related to the task of flattening the nonlinear manifold on which our hypothesized data distribution is supported. A purely constructive approach to flattening the manifold proceeds incrementally, in parallel to what we have seen in Chapters 3 to 5 with the interaction between diffusion, denoising, and compression. In the geometric setting, the incremental *flattening process* corresponding to f_ℓ takes the form of transforming a neighborhood of one point of the manifold to be closer to a flat manifold (i.e., a subspace), and enforcing local consistency with the rest of the data samples; the corresponding incremental operation in the decoder, g_ℓ, undoes this transformation. This procedure precisely incorporates curvature information about the underlying manifold, which is estimated from data samples. Given enough samples from the manifold and a careful instantiation of this conceptual process, it is possible to implement this sketch as a computational procedure that verifiably flattens nonlinear manifolds in a white-box fashion [PPR+24]. However, the approach is limited in its applicability to high-dimensional data distributions such as images due to unfavorable scalability, motivating the development of more flexible methods for incremental autoencoding.

6.1.3 Sparse Autoencoding

In the above autoencoding schemes, the dimension of the feature space d is typically chosen to be much lower than that of the original data space D in order to

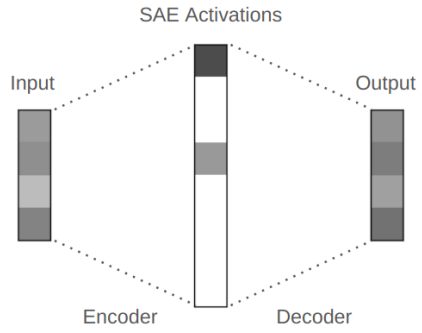


Figure 6.6: Illustration of a sparse autoencoder (SAE), compared to that of a typical autoencoder (AE) in Figure 6.2.

explicitly enforce or promote the learned representation to be low-dimensional. However, in practice, we normally do not know the intrinsic dimension of the data distribution. Hence, the choice of the feature space dimension for autoencoding is often done empirically. In more general situations, the data distribution can be a mixture of a few low-dimensional subspaces or submanifolds. In these cases, it is no longer feasible to enforce a single low-dimensional space for all features together.

The sparse autoencoder is meant to resolve some of these limitations. In particular, the dimension of the feature space can be equal to or even higher than that of the data space, as illustrated in Figure 6.6. However, the features are required to be highly sparse in the feature space. So if we impose sparsity as a measure of parsimony in addition to rate reduction in the objective (6.1.5), we obtain a new objective for sparse autoencoding:

$$\min_{f,g}[\|\boldsymbol{Z}\|_0 - \Delta R_\epsilon(\boldsymbol{Z}) + d(\boldsymbol{X}, \hat{\boldsymbol{X}})], \tag{6.1.11}$$

where the ℓ^0-"norm" $\|\cdot\|_0$ is known to promote sparsity. This is very similar to the sparse rate reduction objective (5.2.4) which we used in the previous Chapter 5 to derive the white-box CRATE architecture.

As a method for learning autoencoding pairs in an end-to-end fashion, sparse autoencoding has been practiced in the past [LRM+12; RPC+06], but nearly all modern autoencoding frameworks are instead based on a different, probabilistic autoencoding framework, which we will study now.

6.1.4 Variational Autoencoding

An improved, more modern methodology for autoencoding that still finds significant application to this day is *variational autoencoding* [KW13a; KW19]. It formulates the representation learning problem in probabilistic terms, as the search for a model $p(\boldsymbol{x}; \boldsymbol{\theta})$ that maximizes the likelihood of a data sample (or

more generally, the population data distribution):

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}}[\log p(\boldsymbol{x};\, \boldsymbol{\theta})].$$

By conditioning we may write $p(\boldsymbol{x}, \boldsymbol{z};\, \boldsymbol{\theta}) = p(\boldsymbol{z};\, \boldsymbol{\theta})p(\boldsymbol{x} \mid \boldsymbol{z};\, \boldsymbol{\theta})$, and

$$\begin{aligned} p(\boldsymbol{x};\boldsymbol{\theta}) &= \int p(\boldsymbol{z};\, \boldsymbol{\theta})p(\boldsymbol{x} \mid \boldsymbol{z};\, \boldsymbol{\theta})\mathrm{d}\boldsymbol{z} \\ &= \mathbb{E}_{\boldsymbol{z}\sim p(\,\cdot\,;\boldsymbol{\theta})}[p(\boldsymbol{x} \mid \boldsymbol{z};\, \boldsymbol{\theta})]. \end{aligned}$$

This implies that a probabilistic model for $\boldsymbol{x}$ can be learned with a *prior* $p(\boldsymbol{z})$ and a conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{z})$, corresponding more generally to a decoder. Technically, Bayes' rule then implies that the *posterior* $p(\boldsymbol{z} \mid \boldsymbol{x})$, necessary for encoding, can be expressed in terms of the prior and the decoder (and samples of $\boldsymbol{x}$), but this is computationally intractable. Hence, in the VAE framework, we introduce an "approximate posterior" $q(\boldsymbol{z} \mid \boldsymbol{x};\, \boldsymbol{\eta})$, corresponding to a separate encoder network, and jointly learn the encoder and decoder through the maximization of a lower bound on the true likelihood, known as the evidence lower bound (ELBO), which is tight when the posterior approximation is tight.

The following instantiation is standard in practice. We use the following Gaussian distributions:

$$\begin{aligned} \boldsymbol{z} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \\ \boldsymbol{x} \mid \boldsymbol{z} &\sim \mathcal{N}(g_1(\boldsymbol{z}), \mathrm{diag}(e^{g_2(\boldsymbol{z})})\boldsymbol{I}), \end{aligned}$$

where $g = (g_1, g_2)$ are deep networks with parameters $\boldsymbol{\theta}$, which correspond to the *decoder* in the autoencoder. Similarly, for the approximate posterior, we use a Gaussian distribution with parameters given by an encoder MLP $f = (f_1, f_2)$ with parameters $\boldsymbol{\eta}$:

$$\boldsymbol{z} \mid \boldsymbol{x} \sim \mathcal{N}(f_1(\boldsymbol{x}), \mathrm{diag}(e^{f_2(\boldsymbol{x})})\boldsymbol{I}).$$

This makes probabilistic encoding and decoding simple: we simply map our data $\boldsymbol{x}$ to the mean and variance parameters of a Gaussian distribution for encoding, and vice versa for decoding. After the aforementioned manipulations, the final VAE objective reads:

$$\max_{\boldsymbol{\theta},\boldsymbol{\eta}} \mathbb{E}_{\boldsymbol{x}}\mathbb{E}_{\boldsymbol{z}\sim q(\,\cdot\,|\boldsymbol{x};\,\boldsymbol{\eta})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z};\, \boldsymbol{\theta})}{q(\boldsymbol{z} \mid \boldsymbol{x};\, \boldsymbol{\eta})}\right] \tag{6.1.12}$$

$$\equiv \max_{\boldsymbol{\theta},\boldsymbol{\eta}} \left\{2\mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\boldsymbol{z}\sim q(\,\cdot\,|\boldsymbol{x};\,\boldsymbol{\eta})}\left[\log p(\boldsymbol{x} \mid \boldsymbol{z};\, \boldsymbol{\theta})\right] + \langle f_2(\boldsymbol{x}) - e^{f_2(\boldsymbol{x})}, \mathbf{1}\rangle - \|f_1(\boldsymbol{x})\|_2^2\right]\right\}. \tag{6.1.13}$$

We can understand this as a regularized autoencoding objective, where there is an interesting additional consistency enforcement in the first term of the ELBO.

At the same time, the reader should note the numerous arbitrary decisions the VAE formalism makes (e.g., the choice and parameterization of the Gaussian distributions). This leads to a complex training objective that remains a

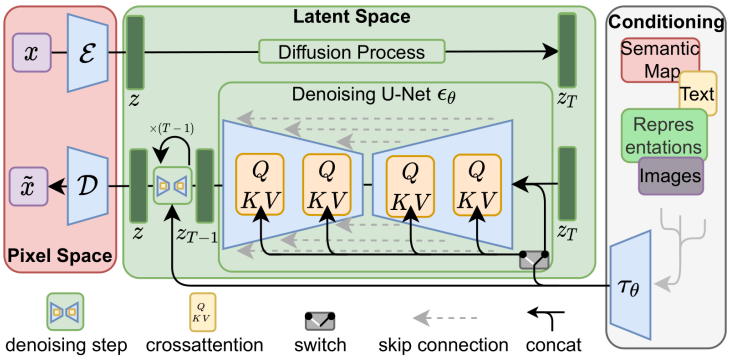


Figure 6.7: Diagram of the Latent Diffusion Model from the work of [RBL+22] which utilizes a variational autoencoder (red box on the left) to learn a latent representation of the data $\boldsymbol{x}$. Then a diffusion/denoising model (green box in the middle) is learned to access the representation $\boldsymbol{z}$. The block on the right is about how to control the denoising sampling process via a (text-based) control signal which we will study in Chapter 7.

challenge to optimize efficiently despite more than ten years of research. Recent representation learning methodologies have made notable improvements to this baseline, and they can even be formulated in the white-box framework we have developed in Chapter 5, as we will soon see.

VAE-Based Latent Diffusion. An important application of VAEs is in the context of latent diffusion models [RBL+22], where a VAE is first trained to learn a more compact and regulated latent representation of high-dimensional data such as images. Then a diffusion model can be trained in the latent space to access the distribution of the representation and hence generate new samples from it, as illustrated in Figure 6.7. This approach enables efficient generation of high-quality images by leveraging the highly-compressed structure of the learned latent space.[6] However, training VAEs can be challenging due to issues such as posterior collapse and balancing the reconstruction and regularization terms in the ELBO. Moreover, the Gaussian posteriors in VAEs are applied per data point and no joint structure is learned across samples, which can severely limit the quality of the learned representations. These challenges have motivated the development of alternative approaches, such as representation autoencoders (RAEs) to be introduced next, which aim to address some of the limitations of VAEs while still providing effective latent representations for generative modeling tasks.

[6] Many popular early image or video generative models are based on this latent diffusion framework, as we describe in further detail in Section 7.4.2 and in implementation terms in Chapter 8.

6.1.5 Joint Representation and Distribution Learning

The previous approaches to training autoencoding pairs for representation learning share a common limitation: they do not *explicitly organize* the representation space $\boldsymbol{Z}$, but only encourage it indirectly (e.g., by enforcing sparsity in the sparse autoencoder or Gaussianity in VAE). As we hinted at the start of the chapter, we have seen previously (in Chapter 4) how to do much better, by learning a representation $\boldsymbol{Z}$ via maximizing the rate reduction $\Delta R_\epsilon(\boldsymbol{Z})$ while ensuring reconstruction consistency. We describe this procedure in more detail in this section, in particular also demonstrating how to *efficiently sample* in the learned representation space $\boldsymbol{Z}$.

As we have seen at the end of Chapter 4 (Section 4.3), we often can learn a good representation $\boldsymbol{Z}$ of the data $\boldsymbol{X}$:

$$f : \boldsymbol{X} \to \boldsymbol{Z}, \tag{6.1.14}$$

based on some end-to-end supervised or self-supervised methods, including the MCR^2 and SimDINO methods introduced in Section 4.3.1 and Section 4.3.2, respectively. As we have also argued, the so-learned representations have better structures than the original data distribution. The representations tend to be a mixture of low-dimensional linear subspaces (or low-rank Gaussians). However, we have not answered the question of how to assess the goodness of the so-learned representations. That is, can we establish from them a reverse decoder:

$$g : \boldsymbol{Z} \to \hat{\boldsymbol{X}} \tag{6.1.15}$$

such that $\hat{\boldsymbol{X}}$ and $\boldsymbol{X}$ are close? In light of the preceding examples of classical autoencoding setups, we should expect that an objective of the form (6.1.5) should work—but what sample-wise consistency loss should be used?

Furthermore, those low-dimensional (linear) structures are still embedded in a high-dimensional representation space. Hence, in order to use the so-learned representations, we need to be able to access their distributions too. In Chapter 3, we have learned that denoising is a general approach to identify and hence access such low-dimensional representations. Hence, we can access the distribution of the learned features via a (learned) denoising process from a random noise:

$$h : \boldsymbol{G} \to \boldsymbol{Z}. \tag{6.1.16}$$

In addition, we have also learned that if the target distribution of $\boldsymbol{Z}$ can be modeled (approximately) as a mixture of subspaces or low-rank Gaussians, their optimal denoising operators share a similar structure to those found in Transformers, or in CRATE introduced in Chapter 5.

Thus, altogether, we want to learn an autoencoder for the data $\boldsymbol{X}$ whose representation $\boldsymbol{Z}$ can be easily accessed. Concretely, we would like to learn an encoder f and a decoder g such that:

1. The representation $\boldsymbol{Z} = f(\boldsymbol{X})$ is a consistent representation of the original data $\boldsymbol{X}$ as defined in Definition 6.1.

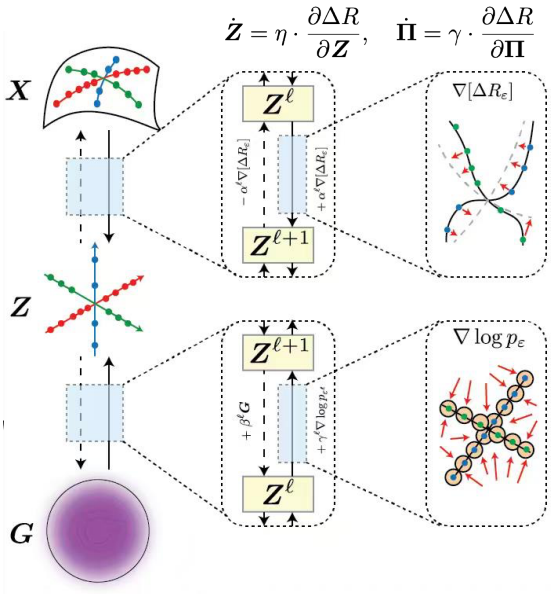


Figure 6.8: Illustration of learning an autoencoding representation $\boldsymbol{Z}$ for the data $\boldsymbol{X}$ (top) and a diffusion/denoising model (bottom) to access the representation $\boldsymbol{Z}$.

2. The structured representation $\boldsymbol{Z}$ can be modeled approximately as a mixture of low-dimensional Gaussians so that we can learn a denoiser to efficiently access the distribution of $\boldsymbol{Z}$ from random Gaussian noise $\boldsymbol{G}$.

Figure 6.8 illustrates the above overall processes.

From Chapter 4, we know that the learned representation $\boldsymbol{Z}$ by optimizing the (unsupervised) MCR^2 objective satisfies the second requirement and can be viewed to have structures close to a mixture of subspaces or low-rank Gaussians. By incorporating a sample-consistency objective, we can extend the rate reduction objective in Section 4.3.2 to learn a consistent representation:

$$\min_{f,g}[-\Delta R_\epsilon(\boldsymbol{Z}) + d(\boldsymbol{X}, \hat{\boldsymbol{X}})], \tag{6.1.17}$$

where $d(\boldsymbol{X}, \hat{\boldsymbol{X}})$ is a measure of similarity between the original data $\boldsymbol{X}$ and the reconstructed data $\hat{\boldsymbol{X}} = g(\boldsymbol{Z})$. In practice, because the objective $\Delta R_\epsilon(\boldsymbol{Z})$ alone already promotes low-dimensional structure of $\boldsymbol{Z}$, we can pretrain the encoder f to maximize $\Delta R_\epsilon(\boldsymbol{Z})$ first, and then learn the decoder g to minimize the reconstruction loss $d(\boldsymbol{X}, \hat{\boldsymbol{X}})$ while fixing the encoder f. This two-step approach has been demonstrated to be effective, as we will see in Section 8.6.2 of Chapter 8.

Once we have learned such a consistent representation $\boldsymbol{Z}$, we can then learn a denoiser to access its distribution, as we have studied in Chapter 3. Concretely,

suppose we have a probabilistic model $p(\boldsymbol{z})$ that captures the (marginal) distribution of the learned representations $\boldsymbol{Z}$ and an initial standard Gaussian distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Our goal is to learn a denoiser $\bar{\boldsymbol{z}}$ such that it iteratively denoises a sample from the initial distribution to the target distribution of the learned representations. This denoiser can be trained using objectives introduced in Chapter 3 (see also Chapter 7 for conditional denoisers). For instance, take the simple case of flow matching (recall Exercise 3.5 and Section 3.2.2). We can define the denoiser as a time-dependent vector field $\bar{\boldsymbol{z}}_\theta(t, \boldsymbol{z}_t)$ parameterized by θ that transforms a sample from the initial distribution to the target distribution over time $t \in [0, 1]$. Here, $\boldsymbol{z}_t$ is an interpolation between $\boldsymbol{z}_0$ and $\boldsymbol{z}_1$ at time t:

$$\boldsymbol{z}_t = (1 - t)\boldsymbol{z}_0 + t\boldsymbol{z}_1, \tag{6.1.18}$$

where $\boldsymbol{z}_0 \sim p(\boldsymbol{z})$ is a sample from the learned representation distribution and $\boldsymbol{z}_1 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The training objective for the denoiser can then be expressed as:

$$\min_\theta \mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{\boldsymbol{z}_0 \sim p(\boldsymbol{z}), \boldsymbol{z}_1 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), t \sim \text{Unif}(0,1)} \left[\| \bar{\boldsymbol{z}}_\theta(t, \boldsymbol{z}_t) - (\boldsymbol{z}_1 - \boldsymbol{z}_0) \|_2^2 \right]. \tag{6.1.19}$$

It is worth noting the difference between learning a denoiser for the representations learned via VAE and those learned via methods like MCR^2 or SimDINO. As discussed in Section 6.1.4, the VAE representations are enforced to approximate a Gaussian posterior *per sample* and no joint distribution of the latent space is modeled, making the representation space less structured and the denoising process potentially more challenging. In contrast, representations learned through MCR^2 or SimDINO tend to have more pronounced linear and low-dimensional structures, such as mixtures of subspaces or low-rank Gaussians (as demonstrated in Section 4.3.2), which can facilitate the denoising process. This structural property can lead to more efficient and effective denoising models, as they can better exploit the underlying geometry of the representation space.

Once we have trained the denoiser, we can use it to generate new samples from the learned representation distribution via a diffusion process, as we have shown in Chapter 3. This completes the learning of a *representation autoencoder* (RAE) with an accessible representation distribution. In Chapter 8 (specifically Section 8.6.2), we will see a specific implementation of RAE and its applications in image generation.

6.2 Learning Self-Consistent Representations

In earlier chapters, we have studied methods that would allow us to learn a low-dimensional distribution via (lossy) compression. As we have mentioned in Chapter 1 and demonstrated in the previous chapters, the progress made in machine intelligence largely relies on finding computationally feasible and efficient solutions to realize the desired compression, not only computable or tractable in theory, but also scalable in practice:

$$\textbf{computable} \implies \textbf{tractable} \implies \textbf{scalable}. \tag{6.2.1}$$

It is even fair to say that the tremendous advancement in machine intelligence made in the past decade or so is largely attributed to the development of scalable models and methods, say by training deep networks via back propagation. Large models with billions of parameters trained with trillions of data points on tens of thousands of powerful GPUs have demonstrated super-human capabilities in memorizing existing knowledge. This has led many to believe that the "intelligence" of such models will continue to improve as their scale continues to go up.

While we celebrate the engineering marvels of such large man-made machine learning systems, we also must admit that, compared to intelligence in nature, this approach to improving machine intelligence is unnecessarily resource-demanding. Natural intelligent beings, including animals and humans, simply cannot afford such a brute-force solution for learning because they must operate with a very limited budget in energy, space, and time, subject to many strict physical constraints.

First, there is strong scientific evidence that our brain does not conduct global end-to-end back propagation to improve or correct its predictions. Instead, it has long been known in neuroscience that our brain corrects errors with local closed-loop feedback, such as predictive coding. This was the scientific basis that inspired Norbert Wiener to develop the theory of feedback control and the Cybernetics program back in the 1940s.

Second, we saw in the previous sections that in order to learn a consistent representation, one needs to learn a bidirectional autoencoding:

$$\boldsymbol{X} \xrightarrow{\mathcal{E}=f} \boldsymbol{Z} \xrightarrow{\mathcal{D}=g} \hat{\boldsymbol{X}}. \tag{6.2.2}$$

It requires enforcing the observed input data $\boldsymbol{X}$ and the decoded $\hat{\boldsymbol{X}}$ to be close by some measure of similarity $d(\boldsymbol{X}, \hat{\boldsymbol{X}})$. However, when the data $\boldsymbol{X}$ is high-dimensional (e.g., images), naive distance measures such as the ℓ^2 norm in ambient space are largely uninformative: two images that differ by a small geometric transformation may be far apart in pixel distance yet perceptually identical. Hence, an outstanding question is: how can we enforce similarity between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ without a meaningful distance in the ambient data space? As we will see in this chapter, the answer lies in closed-loop feedback in a learned feature space, together with the low-dimensionality of the data distribution.

As we know from the previous chapter, in order to ensure that a representation is consistent, we need to compare the generated $\hat{\boldsymbol{X}} \sim p(\hat{\boldsymbol{x}})$ and the original $\boldsymbol{X} \sim p(\boldsymbol{x})$, at least in distribution. Even when we do have access to $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$, technically, computing and minimizing the distance between two distributions can be problematic, especially when the support of the distributions is low-dimensional. The KL-divergence introduced in Chapter 3 is not even well-defined between two distributions that do not have overlapping supports.

As an early attempt to alleviate the above difficulty in computing and minimizing the distance between two (low-dimensional) distributions, people had suggested learning the generator/decoder g via discriminative approaches [Tu07]. This line of thought has led to the idea of *Generative Adversarial Nets*

(GAN) [GPM+14b]. It introduces a discriminator d, usually modeled by a deep network, to discern differences between the generated samples $\hat{\boldsymbol{X}}$ and the real ones $\boldsymbol{X}$:

$$\boldsymbol{Z} \xrightarrow{g(\boldsymbol{z},\eta)} \hat{\boldsymbol{X}},\ \boldsymbol{X} \xrightarrow{d(\boldsymbol{x},\theta)} \{\boldsymbol{0},\boldsymbol{1}\}. \tag{6.2.3}$$

It is suggested that we may attempt to align the distributions between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$ via a *Stackelberg game*—a sequential game in which one player (the leader) commits to a strategy first and the other (the follower) responds optimally (see Section A.3 for a detailed treatment)—between the generator g and the discriminator d:

$$\max_{\theta}\min_{\eta} \mathbb{E}_{p(\boldsymbol{x})}\big[\log d(\boldsymbol{x},\theta)\big] + \mathbb{E}_{p(\boldsymbol{z})}\big[\log\big(1 - d(\underbrace{g(\boldsymbol{z},\eta)}_{\hat{\boldsymbol{x}}\sim p_g},\theta)\big)\big]. \tag{6.2.4}$$

That is, the discriminator d is trying to minimize the cross entropy between the true samples $\boldsymbol{X}$ and the generated ones $\hat{\boldsymbol{X}}$ while the generator g is trying to do the opposite.

One may show that finding an equilibrium for the above Stackelberg game is equivalent to minimizing the *Jensen-Shannon divergence*:

$$\mathcal{D}_{JS}(p(\boldsymbol{x}), p_g(\hat{\boldsymbol{x}})) = \mathcal{D}_{KL}\big(p\|(p+p_g)/2\big) + \mathcal{D}_{KL}\big(p_g\|(p+p_g)/2\big). \tag{6.2.5}$$

Note that, compared to the KL-divergence, the JS-divergence is well-defined even if the supports of the two distributions are non-overlapping. However, JS-divergence does not have a closed-form expression even between two Gaussians, whereas KL-divergence does. In addition, since the data distributions are low-dimensional, the JS-divergence can be highly ill-conditioned to optimize.[7] This may explain why many additional heuristics are typically used in many subsequent variants of GAN.

So, instead, it has also been suggested to replace JS-divergence with the earth mover (EM) distance or the Wasserstein distance.[8] However, both the JS-divergence and Wasserstein distance can only be approximately computed between two general distributions.[9] Furthermore, neither the JS-divergence nor the Wasserstein distance has closed-form formulae, even for Gaussian distributions.[10]

If it is difficult to compare distributions of the data $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$, it may be possible to compare the learned feature $\boldsymbol{Z}$ with its image $\hat{\boldsymbol{Z}}$ under the encoder f:

$$\boldsymbol{X} \xrightarrow{\mathcal{E}=f} \boldsymbol{Z} \xrightarrow{\mathcal{D}=g} \hat{\boldsymbol{X}} \xrightarrow{\mathcal{E}=f} \hat{\boldsymbol{Z}} \tag{6.2.6}$$

[7] As shown in [ACB17].

[8] Roughly speaking, for distributions with potentially non-overlapping low-dimensional supports, the JS-divergence behaves like the ℓ^0-norm, and the EM-distance behaves like the ℓ^1-norm.

[9] For instance, the Wasserstein distance requires one to compute the maximal difference between expectations of the two distributions over all 1-Lipschitz functions.

[10] The (ℓ^1-norm) Wasserstein distance can be bounded by the (ℓ^2-norm) Wasserstein distance which has a closed-form [DDS22]. However, as is well-known in high-dimensional geometry, ℓ^1-norm and ℓ^2 norm deviate significantly in terms of their geometric and statistical properties as the dimension becomes high [WM22]. The bound can become very loose.

This leads to the notion of *self-consistent representation*.

Definition 6.2 (Self-Consistent Representation)**.** Given data $\boldsymbol{X}$, we call a *self-consistent representation* a pair of functions $(f\colon \mathcal{X} \to \mathcal{Z}, g\colon \mathcal{Z} \to \mathcal{X})$, such that the *features* $\boldsymbol{Z} = f(\boldsymbol{X})$ are compact and structured, and the *autoencoding features* $\hat{\boldsymbol{Z}} \doteq f \circ g(\boldsymbol{Z})$ are *close* to $\boldsymbol{Z}$.

1. We say that it is *sample-wise* self-consistent if $\boldsymbol{Z} \approx \hat{\boldsymbol{Z}}$ in a certain norm with high probability.
2. We say that the representation is *distributionally self-consistent* if $\text{Law}(\boldsymbol{Z}) \approx \text{Law}(\hat{\boldsymbol{Z}})$.

6.2.1 Closed-Loop Transcription via Stackelberg Games

How do we try to ensure a learned representation is self-consistent? As usual, let us assume $\boldsymbol{X} = \cup_{k=1}^{K} \boldsymbol{X}_k$ with each subset of samples $\boldsymbol{X}_k$ belonging to a low-dimensional submanifold: $\boldsymbol{X}_k \subset \mathcal{M}_k, k = 1, \ldots, K$. Following the notation in Chapter 3, we use a matrix $\boldsymbol{\Pi}_k(i,i) = 1$ to denote the membership of sample i belonging to class k and $\boldsymbol{\Pi}_k(i,j) = 0$ otherwise. One seeks a continuous mapping $f(\cdot, \boldsymbol{\theta}) : \boldsymbol{x} \mapsto \boldsymbol{z}$ from $\boldsymbol{X}$ to an optimal representation $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N] \subset \mathbb{R}^{d \times N}$:

$$\boldsymbol{X} \xrightarrow{\quad f(\boldsymbol{x}, \boldsymbol{\theta}) \quad} \boldsymbol{Z}, \tag{6.2.7}$$

which maximizes the following coding rate-reduction (MCR^2) objective:

$$\max_{\boldsymbol{Z}} \Delta R_\epsilon(\boldsymbol{Z} \mid \boldsymbol{\Pi}) \doteq \underbrace{\frac{1}{2} \log\det\Big(\boldsymbol{I} + \alpha \boldsymbol{Z}\boldsymbol{Z}^\top\Big)}_{R_\epsilon(\boldsymbol{Z})} - \underbrace{\sum_{k=1}^{K} \frac{\gamma_k}{2} \log\det\Big(\boldsymbol{I} + \alpha_k \boldsymbol{Z}\boldsymbol{\Pi}^k\boldsymbol{Z}^\top\Big)}_{R_\epsilon^c(\boldsymbol{Z}\mid\boldsymbol{\Pi})}, \tag{6.2.8}$$

where $\alpha = d/(N\epsilon^2)$, $\alpha_k = d/(\text{tr}(\boldsymbol{\Pi}_k)\epsilon^2)$, $\gamma_k = \text{tr}(\boldsymbol{\Pi}_k)/N$ for each $k = 1, \ldots, K$.

One issue with learning such a one-sided mapping (6.2.7) via maximizing the above objective (6.2.8) is that it tends to expand the dimension of the learned subspace for features in each class[11], as we have seen in Section 4.2 of Chapter 4. To verify whether the learned features are consistent, meaning neither over-estimating nor under-estimating the intrinsic data structure, we may consider learning a decoder $g(\cdot, \eta) : \boldsymbol{z} \mapsto \boldsymbol{x}$ from the representation $\boldsymbol{Z} = f(\boldsymbol{X}, \boldsymbol{\theta})$ back to the data space $\boldsymbol{x}$: $\hat{\boldsymbol{X}} = g(\boldsymbol{Z}, \boldsymbol{\eta})$:

$$\boldsymbol{X} \xrightarrow{\quad f(\boldsymbol{x}, \boldsymbol{\theta}) \quad} \boldsymbol{Z} \xrightarrow{\quad g(\boldsymbol{z}, \boldsymbol{\eta}) \quad} \hat{\boldsymbol{X}}, \tag{6.2.9}$$

[11] If the dimension of the feature space d is too high, maximizing the rate reduction may over-estimate the dimension of each class. Hence, to learn a good representation, one needs to pre-select a proper dimension for the feature space, as achieved in the experiments in [YCY+20]. In fact the same "model selection" problem persists even in the simplest single-subspace case, which is the classic PCA [Jol86]. To our best knowledge, selecting the correct number of principal components in a heterogeneous noisy situation remains an active research topic [HSD20].

and check how close $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ are. This forms an autoencoding which is what we have studied extensively earlier in this chapter.

Measuring distance in the feature space. However, as we have discussed above, if we do not have the option to compute the distance between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$, we are left with the option of comparing their corresponding features $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}} = f(\hat{\boldsymbol{X}}, \boldsymbol{\theta})$. Notice that under the MCR^2 objective, the distributions of the resulting $\boldsymbol{Z}$ or $\hat{\boldsymbol{Z}}$ tend to be piecewise linear so their distance can be computed more easily. More precisely, since the features of each class, $\boldsymbol{Z}_k$ and $\hat{\boldsymbol{Z}}_k$, are close to being a low-dimensional subspace/Gaussian, their "distance" can be measured by the rate reduction, with (6.2.8) restricted to two sets of equal size:

$$\Delta R_\epsilon\big(\boldsymbol{Z}_k, \hat{\boldsymbol{Z}}_k\big) \doteq R_\epsilon\big(\boldsymbol{Z}_k \cup \hat{\boldsymbol{Z}}_k\big) - \frac{1}{2}\big(R_\epsilon(\boldsymbol{Z}_k) + R_\epsilon(\hat{\boldsymbol{Z}}_k)\big). \tag{6.2.10}$$

The overall distance between $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$ is given by:

$$d(\boldsymbol{Z}, \hat{\boldsymbol{Z}}) \doteq \sum_{k=1}^{K} \Delta R_\epsilon\big(\boldsymbol{Z}_k, \hat{\boldsymbol{Z}}_k\big) = \sum_{k=1}^{K} \Delta R_\epsilon\big(\boldsymbol{Z}_k, f(g(\boldsymbol{Z}_k, \boldsymbol{\eta}), \boldsymbol{\theta})\big). \tag{6.2.11}$$

If we are interested in discerning *any* differences in the distributions of the original data $\boldsymbol{X}$ and their decoded $\hat{\boldsymbol{X}}$, we may view the feature encoder $f(\cdot, \boldsymbol{\theta})$ as a "discriminator" to *magnify* any difference between all pairs of $\boldsymbol{X}_k$ and $\hat{\boldsymbol{X}}_k$, by simply maximizing the distance $d(\boldsymbol{Z}, \hat{\boldsymbol{Z}})$:

$$\max_f d(\boldsymbol{Z}, \hat{\boldsymbol{Z}}) = \max_{\boldsymbol{\theta}} \sum_{k=1}^{K} \Delta R_\epsilon\big(\boldsymbol{Z}_k, \hat{\boldsymbol{Z}}_k\big) = \max_{\boldsymbol{\theta}} \sum_{k=1}^{K} \Delta R_\epsilon\big(f(\boldsymbol{X}_k, \boldsymbol{\theta}), f(\hat{\boldsymbol{X}}_k, \boldsymbol{\theta})\big). \tag{6.2.12}$$

That is, a "distance" between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ can be measured as the maximally achievable rate reduction between all pairs of classes in these two sets. In a way, this measures how well or badly the decoded $\hat{\boldsymbol{X}}$ aligns with the original data $\boldsymbol{X}$—hence measuring the goodness of "injectivity" of the encoder f. Notice that such a discriminative measure is consistent with the idea of GAN (6.2.3) that tries to separate $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ into two classes, measured by the cross-entropy.

Nevertheless, here the discriminative encoder f naturally generalizes to cases when the data distributions are multi-class and multi-modal, and the discriminativeness is measured with a more refined measure—the rate reduction—instead of the typical two-class loss (e.g., cross entropy) used in GANs. That is, we may view the encoder f as a generalized discriminator that replaces the binary classifier d in (6.2.3):

$$\boldsymbol{Z} \xrightarrow{\;g(\boldsymbol{z},\boldsymbol{\eta})\;} \hat{\boldsymbol{X}},\ \boldsymbol{X} \xrightarrow{\;f(\boldsymbol{x},\boldsymbol{\theta})\;} \{\hat{\boldsymbol{Z}}, \boldsymbol{Z}\}. \tag{6.2.13}$$

The overall pipeline can be illustrated by the following "closed-loop" diagram:

$$\boldsymbol{X} \xrightarrow{\;f(\boldsymbol{x},\boldsymbol{\theta})\;} \boldsymbol{Z} \xrightarrow{\;g(\boldsymbol{z},\boldsymbol{\eta})\;} \hat{\boldsymbol{X}} \xrightarrow{\;f(\boldsymbol{x},\boldsymbol{\theta})\;} \hat{\boldsymbol{Z}}, \tag{6.2.14}$$

where the overall model has parameters: $\boldsymbol{\Theta} = \{\boldsymbol{\theta}, \boldsymbol{\eta}\}$. Figure 6.9 shows the overall process.

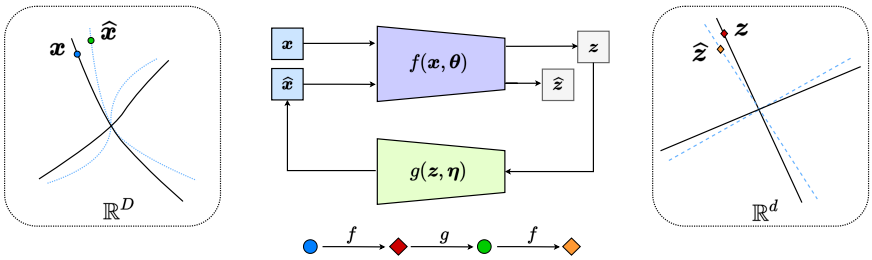


Figure 6.9: **A Closed-loop Transcription.** The encoder f has dual roles: it learns a representation $\boldsymbol{z}$ for the data $\boldsymbol{x}$ via maximizing the rate reduction of $\boldsymbol{z}$ and it is also a "feedback sensor" for any discrepancy between the data $\boldsymbol{x}$ and the decoded $\hat{\boldsymbol{x}}$. The decoder g also has dual roles: it is a "controller" that corrects the discrepancy between $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ and it also aims to minimize the overall coding rate for the learned representation.

Encoder and decoder as a two-player game. Obviously, to ensure the learned autoencoding is self-consistent, the main goal of the decoder $g(\cdot,\boldsymbol{\eta})$ is to *minimize* the distance between $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$. That is, to learn g, we want to minimize the distance $d(\boldsymbol{Z},\hat{\boldsymbol{Z}})$:

$$\min_g d(\boldsymbol{Z},\hat{\boldsymbol{Z}}) \doteq \min_{\boldsymbol{\eta}} \sum_{k=1}^{K} \Delta R_\epsilon\big(\boldsymbol{Z}_k,\hat{\boldsymbol{Z}}_k\big) = \min_{\boldsymbol{\eta}} \sum_{k=1}^{K} \Delta R_\epsilon\big(\boldsymbol{Z}_k, f(g(\boldsymbol{Z}_k,\boldsymbol{\eta}),\boldsymbol{\theta})\big), \tag{6.2.15}$$

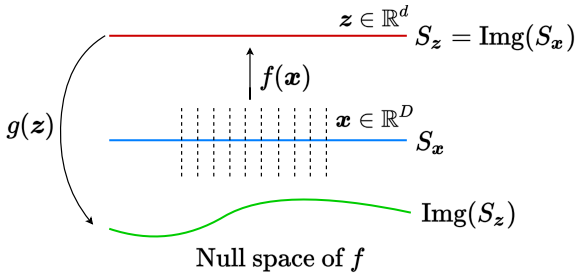
where $\boldsymbol{Z}_k = f(\boldsymbol{X}_k,\boldsymbol{\theta})$ and $\hat{\boldsymbol{Z}}_k = f(\hat{\boldsymbol{X}}_k,\boldsymbol{\theta})$.

Example 6.1. One may wonder why we need the mapping $f(\cdot,\boldsymbol{\theta})$ to function as a discriminator between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ by maximizing $\max_{\boldsymbol{\theta}} \Delta R_\epsilon\big(f(\boldsymbol{X},\boldsymbol{\theta}), f(\hat{\boldsymbol{X}},\boldsymbol{\theta})\big)$. Figure 6.10 gives a simple illustration: there might be many decoders g such that $f \circ g$ is an identity (Id) mapping. $f \circ g(\boldsymbol{z}) = \boldsymbol{z}$ for all $\boldsymbol{z}$ in the subspace $S_{\boldsymbol{z}}$ in the feature space. However, $g \circ f$ is not necessarily an autoencoding map for $\boldsymbol{x}$ in the original distribution $S_{\boldsymbol{x}}$ (here drawn as a subspace for simplicity). That is, $g \circ f(S_{\boldsymbol{x}}) \not\subset S_{\boldsymbol{x}}$, let alone $g \circ f(S_{\boldsymbol{x}}) = S_{\boldsymbol{x}}$ or $g \circ f(\boldsymbol{x}) = \boldsymbol{x}$. One should expect that, without careful control of the image of g, this would be the case with high probability, especially when the support of the distribution of $\boldsymbol{x}$ is extremely low-dimensional in the original high-dimensional data space. ■

Comparing the contractive and contrastive nature of (6.2.15) and (6.2.12) on the same distance measure, we see the roles of the encoder $f(\cdot,\boldsymbol{\theta})$ and the decoder $g(\cdot,\boldsymbol{\eta})$ naturally as "**a two-player game**": *while the encoder f tries to magnify the difference between the original data and their transcribed data, the decoder g aims to minimize the difference.* Now for convenience, let us define the "closed-loop encoding" function:

$$h(\boldsymbol{x},\boldsymbol{\theta},\boldsymbol{\eta}) \doteq f\big(g\big(f(\boldsymbol{x},\boldsymbol{\theta}),\boldsymbol{\eta}\big),\boldsymbol{\theta}\big) : \ \boldsymbol{x} \mapsto \hat{\boldsymbol{z}}. \tag{6.2.16}$$

Ideally, we want this function to be very close to $f(\boldsymbol{x},\boldsymbol{\theta})$ or at least the distributions of their images should be close. With this notation, combining



$$g \circ f \neq \text{Id}, \text{ but } f \circ g = \text{Id}$$

Figure 6.10: **Embeddings of low-dim submanifolds in a high-dim space.** $S_{\boldsymbol{x}}$ (blue) is the submanifold for the original data $\boldsymbol{x}$; $S_{\boldsymbol{z}}$ (red) is the image of $S_{\boldsymbol{x}}$ under the mapping f, representing the learned feature $\boldsymbol{z}$; and the green curve is the image of the feature $\boldsymbol{z}$ under the decoding mapping g.

(6.2.15) and (6.2.12), a closed-loop notion of "distance" between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ can be computed as *an equilibrium point* to the following Stackelberg game (cf. Section A.3) for the same utility in terms of rate reduction:

$$\mathcal{D}(\boldsymbol{X}, \hat{\boldsymbol{X}}) \doteq \max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} \sum_{k=1}^{K} \Delta R_\epsilon\big(f(\boldsymbol{X}_k, \boldsymbol{\theta}), h(\boldsymbol{X}_k, \boldsymbol{\theta}, \boldsymbol{\eta})\big). \tag{6.2.17}$$

Notice that this only measures the difference between (features of) the original data and its transcribed version. It does not measure how good the representation $\boldsymbol{Z}$ (or $\hat{\boldsymbol{Z}}$) is for the multiple classes within $\boldsymbol{X}$ (or $\hat{\boldsymbol{X}}$). To this end, we may combine the above distance with the original MCR^2-type objectives (6.2.8): namely, the rate reduction $\Delta R_\epsilon(\boldsymbol{Z})$ and $\Delta R_\epsilon(\hat{\boldsymbol{Z}})$ for the learned LDR $\boldsymbol{Z}$ for $\boldsymbol{X}$ and $\hat{\boldsymbol{Z}}$ for the decoded $\hat{\boldsymbol{X}}$. Notice that although the encoder f tries to *maximize* the multi-class rate reduction of the features $\boldsymbol{Z}$ of the data $\boldsymbol{X}$, the decoder g should *minimize* the rate reduction of the multi-class features $\hat{\boldsymbol{Z}}$ of the decoded $\hat{\boldsymbol{X}}$. That is, the decoder g tries to use a minimal coding rate needed to achieve a good decoding quality.

Hence, the overall "multi-class" Stackelberg game for learning the closed-loop transcription, named CTRL-Multi, is

$$\max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} \mathcal{T}_{\boldsymbol{X}}(\boldsymbol{\theta}, \boldsymbol{\eta}) \tag{6.2.18}$$

$$\doteq \underbrace{\Delta R_\epsilon\big(f(\boldsymbol{X}, \boldsymbol{\theta})\big)}_{\text{Expansive encode}} + \underbrace{\Delta R_\epsilon\big(h(\boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{\eta})\big)}_{\text{Compressive decode}} + \sum_{k=1}^{K} \underbrace{\Delta R_\epsilon\big(f(\boldsymbol{X}_k, \boldsymbol{\theta}), h(\boldsymbol{X}_k, \boldsymbol{\theta}, \boldsymbol{\eta})\big)}_{\text{Contrastive encode \& Contractive decode}} \tag{6.2.19}$$

$$= \Delta R_\epsilon\big(\boldsymbol{Z}(\boldsymbol{\theta})\big) + \Delta R_\epsilon\big(\hat{\boldsymbol{Z}}(\boldsymbol{\theta}, \boldsymbol{\eta})\big) + \sum_{k=1}^{K} \Delta R_\epsilon\big(\boldsymbol{Z}_k(\boldsymbol{\theta}), \hat{\boldsymbol{Z}}_k(\boldsymbol{\theta}, \boldsymbol{\eta})\big), \tag{6.2.20}$$

subject to certain constraints (upper or lower bounds) on the first term and the

second term.

Notice that, without the terms associated with the generative part h or with all such terms fixed as constant, the above objective is precisely the original MCR^2 objective introduced in Chapter 4. In an unsupervised setting, if we view each sample (and its augmentations) as its own class, the above formulation remains exactly the same. The number of classes K is simply the number of independent samples. In addition, notice that the above game's objective function depends only on (features of) the data $\boldsymbol{X}$, hence one can learn the encoder and decoder (parameters) without the need for sampling or matching any additional distribution (as typically needed in GANs or VAEs).

As a special case, if $\boldsymbol{X}$ only has one class, the Stackelberg game reduces to a special "two-class" or "binary" form,[12] named CTRL-Binary,

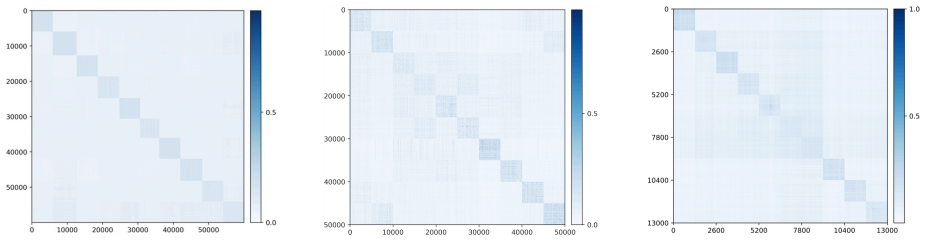
$$\max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} \mathcal{T}^b_{\boldsymbol{X}}(\boldsymbol{\theta}, \boldsymbol{\eta}) \doteq \Delta R_\epsilon\big(f(\boldsymbol{X}, \boldsymbol{\theta}), h(\boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{\eta})\big) = \Delta R_\epsilon\big(\boldsymbol{Z}(\boldsymbol{\theta}), \hat{\boldsymbol{Z}}(\boldsymbol{\theta}, \boldsymbol{\eta})\big), \tag{6.2.21}$$

between $\boldsymbol{X}$ and the decoded $\hat{\boldsymbol{X}}$ by viewing $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ as two classes $\{\mathbf{0}, \mathbf{1}\}$. Notice that this binary case resembles the formulation of the original GAN (6.2.3). Instead of using cross entropy, our formulation adopts a more refined rate-reduction measure, which has been shown to promote diversity in the learned representation in Chapter 3.

Sometimes, even when $\boldsymbol{X}$ contains multiple classes/modes, one could still view all classes together as one class. Then, the above binary objective is to align the union distribution of all classes with their decoded $\hat{\boldsymbol{X}}$. This is typically a simpler task to achieve than the multi-class one (6.2.20), since it does not require learning of a more refined multi-class CTRL for the data, as we will later see in experiments. Notice that one good characteristic of the above formulation is that *all quantities in the objectives are measured in terms of rate reduction for the learned features* (assuming features eventually become subspace Gaussians).

One may notice that the above learning framework draws inspiration from closed-loop error correction widely practiced in feedback control systems. The closed-loop mechanism is used to form an overall feedback system between the two encoding and decoding networks for correcting any "error" in the distributions between the data $\boldsymbol{x}$ and the decoded $\hat{\boldsymbol{x}}$. Using terminology from control theory, one may view the encoding network f as a "sensor" for error feedback while the decoding network g as a "controller" for error correction. However, notice that here the "target" for control is not a scalar nor a finite dimensional vector, but a continuous distribution—in order for the distribution of $\hat{\boldsymbol{x}}$ to match that of the data $\boldsymbol{x}$. This is in general a control problem in an infinite dimensional space. The space of possible diffeomorphisms of submanifolds that f tries to model is infinite-dimensional [Lee02]. Ideally, we hope when the sensor f and the controller g are optimal, the distribution of $\boldsymbol{x}$ becomes a "fixed point" for the closed loop while the distribution of $\boldsymbol{z}$ reaches a compact linear discriminative representation. Hence, the minimax programs (6.2.20) and (6.2.21) can also be interpreted as games between an error-feedback sensor and

[12] As the first two rate reduction terms automatically become zero.



(a) MNIST (b) CIFAR-10 (c) ImageNet

Figure 6.11: Visualizing the alignment between $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$: $|\boldsymbol{Z}^\top \hat{\boldsymbol{Z}}|$ in the feature space for (**a**) MNIST, (**b**) CIFAR-10, and (**c**) ImageNet-10-Class.

an error-reducing controller.

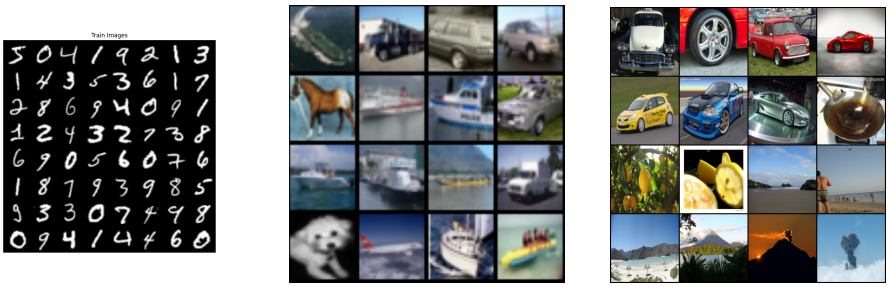
The remaining question is whether the above framework can indeed learn a good (autoencoding) representation of a given dataset? Before we give some formal theoretical justification (in the next subsection), we present some empirical results.

Visualizing correlation of features $\boldsymbol{Z}$ and decoded features $\hat{\boldsymbol{Z}}$. We visualize the cosine similarity between $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$ learned from the multi-class objective (6.2.20) on MNIST, CIFAR-10 and ImageNet (10 classes), which indicates how close $\hat{\boldsymbol{z}} = f \circ g(\boldsymbol{z})$ is from $\boldsymbol{z}$. Results in Figure 6.11 show that $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$ are aligned very well within each class. The block-diagonal patterns for MNIST are sharper than those for CIFAR-10 and ImageNet, as images in CIFAR-10 and ImageNet have more diverse visual appearances.

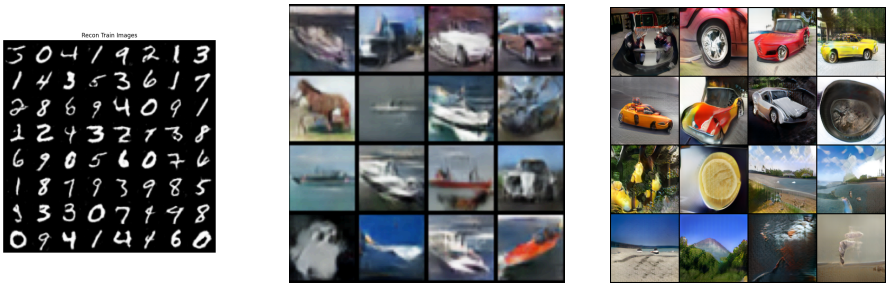
Visualizing autoencoding of the data $\boldsymbol{X}$ and the decoded $\hat{\boldsymbol{X}}$. We compare some representative $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ on MNIST, CIFAR-10 and ImageNet (10 classes) to verify how close $\hat{\boldsymbol{x}} = g \circ f(\boldsymbol{x})$ is to $\boldsymbol{x}$. The results are shown in Figure 6.12, and visualizations are created from training samples. Visually, the autoencoded $\hat{\boldsymbol{x}}$ faithfully captures major visual features from its respective training sample $\boldsymbol{x}$, especially the pose, shape, and layout. For the simpler dataset such as MNIST, autoencoded images are almost identical to the original.

6.2.2 A Mixture of Low-Dimensional Gaussians

In the above, we have argued that it is possible to formulate the problem of learning a data distribution as a closed-loop autoencoding problem. We also saw empirically that such a scheme seems to work. The remaining question is when and why such a scheme should work. It is difficult to answer this question for the most general cases with arbitrary data distributions. Nevertheless, as usual, let us see if we can arrive at a rigorous justification for the ideal case when the data distribution is a mixture of low-dimensional subspaces or low-rank Gaussians. A clear characterization and understanding of this important



(a) MNIST $\boldsymbol{X}$ (b) CIFAR-10 $\boldsymbol{X}$ (c) ImageNet $\boldsymbol{X}$



(d) MNIST $\hat{\boldsymbol{X}}$ (e) CIFAR-10 $\hat{\boldsymbol{X}}$ (f) ImageNet $\hat{\boldsymbol{X}}$

Figure 6.12: Visualizing the auto-encoding property of the learned closed-loop transcription ($\boldsymbol{x} \approx \hat{\boldsymbol{x}} = g \circ f(\boldsymbol{x})$) on MNIST, CIFAR-10, and ImageNet (zoom in for better visualization).

special case would shed light on the more general cases.[13]

To this end, let us first suppose that $\boldsymbol{X}$ is distributed according to a mixture of low-dimensional Gaussians, and the label (i.e., subspace assignment) for $\boldsymbol{X}$ is given by $\boldsymbol{y}$. Then, let us set up a minimax optimization problem to learn the data distribution, say through learning an encoding of $\boldsymbol{X}$ into representations $\boldsymbol{Z}$ which are supported on a mixture of *orthogonal* subspaces, and a decoding of $\boldsymbol{Z}$ back into $\boldsymbol{X}$. Then, the representation we want to achieve is maximized by the earlier-discussed version of the information gain, i.e., $\Delta R_\epsilon(\boldsymbol{Z}) = R_\epsilon(\boldsymbol{Z}) - \sum_{k=1}^K R_\epsilon(\boldsymbol{Z}_k)$, which enforces that the representation $\boldsymbol{Z}_k$ of each class k spans a subspace which is orthogonal to the supporting subspaces of other classes. The way to measure the consistency of the decoding is, as before, given by $\sum_{k=1}^K \Delta R_\epsilon(\boldsymbol{Z}_k, \hat{\boldsymbol{Z}}_k)$, which enforces that the representation $\boldsymbol{Z}_k$ and its autoencoding $\hat{\boldsymbol{Z}}_k$ for each class k span the same subspace. Thus, we can set up a simplified Stackelberg game:

$$\max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} \left\{ \Delta R_\epsilon(\boldsymbol{Z}(\boldsymbol{\theta})) + \sum_{k=1}^K \Delta R_\epsilon(\boldsymbol{Z}_k(\boldsymbol{\theta}), \hat{\boldsymbol{Z}}_k(\boldsymbol{\theta}, \boldsymbol{\eta})) \right\} \tag{6.2.22}$$

[13] As most distributions with low-dimensional structures can be well-approximated by this family of distributions.

Notice that this is a simpler setup than what is used in practice—there is no $\Delta R_\epsilon(\hat{\boldsymbol{Z}})$ term, for instance, and we work in the supervised setting with class labels (although the techniques used to prove the following result are easy to extend to unsupervised formulations). Also, the consistency of the representations is only measured in a distribution-wise sense via ΔR_ϵ (though this may be substituted with a sample-wise distance metric such as the ℓ_2 norm if desired, and equivalent conclusions may be drawn, *mutatis mutandis*).

Under mild conditions, in order to realize the desired encoder and decoder which realize $\boldsymbol{Z}$ from a data source $\boldsymbol{X}$ that is already distributed according to a mixture of correlated low-dimensional Gaussians, we only require a linear encoder and decoder to disentangle and whiten the Gaussians. We then study this setting in the case where $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ parameterize matrices whose operator norm is constrained.

We want to understand what kinds of optima are learned in this setting. One suitable solution concept for this kind of game, where the decoder's optimal solution is defined solely with respect to the encoder (and not, in particular, with respect to some other intrinsic property of the decoder), is a *Stackelberg equilibrium* where the decoder follows the encoder. Namely, at such an equilibrium, the decoder should optimize its objective; meanwhile the encoder should optimize its objective, given that whatever it picks, the decoder will pick an optimal response (which may affect the encoder objective). In game-theoretic terms, it is like the decoder goes *second*: it chooses its weights after the encoder, and both the encoder and decoder attempt to optimize their objective in light of this. It is computationally tractable to learn sequential equilibria via gradient methods via *alternating optimization*, where each side uses different learning rates. Detailed exposition of sequential equilibria is beyond the scope of this book and we provide more technical details in Section A.3. In this setting, we have the following result:

Theorem 6.1 ([PPC+23], Abridged)**.** *Suppose that $\boldsymbol{X}$ is distributed on a mixture of subspaces. Under certain realistic yet technical conditions, it holds that all sequential equilibria of (6.2.22) obey:*

- *The $\boldsymbol{Z}_k$ lie on orthogonal subspaces and are isotropic on those subspaces, i.e., maximizing the information gain.*
- *The autoencoding is self-consistent, i.e., the subspaces spanned by $\boldsymbol{Z}_k$ and $\hat{\boldsymbol{Z}}_k$ are the same for all k.*

This notion of self-consistency is the most one can expect if there are only geometric assumptions on the data, i.e., there are no statistical assumptions. If we assume that the columns of $\boldsymbol{X}$ are drawn from a low-rank Gaussian mixture model, then analogous versions of this theorem certify that $\boldsymbol{Z}_k$ are also low-rank Gaussians whose covariance is isotropic, for instance. Essentially, this result validates, via the simple case of Gaussian mixtures on subspaces, that minimax games to optimize the information gain and self-consistency may achieve optimal solutions.

6.3 Continuous Learning Self-Consistent Representations

6.3.1 Class-wise Incremental Learning

As we have seen, deep neural networks have demonstrated a great ability to learn representations for hundreds or even thousands of classes of objects, in both discriminative and generative contexts. However, networks typically must be trained offline, with uniformly sampled data from all classes simultaneously. It has been known that when an (open-loop) network is updated to learn new classes without data from the old ones, previously learned knowledge will fall victim to the problem of *catastrophic forgetting* [MC89]. This is known in neuroscience as the stability-plasticity dilemma: the challenge of ensuring that a neural system can learn from a new environment while retaining essential knowledge from previous ones [Gro87].

In contrast, natural neural systems (e.g., animal brains) do not seem to suffer from such catastrophic forgetting at all. They are capable of developing new memory of new objects while retaining memory of previously learned objects. This ability, for either natural or artificial neural systems, is often referred to as *incremental learning, continual learning, sequential learning*, or *lifelong learning* [AR20].

While many recent works have highlighted how artificial neural systems can be trained in more flexible ways, the strongest existing efforts toward answering the stability-plasticity dilemma for artificial neural networks typically require either storing raw exemplars [CRE+19; RKS+17] or providing external mechanisms [KPR+17]. Raw exemplars, particularly in the case of high-dimensional inputs like images, are costly and difficult to scale, while external mechanisms—which typically include secondary networks and representation spaces for generative replay, incremental allocation of network resources, network duplication, or explicit isolation of used and unused parts of the network—require heuristics and incur hidden costs.

Here we are interested in an incremental learning setting that is similar to nature. It counters these existing practices with two key qualities.

1. The first is that it should be *memory-based.* When learning new classes, no raw exemplars of old classes are available to train the network together with new data. This implies that one has to rely on a compact and thus structured "memory" learned for old classes, such as incrementally learned generative representations of the old classes, as well as the associated encoding and decoding mappings [KK18].

2. The second is that it should be *self-contained.* Incremental learning takes place in a single neural system with a fixed capacity, and in a common representation space. The ability to minimize forgetting is implied by optimizing an overall learning objective, without external networks, architectural modifications, or resource allocation mechanisms.

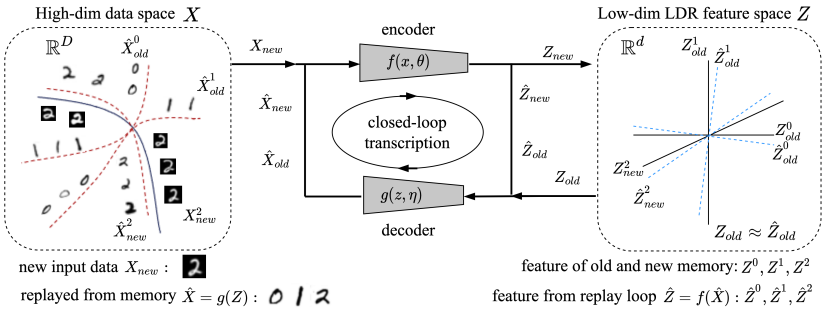


Figure 6.13: **Overall framework** of our closed-loop transcription-based incremental learning for a structured LDR memory. Only a single, entirely self-contained, encoding-decoding network is needed: for a new data class $\boldsymbol{X}_{new}$, a new LDR memory $\boldsymbol{Z}_{new}$ is incrementally learned as a minimax game between the encoder and decoder subject to the constraint that old memory of past classes $\boldsymbol{Z}_{old}$ is intact through the closed-loop transcription (or replay): $\boldsymbol{Z}_{old} \approx \hat{\boldsymbol{Z}}_{old} = f(g(\boldsymbol{Z}_{old}))$.

The incoherent linear structures for features of different classes closely resemble how objects are encoded in different areas of the inferotemporal cortex of animal brains [BSM+20; CT17]. The closed-loop transcription $\boldsymbol{X} \to \boldsymbol{Z} \to \hat{\boldsymbol{X}} \to \hat{\boldsymbol{Z}}$ also resembles popularly hypothesized mechanisms for memory formation [JT20; VST+20]. This leads to a question: since memory in the brain is formed in an incremental fashion, can the above closed-loop transcription framework also support incremental learning?

LDR memory sampling and replay. The simple linear *structures* of LDR make it uniquely suited for incremental learning: the distribution of features $\boldsymbol{Z}_j$ of each previously learned class can be explicitly and concisely represented by a principal subspace $\mathcal{S}_j$ in the feature space. To preserve the memory of an old class j, we only need to preserve the subspace while learning new classes. To this end, we simply sample m representative prototype features on the subspace along its top r principal components, and denote these features as $\boldsymbol{Z}_{j,old}$. Because of the simple linear structures of LDR, we can sample from $\boldsymbol{Z}_{j,old}$ by calculating the mean and covariance of $\boldsymbol{Z}_{j,old}$ after learning class j. The storage required is extremely small, since we only need to store means and covariances, which are sampled from as needed. Suppose a total of t old classes have been learned so far. If prototype features, denoted $\boldsymbol{Z}_{old} \doteq [\boldsymbol{Z}_{1,old}, \ldots, \boldsymbol{Z}_{t,old}]$, for all of these classes can be preserved when learning new classes, the subspaces $\{\mathcal{S}_j\}_{j=1}^t$ representing past memory will be preserved as well. Details about sampling and calculating mean and covariance can be found in the work of [TDW+23].

Incremental learning LDR with an old-memory constraint. Notice that, with the learned autoencoding (6.2.9), one can replay and use the images, say $\hat{\boldsymbol{X}}_{old} = g(\boldsymbol{Z}_{old}, \boldsymbol{\eta})$, associated with the memory features to avoid forgetting

while learning new classes. This is typically how generative models have been used for prior incremental learning methods. However, with the closed-loop framework, explicitly replaying images from the features is not necessary. Past memory can be effectively preserved through optimization exclusively on the features themselves.

Consider the task of incrementally learning a new class of objects.[14] We denote a corresponding new sample set as $\boldsymbol{X}_{new}$. The features of $\boldsymbol{X}_{new}$ are denoted as $\boldsymbol{Z}_{new}(\boldsymbol{\theta}) = f(\boldsymbol{X}_{new}, \boldsymbol{\theta})$. We concatenate them together with the prototype features of the old classes $\boldsymbol{Z}_{old}$ and form $\boldsymbol{Z} = [\boldsymbol{Z}_{new}(\boldsymbol{\theta}), \boldsymbol{Z}_{old}]$. We denote the replayed images from all features as $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{X}}_{new}(\boldsymbol{\theta}, \boldsymbol{\eta}), \hat{\boldsymbol{X}}_{old}(\boldsymbol{\eta})]$ although we do not actually need to compute or use them explicitly. We only need features of replayed images, denoted $\hat{\boldsymbol{Z}} = f(\hat{\boldsymbol{X}}, \boldsymbol{\theta}) = [\hat{\boldsymbol{Z}}_{new}(\boldsymbol{\theta}, \boldsymbol{\eta}), \hat{\boldsymbol{Z}}_{old}(\boldsymbol{\theta}, \boldsymbol{\eta})]$.

Mirroring the motivation for the multi-class CTRL objective (6.2.20), we would like the features of the new class $\boldsymbol{Z}_{new}$ to be incoherent to all of the old ones $\boldsymbol{Z}_{old}$. As $\boldsymbol{Z}_{new}$ is the only new class whose features need to be learned, the objective (6.2.20) reduces to the case where $K = 1$:

$$\min_{\boldsymbol{\eta}} \max_{\boldsymbol{\theta}} \Delta R_\epsilon(\boldsymbol{Z}) + \Delta R_\epsilon(\hat{\boldsymbol{Z}}) + \Delta R_\epsilon(\boldsymbol{Z}_{new}, \hat{\boldsymbol{Z}}_{new}). \tag{6.3.1}$$

However, when we update the network parameters $(\boldsymbol{\theta}, \boldsymbol{\eta})$ to optimize the features for the new class, the updated mappings f and g will change features of the old classes too. Hence, to minimize the distortion of the old class representations, we can try to enforce $\mathrm{Cov}(\boldsymbol{Z}_{j,old}) = \mathrm{Cov}(\hat{\boldsymbol{Z}}_{j,old})$. In other words, while learning new classes, we enforce that the memory of old classes remains "self-consistent" through the transcription loop:

$$\boldsymbol{Z}_{old} \xrightarrow{g(\boldsymbol{z},\boldsymbol{\eta})} \hat{\boldsymbol{X}}_{old} \xrightarrow{f(\boldsymbol{x},\boldsymbol{\theta})} \hat{\boldsymbol{Z}}_{old}. \tag{6.3.2}$$

Mathematically, this is equivalent to setting

$$\Delta R_\epsilon(\boldsymbol{Z}_{old}, \hat{\boldsymbol{Z}}_{old}) \doteq \sum_{j=1}^{t} \Delta R_\epsilon(\boldsymbol{Z}_{j,old}, \hat{\boldsymbol{Z}}_{j,old}) = 0.$$

Hence, the above minimax program (6.3.1) is revised as a *constrained* minimax game, which we refer to as *incremental closed-loop transcription* (i-CTRL). The objective of this game is identical to the standard multi-class CTRL objective (6.2.20), but includes just one additional constraint:

$$\begin{aligned} \min_{\boldsymbol{\eta}} \max_{\boldsymbol{\theta}} \quad & \Delta R_\epsilon(\boldsymbol{Z}) + \Delta R_\epsilon(\hat{\boldsymbol{Z}}) + \Delta R_\epsilon(\boldsymbol{Z}_{new}, \hat{\boldsymbol{Z}}_{new}) \\ & \text{subject to} \quad \Delta R_\epsilon(\boldsymbol{Z}_{old}, \hat{\boldsymbol{Z}}_{old}) = 0. \end{aligned} \tag{6.3.3}$$

In practice, the constrained minimax program can be solved by *alternating* minimization and maximization between the encoder $f(\cdot, \boldsymbol{\theta})$ and decoder $g(\cdot, \boldsymbol{\eta})$ as follows:

$$\max_{\boldsymbol{\theta}} \quad \Delta R_\epsilon(\boldsymbol{Z}) + \Delta R_\epsilon(\hat{\boldsymbol{Z}}) + \lambda \cdot \Delta R_\epsilon(\boldsymbol{Z}_{new}, \hat{\boldsymbol{Z}}_{new}) - \gamma \cdot \Delta R_\epsilon(\boldsymbol{Z}_{old}, \hat{\boldsymbol{Z}}_{old}), \tag{6.3.4}$$

[14] Of course, one may also consider the more general setting where the task contains a small batch of new classes, without serious modification.

$$\min_{\boldsymbol{\eta}} \quad \Delta R_\epsilon(\boldsymbol{Z}) + \Delta R_\epsilon(\hat{\boldsymbol{Z}}) + \lambda \cdot \Delta R_\epsilon(\boldsymbol{Z}_{new}, \hat{\boldsymbol{Z}}_{new}) + \gamma \cdot \Delta R_\epsilon(\boldsymbol{Z}_{old}, \hat{\boldsymbol{Z}}_{old}), \tag{6.3.5}$$

where the constraint $\Delta R_\epsilon(\boldsymbol{Z}_{old}, \hat{\boldsymbol{Z}}_{old}) = 0$ in (6.3.3) has been converted (and relaxed) to a Lagrangian term with a corresponding coefficient γ and sign. We additionally introduce another coefficient λ for weighting the rate reduction term associated with the new data.

Jointly optimal memory via incremental reviewing. As we will see, the above constrained minimax program can already achieve state-of-the-art performance for incremental learning. Nevertheless, developing an optimal memory for *all classes* cannot rely on graceful forgetting alone. Even for humans, if an object class is learned only once, we should expect the learned memory to fade as we continue to learn new ones, unless the memory can be consolidated by reviewing old object classes.

To emulate this phase of memory forming, after incrementally learning a whole dataset, we may go back to review all classes again, one class at a time. We refer to going through all classes once as one reviewing "cycle".[15] If needed, multiple reviewing cycles can be conducted. It is quite expected that reviewing can improve the learned (LDR) memory. But somewhat surprisingly, the closed-loop framework allows us to review even in a "class-unsupervised" manner: when reviewing data of an old class, say $\boldsymbol{X}_j$, the system does not need the class label and can simply treat $\boldsymbol{X}_j$ as a new class $\boldsymbol{X}_{new}$. That is, the system optimizes the same constrained mini-max program (6.3.3) without any modification; after the system is optimized, one can identify the newly learned subspace spanned by $\boldsymbol{Z}_{new}$, and use it to replace or merge with the old subspace $\mathcal{S}_j$. As our experiments show, such a class-unsupervised incremental review process can gradually improve both discriminative and generative performance of the LDR memory, eventually converging to that of a jointly learned memory.

Experimental verification. We show some experimental results on the following datasets: MNIST [LBB+98b] and CIFAR-10 [KNH14]. All experiments are conducted for the more challenging class-IL setting. For both MNIST and CIFAR-10, the 10 classes are split into 5 tasks with 2 classes each or 10 tasks with 1 class each. For the encoder f and decoder g, we adopt a very simple network architecture modified from DCGAN [RMC16], which is merely a *four-layer* convolutional network. Here we only show some qualitative visual results; more experiments and detailed analysis can be found in the work [TDW+23].

Visualizing auto-encoding properties. We begin by qualitatively visualizing some representative images $\boldsymbol{X}$ and the corresponding replayed $\hat{\boldsymbol{X}}$ on MNIST and CIFAR-10. The model is learned incrementally with the datasets split into 5 tasks. Results are shown in Figure 6.14, where we observe that the reconstructed $\hat{\boldsymbol{X}}$ preserves the main visual characteristics of $\boldsymbol{X}$ including shapes and

[15] To distinguish from the term "epoch" used in the conventional joint learning setting.

(a) MNIST $\boldsymbol{X}$ (b) MNIST $\hat{\boldsymbol{X}}$ (c) CIFAR-10 $\boldsymbol{X}$ (d) CIFAR-10 $\hat{\boldsymbol{X}}$

Figure 6.14: Visualizing the auto-encoding property of the learned representation ($\hat{\boldsymbol{X}} = g \circ f(\boldsymbol{X})$).

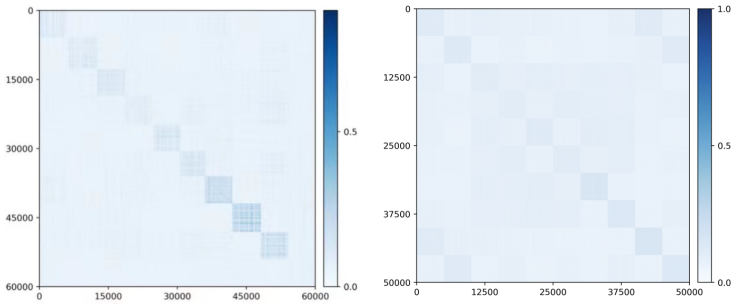


Figure 6.15: Block diagonal structure of $|\boldsymbol{Z}^\top \boldsymbol{Z}|$ in the feature space for MNIST (left) and CIFAR-10 (right).

textures. For a simpler dataset like MNIST, the replayed $\hat{\boldsymbol{X}}$ are almost identical to the input $\boldsymbol{X}$! This is rather remarkable given: (1) our method does not explicitly enforce $\hat{\boldsymbol{x}} \approx \boldsymbol{x}$ for individual samples as most autoencoding methods do, and (2) after having incrementally learned all classes, the generator has not forgotten how to generate digits learned earlier, such as 0, 1, 2. For a more complex dataset like CIFAR-10, we also demonstrate good visual quality, faithfully capturing the essence of each image.

Principal subspaces of the learned features. Most generative memory-based methods utilize autoencoders, VAEs, or GANs for replay purposes. The structure or distribution of the learned features $\boldsymbol{Z}_j$ for each class is unclear in the feature space. The features $\boldsymbol{Z}_j$ of the LDR memory, on the other hand, have a clear linear structure. Figure 6.15 visualizes correlations among all learned features $|\boldsymbol{Z}^\top \boldsymbol{Z}|$, in which we observe clear block-diagonal patterns for both datasets.[16] This indicates that the features for different classes $\boldsymbol{Z}_j$ indeed lie on subspaces that are incoherent from one another. Hence, features of each class can be well modeled as a principal subspace in the feature space.

[16] Notice that these patterns closely resemble the similarity matrix of response profiles of object categories from different areas of the inferotemporal cortex, as shown in Extended DataFig.3 of [BSM+20].

(a) sampled $\hat{\boldsymbol{x}}_{old}$ of '4' (b) sampled $\hat{\boldsymbol{x}}_{old}$ of '7' (c) sampled $\hat{\boldsymbol{x}}_{old}$ of 'horse' (d) sampled $\hat{\boldsymbol{x}}_{old}$ of 'ship'

Figure 6.16: Visualization of 5 reconstructed $\hat{\boldsymbol{x}} = g(\boldsymbol{z})$ from $\boldsymbol{z}$'s with the closest distance to (top-4) principal components of learned features for MNIST (class '4' and class '7') and CIFAR-10 (class 'horse' and 'ship').

Replay images of samples from principal components. Since features of each class can be modeled as a principal subspace, we further visualize the individual principal components within each of those subspaces. Figure 6.16 shows the images replayed from sampled features along the top-4 principal components for different classes, on MNIST and CIFAR-10 respectively. Each row represents samples along one principal component, and they clearly show similar visual characteristics but are distinctively different from those in other rows. We see that the model remembers different poses of '4' after having learned all remaining classes. For CIFAR-10, the incrementally learned memory remembers representative poses and shapes of horses and ships.

Effectiveness of incremental reviewing. We verify how the incrementally learned LDR memory can be further consolidated with an unsupervised incremental reviewing phase described before. Experiments are conducted on CIFAR-10, with 10 steps. Figure 6.17 left shows replayed images of the first class 'airplane' at the end of incremental learning of all ten classes, sampled along the top-3 principal components – every two rows (16 images) are along one principal direction. Their visual quality remains very decent—we observe almost no forgetting. The right figure shows replayed images after reviewing the first class once. We notice a significant improvement in visual quality after reviewing, and principal components of the features in the subspace start to correspond to distinctively different visual attributes within the same class.

6.3.2 Sample-wise Continuous Unsupervised Learning

As we know, the closed-loop CTRL formulation can already learn a decent autoencoding, even without class information, with the CTRL-Binary program:

$$\max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} \quad \Delta R_{\epsilon}(\boldsymbol{Z}, \hat{\boldsymbol{Z}}) \tag{6.3.6}$$

However, note that (6.3.6) is practically limited because it only aligns the dataset $\boldsymbol{X}$ and the regenerated $\hat{\boldsymbol{X}}$ at the distribution level. There is no guarantee that each sample $\boldsymbol{x}$ would be close to the decoded $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$.

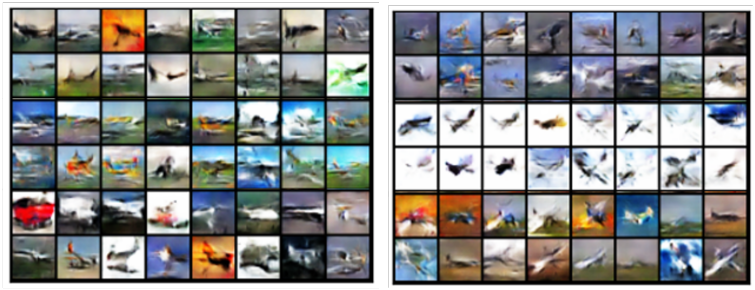


Figure 6.17: Visualization of replayed images $\hat{\boldsymbol{x}}_{old}$ of class 1-'airplane' in CIFAR-10, before (left) and after (right) one reviewing cycle.

Sample-wise constraints for unsupervised transcription. To improve discriminative and generative properties of representations learned in the unsupervised setting, we propose two additional mechanisms for the above CTRL-Binary maximin game (6.3.6). For simplicity and uniformity, here these will be formulated as equality constraints over rate reduction measures, but in practice they can be enforced softly during optimization.

Sample-wise self-consistency via closed-loop transcription. First, to address the issue that CTRL-Binary does not learn a sample-wise consistent autoencoding, we need to promote $\hat{\boldsymbol{x}}$ to be close to $\boldsymbol{x}$ for each sample. In the CTRL framework, this can be achieved by enforcing their corresponding features $\boldsymbol{z} = f(\boldsymbol{x})$ and $\hat{\boldsymbol{z}} = f(\hat{\boldsymbol{x}})$ to be close. To promote sample-wise self-consistency, where $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$ is close to $\boldsymbol{x}$, we want the distance between $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$ to be zero or small, for all N samples. This distance can be measured by the rate reduction:

$$\sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \hat{\boldsymbol{z}}^i) = 0. \tag{6.3.7}$$

Note that this again avoids measuring differences in the image space.

Self-supervision via compressing augmented samples. Since we do not know any class label information between samples in the unsupervised setting, the best we can do is to view every sample and its augmentations (say via translation, rotation, occlusion, etc.) as one "class"—a basic idea behind almost all self-supervised learning methods. In the rate reduction framework, it is natural to compress the features of each sample and its augmentations. In this work, we adopt the standard transformations in SimCLR [CKN+20] and denote such a transformation as τ. We denote each augmented sample $\boldsymbol{x}_a = \tau(\boldsymbol{x})$, and its corresponding feature as $\boldsymbol{z}_a = f(\boldsymbol{x}_a, \boldsymbol{\theta})$. For discriminative purposes, we hope the classifier is *invariant* to such transformations. Hence it is natural to enforce that the features $\boldsymbol{z}_a$ of all augmentations are the same as those $\boldsymbol{z}$ of the original sample $\boldsymbol{x}$. This is equivalent to requiring the distance between $\boldsymbol{z}$ and

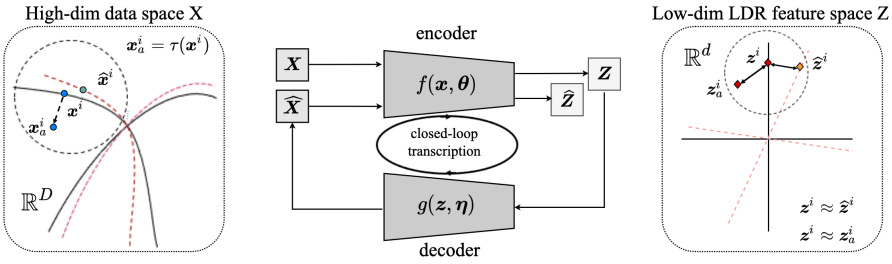


Figure 6.18: **Overall framework** of closed-loop transcription for unsupervised learning. Two additional constraints are imposed on the CTRL-Binary method: 1) self-consistency for sample-wise features $\boldsymbol{z}^i$ and $\hat{\boldsymbol{z}}^i$, say $\boldsymbol{z}^i \approx \hat{\boldsymbol{z}}^i$; and 2) invariance/similarity among features of augmented samples $\boldsymbol{z}^i$ and $\boldsymbol{z}_a^i$, say $\boldsymbol{z}^i \approx \boldsymbol{z}_a^i = f(\tau(\boldsymbol{x}^i), \boldsymbol{\theta})$, where $\boldsymbol{x}_a^i = \tau(\boldsymbol{x}^i)$ is an augmentation of sample $\boldsymbol{x}^i$ via some transformation $\tau(\cdot)$.

$\boldsymbol{z}_a$, measured in terms of rate reduction again, to be zero (or small) for all N samples:

$$\sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \boldsymbol{z}_a^i) = 0. \tag{6.3.8}$$

Unsupervised representation learning via closed-loop transcription. So far, we know the CTRL-Binary objective $\Delta R_\epsilon(\boldsymbol{Z}, \hat{\boldsymbol{Z}})$ in (6.3.6) helps align the distributions while sample-wise self-consistency (6.3.7) and sample-wise augmentation (6.3.8) help align and compress features associated with each sample. Besides consistency, we also want learned representations to be maximally discriminative for different samples (here viewed as different "classes"). Notice that the rate distortion term $R_\epsilon(\boldsymbol{Z})$ measures the coding rate (hence volume) of all features.

Unsupervised CTRL. Putting these elements together, we propose to learn a representation via the following constrained maximin program, which we refer to as *unsupervised CTRL* (u-CTRL):

$$\begin{aligned} \max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} \quad & R_\epsilon(\boldsymbol{Z}) + \Delta R_\epsilon(\boldsymbol{Z}, \hat{\boldsymbol{Z}}) \\ \text{subject to} \quad & \sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \hat{\boldsymbol{z}}^i) = 0, \quad \text{and} \quad \sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \boldsymbol{z}_a^i) = 0. \end{aligned} \tag{6.3.9}$$

Figure 6.18 illustrates the overall architecture of the closed-loop system associated with this program.

In practice, the above program can be optimized by alternating maximization and minimization between the encoder $f(\cdot, \boldsymbol{\theta})$ and the decoder $g(\cdot, \boldsymbol{\eta})$. We adopt the following optimization strategy that works well in practice, which is used for all subsequent experiments on real image datasets:

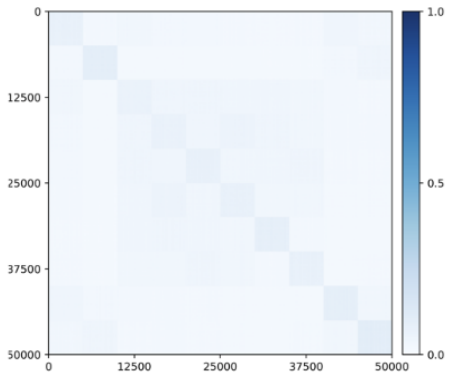


Figure 6.19: Emergence of block-diagonal structures of $|\boldsymbol{Z}^\top \boldsymbol{Z}|$ in the feature space for CIFAR-10.

$$\max_{\boldsymbol{\theta}} \; R_\epsilon(\boldsymbol{Z}) + \Delta R_\epsilon(\boldsymbol{Z}, \hat{\boldsymbol{Z}}) - \lambda_1 \sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \boldsymbol{z}_a^i) - \lambda_2 \sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \hat{\boldsymbol{z}}^i) \quad (6.3.10)$$

$$\min_{\boldsymbol{\eta}} \; R_\epsilon(\boldsymbol{Z}) + \Delta R_\epsilon(\boldsymbol{Z}, \hat{\boldsymbol{Z}}) + \lambda_1 \sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \boldsymbol{z}_a^i) + \lambda_2 \sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \hat{\boldsymbol{z}}^i), \quad (6.3.11)$$

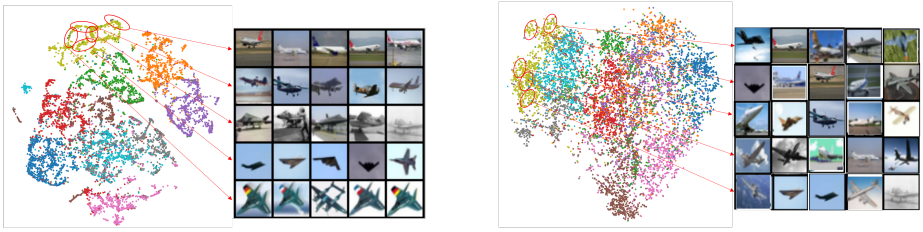
where the constraints $\sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \hat{\boldsymbol{z}}^i) = 0$ and $\sum_{i \in N} \Delta R_\epsilon(\boldsymbol{z}^i, \boldsymbol{z}_a^i) = 0$ in (6.3.9) have been converted (and relaxed) to Lagrangian terms with corresponding coefficients λ_1 and λ_2.[17]

The above representation is learned without class information. In order to facilitate discriminative or generative tasks, it must be highly structured. It has been verified experimentally that this is indeed the case and u-CTRL demonstrates significant advantages over other incremental or unsupervised learning methods [TDC+24]. We here only illustrate some qualitative results with the experiment on the CIFAR-10 dataset [KNH14], with standard augmentations for self-supervised learning [CKN+20]. One may refer to [TDC+24] for experiments on more and larger datasets and their quantitative evaluations.

As one can see from the experiments, specific and unique structure indeed emerges naturally in the representations learned using u-CTRL: globally, features of images in the same class tend to be clustered well together and separated from other classes, as shown in Figure 6.19; locally, features of individual classes cluster into tight, well-separated groups, as shown in Figure 6.20.

Unsupervised conditional image generation via rate reduction. The highly-structured feature distribution also suggests that the learned representa-

[17] Notice that computing the rate reduction terms ΔR for all samples or a batch of samples requires computing the expensive log det of large matrices. In practice, from the geometric meaning of ΔR for two vectors, ΔR can be approximated with an ℓ^2 norm or the cosine distance between two vectors. This approximation is developed further in Section 4.3.2, where it motivates the SimDINO objective.



(a) u-CTRL (b) MoCoV2

Figure 6.20: t-SNE visualizations of learned features of CIFAR-10 with different models.

tion can be very useful for generative purposes. For example, we can organize the sample features into meaningful clusters, and model them with low-dimensional (Gaussian) distributions or subspaces. By sampling from these compact models, we can conditionally regenerate meaningful samples from computed clusters. This is known as *unsupervised conditional image generation* [HKJ+21].

To cluster features, we exploit the fact that the rate reduction framework (4.2.15) is inspired by unsupervised clustering via compression [MDH+07b], which provides a principled way to find the membership $\boldsymbol{\Pi}$. Concretely, we maximize the same rate reduction objective (4.2.15) over $\boldsymbol{\Pi}$, but fix the learned representation $\boldsymbol{Z}$ instead. We simply view the membership $\boldsymbol{\Pi}$ as a nonlinear function of the features $\boldsymbol{Z}$, say $h_{\boldsymbol{\pi}}(\cdot, \xi) : \boldsymbol{Z} \mapsto \boldsymbol{\Pi}$ with parameters ξ. In practice, we model this function with a simple neural network, such as an MLP head right after the output feature $\boldsymbol{z}$. To estimate a "pseudo" membership $\hat{\boldsymbol{\Pi}}$ of the samples, we solve the following optimization problem over $\boldsymbol{\Pi}$:

$$\hat{\boldsymbol{\Pi}} = \arg\max_{\xi} \Delta R_{\epsilon}(\boldsymbol{Z} \mid \boldsymbol{\Pi}(\xi)). \tag{6.3.12}$$

In Figure 6.21, we visualize images generated from the ten unsupervised clusters from (6.3.12). Each block represents one cluster and each row represents one principal component for each cluster. Despite learning and training without labels, the model not only organizes samples into correct clusters, but is also able to preserve statistical diversities within each cluster/class. We can easily recover the diversity within each cluster by computing different principal components and then sampling and generating accordingly. While the experiments presented here are somewhat limited in scale, we will explore more direct and powerful methods that utilize the learned data distributions and representations for conditional generation and estimation in the next chapter.

6.4 Summary and Notes

Historically, autoencoding has been one of the important drivers of research innovation in neural networks for learning, although the most practically impressive demonstrations of deep learning have probably been in other domains

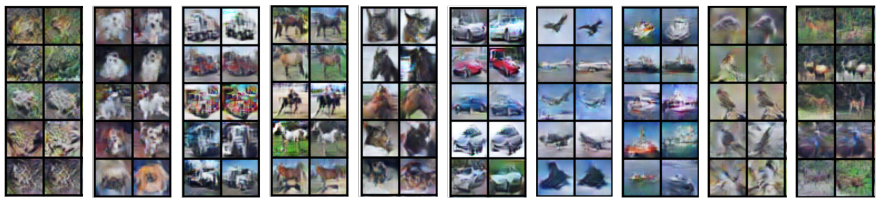


Figure 6.21: Unsupervised conditional image generation from each cluster of CIFAR-10, using u-CTRL. Images from different rows mean generation from different principal components of each cluster.

(such as discriminative classification, with AlexNet [KSH12], or generative modeling with GPT architectures [BMR+20]). Works we have featured throughout the chapter, especially the work of [HS06], served as catalysts of research interest in neural networks during times when they were otherwise not prominent in the machine learning research landscape. In modern practice, autoencoders remain core components of many large-scale systems for generating highly structured data such as visual data, speech data, and molecular data, especially the VQ-VAE approach [OVK17], which builds on the variational autoencoder methodology we discussed in Section 6.1.4. The core problem of autoencoding remains of paramount intellectual importance due to its close connection with representation learning, and we anticipate that it will reappear on the radar of practical researchers in the future as efficiency in training and deploying large models continues to become more important.

Materials presented in the second half of this chapter are based on a series of recent works on the topic of closed-loop transcription: [DTL+22], [PPC+23], [TDW+23], and [TDC+24]. In particular, Section 6.2.1 is based on the pioneering work of [DTL+22]. After that, the work of [PPC+23] has provided strong theoretical justifications for the closed-loop framework, at least for an ideal case. Section 6.3.1 and Section 6.3.2 are based on the works of [TDW+23] and [TDC+24], respectively. They demonstrate that the closed-loop framework naturally supports incremental and continuous learning, either in a class-wise or sample-wise setting. The reader may refer to these papers for more technical and experimental details.

Shallow vs. deep neural networks, for autoencoding and more. In Section 6.1.2, we discussed Cybenko's universal approximation theorem and how it states that in principle, a neural network with a single hidden layer (and suitable elementwise nonlinearities) is sufficient to approximate any suitably regular target function. Of course, the major architectural reason for the dominance of neural networks in practice has been the refinement of techniques for training *deeper* neural networks. Why is depth necessary? From a fundamental point of view, the issue of depth separations, which construct settings where a deeper neural network can approximate a given class of target functions with exponentially-superior efficiency relative to a shallow network, has been studied

at great length in the theoretical literature: examples include [BN20; Tel16; VJO+21]. The ease of training deeper networks in practice has not received as satisfying an answer from the perspective of theory. ResNets [HZR+16b] represent the pioneering empirical work of making deeper networks more easily trainable, used in nearly all modern architectures in some form. Theoretical studies have focused heavily on trainability of very deep networks, quantified via the initial neural tangent kernel [BGW21; MBD+21], but these studies have not given significant insight into the trainability benefits of deeper networks in middle and late stages of training (but see [YH21] for the root of a line of research attempting to address this).

6.5 Exercises and Extensions

Exercise 6.1 (Conceptual Understanding of Manifold Flattening). Consider data lying on a curved manifold $\mathcal{M}$ embedded in $\mathbb{R}^D$ (like a curved surface in 3-dimensional space), as discussed in the manifold flattening subsection of Section 6.1.2. In this exercise, we will describe the basic ingredients of the manifold flattening algorithm from [PPR+24]. A manifold is called flat if it is an open set in Euclidean space (or more generally, an open set in a subspace).

1. Suppose the manifold $\mathcal{M}$ is a *graph*: this means that $\mathcal{M} \subset \mathbb{R}^{D_1} \times \mathbb{R}^{D_2}$ (say), and that there is a function $F : \mathbb{R}^{D_1} \to \mathbb{R}^{D_2}$ such that

$$\mathcal{M} = \{(\boldsymbol{x}, F(\boldsymbol{x})) \mid \boldsymbol{x} \in \mathbb{R}^{D_1}\}. \tag{6.5.1}$$

   Give an integer D_3 and a map $f : \mathbb{R}^{D_1} \times \mathbb{R}^{D_2} \to \mathbb{R}^{D_3}$ that flattens $\mathcal{M}$, and describe the corresponding (lossless) reconstruction procedure from the flattened representation.

2. Now suppose that $\mathcal{M}$ is a general smooth manifold. Smooth manifolds have the property that they are locally well-approximated by subspaces near each point. Describe in intuitive terms how to flatten the manifold $\mathcal{M}$ locally at each point, by relating it to a graph. (The algorithm of [PPR+24] performs a refined version of this local flattening process in a way that allows them to be glued together to form a global flattening.)

Exercise 6.2 (Reproduce Closed-Loop Transcription). Implement a closed-loop transcription pipeline for representation learning on the CIFAR-10 dataset following the methodology in Section 6.2.1. Reference [DTL+22] for useful hyperparameters and architecture settings. Reproduce the result in Figure 6.11.