68 GENERATIVE NETWORKS

The material in the last three chapters focused on the use of neural network structures for the solution of inference (regression and classification) problems. In this chapter, we use the same networks to develop two *generative* methods whose purpose is to generate samples from the same underlying distribution as the training data.

The first method builds upon the variational inference approach from Chapters 33 and 36 and leads to variational autoencoders (VAEs). Each VAE consists of the cascade of two feedforward neural networks: one plays the role of the *encoder* and the other plays the role of the *decoder*. The purpose of the encoder is to compress the feature data down to a small-dimensional latent space. The latent variables are fed into the decoder, which then generates samples from the data distribution at its output. The encoder-decoder combination is trained by means of a backpropagation algorithm, with special attention to the propagation of sensitivity factors through the transition layer linking the encoder to the decoder.

The second method relies on a zero-sum game formulation and leads to a generative adversarial network (GAN). Each GAN consists of a cascade of two feedforward neural networks: one is called the *generator* and the other the *discriminator*. The purpose of the generator is to generate "fake" feature data to "fool" the discriminator, while the purpose of the discriminator is to decide whether the feature data at its input is "real" or "fake." The generator-discriminator combination is again trained by means of a backpropagation algorithm, with special attention to the propagation of sensitivity factors through the coupling layer linking the generator and discriminator. GANs tend to lead to better results (at least subjectively) than VAEs. They are, however, much harder to train due to the competition that occurs between the generator and discriminator sections vying to reach an equilibrium. We discuss VAEs first, followed by GANs.

68.1 VARIATIONAL AUTOENCODERS

Given a collection of feature vectors $\{h_n\}_{n=0}^{N-1} \in \mathbb{R}^M$, we would like to determine a generative model that explains the data and which can be used to generate additional similar samples. One possibility is to learn the underlying probability

distribution, $f_{\boldsymbol{h}}(h)$, from which the data arises. We explained in Chapter 33 that computing such evidences is generally challenging. Moreover, it is often more useful to construct a structure that can sample from $f_{\boldsymbol{h}}(h)$ directly without the need to evaluate the distribution explicitly. This section describes one such method by using variational autoencoders (VAEs). The method exploits to great effect the notion of latent variables and variational inference techniques from Chapters 33 and 36.

68.1.1 Encoder-Decoder Structure

Variational autoencoders (VAEs) employ the same architecture we described before in Fig. 33.1, and which we reproduce here for ease of reference with some adjustment to the notation. Let $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ denote the model for generating the data $\boldsymbol{h}$, based on some latent variable $\boldsymbol{u}$. Recall that we used the symbol $\boldsymbol{z}$ to refer to latent variables in Chapter 33. In this chapter, we will use the symbol $\boldsymbol{u}$. This is because in our study of neural networks, we have reserved the notation $\boldsymbol{z}$ to represent internal variables prior to the activation functions.

It is generally the case that the latent variable $\boldsymbol{u}$ lies in a (much) lower dimensional space than the feature data so that the mapping from $\boldsymbol{h}$ to $\boldsymbol{u}$ amounts to some form of data compression. We refer to the model $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ as the *decoder* since it maps u back to h (i.e., it helps decode the information summarized in u). The distribution $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ represents the likelihood of $\boldsymbol{h}$ given u. On the other hand, the conditional $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$, with the roles of u and h reversed, is known as the *encoder* since it maps h to u and helps encode the information from h into u.

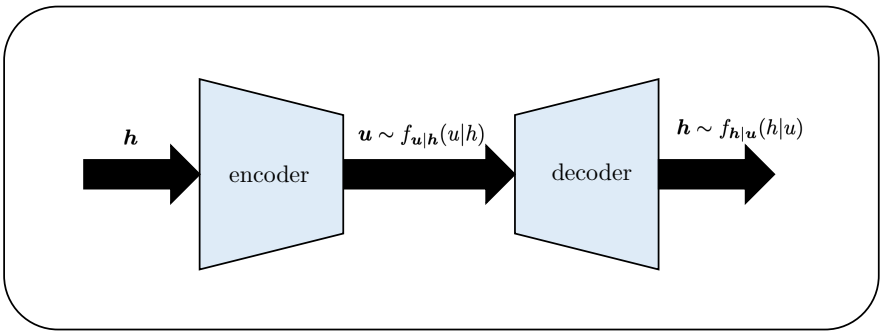

Figure 68.1 The decoder is represented by $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ since it decodes the information contained in the latent variable u, while the encoder is represented by $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ since it codes the information from h into u.

68.1.2 Maximizing the ELBO

We explained in Secs. 36.1 and 36.3 how to learn posterior distributions of the form $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ by maximizing the evidence lower-bound (or ELBO) and by using the mean-field approximation theory. We will again exploit the ELBO in this section, with two main differences in relation to the earlier approach:

(a) (**Data-driven solution**) We will not assume knowledge of the joint distribution $f_{\boldsymbol{h},\boldsymbol{u}}(h,u)$, as was the case with the mean-field approximation theory. We will rely solely on data realizations $\{h_n\}_{n=0}^{N-1}$.

(b) (**Gaussian approximation**) We will resort to a neural network formulation rather than the mean-field theory. In particular, we will assume beforehand a *Gaussian approximation* for the posterior $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ and train a neural network to learn the first and second-order moments of this Gaussian distribution. In contrast, in the mean-field approximation approach, we searched for the optimal posterior approximation by using (36.42).

Thus, let $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ denote an approximation for the actual posterior $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$. Variational inference seeks the distribution $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ that is closest to $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ in the KL divergence sense:

$$q^{\star}_{\boldsymbol{u}|\boldsymbol{h}}(u|h) \;\triangleq\; \underset{q_{\boldsymbol{u}|\boldsymbol{h}}(\cdot)}{\operatorname{argmin}}\; D_{\mathrm{KL}}\Big(q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\,\|\,f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\Big) \tag{68.1}$$

This is of course a challenging problem to solve because $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ is unknown. However, we established a fundamental equality in (6.164) relating the KL divergence of two distributions to a second quantity called the Evidence Lower Bound (ELBO), which we denoted by $\mathcal{L}$. Specifically, we showed that it always holds

$$D_{\mathrm{KL}}\Big(q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\,\|\,f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\Big) \;+\; \mathcal{L}\Big(q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\,\|\,f_{\boldsymbol{h},\boldsymbol{u}}(h,u)\Big) \;=\; \ln f_{\boldsymbol{h}}(h) \tag{68.2}$$

where $\mathcal{L}$ is defined by

$$\mathcal{L}\Big(q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\,\|\,f_{\boldsymbol{h},\boldsymbol{u}}(h,u)\Big) \;\triangleq\; \int_{u\in\mathcal{U}} q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\,\ln\left(\frac{f_{\boldsymbol{h},\boldsymbol{u}}(h,u)}{q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)}\right)du \tag{68.3}$$

If we drop the arguments, we can rewrite (68.2) more succinctly in the form

$$D_{\mathrm{KL}} \;+\; \mathcal{L} \;=\; \ln f_{\boldsymbol{h}}(h) \tag{68.4}$$

where, from (68.3):

$$\mathcal{L}(q) \;\triangleq\; \mathbb{E}_q\Big[\ln f_{\boldsymbol{h},\boldsymbol{u}}(h,u)\Big] \;-\; \mathbb{E}_q\Big[\ln q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\Big] \tag{68.5}$$

and the notation $\mathbb{E}_q$ refers to expectation relative to the conditional distribution $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ (which in turn depends on h). We are also writing $\mathcal{L}(q)$, rather than just $\mathcal{L}$, to highlight the fact that the value of $\mathcal{L}$ depends on q.

Since the joint pdf $f_{\boldsymbol{h},\boldsymbol{u}}(h,u)$ is assumed unavailable in this treatment, we rework expression (68.5) by using Bayes rule and write:

$$\begin{aligned}\mathcal{L}(q) &= \mathbb{E}_q\Big[\ln\Big(f_{\boldsymbol{u}}(u)\, f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)\Big)\Big] \;-\; \mathbb{E}_q\Big[\ln q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\Big] \\ &= \mathbb{E}_q\Big[\ln f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)\Big] - \mathbb{E}_q\left[\ln\Big(\frac{q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)}{f_{\boldsymbol{u}}(u)}\Big)\right] \\ &= \mathbb{E}_q\Big[\ln f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)\Big] - D_{\mathrm{KL}}\Big(q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)\,\|\, f_{\boldsymbol{u}}(u)\Big) \end{aligned} \tag{68.6}$$

where the last term is equal to the KL divergence between $q_{\boldsymbol{u}|\boldsymbol{h}}(\cdot)$ and $f_{\boldsymbol{u}}(\cdot)$. Substituting into (68.4) we conclude that (we are removing the subscripts from the pdf notation for compactness):

$$\ln f(h) = D_{\mathrm{KL}}\Big(q(u|h)\,\|\, f(u|h)\Big) + \underbrace{\mathbb{E}_q\Big(\ln f(h|u)\Big) - D_{\mathrm{KL}}\Big(q(u|h)\,\|\, f(u)\Big)}_{\triangleq\ \textbf{ELBO},\ \mathcal{L}(q)} \tag{68.7}$$

where the two rightmost terms correspond to the ELBO. Since the KL divergence is always nonnegative, we conclude that the ELBO provides a lower bound for the natural logarithm of the evidence:

$$\mathcal{L}(q) \;\leq\; \ln\, f_{\boldsymbol{h}}(h) \tag{68.8}$$

The general result (68.7) states that the ELBO and the KL divergence between the posterior distribution and its approximation always add up to the (natural) logarithm of the evidence. By changing $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$, the values of both $\mathcal{L}$ and $D_{\mathrm{KL}}(q(u|h)\,\|\, f(u|h))$ will adjust with one of them increasing and the other decreasing in such a way that their sum remains invariant. For this reason, the minimization problem (68.1) can be replaced by the equivalent maximization problem:

$$\boxed{q^{\star}_{\boldsymbol{u}|\boldsymbol{h}}(u|h) \;\overset{\Delta}{=}\; \underset{q_{\boldsymbol{u}|\boldsymbol{h}}(\cdot)}{\operatorname{argmax}}\; \mathcal{L}(q)} \tag{68.9}$$

where $\mathcal{L}(q)$ represents the objective function that needs to be maximized and is given by (68.6). We therefore focus on solving (68.9).

68.1.3 Neural Network Structure

To maximize $\mathcal{L}(q)$, we need to explain how to approximate the likelihood $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ and the pdf of the latent variable $f_{\boldsymbol{u}}(u)$, both of which are needed in expression (68.6) for the ELBO. Once these factors are estimated, along with $q^{\star}_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$, then we observe from (68.7) that they essentially determine the evidence $f_{\boldsymbol{h}}(h)$. This is because $q^{\star}_{\boldsymbol{u}|\boldsymbol{h}}(u|h) \approx f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ and, hence, $\ln f_{\boldsymbol{h}}(h) \approx \mathcal{L}(q^{\star})$. However, as explained before, we are less interested in evaluating the evidence $f_{\boldsymbol{h}}(h)$, and

more interested in devising a structure that generates samples from it without explicitly computing this pdf. The structure we will use to determine $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ will be called the *encoder*, while the structure we will use to approximate the likelihood $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ will be called the *decoder*. The encoder and decoder will be connected by means of a transition layer that generates realizations from $f_{\boldsymbol{u}}(u)$. The details are as follows.

Approximating the posterior

To begin with, we assume a particular form for $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ and solve (68.9) over this assumed class. Specifically, we will limit $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ to the family of Gaussian distributions. To determine elements from this family, it is sufficient to identify their mean vector and covariance matrix. We denote the mean and covariance matrix of $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ by $\mu(h)$ and $\Sigma(h)$, respectively:

$$q_{\boldsymbol{u}|\boldsymbol{h}}(u|h) \;=\; \mathcal{N}_{\boldsymbol{u}}\Big(\mu(h), \Sigma(h)\Big) \tag{68.10}$$

Observe that the mean and variance quantities are dependent on the feature h due to the conditioning over h. We denote the individual entries of the $P \times 1$ mean vector $\mu(h)$ by

$$\mu(h) \;\triangleq\; \mathrm{col}\Big\{\mu_1(h), \mu_2(h), \ldots, \mu_P(h)\Big\} \tag{68.11}$$

and assume that Σ is diagonal with entries $\{\sigma_p^2(h)\}$ for $p = 1, 2, \ldots, P$. In this way, the problem of learning $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ in (68.9) amounts to learning the moments:

$$\Big\{\mu_1(h), \ldots, \mu_P(h),\, \sigma_1^2(h),\, \ldots,\, \sigma_P^2(h)\Big\} \tag{68.12}$$

where the $\sigma_p^2(h)$ are nonnegative scalars (but not necessarily equal). To learn these parameters, we will employ a feedforward neural network whose input receives the given feature vectors $\{h_n\}$ and whose output generates the mean and variance parameters. Depending on the application at hand, we may employ a convolutional neural network at this stage. However, we will continue our discussion by considering a traditional feedforward neural network to illustrate the main idea. We explain in the sequel how to train the network in order to learn the parameters. Usually, in practice, we train the encoder to estimate the logarithm of the variance factors rather than the variances themselves for enhanced numerical stability, i.e., the encoder will generate the log-variances:

$$a_p(h) \;\triangleq\; \ln \sigma_p^2(h), \quad p = 1, 2, \ldots, P \tag{68.13}$$

This step ensures that the variances recovered from the $\{a_p(\cdot)\}$ are always nonnegative since

$$\sigma_p^2(h) \;=\; e^{a_p(h)}, \qquad \sigma_p(h) = e^{a_p(h)/2} \tag{68.14}$$

Just like the individual means $\mu_p(h)$ are collected into the $P \times 1$ mean vector $\mu(h)$, we will also collect the log-variances into a vector of similar size and denote

it by

$$a(h) \triangleq \begin{bmatrix} a_1(h) \\ a_2(h) \\ \vdots \\ a_P(h) \end{bmatrix}, \quad (P \times 1) \tag{68.15}$$

Figure 68.2 shows the encoder structure with $M = 3$ attributes at the input layer, latent variable u of size $P = 2$, and 3 hidden layers. We denote the total number of layers in the encoder, including its input and output layers, by L_e. Hence, for the example in the figure, we have $L_e = 5$. The encoder generates the vectors $\{\mu(h), a(h)\}$ at its output.

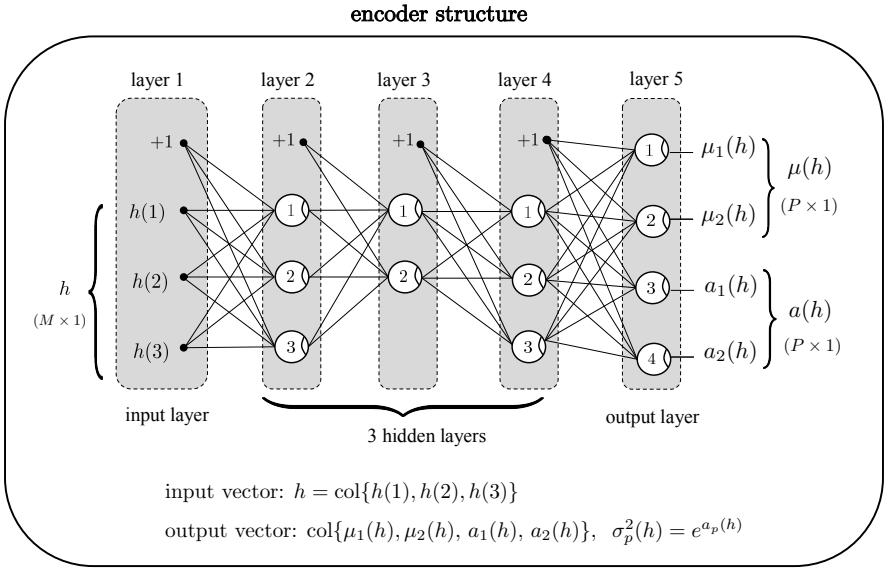

Figure 68.2 An encoder structure to approximate the posterior $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ with $M = 3$ attributes at the input layer (i.e., $h \in \mathbb{R}^3$), 3 hidden layers, and latent variables of size $P = 2$ so that there are 4 nodes at the output layer.

Generating latent variables

Using the mean and variance parameters $\{\mu(h), a(h)\}$, we can generate realizations for the latent variables $\boldsymbol{u}$ since, by assumption,

$$\boldsymbol{u}|\boldsymbol{h} \sim \mathcal{N}_{\boldsymbol{u}}\Big(\mu(h), \Sigma(h)\Big) \tag{68.16}$$

In principle, we could sample directly from this Gaussian distribution. However, as we are going to see, this approach is difficult to incorporate into the training of the encoder-decoder structure by means of the backpropagation procedure. This is because the sampling operation is not differentiable during the backward

stage of the learning process. An alternative approach to generate realizations for $\boldsymbol{u}$ is to rely on a "*re-parametrization trick*." We first generate a realization ϵ from a standard Gaussian distribution, i.e., $\boldsymbol{\epsilon} \sim \mathcal{N}_{\boldsymbol{\epsilon}}(0, I_P)$, and then set

$$\boxed{u = \mu(h) \; + \; \Sigma^{1/2}(h)\epsilon} \tag{68.17}$$

where the entries of $\Sigma^{1/2}(h)$ are the standard deviations $\{\sigma_p(h)\}$. In this way, we relate the realization u directly to the outputs $\{\mu(h), a(h)\}$ of the encoder network, which will facilitate the backpropagation step, as explained in the sequel. Figure 68.3 illustrates this construction, which maps a generic 2−dimensional standard Gaussian variable $\epsilon = \text{col}\{\epsilon(1), \epsilon(2)\}$ into a 2−dimensional latent variable $u = \text{col}\{u(1), u(2)\}$.

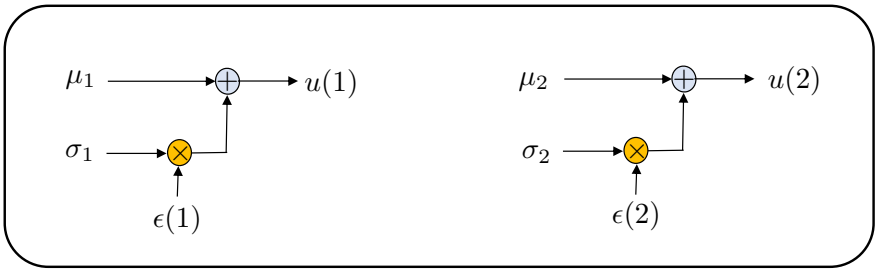

Figure 68.3 Generation of a 2−dimensional latent variable u (i.e., $P = 2$). The structure maps a generic 2−dimensional standard Gaussian variable $\epsilon = \text{col}\{\epsilon(1), \epsilon(2)\}$ into the 2−dimensional latent variable $u = \text{col}\{u(1), u(2)\}$.

Approximating the likelihood

Once the encoder and the re-parameterization trick are combined, we end up with a structure that maps an input feature vector h into a latent-variable realization u. Next, we examine the task of approximating the *reverse distribution* (or likelihood) $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$. Here, instead of approximating the pdf directly, we will construct a second neural network structure that attempts to replicate h — see Fig. 68.4. By doing so, we end up with a neural network structure that is able to generate realizations that arise from the conditional distribution $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$. The input to this network will be the realization u corresponding to h, and the output will be an approximation $\widehat{h}$ for the same h. The mapping from u to $\widehat{h}$ in this second network will be nonlinear and denoted generically by

$$\widehat{h} = g(u) \tag{68.18}$$

for some nonlinear mapping $g(\cdot)$. Figure 68.4 illustrates the structure of the decoder with latent variables of size $P = 2$ at the input layer, 3 hidden layers, and $M = 3$ nodes at the output to generate $\widehat{h}$. We will be using the notation

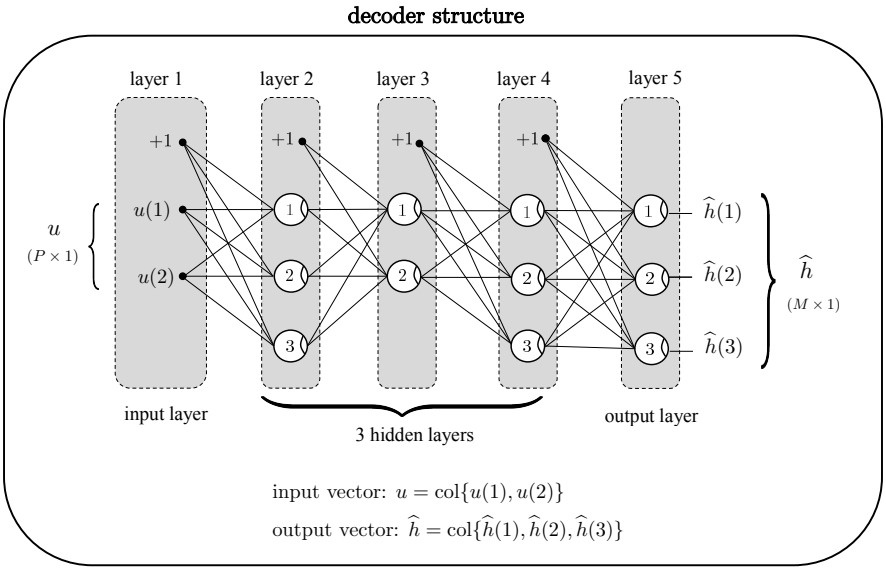

Figure 68.4 A decoder structure to approximate the likelihood $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ with $P = 2$ latent variables at the input layer (i.e., $u \in \mathbb{R}^2$), $M = 3$ nodes at the output layer to generate $\widehat{h}$, and 3 hidden layers.

L_d to refer to the total number of layers in the decoder structure, including its input and output layers. Hence, for the example in the figure we have $L_d = 5$.

If we combine the encoder, decoder, and the generation of latent variables into a single diagram we obtain the structure shown in Fig. 68.5. The purpose of the overall structure is to reproduce the input h, i.e., to "match" $\widehat{h}$ to h. We can pursue this objective by minimizing a certain risk function, as we explain next.

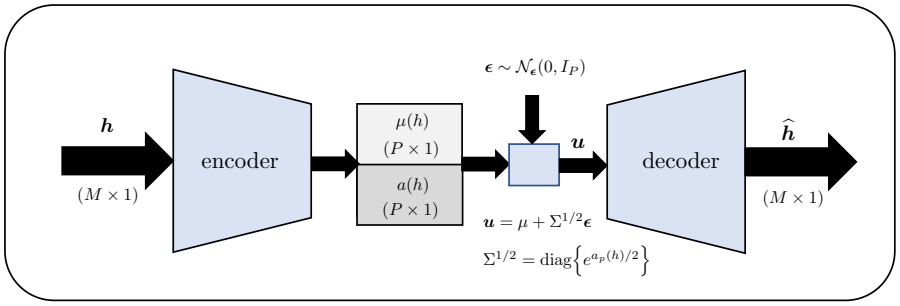

Figure 68.5 Diagram representation of the variational autoencoder (VAE) involving a cascade of encoder and decoder structures with the intermediate step of generating the latent variables $\boldsymbol{u}$ from standard Gaussian realizations ϵ.

Remark 68.1. (Distribution of the latent variables.) Expression (68.16) assumes $\boldsymbol{u}$ is conditionally Gaussian given h. But what about the marginal distribution $f_{\boldsymbol{u}}(u)$?

The nonlinear mapping from u to $\widehat{h}$ in the decoder allows us to assume, without loss in generality, that this *marginal* pdf is a standard Gaussian with zero mean and unit variance, i.e., we can assume

$$\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P) \tag{68.19}$$

This is because, as is known, we can transform a standard Gaussian-distributed random variable to other continuous distributions by means of suitable nonlinear transformations — see Prob. 68.1. Therefore, even if the actual pdf for $\boldsymbol{u}$ is non-Gaussian, the overall nonlinear mapping by the decoder from the assumed Gaussian-distributed $\boldsymbol{u}$ to $\widehat{\boldsymbol{h}}$ will be able to capture the model that maps the *real* $\boldsymbol{u}$ to $\widehat{\boldsymbol{h}}$. This is because the decoder consists of multiple neural layers, and we can regard the initial layers as performing the mapping from the Gaussian-distributed input $\boldsymbol{u}$ to the "actual" latent distribution for $\boldsymbol{u}$, while the remaining layers assist with the generation of $\widehat{\boldsymbol{h}}$.

■

68.2 TRAINING VARIATIONAL AUTOENCODERS

Since both $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ and $f_{\boldsymbol{u}}(u)$ are assumed to be Gaussian-distributed according to (68.16) and (68.19), we can compute the last term in (68.7) by appealing to result (6.65):

$$D_{\mathrm{KL}}\Big(q_{\boldsymbol{u}|\boldsymbol{h}}(u|h) \,\|\, f_{\boldsymbol{u}}(u)\Big) \;= \frac{1}{2}\Big\{-\ln\big(\det(\Sigma(h))\big) - P + \mathrm{Tr}(\Sigma(h)) + \|\mu(h)\|^2\Big\} \tag{68.20}$$

In VAE implementations, the reverse conditional pdf $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ is also approximated by a Gaussian distribution, say,

$$\boldsymbol{h}|\boldsymbol{u} \;\sim\; \mathcal{N}_{\boldsymbol{h}}(g(u), I_M) \tag{68.21}$$

for some mean vector $g(u)$. The decoder focuses on learning $g(u)$ at its output, which we are denoting by $\widehat{h}$. This is because, as we know from mean-square-error inference, the optimal estimate for $\boldsymbol{h}$ conditioned on u is the mean $g(u)$ so that $\widehat{h} = g(u)$. It follows from (68.21) that

$$\ln f_{\boldsymbol{h}|\boldsymbol{u}}(h|u) \;\propto -\frac{1}{2}\|h - g(u)\|^2 \;=\; -\frac{1}{2}\|h - \widehat{h}\|^2 \tag{68.22}$$

Therefore, averaging over multiple feature realizations allows us to approximate the first term in the ELBO expression (68.6) by

$$\mathbb{E}_q\Big(\ln f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)\Big) \approx -\frac{1}{2N}\sum_{n=0}^{N-1}\|h_n - \widehat{h}_n\|^2 \tag{68.23}$$

Combining with (68.20) we find that the design of the encoder-decoder structure will be based on maximizing the following expression over the parameters of the

neural components (weights and biases):

$$\mathcal{L}(q) \approx -\frac{1}{N}\sum_{n=0}^{N-1}\left\{\|h_n - \widehat{h}_n\|^2 \;+\; \sum_{p=1}^{P}\Big(\mu_p^2(h_n) + \sigma_p^2(h_n) - \ln\sigma_p^2(h_n)\Big)\right\} \tag{68.24}$$

We mentioned before that we will be training the encoder to estimate the logarithm of the variance factors rather than the variances themselves. For this reason, we rewrite (68.24) as

$$\mathcal{L}(q) \approx -\frac{1}{N}\sum_{n=0}^{N-1}\left\{\|h_n - \widehat{h}_n\|^2 \;+\; \sum_{p=1}^{P}\Big(\mu_p^2(h_n) + e^{a_p(h_n)} - a_p(h_n)\Big)\right\} \tag{68.25}$$

Notation

We adopt the same notational convention from Chapter 65 on neural networks with some minimal adjustments, such as attaching the subscripts e and d to variables within the encoder and decoder parts, respectively. Figure 68.6 shows an implementation for a variational autoencoder involving a single hidden layer in the encoder and a single hidden layer in the decoder. For the example in the figure, we have that the number of layers in the encoder and decoder is $L_e = 3$ and $L_d = 3$, respectively.

In general, a variational autoencoder will consist of L_e total layers within the encoder and L_d total layers within the decoder so that the encoder will have $L_e - 2$ hidden layers while the decoder will have $L_d - 2$ hidden layers. We denote the weight matrices and bias vectors within the encoder by

$$\Big\{(W_{1,e},\theta_{1,e}),\,(W_{2,e},\theta_{2,e}),\,\ldots,\,(W_{L_e-1,e},\theta_{L_e-1,e})\Big\} \tag{68.26}$$

with a subscript e and sizes

$$W_{\ell,e} : n_{\ell,e} \times n_{\ell+1,e}, \quad \theta_{\ell,e} : n_{\ell+1,e} \times 1 \tag{68.27}$$

and those within the decoder by

$$\Big\{(W_{1,d},\theta_{1,d}),\,(W_{2,d},\theta_{2,d}),\,\ldots,(W_{L_d-1,d},\theta_{L_d-1,d})\Big\} \tag{68.28}$$

with a subscript d and sizes

$$W_{\ell,d} : n_{\ell,d} \times n_{\ell+1,d}, \quad \theta_{\ell,d} : n_{\ell+1,d} \times 1 \tag{68.29}$$

We also denote the pre- and post-activation vectors at the internal layers of the encoder by

$$\Big\{y_{1,e} = h,\,(z_{2,e},y_{2,e}),\,(z_{3,e},y_{3,e}),\ldots,(z_{L_e,e},y_{L_e,e})\Big\} \tag{68.30}$$

with a subscript e and sizes

$$y_{\ell,e} : n_{\ell,e} \times 1, \quad z_{\ell,e} : n_{\ell,e} \times 1, \;\; n_{1,e} = M \tag{68.31}$$

The entries of the $2P \times 1$ output vector in the encoder are

$$y_{L_e,e} = \mathrm{col}\Big\{\mu_1(h),\ldots,\mu_P(h),a_1(h),\ldots,a_P(h)\Big\} \tag{68.32}$$

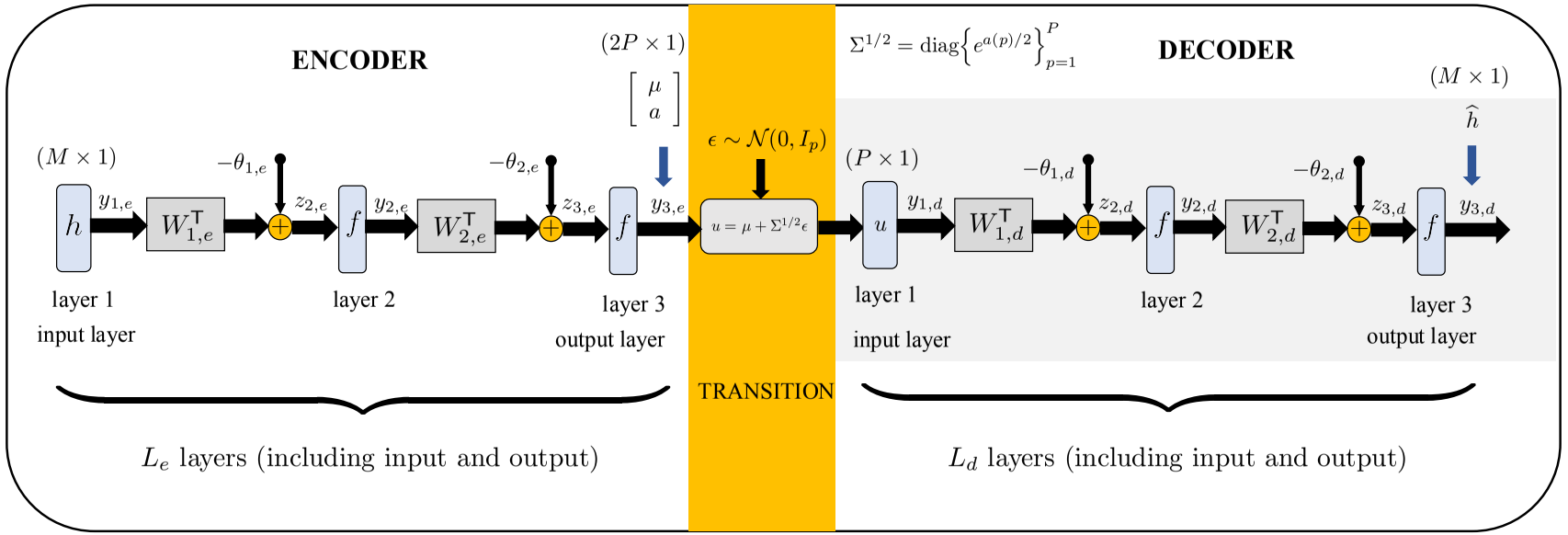

Figure 68.6 Block diagram representation of the cascade of encoder and decoder neural networks with a transition layer, and a single hidden layer in each. The weight matrices and bias vectors within the encoder and decoder are denoted by $\{W_{\ell,e}, \theta_{\ell,e}\}$ and $\{W_{\ell,d}, \theta_{\ell,d}\}$, with subscripts e and d added. The encoder has L_e total layers, including its input and output layers. The decoder has L_d total layers, including its input and output layers.

so that $n_{L_e,e} = 2P$. Likewise, we denote the pre- and post-activation vectors at the internal layers of the decoder by

$$\Big\{y_{1,d} = u,\ (z_{2,d}, y_{2,d}),\ (z_{3,d}, y_{3,d}), \ldots, (z_{L_d,d}, y_{L_d,d})\Big\} \tag{68.33}$$

with a subscript d and sizes

$$y_{\ell,d} : n_{\ell,d} \times 1, \qquad z_{\ell,d} : n_{\ell,d} \times 1, \ \ n_{1,d} = P \tag{68.34}$$

The entries of the input vector are

$$y_{1,d} = u \ \triangleq \ \mu + \Sigma^{1/2}\epsilon \tag{68.35}$$

and the entries of the output vector are

$$y_{L_d,d} = \widehat{h}, \quad n_{L_d} = M \tag{68.36}$$

68.2.1 Empirical Risk Minimization

We refer to the overall weights and biases within the VAE structure by the generic notation (W, θ), and incorporate additional regularization factors over the weight matrices into the risk function. By doing so, we are reduced to *minimizing* the following empirical risk:

$$\begin{aligned}\mathcal{P}(W,\theta) \ \triangleq \ \Bigg\{ &\sum_{\ell=1}^{L_e-1} \rho\|W_{\ell,e}\|_{\mathrm{F}}^2 + \sum_{\ell=1}^{L_d-1} \rho\|W_{\ell,d}\|_{\mathrm{F}}^2 + \frac{1}{N}\sum_{n=0}^{N-1} \|h_n - \widehat{h}_n\|^2 + \\ &\frac{1}{N}\sum_{n=0}^{N-1}\sum_{p=1}^{P}\Big(\mu_p^2(h_n) + e^{a_p(h_n)} - a_p(h_n)\Big)\Bigg\}\end{aligned} \tag{68.37}$$

We can rewrite the empirical risk more compactly by resorting to the vectors $\mu(h_n)$ and $a(h_n)$. We introduce the vector notation:

$$e^{a(h_n)} \ \triangleq \ \mathrm{col}\Big\{e^{a_1(h_n)},\ e^{a_2(h_n)}, \ldots, e^{a_P(h_n)}\Big\} \tag{68.38}$$

That is, the exponential function applied to the vector $a(h_n)$ is a column vector with the exponentiation applied to the individual entries of $a(h_n)$; this is similar to the convention we adopted for activation functions applied to a vector argument. Using this notation, we rewrite (68.37) in the form

$$\begin{aligned}\mathcal{P}(W,\theta) \ \triangleq \ \Bigg\{ &\sum_{\ell=1}^{L_e-1} \rho\|W_{\ell,e}\|_{\mathrm{F}}^2 + \sum_{\ell=1}^{L_d-1} \rho\|W_{\ell,d}\|_{\mathrm{F}}^2 + \frac{1}{N}\sum_{n=0}^{N-1} \|h_n - \widehat{h}_n\|^2 + \\ &\frac{1}{N}\sum_{n=0}^{N-1}\Big(\|\mu(h_n)\|^2 + \mathbb{1}^{\mathsf{T}} e^{a(h_n)} - \mathbb{1}^{\mathsf{T}} a(h_n)\Big)\Bigg\}\end{aligned} \tag{68.39}$$

The training of the variational autoencoder will be similar to the training of a traditional neural network, with both feedforward and backward propagation steps. This is because the variational autoencoder consists of two cascaded

networks representing the encoder and decoder. The main difference relative to what we have seen before while training neural networks is the presence of a *transition* layer, which maps the output of the encoder to the input of the decoder. The transition layer involves generating samples $\boldsymbol{\epsilon} \sim \mathcal{N}_{\boldsymbol{\epsilon}}(0, I_P)$ from a standard Gaussian distribution and using them to produce realizations for the latent variable, u. Therefore, during the forward and backpropagation steps we need to adjust the variables as we move through this transition layer. Everything else remains practically invariant relative to what we already know from training neural networks. The details are spelled out in the sequel.

68.2.2 Feedforward Propagation

Given a feature vector $h \in \mathbb{R}^M$ at the input of the encoder, it is easy to describe the forward propagation of this vector across the encoder and decoder components, as shown in listing (68.41) where $f(\cdot)$ denotes the activation functions. Observe how the feedforward step involves two loops: one over layers of the encoder and a second over layers of the decoder. The transition phase performs the calculations that link the output $\{\mu(h), a(h)\}$ of the encoder to the input u to the decoder. Note that we are denoting the output of the feedforward phase by $\widehat{h}$, and the corresponding pre-activation signal by z so that $\widehat{h} = f(z)$.

68.2.3 Sensitivity Factors

In order to train the autoencoder, we need to evaluate the gradients of the empirical risk $\mathcal{P}(W, \theta)$ defined by (68.39) relative to the individual entries of the weight matrices and bias vectors within the encoder and decoder. We adjust the derivation from Sec. 65.4 to the current context. The main difference will be in handling the propagation of signals through the transition layer linking the encoder and decoder. To simplify the notation, we drop the subscript n from all signals and reinstate it later when we list the final algorithm.

Sensitivity factors through the decoder

We first define sensitivity vectors and propagate them backward through the layers of the decoder. Subsequently, we perform a transition and continue the propagation through the layers of the encoder.

We start with the $M \times 1$ sensitivity vector associated with the last layer of the decoder. Its entries are given by

$$\delta_{L_d,d}(j) = \frac{\partial \|h - \widehat{h}\|^2}{\partial z(j)}, \quad j = 1, 2, \ldots, M \tag{68.40}$$

Feedforward propagation through the variational autoencoder

start with $y_{1,e} = h$.

(propagation through encoder)
repeat for $\ell = 1, \ldots, L_e - 1$:
$\quad z_{\ell+1,e} = W_{\ell,e}^{\mathsf{T}} y_{\ell,e} - \theta_{\ell,e}$
$\quad y_{\ell+1,e} = f\,(z_{\ell+1,e})$
end

$$\text{partition } \; y_{L_e,e} \; \triangleq \; \left[\frac{\mu(h)}{a(h)}\right], \quad (2P \times 1)$$
$$\Sigma^{1/2}(h) = \text{diag}\Big\{e^{a_1(h)/2},\, e^{a_2(h)/2},\, \ldots,\, e^{a_P(h)/2}\Big\} \tag{68.41}$$

(*transition layer)*
generate $\epsilon \sim \mathcal{N}_{\boldsymbol{\epsilon}}(0, I_P)$
$u = \mu(h) + \Sigma^{1/2}(h)\epsilon$

(propagation through decoder)
set $y_{1,d} = u$
repeat for $\ell = 1, \ldots, L_d - 1$:
$\quad z_{\ell+1,d} = W_{\ell,d}^{\mathsf{T}} y_{\ell,d} - \theta_{\ell,d}$
$\quad y_{\ell+1,d} = f\,(z_{\ell+1,d})$
end
$z = z_{L_d,d}$
$\widehat{h} = y_{L_d,d}$

where we are attaching the subscript d to indicate that this sensitivity vector is related to the decoder stage. Using the chain rule for differentiation gives

$$\begin{aligned}
\delta_{L_d,d}(j) &= \sum_{m=1}^{M} \frac{\partial \|h - \widehat{h}\|^2}{\partial \widehat{h}(m)} \, \frac{\partial \widehat{h}(m)}{\partial z(j)} \\
&= \sum_{m=1}^{M} 2(\widehat{h}(m) - h(m)) \, \frac{\partial \widehat{h}(m)}{\partial z(j)} \\
&= 2\Big(\widehat{h}(j) - h(j)\Big) f'(z(j))
\end{aligned} \tag{68.42}$$

since only $\widehat{h}(j)$ depends on $z(j)$ through the relation $\widehat{h}(j) = f(z(j))$. Consequently, using the Hadamard product notation we can write

$$\boxed{\delta_{L_d,d} \;=\; 2(\widehat{h} - h) \odot f'(z)} \qquad (\textbf{terminal sensitivity vector}) \tag{68.43}$$

Next we evaluate the sensitivity vectors $\delta_{\ell,d}$ for the earlier layers within the decoder, i.e., for $\ell = L_d - 1, \ldots, 2$. Since we are performing calculations within

the decoder layers, we will continue to attach the subscript d to all signals and combination weights that appear in the calculations. We can relate $\delta_{\ell,d}$ to $\delta_{\ell+1,d}$ as follows:

$$\begin{aligned}\delta_{\ell,d}(j) &\triangleq \frac{\partial\|h-\widehat{h}\|^2}{\partial z_{\ell,d}(j)} \\ &= \sum_{k=1}^{n_{\ell+1,d}} \frac{\partial\|h-\widehat{h}\|^2}{\partial z_{\ell+1,d}(k)} \frac{\partial z_{\ell+1,d}(k)}{\partial z_{\ell,d}(j)} \\ &= \sum_{k=1}^{n_{\ell+1,d}} \delta_{\ell+1,d}(k) \frac{\partial z_{\ell+1,d}(k)}{\partial z_{\ell,d}(j)} \end{aligned} \tag{68.44}$$

where the right-most term involves differentiating the output of node k in layer $\ell+1$ relative to the (pre-activation) output of node j in the previous layer, ℓ. The summation in the second equality results from the chain rule of differentiation since the entries of $\widehat{h}$ are generally dependent on the $\{z_{\ell+1,d}(k)\}$. The two signals $z_{\ell,d}(j)$ and $z_{\ell+1,d}(k)$ are related by the combination coefficient $w_{jk,d}^{(\ell)}$ present in the combination matrix $W_{\ell,d}$ since

$$z_{\ell+1,d}(k) \;=\; f\left(z_{\ell,d}(j)\right) w_{jk,d}^{(\ell)} \;+\; \text{terms independent of } w_{jk,d}^{(\ell)} \tag{68.45}$$

It follows, as was the case with (65.61), that

$$\boxed{\delta_{\ell,d} = f'\left(z_{\ell,d}\right) \odot \left(W_{\ell,d}\,\delta_{\ell+1,d}\right), \;\; \ell = L_d - 1, \ldots, 3, 2} \tag{68.46}$$

The activation function whose derivative appears in (68.46) is the one associated with the $\ell-$th layer of the decoder.

Sensitivity factors through the encoder

In a similar vein, let $\delta_{\ell,e}$ denote the sensitivity vectors within the encoder with entries:

$$\delta_{\ell,e}(j) \;\triangleq\; \frac{\partial\|h-\widehat{h}\|^2}{\partial z_{\ell,e}(j)}, \;\; \ell = L_e, L_e-1, \ldots, 2 \tag{68.47}$$

A similar argument will show that

$$\boxed{\delta_{\ell,e} = f'\left(z_{\ell,e}\right) \odot \left(W_{\ell,e}\,\delta_{\ell+1,e}\right), \;\; \ell = L_e - 1, \ldots, 3, 2} \tag{68.48}$$

To enable this backward recursion, we need to evaluate its boundary value $\delta_{L_e,e}$. To do so, we need to show how to transfer the end value $\delta_{2,d}$ from the decoder stage (68.46) through the transition layer to get the initial value $\delta_{L_e,e}$ for the encoder stage.

Sensitivity factor at the transition

Thus, note that by definition

$$\begin{aligned}\delta_{L_e,e}(j) &= \frac{\partial\|h-\widehat{h}\|^2}{\partial z_{L_e,e}(j)}, \quad j=1,2,\ldots,2P \\ &= \sum_{i=1}^{n_{2,d}} \frac{\partial\|h-\widehat{h}\|^2}{\partial z_{2,d}(i)} \, \frac{\partial z_{2,d}(i)}{\partial z_{L_e,e}(j)} \\ &= \sum_{i=1}^{n_{2,d}} \delta_{2,d}(i) \, \frac{\partial z_{2,d}(i)}{\partial z_{L_e,e}(j)} \\ &= \sum_{i=1}^{n_{2,d}} \delta_{2,d}(i) \, \left(\sum_{p=1}^{P} \frac{\partial z_{2,d}(i)}{\partial u(p)} \frac{\partial u(p)}{\partial z_{L_e,e}(j)} \right) \end{aligned} \tag{68.49}$$

Recall that the $\{z_{L_e,e}(j)\}$ represent the pre-activation signals prior to the generation of the outputs of the encoder, which consist of the mean and variance parameters, i.e.,

$$\mu_p = f(z_{L_e,e}(p)) \tag{68.50}$$

$$a_p = f(z_{L_e,e}(P+p)), \;\; p=1,2,\ldots,P \tag{68.51}$$

in terms of the activation function at the output layer of the encoder, and where the entries of u are given by

$$\left\{ \begin{array}{ll} u(1) = \mu_1 + \sigma_1 \epsilon(1), & \sigma_1 = e^{a_1/2} \\ u(2) = \mu_2 + \sigma_2 \epsilon(2), & \sigma_2 = e^{a_2/2} \\ \;\;\vdots \;\; = \;\; \vdots & \\ u(P) = \mu_P + \sigma_P \epsilon(P), & \sigma_P = e^{a_P/2} \end{array} \right. \tag{68.52}$$

It follows that

$$\frac{\partial u(p)}{\partial z_{L_e,e}(j)} = \left\{ \begin{array}{ll} f'(z_{L_e,e}(p)), & \text{when } p=j \text{ and } 1 \le j \le P \\ \frac{1}{2}\epsilon(p) e^{a_p/2} f'(z_{L_e,e}(j)), & \text{when } p=j-P \text{ and } P+1 \le j \le 2P \\ 0, & \text{otherwise} \end{array} \right. \tag{68.53}$$

Moreover, since by construction

$$z_{2,d}(i) = \sum_{p=1}^{P} y_{1,d}(p) w_{pi,d}^{(1)} = \sum_{p=1}^{P} u(p) w_{pi,d}^{(1)} \tag{68.54}$$

and $y_{1,d} = u$, we conclude that

$$\frac{\partial z_{2,d}(i)}{\partial u(p)} = w_{pi,d}^{(1)} \tag{68.55}$$

Let $j' = j - P$ for $P+1 \le j \le 2P$. As a result, the index j' runs over $1 \le j' \le P$. Substituting into (68.49) we get

$$\delta_{L_e,e}(j) = \begin{cases} f'(z_{L_e,e}(j)) \left(\displaystyle\sum_{i=1}^{n_{2,d}} \delta_{2,d}(i) w_{ji,d}^{(1)} \right), & 1 \le j \le P \\ \frac{1}{2}\epsilon(j') e^{a_{j'}/2} f'(z_{L_e,e}(j)) \left(\displaystyle\sum_{i=1}^{n_{2,d}} \delta_{2,d}(i) w_{j'i,d}^{(1)} \right), & P+1 \le j \le 2P \end{cases} \tag{68.56}$$

Introduce the $2P \times 1$ vector

$$s \triangleq \operatorname{col}\left\{1, \ldots, 1, \frac{1}{2}\epsilon(1)e^{a_1/2}, \ldots, \frac{1}{2}\epsilon(P)e^{a_P/2}\right\} \tag{68.57}$$

with P leading unit entries. Then, it is easy to verify that the expression for the boundary sensitivity vector is given by

$$\boxed{\delta_{L_e,e} = \Big(s \odot f'(z_{L_e,e})\Big) \odot \begin{bmatrix} W_{1,d}\,\delta_{2,d} \\ W_{1,d}\,\delta_{2,d} \end{bmatrix}} \tag{68.58}$$

This relation tells us how to propagate the sensitivity factor $\delta_{2,d}$ at the leftmost end of the decoder through the transition layer in order to obtain the boundary sensitivity factor at the rightmost end of the encoder.

Second sensitivity factor

There is a second term in the risk function (68.39) for which we also need to evaluate sensitivity factors. Thus, let

$$J(\mu, a) \triangleq \|\mu(h)\|^2 + \mathbb{1}^{\mathsf{T}} e^{a(h)} - \mathbb{1}^{\mathsf{T}} a(h) \tag{68.59}$$

whose terms are affected by the weights and biases within the encoder stage only. We define the $2P \times 1$ sensitivity vector associated with the last layer of the encoder:

$$\lambda_{L_e,e}(j) = \frac{\partial J(\mu, a)}{\partial z_{L_e,e}(j)}, \quad j = 1, 2, \ldots, 2P \tag{68.60}$$

Using the chain rule for differentiation gives

$$\lambda_{L_e,e}(j) = \sum_{p=1}^{2P} \frac{\partial J(\mu, a)}{\partial y_{L_e,e}(p)} \frac{\partial y_{L_e,e}(p)}{\partial z_{L_e,e}(j)} = \frac{\partial J(\mu, a)}{\partial y_{L_e,e}(j)} f'(z_{L_e,e}(j)) \tag{68.61}$$

Recall from (68.32) that the entries of the $2P \times 1$ vector $y_{L_e,e}$ are given by the mean and log-variance parameters. It follows that

$$\frac{\partial J(\mu, a)}{\partial y_{L_e,e}(j)} = \begin{cases} 2\mu_j, & j = 1, 2, \ldots, P \\ e^{a_j} - 1, & j = P+1, \ldots, 2P \end{cases} \tag{68.62}$$

We collect these partial derivatives into the $2P \times 1$ vector:

$$x \triangleq \operatorname{col}\Big\{2\mu_1, \ldots, 2\mu_P, (e^{a_1}-1), \ldots, (e^{a_P}-1)\Big\} \tag{68.63}$$

Then, we conclude that

$$\boxed{\lambda_{L_e,e} = x \odot f'(z_{L_e,e})} \quad (2P \times 1) \tag{68.64}$$

Next we evaluate the sensitivity vectors $\lambda_{\ell,e}$ for the earlier layers within the encoder, i.e., for $\ell = L_e - 1, \ldots, 2$. We can relate $\lambda_{\ell,e}$ to $\lambda_{\ell+1,e}$ as follows:

$$\begin{aligned}
\lambda_{\ell,e}(j) &\triangleq \frac{\partial J(\mu,a)}{\partial z_{\ell,e}(j)} \\
&= \sum_{k=1}^{n_{\ell+1,e}} \frac{\partial J(\mu,a)}{\partial z_{\ell+1,e}(k)} \frac{\partial z_{\ell+1,e}(k)}{\partial z_{\ell,e}(j)} \\
&= \sum_{k=1}^{n_{\ell+1,e}} \lambda_{\ell+1,e}(k) \frac{\partial z_{\ell+1,e}(k)}{\partial z_{\ell,e}(j)}
\end{aligned} \tag{68.65}$$

The two signals $z_{\ell,e}(j)$ and $z_{\ell+1,e}(k)$ are related by the combination coefficient $w_{jk,e}^{(\ell)}$ present in the combination matrix $W_{\ell,e}$ since

$$z_{\ell+1,e}(k) \;=\; f\left(z_{\ell,e}(j)\right) w_{jk,e}^{(\ell)} \;+\; \text{terms independent of } w_{jk,e}^{(\ell)} \tag{68.66}$$

It follows, as was the case with (65.61), that

$$\boxed{\lambda_{\ell,e} = f'\left(z_{\ell,e}\right) \odot \left(W_{\ell,e}\,\lambda_{\ell+1,e}\right), \;\; \ell = L_e - 1, \ldots, 3, 2} \tag{68.67}$$

Comparing this recursion with (68.48) we observe that, if desired, we can group together the δ and λ factors within the encoder and update their sum together as follows. Let

$$\beta_{\ell,e} \triangleq \delta_{\ell,e} + \lambda_{\ell,e} \tag{68.68}$$

Then,

$$\beta_{\ell,e} = f'\left(z_{\ell,e}\right) \odot \left(W_{\ell,e}\,\beta_{\ell+1,e}\right), \;\; \ell = L_e - 1, \ldots, 3, 2 \tag{68.69}$$

with boundary condition

$$\beta_{L_e,e} = \left\{ x + \left(s \odot \begin{bmatrix} W_{1,d}\,\delta_{2,d} \\ W_{1,d}\,\delta_{2,d} \end{bmatrix} \right) \right\} \odot f'(z_{L_e,e}) \tag{68.70}$$

For convenience, we will keep the δ and λ factors separate within the encoder to highlight the fact that one of them arises from the presence of the quadratic term $\|h_n - \widehat{h}_n\|^2$ while the other arises from $J(\mu, a)$; if these factors are changed in the empirical risk, then the updates for δ and λ will need to be adjusted accordingly. Figure 68.7 provides a block diagram representation of the backward updates for the sensitivity factors through the decoder and encoder stages, assuming $L_e = 4$

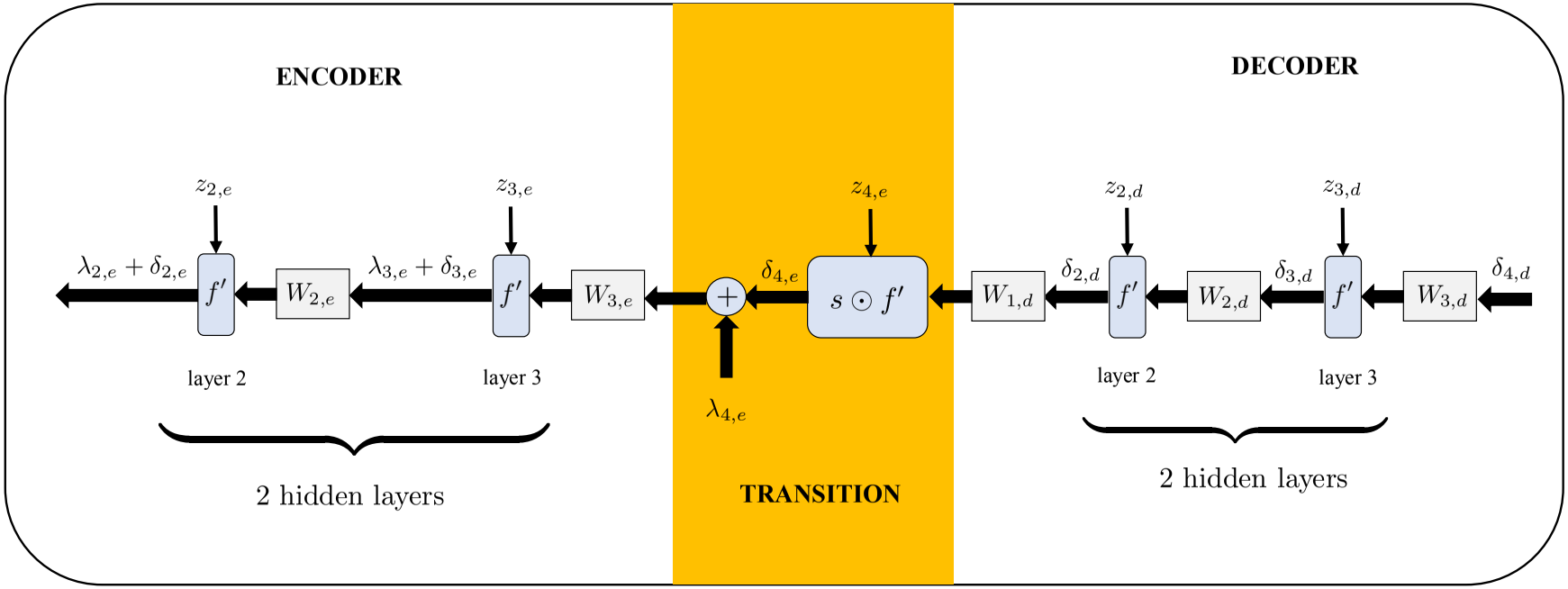

Figure 68.7 Block diagram representation of the backward updates for the sensitivity factors through the decoder and encoder stages, assuming $L_e = 4$ layers within the encoder and $L_d = 4$ layers within the decoder (i.e., each stage has two hidden layers).

layers within the encoder and $L_d = 4$ layers within the decoder (i.e., each stage has two hidden layers).

Expressions for the gradients

We can follow the same arguments from Sec. 65.4.3 to verify similarly that for the decoder we have:

$$\frac{\partial \|h - \widehat{h}\|^2}{\partial W_{\ell,d}} = y_{\ell,d}\delta_{\ell+1,d}^{\mathsf{T}}, \quad \ell = L_d - 1, \ldots, 2, 1 \tag{68.71}$$

$$\frac{\partial \|h - \widehat{h}\|^2}{\partial \theta_{\ell,d}} = -\delta_{\ell+1,d}^{\mathsf{T}}, \quad \ell = L_d - 1, \ldots, 2, 1 \tag{68.72}$$

while for the encoder:

$$\frac{\partial \|h - \widehat{h}\|^2}{\partial W_{\ell,e}} = y_{\ell,e}\delta_{\ell+1,e}^{\mathsf{T}}, \quad \ell = L_e - 1, \ldots, 2, 1 \tag{68.73}$$

$$\frac{\partial \|h - \widehat{h}\|^2}{\partial \theta_{\ell,e}} = -\delta_{\ell+1,e}, \quad \ell = L_e - 1, \ldots, 2, 1 \tag{68.74}$$

and

$$\frac{\partial J(\mu, a)}{\partial W_{\ell,e}} = y_{\ell,e}\lambda_{\ell+1,e}^{\mathsf{T}}, \quad \ell = L_e - 1, \ldots, 2, 1 \tag{68.75}$$

$$\frac{\partial J(\mu, a)}{\partial \theta_{\ell,e}} = -\lambda_{\ell+1,e}, \quad \ell = L_e - 1, \ldots, 2, 1 \tag{68.76}$$

68.2.4 Backpropagation Algorithm

We can now use the forward and backward recursions to train the variational autoencoder by writing down a stochastic-gradient implementation with step-size $\mu > 0$. In this implementation, either one randomly-selected feature vector h_n may be used per iteration or a mini-batch of size B of randomly selected feature vectors $\{h_b\}$ may be used. We describe the implementation in the batch mode. By setting $B = 1$, we recover the version with one data point per iteration. We also restore the subscript n to index the data; and we add a new subscript m as iteration index. Since we are already using two subscripts for each weighting matrix, such as $W_{\ell,e}$, where the second subscript indicates whether the matrix belongs to the encoder or decoder sections, we will adopt the notation $W_{\ell e,m}$ to indicate that this is the $\ell-$th weight matrix in the encoder that is computed at iteration m. Note that we are blending the subscript e into the index ℓ of the weight matrix to avoid repeated commas in the subscript notation (such as writing $W_{\ell,e,m}$). Likewise, we will write $W_{\ell d,m}$, $\theta_{\ell e,m}$, and $\theta_{\ell d,m}$ at iteration

m, where the subscripts e and d refer to layers within the encoder and decoder segments.

In a mini-batch implementation, a collection of $B-$randomly selected feature vectors $\{h_b\}$ are used to approximate the gradients of the risk function. In the statement of the algorithm we assume the following conditions:

$$\begin{cases} \bullet \text{ encoder and decoder structures with } L_e \text{ and } L_d \text{ layers, respectively;} \\ \bullet \text{ output of encoder has dimensions } 2P \times 1; \\ \bullet \text{ random initial parameters } \{\boldsymbol{W}_{\ell e,-1}, \boldsymbol{W}_{\ell d,-1}, \boldsymbol{\theta}_{\ell e,-1}, \boldsymbol{\theta}_{\ell d,-1}\} \\ \bullet \ N \text{ feature vectors } \{h_n \in \mathbb{R}^M\}, \ n = 0, 1, \ldots, N-1. \end{cases} \tag{68.77}$$

The training algorithm does not use any label information and will operate in an unsupervised manner. At the end of the iterations, we set the parameters to

$$\{W^\star_{\ell,e}, W^\star_{\ell,d}, \theta^\star_{\ell,e}, \theta^\star_{\ell,d}\} \leftarrow \{\boldsymbol{W}_{\ell e,m}, \boldsymbol{W}_{\ell d,m}, \boldsymbol{\theta}_{\ell e,m}, \boldsymbol{\theta}_{\ell d,m}\} \tag{68.78}$$

Once the variational autoencoder is trained, it can be used for at least two purposes during testing:

(a) (**Compression**) We can remove the decoder and feed feature vectors into the encoder alone to generate lower-dimensional latent vector representations for them. This amounts to a form of data compression.

(b) (**Generation**) We can remove the encoder and feed random latent realizations $\boldsymbol{u}_n \sim \mathcal{N}_{\boldsymbol{u}_n}(0, I_P)$ into the decoder. This generates estimates $\widehat{h}$. For example, if the original features $\{h_n\}$ arise from some image database, we would then expect $\widehat{h}$ to look similar to the images from that database, as we illustrate in the next example. The inspection here is done visually.

Mini-batch backpropagation algorithm for training the VAE (68.37)

repeat until sufficient convergence over $m = 0, 1, 2, \ldots$**:**

select B random feature vectors $\{\boldsymbol{h}_b\}, b = 0, 1, \ldots, B-1$.

repeat for $b = 0, 1, \ldots, B-1$: *(forward processing/encoder)*

$$\boldsymbol{y}_{1e,b} = \boldsymbol{h}_b$$

repeat for $\ell = 1, 2, \ldots, L_e - 1$

$$\boldsymbol{z}_{\ell+1e,b} = \boldsymbol{W}_{\ell e,m-1}^{\mathsf{T}} \boldsymbol{y}_{\ell e,b} - \boldsymbol{\theta}_{\ell e,m-1}$$

$$\boldsymbol{y}_{\ell+1e,b} = f(\boldsymbol{z}_{\ell+1e,b})$$

end

$$\boldsymbol{y}_{L_e e,b} \triangleq \mathrm{col}\Big\{\boldsymbol{\mu}_b, \boldsymbol{a}_b\Big\}, \quad \boldsymbol{a}_b = \mathrm{col}\Big\{\boldsymbol{a}_{b,p}\Big\}_{p=1}^{P}, \quad \boldsymbol{\mu} = \mathrm{col}\Big\{\boldsymbol{\mu}_{b,p}\Big\}_{p=1}^{P}$$

$$\boldsymbol{\Sigma}_b^{1/2} = \mathrm{diag}\Big\{e^{\boldsymbol{a}_{b,1}/2}, e^{\boldsymbol{a}_{b,2}/2}, \ldots, e^{\boldsymbol{a}_{b,P}/2}\Big\}$$

generate $\boldsymbol{\epsilon}_b \sim \mathcal{N}_{\boldsymbol{\epsilon}}(0, I_P)$; set $\boldsymbol{u}_b = \boldsymbol{\mu}_b + \boldsymbol{\Sigma}_b^{1/2}\boldsymbol{\epsilon}_b$

$$\boldsymbol{y}_{1d,b} = \boldsymbol{u}_b$$

repeat for $\ell = 1, 2, \ldots, L_d - 1$**:** *(forward processing/decoder)*

$$\boldsymbol{z}_{\ell+1d,b} = \boldsymbol{W}_{\ell d,m-1}^{\mathsf{T}} \boldsymbol{y}_{\ell d,b} - \boldsymbol{\theta}_{\ell d,m-1}$$

$$\boldsymbol{y}_{\ell+1d,b} = f(\boldsymbol{z}_{\ell+1d,b})$$

end

$$\boldsymbol{\delta}_{L_d d,b} = 2(\widehat{\boldsymbol{h}}_b - \boldsymbol{h}_b) \odot f'(\boldsymbol{z}_{L_d d,b})$$

$$\boldsymbol{x}_b = \mathrm{col}\Big\{2\boldsymbol{\mu}_{b,1}, \ldots, 2\boldsymbol{\mu}_{b,P}, (e^{\boldsymbol{a}_{b,1}} - 1), \ldots, (e^{\boldsymbol{a}_{b,P}} - 1)\Big\}$$

$$\boldsymbol{\lambda}_{L_e e,b} = \boldsymbol{x}_b \odot f'(\boldsymbol{z}_{L_e e,b})$$

$$\boldsymbol{s}_b = \mathrm{col}\Big\{1, \ldots, 1, \tfrac{1}{2}\boldsymbol{\epsilon}(1)e^{\boldsymbol{a}_b(1)/2}, \ldots, \tfrac{1}{2}\boldsymbol{\epsilon}(P)e^{\boldsymbol{a}_b(P)/2}\Big\}$$

end

repeat for $\ell = L_d - 1, \ldots, 2, 1$**:** *(backward processing/decoder)*

$$\boldsymbol{W}_{\ell d,m} = (1 - 2\mu\rho)\boldsymbol{W}_{\ell d,m-1} \; - \; \frac{\mu}{B}\sum_{b=0}^{B-1} \boldsymbol{y}_{\ell d,b}\boldsymbol{\delta}_{\ell+1d,b}^{\mathsf{T}}$$

$$\boldsymbol{\theta}_{\ell d,m} = \boldsymbol{\theta}_{\ell d,m-1} + \frac{\mu}{B}\sum_{b=0}^{B-1} \boldsymbol{\delta}_{\ell+1d,b}$$

$$\boldsymbol{\delta}_{\ell d,b} = f'(\boldsymbol{z}_{\ell d,b}) \odot \Big(\boldsymbol{W}_{\ell d,m-1}\boldsymbol{\delta}_{\ell+1d,b}\Big), \; \ell \geq 2, b = 0, 1, \ldots, B-1$$

end

$$\boldsymbol{\delta}_{L_e e,b} = (\boldsymbol{s}_b \odot f'(\boldsymbol{z}_{L_e e,b}) \odot \begin{bmatrix} \boldsymbol{W}_{1d,m-1}\boldsymbol{\delta}_{2d,b} \\ \boldsymbol{W}_{1d,m-1}\boldsymbol{\delta}_{2d,b} \end{bmatrix}, \quad b = 0, 1, \ldots, B-1$$

repeat for $\ell = L_e - 1, \ldots, 2, 1$**:** *(backward processing/encoder)*

$$\boldsymbol{W}_{\ell e,m} = (1 - 2\mu\rho)\boldsymbol{W}_{\ell e,m-1} \; - \; \frac{\mu}{B}\sum_{b=0}^{B-1} \boldsymbol{y}_{\ell e,b}\Big(\boldsymbol{\delta}_{\ell+1e,b} + \boldsymbol{\lambda}_{\ell+1e,b}\Big)^{\mathsf{T}}$$

$$\boldsymbol{\theta}_{\ell e,m} = \boldsymbol{\theta}_{\ell e,m-1} + \frac{\mu}{B}\sum_{b=0}^{B-1}\Big(\boldsymbol{\delta}_{\ell+1e,b} + \boldsymbol{\lambda}_{\ell+1e,b}\Big)$$

$$\boldsymbol{\delta}_{\ell e,b} = f'(\boldsymbol{z}_{\ell e,b}) \odot \Big(\boldsymbol{W}_{\ell e,m-1}\boldsymbol{\delta}_{\ell+1e,b}\Big), \; \ell \geq 2, b = 0, 1, \ldots, B-1$$

$$\boldsymbol{\lambda}_{\ell e,b} = f'(\boldsymbol{z}_{\ell e,b}) \odot \Big(\boldsymbol{W}_{\ell e,m-1}\boldsymbol{\lambda}_{\ell+1e,b}\Big), \; \ell \geq 2, b = 0, 1, \ldots, B-1$$

end

end

(68.79)

Example 68.1 (**Generation of handwritten digits using a VAE**) We illustrate the operation of the variational autoencoder algorithm by applying it to the problem of learning to generate "handwritten digits" that are similar to the ones arising from the same MNIST dataset considered earlier in Example 65.9. Recall that the MNIST dataset consists of 60,000 labeled training samples. Each entry in the dataset is a 28×28 grayscale image, which we transform into an $M = 784-$long feature vector, h_n. Each pixel in the image and, therefore, each entry in h_n, assumes integer values in the range $[0, 255]$. Every feature vector (or image) is assigned an integer label in the range 0-9 depending on which digit the image corresponds to. The earlier Fig. 52.6 showed randomly selected images from the training dataset.

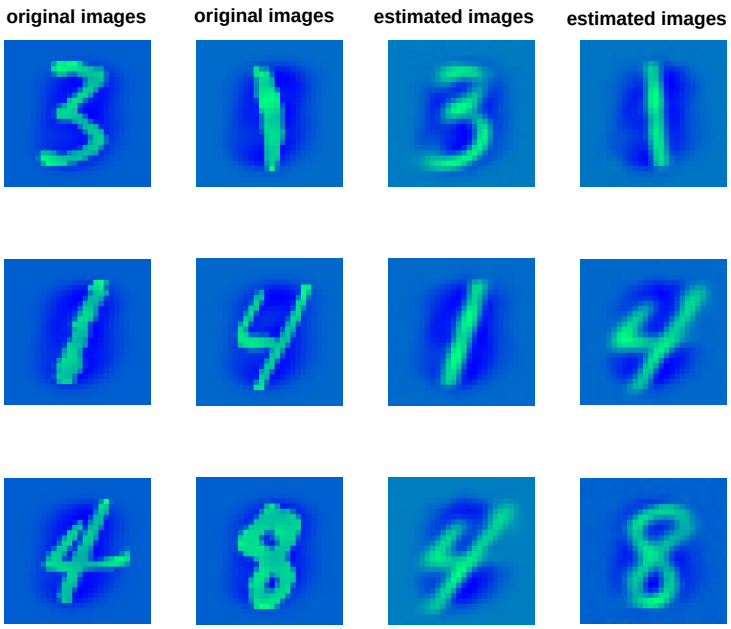

Figure 68.8 The plots in the two columns on the left show six randomly-selected original images from the MNIST dataset used to train the variational autoencoder. The plots in the last two columns on the right show the corresponding approximations generated at the end of the training phase for these images.

We pre-process the images (or the corresponding feature vectors $\{h_n\}$) by scaling their entries by 255 (so that they assume values in the range $[0, 1]$). We subsequently compute the mean feature vector for the training set and center the scaled feature vectors around this mean. The earlier Fig. 52.7 showed randomly selected images for the digits $\{0, 1\}$ before and after processing.

We construct a variational autoencoder with $L_e = 4$ layers in the encoder stage and $L_d = 4$ layers in the decoder stage. In this way, each section has two hidden layers. The size of the input layer for the encoder is $n_{1,e} = 784$ (which agrees with the size of the feature vectors), while the size of each of the two hidden layers in the encoder is $n_{2,e} = n_{3,e} = 512$ neurons. We set the dimension of the latent vector $\boldsymbol{u}$ to $P = 2$ so that

the size of the output layer of the encoder is $n_{L_e,e} = 4$. Likewise, the size of the input layer for the decoder is $n_{1,d} = 2$ (which agrees with the size of the latent variable), while the size of each of the two hidden layers in the decoder is again $n_{2,d} = n_{3,d} = 512$ neurons. The size of the output layer of the decoder is $n_{L_d,d} = 784$ since it needs to match the size of the input feature vector.

We set the activation functions for all hidden layers in both the encoder and decoder to the ReLu function. However, we set the activation functions at the output layer of the encoder to the *linear function* and at the output layer of the decoder to the tanh function.

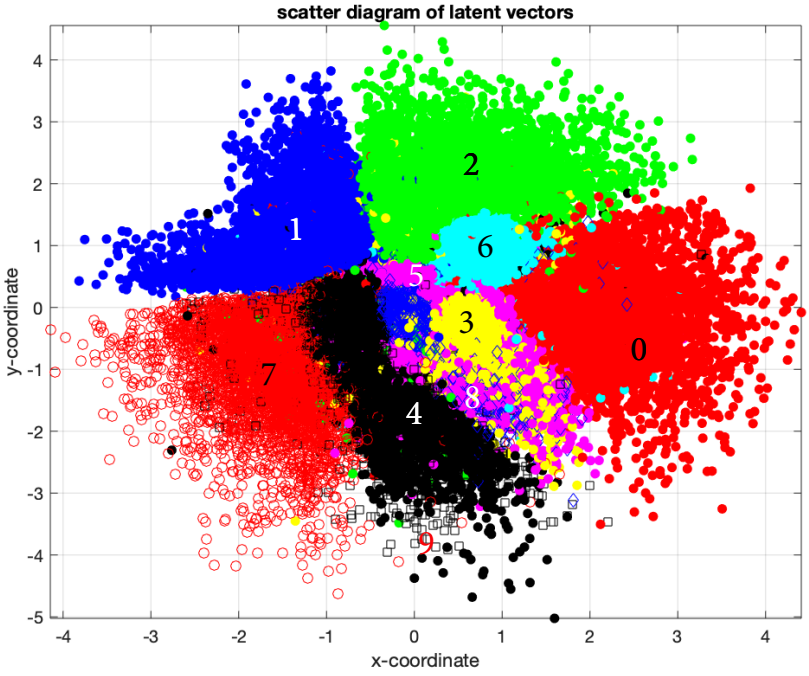

Figure 68.9 The figure plots the latent variables $\{u_n\}$ generated at the end of training for all $60,000$ handwritten digits $\{h_n\}$ in the dataset. The figure is color-coded so that latent variables corresponding to the same digit value appear in the same color. Observe how the latent variables appear clustered together depending on their digit value.

We train the variational autoencoder by using the step-size and regularization parameters:

$$\mu = 0.0001, \quad \rho = 0.0001 \tag{68.80}$$

We run a stochastic-gradient version of the backpropagation algorithm (68.79) with $B = 1$. We perform 200 passes over the training data; the data is randomly reshuffled at the start of each pass. For each input feature vector h, we save the value of the latent variable u that was generated at the end of training, after the variational autoencoder has converged. Figure 68.8 selects six random feature vectors, h_n, and plots them next to the corresponding output vectors, $\widehat{h}_n$, following training.

In Figure 68.9, we plot a scatter diagram for all latent variables $\{u_n\}$ that were saved at the end of the training phase for all 60,000 feature samples $\{h_n\}$ from the dataset. The figure is color-coded so that latent variables corresponding to the same digit value appear in the same color. It is observed that the variables appear clustered together depending on their digit value. We could have chosen latent variables of higher dimension than $P = 2$; we selected this lower value to allow visualization of the clustering effect. Besides, as the subsequent figures show, this low-dimensional latent space is sufficient to illustrate the desired effects and results.

During testing, we disconnect the decoder from the encoder. We generate 12 Gaussian latent variables $\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)$ and feed them into the decoder with the weights and biases fixed at the values obtained at the end of the training phase. We generate the corresponding output vectors $\widehat{h}$ and plot them in image form in Fig. 68.10. These are synthetic or fake images!

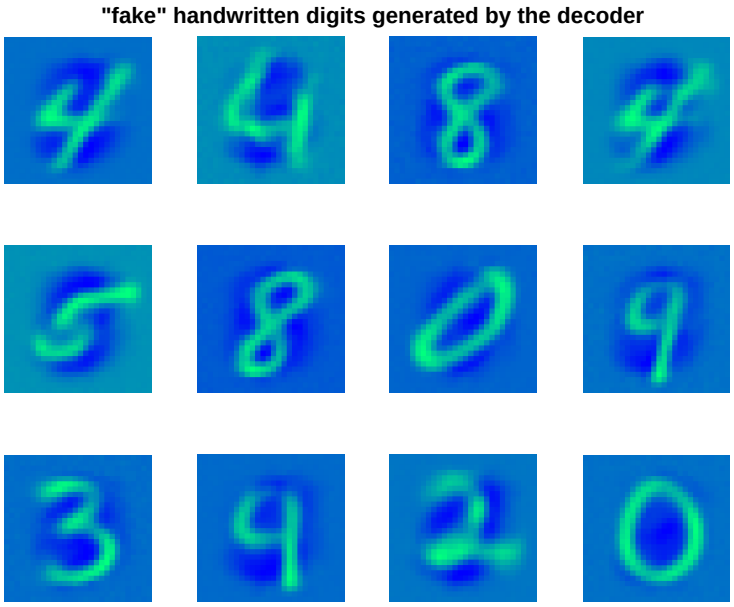

Figure 68.10 Synthetic handwritten images generated by feeding Gaussian latent values $\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)$ through the decoder and plotting the resulting output vectors in image format.

Example 68.2 (**Recommender systems**) We revisit the recommender system studied in Example 16.7. There we introduced a collaborative filtering approach based on matrix factorization to predict ratings by users. We reconsider the user-item $U \times I$ matrix R and replace all missing entries by zeros. That is, if some user u did not rate an item i, we place a zero in the $(u, i)-$th location of R. This formulation allows us to handle additional situations where user feedback is collected in other ways, other than by entering explicit rating scores. For example, the entries of R could correspond to binary flags that assume the values one or zero depending on whether the user clicked on an item or not, or whether the user clicked on "like" or "dislike" buttons. The entries of R could also correspond to counters that count how many times a user visited a particular item's site on an online store. In all these examples, the zero entries correspond to

situations where there has not been any interaction between a user and the item. The objective continues to be to design a recommender system that enables us to predict what the interactions would be for a particular user: whether the user would like an item or not, visit a site or not, click on a button or not, etc.

To do so, we construct a variational autoencoder (VAE) as follows. We associate with each user u an $I \times 1$ feature vector h_u that is formed of the entries in the $u-$th row of R. That is, the vector h_u contains the interactions between user u and the items represented by R (including those for which there were no interactions). In this way, each row of R is transposed and used as a feature vector. The number of feature vectors available for training would be $\{h_u\}$, $u = 1, 2, \ldots, U$, where U is the number of rows in R (i.e., number of users for which data has been collected). Once training is completed, we freeze the parameters of the VAE and feed the feature vectors $\{h_u\}$ through it to obtain estimate vectors, $\{\widehat{h}_u\}$. For each user u, the entries of $\widehat{h}_u$ would contain predicted values for the zero entries from the original feature h_u.

68.3 CONDITIONAL VARIATIONAL AUTOENCODERS

The simulations in Example 68.1 illustrate how the decoder stage of the VAE can be used to generate "fake" handwritten digit images that look similar to the images from the original MNIST database. This is achieved by feeding realizations of the latent variable $\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)$ into the decoder. The generated images will be any of the digits in the range $\{0, 1, 2, \ldots, 9\}$. But what if we are interested in using the decoder to generate synthetic images that correspond to a particular digit number, say, the digit 3? How should the structure of the variational autoencoder be modified to allow us to generate images that belong to a particular class, rather than arbitrarily to any of the possible classes? *Conditional* variational autoencoders address this problem by exploiting the label information, which has not been used so far.

If we refer to the derivation of the variational autoencoder structure, starting from Sec. 68.1.2, we observe that the derivation treated the feature vectors $\boldsymbol{h}$ as input to the encoder and sought first a (Gaussian) approximation for the conditional distribution $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$. Figure 68.1 illustrated this formulation. In the process of determining an optimal approximation for $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$, we were led to the fundamental equality (68.7), which we repeat here for ease of reference (we are again removing the subscripts from the pdf notation for compactness):

$$\ln f(h) = D_{\mathrm{KL}}\Big(q(u|h) \,\|\, f(u|h)\Big) + \underbrace{\mathbb{E}_q\Big(\ln f(h|u)\Big) - D_{\mathrm{KL}}\Big(q(u|h) \,\|\, f(u)\Big)}_{\triangleq \textbf{ELBO, } \mathcal{L}(q)} \tag{68.81}$$

In this expression, $f(h)$ denotes the pdf for the observations, namely, $f_{\boldsymbol{h}}(h)$, while $f(u)$ denotes the pdf for the latent variables, $f_{\boldsymbol{u}}(u)$. Moreover, the notation $f(u|h)$ and $f(h|u)$ refer to the conditional pdfs of the latent variable given the

feature vector and the reverse, i.e., $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ and $f_{\boldsymbol{h}|u}(h|u)$. Likewise, the term $q(u|h)$ stands for $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$ and refers to the approximation for $f_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$.

Now, let $\boldsymbol{\gamma}$ denote some other random (scalar or vector) variable that we wish to condition against, such as the class variable or some representation for it. Straightforward repetition of the derivation that led to (68.7) will show that we can replace (68.81) by — see Prob. 68.2:

$$\ln f(h|\gamma) = D_{\mathrm{KL}}\Big(q(u|h,\gamma)\,\|\,f(u|h,\gamma)\Big) + \underbrace{\mathbb{E}_{\,q}\Big(\ln f(h|u,\gamma)\Big) - D_{\mathrm{KL}}\Big(q(u|h,\gamma)\,\|\,f(u|\gamma)\Big)}_{\triangleq\ \textbf{ELBO},\ \mathcal{L}(q|\gamma)} \tag{68.82}$$

where all distributions are now conditioned on $\boldsymbol{\gamma}$. The same explanation given in Sec. 68.1.3 can then be repeated to motivate the following structures for the encoder and decoder stages.

Architecture

We assume that $q_{\boldsymbol{u}|\boldsymbol{h},\boldsymbol{\gamma}}(u|h,\gamma)$ belongs to the family of Gaussian distributions. We continue to denote its mean and covariance matrix by $\mu(h,\gamma)$ and $\Sigma(h,\gamma)$, respectively, where these quantities are now dependent on *both* h and γ. We further assume that Σ is diagonal with entries $\{\sigma_p^2(h,\gamma)\}$ for $p = 1, 2, \ldots, P$, where P denotes the size of $\boldsymbol{u}$. In this way, the problem of learning $q_{\boldsymbol{u}|\boldsymbol{h},\boldsymbol{\gamma}}(u|h,\gamma)$ reduces to the problem of learning the moments:

$$\Big\{\mu_1(h,\gamma), \ldots, \mu_P(h,\gamma),\ \sigma_1^2(h,\gamma),\ \ldots,\ \sigma_P^2(h,\gamma)\Big\} \tag{68.83}$$

We will learn these moments by using a neural network structure for the encoder with input consisting of both (h,γ); i.e., the feature vector and its label. Usually, *one-hot encoding* is used to represent the class variable γ. For example, if we desire to generate images that belong to class 3 in a problem with a total of $C = 5$ classes, $c \in \{1, 2, 3, 4, 5\}$, then we set γ to the $C-$dimensional vector:

$$\gamma = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \in \{0,1\}^5 \tag{68.84}$$

with zeros everywhere and a unit entry at the location corresponding to class 3. The input to the encoder is then constructed as the extended vector:

$$h' \ \triangleq\ \begin{bmatrix} h \\ \hline \gamma \end{bmatrix}, \quad \text{of size } M + C \tag{68.85}$$

This also means that the dimension $n_{1,e}$ changes from $n_{1,e} = M$ to $n_{1,e} = M+C$.

Next, we approximate the likelihood $f_{\boldsymbol{h}|\boldsymbol{u},\boldsymbol{\gamma}}(h|u,\gamma)$ by constructing a second

neural network structure that attempts to replicate h. The input to this feedforward neural network will be a realization u for the latent variable *augmented* by the same label variable γ, i.e.,

$$u' \triangleq \left[\begin{array}{c} u \\ \hline \gamma \end{array}\right], \quad \text{of size } P+C \tag{68.86}$$

where u is the compressed representation for h and γ represents the label information. As a result, the dimension $n_{1,d}$ changes from $n_{1,d}=P$ to $n_{1,d}=P+C$. The output of the decoder continues to be an approximations $\widehat{h}$ for the feature vector h. Finally, we generate realizations for the conditional latent variables $\boldsymbol{u}|(\boldsymbol{h},\boldsymbol{\gamma})$ by using

$$\boxed{\boldsymbol{u} = \mu(h,\gamma) \;+\; \Sigma^{1/2}(h,\gamma)\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}_{\boldsymbol{\epsilon}}(0, I_P)} \tag{68.87}$$

Figure 68.11 provides a diagram representation for the resulting conditional variational autoencoder. It is worth comparing this structure with the earlier Fig. 68.5. We continue to have two stages: an encoder and a decoder; both of them modeled by means of multi-layer feedforward networks. However, in the conditional implementation shown in Fig. 68.11, we are now assuming knowledge of the class variable (in the form of one-hot encoding vectors). This class variable is fed into *both* the encoder and decoder stages. This formulation is satisfactory in situations where the class information is available during testing. For example, if the purpose of testing is to use the encoder component to perform compression, then this solution will require knowledge of h along with its class variable γ. We consider a different structure in Prob. 68.5 that does not feed γ into the encoder during training in order to avoid the need for it while generating (compressed) latent representations during testing; the class variable (or a representation for it) will still be needed for the generation of "fake" realizations $\widehat{h}$ by the decoder.

Training algorithm

The empirical risk continues to be the same except that the mean and log-variance parameters are now functions of both h_n and γ_n:

$$\begin{aligned}\mathcal{P}(W,\theta) \triangleq \Bigg\{ &\sum_{\ell=1}^{L_e-1} \rho\|W_{\ell,e}\|_{\mathrm{F}}^2 + \sum_{\ell=1}^{L_d-1} \rho\|W_{\ell,d}\|_{\mathrm{F}}^2 + \frac{1}{N}\sum_{n=0}^{N-1} \|h_n - \widehat{h}_n\|^2 + \\ &\frac{1}{N}\sum_{n=0}^{N-1}\sum_{p=1}^{P}\Big(\mu_p^2(h_n,\gamma_n) + e^{a_p(h_n,\gamma_n)} - a_p(h_n,\gamma_n)\Big)\Bigg\}\end{aligned} \tag{68.88}$$

As such, the same training algorithm (68.79) continues to hold with one minor adjustment in relation to the sensitivity factor at the transition layer — Prob. 68.4.

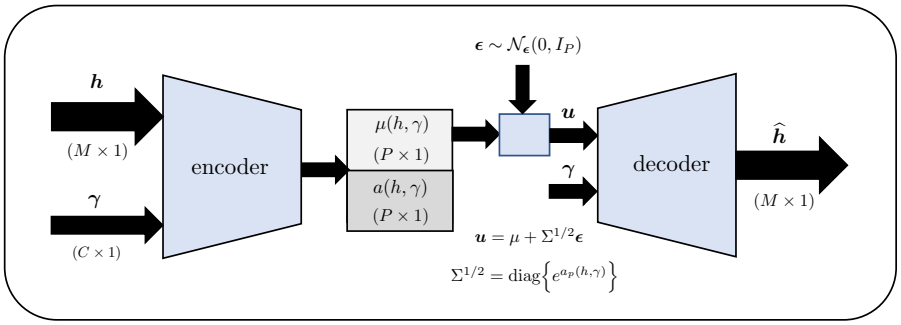

Figure 68.11 Diagram representation of one implementation for the *conditional* variational autoencoder (VAE) involving a cascade of encoder and decoder structures with the intermediate step of generating the latent variables $\boldsymbol{u}$ from standard Gaussian realizations ϵ. The input to the encoder is the extended vector $\mathrm{col}\{h, \gamma\}$, while the input to the decoder is the extended vector $\mathrm{col}\{u, \gamma\}$.

Specifically, recall from (68.49) that by definition

$$\delta_{L_e,e}(j) = \frac{\partial \|h - \widehat{h}\|^2}{\partial z_{L_e,e}(j)}, \quad j = 1, 2, \ldots, 2P \tag{68.89}$$

However, the input to the decoder now involves two terms, u and γ. As such, the derivation that led to (68.49) will need to be adjusted slightly as follows to account for the presence of γ (observe the difference in the last equality):

$$\begin{aligned}
\delta_{L_e,e}(j) &= \sum_{i=1}^{n_{2,d}} \frac{\partial \|h - \widehat{h}\|^2}{\partial z_{2,d}(i)} \frac{\partial z_{2,d}(i)}{\partial z_{L_e,e}(j)} \\
&= \sum_{i=1}^{n_{2,d}} \delta_{2,d}(i) \frac{\partial z_{2,d}(i)}{\partial z_{L_e,e}(j)} \\
&= \sum_{i=1}^{n_{2,d}} \delta_{2,d}(i) \left(\sum_{p=1}^{P} \frac{\partial z_{2,d}(i)}{\partial u(p)} \frac{\partial u(p)}{\partial z_{L_e,e}(j)} + \sum_{c=1}^{C} \frac{\partial z_{2,d}(i)}{\partial \gamma(c)} \frac{\partial \gamma(c)}{\partial z_{L_e,e}(j)} \right)
\end{aligned} \tag{68.90}$$

Note that we are denoting the individual entries of the $C \times 1$ label vector γ by $\{\gamma(c)\}$. However, since the entries of γ are independent of the entries of $z_{L_e,e}$ then

$$\frac{\partial \gamma(c)}{\partial \partial z_{L_e,e}(j)} = 0 \tag{68.91}$$

and the last sum over c disappears. It follows, as before, that

$$\delta_{L_e,e}(j) = \sum_{i=1}^{n_{2,d}} \delta_{2,d}(i) \left(\sum_{p=1}^{P} \frac{\partial z_{2,d}(i)}{\partial u(p)} \frac{\partial u(p)}{\partial z_{L_e,e}(j)} \right) \tag{68.92}$$

so that expression (68.56) continues to hold. The main difference, however, is

that the entries $w_{ji,d}^{(1)}$ that appear in this expression do not correspond to all the entries within $W_{1,d}$. This is because the row dimension of $W_{1,d}$ is now enlarged to $P + C$:

$$W_{1,d} : (P + C) \times n_{2,d}, \quad \text{since } n_{1,d} = P + C \tag{68.93}$$

Only the entries in the leading P rows of $W_{1,d}$ appear in (68.92). We should therefore replace (68.58) by

$$\delta_{L_e,e} = \Big(s \odot f'(z_{L_e,e})\Big) \odot \begin{bmatrix} \bar{W}_{1,d}\,\delta_{2,d} \\ \bar{W}_{1,d}\,\delta_{2,d} \end{bmatrix} \tag{68.94}$$

where $\bar{W}_{1,d}$ contains the leading P rows of $W_{1,d}$:

$$\bar{W}_{1,d} = \begin{bmatrix} I_P & 0 \end{bmatrix} W_{1,d} \tag{68.95}$$

Example 68.3 (**Generation of handwritten digits using a conditional VAE**) We repeat the experiment described in Example 68.1 involving the MNIST database and use the same structures and parameters. Now, however, we train a conditional VAE and use it to generate versions of the digit 3. The main difference is that we now feed the one-hot encoded label vector into both the encoder and decoder. Figure 68.12 illustrates the generation of synthetic handwritten samples for the digit 3.

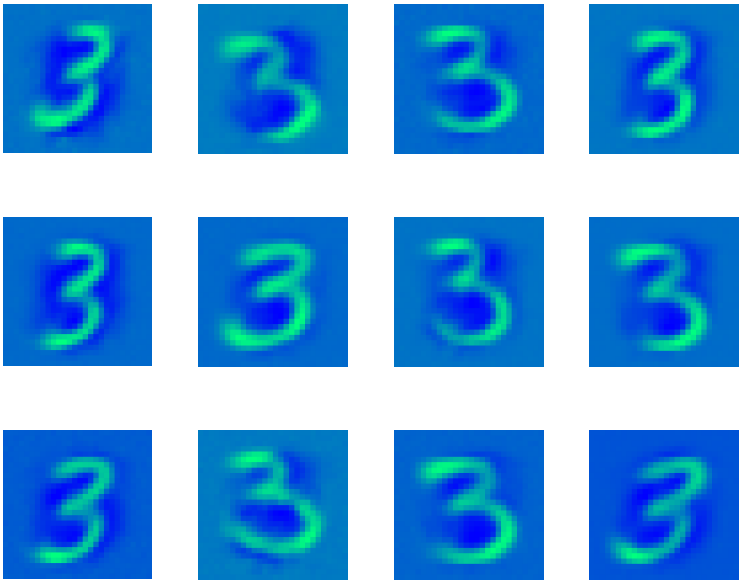

Figure 68.12 Synthetic handwritten images of the digit 3 generated by feeding Gaussian latent variables $\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)$ into the encoder and the label $\gamma = e_4$ (4−th basis vector with a unit entry at the 4−location) into both the encoder and decoder.

68.4 GENERATIVE ADVERSARIAL NETWORKS

In this section we describe a second approach to generative models that relies on the use of adversarial neural networks. In the variational autoencoder formulation from the previous section, the operation of the encoder and decoder components complemented each other: the first performed compression down to the latent space and the second decoded the information hidden in the latent variables to generate samples from the data distribution. In generative adversarial networks (GANs), we will also encounter two network components called the *generator* and *discriminator*. However, these components will now be competing against each other with the generator trying to drive the discriminator away from its objective. The net effect will again be a structure that enables us to generate samples (such as images) from the same distribution as the original data.

68.4.1 Discriminator

We start by describing the discriminator. It takes the form of a feedforward neural network with L_d total layers, including its input and output layers. Depending on the application at hand, we may use a convolutional neural network. It is sufficient for our purposes to illustrate the main ideas by continuing with a traditional feedforward neural structure. Figure 68.13 shows a discriminator with 3 hidden layers, input vectors h of size $M = 3$, and output vectors $\widehat{\gamma}$ of size $Q = 2$. In the figure, we are using little squares in the output layer to refer to the softmax calculation used to generate the entries of $\widehat{\gamma}$, as explained in (68.97) further ahead.

The discriminator receives feature vectors $\{h_n \in \mathbb{R}^M\}$ that arise from one of two possible sources: a *real* source or a *fake* source. For example, the data may amount to images with some of them being genuine/real images and the others being fake or computer-generated images. The purpose of the discriminator is to classify these feature vectors into real or fake. This is similar, for example, to an application where a network is required to discriminate between real and fake currency bills. For this reason, the discriminator will be designed to operate as a classifier. The label vector for the input features will be constructed using *one-hot encoding*; they will have dimension $Q = 2$ and their entries will assume binary values in $\{0, 1\}$:

$$\gamma \in \{0, 1\}^Q \tag{68.96}$$

The first entry of γ will be one when the feature h is real; otherwise, the second entry of γ will be one when h is synthetic. We will assume that the discriminator employs a regularized cross-entropy risk function with a softmax output layer, as was described earlier in Sec. 65.7. In this case, the entries of the output vector

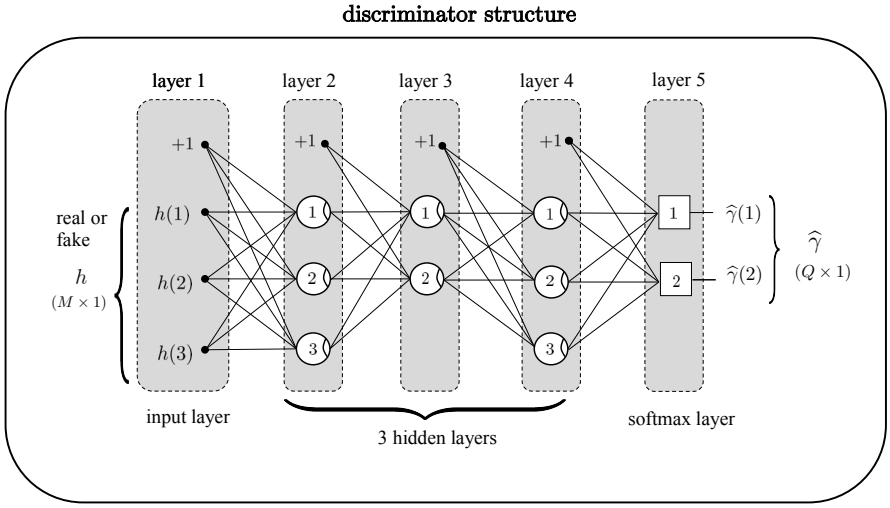

Figure 68.13 A discriminator structure with $M = 3$ attributes at the input layer (i.e., $h \in \mathbb{R}^3$), output vector of size $Q = 2$ (i.e., two classes), and 3 hidden layers. The output layer is softmax.

$\widehat{\gamma}$ are generated by the softmax computation:

$$\widehat{\gamma}(q) \;\triangleq\; \frac{e^{z(q)}}{e^{z(1)} + e^{z(2)}}, \quad q = 1, 2 \tag{68.97}$$

where the $\{z(q)\}$ are the pre-activation signals in the output layer. This construction ensures that the values of $\widehat{\gamma}(q)$ are in the range $(0, 1)$ and add up to one so that they can be interpreted as corresponding to probability values: $\widehat{\gamma}(1)$ specifies the likelihood that h is "real," while $\widehat{\gamma}(2)$ specifies the likelihood that h is "fake."

If we denote the combination matrices and bias vectors within the discriminator by $\{W_{\ell,d}, \theta_{\ell,d}\}$, and consider a collection of N total data pairs $\{h_n, \gamma_n\}$, for $n = 0, 1, \ldots, N-1$, then the empirical risk function that the discriminator seeks to minimize is — recall (65.127):

$$\begin{aligned}
&\mathcal{P}\Big(\{W_{\ell,d}, \theta_{\ell,d}\}\Big) \qquad (68.98)\\
&\triangleq \sum_{\ell=1}^{L_d-1} \rho \|W_{\ell,d}\|_{\mathrm{F}}^2 \;-\; \frac{1}{N} \sum_{n=0}^{N-1} \sum_{q=1}^{Q} \gamma_n(q) \ln\big(\widehat{\gamma}_n(q)\big)\\
&= \sum_{\ell=1}^{L_d-1} \rho \|W_{\ell,d}\|_{\mathrm{F}}^2 \;-\; \frac{1}{N} \sum_{n=0}^{N-1} \Big[\gamma_n(1) \ln\big(\widehat{\gamma}_n(1)\big) + \gamma_n(2) \ln\big(\widehat{\gamma}_n(2)\big)\Big]\\
&= \sum_{\ell=1}^{L_d-1} \rho \|W_{\ell,d}\|_{\mathrm{F}}^2 \;-\; \frac{1}{N} \sum_{n=0}^{N-1} \Big[\gamma_n(1) \ln\big(\widehat{\gamma}_n(1)\big) + (1 - \gamma_n(1)) \ln\big(1 - \widehat{\gamma}_n(1)\big)\Big]
\end{aligned}$$

where $\rho \geq 0$ is a regularization factor. In the last equality, we used the fact that

the entries of each of the vectors γ_n and $\widehat{\gamma}_n$ add up to one. Other choices for the risk function and for the structure of the discriminator are of course possible with proper adjustments to the arguments that follow.

In GAN implementations, we ensure that the feature data feeding into the discriminator during training is more or less equally split between real and fake features. Thus, assume there are N_r *real* features and N_f *fake* features with $N_r + N_f = N$; usually, $N_r = N_f = N/2$. We denote the set of indexes for the real features by $\mathcal{N}_r$ (its cardinality is equal to N_r), and the set of indexes for the fake features by $\mathcal{N}_f$ (its cardinality is equal to N_f). For a real feature vector h we have $\gamma(1) = 1$ and $\gamma(2) = 0$, while for a fake feature vector h we have $\gamma(1) = 0$ and $\gamma(2) = 1$. We use this information to rewrite the empirical risk (68.98) more explicitly as follows, with the sum split over the real and fake data:

$$\begin{aligned} &\mathcal{P}\Big(\{W_{\ell,d}, \theta_{\ell,d}\}\Big) \\ &= \sum_{\ell=1}^{L_d-1} \rho \|W_{\ell,d}\|_{\mathrm{F}}^2 - \frac{1}{N}\left(\sum_{n\in\mathcal{N}_r} \ln \widehat{\gamma}_n(1) \;+\; \sum_{n\in\mathcal{N}_f} \ln\big(1-\widehat{\gamma}_n(1)\big)\right) \end{aligned} \tag{68.99}$$

The discriminator would seek parameters $(\{W_{\ell,d}\}, \{\theta_{\ell,d}\})$ in order to *minimize* this risk. By doing so, the discriminator would be maximizing the likelihood of correct classifications (i.e., of correctly discriminating real from fake data by driving $\widehat{\gamma}_n(1)$ towards one for real data and towards zero for fake data):

$$\Big\{W_{\ell,d}^{\star}, \theta_{\ell,d}^{\star}\Big\} \;=\; \underset{\{W_{\ell,d},\theta_{\ell,d}\}}{\operatorname{argmin}} \; \mathcal{P}\Big(\{W_{\ell,d}, \theta_{\ell,d}\}\Big) \tag{68.100}$$

Stochastic risk interpretation

Observe that expression (68.99) depends only on the leading entry of the output vector, i.e., on $\widehat{\gamma}_n(1)$. In this way, we can view the discriminator as a mapping from h_n to $\widehat{\gamma}_n(1)$. The value of $\widehat{\gamma}_n(1)$ specifies the likelihood that h_n is real (from which we can obviously deduce the likelihood of it being fake). We denote the mapping by $\mathcal{D}(h_n)$, i.e.,

$$\boxed{\widehat{\gamma}_n(1) = \mathcal{D}(h_n)} \tag{68.101}$$

Recall that for $n \in \mathcal{N}_r$, the $\widehat{\gamma}_n(1)$ are generated by *real* feature vectors, and that for $n \in \mathcal{N}_f$, the $\widehat{\gamma}_n(1)$ are generated by *fake* feature vectors. Assuming large enough N_r and N_f, and appealing to the law of large numbers, we can write for the last term in (68.99):

$$\begin{aligned} &\frac{1}{N}\left(\sum_{n\in\mathcal{N}_r} \ln \widehat{\gamma}_n(1) \;+\; \sum_{n\in\mathcal{N}_f} \ln\big(1-\widehat{\gamma}_n(1)\big)\right) \\ &\xrightarrow{N\to\infty} \;\frac{N_r}{N}\mathbb{E}_{\,f_{\boldsymbol{h}}}\, \ln \mathcal{D}(\boldsymbol{h}) \;+\; \frac{N_f}{N}\mathbb{E}_{\,f_{\hat{\boldsymbol{h}}}}\, \ln\Big(1-\mathcal{D}(\boldsymbol{h})\Big) \\ &\approx \;\frac{1}{2}\Big\{\mathbb{E}_{\,f_{\boldsymbol{h}}}\, \ln \mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{\,f_{\hat{\boldsymbol{h}}}}\, \ln\Big(1-\mathcal{D}(\boldsymbol{h})\Big)\Big\} \end{aligned} \tag{68.102}$$

where the last equality is under the generally held condition $N_r = N_f \approx N/2$. Moreover, in the second line, the first expectation is relative to the distribution of the real features, denoted by $f_{\boldsymbol{h}}(h)$, and the second expectation is relative to the distribution of the fake features, denoted by $f_{\widehat{\boldsymbol{h}}}(h)$. That is, the argument of $\mathcal{D}(\boldsymbol{h})$ in the first expectation are feature vectors that arise from the real data distribution, while the argument of $\mathcal{D}(\boldsymbol{h})$ in the second expectation are feature vectors that arise from the fake distribution. These latter vectors will be produced by the generator, which we will describe in the next section, and they will be denoted by $\widehat{\boldsymbol{h}}$ (i.e., we will denote feature vectors corresponding to indexes $n \in \mathcal{N}_f$ by $\widehat{\boldsymbol{h}}_n$). In this way, *apart from regularization* on the weight matrices and assuming the generator structure is fixed so that $f_{\widehat{\boldsymbol{h}}}(\cdot)$ is fixed, we can interpret the operation of the discriminator as seeking parameters $\{W_{\ell,d}, \theta_{\ell,d}\}$ that maximize an objective of the form:

$$\boxed{\left\{W_{\ell,d}^{\star}, \theta_{\ell,d}^{\star}\right\} \;=\; \underset{\{W_{\ell,d},\theta_{\ell,d}\}}{\operatorname{argmax}} \left\{\mathbb{E}_{\,f_{\boldsymbol{h}}} \ln \mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{\,f_{\widehat{\boldsymbol{h}}}} \ln\Big(1-\mathcal{D}(\widehat{\boldsymbol{h}})\Big)\right\}} \tag{68.103}$$

We are using maximization in (68.103) rather than minimization because the term (68.102) appears with a minus sign in the empirical risk expression (68.99).

Example 68.4 (**Optimal discriminator strategy**) Motivated by the stochastic interpretation (68.103), let us consider the following formulation:

$$\mathcal{D}^{\star} \;=\; \underset{\mathcal{D}}{\operatorname{argmax}} \left\{\mathbb{E}_{\,f_{\boldsymbol{h}}} \ln \mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{\,f_{\widehat{\boldsymbol{h}}}} \ln\Big(1-\mathcal{D}(\widehat{\boldsymbol{h}})\Big)\right\} \tag{68.104}$$

where, compared with (68.103), we are not limiting the mapping $\mathcal{D}$ to one that is generated by feedforward neural networks. Instead, we are seeking the *optimal* mapping $\mathcal{D}^{\star}$ that maximizes the cost in (68.104). We denote the cost function by

$$\begin{aligned}
J(\mathcal{D}) \;&\triangleq\; \mathbb{E}_{\,f_{\boldsymbol{h}}} \ln \mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{\,f_{\widehat{\boldsymbol{h}}}} \ln(1-\mathcal{D}(\widehat{\boldsymbol{h}})) \\
&= \int_{h\in\mathcal{H}} f_{\boldsymbol{h}}(h) \ln \mathcal{D}(h) dh \;+\; \int_{\widehat{h}\in\widehat{\mathcal{H}}} f_{\widehat{\boldsymbol{h}}}(\widehat{h}\,) \ln(1-\mathcal{D}(\widehat{h})) d\widehat{h} \\
&\overset{(a)}{=} \int_{h\in\mathcal{H}} f_{\boldsymbol{h}}(h) \ln \mathcal{D}(h) dh \;+\; \int_{h\in\widehat{\mathcal{H}}} f_{\widehat{\boldsymbol{h}}}(h) \ln(1-\mathcal{D}(h)) dh \\
&\overset{(b)}{=} \int_{h\in\mathcal{H}} \Big(f_{\boldsymbol{h}}(h) \ln \mathcal{D}(h) \;+\; f_{\widehat{\boldsymbol{h}}}(h) \ln(1-\mathcal{D}(h))\Big) dh
\end{aligned} \tag{68.105}$$

where in step (a) we introduced a change of variables (i.e., a change in notation) in the second integral from $\widehat{h}$ to h, and in step (b) we assume the domain $\widehat{\mathcal{H}}$ for the fake data agrees with the domain $\mathcal{H}$ for the real data; this latter condition is essential for the discriminator-generator structure to work properly. Since the pdfs are nonnegative for all h, and since $\mathcal{D}(h) \in (0,1)$, the integrand is non-positive. Expression (68.105) is therefore maximized when the integrand is maximized for all h. Differentiating the integrand relative to $\mathcal{D}(h)$ and setting the derivative to zero gives:

$$\frac{f_{\boldsymbol{h}}(h)}{\mathcal{D}^{\star}(h)} \;-\; \frac{f_{\widehat{\boldsymbol{h}}}(h)}{(1-\mathcal{D}^{\star}(h))} = 0 \tag{68.106}$$

so that the optimal discriminator mapping is

$$\mathcal{D}^{\star}(h) = \frac{f_{\boldsymbol{h}}(h)}{f_{\boldsymbol{h}}(h) + f_{\widehat{\boldsymbol{h}}}(h)} = \left(1 + \frac{f_{\widehat{\boldsymbol{h}}}(h)}{f_{\boldsymbol{h}}(h)}\right)^{-1} \tag{68.107}$$

In other words, given a feature vector h, an optimal strategy for the discriminator is to evaluate the evidence under $f_{\boldsymbol{h}}(\cdot)$ and the evidence under $f_{\widehat{\boldsymbol{h}}}(\cdot)$. Then, the likelihood of h being a real image (i.e., the value of $\widehat{\gamma}(1)$) is equal to the ratio (68.107). The discriminator structure based on neural layers will be approximating this optimal mapping. This means that the discriminator will be attempting to learn the ratio of distributions, $f_{\widehat{\boldsymbol{h}}}(h)/f_{\boldsymbol{h}}(h)$, rather than the real data distribution, $f_{\boldsymbol{h}}(h)$. This point represents a difference in relation to the VAE solution where the decoder attempts to learn $f_{\boldsymbol{h}}(h)$ or $f_{\boldsymbol{h}|\boldsymbol{u}}(h|u)$ to generate the samples $\widehat{h}$.

68.4.2 Generator

We describe next the generator structure, which also takes the form of a feedforward neural network with L_g total layers, including its input and output layers. We denote its combination matrices and bias vectors by $\{W_{\ell,g}, \theta_{\ell,g}\}$. Depending on the application at hand, we may again use a convolutional neural network. However, we will continue to present the main ideas by using a traditional feedforward neural structure. The generator structure is shown in Figure 68.14 for a case involving 3 hidden layers, $M = 3$, and latent variables of size $P = 2$.

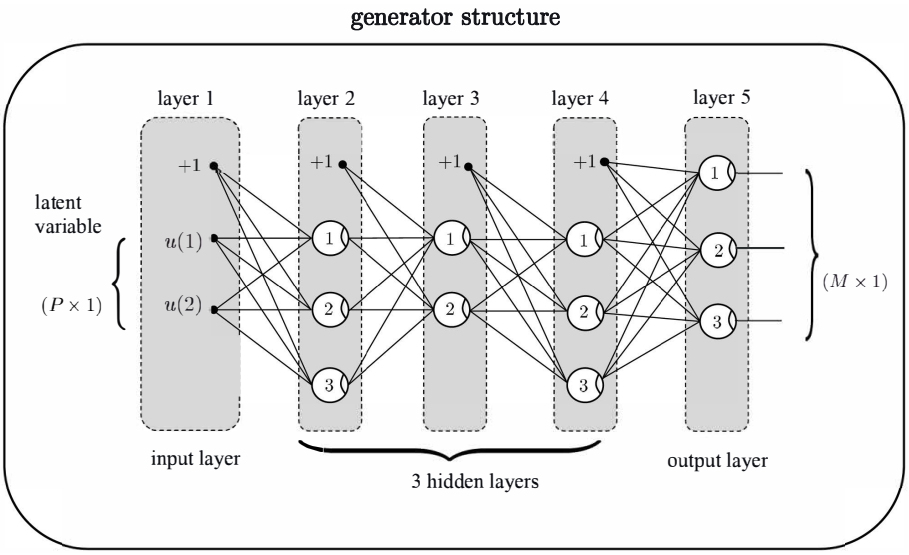

Figure 68.14 A generator structure with $M = 3$ attributes at the output layer (i.e., $\widehat{h} \in \mathbb{R}^3$), latent variables $\boldsymbol{u}$ of size $P = 2$, and 3 hidden layers.

The generator receives latent variables $\{u_n \in \mathbb{R}^P\}$ as input and generates approximate feature vectors $\{\widehat{h}_n \in \mathbb{R}^M\}$ as output. We can view the generator

as a mapping from the latent space u to the space of fake data, $\widehat{h}$. We denote the mapping by $\mathcal{G}(u_n)$, i.e.,

$$\boxed{\widehat{h}_n = \mathcal{G}(u_n)} \tag{68.108}$$

The output vectors $\{\widehat{h}_n\}$ will play the role of the "fake" features that are fed into the discriminator during training. In a sense, the generator in a GAN operates similarly to the decoder in a VAE by decoding latent variables and mapping them to synthetic features. The latent variables u_n are generally chosen as realizations of a Gaussian distribution, $\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)$. In this way, the generator samples from the standard Gaussian distribution and uses the samples $\{u_n\}$ to produce synthetic features $\{\widehat{h}_n\}$ that imitate samples from the (unknown) data distribution $f_{\boldsymbol{h}}(h)$.

We explained in the previous section that the discriminator seeks to maximize its ability to separate real from fake feature vectors. It does so by seeking the maximizer of the risk function (68.103) (or its regularized version). The generator, on the other hand, will seek to generate fake feature vectors that succeed in confusing the discriminator and reducing its ability to distinguish between real and fake data. By doing so, the generator will learn how to generate good synthetic data that look similar to the real data. The generator pursues its objective by seeking to minimize (rather than maximize) the same risk function as the discriminator, i.e., the generator maximizes the opposite risk function used by the discriminator:

$$\Big\{W_{\ell,g}^{\star}, \theta_{\ell,g}^{\star}\Big\} \;=\; \underset{\{W_{\ell,g},\theta_{\ell,g}\}}{\operatorname{argmax}} \; -\Big\{\mathbb{E}_{\,f_{\boldsymbol{h}}} \ln \mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{\,f_{\widehat{\boldsymbol{h}}}} \ln(1-\mathcal{D}(\widehat{\boldsymbol{h}}))\Big\} \tag{68.109}$$

The discriminator and generator components pursue their objectives (68.103) and (68.109) by adjusting their own internal weights and biases; the discriminator has no control over the internal structure of the generator and vice-versa. We are therefore faced with a *zero-sum game*, where the cost functions (also called payoffs in the context of game problems) that are being maximized by both players (the generator and discriminator) are opposites to each other.

68.4.3 Game Problem

If we continue to ignore regularization and persist a little more with the stochastic interpretation, then the objective of the combined generator-discriminator system is to solve the min-max problem:

$$\Big\{W_{\ell,g}^{\star}, \theta_{\ell,g}^{\star}, W_{\ell,d}^{\star}, \theta_{\ell,d}^{\star}\Big\} \;=\; \arg\min_{\substack{\{W_{\ell,g}\}\\ \{\theta_{\ell,g}\}}} \max_{\substack{\{W_{\ell,d}\}\\ \{\theta_{\ell,d}\}}} \Big\{\mathbb{E}_{\,f_{\boldsymbol{h}}} \ln \mathcal{D}(\boldsymbol{h}) + \mathbb{E}_{\,f_{\boldsymbol{u}}} \ln\Big(1-\mathcal{D}(\mathcal{G}(\boldsymbol{u}))\Big)\Big\} \tag{68.110}$$

where we replaced $\widehat{\boldsymbol{h}}$ by $\mathcal{G}(\boldsymbol{u})$ and the second expectation is now over the distribution of the latent variable, $\boldsymbol{u}$. Observe that only the second expectation term is dependent on the parameters of both the generator and discriminator through its

dependence on both $\mathcal{D}$ and $\mathcal{G}$. The first expectation term is dependent solely on the parameters of the discriminator. Assuming the generator structure is fixed, the discriminator would seek to maximize the risk such that $\mathcal{D}(\boldsymbol{h})$ is close to one and $\mathcal{D}(\mathcal{G}(\boldsymbol{u}))$ is close to zero. On the other hand, assuming the discriminator structure is fixed, the generator will seek to minimize the risk such that $\mathcal{D}(\mathcal{G}(\boldsymbol{u}))$ is close to one. Observe that with the discriminator structure fixed, the generator does not have any influence on $\mathcal{D}(\boldsymbol{h})$. Figure 68.15 shows a block diagram representation of the combined generator-discriminator system. The input to the discriminator is selected at random from the given dataset or from the fake samples by the generator.

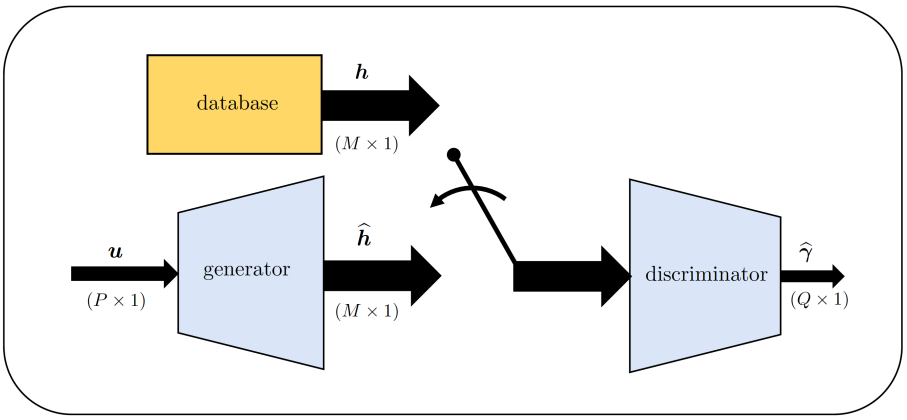

Figure 68.15 A block diagram representation of the GAN structure involving a generator, a discriminator, and a database of original feature vectors for training.

Example 68.5 (**Solution of game problem**) Motivated by the same stochastic formulation (68.110), let us consider the following min-max problem:

$$(\mathcal{G}^\star, \mathcal{D}^\star) \;=\; \arg\min_{\mathcal{G}} \max_{\mathcal{D}} \Big\{ \mathbb{E}_{\,f_{\boldsymbol{h}}} \ln \mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{\,f_{\widehat{\boldsymbol{h}}}} \ln\Big(1 - \mathcal{D}(\widehat{\boldsymbol{h}})\Big) \Big\} \tag{68.111}$$

where we are again ignoring the neural network structure for both the generator and discriminator. We denote the cost function by

$$\begin{aligned} J(\mathcal{G}, \mathcal{D}) \;&\triangleq\; \mathbb{E}_{\,f_{\boldsymbol{h}}} \ln \mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{\,f_{\widehat{\boldsymbol{h}}}} \ln(1 - \mathcal{D}(\widehat{\boldsymbol{h}})) \\ &\overset{(68.105)}{=} \int_{h \in \mathcal{H}} \Big(f_{\boldsymbol{h}}(h) \ln \mathcal{D}(h) \;+\; f_{\widehat{\boldsymbol{h}}}(h) \ln(1 - \mathcal{D}(h)) \Big) dh \end{aligned} \tag{68.112}$$

where in the rightmost term we used a change of variables from $\widehat{h}$ to h. The above expression is a function of both $\mathcal{G}$ and $\mathcal{D}$ since $\widehat{\boldsymbol{h}} = \mathcal{G}(\boldsymbol{u})$. We already know from (68.107) that the maximum of (68.111) over $\mathcal{D}$ (i.e., the optimal discriminator mapping) is given by

$$\mathcal{D}^\star(h) = \frac{f_{\boldsymbol{h}}(h)}{f_{\boldsymbol{h}}(h) + f_{\widehat{\boldsymbol{h}}}(h)} \tag{68.113}$$

We will now verify that the minimum of (68.111) over $\mathcal{G}$ (i.e., the optimal generator mapping) is achieved when $\mathcal{G}$ is able to match $f_{\widehat{\boldsymbol{h}}}(\cdot)$ with the true distribution $f_{\boldsymbol{h}}(\cdot)$. We denote this generator by $\mathcal{G}^\star(\boldsymbol{u})$. We establish the result by showing that the choice $f_{\widehat{\boldsymbol{h}}}(\cdot) = f_{\boldsymbol{h}}(\cdot)$ leads to the smallest possible payoff in (68.111).

Indeed, assume that $f_{\widehat{\boldsymbol{h}}}(\cdot) = f_{\boldsymbol{h}}(\cdot)$. Then, the likelihood value given by (68.113) will be $1/2$ and the discriminator will be confused about assigning h to the real or fake class. Substituting $\mathcal{D}^\star = 1/2$ into expression (68.112) and using $f_{\widehat{\boldsymbol{h}}}(\cdot) = f_{\boldsymbol{h}}(\cdot)$, we find that the corresponding payoff value will be

$$\begin{aligned} J(\mathcal{G}^\star, \mathcal{D}^\star) &= \int_{h\in\mathcal{H}} \Big(f_{\boldsymbol{h}}(h)\ln(1/2) \;+\; f_{\boldsymbol{h}}(h)\ln(1/2) \Big) dh \\ &= 2\ln(1/2) \underbrace{\int_{h\in\mathcal{H}} f_{\boldsymbol{h}}(h) dh}_{=1} \\ &= -\ln 4 \end{aligned} \tag{68.114}$$

If the distribution by the generator does not agree with the actual data distribution, then we can verify that the payoff value will be larger than $-\ln 4$. Indeed, when $f_{\widehat{\boldsymbol{h}}}(\cdot) \neq f_{\boldsymbol{h}}(\cdot)$ we get at the optimal discriminator:

$$\begin{aligned} J(\mathcal{G}, \mathcal{D}^\star) &= J(\mathcal{G}, \mathcal{D}^\star) + \ln 4 - \ln 4 \\ &= J(\mathcal{G}, \mathcal{D}^\star) + 2\ln 2 - \ln 4 \\ &= -\ln 4 + J(\mathcal{G}, \mathcal{D}^\star) + \int_{\mathcal{H}} f_{\boldsymbol{h}}(h)\ln 2\, dh + \int_{\mathcal{H}} f_{\widehat{\boldsymbol{h}}}(h)\ln 2\, dh \\ &\overset{(68.112)}{=} -\ln 4 + \int_{h\in\mathcal{H}} f_{\boldsymbol{h}}(h) \left\{ \ln 2 + \ln\left(\frac{f_{\boldsymbol{h}}(h)}{f_{\boldsymbol{h}}(h) + f_{\widehat{\boldsymbol{h}}}(h)} \right) \right\} dh \;+ \\ &\qquad \int_{\mathcal{H}} f_{\widehat{\boldsymbol{h}}}(h) \left\{ \ln 2 + \ln\left(\frac{f_{\widehat{\boldsymbol{h}}}(h)}{f_{\boldsymbol{h}}(h) + f_{\widehat{\boldsymbol{h}}}(h)} \right) \right\} dh \\ &= -\ln 4 + \int_{h\in\mathcal{H}} f_{\boldsymbol{h}}(h) \ln\left(\frac{f_{\boldsymbol{h}}(h)}{\frac{1}{2}(f_{\boldsymbol{h}}(h) + f_{\widehat{\boldsymbol{h}}}(h))} \right) dh \;+ \\ &\qquad \int_{\mathcal{H}} f_{\widehat{\boldsymbol{h}}}(h) \ln\left(\frac{f_{\widehat{\boldsymbol{h}}}(h)}{\frac{1}{2}(f_{\boldsymbol{h}}(h) + f_{\widehat{\boldsymbol{h}}}(h))} \right) dh \\ &= -\ln 4 + \underbrace{\mathcal{D}_{\mathrm{KL}}\Big(f_{\boldsymbol{h}} \,\|\, 0.5(f_{\boldsymbol{h}} + f_{\widehat{\boldsymbol{h}}}) \Big)}_{\geq 0} + \underbrace{\mathcal{D}_{\mathrm{KL}}\Big(f_{\widehat{\boldsymbol{h}}} \,\|\, 0.5(f_{\boldsymbol{h}} + f_{\widehat{\boldsymbol{h}}}) \Big)}_{\geq 0} \\ &\geq -\ln 4 \end{aligned} \tag{68.115}$$

where the last inequality is because KL divergences are nonnegative. This argument confirms that the smallest payoff for the min-max problem (68.111) is $-\ln 4$ and it is attained by the generator mapping $\mathcal{G}^\star(\boldsymbol{u})$ that leads to $f_{\widehat{\boldsymbol{h}}} = f_{\boldsymbol{h}}$. Another way to arrive at this conclusion is to note that the minimum value of (68.115) is achieved when

$$D_{\mathrm{KL}}\Big(f_{\boldsymbol{h}} \,\|\, 0.5(f_{\boldsymbol{h}} + f_{\widehat{\boldsymbol{h}}}) \Big) + D_{\mathrm{KL}}\Big(f_{\widehat{\boldsymbol{h}}} \,\|\, 0.5(f_{\boldsymbol{h}} + f_{\widehat{\boldsymbol{h}}}) \Big) = 0 \tag{68.116}$$

which occurs when $f_{\widehat{\boldsymbol{h}}} = f_{\boldsymbol{h}}$. We note in passing that for two arbitrary probability distributions $p_{\boldsymbol{x}}(x)$ and $q_{\boldsymbol{x}}(x)$, the quantity defined by

$$D_{\mathrm{JS}}(p\|q) \;\triangleq\; \frac{1}{2}\Big[D_{\mathrm{KL}}\Big(p\|(p+q)/2 \Big) + D_{\mathrm{KL}}\Big(q\|(p+q)/2 \Big) \Big] \tag{68.117}$$

is called the *Jenson-Shannon divergence* between $p_{\boldsymbol{x}}(x)$ and $q_{\boldsymbol{x}}(x)$. Therefore, the optimal generator mapping is seeking to minimize this divergence measure between $f_{\widehat{\boldsymbol{h}}}(h)$

and $f_{\boldsymbol{h}}(h)$. We conclude that the optimal solution of the two-player game problem (68.111) is given by

$$\mathcal{D}^{\star}(h) = \frac{1}{2}, \quad \mathcal{G}^{\star}(\boldsymbol{u}) \text{ such that } f_{\widehat{\boldsymbol{h}}}(h) = f_{\boldsymbol{h}}(h), \quad \text{payoff} = -\ln 4 \tag{68.118}$$

68.5 TRAINING OF GANS

If we return to the original empirical risk notation shown in (68.98), we find that the design of a generative adversarial network (GAN) involves solving a min-max problem over the following risk, where we are re-introducing regularization:

$$\begin{aligned} &\mathcal{P}(W,\theta) \\ &\triangleq \sum_{\ell=1}^{L_g-1} \rho\|W_{\ell,g}\|_{\mathrm{F}}^2 + \sum_{\ell=1}^{L_d-1} \rho\|W_{\ell,d}\|_{\mathrm{F}}^2 \;-\; \frac{1}{N}\sum_{n=0}^{N-1}\sum_{q=1}^{Q} \gamma_n(q)\ln\left(\widehat{\gamma}_n(q)\right) \end{aligned} \tag{68.119}$$

That is, the design of the GAN takes the form of a two-player game:

$$\left\{W_{\ell,g}^{\star}, \theta_{\ell,g}^{\star}, W_{\ell,d}^{\star}, \theta_{\ell,d}^{\star}\right\} \;=\; \underset{\substack{\{W_{\ell,g}\}\\ \{\theta_{\ell,g}\}}}{\arg\max} \; \underset{\substack{\{W_{\ell,d}\}\\ \{\theta_{\ell,d}\}}}{\min} \; \mathcal{P}(W,\theta) \tag{68.120}$$

with one player (the discriminator) attempting to minimize the empirical risk, while the second player (the generator) attempts to maximize it. In practice, one way to solve problem (68.120) is to alternate between the following two steps:

(a) (**Discriminator**) Fix the generator structure (i.e., fix its weight matrices and bias vectors). Perform a number of stochastic gradient *descent* iterations on the following risk (with an added subscript d) to adjust the weights and biases of the discriminator structure:

$$\mathcal{P}_d(W,\theta) \;\triangleq\; \sum_{\ell=1}^{L_d-1} \rho\|W_{\ell,d}\|_{\mathrm{F}}^2 \;-\; \frac{1}{N}\sum_{n=0}^{N-1}\sum_{q=1}^{Q} \gamma_n(q)\ln\left(\widehat{\gamma}_n(q)\right) \tag{68.121}$$

In other words, the discriminator focuses on solving:

$$\boxed{\begin{aligned} &\left\{W_{\ell,d}^{\star}, \theta_{\ell,d}^{\star}\right\} \;=\; \underset{\{W_{\ell,d},\theta_{\ell,d}\}}{\operatorname{argmin}} \\ &\left\{\sum_{\ell=1}^{L_d-1} \rho\|W_{\ell,d}\|_{\mathrm{F}}^2 - \frac{1}{N}\Big(\sum_{n\in\mathcal{N}_r} \ln\widehat{\gamma}_n(1) + \sum_{n\in\mathcal{N}_f} \ln\left(1-\widehat{\gamma}_n(1)\right)\Big)\right\} \end{aligned}} \tag{68.122}$$

This step amounts to training a neural network using the cross-entropy risk function and a softmax output layer. We already know how to perform this step by means of the backpropagation algorithm — recall listing (65.149).

(b) (**Generator**) Fix the discriminator structure (i.e., fix its weight matrices and bias vectors). Perform a number of stochastic gradient *ascent* iterations on the following risk (with an added subscript g) to adjust the weights and biases of the generator structure:

$$\mathcal{P}_g(W,\theta) = \sum_{\ell=1}^{L_g-1} \rho\|W_{\ell,g}\|_{\mathrm{F}}^2 \;-\; \frac{1}{N}\sum_{n=0}^{N-1}\sum_{q=1}^{Q} \gamma_n(q)\ln\left(\widehat{\gamma}_n(q)\right) \tag{68.123}$$

In other words, the generator focuses on solving

$$\left\{W_{\ell,g}^{\star},\theta_{\ell,g}^{\star}\right\} = \underset{\{W_{\ell,g},\theta_{\ell,g}\}}{\operatorname{argmax}} \left\{ \sum_{\ell=1}^{L_g-1} \rho\|W_{\ell,g}\|_{\mathrm{F}}^2 - \frac{1}{N}\sum_{n\in\mathcal{N}_f} \ln\Big(1-\widehat{\gamma}_n(1)\Big) \right\} \tag{68.124}$$

Observe that we are retaining only the fake samples since the generator does not have any control over the real samples.

There is, however, one difficulty with maximizing (68.124). During the early stages of training, when the fake data produced by the generator has not been "perfected" yet, the discriminator will be able to separate them with relative ease from the real data. That is, during the early stages of training it is likely that $\widehat{\gamma}_n(1) \approx 0$ for $n \in \mathcal{N}_f$. This is problematic because the function $\ln(1-x)$ has gradient close to zero when $x \to 0$; this effect is illustrated in Fig. 68.16. The vanishingly small gradient near $\widehat{\gamma}_n(1) \approx 0$ will slow down the training of the generator. For this reason, in practice, the empirical risk for the generator is modified and problem (68.124) is replaced by

$$\boxed{\left\{W_{\ell,g}^{\star},\theta_{\ell,g}^{\star}\right\} = \underset{\{W_{\ell,g},\theta_{\ell,g}\}}{\operatorname{argmax}} \left\{ \sum_{\ell=1}^{L_g-1} \rho\|W_{\ell,g}\|_{\mathrm{F}}^2 + \frac{1}{N}\sum_{n\in\mathcal{N}_f} \ln\left(\widehat{\gamma}_n(1)\right) \right\}} \tag{68.125}$$

This modification does not alter the overall goal for the generator, which is to drive $\widehat{\gamma}_n(1)$ to one. Clearly, by changing the risk function for the generator, the underlying game is no longer zero-sum since one payoff is not the opposite of the other anymore. Nevertheless, this adjustment is necessary to enable proper learning and avoid the vanishing gradient problem.

Notation

We will derive next the recursions for training the GAN. Before we initiate the derivation, we indicate that we will rely on the same notation used so far for neural networks and variational autoencoders. In particular, we will attach subscripts g and d to variables within the generator and discriminator, respectively.

In general, a GAN will consist of L_g total layers within the generator and L_d total layers within the discriminator. We denote the weight matrices and bias

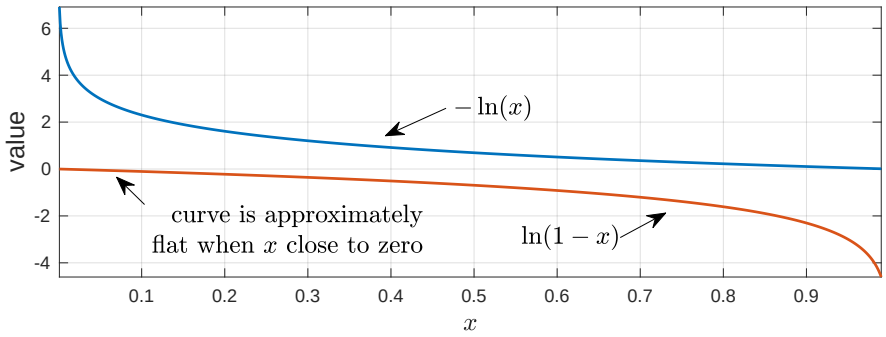

Figure 68.16 The function $\ln(1-x)$ is practically flat close to $x = 0$, which causes a vanishing gradient problem during training by backpropagation. This problem is alleviated by replacing $\ln(1-x)$ by $-\ln(x)$ in the risk function as explained in (68.125). This alternative function is not flat around $x = 0$.

vectors within the generator by

$$\Big\{(W_{1,g},\theta_{1,g}),\,(W_{2,g},\theta_{2,g}),\,\ldots,\,(W_{L_g-1,g},\theta_{L_g-1,g})\Big\} \tag{68.126}$$

with a subsscript g and sizes

$$W_{\ell,g} : n_{\ell,g} \times n_{\ell+1,g}, \quad \theta_{\ell,g} : n_{\ell+1,g} \times 1 \tag{68.127}$$

and those within the discriminator by

$$\Big\{(W_{1,d},\theta_{1,d}),\,(W_{2,d},\theta_{2,d}),\,\ldots,(W_{L_d-1,d},\theta_{L_d-1,d})\Big\} \tag{68.128}$$

with a subscript d and sizes

$$W_{\ell,d} : n_{\ell,d} \times n_{\ell+1,d}, \quad \theta_{\ell,d} : n_{\ell+1,d} \times 1 \tag{68.129}$$

We also denote the pre- and post-activation vectors at the internal layers of the generator by

$$\Big\{y_{1,g} = u,\,(z_{2,g},y_{2,g}),\,(z_{3,g},y_{3,g}),\ldots,(z_{L_g,g},y_{L_g,g})\Big\} \tag{68.130}$$

with a subscript g and sizes

$$y_{\ell,g} : n_{\ell,g} \times 1, \quad z_{\ell,g} : n_{\ell,g} \times 1, \;\; n_{1,g} = P, \quad n_{L_g,g} = M \tag{68.131}$$

The $M \times 1$ output vector of the generator is

$$\widehat{h} = y_{L_g,g} \tag{68.132}$$

Likewise, we denote the pre- and post-activation vectors at the internal layers of the decoder by

$$\Big\{y_{1,d} = h \text{ or } \widehat{h},\,(z_{2,d},y_{2,d}),\,(z_{3,d},y_{3,d}),\ldots,(z_{L_d,d},y_{L_d,d})\Big\} \tag{68.133}$$

with a subscript d and sizes

$$y_{\ell,d} : n_{\ell,d} \times 1, \quad z_{\ell,d} : n_{\ell,d} \times 1, \;\; n_{1,d} = M, \;\; n_{L_d,d} = Q \tag{68.134}$$

The input and output vectors are

$$y_{1,d} = h \text{ or } \widehat{h}, \quad y_{L_d,d} \;=\; \widehat{\gamma} \;\triangleq\; \left[\begin{array}{c} \widehat{\gamma}(1) \\ \widehat{\gamma}(2) \end{array}\right], \;\; \text{since } Q = 2 \tag{68.135}$$

68.5.1 Feedforward Propagation

Given some latent vector $u \in \mathbb{R}^P$ at the input of the generator, it is easy to describe the forward propagation for this vector through the generator and discriminator layers, as shown in listing (68.137) where $f(\cdot)$ denotes the activation function. We denote the softmax operation in the last layer by

$$\widehat{\gamma} = \text{softmax}(z) \iff \widehat{\gamma}(q) = \frac{z(q)}{\sum_{q'=1}^{Q} e^{z(q')}}, \; q = 1, \ldots, Q \tag{68.136}$$

Feedforward propagation through the GAN

start with $y_{1,g} = u \in \mathbb{R}^P$.

(propagation through generator)
repeat for $\ell = 1, \ldots, L_g - 1$:
$\quad z_{\ell+1,g} = W_{\ell,g}^{\mathsf{T}} \, y_{\ell,g} - \theta_{\ell,g}$
$\quad y_{\ell+1,g} = f\,(z_{\ell+1,g})$
end
$\widehat{h} = y_{L_g,g}$

$$y_{1,d} = \left\{\begin{array}{ll} \widehat{h}, & \text{when fake features are used} \\ h, & \text{when real features are used} \end{array}\right. \tag{68.137}$$

(propagation through discriminator)
repeat for $\ell = 1, \ldots, L_d - 1$:
$\quad z_{\ell+1,d} = W_{\ell,d}^{\mathsf{T}} \, y_{\ell,d} - \theta_{\ell,d}$
$\quad y_{\ell+1,d} = f\,(z_{\ell+1,d})$ (use softmax for last layer)
end
$z = z_{L_d,d}$
$\widehat{\gamma} = y_{L_d,d} = \text{softmax}(z)$

Note that we are denoting the output of the feedforward phase by $\widehat{\gamma}$ and the corresponding pre-activation signal by z so that $\widehat{\gamma} = \text{softmax}(z)$. Figure 68.17 shows a block diagram representation of the feedforward propagation scheme for a GAN involving one hidden layer in each of the generator and discriminator sections.

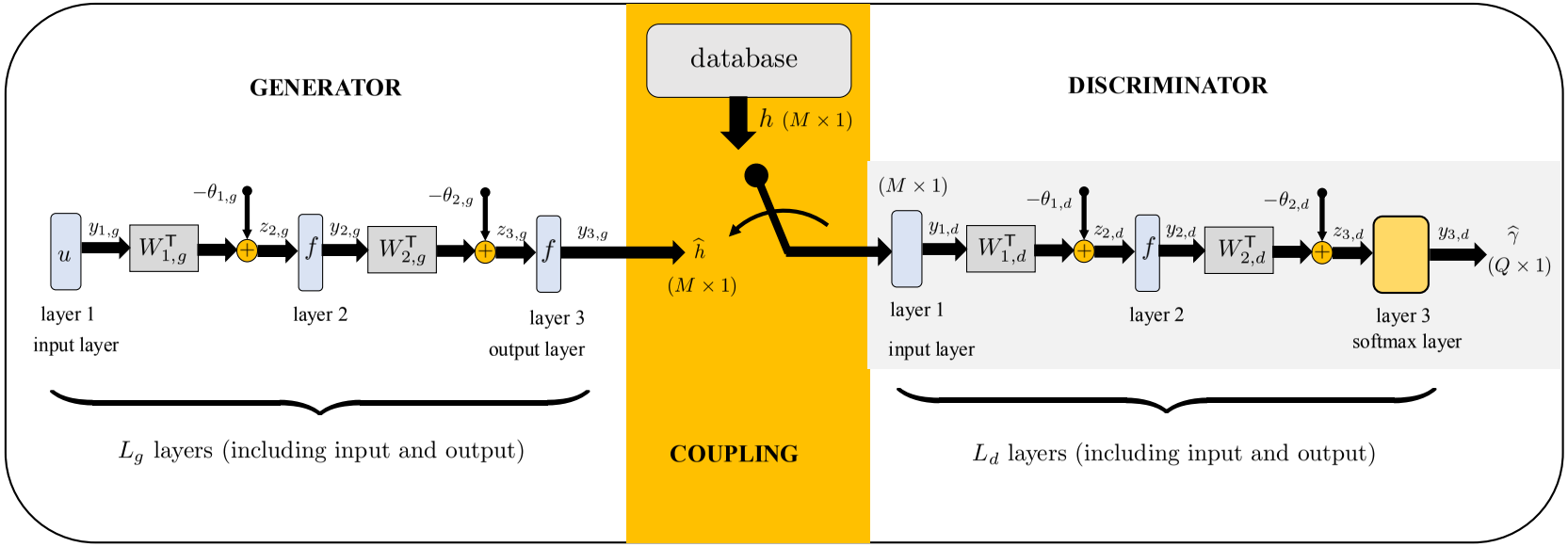

Figure 68.17 Block diagram representation of the cascade of generator and discriminator neural networks with a coupling layer, and a single hidden layer in each. The weight matrices and bias vectors within the generator and discriminator are denoted by $\{W_{\ell,g}, \theta_{\ell,g}\}$ and $\{W_{\ell,d}, \theta_{\ell,d}\}$, with subscripts g and d added. The generator has L_g total layers, including its input and output layers. The discriminator has L_d total layers, including its input and output layers.

68.5.2 Sensitivity Factors

In order to train the discriminator, we need to evaluate the gradients of the empirical risk $\mathcal{P}_d(W,\theta)$ defined by (68.121) relative to the individual entries of its weight matrices and bias vectors. The derivation is identical to what we did earlier in Sec. 65.7 since the discriminator is simply minimizing a regularized cross-entropy risk. We therefore review the main results here. Again, to simplify the notation, we drop the subscript, n, from all signals and reinstate it later.

Sensitivity factors through the discriminator

We first define sensitivity vectors and propagate them backward through the layers of the discriminator. We consider the $Q\times 1$ sensitivity vector associated with the last layer; its entries are given by

$$\begin{aligned}\delta_{L_d,d}(q) &= -\frac{\partial\Big(\sum_{q=1}^{Q}\gamma(q)\ln(\widehat{\gamma}(q))\Big)}{\partial z(q)},\quad q=1,2,\ldots,Q \\ &= -\frac{\partial}{\partial z(q)}\Big(\gamma(1)\ln(\widehat{\gamma}(1))+\gamma(2)\ln(\widehat{\gamma}(2))\Big),\quad \text{since } Q=2\end{aligned} \tag{68.138}$$

where we are attaching the subscript d to indicate that this sensitivity vector is related to the discriminator. We already know from (65.144) that

$$\delta_{L_d,d} = \widehat{\gamma}-\gamma, \tag{68.139}$$

In particular, since the discriminator will receive feature vectors that are either real ($\gamma(1)=1$ and $\gamma(2)=0$) or fake ($\gamma(1)=0$ and $\gamma(2)=1$), the above expression translates into

$$\delta_{L_d,d} = \begin{cases} \begin{bmatrix}\widehat{\gamma}(1)-1\\ \widehat{\gamma}(2)\end{bmatrix}, & \text{when } h \text{ is real} \\ \begin{bmatrix}\widehat{\gamma}(1)\\ \widehat{\gamma}(2)-1\end{bmatrix}, & \text{when } h \text{ is fake}\end{cases} \tag{68.140}$$

Next we evaluate the sensitivity vectors $\delta_{\ell,d}$ for the earlier layers within the discriminator, i.e., for $\ell = L_d-1, L_d-2,\ldots,2$:

$$\begin{aligned}\delta_{\ell,d}(j) &\triangleq -\frac{\partial\Big(\sum_{q=1}^{Q}\gamma(q)\ln(\widehat{\gamma}(q))\Big)}{\partial z_{\ell,d}(j)} \\ &= -\frac{\partial}{z_{\ell,d}(j)}\Big(\gamma(1)\ln(\widehat{\gamma}(1))+\gamma(2)\ln(\widehat{\gamma}(2))\Big)\end{aligned} \tag{68.141}$$

We know from (68.142) that

$$\boxed{\delta_{\ell,d} = f'(z_{\ell,d})\odot(W_{\ell,d}\delta_{\ell+1,d}),\quad \ell = L_d-1,\ldots,3,2} \tag{68.142}$$

Sensitivity factors through the generator

In a similar vein, let $\delta_{\ell,g}$ denote the sensitivity vectors within the generator with entries:

$$\delta_{\ell,g}(j) \triangleq \frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{\ell,g}(j)}, \quad \ell = L_g, L_g - 1, \ldots, 2 \tag{68.143}$$

A similar argument will show that

$$\boxed{\delta_{\ell,g} = f'(z_{\ell,g}) \odot (W_{\ell,g}\, \delta_{\ell+1,g}), \quad \ell = L_g - 1, \ldots, 3, 2} \tag{68.144}$$

To enable this backward recursion, we still need to evaluate its boundary value $\delta_{L_g,g}$. To do so, we need to show how to transfer the end value $\delta_{2,d}$ from the discriminator stage (68.142) to the generator stage.

Sensitivity factor at the transition

Thus, note that by definition

$$\begin{aligned} \delta_{L_g,g}(j) &= \frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{L_g,g}(j)}, \quad j = 1, 2, \ldots, M \\ &= \sum_{i=1}^{n_{2,d}} \frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{2,d}(i)} \frac{\partial z_{2,d}(i)}{\partial z_{L_g,g}(j)} \end{aligned} \tag{68.145}$$

For the second partial derivative we use the relation

$$z_{2,d}(i) = w_{ji,d}^{(1)} f(z_{L_g,g}(j)) - \theta_{1,d}(i) \tag{68.146}$$

so that

$$\frac{\partial z_{2,d}(i)}{\partial z_{L_g,g}(j)} = f'(z_{L_g,g}(j)) w_{ji,d}^{(1)} \tag{68.147}$$

For the first partial derivative in (68.145), we use the definition of $\delta_{2,d}$ to note that

$$\delta_{2,d}(i) \triangleq -\frac{\partial}{\partial z_{2,d}(i)} \Big(\gamma(1) \ln(\widehat{\gamma}(1)) + \gamma(2) \ln(\widehat{\gamma}(2)) \Big) \tag{68.148}$$

However, during the training of the generator (which is when we need to transfer the sensitivity factor from the discriminator stage into the generator stage), the purpose is for the generator to produce fake features and to "trick" the discriminator into believing that these features are real. Thus, the generator will assign label $\gamma(1) = 1$ to its fake features, which also means that $\gamma(2) = 0$ for these same features. It follows that during this stage of training:

$$\delta_{2,d}(i) = -\frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{2,d}(i)} \tag{68.149}$$

which is the term that appears in (68.145). Therefore, we arrive at the relation

$$\boxed{\delta_{L_g,g} = -\Big\{ f'(z_{L_g,g}) \odot \Big(W_{1,d} \delta_{2,d} \Big) \Big\}} \tag{68.150}$$

This expression tells us how to propagate the sensitivity factor $\delta_{2,d}$ at the leftmost end of the discriminator to obtain the boundary sensitivity factor $\delta_{L_g,g}$ at the rightmost end of the generator. Figure 68.18 provides a block diagram representation of the backward updates for the sensitivity factors through the discriminator and generator stages, assuming $L_g = 4$ layers within the generator and $L_d = 4$ layers within the discriminator (i.e., each stage has two hidden layers).

Expressions for the gradients

We can follow the same arguments from Sec. 65.4.3 to verify similarly that for the discriminator we have:

$$-\frac{\partial\Big(\sum_{q=1}^{Q}\gamma(q)\ln(\widehat{\gamma}(q))\Big)}{\partial W_{\ell,d}} = y_{\ell,d}\delta_{\ell+1,d}^{\mathsf{T}}, \quad \ell = L_d-1,\ldots,2,1 \tag{68.151}$$

$$-\frac{\partial\Big(\sum_{q=1}^{Q}\gamma(q)\ln(\widehat{\gamma}(q))\Big)}{\partial \theta_{\ell,d}} = -\delta_{\ell+1,d}, \quad \ell = L_d-1,\ldots,2,1 \tag{68.152}$$

while for the generator:

$$\frac{\partial\ln(\widehat{\gamma}(1))}{\partial W_{\ell,g}} = y_{\ell,g}\delta_{\ell+1,g}^{\mathsf{T}}, \quad \ell = L_g-1,\ldots,2,1 \tag{68.153}$$

$$\frac{\partial\ln(\widehat{\gamma}(1))}{\partial \theta_{\ell,g}} = -\delta_{\ell+1,g}, \quad \ell = L_g-1,\ldots,2,1 \tag{68.154}$$

68.5.3 Backpropagation Algorithm

We can now use the forward and backward recursions to train the generative adversarial network (GAN) by writing down a stochastic-gradient implementation with step-size $\mu > 0$. The recursions will alternate between running a couple of iterations to update the discriminator parameters, with the generator parameters kept fixed, followed by a couple of iterations to update the generator parameters, with the discriminator parameters kept fixed. This process is repeated until sufficient convergence is attained. The listing (68.155) provides a high-level description of the algorithm for ease of reference. The detailed recursions are shown in (68.160)–(68.161).

In description (68.155), we first freeze the parameters of the generator and repeat for K_d iterations the training of the discriminator. For each iteration, we select a random batch of B original feature vectors $\{h_b\}$ from the dataset. Their one-hot encoded labels will be $\{\gamma_b = \mathrm{col}\{1,0\}\}$. We also select B random latent variables $\{u_b \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)\}$ and feed them into the generator to produce B "fake" feature vectors $\{\widehat{h}_b\}$; their corresponding one-hot encoded labels will be $\{\gamma_b = \mathrm{col}\{0,1\}\}$. We feed the collection of $2B$ feature vectors into the discriminator and adjust its weight matrices and bias vectors by means of a backpropagation procedure, as detailed in (68.160).

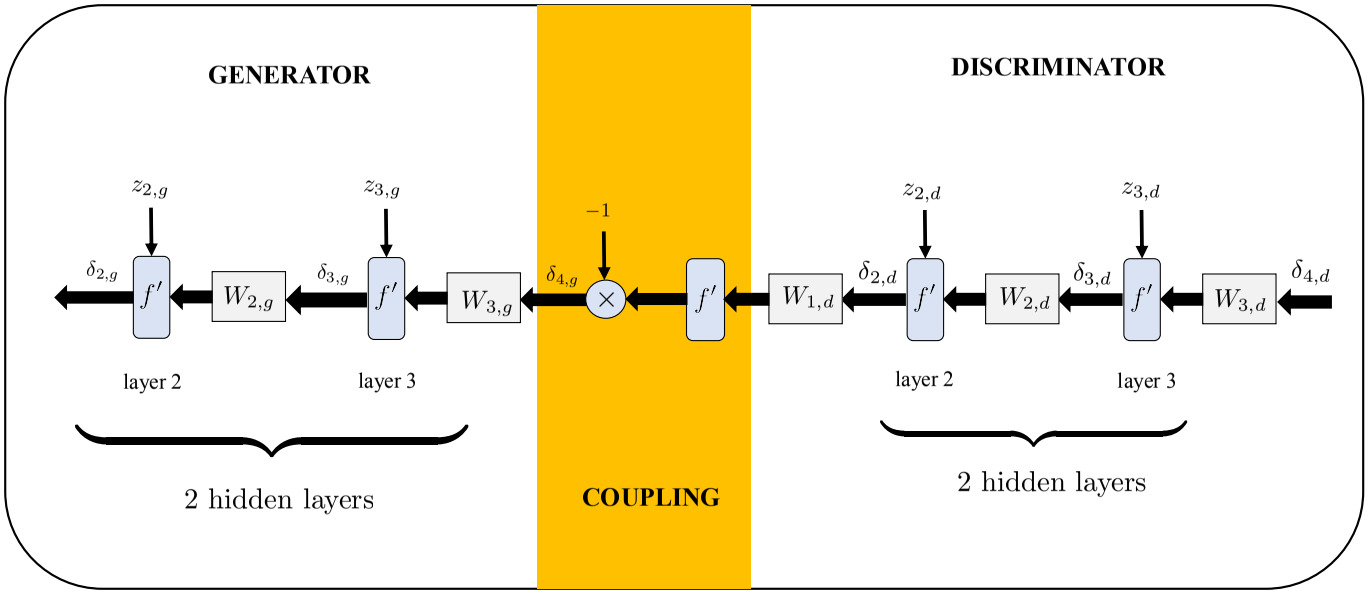

Figure 68.18 Block diagram representation of the backward updates for the sensitivity factors through the discriminator and generator stages, assuming $L_g = 4$ layers within the generator and $L_d = 4$ layers within the discriminator (i.e., each stage has two hidden layers).

High-level description of the training procedure for GANs

repeat until sufficient convergence for $m = 0, 1, \ldots$:
  repeat for K_d **iterations (listing (68.160))** :
    keep the parameters of the generator fixed
    select minibatch of B original feature vectors, $\{\boldsymbol{h}_b\}_{b=0}^{B-1}$
    generate minibatch of B "fake" feature vectors, $\{\widehat{\boldsymbol{h}}_b\}_{b=0}^{B-1}$
    update the parameters of the discriminator
  end
  repeat for K_g **iterations (listing (68.161))** :
    keep the parameters of the discriminator fixed
    select minibatch of B latent vectors $\{\boldsymbol{u}_b \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)\}_{b=0}^{B-1}$
    update the parameters of the generator
  end
end

(68.155)

We then repeat for the generator. We freeze the parameters of the discriminator and repeat for K_g iterations the training of the generator. For each iteration, we select a random batch of B latent vectors $\{u_b \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)\}$ and feed them through the cascade of generator-discriminator sections. We use the output signals $\{\widehat{\gamma}_n\}$ of the network to adjust the weight matrices and bias vectors of the generator by means of a backpropagation procedure, as detailed in (68.161). The process of alternating between training the discriminator and generator continues until sufficient convergence is attained, i.e., until the discriminator is essentially unable to distinguish between real and fake data with the entries of its output vector $\widehat{\gamma}$ split almost equally and close to $1/2$ each. Usually, $K_d = K_g = 1$, but we can also use $K_d > 1$ and $K_g = 1$, or other combinations.

We next provide the detailed implementation for training the generator and discriminator sections in the GAN. In the descriptions (68.160)–(68.161), we restore the subscript n to index the data; and we add a new subscript m as iteration index. Since we are already using two subscripts for each weighting matrix, such as $W_{\ell,g}$, where the second subscript indicates whether the matrix belongs to the generator or discriminator sections, we will adopt the notation $W_{\ell g,m}$ to indicate that this is the $\ell-$th weight matrix in the generator that is computed at iteration m. Note that we are blending the subscript g into the index ℓ of the weight matrix to avoid repeated commas in the subscript notation (such as writing $W_{\ell,g,m}$). Likewise, we will write $W_{\ell d,m}$, $\theta_{\ell g,m}$, and $\theta_{\ell d,m}$ at iteration m, where the subscripts g and d refer to layers within the generator and discriminator segments.

Remark 68.2. (Notation) In the listings (68.160)–(68.161) of the detailed recursions, we deal with both real and fake feature vectors, $\{h_b, \widehat{h}_b\}$. We need to introduce a slight variation in our notation in order to distinguish between internal variables that arise from the propagation of each type of data. We will continue to use the standard notation $\{z_{\ell d,b}, y_{\ell d,b}, \delta_{\ell d,b}\}$ for the signals and sensitivity factors arising from propagating a real feature vector h_b. We will also denote its one-hot encoded label vector by $\gamma_b = \text{col}\{1, 0\}$. In contrast, however, we will employ the notation $\{\widehat{z}_{\ell d,b}, \widehat{y}_{\ell d,b}, \widehat{\delta}_{\ell d,b}\}$, with an upper hat symbol, for the internal signals and sensitivity factors arising from propagating a fake feature vector $\widehat{h}_b$. We will denote its one-hot encoded label vector by $\widehat{\gamma}_b = \text{col}\{0, 1\}$. This latter notation *can be a source of confusion* because we have been using the hat-notation $\widehat{\gamma}$ to refer to the output of the GAN. In the listings (68.160)–(68.161), we denote the outputs of the GAN that correspond to h_b and $\widehat{h}_b$ by y_b and $\widehat{y}_b$, respectively. In particular, these are the output vectors at the last layer of the discriminator, i.e., $y_b = y_{L_d d,b}$ and $\widehat{y}_b = \widehat{y}_{L_d d,b}$:

$$\text{real input } (h_b, \gamma_b) \Longrightarrow \text{output } y_b \text{ and internal signals } \{z_{\ell d,b}, y_{\ell d,b}, \delta_{\ell d,b}\} \tag{68.156}$$

$$\text{fake input } (\widehat{h}_b, \widehat{\gamma}_b) \Longrightarrow \text{output } \widehat{y}_b \text{ and internal signals } \{\widehat{z}_{\ell d,b}, \widehat{y}_{\ell d,b}, \widehat{\delta}_{\ell d,b}\} \tag{68.157}$$

■

In the statement of algorithms (68.160)–(68.161) we assume the following conditions:

$$\begin{cases} \text{generator and discriminator structures with } L_g \text{ and } L_d \text{ layers, respectively;} \\ \text{output of generator has dimensions } M \times 1 \\ \text{random initial parameters } \{\boldsymbol{W}_{\ell g,-1}, \boldsymbol{W}_{\ell d,-1}, \boldsymbol{\theta}_{\ell g,-1}, \boldsymbol{\theta}_{\ell d,-1}\} \\ \text{dataset with real feature vectors } \{h_n \in \mathbb{R}^M\}. \end{cases} \tag{68.158}$$

At the end of the iterations, we set

$$\{W^\star_{\ell,g}, W^\star_{\ell,d}, \theta^\star_{\ell,g}, \theta^\star_{\ell,d}\} \leftarrow \{\boldsymbol{W}_{\ell g,m}, \boldsymbol{W}_{\ell d,m}, \boldsymbol{\theta}_{\ell g,m}, \boldsymbol{\theta}_{\ell d,m}\} \tag{68.159}$$

Once the GAN is trained, it can be used for at least two purposes during testing:

(a) (**Generation**) We can remove the discriminator and feed latent variables $\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)$ into the generator to produce samples $\widehat{h}$ that arise from essentially the same underlying data distribution.

(b) (**Classification**) We can remove the generator and use the discriminator to separate between real and fake data.

Mini-batch backpropagation for training GAN discriminator

repeat for K_d **iterations** :

select B latent vectors $\{\boldsymbol{u}_b \sim \mathcal{N}(0, I_P)\}_{b=0}^{B-1}$;

generate B fake samples $\{\widehat{\boldsymbol{h}}_b\}$ as follows:

repeat for $b = 0, 1, \ldots, B-1$**:**

$\boldsymbol{y}_{1g,b} = \boldsymbol{u}_b$

repeat for $\ell = 1, \ldots, L_g - 1$**:**

$\boldsymbol{z}_{\ell+1g,b} = W_{\ell g,m-1}^{\mathsf{T}}\, \boldsymbol{y}_{\ell g,b} - \theta_{\ell g,m-1}$

$\boldsymbol{y}_{\ell+1g,b} = f(\boldsymbol{z}_{\ell+1g,b})$

end

$\widehat{\boldsymbol{h}}_b = \boldsymbol{y}_{L_g g,b}$

end

(*forward propagation through discriminator*)

select B original feature vectors $\{\boldsymbol{h}_b\}_{b=0}^{B-1}$;

repeat for $b = 0, 1, \ldots, B-1$**:**

$\boldsymbol{y}_{1d,b} = \boldsymbol{h}_b$, $\widehat{\boldsymbol{y}}_{1d,b} = \widehat{\boldsymbol{h}}_b$ (feeding real/fake data)

$\gamma_b = \text{col}\{1, 0\}$, $\widehat{\gamma}_b = \text{col}\{0, 1\}$ (their one-hot encoded labels)

repeat for $\ell = 1, 2, \ldots, L_d - 1$

$\boldsymbol{z}_{\ell+1d,b} = \boldsymbol{W}_{\ell d,m-1}^{\mathsf{T}} \boldsymbol{y}_{\ell d,b} - \boldsymbol{\theta}_{\ell d,m-1}$

$\boldsymbol{y}_{\ell+1d,b} = f(\boldsymbol{z}_{\ell+1d,b})$, (use softmax function for last layer)

$\widehat{\boldsymbol{z}}_{\ell+1d,b} = \boldsymbol{W}_{\ell d,m-1}^{\mathsf{T}} \widehat{\boldsymbol{y}}_{\ell d,b} - \boldsymbol{\theta}_{\ell d,m-1}$

$\widehat{\boldsymbol{y}}_{\ell+1d,b} = f(\widehat{\boldsymbol{z}}_{\ell+1d,b})$, (use softmax function for last layer)

end

$\boldsymbol{y}_b = \boldsymbol{y}_{L_d d,b}$ $\widehat{\boldsymbol{y}}_b = \widehat{\boldsymbol{y}}_{L_d d,b}$, (output vectors for $\boldsymbol{h}_b$ and $\widehat{\boldsymbol{h}}_b$)

$\boldsymbol{\delta}_{L_d d,b} = \boldsymbol{y}_b - \boldsymbol{\gamma}_b$, $\widehat{\boldsymbol{\delta}}_{L_d d,b} = \widehat{\boldsymbol{y}}_b - \widehat{\boldsymbol{\gamma}}_b$ (boundary sensitivity factors)

end

(*backward propagation through discriminator*)

repeat for $\ell = L_d - 1, \ldots, 2, 1$**:**

$$\boldsymbol{W}_{\ell d,m} = (1 - 2\mu\rho)\boldsymbol{W}_{\ell d,m-1} - \frac{\mu}{2B} \sum_{b=0}^{B-1} \Big(\boldsymbol{y}_{\ell d,b} \boldsymbol{\delta}_{\ell+1d,b}^{\mathsf{T}} + \widehat{\boldsymbol{y}}_{\ell d,b} \widehat{\boldsymbol{\delta}}_{\ell+1d,b}^{\mathsf{T}} \Big)$$

$$\boldsymbol{\theta}_{\ell d,m} = \boldsymbol{\theta}_{\ell d,m-1} + \frac{\mu}{2B} \sum_{b=0}^{B-1} \Big(\boldsymbol{\delta}_{\ell+1d,b} + \widehat{\boldsymbol{\delta}}_{\ell+1d,b} \Big)$$

$$\boldsymbol{\delta}_{\ell d,b} = f'(\boldsymbol{z}_{\ell d,b}) \odot \Big(\boldsymbol{W}_{\ell d,m-1} \boldsymbol{\delta}_{\ell+1d,b} \Big),\ \ell \geq 2, b = 0, 1, \ldots, B-1$$

$$\widehat{\boldsymbol{\delta}}_{\ell d,b} = f'(\widehat{\boldsymbol{z}}_{\ell d,b}) \odot \Big(\boldsymbol{W}_{\ell d,m-1} \widehat{\boldsymbol{\delta}}_{\ell+1d,b} \Big),\ \ell \geq 2, b = 0, 1, \ldots, B-1$$

end

end

(68.160)

Mini-batch backpropagation for training the GAN generator

repeat for K_g **iterations** :

select B latent vectors $\{\boldsymbol{u}_b \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)\}_{b=0}^{B-1}$;

generate B fake samples $\{\widehat{\boldsymbol{h}}_b\}$ as follows:

repeat for $b = 0, 1, \ldots, B-1$**:**

$\boldsymbol{y}_{1g,b} = \boldsymbol{u}_b$

repeat for $\ell = 1, \ldots, L_g - 1$**:**

$\boldsymbol{z}_{\ell+1g,b} = \boldsymbol{W}_{\ell g,m-1}^{\mathsf{T}} \boldsymbol{y}_{\ell g,b} - \boldsymbol{\theta}_{\ell g,m-1}$

$\boldsymbol{y}_{\ell+1g,b} = f(\boldsymbol{z}_{\ell+1g,b})$

end

$\widehat{\boldsymbol{h}}_b = \boldsymbol{y}_{L_g g,b}$

end

(*forward propagation through discriminator*)

repeat for $b = 0, 1, \ldots, B-1$**:**

$\widehat{\boldsymbol{y}}_{1d,b} = \widehat{\boldsymbol{h}}_b$, $\widehat{\boldsymbol{\gamma}}_b = \text{col}\{1, 0\}$ (fake data pretending to be real)

repeat for $\ell = 1, 2, \ldots, L_d - 1$

$\widehat{\boldsymbol{z}}_{\ell+1d,b} = \boldsymbol{W}_{\ell d,m-1}^{\mathsf{T}} \widehat{\boldsymbol{y}}_{\ell d,b} - \boldsymbol{\theta}_{\ell d,m-1}$

$\widehat{\boldsymbol{y}}_{\ell+1d,b} = f(\widehat{\boldsymbol{z}}_{\ell+1d,b})$, (use softmax function for last layer)

end

$\widehat{\boldsymbol{y}}_b = \widehat{\boldsymbol{y}}_{L_d d,b}$, (output vector for $\widehat{\boldsymbol{h}}_b$)

$\widehat{\boldsymbol{\delta}}_{L_d d,b} = \widehat{\boldsymbol{y}}_b - \widehat{\boldsymbol{\gamma}}_b$

end

$$\widehat{\boldsymbol{\delta}}_{\ell d,b} = f'(\widehat{\boldsymbol{z}}_{\ell d,b}) \odot \Big(\boldsymbol{W}_{\ell d,m-1} \widehat{\boldsymbol{\delta}}_{\ell+1d,b}\Big), \quad \begin{array}{l} \ell = L_d - 1, \ldots, 3, 2 \\ b = 0, 1, \ldots, B-1 \end{array}$$

$$\boldsymbol{\delta}_{L_g g,b} = -f'(\boldsymbol{z}_{L_g g,b}) \odot \Big(\boldsymbol{W}_{1d,m-1} \widehat{\boldsymbol{\delta}}_{2d,b}\Big), \quad b = 0, 1, \ldots, B-1$$

(*backward propagation through generator*)

repeat for $\ell = L_g - 1, \ldots, 2, 1$ **(backward/generator)** :

$$\boldsymbol{W}_{\ell g,m} = (1 - 2\mu\rho)\boldsymbol{W}_{\ell g,m-1} + \frac{\mu}{B} \sum_{b=0}^{B-1} \boldsymbol{y}_{\ell g,b} \boldsymbol{\delta}_{\ell+1g,b}^{\mathsf{T}}$$

$$\boldsymbol{\theta}_{\ell g,m} = \boldsymbol{\theta}_{\ell g,m-1} - \frac{\mu}{B} \sum_{b=0}^{B-1} \boldsymbol{\delta}_{\ell+1g,b}$$

$$\boldsymbol{\delta}_{\ell g,b} = f'(\boldsymbol{z}_{\ell g,b}) \odot \Big(\boldsymbol{W}_{\ell g,m-1} \boldsymbol{\delta}_{\ell+1g,b}\Big), \ \ell \geq 2, b = 0, 1, \ldots, B-1$$

end

end

(68.161)

68.6 CONDITIONAL GANS

Motivated by the discussion in Sec. 68.3 we can similarly devise a conditional GAN solution that is able to generate synthetic images that belong to a particular class. To do so, we extend the input vectors to the generator and discriminator sections by adding a label vector defined as follows — see Fig. 68.19.

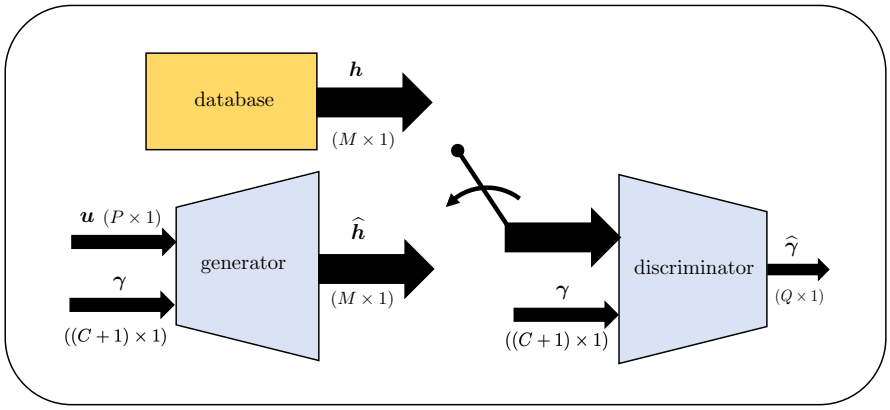

Figure 68.19 Diagram representation of one implementation for a *conditional* generative adversarial network (GAN) involving a cascade of generator and discriminator sections. The input to the generator is the extended vector $\text{col}\{u, \gamma\}$, while the input to the discriminator is either $\text{col}\{h, \gamma\}$ or $\text{col}\{\widehat{h}, \gamma\}$.

Assume there are C classes. That is, the real feature vectors h can belong to any of the classes $c \in \{1, 2, \ldots, C\}$. We augment these possibilities by adding one more class to account for fake feature vectors and employ one-hot encoding to represent the label vector. For example, assume there are $C = 5$ classes and that h happens to be a real vector in class $c = 2$. Then, its label vector will be $6-$dimensional with entries

$$\gamma(h) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ \hline 0 \end{bmatrix} \tag{68.162}$$

The unit entry appears in the second row with all other entries equal to zero; the last entry of $\gamma(h)$ is zero since h is a real (not fake) feature vector; actually, all real feature vectors will have this last entry set to zero. Assume instead that $\widehat{h}$ is a fake feature vector. Then, its label vector will again be $6-$dimensional with

entries

$$\gamma(\widehat{h}\,) = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \hline 1 \end{array}\right] \tag{68.163}$$

with a unit entry in the last row.

We see in Fig. 68.19 that the input to the generator becomes the extended vector

$$u' \triangleq \left[\frac{u}{\gamma}\right], \quad \text{of size } P+C+1 \tag{68.164}$$

This also means that the dimension $n_{1,g}$ changes from $n_{1,g} = P$ to $n_{1,g} = P + C + 1$. Likewise, the input to the discriminator is augmented by γ and becomes

$$h' \triangleq \left[\frac{h \text{ or } \widehat{h}}{\gamma}\right], \quad \text{of size } M+C+1 \tag{68.165}$$

This also means that the dimension $n_{1,d}$ changes from $n_{1,d} = M$ to $n_{1,d} = M + C + 1$. The output of the discriminator continues to be a two-dimensional (i.e., $Q = 2$) label vector $\widehat{\gamma}$ whose entries indicate the likelihood of the feature fed into the discriminator being real or fake. One can modify the structure of the GAN (and its conditional version) by requiring the output of the discriminator to be an extended label vector $\widehat{\gamma}$ of size $C + 1$ (rather than just 2). In this case, the entries of $\widehat{\gamma}$ will provide the likelihood of the feature vector being either fake (which would be given by the last entry of $\widehat{\gamma}$) or belonging to any of the C classes. We can modify the training algorithms (68.160)–(68.161) in a straightforward manner to allow for this extension — see Prob. 68.8.

Training algorithm

The empirical risk functions for training the conditional GAN continue to be the same as in (68.122) and (68.125). As such, the same training algorithms (68.160)–(68.161) continue to hold with one minor adjustment in relation to the sensitivity factor at the transition layer.

Specifically, recall from (68.145) that, by definition,

$$\delta_{L_g,g}(j) = \frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{L_g,g}(j)}, \quad j = 1, 2, \ldots, M \tag{68.166}$$

However, the input to the discriminator now involves two terms, $\widehat{h}$ and γ. As such, the derivation that led to (68.145) will need to be adjusted as follows to

account for the presence of γ (observe the difference in the last equality):

$$\begin{aligned}\delta_{L_g,g}(j) &= \sum_{i=1}^{n_{2,d}} \frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{2,d}(i)} \frac{\partial z_{2,d}(i)}{\partial z_{L_g,g}(j)} \\ &= \sum_{i=1}^{n_{2,d}} \frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{2,d}(i)} \left(\sum_{m=1}^{M} \frac{\partial z_{2,d}(i)}{\partial \widehat{h}(m)} \frac{\partial \widehat{h}(m)}{\partial z_{L_g,g}(j)} + \sum_{c=1}^{C+1} \frac{\partial z_{2,d}(i)}{\partial \gamma(c)} \frac{\partial \gamma(c)}{\partial z_{L_g,g}(j)} \right)\end{aligned} \tag{68.167}$$

Note that we are denoting the individual entries of the input label vector γ by $\{\gamma(c)\}$. However, since the entries of γ are independent of the entries of $z_{L_g,g}$ then

$$\frac{\partial \gamma(c)}{\partial z_{L_g,g}(j)} = 0 \tag{68.168}$$

and the last sum over c disappears. It follows, as before, that

$$\begin{aligned}\delta_{L_g,g}(j) &= \sum_{i=1}^{n_{2,d}} \frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{2,d}(i)} \left(\sum_{m=1}^{M} \frac{\partial z_{2,d}(i)}{\partial \widehat{h}(m)} \frac{\partial \widehat{h}(m)}{\partial z_{L_g,g}(j)} \right) \\ &= \sum_{i=1}^{n_{2,d}} \frac{\partial \ln(\widehat{\gamma}(1))}{\partial z_{2,d}(i)} w_{ji,d}^{(1)} f'(z_{L_g,g}(j)), \quad i = 1, 2, \ldots, M\end{aligned} \tag{68.169}$$

where the second equality follows from the fact that only $\widehat{h}(j)$ is dependent on $z_{L_g,g}(j)$ and

$$z_{2,d}(i) = w_{ji,d}^{(1)} \widehat{h}(j) - \theta_{1,d}(i) \tag{68.170}$$

$$\widehat{h}(j) = f(z_{L_g,g}(j)) \tag{68.171}$$

Therefore, expression (68.145) continues to hold. The main difference, however, is that the entries $w_{ji,d}^{(1)}$ that appear in (68.169) do not correspond to all the entries within $W_{1,d}$. This is because the row dimension of $W_{1,d}$ is now enlarged to $M + C + 1$:

$$W_{1,d} : (M + C + 1) \times n_{2,d}, \quad \text{since } n_{1,d} = M + C + 1 \tag{68.172}$$

Only the entries in the leading M rows of $W_{1,d}$ appear in (68.169). We should therefore replace (68.169) by

$$\delta_{L_g,g} = -f'(z_{L_g,g}) \odot \left(\bar{W}_{1,d} \delta_{2,d} \right) \tag{68.173}$$

where $\bar{W}_{1,d}$ contains the leading M rows of $W_{1,d}$:

$$\bar{W}_{1,d} = \begin{bmatrix} I_M & 0_{M \times C+1} \end{bmatrix} W_{1,d} \tag{68.174}$$

Example 68.6 (**Generation of handwritten digits using a GAN**) We reconsider the same MNIST database from Example 68.1 and train a GAN to generate synthetic handwritten digits. We however pre-process the data so that the normalized feature vectors assume values in the range $[-1, 1]$, since the activation functions at the output layer of

the generator are set to the tanh function.

We construct a GAN with $L_g = 5$ layers in the generator stage and $L_d = 5$ layers in the discriminator stage. In this way, each layer has three hidden layers. The size of the input layer for the generator is $n_{1,g} = 2$, which is the size of the latent variable ($P = 2$), while the size of each of the three hidden layers in the generator is $n_{2,g} = 256$, $n_{3,g} = 512$, and $n_{4,g} = 1024$ neurons. Likewise, the size of the input layer to the discriminator is $n_{1,d} = 784$, which is the size of the feature vectors, while the size of each of the three hidden layers is $n_{2,d} = 1024$, $n_{3,d} = 512$, and $n_{4,g} = 256$ neurons. The size of the output layer of the discriminator is $n_{L_d,d} = 2$ (i.e., $Q = 2$).

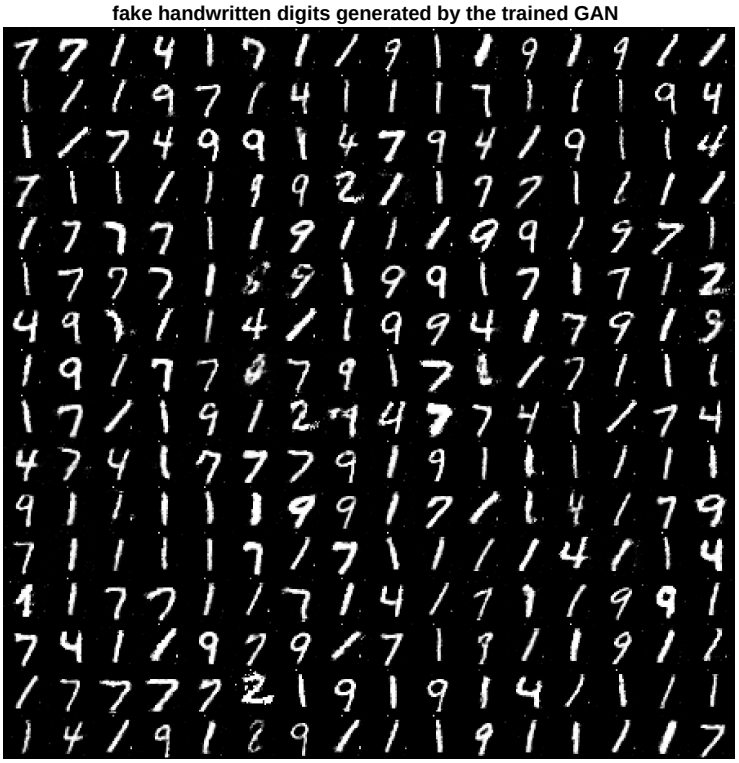

Figure 68.20 Synthetic handwritten images generated by feeding Gaussian latent values $\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)$ through the generator of a trained GAN and plotting the resulting output vectors in image format.

We set the activation functions for all hidden layers in both the generator and discriminator to the ReLu function. However, we set the activation function at the output layer of the generator to the tanh function and at the output layer of the discriminator to the softmax function.

We train the GAN using the ADAM recursions (instead of the stochastic-gradient recursions; we explained earlier in Example 65.3 what adjustments are necessary to run ADAM). We employ the following step-size, regularization, and forgetting factor

parameters:

$$\mu = 0.0002, \quad \rho = 0.0001 \tag{68.175a}$$

$$\beta_{w,1} = \beta_{\theta,1} = 0.500, \quad \text{(for ADAM implementation)} \tag{68.175b}$$

$$\beta_{w,2} = \beta_{\theta,2} = 0.999 \tag{68.175c}$$

$$\epsilon = 1 \times 10^{-8} \tag{68.175d}$$

We run an ADAM-version of the backpropagation algorithm (68.160)–(68.161) with mini-batches of size $B = 128$ samples. We also set $K_d = K_g = 1$. We perform 200 passes over the training data; the data is randomly reshuffled at the start of each pass. The weight parameters are initialized according to the schemes (65.93b)–(65.93c).

The training of GANs is generally difficult and it can fail in many ways due to the competition between the discriminator and generator modules. Sources of instability include the vanishing gradient effect and the *mode collapse problem* where the GAN ends up producing only one or a small subset of images regardless of the input latent vector (this behavior is apparent in Fig. 68.20). Also, during the initial stages of training when the generator has not yet perfected the generation of images that can fool the discriminator, the latter will be able to easily discriminate between real and fake images. Some ad-hoc techniques that can help ameliorate these convergence problems are (a) to decrease the value of the step-size parameter; (b) to add some small zero-mean Gaussian noise to the real or fake feature vectors $\{h_n, \widehat{h}_n\}$ that are fed into the discriminator during training; and (c) to use soft (or smooth) labels for the real features during training, such as setting $\gamma_n = \text{col}\{0.9, 0\}$ when h_n is real instead of the hard value at $\gamma_n = \text{col}\{1, 0\}$. These techniques interfere with the learning ability of the discriminator and provide an opportunity for the generator to improve its performance.

In the plot, the grayscale for the images is normalized so that the maximum value in each image is displayed as white, while the minimum value is displayed as black. During testing, we disconnect the generator from the discriminator. We generate 256 Gaussian latent variables $\boldsymbol{u} \sim \mathcal{N}_{\boldsymbol{u}}(0, I_P)$ and feed them into the generator with the weights and biases fixed at the values obtained at the end of the training phase. We generate the corresponding output vectors $\widehat{h}$ and plot them in image form in Fig. 68.20. These are artificially-generated (fake or synthetic) images.

68.7 COMMENTARIES AND DISCUSSION

Variational autoencoders. The machinery of variational autoencoders was introduced independently by Kingma and Welling (2014,2019) and Rezende, Mohamed, and Wierstra (2014). An overview appears in Doersch (2016). VAEs represent a powerful approach to learning complex distributions; they provide generative models that can be sampled to produce data that share similar statistical properties as the features used to train the VAEs — see, e.g., the discussion in Bengio, Courville, and Vincent (2013). They have been successfully applied to generate different types of data such as handwritten digits and human faces, including in the original references mentioned above. The application to recommender systems in Example 68.2 is from Liang *et al.* (2018).

We explained in the text that there is not much loss in generality in assuming a standard Gaussian model for the latent variable $\boldsymbol{u}$. The main reason was the fact that it is possible to transform this distribution to almost any other distribution by a proper mapping — see, e.g., Devroye (1986), Papoulis (1991), and Leon-Garcia (2008). The resulting VAE structure ends up performing one form of regularization. If we examine

expressions (68.20)–(68.25) and (68.37), we observe that the empirical risk for variational autoencoders consists of two terms: a quadratic term involving the difference $\|h_n - \widehat{h}_n\|^2$ and a regularization term that approximates the KL divergence between the distribution of the latent variable, $f_{\boldsymbol{u}}(u)$, and the conditional approximation $q_{\boldsymbol{u}|\boldsymbol{h}}(u|h)$.

Generative adversarial networks. The framework of generative adversarial networks (GANs) was introduced by Goodfellow *et al.* (2014) in parallel with the introduction of variational autoencoders (VAEs) by Kingma and Welling (2014,2019) and Rezende, Mohamed, and Wierstra (2014). Both classes of GAN and VAE networks enable sampling from an underlying distribution without the need to learn the distribution. GANs tend to lead to sharper synthetic images (at least subjectively) than VAEs and have been received with substantial excitement in the field of machine learning. They have found applications in a broad range of areas, including in advertising, face generation, and video gaming, by allowing computers to generate images that look deceptively realistic. GANs are also driving the emergence of deepfake technology, with "fake" videos showing a person uttering sentences that they have not even spoken.

The superior performance of GANs is largely driven by the competition between its generator and discriminator components, which play a zero-sum game. The generator keeps adjusting its internal structure until the discriminator is not able to distinguish between real and fake data anymore. The results of Examples 68.4 and 68.5 on the optimal discriminator strategy and the solution of the relevant game problem are from Goodfellow *et al.* (2014). Overviews of GANs appear in Cresswell *et al.* (2018), Wang, She, and Ward (2019), and Goodfellow *et al.* (2020). Interestingly, a similar GAN architecture was proposed a few years earlier in a blog post by Niemitalo (2010); it consisted of a generator and a classifier, with the input to the classifier arising either from data produced by the generator or from a dataset. This earlier structure does not include a source of randomness and behaves more like a *conditional* GAN — see Mirza and Osindero (2014).

Generative adversarial networks (GANs) are notoriously hard to train — see Salimans *et al.* (2016); they suffer from several failure modes and difficulty to converge. This is largely due to the competition between the generator and discriminator sections while vying to reach an equilibrium solution. There is also no clear stopping criterion for training and no assessment metric to compare against other solution methods. The training of GANs can employ dropout (especially within the generator) and can also benefit from batch normalization.

There exist many variations of GANs, intended to improve their training, exploit their potential more fully, as well as address their limitations and broaden their applicability. We describe one variation below, known as Wasserstein GAN. In Prob. 68.8 we consider another variation. We explain there that the structure of the GAN can be extended by requiring the output of the discriminator to be an extended label vector $\widehat{\gamma}$ of size $C+1$ (rather than of size 2), where C is the number of classes for the real data. In this case, the entries of the output vector will provide the likelihood of the feature vector being either fake (which is given by the last entry of $\widehat{\gamma}$) or belonging to any of the C real classes – see Springenberg (2016) and Odena (2016). It has been observed in practice that this structure with an extended output label vector leads to improved performance by generating (subjectively) better samples. Some other GAN variations are considered in the problems.

There have also been prior works using adversarial (game-theoretic) techniques for learning, including the adversarial curiosity (AC) and predictability minimization (PM) methods from Schmidhuber (1991,1992a) — see also Schmidhuber, Eldracher, and Foltin (1996). The article by Schmidhuber (2019) comments on how GANs are related to these earlier developments.

Wasserstein GANs. In the GAN formulation described in the text, the discriminator is a classifier that predicts the likelihood of a feature vector being real or fake. The Wasserstein GAN formulation from Arjovsky, Soumith, and Bottou (2017) considers

an alternative construction that leads to more stable performance; it helps alleviate the vanishing gradient problem, as well as the *mode collapse problem* that GANs are subject to. In this latter case, since the GAN is trained to "fool" the discriminator, it often ends up learning to generate a particular "fake" feature vector (or a small subset of feature vectors) and *only* that subset rather than a different feature vector for every different random latent variable at the input. The Wasserstein construction is based on replacing the discriminator by a *critic*, which continues to be a feedforward neural network albeit one with a single output node and a linear output layer. The output of the critic is used to score how realistic the feature vector is. Wasserstein GAN replaces the stochastic formulation (68.111) by one of the form:

$$(\mathcal{G}^\star, \mathcal{D}^\star) \;=\; \operatorname*{argmin}_{\mathcal{G}} \max_{\mathcal{D}} \Big\{ \mathbb{E}_{\,f_{\boldsymbol{h}}} \mathcal{D}(\boldsymbol{h}) \;-\; \mathbb{E}_{\,f_{\widehat{\boldsymbol{h}}}} \mathcal{D}(\widehat{\boldsymbol{h}}) \Big\} \tag{68.176}$$

Observe that the logarithm operation is removed and the cost involves the difference between two expectations: one is the average score over the distribution of real data and the second is the average score over the distribution of fake data. In this way, the discriminator works to maximize the difference between both average scores, while the generator works to bring them closer together. The reason for the designation "Wasserstein" is because given two distributions $p_{\boldsymbol{x}}(x)$ and $q_{\boldsymbol{x}}(x)$, the Wasserstein distance (also called the "Earth mover" distance) between them is defined by

$$W(p\|q) \;\triangleq\; \sup_{\|g\|_L \le 1} \Big\{ \mathbb{E}_{\,\boldsymbol{x}\sim p_{\boldsymbol{x}}}\, g(\boldsymbol{x}) \;-\; \mathbb{E}_{\,\boldsymbol{x}\sim q_{\boldsymbol{x}}}\, g(\boldsymbol{x}) \Big\} \tag{68.177}$$

where the first expectation is over the distribution $p_{\boldsymbol{x}}(x)$ and the second expectation is over the distribution $q_{\boldsymbol{x}}(x)$. The supremum is computed over the space of functions with Lipschitz constant bounded by one, written as $\|g\|_L \le 1$ and used to refer to functions that satisfy:

$$\|g(x) - g(y)\| \le \|x - y\| \tag{68.178}$$

If we imagine each pdf as representing a pile of sand then, informally, the Wasserstein distance gives an indication of the amount of sand that needs to be moved to transform one distribution to the other; hence, the name "Earth mover" distance. The measure (68.177) is also referred to as the maximum mean discrepancy (MMD) distance by Gretton *et al.* (2007), Dziugaite, Roy, and Ghahramani (2015), and Yujia, Swersky, and Zemel (2015). Expression (68.177) is actually a dual representation for a special case of a broader definition of the Wasserstein distance. The term was coined by Dobrushin (1970) after the work of Wasserstein (1969).

In terms of empirical risks, we can replace problem (68.122) by

$$\Big\{ W_{\ell,d}^\star, \theta_{\ell,d}^\star \Big\} \;=\; \operatorname*{argmin}_{\{W_{\ell,d}, \theta_{\ell,d}\}} \left\{ \sum_{\ell=1}^{L_d-1} \rho \|W_{\ell,d}\|_{\mathrm{F}}^2 - \frac{1}{N} \Big(\sum_{n\in\mathcal{N}_r} \mathcal{D}(h_n) - \sum_{n\in\mathcal{N}_f} \mathcal{D}(\widehat{h}_n) \Big) \right\} \tag{68.179}$$

and problem (68.124) by

$$\Big\{ W_{\ell,g}^\star, \theta_{\ell,g}^\star \Big\} \;=\; \operatorname*{argmax}_{\{W_{\ell,g}, \theta_{\ell,g}\}} \left\{ \sum_{\ell=1}^{L_g-1} \rho \|W_{\ell,g}\|_{\mathrm{F}}^2 + \frac{1}{N} \sum_{n\in\mathcal{N}_f} \mathcal{D}(\widehat{h}_n) \right\} \tag{68.180}$$

where we retain the fake feature vectors. We can proceed from here to derive a training algorithm in a manner similar to the arguments used in the text — see Prob. 68.11.

PROBLEMS

68.1 Establish the statement after (68.19), namely, that we can transform a standard Gaussian-distributed random variable to any other distribution by means of suitable nonlinear mappings.

68.2 Establish equality (68.82).

68.3 The expression for the ELBO in (68.7) is sometimes modified to include a scaling term in the following manner:

$$\text{ELBO} \;\triangleq\; \mathbb{E}_q\Big(\ln f(h|u)\Big) - \kappa\, D_{\text{KL}}\Big(q(u|h)\,\|\,f(u)\Big)$$

for some scalar $\kappa \in (0,1]$. Explain how the listing of algorithm (68.79) would be modified in this case.

68.4 Show that the training algorithm (68.79) continues to hold for conditional variational autoencoders, with the main difference being the size of the input vectors to both the encoder and decoder, which are enlarged to $h' = \text{col}\{h,\gamma\}$ and $u' = \text{col}\{u,\gamma\}$ with dimensions $M+C$ and $P+C$, respectively. Here, C is the number of classes and the class variable γ is $C\times 1$.

68.5 The structure shown in the earlier Fig. 68.11 for the conditional variational autoencoder feeds the label variables into both the encoder and decoder. This can be inconvenient during testing if the purpose is to use the trained encoder to perform compression (which should not depend on the label information). An alternative structure would be to consider adding a separate feedforward neural network, in parallel with the encoder network, with the same input h and a softmax output layer that generates a $C\times 1$ vector of probabilities $\widehat{\gamma}$; the $c-$th entry of $\widehat{\gamma}$ would correspond to the likelihood that h belongs to class c. In this way, the feature h feeds through two parallel networks with one network generating output $\{\mu, a\}$ and the other network generating output $\widehat{\gamma}$. The input to the decoder becomes $u' = \text{col}\{u,\widehat{\gamma}\}$. Derive the algorithm for training this structure, which is shown in Fig. 68.21. Write down the corresponding empirical risk.

68.6 How would recursions (68.79) change if we incorporate dropout into the operation of the training algorithm?

68.7 How would recursions (68.79) change if we incorporate batch normalization into the operation of the training algorithm?

68.8 We indicated in Sec. 68.6, right after (68.165), that we can modify the structure of the GAN by requiring the output of the discriminator to be an extended label vector $\widehat{\gamma}$ of size $C+1$ (rather than of size 2). In this case, the entries of this output vector will provide the likelihood of the feature vector being either fake (which is given by the last entry of $\widehat{\gamma}$) or belonging to any of the C real classes. Explain how the training algorithms (68.160)–(68.161) should be modified to account for this variation.

68.9 Design a GAN training algorithm when the binary classification labels generated by the discriminator are flipped, with label 1 corresponding to fake features and label 0 corresponding to real features, so that the stochastic formulation (68.111) is replaced by one of the form:

$$(\mathcal{G}^\star,\mathcal{D}^\star) \;=\; \arg\min_{\mathcal{G}}\max_{\mathcal{D}}\Big\{\mathbb{E}_{f_{\boldsymbol{h}}}\ln\Big(1-\mathcal{D}(\boldsymbol{h})\Big) \;+\; \mathbb{E}_{f_{\boldsymbol{u}}}\ln\mathcal{D}(\mathcal{G}(\boldsymbol{u}))\Big\}$$

Argue further that the above problem can be replaced by

$$(\mathcal{G}^\star,\mathcal{D}^\star) \;=\; \arg\max_{\mathcal{G}}\min_{\mathcal{D}}\Big\{\mathbb{E}_{f_{\boldsymbol{h}}}\ln\mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{f_{\boldsymbol{u}}}\ln\Big(1-\mathcal{D}(\mathcal{G}(\boldsymbol{u}))\Big)\Big\}$$

68.10 The output layer of the discriminator in a GAN implementation need not rely on a softmax calculation, and the empirical risk for the discriminator need not be limited

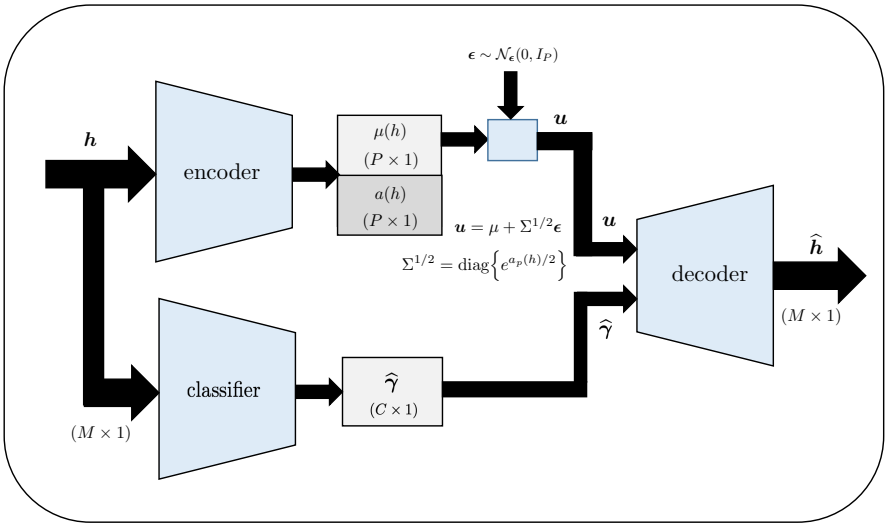

Figure 68.21 An alternative structure for conditional variational autoencoders, described in Prob. 68.5.

to the cross-entropy formulation. For example, design a GAN training algorithm when the stochastic formulation (68.111) is replaced by one of the form:

$$(\mathcal{G}^\star, \mathcal{D}^\star) \;=\; \arg\max_{\mathcal{G}} \min_{\mathcal{D}} \Big\{ \mathbb{E}_{\,f_{\boldsymbol{h}}} \mathcal{D}(\boldsymbol{h}) \;+\; \mathbb{E}_{\,f_{\boldsymbol{u}}} \max\Big\{0, 1 - \mathcal{D}(\mathcal{G}(\boldsymbol{u}))\Big\}\Big\}$$

In this case, the discriminator produces high output values for fake feature vectors and low output values for real feature vectors.

68.11 Use the empirical risks (68.179)–(68.180) to derive recursions for training the generator and critic stages of a Wasserstein GAN.

REFERENCES

Arjovsky, M., C. Soumith, and L. Bottou (2017), "Wasserstein GAN," *Proc. Intern. Conference on Machine Learning* (ICML), PMLR, pp. 214–223, Sydney, Australia. Also available online at arXiv:1701.07875.

Bengio, Y., A. Courville, and P. Vincent (2013), "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828.

Cresswell, A., T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. Bharath (2018), "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65.

Devroye, L. (1986), *Sample-Based Non-Uniform Random Variate Generation*, Springer-Verlag, NY.

Dobrushin, R. L. (1970), "Definition of a system of random variables by conditional distributions," *Teor. Verojatnost. i Primenen.*, vol. 15, pp. 469–497 (in Russian).

Doersch, C. (2016), "Tutorial on variational autoencoders," survey article *available online at arXiv:1606.05908.*

Dziugaite, G. K., D. M. Roy, and Z. Ghahramani (2015), "Training generative neural networks via maximum mean discrepancy optimization," *Proc. Conference on Uncertainty in Artificial Intelligence* (AUI), pp. 258–267, Amsterdam, The Netherlands.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014), "Generative adversarial networks," *Proc. Advances Neural Information Processing Systems* (NIPS), pp. 2672–2680, Montreal, Canada.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2020), "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 1, pp. 139–144.

Gretton, A., K. M. Borgwardt, M. Rasch, B. Scholkopf, and A. J. Smola (2007), "A kernel method for the two-sample problem," in *Proc. Advances in Neural Information Processing Systems* (NIPS), B. Scholkopf, J. C. Platt, and T. Hoffman, *Eds.*, pp. 513–520, MIT Press.

Kingma, D. P. and M. Welling (2014), "Auto-encoding variational Bayes," presented at *International Conference on Learning Representations* (ICLR), pp. 1–14, Banff, Canada. Also available at arXiv:1312.6114.

Kingma, D. P. and M. Welling (2019), "An introduction to variational autoencoders," *Foundations and Trends in Machine Learning*, vol. 12, no. 4, pp. 307–392.

Leon-Garcia, A. (2008), *Probability, Statistics, and Random Processes For Electrical Engineering,* 3rd edition, Prentice Hall, NJ.

Liang, D., R. G. Krishnan, M. D. Hoffman, and T. Jebara (2018), "Variational autoencoders for collaborative filtering," *Proc. World Wide Web Conference*, pp. 689–698, Lyon, France. Also available online at arXiv:1802.05814.

Mirza, M. and S. Osindero (2014), "Conditional generative adversarial nets," *available online at arXiv:1411.1784*.

Niemitalo, O. (2010), "A method for training artificial neural networks to generate missing data within a variable context," *available on Internet Archive 2010* at the link https://web.archive.org/web/20120312111546/http://yehar.com:80/blog/?p=167.

Odena, A. (2016), "Semi-supervised learning with generative adversarial networks," *available online at arXiv:1606.01583*.

Papoulis, A. (1991), *Probability, Random Variables, and Stochastic Processes,* 3rd edition, McGraw-Hill, NY.

Rezende, D. J., S. Mohamed, and D. Wierstra (2014), "Stochastic backpropagation and approximate inference in deep generative models," *Proc. International Conference on Machine Learning* (ICML), PMLR, vol. 32, no. 2, pp. 1278–1286, Beijing, China.

Salimans, T., I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen (2016), "Improved techniques for training GANs," *Proc. Advances Neural Information Processing Systems* (NIPS), pp. 1–9, Barcelona, Spain.

Schmidhuber, J. (1991), "A possibility for implementing curiosity and boredom in model-building neural controllers," in J. A. Meyer and S. W. Wilson, *Eds.*, *Proc. International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 222–227, MIT Press.

Schmidhuber, J. (1992a), "Learning factorial codes by predictability minimization," *Neural Computation*, vol. 4, no. 6, pp. 863–879.

Schmidhuber, J. (2019), "Unsupervised minimax: Adversarial curiosity, generative adversarial networks, and predictability minimization," *available online at the link arXiv:1906.04493*.

Schmidhuber, J., M. Eldracher, and B. Foltin (1996), "Semilinear predictability minimization produces well-known feature detectors," *Neural Computation*, vol. 8, no. 4, pp. 773–786, 1996.

Springenberg, J. T. (2016), "Unsupervised and semi-supervised learning with categorical generative adversarial networks," *Proc. Inter. Conference on Learning Representations* (ICLR), pp. 1–20, San Juan, Puerto Rico. Also available online at arXiv:1511.06390, 20 pages, 2015.

Wang, Z., Q. She, and T. E. Ward (2019), "Generative adversarial networks in computer vision: A survey and taxonomy," *ACM Computing Surveys*, vol. 54, no. 2, pp. 1–38.

Wasserstein, L. N. (1969), "Markov processes over denumerable products of spaces describing large systems of automata," *Probl. Inform. Transmission*, vol. 5, pp. 47–52.

Yujia, L., K. Swersky, and R. Zemel (2015), "Generative moment matching networks," *Proc. International Conference on Machine Learning* (ICML), pp. 1718–1727, Lille, France.