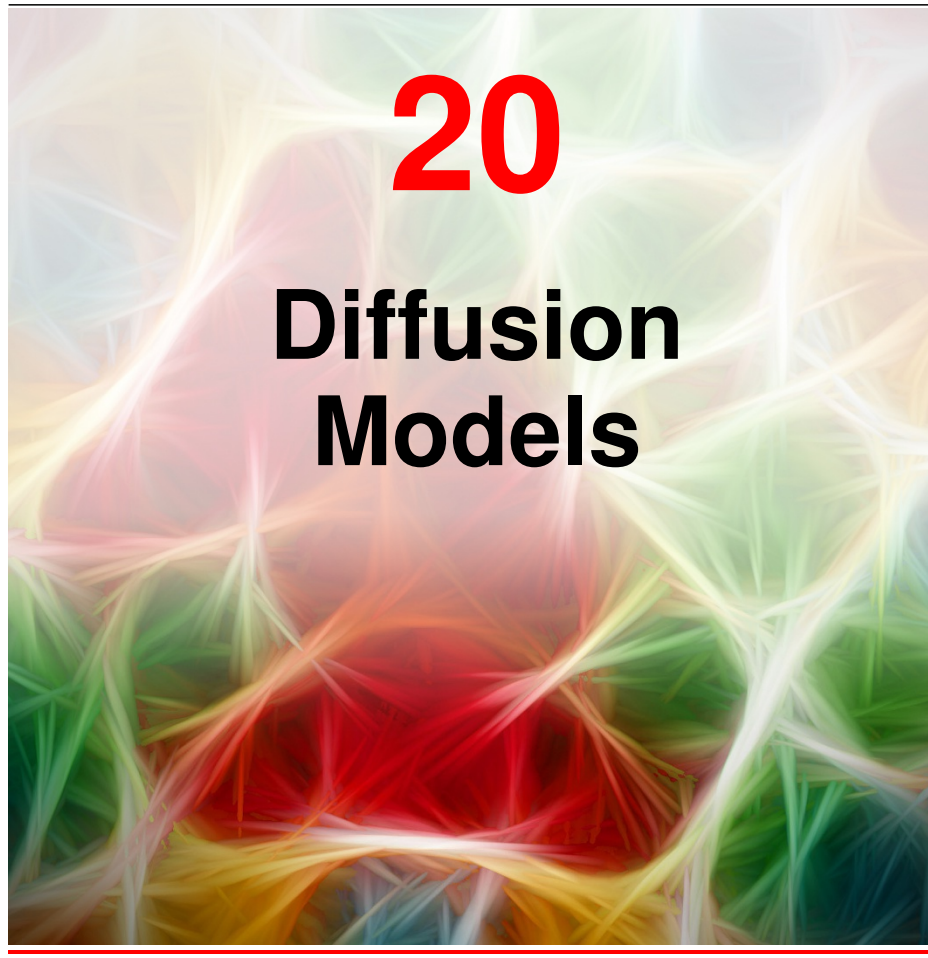

20 Diffusion Models

We have seen that a powerful way to construct rich generative models is to introduce a distribution $p(\mathbf{z})$ over a latent variable $\mathbf{z}$, and then to transform $\mathbf{z}$ into the data space $\mathbf{x}$ using a deep neural network. It is sufficient to use a simple, fixed distribution for $p(\mathbf{z})$, such as a Gaussian $\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$, since the generality of the neural network transforms this into a highly flexible family of distributions over $\mathbf{x}$. In previous chapters we have explored several models which fit within this framework but which take different approaches to defining and training the deep neural network, based on generative adversarial networks, variational autoencoders, and normalizing flows.

Section 16.4.4

In this chapter we discuss a fourth class of models within this general framework, known as *diffusion models*, also called *denoising diffusion probabilistic models*, or DDPMs (Sohl-Dickstein *et al.*, 2015; Ho, Jain, and Abbeel, 2020), which have emerged as the state of the art for many applications. For illustration we will focus on models of image data although the framework has much broader applicabil-

C. M. Bishop, H. Bishop, *Deep Learning*, https://doi.org/10.1007/978-3-031-45468-4_20

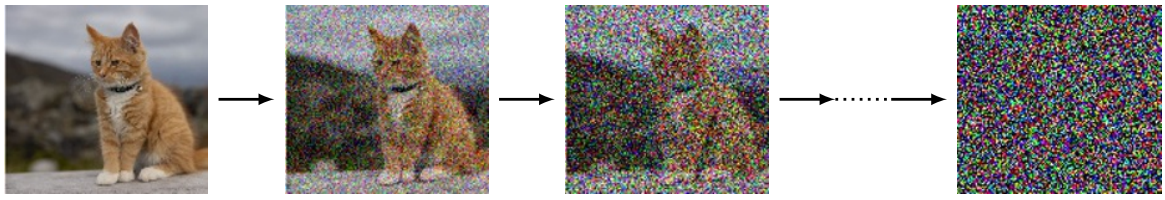

Figure 20.1 Illustration of the encoding process in a diffusion model showing an image $\mathbf{x}$ that is gradually corrupted with multiple stages of additive Gaussian noise giving a sequence of increasingly noisy images. After a large number T of steps the result is indistinguishable from a sample drawn from a Gaussian distribution. A deep neural network is then trained to reverse this process.

ity. The central idea is to take each training image and to corrupt it using a multi-step noise process to transform it into a sample from a Gaussian distribution. This is illustrated in Figure 20.1. A deep neural network is then trained to invert this process, and once trained the network can then generate new images starting with samples from a Gaussian as input.

Section 19.2

Diffusion models can be viewed as a form of hierarchical variational autoencoder in which the encoder distribution is fixed, and defined by the noise process, and only the generative distribution is learned (Luo, 2022). They are easy to train, they scale well on parallel hardware, and they avoid the challenges and instabilities of adversarial training while producing results that have quality comparable to, or better than, generative adversarial networks. However, generating new samples can be computationally expensive due to the need for multiple forward passes through the decoder network (Dhariwal and Nichol, 2021).

20.1. Forward Encoder

Suppose we take an image from the training set, which we will denote by $\mathbf{x}$, and blend it with Gaussian noise independently for each pixel to give a noise-corrupted image $\mathbf{z}_1$ defined by

$$\mathbf{z}_1 = \sqrt{1-\beta_1}\mathbf{x} + \sqrt{\beta_1}\boldsymbol{\epsilon}_1 \tag{20.1}$$

Exercise 20.1

where $\boldsymbol{\epsilon}_1 \sim \mathcal{N}(\boldsymbol{\epsilon}_1|\mathbf{0}, \mathbf{I})$ and $\beta_1 < 1$ is the variance of the noise distribution. The choice of coefficients $\sqrt{1-\beta_1}$ and $\sqrt{\beta_1}$ in (20.1) and (20.3) ensures that the mean of the distribution of $\mathbf{z}_t$ is closer to zero than the mean of $\mathbf{z}_{t-1}$ and that the variance of $\mathbf{z}_t$ is closer to the unit matrix than the variance of $\mathbf{z}_{t-1}$. We can write the transformation (20.1) in the form

Exercise 20.2

$$q(\mathbf{z}_1|\mathbf{x}) = \mathcal{N}(\mathbf{z}_1|\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I}). \tag{20.2}$$

We then repeat the process with additional independent Gaussian noise steps to give a sequence of increasingly noisy images $\mathbf{z}_2, \ldots, \mathbf{z}_T$. Note that in the literature on diffusion models, these latent variables are sometimes denoted $\mathbf{x}_1, \ldots, \mathbf{x}_T$ and the observed variable is denoted $\mathbf{x}_0$. We use the notation of $\mathbf{z}$ for latent variables and $\mathbf{x}$

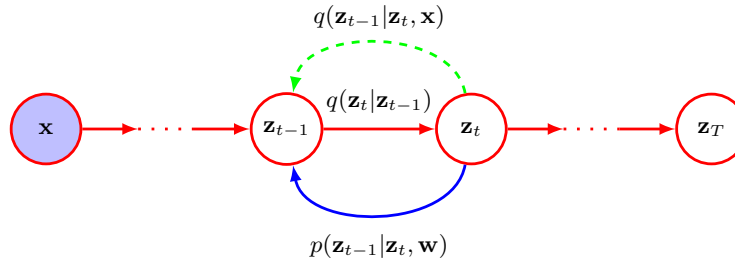

Figure 20.2 A diffusion process represented as a probabilistic graphical model. The original image $\mathbf{x}$ is shown by the shaded node, since it is an observed variable, whereas the noise-corrupted images $\mathbf{z}_1, \ldots, \mathbf{z}_T$ are considered to be latent variables. The noise process is defined by the forward distribution $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ and can be viewed as an encoder. Our goal is to learn a model $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})$ that tries to reverse this noise process and which can be viewed as a decoder. As we will see later, the conditional distribution $q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})$ plays an important role in defining the training procedure.

for the observed variable for consistency with the rest of the book. Each successive image is given by

$$\mathbf{z}_t = \sqrt{1-\beta_t}\mathbf{z}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t \tag{20.3}$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{\epsilon}_t|\mathbf{0}, \mathbf{I})$. Again, we can write (20.3) in the form

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t|\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}). \tag{20.4}$$

Section 11.3 The sequence of conditional distributions (20.4) forms a Markov chain and can be expressed as a probabilistic graphical model as shown in Figure 20.2. The values of the variance parameters $\beta_t \in (0, 1)$ are set by hand and are typically chosen such that the variance values increase through the chain according to a prescribed schedule such that $\beta_1 < \beta_2 < \ldots < \beta_T$.

20.1.1 Diffusion kernel

The joint distribution of the latent variables, conditioned on the observed data vector $\mathbf{x}$, is given by

$$q(\mathbf{z}_1, \ldots, \mathbf{z}_t|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x}) \prod_{\tau=2}^{t} q(\mathbf{z}_\tau|\mathbf{z}_{\tau-1}). \tag{20.5}$$

Exercise 20.3 If we now marginalize over the intermediate variables $\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}$, we obtain the *diffusion kernel*:

$$q(\mathbf{z}_t|\mathbf{x}) = \mathcal{N}(\mathbf{z}_t|\sqrt{\alpha_t}\mathbf{x}, (1-\alpha_t)\mathbf{I}) \tag{20.6}$$

where we have defined

$$\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau). \tag{20.7}$$

We see that each intermediate distribution has a simple closed-form Gaussian expression from which we can directly sample, which will prove useful when training DDPMs as it allows efficient stochastic gradient descent using randomly chosen intermediate terms in the Markov chain without having to run the whole chain. We can also write (20.6) in the form

$$\mathbf{z}_t = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t \tag{20.8}$$

where again $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{\epsilon}_t|\mathbf{0}, \mathbf{I})$. Note that that $\boldsymbol{\epsilon}$ now represents the total noise added to the original image instead of the incremental noise added at this step of the Markov chain.

After many steps the image becomes indistinguishable from Gaussian noise, and in the limit $T \to \infty$ we have

Exercise 20.4

$$q(\mathbf{z}_T|\mathbf{x}) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I}) \tag{20.9}$$

and therefore all information about the original image is lost. The choice of coefficients $\sqrt{1-\beta_t}$ and $\sqrt{\beta_t}$ in (20.3) ensures that once the Markov chain converges to a distribution with zero mean and unit covariance, further updates will leave this unchanged.

Exercise 20.5

Since the right-hand side of (20.9) is independent of $\mathbf{x}$, it follows that the marginal distribution of $\mathbf{z}_T$ is given by

$$q(\mathbf{z}_T) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I}). \tag{20.10}$$

It is common to refer to the Markov chain (20.4) as the *forward process*, and it is analogous to the encoder in a VAE, except that here it is fixed rather than learned. Note, however, that the usual terminology in the literature is the opposite of that typically used in the literature regarding normalizing flows, where the mapping from latent space to data space is considered the forward process.

20.1.2 Conditional distribution

Our goal is to learn to undo the noise process, and so it is natural to consider the reverse of the conditional distribution $q(\mathbf{z}_t|\mathbf{z}_{t-1})$, which we can express using Bayes' theorem in the form

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t) = \frac{q(\mathbf{z}_t|\mathbf{z}_{t-1})q(\mathbf{z}_{t-1})}{q(\mathbf{z}_t)}. \tag{20.11}$$

We can write the marginal distribution $q(\mathbf{z}_{t-1})$ in the form

$$q(\mathbf{z}_{t-1}) = \int q(\mathbf{z}_{t-1}|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x} \tag{20.12}$$

where $q(\mathbf{z}_{t-1}|\mathbf{x})$ is given by the conditional Gaussian (20.6). This distribution is intractable, however, because we must integrate over the unknown data density $p(\mathbf{x})$. If we approximate the integration using samples from the training data set, we obtain a complicated distribution expressed as a mixture of Gaussians.

Instead, we consider the conditional version of the reverse distribution, conditioned on the data vector $\mathbf{x}$, defined by $q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})$, which as we will see shortly turns out to be a simple Gaussian distribution. Intuitively this is reasonable since, given a noisy image, it is difficult to guess which lower-noise image gave rise to it, whereas if we also know the starting image then the problem becomes much easier. We can calculate this conditional distribution using Bayes' theorem:

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \frac{q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x})q(\mathbf{z}_{t-1}|\mathbf{x})}{q(\mathbf{z}_t|\mathbf{x})}. \tag{20.13}$$

We now make use of the Markov property of the forward process to write

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}) = q(\mathbf{z}_t|\mathbf{z}_{t-1}) \tag{20.14}$$

where the right-hand side is given by (20.4). As a function of $\mathbf{z}_{t-1}$, this takes the form of an exponential of a quadratic form. The term $q(\mathbf{z}_{t-1}|\mathbf{x})$ in the numerator of (20.13) is the diffusion kernel given by (20.6), which again involves the exponential of a quadratic form with respect to $\mathbf{z}_{t-1}$. We can ignore the denominator in (20.13) since as a function of $\mathbf{z}_{t-1}$ it is constant. Thus, we see that the right-hand side of (20.13) takes the form of a Gaussian distribution, and we can identify its mean and covariance using the technique of 'completing the square' to give

Exercise 20.6

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \mathcal{N}\left(\mathbf{z}_{t-1}|\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t), \sigma_t^2\mathbf{I}\right) \tag{20.15}$$

where

$$\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t) = \frac{(1-\alpha_{t-1})\sqrt{1-\beta_t}\mathbf{z}_t + \sqrt{\alpha_{t-1}}\beta_t\mathbf{x}}{1-\alpha_t} \tag{20.16}$$

$$\sigma_t^2 = \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t} \tag{20.17}$$

and we have made use of (20.7).

20.2. Reverse Decoder

We have seen that the forward encoder model is defined by a sequence of Gaussian conditional distributions $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ but that inverting this directly leads to a distribution $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$ that is intractable, as it would require integrating over all possible values of the starting vector $\mathbf{x}$ whose distribution is the unknown data distribution $p(\mathbf{x})$ that we wish to model. Instead, we will learn an approximation to the reverse distribution by using a distribution $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})$ governed by a deep neural network, where $\mathbf{w}$ represents the network weights and biases. This reverse step is analogous to the decoder in a variational autoencoder and is illustrated in Figure 20.2. Once the network is trained, we can sample from the simple Gaussian distribution over $\mathbf{z}_T$ and transform it into a sample from the data distribution $p(\mathbf{x})$ through a sequence of reverse sampling steps by repeated application of the trained network.

Chapter 19

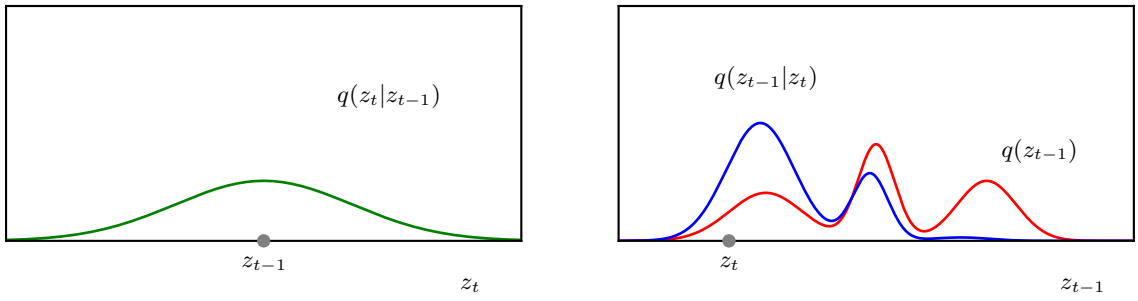

Figure 20.3 Illustration of the evaluation of the reverse distribution $q(z_{t-1}|z_t)$ using Bayes' theorem (20.13) for scalar variables. The red curve on the right-hand plot shows the marginal distribution $q(z_{t-1})$ illustrated using a mixture of three Gaussians, whereas the left-hand plot shows the Gaussian forward noise process $q(z_t|z_{t-1})$ as a distribution over z_t centred on z_{t-1}. By multiplying these together and normalizing, we obtain the distribution $q(z_{t-1}|z_t)$ shown for a particular choice of z_t by the blue curve. Because the distribution on the left is relatively broad, corresponding to a large variance β_t, the distribution $q(z_{t-1}|z_t)$ has a complex multimodal structure.

Intuitively, if we keep the variances small so that $\beta_t \ll 1$ then the change in the latent vector between steps will be relatively small, and hence it should be easier to learn to invert the transformation. More specifically, if $\beta_t \ll 1$ then the distribution $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$ will be approximately a Gaussian distribution over $\mathbf{z}_{t-1}$. This can be seen from (20.11) since the right-hand side depends on $\mathbf{z}_{t-1}$ through $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ and $q(\mathbf{z}_{t-1})$. If $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ is a sufficiently narrow Gaussian then $q(\mathbf{z}_{t-1})$ will vary only a small amount over the region in which $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ has significant mass, and hence $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$ will also be approximately Gaussian. This intuition can be confirmed using a simple example as shown in Figures 20.3 and 20.4. However, since the variances at each step are small, we must use a large number of steps to ensure that the distribution over the final latent variable $\mathbf{z}_T$ obtained from the forward noising process will still be close to a Gaussian, and this increases the cost of generating new samples. In practice, T may be several thousand.

We can see more formally that $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$ will be approximately Gaussian by

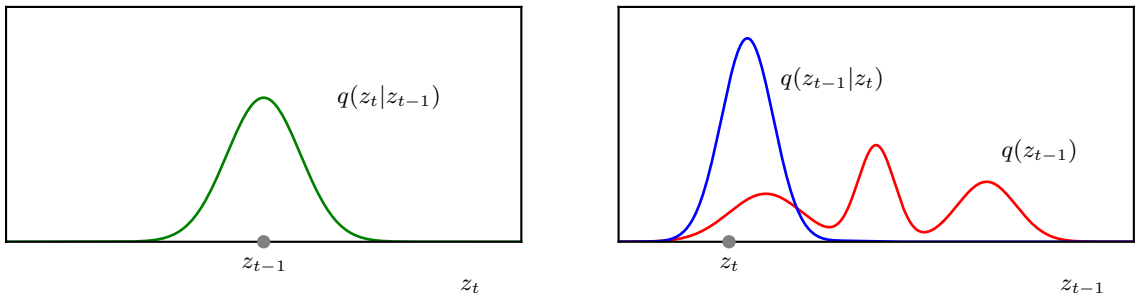

Figure 20.4 As in Figure 20.3 but in which the Gaussian distribution $q(z_t|z_{t-1})$ in the left-hand plot has a much smaller variance β_t. We see that the corresponding distribution $q(z_{t-1}|z_t)$ shown in blue on the right-hand plot is close to being Gaussian, with a similar variance to $q(z_t|z_{t-1})$.

making a Taylor series expansion of $\ln q(\mathbf{z}_{t-1}|\mathbf{z}_t)$ around the point $\mathbf{z}_t$ as a function of $\mathbf{z}_{t-1}$. This also shows that for small variance, the reverse distribution $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ will have a covariance that is close to the covariance $\beta_t\mathbf{I}$ of the forward noise process $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$. We therefore model the reverse process using a Gaussian distribution of the form

Exercise 20.7

$$p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}) = \mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t), \beta_t\mathbf{I}) \tag{20.18}$$

where $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)$ is a deep neural network governed by a set of parameters $\mathbf{w}$. Note that the network takes the step index t explicitly as an input so that it can account for the variation of the variance β_t across different steps of the chain. This allows us to use a single network to invert all the steps in the Markov chain, instead of having to learn a separate network for each step. It is also possible to learn the covariances of the denoising process by incorporating further outputs in the network to account for the curvature in the distribution $q(\mathbf{z}_{t-1})$ in the neighbourhood of $\mathbf{z}_t$ (Nichol and Dhariwal, 2021). There considerable flexibility in the choice of architecture for the neural network used to model $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)$ provided the output has the same dimensionality as the input. Given this restriction, a U-net architecture is a common choice for image processing applications.

Section 10.5.4

The overall reverse denoising process then takes the form of a Markov chain given by

$$p(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_T|\mathbf{w}) = p(\mathbf{z}_T)\left\{\prod_{t=2}^{T} p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})\right\} p(\mathbf{x}|\mathbf{z}_1, \mathbf{w}). \tag{20.19}$$

Here $p(\mathbf{z}_T)$ is assumed to be the same as the distribution of $q(\mathbf{z}_T)$ and hence is given by $\mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I})$. Once the model has been trained, sampling is straightforward because we first sample from the simple Gaussian $p(\mathbf{z}_T)$ and then we sample sequentially from each of the conditional distributions $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})$ in turn, finally sampling from $p(\mathbf{x}|\mathbf{z}_1, \mathbf{w})$ to obtain a sample $\mathbf{x}$ in the data space.

20.2.1 Training the decoder

We next have to decide on an objective function for training the neural network. The obvious choice is the likelihood function, which for data point $\mathbf{x}$ is given by

$$p(\mathbf{x}|\mathbf{w}) = \int \cdots \int p(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_T|\mathbf{w})\, \mathrm{d}\mathbf{z}_1 \ldots \mathrm{d}\mathbf{z}_T \tag{20.20}$$

in which $p(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_T|\mathbf{w})$ is defined by (20.19). This is an instance of the general latent-variable model (16.81) in which the latent variables comprise $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_T)$ and the observed variable is $\mathbf{x}$. Note that the latent variables all have the same dimensionality as the data space, as was the case for normalizing flows but not for variational autoencoders or generative adversarial networks. We see from (20.20) that the likelihood involves integrating over all possible trajectories by which noise samples could give rise to the observed data point. The integrals in (20.20) are intractable as they involve integrating over the highly complex neural network functions.

20.2.2 Evidence lower bound

Since the exact likelihood is intractable, we can adopt a similar approach to that used with variational autoencoders and maximize a lower bound on the log likelihood called the *evidence lower bound* (ELBO), which we re-derive here in the context of diffusion models. For any choice of distribution $q(\mathbf{z})$, the following relation always holds:

Section 16.3

$$\ln p(\mathbf{x}|\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \mathrm{KL}\left(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x},\mathbf{w})\right) \tag{20.21}$$

where $\mathcal{L}$ is the evidence lower bound, also known as the *variational lower bound*, given by

$$\mathcal{L}(\mathbf{w}) = \int q(\mathbf{z}) \ln \left\{ \frac{p(\mathbf{x},\mathbf{z}|\mathbf{w})}{q(\mathbf{z})} \right\} \mathrm{d}\mathbf{z} \tag{20.22}$$

and the Kullback–Leibler divergence $\mathrm{KL}\left(f\|g\right)$ between two probability densities $f(\mathbf{z})$ and $g(\mathbf{z})$ is defined by

Section 2.5.7

$$\mathrm{KL}\left(f(\mathbf{z})\|g(\mathbf{z})\right) = -\int f(\mathbf{z}) \ln \left\{ \frac{g(\mathbf{z})}{f(\mathbf{z})} \right\} \mathrm{d}\mathbf{z}. \tag{20.23}$$

To verify the relation (20.21) first note that, from the product rule of probability, we have

$$p(\mathbf{x},\mathbf{z}|\mathbf{w}) = p(\mathbf{z}|\mathbf{x},\mathbf{w})p(\mathbf{x}|\mathbf{w}). \tag{20.24}$$

Substituting (20.24) into (20.22) and making use of (20.23) gives (20.21). The Kullback–Leibler divergence has the property $\mathrm{KL}\left(\cdot\|\cdot\right) \geqslant 0$ from which it follows that

Exercise 20.8
Section 2.5.7

$$\ln p(\mathbf{x}|\mathbf{w}) \geqslant \mathcal{L}(\mathbf{w}). \tag{20.25}$$

Since the log likelihood function is intractable, we train the neural network by maximizing the lower bound $\mathcal{L}(\mathbf{w})$.

To do this, we first derive an explicit form for the lower bound of the diffusion model. In defining the lower bound we are free to choose any form we like for $q(\mathbf{z})$ as long as it is a valid probability distribution, i.e., that it is non-negative and integrates to 1. With many applications of the ELBO, such as the variational autoencoder, we chose a form for $q(\mathbf{z})$ that has adjustable parameters, often in the form of a deep neural network, and then we maximize the ELBO with respect to those parameters as well as with respect to the parameters of the distribution $p(\mathbf{x},\mathbf{z}|\mathbf{w})$. Optimizing the distribution $q(\mathbf{z})$ encourages the bound to be tight, which brings the optimization of the parameters in $p(\mathbf{x},\mathbf{z}|\mathbf{w})$ closer to that of maximum likelihood. With diffusion models, however, we chose $q(\mathbf{z})$ to be given by the *fixed* distribution $q(\mathbf{z}_1,\ldots,\mathbf{z}_T|\mathbf{x})$ defined by the Markov chain (20.5), and so the only adjustable parameters are those in the model $p(\mathbf{x},\mathbf{z}_1,\ldots,\mathbf{z}_T|\mathbf{w})$ for the reverse Markov chain. Note that we are using the flexibility in the choice of $q(\mathbf{z})$ to select a form that depends on $\mathbf{x}$.

We therefore substitute for $q(\mathbf{z}_1,\ldots,\mathbf{z}_T|\mathbf{x})$ in (20.21) using (20.5), and likewise we substitute for $p(\mathbf{x},\mathbf{z}_1,\ldots,\mathbf{z}_T|\mathbf{w})$ using (20.19), which allows us to write the

ELBO in the form

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_q\left[\ln \frac{p(\mathbf{z}_T)\left\{\prod_{t=2}^{T} p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})\right\} p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})}{q(\mathbf{z}_1|\mathbf{x})\prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x})}\right]$$

$$= \mathbb{E}_q\left[\ln p(\mathbf{z}_T) + \sum_{t=2}^{T} \ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})}{q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x})} - \ln q(\mathbf{z}_1|\mathbf{x}) + \ln p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})\right] \tag{20.26}$$

where we have defined

$$\mathbb{E}_q\left[\,\cdot\,\right] \equiv \int \cdots \int q(\mathbf{z}_1|\mathbf{x}) \prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1})\left[\,\cdot\,\right] \mathrm{d}\mathbf{z}_1 \ldots \mathrm{d}\mathbf{z}_T. \tag{20.27}$$

The first term $\ln p(\mathbf{z}_T)$ on the right-hand side of (20.26) is just the fixed distribution $\mathcal{N}(\mathbf{z}_T|\mathbf{0},\mathbf{I})$. This has no trainable parameters and can therefore be omitted from the ELBO since it represents a fixed additive constant. Similarly, the third term $-\ln q(\mathbf{z}_1|\mathbf{x})$ is independent of $\mathbf{w}$ and so again can be omitted.

The fourth term on the right-hand side of (20.26) corresponds to the reconstruction term from the variational autoencoder. It can be evaluated by approximating the expectation $\mathbb{E}_q\left[\,\cdot\,\right]$ by a Monte Carlo estimate obtained by drawing samples from the distribution over $\mathbf{z}_1$ defined by (20.2) so that

$$\mathbb{E}_q\left[\ln p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})\right] \simeq \sum_{l=1}^{L} \ln p(\mathbf{x}|\mathbf{z}_1^{(l)},\mathbf{w}) \tag{20.28}$$

where $\mathbf{z}_1^{(l)} \sim \mathcal{N}(\mathbf{z}_1|\sqrt{1-\beta_1}\mathbf{x},\beta_1\mathbf{I})$. Unlike with VAEs we do not need to back-propagate an error signal through the sampled value because the q-distribution is fixed and so there is no need here for the reparameterization trick.

Section 19.2.2

This leaves the second term on the right-hand side of (20.26), which comprises a sum of terms each of which is dependent on a pair of adjacent latent-variable values $\mathbf{z}_{t-1}$ and $\mathbf{z}_t$. We saw earlier when we derived the diffusion kernel (20.6) that we can sample from $q(\mathbf{z}_{t-1}|\mathbf{x})$ directly as a Gaussian distribution and we could then obtain a corresponding sample of $\mathbf{z}_t$ using (20.4), which is also a Gaussian. Although this would be a correct procedure in the limit of an infinite number of samples, the use of pairs of sampled values creates very noisy estimates with high variance, so that an unnecessarily large numbers of samples is required. Instead, we rewrite the ELBO in a form that can be estimated by sampling just one value per term.

20.2.3 Rewriting the ELBO

Following our discussion of the ELBO for the variational autoencoder, our goal here is to write the ELBO in terms of Kullback–Leibler divergences, which we can then subsequently express in closed form. The neural network is a model of the distribution in the reverse direction $p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})$ whereas the q-distribution is expressed in the forward direction $q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x})$, and so we use Bayes' theorem to

reverse the conditional distribution by writing

$$q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x}) = \frac{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})q(\mathbf{z}_t|\mathbf{x})}{q(\mathbf{z}_{t-1}|\mathbf{x})}. \tag{20.29}$$

This allows us to write the second term in (20.26) in the form

$$\ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})}{q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x})} = \ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})}{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})} + \ln \frac{q(\mathbf{z}_{t-1}|\mathbf{x})}{q(\mathbf{z}_t|\mathbf{x})}. \tag{20.30}$$

The second term on the right-hand side of (20.30) is independent of $\mathbf{w}$ and so can be omitted. Substituting (20.30) into (20.26), we then obtain

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_q\left[\sum_{t=2}^{T} \ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})}{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})} + \ln p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})\right]. \tag{20.31}$$

Exercise 20.9

Finally, we can rewrite (20.31) in the form

$$\mathcal{L}(\mathbf{w}) = \underbrace{\int q(\mathbf{z}_1|\mathbf{x}) \ln p(\mathbf{x}|\mathbf{z}_1,\mathbf{w})\,\mathrm{d}\mathbf{z}_1}_{\text{reconstruction term}} - \underbrace{\sum_{t=2}^{T} \int \mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\|p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w}))q(\mathbf{z}_t|\mathbf{x})\,\mathrm{d}\mathbf{z}_t}_{\text{consistency terms}} \tag{20.32}$$

where we have simplified the expectation over $q(\mathbf{z}_1,\ldots,\mathbf{z}_T|\mathbf{x})$ in the first term since $\mathbf{z}_1$ is the only latent variable appearing in the integrand. Therefore in the expectation defined by (20.27), all the conditional distributions integrate to unity leaving only the integral over $\mathbf{z}_1$. Likewise, in the second term, each integral involves only two adjacent latent variables $\mathbf{z}_{t-1}$ and $\mathbf{z}_t$, and all remaining variables can be integrated out.

The bound (20.32) is now very similar to the ELBO for the variational autoencoder given by (19.14), except that there are now multiple encoder and decoder stages. The reconstruction term rewards high probability for the observed data sample and can be trained in the same way as the corresponding term in the VAE by using the sampling approximation (20.28). The consistency terms in (20.32) are defined between pairs of Gaussian distributions and therefore can be expressed in closed form, as follows. The distribution $q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})$ is given by (20.15) whereas the distribution $p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})$ is given by (20.18) and so the Kullback–Leibler divergence becomes

Chapter 19

Exercise 20.11

$$\begin{aligned}&\mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\|p(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{w})) \\ &\quad = \frac{1}{2\beta_t}\left\|\mathbf{m}_t(\mathbf{x},\mathbf{z}_t) - \boldsymbol{\mu}(\mathbf{z}_t,\mathbf{w},t)\right\|^2 + \text{const}\end{aligned} \tag{20.33}$$

where $\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t)$ is defined by (20.16) and where any additive terms that are independent of the network parameters $\mathbf{w}$ have been absorbed into the constant term, which plays no role in training. Each of the consistency terms in (20.32) has one remaining integral over $\mathbf{z}_t$, weighted by $q(\mathbf{z}_t|\mathbf{x})$. This can be approximated by drawing a sample from $q(\mathbf{z}_t|\mathbf{x})$, which can be done efficiently using the diffusion kernel (20.6).

We see that the KL divergence (20.33) takes the form of a simple squared-loss function. Since we adjust the network parameters to maximize the lower bound in (20.32), we will be minimizing this squared error because there is a minus sign in front of the Kullback–Leibler divergence terms in the ELBO.

20.2.4 Predicting the noise

One modification that leads to higher quality results is to change the role of the neural network so that instead of predicting the denoised image at each step of the Markov chain it predicts the total noise component that was added to the original image to create the noisy image at that step (Ho, Jain, and Abbeel, 2020). To do this we first take (20.8) and rearrange to give

$$\mathbf{x} = \frac{1}{\sqrt{\alpha_t}}\mathbf{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\boldsymbol{\epsilon}_t. \tag{20.34}$$

If we now substitute this into (20.16) we can rewrite the mean $\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t)$ of the reverse conditional distribution $q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})$ in terms of the original data vector $\mathbf{x}$ and the noise $\boldsymbol{\epsilon}$ to give

Exercise 20.12

$$\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}_t\right\}. \tag{20.35}$$

Similarly, instead of a neural network $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)$ that predicts the denoised image, we introduce a neural network $\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)$ that aims to predict the total noise that was added to $\mathbf{x}$ to generate $\mathbf{z}_t$. Following the same steps that led to (20.35) shows that these two network functions are related by

$$\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}. \tag{20.36}$$

We can now substitute (20.35) and (20.36) into (20.33) to give

$$\begin{aligned}
&\mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})\|p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})) \\
&= \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}\left\|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}_t\right\|^2 + \text{const} \\
&= \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}\left\|\mathbf{g}(\sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}_t\right\|^2 + \text{const}
\end{aligned} \tag{20.37}$$

where in the final line we have substituted for $\mathbf{z}_t$ using (20.8).

The reconstruction term in the ELBO (20.32) can be approximated using (20.28) with a sampled value of $\mathbf{z}_1$. Using the form (20.18) for $p(\mathbf{x}|\mathbf{z}, \mathbf{w})$ we have

$$\ln p(\mathbf{x}|\mathbf{z}_1, \mathbf{w}) = -\frac{1}{2\beta_1}\|\mathbf{x} - \boldsymbol{\mu}(\mathbf{z}_1, \mathbf{w}, 1)\|^2 + \text{const.} \tag{20.38}$$

If we substitute for $\boldsymbol{\mu}(\mathbf{z}_1, \mathbf{w}, 1)$ using (20.36) and we substitute for $\mathbf{x}$ using (20.1) and then make use of $\alpha_1 = (1 - \beta_1)$, which follows from (20.7), we obtain

Exercise 20.13

$$\ln p(\mathbf{x}|\mathbf{z}_1, \mathbf{w}) = -\frac{1}{2(1-\beta_t)}\|\mathbf{g}(\mathbf{z}_1, \mathbf{w}, 1) - \boldsymbol{\epsilon}_1\|^2 + \text{const.} \tag{20.39}$$

This is precisely the same form as (20.37) for the special case $t = 1$, and so the reconstruction and consistency terms can be combined.

Ho, Jain, and Abbeel (2020) found empirically that performance is further improved simply by omitting the factor $\beta_t/2(1 - \alpha_t)(1 - \beta_t)$ in front of (20.37), so that all steps in the Markov chain have equal weighting. Substituting this simplified version of (20.37) into (20.33) gives a training objective function in the form

$$\mathcal{L}(\mathbf{w}) = -\sum_{t=1}^{T} \left\|\mathbf{g}(\sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}_t\right\|^2. \tag{20.40}$$

The squared error on the right-hand side of (20.40) has a very simple interpretation: for a given step t in the Markov chain and for a given training data point $\mathbf{x}$, we sample a noise vector $\boldsymbol{\epsilon}_t$ and use this to create the corresponding noisy latent vector $\mathbf{z}_t$ for that step. The loss function is then the squared difference between the predicted noise and the actual noise. Note that the network $\mathbf{g}(\cdot, \cdot, \cdot)$ is predicting the total noise added to the original data vector $\mathbf{x}$, not just the incremental noise added in step t.

When we use stochastic gradient descent, we evaluate the gradient vector of the loss function with respect to the network parameters for a randomly selected data point $\mathbf{x}$ from the training set. Also, for each such data point we randomly select a step t along the Markov chain, rather than evaluate the error for every term in the summation over t in (20.40). These gradients are accumulated over mini-batches of data samples and then used to update the weights.

Also note that this loss function automatically builds in a form of data augmentation, because every time a particular training sample $\mathbf{x}$ is used it is combined with a fresh sample $\boldsymbol{\epsilon}_t$ of noise. All the above relates to a single data point $\mathbf{x}$ from the training set. The corresponding computation of the gradient is shown in Algorithm 20.1.

20.2.5 Generating new samples

Once the network has been trained we can generate new samples in the data space by first sampling from the Gaussian distribution $p(\mathbf{z}_T)$ and then denoising successively through each step of the Markov chain. Given a denoised sample $\mathbf{z}_t$ at step t, we generate a sample $\mathbf{z}_{t-1}$ in three steps. First we evaluate the output of the neural network given by $\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)$. From this we evaluate $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)$ using (20.36).

Algorithm 20.1: Training a denoising diffusion probabilistic model

Input: Training data $\mathcal{D} = \{\mathbf{x}_n\}$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Network parameters $\mathbf{w}$

---

for $t \in \{1, \ldots, T\}$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ `// Calculate alphas from betas`
end for
repeat
  $\mathbf{x} \sim \mathcal{D}$ `// Sample a data point`
  $t \sim \{1, \ldots, T\}$ `// Sample a point along the Markov chain`
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`
  $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$ `// Evaluate noisy latent variable`
  $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}\|^2$ `// Compute loss term`
  Take optimizer step
until converged
return $\mathbf{w}$

Finally we generate a sample $\mathbf{z}_{t-1}$ from $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}) = \mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t), \beta_t\mathbf{I})$ by adding noise scaled by the variance so that

$$\mathbf{z}_{t-1} = \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon} \tag{20.41}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$. Note that the network $\mathbf{g}(\cdot, \cdot, \cdot)$ predicts the total noise added to the original data vector $\mathbf{x}$ to obtain $\mathbf{z}_t$, but in the sampling step, we subtract off only a fraction $\beta_t/\sqrt{1-\alpha_t}$ of this noise from $\mathbf{z}_{t-1}$ and then add additional noise with variance β_t to generate $\mathbf{z}_{t-1}$. At the final step when we calculate a synthetic data sample $\mathbf{x}$, we do not add additional noise since we are aiming to generate a noise-free output. The sampling procedure is summarized in Algorithm 20.2.

The main drawback of diffusion models for generating data is that they require multiple sequential inference passes through the trained network, which can be computationally expensive. One way to speed up the sampling process is first to convert the denoising process to a differential equation over continuous time and then to use alternative efficient discretization methods to solve the equation efficiently.

Section 20.3.4

We have assumed in this chapter that the data and latent variables are continuous and that we can therefore use Gaussian noise models. Diffusion models can also be defined for discrete spaces (Austin *et al.*, 2021), for example, to generate new candidate drug molecules in which part of the generation process involves choosing atom types from a subset of chemical elements.

We have seen that diffusion models can be computationally intensive because

Algorithm 20.2: Sampling from a denoising diffusion probabilistic model

Input: Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Sample vector $\mathbf{x}$ in data space

$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ `// Sample from final latent space`
for $t \in T, \ldots, 2$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ `// Calculate alpha`
  `// Evaluate network output`
  $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`
  $\mathbf{z}_{t-1} \leftarrow \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$ `// Add scaled noise`
end for
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\left\{\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\mathbf{g}(\mathbf{z}_1, \mathbf{w}, t)\right\}$ `// Final denoising step`
return $\mathbf{x}$

they sequentially reverse a noise process that can have hundreds or thousands of steps. Song, Meng, and Ermon (2020) introduced a related technique called *denoising diffusion implicit models* that relax the Markovian assumption on the noise process while retaining the same objective function for training. This thereby allows one or two orders of magnitude speed-up during sampling without degrading the quality of the generated samples.

20.3. Score Matching

The denoising diffusion models discussed so far in this chapter are closely related to another class of deep generative models that were developed relatively independently and which are based on *score matching* (Hyvärinen, 2005; Song and Ermon, 2019). These make use of the *score function* or *Stein score*, which is defined as the gradient of the log likelihood with respect to the data vector $\mathbf{x}$ and is given by

$$\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \ln p(\mathbf{x}). \tag{20.42}$$

Here it is important to emphasize that the gradient is with respect to the data vector, not with respect to any parameter vector. Note that $\mathbf{s}(\mathbf{x})$ is a vector-valued function of the same dimensionality as $\mathbf{x}$ and that each element $s_i(\mathbf{x}) = \partial \ln p(\mathbf{x})/\partial x_i$ is associated with a corresponding element x_i of $\mathbf{x}$. For example, if $\mathbf{x}$ is an image then $\mathbf{s}(\mathbf{x})$ can also be represented as an image of the same dimensions with corresponding

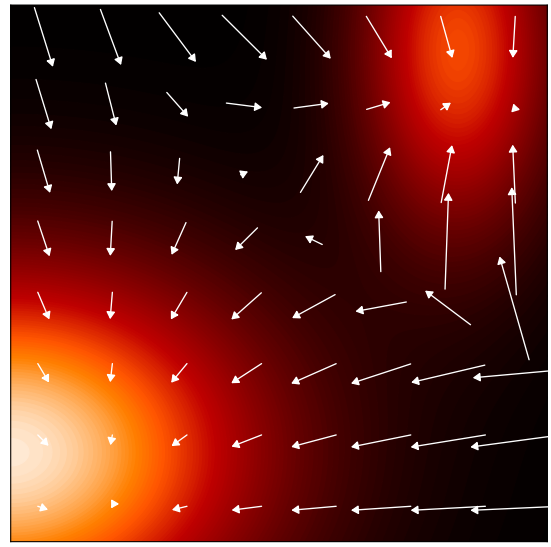

Figure 20.5 Illustration of the score function, showing a distribution in two dimensions comprising a mixture of Gaussians represented as a heat map and the corresponding score function defined by (20.42) plotted as vectors on a regular grid of $\mathbf{x}$-values.

pixels. Figure 20.5 shows an example of a probability density in two dimensions, along with the corresponding score function.

To see why the score function is useful, consider two functions $q(\mathbf{x})$ and $p(\mathbf{x})$ that have the property that their scores are equal, so that $\nabla_{\mathbf{x}} \ln q(\mathbf{x}) = \nabla_{\mathbf{x}} \ln p(\mathbf{x})$ for all values of $\mathbf{x}$. If we integrate both sides of the equation with respect to $\mathbf{x}$ and take exponentials, we obtain $q(\mathbf{x}) = Kp(\mathbf{x})$ where K is a constant independent of $\mathbf{x}$. So if we are able to learn a model $\mathbf{s}(\mathbf{x}, \mathbf{w})$ of the score function then we have modelled the original data density, up to a multiplicative constant.

20.3.1 Score loss function

To train such a model we need to define a loss function that aims to match the model score function $\mathbf{s}(\mathbf{x}, \mathbf{w})$ to the score function $\nabla_{\mathbf{x}} \ln p(\mathbf{x})$ of the distribution $p(\mathbf{x})$ that generated the data. An example of such a loss function is the expected squared error between the model score and the true score, given by

$$J(\mathbf{w}) = \frac{1}{2} \int \|\mathbf{s}(\mathbf{x}, \mathbf{w}) - \nabla_{\mathbf{x}} \ln p(\mathbf{x})\|^2 \, p(\mathbf{x}) \, \mathrm{d}\mathbf{x}. \tag{20.43}$$

Section 14.3.1 As we saw in the discussion of energy-based models, the score function does not require the associated probability density to be normalized, because the normalization constant is removed by the gradient operator, and so there is considerable flexibility in the choice of model. There are broadly two ways to represent the score function $\mathbf{s}(\mathbf{x}, \mathbf{w})$ using a deep neural network. Each element s_i of $\mathbf{s}$ corresponds to one of the elements x_i of $\mathbf{x}$, so the first approach is to have a network with the same number of outputs as inputs. However, the score function is defined to be the gradient of a scalar function (the log probability density), which is a more restricted class of functions. So an alternative approach is to have a network with a single output $\phi(\mathbf{x})$

Exercise 20.14

and then to compute $\nabla_{\mathbf{x}}\phi(\mathbf{x})$ using automatic differentiation. This second approach, however, requires two backpropagation steps and is therefore computationally more expensive. For this reason, most applications simply adopt the first approach.

Exercise 20.15

20.3.2 Modified score loss

One problem with the loss function (20.43) is that we cannot minimize it directly because we do not know the true data score $\nabla_{\mathbf{x}} \ln p(\mathbf{x})$. All we have is the finite data set $\mathcal{D} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ from which we can construct an empirical distribution:

$$p_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} \delta(\mathbf{x} - \mathbf{x}_n). \tag{20.44}$$

Here $\delta(\mathbf{x})$ is the Dirac delta function, which can be thought of informally as an infinitely tall 'spike' at $\mathbf{x} = \mathbf{0}$ with the properties

$$\delta(\mathbf{x}) = 0, \quad \mathbf{x} \neq \mathbf{0} \tag{20.45}$$

$$\int \delta(\mathbf{x})\, \mathrm{d}\mathbf{x} = 1. \tag{20.46}$$

Since (20.44) is not a differentiable function of $\mathbf{x}$, we cannot compute its score function. We can address this by introducing a noise model to 'smear out' the data points and give a smooth, differentiable representation of the density. This is known as a *Parzen estimator* or *kernel density estimator* and is defined by

Section 3.5.2

$$q_\sigma(\mathbf{z}) = \int q(\mathbf{z}|\mathbf{x}, \sigma) p(\mathbf{x})\, \mathrm{d}\mathbf{x} \tag{20.47}$$

where $q(\mathbf{z}|\mathbf{x}, \sigma)$ is the *noise kernel*. A common choice of kernel is the Gaussian

$$q(\mathbf{z}|\mathbf{x}, \sigma) = \mathcal{N}(\mathbf{z}|\mathbf{x}, \sigma^2\mathbf{I}). \tag{20.48}$$

Instead of minimizing the loss function (20.43), we then use the corresponding loss with respect to the smoothed Parzen density in the form

$$J(\mathbf{w}) = \frac{1}{2}\int \left\|\mathbf{s}(\mathbf{z}, \mathbf{w}) - \nabla_{\mathbf{z}} \ln q_\sigma(\mathbf{z})\right\|^2 q_\sigma(\mathbf{z})\, \mathrm{d}\mathbf{z}. \tag{20.49}$$

A key result is that by substituting (20.47) into (20.49) we can rewrite this loss function in an equivalent form given by (Vincent, 2011)

Exercise 20.17

$$J(\mathbf{w}) = \frac{1}{2}\iint \left\|\mathbf{s}(\mathbf{z}, \mathbf{w}) - \nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x}, \sigma)\right\|^2 q(\mathbf{z}|\mathbf{x}, \sigma) p(\mathbf{x})\, \mathrm{d}\mathbf{z}\, \mathrm{d}\mathbf{x} + \text{const}. \tag{20.50}$$

If we substitute for $p(\mathbf{x})$ using the empirical density (20.44), we obtain

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\int \left\|\mathbf{s}(\mathbf{z}, \mathbf{w}) - \nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x}_n, \sigma)\right\|^2 q(\mathbf{z}|\mathbf{x}_n, \sigma)\, \mathrm{d}\mathbf{z} + \text{const}. \tag{20.51}$$

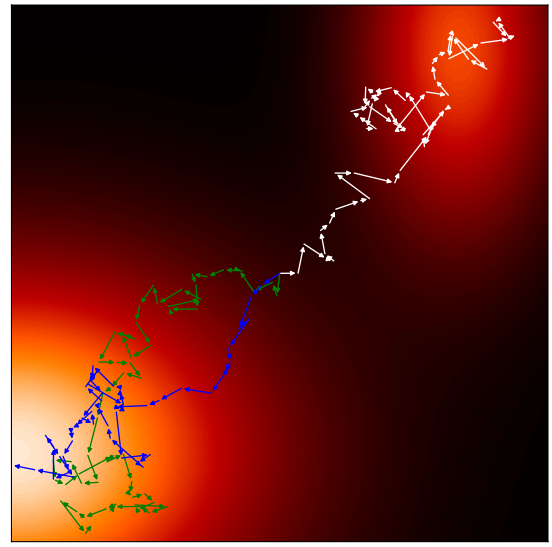

Figure 20.6 Examples of sampling trajectories obtained using Langevin dynamics defined by (14.61) for the distribution shown in Figure 20.5, showing three trajectories all starting at the centre of the plot.

For the Gaussian Parzen kernel (20.48), the score function becomes

$$\nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x}, \sigma) = -\frac{1}{\sigma}\boldsymbol{\epsilon} \tag{20.52}$$

where $\boldsymbol{\epsilon} = \mathbf{z} - \mathbf{x}$ is drawn from $\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$. If we consider the specific noise model (20.6) then we obtain

$$\nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x}, \sigma) = -\frac{1}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}. \tag{20.53}$$

We therefore see that the score loss (20.50) measures the difference between the neural network prediction and the noise $\boldsymbol{\epsilon}$. Therefore, this loss function has the same minimum as the form (20.37) used in the denoising diffusion model, with the score function $\mathbf{s}(\mathbf{z}, \mathbf{w})$ playing the same role as the noise prediction network $\mathbf{g}(\mathbf{z}, \mathbf{w})$ up to a constant scaling $-1/\sqrt{1-\alpha_t}$ (Song and Ermon, 2019). Minimizing (20.50) is known as *denoising score matching*, and we see the close connection to denoising diffusion models. There remains the question of how to choose the noise variance σ^2, and we will return to this shortly.

Having trained a score-based model we then need to draw new samples. Langevin dynamics is well-suited to score-based models because it is based on the score function and therefore does not require a normalized probability distribution, and is illustrated in Figure 20.6.

Section 14.3

20.3.3 Noise variance

We have seen how to learn the score function from a set of training data and how to generate new samples from the learned distribution using Langevin sampling. However, we can identify three potential problems with this approach (Song and

Ermon, 2019; Luo, 2022). First, if the data distribution lies on a manifold of lower dimensionality than the data space, the probability density will be zero at points off the manifold and here the score function is undefined since $\ln p(\mathbf{x})$ is undefined. Second, in regions of low data density, the estimate of the score function may be inaccurate since the loss function (20.43) is weighted by the density. An inaccurate score function can lead to poor trajectories when using Langevin sampling. Third, even with an accurate model of the score function, the Langevin procedure may not sample correctly if the data distribution comprises a mixture of disjoint distributions.

Chapter 16

Exercise 20.18

All three problems can be addressed by choosing a sufficiently large value for the noise variance σ^2 used in the kernel function (20.48), because this smears out the data distribution. However, too large a variance will introduce a significant distortion of the original distribution and this itself introduces inaccuracies in the modelling of the score function. This trade-off can be addressed by considering a sequence of variance values $\sigma_1^2 < \sigma_2^2 < \ldots < \sigma_T^2$ (Song and Ermon, 2019), in which σ_1^2 is sufficiently small that the data distribution is accurately represented whereas σ_T^2 is sufficiently large that the aforementioned problems are avoided. The score network is then modified to take the variance as an additional input $\mathbf{s}(\mathbf{x}, \mathbf{w}, \sigma^2)$ and is trained by using a loss function that is a weighted sum of the loss functions of the form (20.51) in which each term represents the error between the associated network and the corresponding perturbed data set. For a data vector $\mathbf{x}_n$, the loss function then takes the form

$$\frac{1}{2}\sum_{i=1}^{L}\lambda(i)\int \left\|\mathbf{s}(\mathbf{z}, \mathbf{w}, \sigma_i^2) - \nabla_{\mathbf{z}} \ln q(\mathbf{z}|\mathbf{x}_n, \sigma_i)\right\|^2 q(\mathbf{z}|\mathbf{x}_n, \sigma_i)\,\mathrm{d}\mathbf{z} \tag{20.54}$$

where $\lambda(i)$ are weighting coefficients. We see that this training procedure precisely mirrors that used to train hierarchical denoising networks.

Section 20.2.1

Once trained, samples can be generated by running a few steps of Langevin sampling from each of the models for $i = L, L-1, \ldots, 2, 1$ in turn. This technique is called *annealed Langevin dynamics*, and is analogous to Algorithm 20.2 used to sample from denoising diffusion models.

20.3.4 Stochastic differential equations

We have seen that it is helpful to use a large number of steps, often several thousand, when constructing the noise process for a diffusion model. It is therefore natural to ask what happens if we consider the limit of an infinite number of steps, much as we did for infinitely deep neural networks when we introduced neural differential equations. In taking such a limit, we need to ensure that the noise variance β_t at each step becomes smaller in keeping with the step size. This leads to a formulation of diffusion models for continuous time as *stochastic differential equations* or SDEs (Song *et al.*, 2020). Both denoising diffusion probabilistic models and score matching models can then be viewed as a discretization of a continuous-time SDE.

Section 18.3.1

We can write a general SDE as an infinitesimal update to the vector $\mathbf{z}$ in the form

$$\mathrm{d}\mathbf{z} = \underbrace{\mathbf{f}(\mathbf{z}, t)\,\mathrm{d}t}_{\text{drift}} + \underbrace{g(t)\,\mathrm{d}\mathbf{v}}_{\text{diffusion}} \tag{20.55}$$

where the drift term is deterministic, as in an ODE, but the diffusion term is stochastic, for example given by infinitesimal Gaussian steps. Here the parameter t is often called 'time' by analogy with physical systems. The forward noise process (20.3) for a diffusion model can be written as an SDE of the form (20.55) by taking the continuous-time limit.

Exercise 20.19

For the SDE (20.55), there is a corresponding reverse SDE (Song *et al.*, 2020) given by

$$\mathrm{d}\mathbf{z} = \left\{\mathbf{f}(\mathbf{z}, t) - g^2(t)\nabla_{\mathbf{z}} \ln p(\mathbf{z})\right\} \mathrm{d}t + g(t)\,\mathrm{d}\mathbf{v} \tag{20.56}$$

where we recognize $\nabla_{\mathbf{z}} \ln p(\mathbf{z})$ as the score function. The SDE given by (20.55) is to be solved in reverse from $t = T$ to $t = 0$.

To solve an SDE numerically, we need to discretize the time variable. The simplest approach is to use fixed, equally spaced time steps, which is known as the Euler–Maruyama solver. For the reverse SDE, we then recover a form of the Langevin equation. However, more sophisticated solvers can be employed that use more flexible forms of discretization (Kloeden and Platen, 2013).

Section 14.3

For all diffusion processes governed by an SDE, there exists a corresponding deterministic process described by an ODE whose trajectories have the same marginal probability densities $p(\mathbf{z}|t)$ as the SDE (Song *et al.*, 2020). For an SDE of the form (20.56), the corresponding ODE is given by

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{f}(\mathbf{z}, t) - \frac{1}{2} g^2(t)\nabla_{\mathbf{z}} \ln p(\mathbf{z}). \tag{20.57}$$

The ODE formulation allows the use of efficient adaptive-step solvers to reduce the number of function evaluations dramatically. Moreover, it allows probabilistic diffusion models to be related to normalizing flow models, from which the change-of-variables formula (18.1) can be used to provide an exact evaluation of the log likelihood.

Chapter 18

20.4. Guided Diffusion

So far, we have considered diffusion models as a way to represent the unconditional density $p(\mathbf{x})$ learned from a set of training examples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ drawn independently from $p(\mathbf{x})$. Once the model has been trained, we can generate new samples from this distribution. We have already seen an example of unconditional sampling from a deep generative model for face images in Figure 1.3, in that case from a GAN model.

In many applications, however, we want to sample from a conditional distribution $p(\mathbf{x}|\mathbf{c})$ where the conditioning variable $\mathbf{c}$ could, for example, be a class label or a textual description of the desired content for an image. This also forms the basis for applications such as image super-resolution, image inpainting, video generation, and many others. The simplest approach to achieving this would be to treat $\mathbf{c}$ as an additional input into the denoising neural network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t, \mathbf{c})$ and then to train the network using matched pairs $\{\mathbf{x}_n, \mathbf{c}_n\}$. The main limitation of this approach is that

the network can give insufficient weight to, or even ignore, the conditioning variables, so we need a way to control how much weight is given to the conditioning information and to trade this off against sample diversity. This additional pressure to match the conditioning information is called *guidance*. There are two main approaches to guidance depending on whether or not a separate classifier model is used.

20.4.1 Classifier guidance

Suppose that a trained classifier $p(\mathbf{c}|\mathbf{x})$ is available, and consider a diffusion model from the perspective of the score function. Using Bayes' theorem we can write the score function for the conditional diffusion model in the form

$$\begin{aligned}\nabla_{\mathbf{x}} \ln p(\mathbf{x}|\mathbf{c}) &= \nabla_{\mathbf{x}} \ln \left\{ \frac{p(\mathbf{c}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{c})} \right\} \\ &= \nabla_{\mathbf{x}} \ln p(\mathbf{x}) + \nabla_{\mathbf{x}} \ln p(\mathbf{c}|\mathbf{x}) \end{aligned} \tag{20.58}$$

where we have used $\nabla_{\mathbf{x}} \ln p(\mathbf{c}) = 0$ since $p(\mathbf{c})$ is independent of $\mathbf{x}$. The first term on the right-hand side of (20.58) is the usual unconditional score function, whereas the second term pushes the denoising process towards the direction that maximizes the probability of the given label $\mathbf{c}$ under the classifier model (Dhariwal and Nichol, 2021). The influence of the classifier can be controlled by introducing a hyperparameter λ, called the *guidance scale*, which controls the weight given to the classifier gradient. The score function used for sampling then becomes

$$\text{score}(\mathbf{x}, \mathbf{c}, \lambda) = \nabla_{\mathbf{x}} \ln p(\mathbf{x}) + \lambda \nabla_{\mathbf{x}} \ln p(\mathbf{c}|\mathbf{x}). \tag{20.59}$$

If $\lambda = 0$ we recover the original unconditional diffusion model, whereas if $\lambda = 1$ we obtain the score corresponding to the conditional distribution $p(\mathbf{x}|\mathbf{c})$. For $\lambda > 1$ the model is strongly encouraged to respect the conditioning label, and values of $\lambda \gg 1$ may be used, for example $\lambda = 10$. However, this comes at the expense of diversity in the samples as the model prefers 'easy' examples that the classifier is able to classify correctly.

One problem with the classifier-based approach to guidance is that a separate classifier must be trained. Furthermore, this classifier needs to be able to classify examples with varying degrees of noise, whereas standard classifiers are trained on clean examples. We therefore turn to an alternative approach that avoids the use of a separate classifier.

20.4.2 Classifier-free guidance

If we use (20.58) to replace $\nabla_{\mathbf{x}} \ln p(\mathbf{c}|\mathbf{x})$ in (20.59), we can write the score function in the form

Exercise 20.20

$$\text{score}(\mathbf{x}, \mathbf{c}, \lambda) = \lambda \nabla_{\mathbf{x}} \ln p(\mathbf{x}|\mathbf{c}) + (1 - \lambda) \nabla_{\mathbf{x}} \ln p(\mathbf{x}), \tag{20.60}$$

which for $0 < \lambda < 1$ represents a convex combination of the conditional log density $\ln p(\mathbf{x}|\mathbf{c})$ and the unconditional log density $\ln p(\mathbf{x})$. For $\lambda > 1$ the contribution from

the unconditional score becomes negative, meaning the model actively reduces the probability of generating samples that ignore the conditioning information in favour of samples that do.

Furthermore, we can avoid training separate networks to model $p(\mathbf{x}|\mathbf{c})$ and $p(\mathbf{x})$ by training a single conditional model in which the conditioning variable $\mathbf{c}$ is set to a null value, for example $\mathbf{c} = \mathbf{0}$, with some probability during training, typically around 10–20%. Then $p(\mathbf{x})$ is represented by $p(\mathbf{x}|\mathbf{c} = \mathbf{0})$. This is somewhat analogous to dropout in which the conditioning inputs are collectively set to zero for a random subset of training vectors.

Section 9.6.1

Once trained, the score function (20.60) is then used to encourage a strong weighting of the conditional information. In practice, classifier-free guidance gives much higher quality results than classifier guidance (Nichol *et al.*, 2021; Saharia *et al.*, 2022). The reason is that a classifier $p(\mathbf{c}|\mathbf{x})$ can ignore most of the input vector $\mathbf{x}$ as long as it makes a good prediction of $\mathbf{c}$ whereas classifier-free guidance is based on the conditional density $p(\mathbf{x}|\mathbf{c})$, which must assign a high probability to all aspects of $\mathbf{x}$.

Text-guided diffusion models can leverage techniques from large language models to allow the conditioning input to be a general text sequence, known as a *prompt*, and not simply a selection from a predefined set of class labels. This allows the text input to influence the denoising process in two ways, first by concatenating the internal representation from a transformer-based language model with the input to the denoising network and second by allowing cross-attention layers within the denoising network to attend to the text token sequence. Classifier-free guidance, conditioned on a text prompt, is illustrated in Figure 20.7.

Chapter 12

Another application for conditional diffusion models is image super-resolution in which a low-resolution image is transformed into a corresponding high-resolution image. This is intrinsically an inverse problem, and multiple high-resolution images will be consistent with a given low-resolution image. Super-resolution can be achieved by denoising a high-resolution sample from a Gaussian using the low-resolution image as a conditioning variable (Saharia, Ho, *et al.*, 2021). Examples of this method are shown in Figure 20.8. Such models can be cascaded to achieve very high resolution (Ho *et al.*, 2021), for example going from 64×64 to 256×256, and then from 256×256 to 1024×1024. Each stage is typically represented by a U-net architecture, with each U-net conditioned on the final denoised output of the previous one.

Section 10.5.4

This type of cascade can also be used with image-generation diffusion models, in which the image denoising is performed at a lower resolution and the result is subsequently up-sampled using a separate network (which may also take a text prompt as input) to give a final high-resolution output (Nichol *et al.*, 2021; Saharia *et al.*, 2022). This can significantly reduce the computational cost compared to working directly in a high-dimensional space since the denoising process may involve hundreds of passes through the denoising network. Note that these approaches still work within the image space directly but at lower resolution.

A different approach to addressing the high computational cost of applying diffusion models directly in the space of high-resolution images is called *latent diffu-*

Figure 20.7 Illustration of classifier-free guidance of diffusion models, generated from a model called GLIDE using the conditioning text *A stained glass window of a panda eating bamboo*. Examples on the left were generated with $\lambda = 0$ (no guidance, just the plain conditional model) whereas examples on the right were generated with $\lambda = 3$. [From Nichol *et al.* (2021) with permission.]

Section 19.1

sion models (Rombach *et al.*, 2021). Here an autoencoder is first trained on noise-free images to obtain a lower-dimensional representation of the images and is then fixed. A U-net architecture is then trained to perform the denoising within the lower-dimensional space, which itself is not directly interpretable as an image. Finally, the denoised representation is mapped into the high-resolution image space using the output half of the fixed autoencoder network. This approach makes more efficient use of the low-dimensional space, which can then focus on image semantics, leaving the decoder to create a corresponding sharp, high-resolution image from the denoised low-dimensional representation.

There are many other applications of conditional image generation including inpainting, un-cropping, restoration, image morphing, style transfer, colourization, de-blurring, and video generation (Yang, Srivastava, and Mandt, 2022). An example of inpainting is shown in Figure 20.9.

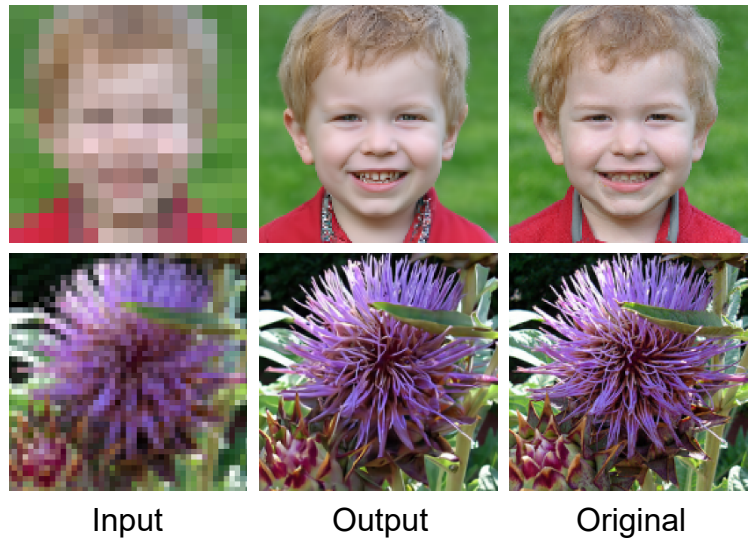

Figure 20.8 Two examples of low-resolution images along with associated samples of corresponding high-resolution images generated by a diffusion model. The top row shows a 16×16 input image and the corresponding 128×128 output image along with the original image from which the input image was generated. The bottom row shows a 64×64 input image with a 256×256 output image, again with the original image for comparison. [From Saharia, Ho, *et al.* (2021) with permission.]

Figure 20.9 Example of inpainting showing the original image on the left, an image with sections removed in the middle, and the image with inpainting on the right. [From Saharia, Chan, Chang, *et al.* (2021) with permission.]

Exercises

20.1 ($\star$) Using (20.3) write down expressions for the mean and covariance of $\mathbf{z}_t$ in terms of the mean and covariance of $\mathbf{z}_{t-1}$. Hence, show that for $0 < \beta_t < 1$ the mean of the distribution of $\mathbf{z}_t$ is closer to zero than the mean of $\mathbf{z}_{t-1}$ and that the covariance of $\mathbf{z}_t$ is closer to the unit matrix $\mathbf{I}$ than the covariance of $\mathbf{z}_{t-1}$.

20.2 ($\star$) Show that the transformation (20.1) can be written in the equivalent form (20.2).

20.3 ($\star\star\star$) In this exercise we use proof by induction to show that the marginal distribution of $\mathbf{x}_t$ for the forward process of the diffusion model, as defined by (20.4), is given by (20.6) where α_t is defined by (20.7). First verify that (20.6) holds when $t = 1$. Now assume that (20.6) is true for some particular value of t and derive the corresponding result for the value $t + 1$. To do this, it is easiest to write the forward process using the representation (20.3) and to make use of the result (3.212), which shows that the sum of two independent Gaussian random variables is itself a Gaussian in which the means and covariances are additive.

20.4 ($\star$) By using the result (20.6), where α_t is defined by (20.7), show that in the limit $T \to \infty$ we obtain (20.9).