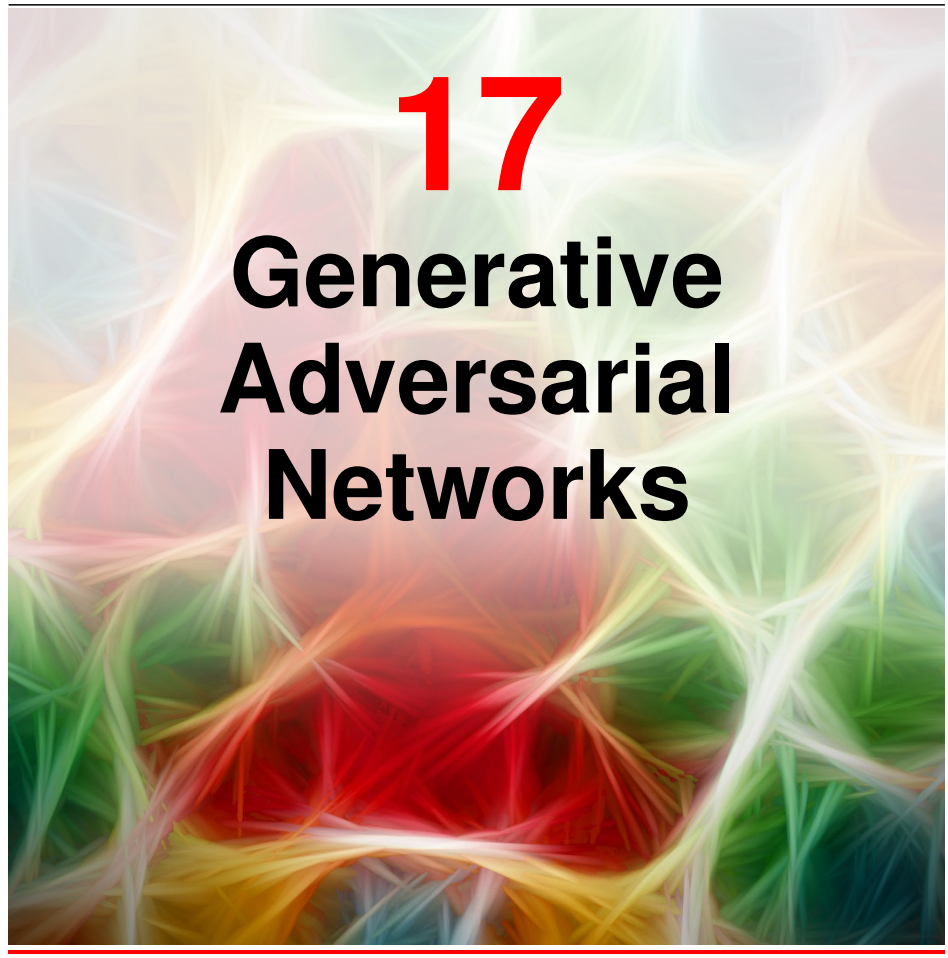

17 Generative Adversarial Networks

Generative models use machine learning algorithms to learn a distribution from a set of training data and then generate new examples from that distribution. For example, a generative model might be trained on images of animals and then used to generate new images of animals. We can think of such a generative model in terms of a distribution $p(\mathbf{x}|\mathbf{w})$ in which $\mathbf{x}$ is a vector in the data space, and $\mathbf{w}$ represent the learnable parameters of the model. In many cases we are interested in conditional generative models of the form $p(\mathbf{x}|\mathbf{c}, \mathbf{w})$ where $\mathbf{c}$ represents a vector of conditioning variables. In the case of our generative model for animal images, we may wish to specify that a generated image should be of a particular animal, such as a cat or a dog, specified by the value of $\mathbf{c}$.

For real-world applications such as image generation, the distributions are extremely complex, and consequently the introduction of deep learning has dramatically improved the performance of generative models. We have already encountered

C. M. Bishop, H. Bishop, *Deep Learning*, https://doi.org/10.1007/978-3-031-45468-4_17

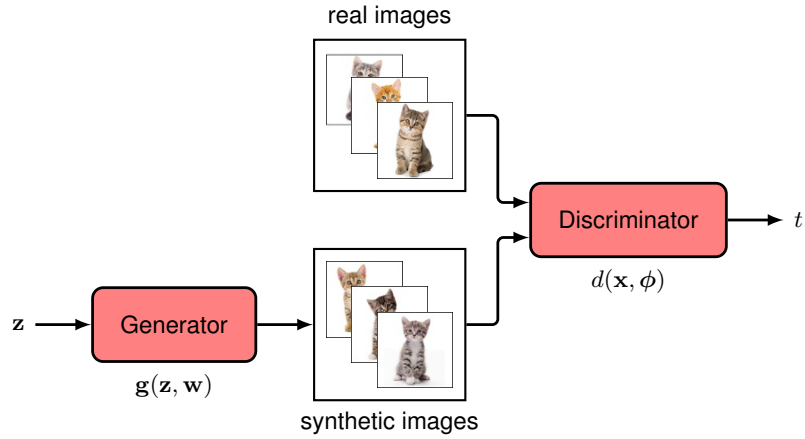

Figure 17.1 Schematic illustration of a GAN in which a discriminator neural network $d(\mathbf{x}, \boldsymbol{\phi})$ is trained to distinguish between real samples from the training set, in this case images of kittens, and synthetic samples produced by the generator network $\mathbf{g}(\mathbf{z}, \mathbf{w})$. The generator aims to maximize the error of the discriminator network by producing realistic images, whereas the discriminator network tries to minimize the same error by becoming better at distinguishing real from synthetic examples.

Chapter 12
Section 16.4.4

an important class of deep generative models when we discussed autoregressive large language models based on transformers. We have also outlined four important classes of generative model based on nonlinear latent variable models, and in this chapter we discuss the first of these, called generative adversarial networks. The other three approaches will be discussed in subsequent chapters.

17.1. Adversarial Training

Consider a generative model based on a nonlinear transformation from a latent space $\mathbf{z}$ to a data space $\mathbf{x}$. We introduce a latent distribution $p(\mathbf{z})$, which might take the form of a simple Gaussian

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}), \tag{17.1}$$

along with with a nonlinear transformation $\mathbf{x} = \mathbf{g}(\mathbf{z}, \mathbf{w})$ defined by a deep neural network with learnable parameters $\mathbf{w}$ known as the *generator*. Together these implicitly define a distribution over $\mathbf{x}$, and our goal is to fit this distribution to a data set of training examples $\{\mathbf{x}_n\}$ where $n = 1, \ldots, N$. However, we cannot determine $\mathbf{w}$ by optimizing the likelihood function because this cannot, in general, be evaluated in closed form. The key idea of *generative adversarial networks*, or GANs, (Goodfellow *et al.*, 2014; Ruthotto and Haber, 2021) is to introduce a second *discriminator* network, which is trained jointly with the generator network and which provides a training signal to update the weights of the generator. This is illustrated in Figure 17.1.

The goal of the discriminator network is to distinguish between real examples from the data set and synthetic, or 'fake', examples produced by the generator network, and it is trained by minimizing a conventional classification error function. Conversely, the goal of the generator network is to maximize this error by synthesizing examples from the same distribution as the training set. The generator and discriminator networks are therefore working against each other, hence the term 'adversarial'. This is an example of a *zero-sum game* in which any gain by one network represents a loss to the other. It allows the discriminator network to provide a training signal, which can be used to train the generator network, and this turns the unsupervised density modelling problem into a form of supervised learning.

17.1.1 Loss function

To make this precise, we define a binary target variable given by

$$t = 1, \quad \text{real data}, \tag{17.2}$$

$$t = 0, \quad \text{synthetic data}. \tag{17.3}$$

The discriminator network has a single output unit with a logistic-sigmoid activation function, whose output represents the probability that a data vector $\mathbf{x}$ is real:

$$P(t = 1) = d(\mathbf{x}, \boldsymbol{\phi}). \tag{17.4}$$

We train the discriminator network using the standard cross-entropy error function, which takes the form

$$E(\mathbf{w}, \boldsymbol{\phi}) = -\frac{1}{N} \sum_{n=1}^{N} \{t_n \ln d_n + (1 - t_n) \ln(1 - d_n)\} \tag{17.5}$$

where $d_n = d(\mathbf{x}_n, \boldsymbol{\phi})$ is the output of the discriminator network for input vector n, and we have normalized by the total number of data points. The training set comprises both real data examples denoted $\mathbf{x}_n$ and synthetic examples given by the output of the generator network $\mathbf{g}(\mathbf{z}_n, \mathbf{w})$ where $\mathbf{z}_n$ is a random sample from the latent space distribution $p(\mathbf{z})$. Since $t_n = 1$ for real examples and $t_n = 0$ for synthetic examples, we can write the error function (17.5) in the form

Section 1.2.4

$$E_{\text{GAN}}(\mathbf{w}, \boldsymbol{\phi}) = -\frac{1}{N_{\text{real}}} \sum_{n \in \text{real}} \ln d(\mathbf{x}_n, \boldsymbol{\phi}) - \frac{1}{N_{\text{synth}}} \sum_{n \in \text{synth}} \ln(1 - d(\mathbf{g}(\mathbf{z}_n, \mathbf{w}), \boldsymbol{\phi})) \tag{17.6}$$

where typically the number N_{real} of real data points is equal to the number N_{synth} of synthetic data points. This combination of generator and discriminator networks can be trained end-to-end using stochastic gradient descent with gradients evaluated using backpropagation. However, the unusual aspect is the adversarial training whereby the error is minimized with respect to $\boldsymbol{\phi}$ but *maximized* with respect to $\mathbf{w}$.

Chapter 7

This maximization can be done using standard gradient-based methods with the sign of the gradient reversed, so that the parameter updates become

$$\Delta\boldsymbol{\phi} = -\lambda\nabla_{\boldsymbol{\phi}}E_n(\mathbf{w}, \boldsymbol{\phi}) \tag{17.7}$$

$$\Delta\mathbf{w} = \lambda\nabla_{\mathbf{w}}E_n(\mathbf{w}, \boldsymbol{\phi}) \tag{17.8}$$

where $E_n(\mathbf{w}, \boldsymbol{\phi})$ denotes the error defined for data point n or more generally for a mini-batch of data points. Note that the two terms in (17.7) and (17.8) have different signs since the discriminator is trained to decrease the error rate whereas the generator is trained to increase it. In practice, training alternates between updating the parameters of the generative network and updating those of the discriminative network, in each case taking just one gradient descent step using a mini-batch, after which a new set of synthetic samples is generated. If the generator succeeds in finding a perfect solution, then the discriminator network will be unable to tell the difference between the real and synthetic data and hence will always produce an output of 0.5. Once the GAN is trained, the discriminator network is discarded and the generator network can be used to synthesize new examples in the data space by sampling from the latent space and propagating those samples through the trained generator network. We can show that for generative and discriminative networks having unlimited flexibility, a fully optimized GAN will have a generative distribution that matches the data distribution exactly. Some impressive examples of synthetic face images generated by a GAN are shown in Figure 1.3.

Exercise 17.1

The GAN model discussed so far generates samples from the unconditional distribution $p(\mathbf{x})$. For example, it could generate synthetic images of dogs if it is trained on dog images. We can also create *conditional* GANs (Mirza and Osindero, 2014), which sample from a conditional distribution $p(\mathbf{x}|\mathbf{c})$ in which the conditioning vector $\mathbf{c}$ might, for example, represent different species of dog. To do this, both the generator and the discriminator network take $\mathbf{c}$ as an additional input, and labelled examples of images, comprising pairs $\{\mathbf{x}_n, \mathbf{c}_n\}$, are used for training. Once the GAN has been trained, images from a desired class can be generated by setting $\mathbf{c}$ to the corresponding class vector. Compared to training separate GANs for each class, this has the advantage that shared internal representations can be learned jointly across all classes, thereby making more efficient use of the data.

17.1.2 GAN training in practice

Although GANs can produce high quality results, they are not easy to train successfully due to the adversarial learning. Also, unlike standard error function minimization, there is no metric of progress because the objective can go up as well as down during training.

Exercise 17.2

One challenge that can arise is called *mode collapse*, in which the generator network weights adapt during training such that all latent-variable samples $\mathbf{z}$ are mapped to a subset of possible valid outputs. In extreme cases the output can correspond to just one, or a small number, of the output values $\mathbf{x}$. The discriminator then assigns the value 0.5 to these instances, and training ceases. For example, a GAN trained on handwritten digits might learn to generate only examples of the digit '3', and while

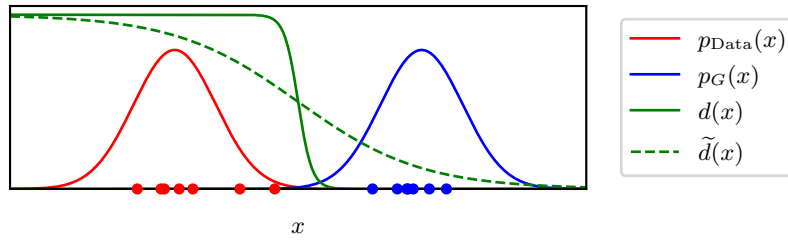

Figure 17.2 Conceptual illustration of why it can be difficult to train GANs, showing a simple one-dimensional data space x with the fixed, but unknown, data distribution $p_{\text{Data}}(x)$ and the initial generative distribution $p_{\text{G}}(x)$. The optimal discriminator function $d(x)$ has virtually zero gradient in the vicinity of either the training or synthetic data points, making learning very slow. A smoothed version $\widetilde{d}(x)$ of the discriminator function can lead to faster learning.

the discriminator is unable to distinguish these from genuine examples of the digit '3', it fails to recognize that the generator is not generating the full range of digits.

Insight into the difficulty of training GANs can be obtained by considering Figure 17.2, which shows a simple one§-dimensional data space x with samples $\{x_n\}$ drawn from the fixed, but unknown, data distribution $p_{\text{Data}}(x)$. Also shown is the initial generative distribution $p_{\text{G}}(x)$ together with samples drawn from this distribution. Because the data and generative distributions are so different, the optimal discriminator function $d(x)$ is easy to learn and has a very steep fall-off with virtually zero gradient in the vicinity of either the real or synthetic samples. Consider the second term in the GAN error function (17.6). Because $d(\mathbf{g}(\mathbf{z}, \mathbf{w}), \boldsymbol{\phi})$ is equal to zero across the region spanned by the generated samples, small changes in the parameters $\mathbf{w}$ of the generative network produce very little change in the output of the discriminator and so the gradients are small and learning proceeds slowly.

This can be addressed by using a smoothed version $\widetilde{d}(x)$ of the discriminator function, illustrated in Figure 17.2, thereby providing a stronger gradient to drive the training of the generator network. The *least-squares GAN* (Mao *et al.*, 2016) achieves smoothing by modifying the discriminator to produce a real-valued output rather than a probability in the range $(0, 1)$ and by replacing the cross-entropy error function with a sum-of-squares error function. Alternatively, the technique of *instance noise* (Sønderby *et al.*, 2016) adds Gaussian noise to both the real data and the synthetic samples, again leading to a smoother discriminator function.

Numerous other modifications to the GAN error function and training procedure have been proposed to improve training (Mescheder, Geiger, and Nowozin, 2018). One change that is often used is to replace the generative network term in the original error function

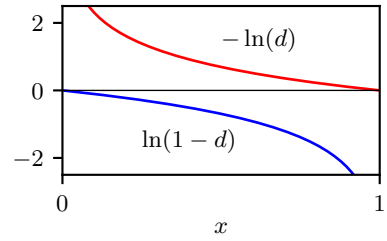

Figure 17.3 Plots of $-\ln(d)$ and $\ln(1-d)$ showing the very different behaviour of the gradients close to $d = 0$ and $d = 1$.

$$-\frac{1}{N_{\text{synth}}}\sum_{n\in\text{synth}}\ln(1-d(\mathbf{g}(\mathbf{z}_n,\mathbf{w}),\boldsymbol{\phi})) \tag{17.9}$$

with the modified form

$$\frac{1}{N_{\text{synth}}}\sum_{n\in\text{synth}}\ln d(\mathbf{g}(\mathbf{z}_n,\mathbf{w}),\boldsymbol{\phi}). \tag{17.10}$$

Although the first form minimizes the probability that the image is fake, the second version maximizes the probability that the image is real. The different properties of these two forms can be understood from Figure 17.3. When the generative distribution $p_{\text{G}}(x)$ is very different from the true data distribution $p_{\text{Data}}(x)$, the quantity $d(\mathbf{g}(\mathbf{z},\mathbf{w}))$ is close to zero, and hence the first form has a very small gradient, whereas the second form has a large gradient, leading to faster training.

A more direct way to ensure that the generator distribution $p_{\text{G}}(x)$ moves towards the data distribution $p_{\text{data}}(x)$ is to modify the error criterion to reflect how far apart the two distributions are in data space. This can be measured using the *Wasserstein distance*, also known as the *earth mover's distance*. Imagine the distribution $p_{\text{G}}(x)$ as a pile of earth that is transported in small increments to construct the distribution $p_{\text{data}}(x)$. The Wasserstein metric is the total amount of earth moved multiplied by the mean distance moved. Of the many ways of rearranging the pile of earth to build $p_{\text{data}}(x)$, the one that yields the smallest mean distance is the one used to define the metric. In practice, this cannot be implemented directly, and it is approximated by using a discriminator network that has real-valued outputs and then limiting the gradient $\nabla_x d(\mathbf{x},\boldsymbol{\phi})$ of the discriminator function with respect to x by using weight clipping, giving rise to the *Wasserstein GAN* (Arjovsky, Chintala, and Bottou, 2017). An improved approach is to introduce a penalty on the gradient, giving rise to the *gradient penalty Wasserstein GAN* (Gulrajani *et al.*, 2017) whose error function is given by

$$\begin{aligned}E_{\text{WGAN-GP}}(\mathbf{w},\boldsymbol{\phi}) = &-\frac{1}{N_{\text{real}}}\sum_{n\in\text{real}}\left[\ln d(\mathbf{x}_n,\boldsymbol{\phi}) - \eta\left(\|\nabla_{\mathbf{x}_n} d(\mathbf{x}_n,\boldsymbol{\phi})\|^2 - 1\right)^2\right] \\ &+\frac{1}{N_{\text{synth}}}\sum_{n\in\text{synth}}\ln d(\mathbf{g}(\mathbf{z}_n,\mathbf{w},\boldsymbol{\phi}))\end{aligned} \tag{17.11}$$

where η controls the relative importance of the penalty term.

17.2. Image GANs

The basic concept of the GAN has given rise to a huge research literature, with many algorithmic developments and numerous applications. One of the most widespread and successful application areas for GANs is the generation of images. Early GAN models used fully connected networks for the generator and discriminator. However, there are many benefits to using convolutional networks, especially for images of higher resolution. The discriminator network takes an image as input and provides a scalar probability as output, so a standard convolutional network is appropriate. The generator network needs to map a lower-dimensional latent space into a high-resolution image, and so a network based on transpose convolutions is used, as illustrated in Figure 17.4.

Chapter 10

Section 10.5.3

High quality images can be obtained by progressively growing both the generator network and the discriminator network starting from a low resolution and then successively adding new layers that model increasingly fine details as training progresses (Karras *et al.*, 2017). This speeds up the training and permits the synthesis of high-resolution images of size 1024×1024 starting from images of size 4×4. As an example of the scale and complexity of some GAN architectures, consider the GAN model for class-conditional image generation called *BigGAN*, whose architecture is shown in Figure 17.5.

17.2.1 CycleGAN

As an example of the broad variety of GANs we consider an architecture called a *CycleGAN* (Zhu *et al.*, 2017). This also illustrates how techniques in deep learning can be adapted to solve different kinds of problems beyond traditional tasks such as

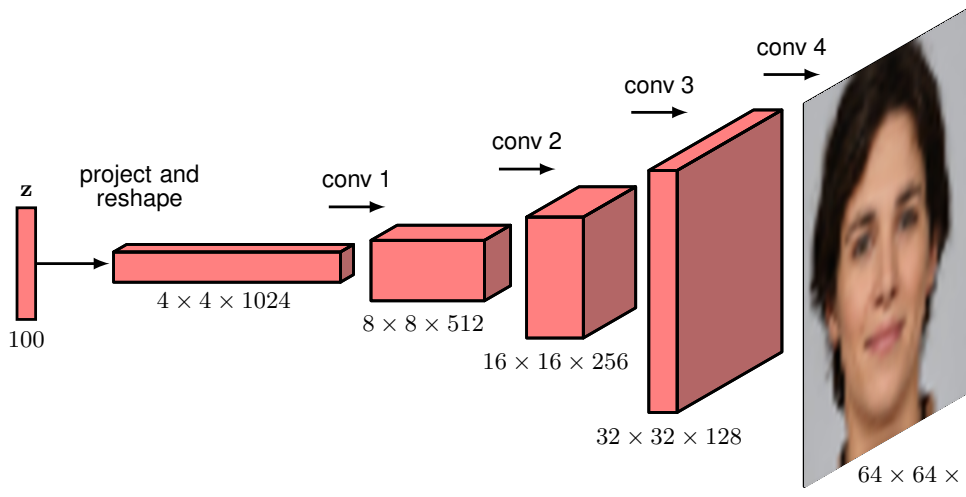

Figure 17.4 Example architecture of a deep convolutional GAN showing the use of transpose convolutions to expand the dimensionality in successive blocks of the network.

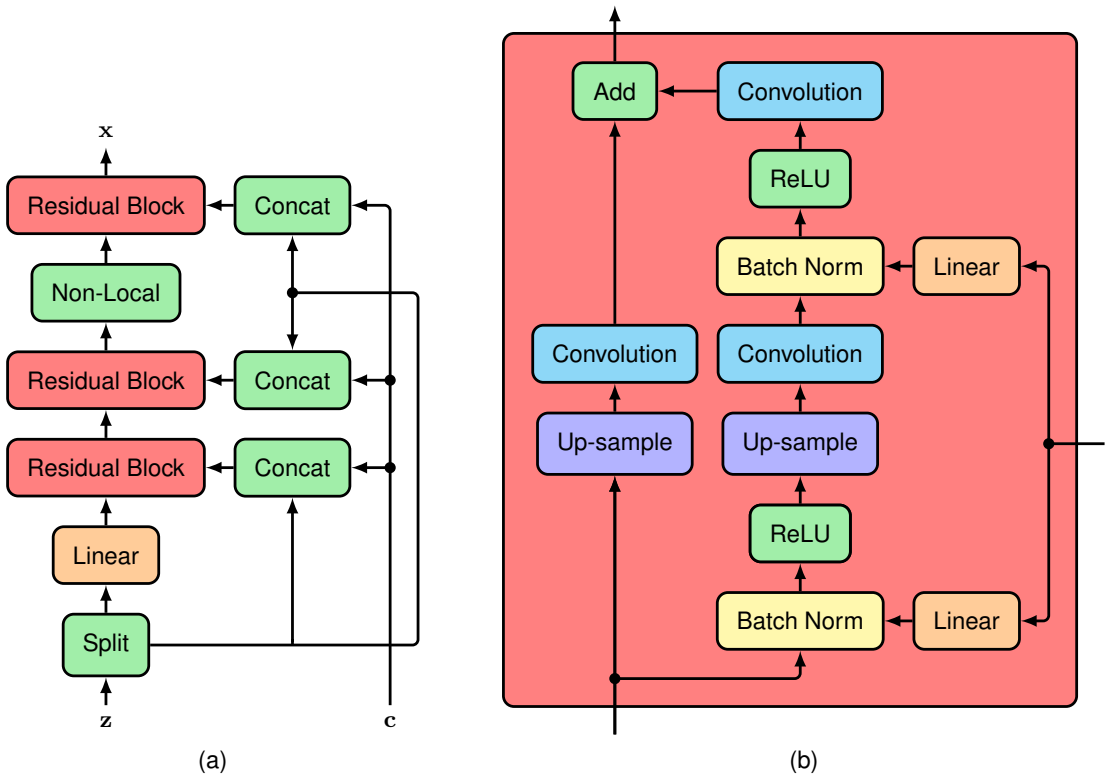

Figure 17.5 (a) Architecture of the generative network in the BigGAN model, which has over 70 million parameters. (b) Details of each of the residual blocks in the generative network. The discriminative network, which has 88 million parameters, has a somewhat analogous structure except that it uses average pooling layers to reduce the dimensionality, instead of using up-sampling to increase the dimensionality. [Based on Brock, Donahue, and Simonyan (2018).]

classification and density estimation. Consider the problem of turning a photograph into a Monet painting of the same scene, or vice versa. In Figure 17.6 we show examples of image pairs from a trained CycleGAN that has learned to perform such an image-to-image translation.

The aim is to learn two bijective (one-to-one) mappings, one that goes from the domain X of photographs to the domain Y of Monet paintings and one in the reverse direction. To achieve this, CycleGAN makes use of two conditional generators, $\mathbf{g}_X$ and $\mathbf{g}_Y$, and two discriminators, d_X and d_Y. The generator $\mathbf{g}_X(\mathbf{y}, \mathbf{w}_X)$ takes as input a sample painting $\mathbf{y} \in Y$ and generates a corresponding synthetic photograph, whereas the discriminator $d_X(\mathbf{x}, \boldsymbol{\phi}_X)$ distinguishes between synthetic and real photographs. Similarly, the generator $\mathbf{g}_Y(\mathbf{x}, \mathbf{w}_Y)$ takes a photograph $\mathbf{x} \in X$ as input and generates a synthetic painting $\mathbf{y}$, and the discriminator $d_Y(\mathbf{y}, \boldsymbol{\phi}_Y)$ distinguishes between synthetic paintings and real ones. The discriminator d_X is therefore trained on a combination of synthetic photographs generated by $\mathbf{g}_X$ and real photographs, whereas d_Y is trained on a combination of synthetic paintings generated by $\mathbf{g}_Y$ and

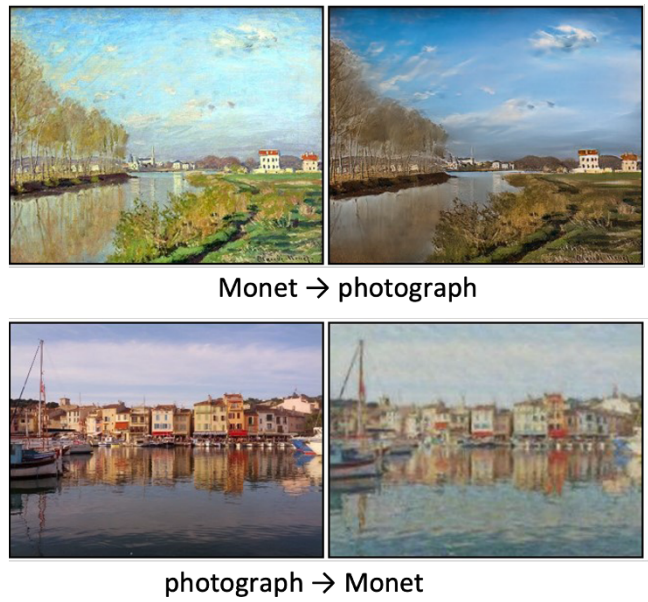

Figure 17.6 Examples of image translation using a CycleGAN showing the synthesis of a photographic-style image from a Monet painting (top row) and the synthesis of an image in the style of a Monet painting from a photograph (bottom row). [From Zhu *et al.* (2017) with permission.]

real paintings.

If we train this architecture using the standard GAN loss function, it would learn to generate realistic synthetic Monet paintings and realistic synthetic photographs, but there would be nothing to force a generated painting to look anything like the corresponding photograph, or vice versa. We therefore introduce an additional term in the loss function called the cycle consistency error, containing two terms, whose construction is illustrated in Figure 17.7.

The goal is to ensure that when a photograph is translated into a painting and then back into a photograph it should be close to the original photograph, thereby ensuring that the generated painting retains sufficient information about the photograph to allow the photograph to be reconstructed. Similarly, when a painting is translated into a photograph and then back into a painting it should be close to the

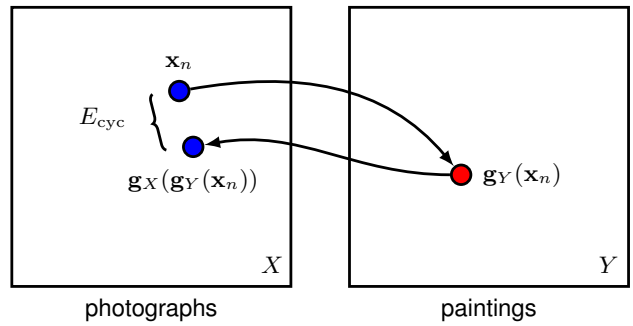

Figure 17.7 Diagram showing how the cycle consistency error is calculated for an example photograph $\mathbf{x}_n$. The photograph is first mapped into the painting domain using the generator $\mathbf{g}_Y$, and the resulting vector is then mapped back into the photograph domain using the generator $\mathbf{g}_X$. The discrepancy between the resulting photograph and the original $\mathbf{x}_n$ defines a contribution to the cycle consistency error. An analogous process is used to calculate the contribution to the cycle consistency error from a painting $\mathbf{y}_n$ by mapping it to a photograph using $\mathbf{g}_X$ and then back to a painting using $\mathbf{g}_Y$.

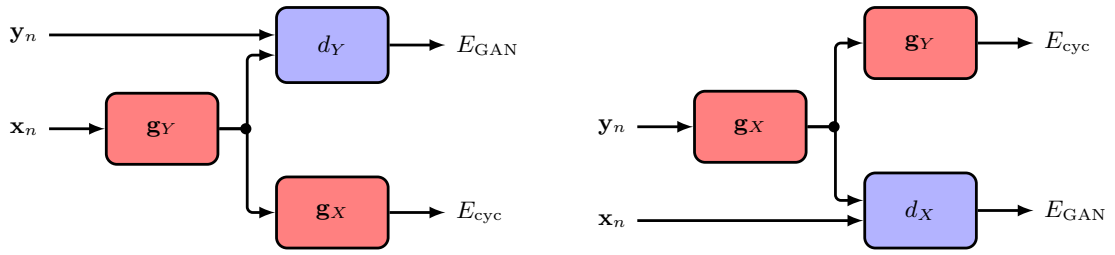

Figure 17.8 Flow of information through a CycleGAN. The total error for the data points $\mathbf{x}_n$ and $\mathbf{y}_n$ is the sum of the four component errors.

original painting. Applying this to all the photographs and paintings in the training set then gives a cycle consistency error of the form

$$E_{\text{cyc}}(\mathbf{w}_X, \mathbf{w}_Y) = \frac{1}{N_X} \sum_{n \in X} \|\mathbf{g}_X(\mathbf{g}_Y(\mathbf{x}_n)) - \mathbf{x}_n\|_1 + \frac{1}{N_Y} \sum_{n \in Y} \|\mathbf{g}_Y(\mathbf{g}_X(\mathbf{y}_n)) - \mathbf{y}_n\|_1 \tag{17.12}$$

where $\|\cdot\|_1$ denotes the L1 norm. The cycle consistency error is added to the usual GAN loss functions defined by (17.6) to give a total error function:

$$E_{\text{GAN}}(\mathbf{w}_X, \boldsymbol{\phi}_X) + E_{\text{GAN}}(\mathbf{w}_Y, \boldsymbol{\phi}_Y) + \eta E_{\text{cyc}}(\mathbf{w}_X, \mathbf{w}_Y) \tag{17.13}$$

where the coefficient η determines the relative importance of the GAN errors and the cycle consistency error. Information flow through the CycleGAN when calculating the error function for one image and one painting is shown in Figure 17.8.

Section 6.3.3

We have seen that GANs can perform well as generative models, but they can also be used for *representation learning* in which rich statistical structure in a data set is revealed through unsupervised learning. When the deep convolutional GAN shown in Figure 17.4 is trained on a data set of bedroom images (Radford, Metz, and Chintala, 2015) and random samples from the latent space are propagated through the trained network, the generated images also look like bedrooms, as expected. In addition, however, the latent space has become organized in ways that are semantically meaningful. For example, if we follow a smooth trajectory through the latent space and generate the corresponding series of images, we obtain smooth transitions from one image to the next, as seen in Figure 17.9.

Moreover, it is possible to identify directions in latent space that correspond to semantically meaningful transformations. For example, for faces, one direction might correspond to changes in the orientation of the face, whereas other directions might correspond to changes in lighting or the degree to which the face is smiling or not. These are called *disentangled representations* and allow new images to be synthesized having specified properties. Figure 17.10 is an example from a GAN trained on face images, showing that semantic attributes such as gender or the presence of glasses correspond to particular directions in latent space.

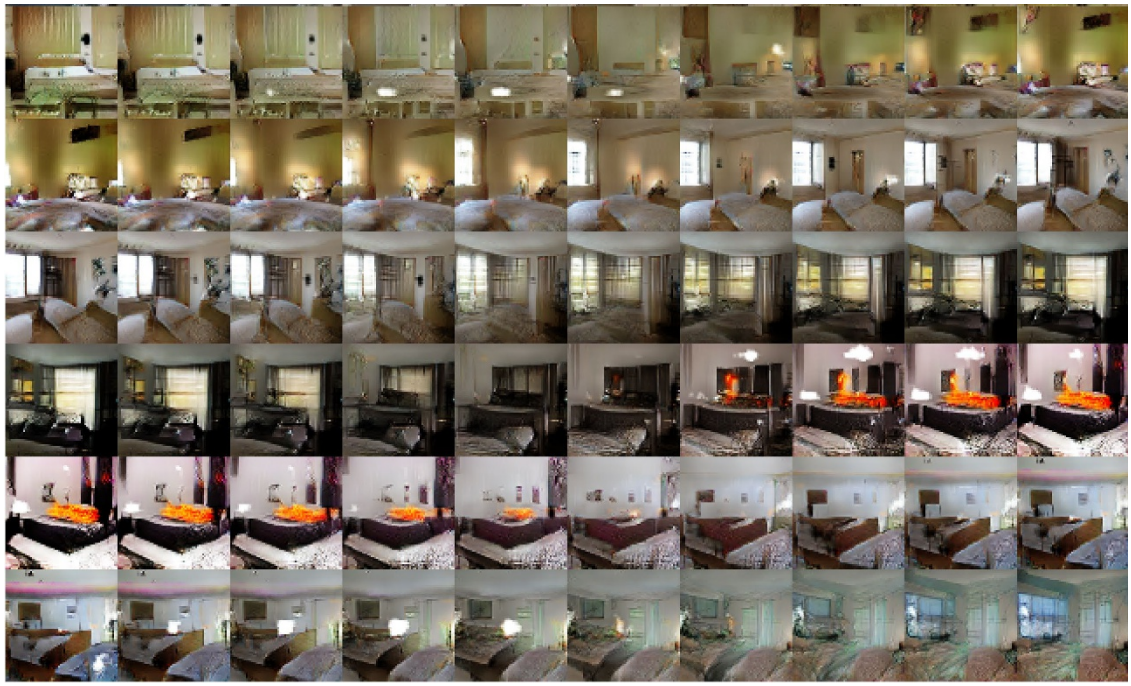

Figure 17.9 Samples generated by a deep convolutional GAN trained on images of bedrooms. Each row is generated by taking a smooth walk through latent space between randomly generated locations. We see smooth transitions, with each image plausibly looking like a bedroom. In the bottom row, for example, we see a TV on the wall gradually morph into a window. [From Radford, Metz, and Chintala (2015) with permission.]

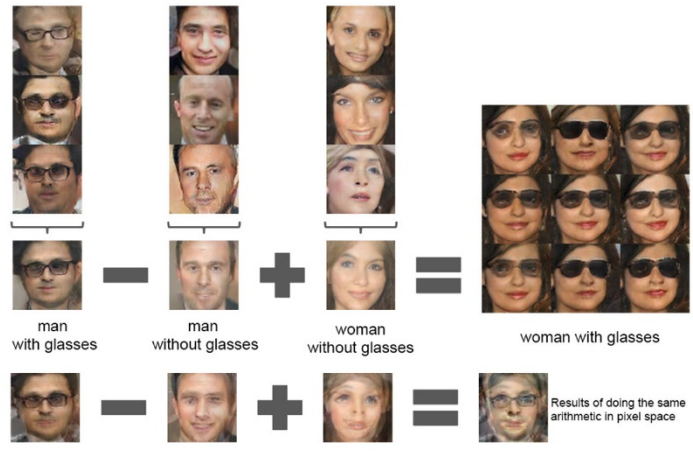

Figure 17.10 An example of vector arithmetic in the latent space of a trained GAN. In each of the three columns, the latent space vectors that generated these images are averaged and then vector arithmetic is applied to the resulting mean vectors to create a new vector corresponding to the central image in the 3×3 array on the right. Adding noise to this vector generates another eight sample images. The four images on the bottom row show that the same arithmetic applied directly in data space simply results in a blurred image due to misalignment. [From Radford, Metz, and Chintala (2015) with permission.]