CS 282A: Deep Neural Network **Fall 2022**

Lecture 26: November 22

Lecturer: Anant Sahai *Scribes: Zheyu Lu, Hyungki Im*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

26.1 Introduction

Generative models have not been around for a long time, but we have already seen waves of advances in this field, such as autoregressive (AR) models, generative adversarial network (GAN), and, more recently, the diffusion models. In this and the subsequent lectures, we will cover these frameworks with both high-level ideas and technical details.

In practice, we are usually interested in generating big objects, in which case we count on the fact that there is some kind of structure that is shared across the object. After all, what makes deep learning work is the idea of weight sharing, where something is replicated across different places in the objects of interest.

In the last lecture, we talked about different approaches for generation and one of them is the AR approach which takes the GPT style. Basically, it treats the object of interest as a sequence and samples one entry at a time conditioned on "past" information. Sampling consumes randomness, *i.e.,* randomness gets used every time for the generation of samples, and this is why we do not get the same result over and over again.

26.2 GAN Approach

26.2.1 GAN Architecture

Basically, the idea behind GAN approach is to leverage the fact that maybe we can tell real from fake and use this to help us generate real things. More concretely speaking, GAN approach uses a trained discriminator that can tell real examples of X from fake ones to train a generator that can produce realistic fakes. Figure 26.1 shows the GAN architecture and the architecture of the generator, and the discriminator that is used in GAN. The randomness comes into the generator as input and passes through a deep network structure, generating an example image X. The deep network used in the generator can contain any inductive biases that may help generate X. The discriminator takes this X (either real or fake) and classifies whether the X is real or fake. One of the advantages of being the discriminator being a deep architecture is that it actually learns the key discrimination between real and fake samples and tells this to the generator by backpropagation.

The core idea of the GAN approach is that the generator and the discriminator are trained together. However, it is very challenging to train the generator and the discriminator together, so we can split the training process into two steps. We discuss this strategy in the following two sections.

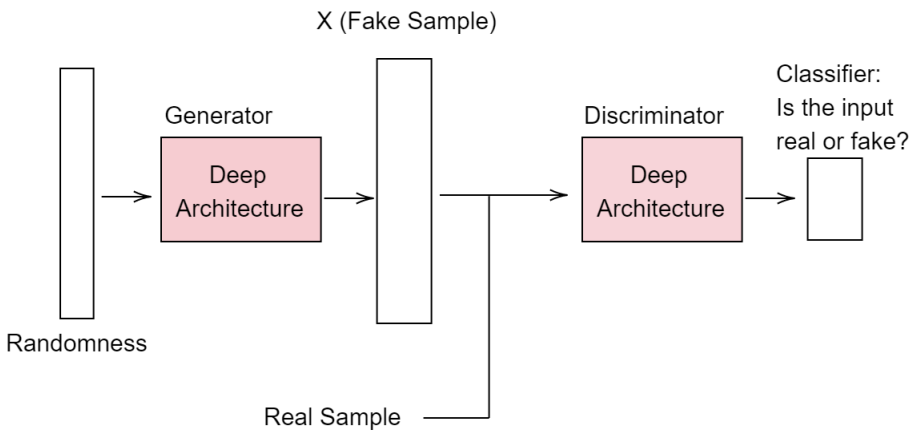

Figure 26.1: GAN Architecture

26.2.2 Step I: Train a Discriminator Given a Frozen Generator

In this step, we train the discriminator given a fixed generator. We can think of this as a classic binary classification problem where there are two categories: "real" and "fake". The frozen generator will generate some fake samples and we mix these fake samples with real samples and use these to train the discriminator.

26.2.3 Step II: Train a Generator Given a Frozen Discriminator

In this case, we can train the generator by inputting random samples (such as sampling from gaussian distribution) and obtain a "real" score or a "fake" score from the discriminator and take SGD steps. If the output of the discriminator is a "real" score, we should maximize it; otherwise, we should minimize it. Figure 26.2 illustrates this step.

Q. Why are we taking the gradient for the input, although it is random? The generator is not tuning to create a specific example that can fool the discriminator. Indeed, it tries to generate a general output that the discriminator can't easily distinguish from the real samples. That is the main reason why we are using the random input only once and throwing it away. This is different from the training procedure, which uses the same dataset again and again.

Now we can iterate between step I and step II to construct a generative model, and figure 26.3 illustrates a simplified version of this training process with a linear classifier as a discriminator.

However, there are several things that can go wrong in this setting. One possible flaw is if the discriminator is too good, then no matter what the generator does locally to its parameter, the discriminator would still say it is still too fake. Then this results in tiny gradients, which are not desirable. Also, if the target distribution is spread out instead of sticking together, it is highly possible for our generator to be oscillating between those target distributions. This is called the problem of mode collapse.

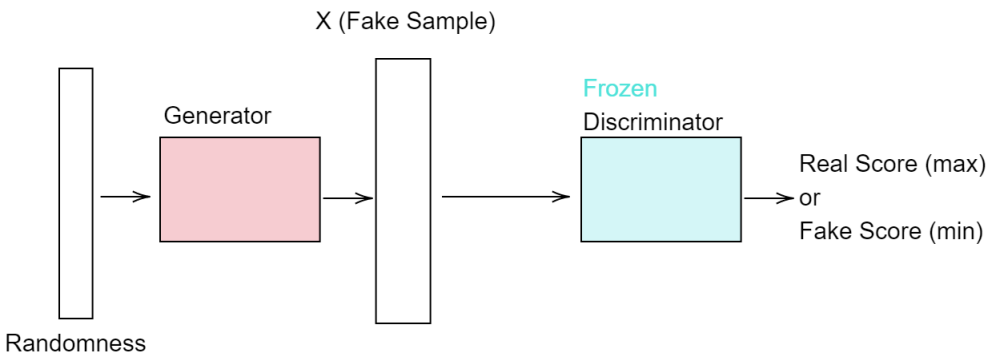

Figure 26.2: Train Generator with Frozen Discriminator

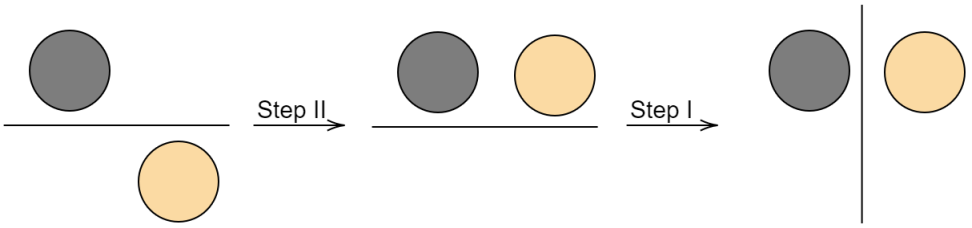

Figure 26.3: Iterating Step I and II: The black circle represents the distribution of real samples while the yellow circle represents the distribution of fake samples generated by the generator. We use a linear classifier as a discriminator and represented as a line in this figure.

26.2.4 Mode Collapse

GAN training is notoriously challenging to do because of the problem called mode collapse. This happens when the generator lacks sufficient diversity. Let's take a look at Figure 26.4.

As explained in the figure, the black dots represent the distribution of "real" data, and the orange cluster represents the distribution of the fake samples that are generated by the generator at some point. Our ultimate goal is to create orange clusters over all the real samples (black dots) in the figure. However, under most of the approaches, the generator will end up clustering at a single point of the real sample cluster just as the left figure in Figure 26.4. If this happens, the discriminator will then classify all the samples in the orange area (including the real samples) as fake. So, the generator will shift the orange area (fake samples) to other real sample clusters (In the Figure 26.4, we moved to counter clockwise.) Again, the discriminator will be trained to classify all the samples in the orange area at the right figure as fake samples, and this will continue on and on. This phenomenon is called mode collapse, and the reason for this naming is that some

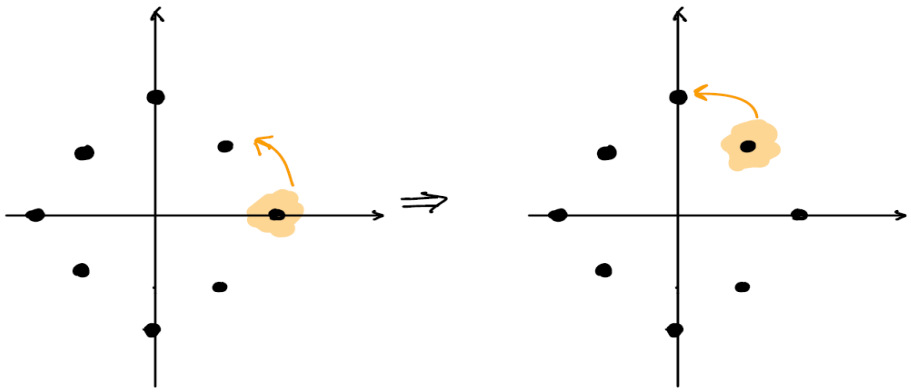

Figure 26.4: Black dots represent the distribution of real samples. The orange cluster represents the distribution of fake samples that are generated by the generator.

modes are not covered by the generator. Mode collapse is deeply rooted in this kind of adversarial learning setting, and this is an active area of research where many people are trying to come up with some methods to solve this issue.

26.3 Diffusion Approach

One of the interesting things about the revolution in deep learning is that sometimes people suddenly realize some specific math techniques that have existed for quite a long time are very important. In the context of diffusion models, these techniques are ordinary differential equations, stochastic differential equations, and thinking about things in continuous time. Before we formally dive into the details of diffusion models, in the following sections, we give two immature ideas that do not work in practice but serve as a good starting point.

26.3.1 First Attempt

Basically speaking, the idea is to use a denoising autoencoder to do generation as shown in Figure 26.5. The idea behind this is that denoising autoencoder has a true example X with noise, compresses X into the latent space and attempts to reconstruct X. However, this simple scheme will not work in practice because if you just add a huge noise to X, the input is dominated by the huge noise and effectively the whole framework becomes reconstructing X from pure noise where the shift is too large.

26.3.2 Second Attempt

Basically speaking, the idea is to do many stages of denoising, *i.e.,* add noise in stages rather than add a single stage of noise in the above first attempt. In this case, as is shown in Figure 26.6, we have a bunch of denoising autoencoders and correspondingly a bunch of targets to reconstruct. We have different noise at different stages.

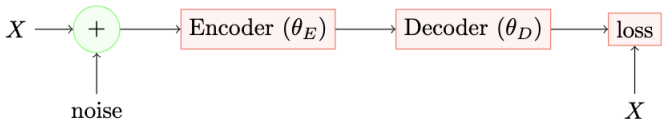

Figure 26.5: Schematics of the first idea where we have a single denoising autoencoder. We add noise to the input real value X and have one encoder with parameters θ_E, one decoder with parameters θ_D, and a loss layer trying to reconstruct X from the noisy data.

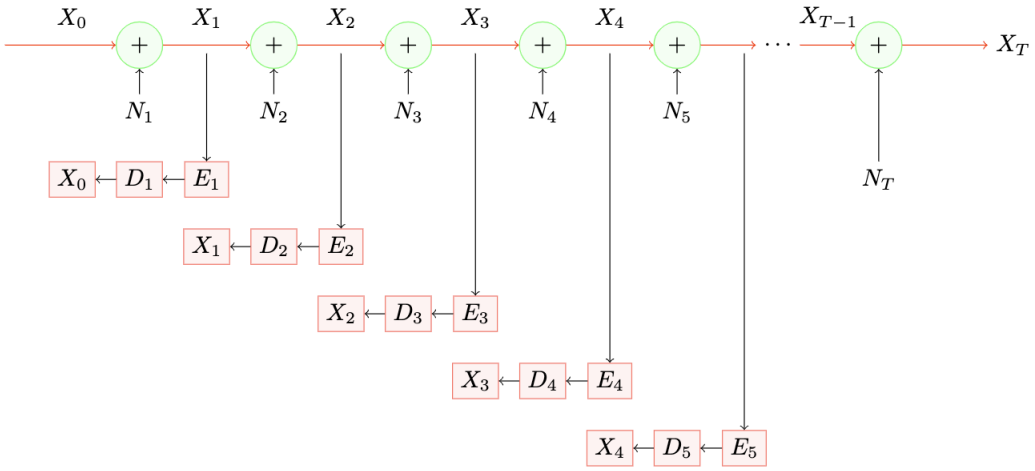

Figure 26.6: Schematics of the second idea where we have multiple denoising autoencoders. At each stage, we add noise $N_i \sim N(0,1)$ to the previous output X_{i-1} and have one encoder E_i and one decoder D_i trying to reconstruct the previous output X_{i-1} from the noisy data X_i.

Thinking about this abstractly is quite hard, so let us think of this using a concrete example on a two-dimensional plane as shown in Figure 26.7 where the true data X_0 are on a line. For the first denoising autoencoder block, we want to reconstruct X_0 from X_1 which means effectively the network will try to push points near the line onto the line, although we do not know what will happen for far away outliers since they may not exist in the training set. For the second denoising autoencoder block, it will also push X_2 towards the line on average although the points may be slightly far away from the line in some places. In general, in each of the denoising autoencoder block, the points will be pushed in the direction towards the line.

Notice that although we add noise with zero mean to X_0 and the resulting X_n has the same mean as X_0, the variance of X_n keeps growing. This is essentially the result of Brownian motion where $\langle x_t^2 \rangle \sim t$. And because the variance is growing over time, we want to standardize it which leads us to a modified design as shown in Figure 26.8 where at each step, we have

$$\begin{aligned} X_{i+1} &= \sqrt{1-\beta_i} X_i + \sqrt{\beta_i} N_i \\ \mathbb{E}[X_{i+1}] &= \sqrt{1-\beta_i}\mathbb{E}[X_i] + \sqrt{\beta_i}\mathbb{E}[N_i] = 0 \\ \mathrm{Var}[X_{i+1}] &= (1-\beta_i)\mathrm{Var}[X_i] + \beta_i \mathrm{Var}[N_i] = 1 \end{aligned} \tag{26.1}$$

Notice that in this design, the real value X_0 will keep shrinking towards 0 over time by a factor of $\prod_{i=1}^{n} \sqrt{1-\beta_i}$. As a result, we transfer whatever we have initially (X_0) to basically just noise at the end.

Also, it is worth noting here that although at each step we keep mean to be 0 and variance to be 1, the distribution is changing over time and we are not getting back to the start. Note that mean and variance alone could not uniquely define a distribution! This could also be illustrated by a two-dimensional example as shown in Figure 26.9. Initially, we have points staying on a circle, as we move forward by adding stages of noises, the circle becomes a cloud of points and the larger β is, the faster the blurring process is. As a result of the attenuation, the circle finally shrinks to a point and noises dominate everything.

Additionally, it is worth noting that standardization is just for controlling things from exploding and has nothing to do with the nature of the directionality of pushing towards the real value. For example, real images are not truly random and nearby pixels have closer values, but the noise will move them in totally different directions. Therefore, for a real image, any kind of divergences may be due to noise while commonalities are more likely due to real common features. Notice that even with these attenuation, the job of each denoising autoencoder block is still very clear, *i.e.,* remove the noise and reconstruct the input.

However, this approach actually still does not work in practice. We will see in the next lecture that how we could further modify the architecture and improve the training procedure to make it work.

26.4 What we wish this lecture also had to make things clearer?

1. It would be better if we provide some reasons why training the generator and the discriminator separately would not be efficient in an organized way.
2. It would be better to introduce some technical concepts to avoid mode collapse.
3. It would be better to introduce some knowledge of stochastic calculus and some technical details of the mathematical description about diffusion.
4. It would be better to add more technical details about the examples used in the lecture to illustrate the ideas behind diffusion models.
5. It would be better to explain more about why the two basic attempts do not work in practice.

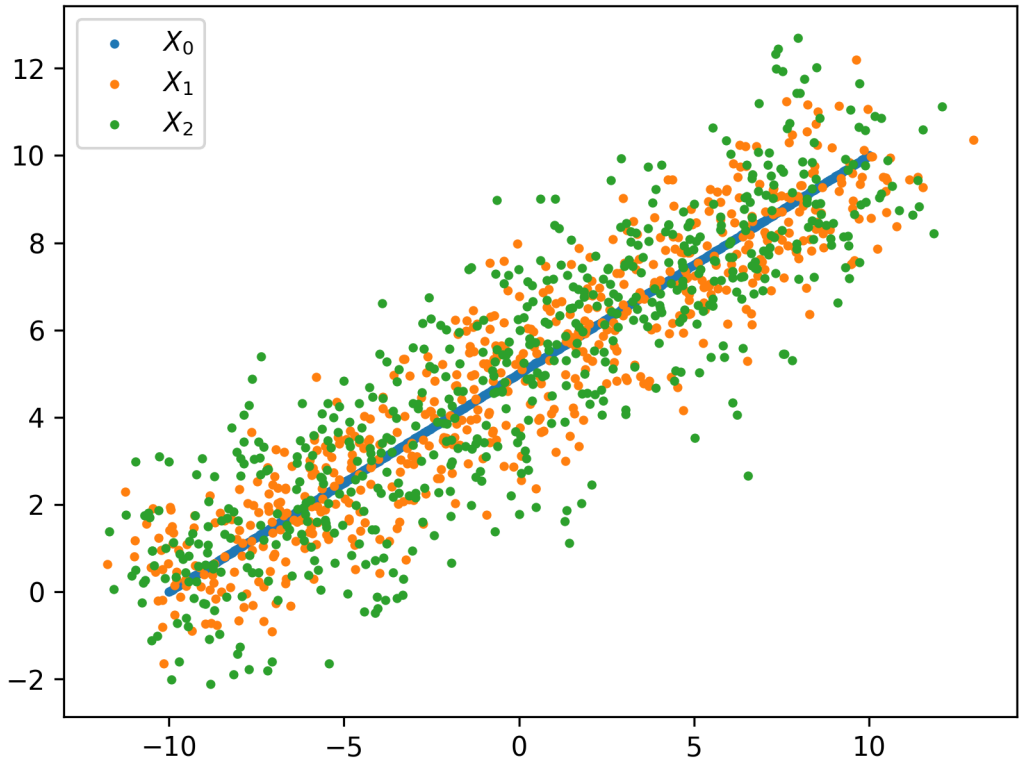

Figure 26.7: A two-dimensional example showing the effect of adding noise at different stages. X_0 are 600 points (x_0, y_0) on a line $y_0 = 0.5x_0 + 5$ ranging from -10 to 10. X_1 are the points (x_1, y_1) satisfying $x_1 = x_0 + n_x^{(1)}$ and $y_1 = y_0 + n_y^{(1)}$ where $n_x^{(1)} \sim N(0,1)$ and $n_y^{(1)} \sim N(0,1)$ are standard normal noise. X_2 are the points (x_2, y_2) satisfying $x_2 = x_1 + n_x^{(2)}$ and $y_2 = y_1 + n_y^{(2)}$ where $n_x^{(2)} \sim N(0,1)$ and $n_y^{(2)} \sim N(0,1)$ are standard normal noise. In this example, we have $\text{Var}(x_0) = 33.44, \text{Var}(y_0) = 8.36, \text{Var}(x_1) = 34.71, \text{Var}(y_1) = 8.95, \text{Var}(x_2) = 35.24, \text{Var}(y_2) = 10.22$. Notice that X_2 has larger variance than X_1 and X_1 has larger variance than X_0.

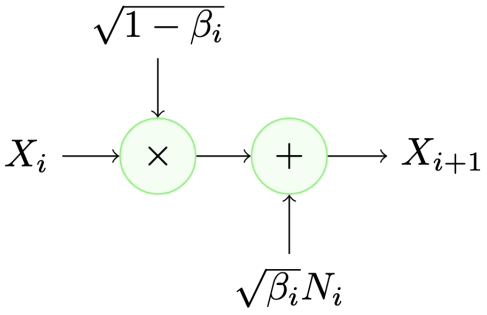

Figure 26.8: Schematics of the attenuation mechanism as a modification to each denosing autoencoder block in the second idea. For each stage, we standardize our result to make it have mean 0 and variance 1, *i.e.*, $X_{i+1} = \sqrt{1-\beta_i}X_i + \sqrt{\beta_i}N_i$ where $N_i \sim N(0,1)$, $\mathbb{E}[X_{i+1}] = 0$, and $\mathrm{Var}[X_{i+1}] = 1$.

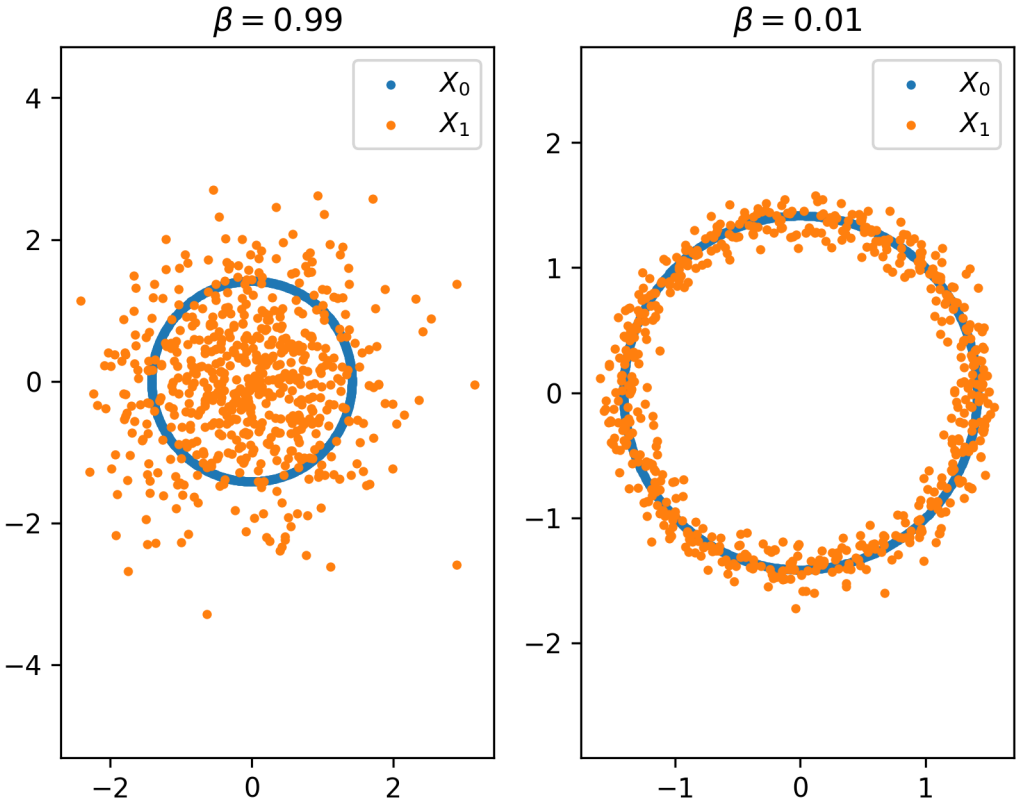

Figure 26.9: A two-dimensional example showing the effect of attenuation. X_0 are the points (x_0, y_0) on a circle centered at origin with radius $\sqrt{2}$. X_1 are the points (x_1, y_1) satisfying $x_1 = \sqrt{1-\beta}x_0 + \sqrt{\beta}n_x$ and $y_1 = \sqrt{1-\beta}y_0 + \sqrt{\beta}n_y$ where $n_x \sim N(0,1)$ and $n_y \sim N(0,1)$ are standard normal noise. On the left, we have $\beta = 0.99$. On the right, we have $\beta = 0.01$.