DIFFUSION-BASED GENERATIVE AI

FIRAS RASSOUL-AGHA[1]

FIGURE 1. Paintings of a mini Schnauzer in Dalístyle. The first two were generated using https://openart.ai/, the next two were generated using https://gemini.google.com/, and the last two used https://chatgpt.com/. The third and last images were generated with the extra prompt to have a labrador reflection.

CONTENTS

Date: April 22, 2025.
Key words and phrases. Artificial Intelligience, AI, Diffusion, Machine Learning, Ohrenstein-Uhlenbeck Process, Stochastic Differential Equation.
[1]Department of Mathematics, University of Utah.

1. Introduction

In these notes, I give a quick basic overview of how diffusion-based generative AI works.

The goal will be to generate a picture based on a prompt. Think specifically of a prompt that says "generate a picture that looks like a Dalípainting."

A picture can be encoded as a matrix. The width k and height ℓ of the picture give, respectively, the number of horizontal and vertical lines. The picture then has $k\ell$ pixels. If the picture is gray-scale, then each pixel is just some real number that indicates the "amount of black" in that pixel. If it is a colored picture, then each pixel consists of three real numbers, indicating the amounts of red, green, and blue in the pixel. Each color can be given by a real number or by an integer, e.g. between 0 and 255, or more, if we want more colors. Thus, a picture can in fact be encoded as a real-valued vector in $\{0, \cdots, 255\}^{3k\ell}$ or, more generally, in $\mathbb{R}^{3k\ell}$. Note that $3k\ell$ is typically a very large number. E.g. a 256×256 RGB image will have $k = 256$ lines (vertical pixels) and $\ell = 256$ pixels across (horizontal pixels), totaling 65,536 pixels and, with colors, the vector would have 196,608 dimensions. Similarly, a 512×512 RGB image would have 786,432 dimensions, a 1024×1024 RGB image has 3,145,728 dimensions, and a picture with 4K resolution has $k = 2160$, $\ell = 3840$, and hence $3k\ell$ is about 25 million[1].

Suppose we have all of Dalí's paintings stored and encoded into vectors as above. To generate a random Dalí, we could simply pick one of these paintings (equally likely). But this would only give us a random Dalípainting. What we want is to generate a painting that is not necessarily in the collection that we have, but that art experts would tell us the painting may well have been painted by Dalíhimself.

To achieve this, we imagine that Dalíwas a random generator of paintings. Suppose that he was a machine with a button and that every time we press the button, the machine generates a random vector in $\mathbb{R}^{3k\ell}$. (In mathematical terms, we have an $\mathbb{R}^{3k\ell}$-valued random variable.) What we are after, then, is guessing what the probability density function (pdf) of this machine (or random variable). Once we have that, we can ourselves generate random vectors from that pdf and we would be generating random paintings that are indistinguisheable from Dalí's.

The problem is that estimating a pdf of a random vector from a space with such a high dimension is a very difficult problem and requires an impossible number of samples. So, instead, we will try to devise an algorithm that considers the existing Dalípaintings as samples and directly generates a new random variable from the pdf, wihtout really estimating the pdf itself.

In a sense, picking a sample at random (which is called Boostrapping) is one way to achieve this. However, as mentioned above, this will not give us new samples. We would like to have an algorithm that gives us new samples that look quite different from our existing samples, but that still come from the same pdf as our samples. That is: we want to generate genuinely new Dalípaintings!

To summarize, here is the problem we want to solve:

The Problem: we are given a set of samples produced by a machine that generates random images, and our task is to use these samples to construct another machine whose output cannot be reliably distinguished from that of the original, even by statistical tests. In other words, any observer–even one with access to many outputs–should not be able to tell which machine produced which image with better-than-chance accuracy.

Here is the general idea of the proposed solution.

[1] 25 million dimensions is quite large even for the methods we will discuss in this note. There are ways to deal with this, though. One method consists of first generating a lower resolution image and then upscaling the resolution of the generated image using smoothing procedures.

We begin by constructing a device that takes an image as input and outputs a new random image. The rules governing how this output is generated are entirely up to us. In mathematical terms, this device defines a Markov process on the space of images and deciding on what its transition probabilities are is completely up to us.

Now, consider the following game: when Dalí's machine generates a random image, we feed it into our device and observe the output. Because we designed the device ourselves, we know its transition probabilities – the rules by which it maps one image to another. This means that if we knew the pdf of Dalí's original images, we could compute the pdf of the output of this game using the law of total probability.

But here's the interesting part: Bayes' rule allows us to reverse this process. That is, we can design a new device – a "backward" device – that takes as input an image produced by the game above and outputs a new image whose distribution exactly matches that of Dalí's original machine.

At first glance, this might seem circular or even pointless. To compute the pdf of the game's final output, we need to know Dalí's pdf. And to apply Bayes' rule to build the reverse device, we again need Dalí's pdf! So how does this help?

We resolve the first issue by repeatedly running our device. Specifically, once Dalí's painting has been processed by our device, we feed the resulting image back into the device, and repeat this many times. The theory of Markov processes tells us that if we choose the transition probabilities of our device carefully, then the distribution of the output will converge to a stationary distribution – the invariant measure of the Markov chain – which depends only on the transition rules of our device, not on the initial input. So, even without knowing Dalí's pdf, we can sample from this limiting distribution and use it as a proxy when running the reverse game!

To address the second issue – that Bayes' rule still requires Dalí's pdf – we take advantage of the fact that we have multiple samples. Suppose we have M samples from Dalí's machine. For each one, we play the game by running it through our device repeatedly N times, yielding M sequences of N images each. We then reverse each of these sequences and use them to estimate the transition probabilities of the reverse process – effectively learning the dynamics of our backward device from data.

Putting all this together, we arrive at an algorithm that allows us to generate new samples resembling those of Dalí's machine:

The algorithm: First, sample an initial image from the invariant measure of our forward device. Next, pass this image through the reverse process N times, using the transition probabilities estimated from data. The final output of this reversed sequence is a new random image that closely approximates a true Dalísample – in the sense that its distribution is statistically close to the original.

Now that we gave a big-picture description of the approach, let us close this introduction by going back and comparing this seemingly roundabout method to the idea of using the given samples to try and directly estimate the underlying pdf.

As mentioned above, the main issue is the high dimensionality of the state space: A 256×256 RGB image lives in a 196,608-dimensional space and – even worse – a 4K colored image lives in a 25 million-dimensional space! Accurately estimating the density over such a space, without any understanding of the structural properties of the density function, is extremely challenging. One complication is that the support of the pdf (the subset of the state space where the pdf is non-zero) is a tiny, complex, highly non-linear manifold in that space – images do not evenly fill the space. Another serious complication is that normalization is hard: An important step in estimating a pdf is to normalized it to something that integrates to one. That means computing an integral over the humongous state space – computationally intractable.

Instead of modeling the full pdf, we learn how to step toward regions of higher probability. This is an easier task because, for example, it requires estimating gradients that are more local and structured and does not require a global normalization.

2. A baby version of generative AI

To explain the idea of the algorithm in more mathematical terms, we will begin with an example on a much smaller state space.

Suppose that instead of having the large state space $\mathbb{R}^{3k\ell}$, we have only three possible outcomes: A, B, and C. Some machine is generating these outcomes at random, with probabilities $\mathbb{P}(A) = \alpha$, $\mathbb{P}(B) = \beta$, and $\mathbb{P}(C) = \gamma$. Here, α, β, γ are nonnegative numbers that add up to one and that are unkown to us.

In this case, the task of figuring out the pdf boils down to figuring out the numbers α, β, and γ, given a number of samples. This is a fairly simple task: look in the sample at the frequencies of A's, B's, and C's, and compute the ratios. E.g., if we are given $1,000$ samples and we see that there are 335 A's, 523 B's, and 142 C's, then we estimate that $\alpha = 0.335$, $\beta = 0.523$, and $\gamma = 0.142$. Now that we know these probabilities, we can generate our own samples from $\{A, B, C\}$.

However, as explained in the introduction, we will not be following the above procedure when the state space is much larger than $\{A, B, C\}$. The larger the size of the sample space is, the more data we need to estimate the probabilities of the various outcomes in the state space. And when the state space is $\mathbb{R}^d$, things get exponentially worse as the dimension d increases. So when d is as large as $200,000$ (in the case of a 256×256 RGB image) or 25 million (as in the example of a 4K color picture), running the above procedure becomes impossible (at least given the current computational power).

Therefore, we will not attempt to estimate α, β, and γ. Instead, we will try to develop an algorithm that uses the provided samples to generate new samples from (approximately) the same probability mass function (pmf) as the original samplest. In other words, we want to generate random samples from $\{A, B, C\}$ that have approximately the probabilities α, β, and γ, but without really learning the values of α, β, and γ.

The main idea will be to take each sample and add a lot of noise to it, generating this way a bunch of noisy versions of our original samples. Then we solve an inverse problem: now that we have noisy versions of our samples, can we denoise them and figure out the original samples? Of course, the answer is NO. We have added noise and removing the noise completely will not be possible. But the added noise will actually still play in our favor! Reversing the process by attempting to remove the noise will give "new initial samples". These are the inputs we are after! That is, these new samples will be new samples (i.e. not simply resampled from our original samples), but will have a pdf that is close to the one our original samples came from.

Suppose we denote the initial distribution by the row vector $\Phi_0 = [\alpha\ \beta\ \gamma]$ and suppose we denote the initial sample by X_0. This means that X_0 is a random variable with values in $\{A, B, C\}$ and that $\mathbb{P}(X_0 = A) = \alpha$, $\mathbb{P}(X_0 = B) = \beta$, and $\mathbb{P}(X_0 = C) = \gamma$.

To add noise to X_0, we run a Markov chain on the state space $\{A, B, C\}$. The transition matrix of this chain is up to us to choose. We denote it by $\overrightarrow{P}$. We are using a "right-arrow" above the P to remind us that this is the transition matrix of the "forward" Markov chain. We will shortly reverse the course of time and construct a "backward" Markov chain.

We denote the state after n steps by X_n and the pmf of this state by the row-vector Φ_n. Thus, $\mathbb{P}(X_n = A) = \Phi_n(A)$, $\mathbb{P}(X_n = B) = \Phi_n(B)$, and $\mathbb{P}(X_n = C) = \Phi_n(C)$. We have

$$\Phi_n = \Phi_{n-1}\overrightarrow{P} \quad \text{and} \quad \Phi_n = \Phi_0 \overrightarrow{P}^n. \tag{2.1}$$

Suppose now we are actually given Φ_n and we would like to devise an algorithm to generate X_0 with pmf given by Φ_0. We will do this recursively by first generating X_{n-1} with pmf Φ_{n-1} and then repeating the process until we get to X_0.

For a Markov chain, it is simpler to generate X_n given X_{n-1}. To generate X_{n-1}, given X_n, we need to use the Bayes rule. That is, we write

$$\mathbb{P}(X_{n-1}=j \mid X_n=i)=\frac{\mathbb{P}(X_{n-1}=j, X_n=i)}{\mathbb{P}(X_n=i)}=\frac{\mathbb{P}(X_n=i \mid X_{n-1}=j)\mathbb{P}(X_{n-1}=j)}{\mathbb{P}(X_n=i)}=\frac{\overrightarrow{P}_{j,i}\Phi_{n-1}(j)}{\Phi_n(i)}.$$

That is, if we are at state i at time n, then the probability we are at state j at time $n-1$ is given by

$$\overleftarrow{P}_{n,i,j}=\frac{\overrightarrow{P}_{j,i}\Phi_{n-1}(j)}{\Phi_n(i)}. \tag{2.2}$$

Thus, we see that when the Markov chain is reversed, we get again a Markov chain. However, since the dependence on the time n does not disappear in the formula (2.2), the time-homogenous forward Markov chain becomes an inhomogenous Markov chain when reversed.

Let us run a sanity check here and make sure that $\overleftarrow{P}_{n,i,j}$ is actually a pmf. For this, we add over all states j and get

$$\sum_j \overleftarrow{P}_{n,i,j}=\sum_j \frac{\overrightarrow{P}_{j,i}\Phi_{n-1}(j)}{\Phi_n(i)}=\frac{\sum_j \Phi_{n-1}(j)\overrightarrow{P}_{j,i}}{\Phi_n(i)}=\frac{(\Phi_{n-1}\overrightarrow{P})_i}{\Phi_n(i)}=\frac{\Phi_n(i)}{\Phi_n(i)}=1.$$

Expalanation: for the second equality, we factored out the common facter $\Phi_n(i)$ (since we are adding over j, not over i), for the third equality we observed that the sum of $\Phi_{n-1}(j)\overrightarrow{P}_{j,i}$ is precisely the i-th entry of the row vector $\Phi_{n-1}\overrightarrow{P}$, and then we used (2.1) to get the second-to-last equality.

Note now that if X_n is chosen according to the mgf given by Φ_n, then X_{n-1} will have mgf Φ_{n-1}. Indeed,

$$\begin{aligned}\mathbb{P}(X_{n-1}=j)&=\sum_i \mathbb{P}(X_{n-1}=j \mid X_n=i)\mathbb{P}(X_n=i)=\sum_i \frac{\overrightarrow{P}_{j,i}\Phi_{n-1}(j)}{\Phi_n(i)}\cdot\Phi_n(i)\\&=\sum_i \overrightarrow{P}_{j,i}\Phi_{n-1}(j)=\Phi_{n-1}(j)\sum_i \overrightarrow{P}_{j,i}=\Phi_{n-1}(j).\end{aligned}$$

Explanation: the first equality applies the law of total probability, the second applies (2.2) and the assumption that X_n has mgf Φ_n, then $\Phi_n(i)$ is cancelled out, $\Phi_{n-1}(j)$ is factored out, and lastly we use the fact that the rows of $\overrightarrow{P}$ add up to one.

The upshot of the above computation is that if we do know Φ_n, then we can generate X_n from this mgf and run the backward Markov chain with (time-dependent) transition probabilities given by (2.2), and once we get to X_0, it will be generated from precisely the mgf Φ_0.

The problem with the above is that to know Φ_n we need to know Φ_0, as is clear from (2.1). But without knowing Φ_n, we cannot generate X_n, nor can we compute $\overleftarrow{P}_{n,i,j}$ using (2.2).

So far, the choice of n was immaterial. That is, the above story would be the same whether we use $n=1$ or $n=1,000,000$. However, we do know that if a Markov chain runs for a long time, then the mgf of X_n will approach a stationary distribution. If we choose $\overrightarrow{P}$ so that our whole state space $\{A,B,C\}$ is the unique communicating class, e.g. if we choose the matrix to have all positive entries, then the matrix $\overrightarrow{P}$ will have a unique left-eigenvector Φ_∞ that has nonnegative entries that add up to one. Furthermore, we know these entries will in fact be all positive and that as $n\to\infty$, $\Phi_0\overrightarrow{P}^n$ will converge to Φ_∞, irrespective of the values of the entries of Φ_0 (as long as they add up to one, of course).

This gives us a solution to our first problem: even though we do not know the value of Φ_n to sample X_n, we can work with a large enough n and sample X_n from Φ_∞ instead. Note here that Φ_∞ only depends on $\overrightarrow{P}$, which was chosen by us and so now we do not need to know Φ_n to generate our samples X_n.

Of course, this will mean that X_n will not be sampled from exactly Φ_n, but from a distribution that is close to Φ_n. So, if everything else works out, we will not get to generate X_0 from exactly Φ_0, but we can hope that our procedure will generate X_0 from a distribution that is close enough to Φ_0 for us to not really see a difference. This should be acceptable, since Dalíwasn't really a random vector generator and didn't really have a pdf. This is only our modeling process. Plus, even if we were trying to solve the problem of guessing the pdf, we were only going to get an estimate of the pdf, based on the samples we have. Thus, the best we can hope for anyway is to manage to generate random variables from a pdf that is close to whatever the true pdf is.

Now that we know how to generate X_n, we are faced with the second problem: how do we run the backward Markov chain, if we do not know Φ_n and hence cannot compute the transition probabilities in (2.2)?

This is where statistical estimation comes in or, as it is called nowadays: statistical learning or machine learning. Here, one estimates the transition probabilities $\overleftarrow{P}_{n,i,j}$, from the available data, instead of computing them explicitly in terms of the unknown Φ_0. To do this, one posites that $\overleftarrow{P}_{n,i,j}$ has a certain form, $p_\theta(n,i,j)$, where θ represents a small number of parameters, and once we know θ, p_θ is a function of n, i, and j, of some given form. The task is then to estimate the function p_θ.

The above is then similar to the classical parameter estimation problem. To think of a familiar classical example, imagine we have independent samples from a normal distribution. We are after figuring out the pdf of the random variables, but we know that the pdf has the special form $f_\theta(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, where here $\theta = (\mu, \sigma^2)$ are the two unkown parameters. We seek to estimate the mean μ and the variance σ^2 from the samples we have. We know in this case that the maximum likelihood estimator of the two parameters is a very good solution to the problem and it tells us that μ can be estimated by the sample mean and σ^2 by the sample variance.

The problem we have is similar. From (2.1), we have that

$$\begin{aligned}
\Phi_{n-1}(A) &= \alpha \overrightarrow{P}^{n-1}_{A,A} + \beta \overrightarrow{P}^{n-1}_{B,A} + \gamma \overrightarrow{P}^{n-1}_{C,A},\\
\Phi_{n-1}(B) &= \alpha \overrightarrow{P}^{n-1}_{A,B} + \beta \overrightarrow{P}^{n-1}_{B,B} + \gamma \overrightarrow{P}^{n-1}_{C,B},\\
\Phi_{n-1}(C) &= \alpha \overrightarrow{P}^{n-1}_{A,C} + \beta \overrightarrow{P}^{n-1}_{B,C} + \gamma \overrightarrow{P}^{n-1}_{C,C},\\
\Phi_{n}(A) &= \alpha \overrightarrow{P}^{n}_{A,A} + \beta \overrightarrow{P}^{n}_{B,A} + \gamma \overrightarrow{P}^{n}_{C,A},\\
\Phi_{n}(B) &= \alpha \overrightarrow{P}^{n}_{A,B} + \beta \overrightarrow{P}^{n}_{B,B} + \gamma \overrightarrow{P}^{n}_{C,B}, \text{ and}\\
\Phi_{n}(C) &= \alpha \overrightarrow{P}^{n}_{A,C} + \beta \overrightarrow{P}^{n}_{B,C} + \gamma \overrightarrow{P}^{n}_{C,C}.
\end{aligned}$$

Then (2.2) gives us

$$\overleftarrow{P}_{n,i,j} = \frac{a_{n,i,j}\alpha + b_{n,i,j}\beta + c_{n,i,j}\gamma}{d_{n,i}\alpha + e_{n,i}\beta + f_{n,i}\gamma}, \tag{2.3}$$

where $a_{n,,j}$, $b_{n,i,j}$, $c_{n,i,j}$, $d_{n,i}$, $e_{n,i}$, and $f_{n,i}$ are quantities that we can explicitly compute from our transition matrix $\overrightarrow{P}$. For example, to get $b_{n,A,C}$, we look at the coefficient in front of β in the numerator of (2.2), with $i = A$ and $j = C$. That is, we look at

$$\overrightarrow{P}_{C,A}\Phi_{n-1}(C) = \overrightarrow{P}_{C,A}\big(\alpha \overrightarrow{P}^{n-1}_{A,C} + \beta \overrightarrow{P}^{n-1}_{B,C} + \gamma \overrightarrow{P}^{n-1}_{C,C}\big)$$

and take $b_{n,C,A}$ to be the coefficient in front of β, i.e.,

$$b_{n,A,C} = \overrightarrow{P}_{C,A}\overrightarrow{P}^{n-1}_{B,C}.$$

Similarly, to get $f_{n,B}$, we look at the denominator of (2.2), with $i = B$, and take $f_{n,B}$ to be the coefficient in front of γ. Thus, we consider

$$\Phi_n(B) = \alpha\overrightarrow{P}^n_{A,B} + \beta\overrightarrow{P}^n_{B,B} + \gamma\overrightarrow{P}^n_{C,B}$$

and take

$$f_{n,B} = \overrightarrow{P}^n_{C,B}.$$

Long story short, $\overleftarrow{P}_{n,i,j}$ turns out to be simply a function of three parameters, α, β, and γ. Specifically, its the ratio of two linear combinations of these parameters. We can at this point work out a maximum likelihood estimator to estimate these parameters.

More precisely, first, take N be a large integer that ensures that Φ_N is quite close to Φ_∞. How large N should be is solely dependent on the matrix $\overrightarrow{P}$, because $\overrightarrow{P}^N$ converges to the matrix whose rows are all equal to Φ_∞ and so we can choose N large enough so that $\overrightarrow{P}^N$ is close to that matrix and this choice does not involve the value of Φ_0.

Next, we are given a number of samples of X_0 (think of Dalípaintings). Denote these by $X_0^{(1)}, X_0^{(2)}, \cdots, X_0^{(M)}$. For each sample, run the Markov chain with transition matrix $\overrightarrow{P}$ for N steps. Denote the result of running $X_0^{(1)}$ for n steps by $X_n^{(1)}$ and, similarly, for $X_0^{(2)}$, $X_0^{(3)}$, etc. For each integer n between 1 and N consider the paired samples $(X_n^{(1)}, X_{n-1}^{(1)}), \cdots, (X_n^{(M)}, X_{n-1}^{(M)})$. This is the data to be used to calculate $\overleftarrow{P}_{n,i,j}$ for $i, j \in \{A, B, C\}$. Since this quantity depends on three parameters, α, β, γ, we can compute the maximum likelihood estimator and estimate these parameters, solving our problem of estimating $\overleftarrow{P}_{n,i,j}$.

HOWEVER, doing things as described above assumes we know the actual form of the thing we are trying to estimate, i.e. we know the formula (2.3), i.e. that $\overleftarrow{P}_{n,i,j}$ is a ratio of two linear functions of α, β, and γ. Even worse, it assumes that the parameters we are trying to estimate are exactly the entries of Φ_0. Thus, at the end of the day, we are doing something that is essentially the same as estimating α, β, and γ by looking at the frequency of A's, B's, and C's in our samples of X_0. This is something that we are trying to avoid, since it would not be feasable in the practical situation where we have a very large sample space. Furthermore, estimating these parameters, in the large state space case, would be equivalent to estimating the pdf of X_0, and computing the maximum likelihood estimator for such a problem is computationally very expensive, which is why we said in the introduction that this is a very hard problem and is not how we should approach things.

But if we pretend we do not know the formula (2.3), then what form should we postulate for $\overleftarrow{P}_{n,i,j}$, and what parameters should it depend on?

We could assume that $\overleftarrow{P}_{n,i,j}$ depends linearly or polynomially on a reasonable number of parameters and run a linear or polynomial regression to estimate the parameters. However, from the actual form of (2.3), we see that neither linear nor polynomial regression would do a good job. The maximum likelihood estimator is out of question because we do not know what parameters we should be using to even compute the likelihood function, let alone maximizing it over these unknown parameters.

In fact, asking the above question is a little bit silly in this simple model, since we already know that the real parameters are α, β, and γ, and that the best solution is to just estimate them from the frequencies of A's, B's, and C's in our samples of X_0. However, the non-linear form of the function in (2.3) almost points the way to the method that is used in the bigger, more realistic,

problem. There, we will see that this estimation step is done using a sort of non-linear regression method that attempts to mimic the way neural networks work in the brain.

Let us summarized what we have learned so far about the generative AI algorithm.

We are given a number of samples $X_0^{(1)}, \cdots, X_0^{(M)}$. We choose a transition matrix $\overrightarrow{P}$ that has an invariant measure Φ_∞ and we pick an integer N that is large enough so that the Markov chain at time N is very close to its invariant measure. We now add noise to the initial samples by running the Markov chain forward, from these initial values. This gives us the random variables $X_1^{(1)}, X_2^{(1)}, \cdots, X_N^{(1)}$, from the first sample, $X_1^{(2)}, X_2^{(2)}, \cdots, X_N^{(2)}$, from the second sample, and so on. We now use these samples to estimate (learn) the transition probabilities $\overleftarrow{P}_{n,i,j}$ of the backward Markov chain. (This is the step that is done using the machine learning algorithm.) Once we learned these transition probabilities, we can release the software to the public, for we have achieved our goal and can perform the task we were after.

To generate a new sample from the (unkown to us) mgf Φ_0, we generate an X_N from the mgf Φ_∞. (Note that we know both N and Φ_∞, because they only depend on our choice of $\overrightarrow{P}$.) We then run the reverse Markov chain using our estimates of the transition probabilities $\overleftarrow{P}_{n,i,j}$. Once we get to $n = 0$, X_0 would be the sample we spit out. Its mgf will not be exactly Φ_0, but it will be close enough to Φ_0 for X_0 to look like it was generated from the same pdf that gave the training samples $X_0^{(1)}, \cdots, X_0^{(M)}$.

3. SDE-based generative AI

Now, we describe the real algorithm. As before, we are given a number of random samples of X_0. These are now vectors in the large state space $\mathbb{R}^{3k\ell}$, where $3k\ell$ is somewhere between 200,000 and 25 million. Instead of adding noise to our samples by running a Markov chain, we will run a stochastic differenital equation (SDE) with the initial condition $X(0)$. Specifically, we will run the Ornstein-Uhlenbeck (OU) process, which is defined as the solution to the following stochastic differential equation (SDE):

$$dX(t) = -\frac{1}{2}\beta(t)X(t)\,dt + \sqrt{\beta(t)}\,dB(t). \tag{3.1}$$

Here, B is a standard Brownian motion in $\mathbb{R}^{3k\ell}$, meaning it consists of $3k\ell$ independent one-dimensional standard Brownian motions. In class, we focused on one-dimensional SDEs, but the theory extends directly to higher dimensions.

In equation (3.1), the term $-\frac{1}{2}\beta(t)X(t)dt$ represents the drift, which contracts $X(t)$ toward zero at a rate controlled by $\beta(t)$. The term $\sqrt{\beta(t)}\,dB(t)$ is the diffusion, which injects random noise into the system. The choice of $\sqrt{\beta(t)}$ will help having an explicit formula when we compute the distribution of $X(t)$ below.

The function $\beta(t)$ is called the noise schedule. It controls both the rate at which the process decays toward zero and the amount of noise added at each moment in time. Typically, $\beta(t)$ is chosen so that the decay rate is small at the beginning (when $t \approx 0$) and increases as t approaches the terminal time T.

A common choice is a linear schedule:

$$\beta(t) = \beta_{\min} + (\beta_{\max} - \beta_{\min})\frac{t}{T}\,,$$

where $\beta_{\min}$ and $\beta_{\max}$ are positive constants. For example, typical values are $\beta_{\min} = 0.01$ and $\beta_{\max} = 5$. This means the decay rate starts near 0.01 and increases to around 5 by time T.

Other commonly used schedules include cosine schedule

$$\beta(t) = 1 - \cos\Big(\frac{\pi}{2} \cdot \frac{t}{T}\Big)$$

and quadratic schedule

$$\beta(t) = \beta_{\min} + (\beta_{\max} - \beta_{\min})\Big(\frac{t}{T}\Big)^2.$$

Each choice of $\beta(t)$ leads to different behavior in the forward diffusion process and can impact the quality of results in downstream generative tasks. While other SDEs are also used in practice, we will focus here on the one given in (3.1).

The SDE (3.1) has an explicit solution given by

$$X(t) = X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr} + \int_0^t e^{-\frac{1}{2}\int_r^t \beta(u)\,du}\sqrt{\beta(r)}\,dB(r). \tag{3.2}$$

The first term on the right-hand side of (3.2) says that $X(t)$ decays to 0 exponentially fast, as expected. The second term is a stochastic integral, which is hence a martingale and represents the added noise.

Note that the initial condition is easily seen to be satisfied because plugging in $t = 0$ in (3.2) sets the integrals $\int_0^t$ to 0 and we get $X(0)$ on the right-hand side. Let us check that the process given in (3.2) does solve the equation (3.1).

Differentiating the first term on the right-hand side of (3.2) gives

$$d\Big(X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}\Big) = X(0)\Big(-\frac{1}{2}\beta(t)\Big)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}\,dt. \tag{3.3}$$

To differentiate the second term we first pull out the dependence on t from inside the integral by first writing the integral $\int_r^t$ as $\int_0^t - \int_0^r$ and then using the fact that the exponential of a sum is the product of exponentials. Hence, we have

$$\begin{aligned}\int_0^t e^{-\frac{1}{2}\int_r^t \beta(u)\,du}\sqrt{\beta(r)}\,dB(r) &= \int_0^t e^{-\frac{1}{2}\int_0^t \beta(u)\,du}\, e^{\frac{1}{2}\int_0^r \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\\ &= e^{-\frac{1}{2}\int_0^t \beta(u)\,du}\int_0^t e^{\frac{1}{2}\int_0^r \beta(u)\,du}\sqrt{\beta(r)}\,dB(r).\end{aligned}$$

Now we can use the product rule, keeping in mind that, by definition,

$$d\Big(\int_0^t e^{\frac{1}{2}\int_0^r \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big) = e^{\frac{1}{2}\int_0^t \beta(u)\,du}\sqrt{\beta(t)}\,dB(t).$$

Thus, we get that

$$\begin{aligned}d\Big(\int_0^t e^{-\frac{1}{2}\int_r^t \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big) &= d\Big(e^{-\frac{1}{2}\int_0^t \beta(u)\,du}\int_0^t e^{\frac{1}{2}\int_0^r \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big)\\
&= \Big(\int_0^t e^{\frac{1}{2}\int_0^r \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big)d\Big(e^{-\frac{1}{2}\int_0^t \beta(u)\,du}\Big) + e^{-\frac{1}{2}\int_0^t \beta(u)\,du}\,d\Big(\int_0^t e^{\frac{1}{2}\int_0^r \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big)\\
&= \Big(\int_0^t e^{\frac{1}{2}\int_0^r \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big)\Big(-\frac{1}{2}\beta(t)\,e^{-\frac{1}{2}\int_0^t \beta(u)\,du}\Big)dt + e^{-\frac{1}{2}\int_0^t \beta(u)\,du}\,e^{\frac{1}{2}\int_0^t \beta(u)\,du}\sqrt{\beta(t)}\,dB(t)\\
&= -\frac{1}{2}\beta(t)\Big(\int_0^t e^{-\frac{1}{2}\int_0^t \beta(u)\,du}e^{\frac{1}{2}\int_0^r \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big)dt + \sqrt{\beta(t)}\,dB(t)\\
&= -\frac{1}{2}\beta(t)\Big(\int_0^t e^{-\frac{1}{2}\int_r^t \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big)dt + \sqrt{\beta(t)}\,dB(t).\end{aligned} \tag{3.4}$$

Adding (3.3) and (3.4) together, we get that

$$\begin{aligned} dX(t) &= X(0)\big(-\frac{1}{2}\beta(t)\big)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}\,dt - \frac{1}{2}\beta(t)\Big(\int_0^t e^{-\frac{1}{2}\int_r^t \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big)dt + \sqrt{\beta(t)}\,dB(t) \\ &= -\frac{1}{2}\beta(t)\Big(X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr} + \int_0^t e^{-\frac{1}{2}\int_r^t \beta(u)\,du}\sqrt{\beta(r)}\,dB(r)\Big)\,dt + \sqrt{\beta(t)}\,dB(t) \\ &= -\frac{1}{2}\beta(t)X(t)dt + \sqrt{\beta(t)}\,dB(t). \end{aligned}$$

We have shown that $X(t)$ indeed solves (3.1).

We can in fact figure out the distribution of $X(t)$, given the value of $X(0)$. For this, recall that if $0 = t_0 < t_1 < \cdots < t_{n-1} < t_n = t$ is a discretization of the interval $[0,t]$, then the random variables $B(t_{i+1}) - B(t_i)$ are independent centered normal random variables. They are in fact $3k\ell$-dimensional normal random variables, with entries that are themselves independent and have variance $t_{i+1} - t_i$. Thus, $B(t_{i+1}) - B(t_i)$ are centered multivariate normal random variables with variance-covariance matrices given by $(t_{i+1} - t_i)$ times the identity matrix (with $3k\ell$ rows and columns). If g is a deterministic function, then $g(t_i)(B(t_{i+1}) - B(t_i))$ are again independent centered multivariate normal random variables, but now with variance-covariance matrices given by $g(t_i)^2(t_{i+1} - t_i)$ times the identity matrix. Their sum is then a centered multivariate normal random variable with a variance-covariance matrix equal to $\sum_{i=0}^{n-1} g(t_i)^2(t_{i+1} - t_i)$ times the identity matrix. If g is continuous, then as we make the discretization finer, $\sum_{i=0}^{n-1} g(t_i)(B(t_{i+1}) - B(t_i))$ converges to $\int_0^t g(r)\,dB(r)$, while $\sum_{i=0}^{n-1} g(t_i)^2(t_{i+1} - t_i)$ converges to $\int_0^t g(r)^2\,dr$. Therefore, we see that $\int_0^t g(r)\,dB(r)$ must be a multivariate normal random variable with mean 0 and a variance-covariance matrix given by $\int_0^t g(r)^2\,dr$ times the identity matrix.

Apply this to get that the stochastic integral term in (3.2) is in fact a multivariate normal random variable with mean 0 and a variance-covariance matrix given by the identity matrix times

$$\begin{aligned} \int_0^t \Big(e^{-\frac{1}{2}\int_r^t \beta(u)\,du}\sqrt{\beta(r)}\Big)^2\,dr &= \int_0^t e^{-\int_r^t \beta(u)\,du}\beta(r)\,dr = e^{-\int_0^t \beta(u)\,du}\int_0^t e^{\int_0^r \beta(u)\,du}\beta(r)\,dr \\ &= e^{-\int_0^t \beta(u)\,du}\int_0^t \Big(e^{\int_0^r \beta(u)\,du}\Big)'\,dr = e^{-\int_0^t \beta(u)\,du}e^{\int_0^r \beta(u)\,du}\Big|_0^t \\ &= e^{-\int_0^t \beta(u)\,du}\Big(e^{\int_0^t \beta(u)\,du} - 1\Big) = 1 - e^{-\int_0^t \beta(u)\,du}. \end{aligned}$$

Note how the choice of $\sqrt{\beta(t)}$ in the diffusion term allowed us do have a perfect derivative under the integral on the second line of the above display.

The first term on the right-hand side of (3.2) is deterministic and so it only changes the mean of the normal random variable. Thus, we conclude that $X(t)$ it a multivariate normal random variable with mean $X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}$ and a variance-covariance matrix equal to the identity matrix times $1 - e^{-\int_0^t \beta(u)\,du}$. Note that as $t \to \infty$, the mean converges to 0 and $1 - e^{-\int_0^t \beta(u)\,du}$ converges to 1. This means that $X(t)$ converges to a standard $3k\ell$-dimensional normal random variable. This is the invariant measure of the OU process and this is the distribution we will draw from when we start the reverse process.

Next, we need to work out how to run the OU process in reverse, from time T back to time 0. Note at this point that one does not really need the full construction of stochastic integrals to define the stochastic integral in (3.2), nor does one need the full stochastic calculus to verify that $X(t)$ in (3.2) satisfies (3.1). Indeed, if g is a continuously differentiable deterministic function, then

we can use integration by parts to write

$$\int_0^t g(r)\,dB(r) = g(t)B(t) - \int_0^t g'(r)B(r)\,dr$$

and turn the stochastic integral into a regular integral. However, to construct the reverse process, one does need to have developed stochastic integrals, stochastic calculus, and, in particular, derived Itô's formula. This was first achieved in a paper titled "Reverse-time diffusion equation models" by Brian Anderson back in 1982.

We begin with a derivation of a partial differential equation (PDE) that the pdf of $X(t)$ satisfies. We will do this for a more genral diffusion process than the one we have in (3.1). So suppose $X(t)$ solves

$$dX(t) = f(t, X(t))dt + g(t, X(t))dB(t),$$

where $f(t,x)$ is valued in $\mathbb{R}^{3k\ell}$ and $g(t,x)$ is real-valued. Let $p(t,x)$ denote the pdf of $X(t)$. Thus, if h is a function from $\mathbb{R}^{3k\ell}$ to $\mathbb{R}$, then

$$\mathbb{E}[h(X_t)] = \int_{-\infty}^{\infty} h(x)p(t,x)\,dx.$$

As we have seen in class, Itô's formula tells us that

$$\begin{aligned} dh(X(t)) &= h'(X(t))dX(t) + \frac{1}{2}h''(X(t))(dX(t))^2 \\ &= h'(X(t))f(t,X(t))dt + h'(X(t))g(t,X(t))dB(t) \\ &\qquad + \frac{1}{2}h''(X(t))(f(t,X(t))dt + g(t,X(t))dB(t))^2. \end{aligned}$$

Expanding the square on the right-hand side gives terms that have $(dt)^2$, $dt\,dB(t)$, and $(dB(t))^2$. The first two can be ignored because $(dt)^2$ and $dt\,dB(t)$ are much smaller than dt. But $(dB(t))^2$ gets replaced by dt (heuristically, because $dB(t)$ is like a centered normal random variable with variance dt and hence its size is of order $\sqrt{dt}$, which when squared becomes dt.) Therefore, we get

$$dh(X(t)) = \big(h'(X(t))f(t,X(t)) + \frac{1}{2}h''(X(t))g(t,X(t))^2\big)dt + h'(X(t))g(t,X(t))dB(t). \tag{3.5}$$

This would be the right formula if X and B were one-dimensional. Since we have vector-valued processes, we have to repeat the above computation using partial derivatives instead. We also need to use the fact that the coordinates of B are independent of each other and hence when we expand the square term, we also get to ignore terms that have $dB_i(t)dB_j(t)$, with $i \neq j$. (Here, we are using the notation $B(t) = (B_1(t), \ldots, B_{3k\ell}(t))$ for the various coordinates of $B(t)$.) The end result is then that

$$dh(X(t)) = \big(\nabla h(X(t)) \cdot f(t,X(t)) + \frac{1}{2}\Delta h(X(t))g(t,X(t))^2\big)dt + g(t,X(t))\nabla h(X(t)) \cdot dB(t).$$

Here, $\nabla h(x)$ is the gradient of h, which is the vector made up of the partial derivatives of h with respect to the various coordinates of x. (Recall that x is a vector in $\mathbb{R}^{3k\ell}$.) Then $\nabla h(X(t))\cdot f(t,X(t))$ is the scalar product of the two vectors, $\nabla h(X(t))$ and $f(t,X(t))$ and $\nabla h(X(t))\cdot dB(t)$ is the scalar product of the two vectors, $\nabla h(X(t))$ and $dB(t)$. On the other hand, $\Delta h(x)$ is the Laplacian of h, calculated at x. That is, if we write $x = (x_1, \ldots, x_{3k\ell})$, then

$$\Delta h(x) = \sum_{i=1}^{3k\ell} \frac{\partial^2 h}{\partial x_i^2}(x).$$

Thus, $\Delta h(x)$ is just a number, not a vector.

The formula (3.5) really means that

$$h(X(t))-h(X(0)) = \int_0^t \nabla h(X(r))\cdot f(r,X(r))\,dr + \int_0^t \frac{1}{2}\Delta h(X(r))g(r,X(r))^2\,dr + \int_0^t g(r,X(r))\nabla h(X(r))\cdot dB(r).$$

Take an expected value of both sides and recall that the mean of a stochastic integral is zero. So we get

$$\mathbb{E}[h(X(t))] - \mathbb{E}[h(X(0))] = \int_0^t \mathbb{E}\big[\nabla h(X(r))\cdot f(r,X(r))\big]\,dr + \frac{1}{2}\int_0^t \mathbb{E}\big[\Delta h(X(r))g(r,X(r))^2\big]\,dr.$$

In the above calculation, we moved the $\mathbb{E}$ inside the integral. This step usually needs a careful analysis because there are conditions one needs to check, but I will ignore this technical issue and I will just say here that this step does work out as intended.

Now we take a derivative of the above in t:

$$\frac{d}{dt}\mathbb{E}[h(X(t))] = \mathbb{E}\big[\nabla h(X(t))\cdot f(t,X(t))\big] + \frac{1}{2}\mathbb{E}\big[\Delta h(X(t))g(t,X(t))^2\big].$$

Next, we use the fact that $X(t)$ has pdf $p(t,x)$ and replace the expected values with integrals against this pdf:

$$\frac{d}{dt}\int_{-\infty}^{\infty} h(x)p(t,x)\,dx = \int_{-\infty}^{\infty} \nabla h(x)\cdot f(t,x)\,p(t,x)\,dx + \frac{1}{2}\int_{-\infty}^{\infty} \Delta h(x)g(t,x)^2\,p(t,x)\,dx.$$

Now, we move the time derivative under the integral. The idea is that an integral is a limit of a sum (think Riemann sums) and the derivative of a sum is the sum of derivatives. There are technical details to verify, of course, because this logic works only when we have a finite sum and here we are taking a limit with more and more terms in the Riemann sum. But I will sweep the technical details under the rug and will write

$$\int_{-\infty}^{\infty} h(x)\frac{\partial p}{\partial t}(t,x)\,dx = \int_{-\infty}^{\infty} \nabla h(x)\cdot f(t,x)\,p(t,x)\,dx + \frac{1}{2}\int_{-\infty}^{\infty} \Delta h(x)g(t,x)^2\,p(t,x)\,dx. \tag{3.6}$$

Now, we use integration by parts to move the derivatives from being on h to being on the other functions in the integrals on the right-hand side.

To this end, consider a real-valued function a and a vector-valued function b. Then

$$\begin{aligned}\int_{-\infty}^{\infty} \nabla a(x)\cdot b(x)\,dx &= \sum_{i=1}^{3k\ell}\int_{-\infty}^{\infty}\frac{\partial a}{\partial x_i}(x)b_i(x)\,dx\\ &= -\sum_{i=1}^{3k\ell}\int_{-\infty}^{\infty} a(x)\frac{\partial b_i}{\partial x_i}(x)\,dx\\ &= -\int_{-\infty}^{\infty} a(x)\nabla\cdot b(x)\,dx,\end{aligned}$$

where for a vector-valued function b, $\nabla\cdot b$ denotes what is called the divergence of b:

$$\nabla\cdot b = \sum_{i=1}^{3k\ell}\frac{\partial b_i}{\partial x_i}.$$

(For a real-valued function a, its gradient ∇a is a vector. On the other hand, for a vector-valued function b, its divergence $\nabla\cdot b$ is a number.)

One thing to be careful about is that in the above integration by parts formula, we did not have the usual boundary terms $a(x)b_i(x)\Big|_{-\infty}^{\infty}$. This is OK if say a decays to 0 both when $x\to\infty$ and when $x\to-\infty$, because then the boundary terms are 0.

Suppose we also have a real-valued function c. Then its gradient ∇c is a vector-valued function. Apply the above integration by pats formula with $b = \nabla c$ to get

$$\int_{-\infty}^{\infty} \nabla a(x) \cdot \nabla c(x)\, dx = -\int_{-\infty}^{\infty} a(x) \nabla \cdot \nabla c(x)\, dx.$$

Note now that $\nabla \cdot \nabla c$ is exactly Δc. Therefore, we have shown that

$$\int_{-\infty}^{\infty} a(x) \Delta c(x)\, dx = -\int_{-\infty}^{\infty} \nabla a(x) \cdot \nabla c(x)\, dx.$$

Now notice that the integral on the right-hand side is symmetric in a and c. So, exchanging the roles of a and c, we also have

$$\int_{-\infty}^{\infty} c(x) \Delta a(x)\, dx = -\int_{-\infty}^{\infty} \nabla c(x) \cdot \nabla a(x)\, dx = -\int_{-\infty}^{\infty} \nabla a(x) \cdot \nabla c(x)\, dx.$$

The last two displays now say that

$$\int_{-\infty}^{\infty} c(x) \Delta a(x)\, dx = \int_{-\infty}^{\infty} a(x) \Delta c(x)\, dx.$$

Applying this formula and the integration by parts formula we have

$$\int_{-\infty}^{\infty} \nabla h(x) \cdot f(t,x) p(t,x)\, dx = -\int_{-\infty}^{\infty} h(x) \nabla \cdot \big(f(t,x) p(t,x)\big)\, dx$$

and

$$\int_{-\infty}^{\infty} \Delta h(x) g(t,x)^2 p(t,x)\, dx = \int_{-\infty}^{\infty} h(x) \Delta\big(g(t,x)^2 p(t,x)\big)\, dx.$$

Going back to (3.6) we get

$$\int_{-\infty}^{\infty} h(x) \frac{\partial p}{\partial t}(t,x)\, dx = -\int_{-\infty}^{\infty} h(x) \nabla \cdot \big(f(t,x) p(t,x)\big)\, dx + \frac{1}{2} \int_{-\infty}^{\infty} h(x) \Delta\big(g(t,x)^2 p(t,x)\big)\, dx.$$

Rearranging, we get

$$\int_{-\infty}^{\infty} h(x) \Big[\frac{\partial p}{\partial t}(t,x) + \nabla \cdot \big(f(t,x) p(t,x)\big) - \frac{1}{2} \Delta\big(g(t,x)^2 p(t,x)\big)\Big]\, dx = 0.$$

This formula is valid for any function h for which we can check the conditions that are necessary for swapping $\mathbb{E}$ and $\int_0^t$ and then moving the time derivative under the integral. Also, we need that h vanishes as $x \to \infty$ and as $x \to -\infty$ so that the boundary terms in the integration by parts computations vanish. It turns out that there are in fact enough such functions to deduce that the only way the above integral would be 0 is by having the quantity in square brackets be equal to zero. (Roughly speaking, we can find enough functions h to approximate the quantity in brackets. This allows us to replace h with the quantity in brackets, in the above formula. But then what we have is that the integral of the square of the quantity in brackets is zero, which means the quantity in brackets must be equal to 0.)

We have derived the PDE that the pdf p satisfies:

$$\frac{\partial p}{\partial t}(t,x) = \frac{1}{2} \Delta\big(g(t,x)^2 p(t,x)\big) - \nabla \cdot \big(f(t,x) p(t,x)\big). \tag{3.7}$$

This is called Kolmogorov's forward equation. It is also sometimes known as the Fokker-Planck equation. Note that it is a heat equation, since we have one time derivative on the left and a space Laplacian on the right. But it also has a drift term $-\nabla \cdot (f(t,x)p(t,x))$.

To apply it to the OU process (3.1), we use $f(t,x) = -\frac{1}{2}\beta(t)x$ and $g(t,x) = \sqrt{\beta(t)}$. Then Plugging this into (3.7), we get Kolmogorov's equation for the pdf of the OU process:

$$\frac{\partial p}{\partial t}(t,x) = \frac{1}{2}\beta(t)\Delta p(t,x) + \frac{1}{2}\beta(t)\nabla\cdot(xp(t,x)).$$

Now we are ready to derive the SDE for the reverse OU process. For this, let $Y(t) = X(T-t)$, for $t \in [0,T]$. This is the reverse-time process we are after. Let $\alpha(t) = \beta(T-t)$ and let $q(t,x) = p(T-t,x)$.

q satisfies a PDE that we can derive from the PDE that p satisfies. Namely, first apply the chain rule to get

$$\frac{\partial q}{\partial t}(t,x) = -\frac{\partial p}{\partial t}(T-t,x).$$

Then use Kolmogrov's equation for p to continue

$$= -\frac{1}{2}\beta(T-t)\Delta p(T-t,x) - \frac{1}{2}\beta(T-t)\nabla\cdot\big(xp(T-t,x)\big).$$

Since the space-derivatives of $p(T-t,x)$ and of $xp(T-t,x)$ are the same as those of $q(t,x)$ and $xq(t,x)$, respectively, we can continue with

$$= -\frac{1}{2}\alpha(t)\Delta q(t,x) - \frac{1}{2}\alpha(t)\nabla\big(xq(t,x)\big).$$

Therefore,

$$\frac{\partial q}{\partial t}(t,x) = -\frac{1}{2}\alpha(t)\Delta q(t,x) - \frac{1}{2}\alpha(t)\nabla\cdot\big(xq(t,x)\big). \tag{3.8}$$

An argument that uses martingales and that I will skip here shows that $Y(t)$ must satisfy an SDE and, more importantly, that its diffusion term has to equal $\sqrt{\alpha(t)}d\overline{B}(t)$, where $\overline{B}$ is a standard Brownian motion. So the claim that I am making without showing its proof is that the diffusion coefficient does not change when we go backward in time and all that changes is the drift coefficient. Let us go with that and figure out what the drift coefficient should be.

Thus, we will postulate that

$$dY(t) = f(t,Y(t))dt + \sqrt{\alpha(t)}d\overline{B}(t)$$

and we want to figure out the formula for f. Since $q(t,x)$ is the pdf of $Y(t)$, we can apply (3.7) with $g(t,x) = \sqrt{\alpha(t)}$ and write

$$\frac{\partial q}{\partial t}(t,x) = \frac{1}{2}\alpha(t)\Delta q(t,x) - \nabla\cdot\big(f(t,x)q(t,x)\big).$$

This and (3.8) imply that

$$-\frac{1}{2}\alpha(t)\Delta q(t,x) - \frac{1}{2}\alpha(t)\nabla\cdot\big(xq(t,x)\big) = \frac{1}{2}\alpha(t)\Delta q(t,x) - \nabla\cdot\big(f(t,x)q(t,x)\big).$$

From this, we get

$$\nabla\cdot\big(f(t,x)q(t,x)\big) = \alpha(t)\Delta q(t,x) + \frac{1}{2}\alpha(t)\nabla\cdot\big(xq(t,x)\big).$$

Recalling that $\Delta q = \nabla\cdot\nabla q$ we can rewrite the above as

$$\nabla\cdot\big(f(t,x)q(t,x)\big) = \nabla\cdot\Big(\alpha(t)\nabla q(t,x) + \frac{1}{2}\alpha(t)xq(t,x)\Big).$$

This leads to the equation

$$f(t,x)q(t,x) = \alpha(t)\nabla q(t,x) + \frac{1}{2}\alpha(t)xq(t,x).$$

Dividing by q and noting that $\frac{\nabla q}{q} = \nabla \log q$ we get that

$$f(t,x) = \alpha(t)\nabla \log q(t,x) + \frac{1}{2}\alpha(t)x.$$

Therefore, the reverse OU process $Y(t)$ satisfies the SDE

$$dY(t) = \frac{1}{2}\alpha(t)\big(Y(t) + 2\nabla \log q(T-t, Y(t))\big)\,dt + \sqrt{\alpha(t)}\,d\overline{B}(t).$$

Returning to the original notation with β and p instead of α and q, we get

$$dY(t) = \frac{1}{2}\beta(T-t)\big(Y(t) + 2\nabla \log p(T-t, Y(t))\big)\,dt + \sqrt{\beta(T-t)}\,d\overline{B}(t). \tag{3.9}$$

We now see that the role of $\overleftarrow{P}_{n,i,j}$ in (2.2) is played here by $\nabla \log p(T-t,x)$. This is called the "score function." Heuristically speaking, this drift term guides the process along the gradient of the log-likelihood function $\log p(T-t,x)$, as it goes backward in time. And the math shows that this drift is exactly what is needed to ensure the backward process matches the forward process in terms of its pdf at every moment in time.

Thus, the next order of business is to estimate the score function $s(t,x) = \nabla \log p(t,x)$. Once we have an estimate $\hat{s}$ of the function s, we can generate samples of $X(0)$ as follows.

The algorithm: Generate a standard $3k\ell$-dimensional normal random variable $\hat{Y}(0)$. Next, use this as the initial condition and numerically solve the SDE

$$d\hat{Y}(t) = \frac{1}{2}\beta(T-t)\big(\hat{Y}(t) + 2\hat{s}(T-t, \hat{Y}(t))\big)\,dt + \sqrt{\beta(T-t)}\,d\overline{B}(t) \tag{3.10}$$

up to time T. Give $\hat{Y}(T)$ as the outcome.

The upshot is that these samples will have a pdf that is close $p(0,x)$, the pdf of $X(0)$.

Note: in (3.10), we used the symbol $\hat{Y}$ instead of Y to disntinguish between the SDE (3.9) that uses $\nabla \log p(t,x)$ in its drift and the SDE (3.10) that uses the approximation $\hat{s}(t,x)$ that is learned from the data.

4. Learning the score function

To estimate (or "learn") the gradient of the score function $s(t,x) = \nabla \log p(t,x)$, we first derive a formula for this function that makes it rather clear how things should go.

First, let us recall that if we know $X(0) = a$, then (3.2) says that $X(t)$ is a multivariate normal random variable with mean $ae^{-\frac{1}{2}\int_0^t \beta(r)\,dr}$ and a variance-covariance matrix $\big(1 - e^{-\int_0^t \beta(r)\,dr}\big)I$. In particular, since $X(t)$ is normal, its coordinates are jointly normal random variables and since the off-diagonal entries of the variance-covariance matrix are zeros, the coordinates of $X(t)$ are uncorrelated (they have zero covariance), which, because the coordinates are jointly normal, implies they are independent. Furthermore, since the diagonal entries of the variance-covariance matrix are all equal to $1 - e^{-\int_0^t \beta(r)\,dr}$, we deduce that the coordinates of $X(t)$ all have the same variance $1 - e^{-\int_0^t \beta(r)\,dr}$. Their means, however, may differ: the i-th coordinate of $X(t)$ has mean equal to the i-th cooridinate of $ae^{-\frac{1}{2}\int_0^t \beta(r)\,dr}$.

Denote the j-th coordinates of $X(t)$ by $X_j(t)$ and let a_j denote the j-th coordinate of a. The pdf of $X_j(t)$, as a function of the variable x_j, is then given by

$$\frac{1}{\sqrt{2\pi\big(1 - e^{-\int_0^t \beta(r)\,dr}\big)}}\exp\Big\{-\frac{\big(x_j - a_j e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}\big)^2}{2\big(1 - e^{-\int_0^t \beta(r)\,dr}\big)}\Big\}.$$

Since the coordinates are independent, the pdf $p(t, x \mid 0, a)$ of $X(t)$ (given $X(0) = a$) is the product of the above pdfs. The first term $1/\sqrt{\cdots}$ is the same in all these pdfs and thus gets raised to the power $3k\ell$, which is the number of coordinates. The product of the exponential functions can be written as an exponential of a sum and, therefore, we have

$$p(t, x \mid 0, a) = \frac{1}{\Big[\sqrt{2\pi\big(1 - e^{-\int_0^t \beta(r)\,dr}\big)}\Big]^{3k\ell}} \exp\Big\{-\sum_{j=1}^{3k\ell} \frac{\big(x_j - a_j e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}\big)^2}{2\big(1 - e^{-\int_0^t \beta(r)\,dr}\big)}\Big\}.$$

Taking a log on both sides we get

$$\log p(t, x \mid 0, a) = -\log\Big[\sqrt{2\pi\big(1 - e^{-\int_0^t \beta(r)\,dr}\big)}\Big]^{3k\ell} - \sum_{j=1}^{3k\ell} \frac{\big(x_j - a_j e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}\big)^2}{2\big(1 - e^{-\int_0^t \beta(r)\,dr}\big)}.$$

The first term on the right-hand side only depends on t and so its derivative with respect to x_i vanishes. When we take derivatives of the terms inside the sum, with respect to x_i, they all vanish, except for the term $j = i$, whose derivative follows the power rule and equals

$$\frac{2(x_i - a_i e^{-\frac{1}{2}\int_0^t \beta(r)\,dr})}{2(1 - e^{-\int_0^t \beta(r)\,dr})} = \frac{x_i - a_i e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}}{1 - e^{-\int_0^t \beta(r)\,dr}}.$$

The vector that has these coordinates is (conveniently!)

$$\frac{x - a e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}}{1 - e^{-\int_0^t \beta(r)\,dr}}.$$

Therefore, we have shown that

$$\nabla \log p(t, x \mid 0, a) = -\frac{x - a e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}}{1 - e^{-\int_0^t \beta(r)\,dr}}. \tag{4.1}$$

We are, however, after a formula for $\nabla \log p(t, x)$. We know from the law of total probability that

$$p(t, x) = \int_{-\infty}^{\infty} p(t, x \mid 0, a) p(0, a)\,da. \tag{4.2}$$

However, the log function is not a linear function and so

$$\log p(t, x) = \log \int_{-\infty}^{\infty} p(t, x \mid 0, a) p(0, a)\,da \neq \int_{-\infty}^{\infty} \big(\log p(t, x \mid 0, a)\big) p(0, a)\,da.$$

So how do we make use of (4.1)?

Here is how:

$$\begin{aligned}
\nabla \log p(t,x) &= \frac{\nabla p(t,x)}{p(t,x)} = \frac{\nabla \int_{-\infty}^{\infty} p(t,x \mid 0,a)p(0,a)\,da}{p(t,x)} \\
&= \frac{\int_{-\infty}^{\infty} \nabla p(t,x \mid 0,a)p(0,a)\,da}{p(t,x)} \\
&= \int_{-\infty}^{\infty} \nabla p(t,x \mid 0,a) \cdot \frac{p(0,a)}{p(t,x)}\,da \\
&= \int_{-\infty}^{\infty} \frac{\nabla p(t,x \mid 0,a)}{p(t,x \mid 0,a)} \cdot \frac{p(t,x \mid 0,a)p(0,a)}{p(t,x)}\,da \\
&= \int_{-\infty}^{\infty} \frac{\nabla p(t,x \mid 0,a)}{p(t,x \mid 0,a)} \cdot p(0,a \mid t,x)\,da \\
&= \int_{-\infty}^{\infty} \big(\nabla \log p(t,x \mid 0,a)\big)p(0,a \mid t,x)\,da \\
&= \int_{-\infty}^{\infty} \frac{ae^{-\frac{1}{2}\int_0^t \beta(r)\,dr} - x}{1 - e^{-\int_0^t \beta(r)\,dr}}\,p(0,a \mid t,x)\,da \\
&= \mathbb{E}\Big[\frac{X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr} - X(t)}{1 - e^{-\int_0^t \beta(r)\,dr}} \,\Big|\, X(t) = x\Big]. \qquad (4.3)
\end{aligned}$$

Explanation: the first equality is expressing the derivative of the log of a function as being the ratio of the derivative of the function and the function itself, the second equality used (4.2), the third equality brought the derivative inside the integral (and, as we have done before, ignored the details that make that possible), the fourht equality brought the denominator inside the integral, the fifth equality used the common math trick of multiplying by 1! (more precisely, we multiplied and divided by $p(t,x \mid 0,a)$), the sith equality used Bayes' rule to recognize that $\frac{p(t,x|0,a)p(0,a)}{p(t,x)} = p(0,a \mid t,x)$, the seventh equality recognized that the ratio of the derivative of a function by the function itself is the derivative of the log of the function, the second-to-last equality used (4.1), and the last equality interpreted the integration over a against $p(0,a \mid t,x)$ as a conditional expected value of a function of $X(0)$, given the information that $X(t) = x$.

Note how in the above computation, we did make use of (4.1), after some manipulations, and, more importantly, that we used the Bayes' rule. So we see that the Bayes' rule is indeed underlying the reversal of the stochastic process, just like it was the main idea in the baby example of Section 2. It was more visible in the Markov chain example and is rather hidden in the real example we are treating here.

It is important to note that in (4.3), $X(t)$ is considered to be known (equal to x) and the expected value is on the random variable $X(0)$. This does go with the intuition that we are computing a quantity that is needed for reversing the course of time for the stochastic process X. Also, note that the terms involving integrals of β are deterministic and known to us. So the quantity inside the expected value is linear in X_0. In other words, the problem of computing $s(t,x) = \nabla \log p(t,x)$ really boils down to computing $u(t,x) = \mathbb{E}\big[X(0) \mid X(t) = x\big]$, the average of $X(0)$ given $X(t) = x$. Observe how this should be a simpler problem than to estimate the pdf of $X(0)$: estimating an average is ought to be easier than estimate the full distribution!

So, we essentially are after estimating the conditional expectation $u(t,x) = \mathbb{E}\big[X(0) \mid X(t) = x\big]$, using samples of $X(0)$. To achieve this, we use the following property of conditional expectations: $u(t,x) = \mathbb{E}\big[X(0) \mid X(t) = x\big]$ is the function that minimizes $\mathbb{E}[||X(0) - u(t,X(t))||^2]$. Here, the expectation $\mathbb{E}$ is over both $X(0)$ and $X(t)$, i.e. neither $X(0)$ nor $X(t)$ are considered to be given

and they are both averaged out. Consequently,

$$s(t,x) = \nabla \log p(t,x) = \mathbb{E}\Big[\frac{X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr} - X(t)}{1 - e^{-\int_0^t \beta(r)\,dr}} \,\Big|\, X(t) = x\Big]$$

is the function that minimizes

$$\mathbb{E}\Big[\Big|\Big|\frac{X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr} - X(t)}{1 - e^{-\int_0^t \beta(r)\,dr}} - s(t,X(t))\Big|\Big|^2\Big]. \tag{4.4}$$

Again, here, both $X(0)$ and $X(t)$ are being averaged out.

With the above minimization property, the problem of estimating the score function s from samples of $X(0)$ becomes a regression problem. I will explain this in steps.

1. Choose the type of regression probelm.

First, we postulate that the approximation of $s(t,x) = \nabla \log p(t,x)$ will come from a certain family of functions $s_\theta : [0,T] \times \mathbb{R}^{3k\ell} \to \mathbb{R}^{3k\ell}$, with θ being a vector of parameters that parametrize the family. We then need to choose a good family of estimation functions, rich enough to get us very close to the true function s we want to estimate. Then, the main task would be use the available data to estimate θ so that we use the corresponding s_θ in place of the unknown s.

A familiar such estimation problem is linear or polynomial regression: If we believe $s(t,x)$ is close to being linear or polynomial in t and the coordinates of x, then we can consider the family of linear or polynomial functions and θ would be the vector that gives the coefficients of the linear or polynomial function. For example, if we only had a two-dimensional problem and so $x = (x_1, x_2)$ and we decided a linear s is good enough, then $s_\theta(t,x)$ (which must also have two coordinates) would be given by $a + bt + cx_1 + dx_2$ as the first coordinate and $a' + b't + c'x_1 + d'x_2$ as the second coordinate. Here, $\theta = (a, b, c, d, a', b', c', d')$ is an eight-dimensional vector. Since in reality x has $3k\ell$ entries, we would need $3k\ell \times (3k\ell + 2)$ parameters to do a full linear regression. A polynomial regression will need even more parameters, depending on the degree of the polynomial we choose to use.

The problem with the above is that s is unlikely to be linear or polynomial in either t or x. The machine learning method suggests a different (neither linear nor polynomial) form for the functions s_θ that attempts to mimic the way neural connections are believed to work in the brain. I will give a brief description of a basic version of this approach at the end of this section.

2. The regression problem.

Suppose we settled on a family of functions s_θ, parametrized by some vector θ. How do we estimate the values of the entries of θ that make the estimated function s_θ closest to the true function s? The answer hides in the fact that s minimizes (4.4).

We are given M random samples of $X(0)$ that we denote by $a^{(1)}, \ldots, a^{(M)}$. Sample times $t^{(1)}, \ldots, t^{(M)}$ independently and uniformly on the interval $[0,T]$. For each $i \in \{1, \ldots, M\}$, sample a multivariante normal $x^{(i)}$, with mean $a^{(i)}e^{-\frac{1}{2}\int_0^{t^{(i)}} \beta(r)\,dr}$ and a variance-covariance matrix $\big(1 - e^{-\int_0^{t^{(i)}} \beta(r)\,dr}\big)I$. This means the coordinates of $x^{(i)}$ are sampled as independent normal random variables with the same variance $1 - e^{-\int_0^{t^{(i)}} \beta(r)\,dr}$ and such that the j-th coordinate of $x^{(i)}$ has mean $a^{(i)}_j e^{-\frac{1}{2}\int_0^{t^{(i)}} \beta(r)\,dr}$, where $a^{(i)}_j$ is the j-th coordinate of $a^{(i)}$. Let

$$s^{(i)} = \frac{a^{(i)}e^{-\frac{1}{2}\int_0^{t^{(i)}} \beta(r)\,dr} - x^{(i)}}{1 - e^{-\int_0^{t^{(i)}} \beta(r)\,dr}}.$$

Then the fact that the true function s minimizes (4.4) tells us that the optimal θ is the one that minimizes the so-called Quadratic Loss Function

$$L(\theta) = \sum_{i=1}^{M} ||s_\theta(t^{(i)}, x^{(i)}) - s^{(i)}||^2. \tag{4.5}$$

3. Minimize the Quadratic Loss Function L.

There are many numerical methods to minimize $L(\theta)$. A very widely used one is "gradient descent". To get the idea, first think of a function of one variable. That is, think of θ as being real-valued. Visualize the graph of L, which is what the point $(\theta, L(\theta))$ would trace if you vary θ. This graph sometimes goes up and sometimes goes down. So it has "mountains" and "valeys" and our task is to locate its deepest valey (the place where L is minimal). We know that when $L'(\theta) > 0$, the function will increase if we move θ to the right, i.e. in the positive direction or, in the direction $L'(\theta)$ is pointing to. Similarly, L will decrease if we move θ to the left, i.e. in the direction $-L'(\theta)$ is pointing to. The same happens if $L'(\theta) < 0$. Thus, we see that if we move from θ in the direction given by $-L'(\theta)$, then the value of L will always go down. A similar story happens if you think of θ as being two-dimensional. In this case, $(\theta, L(\theta))$ will trace a surface, as you vary the two-dimensional θ. Again, we have mountains and valleys, and now may also have saddles (a structure where the surface is concave in one direction and convex in the other - like a horse saddle). Here, the thing that plays the role of the derivative of L is its gradient $\nabla L(\theta)$. This is here a two-dimensional vector and if move from any θ in the direction of $-\nabla L(\theta)$, then L would always go down. You can now imagine that dimension two is not special and this fact holds in all dimensions. The method of "gradient descent" thus goes like this: We start at some "good guess" θ_0, then move to a new value θ_1 such that $\theta_1 = \theta_0 - \gamma_0 \nabla L(\theta_0)$. Here, $\gamma_0 > 0$ indicates how far we move from θ_0 (roughly speaking, how fast you should run downhill). The process is then repeated, i.e. we move from θ_1 to a new $\theta_2 = \theta_1 - \gamma_1 \nabla L(\theta_1)$, and so on. I will not explain in this note how one chooses the starting guess θ_0, nor how one determines the values of $\gamma_0, \gamma_1, \ldots$. That said, these are very important problems. Choose the γ's too small and the method moves in very increments and, as a result, the convergence to the minimum will be very slow. Choose them too large and you will overshoot the minimum and ruin the convergence (think of driving downhill so fast that you are not able to stop in time and end up on the other side of the hill!). The next paragraph explains more about the importance of choosing an appropriate starting point θ_0.

Although gradient descent looks neat and seems intuitive, it does have a big potential problem! If we always go down the hill, then we should indeed be able to locate the bottom of a valley. However, if our initial value θ_0 happens to be in a valley that is not the deepest valley, then we will be stuck in the shallower valley and will never get out of it to find the deeper one. This is fixed by instead using the method called "stochastic gradient descent". Here, when we move from a value of θ, the new value is chosen to go in the direction of $-\nabla L(\theta)$ with some high probability, but it can also go in the direction of $\nabla L(\theta)$ with some small probability. Then, this process most often goes to regions that lower L, but every now and then, it climbs back up instead of going down the hill. It turns out that this probabilistic modification of the original method allows the process to always exist any valley, until it finds the deepest one and then it will eventually end up near the bottom of that valley, finding the value of θ that (globally) minimizes L.

It is noteworthy that the high number of coordinates of θ (typically of at least order $3k\ell$, which is about $200,000$ for a 256×256 RGB image) makes L a function of a large number of variables and as such, it will typically have a huge number of local valleys. This makes it unclear how fast even the stochastic graduent descent method is able to find the deepest valley and settle on the optimal θ.

In short, we have only skimmed over the general idea and there is a large body of mathematical research that is being developped and that still needs to be developped to fully understand how exactly things work.

I close this section by briefly explaining the machine learning method.

4. Estimating the score function s via Machine Learning.

To simplify the notation, we think of t as just another variable and so we think of $s(t,x)$ as a function $s(u)$, where $u \in [0,T] \times \mathbb{R}^{3k\ell}$.

In one of the simplest machine learning settings, we use the family of functions

$$s_\theta(u) = W_2(W_1 u + b_1)^+ + b_2. \tag{4.6}$$

Here, u is viewed as a column vector, W_1 is an $m \times (3k\ell+1)$ matrix, where m is an integer that is usually taken to be much smaller than $3k\ell$, W_2 is a $3k\ell \times m$ matrix, b_1 is an m-vector, and b_2 is a vector with $3k\ell$ entries. This way, the matrix multiplications make sense (recall that u has $3k\ell+1$ entries) and $s_\theta(u)$ ends up having $3k\ell$ entries. The function that is applied to W_1u+b_1 is called "the activation function". Here, we used the function x^+. If x is a real number, then the function x^+ is simply the maximum of 0 and x: when x is positive, the function simply returns x, but when x becomes negative, the function cuts it off and replaces it by a 0. If x is a vector, then x^+ means we apply this function to each coordinate. In the machine learning literature, this particular function is called the Rectified Linear Unit function and the notation for it is $\mathrm{ReLU}(x) = x^+$. Other non-linear activation functions are also sometimes used in place of ReLU.

Note how if we did not use the activation function, then $W_2(W_1u+b_1)+b_2$ can be rewritten as $(W_2W_1)u + (W_2b_1+b_2)$, which is of the form $Wu+b$ and we would then be just doing a linear regression. So it is the activation function that creates the nonlinearity in s_θ.

Including t as just another variable is often too simple. Instead, to reflect a more complex dependence on time, one uses what is called a "time embedding". A simple way to do this is to use Fourier modes. That is, instead of taking $u=(t,x)$, one takes

$$u = \big(\sin(2\pi t/T), \cos(2\pi t/T), \dots, \sin(2\pi Lt/T), \cos(2\pi Lt/T), x\big),$$

where L is the desired number of modes, e.g. $L=10$. This increases the dimension of u from $3k\ell+1$ to $3k\ell+2L$. Then W_1 needs to be an $m \times (3k\ell+2L)$ matrix.

The parameters of the model (i.e. the entries of θ) are precisely the entries of the matrices W_1 and W_2 and the vectors b_1 and b_2. So these matrices and vectors are what we want to estimate. Consequently, we have here $(3k\ell+1)m + 3k\ell m + m + 3k\ell = 3k\ell(2m+1)+2m$ parameters to estimate ($3k\ell(2m+1)+(2L+1)m$, if using a time embedding). Even though this is a large number of parameters, this is still an easier problem to handle than the problem of estimating the pdf $p(0,x)$ of $X(0)$ directly (e.g. we do not have to normalize things to integrate to one).

Summary. Now one can use (4.6) to compute the loss function $L(\theta)$ in (4.5). We apply, for example, the stochastic gradient descent method to find the optimal θ that minimizes this L. Using this optimal θ, we now can approximate the true score function $s(t,x) = \nabla \log p(t,x)$ by $\hat{s}(t,x) = s_\theta(t,x)$. Now that we learned an approximation of $s(t,x)$, we can produce as many new images as we wish. Namely, we run the reverse OU process $\hat{Y}$ (i.e. numerically solve the SDE (3.10)) from its starting point $\hat{Y}(0)$, chosen as a multivariate standard normal, to its terminal point $\hat{Y}(T)$, which presumably will have a pdf that is close to that of $X(0)$.

Before closing the section, let us touch on the connection with neural networks.

(4.6) is what is called a two-layer fully-connected neural network. The input layer is the vector u, which you should think of as $3k\ell+1$ neurons, each sending a signal to another (hidden) layer

of m neurons. The m neurons process the signals from the input layer and each of these neurons sends out a signal. This is modeled by $W_1u + b_1$, where W_1 summarizes the weights that the m neurons put on the various signals coming from the input neurons and b_1 models the biases that the intermediate neurons add to the signals. The neurons also apply a non-linear step, modeled here by the ReLU function. Thus, the output of the m neurons is in fact given by $(W_1u + b_1)^+$. These signals are sent to a second layer of $3k\ell$ neurons that apply their own matrix and bias vector and end up outputing $W_2(W_1u + b_1)^+ + b_2$.

One can add more layers to enlarge the family of estimation functions s_θ, much like increasing the degree in polynomial regression expands the space of approximating functions.

5. Incorporating a prompt

A prompt is a chunk of text that provides guidance on what should be generated. For example, the prompt "a painting of a dog in the style of Salvador Dalí" suggests generating an image of a dog with characteristics reminiscent of Dalí's artistic style.

To work with such prompts computationally, we represent them as sets of tags, such as "painting", "dog", and "Salvador Dalí". In fact, for our purposes, we will treat a prompt and a set of tags as equivalent. Just as a prompt can be broken down into individual tags, you do not need to use a single coherent text as your prompt and can specify the prompt directly as a collection of tags from the outset.

This prompt–or more precisely, this set of tags–is then converted into a numerical representation: a high-dimensional vector, which we denote as π. This encoding process is analogous to how an image is transformed into a vector x, although we will not touch on the details here.

Our goal is to generate an image vector that corresponds to a given prompt vector. To formalize this, we define the state space as the set of pairs (x, π). We assume a probability distribution function (pdf) $p(0, x, \pi)$ over this space. This pdf is not known to us and the task is to develop an algorithm that, given a prompt π, samples an image x from the conditional distribution $p(0, x \mid \pi)$. In other words, we aim to generate an image that is consistent with the given prompt.

The procedure with a prompt is almost identical to the case without one, but we must carefully adjust the problem setup. Previously, in the forward diffusion process, we assumed the initial state $X(0)$ had an unknown pdf denoted by $p(0, x)$. Now, however, we introduce an additional random variable: the prompt Π. At time $t = 0$, we consider the joint distribution of the pair $(X(0), \Pi)$, which is governed by the joint pdf $p(0, x, \pi)$.

Importantly, noise is added only to the image variable, not to the prompt. That is, the forward stochastic process remains the same Ornstein-Uhlenbeck (OU) process $X(t)$ solving the SDE in equation (3.1). The prompt Π remains fixed throughout the entire process.

As we have already seen, for sufficiently large T, the distribution of $X(T)$ approaches a standard normal in $\mathbb{R}^{3k\ell}$. Therefore, when initializing the backward process to generate new samples, we continue to sample the starting point $Y(0)$ from a $3k\ell$-dimensional standard normal distribution.

The key difference now lies in the drift term of the reverse-time SDE (3.9). Instead of using the gradient $\nabla \log p(t, x)$, we must now use the conditional gradient $\nabla \log p(t, x \mid \pi)$, where $p(t, x \mid \pi)$ denotes the conditional pdf of $X(t)$ given the prompt $\Pi = \pi$. That is, the SDE for the backward process is now

$$dY(t) = \frac{1}{2}\beta(T-t)\big(Y(t) + 2\nabla \log p(T-t, Y(t) \mid \pi)\big)\, dt + \sqrt{\beta(T-t)}\, d\overline{B}(t). \tag{5.1}$$

The computation that led to (4.3) goes the same way, except that we now replace the fact that

$$p(t,x) = \int_{-\infty}^{\infty} p(t,x \mid 0,a)p(0,a)\,da$$

with the fact that

$$p(t,x \mid \pi) = \int_{-\infty}^{\infty} p(t,x \mid 0,a)p(0,a \mid \pi)\,da.$$

Thus, $p(0,a)$ is replaced by $p(0,a \mid \pi)$ and the change appears when we use the Bayes formula:

$$\begin{aligned}
\frac{p(t,x \mid 0,a)p(0,a \mid \pi)}{p(t,x \mid \pi)} &= \frac{p(t,x \mid 0,a)p(0,a,\pi)/p(\pi)}{p(t,x,\pi)/p(\pi)} \\
&= \frac{p(t,x \mid 0,a)p(0,a,\pi)}{p(t,x,\pi)} \\
&= \frac{p(t,x \mid 0,a,\pi)p(0,a,\pi)}{p(t,x,\pi)} \\
&= \frac{p(\pi,0,a,t,x)}{p(t,x,\pi)} \\
&= p(0,a \mid \pi,t,x).
\end{aligned}$$

Explanation: The first equality follows from the definition of conditional pdfs, where $p(\pi)$ denotes the pdf of the random variable Π. The second equality results from canceling out the common factor $p(\pi)$. The third equality relies on the fact that the evolution of the Markov process is independent of the prompt–i.e., the distribution of $X(t)$ depends only on the initial condition $X(0) = a$, regardless of whether we condition on $\Pi = \pi$ or not. The third and fourth equalities again apply the definition of conditional pdfs (with $p(\pi,0,a,t,x)$ denoting the joint pdf of of the triple $(\Pi, X(0), X(t))$).

With the above change, we get a formula for the conditional score function:

$$s(t,x \mid \pi) = \nabla \log p(t,x \mid \pi) = \mathbb{E}\Big[\frac{X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr} - X(t)}{1 - e^{-\int_0^t \beta(r)\,dr}} \,\Big|\, X(t) = x, \Pi = \pi\Big].$$

As before, this implies that $s(t,x \mid \pi)$ is the function that minimizes

$$\mathbb{E}\Big[\Big|\Big|\frac{X(0)e^{-\frac{1}{2}\int_0^t \beta(r)\,dr} - X(t)}{1 - e^{-\int_0^t \beta(r)\,dr}} - s(t,X(t) \mid \Pi)\Big|\Big|^2\Big]. \tag{5.2}$$

Now, all three random variables $(\Pi, X(0), X(t))$ are being averaged out.

Note how now the function we need to learn is a function of three variables, t, x, and also π. As before, we postulate that we are working with a given family of estimators $s_\theta(t,x \mid \pi)$ and the problem becomes a regression problem. We just have more dimensions than before, because we have the extra variable π. So, e.g., if we decide to use machine learning, then we can use the same formula (4.6), but the vector u will now contain t (or the time embedding), x, and also π.

The learning process is similar to before, though now the data that is given to us consists of image-prompt pairs[2] $(a^{(i)}, \pi^{(i)})$. We still sample the times $t^{(i)}$, independently and uniformly on

[2]The images in the training data are typically tagged. They are either tagged by the users who produced them (like your phone camera tags the pictues you take with it) or they are tagged by web crawlers that scour the web and collect the images. These bots tag the images by either scraping text they find around where the images appear (like on Wikipedia) or by running the images through a pattern-recognition software that tags them.

$[0,T]$, and then sample $x^{(i)}$ as a multivariate normal with mean $a^{(i)}e^{-\frac{1}{2}\int_0^{t^{(i)}}\beta(r)\,dr}$ and variance-covariance matrix $\big(1-e^{-\int_0^{t^{(i)}}\beta(r)\,dr}\big)I$. We still define

$$s^{(i)}=\frac{a^{(i)}e^{-\frac{1}{2}\int_0^{t^{(i)}}\beta(r)\,dr}-x^{(i)}}{1-e^{-\int_0^{t^{(i)}}\beta(r)\,dr}}$$

as before. But now, since $s(t,x\mid\pi)$ minimizes (5.2), we use the quadratic loss function

$$L(\theta)=\sum_{i=1}^{M}||s_\theta(t^{(i)},x^{(i)}\mid\pi^{(i)})-s^{(i)}||^2. \tag{5.3}$$

As before, we find the parameter θ that minimizes the loss function and then use the corresponding s_θ from the family we chose and plug $s_\theta(T-t,Y(t)\mid\pi)$ in place of $\nabla\log p(T-t,Y(t)\mid\pi)$ in (5.1), when we run that SDE to generate new samples. Then, $Y(T)$ will presumably have a pdf close to $p(0,a\mid\pi)$, as desired.

Several tools have been developed to enhance the alignment of the generated image with the given prompt. For example, during the training period, both the unconditional and the conditional gradients $\nabla\log p(t,x)$ and $\nabla\log p(t,x\mid\pi)$ are learned. So the first one is estimated using the loss function (4.5) and all the available images, without regard to their tags, and the second one is estimated using the loss function (5.3) and all the available images with their tags. Denote the estimators by $\hat{s}(t,x)$ and $\hat{s}(t,x\mid\pi)$, respectively. Then, when a prompt π is given, a sample is generated by choosing $Y(0)$ as a standard multivariate normal and then runnig the SDE

$$dY(t)=\frac{1}{2}\beta(T-t)\big(Y(t)+2\big[(1+\lambda)\hat{s}(T-t,Y(t)\mid\pi)-\lambda\hat{s}(T-t,Y(t))\big]\big)\,dt+\sqrt{\beta(T-t)}\,d\overline{B}(t)$$

and returning $Y(T)$ as the output. Here, $\lambda>0$ is the "guidance scale"–a parameter that is tuned to optimize the outcome. Basically, the term in the square brackets rewards alignment with the prompt and penalizes deviations from it, in an attempt to have $Y(T)$ more aligned with the given prompt π.