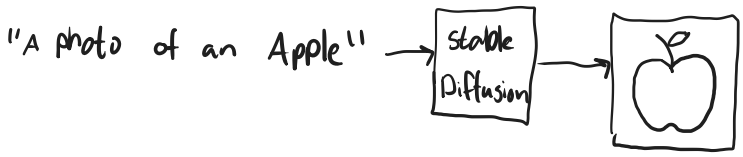
Diffusion and Score Based Models

Gijs Bellaard

Presentation date: March 24, 2022
Compiled date: April 10, 2023

Bart showed and explained to us *Stable Diffusion* two weeks ago. Stable diffusion accomplishes the, in my opinion very impressive, feat of generating natural images from text descriptions.



Stable diffusion is, as the name suggest, a *denoising diffusion probabilistic model* (DDPM). I will start by showing the general idea and mathematics behind diffusion models. Afterwards I'll introduce the *score function*, *Langevin dynamics*, and *Langevin sampling*. Only then I will start discussing *score based generative modelling*. So, in fact, this presentation will mostly only encapsulate the background section from [4, Sec.2].

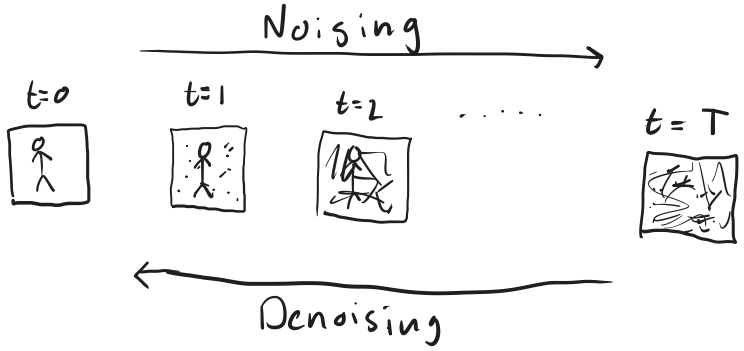
A warning before we start: I am going to butcher notation. The same symbol can be a realization, an outcome, a random variable, the corresponding distribution, or a probability density function, all at the same time. Also, I have simplified the theory to it bare minimum wherever I could, at the expensive of being precise. I hope that we can get through this by lots of hand-waving.

1 Denoising Diffusion Probabilistic Models (DDPM)

Diffusion models [1, 6] are *generative models*: that being a way to generate images that *look like* images from a given dataset of interest. Let me emphasize the *look like* part here: we don't want a generative model that did not *generalize* and just returns exact copies from the dataset.

The general idea of diffusion models is extremely simple: we train a network to denoise images that have been degraded/noised. Once done training we are

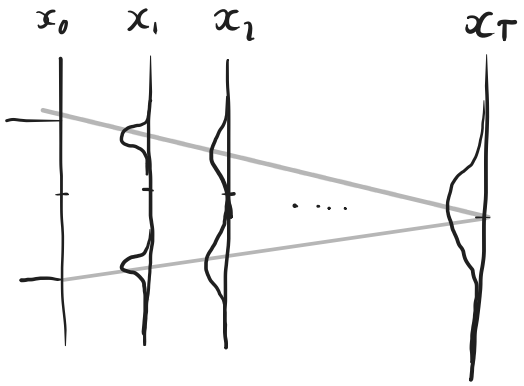
able to pass the network pure noise and out comes an image.



Let us put this idea on mathematical footing. Pick a sample x_0 from a dataset. Let us define a forward diffusion process in which we add small amount of Gaussian noise to the sample x_0 in T steps, producing a sequence of noisy samples $x_1; \cdots; x_T$. The step sizes are controlled by an increasing noise *variance schedule* $0 \quad {}^2_1 \quad {}^2_T < 1$.

$$x_t = \ {}_t x_{t\ 1} + \ {}_t \text{"}_t; \text{ where } {}^2_t + {}^2_t = 1 \tag{1}$$

and $\text{"}_t \quad N(0; 1)$ is standard normal distributed, and the subscript t in "_t indicates that these are different random variables. At time $t = T$ we have $x_T \quad N(0; 1)$ is approximately standard Gaussian noise if we choose ${}^2_T$ close enough to 1, i.e. the original image is essentially completely noised away.



Note that I use a slightly different notation than can be found in the literature: for some reason the original authors [1] thought it was a good idea to write down variances without a square, meaning that they need to carry around a bunch of square roots to make the formulas work. I'd rather carry around squares.

To sample from, for example, x_t at $t = 30$ we don't need to go through (1) thirty times. This is due to the fact that normal distributions keep being normal distributions under addition and scaling. To be precise, we have that x_t is distributed as:

$$x_t \quad {}^{-}{}_t x_0 + {}^{-}{}_t \text{"}_t; \text{ where } {}^{-}{}_t = \prod_{s=1}^{s\forall t} {}_s \text{ and } {}^{-2}_t + {}^{-2}_t = 1: \tag{2}$$

The equation for ${}^{-}{}_t$ can be quickly deduced if we imagine for a moment that $x_0 \quad N(0; 1)$: we see then that (1) always keeps it so the variance of x_t is 1, so we have to conclude that ${}^{-2}_t + {}^{-2}_t = 1$. Furthermore, we have the relation:

$${}^{-}{}_t = {}^{2}_t + (\quad {}^{-}{}_{t\ 1})^2: \tag{3}$$

If we could train a network $\Phi = \Phi$, with parameters , that in some way reverses the forward process (1) we could start with standard Gaussian noise and (hopefully) get back an image by reversing from $t = T$ all the way back to $t = 0$.

At this point we have multiple choices:

- we can sample $x_{t\ 1} = {}^{-}{}_{t\ 1} x_0 + {}^{-}{}_{t\ 1} \text{"}_{t\ 1}$ and $x_t = \quad {}_t x_{t\ 1} + \quad {}_t \text{"}_t$, and train our network Φ to try and reconstruct $x_{t\ 1}$ from x_t, which means our loss L for a single data point is something like:

$$L = k x_{t\ 1} \quad \Phi(x_t; t)k; \tag{4}$$

- or we could train the network to predict the last incremental noise "_t that was added:

$$L = k\text{"}_t \quad \Phi(x_t; t)k: \tag{5}$$

- Or we can be bold an try to train the network to reconstruct x_0 directly from a sample $x_t = {}^{-}{}_t x_0 + {}^{-}{}_t \text{"}_t$:

$$L = k x_0 \quad \Phi(x_t; t)k; \tag{6}$$

- or we could train the network to predict the total noise "_t:

$$L = k\text{"}_t \quad \Phi(x_t; t)k: \tag{7}$$

Now to me these last two seem too bold, to good to be true, but for reasons, that are unclear to me, it turns out empirically that the bold option (7) is the best [1]. Coincidentally, the easiest to analyse method is also (7).

Great, so after training we have a network $\Phi(x_t; t)$ that can approximately predict ϵ_t from $x_t = \bar{\alpha}_t x_0 + \bar{\alpha}_t \epsilon_t$ and t, now how do we actually step backwards through time?

Simple:

$$x_0 \quad \frac{1}{\bar{\alpha}_t} \quad x_t \quad \bar{\alpha}_t \Phi(x_t; t) \tag{8}$$

Well... No. This, unsurprisingly, gives horrendous results in practice, instead we combine (8) with suitable guess for $x_{t\ 1}$. One choice is the tractable normal distribution of $x_{t\ 1}$ conditioned on the knowledge of x_t and x_0:

$$x_{t\ 1} \, j \, x_0; x_t \quad {}_t + \ {}_t \epsilon; \tag{9}$$

where

$${}_t(x_0; x_t) = \frac{1}{\bar{\alpha}_t^2} \quad {}_t \ \bar{\alpha}_{t\ 1} x_t + \bar{\alpha}_t \ {}_{1\ t}^{2} x_0 \quad \text{and} \quad {}_t = \frac{\bar{\alpha}_{t\ 1}}{\bar{\alpha}_t} \ {}_t \tag{10}$$

Filling in our approximation for x_0 (8) into the mean of (9) and simplifying we get that

$${}_t \quad {}_t^{0}(x_t) = \frac{1}{\alpha_t} \quad x_t \quad \frac{1 - \alpha_t^2}{\bar{\alpha}_t} \Phi(x_t; t) \tag{11}$$

so we take a sample $x_{t\ 1}$ as:

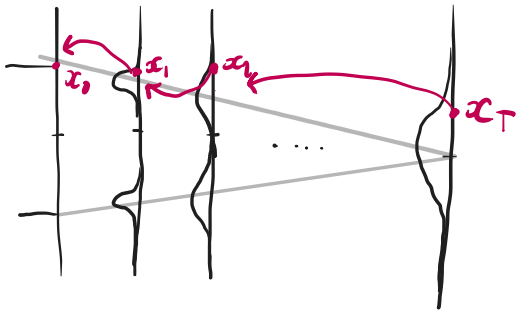
$$x_{t\ 1} \quad {}_t^{0} + \ {}_t \epsilon \tag{12}$$

and continue iterating from there.

Now what I find suspicious here is that after simplifying we got (11), which looks a lot like if we had inverted the relation $x_t = \ {}_t x_{t\ 1} + \ {}_t \epsilon_t$:

$$x_{t\ 1} = \frac{1}{\alpha_t} (x_t \quad {}_t \epsilon_t) \ \text{ with } \ \bar{\alpha}_t \epsilon_t \quad {}_t \Phi(x_t; t) + \ {}_t \ \bar{\alpha}_{t\ 1} \epsilon; \tag{13}$$

which looks suspiciously like (3). So it seems to me there is a better way to derive (11), I however been unable to nail it down completely.



So, to conclude, the (original) diffusion model algorithm [1] is nothing but:

Algorithm 1 Diffusion Training Step

Pick a data point x_0 randomly from your dataset
Pick a random step amount 1 t T
Sample some Gaussian noise ε_t $\mathcal{N}(0; 1)$
Degrade the data point $x_t = \bar{\alpha}_t x_0 + \bar{\beta}_t \varepsilon_t$
Calculate the loss $L = k\varepsilon_t \Phi(x_t; t)k$
Change the parameters such that the loss L becomes smaller.

Algorithm 2 Diffusion Sampling

Sample some Gaussian noise x_T $\mathcal{N}(0; 1)$
for $t = T; \dots; 1$ **do**
  Calculate $\mu_t = \frac{1}{\alpha_t} x_t \frac{\beta_t^2}{\bar{\beta}_t} \Phi(x_t; t)$
  Sample some Gaussian noise ε $\mathcal{N}(0; 1)$
  $x_{t-1} = \mu_t + \beta_t \varepsilon$
end for
return x_0

2 Score Function, Langevin Dynamics and Sampling

Yang Song [2–4] defines the *score function* as:

$$(\nabla_x \log p)(x); \tag{14}$$

that being the gradient of probability density function $p(x)$ w.r.t. the variable(s) x. Now the annoying thing is that normally statistical *score* is defined differently. In this case we have a model density function $p(x)$ with parameters , and the score is the gradient w.r.t the parameters of the log-likelihood of some observation(s) x :

$$\nabla (\log p(x)) \tag{15}$$

It seems Song was inspired by statistical score and therefor called it a score function. Now to remove all confusion we follow the (possibly bad) precedent set by Song from here on out: when I say score function I really mean (14).

One specific example of a particularly nice score function is the one of a normal distribution:

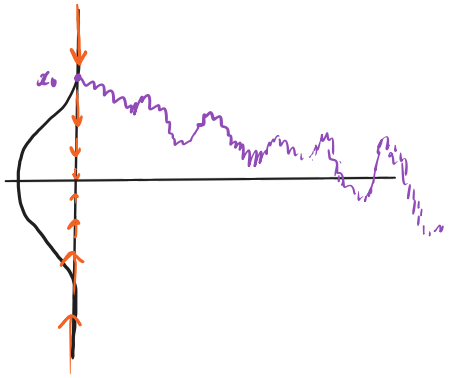
$$p(x) \quad \mathcal{N}(0; \ ^2) \text{ then } (\nabla_x \log p)(x) = \quad \frac{x}{^2} \tag{16}$$

And this is one to remember for later.

Consider the following stochastic differential equation:

$$dx_t = \frac{1}{2}(\nabla_x \log p)(x_t)dt + dw_t \tag{17}$$

where w is standard Brownian motion, i.e. the Wiener process. This equation is called the Langevin Itô diffusion. The interesting thing about this SDE is that when we let $t \to \infty$ we get that $x_t \sim p(x)$.



By discretizing (17) in discrete time steps t_i with $t_{i+1} - t_i = \Delta t$, i.e. by applying the Euler–Maruyama method, we get the Langevin Markov chain Monte Carlo (MCMC):

$$x_{i+1} = x_i + \frac{1}{2}(\nabla_x \log p)(x_i)\Delta t + \varepsilon\sqrt{\Delta t} \tag{18}$$

where again $\varepsilon \sim \mathcal{N}(0, 1)$. Again, if we let $i \to \infty$ and $\Delta t \to 0$ we get that $x_i \sim p(x)$. Now this is very interesting: we have achieved a way to sample from $p(x)$ on a computer using just its score function.

Algorithm 3 Langevin Sampling

Pick some x_0
for $i = 1, \ldots, n$ **do**
  Sample some Gaussian noise $\varepsilon \sim \mathcal{N}(0, 1)$
  $x_i = x_{i-1} + \frac{1}{2}(\nabla_x \log p)(x_{i-1})\Delta t + \varepsilon\sqrt{\Delta t}$
end for
return x_n

This sampling method is referred to as Langevin sampling, Langevin algorithm, Langevin method, or Langevin dynamics. I will refer to it as Langevin sampling.

Also, personally, anyone who refers to Langevin sampling as Langevin dynamics is wrong.

Furthermore, It seems people are confusing the above Langevin Sampling with *Stochastic gradient Langevin dynamics* (SGLD) by Welling [5], which is explicitly *not* about generating samples from a dataset distribution. Instead Welling relates Stochastic gradient descent and Langevin dynamics and creates SGLD as another way to optimize for *parameters*, i.e. it samples the *posterior distribution* $p(\ \ j\, x)$ given the dataset x.

3 Denoising Score Matching with Langevin Dynamics (SMLD)

If we could create a network $\Phi\ \ $ that approximates the score function $\ r_{\,x} \log p$:

$$L = k(\ r_{\,x} \log p)(x) \quad \Phi\ \ (x)k; \tag{19}$$

we could in theory generate samples from our data density function $p(x)$ using Langevin sampling. This is the key idea of *Score Based Generative Modelling* [3].

But wait, how do even train this? We don't have access to the score function after all. This is where *score matching* comes into play. Score matching is any way of getting the network to approximate the desired score function. For now lets assume that it can be done.

However, if one tries to tackle this problem naively, that is try to estimate the score function and apply Langevin sampling directly, things not work out at all, as discussed in [3, Sec.3]. To quote them

> "The manifold hypothesis states that data in the real world tend to concentrate on low dimensional manifolds embedded in a high dimensional space (a.k.a., the ambient space). This hypothesis empirically holds for many datasets, and has become the foundation of manifold learning.
>
> Under the manifold hypothesis, score-based generative models will face two key difficulties:
>
> First, since the score is a gradient taken in the ambient space, it is undefined when x is confined to a low dimensional manifold.
>
> Second, the score matching objective provides a consistent score estimator only when the support of the data distribution is the whole space, and will be inconsistent when the data reside on a low-dimensional manifold. "

So what do we do? Song [3] suggest perturbing the dataset with random Gaussian noise makes the data distribution more amenable to score-based generative

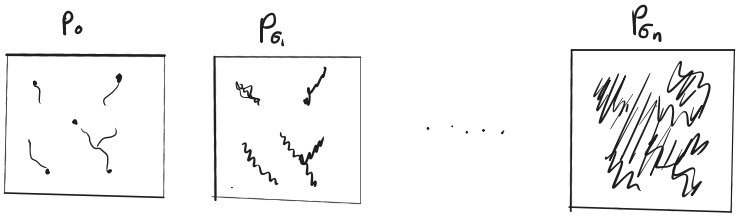
modeling. This helps as perturbing the dataset spreads out the distribution, making the data manifold nicer to work with.

So, consider a noise variance ^{2}, and the perturbed data distributions

$$\tilde{x} = x + \quad ''; \text{ where } x \quad p(x) \text{ and } '' \quad N(0;1) \tag{20}$$

i.e.

$$p\ (\tilde{x}) = \int_x p(x) f\ (\tilde{x} \quad x) dx; \text{ where } f\ (x) \quad N(0; \quad) \tag{21}$$



Instead of training the model $\Phi\ (x)$ to predict the score function directly, we instead train a network $\Phi\ (\tilde{x}; \quad)$ it to estimate the scores of perturbed data distributions:

$$L = k(\ r_x \log p\ )(\tilde{x}) \quad \Phi\ (\tilde{x}; \quad)k \tag{22}$$

Song calls $\Phi\ (\tilde{x}; \quad)$ a Noise Conditional Score Network [3].

Now, on to the question of how to actually train this. Suppose we first sample a data point x_0 from our dataset, pick a noise variance , and then sample a perturbed data point $\tilde{x} = x + \quad ''$ by adding some noise. Now, the following conditional score function is trivial:

$$r_x \log p\ (\tilde{x}\ j\ x) = \quad \frac{\tilde{x} \quad x}{^2} \tag{23}$$

Now, without further clarification, because I don't even fully understand this, we can instead try to train the network estimate this score function. All in all we get the following algorithm [3]

Algorithm 4 Noise Conditional Score Training Step

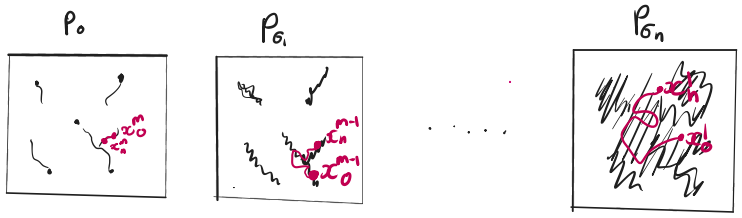
Pick a data point x randomly from your dataset
Pick a random noise variance ^{2}
Sample some Gaussian noise $'' \quad N(0;1)$
Perturb the data point $\tilde{x} = x + \quad ''$
Calculate the loss $L = k \quad \frac{x \quad x}{^2} \quad \Phi\ (\tilde{x}; \quad)k = k \quad \frac{''}{} \quad \Phi\ (\tilde{x}; \quad)k$
Change the parameters such that the loss L becomes smaller.

Now this looks an awful lot like the diffusion training step we have seen before: The network is effectively trained to predict the (minus of the) noise ϵ that was added to a dataset sample x.

As for sampling, Yong [3] suggests what he calls *Annealed Langevin dynamics*. The idea is to start Langevin sampling in a highly perturbed data distributions with a large time step Δt for a predetermined amount of steps n. Once this is done we do this again but for a slightly less perturbed data distribution with a slightly smaller time step Δt. We repeat this process for m noise scales. We encode the process in the variables x_j^i where i indicates a noise scale, and j the Langevin steps.

Algorithm 5 Annealed Langevin Sampling

Pick some x_0^1
for $i = 1, \ldots, m$ **do**
  Pick an appropriate and Δt
  for $j = 1, \ldots, n$ **do**
    Sample some Gaussian noise $\epsilon \sim \mathcal{N}(0, 1)$
    $x_j^i = x_{j-1}^i + \frac{1}{2}\Phi(x_{j-1}^i, \ldots)\Delta t + \epsilon\sqrt{\Delta t}$
  end for
  $x_0^{i+1} = x_n^i$
end for



If we choose to be, again, bold and set $n = 1$ we get that annealed Langevin sampling looks a lot like diffusion sampling. If it wasn't obvious already: there seems to be a direct relation between diffusion and score-based models.

The whole framework is later referred to as Denoising Score Matching with Langevin Dynamics (SMLD) by Song [3].

4 Generative Modelling with SDEs

Now, as shown in [4], both DDPM and SMLD can be interpreted as two different discretizations of the same SDEs. The forward process of both of them can seen