Stochastic project: Diffusion Models

Wennan Wang **Jingrong Guan** **Lecong Ding**

Abstract

Diffusion models have recently emerged as a popular type of generative model. This paper explores their principles, improvements, and applications. Firstly, diffusion models are categorized into conditional and unconditional diffusion models based on the task type. Next, a brief introduction to the basic principles of diffusion models is provided. Following this, two commonly used unconditional diffusion models, Denoising Diffusion Probability Model(DDPM) and Score-based model, are introduced in sequence. Building on this foundation, we summarize two improvement strategies, incorporating our insights and existing enhanced models. Finally, we offer some observations on the applications and future development of diffusion models.

1 Introduction

Diffusion models are a class of probability-based generative models. The concept of the models first appeared in statistical physics, used to describe the process of particles moving. In 2015, Sohl-Dickstein of Stanford introduced this concept to machine learning. He initially proposed systematically and slowly destroying the structure in the data distribution through an iterative forward diffusion process, and then learning an inverse diffusion process that restores the structure in the data, resulting in a highly flexible and easy-to-handle generative model of the data. This became one of the key breakthroughs in diffusion modelling. In 2020, Jonathan Ho of the University of California, Berkeley, proposed the **Denoising Diffusion Probabilistic Model**(DDPM) based on diffusion modelling. This technique led to a huge improvement in the quality of the generated images. Since then, diffusion models have shown potential in more and more fields such as prediction and generation, especially in the field of **high-resolution image generation**. Using AI to generate images is a popular direction of research nowadays. In fact, many people have already used generative models based on diffusion models such as DALL-E 3, Stable diffusion 3, Sora in their lives. And diffusion models have replaced the original common generative models, such as GAN, VAE, etc., and become the first choice of Stability AI, OpenAI, Google Brain, etc. to build models.

Figure 1: Generated samples from Sable Diffusion 3

Therefore, in this paper, we will introduce in detail the mathematical principles behind several typical diffusion models, such as **denoising diffusion probabilistic models(DDPMs)** and **score-based**

stochastic differential equations(Score SDEs), from a mathematical point of view. The aim is to understand the application methods and explore the application areas of diffusion models based on the stochastic process course of this semester. Finally, we conclude the paper with a summary of what we have covered in the paper. And we have made an outlook on the direction of our further work and the future progress of the model.

2 Overview of the model

Typically, we classify diffusion models into two main categories: unconditional and conditional diffusion models.

The unconditional diffusion models work in an unsupervised manner. Generating data samples relies on self-information without the need for supervised signals. Within this category, models can be further classified into probability-based diffusion models and score-based diffusion models. Examples include denoising diffusion probability models (DDPMs) and score-based stochastic differential equations (Score SDEs).

The conditional model builds on the unconditional diffusion model by introducing additional information (e.g., category labels, conditional constraints, etc.) to train the network. This enhances the performance of the model and makes the results more consistent with certain objective laws. Empirical studies have shown that conditional generation models using data labels are easier to train and perform better compared to unconditional models. These conditional mechanisms are more conducive to the generation of application-speciffc ffelds by using the control of other information to generate results.

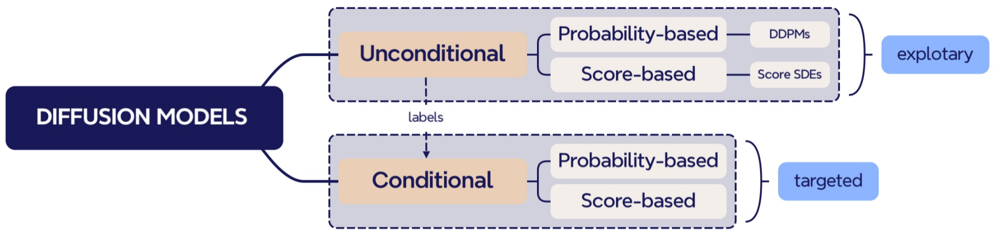


Figure 2: Classification of diffusion models

Therefore, unconditional diffusion model generation is innovative and versatile, suitable for exploratory application scenarios. While conditional diffusion model can control the output more specifically, and is the most commonly used means of generation nowadays.

We build a generative Markov chain which converts a simple known distribution (e.g. a Gaussian) into a target (data) distribution using a diffusion process. Rather than use this Markov chain to approximately evaluate a model which has been otherwise defined, we explicitly define the probabilistic model as the endpoint of the Markov chain.

The implementation of the diffusion model is specifically divided into two parts: the diffusion process and the generation process. The first is the diffusion process (forward process). Gradually add some kind of noise to the data, i.e., gradually impose the so-called diffusion kernel (Gaussian kernel or binomial kernel) on the target distribution. We see the every image as a situation in this process. And finally get the smooth distribution of this Markov process which is the known simple distribution. The second is the generation process (backward process). Specifically, the neural network is trained to trace the forward process and learn the diffusion kernel's parameters. Then using the learned parameters, starting from the known simple distribution and imposing the same form of diffusion as the forward process. At last, the target distribution is generated after a parameterised Markov chain.

In the following two sections, we will specifically introduce two typical unconditional diffusion models.

3 Denoising Diffusion Probability Model

A DDPM is formed by two parametrized Markov chains:

1. The forward chain is constructed as
$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t, \quad t = 1, 2, ... \tag{3.1}$$
It's designed to add Gaussian white noise in each step in the Markov chain so that the transition probability is Gaussian $q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$ for some parameters $\{\beta_t\}_{t=1}^{\infty}$, where x_t represents an image. We can verify that as $t \to \infty$, there exists limiting distribution and moreover, the limit is standard Gaussian: $q(x_t|X_0) \to \mathcal{N}(0, I), t \to \infty$ under the condition that $\beta_t << 1$.

   The reason why the sum of the squares of the coefficients is 1:
$$\begin{aligned} x_t &= a_t x_{t-1} + b_t\epsilon_t \\ &= \dots \\ &= (a_t \dots a_1)x_0 + (a_t \dots a_2)b_1\epsilon_1 + (a_t \dots a_3)b_2\epsilon_2 + \cdots + a_t b_{t-1}\epsilon_{t-1} + b_t\epsilon_t \end{aligned}$$
The sum of Gaussian white noise is still a Gaussian white noise,so we have
$$x_t = (a_t \dots a_1)x_0 + \sqrt{(a_t \dots a_2)^2 b_1^2 + (a_t \dots a_3)^2 b_2^2 + \cdots + a_t^2 b_{t-1}^2 + b_t^2}\bar{\epsilon}_t, \quad \bar{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
By adding the constraint that $a_t^2 + b_t^2 = 1$,we will have:
$$\begin{aligned} &(a_t \dots a_1)^2 + (a_t \dots a_2)^2 b_1^2 + (a_t \dots a_3)^2 b_2^2 + \cdots + a_t^2 b_{t-1}^2 + b_t^2 \\ =&a_t^2\left(a_{t-1}^2\left(\dots\left(a_2^2(a_1^2+b_1^2)+b_2^2\right)+\dots\right)+b_{t-1}^2\right)+b_t^2 \\ =&1 \end{aligned}$$
This gives us great convenience in the later derivation of the formula.

2. The backward(reverse) process is the "denoising" inference process of diffusion. The reverse transition probability is given by Bayesian formula
$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t|x_{t-1})q(x_{t-1}|x_0)}{q(x_t|x_0)}$$
Actually $q(x_t|x_{t-1}) = q(x_t|x_{t-1}, x_0)$in the above formula because of the property of a Markov chain.
Unfortunately when we take our model into application, x_0 is usually unknown. So we have to build up a deep learning net to train $p_\theta(x_{t-1}|x_t)$,which is used to approximate $q(x_{t-1}|x_t, x_0)$ when we just have x_t and t.
A fact that $q(x_{t-1}|x_t, x_0)$ is still Gaussian, given that $q(x_t|x_{t-1})$ is Gaussian and $\beta_t << 1$. So we can model p_θ as Gaussian, then our training goal is to approximate the parameter of the Gaussian distribution:μ_θ and β_θ
$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \beta_\theta(x_t, t)I)$$

3.1 Forward process

Proposition 3.1.1 (Convergence of forward process) *Suppose the forward process is a Markov chain that*
$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I) \text{ i.i.d.}$$
i.e. the transition probability is Gaussian $x_t|x_{t-1} \sim \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t I)$. If for each $t = 1, 2, ...$, $0 < \beta_t < 1$ such that $\lim_{t\to\infty}\prod_{i=1}^{t}(1-\beta_t) = 0$, then the forward Markov chain possesses a limiting distribution and
$$q(x_t|x_0) \to \mathcal{N}(0, I), \text{ as } t \to \infty \tag{3.2}$$
for any given $x_0 \in \mathbb{R}^n$.

3.2 Backward process

There is an analytic formula for $q(x_{t-1}|x_t, x_0)$ if we know the final generated sample x_0 from backward process!

Proposition 3.2.1 *Denote $A_t = \prod_{i=1}^{t} \alpha_t$. For given generated sample x_0, we have*

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \widetilde{\mu}_t(x_t, x_0), \widetilde{\beta}_t I)$$

where

$$\widetilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_t}(1 - A_{t-1})}{1 - A_t} x_t + \frac{\sqrt{A_{t-1}}\beta_t}{1 - A_t} x_0 \tag{3.3}$$

$$\widetilde{\beta}_t = \frac{1 - A_{t-1}}{1 - A_t} \beta_t \tag{3.4}$$

3.3 Approximation

In general case, there is no explicit formula for $q(x_{t-1}|x_t)$ when x_0 is unknown. However we konw that

$$x_0 = \frac{1}{\sqrt{A_t}}(x_t - (1 - A_t)z_t), \quad z_t \sim \mathcal{N}(0, I) \tag{3.5}$$

where the Gaussian noise $z_t \sim \mathcal{N}(0, I)$. Take it into (3.3), we obtain that

$$\widetilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - A_t}} z_t \right) \tag{3.6}$$

We can set $z_\theta(x_t, t)$ to be approximator intended to predict z_t,which is the Gaussian white noise added into x_0 to obtain x_t, using deep neural net.

Here we assume that $\sigma_\theta(x_t, t)$ is the same as the variance in the forward process.

Hence the parameters for backward process model can be

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - A_t}} z_\theta(x_t, t) \right) \tag{3.7}$$

$$\sigma_\theta(x_t, t) = \widetilde{\beta}_t = \frac{1 - A_{t-1}}{1 - A_t} \beta_t \tag{3.8}$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_\theta(x_t, t) I) \tag{3.9}$$

where $p_\theta(x_{t-1}|x_t)$ is the modelled probability to approximate real $q(x_{t-1}|x_t, x_0)$

When training p_θ,we need to design the loss function and then minmize it.

3.4 Design of the loss function

Under the given data distribution $x_0 \sim q(x_0)$,by the property of a Markov process $p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$, it's reasonable to maximize the log-likelyhood of $p_\theta(x_0)$, i.e. to minimize the cross entropy of $p_\theta(x_0)$ under $q(x_0)$:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta, x_0) = \arg\min_\theta \mathbb{E}_{q(x_0)}[-\log p_\theta(x_0)] \tag{3.10}$$

But directly compute $p_\theta(x_0)$ is intractable, so we have to use some tools to simplify the loss.

Lemma 3.4.1 *By using Jensen inequality,the negative log-likelyhood can be bounded by the* ***Variational Lower Bound****: $\mathcal{L}_{VLB}$, i.e.*

$$\mathcal{L} = \mathbb{E}_{q(x_0)}[-\log p_\theta(x_0)] \leq \mathbb{E}_{q(x_{0:T})} \left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} \right] \triangleq \mathcal{L}_{VLB} \tag{3.11}$$

Instead of directly minimizing $\mathcal{L}$, we choose to optimize $\mathcal{L}_{\text{VLB}}$ since it's more computable by the follwing Lemma.

Lemma 3.4.2 (Decomposition of $\mathcal{L}_{\text{VLB}}$) *$\mathcal{L}_{VLB}$ can be decomposed into a sum of several KL-divergence:*

$$\mathcal{L}_{VLB} = L_T + L_{T-1} + ... + L_0 \tag{3.12}$$

$$L_T = D_{KL}(q(x_T|x_0)||p_\theta(x_T)) \tag{3.13}$$

$$L_t = D_{KL}(q(x_t|x_{t+1}, x_0)||p_\theta(x_t|x_{t+1})), \quad 1 \leq t \leq T-1 \tag{3.14}$$

$$L_0 = -\log p_\theta(x_0|x_1) \tag{3.15}$$

Actually $q(x_0)$ has no parameter, $p_\theta(x_T)$ is pure noise, so L_T can be seen as constant to the model. For the fact that $q(x_t|x_{t+1}, x_0$ is Gaussian distribution and we have supposed that $p_\theta(x_t|x_{t+1}))$ is Gaussian distribution, by solving the KL divergence of Gaussian distribution, we have that

$$L_t = \mathbb{E}_q\left[\frac{1}{2|\sigma_\theta(x_t,t)|^2}\|\widetilde{\mu}_t(x_t,x_0) - \mu_\theta(x_t,t)\|_2^2\right] + C, \quad 1 \leq t \leq T-1 \tag{3.16}$$

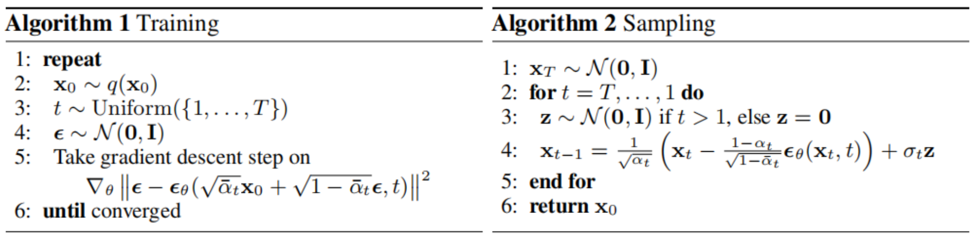
Combining (3.5),(3.6),(3.7), we can further derive that

$$\begin{aligned} L_t &= \mathbb{E}_{x_0,z_t}\left[\frac{1}{2\|\sigma_\theta(x_t,t)\|^2}\|\widetilde{\mu}_t(x_t,x_0) - \mu_\theta(x_t,t)\|^2\right] \\ &= \mathbb{E}_{x_0,z_t}\left[\frac{\beta_t^2}{2\alpha_t(1-A_t)\widetilde{\beta}_t^2}\left\|z_t - z_\theta(\sqrt{A_t}x_0 + \sqrt{1-A_t}z_t, t)\right\|^2\right] \end{aligned}$$

For simplicity we can ignore the coefficients in L_t, then we have

$$L_t^{\text{simple}} = \mathbb{E}_{x_0,z_t}\left[\left\|z_t - z_\theta(\sqrt{A_t}x_0 + \sqrt{1-A_t}z_t, t)\right\|^2\right]$$

In this picture of algorithm, ϵ is z_t in our formula and ϵ_θ is z_θ in our formula.



```
Algorithm 1 Training
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \text{Uniform}(\{1, \dots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$
6: until converged
```

```
Algorithm 2 Sampling
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = T, \dots, 1$ do
3:   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4:   $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

4 Score-based model

In order to build such a generative model, we first need a way to represent a probability distribution. One such way, as in likelihood-based models, is to directly model the probability density function (p.d.f.) or probability mass function (p.m.f.). Let $f_\theta(x)$ be a real-valued function parameterized by a learnable parameter . We can define a p.d.f. via

$$p_\theta(x) = \frac{e^{-f_\theta(x)}}{Z_\theta}$$

where $Z_\theta > 0$ is a normalizing constant dependent on θ, such that $\int p_\theta(x)\mathrm{d}x = 1$. Here the function is often called an **unnormalized probabilistic model**, or **energy-based model**.

We can train $p_\theta(x)$ by maximizing the log-likelihood of the data

$$\max_\theta \sum_{i=1}^{N} \log p_\theta(x)$$

This is undesirable because in order to compute $p_\theta(x)$, we must evaluate the normalizing constant Z_θ, a typically intractable quantity for any general $f_\theta(x)$. Thus to make maximum likelihood training feasible, likelihood-based models must either restrict their model architectures.

By modeling the score function instead of the density function, we can sidestep the difficulty of intractable normalizing constants.

4.1 Score function

Definition 4.1.1 *The* ***score function*** *of a distribution $p(x)$ is defined as*

$$\nabla_x \log p(x)$$

and a model for the score function is called a ***score-based model****, which we denote as $s_\theta(x)$, for some normalized probability model*

$$p_\theta(x) = \frac{\widetilde{p}_\theta(x)}{Z_\theta}$$

Example 4.1.2 *we can easily parameterize a score-based model with the energy-based model via*

$$s_\theta(x) = \nabla \log p_\theta(x) = -\nabla_x f_\theta(x) - \underbrace{\nabla_x Z_\theta}_{=0} = -\nabla_x f_\theta(x)$$

Note that the score-based model is independent of the normalizing constant Z_θ! This significantly expands the family of models that we can tractably use, since we don't need any special architectures to make the normalizing constant tractable

4.2 Langevin dynamics

Langevin dynamics provides an MCMC procedure to sample from data distribution $p_{data}(x)$ using only its score function. Specifically, it initializes the chain from an arbitrary prior distribution $x_0 \sim \pi(x)$, and then iterates the following

$$x_{i+1} = x_i + \delta \nabla_x \log p_{data}(x) + \sqrt{2\delta}\epsilon_i, \quad i = 0, 1, ..., T \tag{4.1}$$

where $\epsilon_i \sim \mathcal{N}(0, I)$. When $\delta \to 0$ and $T \to \infty$, x_T obtained from the procedure converges to a sample from $p(x)$ under some regularity conditions. The continuous form of (2.5) is an SDE:

$$\mathrm{d}x_t = \delta \nabla_x \log p(x)\mathrm{d}t + \sqrt{2\delta}\mathrm{d}W_t \tag{4.2}$$

Unfortunately we don't know the actual distribution $p_{data}(x)$, we can only only sampled from modeled probability $p_\theta(x)$ using Langevin dynamics

$$\begin{aligned} x_{i+1} &= x_i + \delta \nabla_x \log p_\theta(x) + \sqrt{2\delta}\epsilon_i & (4.3)\\ &= x_i + \delta s_\theta(x) + \sqrt{2\delta}\epsilon_i & (4.4)\end{aligned}$$

And the core of score-matching lies in that using a best estimator(e.g. nerual network) $s_\theta(x)$ to replace unknown $\nabla_x \log p_{data}(x)$, while the "prediction error" $\|s_\theta(x) - \nabla_x \log p_{data}(x)\|$ is minimized.

4.3 Scored-based modeling with forward and reverse SDE

When the number of noise scales approaches infinity, we essentially perturb the data distribution with continuously growing levels of noise. In this case, the noise perturbation procedure is a continuous-time stochastic process. Usually presented as an SDE(diffusion process):

$$\mathrm{d}x_t = f(x_t, t)\mathrm{d}t + g(x_t, t)\mathrm{d}W_t \tag{4.5}$$

where $f(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d, g(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^{d\times d}$.

DDPM can be seen as a discretised version of a special SDE, since the forward process of

$$x_t - x_{t-1} = (\sqrt{1-\beta_t} - 1)x_{t-1} + \sqrt{\beta_t}\epsilon_t$$

When t is continuous variable, the equation becomes

$$\mathrm{d}x_t = (\sqrt{1-\beta_t} - 1)x_t + \sqrt{\beta_t}\mathrm{d}W_t \tag{4.6}$$

In empirical practice, the hyper-parameters $\{\beta_t\}_{t\geq 1}$ is very small that $\beta_t \in (0.01, 0.1)$, so by Taylor expansion $\sqrt{1-\beta_t} - 1 \approx -\beta_t/2$, so the corresponding SDE can be simplified as:

$$\mathrm{d}x_t = -\frac{\beta_t}{2}x_t\mathrm{d}t + \sqrt{\beta_t}\mathrm{d}W_t \tag{4.7}$$

$$x_t - x_{t-1} \approx -\frac{\beta_t}{2}x_{t-1} + \sqrt{\beta_t}\epsilon_t \tag{4.8}$$

4.4 Reverse SDE and probability flow ODE of diffusion process

Recall that with a finite number of noise scales, we can generate samples by reversing the perturbation process with Langevin dynamics For infinite noise scales, we can analogously reverse the perturbation process for sample generation by using the reverse SDE.

Theorem 4.4.1 *For diffusion process defined by equation (2.25), if it is invertible, then the reverse process is characterised by reverse SDE:*

$$\mathrm{d}x_t = \left[f(x_t,t) - \nabla_x \cdot gg^T(x_t,t) - gg^T(x_t,t)\nabla_x \log p_t(x)\right]\mathrm{d}t + g(x_t,t)\mathrm{d}\overline{W}_t \tag{4.9}$$

where the divergence of the matrix is

$$\nabla \cdot gg^T := \begin{bmatrix} \nabla \cdot (gg^T)_1 \\ \vdots \\ \nabla \cdot (gg^T)_d \end{bmatrix}$$

$p_t(x)$ is the density of forward process. $\overline{W}_t$ is reverse Wiener process(Brownian motion).

Also, the SDE of form (2.25) can be converted into an ordinary differential equation (ODE) without changing its marginal distributions $\{p_t(x)\}_{t=0}^{T}$. Thus by solving this ODE, we can sample from the same distributions as the reverse SDE. The corresponding ODE of an SDE is named **probability flow ODE**.

Lemma 4.4.2 *SDE in form of (2.25) have one-to-one correspondence to a* ***probability flow ODE****:*

$$\mathrm{d}x_t = \left[f(x_t,t) - \frac{1}{2}\nabla \cdot gg^T(x_t,t) - \frac{1}{2}gg^T(x_t,t)\nabla_x \log p_t(x)\right]\mathrm{d}t \tag{4.10}$$

The proof of Lemma 2.5.2 and Theorem 2.5.1 are based on the **Fokker-Plank equation(Kolmogorov forward equation)**, details are provided in Appendix.

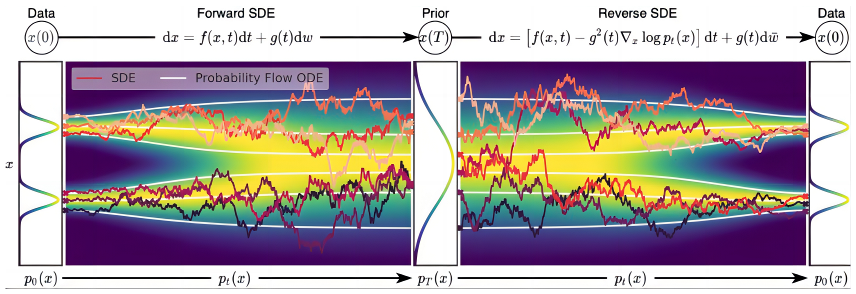

Data
Forward SDE
Prior
Reverse SDE
Data
$x(0)$
$\mathrm{d}x = f(x,t)\mathrm{d}t + g(t)\mathrm{d}w$
$x(T)$
$\mathrm{d}x = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]\,\mathrm{d}t + g(t)\mathrm{d}\bar{w}$
$x(0)$
SDE
Probability Flow ODE
x
$p_0(x)$
$p_t(x)$
$p_T(x)$
$p_t(x)$
$p_0(x)$

Figure 3: Comparison of SDE and probability flow ODE

When the score function is approximated via a time-dependent scored-based model, especially being a neural network, using this probability flow ODE allows faster sampling $x_0 \sim p_0 = p_{data}$ under the terminal condition $x_T \sim p_T$, which is almost a white noise. Typical solving method of this ODE relies on the discretization strategy.

4.5 Continuous DDPM

As for the continuous DDPM (2.27), there is an analytic solution for x_t that

$$x_t = x_0 e^{-\int_0^t \frac{\beta_s}{2}\mathrm{d}s} + \int_0^t \sqrt{\beta_s} e^{-\int_s^t \frac{\beta_r}{2}\mathrm{d}r} \mathrm{d}W_s \tag{4.11}$$

Then the distribution of x_t is

$$q(x_t|x_0) = \mathcal{N}\left(x_t; \underbrace{x_0 e^{-\int_0^t \frac{\beta_s}{2}\mathrm{d}s}}_{\mu_t}, \underbrace{\left(\int_0^t \beta_s e^{-\int_s^t \beta_r \mathrm{d}r}\mathrm{d}s\right)}_{\sigma_t} I_d \right) \tag{4.12}$$

where $\mu_t \to 0, t \to \infty$, $\sigma_t = O(1)$ Hence we can directly compute the reverse(backward) SDE of continuous DDPM:

$$\mathrm{d}x_t = \left[-\frac{\beta_t}{2} x_t + \beta_t \frac{x_t - \mu_t}{\sigma_t} \right] \mathrm{d}t + \sqrt{\beta_t}\mathrm{d}\overline{W}_t \tag{4.13}$$

4.6 Numerical method of forward SDE and probability flow ODE

The forward SDE (4.5) can be discretized as

$$x_{i+1} = x_i + f(x_i, t_i)\Delta t + g(x_i, t_i)\Delta W_i \tag{4.14}$$

where $\Delta W_i = W_{t_{i+1}} - W_{t_i} = \epsilon_i \sqrt{\Delta t}$ with $\epsilon_i \sim \mathcal{N}(0, I)$ i.i.d. It's a simple Euler-Maruyama scheme.

Similarly, suppose we use a time-dependent neural network $s_\theta(x_t, t)$ to estimate $\nabla_{x_t} \log p_t(x_t)$. After training to obtain an optimal approximator $s_{\theta^*}(x_t, t) \approx \nabla_{x_t} \log p_t(x_t)$, the probability flow (4.10) for the reverse process is discretized as

$$x_i = x_{i+1} - \left(f(x_{i+1}, t_{i+1}) - \frac{1}{2}\nabla_x \cdot gg^T(x_{i+1}, t_{i+1}) - \frac{1}{2} gg^T(x_{i+1}, t_{i+1}) s_{\theta^*}(x_{i+1}, t_{i+1}) \right) \Delta t \tag{4.15}$$

If for reducing complexity of computation, we can model $g(x_t, t) = g(t)$ to be x-independent, then the numerical discretization can be simplified as

$$x_i = x_{i+1} - \left(f(x_{i+1}, t_{i+1}) - \frac{1}{2} gg^T(x_{i+1}, t_{i+1}) s_{\theta^*}(x_{i+1}, t_{i+1}) \right) \Delta t \tag{4.16}$$

This scheme enables us to generate samples $x_0 = x_{t_0}$ from $x_{t_N} = x_T \sim \mathcal{N}(0, I)$.

4.7 Construction of Loss by Score-matching

The core idea of score matching is to match the score function (gradient of the log PDF) of a model to that of the true data distribution, rather than directly modeling the PDF itself. In our case, we plan to use a neural network $s_\theta(x_t, t)$ to estimate $\nabla_{x_t} \log p_t(x_t)$ through minimizing the Fisher divergence. By doing so, score matching can ignore the need for explicitly calculating the normalization constant of the PDF.

4.7.1 Time-independent Loss function

To approximate the score function, We use a neural network $s_\theta(x) = \nabla_x \log p_\theta(x)$ with parameter θ.

Similar to likelihood-based models, we can train score-based models by minimizing the Fisher divergence 2 between the model and the data distributions, defined as

$$L(\theta) = \frac{1}{2}\mathbb{E}_{p_{data}(x)} \left[\|\nabla_x \log p_{data}(x) - s_\theta(x)\|_2^2 \right] \tag{4.17}$$

intuitively, the Fisher divergence compares the squared ℓ^2 distance between the ground-truth data score and the score-based model. Directly computing this divergence, however, is infeasible because it requires access to the unknown data score $\nabla_x p_{data}(x)$.

Fortunately, there exists a family of methods called score matching that minimize the Fisher divergence without knowledge of the ground-truth data score. Score matching objectives can directly be estimated on a dataset and optimized with stochastic gradient descent, analogous to the log-likelihood objective for training likelihood-based models

Theorem 4.7.1 *Assume that the model score function $s_\theta(x)$ is differentiable, as well as some weak regularity conditions. Then, the objective function $L(\theta)$ can be expressed as $L(\theta) = J(\theta) + C$, where*

$$J(\theta) = \mathbb{E}_{p_{data}(x)}\left[tr\left(\nabla_x s_\theta(x)\right) + \frac{1}{2}\|s_\theta(x)\|_2^2\right] \tag{4.18}$$

C is a constant that does not depend on θ, and

$$\nabla_x s_\theta(x) = \nabla_x^2 \log p_\theta(x) \tag{4.19}$$

is the Hessian of the log-density function.

The proof is simply the integration by parts, provided in Appendix.

Remark 4.7.2 *The constant can be ignored and the following unbiased estimator of the remaining terms is used to train $p_\theta(x)$:*

$$\widehat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[tr\left(\nabla_x s_\theta(x_0^i)\right) + \frac{1}{2}\|s_\theta\|_2^2\right] \tag{4.20}$$

where a collection of N data points $\{x_0^1, ..., x_0^N\}$ are collected samples from $p_{data}(x_0)$.

4.7.2 Time-dependent Loss function

By build up a time-dependent neural network $s_\theta(x_t, t) = \nabla_{x_t}\log p_\theta(x_t)$ to estimate $\nabla_{x_t}\log p_t(x_t)$ from the forward SDE, the loss function is designed as

$$\theta^* = \arg\min_\theta \mathbb{E}_t\left\{\lambda(t)\mathbb{E}_{x_0}\mathbb{E}_{x_t|x_0}\left[\|s_\theta(x_t,t) - \nabla_{x_t}\log p_t(x_t|x_0)\|_2^2\right]\right\} \tag{4.21}$$

$$\approx \arg\min_\theta \mathbb{E}_t\left\{\frac{\lambda(t)}{N}\sum_{i=1}^{N}\mathbb{E}_{x_t|x_0^i}\left[\|s_\theta(x_t,t) - \nabla_{x_t}\log p_t(x_t|x_0^i)\|_2^2\right]\right\} \tag{4.22}$$

where $\lambda : [0, T] \to \mathbb{R}_+$ is a positive weighting function, t is uniformly sampled from $\mathcal{U}[0, T]$. The idea of this design aims to minimize the average ℓ^2 distance between $s_\theta(x_t, t)$ and $\nabla_{x_t}\log p_t(x_t)$, under uniformly sampled time variable t and some given weight $\lambda(t)$.

When $\lambda(t) = g^2(t)$, where $g(t)$ is diffusion coefficient independent about x_t, we have an important connection between our weighted combination of Fisher divergences and the KL divergence from $p_0 = p_{data}$ to p_θ under some regularity conditions:

$$D_{KL}(p_0(x)||p_\theta(x)) \leq \frac{T}{2}\mathbb{E}_t\mathbb{E}_{x_t}\left[\lambda(t)\|s_\theta(x_t,t) - \nabla_{x_t}\log p_t(x_t|x_0)\|_2^2\right] + D_{KL}(p_T||\pi) \tag{4.23}$$

After optimization $\theta^* = \arg\max_\theta J(\theta)$, $s_{\theta^*}(x)$ has best learned the main characters of input data(such as image and video data) in the sense of Fisher divergence. While it certainly differs from the real gradient of logarithm probability $\nabla_x p_{\theta^*}(x)$, which means the generated samples from $p_{\theta^*}(x)$(e.g. from Langevin dynamics),is not identically the real distribution $p_{data}(x)$. So the generated images may seem unreasonable or not possible to happen in real life.

5 Improvements

5.1 Sampling acceleration method

For an image generation model, there are three considerations: **1)** high quality samples, **2)** generating diversity, **3)** efficient and fast sampling. However, it is often difficult to trade-off between these three. Compared with traditional models for image generation (e.g. GAN,VAE), Diffusion models generate results of higher quality or even can exceed GAN, and the results have good diversity. However, Diffusion requires hundreds or even thousands of sampling steps, which leads to very slow training and inference. The data shows that the diffusion model may take nearly 1000 hours to sample 50,000 256*256 images on the same GPU. Therefore, most of the current optimisations of diffusion models focus on reducing the number of sampling steps, or increasing the sampling speed.

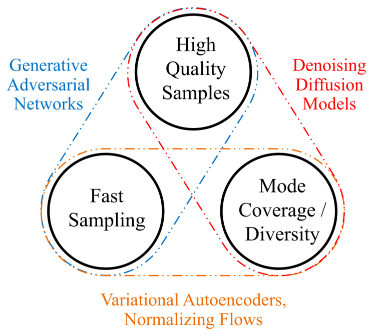

High Quality Samples
Generative Adversarial Networks
Denoising Diffusion Models
Fast Sampling
Mode Coverage / Diversity
Variational Autoencoders, Normalizing Flows

Figure 4: Generative learning trilemma

5.1.1 "Skipping steps"

In the discrete case, we consider that the parameter β_t is very small and therefore the noise removed in each generation step is weak. This results in an image that does not change much in successive steps, so we would like to eliminate some computations by 'skipping steps' while maintaining performance. In practice, this is not feasible in the denoising diffusion model. Because DDPM is based on a very important assumption, Markov property, we have to do the noise reduction step by step. Even if we predict the noise, we can't get x_0 directly from x_t. So if we make the process not depend on Markov property anymore, then it seems that we have a chance to implement the idea of skipping steps. After searching for information, we found that **Denoising Diffusion Implicit Model(DDIM)** implements this idea of ours.

Actually, DDIM is only a sampling model. Different from DDPM, we are not assuming that the forward process is a Markov chain in DDIM. Thereby $q(x_t|x_{t-1}, x_0)$ can not be replaced by $q(x_t|x_{t-1})$ in the backward process. Since the forward process in DDPM only uses that

$$x_T = \sqrt{A_T}x_0 + \sqrt{1 - A_T}\epsilon, \epsilon \sim \mathcal{N}(0, I) \tag{5.1}$$

This allows us to set any probability distribution that satisfies the equation.We can therefore adopt a form similar to the DDPM, and thus use the method of undetermined coefficients, assuming that

$$q(x_{t-1}|x_t, x_0) \sim \mathcal{N}(kx_0 + mx_t, \sigma^2 I) \tag{5.2}$$

Then we have $x_{t-1} = kx_0 + mx_t + \sigma\epsilon$. Since we still have $x_t = \sqrt{A_t}x_0 + \sqrt{1 - A_t}\epsilon'$, where ϵ and ϵ' are both follows a standard normal distribution, we have

$$\begin{aligned} x_{t-1} &= kx_0 + m(\sqrt{A_t}x_0 + \sqrt{1 - A_t}\epsilon') + \sigma\epsilon \\ &= (k + m\sqrt{A_t})x_0 + (m\sqrt{1 - A_t})\epsilon' + \sigma\epsilon \\ &= (k + m\sqrt{A_t})x_0 + \sqrt{m^2(1 - A_t) + \sigma^2}\epsilon \end{aligned}$$

Bringing back to the condition, we will get

$$k + m\sqrt{A_t} = \sqrt{A_{t-1}} \tag{5.3}$$
$$m^2(1 - A_t) + \sigma^2 = 1 - A_{t-1} \tag{5.4}$$

And by elementary arithmetic, we can easily conclude that $m = \frac{\sqrt{1-A_{t-1}-\sigma^2}}{\sqrt{1-A_t}}$ and $k = \sqrt{A_{t-1}} - \sqrt{1 - A_{t-1} - \sigma^2}\frac{\sqrt{A_t}}{\sqrt{1-A_t}}$. Finally we get the new distribution of $q(x_{t-1}|x_t, x_0)$,

$$q(x_{t-1}|x_t, x_0) \sim \mathcal{N}(\sqrt{A_{t-1}} - \sqrt{1 - A_{t-1} - \sigma^2}\frac{\sqrt{A_t}}{\sqrt{1 - A_t}}x_0 + \frac{\sqrt{1 - A_{t-1}} - \sigma^2}{\sqrt{1 - A_t}}x_t, \sigma^2 I) \tag{5.5}$$

Since we no longer need to obey the Markov property, we obey the following form when sampling:

$$x_s = \sqrt{A_s}(\frac{x_k - \sqrt{1 - A_t}\epsilon_\theta(x_t)}{\sqrt{A_k}}) + \sqrt{1 - A_s - \sigma^2}\epsilon_\theta(x_k) + \sigma\epsilon \tag{5.6}$$

Here, s should strictly satisfies $s < k$. Thus, we can then take a random ascending subsequence of length l from the time series $\{0, \cdots, T\}$. Sampling through the above equation iteratively l times finally gives us the x_0 we want. Then there is still a question about σ. In Denoising Diffusion Implicit Models(DDIM) [Song et al., 2022], the author have proved that whatever the σ is, it will not influence forward process condition holds. So we can pick an arbitrary value, such as $\sigma = \eta\sqrt{\frac{1-A_{t-1}}{1-A_t}\beta_t}$, $\eta \in [0, 1]$ ($\frac{1-A_{t-1}}{1-A_t}\beta_t$ is the variance in DDPM). When $\eta = 1$, it is DDPM. When $\eta = 0$, it is DDIM.

5.1.2 Integration with other generative models

In the field of image generation, we often compensate for each other's shortcomings by combining models. For example, combining Generative Adversarial Networks(GAN) with Diffusion Models(DM) can compensate for the low diversity of GAN generated samples and the slow generation of DM samples. However, since GAN approximates the generated samples to the real samples through the confrontation between the generator and the discriminator, it also often faces pattern collapse. And combining with DM does not improve this. Therefore the more common approach nowadays is to combine Variational Autoencoders(VAE) with DM.

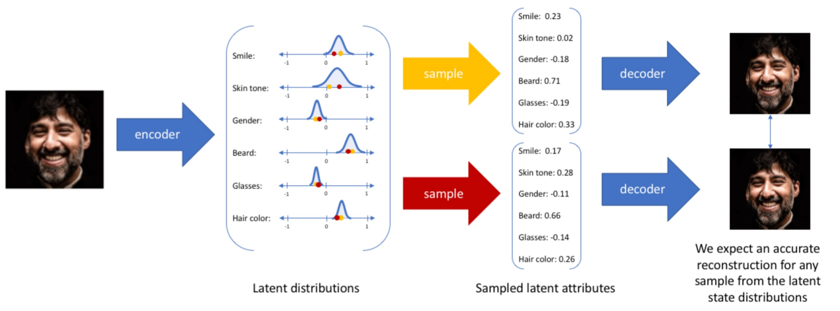


Figure 5: The process of VAE

In brief, Variational Autoencoder is a variant of Autoencoder. It maps the input image first into a probability distribution in the hidden space (one can assume that the prior distribution is a Gaussian distribution). A probabilistic decoder is then trained to achieve the mapping from the hidden space distribution to the real data distribution. Since the information in the hidden layer space is much smaller than the original image, the significant disadvantage of the VAE model is that the generated image is blurrier. In combination with the diffusion model, not only can the number of steps required for the convergence of the diffusion model be accelerated by appropriately reducing the dimensionality of the training data, which in turn speeds up the speed when sampling. It is also possible to make the image have higher clarity than VAE by training the diffusion model with multiple steps.

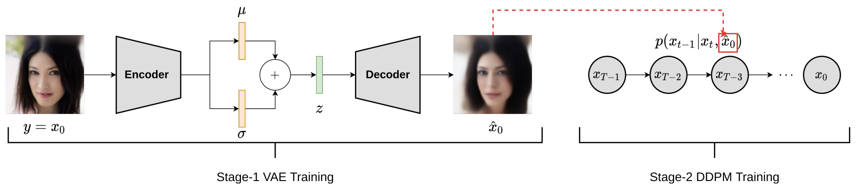


Figure 6: The training process of DiffusionVAE

5.2 Outlook and future opportunities

In this subsection, we point out some future research directions of diffusion models that are worthy of further investigation.

5.2.1 Efficiency Issues

Efficiency remains an issue that needs to be further improved in diffusion models, and future research could continue to explore lighter and faster diffusion models that maintain good performance while significantly reducing computational requirements. In addition to this, network lightweighting can be attempted. Make the network structure simple and improve the training efficiency.

5.2.2 Robustness and Generalization

On the one hand, real-world images may inherently have a wide variety of situations such as noise or missing data, so it is essential to study and enhance the diffusion model so that it can cope with complex image information. On the other hand, a superior generalisation capability can also increase the scalability of the model, for example by allowing the application of the model to be upgraded from 2D to 3D, which at the same time relies on more efficient sampling.

5.2.3 Integration of LLMs and Diffusion Models

Current large language models are capable of generating high-quality text, but for practical use it is necessary to have the model generate text that meets our desired requirements. Existing methods are either unable to incorporate multiple requirements or have strong limitations. The combination of a continuous diffusion model with a non-autoregressive language model can perform complex control tasks in a simpler way. And it has been experimentally demonstrated that the model achieves excellent results on tasks such as text length and fill-in-the-blank tasks.

6 Application

6.1 Generation

The most important application of diffusion models is still in the generative domain. As mentioned in the introduction section, popular generative models such as DALL-E 3, Stable diffusion 3, Sora, etc. all coincidentally use diffusion models to generate creative, high-quality images or videos. The transformation of text and images is achieved. In addition, the diffusion model can also be used for image restoration and enhancement. It is a major breakthrough in the field of computer vision.

6.2 Forecasting

Spatio-temporal data analyses rely fundamentally on a deep understanding of their intrinsic temporal dynamics. At the core of these forecasting is the generation of temporal data samples for specific purposes in a conditional or unconditional manner. In this regard, diffusion model serves as a powerful generative framework that effectively fills the gap in spatio-temporal data generation by training on large-scale temporal data. A wide variety of predictive models based on diffusion models are available for a wide range of domains such as healthcare, climate and weather, energy and power, etc.. And it exhibits signiffcant potential in solving the puzzle of next-generation, LLM-empowered temporal data-centric agents.

7 Conclusion

In this research, we present diffusion models that are widely used today for generative and predictive tasks. Firstly we classify diffusion models into conditional and unconditional diffusion models according to the type of task. Secondly a brief overview of the basic principles of diffusion models is given. Then we introduce two commonly used unconditional diffusion models in turn.In DDPM, we use Bayesian formula to calculate $q(x_{t-1}|x_t, x_0)$ and model $p_\theta(x_{t-1}|x_t)$ as Gaussian to approximate it, which equals to approximating the mean.We decompose VLB to minimize loss function and at last our target is to predict the noise generated from the last step. In Scored-based model, we use a neural network $s_\theta(x_t, t)$ to estimate the score funtion $\nabla_{x_t} \log p_t(x_t)$, which avoid the direct computation of intractable normalizer, and enables us to approximately sample from real data distribution via Langevin dynamics. Apart from that score-based model can be combined with SDE and probability flow ODE, that can generate new images/video data through solving reverse SDE/ODE from pure white noises. Based on this, we summarise two improved scenarios by combining our reflections and some existing improved models. Finally, we offer some observations on the application and future development of diffusion models.

8 Contribution

In this project, all three of us worked on the presentation and this research. The specific contribution in terms of content is as follows:

- Wennan Wang is responsible for the details of score-based diffusion modelling and DDPM.
- Jingrong Guan is responsible for the background and improvements(DDIM and VAE).
- Lecong Ding is responsible for the details in denoising diffusion probability model.

References

[1] William K. Bertram, Analytic solutions for optimal statistical arbitrage trading, Physica A: Statistical Mechanics and its Applications, Volume 389, Issue 11, 2010, Pages 2234-2243, ISSN 0378-4371, https://doi.org/10.1016/j.physa.2010.01.045.

[2] Santos, Javier E. and Yen Ting Lin. "Using Ornstein-Uhlenbeck Process to understand Denoising Diffusion Probabilistic Model and its Noise Schedules." ArXiv abs/2311.17673 (2023): n. pag.

[3] Song, Yang, Conor Durkan, Iain Murray and Stefano Ermon. "Maximum Likelihood Training of Score-Based Diffusion Models." Neural Information Processing Systems (2021).

[4] Turner, Richard E., Cristiana-Diana Diaconu, Stratis Markou, Aliaksandra Shysheya, Andrew Y. K. Foong and Bruno Mlodozeniec. "Denoising Diffusion Probabilistic Models in Six Simple Steps." ArXiv abs/2402.04384 (2024): n. pag.

[5] Ho, Jonathan, Ajay Jain and P. Abbeel. "Denoising Diffusion Probabilistic Models." ArXiv abs/2006.11239 (2020): n. pag.

[6] Yang, Yiyuan, Ming Jin, Haomin Wen, Chaoli Zhang, Yuxuan Liang, Lintao Ma, Yi Wang, Cheng-Ming Liu, Bin Yang, Zenglin Xu, Jiang Bian, Shirui Pan and Qingsong Wen. "A Survey on Diffusion Models for Time Series and Spatio-Temporal Data." (2024).

[7]Nichol, Alex and Prafulla Dhariwal. "Improved Denoising Diffusion Probabilistic Models." ArXiv abs/2102.09672 (2021): n. pag.

[8] Sohl-Dickstein, Jascha Narain, Eric A. Weiss, Niru Maheswaranathan and Surya Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics." ArXiv abs/1503.03585 (2015): n. pag.

[9] Xiao, Zhisheng, Karsten Kreis and Arash Vahdat. "Tackling the Generative Learning Trilemma with Denoising Diffusion GANs." ArXiv abs/2112.07804 (2021): n. pag.

[10] Dhariwal, Prafulla and Alex Nichol. "Diffusion Models Beat GANs on Image Synthesis." ArXiv abs/2105.05233 (2021): n. pag.

Appendix: Detailed proofs about theorems and properties

A Convergence of forward process in DDPM

Proof 1 (Proof of Proposition (3.1.1)**)** *Denote* $\alpha_t = 1 - \beta_t \in (0,1)$*, then we have*

$$\begin{aligned}
x_t &= \sqrt{\alpha_t} x_{t-1} + \sqrt{\beta_t}\epsilon_t \\
&= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{\beta_{t-1}}\epsilon_{t-1}) + \sqrt{\beta_t}\epsilon_t \\
&= \sqrt{\alpha_t\alpha_{t-1}} x_{t-2} + \underbrace{\left(\sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-1} + \sqrt{1-\alpha_t}\epsilon_t\right)}_{\sim\mathcal{N}(0,(\alpha_t(1-\alpha_{t-1})+(1-\alpha_t))I)=\mathcal{N}(0,(1-\alpha_t\alpha_{t-1})I)} \\
&= \sqrt{\alpha_t\alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\cdot z_2, \qquad z_2 \sim \mathcal{N}(0,I) \\
&= \sqrt{\alpha_t\alpha_{t-1}}(\sqrt{\alpha_{t-2}} x_{t-3} + \sqrt{1-\alpha_{t-2}}\cdot\epsilon_{t-2}) + \sqrt{1-\alpha_t\alpha_{t-1}}\cdot\bar{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}\alpha_{t-2}} x_{t-3} + \underbrace{\left(\sqrt{\alpha_t\alpha_{t-1}(1-\alpha_{t-2})}\cdot\epsilon_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\cdot\bar{\epsilon}_{t-1}\right)}_{\sim\mathcal{N}(0,(1-\alpha_t\alpha_{t-1}\alpha_{t-2})I)} \\
&= \sqrt{\alpha_t\alpha_{t-1}\alpha_{t-2}} x_{t-3} + \sqrt{1-\alpha_t\alpha_{t-1}\alpha_{t-2}}\cdot z_3, \qquad z_3 \sim \mathcal{N}(0,I)
\end{aligned}$$

Following this, we can use induction to derive that marginal distribution of x_t *that*

$$x_t = \sqrt{\prod_{i=1}^{t}\alpha_t}\cdot x_0 + (1 - \prod_{i=1}^{t}\alpha_t)\cdot z_t, \quad z_t \sim \mathcal{N}(0,I) \tag{A.1}$$

Then $x_t \sim \mathcal{N}(\sqrt{\prod_{i=1}^t \alpha_i}\cdot x_0, (1-\sqrt{\prod_{i=1}^t \alpha_i})I)$.

If $\prod_{i=1}^t \alpha_i \to 0$ *as* $t\to\infty$*, then obviously we have*

$$q(x_t|x_0) = \mathcal{N}(\sqrt{\prod_{i=1}^{t}\alpha_i}\cdot x_0, (1-\sqrt{\prod_{i=1}^{t}\alpha_i})I) \to \mathcal{N}(0,I), \text{ as } t\to\infty$$

B Backward process of DDPM

Proof 2 (Proof of Proposition (3.2.1)**)**

$$\begin{aligned}
q(x_{t-1}|x_t,x_0) &= \frac{q(x_t|x_{t-1},x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)} \\
&\propto \exp\left[-\frac{1}{2}\left(\frac{(x_t-\sqrt{\alpha_t}x_{t-1})^2}{\beta_t} + \frac{(x_{t-1}-\sqrt{A_t}x_0)^2}{1-A_{t-1}} - \frac{(x_t-\sqrt{A_t}x_0)^2}{1-A_t}\right)\right] \\
&\propto \exp\left[-\frac{1}{2}\left(\underbrace{\left(\frac{\alpha_t}{\beta_t}+\frac{1}{1-A_{t-1}}\right)}_{=1/\widetilde{\beta}_t\text{: variance of } x_{t-1}} x_{t-1}^2 - 2\underbrace{\left(\frac{\sqrt{\alpha_t}}{\beta_t}x_t + \frac{\sqrt{A_t}}{1-A_{t-1}}x_0\right)}_{=:u(x_t,x_0)} x_{t-1}\right)\right] \\
&\propto \exp\left[-\frac{1}{2\widetilde{\beta}_t}\left(x_{t-1}-\widetilde{\beta}_t u(x_t,x_0)\right)^2\right]
\end{aligned}$$

Hence we have

$$\begin{aligned}\widetilde{\beta}_t &= \frac{1}{\frac{\alpha_t}{\beta_t}+\frac{1}{1-A_{t-1}}} = \frac{1-A_{t-1}}{\alpha_t(1-A_{t-1})+\beta_t}\beta_t \\ &= \frac{1-A_{t-1}}{1-A_t}\beta_t \\ \widetilde{\mu}(x_t,x_0) &= \frac{1-A_{t-1}}{1-A_t}\beta_t\cdot\left(\frac{\sqrt{\alpha_t}}{\beta_t}x_t+\frac{\sqrt{A_t}}{1-A_{t-1}}x_0\right) \\ &= \frac{\sqrt{\alpha_t}(1-A_{t-1})}{1-A_t}x_t+\frac{\sqrt{A_{t-1}}\beta_t}{1-A_t}x_0\end{aligned}$$

C Design of Loss function via Variational Lower Bound in DDPM

Proof 3 (Proof of Lemma (3.4.1)**)**

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{q(x_0)}[-\log p_\theta(x_0)] = -\mathbb{E}_{q(x_0)}\left[\log\left(\mathbb{E}_{p_\theta(x_{1:T})}[p_\theta(x_{0:T})]\right)\right] \\ &= -\mathbb{E}_{q(x_0)}\left[\log\left(\int\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}q(x_{1:T}|x_0)\mathrm{d}x_{1:T}\right)\right] \\ &\overset{\textit{Jensen}}{\leq} -\mathbb{E}_{q(x_0)}\left[\int\log\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}q(x_{1:T}|x_0)\mathrm{d}x_{1:T}\right] \\ &= \int q(x_0)\mathrm{d}x_0\int\log\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}q(x_{1:T}|x_0)\mathrm{d}x_{1:T} \\ &= \int\log\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}q(x_{0:T})\mathrm{d}x_{0:T} \\ &= \mathbb{E}_{q(x_{0:T})}\left[\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right] = \mathcal{L}_{\mathit{VLB}}\end{aligned}$$

Proof 4 (Proof of Lemma (3.4.2)**)**

$$
\begin{aligned}
\mathcal{L}_{VLB} &= \mathbb{E}_{q(x_{0:T})}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right] = \mathbb{E}_{q(x_{0:T})}\left[\log \frac{\prod_{t=1}^{T} q(x_t|x_{t-1})}{p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)}\right] \\
&= \mathbb{E}_{q(x_{0:T})}\left[-\log p_\theta(x_T) + \sum_{t=1}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)}\right] \\
&= \mathbb{E}_{q(x_{0:T})}\left[-\log p_\theta(x_T) + \sum_{t=2}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}\right] \\
&= \mathbb{E}_{q(x_{0:T})}\left[-\log p_\theta(x_T) + \sum_{t=2}^{T} \log \left(\frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)} \cdot \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)}\right) + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}\right] \\
&= \mathbb{E}_{q(x_{0:T})}\left[-\log p_\theta(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)} + \sum_{t=2}^{T} \log \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}\right] \\
&= \mathbb{E}_{q(x_{0:T})}\left[-\log p_\theta(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_T|x_0)}{q(x_1|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}\right] \\
&= \mathbb{E}_{q(x_{0:T})}\left[\underbrace{\log \frac{q(x_T|x_0)}{p_\theta(x_T)}}_{L_T} + \sum_{t=2}^{T} \underbrace{\log \frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)}}_{L_{t-1}} \underbrace{- \log p_\theta(x_0|x_1)}_{L_0}\right] \\
&= \mathbb{E}_{q(x_0)}\left[\underbrace{D_{KL}(q(x_T|x_0)||p_\theta(x_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{KL}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{- \log p_\theta(x_0|x_1)}_{L_0}\right]
\end{aligned}
$$

D Reverse SDE of diffusion process

We first verify the correspondence between SDE and probability flow ODE.

Proof 5 (Proof of Lemma (4.4.2)**)** *Recall that the forward density $p_t(x)$ satisfies Fokker-Planck equation:*

$$\frac{\partial p_t(x)}{\partial t} = -\nabla_x \cdot (f(x_t,t)p_t(x)) + \frac{1}{2}\nabla_x \cdot \left(\nabla_x \cdot \left(gg^T(x_t,t)p_t(x)\right)\right)$$

using some simple calculas we have

$$
\begin{aligned}
\frac{\partial p_t(x)}{\partial t} &= -\nabla_x \cdot \left[f(x_t,t)p_t(x) - \frac{1}{2}\left(\nabla_x \cdot gg^T(x_t,t)\right)p_t(x) - \frac{1}{2}gg^T(x_t,t)\nabla_x p_t(x)\right] \\
&= -\nabla_x \cdot \left[\left(f(x_t,t) - \frac{1}{2}\nabla_x \cdot gg^T(x_t,t) - \frac{1}{2}gg^T(x_t,t)\frac{1}{p_t(x)}\nabla_x p_t(x)\right)p_t(x)\right] \\
&= -\nabla_x \cdot \left[\left(f(x_t,t) - \frac{1}{2}\nabla_x \cdot gg^T(x_t,t) - \frac{1}{2}gg^T(x_t,t)\nabla_x \log p_t(x)\right)p_t(x)\right] \\
&\triangleq -\nabla \cdot (\widetilde{f}(x_t,t)p_t(x))
\end{aligned}
$$

Compare the result with above Fokker-Planck equation, the same stochastic process is equivalent to the following ***probability flow ODE****:*

$$\mathrm{d}x_t = \widetilde{f}(x_t,t)\mathrm{d}t = \left(f(x_t,t) - \frac{1}{2}\nabla_x \cdot gg^T(x_t,t) - \frac{1}{2}gg^T(x_t,t)\nabla_x \log p_t(x)\right)\mathrm{d}t \tag{D.1}$$

or $\frac{\mathrm{d}x_t}{\mathrm{d}t} = \widetilde{f}(x_t,t)$.

Proof 6 (Proof of Theorem (4.4.1)**)** *The benefit to reduce SDE into ODE is that the reverse probability flow ODE is very simple to express: Let T be the time terminal, and $\tau = T - t$ the reverse time, then the reverse ODE is*

$$\frac{\mathrm{d}x}{\mathrm{d}\tau} = -\widetilde{f}(x_{T-\tau}, T-\tau)$$

Denote $\widetilde{p}_\tau(x) = p_{T-\tau}(x)$, $\widetilde{x}_\tau = x_{T-\tau}$ the density and state of reverse process, the Fokker-Planck equation for the reverse process is given by

$$\begin{aligned}
\frac{\partial \widetilde{p}_\tau(x)}{\partial \tau} &= -\nabla_x \cdot \Big(-\widetilde{f}(\widetilde{\tau}, T-\tau)\widetilde{p}_\tau(x)\Big) \\
&= -\nabla_x \cdot \left[\widetilde{p}_\tau(x) \left(-f(\widetilde{x}_\tau, T-\tau) + \frac{1}{2}\left(\nabla_x \cdot gg^T(\widetilde{x}_\tau, T-\tau) + gg^T(\widetilde{x}_\tau, T-\tau)\nabla_x \log \widetilde{p}_\tau(x)\right)\right)\right] \\
&= -\nabla_x \cdot \left[\widetilde{p}_\tau(x)\left(-f(\widetilde{x}_\tau, T-\tau) + \left(\nabla_x \cdot gg^T(\widetilde{x}_\tau, T-\tau) + gg^T(\widetilde{x}_\tau, T-\tau)\nabla_x \log \widetilde{p}_\tau(x)\right)\right)\right] \\
&+ \frac{1}{2}\nabla_x \cdot \left[\widetilde{p}_\tau(x)\left(\nabla_x \cdot gg^T(\widetilde{x}_\tau, T-\tau) + gg^T(\widetilde{x}_\tau, T-\tau)\nabla_x \log \widetilde{p}_\tau(x)\right)\right] \\
&= -\nabla_x \cdot \left[\widetilde{p}_\tau(x) \underbrace{\left(-f(\widetilde{x}_\tau, T-\tau) + \left(\nabla_x \cdot gg^T(\widetilde{x}_\tau, T-\tau) + gg^T(\widetilde{x}_\tau, T-\tau)\nabla_x \log \widetilde{p}_\tau(x)\right)\right)}_{\triangleq f_r(\widetilde{x}_\tau, T-\tau)}\right] \\
&+ \frac{1}{2}\nabla \cdot \left(\nabla \cdot \left(gg^T(\widetilde{x}_\tau, T-\tau)\widetilde{p}_\tau(x)\right)\right) \\
&= -\nabla \cdot \left(f_r(\widetilde{x}_\tau, T-\tau)\widetilde{p}_\tau(x)\right) + \frac{1}{2}\nabla \cdot \left(\nabla \cdot \left(gg^T(\widetilde{x}_\tau, T-\tau)\widetilde{p}_\tau(x)\right)\right)
\end{aligned}$$

Hence we have the first version of reverse SDE:

$$\mathrm{d}\widetilde{x}_\tau = f_r(\widetilde{x}_\tau, T-\tau)\mathrm{d}\tau + g(\widetilde{x}_\tau, T-\tau)\mathrm{d}W_\tau \tag{D.2}$$

If changing the time variable from τ to t, we finally obtain that

$$\begin{aligned}
\mathrm{d}x_t &= -f_r(x_t, t)\mathrm{d}t + g(x_t, t)\mathrm{d}W_t \\
&= \left[f(x_t, t) - \nabla_x \cdot gg^T(x_t, t) - gg^T(x_t, t)\nabla_x \log p_t(x)\right]\mathrm{d}t + \mathrm{d}W_t
\end{aligned}$$

Proof 7 (Proof of Theorem (4.7.1)**)**

$$\begin{aligned}
L(\theta) &= \frac{1}{2}\mathbb{E}_{p_{data}(x)}\left[\|\nabla_x \log p_{data}(x) - s_\theta(x)\|_2^2\right] \\
&= \int \left(\underbrace{\frac{1}{2}\|\nabla_x \log p_{data}(x)\|_2^2}_{= \text{constant w.r.t. } \theta} + \frac{1}{2}\|s_\theta(x)\|_2^2 - s_\theta^T(x)\nabla_x \log p_{data}(x)\right) p(x)\mathrm{d}x \\
&= \int \left(\frac{1}{2}\|s_\theta(x)\|_2^2 - s_\theta^T(x)\frac{\nabla_x p_{data}(x)}{p_{data}(x)}\right) p_{data}(x)\mathrm{d}x + C \\
&= \mathbb{E}_{p_{data}(x)}\left[\frac{1}{2}\|s_\theta(x)\|_2^2\right] - \int \sum_{i=1}^d s_\theta^i(x)\partial_i p_{data}(x)\mathrm{d}x + C \\
&= \mathbb{E}_{p_{data}(x)}\left[\frac{1}{2}\|s_\theta(x)\|_2^2\right] + \sum_{i=1}^d \int p_{data}(x)\partial_i s_\theta^i(x)\mathrm{d}x + C \\
&= \mathbb{E}_{p_{data}(x)}\left[\frac{1}{2}\|s_\theta(x)\|_2^2 + tr(\nabla_x s_\theta(x))\right] + C \\
&\triangleq J(\theta) + C
\end{aligned}$$