Diffusion models

A straight to the point yet technical explanation of

Diffusion Models

Author: Diego Biagini

The diffusion process

The basic diffusion process

Diffusion models are generative models based around the idea of a **Diffusion Process**

Let's say our samples come from a ground truth data distribution: $x_0 \sim q(x_0)$

A discrete-time diffusion process can be obtained by adding a small amount of noise to an original image, in T steps, this results in a sequence of increasingly noisier images $x_1, \ldots, x_T$

An image which has passed through t noising steps will be referred from now on as x_t

The noise that is usually added is Gaussian noise, this means that for high values of t these new noised images will resemble more and more pure Gaussian noise

In particular we **want** the distribution $q(x_T|x_0)$ to resemble a standard Gaussian

Diffusion step

We can define this noising transformation through a set of hyperparameters, which are called a **variance schedule:**

$$\{\beta_t \in (0,1)\}_{t=1}^{T}$$

Using these parameters we can define the distribution of each x_t given the slightly less noisy image before it:

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}\, x_{t-1}, \beta_t I\right)$$

However to obtain a sample noised t times we don't actually need to sample from these distributions for each noising step, we can actually obtain a sample at a certain noise level directly knowing x_0:

Let: $\alpha_t = 1 - \beta_t$ $\bar{\alpha}_t = \prod_{i=0}^{t} \alpha_t$ Then we can define: $q(x_t|x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)I\right)$

Noising schedule

Many possible noising schedules β_t have been devised, in general it is very preferable to have a strictly increasing schedule $\beta_1 < \beta_2 < \cdots < \beta_T$

Various possibilities are the following ones:
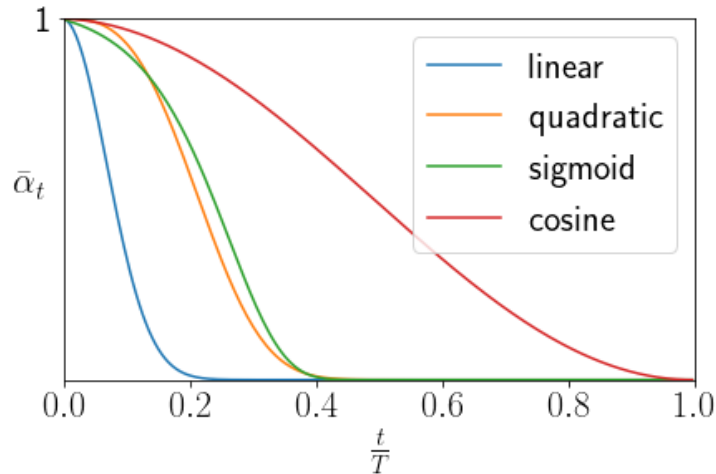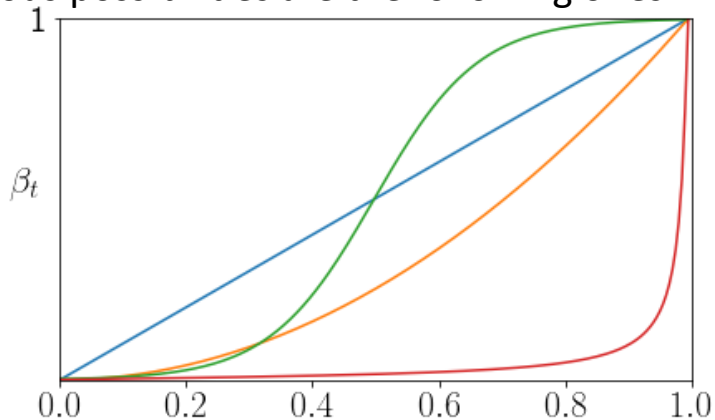
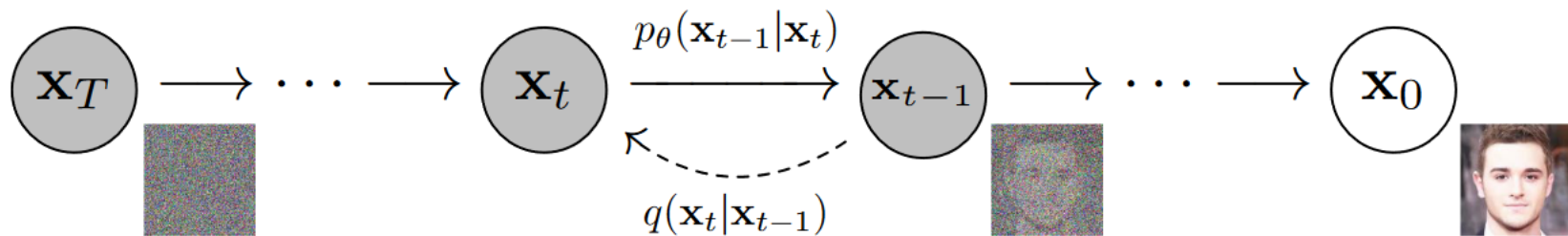For details on each of these schedules refer to: lucidrains/denoising-diffusion-pytorch

Image credit: DDPM paper

Reversing the diffusion process

Reverse diffusion process

To generate samples from the data-generating distribution $q(x_0)$ we need to have a way to know the reverse diffusion process, i.e. we need to know the distribution: $q(x_{t-1}|x_t)$

If we have such a distribution we can generate a sample $x_0 \sim q(x_0)$ by:

- Sampling $x_T \sim q(x_T) = \mathcal{N}(0, I)$
- Following the reverse diffusion chain by iteratively sampling from $q(x_{t-1}|x_t)$ until we reach x_0

Thankfully for us, if β_t is small enough then $q(x_{t-1}|x_t)$ is actually a Gaussian

And what we want to do is use a neural network to estimate the moments of this Gaussian:
$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\big(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\big)$$

Estimating $q(x_{t-1}|x_t)$

Why do we need a neural network estimation to obtain $q(x_{t-1}|x_t)$ instead of finding it analytically?

Because deriving it would require integrating over the entire data distribution!

However it becomes tractable if we know the original sample x_0:

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

In fact using Bayes' rule we have that: $q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0) \frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}$

This is a product of Gaussians which ultimately results in a Gaussian with parameters:

$$\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon_t\right) \qquad \tilde{\beta}(t) = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t$$

Where ϵ_t is the standard Gaussian noise applied to x_0 to obtain x_t

Obtaining the best generator(1)

Our objective is to maximize the likelihood of our novel data-generating distribution: $p_\theta(x_0)$

This objective can be obtained by minimizing the divergence between the estimated reverse distribution and the actual reverse distribution of the entire process

This is pretty much the same setup as a VAE, we can say that we want to minimize the following loss function, which amounts to maximizing the ELBO:

$$L_{VLB} = \mathbb{E}_{q(x_{0:T})}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right] \geq -\mathbb{E}_{q(x_0)}[\log p_\theta(x_0)]$$

After a lot of steps, this loss can be decomposed in a list of terms, one for each noising level t:

$$L_{VLB} = L_T + L_{T-1} + \cdots + L_0$$

Obtaining the best generator (2)

Such loss terms are defined as follows:

$$L_T = KL(q(x_T|x_0) \parallel p_\theta(x_T))$$
$$L_t = KL(q(x_t|x_{t+1}, x_0) \parallel p_\theta(x_t|x_{t+1})) \text{ for } 1 \leq t \leq T - 1$$
$$L_0 = -\log p_\theta(x_0|x_1)$$

We can notice that each of them other than L_0 is a KL-divergence between Gaussians, which can be computed explicitly.

Furthermore L_T is a constant and can be ignored.

Finally L_0 can be optimized by computing a discretized Gaussian log-likelihood, since it is our first step and we know that x_0 has no noise.

What is important to notice is the fact that we can optimize over these terms **one by one**.

The full training procedure

Algorithm 2 Optimizing L_{vlb}

repeat

    sample real data $x_0 \sim q(x_0)$

    sample noise level t

    $\epsilon \sim \mathcal{N}(0, I)$

    $x_t \leftarrow \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$

    Obtain the true posterior moments $\tilde{\mu}_t(x_t, x_0)$ $\tilde{\beta}_t$

    Get noise and variance predictions from the model: $\epsilon_\theta(x_t, t)$ $\Sigma_\theta(x_t, t)$

    Obtain predicted unnoised image: $\tilde{x}_0(x_t, \epsilon_\theta(x_t, t))$

    if $t \neq 1$ **then**

        Compute KL-div between true and estimated Gaussian

        $L_{vlb} \leftarrow KL(\mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I), \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, \tilde{x}_0), \Sigma_\theta(x_t, t)I))$

    else

        Compute discretized Gaussian log likelihood

        $L_{vlb} \leftarrow -DGLL(x_0; \mathcal{N}(\tilde{x}_0; \tilde{\mu}(x_t, \tilde{x}_0), \Sigma_\theta(x_t, t)I))$

    end if

    Optimization step

until converged

In practice

Simplified training procedure

We don't need our model to learn the complex $\tilde{\mu}_t(x_t, t)$ term, or even x_0, we can just let it predict the noise ϵ_t (which when added to x_0 results in x_t) and use this result to rewrite L_t:

$$L_t = \mathbb{E}_{x_0,\epsilon}[\frac{(1 - \alpha_t)^2}{2\alpha_t(1 - \bar{\alpha}_t) \parallel \Sigma \parallel_2^2} \parallel \epsilon_t - \epsilon_\theta(x_t, t) \parallel_2^2]$$

It also turns out that we don't need the weighting term and we can use a simplified objective:

$$L_t^{simple} = \mathbb{E}_{t\sim[1,T],x_0,\epsilon}[\parallel \epsilon_t - \epsilon_\theta(x_t, t) \parallel_2^2]$$

Which is used in the following training procedure:

Algorithm 1 Training

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

Basic sampling(DDPM)

The sampling procedure consists in traversing the chain of diffusion steps backwards, starting from pure noise.

Each of these steps consists in:

- Use the network to estimate the added noise $\epsilon_\theta(x_t, t)$
- Obtain estimates for the moments of the reverse process: $\mu_\theta(x_t, t), \tilde{\beta}_t$
- Sample from it: $x_{t-1} \sim \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \tilde{\beta}_t I)$

The shorthand algorithm is the following:

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **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** $\mathbf{x}_0$

Strided Sampling

Usually Diffusion models are trained with a fairly high amount of noise levels (~ 1000), this means that we would need to perform 1000 forward passes to sample an image.

An alternative is to use a strided schedule.

Let's say our original schedule is composed of T steps but we want to only perform a subset S of them.

We can consider the strided values $\bar{\alpha}_{S_t}$ and use them to obtain corresponding sampling variances:

$$\beta_{S_t} = 1 - \frac{\bar{\alpha}_{S_t}}{\bar{\alpha}_{S_{t-1}}}, \qquad\qquad \tilde{\beta}_{S_t} = \frac{1 - \bar{\alpha}_{S_{t-1}}}{1 - \bar{\alpha}_{S_t}} \beta_{S_t}$$

Now we can traverse the chain using the distribution: $p_\theta\left(x_{S_{t-1}}|x_{S_t}\right) = \mathcal{N}\left(\mu_\theta\left(x_{S_t}, S_t\right), \tilde{\beta}_{S_t} I\right)$

DDIM

The strided sampling approach works well but it still requires a reasonable amount of steps to obtain good results.

Many sampling methods have been devised to overcome this limitation, one of them is DDIM.

The core idea of DDIM is to control how much stochasticity we inject in each of the steps, through a parameter η.

In fact we define the variance of the reverse process as: $\sigma_t^2 = \eta \cdot \tilde{\beta}_t$

And we define the reverse process distribution as:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; x_{0,\theta}\sqrt{\bar{\alpha}_t} + \sqrt{1 - \bar{\alpha}_t - \sigma^2}\epsilon_\theta, \sigma_t^2 I)$$

With $\eta = 1$ we obtain DDPM, with $\eta = 0$ we have pure DDIM, which exhibits better results when we provide it with a strided sampling schedule with few steps.
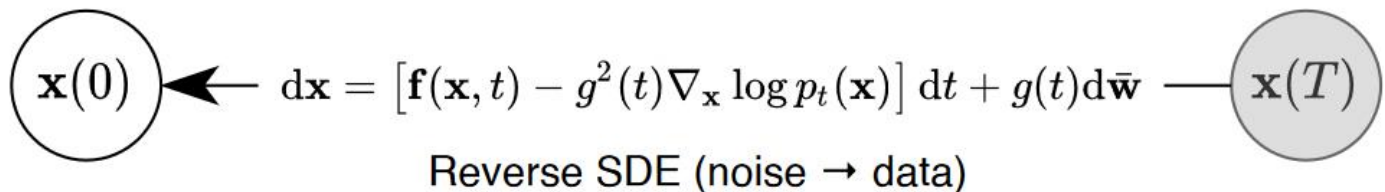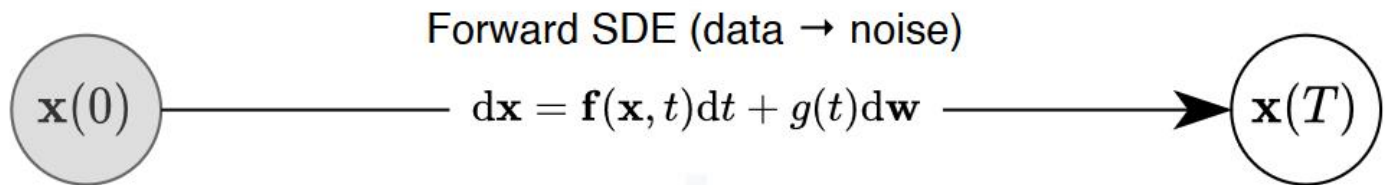
Forward SDE (data → noise)

$\mathbf{x}(0)$ ——————— $\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$ ——————→ $\mathbf{x}(T)$

$\mathbf{x}(0)$ ←——— $\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$ ——— $\mathbf{x}(T)$

Reverse SDE (noise → data)

Image credit: Song et al. (2021)

Continuous Diffusion

Diffusion models and NCSN

Diffusion models have many equivalences with similar families of models, one of these are Noise Conditioned Score Networks(NCSN).

NCSN are able to generate samples from a distribution $q(x)$ through the so-called **score function**, defined as $\nabla_x \log q(x)$ and estimated through a score network s_θ, trained such that $s_\theta(x) \approx \nabla_x \log q(x)$

If we consider a DDPM we have the following equivalency with NCSN's score network/function:

$$\nabla_{x_t} \log q(x_t) \approx s_\theta(x_t, t) = -\frac{\epsilon_\theta(x_t, t)}{\sqrt{1 - \bar{\alpha}_t}}$$

This formulation of the score function will come in handy in many different places.

SDE formulation

Another way to formulate a diffusion process is through a continuous formulation, which employs stochastic differential equations.

In this approach the diffusion process is defined as $\{x_t\}_{t=0}^T$, indexed by a **continuous** time variable t , such that $x_0 \sim q$ are samples from the data distribution and $x_T \sim q_T$ are noise samples.

Furthermore the schedule is now defined as a continuous function $\beta(t): \mathbb{R} \to \mathbb{R}$

This forward process can be modeled as the solution to an Itō SDE:
$$dx = f(x,t)dt + g(t)dw$$

From this we can obtain a reverse process, whose evolution is given by the reverse-time SDE:
$$dx = \left[f(x,t) - g(t)^2 \nabla_{x_t} \log q(x_t)\right]dt + g(t)d\bar{w}$$

Sampling from an SDE

If we can compute the score function for all t we can effectively derive the reverse diffusion process and simulate it to sample from the data-generating distribution.

In particular we an either use specific or general purpose solvers, one of these is the Euler-Maruyama procedure:

Initialize $\quad t \leftarrow T, \quad \mathbf{x} \sim p_T(\mathbf{x})$

Repeat until $t = 0$ $\begin{cases} \Delta\mathbf{x} \leftarrow [\boldsymbol{f}(\mathbf{x}, t) - g^2(t)\boldsymbol{s_\theta}(\mathbf{x}, t)]\Delta t + g(t)\mathbf{z} \\ \mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x} \\ t \leftarrow t + \Delta t \end{cases}$

source

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, |\Delta t|\boldsymbol{I})$$

ODE Formulation

For all reverse diffusion processes defined through SDEs there exists a corresponding deterministic process (so defined through an ordinary differential equation) whose trajectories share the same marginal probability densities as the SDE.

This is called a **probability flow ODE**:

$$dx = \left[f(x,t) - \frac{1}{2} g(t)^2 \nabla_x \log p_t(x) \right] dt$$

Samplers can work on either the SDE or ODE formulation.

A big difference between the two is the fact that unlike SDE reverse diffusion samplers or discrete sampling with DDPM, with ODEs there is no additional randomness once the initial sample is obtained from the prior distribution.

RELATIONSHIP BETWEEN DISCRETE AND CONTINuous

We can see that DDPM is actually a discretization of a certain SDE (called the VP-SDE).

$$dx = -\frac{1}{2}\beta(t)x\, dt + \sqrt{\beta(t)}\, dw$$

And the process of following the reverse diffusion chain is a discretization of the reverse-time SDE and is called ancestral sampling*.

Meanwhile pure DDIM is a discretization of the Probability Flow ODE obtained by removing the stochastic term of the VP-SDE, after all in both cases we are removing the stochasticity.
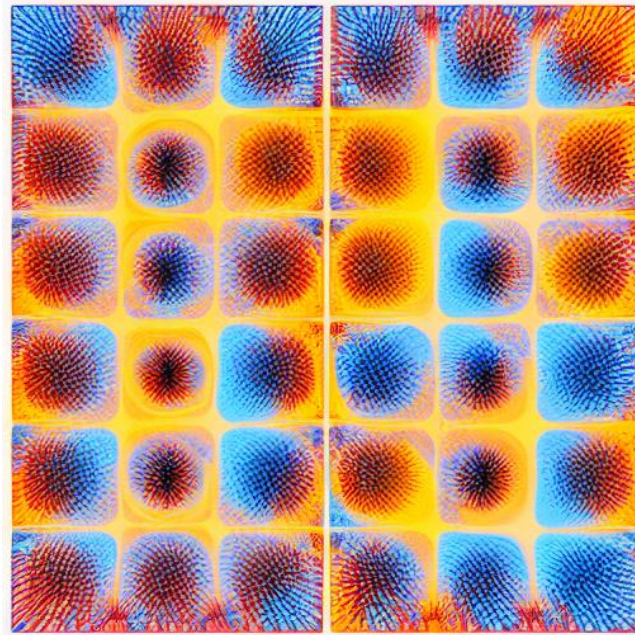
Elucidating diffusion

Elucidating diffusion formulation

Karras et al. gave a generalization of all the different formulations, which is presented in the next 2 slides.

Let's now define a series of noise levels $\{\sigma_t\}_{t=0}^{N}$, such that $\sigma_0 = \sigma_{\max}$ is the noise level of a standard Gaussian, our Diffusion models will start with an "image" at a noise level σ_0 and gradually denoise it into an image with no noise, belonging to the original data distribution.

$\sigma(t)$ can be an arbitrary function which defines the noise level at time t.

The forward noising process is given by the following probability flow ODE:
$$dx = -\dot{\sigma}(t)\sigma(t)\nabla_{x}\log p\big(x; \sigma(t)\big)dt$$

Different choices of time steps t and noise schedule $\sigma(t)$ result in different scheduling, like DDPM with a linear β schedule for example.

Denoiser

What we need to have is a model called a denoiser $D(x; \sigma)$ that minimizes the denoising error for samples drawn from the data generating distribution for every noise level.

From that we can obtain the score function.

To be as general as possible this denoiser is defined as follows:

$$D_\theta(x; \sigma) = c_{skip}(\sigma)x + c_{out}(\sigma)F_\theta(c_{in}(\sigma)x; c_{noise}(\sigma))$$

Where the neural network output is F_θ and the other functions define

$$\mathbb{E}_{\sigma, y, n}\left[\underbrace{\lambda(\sigma)\, c_{\text{out}}(\sigma)^2}_{\text{effective weight}} \left\| \underbrace{F_\theta\left(c_{\text{in}}(\sigma) \cdot (x_0 + n); c_{\text{noise}}(\sigma)\right)}_{\text{network output}} - \underbrace{\frac{1}{c_{\text{out}}(\sigma)}\left(x_0 - c_{\text{skip}}(\sigma) \cdot (x_0 + n)\right)}_{\text{effective training target}} \right\|_2^2 \right]$$

Sampling

The general sampling procedure (solver and formulation agnostic) can be summarized as:

- Obtain the time-steps $t_{i<T}$ given the total number of diffusion steps T
- Obtain the noise level $\sigma(t)$ for each of those steps, according to the schedule
- Obtain a starting sample of Gaussian noise from $\mathcal{N}(0, \sigma_{max}^2 I)$
- Run the preferred sampler on the schedule-transformed steps

In this case the choice of performing ancestral sampling, SDE or ODE solving is completely left to the sampler.

There even exist adaptive samplers which do not require a number of steps to be set.

Samplers

There are many possible samplers that we can use, some of them have a correlated paper, most of them are impenetrable horrors whose inner workings are only privy to those who made them.

- Euler
- Euler a
- LMS
- DPM2
- DPM2 a
- DPM++ 2S a

▶ DPM++ SM

▶ DPM++ SDE

▶ DPM fast

▶ DPM adaptive

▶ LMS Karras

▶ DPM2 Karras

▶ DPM2 a Karras

▶ DPM++ 2S a Karras

▶ DPM++ 2M Karras

▶ DPM++ SDE

▶ DDIM

▶ PLMS

▶ UniPC

Despite all of this it turns out that sampler choice does not matter much if we use more than 20 steps. The biggest difference considering results seems to be between ancestral (a), non-ancestral and SDE-based samplers.
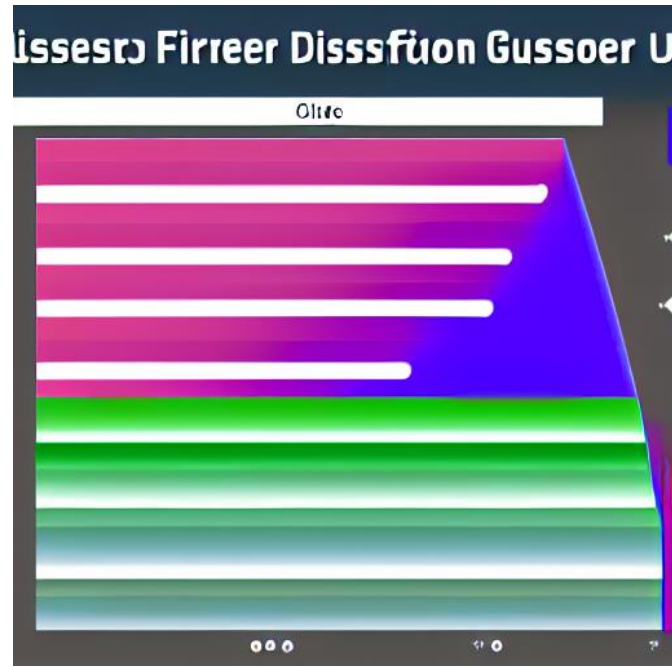
Guiding diffusion

Classifier guidance

Given a diffusion model $q(x_{t-1}|x_t)$ we use the term guidance to refer to the process of using some additional signal to inject conditioning information into the process.

The most basic way of doing this is called **Classifier guidance**

With this method a classifier $f_\phi(y|x_t)$ trained on noisy images is used to guide the diffusion process toward a certain class.

We can notice that: $\nabla_{x_t} \log q(x_t, y) = \nabla_{x_t} \log q(x_t) + \nabla_{x_t} \log q(y|x_t) = \nabla_{x_t} \log q(x_t) + \nabla_{x_t} \log f_\phi(y|x_t)$

And our weighted and guided noise prediction becomes:

$$\bar{\epsilon}_\theta(x_t, t, y) = \epsilon_\theta(x_t, t) - \sqrt{1 - \bar{\alpha}_t} w \nabla_{x_t} \log f_\phi(y|x_t)$$

However this method is very inconvenient, since it forces us to train an additional model from scratch.

Classifier-Free guidance

Instead of having a classifier in addition to our denoising network we could just use our inner network to parametrize two distributions, $p_\theta(x)$ and $p_\theta(x|y)$, obtained by predicting respectively $\epsilon_\theta(x_t, t)$ and $\epsilon_\theta(x_t, t, y)$; this approach is called classifier-free guidance.

It holds that: $\nabla_{x_t} \log q(y|x_t) = \nabla_{x_t} \log q(x_t|y) - \nabla_{x_t} \log q(x_t)$

And if we substitute this gradient in the formula for classifier guidance we obtain:
$$\bar\epsilon_\theta(x_t, t, y) = \epsilon_\theta(x_t, t, y) + w(\epsilon_\theta(x_t, t, y) - \epsilon_\theta(x_t, t))$$

Using this "doubled" denoiser is also easy, if we want to make our model behave in the unconditioned way(to get the second term) we can just replace y with a null value.

Of course during training we have to sometimes drop the conditioning information to let the model learn $p_\theta(x)$.
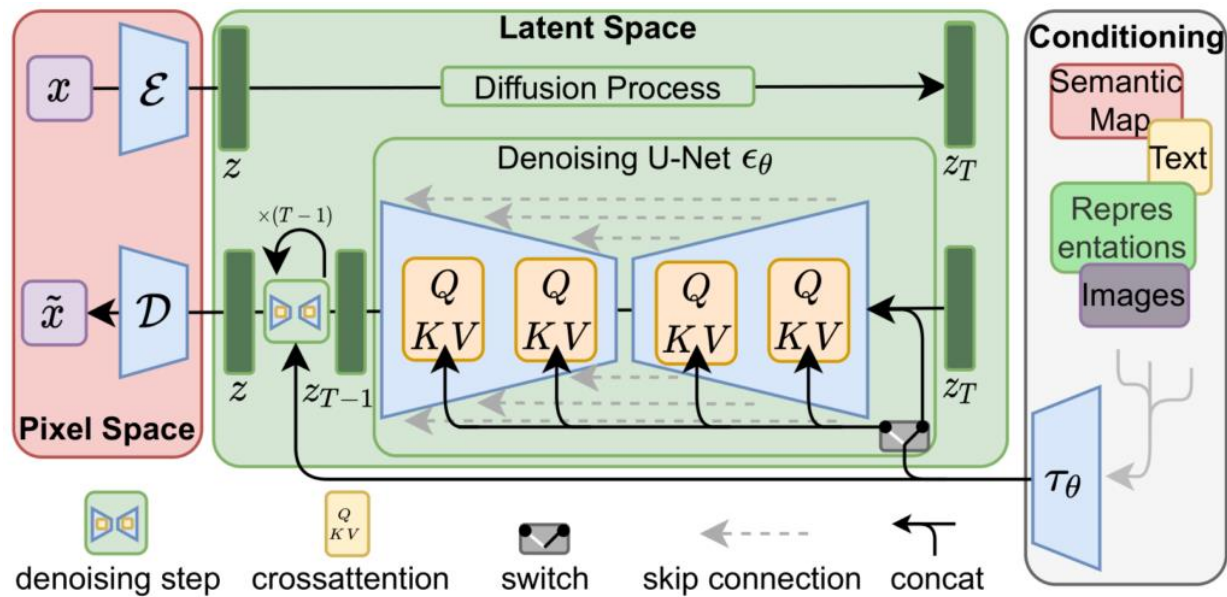
Image credit: Rombach et al. (2022)

Latent diffusion

Running diffusion in the latent space

Both inference and training of diffusion models is quite expensive, and the computational cost scales quadratically with the resolution of input images.

Can we do better?

The main idea of the **Latent Diffusion** approach is to run the diffusion process in the latent space of an autoencoder, this makes our denoiser work on lower-dimensional data than before.

We now have to train two (partially independent) models, with the denoiser only taking latents as inputs and only providing latents as outputs.

If we want to re-obtain real images we just need to pass the denoised latents to the decoder.

Another advantage of this approach is the newly gained ability of manipulating the latent space.

The Autoencoder

The autoencoder is actually a variational autoencoder, i.e. we learn the parameters of a distribution in the latent space, and it is composed of an encoder part $\mathcal{E}$ and a decoder part $\mathcal{D}$.

The training objective has three components:

- $L_{rec}(x, \mathcal{D}(\mathcal{E}(x)))$: the reconstruction loss, MSE between real and reconstructed image, often replaced with a learned perceptual loss instead

- $L_{reg}(x; \mathcal{E}, \mathcal{D})$: regularization, two possibilities, either the classical VAE regularization of the learned distribution towards a standard Gaussian, or/and a vector-quantization regularization

- $L_{adv}(x, \mathcal{D}(\mathcal{E}(x)), \mathcal{D}_\phi)$: adversarial objective; train a patch-based discriminator D_ψ in conjunction with the autoencoder, whose objective is to discriminate between images and their reconstruction

The denoiser

The denoiser consists of a U-Net architecture, which receives in input a noised image x_t and a timestep t and is trained to be a noise predictor.

An important consideration is how to encode the scalar-valued timestep into a proper higher-dimensional (d_t dimensional) value, to do so the authors employ the classical positional embedding found in the transformer architecture.
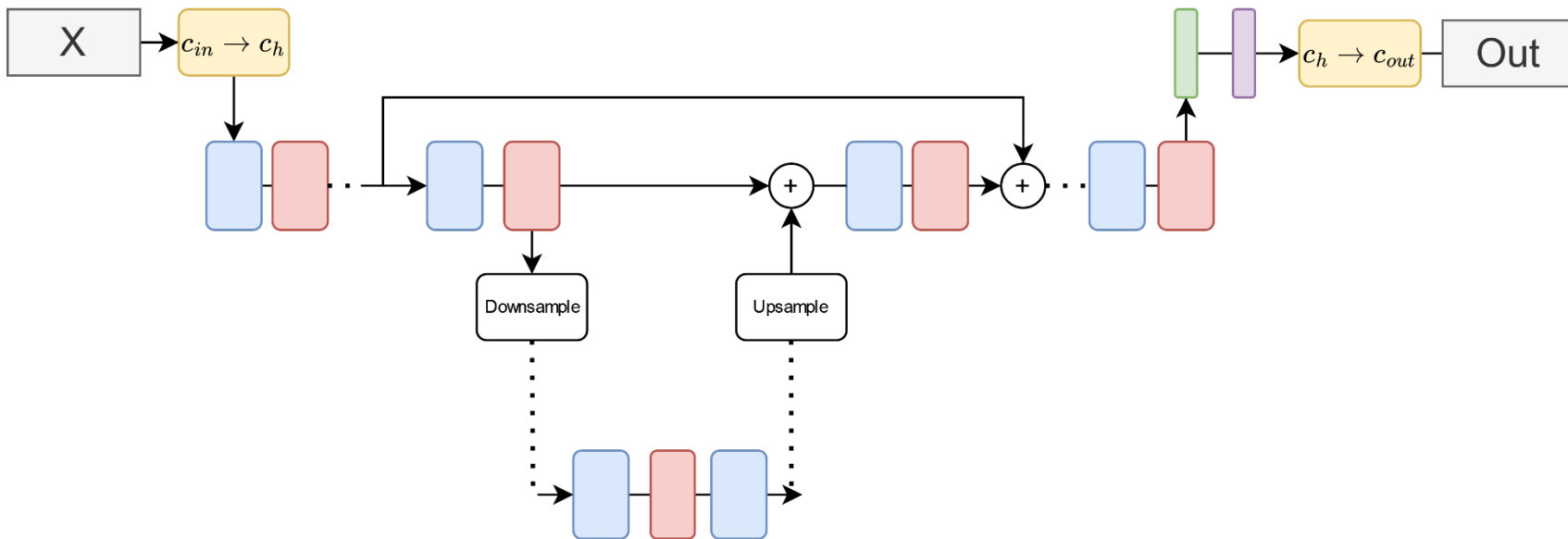
$$PE_{(t,2i)} = \sin\left(\frac{t}{10000^{2i/d_t}}\right), \qquad PE_{(t,2i+1)} = \cos\left(\frac{t}{10000^{2i/d_t}}\right)$$

Each level of our U-Net will consist of residual blocks interleaved with attention blocks.
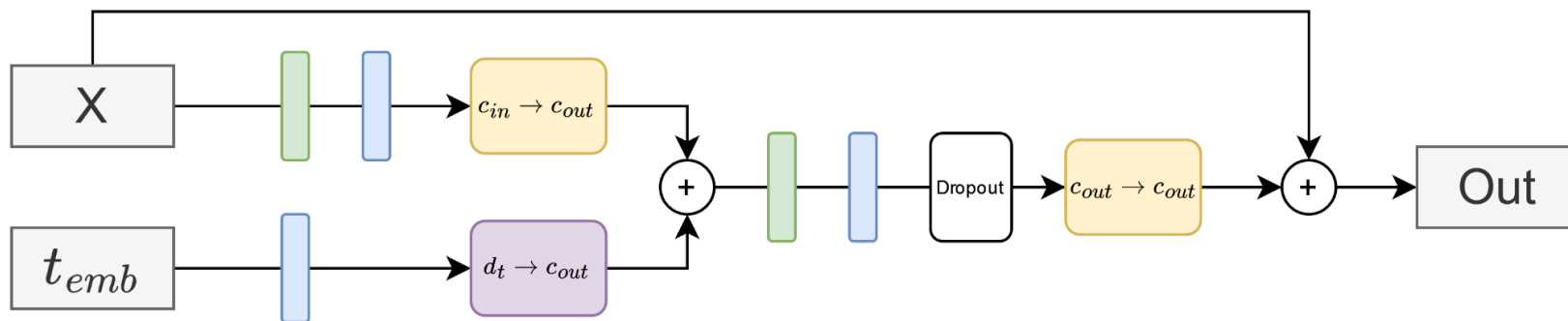
These attention blocks can either implement spatial self-attention or spatial cross-attention, the latter of which is the preferred way to add conditioning information into our diffusion process.
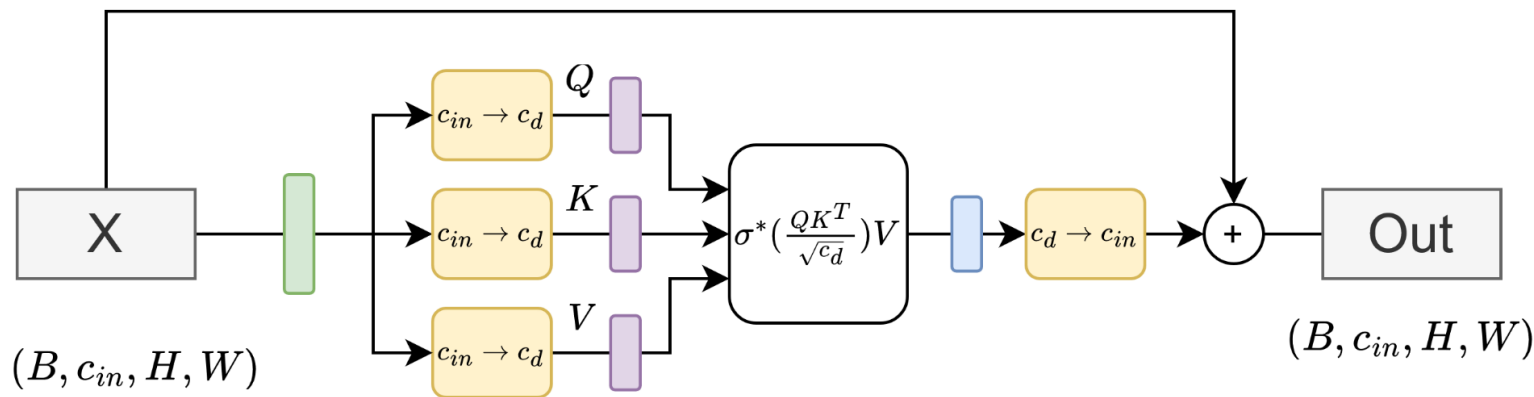
u-Net architecture

35

Residual block

Legend:

- Swish
- Groupnorm
- Linear
- 3×3 conv

Spatial Attention

$$\sigma^*\left(\frac{QK^T}{\sqrt{c_d}}\right)V$$

(B, c_{in}, H, W)

(B, c_{in}, H, W)

Legend:

- 1×1 conv
- Groupnorm
- Reshape to (B, C, HW)
- Reshape to (B, C, H, W)

σ^* is the softmax function

To have cross attention K, V are projections of the conditioning information

37

Conditioning and Tricks

General conditioning

Having a latent conditional diffusion model amounts to having a noise predictor model which computes $\epsilon_\theta(z_t, t, y)$.

To inject various types of conditioning information into our model what we do is we use a domain-specific encoder $\tau_\theta(y)$, which maps arbitrary conditioning information (class, text, images, segmentation maps...) into a d_τ-dimensional space. This code is then used to condition the denoiser model through cross attention.

The conditional objective thus becomes:

$$L_{LDM} = \mathbb{E}_{\mathcal{E}(x), y, \epsilon \sim \mathcal{N}(0,I), t}[\| \epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y)) \|_2^2]$$

As an example, the standard way of providing text conditioning is to use a frozen language model (usually CLIP due to its multimodal nature) as τ_θ.

Negative prompting

We have seen how to give positive conditioning information, that is nudging the model toward a y-conditioned generation, is there a way to also nudge it away from a certain $\bar{y}$-conditioned generation?

We can just reformulate the unconditional term in classifier-free guidance and replace it with a negative-conditional term, in this way CFG generation becomes:

$$\bar{\epsilon}_\theta(z_t, t, y, \bar{y}) = \epsilon_\theta(z_t, t, y) + w(\epsilon_\theta(z_t, t, y) - \epsilon_\theta(z_t, t, \bar{y}))$$

This is especially used in text-guided generation, since in this setting the space of generation possibilities is so big, latents of different objects might be close together, resulting in an unwanted coupled generation.

We can fix this by setting as negative conditioning the intrusive concept.

Img2IMG

Image conditioning can be done through the cross-attention mechanism as we have seen, however if we want to really mimic the image the conditioning signal might not be strong enough.

In this case we can use a different type of conditioning, namely conditioning by starting with a pre-existing latent.

Given an image that we want to transform we can proceed as follows:

- Encode the input image with the encoder to get the estimated moments

- Sample from a Gaussian parametrized in that way

- Run the sampled latent under the diffusion process, adding other conditioning information through cross-attention

The more steps of the diffusion process we run the more different our output latent will be to the original, we can control this duration through a parameter referred to as **denoising strength**.

Inpainting

Inpainting refers to the procedure of taking an input image, defining a mask and then modifying only the content of that mask

Whole image inpainting
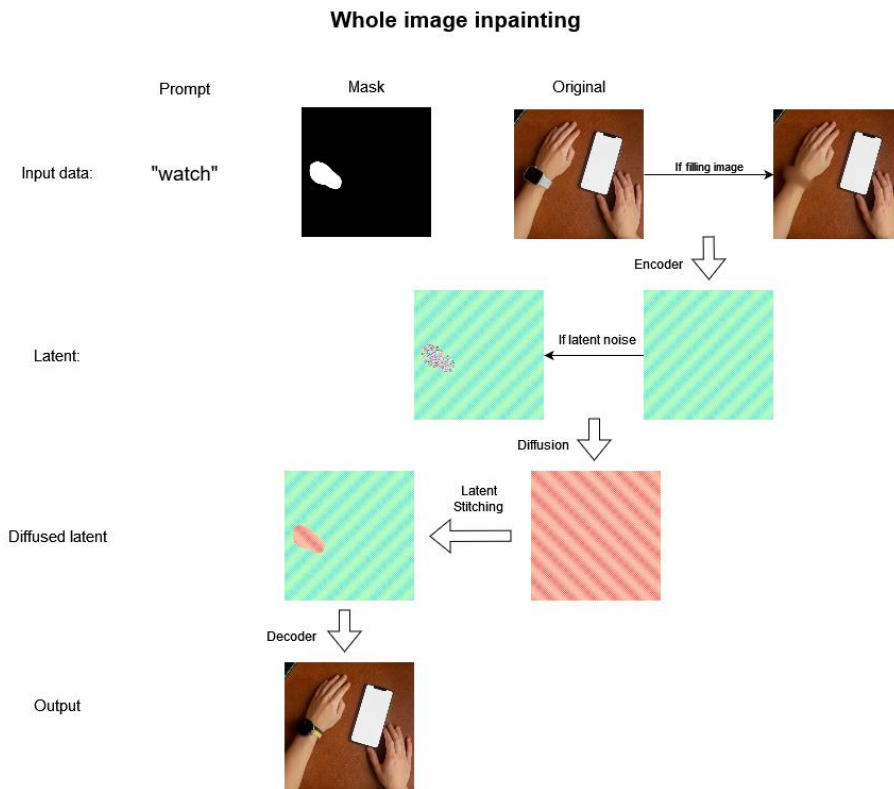
Resources

Main Papers

- What are diffusion models?
- Denoising Diffusion Probabilistic Models
- Improved Denoising Diffusion Probabilistic Models
- Generative Modeling by Estimating Gradients of the Data Distribution
- Score-Based Generative Modeling Through Stochastic Differential Equations
- Elucidating the Design Space of Diffusion-Based Generative Models
- High-Resolution Image Synthesis with Latent Diffusion Models
- Classifier-Free Diffusion Guidance
- Denoising Diffusion Implicit Models
- Pseudo Numerical Methods for Diffusion Models on Manifolds

For all the resources I've used as reference take a look at:
https://github.com/DiegoBiagini/UsefulDiffusion

Repositories and libraries

- https://github.com/hojonathanho/diffusion
- https://github.com/lucidrains/denoising-diffusion-pytorch
- https://github.com/mikonvergence/DiffusionFastForward
- https://github.com/crowsonkb/k-diffusion
- https://github.com/CompVis/latent-diffusion
- https://github.com/CompVis/stable-diffusion
- https://github.com/AUTOMATIC1111/stable-diffusion-webui
- https://huggingface.co/docs/diffusers/index

Proofs / Derivations

1

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I})$$

Using Bayes' rule, we have:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)}$$

$$\propto \exp\Big(-\frac{1}{2}\big(\frac{(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{\beta_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2}{1 - \bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1 - \bar{\alpha}_t}\big)\Big)$$

$$= \exp\Big(-\frac{1}{2}\big(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \alpha_t\mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \bar{\alpha}_{t-1}\mathbf{x}_0^2}{1 - \bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1 - \bar{\alpha}_t}\big)\Big)$$

$$= \exp\Big(-\frac{1}{2}\big((\frac{\alpha_t}{\beta_t} + \frac{1}{1 - \bar{\alpha}_{t-1}})\mathbf{x}_{t-1}^2 - (\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1 - \bar{\alpha}_{t-1}}\mathbf{x}_0)\mathbf{x}_{t-1} + C(\mathbf{x}_t, \mathbf{x}_0)\big)\Big)$$

where $C(\mathbf{x}_t, \mathbf{x}_0)$ is some function not involving $\mathbf{x}_{t-1}$ and details are omitted. Following the standard Gaussian density function, the mean and variance can be parameterized as follows (recall that $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$):

$$\tilde{\beta}_t = 1/(\frac{\alpha_t}{\beta_t} + \frac{1}{1 - \bar{\alpha}_{t-1}}) = 1/(\frac{\alpha_t - \bar{\alpha}_t + \beta_t}{\beta_t(1 - \bar{\alpha}_{t-1})}) = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \cdot \beta_t$$

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = (\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1 - \bar{\alpha}_{t-1}}\mathbf{x}_0)/(\frac{\alpha_t}{\beta_t} + \frac{1}{1 - \bar{\alpha}_{t-1}})$$

$$= (\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1 - \bar{\alpha}_{t-1}}\mathbf{x}_0)\frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \cdot \beta_t$$

$$= \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x}_0$$

Thanks to the nice property, we can represent $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t)$ and plug it into the above equation and obtain:

$$\tilde{\boldsymbol{\mu}}_t = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t)$$

$$= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$$

$$-\log p_\theta(\mathbf{x}_0) \leq -\log p_\theta(\mathbf{x}_0) + D_{\mathrm{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)\|p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0))$$

$$= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})/p_\theta(\mathbf{x}_0)}\right]$$

$$= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log p_\theta(\mathbf{x}_0)\right]$$

$$= \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]$$

$$\text{Let } L_{\mathrm{VLB}} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \geq -\mathbb{E}_{q(\mathbf{x}_0)}\log p_\theta(\mathbf{x}_0)$$

$$L_{\text{VLB}} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]$$

$$= \mathbb{E}_q\left[\log\frac{\prod_{t=1}^{T}q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_T)\prod_{t=1}^{T}p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right]$$

$$= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=1}^{T}\log\frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right]$$

$$= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T}\log\frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log\frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right]$$

$$= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T}\log\left(\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\cdot\frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}\right) + \log\frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right]$$

$$= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T}\log\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \sum_{t=2}^{T}\log\frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)} + \log\frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right]$$

$$= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T}\log\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log\frac{q(\mathbf{x}_T|\mathbf{x}_0)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log\frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right]$$

$$= \mathbb{E}_q\left[\log\frac{q(\mathbf{x}_T|\mathbf{x}_0)}{p_\theta(\mathbf{x}_T)} + \sum_{t=2}^{T}\log\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right]$$

$$= \mathbb{E}_q[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T}\underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}]$$

4

$$L_t = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{1}{2\|\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)\|_2^2} \|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2 \right]$$

$$= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{1}{2\|\boldsymbol{\Sigma}_\theta\|_2^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_t \right) - \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) \right\|^2 \right]$$

$$= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{(1 - \alpha_t)^2}{2\alpha_t(1 - \bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2} \|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2 \right]$$

$$= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{(1 - \alpha_t)^2}{2\alpha_t(1 - \bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2} \|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2 \right]$$