LATENT DIFFUSION MODEL

KAITIAN CHAO, BANGJIE WANG, AND GUANQIU WU

Abstract. Diffusion probabilistic models have demonstrated state-of-the-art performance in various image synthesis benchmarks. However, they suffer from a lack of low-dimensional, interpretable latent space and slow generation speed. Due to operating directly in pixel space, optimizing powerful diffusion models often requires hundreds of GPU days, and inference time is costly due to sequential evaluations. In contrast, Variational Autoencoders (VAEs) provide access to a low-dimensional latent space, but despite recent advances, they continue to exhibit subpar sample quality.

To address these limitations according to the paper (Robin Rombach et al., 2022)[4], Latent Diffusion Model(LDWM) was introduced as a combination of VAE and Diffusion Model enabling efficient training on limited computational resources without compromising quality and flexibility.

In our project, we explain the motivation and mathematical derivation of both VAE and the Diffusion Model and discover that their mathematical principles are deeply rooted in Markov chains, sampling, and other stochastic processes. Then we introduce the Latent Diffusion Model as a combination of both. Furthermore, we implemented the code for VAE, Diffusion Model, and Latent Diffusion model, and trained all models and got some results.

Contents

1. Introduction

1.1. Summary of Our Work.

- **Theoretical Part:** During completing the project, our group conducted an in-depth study of Variational Autoencoders (VAEs) and Diffusion Models, gaining a profound understanding of both models. We proved and clarified all the mathematical derivations, understanding all underlying stochastic mathematical principles. Remarkably, we also understand the motivations behind these model formulations and mathematical derivations. Furthermore, we successfully discover the seemingly indirect relationship between VAEs and Diffusion Models.
- **Practical Part:** We successfully implemented the code for the VAE model, the Diffusion model, and the Latent Diffusion model, which is the combination of the VAE and Diffusion models.

  Specifically:

  1. We implemented a VAE model capable of compressing handwritten digits into a low-dimensional latent space and reconstructing them generatively.
  (Stochastic_Process_Project_VAE.ipynb)

  2. We developed a Diffusion model that can generate images of cars.
  (Stochastic_Process_Project_DM.ipynb)

  3. We combined the two models to create a Latent Diffusion model, capable of generating new images of handwritten digits.
  (Stochastic_Process_Project_LDM.ipynb)

1.2. Introduction to VAE.

In machine learning, a variational autoencoder (VAE) is an artificial neural network architecture.

In addition to being seen as an autoencoder neural network architecture, variational autoencoders can also be studied within the mathematical formulation of variational Bayesian methods, connecting a neural encoder network to its decoder through a probabilistic latent space (for example, as a multivariate Gaussian distribution) that corresponds to the parameters of a variational distribution.

Thus, the encoder maps each point (such as an image) from a large complex dataset into a distribution within the latent space, rather than to a single point in that space. The decoder has the opposite function, which is to map from the latent space to the input space, again according to a distribution. By mapping a point to a distribution instead of a single point, the network can avoid overfitting the

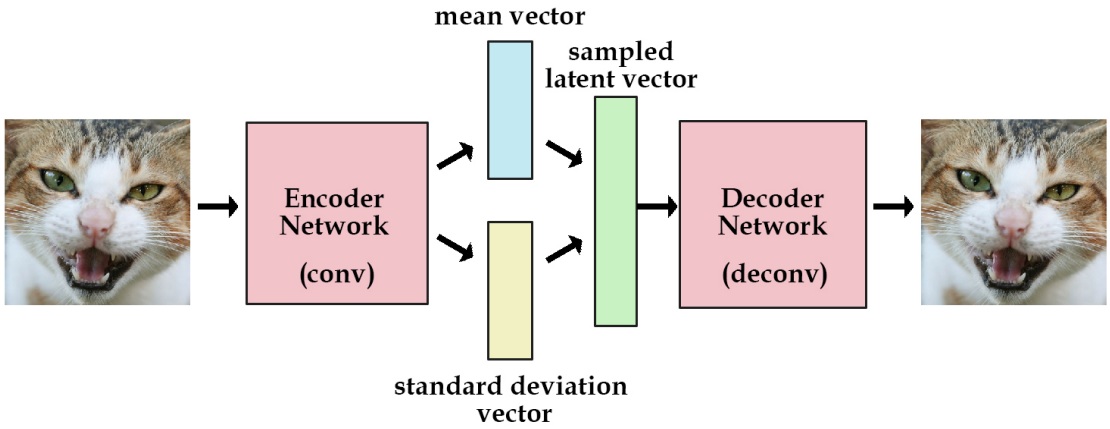


FIGURE 1. An illustration that shows how VAE works

training data. Both networks are typically trained together with the usage of the reparameterization trick, although the variance of the noise model can be learned separately.

1.3. **Introduction to Diffusion Model.** Diffusion Models are a class of generative models that create new data samples by iteratively denoising them. They first add noise to the data in a series of steps and then learn the reverse process to remove the noise, ultimately generating realistic data from pure noise.

In a word, the Diffusion Model consists of two processes:

Forward Process:

(1) Start with a clean image x_0.
(2) Add noise step-by-step according to the noise schedule β_t.
(3) After T steps, obtain a highly noisy image x_T.

Reverse Process:

(1) Start with a random noise sample x_T.
(2) Apply the learned reverse denoising steps iteratively.
(3) After T steps, obtain a generated image x_0 that resembles the training data.

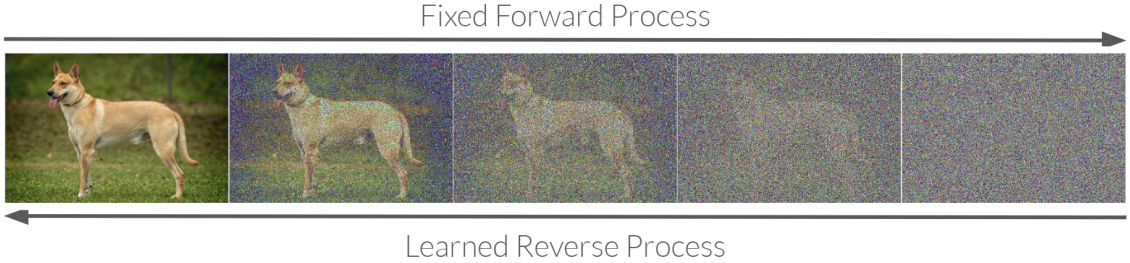


FIGURE 2. Forward and reverse diffusion process

This is a good figure illustrating how the forward and reverse diffusion process looks like, but it may cause some confusion. Note that, in the Diffusion Model, there is 0 possibility of recovering the original figure (and as it is a generative model, it is also not what we want), so running the learned reversed process, we will not get the original dog picture back, but we expect to obtain a new dog picture resembling the original dog picture.

So an intuitive and simplified explanation of what the Diffusion Model is doing is that: We first learned a model which could step by step denoise from a noisy image to our training data, then we apply this model to the Gaussian noise, and denoise it step by step, then we could have a generated image that resemble our training data.

2. VAE

2.1. **Motivation of VAE.** There are 2 ways to understand VAE, short for Variational Autoencoder.

(1) From the perspective of an autoencoder
(2) From the perspective of a generative model

2.1.1. *From the perspective of an autoencoder.* The fisrt way is to view VAE as a variant of regular autoencoders. Autocoder is invented to reconstruct high-dimensional data using a neural network model with a narrow bottleneck layer in the middle. A nice byproduct is dimension reduction: the bottleneck layer captures a compressed latent encoding. Such a low-dimensional representation can be used as an embedding vector in various applications (i.e. search), help data compression, or reveal the underlying data generative factors.

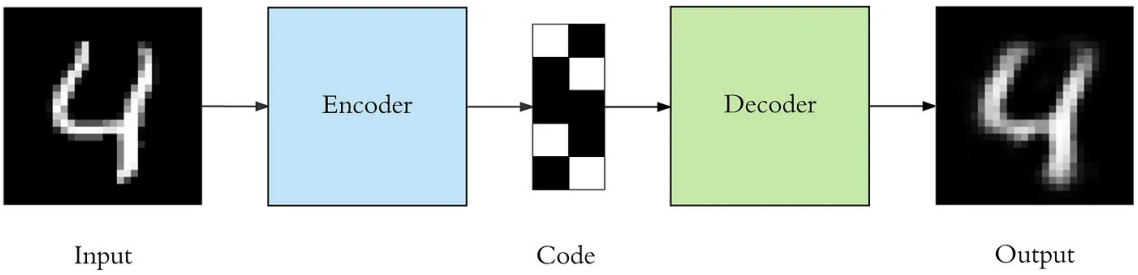


FIGURE 3. An illustration of a handwritten digit 4 compressed and reconstructed by an autoencoder (Image source: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798)

Autoencoder is a neural network designed to learn an identity function in an unsupervised way to reconstruct the original input while compressing the data in the process so as to discover a more efficient and compressed representation.It consists of two networks:

- Encoder network: It translates the original high-dimension input into the latent low-dimensional code. The input size is larger than the output size.
- Decoder network: The decoder network recovers the data from the code, likely with larger and larger output layers.

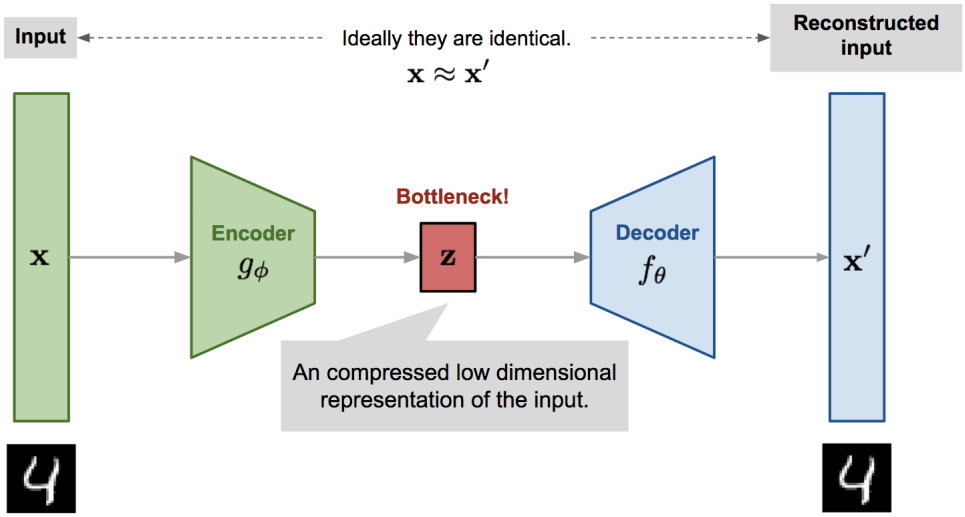


FIGURE 4. Illustration of autoencoder model architecture.[7]

The encoder network essentially accomplishes the dimensionality reduction. In addition, the autoencoder is explicitly optimized for the data reconstruction from the code. A good intermediate representation not only can capture latent variables, but also benefits a full decompression process.

The model contains an encoder function $g(.)$ parameterized by ϕ and a decoder function $f(.)$ parameterized by θ. The low-dimensional code learned for input $\mathbf{x}$ in the bottleneck layer is $\mathbf{z} = g_\phi(\mathbf{x})$ and the reconstructed input is $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$.

The parameters (ϕ, θ) are learned together to output a reconstructed data sample same as the original input, $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$, or in other words, to learn an identity function. There are various metrics to quantify the difference between two vectors, such as cross entropy when the activation function is sigmoid, or as simple as MSE loss:

$$L_{\text{AE}}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$$

The idea of Variational Autoencoder[3] is to add randomness into the encoder and the decoder, which heavily relies on the methods of variational bayesian and graphical model. Instead of mapping the input into a fixed vector, we want to map it into a distribution. If we label this distribution as p_θ, parameterized by θ. The relationship between the data input $\mathbf{x}$ and the latent encoding vector $\mathbf{z}$ can be fully defined by:

- Prior: $p_\theta(\mathbf{z})$
- Likelihood: $p_\theta(\mathbf{x}|\mathbf{z})$

- Posterior: $p_\theta(\mathbf{z}|\mathbf{x})$

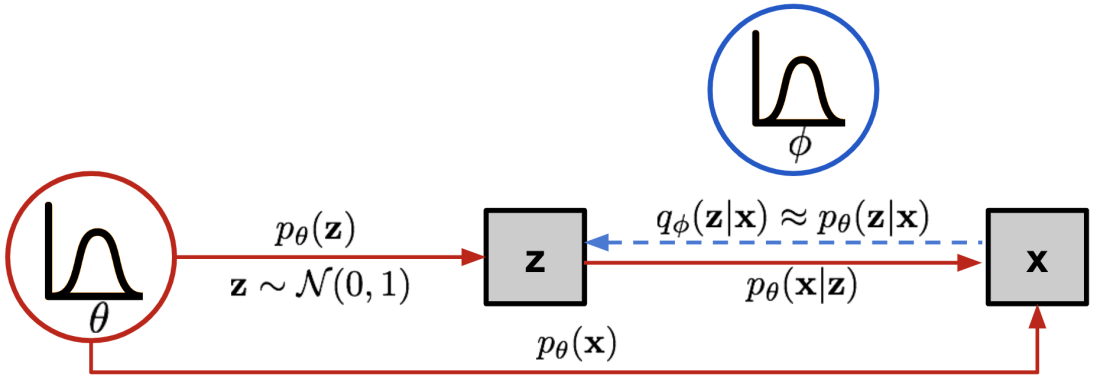


FIGURE 5. The graphical model involved in Variational Autoencoder. Solid lines denote the generative distribution $p_\theta(.)$ and dashed lines denote the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate the intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$.[7]

Assuming that we know the real parameter θ^* for this distribution. In order to generate a sample that looks like a real data point $\mathbf{x}^{(i)}$, we follow these steps:

First, sample a $\mathbf{z}^{(i)}$ from a prior distribution $p_{\theta^*}(\mathbf{z})$.Then a value $\mathbf{x}^{(i)}$ is generated from a conditional distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z} = \mathbf{z}^{(i)})$. The optimal parameter θ^* is the one that maximizes the probability of generating real data samples:

$$\theta^* = \arg\max_\theta \prod_{i=1}^{n} p_\theta(\mathbf{x}^{(i)})$$

Commonly we use the log probabilities to convert the product on RHS to a sum:

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)})$$

Now let's update the equation to better demonstrate the data generation process so as to involve the encoding vector:

$$p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}$$

Unfortunately it is not easy to compute $p_\theta(\mathbf{x}^{(i)})$ in this way, as it is very expensive to check all the possible values of $\mathbf{z}$ and sum them up. To narrow down the value space to facilitate faster search, we would like to introduce a new approximation function to output what is a likely code given an input $\mathbf{x}$, $q_\phi(\mathbf{z}|\mathbf{x})$, parameterized by ϕ.

Now the structure looks a lot like an autoencoder, but instead of a deterministic vector in the latent space, VAE matches the input $\mathbf{x}$ to a Gaussian distribution with parameters μ, σ, and gets a sample $\mathbf{z} \sim \mathcal{N}(\mu, \sigma)$:

- The conditional probability $p_\theta(\mathbf{x}|\mathbf{z})$ defines a generative model, similar to the decoder $f_\theta(\mathbf{x}|\mathbf{z})$ introduced above. $p_\theta(\mathbf{x}|\mathbf{z})$ is also known as probabilistic decoder.
- The approximation function $q_\phi(\mathbf{z}|\mathbf{x})$ is the probabilistic encoder, playing a similar role as $g_\phi(\mathbf{z}|\mathbf{x})$ above.

2.1.2. *From the perspective of a generative model.* Another way to understand the mechanics of VAE is to view it as a generative model that can generate new images of a particular object, style, etc., based on the materials by which it is trained. For example we want to generate new images of a car, having a dataset $\mathcal{D}$ of images of cars already. Firstly we have a low-dimensional image $\mathbf{z}$ of pure Gaussian noise, that is $\mathbf{z} \sim \mathcal{N}(0, I)$, where I is the identity matrix of corresponding size as ‡. We want to have a neural network $p_\theta(\mathbf{x}|\mathbf{z})$, where θ is its parameter, that having $\mathbf{z}$ as input, can generate an image $\mathbf{x}_0$ that is

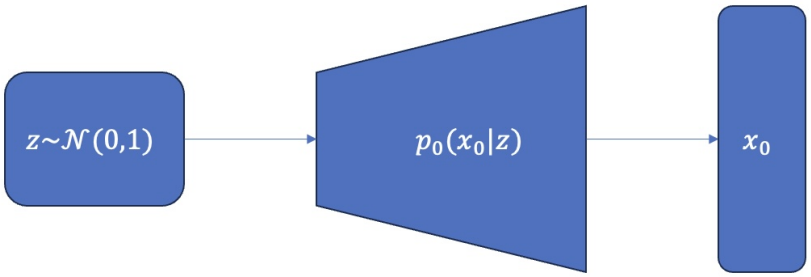


FIGURE 6. The goal to generate an image of request from Gausian noise by the model$p_\theta(\mathbf{x}|\mathbf{z})$

We want to maximize the likelihood of successesifully generating the desired $\mathbf{x}_0$, that is

$$p_\theta(x_0) := \int p_\theta(z)dz$$

but we can not check all possible z to compute $p_\theta(x_0)$ above. A natural idea is that we can only consider the $\mathbf{z}$s that are highly likely to be sampled from the posterior distribution of the generate model, $\mathbf{z} \sim p_\theta(\mathbf{z}|\mathbf{x}_0)$. But this posterior distribution is also unknown.

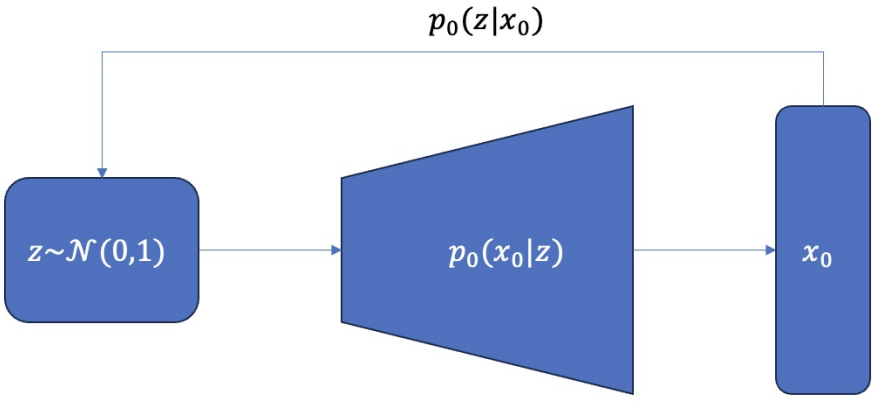


FIGURE 7. Consider only the **z**s that are highly likely to be sampled from the posterior distribution of the generate model

Therefore we turn to build another neural network $q_\phi(\mathbf{z}|\mathbf{x_0})$, where as before ϕ is the parameter of the neural network, to approximate this posterior distribution. This neural network has data sample $\mathbf{x} \in \mathcal{D}$ from the data set $\mathcal{D}$ as its input, and gives out a Gaussian distribution of lower dimension with mean μ_ϕ and variance σ_ϕ. Then through this neural network, we get the degenerated version $\mathbf{z}$ of the sample $\mathbf{x}$ from $\mathcal{D}$.

The the 2 neural networks $p_\theta(\mathbf{x}|\mathbf{z})$ and $q_\phi(\mathbf{z}|\mathbf{x_0})$ are trained together, and now in order to generate $\mathbf{x}_0$, we need to:

- minimize the KL divergence:

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x_0}) \|_\theta (\mathbf{z}|\mathbf{x}_0))$$

- maximize the log-likelihood of generating the desired result$\mathbf{x}_0$:

$$\log p_\theta(\mathbf{x}_0)$$

So instead of maximizing the likelihood $\log p_\theta(\mathbf{x}_0)$ directly, we are essentially adding constraints and maximizing the lower bound of it. That is

$$\log p_\theta(\mathbf{x}_0) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x_0}) \|_\theta (\mathbf{z}|\mathbf{x}_0))$$

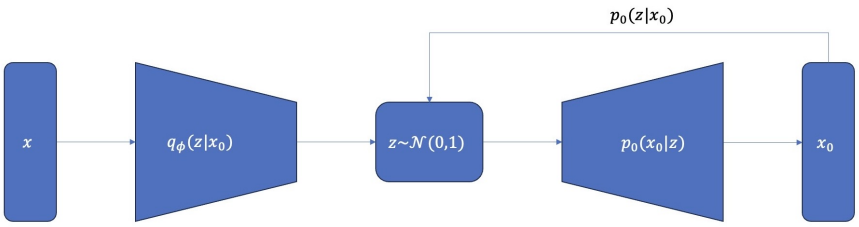


FIGURE 8. The 2 neural networks

Therefore, the loss function can be defined as

$$L_{VAE} = -\log p_\theta(\mathbf{x}_0) + D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x_0}) \parallel p_\theta(\mathbf{z}|\mathbf{x}_0)\right)$$

Using methods like gradient descent we get the optimized parameters θ^*, ϕ^*.

2.2. **Mathematical Derivation of VAE.**

2.2.1. *KL Divergence and the Loss Function.* Firstly we need to formally define the KL-divergence, the standard measure of how different a distribution p is from distribution q:

The KL-divergence of two distribution p and q is

$$D_{KL}(p \parallel q) = -\mathbb{E}_{X\sim}\left[\log\left(\frac{q(X)}{p(X)}\right)\right]$$

An interpretation of KL-divergence from information theory: If you're trying to encode values from true distribution p but use an encoder optimized for q, how many extra bits will you need per value?

It is worth mentioning the choice of using $D_{\mathrm{KL}}(q_\phi|p_\theta)$ (reversed KL) instead of $D_{\mathrm{KL}}(p_\theta|q_\phi)$ (forward KL)? Eric Jang has a great explanation in his post on Bayesian Variational methods. As a quick recap:

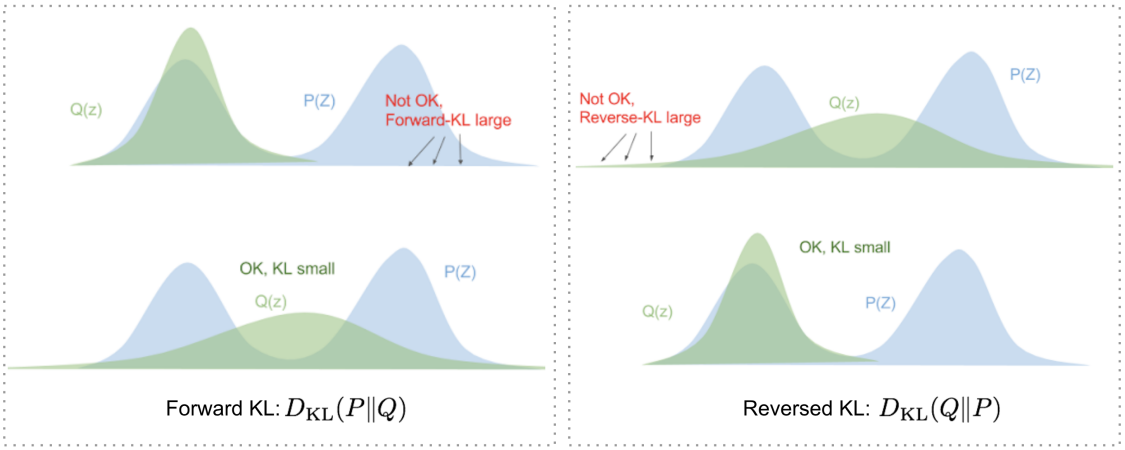


FIGURE 9. Forward and reversed KL divergence have different demands on how to match two distributions. (Image source: blog.evjang.com/2016/08/variational-bayes.html)

- Forward KL divergence: $D_{\mathrm{KL}}(P|Q) = \mathbb{E}_{z\sim P(z)} \log \frac{P(z)}{Q(z)}$; we have to ensure that $Q(z) > 0$ wherever $P(z) > 0$. The optimized variational distribution $Q(z)$ has to cover over the entire $P(z)$.
- Reversed KL divergence: $D_{\mathrm{KL}}(Q|P) = \mathbb{E}_{z\sim Q(z)} \log \frac{Q(z)}{P(z)}$; minimizing the reversed KL divergence squeezes the $Q(z)$ under $P(z)$.

Let's now expand the equation:

$$
\begin{aligned}
& D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} && \text{Because } p(z|x)=p(z,x)/p(x) \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \big(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} \big) d\mathbf{z} \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} && \text{Because } \textstyle\int q(z|x)dz=1 \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} && \text{Because } p(z,x)=p(x|z)p(z) \\
&= \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} [\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})] \\
&= \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})
\end{aligned}
$$

So we have:

$$
D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})
$$

Once rearrange the left and right hand side of the equation,

$$
\log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))
$$

The LHS of the equation is exactly what we want to maximize when learning the true distributions: we want to maximize the (log-)likelihood of generating real data (that is $\log p_\theta(\mathbf{x})$) and also minimize the difference between the real and estimated posterior distributions (the term D_{KL} works like a regularizer). Note that $p_\theta(\mathbf{x})$ is fixed with respect to q_ϕ.

The negation of the above defines our loss function:

$$
\begin{aligned}
L_{\mathrm{VAE}}(\theta,\phi) &= -\log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \\
&= -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \\
\theta^*, \phi^* &= \arg\min_{\theta,\phi} L_{\mathrm{VAE}}
\end{aligned}
$$

In Variational Bayesian methods, this loss function is known as the variational lower bound, or evidence lower bound. The "lower bound" part in the name comes from the fact that KL divergence is always non-negative and thus $-L_{\mathrm{VAE}}$ is the lower bound of $\log p_\theta(\mathbf{x})$.

Theorem 2.1. *KL-divergence is always nonnegative, and equals 0 only when p and q are the same distribution.*

The proof of this theorem can be found in, for example Wikipedia.

Therefore by minimizing the loss, we are maximizing the lower bound of the probability of generating real data samples.

2.2.2. *Reparameterization Trick.* The expectation term in the loss function invokes generating samples from $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Sampling is a stochastic process and therefore we cannot backpropagate the gradient. To make it trainable, the reparameterization trick is introduced: It is often possible to express the random variable $\mathbf{z}$ as a deterministic variable $\mathbf{z} = \mathcal{T}_\phi(\mathbf{x}, \boldsymbol{\epsilon})$, where $\boldsymbol{\epsilon}$ is an auxiliary independent random variable, and the transformation function $\mathcal{T}_\phi$ parameterized by ϕ converts $\boldsymbol{\epsilon}$ to $\mathbf{z}$.

For example, a common choice of the form of $q_\phi(\mathbf{z}|\mathbf{x})$ is a multivariate Gaussian with a diagonal covariance structure:

$$\begin{aligned}
&\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \\
&\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}) \quad \text{; Reparameterization trick.}
\end{aligned}$$

where $\odot$ refers to element-wise product.

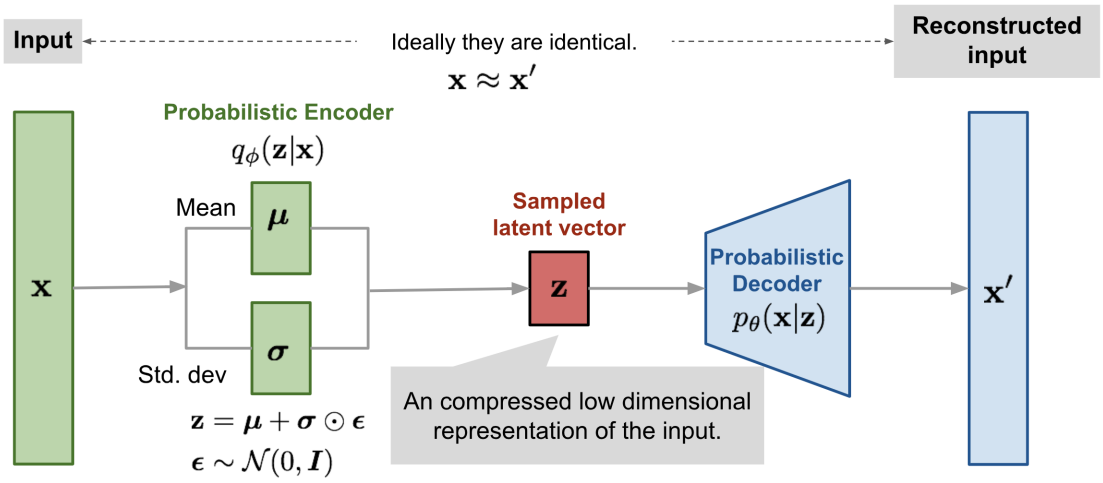


FIGURE 10. Illustration of variational autoencoder model with the multivariate Gaussian assumption.[7]

The reparameterization trick works for other types of distributions too, not only Gaussian. In the multivariate Gaussian case, we make the model trainable by learning the mean and variance of the distribution, and , explicitly using the reparameterization trick, while the stochasticity remains in the random variable $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$.

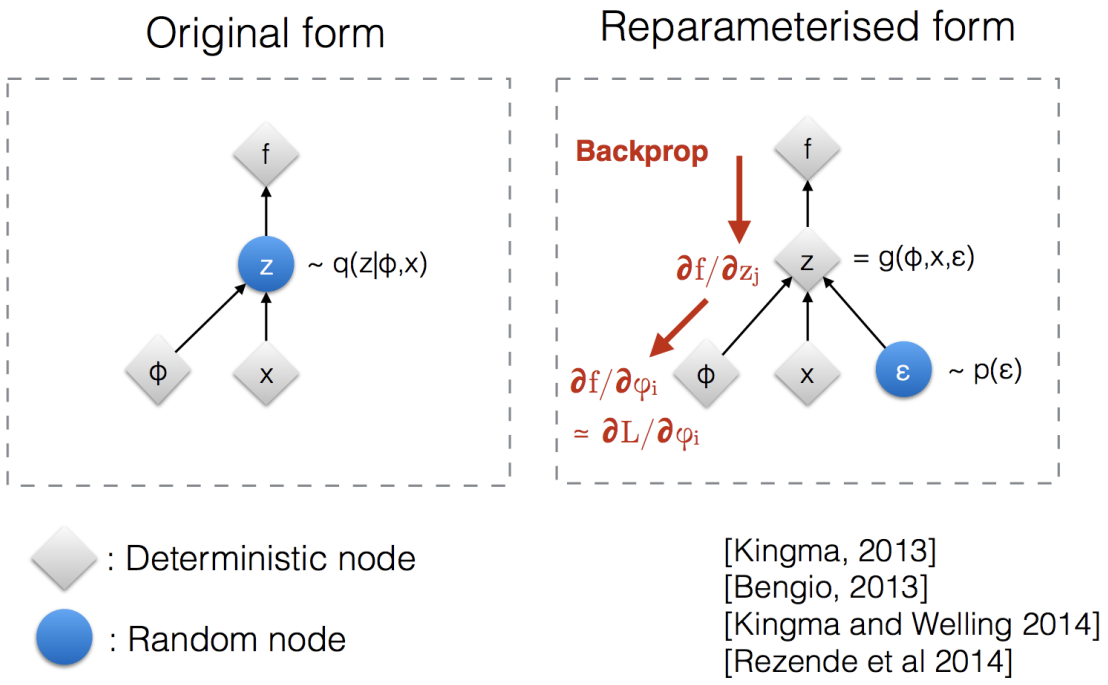


FIGURE 11. Reparameterization trick.(Image source: Slide 12 in Kingma's NIPS 2015 workshop talk)

3. Diffusion Model

3.1. **Motivation of Diffusion Model.** In this section, I want to explain the following:

(1) Why Diffusion Model is designed like this?
(2) Why the Diffusion Models are philosophically similar to VAE?

3.1.1. *How Diffusion Model work.* For a generative model, our naive wish is:

$$\begin{gathered}\text{Under the following model:}\\ \underset{\mathcal{N}(0,1)}{\mathbf{x}_T} \overset{\text{model}_\theta}{\longrightarrow} \underset{p_\theta(\mathbf{x}_0|\mathbf{x}_T)}{\mathbf{x}_0}\\ \text{we want to maximize } \log p_\theta(\mathbf{x}_0)\\ \text{(which is the possibility of generating pictures we want)}\end{gathered}$$

here X_T is a random Gaussian distribution, so our naive wish is to make the model have the largest ability to use such random input to generate the figures we want. (Note that the ability to generate pictures using totally random input is vital for the generative model)

However, generate high-quality samples in one step is really hard, so we decompose it into several steps, as following:

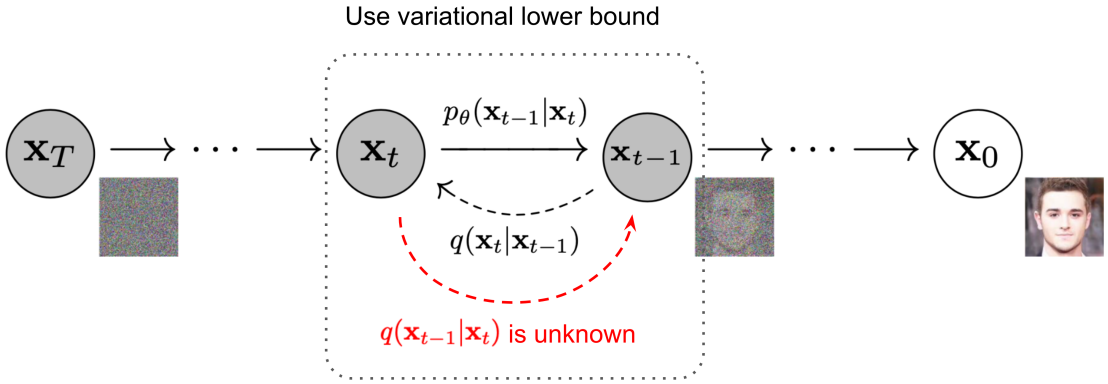


FIGURE 12. Step by step generation

here $p_\theta(\mathbf{x}_0|\mathbf{x}_T)$ is decomposed into several $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})$ and we call such process 'generate process' (or what people called the 'reverse process')

However, for $\mathbf{x}_t$ in each step, how could we know which $\mathbf{x}_{t-1}$ is the 'best' $\mathbf{x}_{t-1}$ to generated in next step?

So we need to have samples to teach the model how to generate the 'best' $\mathbf{x}_{t-1}$ in the next step.

Thus we need to establish the 'degenerate process' (or what people called the 'forward process'), which adds Gaussian noise to a clean image step by step. Here $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ represents the degenerate process in the figure above.

So our target is:

(1) We want the model to generate $\mathbf{x}_t$ sample at time stamp t, which are highly likely to be sampled during the diffusion process at t.
(2) We want to train the model so that we can approximate the posterior $p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)$ using $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ as well as possible.

So on the one hand, we want to maximize $\log p_\theta(\mathbf{x}_0)$, on the other hand, we want to minimize $D_{\mathrm{KL}}(q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \| p_\theta(\mathbf{x}_{1:T} \mid \mathbf{x}_0))$.

So finally, it explain why we pick

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

as our loss function in training(We will show this later in mathematical derivation part).

3.1.2. *Why the Diffusion Models are philosophically similar to VAE.* Recall that our naive wish is to generate samples using the following model:

$$\underset{\mathcal{N}(0,1)}{\mathbf{x}_T} \xrightarrow[p_\theta(\mathbf{x}_0|\mathbf{x}_T)]{\text{model}_\theta} \mathbf{x}_0$$

However, it is hard for us to generate high-quality samples in one step. So we first decompose this into several steps. And to generate the learning data we need

in these middle steps, we construct a degenerate process to do the following:

$$\underset{\text{dataset}}{\mathbf{x}_0} \overset{\text{degenerate}}{\underset{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{\longrightarrow}} \mathbf{x}_T$$

and our goal is to using the data generated by the degenerate process to teach the model to generate samples that resemble our original dataset $\mathbf{x}_0$.

So the complete digram looks like this:

$$\underset{\text{dataset}}{\mathbf{x}_0} \overset{\text{degenerate}}{\underset{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{\longrightarrow}} \mathbf{x}_T \overset{\text{generate}}{\underset{p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})}{\longrightarrow}} \tilde{\mathbf{x}}_0$$

where we want somehow $\mathbf{x}_0 \approx \tilde{\mathbf{x}}_0$.

This is philosophically identical to VAE!

In VAE, what we want is making $p_\theta(\mathbf{x}|\mathbf{z})$ a good inverse to $q_\phi(\mathbf{x}|\mathbf{z})$.

In the Diffusion Model, what we want is also making $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ a good inverse to $q_\phi\left(\mathbf{x}_t|\mathbf{x}_{t+1}\right)$.

So these two models are isomorphic to each other indeed.

The difference is Diffusion Model decomposes the single step in VAE into several steps, which gives the model a better competence of generating high-quality new samples. As in the following figure:

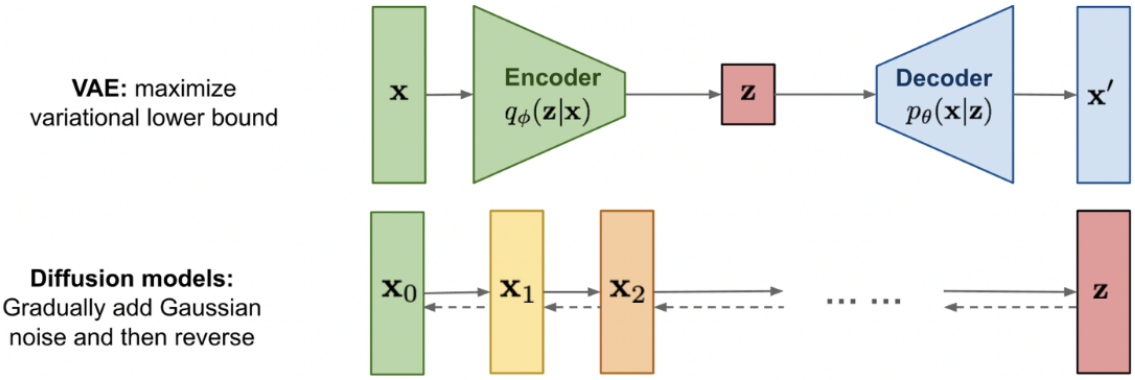


FIGURE 13. Diffusion and VAE

3.2. **Mathematical Derivation of Diffusion Model.**

3.2.1. *Forward Diffusion Process.* Given a data point sampled from a real data distribution $\mathbf{x}_0 \sim q(\mathbf{x})$, let us define a forward diffusion process in which we add small amount of Gaussian noise to the sample in T steps, producing a sequence of noisy samples $\mathbf{x}_1, \ldots, \mathbf{x}_T$. The step sizes are controlled by a variance schedule:

$$\{\beta_t \in (0,1)\}_{t=1}^T \text{ One choice is } \beta_1 = 10^{-4} \text{ to } \beta_T = 0.02, T = 1000.$$

$$q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \quad q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) = \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)$$

The data sample $\mathbf{x}_0$ gradually loses its distinguishable features as the step t becomes larger. Eventually when $T \to \infty, \mathbf{x}_T$ is equivalent to an isotropic Gaussian distribution.

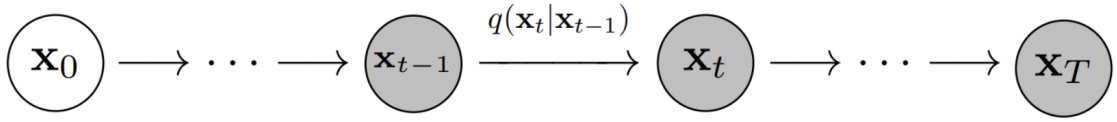


FIGURE 14. Forward diffusion process

FIGURE 15. A visual representation of the forward diffusion process

A nice property of the above process is that we can sample $\mathbf{x}_t$ at any arbitrary time step t in a closed form using reparameterization trick. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\overline{\boldsymbol{\epsilon}}_{t-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \\
q\left(\mathbf{x}_t \mid \mathbf{x}_0\right) &= \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \left(1-\bar{\alpha}_t\right)\mathbf{I}\right)
\end{aligned}$$

where $\boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Note that we denote $\overline{\boldsymbol{\epsilon}}_{t-2}$ as the merge of two Gaussians $(*)$.

Recall that when we merge two Gaussians with different variance, $\mathcal{N}\left(\mathbf{0}, \sigma_1^2\mathbf{I}\right)$ and $\mathcal{N}\left(\mathbf{0}, \sigma_2^2\mathbf{I}\right)$, the new distribution is $\mathcal{N}\left(\mathbf{0}, \left(\sigma_1^2 + \sigma_2^2\right)\mathbf{I}\right)$. Here the merged standard deviation is

$$\sqrt{\left(1-\alpha_t\right) + \alpha_t\left(1-\alpha_{t-1}\right)} = \sqrt{1-\alpha_t\alpha_{t-1}}.$$

Usually, we can afford a larger update step when the sample gets noisier, so $\beta_1 < \beta_2 < \cdots < \beta_T$ and therefore $\bar{\alpha}_1 > \cdots > \bar{\alpha}_T$.

In practice, we use $\beta_1 = 0.001$ to $\beta_T = 0.02$ with linear growth between β_1 to β_T, or we use cosine function to create nonlinear growth between β_1 to β_T.

The following graph shows how $\bar{\alpha}_t$ changes along with timestep.

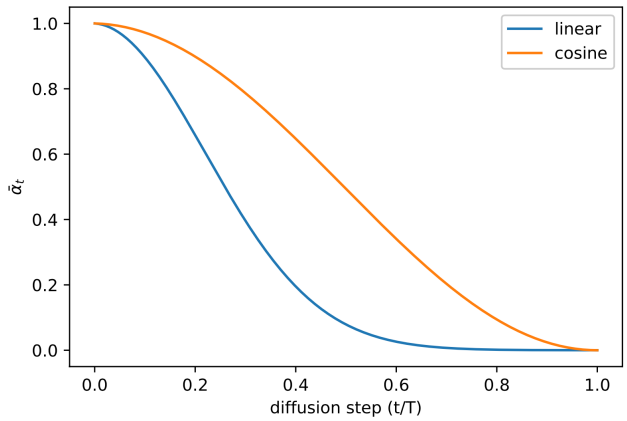


FIGURE 16. Linear and cosine-based scheduling

3.2.2. *Reverse diffusion process.* If we can reverse the above process and sample from $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$, we will be able to recreate the true sample from a Gaussian noise input, $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

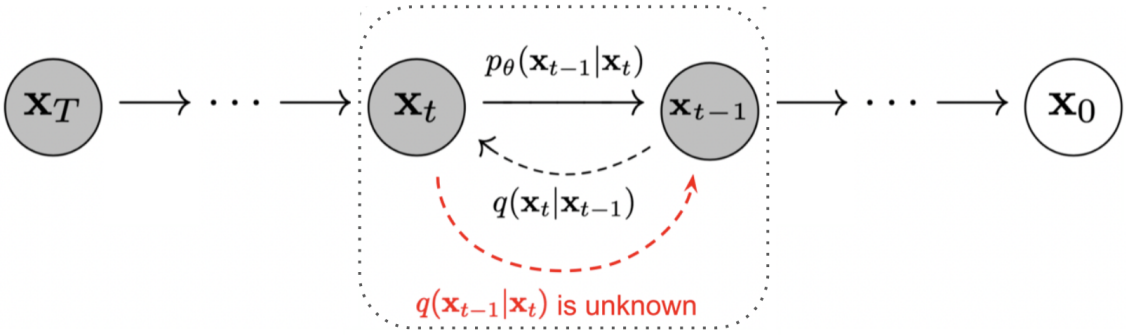


FIGURE 17. Reverse diffusion process

Note that if β_t is small enough, $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ will also be Gaussian. Unfortunately, we cannot easily estimate $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ because it needs to use the entire dataset and therefore we need to learn a model p_θ to approximate these conditional probabilities in order to run the reverse diffusion process.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

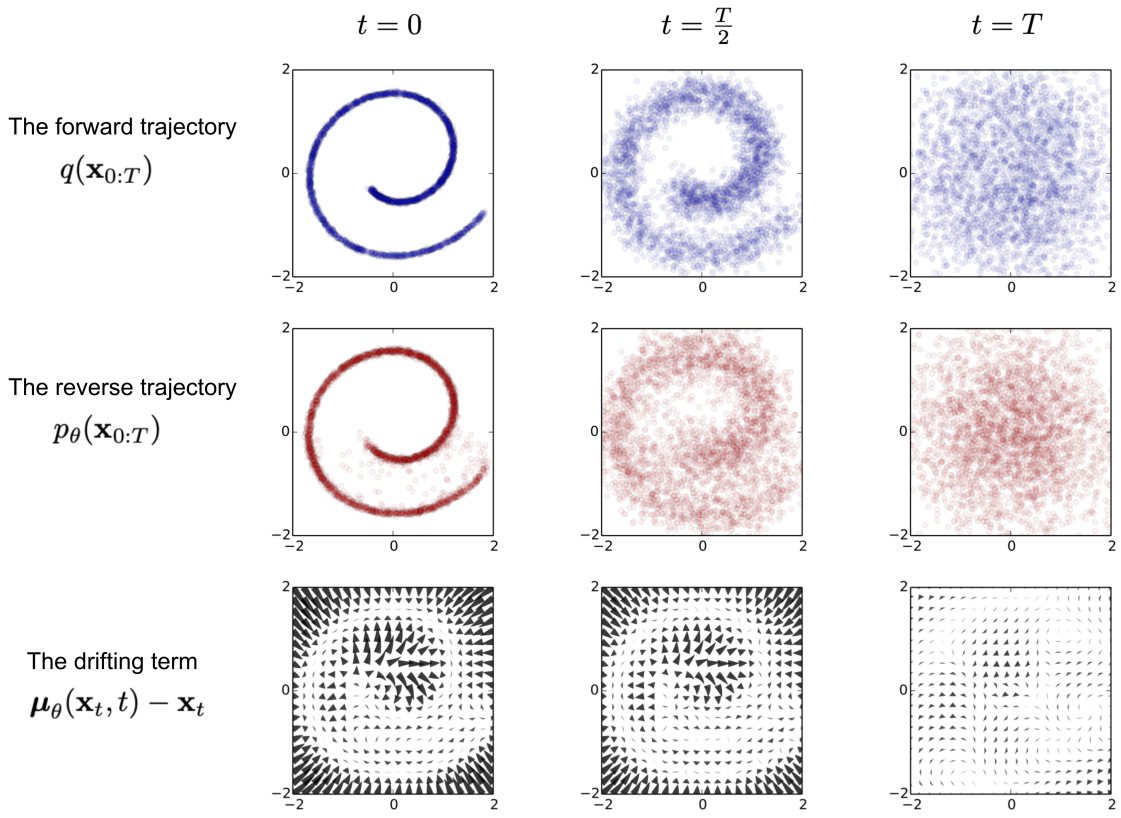


FIGURE 18. An example of training a diffusion model for modeling a 2D swiss roll data. (Image source: Sohl-Dickstein et al., 2015[5])

It is noteworthy that the reverse conditional probability is tractable when conditioned on $\mathbf{x}_0$:

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

Using Bayes' rule, we have:

$$\begin{aligned}
& q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \\
=& q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) \frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)} \\
\propto & \exp \left(-\frac{1}{2}\left(\frac{\left(\mathbf{x}_t-\sqrt{\alpha_t} \mathbf{x}_{t-1}\right)^2}{\beta_t}+\frac{\left(\mathbf{x}_{t-1}-\sqrt{\bar{\alpha}_{t-1}} \mathbf{x}_0\right)^2}{1-\bar{\alpha}_{t-1}}-\frac{\left(\mathbf{x}_t-\sqrt{\bar{\alpha}_t} \mathbf{x}_0\right)^2}{1-\bar{\alpha}_t}\right)\right) \\
= & \exp \left(-\frac{1}{2}\left(\frac{\mathbf{x}_t^2-2 \sqrt{\alpha_t} \mathbf{x}_t \mathbf{x}_{t-1}+\alpha_t \mathbf{x}_{t-1}^2}{\beta_t}+\frac{\mathbf{x}_{t-1}^2-2 \sqrt{\bar{\alpha}_{t-1}} \mathbf{x}_0 \mathbf{x}_{t-1}+\bar{\alpha}_{t-1} \mathbf{x}_0^2}{1-\bar{\alpha}_{t-1}}-\frac{\left(\mathbf{x}_t-\sqrt{\bar{\alpha}_t} \mathbf{x}_0\right)^2}{1-\bar{\alpha}_t}\right)\right) \\
= & \exp \left(-\frac{1}{2}\left(\left(\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}}\right) \mathbf{x}_{t-1}^2-\left(\frac{2 \sqrt{\alpha_t}}{\beta_t} \mathbf{x}_t+\frac{2 \sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}} \mathbf{x}_0\right) \mathbf{x}_{t-1}+C\left(\mathbf{x}_t, \mathbf{x}_0\right)\right)\right)
\end{aligned}$$

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 $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$):

$$\begin{aligned}
\tilde{\beta}_t &= 1/\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}\right) \\
&= 1/\left(\frac{\alpha_t - \bar{\alpha}_t + \beta_t}{\beta_t(1-\bar{\alpha}_{t-1})}\right) \\
&= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \\
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \left(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0\right) / \left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}\right) \\
&= \left(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0\right) \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \\
&= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}\beta_t}}{1-\bar{\alpha}_t}\mathbf{x}_0
\end{aligned}$$

Thanks to the nice property, if we are given sample $\mathbf{x}_t$ and ϵ_t, we can represent $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_t\right)$ and plug it into the above equation and obtain:

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

That is, given sample $\mathbf{x}_t.\epsilon$, we can recover such mean in the reverse diffusion process.

3.2.3. *Loss Function of Diffusion Model.* Such a setup is very similar to VAE and thus we can use the variational lower bound to optimize the negative log-likelihood.

$$\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &\leq -\log p_\theta(\mathbf{x}_0) + D_{\mathrm{KL}}\left(q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \| p_\theta(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\right) \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})/p_\theta(\mathbf{x}_0)}\right] \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log p_\theta(\mathbf{x}_0)\right] \\
&= \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \\
\text{Let } L_{\mathrm{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \geq -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0)
\end{aligned}$$

To convert each term in the equation to be analytically computable, the objective can be further rewritten to be a combination of several KL-divergence and entropy terms (See the detailed step-bystep process in Appendix B in Sohl-Dickstein et al., 2015):

$$
\begin{aligned}
L_{\text{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] \\
&= \mathbb{E}_q\Big[\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)}\Big] \\
&= \mathbb{E}_q\Big[-\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)}\Big] \\
&= \mathbb{E}_q\Big[-\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)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \Big(\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)}\Big) + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\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)}\Big] \\
&= \mathbb{E}_q\Big[-\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)}\Big] \\
&= \mathbb{E}_q\Big[\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)\Big] \\
&= \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}]
\end{aligned}
$$

Let's label each component in the variational lower bound loss separately:

$$
\begin{aligned}
L_{\text{VLB}} &= L_T + L_{T-1} + \cdots + L_0 \\
\text{where } L_T &= D_{\text{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_T\right)\right) \\
L_t &= D_{\text{KL}}\left(q\left(\mathbf{x}_t \mid \mathbf{x}_{t+1}, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_t \mid \mathbf{x}_{t+1}\right)\right) \text{ for } 1 \leq t \leq T-1 \\
L_0 &= -\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)
\end{aligned}
$$

Every KL term in L_{VLB} (except for L_0) compares two Gaussian distributions and therefore they can be computed in closed form. L_T is constant and can be ignored during training because q has no learnable parameters and $\mathbf{x}_T$ is a Gaussian noise. Ho et al. 2020[2] models L_0 using a separate discrete decoder derived from $\mathcal{N}\left(\mathbf{x}_0; \boldsymbol{\mu}_\theta\left(\mathbf{x}_1, 1\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_1, 1\right)\right)$.

3.2.4. *Parameterization of L_t for Training Loss.* Recall that we need to learn a neural network to approximate the conditioned probability distributions in the reverse diffusion process, $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$. We would like to train $\boldsymbol{\mu}_\theta$ to predict $\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$. Because $\mathbf{x}_t$ is available as input at

training time, we can reparameterize the Gaussian noise term instead to make it predict $\boldsymbol{\epsilon}_t$ from the input $\mathbf{x}_t$ at time step t :

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

$$\text{Thus } \mathbf{x}_{t-1} = \mathcal{N}(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

And indeed, as our goal is to approximate $q\left(\mathbf{x}_t \mid \mathbf{x}_{t+1}, \mathbf{x}_0\right)$ by $p_\theta\left(\mathbf{x}_{t-1}\right)$, so in partice, we can directly take

$$\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \tilde{\beta}_t$$

To visualize such reparameterize, we show the following sequence of figures:

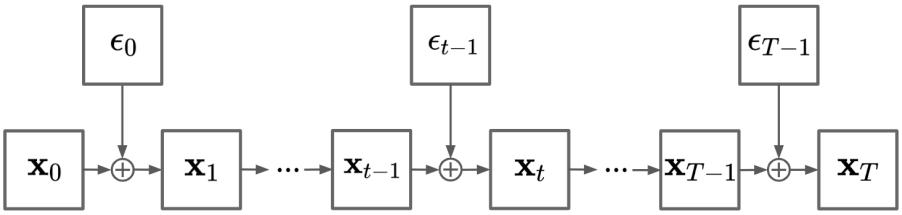


FIGURE 19. Forward process

In the forward process, we generate $\mathbf{x}_t$ by Gaussian noise ϵ_{t-1} and $\mathbf{x}_{t-1}$.

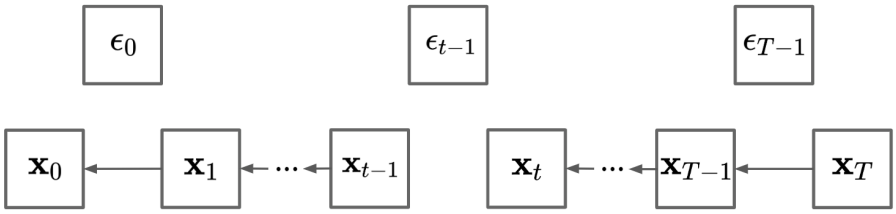


FIGURE 20. Forward process

In the reverse process, we don't know the Gaussian noise ϵ_{t-1} for give $\mathbf{x}_t$.

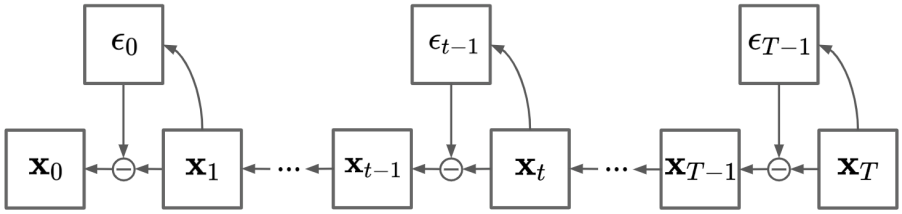


FIGURE 21. Forward process

So the goal, under reparameterize, is transferred to estimate the Gaussian we expected to add in the forward process.

In other words,

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, I)$$

Don't predict this Predict this instead

FIGURE 22. Effect of reparameterize

Here, to continue on mathematical derivation, we need to refer to a classical result in Statistics[1]

Proposition 3.1. *For two d-dimension Gaussian distributions* $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ *and* $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$*, we have:*

$$\begin{aligned} & D_{\mathrm{KL}}(\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \| \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)) \\ =& \frac{1}{2}\left(\operatorname{tr}(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - d + \log \frac{\det \boldsymbol{\Sigma}_2}{\det \boldsymbol{\Sigma}_1}\right). \end{aligned}$$

Proof. KL divergence between two distributions P and Q of a continuous random variable is given by:

$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)}$$

And probabilty density function of multivariate Normal distribution is given by:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

Now, let our two Normal distributions be $\mathcal{N}(\boldsymbol{\mu_1}, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu_2}, \boldsymbol{\Sigma}_2)$, both k dimensional.

$$\begin{aligned} D_{KL}(p\|q) &= \mathbb{E}_p[\log(p) - \log(q)] \\ &= \int \left[\frac{1}{2}\log\frac{|\boldsymbol{\Sigma_2}|}{|\boldsymbol{\Sigma_2}|} - \frac{1}{2}(x - \boldsymbol{\mu_1})^T \boldsymbol{\Sigma_2}^{-1}(x - \boldsymbol{\mu_1}) + \frac{1}{2}(x - \boldsymbol{\mu_2})^T \boldsymbol{\Sigma_2}^{-1}(x - \boldsymbol{\mu_2})\right] \times p(x)dx \\ &= \frac{1}{2}\log\frac{|\boldsymbol{\Sigma_2}|}{|\boldsymbol{\Sigma_2}|} - \frac{1}{2}\operatorname{tr}\left\{E[(x - \boldsymbol{\mu_1})(x - \boldsymbol{\mu_1})^T]\ \boldsymbol{\Sigma_2}^{-1}\right\} + \frac{1}{2}E[(x - \boldsymbol{\mu_2})^T \boldsymbol{\Sigma_2}^{-1}(x - \boldsymbol{\mu_2})] \\ &= \frac{1}{2}\log\frac{|\boldsymbol{\Sigma_2}|}{|\boldsymbol{\Sigma_2}|} - \frac{1}{2}\operatorname{tr}\{I_d\} + \frac{1}{2}(\boldsymbol{\mu_1} - \boldsymbol{\mu_2})^T \boldsymbol{\Sigma_2}^{-1}(\boldsymbol{\mu_1} - \boldsymbol{\mu_2}) + \frac{1}{2}\operatorname{tr}\{\boldsymbol{\Sigma_2}^{-1}\boldsymbol{\Sigma_2}\} \\ &= \frac{1}{2}\left[\log\frac{|\boldsymbol{\Sigma_2}|}{|\boldsymbol{\Sigma_2}|} - d + \operatorname{tr}\{\boldsymbol{\Sigma_2}^{-1}\boldsymbol{\Sigma_2}\} + (\boldsymbol{\mu_2} - \boldsymbol{\mu_1})^T \boldsymbol{\Sigma_2}^{-1}(\boldsymbol{\mu_2} - \boldsymbol{\mu_1})\right]. \end{aligned}$$

□

So by applying this proposition, we could have a close form of L_t as following(recall that we have already taken $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \tilde{\beta}_t$):

$$\begin{aligned} L_t &= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[D_{\mathrm{KL}}\left(q\left(\mathbf{x}_t \mid \mathbf{x}_{t+1}, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_t \mid \mathbf{x}_{t+1}\right)\right)\right] \\ &= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[\frac{1}{2}\left(\operatorname{tr}(\tilde{\beta}_t^{-1}\mathbf{I}\tilde{\beta}_t\mathbf{I}) + \frac{\left\|\boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right) - \tilde{\boldsymbol{\mu}}\left(\mathbf{x}_t, t\right)\right\|_2^2}{\tilde{\beta}_t} - d + \log(\frac{\det(\tilde{\beta}_t\mathbf{I})}{\det(\tilde{\beta}_t\mathbf{I})}\right)\right] \\ &= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[\frac{\left\|\boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right) - \tilde{\boldsymbol{\mu}}\left(\mathbf{x}_t, t\right)\right\|_2^2}{2\tilde{\beta}_t}\right] \end{aligned}$$

So the loss term L_t is parameterized to minimize the difference from $\tilde{\boldsymbol{\mu}}$:

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

Empirically, Ho et al.2020[2] found that training the diffusion model works better with a simplified objective that ignores the weighting term:

$$
\begin{aligned}
L_t^{\text{simple}} &= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\epsilon_t}\left[\left\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t,t\right)\right\|^2\right] \\
&= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\epsilon_t}\left[\left\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t,t\right)\right\|^2\right]
\end{aligned}
$$

The final simple objective is:

$$
L_{\text{simple}} = L_t^{\text{simple}} + C
$$

where C is a constant not depending on θ.

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** $\mathbf{x}_0$

FIGURE 23. The training and sampling algorithms in DDPM (Image source: Ho et al. 2020[2])

4. Latent Diffusion Model

Diffusion models are known for their powerful generative capabilities, producing high-quality and diverse outputs. However, they require significant computational resources and are challenging to train due to their high-dimensional pixel space operations. To address these challenges, we introduce the Latent Diffusion Model (LDM).

Latent Diffusion Model (LDM) is a powerful generative framework that combines the strengths of Variational Autoencoders (VAEs) and Diffusion Models. LDMs operate by first mapping data to a lower-dimensional latent space via a VAE, and

then employing a diffusion process within this latent space to generate high-quality, diverse outputs.

By transforming the training process from the pixel space to a lower-dimensional latent space, LDMs reduce computational demands and significantly shorten training time, while maintaining the generative strength of diffusion models.

In a word, LDM has the following advantages:

(1) Training in the latent space, which has a lower dimension than the pixel space, reduces the requirement of computation resources and saves training time.
(2) Our dataset could contain more pictures, greatly enhancing the output's quality.

The following pictures demonstrate the structure of the latent diffusion model.

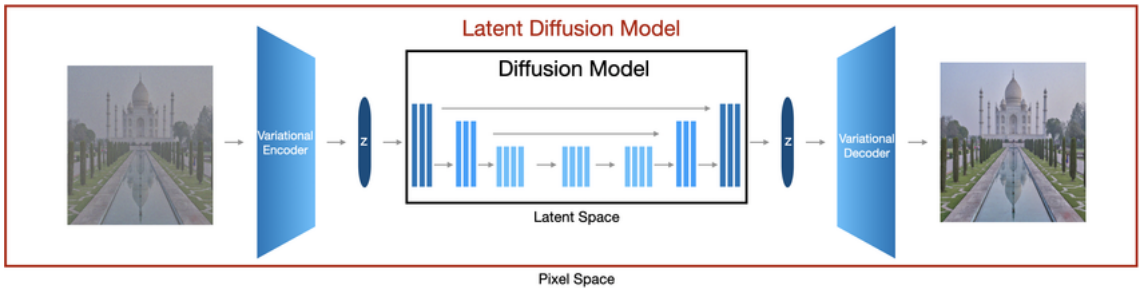
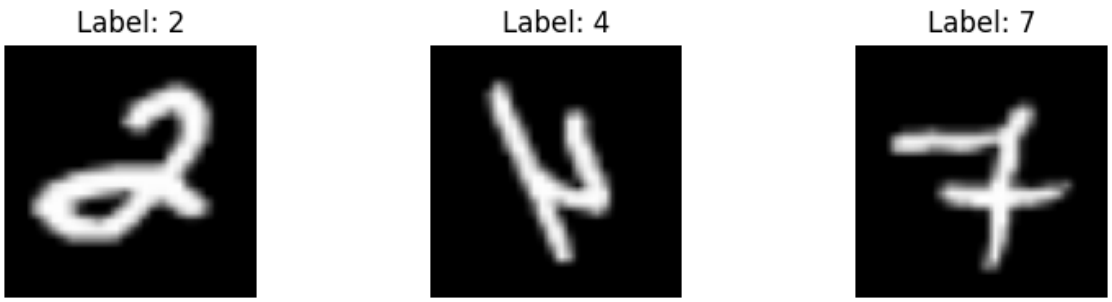


FIGURE 24. Latent diffusion model(Image source: Verma 2023[6])

5. Implementation and Results

5.1. **Coding part 1: VAE Implementation.** We build a VAE step by step (Stochastic_Process_Project_VAE.ipynb). It is responsible to generate the latent representation in latent space for the diffusion model to be trained with.

The following are examples in our dataset:



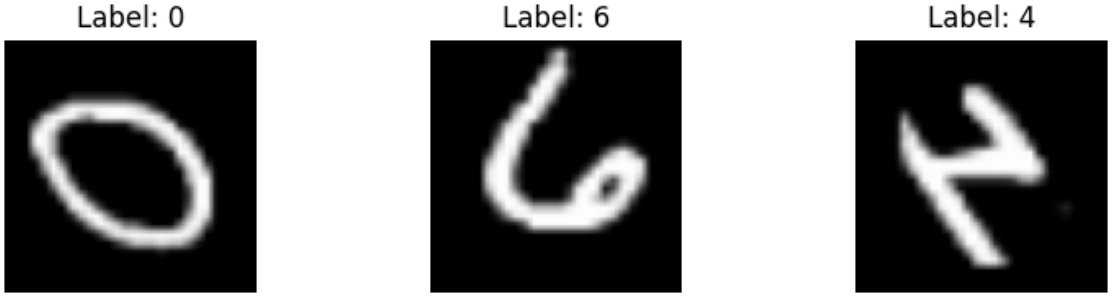


FIGURE 26. Dataset used in VAE

The following the is the training process, You can find that the model behaves better along with epoch grows:

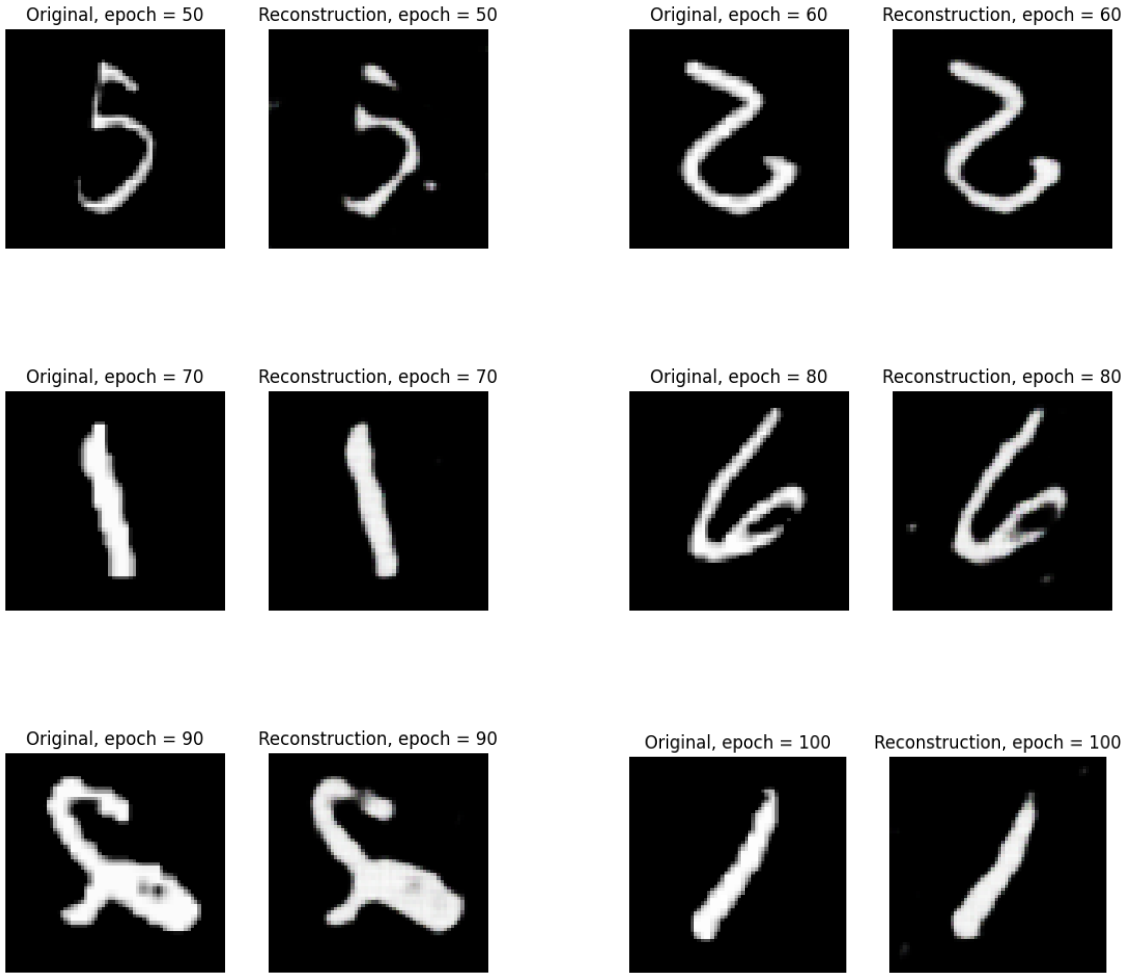


FIGURE 29. Training process in VAE

We are also interested in what latent space looks like exactly.
The following shows pictures from the latent spaces:

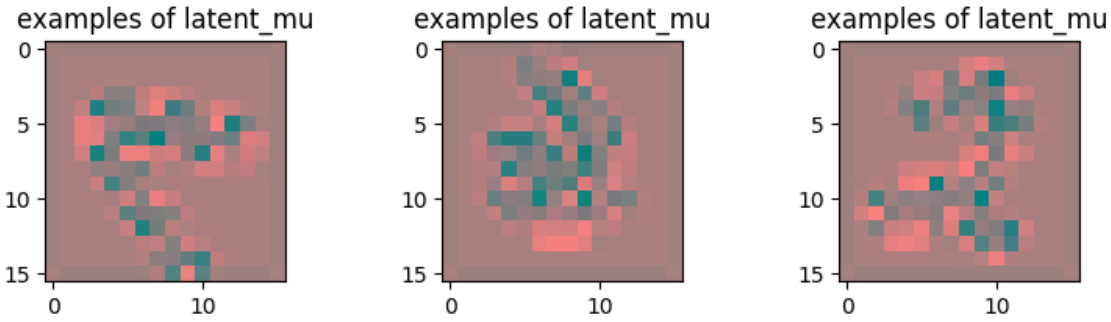
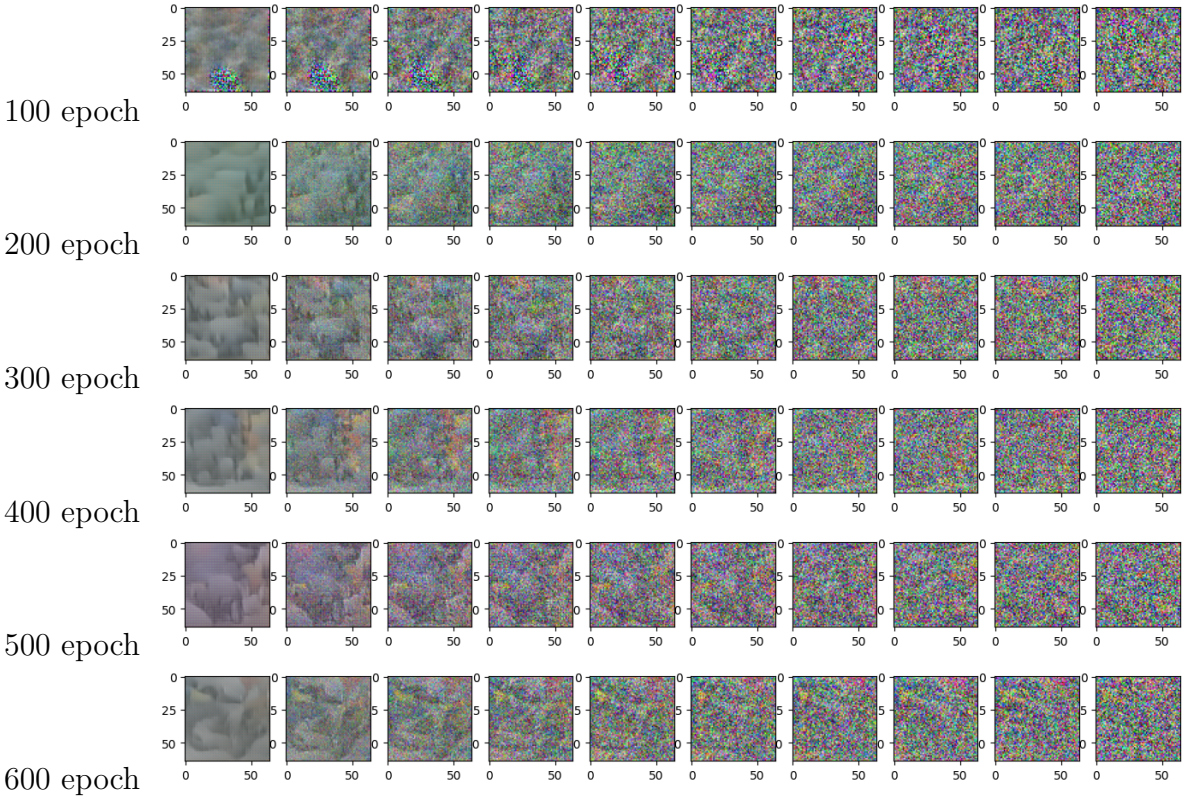
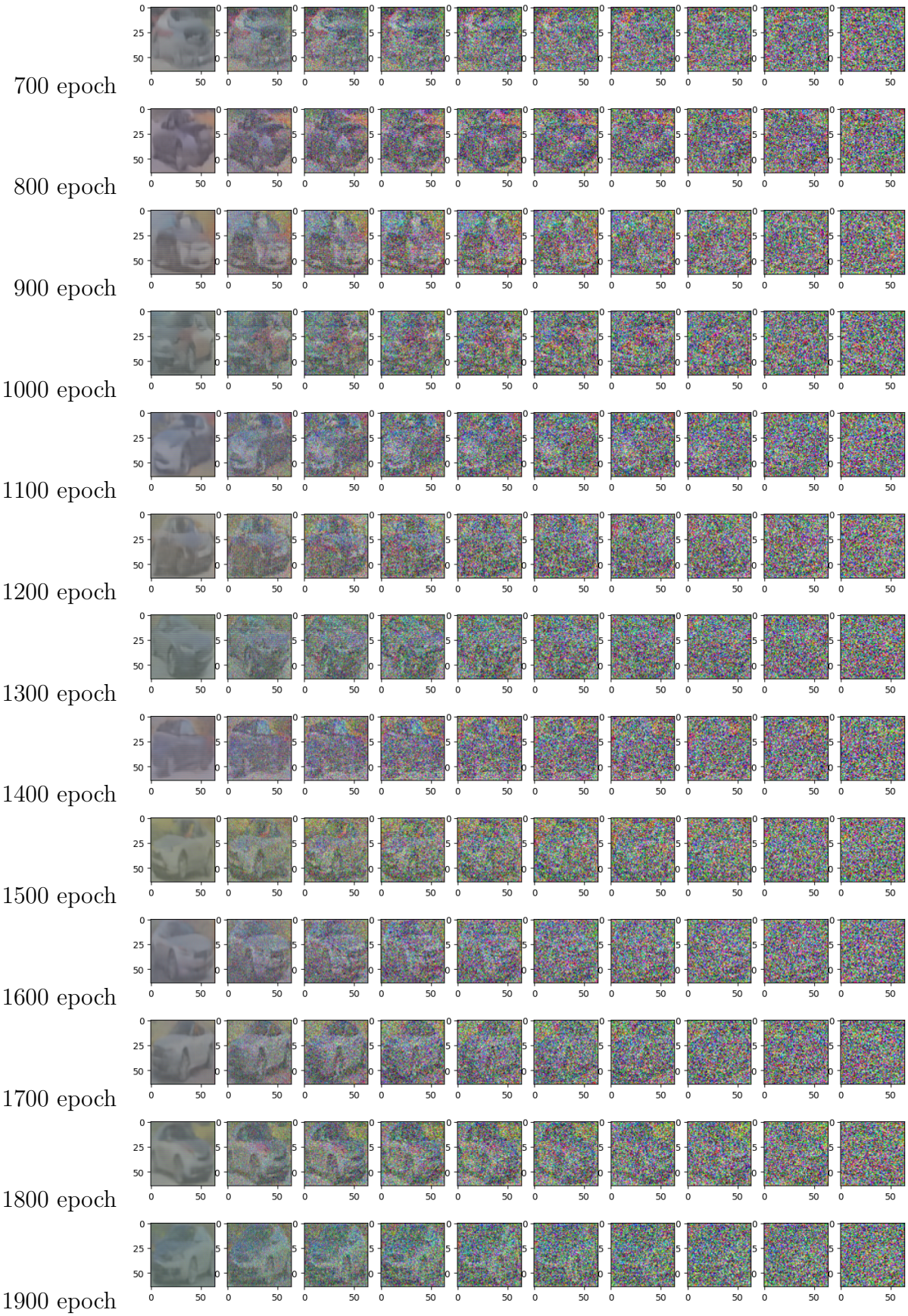
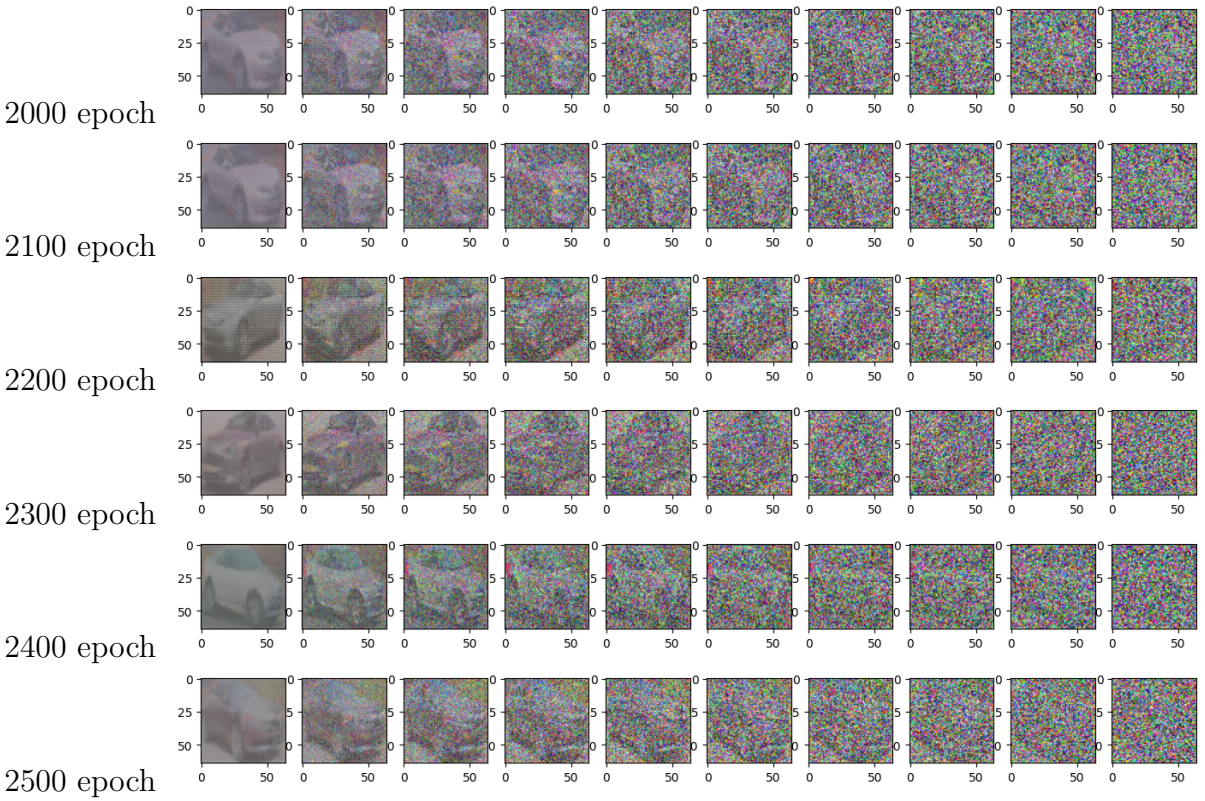


FIGURE 30. The visualization of the samples in the latent space

5.2. **Coding part 2: Diffusion Model Demo.** In the coding part, we first provide a complete version of diffusion model which is trained to generate car images(Stochastic_Process_Project_DM.ipynb). This notebook has a complete implementation of diffusion model pipeline. We visualized the forward process of diffusion and trained the U-Net architecture to predict noise in the backward diffusion process. We trained the model for 2500 epochs and demonstrated the intermediate performance of the model to generate cars from random noise every 100 epochs. After about 700 epochs of training, the generated car is already of nice looking. And here is the final performance of our model after 2500 epochs of training, which took approximately 40 min:




700 epoch
800 epoch
900 epoch
1000 epoch
1100 epoch
1200 epoch
1300 epoch
1400 epoch
1500 epoch
1600 epoch
1700 epoch
1800 epoch
1900 epoch



5.3. **Coding part 3: LDM Implementation.** The third notebook is about LDM with pre-trained VAE(Stochastic_Process_Project_LDM.ipynb), which appears not difficult at all as it is just a combination of the VAE and the diffusion model we have became familiar with in the first two notebooks. We first load the pretrained VAE model to generate a latent representation dataset of MNIST. Then we walked through the implemention of the diffusion process in this latent space, including the forward diffusion process, the U-Net for backward process and the loss function. We first train our model using the whole MNIST dataset. But considering the 0 to 9 digits are different in shape, for better performance, we then make our LDM only learn to generate images of "4", which comes out better as we expected. As in diffusion model notebook, we show our intermediate performance of our LDM model every 50 epochs during training. It is worth noting that, there is an interesting modification on the U-Net architecture in this LDM part compared to the U-Net we use in the ordinary diffusion part. There is a discussion about the necessity of this in our notebook. The reason in short is that we are applying diffusion model in the latent space where our images are compressed. So we can not downsample our latents as many times as we do in ordinary diffusion.

Here we generate our training dataset using forward diffusion process:

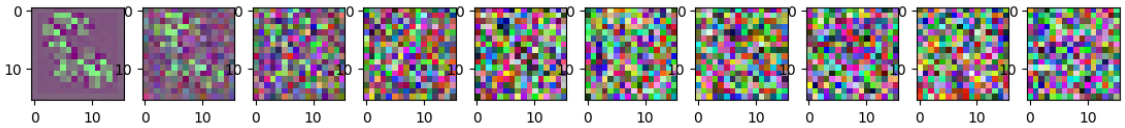
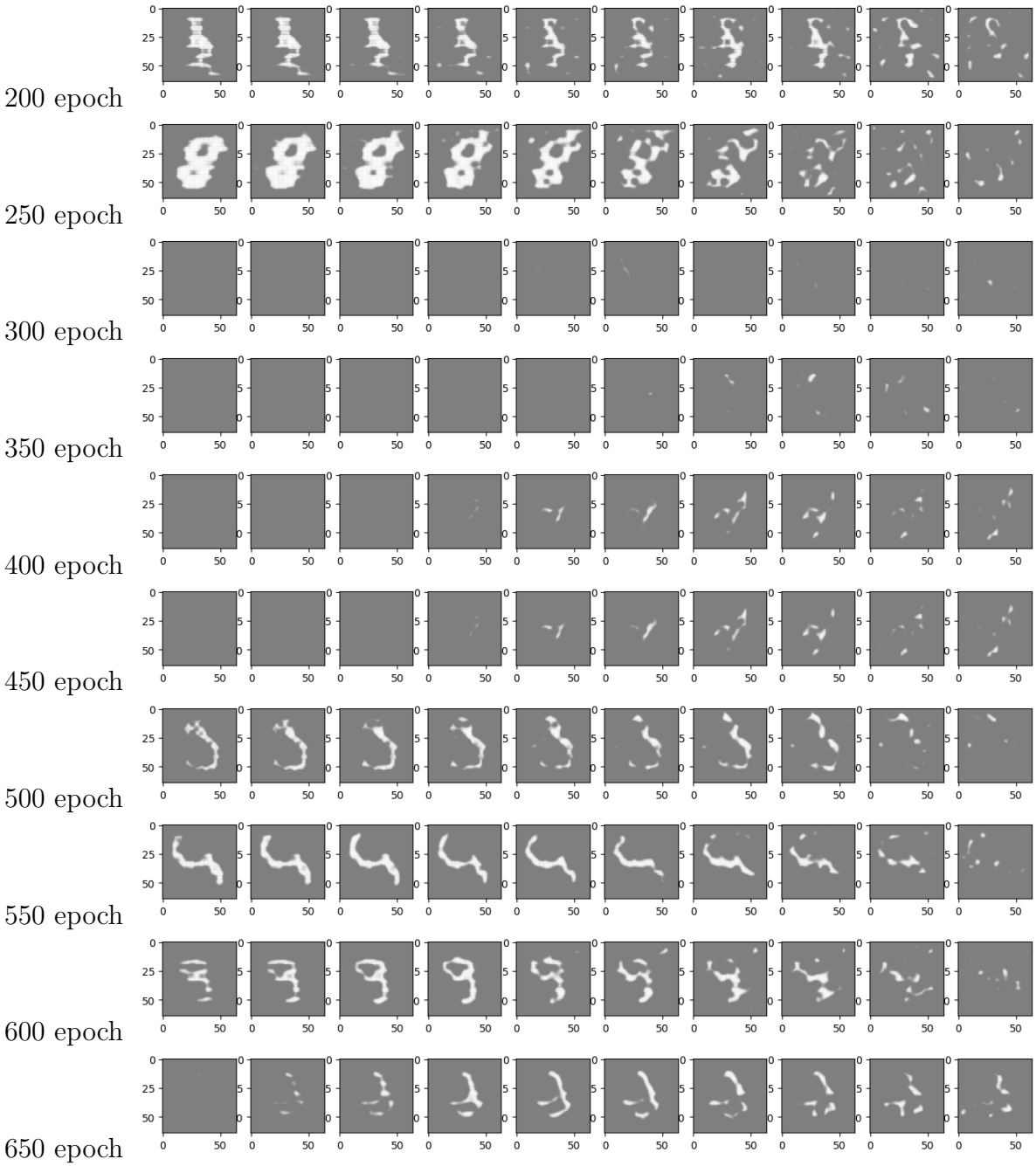


FIGURE 31. Forward diffusion process of LDM in latent space

When we are training our LDM model, first we train it without specifying the exact number to generate, so it turns our that we fail to generate meaningful pictures!



Then we tried to train with datasets that only contained 4 and generated only 4, and we succeeded. It somehow tells us that if we want to generate all numbers, text code will be needed.

Here we generate our training dataset for '4' using forward diffusion process:

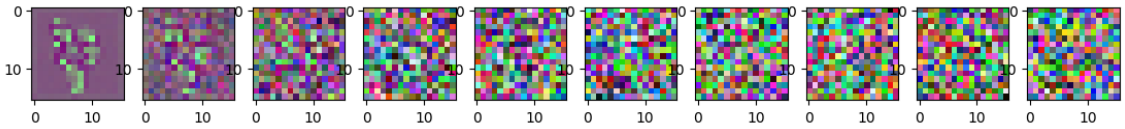
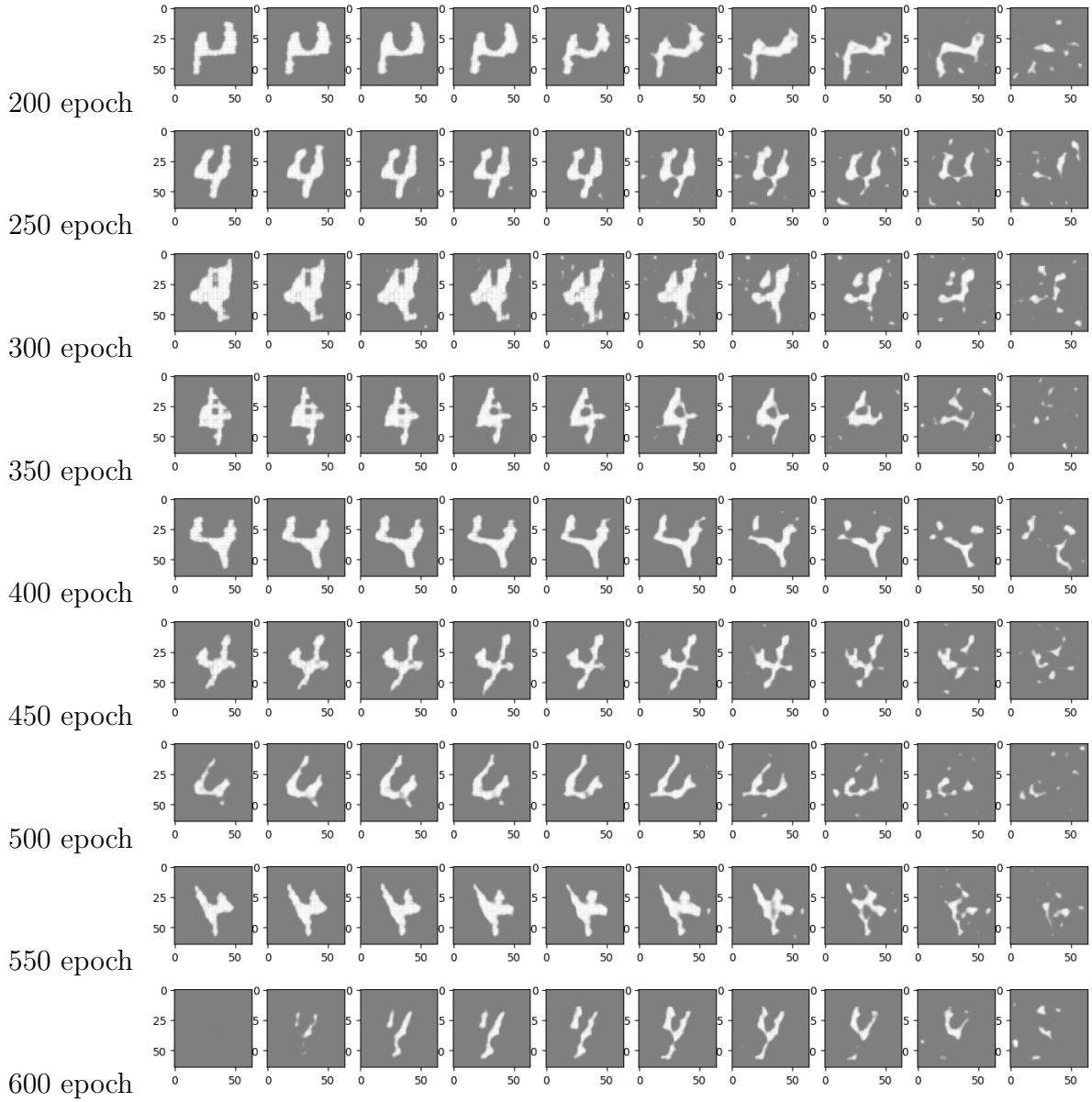
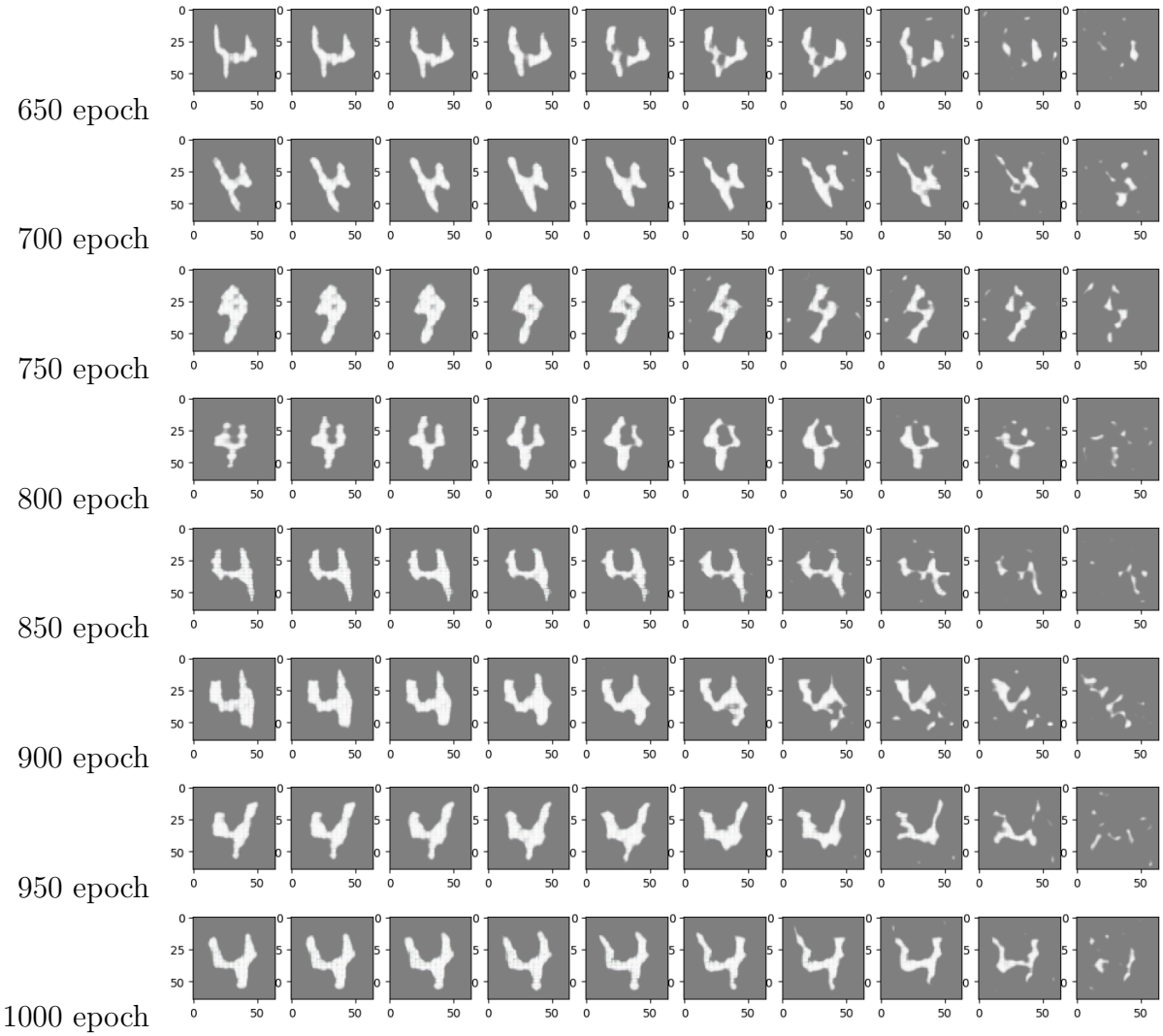


FIGURE 32. Forward diffusion process of LDM in latent space for only 4

Then we successfully trained our model which could generate '4' and it behaved better along with the epoch growth.





6. Conclusions and Discussion

In this project, we conducted an in-depth investigation of three generative models: Variational Autoencoder (VAE), Diffusion Model (DM), and Latent Diffusion Model (LDM). These models are intrinsically linked to key concepts in stochastic processes, such as sampling, Markov Chains, and Gaussian Processes.

For each model, we first derived the underlying mathematics to grasp the intuition behind their respective mathematical formulations. Subsequently, we implemented and trained the VAE and diffusion model independently, providing visualizations to demonstrate their performance.

To enhance both training and generation speed, we integrated the VAE with the diffusion model, enabling the diffusion process to occur in the latent space created by the VAE. This latent space has a lower dimensionality compared to the original pixel space, leading to significant computational efficiency.

Performance visualizations for the combined model highlight the improvements in efficiency and effectiveness, showcasing the advantages of operating within a reduced dimensional latent space.

Overall, our investigation and implementation underscore the potential of combining generative models to leverage their individual strengths, offering valuable insights for future research and applications in this domain.

7. Contribution

The contributions of each group member to this project are detailed below:

Kaitian Chao: Mathematical derivations; programming and coding implementation; presentation delivery; report writing.

Bangjie Wang: Mathematical derivations; presentation delivery; creation of presentation slides; report writing.

Guanqiu Wu: Mathematical derivations; creation of presentation slides; report writing.

References

1. Christopher M Bishop and Nasser M Nasrabadi, *Pattern recognition and machine learning*, vol. 4, Springer, 2006.
2. Jonathan Ho, Ajay Jain, and Pieter Abbeel, *Denoising diffusion probabilistic models*, Advances in neural information processing systems **33** (2020), 6840–6851.
3. Diederik P Kingma and Max Welling, *Auto-encoding variational bayes*, 2022.
4. Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer, *High-resolution image synthesis with latent diffusion models*, Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022, pp. 10684–10695.
5. Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, International conference on machine learning, PMLR, 2015, pp. 2256–2265.
6. Nikhil Verma, *Diffusion idea exploration for art generation*, arXiv preprint arXiv:2307.04978 (2023).
7. Lilian Weng, *From autoencoder to beta-vae*, lilianweng.github.io (2018).

School of Information Science and Technology, ShanghaiTech University
Email address: chaokt@shanghaitech.edu.cn

Institute of Mathematical Sciences, ShanghaiTech University
Email address: wangbj@shanghaitech.edu.cn

Institute of Mathematical Sciences, ShanghaiTech University
Email address: wugq@shanghaitech.edu.cn