A replication of DDPM and DDIM on MNIST

Yifan Li
Peking University
2100012520@stu.pku.edu.cn

Abstract

Denoising diffusion probabilistic models (DDPMs) have achieved high quality image generation without adversarial training. Diffusion Models learn to generate an image by taking successive denoising steps starting from a random noise. It is so elegant in math, and I make a replication of DDPM and its faster sample method, DDIM, on MNIST, to explore this powerful generative model and enjoy the combination of statistics and coding. The base of the code is refer to this colab jypyter notebook and I make some modifications on the DDIM part and the training process. The code URL is here.

1. Background

1.1. Diffusion: Start from Noise

Given samples from a data distribution $q(x_0)$, the goal of generative models is to learn a distribution $p_\theta(x_0)$ that approximates $q(x_0)$ and easy to sample from. Denoising diffusion probabilistic models (DDPMs) [1, 4] are latent variable models of the form

$$p_\theta(x_0) = \int p_\theta(x_{0:T}) \mathrm{d}x_{1:T}, \tag{1}$$

where $p_\theta(x_{0:T}) := \prod_{t=1}^{T} p_\theta^{(t)}(x_{t-1}|x_t)$, and $x_1, ..., x_T$ are latent variables in the same sample space as x_0. The parameters θ are learned to fit the data distribution $q(x_0)$ by maximizing a variational lower bound:

$$\begin{aligned} &\max_\theta \mathbb{E}_{x_0 \sim q(x_0)}[\log p_\theta(x_0)] \\ &\leq \max_\theta \mathbb{E}_{q(x_0, x_1, ..., x_T)}[\log p_\theta(x_{0:T}) - \log q(x_{1:T}|x_0)]. \end{aligned} \tag{2}$$

Unlike typical latent variable models (such as the variational autoencoder [2], DDPM [1] is learned with a fixed (rather than trainable) inference procedure $q(x_{1:T}|x0)$:

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{\alpha_t} x_{t-1}, (1 - \alpha_t)\boldsymbol{I}), \tag{3}$$

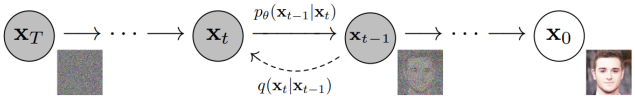

Figure 1. Illustration of DDPM

Thanks to the elegant properties of normal distribution and Markov chain, we can get:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)\boldsymbol{I}), \tag{4}$$

where α_t is a hyperparameter and $\bar{\alpha}_t := \prod_{i=1}^{t} \alpha_i$. More concreate mathmetics can be seen at this nice blog [6].

This is called the *forward process* due to the autoregressive nature of the sampling procedure (from x_0 to x_T). We call the latent variable model $p_\theta(x_{0:T})$, which is a Markov chain that samples from x_T to x_0, the *generative process*, since it approximates the intractable reverse process $q(x_{t-1}|x_t)$. Intuitively, the forward process progressively adds noise to the observation x_0, whereas the generative process progressively denoises a noisy observation, as showed in Fig.1.

1.2. Faster and Demterministic: DDIM

As we know how DDPM works, we can find some disadvantages of it. The most severe one is that DDPM has to sample T times sequentially to generate an image. As a result, sampling from DDPM is much slower than sampling from other deep generative models, which makes them impractical for tasks where compute is limited and latency is critical.

There are some cheap ideas to reduce the time complexity of DDPM straightforwardly. The first idea is to shrink the sampling timesteps. But this idea doesn't work well, because the setting of α_t is strict. On one hand, during the forward process when we add noises from x_{t-1} to x_t, from the definition of Eq.(3) we have to set α_t to approximately be 1 so that the difference between x_{t-1} and x_t will not damage the training of the denoising model and we can maintain x_t is in normal distribution for the small seting of the variance of the noises. On the other hand, after we derive the

Eq.(4), we have to let $\bar{\alpha}_t$ to be near 0 so that x_T will approximately be a gaussian noise which is required by the essential philosophy of DDPM. So we have to set a rather big T (usually ~1,000) to meet the above constraints with the proper sequence of $\alpha_t, t = 0, 1, ..., T$.

Another idea to decrease the sampling steps in the inference time is to skip some steps in the above Markov chain. However, in the setting of DDPM we cannot do this, because in the derivation of the posterior probablity, the nice property of Markov process is utilized necessarily:

$$\begin{aligned} P(x_{t-1}|x_t, x_0) &= \frac{P(x_{t-1}, x_t, x_0)}{P(x_t, x_0)} \quad \text{(Bayes Formula)} \\ &= \frac{P(x_t|x_{t-1}, x_0) \cdot P(x_{t-1}|x_0) \cdot P(x_0)}{P(x_t|x_0) \cdot P(x_0)} \quad \text{(Bayes Formula)} \\ &= \frac{P(x_t|x_{t-1}) \cdot P(x_{t-1}|x_0)}{P(x_t|x_0)} \quad \text{(Markov)}. \end{aligned} \tag{5}$$

As Eq.(4) is just a reparameterization trick because of the normal distribution, the real forward process is Eq.(3). So if we skip some steps during sampling, the process which is standing on Markov chain is no longer correct.

From this point of view, we can see that one of the basic setting of DDPM [1], the Markov chain process has been the most difficult barrier to accelerate the sampling procedure. Why not we drop this property off to discover a non-Markov process in the forward process? That is what DDIM exactly do.

Because of the discard of Markov, we have to stop at the second step in Eq.(5). Besides, Eq.(4) is no longer right without Markov property. To maintain the consistency of DDPM model in the training time, we have to keep Eq.(4) without the prior Markov chain setting, that is

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{6}$$

Then the core problem is to find a distribution $P(x_{t-1}|x_t, x_0)$ under this constraint. We can set this distribution in an easy way, for example, let

$$P(x_{t-1}|x_t, x_0) \sim \mathcal{N}(kx_0 + mx_t, \sigma^2 I). \tag{7}$$

Then we can use Eq.(6) to get the following equation:

$$\begin{aligned} x_{t-1} &= kx_0 + m(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon) + \sigma\epsilon', \\ \epsilon, \epsilon' &\sim \mathcal{N}(0, I). \end{aligned} \tag{8}$$

We can also do some identity transformations and merge the two gaussian noise into one. After that, we can get

$$\begin{aligned} x_{t-1} &= (k + m\sqrt{\bar{\alpha}_t})x_0 + \sqrt{m^2(1 - \bar{\alpha}_t) + \sigma^2}\epsilon, \\ \epsilon &\sim \mathcal{N}(0, I). \end{aligned} \tag{9}$$

We can also extend Eq.(6) to

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}} x_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{10}$$

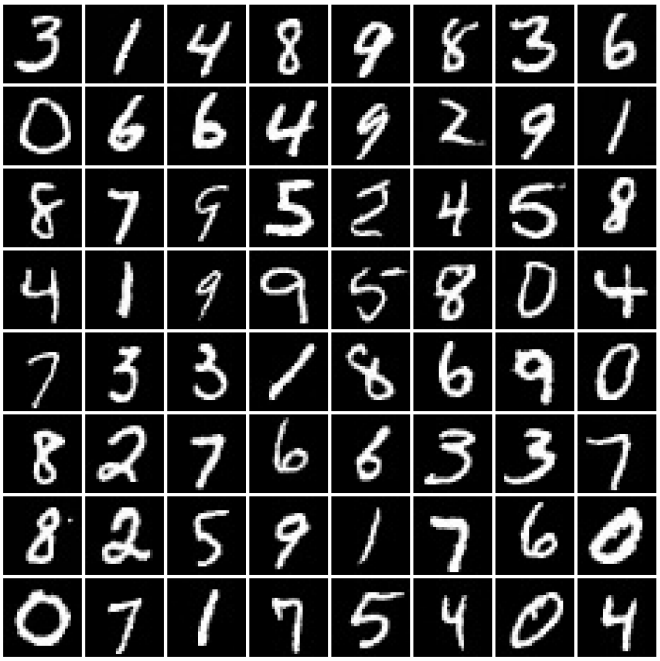

Figure 2. results of DDPM

To maintain Eq.(4) we have mentioned before, we have the following two equations:

$$\begin{aligned} k + m\sqrt{\bar{\alpha}_t} &= \sqrt{\bar{\alpha}_{t-1}}, \\ \text{and } m^2(1 - \bar{\alpha}_t) + \sigma^2 &= 1 - \bar{\alpha}_{t-1}. \end{aligned} \tag{11}$$

Finally, we can get m and k easily:

$$\begin{aligned} m &= \frac{\sqrt{1 - \bar{\alpha}_{t-1} - \sigma^2}}{\sqrt{1 - \bar{\alpha}_t}}, \\ \text{and } k &= \sqrt{\bar{\alpha}_{t-1}} - \sqrt{1 - \bar{\alpha}_{t-1} - \sigma^2}\frac{\sqrt{\bar{\alpha}_t}}{\sqrt{1 - \bar{\alpha}_t}}. \end{aligned} \tag{12}$$

That is what DDIM [5] defines in the forward process which benifits both on non-Markov property to accelerate the inference and one-step adding noise use Eq.(4) to make the training between DDPM and DDIM consistently. When we set σ=0, we got a deterministic sampling process.

2. Experiments for Replication

I replicate both DDPM training and DDIM sampling on Mnist dataset. Due to the lack of computing resources, I didn't try more dataset with a large resolution, but to tune the model to perform well on Mnist. All training is done on a single GeForce RTX 2080 Ti GPU.

For more experimental details, I use UNet architechture as the denoise model. I set the initial dimension as 64, learning rate as 5e-4, batch size as 64, classic ResNet architechture in UNet and the number of attention head as 4.

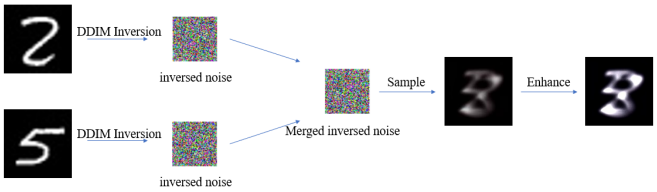

Figure 3. use Inversion-Merge to make "2+5=8" come true

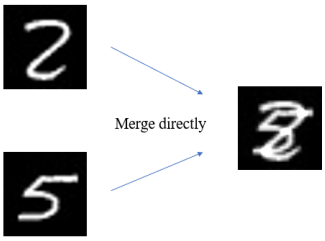

Figure 4. merge two images straightforwardly may cause chaos

More details can be found in my source codes. I compute FID score with this pytorch-fid tool repository with 2048 images from original dataset and results. After 186k iters, the model can generate some good results with a FID=9.32. Although it is somewhat poor, the main problem is the inaccuracy of some hyperparameters but not the implementations. So I think this score and visulization results in Fig.2 is satisfying.

For the implementation of DDIM sampling, I adopted part of codes in this colab notebook. It is so interesting to see DDIM sampling results in 1.5 seconds while DDPM needs about 43 seconds for batch size=64. the FID of DDIM sampling method is 11.61 with the same setting of DDPM. You can also see this part of my code in this URL.

3. Limitations and Future Work

3.1. Small Resolution of dataset

I have trained DDPM on Mnist and got a satisfying result both on visulization and FID, but Mnist is a rather small resolution dataset which is easy to fit its distribution. I once tried to train DDPM on LSUN, a dataset of indoor bedroom scene with 256x256 resolution and over 270k images. But I failed to run it even the batch size is 1. I think this experiment is somehow useless because the experiment on Mnist has made me feel the power of DDPM and DDIM, and the coding experience made me concentrate on more details which may be missed before.

3.2. Inversion-Merge Method

Probably I can try more inversion method based on DDIM because of DDIM provides a deterministic sampling process. There will also be a great fun to mix two latent noisy x_T and then denoise from it to see what will happen. For example, if I merge the latent noisy x_T of 5 and 2, will 8 comes out?

I adopted the architecture and pretrained model of Stable Diffusion [3] to implement the DDIM Inversion. Details can be found in Fig.(3). In a word, I want to merge two numbers which are complementary to each other to create a new number. I use simple linear interpolation with constant coefficient to merge the inversed noise to get the origin noise of the new number, but probably it is not the best way.

I tried for some other methods to merge. For example, use the difference ratio as the coefficient to construct the merged noise. But this doesn't work because it damages the origin noises a lot.

As for the explaination, the inversed noises can preserve the basic structure information of the origin images. When I merge two inversed noises, the merged noise load the structure information of two images. If this two images are complementary in structure, then a new meaningful images can be created, such as "2+5=8".

It is worthy to add a comparison to illustrate the difference between the Inversion-Merged method and merge from images straightforwardly, which can be seen at Fig.(4). Merge two images straightforwardly may cause chaos.

As a black-box model, DDPM provides us few to discover about its interpretation. If the relationship between the above merged inversed noise and the true inversed noise of "8" can be learned more clearly, I think it's a kind of progress to uncover the mystery of Diffusion Models.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1, 2

[2] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014. 1

[3] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 3

[4] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015. 1

[5] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 2

[6] Lilian Weng. What are diffusion models? *lilianweng.github.io*, Jul 2021. 1