Progressive Distillation of Diffusion Models

Pramook Khungurn

January 26, 2023

This note is written as I read "Progressive Distillation for Fast Sampling of Diffusion Models" by Salimans and Ho [SH22].

1 Introduction

- DDPMs can generate high quality samples. However, they are slow at sampling. Vanilla DDPMs (circa 2021) takes hundreds and thousands of models evaluations to sample one data item.
  - Note, however, that improvements by Karras et al. [KAAL22], for example, reduces the number of sampling steps to tens (20 to 50) in practice.
- The paper mentions that sampling a DDPM can be fast in "strongly conditioned settings."
  - Text-to-speech.
  - Image super-resolution.
  - Classifier-guided sampling [DN21].
- It is the slowest when less conditioning information is available.
  - Unconditional sampling.
  - Class conditional sampling.
- The paper claims to make three contributions.
  - A new parametermizations of DDPMs that make them more stable when sampling with a few steps.
  - A method to distill a deterministic sampler that use many steps to one that uses a few steps.
  - Showing that the distillation process does not take more time than the process to train the original model.
- The core algorithm's specification.
  - **Input.** A pre-trained DDPM that takes N steps to sample.
  - **Output.** A new DDPM that takes $N/2$ steps to sample.
- Progressive distillation = keep applying the core algorithm until it takes a few, say 4, steps to sample.
- Summary of results.
  - Start out with SOTA DDPM that takes 8192 steps to sample.
  - Distill it down to 4 steps.
  - Perceptual quality does not suffer much: FID of 3.0 on CIFAR-10 in 4 steps.

2 Background

- The paper uses the notations in introduced in Kingma et al.'s "Variational Diffusion Models" paper [KSPH21].
- A data item is denoted by $\mathbf{x}$. The data distribution is denoted by p_{data}.
- Diffusion models work on **latent variables** $\{\mathbf{z}_t : t \in [0,1]\}$. The latent variables form a Gaussian process that evolves over time according to the **forward process** $q(\mathbf{z}_t|\mathbf{x})$ where

$$q(\mathbf{z}_t|\mathbf{x}) = \mathcal{N}(\mathbf{z}_t; \alpha_t \mathbf{x}, \sigma_t^2 I)$$

  where α_t and σ_t are differentiable functions of t, collectively known as the **noise schedule**.
- Let $\lambda_t = \log(\alpha_t^2/\sigma_t^2)$. We call it the **signal-to-noise ratio (SNR)**. We require that the SNR decreases monotonically with t and that

$$q(\mathbf{z}_t|\mathbf{z}_s) = \mathcal{N}\left(\mathbf{z}_t; \frac{\alpha_t}{\alpha_s}\mathbf{z}_s; \sigma_{t|s}^2 I\right)$$

  where $0 \le s < t \le 1$ and

$$\sigma_{t|s}^2 = (1 - e^{\lambda_t - \lambda_s})\sigma_t^2.$$

- We train a neural network $\hat{\mathbf{x}}_{\boldsymbol{\theta}}$ so that $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) \approx \mathbf{x}$. In other words, $\hat{\mathbf{x}}(\mathbf{z}_t, \lambda_t)$ denoise $\mathbf{z}_t$ back to $\mathbf{x}$. The network is trained with the following loss:

$$E_{\mathbf{x}\sim p_{\text{data}}, t\sim\mathcal{U}([0,1]), \mathbf{z}_t\sim\mathcal{N}(\alpha_t\mathbf{x}, \sigma_t^2 I)}[w(\lambda_t)\|\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \mathbf{x}\|^2].$$

  Here, $\mathcal{U}([0,1])$ is the uniform distribution over the interval $[0,1]$. The weighting function $w(\lambda_t)$ will be discussed later.
- We can deduce that, for any $0 \le s < t \le 1$,

$$q(\mathbf{z}_s|\mathbf{z}_t, \mathbf{x}) = \mathcal{N}(\mathbf{z}_s; \tilde{\boldsymbol{\mu}}_{s|t}(\mathbf{z}_t, \mathbf{x}), \tilde{\sigma}_{s|t}^2 I)$$

  where

$$\tilde{\boldsymbol{\mu}}_{s|t}(\mathbf{z}_t, \mathbf{x}) = e^{\lambda_t - \lambda_s}\frac{\alpha_s}{\alpha_t}\mathbf{z}_t + (1 - e^{\lambda_t - \lambda_s})\alpha_s \mathbf{x},$$
$$\tilde{\sigma}_{s|t}^2 = (1 - e^{\lambda_t - \lambda_s})\sigma_s^2.$$

- We can use the above equation as a basis for the **ancestral sampling algortihm**.
  1. We have a sequence of times $0 = t_0 < t_1 < t_2 < \ldots < t_K = 1$.
  2. We start with $\mathbf{z}_1 \sim \mathcal{N}(0, I)$.
  3. Given that we have a sample at time t and we want sample at another time $s < t$, then,

$$\mathbf{z}_s = \tilde{\boldsymbol{\mu}}_{s|t}(\mathbf{z}_t, \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)) + \sqrt{(\tilde{\sigma}_{s|t}^2)^{1-\gamma}(\sigma_{t|s}^2)^{\gamma}}\,\boldsymbol{\xi}$$

     where $\xi \sim \mathcal{N}(\mathbf{0}, I)$, and γ is a hyperparameter that controls how much noise is added during sampling [ND21].

- Alternatively, $\mathbf{z}_t$ satisfies the following **probability flow ODE**:

$$\mathrm{d}\mathbf{z}_t = \left(f(\mathbf{z}_t, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{z}} \log \hat{p}_{\boldsymbol{\theta}}(\mathbf{z}_t) \right) \mathrm{d}t$$

where

$$\begin{aligned} \nabla_{\mathbf{z}} \log \hat{p}_{\boldsymbol{\theta}}(\mathbf{z}_t) &= \frac{\alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \mathbf{z}_t}{\sigma_t^2}, \\ f(\mathbf{z}_t, t) &= \frac{\mathrm{d} \log \alpha_t}{\mathrm{d}t} \mathbf{z}_t, \\ g(t)^2 &= \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2 \frac{\mathrm{d} \log \alpha_t}{\mathrm{d}t} \sigma_t^2. \end{aligned}$$

Sampling can then be done by simulating the ODE backward in time from $t = 1$ to $t = 0$.

- The DDIM sampler proposed by Song et al. [SME20] is given by

$$\begin{aligned} \mathbf{z}_s &= \alpha_s \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t) + \sigma_s \frac{\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t)}{\sigma_t} \\ &= e^{(\lambda_t - \lambda_s)/2} (\alpha_s / \alpha_t) \mathbf{z}_t + (1 - e^{(\lambda_t - \lambda_s)/2}) \alpha_s \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t). \end{aligned}$$

The paper claims that this can be understood in terms of integrating the probability flow ODE.

3 Progressive Distillation

- The core algorithm receives as input a **teacher model** that is trained in the standard way.

STANDARD-DIFFUSION-TRAINING

while not converged
  $\mathbf{x} \sim p_{\text{data}}$
  $t \sim \mathcal{U}([0, 1])$
  $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, I)$
  $\mathbf{z}_t \leftarrow \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\theta}$
  $\lambda_t \leftarrow \log(\alpha_t^2 / \sigma_t^2)$
  $L_{\boldsymbol{\theta}} \leftarrow w(\lambda_t) \| \mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) \|_2^2$
  Update $\boldsymbol{\theta}$ according to $\nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}}$.

- The result of the distillation process is a **student model**, which has the same architecture as the teacher model.
- The distillation process proceeds in iterations.
  - Each iteration halves the number of sampling steps required.
  - At the start of each iteration, the student model is initialized with the paremeters of the teacher model.
  - At the end of each iteration, the student model becomes the teacher model of the next round.
  - The training process in each iteration is similar to the standard training. However, the main difference is the target that $\hat{\mathbf{x}}_{\boldsymbol{\theta}}$ is asked to approximate from $\mathbf{z}_t$.
    - For the standard training, the target is $\mathbf{x}$.

    - For the distillation process, the target is the denoised data that would have been predicted by the teacher in two distillation steps.
  - Another difference is that the distillation process always work in discrete time instead of continuous time like the standard training.

- Here's the distillation algorithm.

PROGRESSIVE-DISTILLATION($\boldsymbol{\eta}, N$)

// $\boldsymbol{\eta}$ denotes the parameters of the trained teacher model.
// N is the number of iterations that the teacher model takes to sample.
for K iterations
  $N \leftarrow N/2$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\eta}$
  while not converged
    $\mathbf{x} \sim p_{\text{data}}$
    $i \sim \mathcal{U}(\{1, 2, \ldots, N\})$
    $t \leftarrow i/N$
    $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, I)$
    // 2 steps of DDIM sampling with teacher model
    $t' \leftarrow t - 0.5/N; \quad t'' \leftarrow t - 1/N$
    $\lambda_t \leftarrow \log(\alpha_t^2/\sigma_t^2); \quad \lambda_{t'} \leftarrow \log(\alpha_{t'}^2/\sigma_{t'}^2)$
    $\mathbf{z}_{t'} \leftarrow \alpha_{t'}\hat{\mathbf{x}}_{\boldsymbol{\eta}}(\mathbf{z}_t, \lambda_t) + \frac{\sigma_{t'}}{\sigma_t}\big(\mathbf{z}_t - \alpha_t\hat{\mathbf{x}}_{\boldsymbol{\eta}}(\mathbf{z}_t, \lambda_t)\big)$
    $\mathbf{z}_{t''} \leftarrow \alpha_{t''}\hat{\mathbf{x}}_{\boldsymbol{\eta}}(\mathbf{z}_{t'}, \lambda_{t'}) + \frac{\sigma_{t''}}{\sigma_{t'}}\big(\mathbf{z}_{t'} - \alpha_{t'}\hat{\mathbf{x}}_{\boldsymbol{\eta}}(\mathbf{z}_{t'}, \lambda_{t'})\big)$
    $\tilde{\mathbf{x}} \leftarrow \frac{\mathbf{z}_{t''} - (\sigma_{t''}/\sigma_t)\mathbf{z}_t}{\alpha_{t''} - (\sigma_{t''}/\sigma_t)\alpha_t}$
    $L_{\boldsymbol{\theta}} \leftarrow w(\lambda_t)\|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)\|^2$
    Update $\boldsymbol{\theta}$ according to $\nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}}$
  // Make the converged student the teacher of the next round.
  $\boldsymbol{\eta} \leftarrow \boldsymbol{\theta}$

- The paper notes that using the value predicted by the teacher in two steps as a target makes the prediction task for the student model much easier.
  - If using $\mathbf{x}$ as the target, the model must predict the average of possible $\mathbf{x}$ value, which produces blurry predictions.
  - However, the value predicted by the teacher model is completely determined by the teacher model and $\mathbf{z}_t$. So, the prediction is sharp.

  As a result, the student model can make progress much faster than the vanilla teacher model can during sampling.

4 Parameterization and Training Loss

- This section is about the details of the models: α_t, σ_t, $w(\lambda_t)$, and what $\hat{\mathbf{x}}_{\boldsymbol{\theta}}$ is.
- The paper assumes a variance preserving model: $\sigma_t^2 = 1 - \alpha_t^2$.
- Cosine schedule is used: $\alpha_t = \cos(0.5\pi t)$.

4.1 Model Parameterization

- Many works on DDPM parameterize $\hat{\mathbf{x}}_{\boldsymbol{\theta}}$ by requiring it to predict the noise $\boldsymbol{\xi}$ that is used to make $\mathbf{z}_t$ from $\mathbf{x}$. In other words, our network is $\hat{\boldsymbol{\xi}}_{\boldsymbol{\theta}}$, and we take

$$\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) = \frac{1}{\alpha_t}\big(\mathbf{z}_t - \sigma_t \hat{\boldsymbol{\xi}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)\big).$$

- While the above parametermization might work well for training the original model, the paper argues that this does not work well with distillation.
  - As distillation progresses, we increasingly evaluate at lower and lower signal-to-noise ratios.
  - As $\alpha_t \to 0$, the changes made by $\hat{\boldsymbol{\xi}}_{\boldsymbol{\theta}}$ gets amplified drastically. This is because

$$\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) = \frac{1}{\alpha_t}\big(\mathbf{z}_t - \sigma_t \hat{\boldsymbol{\xi}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)\big).$$

    So, changes are multipled by $1/\alpha_t$.
  - If we distill down to a single sampling step, the input to the model is pure noise $\boldsymbol{\xi}$, and the distilled $\hat{\boldsymbol{\xi}}_{\boldsymbol{\theta}}$ would have to predict $\boldsymbol{\xi}$. The link between $\boldsymbol{\xi}$-prediction and $\mathbf{x}$-prediction breaks down completely.
- So, for distillation to work, we need to parametermize the model so that the prediction $\hat{\mathbf{x}}_{\boldsymbol{\theta}}$ remains stable as the SNR varies. The paper tried a number of combinations and found all of them to work well.
  - Predicting $\mathbf{x}$ directly.
  - Predicting both $\mathbf{x}$ and $\boldsymbol{\xi}$, resulting in $\tilde{\mathbf{x}}$ and $\tilde{\boldsymbol{\xi}}$, and then merging via

$$\hat{\mathbf{x}} = \sigma_t^2 \tilde{\mathbf{x}} + \alpha_t(\mathbf{z}_t - \sigma_t \tilde{\boldsymbol{\xi}}).$$

    Note that $\sigma_t^2 \tilde{\mathbf{x}} \approx \sigma_t^2 \mathbf{x}$, and

$$\alpha_t(\mathbf{z}_t - \sigma_t \tilde{\boldsymbol{\xi}}) = \alpha_t(\alpha_t \mathbf{x} + \sigma_t \boldsymbol{\theta} - \sigma_t \tilde{\boldsymbol{\xi}}) \approx \alpha_t^2 \mathbf{x}.$$

    Because $\alpha_t^2 + \sigma_t^2 = 1$, we have that $\sigma_t^2 \tilde{\mathbf{x}} + \alpha_t(\mathbf{z}_t - \sigma_t \tilde{\boldsymbol{\xi}}) \approx \sigma_t^2 \mathbf{x} + \alpha_t^2 \mathbf{x}$ is an interpolation between the $\mathbf{x}$-prediction and the implied $\mathbf{x}$ from $\boldsymbol{\xi}$-prediction.
  - Predicting $\mathbf{v} := \alpha_t \boldsymbol{\xi} - \sigma_t \mathbf{x}$ and setting $\hat{\mathbf{x}} := \alpha_t \mathbf{z}_t - \sigma_t \hat{\mathbf{v}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)$.
- The paper also tried all the above combinations when training the original models, and it found that they worked well too.

4.2 Weighting Function

- In the 2020 work by Ho et al. [HJA20], they sample the time uniformly and compute the squared L2-distance between $\boldsymbol{\xi}$ and $\hat{\boldsymbol{\xi}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)$. One can deduce that

$$L_{\boldsymbol{\theta}} = \|\boldsymbol{\xi} - \boldsymbol{\xi}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)\|^2 = \frac{\alpha_t^2}{\sigma_t^2}\|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)\|^2.$$

  This is called the **SNR** weighting scheme. This means that the loss gives zero weight to data with zero SNR. This is not a suitable loss for distillation.
- The paper tried the following choices of weighting scheme.

- **Truncated SNR**: $L_{\boldsymbol{\theta}} = \max(\|\mathbf{x} - \hat{\mathbf{x}}\|^2, \|\boldsymbol{\xi} - \hat{\boldsymbol{\xi}}\|^2) = \max(\frac{\alpha_t^2}{\sigma_t^2}, 1)\|\mathbf{x} - \hat{\mathbf{x}}\|^2$.
- **SNR+1**: $L_{\boldsymbol{\theta}} = \|\mathbf{v} - \hat{\mathbf{v}}\| = (1 + \frac{\alpha_t^2}{\sigma_t^2})\|\mathbf{x} - \hat{\mathbf{x}}\|^2$.

It found that both worked well.

- The authors did not put much attention on how the time is sampled. They just sampled it uniformly randomly from $[0, 1]$.

5 Experiments

5.1 Model Parameterization and Training Loss

- The paper tried the 4 combinations of model parameterization and 3 combination of weighting schemes.
  - One case lead to training divergence: predicting $\boldsymbol{\xi}$ and using the truncated SNR weighting scheme.
  - In all other cases, the numbers are similar.
- The paper observed that predicting $\mathbf{v}$ is the most stable method because it makes the DDIM step-sizes independent of the SNR. However, predicting $\mathbf{x}$ gives the best empirical results.

5.2 Progressive Distillation

- The authors started with a teacher model trained on continous time. They then started distilling from $N = 8192$ (or $N = 1024$ for bigger models) down to 1.
- In each distillation iteration, they optimize the model for 50 000 parameter updates. The exceptions are when distilling to 2 or 1 step(s), which they took 100 000 updates. They claimed that the total time used to distill does not exceed that used to train the original model.
- FID scores of distilled models increases very slowly until $N = 4$, after which they rises very rapidly. So, $N = 4$ seems to be a sweet spot between speed and quality.
- Undistilled models' performance degrades very fast after $N = 128$.

A Relationship between DDIM sampling procedure and the probability flow ODE

- The probability flow ODE [SSDK+21] reads

$$\mathrm{d}\mathbf{z}_t = \left(f(\mathbf{z}_t, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{z}} \log p_t(\mathbf{z}) \right) \mathrm{d}t.$$

So, computing $\mathbf{z}_s$ from $\mathbf{z}_t$ according to it would look like

$$\mathbf{z}_s = \left(f(\mathbf{z}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{z}} \log p_t(\mathbf{z}) \right)(s - t)$$

- However, DDIM's update rule is

$$\mathbf{z}_s = \alpha_s \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) + \sigma_s \frac{\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)}{\sigma_t},$$

which, from the look of it, is very different from the update one would derive naturally from the probability flow ODE.

- However, the two updates are related. We just have not written the probability flow ODE in the right form. To do so, we first need to write the probability flow ODE in terms of the log SNR $\lambda_t = \log(\alpha_t^2/\sigma_t^2)$.
- The first task is to write $f(\mathbf{z}_t, t)$ and $g(t)$ in terms of α_t and σ_t. We first observe that $f(\mathbf{z}_t, t)$ is actually $f(t)\mathbf{z}_t$ in the settings of DDIM. So, the SDE corresponding to the probability flow ODE becomes

$$\mathrm{d}\mathbf{z}_t = f(t)\mathbf{z}_t \,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{W}.$$

- Solving the SDE using the derivation in Karras et al.'s paper [KAAL22], we have that

$$\alpha_t = \exp\left(\int_0^t f(u)\,\mathrm{d}u\right).$$

So,

$$f(t) = \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}.$$

- Moreover, we have that

$$\sigma_t^2 = \alpha_t^2 \int_0^t \frac{g(u)^2}{\alpha_u^2}\,\mathrm{d}u$$

So,

$$\begin{aligned}
\frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} &= \frac{\mathrm{d}\alpha_t^2}{\mathrm{d}t}\left(\int_0^t \frac{g(u)^2}{\alpha_u^2}\,\mathrm{d}u\right) + \alpha_t^2\frac{g(t)^2}{\alpha_t^2} \\
\frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} &= 2\alpha_t\frac{\mathrm{d}\alpha_t}{\mathrm{d}t}\cdot\frac{\sigma_t^2}{\alpha_t^2} + g(t)^2 \\
\frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} &= 2\sigma_t^2\left(\frac{1}{\alpha_t}\frac{\mathrm{d}\alpha_t}{\mathrm{d}t}\right) + g(t)^2 \\
\frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} &= 2\sigma_t^2\frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t} + g(t)^2 \\
g(t)^2 &= \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2\sigma_t^2\frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}.
\end{aligned}$$

- Assuming a variance preserving process, we have that $\alpha_t^2 + \sigma_t^2 = 1$. So,

$$\mathrm{sigmoid}(\lambda_t) = \frac{1}{1+\exp(-\log(\alpha_t^2/\sigma_t^2))} = \frac{1}{1+\sigma_t^2/\alpha_t^2} = \frac{\alpha_t^2}{\alpha_t^2+\sigma_t^2} = \alpha_t^2.$$

Similarly,

$$\mathrm{sigmoid}(-\lambda_t) = \sigma_t^2.$$

As a result,

$$\begin{aligned}
\frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t} &= \frac{1}{2}\frac{\mathrm{d}\log\alpha_t^2}{\mathrm{d}t} = \frac{1}{2}\frac{\mathrm{d}\log(\mathrm{sigmoid}(\lambda_t))}{\mathrm{d}t} = \frac{1}{2}\frac{\mathrm{d}\log(\mathrm{sigmoid}(\lambda_t))}{\mathrm{d}\lambda_t}\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} = \frac{1}{2}\mathrm{sigmoid}(-\lambda_t)\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} \\
&= \frac{1}{2}\sigma_t^2\frac{\mathrm{d}\lambda_t}{\mathrm{d}t}.
\end{aligned}$$

Moreover,

$$\begin{aligned}
g(t)^2 &= \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2\sigma_t^2\frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t} = \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}\lambda_t}\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} - \sigma_t^4\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} = \frac{\mathrm{d}\,\mathrm{sigmoid}(-\lambda_t)}{\mathrm{d}\lambda_t}\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} - \sigma_t^4\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} \\
&= -\mathrm{sigmoid}(-\lambda_t)(1-\mathrm{sigmoid}(-\lambda_t))\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} - \sigma_t^4\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} = (\sigma_t^4-\sigma_t^2)\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} - \sigma_t^4\frac{\mathrm{d}\lambda_t}{\mathrm{d}t} \\
&= -\sigma_t^2\frac{\mathrm{d}\lambda_t}{\mathrm{d}t}.
\end{aligned}$$

- Because

$$\nabla_{\mathbf{z}} \log p_t(\mathbf{z}) \approx \frac{\alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \mathbf{z}_t}{\sigma_t^2},$$

we have that the probability flow ODE can be written as

$$\begin{aligned}
\mathrm{d}\mathbf{z}_t &= \left(\frac{1}{2}\sigma_t^2 \frac{\mathrm{d}\lambda_t}{\mathrm{d}t}\mathbf{z}_t + \frac{1}{2}\sigma_t^2 \frac{\mathrm{d}\lambda_t}{\mathrm{d}t} \frac{\alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \mathbf{z}_t}{\sigma_t^2} \right) \mathrm{d}t \\
&= \frac{1}{2}\Big(\sigma_t^2 \mathbf{z}_t + \alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \mathbf{z}_t \Big) \mathrm{d}\lambda_t \\
&= \frac{1}{2}\Big(\alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) + (\sigma_t^2 - 1)\mathbf{z}_t \Big) \mathrm{d}\lambda_t \\
&= \frac{1}{2}\Big(\alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \alpha_t^2 \mathbf{z}_t \Big) \mathrm{d}\lambda_t.
\end{aligned}$$

In other words,

$$\frac{\mathrm{d}\mathbf{z}_t}{\mathrm{d}\lambda_t} = \frac{1}{2}\Big(\alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \alpha_t^2 \mathbf{z}_t \Big).$$

- Let's take a brief detour. We know that

$$\begin{aligned}
\frac{\mathrm{d}\alpha_t^2}{\mathrm{d}\lambda_t} &= 2\alpha_t \frac{\mathrm{d}\alpha_t}{\mathrm{d}\lambda_t} \\
\frac{\mathrm{d}\,\mathrm{sigmoid}(\lambda_t)}{\mathrm{d}\lambda_t} &= 2\alpha_t \frac{\mathrm{d}\alpha_t}{\mathrm{d}\lambda_t} \\
\mathrm{sigmoid}(\lambda_t)(1 - \mathrm{sigmoid}(\lambda_t)) &= 2\alpha_t \frac{\mathrm{d}\alpha_t}{\mathrm{d}\lambda_t} \\
\alpha_t^2 \sigma_t^2 &= 2\alpha_t \frac{\mathrm{d}\alpha_t}{\lambda_t} \\
\frac{\mathrm{d}\alpha_t}{\mathrm{d}\lambda_t} &= \frac{1}{2}\alpha_t \sigma_t^2.
\end{aligned}$$

Similarly, one can show that

$$\frac{\mathrm{d}\sigma_t}{\mathrm{d}\lambda_t} = -\frac{1}{2}\sigma_t \alpha_t^2.$$

- Now, let us get back to the DDIM update rule.

$$\mathbf{z}_s = \sigma_s \frac{\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)}{\sigma_t} + \alpha_s \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t).$$

Differentiating both sides with respect to λ_s, we have that

$$\begin{aligned}
\frac{\mathrm{d}\mathbf{z}_s}{\mathrm{d}\lambda_s} &= \frac{\mathrm{d}\sigma_s}{\mathrm{d}\lambda_s} \frac{\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)}{\sigma_t} + \frac{\mathrm{d}\alpha_s}{\mathrm{d}\lambda_s} \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) \\
&= -\frac{1}{2}\sigma_s \alpha_s^2 \frac{\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)}{\sigma_t} + \frac{1}{2}\alpha_s \sigma_s^2 \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t).
\end{aligned}$$

Evaluating the derivative at $s = t$, we have

$$\begin{aligned}
\left.\frac{\mathrm{d}\mathbf{z}_s}{\mathrm{d}\lambda_s}\right|_{s=t} &= -\frac{1}{2}\sigma_t \alpha_t^2 \frac{\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t)}{\sigma_t} + \frac{1}{2}\alpha_t \sigma_t^2 \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) \\
&= \frac{1}{2}\big(\alpha_t(\alpha_t^2 + \sigma_t^2)\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \alpha_t \mathbf{z}_t^2 \big) \\
&= \frac{1}{2}\big(\alpha_t \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t, \lambda_t) - \alpha_t \mathbf{z}_t^2 \big).
\end{aligned}$$

The above result agrees with the formula for $\mathrm{d}\mathbf{z}_t/\mathrm{d}t$ derived from the probability flow ODE. So, one can say that the DDIM an an integration rule for the probabilty flow ODE.

References

[DN21] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 2021.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

[KAAL22] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models, 2022.

[KSPH21] Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models, 2021.

[ND21] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

[SH22] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *CoRR*, abs/2202.00512, 2022.

[SME20] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2020.

[SSDK+21] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.