MATHÉMATIQUES VISION APPRENTISSAGE

école normale supérieure paris-saclay

Denoising score matching for diffusion models

Probabilistic Graphical Models and Deep Generative Models

December 2024

Julien Boudier, Martin Jolif

1 Introduction

We worked on the **Denosing score matching for diffusion models** project. To do so, we studied three articles: "A Connection Between Score Matching and Denoising Autoencoders" [6], "Generative Modeling by Estimating Gradients of the Data Distribution" [5] and "Denoising Diffusion Probabilistic Models" [1]. This allowed us to better understand the mathematical background behind diffusion models. We focused specifically on the third paper and developed a PyTorch implementation of the diffusion model it proposes (available on GitHub) for image generation.

Denoising score matching is a method used in diffusion models to learn the score of the data distribution. In this approach, noise is gradually added to the data, and a neural network is trained to predict how to remove this noise to recover the original data. By learning how to reverse the noise process, the model can create new samples by starting with random noise and removing it step by step, making it useful for tasks like image generation.

2 Origin of Denoising score matching for diffusion models

In this section, we will provide a brief historical overview to explain the origins of diffusion models and their connection to denoising score matching.

2.1 Explicit Score Matching

Explicit Score Matching (ESM) was introduced by Hyvärinen [2] as a technique to learn the parameters θ of probability density models $p_\theta(x)$ of the form $\frac{1}{Z(\theta)}e^{-E(x,\theta)}$ such that the score of $p_\theta(x)$ distribution best matches the corresponding score of the true distribution $q_{data}(x)$, where E is called the energy function and the score refers to the derivative of the log likelihood with respect to the data: $\frac{\partial \log p_\theta(x)}{\partial x}$. The corresponding objective function to be minimized is the expected squared error between these two vectors:

$$J_{ESM_{q_{data}}}(\theta) = \mathbb{E}_{q_{data}(x)}\left(\frac{1}{2}\left\|\frac{\partial \log p_\theta(x)}{\partial x} - \frac{\partial \log q_{data}(x)}{\partial x}\right\|^2\right) \tag{1}$$

However, since q_{data} is unknown, we do not have explicit regression targets for $\frac{\partial \log q_{data}(x)}{\partial x}$. This formulate the goal of estimating the true data distribution by a distribution $p_\theta(x)$ of the form $\frac{1}{Z(\theta)}e^{-E(x,\theta)}$ (so it could be a gaussian distribution, choice that will be done for diffusion models). However, it's impossible to train since the true distribution is unknown.

2.2 Denoising Score Matching

To address this problem, the first paper "A Connection Between Score Matching and Denoising Autoencoders" [6] considers a slightly different objective that is inspired by both Explicit Score Matching (ESM) and Denoising Autoencoder (a Denoising Autoencoder is a modification of a classical autoencoder that is trained, not to reconstruct it's input, but to denoise a noisy version of it's input).

Let $\tilde{x} = x + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2 I_d)$ the corrupted version of a training input x. They define in this paper [6], the following Denoising Score Matching (DSM) objective:

$$J_{DSM_{q_\sigma}}(\theta) = \mathbb{E}_{q_\sigma(x,\tilde{x})}\left(\frac{1}{2}\left\|\frac{\partial \log p_\theta(\tilde{x})}{\partial \tilde{x}} - \frac{\partial \log q_\sigma(\tilde{x}|x)}{\partial x}\right\|^2\right) \tag{2}$$

where in this case of an isotropic Gaussian of variance σ^2 noise model, $q_\sigma(\tilde{x}|x) = \frac{1}{(2\pi)^{d/2}\sigma^d}e^{-\frac{1}{2\sigma^2}\|\tilde{x}-x\|^2} = \mathcal{N}(\tilde{x}; x, \sigma^2 I_d)$ and $q_\sigma(x, \tilde{x}) = q_\sigma(\tilde{x}|x)q_{data}(x)$.

The first paper [6] provides a proof that $J_{DSM_{q_\sigma}}(\theta)$ and $J_{ESM_{q_\sigma(\tilde{x})}}(\theta)$ are equivalent optimization objectives.

2.3 Denoising Score Matching for image Generation

The second paper, "Generative Modeling by Estimating Gradients of the Data Distribution" [5] introduces a new generative model where samples are produced via Langevin dynamics using gradients of

the data distribution estimated with denoising score matching (DSM) as explained in the last section. However this approach faces two main challenges:

- Low-dimensional manifold issue: When data lies on a low-dimensional manifold, the score (gradient of the log-density) becomes undefined in the ambient space, making score matching inconsistent.
- Scarcity in low-density regions: In regions with few data points (far from the manifold), score estimation becomes inaccurate, which negatively impacts Langevin dynamics sampling. Poor score estimates slow down the mixing process, as Langevin dynamics often starts in these low-density areas, and transitioning between distribution modes requires moving through these regions.

To address these issues, the second paper [5] proposes to perturb the data with random Gaussian noise of various magnitudes and to estimate simultaneously scores corresponding to all noise levels by training a single denoising network using equation (2). Let $\{\sigma_i\}_{i=1}^L$ be a positive geometric sequence that satisfies $\frac{\sigma_1}{\sigma_2} = \cdots = \frac{\sigma_{L-1}}{\sigma_L} > 1$. Let $q_\sigma(\tilde{x}) = \int q_{\text{data}}(t) q_\sigma(t|x) dt$ (where $q_\sigma(t|x) = \mathcal{N}(x; t, \sigma^2 I)$) denote the perturbed data distribution. The denoising score matching objective is the following (similar as equation 2):

$$l(\theta, \sigma) = \mathbb{E}_{q_{data}(x)} \left[\mathbb{E}_{q_\sigma(x,\tilde{x})} \left(\frac{1}{2} \left\| s_\theta(\tilde{x}, \sigma) - \frac{\partial \log q_\sigma(\tilde{x}|x)}{\partial x} \right\|^2 \right) \right] \tag{3}$$

where $s_\theta(x, \sigma)$ is the denoising network called Noise Conditional Score Network (NCSN).
Then, they combine equation (3) for all $\sigma \in \{\sigma_i\}_{i=1}^L$ to get a unified objective:

$$\mathcal{L}(\theta, \{\sigma_i\}_{i=1}^L) = \frac{1}{L} \sum_{i=1}^{L} \lambda(\sigma_i) l(\theta, \sigma_i) \tag{4}$$

where $\lambda(\sigma_i) > 0$ is a function of σ_i.

Once the denoising network is trained, during the inference, the samples are generated via annealed Langevin dynamics:

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^L, \epsilon, T$.
1: Initialize $\tilde{x}_0$
2: **for** $i \leftarrow 1$ to L **do**
3: $\quad \alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
4: $\quad$ **for** $t \leftarrow 1$ to T **do**
5: $\quad\quad$ Draw $z_t \sim \mathcal{N}(0, I)$
6: $\quad\quad \tilde{x}_t \leftarrow \tilde{x}_{t-1} + \frac{\alpha_i}{2} s_\theta(\tilde{x}_{t-1}, \sigma_i) + \sqrt{\alpha_i} z_t$
7: $\quad$ **end for**
8: $\quad \tilde{x}_0 \leftarrow \tilde{x}_T$
9: **end forreturn** $\tilde{x}_T$

Where ϵ, T are two parameters from the Langevin dynamics that should be respectively close to zero and as large as possible ($+\infty$). This algorithm uses the final sample from Langevin dynamics $q_{\sigma_{i-1}}(x)$ as the starting sample for $q_{\sigma_i}(x)$. In the final step, Langevin dynamics is used to sample from $q_{\sigma_L}(x)$, which closely approximates $q_{\text{data}}(x)$ as σ_L approaches zero.

3 Denoising Diffusion Probabilistic Models [1]

3.1 General Idea

Some simple explanation before diving into the mathematical explanation. To generate new images following the DDPM model [1] there are two main phases. The first one is the training phase, for all training images, we will progressively add noise to them ($q(x_t|x_{t-1})$ in figure 1), at the final step T, we suppose that our training images follow an isotropic gaussian distribution. The goal is to predict the noise added during this noising process to be able to go back to the original image from the noisy one. To do so, we train a denoising neural network ($p_\theta(x_{t-1}|x_t)$ in figure 1) to predict this noise by minimizing the squared difference between the real noise and the predicted noise. Once this denoising neural network is trained, we start by sampling one image at time step T (a sample from an isotropic gaussian distibution) and we pass this sample through the denoising neural network, which will predict some noise, that we will "remove" from this sample: this will generate a new image that follows the

probability distribution of the training set images. With all of this in mind, we can go through the mathematical explanation.

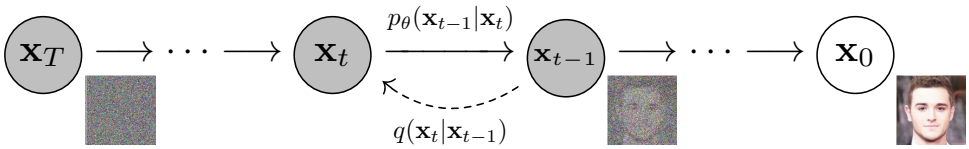

Figure 1: Directed graphical model considered in [1]

3.2 Diffusion Process

In this section, the notation are a little bit different than the ones used in section 2. Here, the true distribution $q_{data}(x)$ is noted $q(x_0)$. Moreover, here the time steps are in the opposite direction compared to the notations of Algorithm 1 (line 4 to 6): in this section, to denoise the samples we go from time step t to time step $t-1$ (see figure 1).

As before, we suppose that for a big time step T, $p(x_T) = \mathcal{N}(0, I)$. However, this time, the authors propose another diffusion process as the one proposed in [5]:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I_d) \tag{5}$$

where the forward process variances β_t can be learned by reparametrization or held constant as hyperparameters. Therefore,

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t) I_d) \tag{6}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$

3.3 Training Objective

The goal remains the same, learning the reverse process to be able to generate a sample from the original distribution given a noisy sample. We want to find the optimal parameters θ such that:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{7}$$

Training can be performed by optimizing the usual variational bound on negative log likelihood which can be rewritten as:

$$\mathbb{E}_q \left(D_{KL}\left[q(x_T|x_0)||p(x_T)\right] + \sum_{t>1} D_{KL}\left[q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)\right] - \log p_\theta(x_0|x_1) \right) \tag{8}$$

See [1] for the proof. Where,

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I) \tag{9}$$

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t \text{ and } \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{10}$$

First, the authors set $\Sigma_\theta(x_t, t) = \sigma_t^2 I$, where $\sigma_t^2 = \frac{(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\beta_t$, which is an optimal choice for x_0 deterministically set to one point.

Therefore by denoting, $L_{t-1} = \mathbb{E}_q \left(D_{KL}\left[q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)\right]\right)$, using the fact that the Kullback-Leibler divergence of two normal distributions is known, we obtain:

$$L_{t-1} = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t) \right\|^2 \right] + C \tag{11}$$

with C a constant independent of theta.

Then using a reparametrization of equation (6): $x_t(x_0, \epsilon) = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ for $\epsilon \sim \mathcal{N}(0, I)$, we can obtain:

$$L_{t-1} = \mathbb{E}_{x_0,\epsilon}\left[\frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(x_t(x_0,\epsilon) - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) - \mu_\theta(x_t(x_0,\epsilon),t)\right\|^2\right] + C \tag{12}$$

Equation (12) reveals that, we may choose the following parametrization for μ_θ:

$$\mu_\theta(x_t,t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)\right) \tag{13}$$

where ϵ_θ is a function approximator intended to predict the noise ϵ from x_t. In our case, with an image dataset, the authors designed a U-Net architecture with Transformer sinusoidal position embedding to process time as well as the input image. Furthermore, with the last parametrization from equation (13), equation (12) can be simplified as follows:

$$\mathbb{E}_{x_0,\epsilon}\left(\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\alpha_t)}\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2\right) \tag{14}$$

However, the authors decided to use the following simplified loss function term to simplify the implementation and after finding it beneficial (in term of FID see equation 16) :

$$\mathbb{E}_{t,x_0,\epsilon}\left(\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2\right) \tag{15}$$

This comes to the following training process:

Algorithm 2 Training

repeat
  $x_0 \sim q(x_0)$
  $t \sim \text{Uniform}(\{1,\dots,T\})$
  $\epsilon \sim \mathcal{N}(0,I)$
  Take gradient descent step on
    $\nabla_\theta \left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2$
until converged

3.4 Sampling process

For the sampling process, once we have trained our network, we know the parameters θ and $\mu_\theta(x_t,t)$, $\Sigma_\theta(x_t,t)$, we can sample x_{t-1} given x_t, with equation (7). The full sampling process is described in the following algorithm:

Algorithm 3 Sampling

$x_T \sim \mathcal{N}(0,I)$
for $t = T,\dots,1$ **do**
  $z \sim \mathcal{N}(0,I)$ if $t > 1$, else $z = 0$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)\right) + \sigma_t z$
end for
return x_0

3.5 Images Generation

To test their diffusion model on real data, the authors decided to generate images, to do that, they used the CIFAR-10 dataset as training data. To be able to have a quantitative idea of the results of their model, they used a metric called Fréchet-Inception-Distance (FID), which compares the mean and standard deviation of the deepest layer in Inceptionv3 model of the training data and generated data distributions as follows:

$$d_F(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\mu',\Sigma'))^2 = \|\mu - \mu'\|_2^2 + \text{tr}\left(\Sigma + \Sigma' - 2\left(\Sigma\Sigma'\right)^{\frac{1}{2}}\right) \tag{16}$$

Therefore, a lower FID score indicates better performance.

4 Experiments on Beta Schedules

First, to be able to experience the theoretical results seen in this article, we used a GitHub repository trained on 10 000 images from the CelebA dataset (resized to 64*64) to generate the faces from the cover page.

Then, to go deeper in the implementation, we decided to create a GitHub repository in which we provide a PyTorch implementation of the Diffusion model proposed in [1] based on a notebook [4]. In this repository, you can find a README file explaining the steps to train and sample the model on the fashion-mnist dataset. The repository is structured such that you can easily change/adapt a single part of the global architecture.

In the original paper [1], the authors used a linear schedule for the noise variances β_t in the forward diffusion process. However, other works, such as "Improved Denoising Diffusion Probabilistic Models" [3], have shown that a cosine schedule can lead to better results. This motivated us to explore the impact of different β_t schedules on the performance of DDPMs.

To do our experiment, we trained the DDPM diffusion model using our implementation with different existing beta schedulers and tried to propose some new schedulers. Then we tried to compare the results of the models trained with these different schedulers qualitatively by looking at the generated samples. We also tried to compare them quantitatively by using the FID distance metric, unfortunately, we faced a memory issue, and we were not able to compute it.

Here are the different schedulers that we tested:

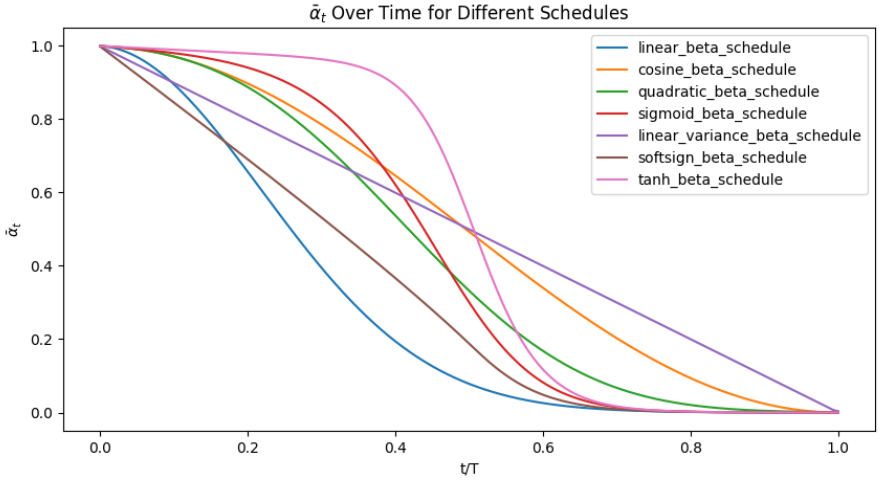

Figure 2: Evolution of $\bar{\alpha}_t$

The first schedule we proposed is the "linear variance beta schedule", the goal is that $\bar{\alpha}_t$ be a linear function of time steps t and decrease from 1 to 0. To obtain the corresponding β_t function we do a computation that you can find in Appendix.

The two others (soft sign and tanh beta schedule) are inspired on the sigmoid beta schedule as a rescale of the $Softsign$ and $tanh$ function are more or less similar to the sigmoid function. However these two schedulers might not be optimal as the values are close to zero almost 30% of the time.

Figure 2 visualizes the evolution of $\bar{\alpha}_t$ across different schedules. We observe significant differences in their behavior. Notably, the linear schedule proposed in the original paper exhibits a rapid decrease in $\bar{\alpha}_t$, approaching zero for nearly a third of the diffusion process. This suggests that a substantial portion of the forward process might be spent on excessively noisy images, potentially hindering the model's ability to learn meaningful information during these steps. In contrast, the cosine schedule and our proposed linear variance decrease schedule maintain a more gradual decrease in $\bar{\alpha}_t$. As shown in figure 3, this slower decrease in $\bar{\alpha}_t$ translates to images with less noise at the same time step t during the forward process.

Figure 4 presents generated samples obtained with different schedules and varying hyperparameters. We made several key observations:

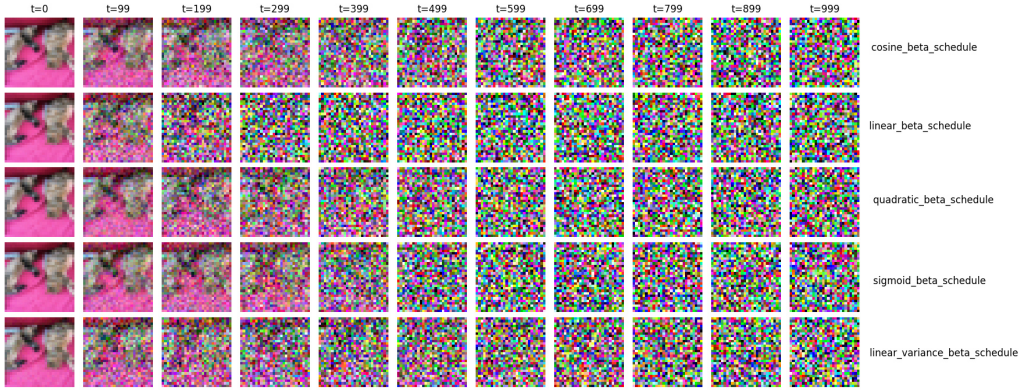

Figure 3: Forward process with different schedules

- **Sensitivity to Hyperparameters:** The performance of each schedule is highly dependent on the chosen hyperparameters, particularly the number of training epochs and diffusion timesteps T. Some schedules fail to produce satisfactory results under certain hyperparameter combinations.
- **Overfitting and Training Epochs:** Certain schedules, like our linear variance schedule, appear more prone to overfitting or require fewer epochs to reach satisfactory results. The linear variance decrease schedule seems to require fewer epochs than the original linear schedule to converge. For instance, we observed good results with just 2 epochs of training.
- **Timestep Requirements:** The linear variance decrease schedule seems to require fewer timesteps than the original linear schedule to produce high-quality samples. This is consistent with the observation that the original linear schedule spends a significant portion of the process with excessively noisy images. For example, when generating with $T = 500$ on a model trained with $T = 1000$, the linear beta schedule seems to still be able to produce great results.

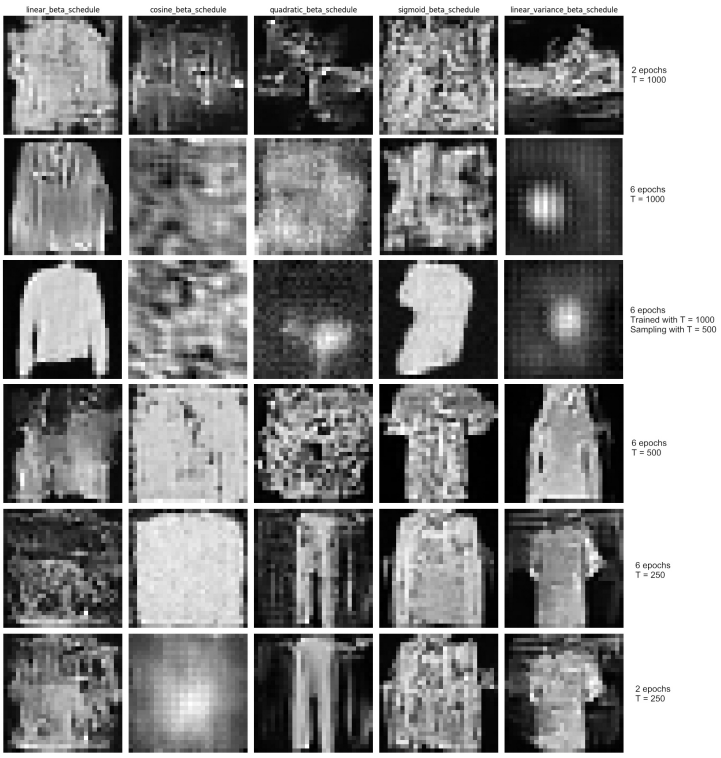

Figure 4: Generated samples on Fashion MNIST

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[2] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 01 2005.

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

[4] Niels Rogge and Kashif Rasul. The annotated diffusion model.

[5] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2020.

[6] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.

A Derivation of Linear Variance Decrease Schedule and Implementation Details

This appendix provides details on the derivation of the linear variance decrease β_t schedule and other implementation specifics.

We aimed to design a β_t schedule that ensures a linear decrease in the variance of the signal component ($\sqrt{\bar{\alpha}_t}x_0$) over the diffusion process. The variance can be expressed as:

$$Var(\sqrt{\bar{\alpha}_t}x_0) = \bar{\alpha}_t Var(x_0)$$

Assuming $Var(x_0) = 1$ for simplicity, we want:

$$\bar{\alpha}_t = C - ct$$

where C and c are constants. To determine C and c, we impose the following constraints:

- $0 \leq \bar{\alpha}_t \leq 1$ for all $t \in \{1, ..., T\}$
- $\bar{\alpha}_0 = 1$ (initial signal is unchanged)

From $\bar{\alpha}_0 = 1$, we get $C = 1$. From $0 \leq \bar{\alpha}_T$, we get $c \leq \frac{1}{T}$. Choosing $c = \frac{1}{T}$ to achieve the maximum linear decrease, we have:

$$\bar{\alpha}_t = 1 - \frac{t}{T}$$

Now, we can compute α_t:

$$\alpha_t = \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}} = \frac{1 - \frac{t}{T}}{1 - \frac{t-1}{T}} = \frac{T - t}{T - t + 1}$$

Finally, we obtain β_t:

$$\beta_t = 1 - \alpha_t = 1 - \frac{T - t}{T - t + 1} = \frac{1}{T - t + 1}$$

Therefore, the linear variance decrease schedule is given by:

$$\beta_t = \frac{1}{T - t + 1}$$

To ensure numerical stability and prevent issues arising from β_t values extremely close to 0 or 1, we clip the computed β_t values to the range $[0.0001, 0.9999]$ in all our experiments similarly as the cosine beta schedule.

B Contributions statement

- Martin Jolif: based on the following two GitHub repositories created a notebook to details the DDPM model as it is presented in the paper and run it on the CelebA dataset (official repository, PyTorch Implementation)
- Julien Boudier: created the Github repository based on a notebook [4]

We decided to use Julien's GitHub to do our experiments on beta schedulers as it was more adapted to add new schedulers.