**M2 IASD - PSL University**

Report - Data Science Project

# Diffusion Probabilistic Models

**no-name-idea-fusion-Team**
Adrien Golebiewski
Oumaïma Bendramez
Yann Trémenbert

November 27th, 2022

## 1 Introduction

Diffusion models were inspired by non-equilibrium thermodynamics. Let's take the example of a drop of ink in a glass of water. By the laws of physics, the ink will diffuse and spread until it reaches an equilibrium. In physics this phenomenon is irreversible. In Machine Learning, we can consider that the drop of ink in the water is the initial image. And as the diffusion process goes on, the percentage of information from the initial image that we keep decreases until the image is completely noised. In Machine Learning, we can learn to go from a completely noised image to a clear one.

## 2 Diffusion Models : The theory

Diffusion models rely on a Markov chain because the noise distribution at every time step depends on the previous step. We gradually add a small amount of gaussian noise to the original image at every time step until we reach a normal distribution of noise $\mathcal{N}(0,1)$. That is called the forward process. A nice property of diffusion models is that the latent space have the same dimentionnality as the input.

### 2.1 Forward and Reverse processes

$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$ describes the way that noise is sampled which corresponds to a Gaussian distribution with a mean that depends on the previous image and a fixed variance. We can not easily estimate $q(x_{t-1}|x_t)$. Therefore, learning the reverse process is achieved by a parameterized Neural network by approximating the model distribution :

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

### 2.2 Forward processes and variance schedule

More precisely, in the forward process we add noise progressively according to a variance schedule of betas. We can compute the image at a given time step t using the following formula : $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ where $\epsilon \sim \mathcal{N}(0, I)$.
Basically, $\bar{\alpha}_t$ translates "how much information we keep from the original image". As t tends to T, $\bar{\alpha_T} = 0$ meaning that we end up with pure Gaussian noise as illustrated in figure 2.
In the following plot, we plotted $\bar{\alpha}_t$ according to the time step in order to compare the different variance schedules.

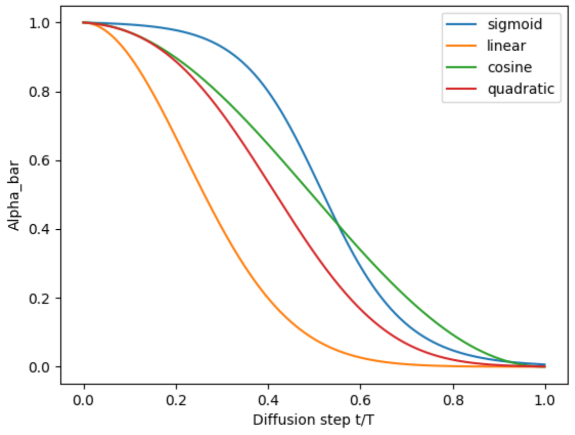

Figure 1: Cumulative product of the alphas according to t/T with alpha = 1-beta

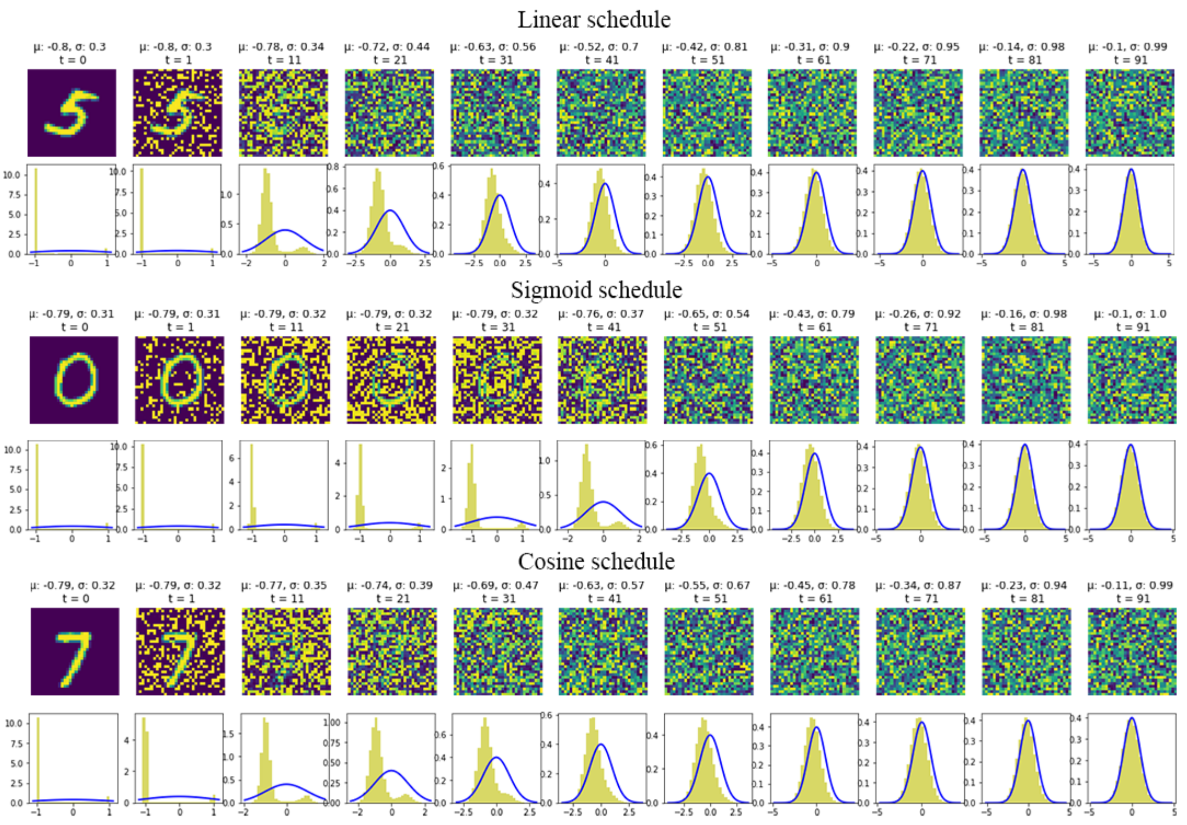

Figure 2: Forward images samples and their associated pixel value distributions for T=100 (gaussian distribution is represented by the blue curve)

The papers precise that the most effective noise scheduler is cosine because ideally we want a linear relation between $\bar{\alpha}_t$ and $t/T$.

### 2.3 Reverse process : The Training

We model the reverse diffusion process as a Markov chain that converts the noise distribution into the target distribution in order to generate a clear image. We use a Neural Network to do the reverse process.

The parameters of the network are optimized in a way to maximize $p_\theta(x_0)$ which can be seen as the probability of generating a realistic sample with the whole denoising process:

$$\theta^* = \arg\max_\theta \sum_{i=1}^n logp_\theta(x_0^{(i)})$$

Introducing the Kullback-Leibler divergence, the *Denoising Diffusion Probabilistic Models (DDPM)* paper's authors derive a simpler loss defined as follows:

$$L_{simple}(\theta) := E_{t,x_0,\epsilon}[\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2]$$

calculating the difference between the true added noise $\epsilon$ and the predicted noise $\epsilon_\theta$. This is the loss we use for our experiments.

## 3 A neural architecture approach: the Unet

A neural network is trained to predict this noise based on the corrupted image $x_t$ (i.e. noise applied on x0x0 based on known schedule $\beta t$) The neural network needs to take in a noised image at a particular time step and return the predicted noise. Note that the predicted noise is a tensor that has the same size/resolution as the input image. So technically, the network takes in and outputs tensors of the same shape.

In terms of architecture, the Diffusion Model authors went for a U-Net, a u-shaped neural network.
Like any autoencoder, this network consists of a bottleneck in the middle that makes sure the network learns only the most important information. The encoder block has a constant reduction of image size with the help of the max-pooling layers. We also have repeated convolutional layers with an increasing number of filters in the encoder architecture. The encoder first encodes an image into a smaller hidden representation called a "bottleneck". Then, the decoder decodes that hidden representation back into an actual image. We notice the number of filters in the convolutional layers start to decrease along with a gradual upsampling in the following layers all the way to the top.

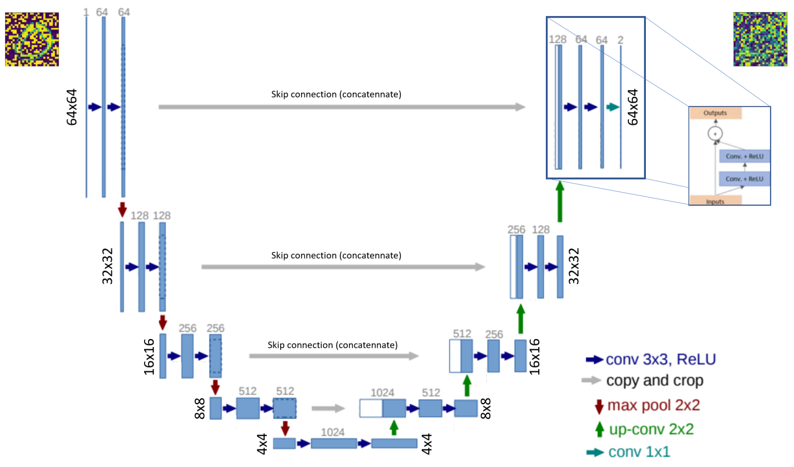

Figure 3: Simple Unet architecture

With a low-level spatial information is exchanged between the input and output, it would be preferable to transfer this information directly across the network. Concatenation of prior feature maps is another method for achieving skip connections. These concatenations are realized by “skip connections” and favors the work of reconstruction of the image noise.
Finally, as the parameters of the neural network are shared across time (noise level), position embeddings encode the timestep t, inspired by the Transformer architecture. This makes the neural network "know" at which particular time step (noise level) it is operating, for every image in a batch.

## 4 Implementation and results comparative

In this work, we want to focus on the variance schedule and infer strategies on how to optimize it. Variance is introduced in the forward process through a list of $\beta_t$ as defined in section 2.1 in order to infer the noisier image $x_{t+1}$ from the previous one $x_t$. We define $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$.

As illustrated on figure 1, the slope of the list of $\bar{\alpha}_t$ corresponds to the speed at which we lose information from the previous images. Depending on where are located the smooth or abrupt slopes, we will force our algorithm to learn different denoising strategies. Furthermore, one more interesting hyperparameter is the number of steps $T$ necessary to denoise a random gaussian noise. We set $T$ to either 50, 100, 200, 400 or 800 and compare its consequences on the training phase and on the quality of the samplings. Here, we decide to compare three different schedulers: linear, cosine, and sigmoid, defined as follows:

```python
s = 0.002 # smallest value which adds noise for cosine schedule at first step for T=800 (otherwise no noise is added and x_1 is the same as x_0...)
beta_min = 1e-5 # smallest value which adds noise for linear and sigmoid schedules at first step (otherwise no noise is added and x_1 is the same as x_0...)
beta_max = 0.01 # smallest value to give almost gaussian noise at the end of the forwars process
start = (beta_min*1000)/T
end = (beta_max*1000)/T # That way, we keep the same shape of curve alphas_cumprod for any T

def linear_beta_schedule(timesteps, start, end):
    return torch.linspace(start, end, timesteps)

def sigmoid_beta_schedule(timesteps, start, end):
    betas = torch.linspace(-10, 10, timesteps)
    return torch.sigmoid(betas) * (end - start) + start

def cosine_beta_schedule(timesteps, s):
    steps = timesteps + 1
    x = torch.linspace(0, timesteps, steps)
    alphas_cumprod = torch.cos(((x / timesteps) + s) / (1 + s) * torch.pi * 0.5) ** 2
    alphas_cumprod = alphas_cumprod / alphas_cumprod[0]
    betas = 1 - (alphas_cumprod[1:] / alphas_cumprod[:-1])
    return torch.clip(betas, 0.00001, 0.9999)
```

As a consequence, we obtain 15 different models and compare their learning curves over 5 epochs of training. Those can be seen in figure 4. Concerning the loss minimization, almost all models perform similarly one to another, independently of the the number of time steps $T$. We observe that the sigmoid schedule is slightly outperformed by the other two schedules.

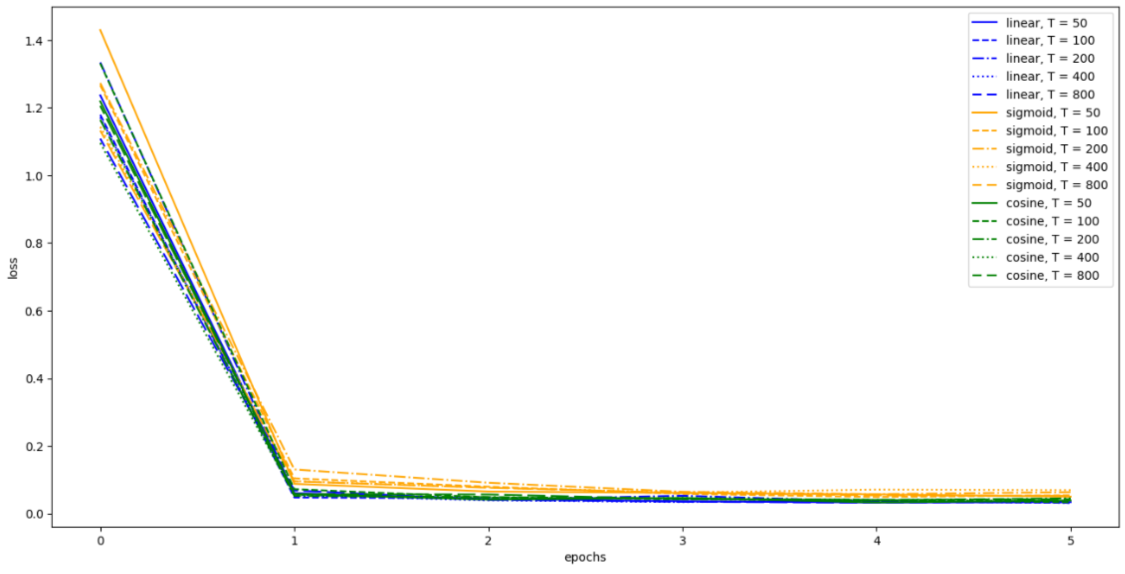

Figure 4: Loss as a function of epochs for the 3 schedulers (linear, sigmoid, cosine) and 5 number of time steps (50, 100, 200, 400, 800)

For each of these models, we generate 32 sample images to evaluate correlation between the loss value and the quality of the sampling process. We find that the loss value is not representative of the quality of such sampling as all the models produce very different result images while obtaining almost the same loss value. We showed some of the produced images in figure 5.

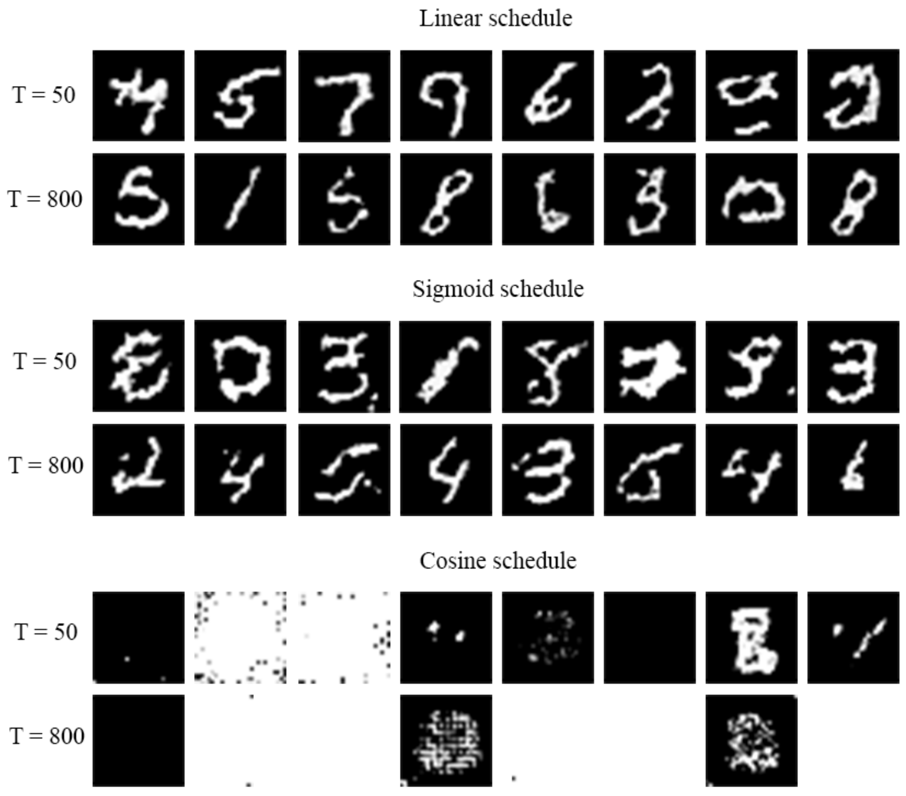

Figure 5: Generation samples for different models

The first point to underline is that even though the cosine schedule models usually have smaller loss values than sigmoid schedule models, they tend to produce worse images. Hence, rather than choosing a model based on the loss value, we advise choosing a DDPM model based on a distance to a dataset (MNIST here), like the Frechet Inception Distance (FID) such as used in the *Denoising Diffusion Implicit Models* paper.

Moreover, the fact that the cosine schedule performs worse than the other two may be explained by its (almost) linear slope. Meanwhile many papers argue that this slope allows a smooth transition from clear to noised image, we argue here that it requires a large number of timesteps in order to be effective. Most of the papers that have chosen a cosine schedule chose $T = 4000$, allowing a small slope during the whole forward and reverse processes. But using either a linear or a sigmoid schedule with smaller $T$ also allows to have a small slope for a large amount of timesteps, which could explain the result.

The second point is that the number of timesteps $T$ has only a small effect on the quality of the produced images. One way to further study this phenomenon would be to sample a lot of images for each $T$, use the FID as a quality criteria and observe a trend in the function $FID = f(T)$.

## 5 Conclusion

Given the time constraints, we were not able to explore the others papers neatly, however we have very good understanding of the Data Generation problem, and more specifically the Diffusion Model particularities.

Due this same lack of time and compute power, we could not perform the analysis about the number of timesteps $T$.

Nevertheless, we highlighted the significant importance of the variance scheduler and argue that a well chosen schedule allows to decrease the number of timesteps $T$. This is of paramount importance for generating new images in shorter amounts of time for the sampling part.

Moreover, through this scientific study, we have tackled a new architecture, the U-Net, which is a conventional network architecture for Diffusion Model networks. Although our generation results obtained are quite suitable, the network architecture and our general approach can be further complicated.
In the state of the art, the existence of U-Transformer with attention module between the convolutional blocks, highlights only the relevant activations during training and offer better results. For an extensive explanation of the attention mechanism, we refer the reader to the bibliography.

## References

[1] A POINT CLOUD GENERATIVE MODEL BASED ON NON EQUILIBRIUM THERMODYNAMICS [Anonymous authors] https://openreview.net/pdf?id=O1GEH9X8848

[2] Ho, J., Jain, A., Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems, 33, 6840-6851.

[3] Song, J., Meng, C., Ermon, S. (2020). Denoising diffusion implicit models. arXiv preprint arXiv:2010.02502.

[4] Nichol, A. Q., Dhariwal, P. (2021, July). Improved denoising diffusion probabilistic models. In International Conference on Machine Learning (pp. 8162-8171). PMLR.

[5] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems, 30.

[6] Niels Rogge, Kashif Rasul, Huggingface Notebook, https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/annotated_diffusion.ipynbscrollTo=3a159023