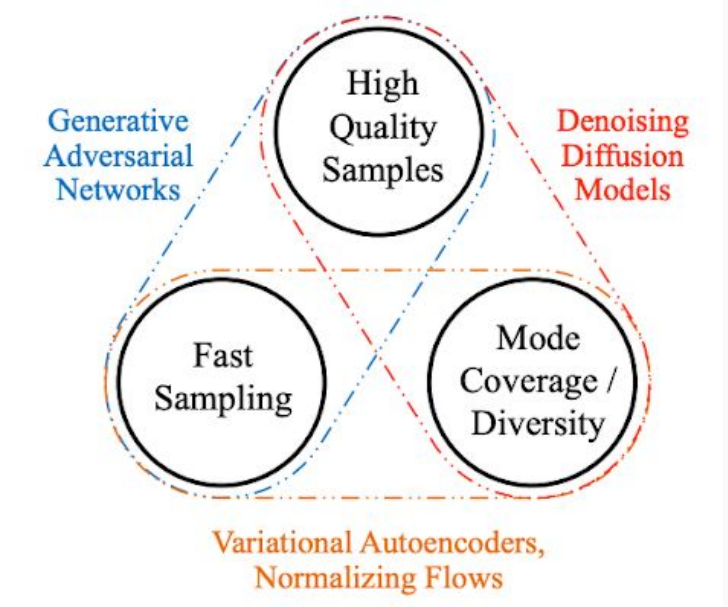
Diffusion models

A tryout project by Alexander Chernyavskiy

Project repository

Motivation

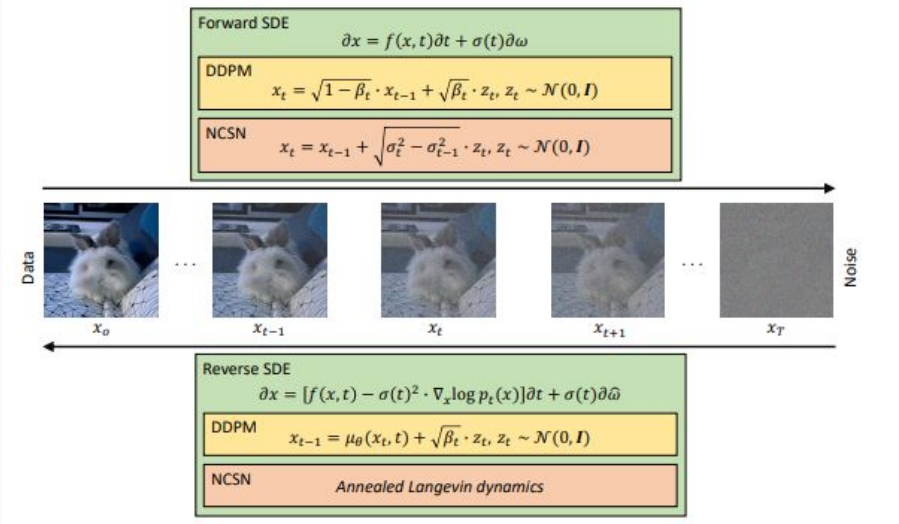
1. Derive for myself DDPMs from the first principles – not only from variational perspectives and form SDEs and stochastic processes
2. Code a minimal example using simple datasets
3. Study how choice of scheduler, its parameters affect generation quality on different datasets



Short reminder about DDPMS

During the lectures ones were derived from variational principles but there exists an equivalent NCSN approach – inspired by Langevin's thermodynamics – and there are some methods to design these models aka solving SDEs/ODEs.

pic source



General algorithmic details

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** $\mathbf{x}_0$

The objective

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

source

Scheduler

Whereas diffusion models usually simply define the forward pass from noise to a less noisy sample, schedulers define the whole denoising process, *i.e.*:

- How many denoising steps?
- Stochastic or deterministic?
- What algorithm to use to find the denoised sample

They can be quite complex and often define a trade-off between denoising speed and denoising quality. It is extremely difficult to measure quantitatively which scheduler works best for a given diffusion pipeline, so it is often recommended to simply try out which works best.

HuggingFace API link

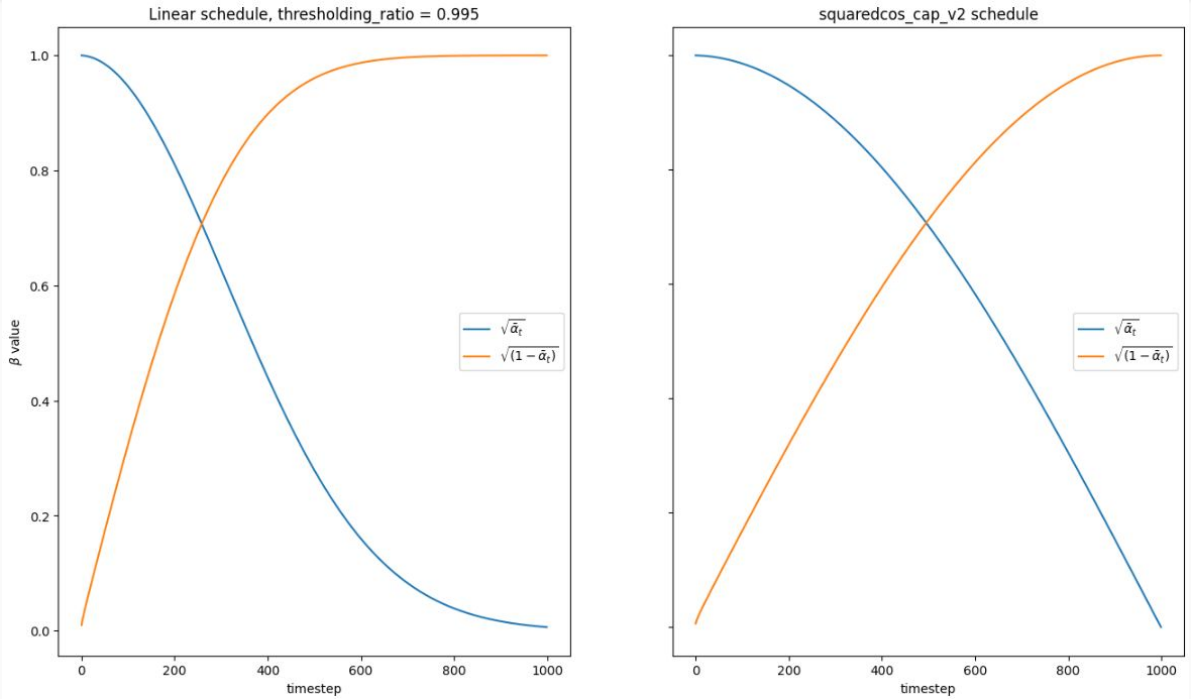
β-schedule

To address this problem, we construct a different noise schedule in terms of $\bar{\alpha}_t$:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2 \tag{17}$$

To go from this definition to variances β_t, we note that $\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$. In practice, we clip β_t to be no larger than 0.999 to prevent singularities at the end of the diffusion process near $t = T$.

Cosine schedule drops more steadily than a linear one



Used schedulers

DDIM schedule – faster sampling[link]

Whereas DDIM performs deterministic sampling, I used this scheduler for sampling in my experiments. During training I tried both to compare efficiency of choosing timesteps stochastically/deterministically

For another approach, let's rewrite $q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ to be parameterized by a desired standard deviation σ_t according to the nice property:

$$
\begin{aligned}
\mathbf{x}_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\boldsymbol{\epsilon}_t + \sigma_t\boldsymbol{\epsilon} \\
&= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}} + \sigma_t\boldsymbol{\epsilon} \\
q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}, \sigma_t^2\mathbf{I})
\end{aligned}
$$

Recall that in $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I})$, therefore we have:

$$
\tilde{\beta}_t = \sigma_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t
$$

Let $\sigma_t^2 = \eta \cdot \tilde{\beta}_t$ such that we can adjust $\eta \in \mathbb{R}^+$ as a hyperparameter to control the sampling stochasticity. The special case of $\eta = 0$ makes the sampling process *deterministic*. Such a model is named the *denoising diffusion implicit model* (**DDIM**; Song et al., 2020). DDIM has the same marginal noise distribution but deterministically maps noise back to the original data samples.

During generation, we only sample a subset of S diffusion steps $\{\tau_1, \dots, \tau_S\}$ and the inference process becomes:

$$
q_{\sigma,\tau}(\mathbf{x}_{\tau_{i-1}}|\mathbf{x}_{\tau_i}, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{\tau_{i-1}}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\mathbf{x}_{\tau_i} - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}, \sigma_t^2\mathbf{I})
$$

PNDM schedule – ODE perspective[link]

Faster sampling via deterministically integrating DDPM as an ODE with distributional initial conditions

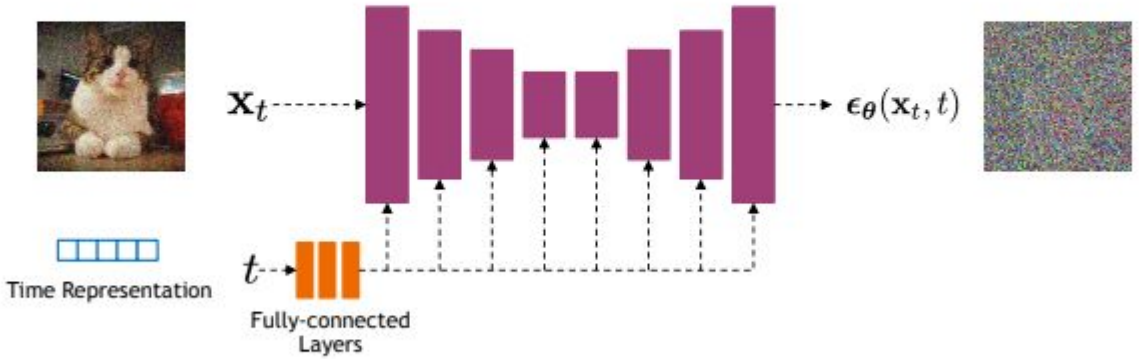
Algorithm 3 Deterministic sampling using general 2nd order Runge–Kutta, $\sigma(t) = t$ and $s(t) = 1$.

procedure ALPHASAMPLER($D_\theta(\boldsymbol{x}; \sigma)$, $t_{i\in\{0,\dots,N\}}$, α)
  sample $\boldsymbol{x}_0 \sim \mathcal{N}(\mathbf{0}, t_0^2\,\mathbf{I})$
  for $i \in \{0, \dots, N-1\}$ **do**
    $h_i \leftarrow t_{i+1} - t_i$ ▷ Step length
    $\boldsymbol{d}_i \leftarrow \big(\boldsymbol{x}_i - D_\theta(\boldsymbol{x}_i; t_i)\big)/t_i$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $(\boldsymbol{x}, t_i)$
    $(\boldsymbol{x}_i', t_i') \leftarrow (\boldsymbol{x}_i + \alpha h \boldsymbol{d}_i, t_i + \alpha h)$ ▷ Additional evaluation point
    if $t_i' \neq 0$ **then**
      $\boldsymbol{d}_i' \leftarrow \big(\boldsymbol{x}_i' - D_\theta(\boldsymbol{x}_i'; t_i')\big)/t_i'$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $(\boldsymbol{x}_i', t_i')$
      $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + h\Big[\Big(1 - \frac{1}{2\alpha}\Big)\boldsymbol{d}_i + \frac{1}{2\alpha}\boldsymbol{d}_i'\Big]$ ▷ Second order step from t_i to t_{i+1}
    else
      $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + h\boldsymbol{d}_i$ ▷ Euler step from t_i to t_{i+1}
  return $\boldsymbol{x}_N$

Implementation Considerations

Network Architectures

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



Time representation: sinusoidal positional embeddings or random Fourier features.

Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see Dharivwal and Nichol NeurIPS 2021)

Classifier-free guidance

w – guidance rate

- w = 0 – fully conditional generation
- large w – unconditional generation

For large rates it is good to use dynamic thresholding to cope with pixel saturation

$$
\begin{aligned}
\nabla_{\mathbf{x}_t} \log p(y|\mathbf{x}_t) &= \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|y) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) \\
&= -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \Big(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \Big) \\
\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t, y) &= \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \sqrt{1-\bar{\alpha}_t}\, w \nabla_{\mathbf{x}_t} \log p(y|\mathbf{x}_t) \\
&= \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) + w\big(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\big) \\
&= (w+1)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - w\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)
\end{aligned}
$$

Metrics

FID (distinguish between synthetic and generated images) and IS (I went for KID (Kernel Inception Distance for more stability instead)) (quality and diversity)

Calculates Fréchet inception distance (FID) which is used to access the quality of generated images. Given by.

$$FID = |\mu - \mu_w| + tr\left(\Sigma + \Sigma_w - 2(\Sigma\Sigma_w)^{\frac{1}{2}}\right)$$

where $\mathcal{N}(\mu, \Sigma)$ is the multivariate normal distribution estimated from Inception v3 (fid ref1) features calculated on real life images and $\mathcal{N}(\mu_w, \Sigma_w)$ is the multivariate normal distribution estimated from Inception v3 features calculated on generated (fake) images. The metric was originally proposed in fid ref1.

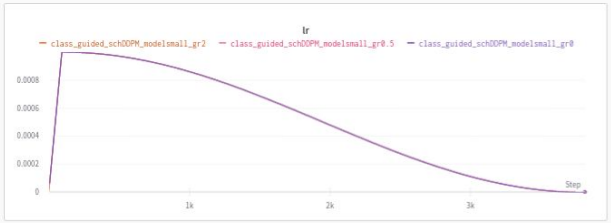
metrics source

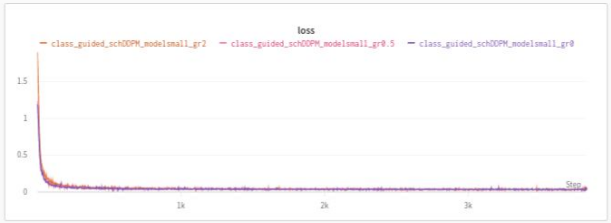
Experiments

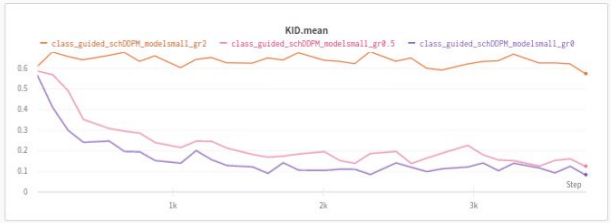
Experimental Setup

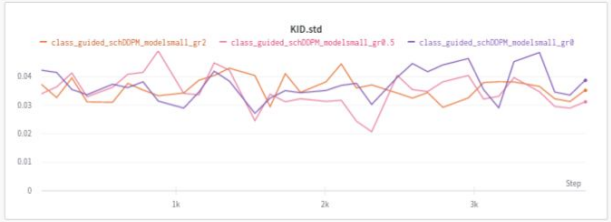
- Two models: **small** (1’780’803 trainable parameters) and **large** (49’558’083 trainable parameters)
- Datasets: **MNIST** (1, 28, 28), **Fashion MNIST** (1, 28, 28), **Flowers102** (3, 128, 128)
- Schedulers: **DDPM, DDIM, PNDM**
- β-schedules: **linear**, **cosine**
- guidance rates: **0** (conditional), **0.5**(mixed), **2**(unconditional)

Guidance





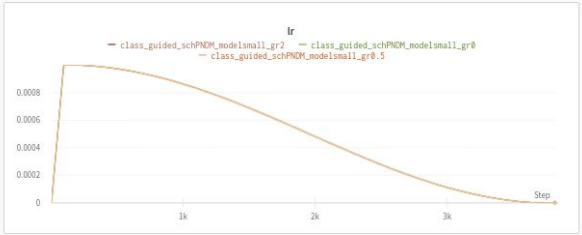


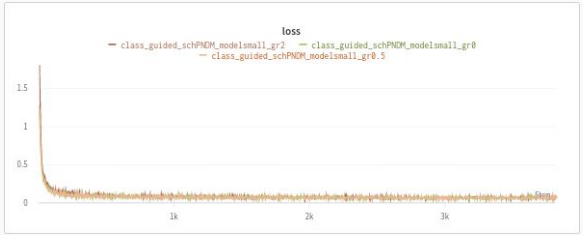


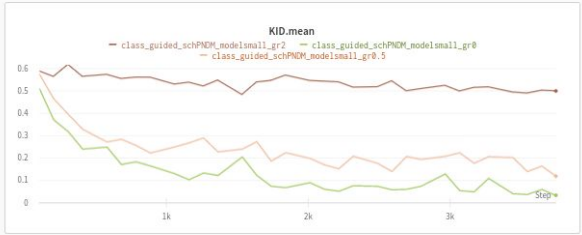
Unconditional guidance is the best to perform in terms of FID and IS, as the guidance rate grows, the results become more hallucinated (I set 100 sampling to make guidance work)

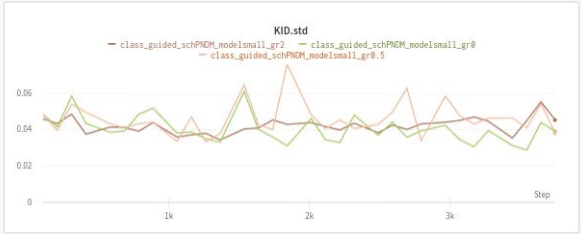
0 0.5 2

For Fashion MNIST results are similar

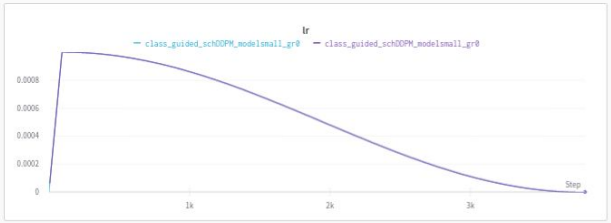


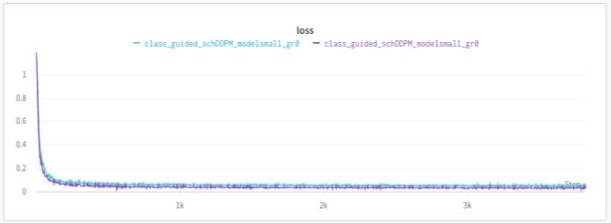


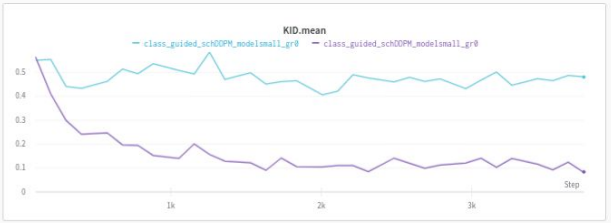


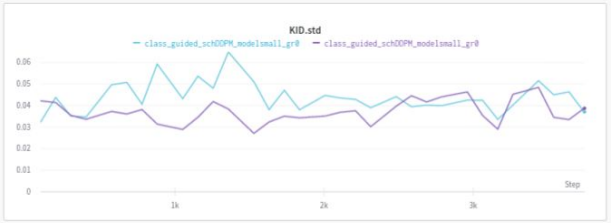


β-schedule type



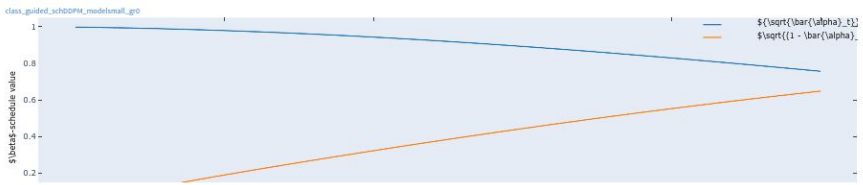


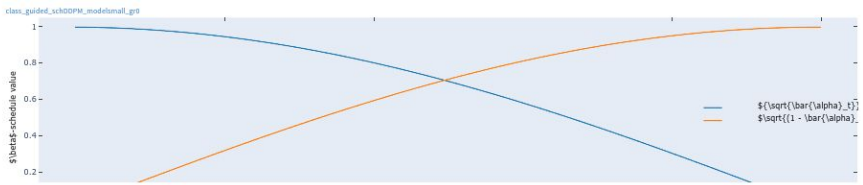




Linear schedule is blue here, as it was shown in DDPM paper, cosine schedule works much better

Sample generations (linear – left)





As expected, on distance cosine schedule is more stable, on Fashion MNIST results pretty much the same

Training scheduler type

For MNIST/Fashion MNIST/Flowers results are identical – choice of a scheduler does not have any quantitative impact but a performance boost during sampling:

DDIM is the fastest, then are methods with ODE solving, and DDPM is the slowest (2x-3x), so I used only DDIM for sampling to accelerate training.

Quantitative results:

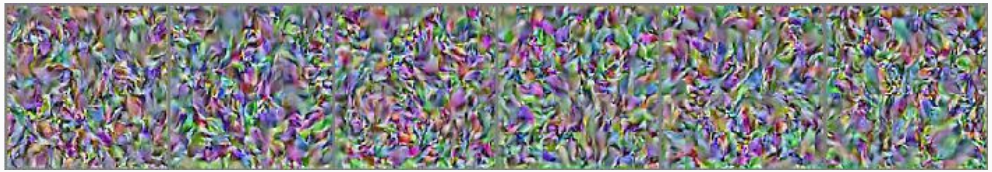
- for MNIST/Fashion MNIST – **training**: around 2 minutes per epoch, **sampling**: around 30 seconds *(DDIM, PNDM)* | around 1.5-2 minutes (*DDPM*) [Tested on Nvidia 1080 Ti]
- For Flowers102: – **training**: around 4 minutes per epoch (same number of training samples, but images 16 times larger), **sampling**: around 70 seconds (*DDIM*, *PNDM*) | around 3-3.5 minutes (*DDPM*) [Tested on free Google Colab]

Performance note

I did not find much success experimenting with RGB images because there is too much GPU overhead during training from scratch (I do not have multi-GPU setup for 1080Ti), it works on Colab but with many performance sacrifices – possibly, my code needs optimisation.

Everything I did is to went through this jupyter – no quantitative results available.

For Flowers dataset everything I got with my resources – these nice hallucinations



Generation quality/performance tradeoff

- On MNIST/Fashion MNIST datasets it was enough around one-two train epochs to generate plausible samples, but if the guidance rate is high enough (unconditional generation), a bit longer training is necessary because training becomes non-stationary due to lack of goal
- For RGB images (fine tuning case), around 5-10 epochs of training is good, and a dataset should be larger by an order of magnitude, with unconditional generation have to train longer, 2-3 times, as I experienced

Additionally: text guidance

I implemented a small text guided model on MNIST

Prompt: “nine”. Text vectorizer – from open-CLIP model and the best settings from previous experiments. Looks are kinda 9 – but, I suppose, model is too small and the dataset is too simple to judge significantly.

Conclusions

1. DDPM is a powerful generative model allowing generate variously conditioned images from gaussian noise;
2. It does not matter what noise scheduler to choose for training, but sampling is a bottleneck – DDIM or numerical ODE samplers are good choices
3. To get cool generations without any additional model, classifier-free guidance is the thumb up! large guidance rate would let the model to be creative
4. β-schedule is the thing: actually, the choice is arbitrary but works well in practice
5. Do not train DDPM from scratch for your RGB dataset – it is better to finetune one of pipelines for your needs!