Notes on Diffusion Models

Master in Artificial Intelligence
University of Bologna A.Y. 2022/2023

Diego Biagini

Contents

1 Theory **1**

1.1 The diffusion process 1

1.1.1 Discrete-time diffusion 1

1.1.2 Continuous Time Diffusion 5

1.1.3 Guiding the diffusion process 10

1.2 Latent Diffusion Models 11

1.2.1 Autoencoder 11

1.2.2 U-Net 12

1.2.3 Conditioning 13

1.2.4 Exploring Automatic's UI 14

Chapter 1

Theory

1.1 The diffusion process

Diffusion models are generative models based around the idea of a **diffusion process**.

A diffusion process can be formally defined through two formulations, a discrete-time one and a continuous-time one, the objective of both of them is to gradually perturb samples from a data-generating distribution $x_0 \sim q$ by iteratively injecting noise and transforming it into a simple prior distribution.

1.1.1 Discrete-time diffusion

Forward process

A discrete-time diffusion process can be defined as follows: given our initial sample x_0 a discrete time diffusion process is obtained by adding a small amount of noise in T steps, resulting in a sequence of increasingly noisier images $x_1, \dots, x_T$.
The amount of noise added after each of these steps is defined by a variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$
The distribution of each of these steps is thus defined as:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

If we know the original sample we can iteratively apply this definition to obtain a joint distribution over all the steps:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

It turns out that we don't need to apply noise many times to obtain a sample x_t at a wanted noise level, we can reparametrize $q(x_t)$ to just depend on the original data sample $q(x_t|x_0)$.
Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=0}^{t} \alpha_t$, we can define:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

It follows that a sample can be obtained by sampling from a standard Gaussian as:

$$x_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\mathcal{N}(0,1)$$

When we refer to a particular standard Gaussian noise introduced at time t we will use the notation ϵ_t.

Noising schedule

In general our schedule shall add more noise the farther along the diffusion process we are, this translates into: $\beta_1 < \beta_2 < \cdots < \beta_T$. Many schedules are possible, all of them are defined according to the number of noising steps T and some additional parameters, some of the most popular are:

- **Linear** schedule, where $\beta_t = \beta_{start} + (\beta_{end} - \beta_{start}) \cdot \frac{t}{T}$, i.e. evenly divide the real segment $[\beta_{start}, \beta_{end}]$
- **Cosine** schedule, given a scale parameter s, define $\bar{\alpha_t} = cos(\frac{t+s}{1+s} \cdot \frac{\pi}{2})^2$, after dividing α_t by α_0 the schedule is defined as $\beta_t = 1 - \frac{\bar{\alpha_t}}{\bar{\alpha_{t-1}}}$, this creates a schedule with values between 0 and 1, the latter of which seem to be far too high
- **Sigmoid**, given parameters $start, end, \tau$, we define

  $$v_{start} = \sigma(\frac{start}{\tau}) \qquad v_{end} = \sigma(\frac{end}{\tau})$$

  and from these

  $$\bar{\alpha_t} = \frac{\sigma(\frac{t \cdot (end - start) + start}{\tau} + v_{end})}{v_{end} - v_{start}}$$

  after dividing α_t by α_0 the schedule is defined as $\beta_t = 1 - \frac{\bar{\alpha_t}}{\bar{\alpha_{t-1}}}$

It turns out that a non linear schedule is experimentally beneficial[4], however even stable diffusion uses a linear schedule.
Schedules taken from github/denoising-diffusion-pytorch.

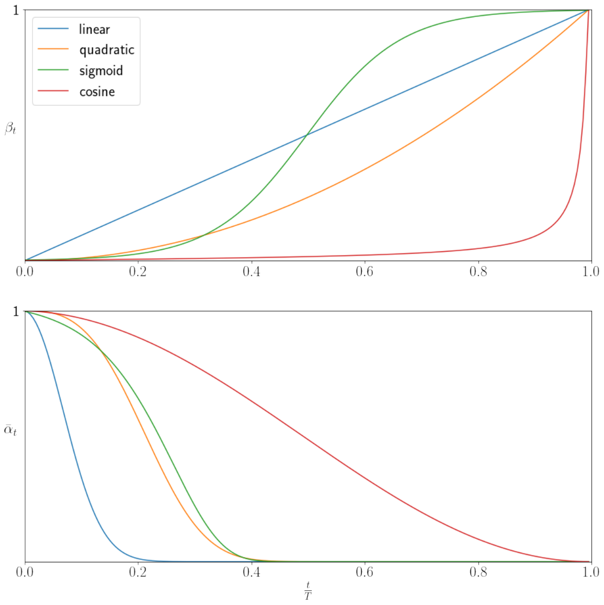


Figure 1.1: Different noising schedules with $\beta_{start} = 0, \beta_{end} = 1$; for cosine $s = 0.008$

Reverse diffusion process

To generate new samples from a distribution similar to $q(x_0)$ we need to reverse the diffusion process, that is derive $q(x_{t-1}|x_t)$.
A nice property of diffusion processes is the fact that if β_t is small enough $q(x_{t-1}|x_t)$ will be a Gaussian.
Obtaining $q(x_{t-1}|x_t)$ requires summing over the entire data distribution, however it is tractable if we know the original sample x_0:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I})$$

In fact using Bayes' rule we have:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)}$$

This is a product of Gaussians which ultimately results in a Gaussian with parameters:

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\Big)$$

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

Our objective is thus to estimate this reverse distribution using a neural network parametrized by θ, i.e. obtain a distribution:

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

Optimization

Our objective is to maximize the likelihood of our novel data-generating distribution $p_\theta(x_0)$, to do so we can proceed in a way similar to a VAE, i.e. minimizing the KL-divergence between the true and estimated distributions of the reverse process ($D_{KL}(p_\theta||q)$), which corresponds to maximizing the variational lower bound:

$$-\log p_\theta(\mathbf{x}_0) \leq -\log p_\theta(\mathbf{x}_0) + D_{\text{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)\|p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0))$$

Which written in loss terms amounts to finding:

$$L_{\text{VLB}} = \mathbf{E}_{q(\mathbf{x}_{0:T})}\log\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} \geq -\mathbf{E}_{q(\mathbf{x}_0)}\log p_\theta(\mathbf{x}_0)$$

This loss/ELBO can actually be decomposed in multiple terms:

$$\begin{aligned}
L_{\text{VLB}} &= L_T + L_{T-1} + \cdots + L_0 \\
\text{where } L_T &= D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T)) \\
L_t &= D_{\text{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})) \text{ for } 1 \leq t \leq T-1 \\
L_0 &= -\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)
\end{aligned}$$

Except for L_0 each of these terms is a KL-divergence between Gaussians, which can be computed explicitly. L_T is a constant and can be ignored ($p_\theta(x_T)$ is a normal gaussian). On the other hand L_0 can be optimized by computing a discretized likelihood which has the form:

$$p_\theta(x_0|x_1) = \prod_{i=1}^{D}\int_{\delta_-(x_0^i)}^{\delta_+(x_0^i)}\mathcal{N}(x; \mu_\theta^i(x_1, 1), \sigma_1^2)dx$$

Despite the loss having this form, in practice what is done is optimizing each of the terms separately, i.e. a sample is first perturbed to a certain noise level and then the term relative to that noise level is optimized.

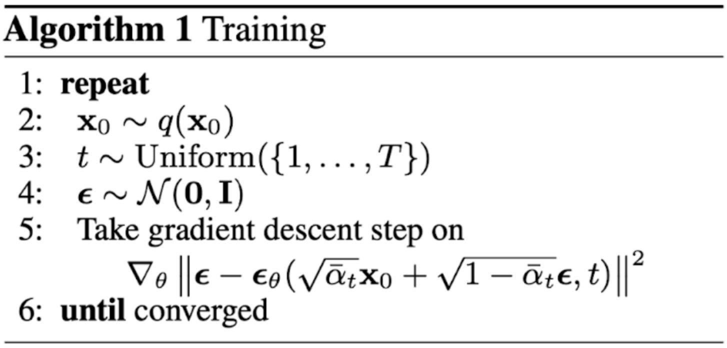
This objective can actually be made simpler if we consider our network as a noise predictor:

$$\begin{aligned}
L_t &= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\Big[\frac{1}{2\|\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)\|_2^2}\|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2\Big] \\
&= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\Big[\frac{1}{2\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\Big) - \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big)\|^2\Big] \\
&= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\Big[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\Big]
\end{aligned}$$

And if we further remove the weighting term we obtain:

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

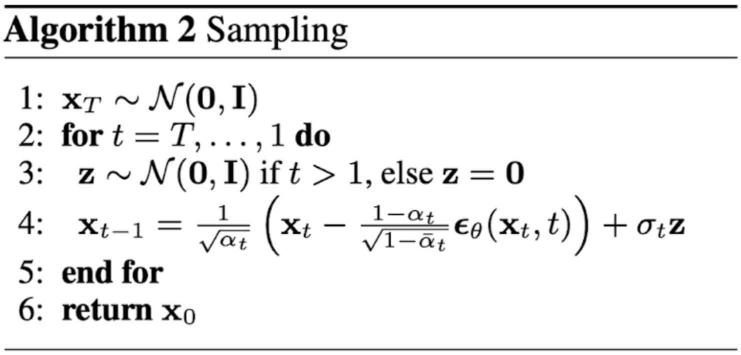
Which is the objective used in [2].
The final training algorithm looks like:



```
Algorithm 1 Training
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
      $\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$
until converged
```

Sampling

The basic sampling procedure consists in traversing the entire diffusion process backwards.



```
Algorithm 2 Sampling
$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ do
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$
end for
return $\mathbf{x}_0$
```

Each step of this process is performed by first using the network to estimate the noise added to the original image (in the noise-predictor formulation), obtaining the estimates for the moments of the reverse process and sampling from such a parametrized distribution to get a slightly less noisy image.
A big problem with such an algorithm is the fact that we need to traverse the entire chain of diffusion steps, effectively forcing us to perform thousands of forward passes. A first approach is to follow a strided sampling schedule.
For a model trained with T diffusion steps it is also possible to sample using an arbitrary subsequence S of t values.
Given the training noise schedule $\bar{\alpha}_t$, for a given sequence S we can obtain the sampling noise schedule $\bar{\alpha}_{S_t}$, the corresponding variances are:

$$\beta_{S_t} = 1 - \frac{\bar{\alpha}_{S_t}}{\bar{\alpha}_{S_{t-1}}} \qquad \tilde{\beta}_{S_t} = \frac{1-\bar{\alpha}_{S_{t-1}}}{1-\bar{\alpha}_{S_t}}\beta_{S_t}$$

With this new schedule we can now traverse a surrogate diffusion chain, where each step is defined as:

$$p_\theta(x_{S_{t-1}}|x_{S_t}) = \mathcal{N}(\mu_\theta(x_{S_t}, S_t), \Sigma_\theta(x_{S_t}, S_t)$$

Another approach is instead to use a different formulation for the sampling process, called **DDIM**.
The main idea behind this approach is control how much stochasticity is added to our reverse diffusion process, according to an hyperparameter η.
Like in strided DDPM we define a subset of steps over which to diffuse.
We then inject our stochasticity parameter into the variance:

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

In the end our reverse process is defined through:

$$p_\theta(x_{S_{t-1}}|x_{S_t}) = \mathcal{N}(x_{S_{t-1}}; x_{0_\theta}(x_{S_t})\sqrt{\bar{\alpha}_{S_t}} + \sqrt{1-\bar{\alpha}_{S_t}-\sigma^2}\epsilon_\theta(x_{S_t}), \sigma^2_{S_t} I)$$

With $\eta = 0$ we have a purely deterministic process, while with $\eta = 1$ we obtain DDPM. It has been observed that the purely deterministic formulation of DDIM is advantageous when we run a strided sampling schedule with few steps.

Variance parametrization

We have seen that in [2] the variance is fixed to a particular value $\sigma^2 = \tilde{\beta}$.
This value is actually an extreme between two choices of variance schedule: isotropic Gaussian noise, given by β_t, and a delta function, given by $\tilde{\beta}_t$. The choice between the two extremes turned out not to matter too much, with regards to sample quality, due to the fact that β_t and $\tilde{\beta}_t$ are almost equal except when $t = 0$, however learning it benefits the likelihood.
What is proposed in [4] is to let our model output a vector v, a value for each sample dimension, which represents an interpolation weight:

$$\mathbf{\Sigma}_\theta(\mathbf{x}_t, t) = \exp(\mathbf{v}\log\beta_t + (1-\mathbf{v})\log\tilde{\beta}_t)$$

However to learn these weights we cannot use the simple objective L_{simple}, we have to use the full training loss (the variational lower bound) taking care of not optimizing the mean computation in doing so if we are still using the simpler objective to learn the mean.
The overall loss function is thus: $L_{hybrid} = L_{simple} + \lambda L_{vlb}$, the full training process is found in algorithm 1.

1.1.2 Continuous Time Diffusion

There exists another formulation of the diffusion process which makes heavy use of the score matching formulation. The core idea behind this other formulation is expressing the diffusion process in a continuous framework through stochastic differential equations.
This section is an agglomerate of [8] and [3]

Connection with Noise-Conditioned score networks

Diffusion models have many equivalencies with similar models, one of these are score-based generative models.
Proposed in [7], NCSN produce samples (i.e. perform density estimation) through the score function $\nabla_x \log q(x)$, which is estimated through a score network s_θ, trained such that $s_\theta(x) \approx \nabla_x \log q(x)$.
If we go back to the diffusion-like approach of estimating the score function at different noise levels we obtain the following equivalency between NCSN and noise-learning DDPMs:

$$\mathbf{s}_\theta(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t}\log p(\mathbf{x}_t) = \mathbb{E}_{p(\mathbf{x}_0)}[\nabla_{\mathbf{x}_t} p(\mathbf{x}_t|\mathbf{x}_0)] = \mathbb{E}_{p(\mathbf{x}_0)}\Big[-\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}\Big] = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}$$

This is due to the fact that if we have a Gaussian distribution p we have that:

$$\nabla_{\mathbf{x}}\log p(\mathbf{x}) = \nabla_{\mathbf{x}}\Big(-\frac{1}{2\sigma^2}(\mathbf{x}-\boldsymbol{\mu})^2\Big) = -\frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma^2} = -\frac{\boldsymbol{\epsilon}}{\sigma}$$

Algorithm 1 Optimizing L_{hybrid}

```
repeat
    sample real data $x_0 \sim q(x_0)$
    sample noise level $t$
    $\epsilon \sim \mathcal{N}(0, I)$
    $x_t \leftarrow \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$
    Obtain the true posterior moments $\tilde{\mu}_t(x_t, x_0) \quad \tilde{\beta}_t$
    Get noise and interpolated-variance predictions from the model: $\epsilon_\theta(x_t, t) \quad v_\theta(x_t, t)$
    Obtain predicted unnoised image: $\tilde{x}_0(x_t, \epsilon_\theta(x_t, t))$
    $\Sigma_\theta(x_t, t) \leftarrow \exp(v_\theta \log \beta_t + (1 - v_\theta) \log \tilde{\beta}_t)$
    if $t \neq 1$ then
        Compute KL-div between true and estimated Gaussian
        $L_{vlb} \leftarrow KL(\mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I), \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, \tilde{x}_0), \Sigma_\theta(x_t, t) I))$
    else
        Compute discretized Gaussian log likelihood (in practice done with log-variance)
        $L_{vlb} \leftarrow -DGLL(x_0; \mathcal{N}(\tilde{x}_0; \tilde{\mu}(x_t, \tilde{x}_0), \Sigma_\theta(x_t, t) I))$
    end if
    $L_{simple} \leftarrow ||\epsilon - \epsilon_\theta(x_t, t)||^2$
    $L_{hybrid} \leftarrow L_{simple} + \lambda L_{vlb}$
    Optimization step
until converged
```

Algorithm 2 Optimizing L_{vlb}

```
repeat
    sample real data $x_0 \sim q(x_0)$
    sample noise level $t$
    $\epsilon \sim \mathcal{N}(0, I)$
    $x_t \leftarrow \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$
    Obtain the true posterior moments $\tilde{\mu}_t(x_t, x_0) \quad \tilde{\beta}_t$
    Get noise and variance predictions from the model: $\epsilon_\theta(x_t, t) \quad \Sigma_\theta(x_t, t)$
    Obtain predicted unnoised image: $\tilde{x}_0(x_t, \epsilon_\theta(x_t, t))$
    if $t \neq 1$ then
        Compute KL-div between true and estimated Gaussian
        $L_{vlb} \leftarrow KL(\mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I), \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, \tilde{x}_0), \Sigma_\theta(x_t, t) I))$
    else
        Compute discretized Gaussian log likelihood
        $L_{vlb} \leftarrow -DGLL(x_0; \mathcal{N}(\tilde{x}_0; \tilde{\mu}(x_t, \tilde{x}_0), \Sigma_\theta(x_t, t) I))$
    end if
    Optimization step
until converged
```

SDE Formulation

Just like in the discrete approach, our goal is to construct a diffusion process $\{x_t\}_{t=0}^{T}$ indexed by a **continuous** time variable t, such that $x_0 \sim q$ are samples taken from the data-generating distribution and $x_T \sim q_T$ are samples from a prior distribution, in most cases a simple Gaussian.
The diffusion process can be modeled as the solution to an Itō SDE:

$$\mathrm{d}\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)\mathrm{d}t + g(t)\mathrm{d}\boldsymbol{w}$$

Where $\boldsymbol{w}$ is the standard Wiener process, responsible for the stochasticity, $\boldsymbol{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is a vector- valued function called the drift coefficient, and $g(\cdot) : \mathbb{R} \to \mathbb{R}$ is a scalar function known as the diffusion coefficient.

As always our objective is to reverse this diffusion process in order to turn samples from the prior distribution p_T into samples from the data-generating distribution p_0.
It turns out that the reverse of a diffusion process is also a diffusion process whose evolution is given by the following reverse-time SDE:

$$d\boldsymbol{x} = [\boldsymbol{f}(\boldsymbol{x}, t) - g(t)^2 \nabla_x \log p_t(\boldsymbol{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\boldsymbol{w}}$$

Thus if we can compute the score function $\nabla_x \log p_t(x)$ for all t we can effectively derive the reverse diffusion process and simulate it to sample from the data-generating distribution.

Equivalence with DDPM and SMLD

The noise perturbations used in SMLD and DDPM are actually discretizations of two different SDEs.
Let's consider DDPM whose discrete Markov chain is defined as:

$$x_i = \sqrt{1 - \beta_{S_{t_i}}} x_{i-1} + \sqrt{\beta_{S_{t_i}}} z, \quad i = 1, \dots, N$$

As $N \to \infty$ this converges to the following SDE:

$$\mathrm{d}\boldsymbol{x} = -\frac{1}{2}\beta_t \boldsymbol{x}\mathrm{d}t + \sqrt{\beta_t}\mathrm{d}\boldsymbol{w}$$

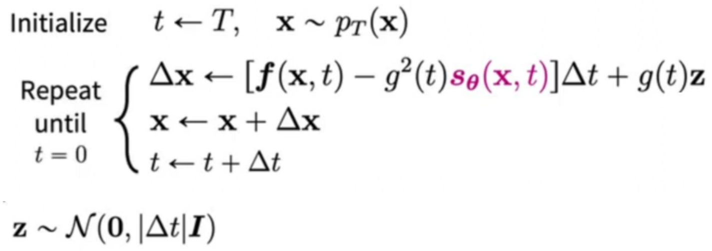
Sampling from an SDE

In DDPM after we have obtained the optimal model $s_\theta(x_t, t)$ samples are generated by starting from $x_T \sim \mathcal{N}(0, I)$ and following the estimated Markov chain:

$$x_{t-1} = \frac{1}{\sqrt{1 - \beta_t}}(x_t + \beta_t s_\theta(x_t, t)) + \sqrt{\beta_t}\mathcal{N}(0, I)$$

This method is called ancestral sampling and corresponds to a particular discretization of the reverse-time SDE obtained when $N \to \infty$.
Such a process is generalizable to arbitrary SDEs, i.e. after training a time-dependent score-based model, we can use it to construct the reverse-time SDE and then simulate it with numerical approaches to generate samples.
In particular we can use general purpose numerical solvers to approximate the trajectory of the SDE, like the Euler-Maruyama and the stochastic Runge-Kutta methods[1.2].

Figure 1.2: The Euler-Maruyama procedure for solving SDEs



```
Initialize    $t \leftarrow T, \quad \mathbf{x} \sim p_T(\mathbf{x})$
Repeat until $t = 0$
    $\Delta\mathbf{x} \leftarrow [\boldsymbol{f}(\mathbf{x}, t) - g^2(t)\boldsymbol{s_\theta}(\mathbf{x}, t)]\Delta t + g(t)\mathbf{z}$
    $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$
    $t \leftarrow t + \Delta t$
$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, |\Delta t|\boldsymbol{I})$
```

ODE approach

For all diffusion processes, there exists a corresponding deterministic process whose trajectories share the same marginal probability densities $\{p_t(x)\}_{t=0}^{T}$ as the SDE.
This deterministic process satisfies an ODE, often called probability flow ODE:

$$\mathrm{d}\boldsymbol{x} = [\boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2}g(t)^2\nabla_x \log p_t(\boldsymbol{x})]\mathrm{d}t$$

And we can use numerical methods to integrate it backwards.
Samplers can work on either the SDE or ODE formulation, however a big difference is the fact that unlike reverse diffusion samplers or ancestral sampling, there is no additional randomness once the initial sample is obtained from the prior distribution.
At this point [3] tries to take another generalization step and they define the ODE based on $\sigma(t)$, a real-valued function which defines the desired noise level at time t. We can sample from such a noise level by sampling from $\mathcal{N}(x_0, \sigma(t))$
Their final probability flow ODE is defined as:

$$\mathrm{d}\boldsymbol{x} = -\dot{\sigma}(t)\sigma(t)\nabla_x \log p(x; \sigma(t))\mathrm{d}t$$

Where $p(x; , \sigma) = p_0 * \mathcal{N}(0, \sigma(t)^2 I)$, the convolution/sum between the noise distribution and the data-generating distribution.
A further generalization of this formulation also employs an addition scale schedule $s(t)$ and considers $x = s(t)\hat{x}$ as a scaled version of the original variable $\hat{x}$.
Now we want a way to express the score function in this new framework, what we do is use a denoiser function $D(x; \sigma)$ that minimizes the denoising error for samples drawn from the data-generating distribution.

From this we can define: $\nabla_x \log p(x; \sigma) = \frac{D(x:\sigma)-x}{\sigma^2}$
After substituting the score function into the ODE we can find its solution through numerical integration, which requires the definition of the integration method (e.g. Euler) and the discrete sampling times.

Furthermore they define the denoiser function through a number of hyperparameters, different choices of which can obtain the same behaviour as other frameworks (like DDPM):

$$D_\theta(x; \sigma) = c_{skip}(\sigma)x + c_{out}(\sigma)F_\theta(c_{in}(\sigma)x; c_{noise}(\sigma))$$

Where F_θ is the raw neural network.
As always, after reframing everything in this framework we can go back to another SDE formulation, which usually generates better samples. In practice this new SDE is a sum of the probability flow ODE and a time-varying Langevin diffusion SDE.

Training in Karras

The algorithms found in this and the following sections are taken from the repository k-diffusion .
Using Karras' notation the overall training loss can be defined as:

$$E_{\sigma,x_0,n}[\lambda(\sigma)||D(x_0+n;\sigma)-y||_2^2]$$

where σ is the noise level, sampled from a training noise distribution p_{data}, and $n \sim \mathcal{N}(0,\sigma^2 I)$ is Gaussian noise. A weighting term $\lambda(\sigma)$ is also added to give more or less importance to the noise level σ.
To have a look at all the hyperparameters refer to [3]
We can equivalently express this loss with respect to the raw network output as follows:

$$\mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\Big[\underbrace{\lambda(\sigma)\, c_{\text{out}}(\sigma)^2}_{\text{effective weight}}\Big\|\underbrace{F_\theta\big(c_{\text{in}}(\sigma)\cdot(x_0+\boldsymbol{n}); c_{\text{noise}}(\sigma)\big)}_{\text{network output}} - \underbrace{\tfrac{1}{c_{\text{out}}(\sigma)}\big(x_0 - c_{\text{skip}}(\sigma)\cdot(x_0+\boldsymbol{n})\big)}_{\text{effective training target}}\Big\|_2^2\Big]$$

The overall training procedure in practice is found in 3.

Algorithm 3 Training Karras

```
repeat
    sample real data $x_0$
    sample from the noise distribution to get $\sigma$ (In Karras $\ln(\sigma) \sim \mathcal{N}(P_{mean}, P_{std}^2)$
    sample base noise $\epsilon \sim \mathcal{N}(0, I)$
    obtain the noised input $x_\sigma \leftarrow x_0 + \sigma\epsilon$
    obtain model output, noise and variance $\epsilon_\theta(x_\sigma + c_{in}, c_{noise}(\sigma)), \Sigma_\theta(x_\sigma + c_{in}, c_{noise}(\sigma))$
    define the target as $\epsilon_\sigma \leftarrow \frac{(x_0 - c_{skip} x_\sigma)}{c_{out}}$
    compute the loss as $L \leftarrow \frac{1}{2}(\frac{(\epsilon_\theta - \epsilon_\sigma)^2}{\Sigma_\theta} + \log \Sigma_\theta)$ (weighting already considered and simplified)
    Optimization step
until converged
```

Sampling in Karras

Algorithm 4 Sampling using with ODE/SDE

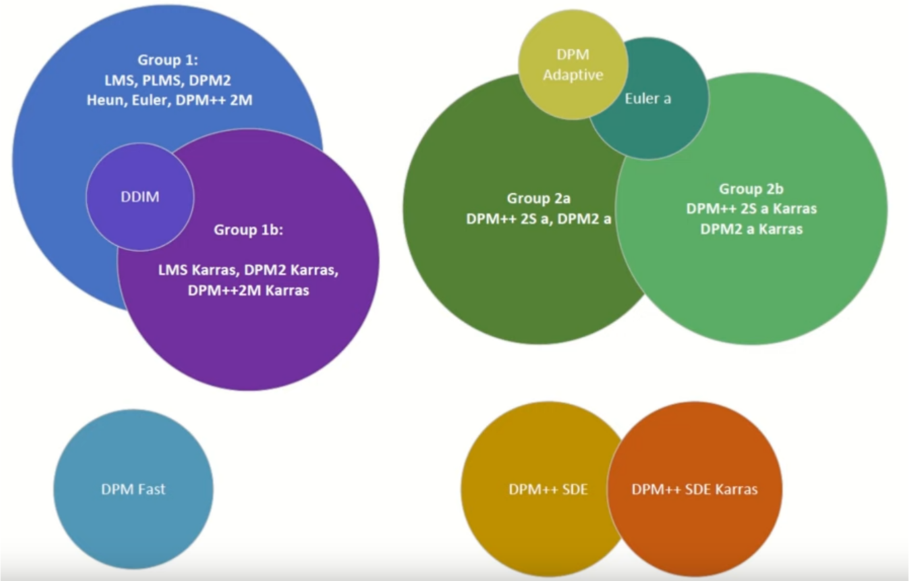
```
Obtain the time-steps $t_{i<T}$ given the total number of diffusion steps $T$
Obtain the noise level $\sigma(t)$ for each of those steps, according to schedule
Obtain a starting sample of Gaussian noise from $\mathcal{N}(0, \sigma_{max}^2 I)$
Run the preferred sampler on the schedule-transformed steps
```

How they(kcrowson) came up with all these different samplers is an unsolved mystery, but here are some ideas of how they work (from @iScienceLuvr) (copy pasted, tbh I find this explanation highly inconsistent):

- Euler: sampling with Euler's method from the DDIM probability flow ODE
- Euler ancestral: ancestral sampling with Euler's method from the VE-SDE for a DDPM
- Heun: sampling with Heun's method from the DDIM probability flow ODE
- LMS: Sampling with linear multi step method (4th order Adams-Bashforth) of the DDIM probability flow ODE

- LMS ancestral: DDPM as an ODE with LMS then add noise
- DPM-2: DPM solver with the DDIM probability flow ODE plus something else
- DPM-2 ancestral:DPM-2 with DDPM as an ODE then add noise

Ultimately sampler choice seems to not matter too much, more often than not different samplers will obtain the same results, especially when diffusing on a latent space and when the number of steps is greater than 20.
Experimental results show the following similarities between groups of samplers (credit https://www.youtube.com/@siliconthaumaturgy7593) tldw: ancestral and non-ancestral is the biggest discriminator.



Furthermore there exist adaptive samplers which do not require a number of steps to be set, however they are often very inefficient since after a while increasing the number of steps does not matter much.

1.1.3 Guiding the diffusion process

Given a diffusion model $q(x_{t-1}|x_t)$, we use the term guidance to refer to the process of using some additional signal (usually weighted) to inject conditioning information into the process.

Classifier-guided diffusion

In [1] the authors explored the usage of an additional network, a classifier $f_\phi(y|x_t)$ trained on **noisy** images, to guide the diffusion process through its gradient $\nabla_x \log f_\phi(y|x_t)$.
To understand how this works we have to go back to the score-model formulation and consider the joint distribution of the model with the added conditioning:

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

Our weighted guided noise prediction thus becomes:

$$\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t, y) = \boldsymbol{\epsilon}_\theta(x_t, t) - \sqrt{1-\bar{\alpha}_t}\; w\nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t)$$

This approach however is extremely inconvenient, it requires the training of an additional network which has to be trained on the diffusion process itself with a known amount of classes.

Classifier-free guidance

The idea behind classifier-free guidance is to still have a guiding signal without having to train two different networks.
We do so by parametrizing two distributions $p_\theta(x)$ ($\epsilon_\theta(x_t, t)$) and $p_\theta(x|y)$ ($\epsilon_\theta(x_t, t, y)$), this is done by training our noise predictors respectively on the non-conditional case (achieved by setting the conditioning information to a null value) and the paired data (x, y).
In this way we are able to obtain an implicit classifier, whose gradient can be used to guide the diffusion process:

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

1.2 Latent Diffusion Models

The idea behind Latent diffusion is models is to perform the denoising diffusion process in a latent space tightly related to the original image-space formulation.
To obtain such a representation we can train another network, an autoencoder, whose objective after being trained is to downsample the original image by a factor f, such that if the original image was $x \in \mathbf{R}^{H \times W \times 3}$ the latent representation is $z \in \mathcal{R}^{h \times w \times c}$.

1.2.1 Autoencoder

The autoencoder model is actually trained using multiple objectives:

- Reconstruction objective: your classical perceptual loss for autoencoders, this is often replaced with a learned perceptual loss (LPIPS)

  $L_{rec}(x, \mathcal{D}(\mathcal{E}(x))) = ||x - \mathcal{D}(\mathcal{E}(x))||_2^2$

- Adversarial objective: in addition to the AE a patch-based discriminator D_ψ is trained to differentiate images from the reconstruction, this results in a GAN-like adversarial loss:

  $L_{adv}(x, \mathcal{D}(\mathcal{E}(x)), D_\psi) = [\log D_\psi(x) + \log(1 - \mathcal{D}(\mathcal{E}(x)))]$

- Regularization objective: two possibilities, a classical VAE regularization towards a standard Gaussian:

  $L_{reg}(x; \mathcal{E}, \mathcal{D}) = KL(\mathcal{N}(\mathcal{E}_\mu, \mathcal{E}_\sigma), \mathcal{N}(0, I))$

  (Is the direction of the KL correct?)

  or a vector-quantization regularization, such that we learn a codebook $|\mathcal{Z}|$, in this case our latent distribution is defined as

  $$q(z|x) = \begin{cases} 1 & \text{for } k = \text{argmin}_j ||\mathcal{E}(x) - e_j|| \\ 0 & \text{otherwise} \end{cases}$$

  Where $e_j \in \mathbf{R}^d$ are d dimensional quantized embedding vectors, effectively we are parametrizing a dirac distribution whose mass is concentrated on a single point, the quantized code (found through NN search) of the encoder.

The overall loss has 2 components, a Vector-Quantization component, necessary to move the embedding vectors towards the encoder outputs, and a commitment component, to make sure that the encoder commits to a particular (quantized) embedding and does not wander too much around the space of possible embeddings

$$L_{reg} = ||\text{sg}[\mathcal{E}(x)] - e_j||_2^2 + \beta||\mathcal{E}(x) - \text{sg}[e_j]||_2^2$$

(e_j is the closest vector in the codebook, sg is the stop gradient operator, $\beta = 0.25$ in [5])

(I could not find the code for the quantized regularization loss, so I just used [5] as reference)

In both cases the regularization weight is quite low ($\sim 10^{-6}$)

Overall the unweighted AE objective is:

$$L_{AE} = \text{min}_{\mathcal{E},\mathcal{D}}\text{max}_{\psi}(L_{rec}(x, \mathcal{D}(\mathcal{E}(x))) + L_{adv}(x, \mathcal{D}(\mathcal{E}(x))) + L_{reg}(x; \mathcal{E}, \mathcal{D}))$$

Since this is trained adversarially D_ψ and the autoencoder take alternating optimization steps. An additional detail when using the KL-regularization is the addition of a (batch-spatial-channel, everything) normalization operation for the moments which parametrize the latent-space Gaussian.

1.2.2 U-Net

The basic denoiser model is that of a U-Net, the implementation that we are considering in this section is from CompVis/stable-diffusion.
The first consideration to be taken is how to encode the input timestep t, to do so a sinusoidal position embedding into a d_t dimensionality is performed (same as positional embedding for transformers):

$$PE_{(t,2i)} = \sin(\frac{t}{10000^{\frac{2i}{d_t}}}) \qquad PE_{(t,2i+1)} = \cos(\frac{t}{10000^{\frac{2i}{d_t}}})$$

The first component of the U-Net that we have to define is the base block, which takes in input both the data and the timestep embedding.

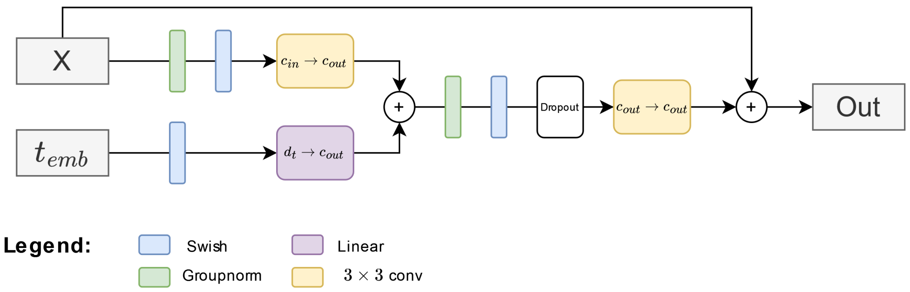


Figure 1.3: Resnet block

After this we have to define the attention block, which can represent either self-attention or cross-attention depending on the needs, in particular for cross attention the K and V are mappings of the context/conditioning information.

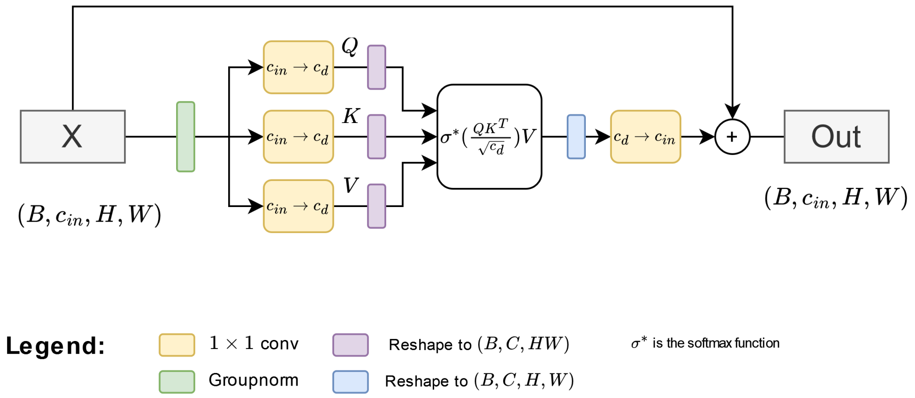


Figure 1.4: Spatial attention block

The last pair of components are the upsampling and downsampling operation, in particular:

- Downsampling: can either be implemented with a 2-pixel-stride 1×1 convolution or through a 2×2 avg-pool
- Upsampling: implemented with a 2x nearest neighbor interpolation plus an optional 1×1 convolution

Finally we can put everything together in the U-Net

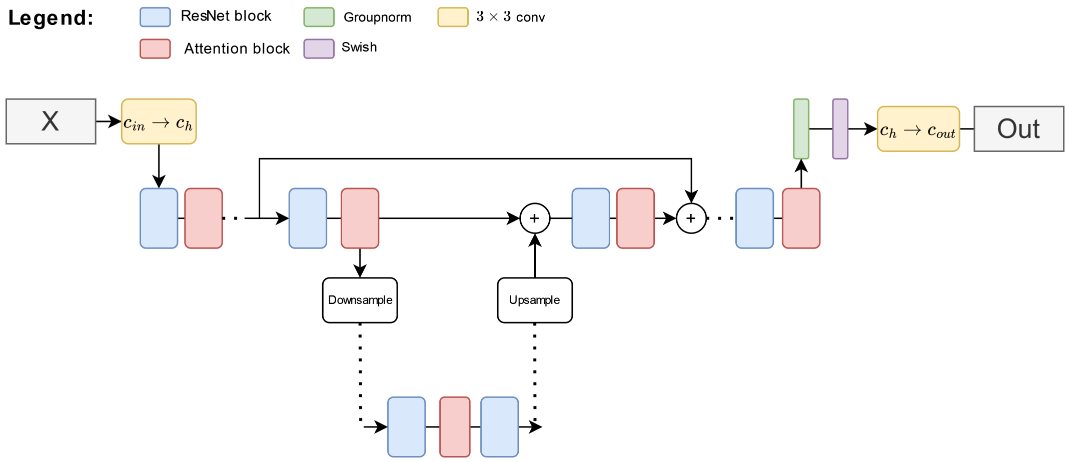


Figure 1.5: U-Net architecture

One thing to take note of is the fact that the attention blocks are not necessarily present at all resolution levels of the U-Net, however having them in the last one is highly advised. Something that is not shown in this picture is the fact that all Resnet blocks take in input the time-step embedding as well.

The choice of denoising network does not have to end up with a U-Net, in [6] it is shown how ViTs are very effective, with a lot of customizability, even in the conditioning approach (cross attention is not the be-all end-all, adaptive normalization can also be used for conditioning).

1.2.3 Conditioning

Having a latent conditional diffusion model amounts to having a noise predictor model which computes $\epsilon_\theta(z_t, t, y)$.

To work on conditioning using various type of modalities we can use a domain-specific encoder $\tau_\theta(y)$ which maps conditioning into a d_τ space.

The generating U-Net is then conditioned through a cross attention mechanism, whose attention is defined as follows: $Att(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}}) \cdot V$

Where the Q, K, V projections are defined as:

$$Q = W_Q \cdot \varphi(z_t), K = W_K \cdot \tau_\theta(y), V = W_V \cdot \tau_\theta(y)$$

Where $\varphi(z_t)$ obtains a flattened representation of the noised image. (Single head formulation, multi-head is the same)
The conditional objective thus becomes:

$$L_{LDM} = \mathbf{E}_{\mathcal{E}(x),y,\epsilon\sim\mathcal{N}(0,I),t}[||\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))||_2^2]$$

For example to have a text-to-image model what they did is train a transformer as a mapper τ_θ.

1.2.4 Exploring Automatic's UI

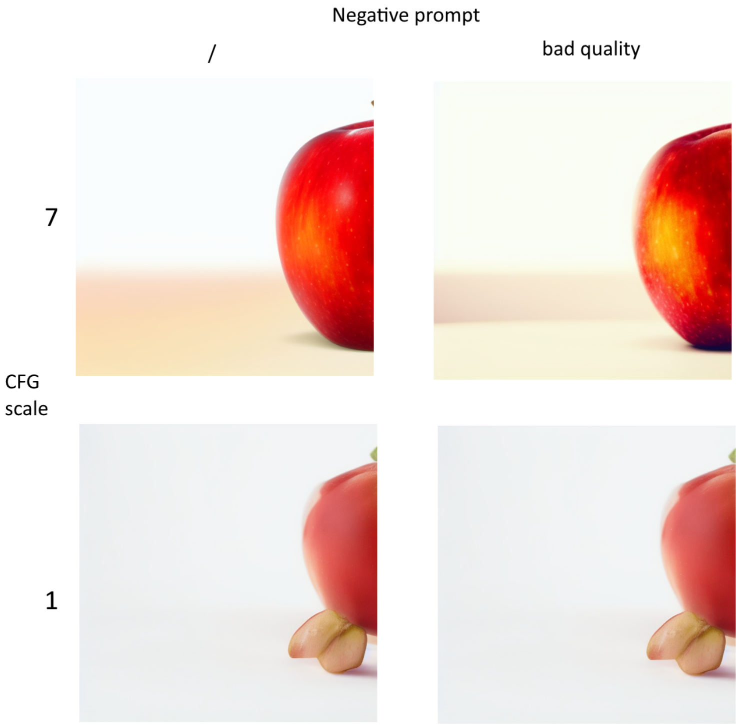
IMG2IMG

Given an input img, do the following:

- encode the input image through the AE and get moments
- Sample from said posterior and start from those sampled latents
- Run the latent under the diffusion process, adding prompt conditioning via cross-attention etc. etc.

The negative prompt is given to the cross-attention input by stacking it together with the positive. After that you obtain 2 outputs and you combine them according to a weight, before performing the next denoising step.
WHAT IS THIS NO PAPER MENTIONS THIS??? Because this is the classifier free guidance term, but of course no-one tells you that the negative prompt amounts to the unconditioned part in CFG, which in theory should be the result computed with no conditioning information (instead of the negative one). So, with $CFGs = 1$ (no guidance) we have that the result obtained with an empty negative prompt is the same as the result obtained with a given negative prompt

Figure 1.6: Effect of CFG-scale and negative prompt, all four images have the same positive prompt("an apple under a tree"), sampler, steps, denoising strenght(euler-a, 20, 1) and starting seed(42)



Denoising strenght s: what this amounts to is a reshaping of the schedule such that only the first $s \cdot steps$ are taken. This makes sense with img2img using the encoded og image as starting latent, since what we are effectively doing is perturbing it and then reconstructing it, the more perturbation steps(higher denoising strenght) we take the more the final output will be different.
In particular the effect of denoising strenght depends on the noising schedule, in a non-linear schedule where β is increased by a bigger amount the farther along the schedule we are (which is the type schedule we usually use) the difference between $s = 0.1$ and $s = 0.2$ is **very** different from the difference between $s = 0.6$ and $s = 0.7$.

Inpaint

We can perform inpainting with a variety of settings/parameters. The first choice is which area to inpaint:

- Only the masked area, in this case the input to our model will be only the masked area, possibly with padding (in case of padding what we will do is obtain a mask which isolates padding information from the initial mask, after defining this proceed like the other possibility below)
- The entire image

The second choice concerns what to do with the masked area, in particular what will the latents of the masked area be :

- Fill: before encoding apply heavy gaussian blur to the masked area
- Original: just encode the entire image
- Latent noise: encode what is outside the mask normally, but replace the latent mask with random noise

Overall the inpainting procedure works as follows:

- You have an image and a mask
- Optionally apply gaussian blur to the edges of the mask which will be considered only when putting mask and unmasked results back together
- Define the latent of the mask according to parameter 2
- Backward diffusion procedure for the entire latent
- The output is the initial latent where the mask was not and the sampled values where the mask is

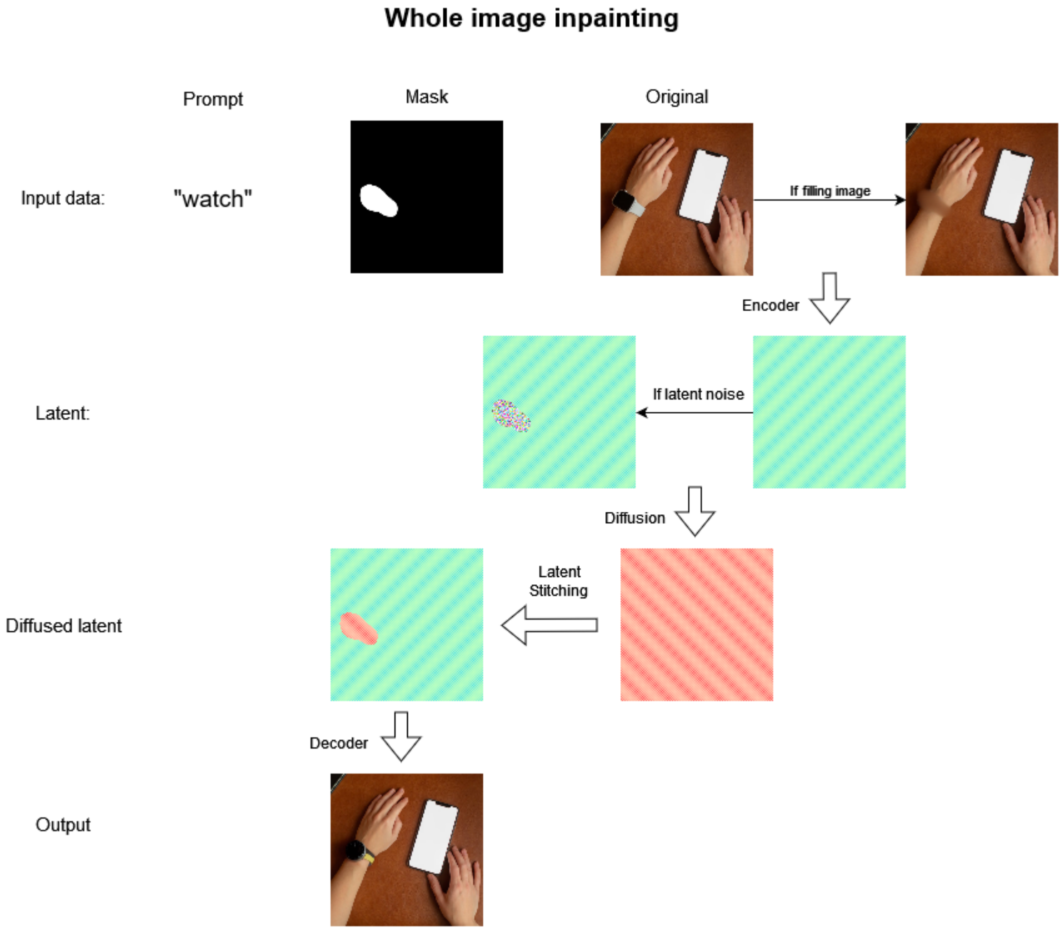


Figure 1.7: Inpainting procedure

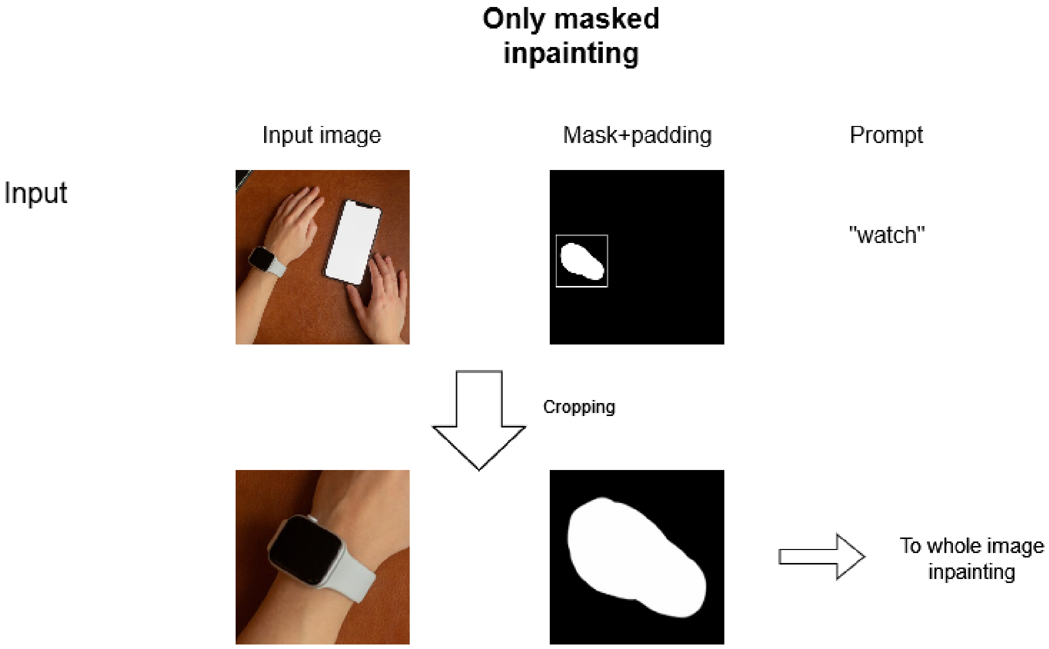


Figure 1.8: Only mask inpainting

Bibliography

[1] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. DOI: 10.48550/ARXIV.2105.05233. URL: https://arxiv.org/abs/2105.05233.

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. DOI: 10.48550/ARXIV.2006.11239. URL: https://arxiv.org/abs/2006.11239.

[3] Tero Karras et al. *Elucidating the Design Space of Diffusion-Based Generative Models*. 2022. DOI: 10.48550/ARXIV.2206.00364. URL: https://arxiv.org/abs/2206.00364.

[4] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. DOI: 10.48550/ARXIV.2102.09672. URL: https://arxiv.org/abs/2102.09672.

[5] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. *Neural Discrete Representation Learning*. 2017. DOI: 10.48550/ARXIV.1711.00937. URL: https://arxiv.org/abs/1711.00937.

[6] William Peebles and Saining Xie. *Scalable Diffusion Models with Transformers*. 2022. DOI: 10.48550/ARXIV.2212.09748. URL: https://arxiv.org/abs/2212.09748.

[7] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. 2019. DOI: 10.48550/ARXIV.1907.05600. URL: https://arxiv.org/abs/1907.05600.

[8] Yang Song et al. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2020. DOI: 10.48550/ARXIV.2011.13456. URL: https://arxiv.org/abs/2011.13456.