Deep Generative Models

Diffusion Models

Hamid Beigy

Sharif University of Technology

June 2, 2025

Table of contents

1. Introduction

2. Diffusion models

3. Denoising diffusion probabilistic models

4. Score-based generative models

5. Continuous time models using differential equations

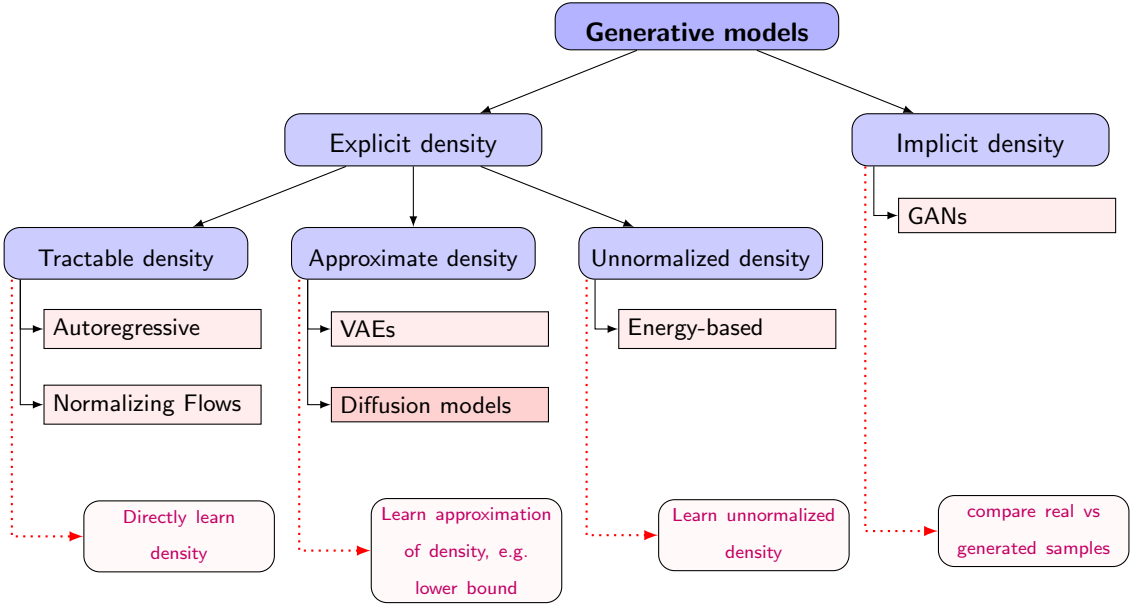
6. Latent Diffusion Models

7. Summary

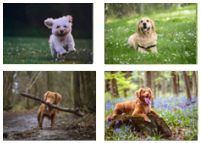
8. References

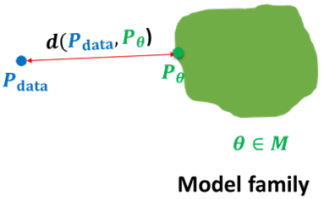
Introduction

Generative models categories



1. Assume that the observed variable $\mathbf{x}$ is a random sample from an underlying process, whose true distribution $p_{\mathrm{d}}(\mathbf{x})$ is unknown.





2. We attempt to approximate this process with a chosen model, $p_\theta(\mathbf{x})$, with parameters θ such that $\mathbf{x} \sim p_\theta(\mathbf{x})$.

3. Learning is the process of searching for the parameter θ such that $p_\theta(\mathbf{x})$ well approximates $p_{\mathrm{d}}(\mathbf{x})$ for any observed $\mathbf{x}$, i.e.

$$p_\theta(\mathbf{x}) \approx p_{\mathrm{d}}(\mathbf{x})$$

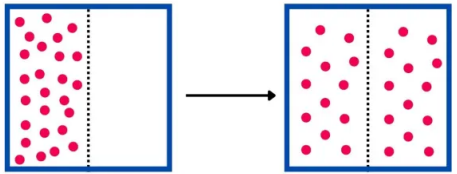
4. We wish $p_\theta(\mathbf{x})$ to be sufficiently flexible to be able to adapt to the data for obtaining sufficiently accurate model and to be able to incorporate prior knowledge.

Credit: Aditya Grover

Diffusion models

What is Diffusion?

1. Diffusion is a fundamental natural phenomenon observed in various systems, including physics, chemistry, and biology.
2. Diffusion is a natural physical phenomenon where particles and energy move from a region of higher concentration to that of lower concentration.
3. At microscopic level, diffusion is driven by random and chaotic movement of particles.
4. This phenomenon is referred to as Brownian motion.
5. This process takes place as systems seek to reach equilibrium, a state where particle concentrations are evenly dispersed throughout the system.



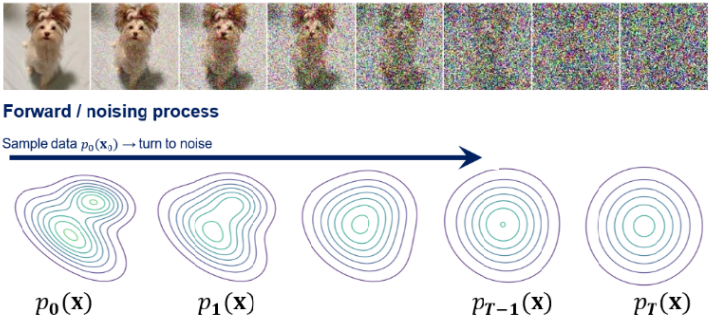
6. The concept of diffusion in AI models is similar to the physical process of diffusion.

What Is Diffusion in the Context of Machine Learning?

1. Diffusion models in Machine Learning function by **learning how to reverse a diffusion process**.
2. This process **begins with a distribution of random noise** and then **gradually transforms it into a structured data such as an image**.
3. Training a model in such a way allows us to pass randomly sampled noise to the model and constructs a new data point.
4. The process itself can be broken down into two main phases:
   - Forward diffusion
   - Reverse diffusion

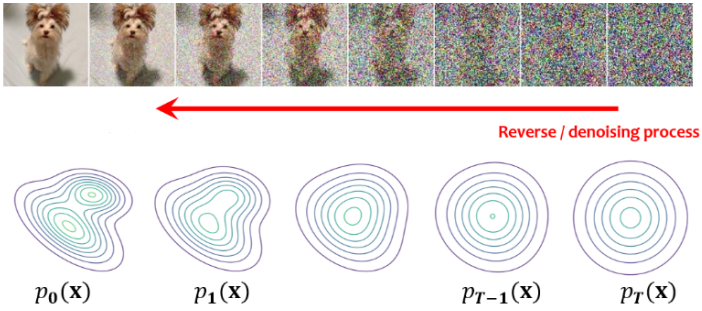
What Is Forward Diffusion?

1. **Forward diffusion** is the process of deconstructing some original data point.
2. Noise is added to the original data point through multiple iterations.
3. In each iteration, we add a small amount of noise until we reach a pure noise state.
4. At this point, we can no longer recognize the original data point.
5. A degree of randomness is introduced at each step as a byproduct of the noise addition.
6. However, the entire process is still designed to be both predictable and reversible.



What Is Reverse Diffusion?

1. The reverse diffusion process aims to **reverse** what was done during forward diffusion.
2. The data is reconstructed from a noisy state back to its original form or a new coherent form through multiple steps.
3. The reverse process is possible because of the precise knowledge of the amount and nature of noise added at each step.
4. This information makes the reverse diffusion possible and enables the reconstruction of the data from the pure noise generated as a result of the forward diffusion.
5. This process is important because performing inference with a trained model is equivalent to performing reverse diffusion.



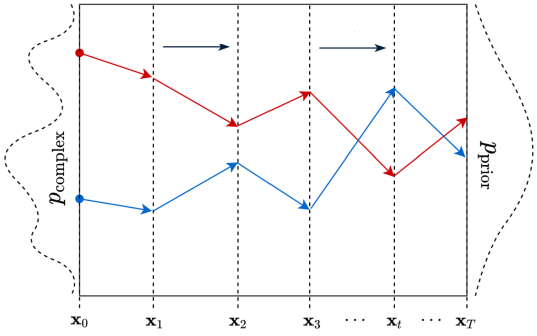
Training and sampling

1. The training objective of diffusion models is by

   maximizing the log-likelihood of the sample generated (at the end of the reverse process) belonging to the original data distribution.

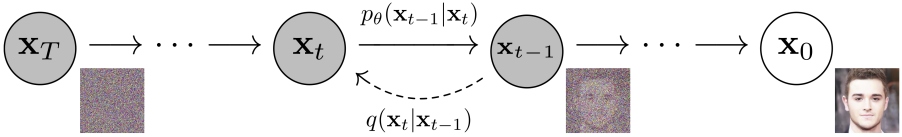
2. To generate samples from diffusion models,

   the reverse diffusion, diffusion models can generate new data samples by starting from a point in the simple distribution and diffusing it step-by-step to the desired complex data distribution.

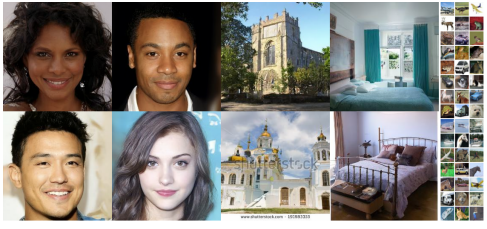


Denoising diffusion probabilistic models

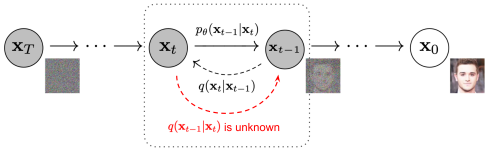
1. **Denoising Diffusion Probabilistic Models** (DDPM) are based on a **Markov model** (Ho, Jain, and Abbeel 2020):



2. In this model
   - We don't know how to sample from $q(\mathbf{x}_0) = p_{\text{d}}(\mathbf{x})$.
   - We know how to sample from $q(\mathbf{x}_T)$.
   - We know how to go from $\mathbf{x}_0$ to $\mathbf{x}_T$.
   - We learn how to go from $\mathbf{x}_T$ to $\mathbf{x}_0$.



1. Given a training point ($\mathbf{x} \sim q(\mathbf{x})$), in forward process, we add small amount of Gaussian noise to the sample in T steps, producing a sequence of noisy samples $\mathbf{x}_1, \ldots, \mathbf{x}_T$.



2. Data sample $\mathbf{x}_0$ gradually loses its distinguishable features as the step t becomes larger.
3. Eventually when $T \to \infty$, $\mathbf{x}_T$ is equivalent to an isotropic Gaussian distribution.
4. The forward process is defined as

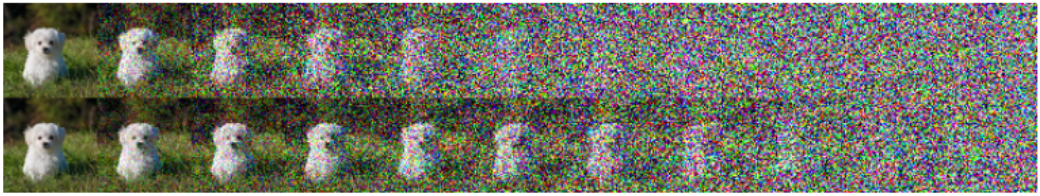
$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$$
$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

5. The term $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is known as the forward diffusion kernel (FDK).
6. The step sizes / diffusion rates β_t are controlled by a variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$.

1. A property of this process is that we can sample $\mathbf{x}_t$ at any arbitrary time step t in a closed form using **reparameterization trick**.
2. Since $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$, we can sample at step t as

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}$$
$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

3. In practice, the authors of DDPMs used a linear variance scheduler and defined $\beta_t \in [0.0001, 0.02]$ and set the total time steps $T = 1000$.
4. Experimental results shows that cosine schedule works better than linear (linear (top) and cosine (bottom) schedules).



1. Let $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{i=1}^{t} \alpha_i$.

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} && \text{where} \quad \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\overline{\boldsymbol{\epsilon}}_{t-2} && \text{where } \overline{\boldsymbol{\epsilon}}_{t-2} \text{ is merge of two Gaussian} \textbf{(1)} \\
&= \vdots \\
&= \sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\overline{\alpha}_t}\boldsymbol{\epsilon} \\
q(\mathbf{x}_t \mid \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\overline{\alpha}_t}\mathbf{x}_0, (1-\overline{\alpha}_t)\mathbf{I})
\end{aligned}
$$

2. **Recall (1):**
   - When we merge two Gaussian with different variance, $\mathcal{N}(\mathbf{0}, \sigma_1^2\mathbf{I})$ and $\mathcal{N}(\mathbf{0}, \sigma_2^2\mathbf{I})$, the new distribution is $\mathcal{N}(\mathbf{0}, (\sigma_1^2 + \sigma_2^2)\mathbf{I})$.
   - Here, the merged distribution, $q(\mathbf{x}_t \mid \mathbf{x}_{t-2})$, is Gaussian and the merged standard deviation is $\sqrt{(1-\alpha_t) + \alpha_t(1-\alpha_{t-1})} = \sqrt{1-\alpha_t\alpha_{t-1}}$.
3. Usually, we use a larger update step when the sample gets noisier, $\beta_1 < \beta_2 < \cdots < \beta_T$ and hence $\overline{\alpha}_1 > \cdots > \overline{\alpha}_T$.

Reverse diffusion process

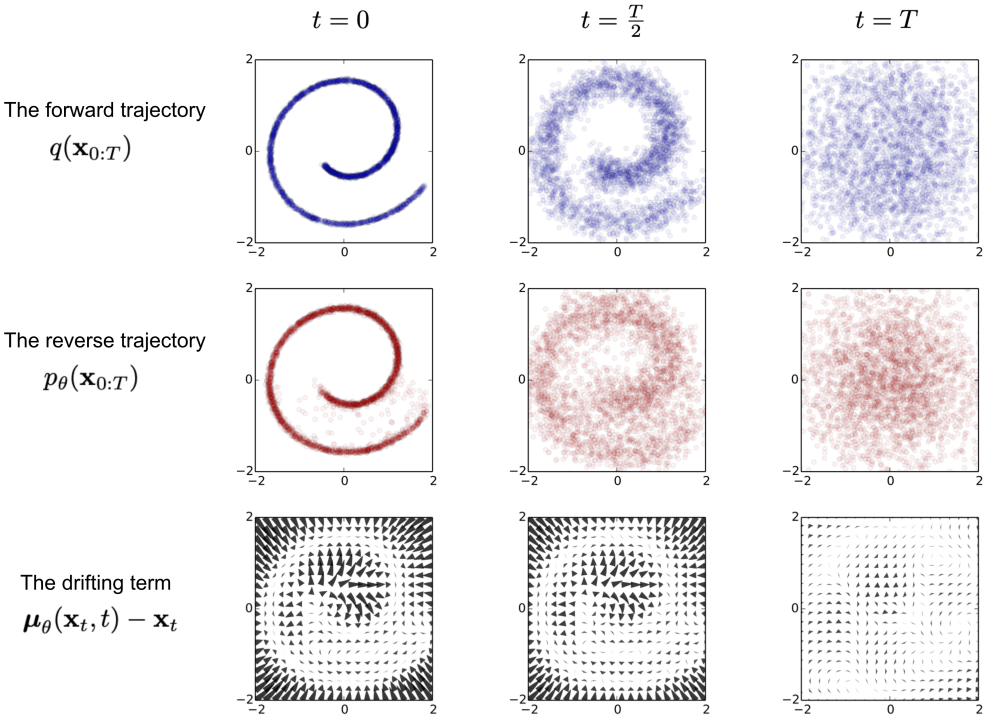
1. If we can reverse the forward process and sample from $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$, we will be able to recreate the true sample from a Gaussian noise input $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
2. if β_t is small enough, $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ will also be Gaussian.
3. Unfortunately, we cannot easily estimate $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ because it needs to use the entire dataset.
4. Therefore we need to learn a model $p_\theta(\mathbf{x})$ to approximate these conditional probabilities for running the reverse diffusion process.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

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

5. The term $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is known as the reverse diffusion kernel (RDK).

Reverse diffusion process



Reverse diffusion process

1. Using Bayes' rule to calculate $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ from $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$:

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

2. Since we do not have access to the marginals $q(\mathbf{x}_{t-1})$ and $q(\mathbf{x}_t)$ would be as follows

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

3. This means that we will need to not only represent the distribution $q(\mathbf{x}_0)$, but also be able to integrate over it.
4. Only for $t > 0$ and conditionally on the starting-point $\mathbf{x}_0$, **all variables are Gaussians**.
5. As a result, we will not be able to calculate $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ directly, but need to do something else.
6. The idea is instead to find a suitable approximation to $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ and let that approximation define the reverse process.

1. The reverse conditional probability is tractable when conditioned on $\mathbf{x}_0$:

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

2. By using Bayes' rule, we obtain:

$$\begin{aligned}
q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) &= q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \\
&\propto \exp\left(-\frac{1}{2}\left(\frac{\left(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1}\right)^2}{\beta_t} + \frac{\left(\mathbf{x}_{t-1} - \sqrt{\overline{\alpha}_{t-1}}\mathbf{x}_0\right)^2}{1-\overline{\alpha}_{t-1}} - \frac{\left(\mathbf{x}_t - \sqrt{\overline{\alpha}_t}\mathbf{x}_0\right)^2}{1-\overline{\alpha}_t}\right)\right) \\
&= \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \alpha_t \mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\overline{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \overline{\alpha}_{t-1}\mathbf{x}_0^2}{1-\overline{\alpha}_{t-1}}\right)\right) \\
&\quad \times \exp\left(\frac{1}{2}\left(\frac{\left(\mathbf{x}_t - \sqrt{\overline{\alpha}_t}\mathbf{x}_0\right)^2}{1-\overline{\alpha}_t}\right)\right) \\
&= \exp\left(-\frac{1}{2}\left(\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\overline{\alpha}_{t-1}}\right)\mathbf{x}_{t-1}^2 - \left(\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{2\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_{t-1}}\mathbf{x}_0\right)\mathbf{x}_{t-1}\right)\right) \\
&\quad \times \exp\left(-\frac{1}{2}C(\mathbf{x}_t, \mathbf{x}_0)\right)
\end{aligned}$$

where $C(\mathbf{x}_t, \mathbf{x}_0)$ is some function not involving $\mathbf{x}_{t-1}$ and details are omitted.

1. Let $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{i=1}^{T} \alpha_i$.
2. Following the standard Gaussian density function, the mean and variance can be parameterized as

$$\tilde{\beta}_t = \frac{1}{\frac{\alpha_t}{\beta_t} + \frac{1}{1-\overline{\alpha}_{t-1}}} = \frac{1}{\frac{\alpha_t - \overline{\alpha}_t + \beta_t}{\beta_t(1-\overline{\alpha}_{t-1})}} = \beta_t \frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}$$

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \frac{\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_{t-1}}\mathbf{x}_0}{\frac{\alpha_t}{\beta_t} + \frac{1}{1-\overline{\alpha}_{t-1}}} \\
&= \left(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_{t-1}}\mathbf{x}_0\right)\frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t} \cdot \beta_t \\
&= \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\overline{\alpha}_{t-1}}\beta_t}{1-\overline{\alpha}_t}\mathbf{x}_0
\end{aligned}$$

3. Since $\mathbf{x}_t = \sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\overline{\alpha}_t}\boldsymbol{\epsilon}$, we can represent $\mathbf{x}_0 = \frac{1}{\sqrt{\overline{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\overline{\alpha}_t}\boldsymbol{\epsilon}_t\right)$ and plug it into the last equation. We obtain:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_t &= \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\overline{\alpha}_{t-1}}\beta_t}{1-\overline{\alpha}_t}\frac{1}{\sqrt{\overline{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\overline{\alpha}_t}\boldsymbol{\epsilon}_t\right) \\
&= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\boldsymbol{\epsilon}_t\right)
\end{aligned}$$

Reverse diffusion process

1. We don't have $q(\mathbf{x}_0)$, but we only have access to samples from it.
2. We use these samples $(\mathbf{x} \sim p_{\text{d}}(\mathbf{x}))$ to approximate $q(\mathbf{x}_0)$.
3. An alternative way is to use Bayes theorem and create a step-wise noise reduction process $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ to approximate $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$.
4. Since the reverse process must have the same functional form as the forward process, each reverse step can be parameterized as a Gaussian, and the parameters can be learned by fitting a neural network.
5. This means that we only have to estimate the mean and variance of the distribution $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ to draw samples from it.
6. Let a neural network parameterized by θ represent this distribution (produce parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, we have

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

7. The neural network takes as input the sample at time t $(\mathbf{x}_t)$ in addition to the time step t itself, in order to account for the variance schedule of the forward process.

Reverse diffusion process

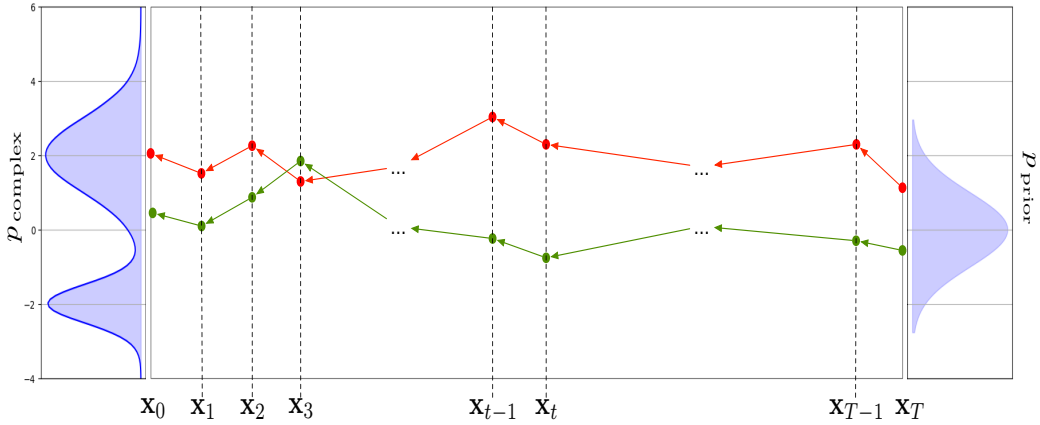
1. Like the forward process, the reverse process is a Markov chain, and we can write the joint probability of a sequence of samples as a product of conditionals and the marginal probability of $\mathbf{x}_T$,

$$p_\theta(\mathbf{x}_{0:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

2. Here, we have : $p_\theta(\mathbf{x}_T) = q(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; 0, \mathbf{I})$.
3. This means that the generation process starts with Gaussian noise, followed by sampling from the learned individual steps of the reverse process.
4. While the original data-samples do not have zero off-diagonal covariance, the noise added to the original samples was diagonal.
5. This means that we can assume that the variance of the removed noise is also diagonal: $\mathbf{\Sigma} = \sigma \mathbf{I}$ for some scalar value σ.
6. When $\mathbf{\Sigma}$ is diagonal, then mean and variance of each dimension can be estimated separately, and the multivariate density function can be described in terms of a product of univariate Gaussian.

1. If the variance is given, we need to predict the mean.
2. We will predict only the mean of the reversed diffusion process-distribution, while the variance follows a schedule parameterized by t.

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



Training the model

1. What objective are we optimizing when training the neural network to learn the following function?

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

2. All generative models attempt to learn the distribution of their training data. Hence, it would make sense to maximize the likelihood assigned to $\mathbf{x}_0$ by the model.
3. Calculating this would require us to marginalize overall steps from $t = T$ down to $t = 1$

$$p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{1:T}) d\mathbf{x}_{1:T}$$

4. Maximizing this gives the process $p_\theta(\mathbf{x})$ over $\mathbf{x}_T \to \mathbf{x}_{T-1} \to \ldots \mathbf{x}_1 \to \mathbf{x}_0$ that has the largest log likelihood of producing the observed $\mathbf{x}_0$ from the noise $\mathbf{x}_T$.
5. However, evaluating this expression involves integrating over all possible trajectories from noise to the data manifold, **which is intractable**.

1. Since $q(\mathbf{x}_{t-1} \mid \mathbf{x})$ is unknown, the problem setting is very similar to VAE.
2. Thus we can use the variational lower bound to optimize the negative log-likelihood.

$$
\begin{aligned}
-\log\, p_\theta(\mathbf{x}_0) &\leq -\log\, p_\theta(\mathbf{x}_0) + \mathrm{D}_{KL}(\, q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{1:T} \mid \mathbf{x}_0)) \\
&= -\log\, p_\theta(\mathbf{x}_0) + \mathbb{E}_{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T}) / p_\theta(\mathbf{x}_0)}\right] \\
&= -\log\, p_\theta(\mathbf{x}_0) + \mathbb{E}_{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log\, p_\theta(\mathbf{x}_0)\right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]
\end{aligned}
$$

3. Let L_{VLB} be variational lower bound, then by taking expectation with respect to $q(\mathbf{x}_0)$, we have

$$
\begin{aligned}
L_{VLB} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \\
&\geq -\mathbb{E}_{q(\mathbf{x}_0)}[\log\, p_\theta(\mathbf{x}_0)]
\end{aligned}
$$

4. It is also straightforward to get the same result using Jensen's inequality. **Derive it as a homework.**

1. To convert each term in the equation to be analytically computable, the objective can be further rewritten to be a combination of several KL-divergence and entropy terms: See Appendix B of (Sohl-Dickstein et al. 2015).

$$
\begin{aligned}
L_{VLB} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \\
&= \mathbb{E}_q\left[\log \frac{\prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=1}^{T} \log \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} + \log \frac{q(\mathbf{x}_1 \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T} \log \left(\frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} \times \frac{q(\mathbf{x}_t \mid \mathbf{x}_0)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}\right) + \log \frac{q(\mathbf{x}_1 \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} + \sum_{t=2}^{T} \log \frac{q(\mathbf{x}_t \mid \mathbf{x}_0)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1 | \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}\right]
\end{aligned}
$$

1. By simplifying the equation, we obtain

$$
\begin{aligned}
L_{VLB} &= \mathbb{E}_q \left[-\log\, p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} + \sum_{t=2}^{T} \log \frac{q(\mathbf{x}_t \mid \mathbf{x}_0)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1 | \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)} \right] \\
&= \mathbb{E}_q \left[-\log\, p_\theta(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)} + \log \frac{q(\mathbf{x}_T \mid \mathbf{x}_0)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1 \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)} \right] \\
&= \mathbb{E}_q \left[\log \frac{q(\mathbf{x}_T \mid \mathbf{x}_0)}{p_\theta(\mathbf{x}_T)} + \sum_{t=2}^{T} \log \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)} - \log\, p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) \right] \\
&= \mathbb{E}_q \left[\underbrace{\mathrm{D}_{KL}(\, q(\mathbf{x}_T \mid \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_T))}_{L_T} \right] \\
&+ \mathbb{E}_q \left[\sum_{t=2}^{T} \underbrace{\mathrm{D}_{KL}(\, q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))}_{L_{t-1}} \right] \mathbb{E}_q \left[\underbrace{-\log\, p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}_{L_0} \right]
\end{aligned}
$$

2. Consider each component in the variational lower bound loss

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

Training the model

1. Every KL term in L_{VLB} (except for L_0) compares two Gaussian distributions and therefore they can be computed in closed form instead of high variance Monte Carlo estimates.
2. L_T is constant and can be ignored during training because $q(\mathbf{x})$ has no learnable parameters and $\mathbf{x}_T$ is a Gaussian noise.
3. L_0 was modeled using a separate discrete decoder derived from $\mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \boldsymbol{\Sigma}_\theta(\mathbf{x}_1, 1))$ (Ho, Jain, and Abbeel 2020).
4. We need to learn a neural network to approximate the conditioned probability distributions in the reverse diffusion process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$.
5. Using $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$, we would like to train $\boldsymbol{\mu}_\theta$ to predict $\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$.
6. Since $\mathbf{x}_t$ is available as input at training time, we can reparameterize the Gaussian noise term instead to make it predict $\boldsymbol{\epsilon}_t$ from the input $\mathbf{x}_t$ at time step:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)$$

$$\mathbf{x}_{t-1} = \mathcal{N}\left(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)\right)$$

1. For estimating $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})$, the loss term L_t is parameterized to minimize the difference from $\tilde{\boldsymbol{\mu}}$ as

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

1. Training the diffusion model works better with a simplified objective that ignores the weighting term (Ho, Jain, and Abbeel 2020):

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

2. The final simple objective is $L_{simple} = L_t^{simple} + C$, where C is a constant not depending on θ.

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

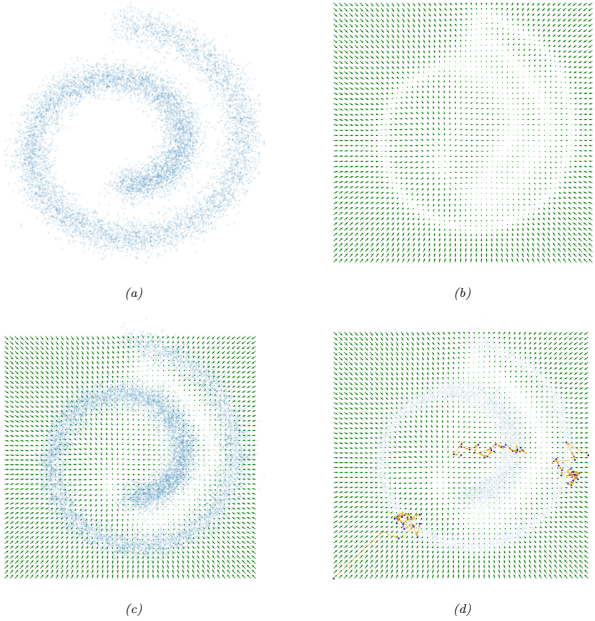
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,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

Score-based generative models

Score-based generative models

1. In EBMs, we discussed how to fit EBMs using score matching.
2. It adjusts the parameters of EBM so that the score function of the model, $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$, matches the score function of the data, $\nabla_{\mathbf{x}} \log p_{\mathrm{d}}(\mathbf{x})$.
3. An alternative to fitting a scalar energy function and computing its score is **to directly learn the score function**.
4. This is called a **score-based generative model** (SGM).
5. We can optimize **score function** $s_\theta(\mathbf{x})$ using
   - basic score matching,
   - denoising score matching, or
   - sliced score matching.

Score-based generative models (example)



(a) Training set. (b) Learned score function trained (MLP with 2 hidden layers, each with 128 hidden units) using the basic score matching. (c) Superposition of learned score function and empirical density. (d) Langevin sampling applied to the learned model.

Score matching problems

1. In general, score matching can have difficulty when there are regions of low data density.
2. Let $p_{\mathrm{d}}(\mathbf{x}) = \pi\, p_0(\mathbf{x}) + (1-\pi)\, p_1(\mathbf{x})$.
3. Let $S_0 = \{\mathbf{x} \mid p_0(\mathbf{x}) > 0\}$ and $S_1 = \{\mathbf{x} \mid p_1(\mathbf{x}) > 0\}$ be the supports of $p_0(\mathbf{x})$ and $p_1(\mathbf{x})$, respectively.
4. When sets S_0 and S_1 are disjoint from each other, the score of $p_{\mathrm{d}}(\mathbf{x})$ is

$$\nabla_{\mathbf{x}} \log p_{\mathrm{d}}(\mathbf{x}) = \begin{cases} \nabla_{\mathbf{x}} \log p_0(\mathbf{x}) & \mathbf{x} \in S_0 \\ \\ \nabla_{\mathbf{x}} \log p_1(\mathbf{x}) & \mathbf{x} \in S_1 \end{cases}$$

5. The score, $\nabla_{\mathbf{x}} \log p_{\mathrm{d}}(\mathbf{x})$, does not depend on the weight π.
6. Hence, score matching cannot correctly recover the true distribution.
7. As a result, Langevin sampling will have difficulty traversing between modes.

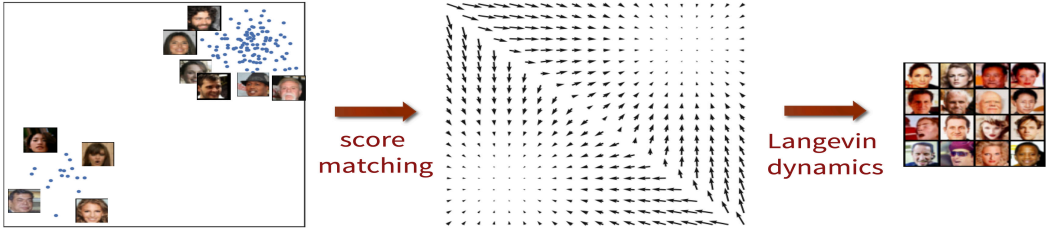
Adding noise at multiple scales

1. We can overcome this difficulty by perturbing training data with different scales of noise as

$$q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}} \mid \mathbf{x}, \sigma^2 \mathbf{I})$$

$$q_\sigma(\tilde{\mathbf{x}}) = \int p_{\text{d}}(\mathbf{x})\, q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) d\mathbf{x}$$

2. For a large noise perturbation, different modes are connected due to added noise, and the estimated weights between them are therefore accurate.
3. For a small noise perturbation, different modes are more disconnected, but the noise-perturbed distribution is closer to the original unperturbed data distribution.



1. The score will be approximated by a deep neural network $s_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log\, p_{\mathrm{d}}(\mathbf{x})$, where $s_\theta(\mathbf{x}) : \mathbb{R}^D \mapsto \mathbb{R}^D$.
2. The objective is to minimize the expectation of the L_2 norm of the difference between the true score and the approximated one as

$$\theta^* = \arg\min_\theta \mathbb{E}_{p_{\mathrm{d}}(\mathbf{x})}\left[\|\nabla_{\mathbf{x}} \log\, p_\theta(\mathbf{x}) - s_\theta(\mathbf{x})\|_2^2\right]$$

3. This requires to have access to the score ground truth and thus makes the **optimization infeasible** as we do not have them.
4. It has been shown that the objective is equivalent to

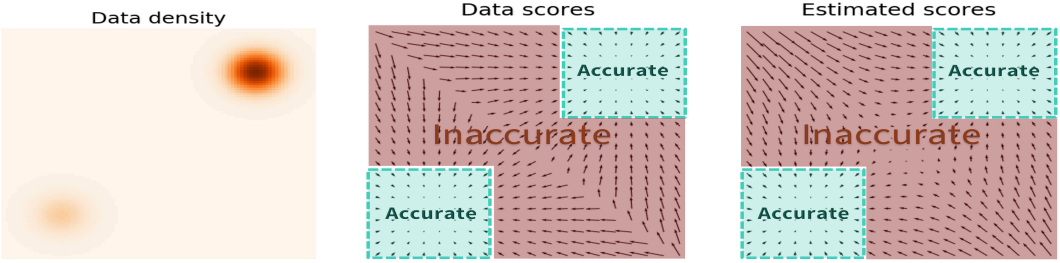
$$\theta^* = \arg\min_\theta \mathbb{E}_{p_{\mathrm{d}}(\mathbf{x})}\left[\mathrm{tr}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x})) + \left\|\frac{1}{2}\nabla_{\mathbf{x}} s_\theta(\mathbf{x})\right\|_2^2\right]$$

5. This can be solved by typical gradient descent algorithms coupled with automatic differentiation, it is computationally exorbitant as it requires computing the Jacobian of $s_\theta(\mathbf{x})$ with respect to $\mathbf{x}$.
6. Several techniques emerged to deal with this problem such as denoising score matching or sliced score matching.

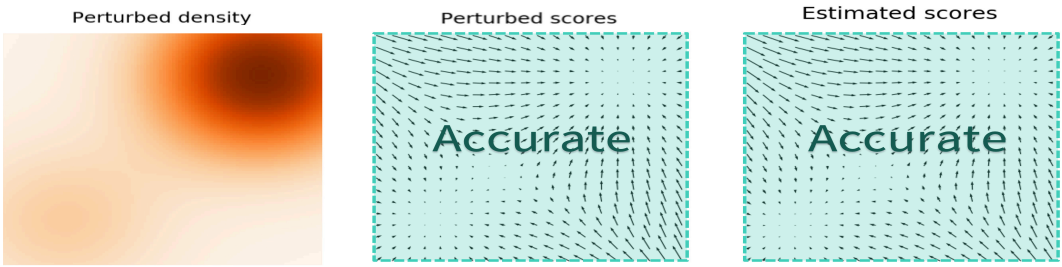
1. The score estimation $s_\theta(\mathbf{x})$ is trained by minimizing an expected loss over the true distribution

$$\begin{aligned}\theta^* &= \arg\min_\theta \mathbb{E}_{p_{\mathrm{d}}(\mathbf{x})}\left[\|\nabla_{\mathbf{x}} \log\, p_{\mathrm{d}}(\mathbf{x}) - s_\theta(\mathbf{x})\|_2^2\right] \\ &= \arg\min_\theta \int p_{\mathrm{d}}(\mathbf{x})\|\nabla_{\mathbf{x}} \log\, p_{\mathrm{d}}(\mathbf{x}) - s_\theta(\mathbf{x})\|_2^2 d\mathbf{x}\end{aligned}$$

2. In practice it consists in randomly sampling a sample of the data set and computing the loss afterwards.
3. The data set does not fully fill the space in which its samples live but only a tiny fraction of it. Hence, we have inaccurate score estimation.



1. How can we bypass the difficulty of accurate score estimation in regions of low data density?
2. When the noise magnitude is sufficiently large, it can populate low data density regions to improve the accuracy of estimated scores.

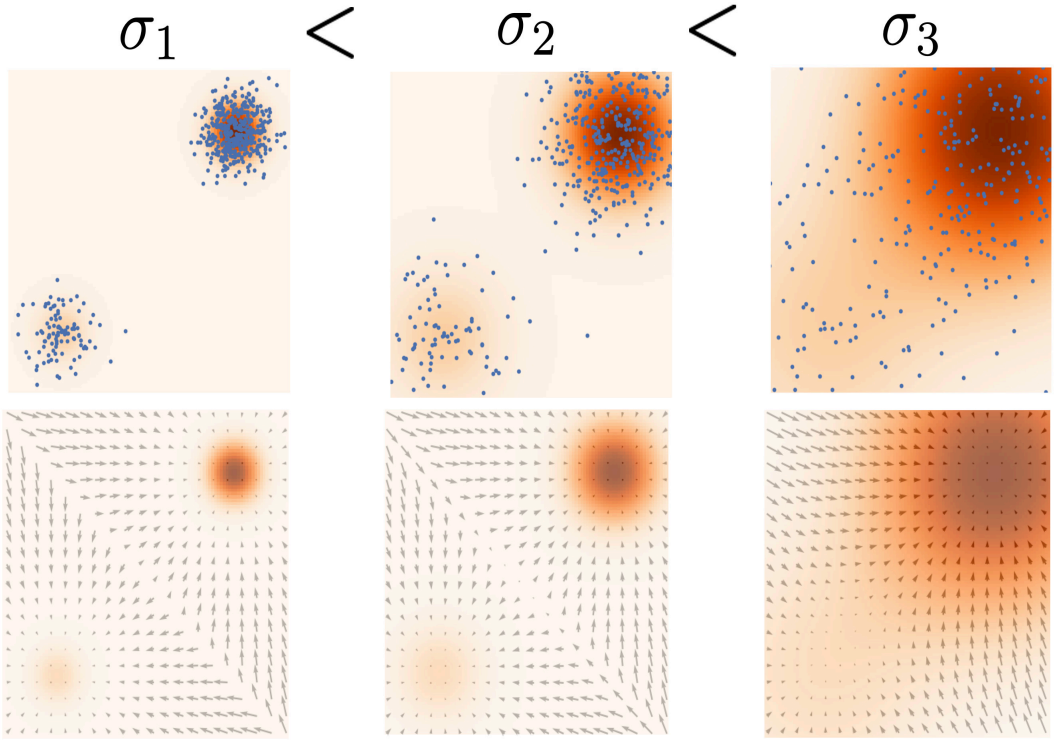


3. How do we choose an appropriate noise scale for the perturbation process?
4. Larger noise can obviously cover more low density regions for better score estimation, but it **over-corrupts the data** and alters it significantly from the original distribution.
5. Smaller noise **causes less corruption** of the original data distribution, but **does not cover the low density regions** as well as we would like.

1. To achieve the best of both worlds, multiple scales of noise perturbations simultaneously are used.
2. Let we always perturb the data with isotropic Gaussian noise, and let there be a total of L increasing standard deviations $\sigma_1 < \sigma_2 < \ldots < \sigma_L$.
3. We first perturb the data distribution $p_{\mathrm{d}}(\mathbf{x})$ with each of the Gaussian noise $\mathcal{N}(0, \sigma_i^2 \mathbf{I})$ for $i = 1, 2, \ldots, L$ to obtain a noise-perturbed distribution

$$p_{\sigma_i}(\mathbf{x}) = \int p_{\mathrm{d}}(\mathbf{y}) \mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma_i^2 \mathbf{I}) d\mathbf{y}$$

4. We can easily draw samples from $p_{\sigma_i}(\mathbf{x})$ by sampling $\mathbf{x} \sim p_{\mathrm{d}}(\mathbf{x})$ and computing $\mathbf{x} + \sigma_i \mathbf{z}$ with $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
5. Next, estimate the score function of each noise-perturbed distribution, $\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$, by training a Noise Conditional Score-Based Network, when parameterized with a neural network) with score matching, such that $s_\theta(\mathbf{x}, i) \approx \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$ for all $i = 1, 2, \ldots, L$.


σ_1
<
σ_2
<
σ_3

Noise Conditional Score-Based Model

1. The training objective for $s_\theta(\mathbf{x}, i)$ is a weighted sum of Fisher divergences for all noise scales.

$$\theta^* = \arg\min_\theta \sum_{i=1}^{L} \lambda(i)\, \mathbb{E}_{p_{\sigma_i}(\mathbf{x})}\left[\|\nabla_{\mathbf{x}} \log\, p_{\sigma_i}(\mathbf{x}) - s_\theta(\mathbf{x}, i)\|_2^2\right]$$

2. where $\lambda(i) \in \mathbb{R}_+$ is a positive weighting function, often chosen to be $\lambda(i) = \sigma_i^2$.
3. This objective function can be optimized with score matching, exactly as in optimizing the naive score-based model $s_\theta(\mathbf{x})$.

Noise Conditional Score-Based Model

Continuous time models using differential equations

Introduction

1. Adding multiple noise scales is critical to the success of score-based generative models.
2. By generalizing the number of noise scales to infinity, we obtain
   - higher quality samples,
   - exact log-likelihood computation, and
   - controllable generation.
3. When the number of noise scales approaches infinity, we essentially perturb the data distribution with continuously growing levels of noise.
4. In this case, the noise perturbation procedure is a continuous-time stochastic process.

1. Consider the following ODE

$$\frac{d\mathbf{x}_t}{dt} = f(\mathbf{x}_t, t)$$

with some initial condition $\mathbf{x}_0$.

2. We can solve this ODE using numerical methods. For example, Euler's method starts from $t = 0$ and proceeding to $t = 1$ with step Δt.

$$\mathbf{x}_{t+\Delta t} - \mathbf{x}_t = f(\mathbf{x}_t, t) \times \Delta t$$

3. Sometimes, it necessary to run from $t = 1$ to $t = 0$. By applying Euler's method backwards, we obtain

$$\mathbf{x}_t = \mathbf{x}_{t+\Delta t} - f(\mathbf{x}_{t+\Delta t}, t + \Delta t) \times \Delta t$$

4. In general, we can think SDEs as ODEs whose trajectories are random and distributed according to $p_t(\mathbf{x})$ at each t.

1. SEDs could be defined as

$$d\mathbf{x}_t = \underbrace{f(\mathbf{x}_t, t)}_{\textit{Drift}}\, dt + \underbrace{g(t)}_{\textit{Diffusion}}\, dw(t)$$

2. $w(t)$ is a standard Wiener process.
3. $f(\mathbf{x}_t, t) \in \mathbb{R}^d_+$ and $g(t) \in \mathbb{R}$.
4. An important property of this SDE is the existence of a corresponding ODE whose solutions follow the same distribution if we start with data point $\mathbf{x}_0$.
5. The SDE $d\mathbf{x} = e^t d\mathbf{w}$ perturbs data with a Gaussian noise of mean zero and exponentially growing variance, such as

$$\mathcal{N}(0, \sigma_1^2\mathbf{I}), \mathcal{N}(0, \sigma_2^2\mathbf{I}), \ldots, \mathcal{N}(0, \sigma_L^2\mathbf{I})$$

when

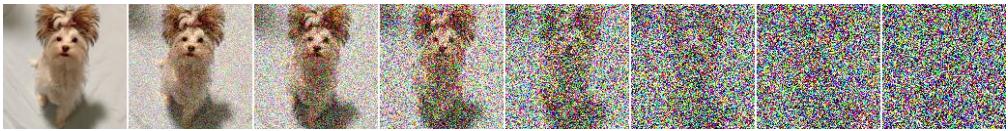
$$\sigma_1 < \sigma_2 < \cdots < \sigma_L$$

Wiener process

1. The Wiener process is a real-valued continuous-time stochastic process that describes the mathematical properties of the one-dimensional Brownian motion.
2. The Wiener process w_t is characterized by the following properties:
   - $w_0 = 0$
   - w has independent increments:

     for every $t > 0$ and $u \geq 0$, $w_{t+u} - w_t$ are independent of the past values w_s for all $s < t$.
   - w has Gaussian increments: $(w_{t+u} - w_t) \sim \mathcal{N}(0, u)$.
   - w_t is almost surely continuous in t.
3. The Wiener process also called Brownian noise.

$$\mathbf{x}(0) \longrightarrow \mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w} \longrightarrow \mathbf{x}(T)$$



$$\mathbf{x}(0) \longleftarrow \mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}} \longleftarrow \mathbf{x}(T)$$

Estimating the reverse SDE

1. Solving reverse SDE requires knowledge of $p_T(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$ and $\nabla_{\mathbf{x}} \log\, p_t(\mathbf{x}_t)$.
2. The $p_T(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$ is fully tractable.
3. To estimate $\nabla_{\mathbf{x}} \log\, p_t(\mathbf{x}_t)$, a **Time-Dependent Score-Based Model** $\mathbf{s}_\theta(\mathbf{x}, t)$ is trained such that $\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log\, p_t(\mathbf{x}_t)$.
4. This is analogous to the noise-conditional score-based model used for finite noise scales.
5. The objective for $\mathbf{s}_\theta(\mathbf{x}, t)$ is a continuous weighted combination of Fisher divergences,

$$\theta^* = \arg\min_\theta\, \mathbb{E}_{t \in U(0,T)} \Big[\mathbb{E}_{p_t(\mathbf{x})} \Big[\lambda(t) \| \nabla_{\mathbf{x}} \log\, p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x}, t) \|_2^2 \Big] \Big]$$

Forward diffusion SDE

1. Consider a diffusion process with noise level β_t.
2. If noise levels β_t are small enough and the number of steps are large enough, we can replace β_t with an infinitesimal function $\beta(t)\Delta t$ such that at each step, instead of moving one unit forward in time, we move Δt units.
3. With this approximation, we have:

$$\begin{aligned}\mathbf{x}_t &= \sqrt{1-\beta_t}\ \mathbf{x}_{t-1} + \sqrt{\beta_t}\ \mathcal{N}(0, \mathbf{I}) \\ &= \sqrt{1-\beta(t)\Delta t}\ \mathbf{x}_{t-1} + \sqrt{\beta(t)\Delta t}\ \mathcal{N}(0, \mathbf{I})\end{aligned}$$

4. If Δt is small, we can approximate the first term with the first-order Taylor series expansion to get

$$\mathbf{x}_t \approx \mathbf{x}_{t-1} - \frac{\beta(t)\Delta t}{2}\mathbf{x}_{t-1} + \sqrt{\beta_t \Delta t}\ \mathcal{N}(0, \mathbf{I})$$

5. **Recall:** The first-order Taylor series expansion of $f(x) = \sqrt{1-x}$ is

$$f(x) = 1 - \frac{1}{2}x$$

1. For small Δt we have

$$\frac{\mathbf{x}_t - \mathbf{x}_{t-1}}{\Delta t} \approx -\frac{\beta(t)}{2}\mathbf{x}_{t-1} + \frac{\sqrt{\beta_t}}{\sqrt{\Delta t}}\,\mathcal{N}(0, \mathbf{I})$$

2. Now, We can now switch to the continuous time limit, and write this as the following stochastic differential equation:

$$\frac{d\mathbf{x}(t)}{dt} = -\frac{1}{2}\beta(t)\mathbf{x}(t) + \sqrt{\beta(t)}\frac{dw(t)}{dt}$$

where $w(t)$ represents a standard Wiener process.

3. In general, we can write these SDEs as

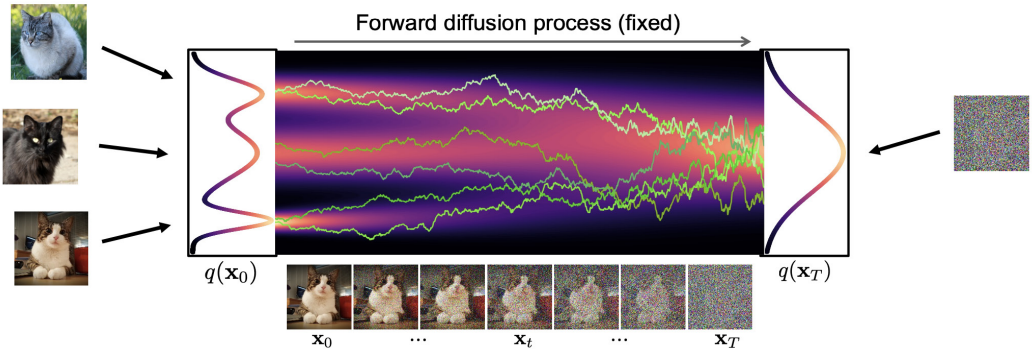
$$d\mathbf{x} = \underbrace{f(\mathbf{x}, t)}_{\text{Drift coefficient}}\,dt + \underbrace{g(t)}_{\text{Diffusion coefficient}}\,dw$$

4. It can be shown that SDE corresponding to DDPMs in $T \to \infty$ limit is (Song et al. 2021).

$$d\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}dt + \sqrt{\beta(t)}dw$$

Forward diffusion SDE

1. Yellow lines are sample paths from the SDE.
2. Heat map represents the marginal distribution.



3. The drift term pulls towards mode.
4. The diffusion term injects noise.

1. To generate samples from this model, we need to be able to reverse the SDE.
2. Any forwards SDE (in the mentioned form) can be reversed to get the following reverse-time SDE:

$$d\mathbf{x} = \left[f(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log\, q_t(\mathbf{x})\right] dt + g(t) d\overline{w}$$

   where $\overline{w}$ is the standard Wiener process when time flows backwards.
3. In the case of the DDPM, the reverse SDE has the following form:

$$d\mathbf{x}_t = \left[-\frac{1}{2}\beta(t)\mathbf{x}_t - \beta(t)\nabla_{\mathbf{x}_t} \log\, q_t(\mathbf{x}_t)\right] dt + \sqrt{\beta(t)} d\overline{w}_t$$

4. To estimate the score function, we can use denoising score matching to get

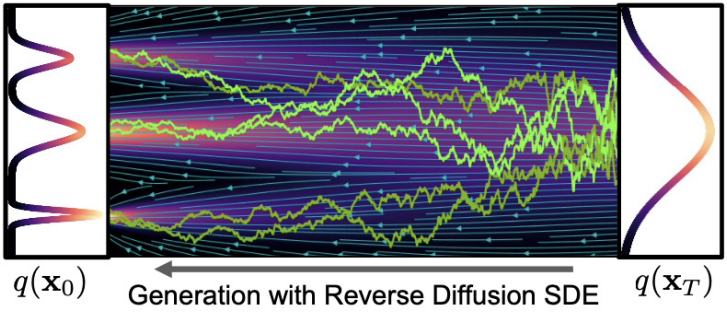
$$\nabla_{\mathbf{x}_t} \log\, q_t(\mathbf{x}_t) \approx s_\theta(\mathbf{x}_t, t)$$

1. Then, SDE becomes

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t)[\mathbf{x}_t + 2s_\theta(\mathbf{x}_t, t)]dt + \sqrt{\beta(t)}d\overline{w}_t$$

2. After fitting the score network, we can sample from it

$$\mathbf{x}_{t-1} = \mathbf{x}_t + \frac{1}{2}\beta(t)[\mathbf{x}_t + 2s_\theta(\mathbf{x}_t, t)]\Delta t + \sqrt{\beta(t)\Delta t}\mathcal{N}(\mathbf{0}, \mathbf{I})$$

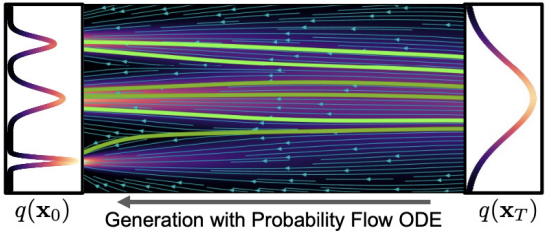


1. Instead of adding Gaussian noise at every step, we can sample the initial state, and then let it evolve deterministically over time according to the ODE

$$d\mathbf{x} = \underbrace{\left[f(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log\, p_t(\mathbf{x}) \right]}_{h(\mathbf{x},t)} dt$$

2. This is called the probability flow ODE. We can compute the state at any moment

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t h(\mathbf{x}, \tau) d\tau$$



3. The induced distribution over paths will have the same marginals as the SDE model.

Reverse diffusion ODE

1. We can derive the probability flow ODE from the reverse-time SDE

$$d\mathbf{x}_t = \left[f(\mathbf{x}, t) - g(t)^2 s_\theta(\mathbf{x}_t, t)\right] dt$$

2. By setting $f(\mathbf{x}, t) = -\frac{1}{2}\beta(t)$ and $g(t) = \sqrt{\beta(t)}$, we obtain

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t)[\mathbf{x}_t + s_\theta(\mathbf{x}_t, t)]dt$$

3. A simple way to solve this ODE is to use Euler's method

$$\mathbf{x}_{t-1} = \mathbf{x}_t + \frac{1}{2}\beta(t)[\mathbf{x}_t + s_\theta(\mathbf{x}_t, t)]\Delta t$$

4. In practice one can get better results using higher-order ODE solvers, such as Heun's method.

Comparison of the SDE and ODE approach

1. We can see the connection between ODE and SDE methods by rewriting the SDE

$$
\begin{aligned}
d\mathbf{x} &= \left[f(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log\, q_t(\mathbf{x})\right] dt + g(t) d\overline{w} \\
&= \underbrace{-\frac{1}{2}\beta(t)[\mathbf{x}_t + s_\theta(\mathbf{x}_t, t)]dt}_{\text{Probability flow ODE}} \underbrace{- \frac{1}{2}\beta(t) s_\theta(\mathbf{x}_t, t)dt + \sqrt{\beta(t)} d\overline{w}}_{\text{Langevin diffusion SDE}}
\end{aligned}
$$

2. Generative Diffusion SDE has the following properties:

   Pros: Continuous noise injection can help to compensate errors during diffusion process.

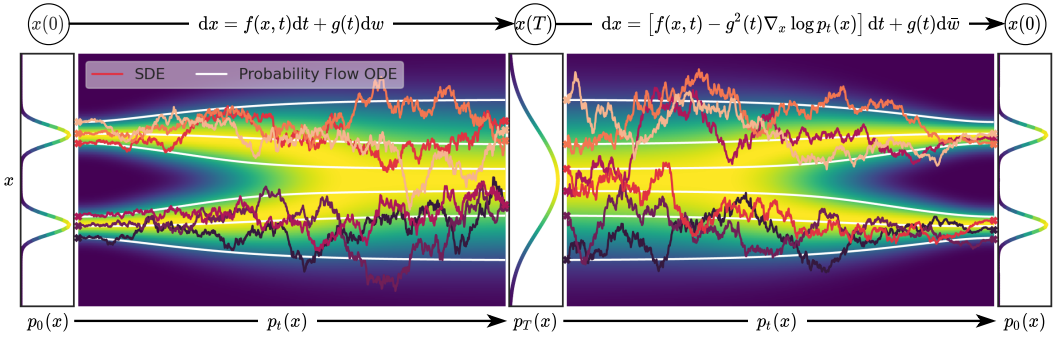
   Cons: Often slower, beacuse the stochastic terms themselves require fine discretization during solve.

3. Probability Flow ODE has the following properties:

   Pros: Can leverage fast ODE solvers. Best when targeting very fast sampling.

   Cons: No stochastic error correction, often slightly lower performance than stochastic sampling

ODE results


$x(0)$
$\mathrm{d}x = f(x,t)\mathrm{d}t + g(t)\mathrm{d}w$
$x(T)$
$\mathrm{d}x = \left[f(x,t) - g^2(t)\nabla_x \log p_t(x)\right]\mathrm{d}t + g(t)\mathrm{d}\bar{w}$
$x(0)$
SDE
Probability Flow ODE
x
$p_0(x)$
$p_t(x)$
$p_T(x)$
$p_t(x)$
$p_0(x)$

Comparison of the SDE and ODE approach

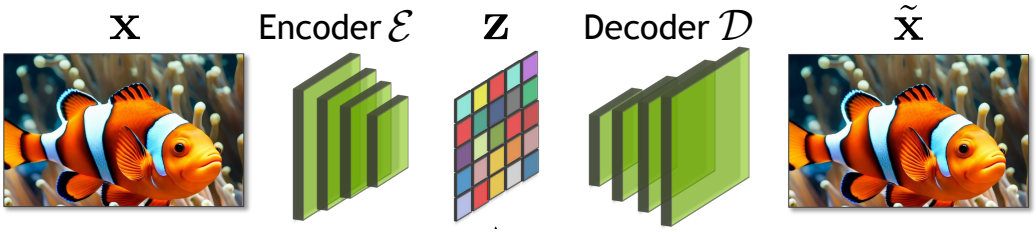
Latent Diffusion Models

Latent Diffusion Models

1. DPMs belong to the class of likelihood-based models.
2. In image synthesis, DPMs typically operate directly in pixel space, and therefore optimizing a high-resolution image generating DPM is GPU intensive.
3. Inference is also expensive due to sequential evaluations.
4. Latent diffusion models introduced to (Rombach et al. 2022)
   - reduce the computational complexity to enable DPM training on limited computational resources
   - retaining their quality and flexibility

The key idea of LDMs is to separate the training into two phases:

1. **Perceptual image compression:** This is the first stage of training in which an autoencoder is trained which provides a lower-dimensional representational space which is perceptually equivalent to the data space.
2. **Latent Diffusion:** In this second phase, a DPM is trained on the learned lower-dimensional latent space from the trained autoencoder, instead of the high-dimensional pixel space.



Summary

Summary

1. Pros of diffusion models
   - Capacity for producing high-quality outputs that often surpass GANs in terms of realism and diversity
   - Skilled at handling complex distributions, making them versatile for various applications.
   - A more reliable training process than GANs, avoiding the issue of mode collapse.
2. Cons of diffusion models
   - Require significant resources for training and generation, which can limit accessibility.
   - Generating data through iterative denoising is much more time-consuming compared to direct generation methods used by GANs.

Comparison of deep generative models

Model	Density	Sampling	Training	Latents	Architecture
PGM-D	Exact, fast	Fast	MLE	Optional	Sparse DAG
PGM-U	Approx, slow	Slow	MLE-Approx	Optional	Sparse Graph
VAE	LB, fast	Fast	MLE-LB	$\mathbb{R}^L$	Encoder-Decoder
ARM	Exact, fast	Slow	MLE	None	Sequential
Flows	Exact, slow/fast	Slow	MLE	$\mathbb{R}^D$	Invertible
EBM	Approx, slow	Slow	MLE-Approx	Optional	Discriminative
Diffusion	LB	Slow	MLE-LB	$\mathbb{R}^D$	Encoder-Decoder
GAN	N/A	Fast	Min-Max	$\mathbb{R}^L$	Generator-Discriminator

References

Reading

1. Chapter 25 of Probabilistic Machine Learning: Advanced Topics (Murphy 2023).
2. Chapter 9 of Deep Generative Modeling (Tomczak 2024).

References i

- Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems*.
- Murphy, Kevin P. (2023). *Probabilistic Machine Learning: Advanced Topics*. The MIT Press.
- Rombach, Robin et al. (2022). "High-Resolution Image Synthesis with Latent Diffusion Models". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10674–10685.
- Sohl-Dickstein, Jascha et al. (2015). "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *International Conference on Machine Learning*. Vol. 37, pp. 2256–2265.
- Song, Yang et al. (2021). "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*.
- Tomczak, Jakub M. (2024). *Deep Generative Modeling*. Springer.

Questions?