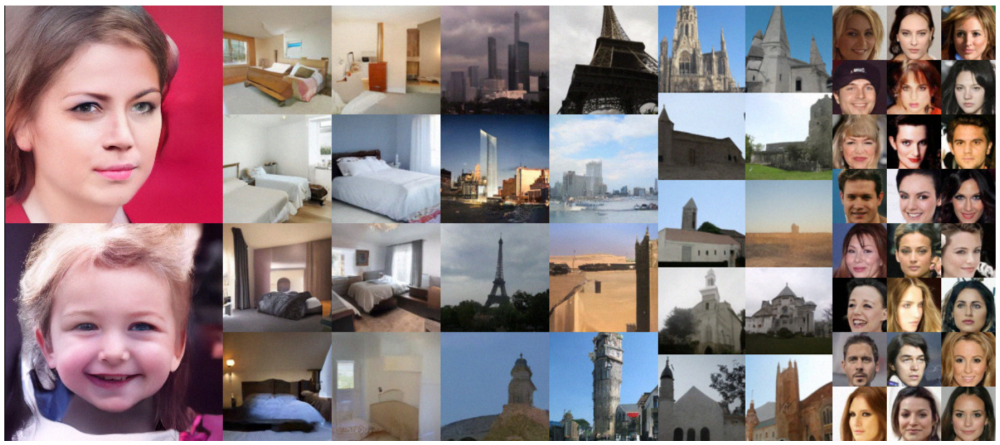
Score-Based Generative Modeling

Please note that this is an optional notebook meant to introduce more advanced concepts. If you're up for a challenge, take a look and don't worry if you can't follow everything. There is no code to implement—only some cool code for you to learn and run!

Goals

This is a hitchhiker's guide to score-based generative models, a family of approaches based on estimating gradients of the data distribution. They have obtained high-quality samples comparable to GANs (like below, figure from this paper) without requiring adversarial training, and are considered by some to be the new contender to GANs.



Introduction

Score and Score-Based Models

Given a probablity density function $p(\mathbf{x})$, we define the *score* as

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}).$$

As you might guess, score-based generative models are trained to estimate $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. Unlike likelihood-based models such as flow models or autoregressive models, score-based models do not have to be normalized and are easier to parameterize. For example, consider a non-normalized statistical model $p_\theta(\mathbf{x}) = \frac{e^{-E_\theta(\mathbf{x})}}{Z_\theta}$, where $E_\theta(\mathbf{x}) \in \mathbb{R}$ is called the energy function and Z_θ is an unknown normalizing constant that makes $p_\theta(\mathbf{x})$ a proper probability density function. The energy function is typically parameterized by a flexible neural network. When training it as a likelihood model, we need to know the normalizing constant Z_θ by computing complex high-dimensional integrals, which is typically intractable. In constrast, when computing its score, we obtain $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x})$ which does not require computing the normalizing constant Z_θ.

In fact, any neural network that maps an input vector $\mathbf{x} \in \mathbb{R}^d$ to an output vector $\mathbf{y} \in \mathbb{R}^d$ can be used as a score-based model, as long as the output and input have the same dimensionality. This yields huge flexibility in choosing model architectures.

Perturbing Data with a Diffusion Process

In order to generate samples with score-based models, we need to consider a diffusion process that corrupts data slowly into random noise. Scores will arise when we reverse this diffusion process for sample generation. You will see this later in the notebook.

A diffusion process is a stochastic process similar to Brownian motion. Their paths are like the trajectory of a particle submerged in a flowing fluid, which moves randomly due to unpredictable collisions with other particles. Let $\{\mathbf{x}(t) \in \mathbb{R}^d\}_{t=0}^T$ be a diffusion process, indexed by the continuous time variable $t \in [0, T]$. A diffusion process is governed by a stochastic differential equation (SDE), in the following form

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w},$$

where $\mathbf{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is called the *drift coefficient* of the SDE, $g(t) \in \mathbb{R}$ is called the *diffusion coefficient*, and $\mathbf{w}$ represents the standard Brownian motion. You can understand an SDE as a stochastic generalization to ordinary differential equations (ODEs). Particles moving according to an SDE not only follows the deterministic drift $\mathbf{f}(\mathbf{x}, t)$, but are also affected by the random noise coming from $g(t)d\mathbf{w}$.

For score-based generative modeling, we will choose a diffusion process such that $\mathbf{x}(0) \sim p_0$, where we have a dataset of i.i.d. samples, and $\mathbf{x}(T) \sim p_T$, for which we have a tractable form to sample from.

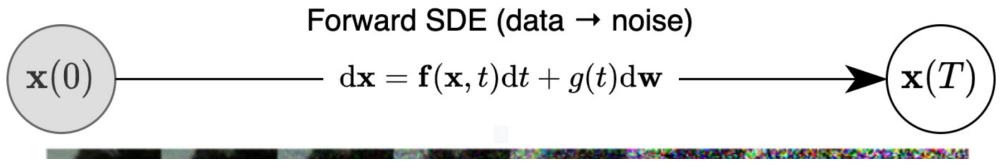
Reversing the Diffusion Process Yields Score-Based Generative Models

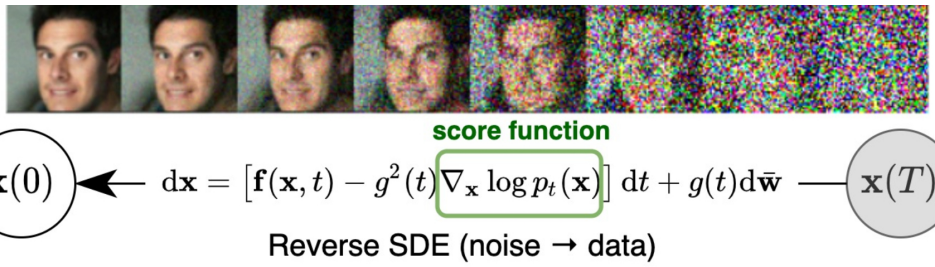
By starting from a sample from p_T and reversing the diffusion process, we will be able to obtain a sample from p_{data}. Crucially, the reverse process is a diffusion process running backwards in time. It is given by the following reverse-time SDE

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

where $\bar{\mathbf{w}}$ is a Brownian motion in the reverse time direction, and dt here represents an infinitesimal negative time step. Here $p_t(\mathbf{x})$ represents the distribution of $\mathbf{x}(t)$. This reverse SDE can be computed once we know the drift and diffusion coefficients of the forward SDE, as well as the score of $p_t(\mathbf{x})$ for each $t \in [0, T]$.

The overall intuition of score-based generative modeling with SDEs can be summarized in the illustration below





Score Estimation

Based on the above intuition, we can use the time-dependent score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ to construct the reverse-time SDE, and then solve it numerically to obtain samples from p_0 using samples from a prior distribution p_T. We can train a time-dependent score-based model $s_\theta(\mathbf{x}, t)$ to approximate $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, using the following weighted sum of denoising score matching objectives.

$$\min_\theta \mathbb{E}_{t\sim\mathcal{U}(0,T)}[\lambda(t)\mathbb{E}_{\mathbf{x}(0)\sim p_0(\mathbf{x})}\mathbf{E}_{\mathbf{x}(t)\sim p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))}[\|s_\theta(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\|_2^2]],$$

where $\mathcal{U}(0, T)$ is a uniform distribution over $[0, T]$, $p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))$ denotes the transition probability from $\mathbf{x}(0)$ to $\mathbf{x}(t)$, and $\lambda(t) \in \mathbb{R}^+$ denotes a continuous weighting function.

In the objective, the expectation over $\mathbf{x}(0)$ can be estimated with empirical means over data samples from p_0. The expectation over $\mathbf{x}(t)$ can be estimated by sampling from $p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))$, which is efficient when the drift coefficient $\mathbf{f}(\mathbf{x}, t)$ is affine. The weight function $\lambda(t)$ is typically chosen to be inverse proportional to $\mathbb{E}[\|\nabla_{\mathbf{x}} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\|_2^2]$.

Time-Dependent Score-Based Model

There are no restrictions on the network architecture of time-dependent score-based models, except that their output should have the same dimensionality as the input, and they should be conditioned on time.

Several useful tips on architecture choice:

- It usually performs well to use the U-net architecture as the backbone of the score network $s_\theta(\mathbf{x}, t)$,
- We can incorporate the time information via Gaussian random features. Specifically, we first sample $\omega \sim \mathcal{N}(\mathbf{0}, s^2\mathbf{I})$ which is subsequently fixed for the model (i.e., not learnable). For a time step t, the corresponding Gaussian random feature is defined as
$$[\sin(2\pi\omega t); \cos(2\pi\omega t)],$$
where $[\vec{a}; \vec{b}]$ denotes the concatenation of vector $\vec{a}$ and $\vec{b}$. This Gaussian random feature can be used as an encoding for time step t so that the score network can condition on t by incorporating this encoding. We will see this further in the code.
- We can rescale the output of the U-net by $1/\sqrt{\mathbb{E}[\|\nabla_{\mathbf{x}} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\|_2^2]}$. This is because the optimal $s_\theta(\mathbf{x}(t), t)$ has an ℓ_2-norm close to $\mathbb{E}[\|\nabla_{\mathbf{x}} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\|_2]$, and the rescaling helps capture the norm of the true score. Recall that the training objective contains sums of the form
$$\mathbf{E}_{\mathbf{x}(t)\sim p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))}[\|s_\theta(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\|_2^2].$$
Therefore, it is natural to expect that the optimal score model $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))$.
- Use exponential moving average (EMA) of weights when sampling. This can greatly improve sample quality, but requires slightly longer training time, and requires more work in implementation. We do not include this in this tutorial, but highly recommend it when you employ score-based generative modeling to tackle more challenging real problems.

Defining a time-dependent score-based model (double click to expand or collapse)

Training with Weighted Sum of Denoising Score Matching Objectives

Now let's get our hands dirty on training. First of all, we need to specify an SDE that perturbs the data distribution p_0 to a prior distribution p_T. We choose the following SDE

$$d\mathbf{x} = \sqrt{\frac{d[\sigma^2(t)]}{dt}} d\mathbf{w},$$

where $\sigma(t) = \sigma_{\min}(\frac{\sigma_{\max}}{\sigma_{\min}})^t, t \in [0, 1]$. In this case,

$$p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}(\mathbf{x}(t); \mathbf{x}(0), [\sigma^2(t) - \sigma^2(0)]\mathbf{I})$$

and $\lambda(t) \propto \sigma^2(t) - \sigma^2(0)$.

When $\sigma_{\max}$ is large enough, the distribution of p_1 is

$$\int p_0(\mathbf{y})\mathcal{N}(\mathbf{x}; \mathbf{y}, [\sigma_{\max}^2 - \sigma_{\min}^2]\mathbf{I})d\mathbf{y} \approx \mathbf{N}(\mathbf{x}; \mathbf{0}, [\sigma_{\max}^2 - \sigma_{\min}^2]\mathbf{I}),$$

which is easy to sample from.

Intuitively, this SDE captures a continuum of Gaussian perturbations with variance function $\sigma(t)^2 - \sigma^2(0)$, where $\sigma(t)$ is a strictly increasing function that grows exponentially fast. This continuum of perturbations allows us to gradually transfer samples from a data distribution p_0 to a simple Gaussian distribution p_1.

Loss function (double click to expand or collapse)

Training (double click to expand or collapse)

device: cuda

sigma_min: 0.01

sigma_max: 22

n_epochs: 50

batch_size: 32

lr: 1e-4

```
100% 1875/1875 [24:45<00:00, 1.26it/s]

epoch: 0, average loss: 479.2823524576823
100% 1875/1875 [00:44<00:00, 41.75it/s]

epoch: 1, average loss: 191.41421276448568
100% 1875/1875 [00:22<00:00, 83.29it/s]

epoch: 2, average loss: 143.56634580485027
100% 1875/1875 [23:36<00:00, 1.32it/s]

epoch: 3, average loss: 125.4686314066569
100% 1875/1875 [03:47<00:00, 8.23it/s]

epoch: 4, average loss: 115.43633513997396
100% 1875/1875 [03:25<00:00, 9.13it/s]

epoch: 5, average loss: 107.28631278686524
100% 1875/1875 [03:02<00:00, 10.27it/s]

epoch: 6, average loss: 102.80460919799805
100% 1875/1875 [02:39<00:00, 11.74it/s]

epoch: 7, average loss: 98.79836651204427
100% 1875/1875 [02:16<00:00, 13.70it/s]

epoch: 8, average loss: 95.33749889933269
100% 1875/1875 [01:54<00:00, 16.43it/s]

epoch: 9, average loss: 93.04867509765624
100% 1875/1875 [01:31<00:00, 20.52it/s]

epoch: 10, average loss: 90.20109106648763
100% 1875/1875 [01:08<00:00, 27.34it/s]

epoch: 11, average loss: 88.1472403523763
100% 1875/1875 [00:45<00:00, 41.18it/s]

epoch: 12, average loss: 86.72806940511067
100% 1875/1875 [00:22<00:00, 82.53it/s]

epoch: 13, average loss: 84.76441270141602
100% 1875/1875 [19:26<00:00, 1.61it/s]

epoch: 14, average loss: 82.9646538655599
```

Sampling with Numerical SDE Solvers

Recall that for any SDE of the form

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w},$$

the reverse-time SDE is given by

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) + g(t)^2\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]dt + g(t)d\bar{\mathbf{w}}.$$

Since we have chosen the forward SDE to be

$$d\mathbf{x} = \sqrt{\frac{d[\sigma^2(t)]}{dt}}d\mathbf{w},$$

where $\sigma(t) = \sigma_{\min}(\frac{\sigma_{\max}}{\sigma_{\min}})^t$, $t \in [0, 1]$. The reverse-time SDE is given by

$$d\mathbf{x} = -\frac{d[\sigma^2(t)]}{dt}\nabla_{\mathbf{x}} \log p_t(\mathbf{x})dt + \sqrt{\frac{d[\sigma^2(t)]}{dt}}d\bar{\mathbf{w}}.$$

To sample from our time-dependent score-based model $s_\theta(\mathbf{x}, t)$, we can first draw a sample from $p_1 \approx \mathbf{N}(\mathbf{x}; \mathbf{0}, [\sigma^2_{\max} - \sigma^2_{\min}]\mathbf{I})$, and then solve the reverse-time SDE with numerical methods.

Specifically, using our time-dependent score-based model, the reverse-time SDE can be approximated by

$$d\mathbf{x} = -\frac{d[\sigma^2(t)]}{dt}s_\theta(\mathbf{x}, t)dt + \sqrt{\frac{d[\sigma^2(t)]}{dt}}d\bar{\mathbf{w}}$$

Next, one can use numerical methods to solve for the reverse-time SDE, such as the Euler-Maruyama approach. It is based on a simple discretization to the SDE, replacing dt with Δt and $d\mathbf{w}$ with $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, g^2(t)\Delta t\mathbf{I})$. When applied to our reverse-time SDE, we can obtain the following iteration rule

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t + \frac{d[\sigma^2(t)]}{dt}s_\theta(\mathbf{x}_t, t)\Delta t + \sqrt{\frac{d[\sigma^2(t)]}{dt}\Delta t}\mathbf{z}_t,$$

where $\mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Sampling with Predictor-Corrector Methods

Aside from generic numerical SDE solvers, we can leverage special properties of our reverse-time SDE for better solutions. Since we have an estimate of the score of $p_t(\mathbf{x}(t))$ via the score-based model, i.e., $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$, we can leverage score-based MCMC approaches, such as Langevin MCMC, to correct the solution obtained by numerical SDE solvers.

Score-based MCMC approaches can produce samples from a distribution $p(\mathbf{x})$ once its score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ is known. For example, Langevin MCMC operates by running the following iteration rule for $i = 1, 2, \cdots, N$:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon\nabla_{\mathbf{x}} \log p(\mathbf{x}_i) + \sqrt{2\epsilon}\mathbf{z}_i,$$

where $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\epsilon > 0$ is the step size, and $\mathbf{x}_1$ is initialized from any prior distribution $\pi(\mathbf{x}_1)$. When $N \to \infty$ and $\epsilon \to 0$, the final value $\mathbf{x}_{N+1}$ becomes a sample from $p(\mathbf{x})$ under some regularity conditions. Therefore, given $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, we can get an approximate sample from $p_t(\mathbf{x})$ by running several steps of Langevin MCMC, replacing $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ with $s_\theta(\mathbf{x}, t)$ in the iteration rule.

Predictor-Corrector samplers combine both numerical solvers for the reverse-time SDE and the Langevin MCMC approach. In particular, we first apply one step of numerical SDE solver to obtain $\mathbf{x}_{t-\Delta t}$ from $\mathbf{x}_t$, which is called the "predictor" step. Next, we apply several steps of Langevin MCMC to refine $\mathbf{x}_t$, such that $\mathbf{x}_t$ becomes a more accurate sample from $p_{t-\Delta t}(\mathbf{x})$. This is the "corrector" step as the MCMC helps reduce the error of the numerical SDE solver.

Sampling with Numerical ODE Solvers

For any SDE of the form

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w},$$

there exists an associated ordinary differential equation (ODE)

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

such that their trajectories have the same mariginal probability density $p_t(\mathbf{x})$. We call this ODE the *probability flow ODE*.

Therefore, we can start from a sample from p_T, integrate the ODE in the reverse time direction, and then get a sample from $p_0 = p_{\text{data}}$. In particular, for our chosen forward SDE, we can integrate the following SDE from $t = T$ to 0 for sample generation

$$d\mathbf{x} = -\frac{1}{2}\frac{d[\sigma^2(t)]}{dt}s_\theta(\mathbf{x}, t)dt.$$

This can be done using many heavily-optimized black-box ODE solvers provided by packages such as `scipy`.

SDE sampling (double click to expand or collapse)

num_steps: 500

PC sampling (double click to expand or collapse)

signal_to_noise_ratio: 0.15

num_steps: 500

ODE sampling (double click to expand or collapse)

error_tolerance: 1e-5

Sampling (double click to expand or collapse)

device: cuda

sample_batch_size: 64

sampler: pc_sampler

100% 500/500 [00:04<00:00, 103.49it/s]

Likelihood Computation

A by-product of the probability flow ODE formulation is likelihood computation. Suppose we have a differentiable one-to-one mapping $\mathbf{h}$ that transforms a data sample $\mathbf{x} \sim p_0$ to a prior distribution $\mathbf{h}(\mathbf{x}) \sim p_1$. We can compute the likelihood of $p_0(\mathbf{x})$ via the following change-of-variable formula

$$p_0(\mathbf{x}) = p_1(\mathbf{h}(\mathbf{x}))|\det(J_{\mathbf{h}}(\mathbf{x}))|,$$

where $J_{\mathbf{h}}(\mathbf{x})$ represents the Jacobian of the mapping $\mathbf{h}$, and we assume it is efficient to evaluate the likelihood of the prior distribution p_1.

Similarly, an ODE is also a one-to-one mapping from $\mathbf{x}(0)$ to $\mathbf{x}(1)$. For ODEs of the form

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

there exists an instantaneous change-of-variable formula that connects the probability of $p_0(\mathbf{x})$ and $p_1(\mathbf{x})$, given by

$$p_0(\mathbf{x}(0)) = e^{\int_0^1 \operatorname{div}\mathbf{f}(\mathbf{x}(t),t)dt}p_1(\mathbf{x}(1)),$$

where div denotes the divergence function (trace of Jacobian).

In practice, this divergence function can be hard to evaluate for general vector-valued function $\mathbf{f}$, but we can use an unbiased estimator, named Skilling-Hutchinson estimator, to approximate the trace. Let $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The Skilling-Hutchinson estimator is based on the fact that

$$\operatorname{div}\mathbf{f}(\mathbf{x}) = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\boldsymbol{\epsilon}^\intercal J_{\mathbf{f}}(\mathbf{x})\boldsymbol{\epsilon}].$$

Therefore, we can simply sample a random vector $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and then use $\boldsymbol{\epsilon}^\intercal J_{\mathbf{f}}(\mathbf{x})\boldsymbol{\epsilon}$ to estimate the divergence of $\mathbf{f}(\mathbf{x})$. This estimator only requires computing the Jacobian-vector product $J_{\mathbf{f}}(\mathbf{x})\boldsymbol{\epsilon}$, which is typically efficient.

As a result, for our probability flow ODE, we can compute the (log) data likelihood with the following

$$\log p_0(\mathbf{x}(0)) = \log p_1(\mathbf{x}(1)) - \frac{1}{2}\int_0^1 \frac{d[\sigma^2(t)]}{dt}\operatorname{div} s_\theta(\mathbf{x}(t), t)dt.$$

With the Skilling-Hutchinson estimator, we can compute the divergence via

$$\operatorname{div} s_\theta(\mathbf{x}(t), t) = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\boldsymbol{\epsilon}^\intercal J_{s_\theta}(\mathbf{x}(t), t)\boldsymbol{\epsilon}].$$

Afterwards, we can compute the integral with numerical integrators. This gives us an unbiased estimate to the true data likelihood, and we can make it more and more accurate when we run it multiple times and take the average. The numerical integrator requires $\mathbf{x}(t)$ as a function of t, which can be obtained by solving the original probability flow ODE.

Likelihood function (double click to expand or collapse)

Computing likelihood on the dataset (double click to expand or collapse)

device: cuda

0% 8/1875 [01:51<7:13:07, 13.92s/it]

```
bpd (running average): 2.3032925728249083
bpd (running average): 2.308977756601495
bpd (running average): 2.377123575058273
bpd (running average): 2.389343318526081
bpd (running average): 2.370500455404197
bpd (running average): 2.378673716580655
bpd (running average): 2.382435482845119
bpd (running average): 2.4092268406606396
```

Further Resources

If you're interested in learning more about score-based generative models, the following papers would be a good start:

- Yang Song, and Stefano Ermon. "[Generative modeling by estimating gradients of the data distribution.]" Advances in Neural Information Processing Systems. 2019.
- Yang Song, and Stefano Ermon. "[Improved Techniques for Training Score-Based Generative Models.]" Advances in Neural Information Processing Systems. 2020.
- Jonathan Ho, Ajay Jain, and Pieter Abbeel. "[Denoising diffusion probabilistic models.]" Advances in Neural Information Processing Systems. 2020.