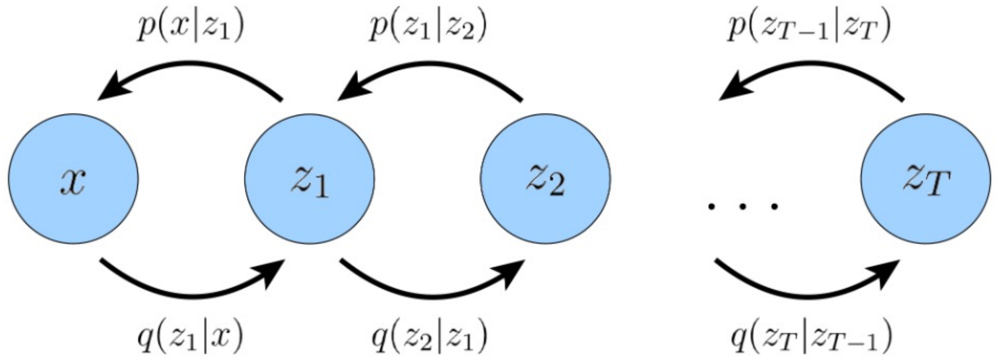
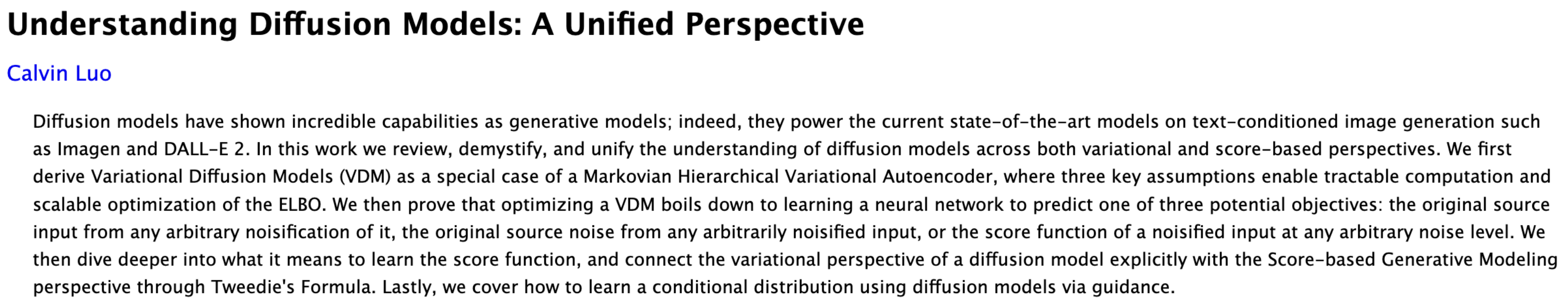
An introduction to diffusion models

Suppose you've seen Stable Diffusion

https://stability.ai/

And you want to build diffusion from scratch



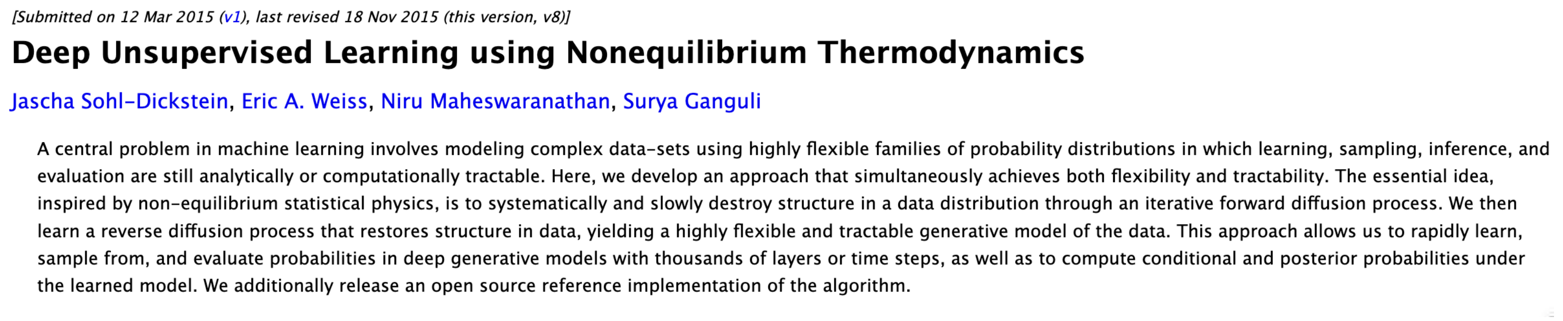


Understanding Diffusion Models: A Unified Perspective

Calvin Luo

Diffusion models have shown incredible capabilities as generative models; indeed, they power the current state-of-the-art models on text-conditioned image generation such as Imagen and DALL-E 2. In this work we review, demystify, and unify the understanding of diffusion models across both variational and score-based perspectives. We first derive Variational Diffusion Models (VDM) as a special case of a Markovian Hierarchical Variational Autoencoder, where three key assumptions enable tractable computation and scalable optimization of the ELBO. We then prove that optimizing a VDM boils down to learning a neural network to predict one of three potential objectives: the original source input from any arbitrary noisification of it, the original source noise from any arbitrarily noisified input, or the score function of a noisified input at any arbitrary noise level. We then dive deeper into what it means to learn the score function, and connect the variational perspective of a diffusion model explicitly with the Score-based Generative Modeling perspective through Tweedie's Formula. Lastly, we cover how to learn a conditional distribution using diffusion models via guidance.

But wait. How does diffusion work?

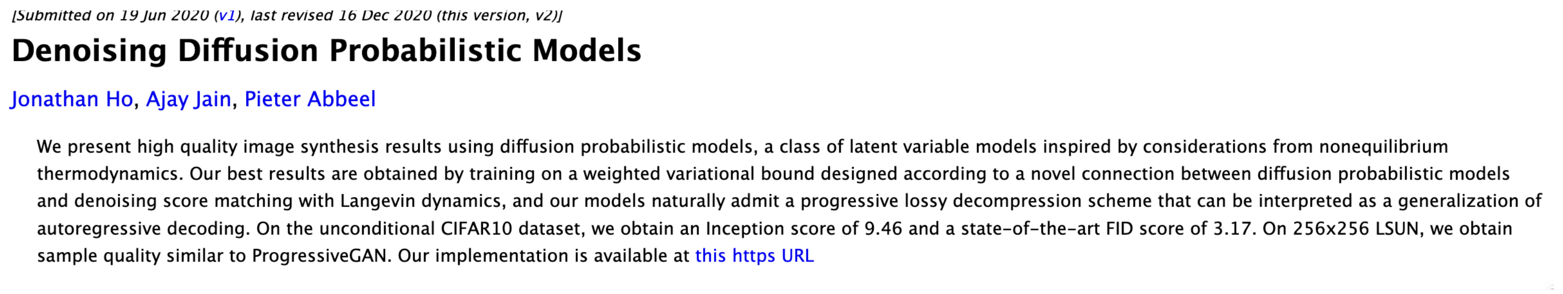


[Submitted on 12 Mar 2015 (v1), last revised 18 Nov 2015 (this version, v8)]

Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, Surya Ganguli

A central problem in machine learning involves modeling complex data-sets using highly flexible families of probability distributions in which learning, sampling, inference, and evaluation are still analytically or computationally tractable. Here, we develop an approach that simultaneously achieves both flexibility and tractability. The essential idea, inspired by non-equilibrium statistical physics, is to systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process. We then learn a reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data. This approach allows us to rapidly learn, sample from, and evaluate probabilities in deep generative models with thousands of layers or time steps, as well as to compute conditional and posterior probabilities under the learned model. We additionally release an open source reference implementation of the algorithm.

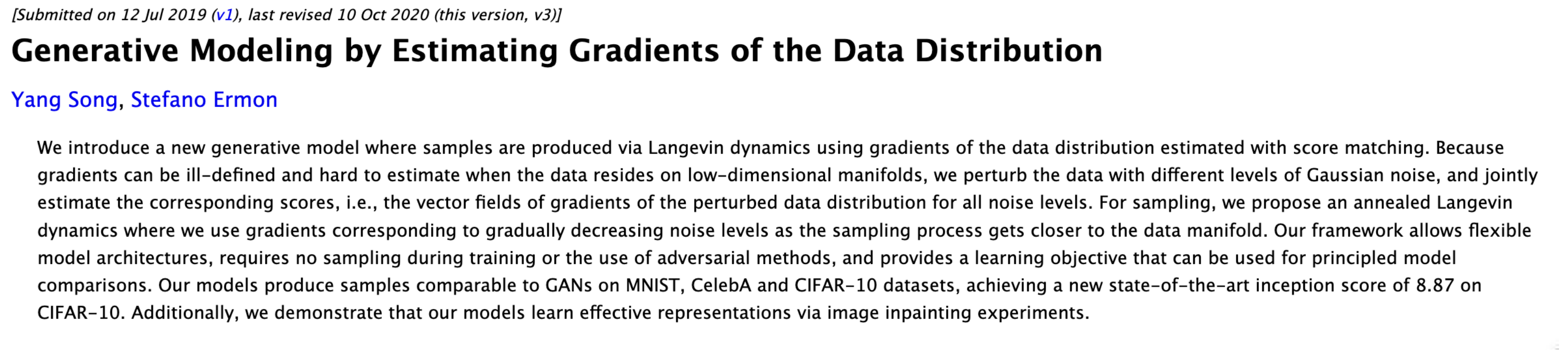


[Submitted on 19 Jun 2020 (v1), last revised 16 Dec 2020 (this version, v2)]

Denoising Diffusion Probabilistic Models

Jonathan Ho, Ajay Jain, Pieter Abbeel

We present high quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics. Our best results are obtained by training on a weighted variational bound designed according to a novel connection between diffusion probabilistic models and denoising score matching with Langevin dynamics, and our models naturally admit a progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding. On the unconditional CIFAR10 dataset, we obtain an Inception score of 9.46 and a state-of-the-art FID score of 3.17. On 256x256 LSUN, we obtain sample quality similar to ProgressiveGAN. Our implementation is available at this https URL



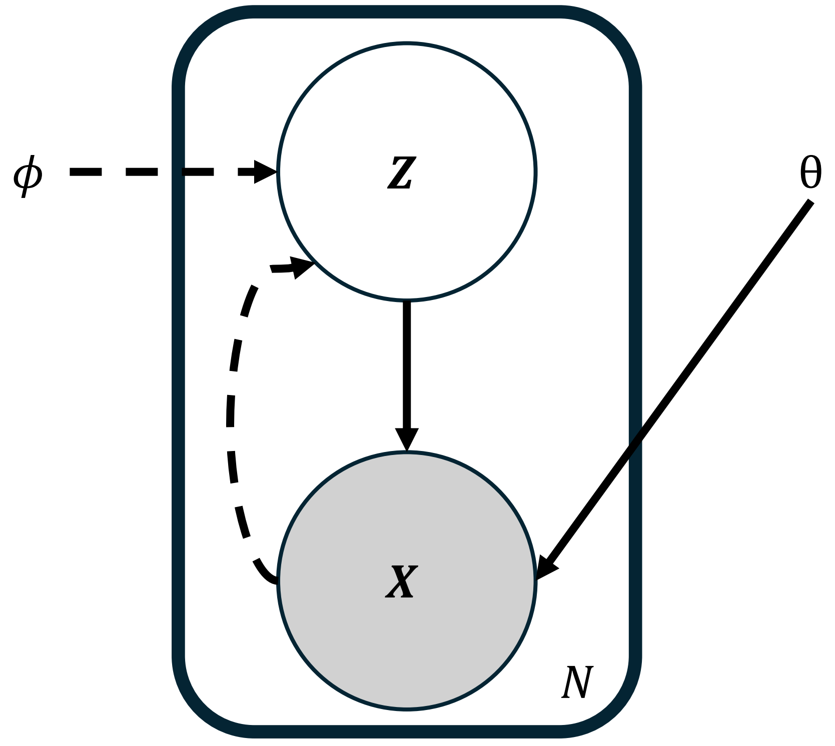
[Submitted on 12 Jul 2019 (v1), last revised 10 Oct 2020 (this version, v3)]

Generative Modeling by Estimating Gradients of the Data Distribution

Yang Song, Stefano Ermon

We introduce a new generative model where samples are produced via Langevin dynamics using gradients of the data distribution estimated with score matching. Because gradients can be ill-defined and hard to estimate when the data resides on low-dimensional manifolds, we perturb the data with different levels of Gaussian noise, and jointly estimate the corresponding scores, i.e., the vector fields of gradients of the perturbed data distribution for all noise levels. For sampling, we propose an annealed Langevin dynamics where we use gradients corresponding to gradually decreasing noise levels as the sampling process gets closer to the data manifold. Our framework allows flexible model architectures, requires no sampling during training or the use of adversarial methods, and provides a learning objective that can be used for principled model comparisons. Our models produce samples comparable to GANs on MNIST, CelebA and CIFAR-10 datasets, achieving a new state-of-the-art inception score of 8.87 on CIFAR-10. Additionally, we demonstrate that our models learn effective representations via image inpainting experiments.

Variational autoencoder (VAE)



The full likelihood:

$$p(z)p_\theta(x|z) = N(z|0, I)N(x|\mu_\theta(z), \Sigma_\theta(z))$$

The posterior: $p(z|x) = ?$

The variational approximation:

$$q_\phi(z|x) = N(z|\mu_\phi(x), \Sigma_\phi(x))$$

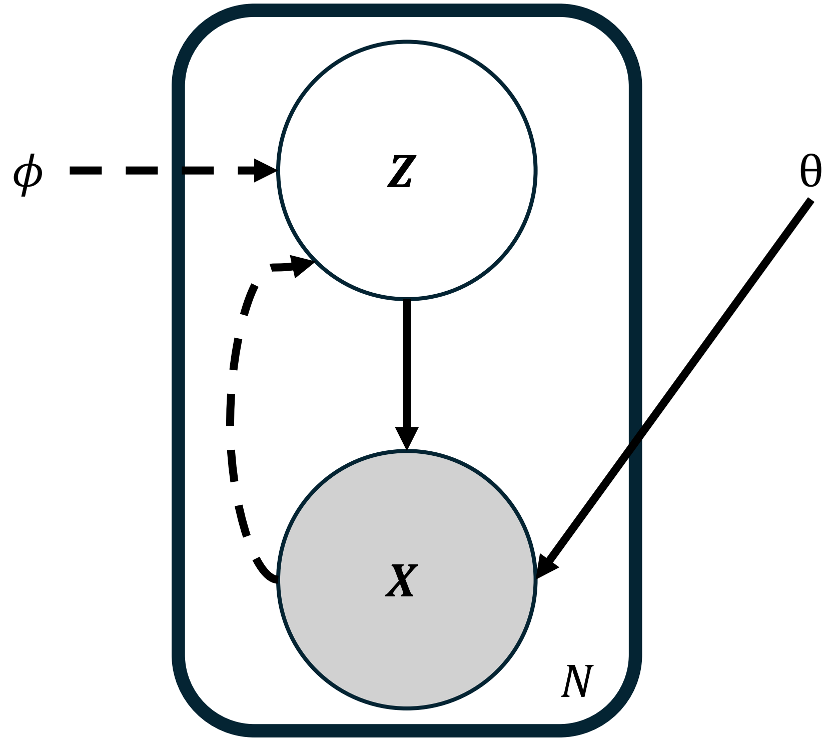
Via the Jensen's Inequality:

$$\log p_\theta(x) \geq \int q_\phi(z|x) \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} dz$$

$$= \int q_\phi(z|x) \log p_\theta(x|z) dz - D_{KL}(q_\phi(z|x)||p(z))$$

ELBO: Evidence Lower Bound

Variational autoencoder (VAE)

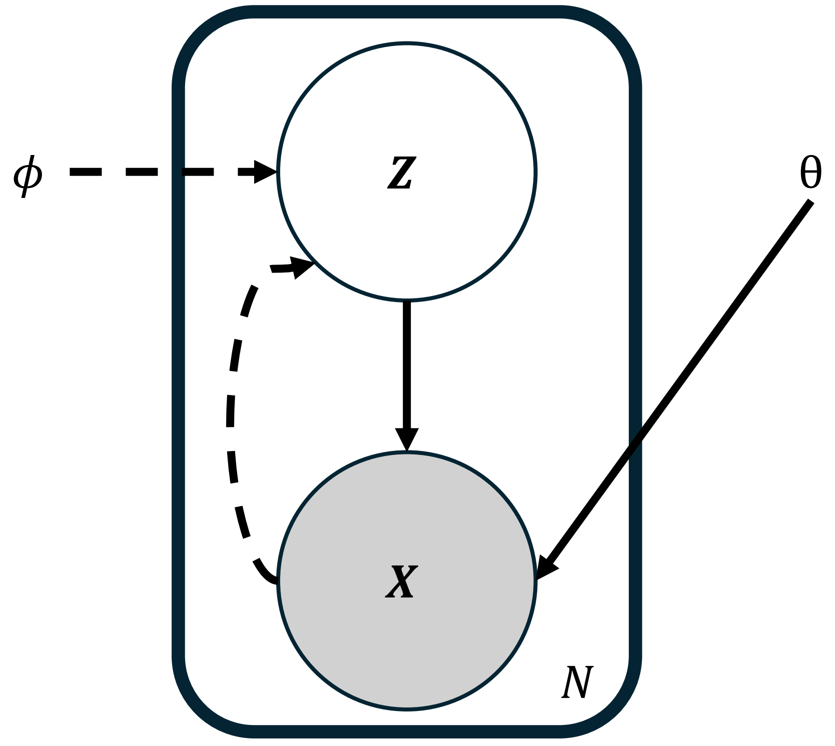


$D_{KL}(q_\phi(z|x)||p(z))$ has an analytic form

$\int q_\phi(z|x) \log p_\theta(x|z)dz = E_{z\sim q_\phi}(\log p_\theta(x|z))$ can be approximated using Monte Carlo integration

$$\frac{1}{K}\sum_{k=1}^{K} \log p_\theta\left(x\middle|z^k\right), z^k \sim N(z|\mu_\phi(x), \Sigma_\phi(x))$$

Variational autoencoder (VAE)



$$\frac{1}{K}\sum_{k=1}^{K} \log p_\theta\left(x \middle| z^k\right), z^k \sim N(z|\mu_\phi(x), \Sigma_\phi(x))$$

Sampling is not differentiable and
$\nabla_\phi E_{z\sim q_\phi}(\log p_\theta(x|z)) \neq E_{z\sim q_\phi}(\nabla_\phi \log p_\theta(x|z))$

Reparameterization:

$$\epsilon \sim N(\epsilon|0, I), z^k = \sum\nolimits_\phi^{\frac{1}{2}}(x)\,\epsilon + \mu_\phi(x)$$

LOTUS:
$E_{z\sim q_\phi}(\log p_\theta(x|z)) = E_{\epsilon\sim N(0,I)}(\log p_\theta(x|z(\epsilon)))$

HVAE

- Just stack a lot of VAEs

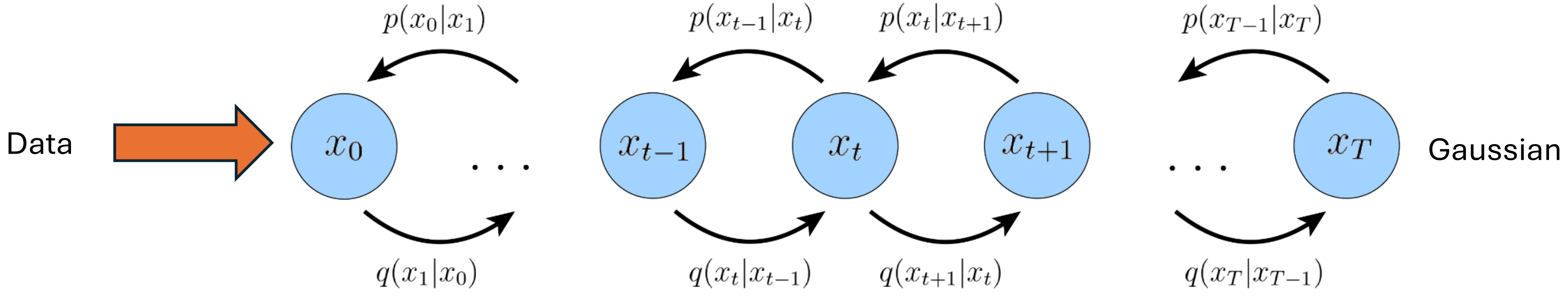
The full likelihood:

$$p(z_{1:T})p_\theta(x|z_1) = p(z_T)\Pi_t p(z_{t-1}|z_t)p(x|z_1)$$

Canonical diffusion

- A special case of HAVE

1. Latent dimensions = data dimension
2. Encoder structure is pre-defined
3. At time T, the distribution is an isotropic Gaussian



$$q(x_t|x_{t-1}) = N(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I)$$

α_t fixed (but can be learned)

ELBO

$$logp(x_0) \geq \int q(x_{1:T}|x_0) log \frac{p(x_{0:T})}{q(x_{1:T}|x_0)} dx_{1:T}$$

$$= \int q(x_{1:T}|x_0)[logp(x_0|x_1) + log \frac{p(x_T)}{q(x_T|x_{T-1})} + \sum_{t=1}^{T-1} log \frac{p(x_t|x_{t+1})}{q(x_t|x_{t-1})}]dx_{1:T}$$

$$= E_{q(x_1|x_0)}[logp(x_0|x_1)] - E_{q(x_{T-1}|x_T)}[D_{KL}(q(x_T|x_{T-1})||p(x_T))] - \sum_{t=1}^{T-1} E_{q(x_{t-1},x_{t+1}|x_0)}[D_{KL}(q(x_t|x_{t-1})||p(x_t|x_{t+1}))]$$

Alternative ELBO

$$q(x_t|x_{t-1}) = q(x_t|x_{t-1}, x_0)$$

$$logp(x_0) \geq \int q(x_{1:T}|x_0) log \frac{p(x_{0:T})}{q(x_{1:T}|x_0)} dx_{1:T}$$

$$= \int q(x_{1:T}|x_0)[log \frac{p(x_T)p(x_0|x_1)}{q(x_1|x_0)} + \sum_{t=2}^{T} log \frac{p(x_{t-1}|x_t)}{q(x_t|x_{t-1}, x_0)}]dx_{1:T}$$

$$= \int q(x_{1:T}|x_0)[log \frac{p(x_T)p(x_0|x_1)}{q(x_1|x_0)} + \sum_{t=2}^{T} log \frac{p(x_{t-1}|x_t)}{\frac{q(x_{t-1}|x_t, x_0)q(x_t|x_0)}{q(x_{t-1}|x_0)}}]dx_{1:T}$$

Alternative ELBO

$$= E_{q(x_1|x_0)}[logp(x_0|x_1)] - D_{KL}(q(x_T|x_0||p(x_T))) - \sum_{t=2}^{T} E_{q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t, x_0)||p(x_{t-1}|x_t))]$$

A separate decoder is training in Denoising diffusion probabilistic models

$q(x_t|x_0)$

$$
\begin{aligned}
x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon^*_{t-1} \\
&= \sqrt{\alpha_t}\left[\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon^*_{t-2}\right] + \sqrt{1-\alpha_t}\epsilon^*_{t-1} = \cdots \\
&= \sqrt{\Pi_{i=1}^{t}\alpha_i}\,x_0 + \sqrt{1-\Pi_{i=1}^{t}\alpha_i}\,\epsilon_0
\end{aligned}
$$

$$\Downarrow$$

$$N(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

$q(x_{t-1}|x_t, x_0)$

$$q(x_{t-1}|x_t, x_0) = \frac{N(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I)N(x_t; \sqrt{\bar{\alpha}_{t-1}}x_0, (1-\bar{\alpha}_{t-1})I)}{N(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)}$$

$$\Downarrow$$

$$N(x_{t-1}; \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_t)x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t}, \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}I)$$

$$D_{KL}(q(x_{t-1}|x_t, x_0)||p(x_{t-1}|x_t))$$

If we let $p(x_{t-1}|x_t)$ be a Gaussian $N(\mu_\theta, \Sigma)$, this now has an analytic form

If we further match the variances, the KL is just

$$\frac{1}{2(\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t})}\left\|\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_t)x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} - \mu_\theta\right\|^2$$

We can set $\mu_\theta = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_t)x_t+\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_\theta(x_t,t)}{1-\bar{\alpha}_t}$

Objective

$$\frac{1}{2\left(\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\right)} \frac{\bar{\alpha}_{t-1}(1-\alpha_t)^2}{(1-\bar{\alpha}_t)^2} \|x_\theta(x_t, t) - x_0\|^2$$

Optimizing over all time steps can be further approximated by random sampling

$$E_{t\sim U\{2,\ldots,T\}}[E_{q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t, x_0)||p(x_{t-1}|x_t))]]$$

Alternative Objective

$$x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_0}{\sqrt{\bar{\alpha}_t}}$$

$$\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_t)x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} = \frac{1}{\sqrt{\alpha_t}}x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\epsilon_0$$

We can set $\mu_\theta = \frac{1}{\sqrt{\alpha_t}}x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\epsilon_0(x_t, t)$

Objective (part): $\frac{1}{2\left(\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\right)}\frac{(1-\alpha_t)^2}{(1-\bar{\alpha}_t)\alpha_t}||\epsilon_0 - \epsilon_0(x_t, t)||^2$

Back to the main topic: Git

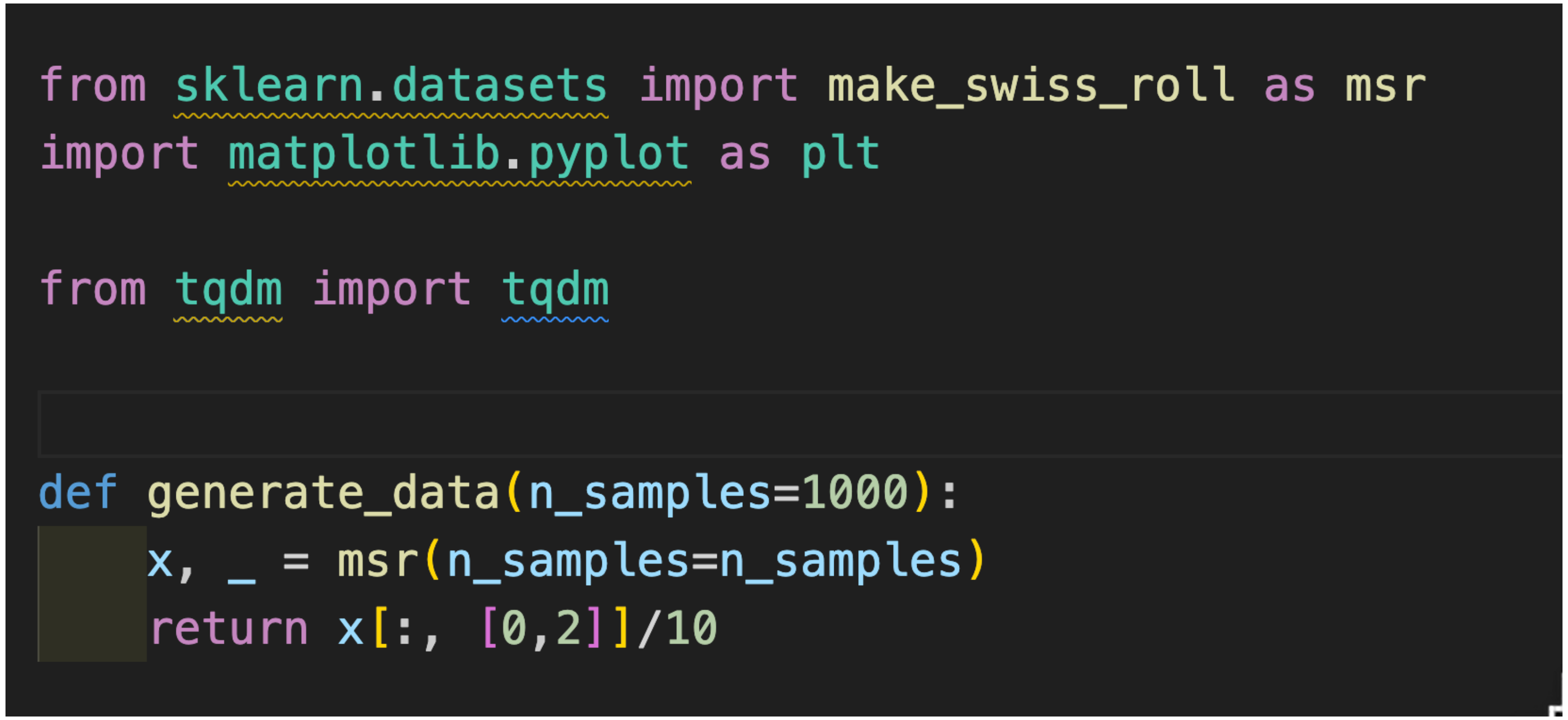
- We need:
  - A function to sample from some nontrivial distribution (data distribution)
  - A neural network (not a complicated one)
  - Forward and reverse processes
  - Of course, loss
  - Some other functions (plotting, training)

```
>touch utility.py
>touch simple_diffusion.py
```

(Adapted from https://github.com/MaximeVandegar/Papers-in-100-Lines-of-Code/blob/main/Deep_Unsupervised_Learning_using_Nonequilibrium_Thermodynamics/diffusion_models.py)

utility.py

A function to generate from a 2d
Swiss roll



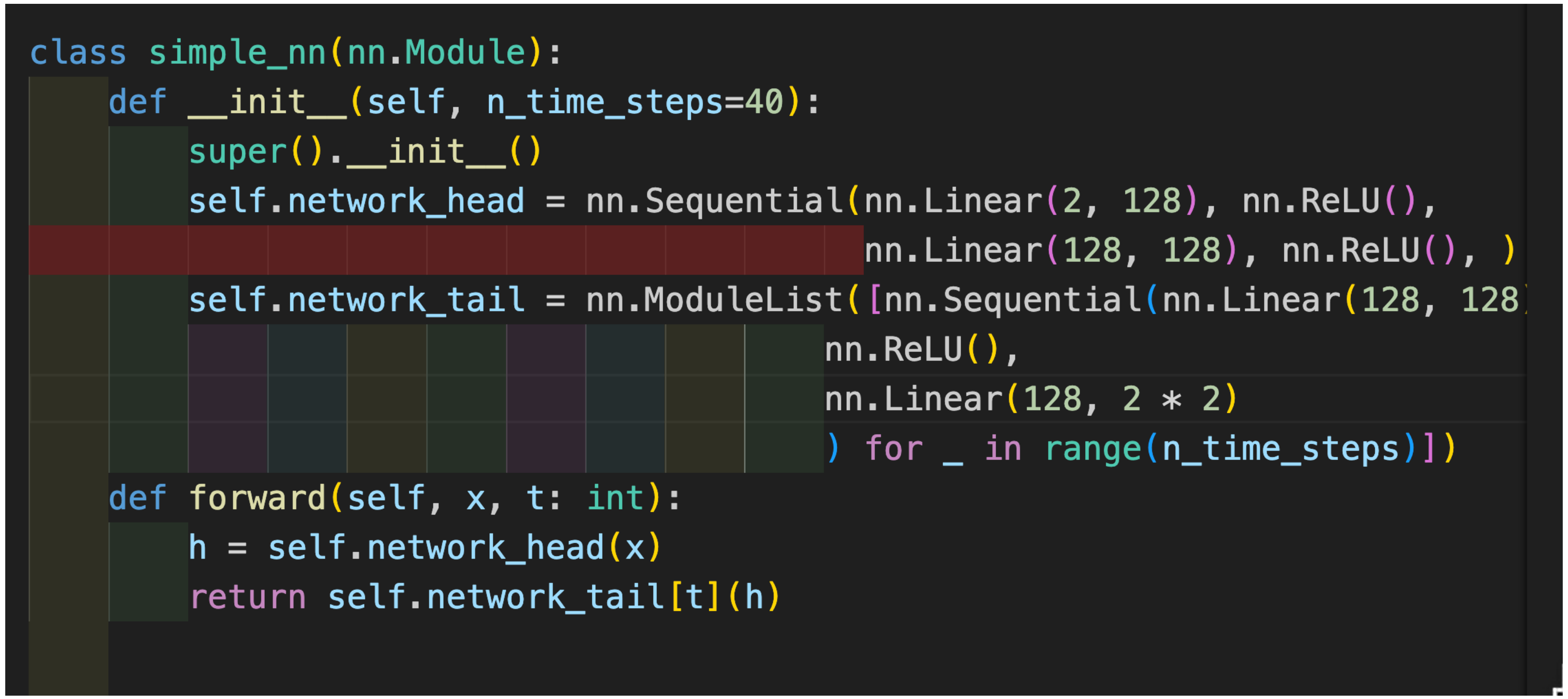
```python
from sklearn.datasets import make_swiss_roll as msr
import matplotlib.pyplot as plt

from tqdm import tqdm


def generate_data(n_samples=1000):
    x, _ = msr(n_samples=n_samples)
    return x[:, [0,2]]/10
```

utility.py

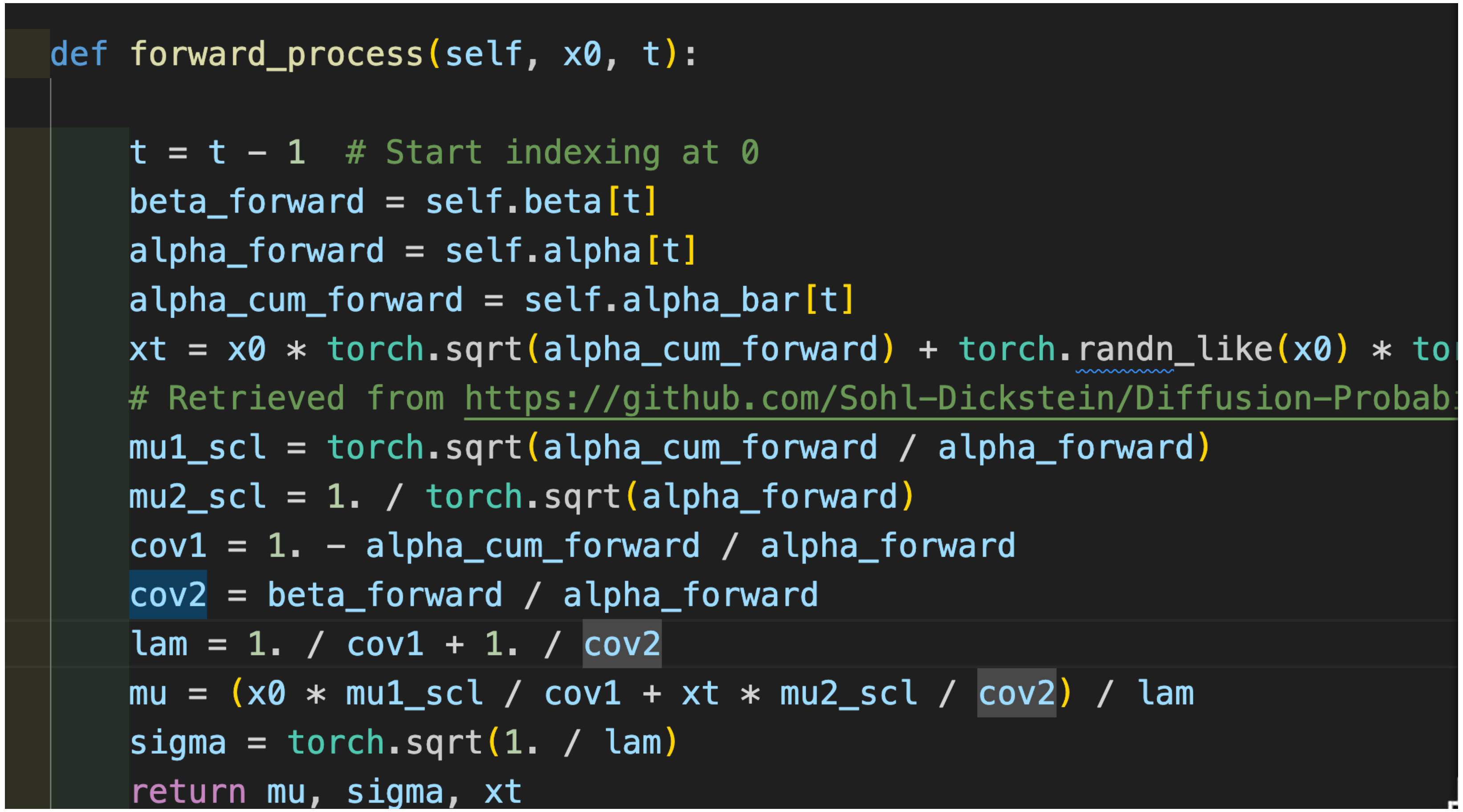
And a simple neural network



```python
class simple_nn(nn.Module):
    def __init__(self, n_time_steps=40):
        super().__init__()
        self.network_head = nn.Sequential(nn.Linear(2, 128), nn.ReLU(),
                                          nn.Linear(128, 128), nn.ReLU(), )
        self.network_tail = nn.ModuleList([nn.Sequential(nn.Linear(128, 128)
                                         nn.ReLU(),
                                         nn.Linear(128, 2 * 2)
                                         ) for _ in range(n_time_steps)])
    def forward(self, x, t: int):
        h = self.network_head(x)
        return self.network_tail[t](h)
```

simple_diffusion.py

Forward process

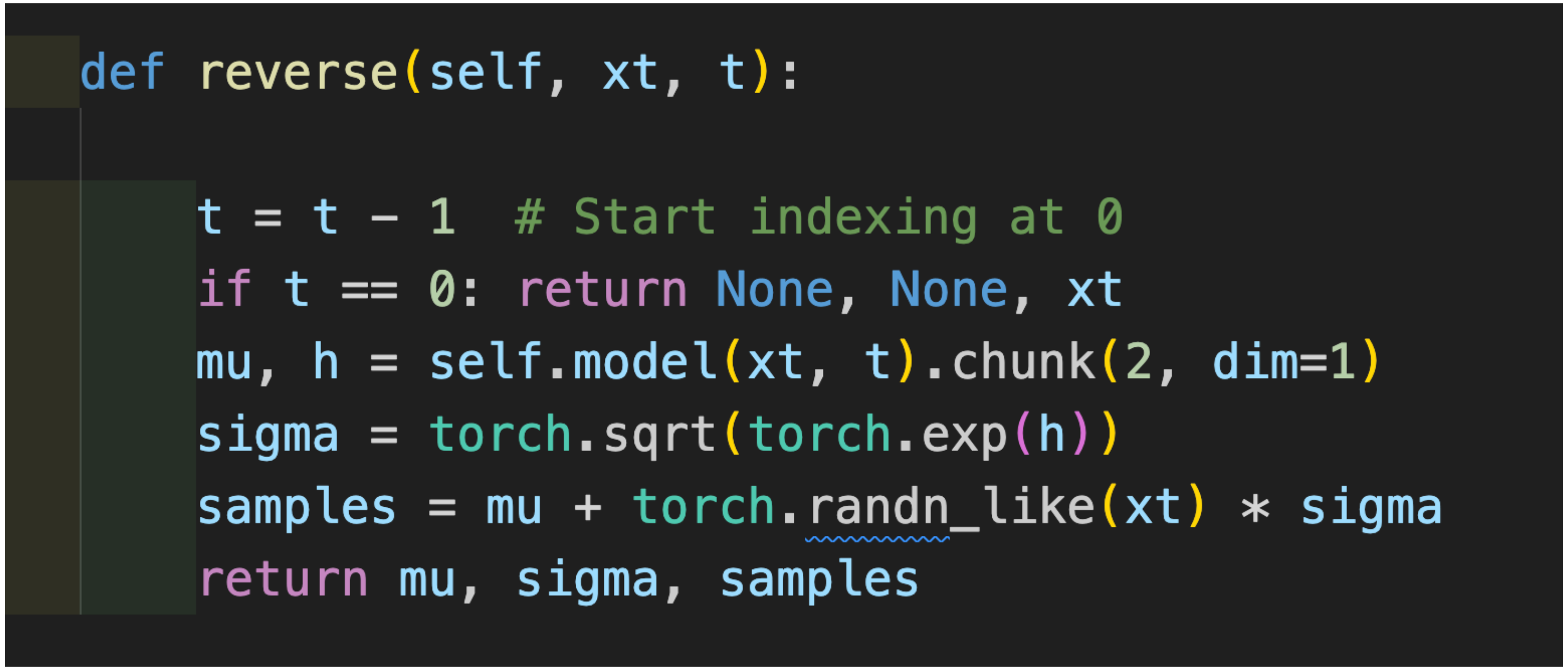


```
def forward_process(self, x0, t):

    t = t - 1  # Start indexing at 0
    beta_forward = self.beta[t]
    alpha_forward = self.alpha[t]
    alpha_cum_forward = self.alpha_bar[t]
    xt = x0 * torch.sqrt(alpha_cum_forward) + torch.randn_like(x0) * to
    # Retrieved from https://github.com/Sohl-Dickstein/Diffusion-Probab
    mu1_scl = torch.sqrt(alpha_cum_forward / alpha_forward)
    mu2_scl = 1. / torch.sqrt(alpha_forward)
    cov1 = 1. - alpha_cum_forward / alpha_forward
    cov2 = beta_forward / alpha_forward
    lam = 1. / cov1 + 1. / cov2
    mu = (x0 * mu1_scl / cov1 + xt * mu2_scl / cov2) / lam
    sigma = torch.sqrt(1. / lam)
    return mu, sigma, xt
```

simple_diffusion.py

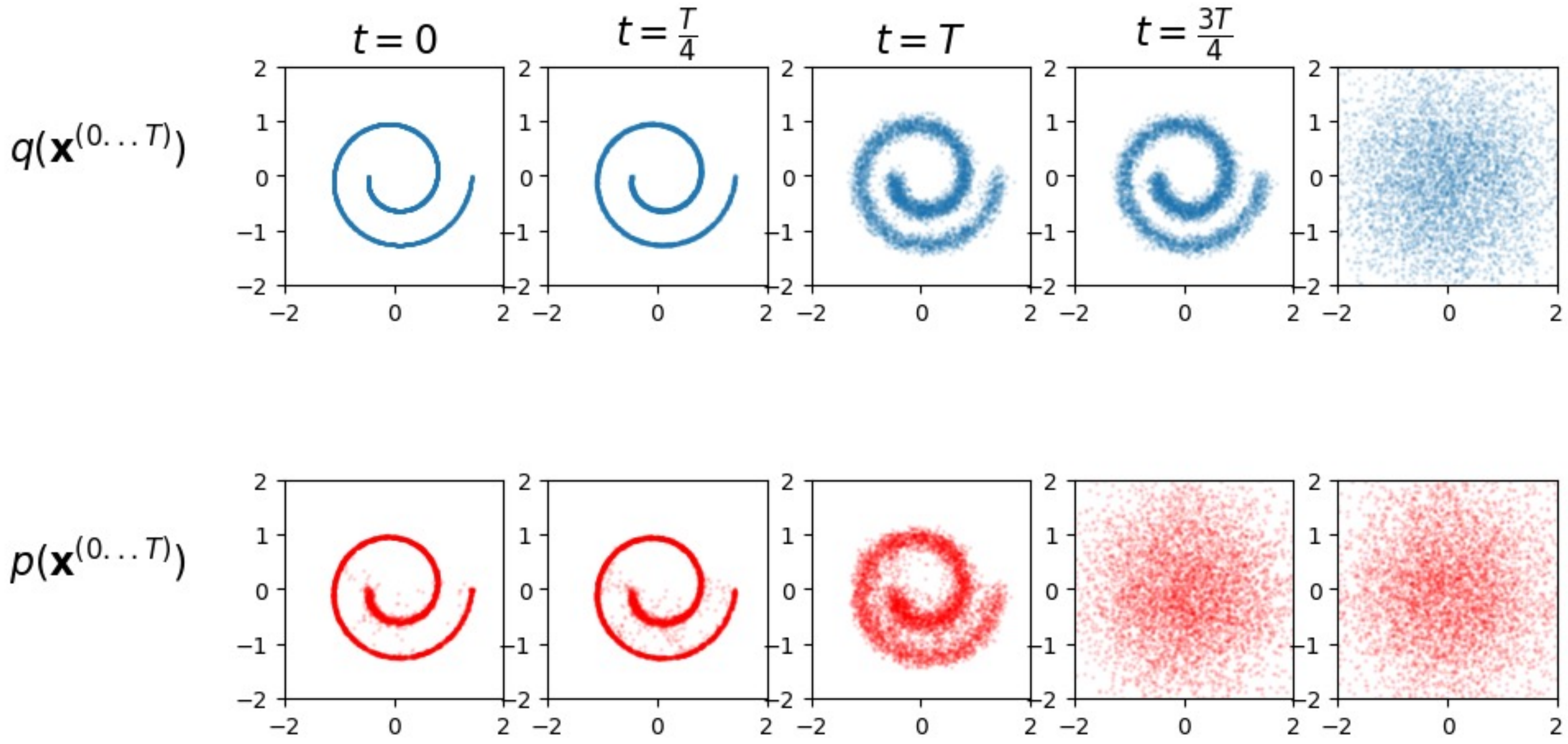
Reverse process



```
def reverse(self, xt, t):

    t = t - 1  # Start indexing at 0
    if t == 0: return None, None, xt
    mu, h = self.model(xt, t).chunk(2, dim=1)
    sigma = torch.sqrt(torch.exp(h))
    samples = mu + torch.randn_like(xt) * sigma
    return mu, sigma, samples
```

After training


$t = 0$
$t = \frac{T}{4}$
$t = T$
$t = \frac{3T}{4}$
$q(\mathbf{x}^{(0...T)})$
$p(\mathbf{x}^{(0...T)})$