COMP 7070 Advanced Topics in AI and ML April 16, 2025

Lecture 12: From VAE to Diffusion Models

Instructor: Yifan Chen *Scribes: Meng Shen, Jinwen Xu* *Proof reader: Xiong Peng*

12.1 Variational Diffusion Models

12.1.1 Markovian Hierarchical Variational Autoencoders

A Hierarchical Variational Autoencoder (HVAE) is a generalization of a VAE that extends to multiple hierarchies over latent variables. Under this formulation, latent variables themselves are interpreted as generated from other higher-level, more abstract latents. As a special case of HVAE, Markovin HVAE (MHVAE) models the generative process as a Markovin chain. In MHVAE, each transition down the hierarchy is Markovian, where decoding each latent $\mathbf{z}_t$ only conditions on previous latent $\mathbf{z}_{t+1}$. Mathematically, we represent the joint distribution and the posterior of a MHVAE as,

$$\begin{aligned} p(\mathbf{x}, \mathbf{z}_{1:T}) &= p(\mathbf{z}_T) \cdot p_\theta(\mathbf{x}|\mathbf{z}_1) \cdot \prod_{t=2}^{T} p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t), \\ q_\phi(\mathbf{z}_{1:T}|\mathbf{x}) &= q_\phi(\mathbf{z}_1|\mathbf{x}) \cdot \prod_{t=2}^{T} q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}). \end{aligned} \tag{12.1}$$

Then, we can write the ELBO as,

$$\begin{aligned} \log p(\mathbf{x}) &= \log \int p(\mathbf{x}, \mathbf{z}_{1:T}) \mathrm{d}\mathbf{z}_{1:T} \\ &= \log \int \frac{p(\mathbf{x}, \mathbf{z}_{1:T}) \cdot q_\phi(\mathbf{z}_{1:T}|\mathbf{x})}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x})} \mathrm{d}\mathbf{z}_{1:T} \\ &= \log \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x})} \left[\frac{p(\mathbf{x}, \mathbf{z}_{1:T})}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x})} \right] \\ &\geq \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x})} \left[\log \frac{p(\mathbf{z}, \mathbf{z}_{1:T})}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x})} \right]. \end{aligned} \tag{12.2}$$

We can then substitute the joint distribution and posterior in eq. (12.1) into eq. (12.2) to produce an alternate form, i.e.,

$$\mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x})} \left[\log \frac{p(\mathbf{z}, \mathbf{z}_{1:T})}{q_\phi(\mathbf{z}_{1:T}|\mathbf{x})} \right] = \mathbb{E}_{q_\phi(\mathbf{z}_{1:T}|\mathbf{x})} \left[\log \frac{p(\mathbf{z}_T) \cdot p_\theta(\mathbf{x}|\mathbf{z}_1) \cdot \prod_{t=2}^{T} p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t)}{q_\phi(\mathbf{z}_1|\mathbf{x}) \cdot \prod_{t=2}^{T} q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1})} \right]. \tag{12.3}$$

When we investigate Variational Diffusion Models below, this objective can be further decomposed into interpretable components.

12.1.2 Make MHVAE becomes Variational Diffusion Models

The easiest way to think of a Variational Diffusion Model (VDM) is simply as a MHVAE with three key restrictions as follows,

- The latent dimension is exactly equal to the data dimension.

- The structure of the latent encoder at each timestep is not learned; it is pre-defined as a linear Gaussian model. In other words, it is a Gaussian distribution centered around the output of the previous timestep.
- The Gaussian parameters of the latent encoders vary over time in such a way that the distribution of the latent at final timestep T is a standard Gaussian.

From the first restriction, we can represent both true data samples and latent variables as $\mathbf{x}_t$, where $t = 0$ refers true data samples and $t \in [1 : T]$ represents a corresponding latent with hierarchical index by t. The VDM posterior is the same as MHVAE in eq. (12.1), which can be rewritten as,

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}). \tag{12.4}$$

From the second assumption, we know that the distribution of each latent variable in the encoder is a Gaussian centered around its previous hierarchical latent. The encoder is fixed as a linea Gaussian model, where the mean and standard deviation can be set beforehand as hyperparameters or learned as parameters. We parameterize the Gaussian encoder with mean $\mu_t(\mathbf{x}_t) = \sqrt{\alpha_t}\mathbf{x}_{t-1}$ and variance $\sum_t(\mathbf{x}_t) = (1 - \alpha_t)\mathbf{I}$, where the form of the coefficients are chosen such that the variance of the latent variables stays at a similar scale. Mathematically, encoder transitions are denoted as follows,

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1 - \alpha_t)\mathbf{I}). \tag{12.5}$$

From the third assumption, we know that α_t evolves over time according to a fixed or learnable schedule structured such that the distribution of the final latent, i.e., $p(\mathbf{x}_T)$ is a standard Gaussian. We can then update the joint distribution of a MHVAE in eq. (12.1) to write the joint distribution of VDM as,

$$p(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \cdot \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \text{ where } p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}). \tag{12.6}$$

Note that the encoder distributions $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ are not parameterized by ϕ, as they are completely modeled as Gaussian with defined mean and variance parameters at each timestep. Therefore, in a VDM, the key is to learn conditionals $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, so that we can simulate new data. After optimizing the VDM, the sampling procedure is as simple as sampling Gaussian noise from $p(\mathbf{x}_T)$ and iteratively runing the denoising transitions $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ for T steps to generate a novel x_0.

Like any HVAE, the VDM can be optimized by maximizing the ELBO, which can be derived as,

$$\begin{aligned}
\log p(\mathbf{x}) &= \log \int p(\mathbf{x}_{0:T}) \mathrm{d}\mathbf{x}_{1:T} \\
&= \log \int \frac{p(\mathbf{x}_{0:T}) \cdot q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \mathrm{d}\mathbf{x}_{1:T} \\
&= \log \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \\
&\geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) \cdot \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) \cdot p_\theta(\mathbf{x}_0|\mathbf{x}_1) \cdot \prod_{t=2}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_T|\mathbf{x}_{T-1}) \cdot \prod_{t=1}^{T-1} q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) \cdot p_\theta(\mathbf{x}_0|\mathbf{x}_1) \cdot \prod_{t=1}^{T-1} p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_T|\mathbf{x}_{T-1}) \cdot \prod_{t=1}^{T-1} q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) \cdot p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \prod_{t=1}^{T-1} \frac{p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right]
\end{aligned} \tag{12.7}$$

$$
\begin{aligned}
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})}\right] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\sum_{t=1}^{T-1} \log \frac{p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] + \mathbb{E}_{q(\mathbf{x}_{T-1},\mathbf{x}_T|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})}\right] \\
&\qquad + \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{t-1},\mathbf{x}_t,\mathbf{x}_{t+1}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \\
&= \underbrace{\mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right]}_{\text{Reconstruction Term}} - \underbrace{\mathbb{E}_{q(\mathbf{x}_{T-1}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_{T-1}) \mid\mid p(\mathbf{x}_T))\right]}_{\text{Prior Matching Term}} \\
&\qquad - \sum_{t=1}^{T-1} \underbrace{\mathbb{E}_{q(\mathbf{x}_{t-1},\mathbf{x}_{t+1}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t-1}) \mid\mid p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}))\right]}_{\text{Consistency Term}}.
\end{aligned}
\tag{12.8}
$$

Each term of the derived form of the ELBO in eq. (12.7) can be interpreted as follows. (a) **Reconstruction Term** is the prediction of the log likelihood of the orginal data given the first-step latent. (b) **Prior Matching Term** reaches the minimum when the final latent distribution matches the Gaussian prior. (c) **Consistency Term** constrains the distribution at $\mathbf{x}_t$ being consistent, in both forward and backward processes.

Since the consistency term in eq. (12.7) is computed as an expectation over two random variables, i.e., $\{\mathbf{x}_{t-1}, \mathbf{x}_{t+1}\}$ for every timestep, its Monte Carlo estimate could potentially be higher than a term that is estimated using only one random variable per timestep. Thus, we instead derive a form for ELBO where each term is computated as an expectation over only one random variable at a time. The key insight is that we can rewrite encoder transitions as $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)$, where the extra conditioning term is superfluous due to the Markov property. Then, according to Bayes rule, we can rewrite each transition as,

$$
q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \cdot q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}. \tag{12.9}
$$

With eq. (12.9), we can retry the deriviation resuming from the ELBO in eq. (12.7) as,

$$
\begin{aligned}
\log p(\mathbf{x}) &\geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T) \cdot \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T) \cdot p_\theta(\mathbf{x}_0|\mathbf{x}_1) \cdot \prod_{t=2}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_1|\mathbf{x}_0) \cdot \prod_{t=2}^T q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_T) \cdot p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \prod_{t=2}^T \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_T) \cdot p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \prod_{t=2}^T \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \cdot q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_T) \cdot p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{q(\mathbf{x}_T|\mathbf{x}_0)} + \log \prod_{t=2}^T \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_T) \cdot p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \sum_{t=2}^T \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)}\right] + \sum_{t=2}^T \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}\right]
\end{aligned}
\tag{12.10}
$$

$$
\begin{aligned}
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}\right] \\
&= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] + \mathbb{E}_{q(\mathbf{x}_T|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t,\mathbf{x}_{t-1}|\mathbf{x}_0)}\left[\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}\right] \\
&= \underbrace{\mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right]}_{\text{Reconstruction Term}} - \underbrace{\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \mid\mid p(\mathbf{x}_T))}_{\text{Prior Matching Term}} - \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) \mid\mid p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))\right]}_{\text{Denoising Matching Term}}
\end{aligned}
\tag{12.11}
$$

The ELBO in eq. (12.10) can be estimated with lower variance, since each term is computed as an expectation of at most one random variable at a time. Each term of eq. (12.10) can be interpreted as follows. **(a) Reconstruction Term** can be approaximated and optimized using a Monte Carlo estimate. **(b) Prior Matching Term** represents how close the distribution of the final noisified input is to the standard Gaussian prior. **(c) Denoising Matching Term** can be minimized when the two denoising steps match as closely as possible.

12.1.3 Computation of Loss

With the derived ELBO, we can define the loss function based on ELBO for model training. In eq. (12.10), **the key is to compute the denoising matching term** which requires a large cost of computation. However, the KL divergence term $\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$ is difficult to minimize. In VDM, we can leverage the Gaussian transition assumption to make optimization tractable. By Bayes rule, we have,

$$
q(\mathbf{x}_{t-1}|\mathbf{x}_t, x_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0)\cdot q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)}. \tag{12.12}
$$

where $q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I})$. Thus, the key is *how can we obtain* $q(\mathbf{x}_{t-1}|\mathbf{x}_0)$ *and* $q(\mathbf{x}_t|\mathbf{x}_0)$? The details are as follows.

Under the reparameterization trick, samples $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_{t-1})$ can be rewritten as,

$$
\mathbf{x}_t = \sqrt{\alpha_t}\cdot\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\cdot\epsilon, \quad \epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I}). \tag{12.13}
$$

Then, $q(\mathbf{x}_t|\mathbf{x}_0)$ can be derived by repeating the reparameterization trick. Assume we have access to $2T$ random noise variables $\{\epsilon_t^*, \epsilon_t\}_{t=0}^T \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I})$. For an arbitrary sample $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$, it can be rewritten as the Gaussian form as follows,

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\cdot\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\cdot\epsilon_{t-1}^* \\
&= \sqrt{\prod_{i=1}^{t}\alpha_i}\cdot\mathbf{x}_0 + \sqrt{1-\prod_{i=1}^{t}\alpha_i}\cdot\epsilon_0 \\
&= \sqrt{\bar{\alpha}_t}\cdot\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\cdot\epsilon_0 \\
&\sim \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\cdot\mathbf{x}_0, (1-\bar{\alpha}_t)\cdot\mathbf{I}).
\end{aligned}
\tag{12.14}
$$

So far, we have derived the Gaussian form of $q(\mathbf{x}_t|\mathbf{x}_0)$ in eq. (12.14). Similarly, this derivation can be modified to yield the Gaussian parameterization describing $q(\mathbf{x}_{t-1}|\mathbf{x}_0) \sim \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_t}\cdot\mathbf{x}_0, (1-\bar{\alpha}_t)\cdot\mathbf{I})$. By substituting the Gaussion forms into eq. (12.12), we have,

$$
\begin{aligned}
q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) &= \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0)\cdot q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \\
&= \frac{\mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I})\cdot\mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\cdot\mathbf{I})}{\mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\cdot\mathbf{x}_0, (1-\bar{\alpha}_t)\cdot\mathbf{I})} \\
&\propto \mathcal{N}(\mathbf{x}_{t-1}; \mu_q(\mathbf{x}_t,\mathbf{x}_0), \sum_q(t)),
\end{aligned}
\tag{12.15}
$$

where $\mu_q(\mathbf{x}_t,\mathbf{x}_0) = \frac{\sqrt{\alpha_t}\cdot(1-\bar{\alpha}_{t-1})\cdot\mathbf{x}_t + \sqrt{\bar{\alpha}_{t-1}}\cdot(1-\alpha_t)\cdot\mathbf{x}_0}{1-\bar{\alpha}_t}$ and $\sum_q(t) = \frac{(1-\alpha_t)\cdot(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\cdot\mathbf{I}$. With $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ in eq. (12.15), we can define the loss function based on it. Specifically, we set the variances of the two

Gaussians to match exactly, optimizing the KL divergence term reduces to minimizing the difference between the means of the two distributions, i.e.,

$$\begin{aligned}
&\arg\min_\theta \mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) \;||\; p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \\
=& \arg\min_\theta \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(\mathbf{x}_{t-1};\mu_q(\mathbf{x}_t,\mathbf{x}_0),\sum_q(t)) \;||\; \mathcal{N}(\mathbf{x}_{t-1};\mu_\theta(\mathbf{x}_t,t),\sum_q(t))) \\
=& \arg\min_\theta \frac{1}{2}\left[\log\frac{|\sum_q(t)|}{|\sum_q(t)|} - d + \mathrm{tr}(\sum_q(t)^{-1}\sum_q(t)) + (\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,\mathbf{x}_0))^\top\cdot\sum_q(t)^{-1}\cdot(\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,\mathbf{x}_0))\right] \\
=& \arg\min_\theta \frac{1}{2}\left[\log 1 - d + d + (\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,t))^\top\cdot\sum_q(t)^{-1}\cdot(\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,t))\right] \\
=& \arg\min_\theta \frac{1}{2}\left[(\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,\mathbf{x}_0))^\top\cdot\sum_q(t)^{-1}\cdot(\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,\mathbf{x}_0))\right] \\
=& \arg\min_\theta \frac{1}{2}\left[(\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,\mathbf{x}_0))^\top\cdot(\sigma_q^2(t)\cdot\mathbf{I})^{-1}\cdot(\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,\mathbf{x}_0))\right] \\
=& \arg\min_\theta \frac{1}{2\sigma_q^2(t)}\left[||\mu_\theta(\mathbf{x}_t,t)-\mu_q(\mathbf{x}_t,\mathbf{x}_0)||_2^2\right],
\end{aligned} \tag{12.16}$$

where $\mu_q(\mathbf{x}_t,\mathbf{x}_0)=\frac{\sqrt{\alpha_t}\cdot(1-\bar{\alpha}_{t-1})\cdot\mathbf{x}_t+\sqrt{\bar{\alpha}_{t-1}}\cdot(1-\alpha_t)\cdot\mathbf{x}_0}{1-\bar{\alpha}_t}$, $\mu_\theta(\mathbf{x}_t,t)=\frac{\sqrt{\alpha_t}\cdot(1-\bar{\alpha}_{t-1})\cdot\mathbf{x}_t+\sqrt{\bar{\alpha}_{t-1}}\cdot(1-\alpha_t)\cdot\hat{\mathbf{x}}_\theta(\mathbf{x}_t,t)}{1-\bar{\alpha}_t}$. And $\hat{\mathbf{x}}_\theta(\mathbf{x}_t,t)$ is parameterized by a neural network that seeks to predict $\mathbf{x}_0$ from noisy image $\mathbf{x}_t$ and time index t. Then, the optimization objective can be simplified as follows,

$$\begin{aligned}
&\arg\min_\theta \mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) \;||\; p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \\
=& \arg\min_\theta \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(\mathbf{x}_{t-1};\mu_q(\mathbf{x}_t,\mathbf{x}_0),\sum_q(t)) \;||\; \mathcal{N}(\mathbf{x}_{t-1};\mu_\theta(\mathbf{x}_t,t),\sum_q(t))) \\
=& \arg\min_\theta \frac{1}{2\sigma_q^2(t)}\frac{\bar{\alpha}_{t-1}\cdot(1-\alpha)^2}{(1-\bar{\alpha}_t)^2}\left[||\hat{\mathbf{x}}_\theta(\mathbf{x}_t,t)-\mathbf{x}_0||_2^2\right].
\end{aligned} \tag{12.17}$$

Thus, the optimization of VDM is equal to learn a neural network to predict the original image from a random Gaussian noise. Furthermore, minimizing the summation term of the derived ELBO objctive across all noise levels can be approximated by minimizing the expectation over all timesteps as,

$$\arg\min_\theta \mathbb{E}_{t\sim U\{2,T\}}\left[\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) \;||\; p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))\right]. \tag{12.18}$$

Based on eq. (12.18), the optimization problem can be further specified as,

$$\begin{aligned}
&\arg\min_\theta \mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) \;||\; p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \\
=& \arg\min_\theta \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(\mathbf{x}_{t-1};\mu_q,\sum_q(t)) \;||\; \mathcal{N}(\mathbf{x}_{t-1};\mu_\theta,\sum_q(t))) \\
=& \arg\min_\theta \frac{1}{2\sigma_q^2(t)}\left[\left\|\frac{1}{\sqrt{\alpha_t}}\cdot\mathbf{x}_t+\frac{1-\alpha_t}{\sqrt{\alpha_t}}\cdot s_\theta(\mathbf{x}_t,t)-\frac{1}{\sqrt{\alpha_t}}\cdot\mathbf{x}_t-\frac{1-\alpha_t}{\sqrt{\alpha_t}}\cdot\nabla\log p(\mathbf{x}_t)\right\|_2^2\right] \\
=& \arg\min_\theta \frac{1}{2\sigma_q^2(t)}\left[\left\|\frac{1-\alpha_t}{\sqrt{\alpha_t}}\cdot s_\theta(\mathbf{x}_t,t)-\frac{1-\alpha_t}{\sqrt{\alpha_t}}\cdot\nabla\log p(\mathbf{x}_t)\right\|_2^2\right] \\
=& \arg\min_\theta \frac{1}{2\sigma_q^2(t)}\left[\left\|\frac{1-\alpha_t}{\sqrt{\alpha_t}}\cdot s_\theta(\mathbf{x}_t,t)-\nabla\log p(\mathbf{x}_t)\right\|_2^2\right] \\
=& \arg\min_\theta \frac{1}{2\sigma_q^2(t)}\cdot\frac{(1-\alpha_t)^2}{\alpha_t}\left[\|s_\theta(\mathbf{x}_t,t)-\nabla\log p(\mathbf{x}_t)\|_2^2\right],
\end{aligned} \tag{12.19}$$

where $s_\theta(\mathbf{x}_t,t)$ is a neural network that learns to predict the score function $\nabla_{\mathbf{x}_t}p(\mathbf{x}_t)$, which is the gradient of $\mathbf{x}_t$ for any arbitrary noise level t.

12.1.4 Three Equivalent Objectives of VDM

The optimization of a VDM has three equivalent objectives as shown in Tab. (12.1). In eq. (12.18), it shows that a VDM can be trained by learning a neural network to predict the original image $\mathbf{x}_0$ from an arbitrary noised version $\mathbf{x}_t$ and its time index t. Furthermore, we conclude another two interpretations for VDM optimization as follows. One is to learn a neural network for predicting the source noise $\epsilon_0 \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I})$ determining $\mathbf{x}_t$ from $\mathbf{x}_0$, which is a basic idea in DDPM. Another one is learning a neural network to predict the score of the image at an arbitrary noise level $\nabla \log p(\mathbf{x}_t)$.

Table 12.1: Three equivalent objectives to optimize a VDM.

Model	Objective	Loss
VDM	Predict original image $\mathbf{x}_0$	$\|\|\mathbf{x}_0 - \hat{\mathbf{x}}_\theta(\mathbf{x}_t, t)\|\|_2^2$
DDPM	Predict source noise ϵ_0	$\|\|\epsilon_0 - \hat{\epsilon}_\theta(\mathbf{x}_t, t)\|\|_2^2$
Score-based Models	Predict the score $\nabla \log p(\mathbf{x}_t)$	$\|\|s_\theta(\mathbf{x}_t, t) - \nabla \log p(\mathbf{x}_t)\|\|_2^2$

12.2 Score-based Models

12.2.1 Energy-based Models

To understand why optimizing a score function make sense, we first revist energy-based models. Arbitrarily fexible probability distribution can written based on energy function as follows,

$$p_\theta(\mathbf{x}) = \frac{1}{Z_\theta} e^{-f_\theta(\mathbf{x})}, \tag{12.20}$$

where $f_\theta(\mathbf{x})$ refers to the energy function, Z_θ is a normalizing constant to ensure that $\int p_\theta(\mathbf{x})\mathrm{d}\mathbf{x} = 1$. One straightforward way to learn such distribution is maximum likelihood. However, the normalizing constant $Z_\theta = \int e^{-f_\theta(\mathbf{x})}\mathrm{d}\mathbf{x}$ is untractable, due to the impossibility of computing $f_\theta(\mathbf{x})$. To avoid calculating or modeling the normalization constant, we can use a neural network $s_\theta(\mathbf{x})$ to learn the score function $\nabla \log p(\mathbf{x})$ of distribution $p(\mathbf{x})$. Take the derivative of the log of both sides of eq. (12.20), we can have,

$$\begin{aligned}
\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) &= \nabla_{\mathbf{x}} \log(\frac{1}{Z_\theta} e^{-f_\theta(\mathbf{x})}) \\
&= \nabla_{\mathbf{x}} \log \frac{1}{Z_\theta} + \nabla_{\mathbf{x}} \log e^{-f_\theta(\mathbf{x})} \\
&= -\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) \\
&\approx s_\theta(\mathbf{x}).
\end{aligned} \tag{12.21}$$

According to eq. (12.21), the objective of score-based models can be defined as the minimization of the Fisher Divergence between $\nabla \log p(x)$ and $s_\theta(x)$ as follows,

$$\mathbb{E}_{p(\mathbf{x})}\left[||s_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p(\mathbf{x})||_2^2\right]. \tag{12.22}$$

12.2.2 Score Matching Langevin Dynamics

To sample the data from the $p(\mathbf{x})$ using score function, we use Langevin Dynamics sampling, which can be formulated as,

$$\mathbf{x}_{i+1} = \mathbf{x}_i + c \cdot \nabla_{\mathbf{x}} \log p(\mathbf{x}_i) + \sqrt{2c} \cdot \mathbf{z}_i, \quad \mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad i = 0, 1, ..., K, \tag{12.23}$$

where c is a small constant, $\mathbf{x}_0$ is randomly intialized and updated through in an iterative process formulated in eq. (12.23). When the number of iteration, i.e., K is large enough, $\mathbf{x}$ will converge to a data sampled from the distribution, this can also be interpreted as a generative process.

12.2.3 Problems of Vanilla Score Matching

There are two main problems of vanilla score matching. **(a)** The score function in eq. (12.22) is ill-defined, since small value of $p(\mathbf{x})$ leads to larger error. **(b)** Langevin dynamics sampling may not mix, even if it is performed using the ground truth scores. Suppose that the true data distribution is a mixture of two disjoint distributions, i.e.,

$$p(\mathbf{x}) = c_1 \cdot p_1(\mathbf{x}) + c_2 \cdot p_2(\mathbf{x}). \tag{12.24}$$

Then, when the score is computed, these mixing coefficients are lost, since the log operation splits the coefficient from the distribution and the gradient operation zeros it out. Langevin dynamics sampling from the depicted initialization point has a roughly equal chance of arriving at each mode, despite the bottom right mode having a higher weight in the actual Mixture of Gaussians.

12.2.4 Solution: Adding Multiple Levels of Gaussian Noise

By adding multiple levels of Gaussian noise, there are several advantages. **(a)** A perturbed data will not be confined to a low-dimensional manifold, since the support of a Gaussian noise distribution is the entire space. **(b)** The area of each mode convering in the data distribution will be increased, adding more training signals in low density regions. **(c)** The intermediate distributions that respect ground truth mixing coefficients can be obtained.

Given a positive sequence of noise levels $\{\sigma_t\}_{t=1}^T$, a sequence of progressively perturbed data distribution can be formulated as,

$$p_{\sigma_t}(\mathbf{x}_t) = \int p(\mathbf{x}) \cdot \mathcal{N}(\mathbf{x}_t; \mathbf{x}, \sigma_t^2 \mathbf{I}) \mathrm{d}\mathbf{x}. \tag{12.25}$$

12.3 Unify VDM and SMLD

12.3.1 Unified Understanding from Two Perspectives

The Variational Diffusion Models (VDM) and Score Matching Langevin Dynamics (SMLD) can be unfied from the following two perspectives. **(a) Training Objective.** A neural network $s_\theta(\mathbf{x}_t, t)$ can be trained using score matching to learn the score function for all noise levels simultaneously, i.e.,

$$\arg\min_\theta \sum_{t=1}^T \lambda(t) \cdot \mathbb{E}_{p_{\sigma_t}(\mathbf{x}_t)} \left[||s_\theta(\mathbf{x}, t) - \nabla \log p_{\sigma_t}(\mathbf{x}_t)||_2^2 \right], \tag{12.26}$$

where $\lambda(t)$ is a positive weighting function that conditions on noise level t. The training objective of SMLD in eq. (12.26) is equivalent to the training objective of VDM in eq. (12.19). **(b) Sampling Procedure.** In Langevin dynamics sampling of score-based models, the samples eventually converge into a true mode with the noise levels steadily decreasing over timesteps. In the sampling performed in the MHVAE interpretation of a VDM, a randomly initialized data vector is iteratively refined over decreasing noise levels. This indicates the unified sampling procedure of VDM and SMLD.

12.3.2 Unify with SDE

Different Stochastic Differential Equations (SDEs) represent different form of adding noise. With SDE, VDM and SMLD can be written as SDE form, in which the Lagrangian form can be written as,

$$\mathrm{d}\mathbf{x} = f(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}W, \tag{12.27}$$

where $f(\cdot)$ is drift coefficient, and $g(\cdot)$ is diffusion coefficient, W is a standard Brownian motion, and $\mathrm{d}W$ is a noise.
(a) VDM. The adding noise process in VDM can be formulated as,

$$\mathbf{x}_t = \sqrt{\alpha_t} \cdot \mathbf{x}_{t-1} + \sqrt{1 - \alpha_t} \cdot \epsilon. \tag{12.28}$$

When $K \to \infty$, we have,

$$\begin{aligned}\mathbf{x}(t+\Delta t) &= \sqrt{1-\alpha(t+\Delta t)\Delta t}\mathbf{x}(t) + \sqrt{\alpha(t+\Delta t)\Delta t}\epsilon(t)\\ &\approx \mathbf{x}(t) - \frac{1}{2}\alpha(t+\Delta t)\mathbf{x}(t)\Delta t + \sqrt{\alpha(t+\Delta t)\Delta t}\epsilon(t)\\ &\approx \mathbf{x}(t) - \frac{1}{2}\alpha(t)\mathbf{x}(t)\Delta t + \sqrt{\alpha(t)\Delta t}\epsilon(t).\end{aligned} \tag{12.29}$$

Then, we have,

$$\begin{aligned}\mathbf{x}(t+\Delta t) - \mathbf{x}(t) &= -\frac{1}{2}\alpha(t)\mathbf{x}(t)\Delta t + \sqrt{\alpha(t)\Delta t}\epsilon(t)\\ \frac{\mathbf{x}(t+\Delta t) - \mathbf{x}(t)}{\Delta t} &= \underbrace{\mathrm{d}\mathbf{x} = -\frac{1}{2}\alpha(t)\mathbf{x}\mathrm{d}t + \sqrt{\alpha(t)}\mathrm{d}W}_{\text{SDE}}\end{aligned} \tag{12.30}$$

(b) SMLD. The adding noise process in VDM can be formulated as,

$$\mathbf{x}_t == \mathbf{x}_t + \sigma_t\mathbf{z}. \tag{12.31}$$

When $K \to \infty$, we have,

$$\begin{aligned}\mathbf{x}(t+\Delta t) &= \mathbf{x}(t) + \sqrt{\sigma^2(t+\Delta t) - \sigma^2(t)}\mathbf{z}(t)\\ &\approx \mathbf{x}(t) + \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}\Delta t}\mathbf{z}(t)\\ &= \mathbf{x}(t) + \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}\sqrt{\Delta t}\mathbf{z}(t).\end{aligned} \tag{12.32}$$

Then, we have,

$$\begin{aligned}\mathbf{x}(t+\Delta t) - \mathbf{x}(t) &= \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}\sqrt{\Delta t}\mathbf{z}(t)\\ \frac{\mathbf{x}(t+\Delta t) - \mathbf{x}(t)}{\Delta t} &= \underbrace{\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}\mathrm{d}W}_{\text{SDE}}.\end{aligned} \tag{12.33}$$

According to eq. (12.32) and eq. (12.33), we can conclude that the VDM and SMLD can be unified with SDE.

12.3.3 From SDE to ODE

Consider the movement of particles in a domain, the Lagrangian form focuses on how each particle moves, which is formulated as,

$$\mathrm{d}\mathbf{x} = f(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}W. \tag{12.34}$$

The Euler form focuses on the transition of density $\rho(\mathbf{x}, t)$, which is given by,

$$\mathrm{d}\mathbf{x} = \left[f(\mathbf{x}, t) - \frac{1}{2}(g^2(t) - \sigma^2(t))\nabla_{\mathbf{x}}\log p_t(\mathbf{x})\right]\mathrm{d}t + \sigma(t)\mathrm{d}W. \tag{12.35}$$

Since two SDEs in eq. (12.34) and eq. (12.35) are equivalent, their $p_t(\mathbf{x})$ are the same, which means that we can alter the variance $\sigma(t)$ of the second SDE. When the variance is set to zero, i.e., $\sigma(t) = 0$, we can obtain an Ordinary Differential Equation (ODE), i.e.,

$$\mathrm{d}\mathbf{x} = \left[f(\mathbf{x}, t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}}\log p_t(\mathbf{x})\right]\mathrm{d}t. \tag{12.36}$$

13.1 DDIM

13.1.1 Background

Given samples from a data distribution $q(x_0)$, we are interested in learning a model distribution $p_\theta(x_0)$ that approximates $q(x_0)$ and is easy to sample from. Denoising diffusion probabilistic models (DDPMs) are latent variable models of the form:

$$p_\theta(x_0) = \int p_\theta(x_{0:T})\, dx_{1:T}, \quad p_\theta(x_{0:T}) := p_\theta(x_T) \prod_{t=1}^{T} p_\theta^{(t)}(x_{t-1} \mid x_t). \tag{13.1}$$

where $x_1, \dots, x_T$ are latent variables in the same sample space as x_0 (denoted as $\mathcal{X}$). The parameters θ are learned to fit the data distribution $q(x_0)$ by maximizing a variational lower bound:

$$\max_\theta \mathbb{E}_{q(x_0)}[\log p_\theta(x_0)] \;\le\; \max_\theta \mathbb{E}_{q(x_{0:T})}\big[\log p_\theta(x_{0:T}) - \log q(x_{1:T} \mid x_0)\big], \tag{13.2}$$

where $q(x_{1:T} \mid x_0)$ is some inference distribution over the latent variables. Unlike typical latent variable models (such as the variational autoencoder), DDPMs are learned with a fixed (rather than trainable) inference procedure $q(x_{1:T} \mid x_0)$, and latent variables are relatively high dimensional. For example, DDPM considered the following Markov chain with Gaussian transitions parameterized by a decreasing sequence $\alpha_{1:T} \in (0,1]^T$:

$$q(x_{1:T} \mid x_0) := \prod_{t=1}^{T} q(x_t \mid x_{t-1}), \quad q(x_t \mid x_{t-1}) := \mathcal{N}\Big(\sqrt{\tfrac{\alpha_t}{\alpha_{t-1}}}\, x_{t-1},\; \big(1 - \tfrac{\alpha_t}{\alpha_{t-1}}\big)\, I\Big). \tag{13.3}$$

where the covariance matrix is ensured to have positive terms on its diagonal. This is called the forward process due to the autoregressive nature of the sampling procedure (from x_0 to x_T). We call the latent variable model $p_\theta(x_{0:T})$, which is a Markov chain that samples from x_T to x_0, the generative process, since it approximates the intractable reverse process $q(x_{1:T}|x_0)$. Intuitively, the forward process progressively adds noise to the observation x_0, whereas the generative process progressively denoises a noisy observation.

A special property of the forward process is that:

$$q(x_t \mid x_0) := \int q(x_t \mid x_{t-1})\, q(x_{t-1} \mid x_0)\, dx_{t-1} = \mathcal{N}(x_t;\, \sqrt{\alpha_t}\, x_0,\, (1-\alpha_t)\, I). \tag{13.4}$$

When $\alpha_T \approx 0$, $q(x_T \mid x_0) \approx \mathcal{N}(0, I)$, so one sets $p_\theta(x_T) = \mathcal{N}(0, I)$. If all the conditionals are modeled as Gaussians with trainable mean functions and fixed variances, the objective in Eq. (13.2) can be simplified to:

$$\mathcal{L}_\gamma(\epsilon_\theta) := \sum_{t=1}^{T} \gamma_t\, \mathbb{E}_{\substack{x_0 \sim q(x_0)\\ \epsilon_t \sim \mathcal{N}(0,I)}} \big\|\epsilon_\theta^{(t)}\big(\sqrt{\alpha_t}\, x_0 + \sqrt{1-\alpha_t}\, \epsilon_t\big) - \epsilon_t\big\|_2^2, \tag{13.5}$$

where $\epsilon_\theta^{(t)} := \epsilon_\theta^{(t)}(x_t)$ is a set of T functions, each $\epsilon_\theta^{(t)} : \mathcal{X} \to \mathcal{X}$ (indexed by t) with trainable parameters $\theta^{(t)}$, and $\gamma := [\gamma_1, \dots, \gamma_T]$ is a vector of positive coefficients in the objective that depends on $\alpha_{1:T}$. **Practical ML insight:** The noise schedule $\alpha_{1:T}$ in DDPM/DDIM acts like a *curriculum*: early steps (large α_t) are easy (little noise), later steps harder (more noise). In practice, learning benefits if one trains with a *progressively increasing noise level*, akin to curriculum learning in deep nets. Moreover, the choice of weighting γ_t parallels the reweighting of per-step losses in multi-task learning, trading off bias and variance.

13.1.2 Variational Inference for Non-Markovian Forward Processes

Because the generative model approximates the reverse of the inference process, we need to rethink the inference process in order to reduce the number of iterations required by the generative model. Our key observation is that the DDPM objective in the form of $\mathcal{L}_\gamma$ only depends on the marginals $q(x_t \mid x_0)$, but not directly on the joint $q(x_{1:T} \mid x_0)$. Since there are many inference distributions (joints) with the same marginals, we explore non-Markovian alternative inference processes, which leads to new generative processes. These non-Markovian inference processes lead to the same surrogate objective function as DDPMs, as we will show below.

Non-Markovian Forward Processes

Let us consider a family $\mathcal{Q}$ of inference distributions, indexed by a real vector $\sigma \in \mathbb{R}^T_{\geq 0}$:

$$q_\sigma(x_{1:T} \mid x_0) := q_\sigma(x_T \mid x_0) \prod_{t=2}^{T} q_\sigma(x_{t-1} \mid x_t, x_0), \tag{13.6}$$

where

$$q_\sigma(x_T \mid x_0) = \mathcal{N}\big(\sqrt{\alpha_T}\, x_0,\ (1-\alpha_T)\, I\big)$$

and for all $t > 1$,

$$q_\sigma(x_{t-1} \mid x_t, x_0) = \mathcal{N}\Big(\sqrt{\alpha_{t-1}}\, x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\ \frac{x_t - \sqrt{\alpha_t}\, x_0}{\sqrt{1-\alpha_t}}\ ,\ \sigma_t^2\, I\Big)\,. \tag{13.7}$$

The mean function is chosen in order to ensure that $q_\sigma(x_t \mid x_0) = \mathcal{N}(\sqrt{\alpha_t}\, x_0,\ (1-\alpha_t)I)$ for all t, so that it defines a joint inference distribution that matches the "marginals" as desired. The forward process can be derived from Bayes' rule:

$$q_\sigma(x_{t-1} \mid x_t, x_0) = \frac{q_\sigma(x_t \mid x_{t-1}, x_0)\; q_\sigma(x_{t-1} \mid x_0)}{q_\sigma(x_t \mid x_0)}. \tag{13.8}$$

which is also Gaussian (although we do not use this fact for the remainder of this paper). Unlike the diffusion process in Eq. (13.3), the forward process here is no longer Markovian, since each x_t could depend on both x_{t-1} and x_0. The magnitude of σ controls the degree of stochasticity of the forward process; when $\sigma \to 0$, we reach an extreme case where as long as we observe x_0 and x_t for some t, then x_{t-1} becomes known and fixed.

Generative Process and Unified Variational Inference Objective

Next, we define a trainable generative process $p_\theta(x_{0:T})$ where each $p_\theta^{(t)}(x_{t-1} \mid x_t)$ leverages knowledge of $q_\sigma(x_{t-1} \mid x_t, x_0)$. Intuitively, given a noisy observation x_t, we first predict the corresponding x_0, and then use it to obtain a sample x_{t-1} through the reverse conditional distribution $q_\sigma(x_{t-1} \mid x_t, x_0)$, which we have defined.

For some $x_0 \sim q(x_0)$ and $\epsilon_t \sim \mathcal{N}(0, I)$, x_t can be obtained using Eq. (13.4). The model $\epsilon_\theta^{(t)}(x_t)$ then attempts to predict ϵ_t from x_t, without knowledge of x_0. By rewriting Eq. (13.4), one can then predict the denoised observation, which is a prediction of x_0 given x_t:

$$f_\theta^{(t)}(x_t) \ :=\ \frac{x_t \ -\ \sqrt{1-\alpha_t}\, \epsilon_\theta^{(t)}(x_t)}{\sqrt{\alpha_t}}. \tag{13.9}$$

We can then define the generative process with a fixed prior $p_\theta(x_T) = \mathcal{N}(0, I)$ and

$$p_\theta^{(t)}(x_{t-1} \mid x_t) = \begin{cases} \mathcal{N}\big(f_\theta^{(1)}(x_t),\ \sigma_t^2 I\big), & t = 1, \\ q_\sigma\big(x_{t-1} \mid x_t,\ f_\theta^{(t)}(x_t)\big), & \text{otherwise.} \end{cases} \tag{13.10}$$

where $q_\sigma\big(x_{t-1} \mid x_t,\ f_\theta^{(t)}(x_t)\big)$ is defined as in Eq. (13.7) with x_0 replaced by $f_\theta^{(t)}(x_t)$. We add some Gaussian noise (with covariance $\sigma_t^2 I$) for the case of $t = 1$ to ensure that the generative process is supported everywhere.

We optimize θ via the following variational inference objective (which is functional over ϵ_0):

$$J_\sigma(\epsilon_0) \ :=\ \mathbb{E}_{x_0 \sim q(x_0)}\big[\log q_\sigma(x_{1:T} \mid x_0) \ -\ \log p_\theta(x_{0:T})\big]. \tag{13.11}$$

By factorizing $q_\sigma(x_{1:T} \mid x_0)$ and $p_\theta(x_{0:T})$ we obtain

$$\begin{aligned} J_\sigma(\epsilon_0) = \mathbb{E}_{x_0 \sim q(x_0)} \Big[&\log q_\sigma(x_T \mid x_0) + \sum_{t=2}^{T} \log q_\sigma(x_{t-1} \mid x_t, x_0) \\ &- \sum_{t=1}^{T} \log p_\theta^{(t)}(x_{t-1} \mid x_t) \ -\ \log p_\theta(x_T)\Big], \end{aligned} \tag{13.12}$$

where we factorize $q_\sigma(x_{1:T} \mid x_0)$ according to Eq. (13.6) and $p_\theta(x_{0:T})$ according to Eq. (13.1).

13.1.3 Denoising Diffusion Implicit Models

From $p_\theta(x_{1:T})$ in Eq. (13.10), one can generate a sample x_{t-1} from a sample x_t via:

$$x_{t-1} = \sqrt{\alpha_{t-1}}\Big(\frac{x_t - \sqrt{1-\alpha_t}\,\epsilon_\theta^{(t)}(x_t)}{\sqrt{\alpha_t}}\Big) + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\,\epsilon_\theta^{(t)}(x_t) + \sigma_t\,\epsilon_t\,, \tag{13.13}$$

where $\epsilon_t \sim \mathcal{N}(0, I)$ is standard Gaussian noise independent of x_t, and we define $\alpha_0 = 1$. Different choices of σ or values result in different generative processes, all while using the same model ϵ_0, so re-training the model is unnecessary. When

$$\sigma_t = \sqrt{\frac{1-\alpha_{t-1}}{1-\alpha_t}}\sqrt{\frac{1-\alpha_t}{\alpha_{t-1}}} \quad \text{for all } t,$$

the forward process becomes Markovian, and the generative process becomes a DDPM.

We note another special case when $\sigma_t = 0$ for all t: the forward process becomes deterministic given x_{t-1} and x_0, except for $t = 1$; in the generative process, the coefficient before the random noise ϵ_t becomes zero. The resulting model becomes an implicit probabilistic model, where samples are generated from latent variables with a fixed procedure (from x_T to x_0). We name this the denoising diffusion implicit model (DDIM) because it is an implicit probabilistic model trained with the DDPM objective (despite the fact that the forward process is no longer a diffusion).

13.2 Flow Matching

13.2.1 Preliminaries: Continuous Normalizing Flows

Let $\mathbb{R}^d$ denote the data space with data points

$$x = (x^1, \ldots, x^d) \in \mathbb{R}^d.$$

Two important objects we use in this paper are:

- the *probability density path*
$$p : [0,1] \times \mathbb{R}^d \longrightarrow \mathbb{R}_{>0},$$
which is a time-dependent probability density function, i.e. $\int p_t(x)\,\mathrm{d}x = 1$ for all $t \in [0,1]$, and
- a time-dependent vector field
$$v : [0,1] \times \mathbb{R}^d \longrightarrow \mathbb{R}^d.$$

A vector field $v_t(\cdot) = v(t, \cdot)$ can be used to construct a time-dependent diffeomorphic map, called a *flow*,

$$\phi : [0,1] \times \mathbb{R}^d \longrightarrow \mathbb{R}^d,$$

defined via the ordinary differential equation (ODE):

$$\frac{\mathrm{d}}{\mathrm{d}t}\,\phi_t(x) = v_t\big(\phi_t(x)\big), \quad \phi_0(x) = x. \tag{13.14}$$

A Continuous Normalizing Flow (CNF) can be used to reshape a simple prior density p_0 (e.g., pure noise) to a more complicated one, p_1, via the push-forward equation:

$$p_t \;=\; [\phi_t]_* p_0, \tag{13.15}$$

where the push-forward (or change of variables) operator $*$ is defined by:

$$[\phi_t]_* p_0(x) \;=\; p_0\big(\phi_t^{-1}(x)\big)\ \det\Big[\frac{\partial \phi_t^{-1}}{\partial x}\Big]. \tag{13.16}$$

A vector field v_t is said to *generate* a probability density path p_t if its flow ϕ_t satisfies equation (13.15).

13.2.2 Flow Matching

Let x_1 denote a random variable distributed according to some unknown data distribution $q(x_1)$. We assume we only have access to data samples from $q(x_1)$ but no access to the density function itself. Furthermore, let p_t be a probability path such that $p_0 = p$ is a simple distribution, e.g. the standard normal distribution $p(x) = \mathcal{N}(x \mid 0, I)$, and let p_1 be approximately equal in distribution to q. We will later discuss how to construct such a path. The Flow Matching objective is then designed to match this target probability path, which will allow us to flow from p_0 to p_1.

We define the Flow Matching (FM) objective as:

$$\mathcal{L}_{\mathrm{FM}}(\theta) \;=\; \mathbb{E}_{t\sim U[0,1],\, x\sim p_t(x)}\left[\| \, v_t(x) - u_t(x)\|^2\right], \tag{13.17}$$

where θ denotes the learnable parameters of the CNF vector field v_t, $t \sim U[0,1]$ is drawn uniformly, and $x \sim p_t(x)$. Simply put, the FM loss regresses the vector field u_t with a neural network v_t. Upon reaching zero loss, the learned CNF model will generate samples from $p_t(x)$.

Flow Matching is a simple and attractive objective, but naively on its own it is intractable to use in practice since we have no prior knowledge for what an appropriate p_t and u_t are.

13.2.3 Constructing p_t, u_t from Conditional Probability Paths and Vector Fields

A simple way to construct a target probability path is via a mixture of simpler probability paths. Given a particular data sample x_1, we denote by $p_t(x \mid x_1)$ a conditional probability path such that $p_0(x \mid x_1) = p(x) \quad$ at t=0, and we design $p_1(x \mid x_1)$ at $t = 1$ to be a distribution concentrated around $x = x_1$, e.g. $p_1(x \mid x_1) = \mathcal{N}(x \mid x_1, \sigma^2 I)$, a normal distribution with mean x_1 and sufficiently small standard deviation $\sigma > 0$.

Marginalizing the conditional probability paths over $q(x_1)$ gives rise to the marginal probability path:

$$p_t(x) = \int p_t(x \mid x_1)\, q(x_1)\, \mathrm{d}x_1. \tag{13.18}$$

Interestingly, we can also define a marginal vector field by "marginalizing" over the conditional vector fields in the following sense (we assume $p_t(x) > 0$ for all t and x):

$$u_t(x) = \int u_t(x \mid x_1)\, \frac{p_t(x \mid x_1)\, q(x_1)}{p_t(x)}\, \mathrm{d}x_1. \tag{13.19}$$

Theorem 12.1 *Given vector fields $u_t(x \mid x_1)$ that generate conditional probability paths $p_t(x \mid x_1)$ for any distribution $q(x_1)$, the marginal vector field u_t in Eq. (13.19) generates the marginal probability path p_t in Eq. (13.18).*

13.2.4 Conditional Flow Matching

Unfortunately, due to the intractable integrals in the definitions of the marginal probability path and vector field (Eqs. (13.18) and (13.19)), it is still intractable to compute u_t, and consequently intractable to naively compute an unbiased estimator of the original Flow Matching objective. Instead, we propose a simpler objective, which surprisingly will result in the same optima as the original objective. Specifically, we consider the Conditional Flow Matching (CFM) objective:

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{\,t\sim U[0,1],\, x_1\sim q(x_1),\, x\sim p_t(x|x_1)}\left[\| \, v_t(x) - u_t(x \mid x_1)\|^2\right], \tag{13.20}$$

where $t \sim U[0,1]$, $x_1 \sim q(x_1)$, and $x \sim p_t(x \mid x_1)$. Unlike the FM objective, the CFM objective allows us to easily sample unbiased estimates as long as we can efficiently sample from $p_t(x \mid x_1)$ and compute $u_t(x \mid x_1)$, both of which can be done on a per-sample basis.

Our key observation is therefore:

The FM (Eq. (13.17)) and CFM (Eq. (13.20)) objectives have identical gradients w.r.t. θ.

That is, optimizing the CFM objective is equivalent (in expectation) to optimizing the FM objective. Consequently, this allows us to train a CNF to generate the marginal probability path p_t—which in particular approximates the unknown data distribution q at $t = 1$—without ever needing access to either the marginal probability path or the marginal vector field. We simply need to design suitable conditional probability paths and vector fields. We formalize this property in the following theorem.

Theorem 12.2 *Assuming that $p_t(x) > 0$ for all $x \in \mathbb{R}^d$ and $t \in [0,1]$, then, up to a constant independent of θ, $\mathcal{L}_{\mathrm{CFM}}$ and $\mathcal{L}_{\mathrm{FM}}$ are equal. Hence,*

$$\nabla_\theta \mathcal{L}_{\mathrm{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\mathrm{CFM}}(\theta).$$

13.2.5 Gaussian Conditional Probability Paths and Vector Fields

The CFM objective works with any choice of conditional probability path and vector field $u_t(x \mid x_1)$. In this section, we discuss the construction of $p_t(x \mid x_1)$ and $u_t(x \mid x_1)$ with a particular focus on conditional probability paths. Namely, we consider conditional Gaussian paths:

$$p_t(x \mid x_1) = \mathcal{N}\big(x \mid \mu_t(x_1),\, \sigma_t(x_1)^2 I\big), \tag{13.21}$$

where $\mu_t(x_1)$ is the time-dependent mean of the Gaussian distribution, while $\sigma_t(x_1)$ is a time-dependent scalar standard deviation (std). We set $\mu_0(x_1) = 0$ and design conditional paths such that all conditional probability paths converge to the same standard Gaussian at $t = 0$, e.g. $\mathcal{N}(x \mid 0, I)$. We then set $\mu_1(x_1) = x_1$ and $\sigma_1(x_1) = \sigma_{\min}$, which ensures that $p_1(x \mid x_1)$ is a concentrated Gaussian distribution centered at x_1.

Let $u_t(x \mid x_1)$ denote the vector field that generates the conditional probability path:

$$\frac{\mathrm{d}}{\mathrm{d}t}\,\phi_t(x) \;=\; u_t\big(\phi_t(x) \mid x_1\big). \tag{13.22}$$

Plugging equation (13.22) into the CFM loss we obtain:

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{\substack{t\sim U[0,1]\\ x_1\sim q(x_1)\\ x_0\sim p_0(x)}} \Big[\big\|\, v_t\big(\psi_t(x_0)\big) - \tfrac{\mathrm{d}}{\mathrm{d}t}\,\psi_t(x_0)\big\|^2\Big]. \tag{13.23}$$

Since ψ_t is a simple (invertible) affine map, we can use equation (13.22) to solve for u_t in closed form.

13.3 Rectified flow

Given empirical observations $X_0 \sim \pi_0$, $X_1 \sim \pi_1$, the rectified flow induced from (X_0, X_1) is an ordinary differentiable model (ODE) on time $t \in [0,1]$:

$$\mathrm{d}Z_t = v(Z_t, t)\,\mathrm{d}t,$$

which converts Z_0 from π_0 to Z_1 following π_1. The drift force

$$v : \mathbb{R}^d \to \mathbb{R}^d$$

is set to drive the flow to follow the direction $(X_1 - X_0)$ of the linear path pointing from X_0 to X_1 as much as possible, by solving a simple least-squares regression problem:

$$\min_v \int_0^1 \mathbb{E}\big[\|\,(X_1 - X_0) \;-\; v(X_t,t)\|^2\big]\,\mathrm{d}t, \quad \text{with} \quad X_t = t\,X_1 + (1-t)\,X_0. \tag{13.24}$$

where X_t is the linear interpolation of X_0 and X_1. Naively, X_t follows the ODE

$$\mathrm{d}X_t = (X_1 - X_0)\,\mathrm{d}t,$$

which is non-causal (or anticipating) as the update of X_t requires the information of the final point X_1. By fitting the drift v with $X_1 - X_0$, the rectified flow *causalizes* the linear interpolation X_t paths, yielding an ODE flow that can be simulated without seeing the future.

In practice, we parameterize v with a neural network or other nonlinear models and solve Eq. (13.24) with any off-the-shelf stochastic optimizer, such as stochastic gradient descent, with empirical draws of (X_0, X_1). See the rectified flow algorithm.

Algorithm 1. After we get v, we solve the ODE starting from $Z_0 \sim \pi_0$ to transfer π_0 to π_1, backwardly starting from $Z_1 \sim \pi_1$ to transfer π_1 to π_0. Specifically, for backward sampling, we simply ... Solve

$$\mathrm{d}\tilde{X}_t = -\, v(\tilde{X}_t, t)\,\mathrm{d}t, \quad \tilde{X}_0 \sim \pi_1, \quad \tilde{X}_t = \tilde{X}_{1-t}.$$

The forward and backward sampling are equally favored by the training algorithm because the objective in Eq. (13.24) is *time-symmetric* in that it yields the equivalent problem if we exchange X_0 and X_1 and flip the sign of v.

Flows Avoid Crossing

A key to understanding the method is the non-crossing property of flows: the different paths following a well-defined ODE

$$\mathrm{d}Z_t = v(Z_t, t)\,\mathrm{d}t,$$

whose solution exists and is unique, cannot cross each other at any time $t \in [0, 1]$. Specifically, there exists no location $z \in \mathbb{R}^d$ and time $t \in [0, 1]$ such that two paths go across z at time t along different directions, because otherwise the solution of the ODE would be non-unique. On the other hand, the paths of the interpolation process X_t may intersect with each other, which makes it non-causal. Hence, the rectified flow *rewires* the individual trajectories passing through the interpolation points to avoid crossing while tracing out the same density map as the linear interpolation paths due to the optimization of Eq. (13.24). We can view the linear interpolation X_t as building roads (or tunnels) to connect π_0 and π_1, and the rectified flow as traffic of particles passing through the roads in a myopic, memoryless, non-crossing way, which allows them to ignore the global path information of how X_0 and X_1 are paired and rebuild a more deterministic pairing of (Z_0, Z_1).

Rectified Flows Reduce Transport Costs

If Eq. (13.24) is solved exactly, the pair (Z_0, Z_1) of the rectified flow is guaranteed to be a valid coupling of π_0, π_1 (Theorem 3.3), that is, $Z_1 \sim \pi_1$ if $Z_0 \sim \pi_0$. Moreover, (Z_0, Z_1) yields no larger transport cost than the data pair (X_0, X_1) simultaneously for all convex cost functions c (Theorem 3.5). The data pair (X_0, X_1) can be an arbitrary coupling of π_0, π_1, typically independent (i.e. $(X_0, X_1) \sim \pi_0 \times \pi_1$) as dictated by the lack of meaningfully paired observations in practice. In comparison, the rectified coupling (Z_0, Z_1) has a deterministic dependency since it is constructed from an ODE model. Denote by

$$(Z_0, Z_1) \;=\; \mathrm{Rectify}\big((X_0, X_1)\big)$$

the mapping from (X_0, X_1) to (Z_0, Z_1). Hence, Rectify converts an arbitrary coupling into a deterministic coupling with lower convex transport costs.

Straight Line Flows Yield Fast Simulation

Following Algorithm 1, denote by

$$Z \;=\; \mathrm{RectFlow}\big((X_0, X_1)\big)$$

the rectified flow induced from (X_0, X_1). Applying this operator recursively yields a sequence of rectified flows

$$Z^{k+1} \;=\; \mathrm{RectFlow}\big((Z_0^k, Z_1^k)\big), \quad (Z_0^0, Z_1^0) = (X_0, X_1),$$

where Z^k is the k-th rectified flow (or k-rectified flow) induced from (X_0, X_1).

This *reflow* procedure not only decreases transport cost but also straightens the flow paths, making them nearly linear. Such nearly straight flows incur small time-discretization error in numerical simulation; indeed, perfectly straight paths can be simulated exactly with a single Euler step, yielding an effective one-step model. This addresses the bottleneck of high inference cost in existing continuous-time ODE/SDE models.

13.4 Consistency Model

13.4.1 Background

Consistency models are heavily inspired by the theory of continuous-time diffusion models. Diffusion models generate data by progressively perturbing data to noise via Gaussian perturbations, then creating samples from noise via sequential denoising steps. Let $p_{\text{data}}(x)$ denote the data distribution. Diffusion models start by diffusing $p_{\text{data}}(x)$ with a stochastic differential equation (SDE):

$$\mathrm{d}x_t \;=\; \mu(x_t,t)\,\mathrm{d}t \;+\; \sigma(t)\,\mathrm{d}w_t, \tag{13.25}$$

where $t \in [0,T]$, $T > 0$ is a fixed constant, $\mu(\cdot,\cdot)$ and $\sigma(\cdot)$ are the drift and diffusion coefficients respectively, and $\{w_t\}_{t\in[0,T]}$ denotes the standard Brownian motion. we denote the law of x_t by $p_t(x)$, so that $p_0(x) \equiv p_{\text{data}}(x)$.

A remarkable property of this SDE is that there exists an equivalent ordinary differential equation (the *Probability Flow* ODE) whose solution $\{x_t\}$ has marginal p_t at time t:

$$\mathrm{d}x_t \;=\; \Big[\mu(x_t,t) \;-\; \tfrac{1}{2}\,\sigma(t)^2\,\nabla \log p_t(x_t)\Big]\,\mathrm{d}t. \tag{13.26}$$

Here $\nabla \log p_t(x)$ is the *score* of p_t.

Typically one chooses $\mu(x,t) = 0$ and $\sigma(t) = \sqrt{2t}$, so that

$$p_t(x) = p_{\text{data}}(x) \;\otimes\; \mathcal{N}(0,t^2 I), \qquad p_T(x) \approx \mathcal{N}(0,T^2 I).$$

We then train a neural score model $s_\phi(x,t) \approx \nabla \log p_t(x)$ by score matching, and plug it into (**??**) to obtain an empirical Probability Flow ODE of the form

$$\frac{\mathrm{d}x_t}{\mathrm{d}t} \;=\; -\,t\,s_\phi(x_t,t). \tag{13.27}$$

We call Eq. (13.27) the *empirical PF ODE.* Next, we sample

$$x_T \sim \pi = \mathcal{N}(0,\,T^2 I)$$

to initialize the empirical PF ODE and solve it backwards in time with any numerical ODE solver such as Euler or Heun to obtain the solution trajectory $\{x_t\}_{t\in[0,T]}$. The resulting x_0 can then be viewed as an approximate sample from the data distribution $p_{\text{data}}(x)$. To avoid numerical instability, one typically stops the solver at $t = \epsilon$, which is a fixed small positive number, and accepts x_ϵ as the approximate sample.

Definition Given a solution trajectory $\{x_t\}_{t\in[\epsilon,T]}$ of the PF ODE in Eq. (13.26), we define the *consistency function*

$$f : (x_t,t) \;\mapsto\; x_\epsilon.$$

A consistency function has the property of *self-consistency*: its outputs are consistent for arbitrary pairs (x_t,t) that belong to the same PF ODE trajectory, i.e.

$$f(x_t,t) \;=\; f(x_{t'},t') \quad \text{for all } t,t' \in [\epsilon,T].$$

The goal of a consistency model, symbolized as f_θ, is to estimate this consistency function f from data by learning to enforce the self-consistency property. Note that a similar definition is used for neural flows, and the invertibility constraint is not required here.

Parameterization For any consistency function $f(\cdot,\cdot)$, we have

$$f(x_\epsilon, \epsilon) \;=\; x_\epsilon,$$

i.e. $f(\cdot,\epsilon)$ is an identity function. We call this constraint the *boundary condition.* All consistency models have to meet this boundary condition, as it plays a crucial role in the successful training of consistency models.

$$f_\theta(x,t) = \begin{cases} x, & t = \epsilon, \\ F_\theta(x,t), & t \in (\epsilon, T], \end{cases} \tag{13.28}$$

$$f_\theta(x,t) = c_{\text{skip}}(t)\, x \;+\; c_{\text{out}}(t)\, F_\theta(x,t), \tag{13.29}$$

where $c_{\text{skip}}(t)$ and $c_{\text{out}}(t)$ are differentiable functions satisfying

$$c_{\text{skip}}(\epsilon) = 1, \quad c_{\text{out}}(\epsilon) = 0.$$

13.4.2 Training Consistency Models via Distillation

We present our first method for training consistency models based on distilling a pre-trained score model $s_\phi(x,t)$. Our discussion revolves around the empirical PF ODE in Eq. (13.27), obtained by plugging the score model $s_\phi(x,t)$ into the PF ODE. Consider discretizing the time horizon $[\epsilon, T]$ into $N-1$ sub-intervals, with boundaries

$$t_1 = \epsilon \;<\; t_2 \;<\; \cdots \;<\; t_N = T.$$

In practice, we determine the boundaries with the formula

$$t_i \;=\; \Big(\epsilon^{1/\rho} \;+\; \frac{i-1}{N-1}\big(T^{1/\rho} - \epsilon^{1/\rho}\big)\Big)^{\rho}, \quad \rho = 7. \tag{13.30}$$

When N is sufficiently large, we can obtain an accurate estimate of x_{t_n} from $x_{t_{n+1}}$ by running one discretization step of a numerical ODE solver. This estimate, which we denote as $\hat{x}^{\phi}_{t_n}$, is defined by

$$\hat{x}^{\phi}_{t_n} \;:=\; x_{t_{n+1}} \;+\; (t_n - t_{n+1})\, \Phi\big(x_{t_{n+1}}, t_{n+1}; \phi\big), \tag{13.31}$$

where $\Phi(\cdot)$ represents the update function of a one-step ODE solver applied to the empirical PF ODE. For example, when using the Euler solver, we have $\Phi(x,t;\phi) = -t\, s_\phi(x,t)$, which corresponds to the following update rule:

Definition 12.3 (Consistency distillation loss) *The consistency distillation loss is defined as*

$$\mathcal{L}^N_{\text{CD}}(\theta, \theta^-; \phi) := \mathbb{E}_{\substack{x\sim p_{\text{data}},\, n\sim\mathcal{U}[1,N-1],\\ x_{t_{n+1}}\sim\mathcal{N}(x;\, t^2_{n+1} I)}}\big[d\big(f_\theta(x_{t_{n+1}}, t_{n+1}),\, f_{\theta^-}(\hat{x}^{\phi}_{t_n}, t_n)\big)\big], \tag{13.32}$$

where $\mathcal{U}[1, N-1]$ denotes the uniform distribution over $\{1, 2, \ldots, N-1\}$, $\omega(t_n)$ is a positive weighting function, $\hat{x}^{\phi}_{t_n}$ is given by Eq. (13.31), θ^- denotes a running average of the past values of θ during optimization, and $d(\cdot,\cdot)$ is a metric satisfying $d(x,y) \geq 0$ and $d(x,y) = 0$ if and only if $x = y$.