COMP760:

Geometry and Generative Models

Week 4: Generative Models Primer II

Notes by: Joey Bose and Prakash Panangaden

1

DIFFUSION MODELS

Diffusion models represent a combination of many familiar elements to other likelihood generative models like Normalizing Flows and VAEs—in fact as we will they are intimately related—but they also enjoy generated sample quality and likelihood scores that rivals state of the art GANs and autoregressive models respectively. While there are a couple of different perspectives to diffusion models we will begin our treatment with perhaps the most conceptually simplest discrete time version [Sohl-Dickstein et al., 2015, Ho et al., 2020].

1.1 Denoising Diffusion Models

In a nutshell, diffusion models consist of two processes. The first one, dubbed the diffusion process, starts from a clean data sample $x \sim q(x_0)$ and progressively injects noise (e.g. Gaussian noise) at every time step $i \in [T]$ until a terminal step T where the resulting sample consists of pure noise. The reverse generative process takes a sample from a noise distribution and denoises the sample by progressively adding back structure until we return to a sample that resembles being drawn from the empirical data distribution $p(x)$. Both the diffusion and denoising processes are illustrated below for natural images in Fig. 1.1.

Forward Diffusion Process. Let us define $q(x_t|x_{t-1})$ as the forward diffusion distribution modelled as a Gaussian centered around the sample at the previous timestep x_{t-1}:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I), \tag{1.1}$$

where β_t is a known variance schedule. As (1.1) is a conditional Gaussian with mean $\mu_t = \sqrt{1-\beta_t}x_{t-1}$ and variance $\sigma_t^2 = \beta_t I$. Sampling from this distribution is then simply: $x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}$, where $\epsilon_{t-1} \sim \mathcal{N}(0, I)$. Note that this is a reparametrized sample in the sense of the reparametrization trick common in VAEs, but sometimes we will push β_t into the variance of ϵ_t to allow for convenient algebraic manipulation later. Given a starting point, x_0 which we take as the clean sample the joint

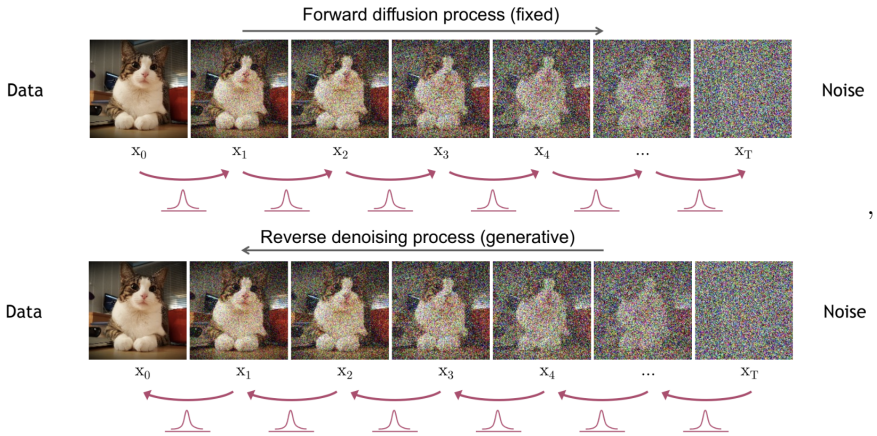

Figure 1.1: **Top:** Forward diffusion process that progressively corrupts data. **Bot:** Reverse generative process that denoises a data point. Figure taken from Kreis et al. [2022].

distribution over all timesteps can be written as follows,

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}). \tag{1.2}$$

Now, what if we wish to craft a noisy sample $x_t \sim q(x_t|x_0)$ at a specific timestep? Naively, we could run the forward diffusion process for t steps but this is wasteful. Instead, we can recognize that our forward diffusion process progressively adds Gaussian noise which enables us to use a few clever algebraic tricks and sample directly from $q(x_t|x_0)$. First let us define $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t}(\alpha_i)$, then a sample at timestep t is:

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \tag{1.3}$$

$$= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon_{t-2} \tag{1.4}$$

$$= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\alpha_t}\epsilon_0. \tag{1.5}$$

Here the main trick used in the middle line comes merging two Gaussian's with different variances $\mathcal{N}(0, \sigma_1^2 I)$, and $\mathcal{N}(0, \sigma_2^2 I)$ which has a new distribution as $\mathcal{N}(0, (\sigma_1^2 + \sigma_2^2)I)$. Applying this trick to the middle line we get $\sqrt{1-\alpha_t + \alpha_t(1-\alpha_{t-1})} = \sqrt{1-\alpha_t\alpha_{t-1}}$. Using this result we can write the Gaussian conditioned on x_0 at timestep t as $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$. It is important to note that the noise schedule β_t is designed such that the terminal conditional distribution $q(x_T|x_0) \approx \mathcal{N}(0, I)$. Finally, we

can consider the marginal noisy distribution $q(x_t)$ which we can write using our forward diffusion conditional:

$$q(x_t) = \int q(x_0, x_t) dx_0 = \int q(x_0) q(x_t|x_0) dx_0. \tag{1.6}$$

Reverse Denoising Process. It is tempting to think that the reverse process—the one that takes noise to real data—can be easily performed with access to $q(x_{t-1}|x_t)$ for each timestep. But herein, lies the computational difficulty as $q(x_{t-1}|x_t) = \frac{q(x_t|x_{t-1})q(x_{t-1})}{q(x_t)}$ which needs marginals $q(x_t)$ and $q(x_{t-1})$. As we see in (1.6) computing these marginals requires integrating over the entire support of $q(x_0)$ but we do not have access to this distribution in practice. Instead, we only have samples—i.e. the dataset for training—and thus computing this integral is computationally infeasible. A natural question if we cannot exactly compute $q(x_{t-1}|x_t)$ is whether we can approximate it. Luckily, if β_t in the forward diffusion is small enough then $q(x_{t-1}|x_t) \propto q(x_t|x_{t-1})q(x_{t-1})$ is also Gaussian.

Note: We can match distributions using a variety of divergences or metrics, but the most common one in generative models is perhaps the KL-Divergence.

As is common practice in much of deep learning to approximate a distribution we can parametrize it using another distribution p_θ, with learnable parameters θ such that optimization leads us to "match" the distribution of p, q. Under the assumption of sufficiently small β_t we know that $q(x_{t-1}|x_t)$ is Gaussian and thus we can parametrize $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$ as a Gaussian with learned a mean μ_θ. We could also parametrize and learn the variance but let we can forego this for simplicity of exposition. Due to this parametrization, we can immediately state the following facts:

$$p(x_T) = \mathcal{N}(0, I) \tag{1.7}$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I) \tag{1.8}$$

$$p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t). \tag{1.9}$$

Training Diffusion Models. Now that we have defined p_θ we have all the components needed to train diffusion models. Ultimately we seek to model $p(x_0)$ which when fully trained approximates $q(x_0)$ and as a result, we can marginalize the joint distribution (1.9) and maximize the marginal log-likelihood of the data. Recall, that maximizing the log-likelihood—when possible—is a common design principle for crafting generative models, and as such being able to train diffusion models makes them attractive to GANs that need adversarial optimization.

$$\log p(x_0) = \log \int p(x_{0:T}) dx_{1:T} \tag{1.10}$$

$$= \log \int p(x_{0:T}) \frac{q(x_{1:T}|x_0)}{q(x_{1:T}|x_0)} dx_{1:T} \tag{1.11}$$

$$= \log \int q(x_{1:T}|x_0) p(x_T) \prod_{t=1}^{T} \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} dx_{1:T} \tag{1.12}$$

$$= \log \left(\mathbb{E}_{q(x_{1:T}|x_0)} [p(x_T) \prod_{t=1}^{T} \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}] \right) \tag{1.13}$$

We can now form a variational upper bound, similar to VAEs, on the marginal log-likelihood by leveraging Jensens inequality like so,

$$\log p(x_0) \leq \mathbb{E}_{q(x_{1:T}|x_0)} \left[\log \left(p(x_T) \prod_{t=1}^{T} \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \right) \right] := -\mathcal{L}. \tag{1.14}$$

We can also decompose (1.14) revealing the following terms of interest:

$$\mathcal{L} = -\mathbb{E}_{q(x_{1:T}|x_0)} \left[\log \left(p(x_T) \prod_{t=1}^{T} \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \right) \right] \tag{1.15}$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)} \left[-\log p(x_T) - \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \right] \tag{1.16}$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)} \left[-\log p(x_T) - \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} - \log \frac{p_\theta(x_0|x_1)}{q(x_1|x_0)} \right] \tag{1.17}$$

Now we employ a convenient trick by conditioning on x_0 for the conditional $q(x_t|x_{t-1})$ and then applying Bayes Rule. Note, that this is possible because x_0 is sampled from $q(x_0)$ which is our empirical data distribution, in other words, our training dataset.

Note that we conditioned on x_0 in (1.19) and applied Bayes Rule.

$$\mathcal{L} = \mathbb{E}_{q(x_{1:T}|x_0)}\Bigg[-\log p(x_T) - \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} \cdot \frac{q(x_{t-1}|x_0)}{q(x_t)|x_0)} - \log \frac{p_\theta(x_0|x_1)}{q(x_1|x_0)} \Bigg] \tag{1.18}$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)}\left[-\log \frac{p(x_T)}{q(x_T|x_0)} - \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} - \log p_\theta(x_0|x_1) \right] \tag{1.19}$$

$$= \mathbb{E}_q \Bigg[\underbrace{D_{KL}(p(x_T) \,||\, q(x_T|x_0))}_{\mathcal{L}_T} - \sum_{t=2}^{T} \underbrace{D_{KL}(p_\theta(x_{t-1}|x_t) \,||\, q(x_{t-1}|x_t, x_0))}_{\mathcal{L}_{t-1}} - \underbrace{\log p_\theta(x_0|x_1)}_{\mathcal{L}_0} \Bigg]. \tag{1.20}$$

Decomposing the variational upper bound as in (1.20) we see that we have a KL divergence between the initial denoising distribution $p(x_T)$ and $q(x_T|x_0)$ in $\mathcal{L}_T$, a per timestep KL divergence in $\mathcal{L}_{t-1}$ between the denoising distribution $p_\theta(x_{t-1}|x_t)$ and the forward diffusion distribution which is additionally conditioned on x_0, $q(x_{t-1}|x_t, x_0)$. It is important to note that both $\mathcal{L}_T$ and $\mathcal{L}_{t-1}$ are comparisons between two Gaussian distributions and thus can be computed in closed form. Moreover, $\mathcal{L}_T$ uses $p(x_T) = \mathcal{N}(0, I)$ which is *not* a function of any learnable parameters θ and does not contribute any gradient information which means we can simply ignore computing it. Finally, the last term $\mathcal{L}_0$ can be treated separately as in Ho et al. [2020] using a discrete decoder $\mathcal{N}(x_0; \mu_\theta(x_1, 1), \sigma_1^2 I$.

One important observation, and perhaps the main trick we used to get a computational benefit in (1.19) is conditioning on x_0 which makes the previously intractable $q(x_{t-1}|x_t, x_0)$ into a tractable conditional $q(x_{t-1}|x_t, x_0)$. To see this let $q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}, \tilde{\mu}(x_t, x_0), \tilde{\beta}_t)$ be the Gaussian whose parameters have the following convenient closed-form expression:

$$\mathcal{Q} := q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0)\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} \tag{1.21}$$

Note: After a bit of algebraic manipulation we get (1.23). Also $C(x_t, x_0)$ is a some function but crucially does not depend on x_{t-1}.

Expanding this we get:

$$\mathcal{Q} \propto \exp\left(-\frac{1}{2}\left(\frac{(x_t - \sqrt{\alpha_t}x_{t-1})^2}{\beta_t} + \frac{(x_{t-1} - \sqrt{1-\bar{\alpha}_{t-1}}x_0)^2}{1-\alpha_{t-1}} - \frac{(x_t - \sqrt{\bar{\alpha}_t}x_0)^2}{1-\bar{\alpha}_t}\right)\right) \tag{1.22}$$

$$= \exp\left(-\frac{1}{2}\left(\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha_{t-1}}}\right)x_{t-1}^2 - \left(\frac{2\sqrt{\alpha_t}}{\beta_t}x_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}x_0\right)x_{t-1} + C(x_t, x_0)\right)\right). \tag{1.23}$$

From (1.23) we begin to see some form of the Gaussian emerging but we can further clean this up by rearranging terms so that the mean $\tilde{\mu}(x_t, x_0)$ and variance $\tilde{\beta}_t$ become more explicit .

Note: For (1.25) we used $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$

$$\tilde{\beta}_t = 1/\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha_{t-1}}}\right) \tag{1.24}$$

$$= 1/\frac{\alpha_t - \bar{\alpha}_t + \beta_t}{\beta_t(1-\bar{\alpha}_{t-1})} \tag{1.25}$$

$$= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{1.26}$$

Similarly for the mean $\tilde{\mu}(x_t, x_0)$ we can use (1.26) to factor out $\tilde{\beta}_t$ and look at the middle term in (1.23) .

Note: In (1.29) we used $x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_t)$

$$\tilde{\mu}(x_t, x_0) = \left(\frac{\sqrt{\alpha_t}}{\beta_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\right) \cdot \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{1.27}$$

$$= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 \tag{1.28}$$

$$= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \left(\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\right) \cdot \left(\frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_t)\right) \tag{1.29}$$

$$= \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right) \tag{1.30}$$

Armed with (1.30) we can rewrite the per timestep loss $\mathcal{L}_{t-1}$. Recall that this objective is a KL-divergence between the two Gaussians which admits a closed form expression:

$$\mathcal{L}_{t-1} = \frac{1}{2}\mathbb{E}_q\left[\frac{1}{\sigma_t^2}||\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)||^2\right] + C, \tag{1.31}$$

where C is a constant that does not depend on θ. The above equation also suggests a nice parametrization for μ_θ if we notice that in (1.30) we can predict the noise level at timestep t. That is instead of a mean prediction network we can parametrize μ_θ to be a noise prediction network $\epsilon_\theta(x_t, t)$ that predicts the amount of noise in x_t.

$$\mu_\theta = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right). \tag{1.32}$$

Putting everything together in $\mathcal{L}_{t-1}$,

$$\begin{aligned}\mathcal{L}_{t-1} &= \frac{1}{2}\mathbb{E}_q\left[\frac{1}{\sigma_t^2}||\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right) - \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)||^2\right] \\ &= \frac{1}{2}\mathbb{E}_{x\sim q(x_0),\epsilon_t}\left[\underbrace{\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}}_{\lambda_t}||\epsilon_t - \epsilon_\theta(x_t, t)||^2\right].\end{aligned} \tag{1.33}$$

From (1.33) we see that every timestep t the goal is to predict the noise level ϵ_t from a noisy sample x_t which is then weighted by λ_t. This weighting is of course not unique—in fact, the one above is derived specifically to maximize the marginal log-likelihood. In practice, we might want to tradeoff a slightly worse likelihood for higher visual fidelity as λ_t is often large for small t. A simplification that works well in practice is to simply set $\lambda_t = 1$ leading to what Ho et al. [2020] call $\mathcal{L}_{\text{simple}}$:

$$\mathcal{L}_{\text{simple}} = \frac{1}{2}\mathbb{E}_{x\sim q(x_0),\epsilon_t}\left[||\epsilon_t - \epsilon_\theta(x_t, t)||^2\right]. \tag{1.34}$$

It is also important to note that there are also other weighting schemes for λ_t that we do not cover here. These schemes will play a larger role when we try to realize the theory of diffusion models but in continuous time as they will correspond to specific instantiations of various Stochastic Differential Equations. This viewpoint is covered later in Chapter 3 along with the connection to Score-based generative models which we cover in Chapter 2 in detail.

Training and Sampling. We are finally in a position to state the complete training and sampling algorithms. One important observation is that for training the loss is temporarily decorrelated in the sense that we can uniformly sample a timestep t and compute $\mathcal{L}_{t-1}$ rather than in any specific ordering as each loss term is independent of any other timestep. The pseudocode for likelihood-based training is presented in Alg. 1 below.

Algorithm 1 Training

while not converged **do**
  $x_0 \sim q(x_0)$
  $t \sim \text{Uniform}(\{0, \dots, T\})$
  $\epsilon \sim \mathcal{N}(0, I)$
  $\nabla_\theta \mathcal{L}_t = \nabla_\theta \left[\lambda_t ||\epsilon_t - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2\right]$
end while

Once fully trained we can generate samples from the model by denoising a noisy sample backward in time. The pseudocode for sampling from a trained diffusion model is presented in Alg. 2.

Algorithm 2 Sampling

$x_T \sim \mathcal{N}(0, I)$
for $t = T, \dots, 1$ **do**
  $\epsilon \sim \mathcal{N}(0, I)$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\epsilon(x_t, t)}\right) + \sigma_t \epsilon$
end for

2

SCORE-BASED GENERATIVE MODELLING

As attractive as many likelihood-based generative models are they often need to impose certain restrictions on the function class to achieve their purpose. For example, normalizing flows must be invertible, autoregressive models impose a specific ordering over their inputs, and VAEs only provide a lower bound to the log-likelihood. Score-based generative models [Song and Ermon, 2019] bypass these constraints by modelling the score function $\nabla_x \log p(x)$ of the data distribution as opposed to $\log p(x)$ directly. As we will see such a design choice enables greater flexibility in the architectures used to represent the model density. Moreover, score-based models have tight connections to diffusion models but to keep this discussion accessible we relegate the magic of these details to later.

2.1 Modelling Data Scores with Score Models

To arrive at that the motivation for score-based models we can start by attempting to fit an empirical data distribution with a model. One simple way to do just this is to consider $E_\theta(x) \in \mathbb{R}$ be a real-valued function for an input point $x \in \mathbb{R}^n$. In certain literature, like statistical thermodynamics, E_θ is also called the Energy function and provides an estimate of scalar energy of a certain configuration. Now we can define a valid probability density as follows:

$$p_\theta(x) = \frac{e^{-E_\theta(x)}}{Z_\theta}, \tag{2.35}$$

where Z_θ is the normalization constant such that $\int p_\theta(x)dx = 1$. Note that Z_θ is a function of θ and any likelihood-based model that requires gradients of the form ∇_θ needs to elegantly handle this term. Instead of modelling the density we can choose to model the score function $\mathbf{s}_\theta(x) \approx \nabla_x \log p_(x)$ leading to the following simplifications:

$$\mathbf{s}_\theta(x) \approx \nabla_x \log p_{\text{data}}(x) = -\nabla_x E_\theta(x) - \underbrace{\nabla_x Z_\theta}_{=0}. \tag{2.36}$$

Conveniently, the troublesome Z_θ term drops out and as a result our parametrization E_θ can be more unrestricted—e.g. no invertibility needed. Henceforth, we will refer to our desired target distribution which we seek

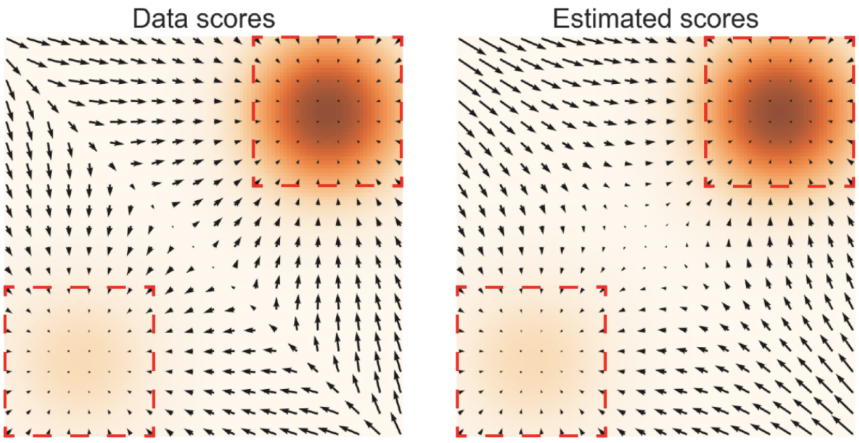

Figure 2.1: **Left:** True data scores $\nabla_x \log p(x)$. **Right:** Estimated model scores from $\mathbf{s}_\theta$. Figure taken from Song and Ermon [2019].

to approximate with as p_{data}. Naively, we can try to train score-based generative models by minimizing a suitable divergence between the real data score $\nabla_x \log p_{\text{data}}(x)$ and our approximation $\mathbf{s}_\theta(x)$:

$$\mathcal{L} = \mathbb{E}_{x \sim p_{\text{data}}(x)}[||\nabla_x \log p_{\text{data}}(x) - \mathbf{s}_\theta(x)||_2^2]. \tag{2.37}$$

However, in this current form (2.37) has a critical problem; namely, the real data score $\nabla_x \log p_{\text{data}}(x)$ is not known as we would need to have access to the data distribution $p_{\text{data}}(x)$ but instead we only have samples in the form of a dataset. This motivates the main technical distinctions in various score-matching methods that enable feasible computation of the naive objective in (2.37).

Now to get a conceptual understanding of what score-based models aim to achieve we can plot in Fig. 2.1 the vector field of both the data score and the estimated model score for a toy density of two 2D Gaussians. In this figure, data density is represented using an orange colormap with a darker shade indicating higher density regions. Intuitively, the estimated scores via $\mathbf{s}_\theta$ attempt to push us towards higher density regions which is needed when we want to sample from the model—i.e. we want to sample a point in high-density region because that's where the data actually lives.

Sampling using Langevin Dynamics. Let us assume we are given a fully trained score model and we wish to generate samples from it like any other generative model. How might we do so? One way to do so is to utilize an iterative procedure called Langevin dynamics and follow the vector field provided by the score to samples in high-density regions. Specifically, given

a fixed step size $\alpha > 0$ and an initial sample from a prior distribution $x_0 \sim p(x_0)$, e.g. a standard Gaussian, Langevin dynamics can produce samples from a desired target distribution $p_{\text{data}}(x)$ by recursively applying the following update rule:

$$x_t = x_{t-1} + \frac{\alpha}{2}\nabla_x \log p_{\text{data}}(x_{t-1}) + \sqrt{\alpha}\epsilon_t, \tag{2.38}$$

where $\epsilon_t \sim \mathcal{N}(0, I)$. The distribution over x_T equals the target $p_{\text{data}}(x)$ when we take $\alpha \to 0$ and $T \to \infty$. In practice, we run for a finite number of steps $T < \infty$ and do not take $\alpha \to 0$ and although this incurs an error this seems to not be significant. It is important to again reemphasize that sampling requires $\nabla_x \log p(x)$, which is the true data score but we do not have access to it and instead rely on an approximate score given to us by our model $\mathbf{s}_\theta(x) \approx \nabla_x \log p_{\text{data}}(x)$.

2.2 Score Matching for Score Models

As we work with deep learning models that are trained on high-dimensional data such as images we seek a scalable way to train score-based generative models. One mechanism to do this is to inject noise into clean data and train score models on the noisy data instead. The chief benefit of this scheme is that it avoids a rather large pitfall of sampling via Langevin Dynamics. The crux of the issue is that the prior distribution which we use to initialize Langevin is significantly more likely to sample a point in a low-density region with respect to p_{data} and as we typically train score based models using the empirical data distribution our trained model $\mathbf{s}_\theta$ may not have good score approximates in low-density regions and as result, it may be challenging to converge to a sample from a high-density region. By training on noisy data across noisy data, we flood the low-density regions and training now can make use of this information to provide more accurate score estimates which aid the downstream application of Langevin during sampling.

Explicit Score Matching. One way to mollify the objective in (2.37) is to consider a parzen window density estimator with a Gaussian kernel q_σ. This leads to the Explicit score matching objective (ESM):

$$\mathcal{L}_{ESM} = \mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}) - \mathbf{s}_\theta(\tilde{x})||_2^2]. \tag{2.39}$$

This objective has the benefit that the data score of $\nabla_{\tilde{x}} q_\sigma(x)$ is now computable if $\sigma > 0$ and q_σ is differentiable. However, choosing an appropriate σ and scaling this up such that low-density noise regions are not too washed out is still challenging.

Denoising Score Matching. While the idea of training on perturbed data is indeed interesting, one might wonder how one can choose an appropriate

noise scale. Should it be dependent on the data? The model complexity? Or a hyperparameter during training. Fortunately, we do not need to concern ourselves with these design decisions as we can simply train on all noise scales within an upper and lower bound simultaneously. This is the key idea of Denoising score matching (DSM). In particular, we can perturb clean data isotropic Gaussian noise with a finite number of levels L with increasing standard deviations $\sigma_1, \sigma_2, \ldots, \sigma_L$. Each noise level $i \in [L]$ defines a corresponding conditional distribution $q_{\sigma_i}(\tilde{x}|x)$ whose marginal over $p_{\text{data}}(x)$ is given by:

$$q_{\sigma_i}(\tilde{x}) = \int q_{\sigma_i}(\tilde{x}|x) p_{\text{data}}(x) dx. \tag{2.40}$$

Like previously seen in other generative models we can draw samples from $q_{\sigma_i}(\tilde{x}|x)$ using the reparametrization trick: $\tilde{x} = x + \sigma_i \epsilon$ where $x \sim p_{\text{data}}(x)$ and $\epsilon = \mathcal{N}(0, I)$. To avoid notational clutter we drop the noise level index i but it is implied in all the remaining equations. We are now ready to state the Denoising score matching objective:

$$\mathcal{L}_{DSM} = \mathbb{E}_{q_\sigma(\tilde{x},x)}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) - \mathbf{s}_\theta(\tilde{x})||_2^2]. \tag{2.41}$$

Intuitively, the gradient of the log-density approximated by $\mathbf{s}_\theta(x)$ should move us from a corrupted sample $\tilde{x}$ to its corresponding clean one x. To see this more explicitly note that $q(\tilde{x}|x)$ is a Gaussian and its gradient with respect to the noisy sample has a closed form expression $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) = -\frac{1}{\sigma^2}(\tilde{x} - x)^2$. At this point, a natural question that arises is how does the DSM objective (2.41) relate back to the Explicit score matching objective in (2.39)? As we will now see there's an explicit connection between the two different objectives. First, let us start from the ESM objective and rewrite it by expanding the terms.

$$\begin{aligned} \mathcal{L}_{ESM} &= \mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}) - \mathbf{s}_\theta(\tilde{x})||_2^2] \qquad (2.42) \\ &= \mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}[||\mathbf{s}_\theta(\tilde{x})||_2^2] - \underbrace{\mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}\left[\langle \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}), \mathbf{s}_\theta(\tilde{x})\rangle\right]}_{\Gamma_\theta} \\ &+ \underbrace{\mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x})||_2^2]}_{\text{does not depend on } \theta}. \qquad (2.43) \end{aligned}$$

Looking at (2.43) we notice the final term does not depend on θ and can be ignored as it is a constant for optimization purposes. Now analyzing the

inner product term Γ_θ further,

$$\begin{aligned}
\Gamma_\theta &= \mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}\left[\langle \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}), \mathbf{s}_\theta(\tilde{x})\rangle\right] && (2.44)\\
&= \int q_\sigma(\tilde{x}) \langle \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}), \mathbf{s}_\theta(\tilde{x})\rangle d\tilde{x} && (2.45)\\
&= \int q_\sigma(\tilde{x}) \left\langle \frac{\nabla_{\tilde{x}} q_\sigma(\tilde{x})}{q_\sigma(\tilde{x})}, \mathbf{s}_\theta(\tilde{x}) \right\rangle d\tilde{x} && (2.46)\\
&= \int \langle \nabla_{\tilde{x}} q_\sigma(\tilde{x}), \mathbf{s}_\theta(\tilde{x})\rangle d\tilde{x} && (2.47)\\
&= \int \left\langle \nabla_{\tilde{x}} \int q_\sigma(\tilde{x}|x) p_{\text{data}}(x) dx, \mathbf{s}_\theta(\tilde{x}) \right\rangle d\tilde{x} && (2.48)\\
&= \int \left\langle \int \nabla_{\tilde{x}} q_\sigma(\tilde{x}|x) p_{\text{data}}(x) dx, \mathbf{s}_\theta(\tilde{x}) \right\rangle d\tilde{x} && (2.49)\\
&= \int \left\langle \int q_\sigma(\tilde{x}|x) \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) p_{\text{data}}(x) dx, \mathbf{s}_\theta(\tilde{x}) \right\rangle d\tilde{x} && (2.50)\\
&= \int \int q_\sigma(\tilde{x}|x) p_{\text{data}}(x) \left\langle \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x), \mathbf{s}_\theta(\tilde{x}) \right\rangle dx d\tilde{x} && (2.51)\\
&= \int \int q_\sigma(\tilde{x}, x) \left\langle \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x), \mathbf{s}_\theta(\tilde{x}) \right\rangle dx d\tilde{x} && (2.52)\\
&= \mathbb{E}_{q_\sigma(\tilde{x},x)}[\langle \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x), \mathbf{s}_\theta(\tilde{x})]\rangle] && (2.53)\\
& && (2.54)
\end{aligned}$$

Plugging this back into (2.43) we get:

$$\begin{aligned}
\mathcal{L}_{ESM} &= \mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}[||\mathbf{s}_\theta(\tilde{x})||_2^2] - \mathbb{E}_{q_\sigma(\tilde{x},x)}[\langle \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x), \mathbf{s}_\theta(\tilde{x})]\rangle] \\
&\quad + \mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x})||_2^2] && (2.55)
\end{aligned}$$

To see how this connects back to the DSM objective we can work backward and rewrite the DSM objective by expanding out the terms and matching them.

$$\begin{aligned}
\mathcal{L}_{DSM} &= \mathbb{E}_{q_\sigma(\tilde{x},x)}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) - \mathbf{s}_\theta(\tilde{x})||_2^2] && (2.56)\\
&= \mathbb{E}_{\tilde{x}\sim q_\sigma(\tilde{x})}[||\mathbf{s}_\theta(\tilde{x})||_2^2] - \mathbb{E}_{q_\sigma(\tilde{x},x)}[\langle \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x), \mathbf{s}_\theta(\tilde{x})]\rangle] \\
&\quad + \underbrace{\mathbb{E}_{q_\sigma(\tilde{x},x)}[||\nabla_{\tilde{x}} q_\sigma(\tilde{x}|x)||_2^2]}_{\text{does not depend on } \theta}. && (2.57)
\end{aligned}$$

Inspecting (2.57) we notice that ignoring constants the ESM objective and the DSM objective are identical for a specific σ. However, due to the varying

noise scales the DSM objective is easier to optimize for generative models when we actually need to sample via Langevin dynamics.

Noise Conditional Score Networks. Under the DSM objective, we need to estimate the score for every noise level. To this end we can incorporate the noise level as additional input to the score network; thereby conditioning the noise. Thus, a noise conditional score network seeks to approximate $\mathbf{s}_\theta(\tilde{x}, i) \approx \nabla_{\tilde{x}} \log q_{\sigma_i}(\tilde{x})$ resulting in the following loss.

$$\mathcal{L}_{DSM,\sigma_i} = \mathbb{E}_{q_{\sigma_i}(\tilde{x},x)} \left[|| \frac{\tilde{x} - x}{\sigma_i^2} + \mathbf{s}_\theta(\tilde{x}, \sigma_i) ||_2^2 \right]. \tag{2.58}$$

We can combine this objective across all the noise levels $\sigma_i \in \{\sigma_i\}_{i=1}^L$ resulting in a single loss for optimization,

$$\mathcal{L} = \frac{1}{L} \sum_{i=1}^{L} \lambda(\sigma_i) \mathcal{L}_{DSM,\sigma_i}, \tag{2.59}$$

where $\lambda(\sigma_i) > 0$ is a function that depends on the noise level σ_i. An important insight in (2.59) is the minimizer of the optimization problem is the network $\mathbf{s}^*$ if and only if $\mathbf{s}^*(\tilde{x}, \sigma_i) = \nabla_{\tilde{x}} \log q_{\sigma_i}(\tilde{x})$ for all $\sigma_i \in \{\sigma_i\}_{i=1}^L$. We can reason this as (2.59) is a conical combination of DSM objectives across the various noise scales.

3

CONTINUOUS TIME DIFFUSION MODELS

In the previous two chapters, we considered two seemingly different approaches to generative modeling with diffusion and score-based generative models. However, the astute reader may have observed a rather intriguing connection between both approaches; namely, for tractable and efficient training we need a denoising step with a Gaussian distribution. In fact, the connection becomes more intimate if we take a continuum of noise scales—as opposed to a finite number—in denoising diffusion models. This chapter serves to investigate the tight and in some sense dual relationship between diffusion and score-based generative models. To fully appreciate the following material we would need to present the material through the lens of stochastic calculus and stochastic differential equations, which itself is a broad and rich subject. To keep this material as accessible as possible, we forego much of the mathematical nuance in favor of intuition which is perhaps customary whenever deep learning is concerned.

3.1 Score-based Generative Modeling with SDEs

Our starting point is to consider the setting when we take a diffusion model with L finite noise scales. Now if we $L \to \infty$ we perturb the real data distribution into a continuum of progressively growing noise. Such a setting

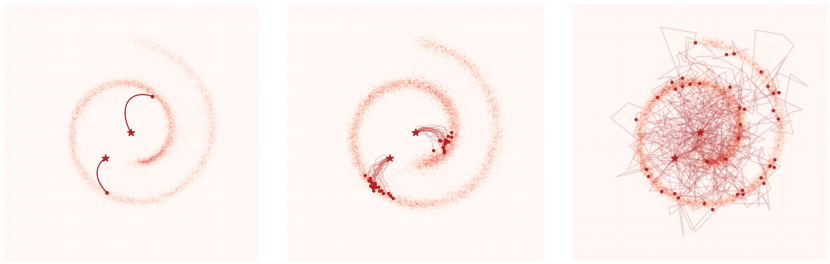

Figure 3.1: Three special cases of generative SDEs. The stars indicate the initial values, followed by some random sample paths. **Left**: trained with no diffusion $\sigma = 0$ (i.e. neural ODE). **Middle**: trained with some fixed diffusion $\sigma > 0$. **Right**: trained with a fixed inference process (diffusion SDE), μ and σ. Figure and caption taken from Huang et al. [2021].

is exactly a continuous stochastic process that can be represented as the solution of Stochastic Differential Equation (SDE). There are typically two equivalent ways to define an SDE in the literature—each with its own advantage—we adopt the convenient form offered via Itô calculus.

A diffusion model can be defined as the solution to the (Itô) Stochastic Differential Equation [Øksendal, 2003],

$$\mathrm{d}x = \mu\,\mathrm{d}t + \sigma\,\mathrm{d}B_t, \tag{3.60}$$

with the initial condition x_T following some unstructured prior $p(x_T)$ such as the standard normal distribution. In the Itô SDE literature, the function $\mu : \mathbb{R}^d \to \mathbb{R}^d$ is known as the drift coefficient while σ is a real-valued function called the diffusion coefficient. Finally, B_t is a standard Brownian motion that can be viewed as infinitesimal white noise. The drift and diffusion coefficients control the deterministic forces and the amount of noise injected at each time step that drives the evolution of the state. The solution to the SDE is a collection of random variables $\{x_t\}_{t\in T}$, indexed continuously through an upper time index T, which trace stochastic trajectories. As previously, $p(x_0)$ is the real data distribution that we have access to only via samples in the form of a dataset.

Let us call the SDE in (3.60) the diffusion SDE which perturbs the data until we reach a prescribed prior. It is important to note that the SDE is hand designed and should be considered part of the model in the sense that how we inject noise and its progression are not unique choices. For every SDE of the form in (3.60) it is possible to define a reverse denoising SDE that can be thought of as going backward in time and reversing the noise injection process. We term this the reverse generative SDE which in math is defined as follows:

$$\mathrm{d}x = (-\sigma\sigma^T \nabla_x \log p(x_t) + \mu)\,\mathrm{d}t + \sigma\,\mathrm{d}B_t, \tag{3.61}$$

where the SDE above needs to be solved backward in time ($t = T$ to $t = 0$). As an observant reader might notice, a key component in (3.61) is access to the score function of the noisy data $\nabla_x \log p(x_t)$ at timestep t. This is precisely the connection between diffusion models and score-based generative models. Namely, in the continuous time limit diffusion models correspond to solutions to a pair of SDEs that propagate forward and backward in time respectively and the reverse generative SDE explicitly utilizes the data score. Like previously, we can train a score network to approximate the data score $\mathbf{s}_\theta(x_t, t) \approx \nabla_x \log p(x_t)$. Our score network this time is conditioned directly on the time index t, which is analogous

to conditioning on the noise level in the discrete setting. Training time-conditioned score networks take the following form:

$$\mathcal{L} = \frac{1}{2}\mathbb{E}_{t\sim\text{Unif}(0,T),p(x_t)}\left[\lambda_t||\mathbf{s}_\theta(x_t,t) - \nabla_x \log p(x_t)||_2^2\right], \tag{3.62}$$

where λ_t is a time-dependent positive weighting function and $\text{Unif}(a,b)$ is the Uniform distribution supported on the interval$[a,b]$. We can also design a continuous version of the denoising score matching:

$$\mathcal{L}_{DSM} = \frac{1}{2}\mathbb{E}_{t\sim\text{Unif}(0,T),p(x_0)p(\tilde{x}_t|x_0)}\left[\lambda_t||\mathbf{s}_\theta(\tilde{x}_t,t) - \nabla_x \log p(\tilde{x}_t|x_0)||_2^2\right]. \tag{3.63}$$

In the specific case where the drift coefficient is linear in x, the transition density $p(\tilde{x}_t|x_0)$ is a tractable Gaussian distribution. Once training is finished to generate samples we can substitute our trained score model in the time-reversed generative SDE,

$$\mathrm{d}x = (-\sigma\sigma^T\mathbf{s}_\theta(x,t) + \mu)\,\mathrm{d}t + \sigma\,\mathrm{d}B_t. \tag{3.64}$$

Solving the generative SDE with our trained score network involves first sampling from the prior and tracing a stochastic trajectory until we arrive at a sample that resembles one from the data distribution. In practice, we often rely on numerical SDE solvers that enable us to simulate the SDE within a prescribed tolerance. The literature on numerical SDE solvers is vast and there are a variety of ones that are in use today such as Runge-Kutta and Euler-Maruyama to name a couple. We skip the discussion on the intricacies of these solvers as they are orthogonal to the thrust of the main goal which is to understand continuous-time diffusion models.

3.2 Variational Framework for Continuous Diffusion Models

In this section, we present the theory involved in constructing a variational framework for continuous diffusion models. A variational framework is ideal as it allows for a principled way to optimize the log-likelihood of the marginal density, which is a core design principle of many modern generative models. This section can be safely skipped as it assumes a little bit of mathematical maturity and many concepts require adequate prior knowledge to properly digest.

To train a continuous-time diffusion model via maximum likelihood, we require an expression for the log marginal density $\log p(x_0)$, which like in the discrete setting is computationally intractable.

To give a preview to our final result, the marginal likelihood of a continuous diffusion model can be represented using a stochastic instantaneous change-of-variable formula leading to a continuous-time evidence lower bound (CT-ELBO) [Huang et al., 2021, Song et al., 2021]. We will now derive this lower bound in detail. Our starting point is stating the instantaneous change in density $\partial_t p(x_t)$ which we can express using the *Fokker-Plank*, alternatively called the *Kolmogorov Forward* equation :

Note: In a slight abuse of notation we use upper indices, e.g. x^j to denote the j-th element of a vector.

$$\partial_t p(x_t) = -\sum_j \partial x_t^j [\mu^j(x_t) p(x_t)] + \sum_{i,j} \partial^2 x^i x^j [D_{i,j}(x_t) p(x_t)], \tag{3.65}$$

where $D = \frac{1}{2}\sigma\sigma^T$ is the diffusion matrix. We also adopted a convention—to avoid notational clutter—to drop the time dependence on the density which is implied through the sample at the timestep t, i.e. $p_t(x_t) := p(x_t)$. However, sometimes we will also be explicit especially if we want to evaluate a sample under our prior e.g. $p_T(x_T)$. Now returning to our main focus by expanding (3.65) and grouping coefficients of terms of the same order we get,

Note: $\nabla\cdot$ is the Euclidean divergence.

$$\begin{aligned}\partial_t p(x_t) = &\left[-\nabla\cdot\mu(x_t) + \sum_{i,j} \partial^2 x^i x^j D_{i,j}(x_t)\right] p(x_t) \\ &+ \sum_i \left[-\mu_i(x_t) + 2\sum_j \partial x^j D_{i,j}(x_t)\right] \partial x^i p(x_t) \\ &+ \sum_{i,j} D_{i,j}(x_t) \partial^2 x^i x^j p(x_t).\end{aligned} \tag{3.66}$$

We can further simplify this if we assume the diffusion coefficient σ is independent of the state x.

$$\partial_t p(x_t) = -(\nabla\cdot\mu(x_t))p(x_t) - \mu(x_t)^T \nabla p(x_t) + \langle D, H_p(x_t)\rangle_F, \tag{3.67}$$

where $\langle\cdot,\cdot\rangle_F$ is the Frobenius norm between matrices and H_p is the hessian matrix. Equation 3.67 is a second order linear partial differential equation whose solution admits a probabilistic representation using the *Feynmann-Kac* formula[1]. A direct application of this to (3.67) results in:

$$p(x_T) = \mathbb{E}\left[p_0(y_T)\exp\left(\int_0^T -\nabla\cdot\mu_{T-s}(y_s)ds\right)\middle| y_0 = x\right]. \tag{3.68}$$

[1] For more details on the Feynman-Kac representation see Theorem 3 in http://www.stat.uchicago.edu/~lalley/Courses/391/Lecture12.pdf

Note that y_s follows the diffusion SDE (the generative coefficients are evaluated in reversed time, i.e. $T - s$):

$$\mathrm{d}y = -\mu_{T-s}(y)\,\mathrm{d}s + \sigma_{T-s}\,\mathrm{d}\hat{B}_s \tag{3.69}$$

with $\hat{B}_s$ being another Brownian motion. The first immediate consequence of (3.69) is that we can interpret this is a mixture of continuous time normalizing flows. To see this, we notice that the diffusion coefficient is independent of the spatial variable x so it can be viewed as constant additive noise which is volume preserving. The sole contributor to the change in density is thus $x \to x + \mu\Delta t$ whose log determinant needs to be accounted for when applying the instantaneous change of variable formula.

Our main goal remains to calculate the log marginal density, but under our current progress, this would involve integrating out all possible Brownian motion paths which is computationally intractable. Instead, we can view the Brownian motion as an infinite dimensional latent variable and the inference task is to assign higher probability mass to more likely paths. In some sense, we can think of this setting as a VAE with an infinite dimensional latent variable. Such forward thinking also tells us that we can apply the common trick in VAEs by introducing a new probability measure $\mathbb{Q}$ in place of $\mathbb{P}$. This allows us to introduce the logarithm via an application of Jensen's inequality.

Note: Technically we need to define a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ for which $\hat{B}_s$ is a Brownian motion and $\mathbb{Q}$ to be a measure on the same probability space.

$$\log p(x_T) \geq \mathbb{E}_{\mathbb{Q}}\left[\log \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} + \log p_0(y_T) - \int_0^T \nabla \cdot \mu_{T-s}(y_s) ds \,\middle|\, y_0 = x\right]. \tag{3.70}$$

Observe that $\frac{\mathrm{d}P}{\mathrm{d}Q}$ is the *Radon-Nikodym* derivative and when both measures are absolutely continuous with respect to a third measure (e.g. Lebesgue) then the derivative can be written as a ratio of two densities. Despite this, it is unclear how one can reason about this quantity as we are dealing with infinite dimensional spaces. Fortunately, we can utilize the *Girsanov* theorem which allows us to compute a change of measure of Gaussian random variables under additive perturbation.

Theorem 3.2.1 (**Girsanov theorem**, Theorem 8.6.3 of Øksendal [2003]) *Let $\hat{B}_s$ be an Itô process solving*

$$\mathrm{d}\hat{B}_s = -a(\omega, s)\,\mathrm{d}s + \mathrm{d}B_s, \tag{3.71}$$

for $\omega \in \Omega$, $0 \leq s \leq T$ and $\hat{B}_0 = 0$, where $a(\omega, s)$ satisfies the Novikov's condition $\mathbb{E}\left[\exp\left(\frac{1}{2}\int_0^T a^2\,\mathrm{d}s\right)\right] < \infty$. Then $\hat{B}_s$ is a Brownian motion with

respect to $\mathbb{Q}$ *where*

$$\frac{\mathrm{d}\mathbb{Q}}{\mathrm{d}\mathbb{P}}(\omega) := \exp\left(\int_0^T a(\omega, s) \cdot \mathrm{d}B_s - \frac{1}{2}\int_0^T ||a(\omega, s)||_2^2 \,\mathrm{d}s\right). \tag{3.72}$$

Equation (3.71) provides a standarization formula of B'_s under $\mathbb{Q}$, which means we can "invert" it to reparameterize B_s. Applying this idea back to (3.70) we arrive at out lower bound:

$$\log p(x_T) \geq \mathbb{E}\left[\log p_0(y_T) - \int_0^T \left(\frac{1}{2}\left\|a(y_s, s)\right\|_2^2 + \nabla \cdot \mu_{T-s}(Y_s)\right) \mathrm{d}s \,\middle|\, y_0 = x\right] \tag{3.73}$$

where a is the variational degree of freedom, $\nabla\cdot$ denotes the (Euclidean) divergence operator. Equation (3.73) is known as the continuous-time evidence lower bound and the relationship between a and a score network is thus $a = \sigma^T \mathbf{s}_\theta$. Having, the CT-ELBO allows for a principled optimization objective but it should be noted that in the literature there are other objectives that do not necessarily maximize the likelihood directly. But for conciseness of presentation, we do not discuss these topics further.

Note: We skipped a few algebraic steps and substitutions. Details of this are provided in Table 2 Huang et al. [2021].

Probability Flow ODE and CNFs. The continuous time diffusion model induces two different probabilistic models for which we can define likelihoods for. The first is the likelihood induced by the generative reverse SDE which induces a likelihood on the marginal $p(x_0)$. The other model which we extract a likelihood from is known as the probability flow ODE which is associated and also available to every SDE. The probability flow ODE enjoys having the same marginal distribution $p_t(x_t)$ at every timestep t as its associated SDE. The probability flow ODE associated with the SDE in (3.60) is:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \mu(x_t) - \frac{1}{2}\sigma\sigma^T \nabla_x \log p_t(x_t). \tag{3.74}$$

To extract a probability flow ODE from our trained diffusion model we substitute our trained score network $\mathbf{s}_\theta$ for the data score $\nabla_x \log p_t(x_t)$. The remarkable thing about the probability flow ODE is that it is completely deterministic and this means it is infact an instantiation of a Continuous Normalizing Flow. This immediately implies that we can compute the exact marginal log-likelihood—and not just a lower bound—via the instantaneous change of variable formula which enables likelihood evaluation of continuous diffusion models.

Bibliography

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

C.-W. Huang, J. H. Lim, and A. Courville. A variational perspective on diffusion-based generative models and score matching. *arXiv preprint arXiv:2106.02808*, 2021.

K. Kreis, R. Guo, and A. Vahdat. Denoising diffusion-based generative modeling: Foundations and applications. In *Computer Vision and Pattern Recognition*, 2022.

B. Øksendal. Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer, 2003.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

Y. Song, C. Durkan, I. Murray, and S. Ermon. Maximum likelihood training of score-based diffusion models. *Advances in Neural Information Processing Systems*, 34, 2021.