Bridging Energy, Score, and Diffusion in Generative Modeling (a brief note)

Notes are based on the Deep Generative Models lectures, by Stefano Ermon

Sadegh Aliakbarian

March 2025

Abstract

This note presents a concise tutorial on energy-based generative modeling and its natural progression toward score matching and diffusion models. It begins by motivating energy-based models, emphasizing their flexibility alongside the challenges posed by unnormalized probabilities. The tutorial then introduces score matching and its denoising variant as effective alternatives for learning in high-dimensional spaces. Subsequently, it connects these concepts to diffusion models, demonstrating how iterative noise removal techniques offer a unified framework for modern generative modeling. Inspired by Stefano Ermon's lectures [1], this note serves purely as an educational tutorial based on existing materials, aiming to distill key ideas into an accessible introduction for those interested in state-of-the-art generative techniques without proposing any novel methods.

Contents

1 Energy-based Models

Probability distributions $p(x)$ are fundamental components of generative modeling. When constructing models to learn the underlying distribution of a dataset, it is essential that these distributions satisfy basic properties:

A valid probability distribution $p(x)$ must be non-negative and integrate (or sum, in the discrete case) to one:

$$p(x) \geq 0, \quad \text{and} \quad \int p(x)\,dx = 1. \tag{1}$$

In the context of neural network-based generative modeling, ensuring non-negativity is relatively straightforward—for instance, by squaring network outputs or applying exponential functions. However, enforcing that the distribution sums to one, i.e., normalization, is significantly more challenging in practice.

1.1 From Unnormalized Functions to Energy Functions

When modeling probability distributions with neural networks, ensuring non-negativity is relatively straightforward—one can, for instance, design a model such that $g_\theta(x) \geq 0$. However, enforcing that $g_\theta(x)$ sums (or integrates) to one over all possible outcomes is substantially more difficult. If $\sum_x g_\theta(x) \neq 1$, the function does not define a valid probability mass function or density.

One common strategy to address this challenge is to start with an unnormalized function. Instead of directly modeling a function $g_\theta(x) \geq 0$, we define an energy function $f_\theta(x)$, and express the distribution as:

$$p_\theta(x) = \frac{\exp(f_\theta(x))}{Z(\theta)}, \tag{2}$$

where

$$Z(\theta) = \int \exp(f_\theta(x))\,dx \tag{3}$$

is the *partition function*, which normalizes the distribution by accounting for the total "volume" of the unnormalized density. In some simple cases, such as the Gaussian distribution, the partition function can be computed analytically. For instance, for a Gaussian $g_{(\mu,\sigma)}(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, the partition function evaluates to $\sqrt{2\pi\sigma^2}$. However, for most models of interest, computing $Z(\theta)$ in closed form is intractable.

The exponential function is chosen because it not only guarantees non-negativity but also naturally captures large variations in the data. This formulation is closely connected to statistical physics, where similar expressions emerge from the principle of maximum entropy.

1.2 Energy-based Models

In energy-based models [2], the probability distributions are typically defined in the following form:

$$p_\theta(x) = \frac{1}{\int \exp(f_\theta(x))\, dx} \exp(f_\theta(x)), \tag{4}$$

where $\exp(f_\theta(x))$ ensures that the function is non-negative [1] and the normalization factor $\frac{1}{\int \exp(f_\theta(x))\, dx}$ guarantees that the resulting function is a valid probability density. In this framework, we are free to use any neural network architecture for $f_\theta(x)$, and Eq. 4 ensures that the output defines a valid probability distribution.

The Challenge of Computing the Partition Function While this formulation is highly flexible, computing the partition function $Z(\theta)$ exactly is often intractable, particularly in high-dimensional spaces. The integral may involve an exponentially large number of terms, making direct computation practically impossible. This intractability underscores the need for alternative training and sampling methods that circumvent the direct evaluation of $Z(\theta)$.

A useful observation is that not every operation involving such a function requires computing the partition function. For example, when comparing two samples x and x', the partition function cancels out:

$$\frac{p_\theta(x)}{p_\theta(x')} = \frac{\exp(f_\theta(x))}{\exp(f_\theta(x'))} = \exp\Big(f_\theta(x) - f_\theta(x')\Big). \tag{5}$$

This property is particularly advantageous in applications such as anomaly detection or *denoising* (spoiler alert), where relative probabilities are more important than absolute likelihood values.

1.3 Training Intuition

When training an energy-based model, the objective is to maximize

$$\max_\theta \frac{\exp f_\theta(x_{\text{train}})}{Z(\theta)}, \tag{6}$$

for a given training dataset x_{train}. Intuitively, the goal is to increase the numerator while simultaneously decreasing the denominator. As illustrated in Fig. 1, the desired outcome is to reduce the energy for correct points while increasing the energy for incorrect points, thereby minimizing $Z(\theta)$.

To achieve this, instead of computing $Z(\theta)$ exactly, we use a Monte Carlo estimate. Specifically, we sample a set of points from the model (to form the negative samples) and use the points from x_{train} as the positive samples, learning to distinguish between them.

[1] The exponential family defined here arises under general assumptions from statistical physics (e.g., maximum entropy and the second law of thermodynamics). In particular, the term $-f_\theta(x)$ is referred to as the energy. Intuitively, configurations x with low energy (i.e., high $f_\theta(x)$) are more probable.

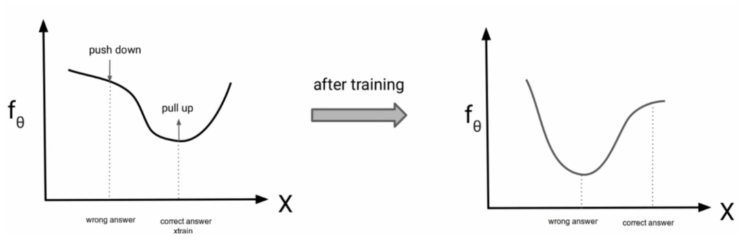


Figure 1: The goal of training an energy-based model.

Contrastive Divergence Algorithm. To train an energy-based model using the contrastive divergence algorithm [3, 4], we sample $x_{\text{sample}} \sim p_\theta$ and perform a gradient step on

$$\nabla_\theta \Big(f_\theta(x_{\text{train}}) - f_\theta(x_{\text{sample}}) \Big). \tag{7}$$

Examining Eq. 6, maximizing the log-likelihood corresponds to solving the following optimization problem:

$$\max_\theta \Big[f_\theta(x_{\text{train}}) - \log Z(\theta) \Big]. \tag{8}$$

Note that the partition function $Z(\theta)$ depends on θ. Therefore, the gradient we need to compute takes the form

$$\nabla_\theta f_\theta(x_{\text{train}}) - \nabla_\theta Z(\theta). \tag{9}$$

The gradient of the log-likelihood consists of two components:

1. The gradient of $f_\theta(x_{\text{train}})$, which is straightforward to compute.
2. A term involving the gradient of $Z(\theta)$, which can be expressed as an expectation over the model's distribution.

Computing the second term of the gradient is somewhat subtle. Given that the gradient of Eq. 8 is

$$\nabla_\theta f_\theta(x_{\text{train}}) - \frac{\nabla_\theta Z(\theta)}{Z(\theta)}, \tag{10}$$

we can rewrite this expression as

$$\begin{aligned} &\nabla_\theta f_\theta(x_{\text{train}}) - \frac{1}{Z(\theta)} \int \nabla_\theta \exp\big(f_\theta(x)\big)\, dx \\ &= \nabla_\theta f_\theta(x_{\text{train}}) - \int \frac{\exp\big(f_\theta(x)\big)}{Z(\theta)} \nabla_\theta f_\theta(x)\, dx. \end{aligned} \tag{11}$$

Notice that in the second term, $\frac{\exp(f_\theta(x))}{Z(\theta)}$ corresponds to the probability assigned by the model to a data point x. Thus, the gradient we seek to compute for optimizing the model parameters takes the form

$$\nabla_\theta f_\theta(x_{\text{train}}) - \mathbb{E}_{x_{\text{sample}} \sim p_\theta}\left[\nabla_\theta f_\theta(x_{\text{sample}})\right], \tag{12}$$

where the first term is the gradient of the energy evaluated at a data point, and the second term is the expected gradient with respect to the model's distribution. Using a Monte Carlo approximation, we can estimate the expectation in the second term by sampling $x_{\text{sample}} \sim p_\theta = \frac{\exp(f_\theta(x))}{Z(\theta)}$, yielding

$$\nabla_\theta f_\theta(x_{\text{train}}) - \nabla_\theta f_\theta(x_{\text{sample}}). \tag{13}$$

1.4 The Sampling Process

Unlike autoregressive or flow-based models, sampling from an energy-based model is non-trivial due to the difficulty in directly computing the probability of individual samples. However, as previously mentioned, we can still compare the relative likelihoods of two samples, x and x', which is useful for sampling. To leverage this property, we employ an iterative approach known as Markov Chain Monte Carlo (MCMC) [5].

The process begins by initializing a sample x^0 randomly at $t = 0$. At each step, the sample is perturbed to generate a new candidate,

$$x' = x^t + \text{noise}.$$

If the proposed sample has a higher likelihood, i.e., $f_\theta(x') > f_\theta(x^t)$, we accept it as the next sample:

$$x^{t+1} = x'.$$

Otherwise, we accept it with probability $\exp(f_\theta(x') - f_\theta(x^t))$, ensuring occasional downward moves to explore the distribution and avoid local optima. This procedure is repeated for T iterations. In theory, as $T \to \infty$, the distribution of samples converges to $p_\theta(x)$. However, in practice, this method suffers from slow convergence, particularly in high-dimensional spaces.

Improved Sampling with Langevin Dynamics A notable enhancement to standard MCMC is the unadjusted Langevin Monte Carlo (Langevin MCMC) [6], which incorporates gradient information to accelerate convergence. Instead of a simple random perturbation, the update step follows:

$$x^{t+1} = x^t + \epsilon \nabla_x \log p_\theta(x)\Big|_{x=x^t} + \sqrt{2\epsilon} z^t, \tag{14}$$

where $z^t \sim \mathcal{N}(0, \mathbf{I})$. This update rule combines a gradient ascent step that moves the sample toward regions of higher probability with an additional noise term to ensure exploration. Under appropriate conditions ($T \to \infty$ and $\epsilon \to 0$), this process guarantees convergence to the true distribution $p_\theta(x)$.

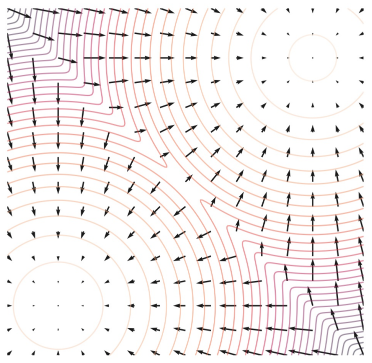


Figure 2: Score function provides an alternative view of the original function where we look at data in terms of gradients instead of likelihoods.

A key advantage of Langevin MCMC in the context of energy-based models is that the score function simplifies as $\nabla_x \log p_\theta(x) = \nabla_x f_\theta(x)$, eliminating the need to compute the partition function. While this makes gradient computation feasible, the method still suffers from slow mixing in high-dimensional spaces, posing a challenge for effective sampling. This issue is particularly problematic since sampling is essential for contrastive training of energy-based models, as shown in Eq. 12.

Having established the fundamental role of energy-based models in capturing complex distributions—along with the inherent challenge of computing the partition function—we now transition to *score matching*. This approach sidesteps the normalization issue by focusing on the gradients of log-density, setting the stage for a more tractable learning objective that avoids direct partition function estimation.

2 Score Matching

2.1 The Score Function

Recall that for an energy-based model, the probability density is defined as in Eq. 4, so that the log-density can be expressed as

$$\log p_\theta(x) = f_\theta(x) - \log Z(\theta). \tag{15}$$

The *score function* [7] is defined as the gradient of the log-density with respect to x:

$$s_\theta(x) := \nabla_x \log p_\theta(x). \tag{16}$$

Since $\log Z(\theta)$ is independent of x — as the partition function normalizes the distribution and does not depend on individual data points — we can write

$$s_\theta(x) = \nabla_x \log p_\theta(x) = \nabla_x f_\theta(x) - \underbrace{\nabla_x \log Z(\theta)}_{=0} = \nabla_x f_\theta(x). \tag{17}$$

Thus, the score function can be computed directly from the energy function without requiring evaluation of $Z(\theta)$.

A simple example is the Gaussian distribution. In this case,

$$p_\theta(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{18}$$

where the normalization constant is $\frac{1}{\sqrt{2\pi\sigma^2}}$. If we compute the score function for this distribution, we obtain $s_\theta(x) = -\frac{x-\mu}{\sigma^2}$.

Fig. 2 illustrates an intuitive interpretation of what a score function represents. Essentially, the score function provides an alternative perspective on the underlying distribution: rather than describing the likelihood at each point, it describes how the log-likelihood changes — that is, it gives the gradient with respect to x. As shown in Fig. 2, the score function forms a vector field, where each arrow indicates the direction of steepest ascent in log-likelihood. In the case of Gaussian distributions, these vectors naturally point toward the means of the Gaussians, indicating the most likely regions.

2.2 Training Objective

The training objective is defined by comparing two probability distributions p and q through their respective vector fields of gradients, known as score functions [8]. This comparison is formalized using the *Fisher divergence*:

$$D_F(p, q) := \frac{1}{2} \mathbb{E}_{x\sim p} \left[\|\nabla_x \log p(x) - \nabla_x \log q(x)\|_2^2 \right]. \tag{19}$$

The key idea behind this formulation is that if p and q are similar, then their respective score functions should also be close. Recall that the goal is to train an energy-based model, where p represents the data distribution and q denotes the model distribution. Importantly, this loss function in Eq. 19 depends only on the score functions and therefore avoids explicit computation of the (log) partition function.

2.3 Score Matching

The goal of score matching is to train the model without having to compute the intractable partition function. Instead of matching the probability densities directly, as in maximum likelihood estimation, score matching focuses on aligning their gradients—specifically, by minimizing the discrepancy between the score of the data distribution $\nabla_x \log p_{\text{data}}(x)$ and the model's score $s_\theta(x)$. A natural way to quantify this discrepancy is through the *Fisher divergence*:

$$\frac{1}{2}\,\mathbb{E}_{x\sim p_{\text{data}}}\left[\left\|\nabla_x \log p_{\text{data}}(x) - \underbrace{s_\theta(x)}_{\nabla_x f_\theta(x)}\right\|_2^2\right]. \tag{20}$$

Minimizing this divergence compels the model's score function to closely approximate the score function of the data distribution.

However, while this formulation is elegant, it is not directly applicable in practice because it involves $\nabla_x \log p_{\text{data}}(x)$, which is unknown. In practice, we typically only have access to *samples* drawn from $p_{\text{data}}(x)$. This raises a crucial question: how can we approximate $\nabla_x \log p_{\text{data}}(x)$ using only these samples?

2.4 Overcoming the Unknown Data Score

The true score $\nabla_x \log p_{\text{data}}(x)$ is unknown, as we only have access to samples from p_{data}. The key insight is to reformulate the objective in such a way that the unknown term can be eliminated through integration by parts [7].

To illustrate the concept of integration by parts, let's first consider the univariate case (1D).

$$\begin{aligned}&\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\left[(\nabla_x \log p_{\text{data}}(x) - \nabla_x \log p_\theta(x))^2\right]\\ =&\frac{1}{2}\int_{x\sim p_{\text{data}}}\left[(\nabla_x \log p_{\text{data}}(x) - \nabla_x \log p_\theta(x))^2\right]\end{aligned} \tag{21}$$

It is a square of the difference, so expanding it results in three terms,

$$\frac{1}{2}\int p_{\text{data}}(x)(\nabla_x \log p_{\text{data}}(x))^2 dx + \frac{1}{2}\int p_{\text{data}}(x)(\nabla_x \log p_\theta(x))^2 dx - \\ \frac{1}{2}\int p_{\text{data}}(x)\nabla_x \log p_{\text{data}}(x)\nabla_x \log p_\theta(x) dx \tag{22}$$

The first term is independent of θ and can thus be ignored during optimization. The second term depends solely on the model, making it relatively straightforward to compute. The third term is more interesting: it represents the cross product between the model and the data distribution, and it still involves the log of the data density, which is non-trivial to compute. To address this, we apply integration by parts.

Integration by Parts. Recall the integration by parts formula: $\int f'g\,dx = fg - \int g'f\,dx$. Applying this to the third term in Eq. 22, we get:

$$
\begin{aligned}
&- \int p_{\text{data}}(x) \nabla_x \log p_{\text{data}}(x) \nabla_x \log p_\theta(x) dx \\
&= - \int \cancel{p_{\text{data}}(x)} \frac{1}{\cancel{p_{\text{data}}(x)}} \nabla_x p_{\text{data}}(x) \nabla_x \log p_\theta(x) dx \\
&= \underbrace{-p_{\text{data}}(x) \nabla_x \log p_\theta(x)|_{x=-\infty}^{\infty}}_{=0} + \int p_{\text{data}}(x) \nabla_x^2 \log p_\theta(x) dx \qquad (23)
\end{aligned}
$$

In fact, we have now expressed it in terms of an expectation with respect to the data distribution over the second derivative of the model's score. This gives us a more convenient term $\int p_{\text{data}}(x)\nabla_x^2 \log p_\theta(x)\,dx$, which no longer depends on the *score* of the data density. The only remaining issue is the expectation with respect to the data density $\int p_{\text{data}}(x)$, but it no longer involves the score of the data.

Thus, returning to Eq. 21 and using the derived term from Eq. 23, we can express the univariate score matching objective as:

$$
\begin{aligned}
&\underbrace{\frac{1}{2} \int p_{\text{data}}(x)(\nabla_x \log p_{\text{data}}(x))^2 dx}_{\text{constant wrt } \theta} + \frac{1}{2} \int p\text{data}(x)(\nabla_x \log p_\theta(x))^2 dx \\
&\qquad + \int p_{\text{data}}(x) \nabla_x^2 \log p_\theta(x) dx \\
&= \mathbb{E}_{x\sim p_{\text{data}}}\Big[\frac{1}{2}(\nabla_x \log p_\theta(x))^2 + \nabla_x^2 \log p_\theta(x)\Big] + \text{const.} \qquad (24)
\end{aligned}
$$

which we can compute and evaluate. Extending this to the multivariate case, we have:

$$
\begin{aligned}
&\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\Big[||\nabla_x \log p_{\text{data}}(x) - \nabla_x \log p_\theta(x)||_2^2\Big] \\
&= \mathbb{E}_{x\sim p_{\text{data}}}\Big[\frac{1}{2}||\nabla_x \log p_\theta(x)||_2^2 + \text{tr}(\underbrace{\nabla_x^2 \log p_\theta(x)}_{\text{Hessian of } \log p_\theta(x)})\Big] + \text{const.} \qquad (25)
\end{aligned}
$$

where the second term represents the trace of the Hessian.

2.5 Refined Training Paradigm

With the objective function in Eq. 25, we can train the model using score matching through stochastic gradient descent. Given a mini-batch of samples $\{x_1, x_2, \ldots, x_n\} \sim p_{\text{data}}(x)$, we can estimate the score matching loss of Eq. 25 with the empirical mean:

$$
L = \frac{1}{n}\sum_{i=1}^{n}\Big[\frac{1}{2}||\nabla_x f_\theta(x_i)||_2^2 + \text{tr}(\nabla_x^2 f_\theta(x_i))\Big] \qquad (26)
$$

Luckily, there is no need to sample from the energy-based model in order to train the model.

While score matching offers a compelling alternative to likelihood-based training, it introduces computational challenges when evaluating higher-order derivatives. To address this, we shift our focus to *denoising score matching*, where the introduction of controlled noise simplifies the estimation of score functions. This not only alleviates computational burdens but also improves robustness in high-dimensional settings.

3 Denoising Score Matching

Denoising score matching [8] is a method designed to address the computational challenges posed by the trace of the Hessian in high-dimensional settings with large models. The key idea is that, rather than estimating the gradient of the clean data distribution, this approach aims to estimate the gradient of the data that has been *perturbed with noise.* It turns out that it is computationally easier to estimate the score of the distribution of the noisy data compared to estimating the score of the clean data. If the added noise is relatively small, this method provides a good approximation of the score of the clean data.

Denoising score matching. Given the data distribution $p_{\text{data}}(x)$, we use a noising mechanism to add noise to the samples, $q_\sigma(\tilde{x}|x)$, giving us the distribution of noisy data $q_\sigma(\tilde{x})$. Then the goal is to match the score of the noise-perturbed distribution via the Fisher-divergence:

$$\frac{1}{2}\mathbb{E}_{\tilde{x}\sim q_\sigma}\left[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}) - s_\theta(\tilde{x})||_2^2\right] = \frac{1}{2}\int q_\sigma(\tilde{x})||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}) - s_\theta(\tilde{x})||_2^2\, d\tilde{x} \tag{27}$$

Similar to before, we can expand this term by integration by parts, leading to

$$\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}(x),\tilde{x}\sim q_\sigma(\tilde{x}|x)}\left[\left|\left|s_\theta(\tilde{x}) - \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)\right|\right|_2^2\right] + \text{const.} \tag{28}$$

in which computing $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)$ is very easy, assuming it is a Gaussian, $q_\sigma(\tilde{x}|x) = \mathcal{N}(\tilde{x}|x, \sigma^2\mathbf{I})$. This means, $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) = -\frac{\tilde{x}-x}{\sigma^2}$. Putting this term in Eq. 28 (basically a denoising objective), it becomes very efficient to optimize for very high-dimensional data. On the negative side, this cannot estimate the score of the noise-free data.

3.1 Training with SGD

Now, given a mini-batch of data points $\{x_1, x_2, \ldots, x_n\} \sim p_{\text{data}}(x)$, we can have a perturbed mini-batch $\{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n\} \sim q_\sigma(\tilde{x})$ where $\tilde{x}_i \sim q_\sigma(\tilde{x}|x)$. Then in

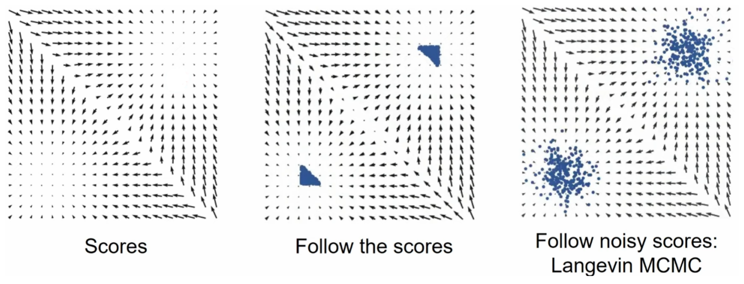


Figure 3: Left: Computed scores $s_\theta(x)$. Middle: Via MCMC and following the scores, the model generate samples that fall into local minima. Right: Via Langevin MCMC and following noisy scores, it is guaranteed to generate samples from the underlying data density.

the case of Gaussian perturbation, the objective becomes as simple as

$$\frac{1}{2n}\sum_{i=1}^{n}\left[\left\|s_\theta(\tilde{x}_i) - \frac{\tilde{x}_i - x_i}{\sigma^2}\right\|_2^2\right] \tag{29}$$

To optimize this term, what s_θ is trying to achieve is to match the amount of Gaussian noise that has been added to the data. Writing the loss in expectation, and by reparameterizing with $\tilde{x} = x + \sigma z$ where $z \sim \mathcal{N}(0, \mathbf{I})$, the loss becomes:

$$\frac{1}{2}\,\mathbb{E}_{x\sim p_{\text{data}},\, z\sim\mathcal{N}(0,\mathbf{I})}\left[\left\|s_\theta(x+\sigma z) + \frac{z}{\sigma}\right\|_2^2\right]. \tag{30}$$

Alternatively, by defining a noise-prediction network $\epsilon_\theta(\cdot)$ such that $\epsilon_\theta(x + \sigma z) = -\sigma\, s_\theta(x + \sigma z)$, the objective can be written as:

$$\frac{1}{2}\,\mathbb{E}_{x,z}\left[\|\epsilon_\theta(x+\sigma z) - z\|_2^2\right]. \tag{31}$$

How to sample? Via MCMC and given a computed score function $s_\theta(x)$ (which is basically similar to the vector field depicted in Fig. 2), we can follow the scores in an iterative process and update an initial pure noise sample into $\tilde{x}_{t+1} = \tilde{x}_t + \frac{\epsilon}{2}s_\theta(\tilde{x}_t)$. However, we know that this will lead to converging into local optima. Same as before, we can use Langevin MCMC and follow the noisy scores. In this case, we first need to sample a noise vector $z_t \sim \mathcal{N}(0, \mathbf{I})$ and the update would be $\tilde{x}_{t+1} = \tilde{x}_t + \frac{\epsilon}{2}s_\theta(\tilde{x}_t) + \sqrt{\epsilon}z_t$. If doing this iterative sampling for long enough, it is guaranteed to produce samples from the underlying density, as illustrated in Fig. 3.

3.2 Langevin Dynamics Sampling

The Langevin dynamics sampling follows these steps to sample from $p(x)$ using only the scores $\nabla_x \log p(x)$:

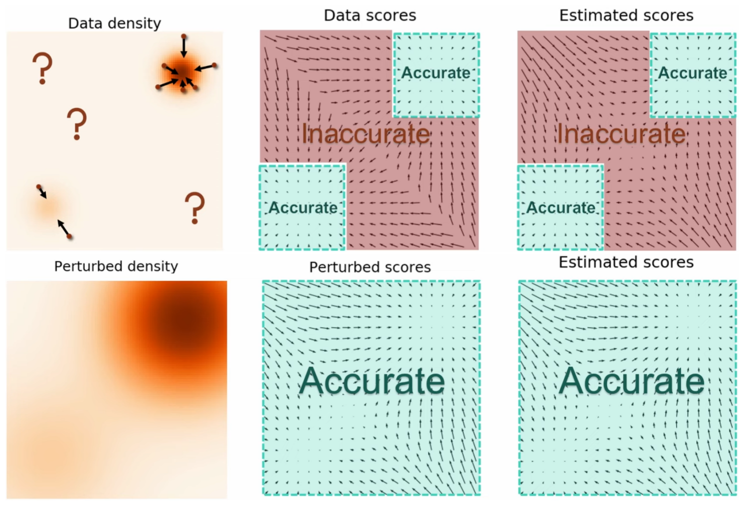


Figure 4: (Top) In the region that data density is high, the estimated scores are more accurate compared to regions where low data density. (Bottom) If we add noise to the data, we cover more space and thus estimated scores will be more accurate.

- Initialize $x^0 \sim \pi(x)$, some noise distribution
- Repeat for $t \leftarrow 1, 2, \dots T$:
  - $z^t \sim \mathcal{N}(0, \mathbf{I})$
  - $x^t = x^{t-1} + \frac{\epsilon}{2} \nabla_x \log p(x^{t-1}) + \sqrt{\epsilon} z^t$
- If $\epsilon \to 0$ and $T \to \infty$, it is guaranteed to have $x^T \sim p(x)$.

We can combine Langevin dynamics and score estimation, then $s_\theta(x) \approx \nabla_x \log p(x)$.

Unfortunately, on high-dimensional *real* data, such as images, the simple Langevin dynamics sampling fails to work. There are multiple reasons for this. One main challenge is related to dealing with low data density regions. As illustrated in Fig. 4 (top), for regions where we have no samples, the estimated scores are far from accurate, since the model does not see any samples to minimize its error on the scores. Another issue with Langevin dynamics sampling is that it basically cannot handle mixing dynamics between data modes. Assuming we have two modes where we have far more samples on one mode than the other, the model basically cannot correctly model that. If we sample many points, we see that while the locations of points may be correct, the density is not.

These challenges can be addressed, to some extent, by adding (Gaussian) noise to the data. In fact, if we add a large amount of noise to, e.g., the data distribution depicted in Fig. 4 (bottom), then samples will be spread across the entire space, and thus the added noise provides useful directional information for Langevin dynamics. This, however, comes at the cost of not approximating

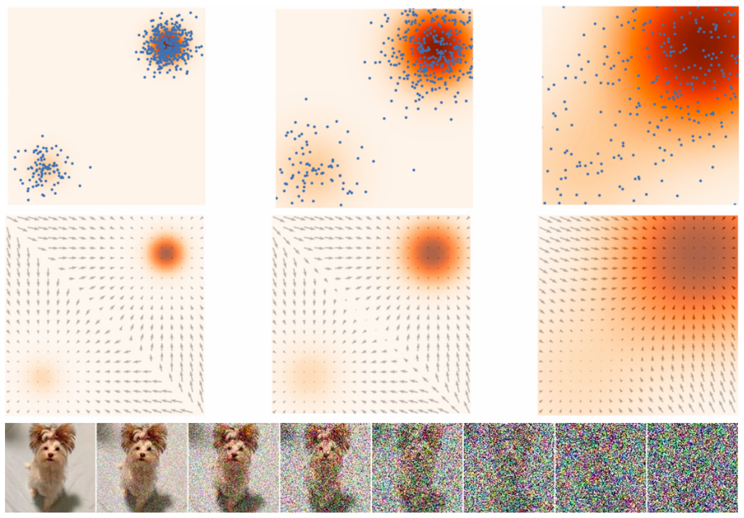


Figure 5: (Top) Multiple noise scales used to add noise to the data. (Middle) The estimated score function given different noise scales. (Bottom) Example of generating data with different noise scales. Process starts from right to left.

the true data density (but a highly noisy version of it). In other words, in this scenario, the model learns to estimate a more accurate score of a less accurate (noisy) data. Note that the difference between this and the denoising score matching discussed above is the *scale* of the noise. The assumption of the denoising score matching is that the noise that is being added to the data is small, and here the problem of low-density regions is being addressed by adding a large amount of noise.

The solution to this problem is *Annealed Langevin Dynamics* [8], in which the idea is that instead of choosing one noise scale, either small or large, we choose to have multiple noise scales, annealed from large to small.

As illustrated in Fig. 5, the annealed Langevin dynamics starts by sampling data points using different amounts of noise scales $\sigma_1, \sigma_2, \ldots, \sigma_L$, sequentially. Starting from the largest noise scale, we generate samples that initialize the process for the following noise scales. As shown in Fig. 5 (Bottom), the initial samples look like random noise simply because the noise scale is too large. However, such samples initialize another Langevin dynamics procedure with a smaller noise scale. We continue this process, making the noise scale smaller, and thus see more structure in the generated samples. This is achieved by training a noise-conditioned score-based model, wherein the noise scale is also provided as the input to the model.

3.3 Noise Conditional Score-based Model

This is very similar to denoising score matching since the goal is to estimate the score of the perturbed data distributions. The objective function to train such a model is thus the weighted combination of denoising score matching losses (for all noise scales):

$$\frac{1}{L}\sum_{i=1}^{L}\lambda(\sigma_i)\mathbb{E}_{q_{\sigma_i}(x)}\left[||\nabla_x \log q_{\sigma_i}(x) - s_\theta(x,\sigma_i)||_2^2\right] \tag{32}$$

When dealing with data at different noise levels, we use an objective function in the form of

$$\begin{aligned}&\frac{1}{L}\sum_{i=1}^{L}\lambda(\sigma_i)\mathbb{E}_{x\sim p_{\text{data}},z\sim\mathcal{N}(0,\mathbf{I})}\left[||\nabla_{\tilde{x}} \log q_{\sigma_i}(\tilde{x}|x) - s_\theta(\tilde{x},\sigma_i)||_2^2\right] + \text{const.}\\ &= \frac{1}{L}\sum_{i=1}^{L}\lambda(\sigma_i)\mathbb{E}_{x\sim p_{\text{data}},z\sim\mathcal{N}(0,\mathbf{I})}\left[\left|\left|s_\theta(x+\sigma_i z,\sigma_i) + \frac{z}{\sigma_i}\right|\right|_2^2\right] + \text{const.}\end{aligned} \tag{33}$$

in which $\lambda(\sigma_i)$ determines how much we should care about the quality of the denoiser at the noise level i. The goal of this term is to balance different score matching losses in the sum, thus one option is $\lambda(\sigma_i) = \sigma_i^2$. Adding this term to Eq. 33 and bringing it into the sum, we can re-write the objective as

$$\begin{aligned}&\frac{1}{L}\sum_{i=1}^{L}\mathbb{E}_{x\sim p_{\text{data}},z\sim\mathcal{N}(0,\mathbf{I})}\left[||\sigma_i s_\theta(x+\sigma_i z,\sigma_i) + z||_2^2\right]\\ &= \frac{1}{L}\sum_{i=1}^{L}\mathbb{E}_{x\sim p_{\text{data}},z\sim\mathcal{N}(0,\mathbf{I})}\left[|\epsilon_\theta(x+\sigma_i z,\sigma_i) + z||_2^2\right]\end{aligned} \tag{34}$$

3.4 Training Noise Conditional Score-based Model

Training with stochastic gradient descent, similar to the other cases we saw earlier, starts with sampling a mini-batch of data points $\{x_1, x_2, \ldots, x_n\} \sim p_{\text{data}}(x)$. We also sample a mini-batch of noise scale indices $\{i_1, i_2, \ldots, i_n\} \sim \mathcal{U}\{1, 2, \ldots, L\}$, where L is the number of noise levels. We then sample a mini-batch of Gaussian noise vectors $\{z_1, z_2, \ldots, z_n\} \sim \mathcal{N}(0, \mathbf{I})$. This would be enough to estimate the weighted mixture of score matching losses

$$\frac{1}{n}\sum_{k=1}^{n}\left[||\sigma_{i_k} s_\theta(x_k + \sigma_{i_k} z_k, \sigma_{i_k}) + z_k||_2^2\right] \tag{35}$$

to encourage the score network to solve the denoising problem for every data point at different noise levels. This is as efficient as training a single non-conditional score-based model.

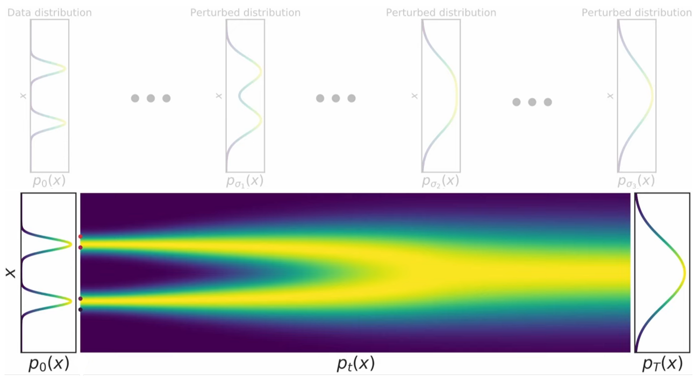


Figure 6: Infinite noise levels: Finite noise levels can be seen as slices of the infinite noise levels.

How many noise levels? The noise conditional score matching is very elegant. However, there are a few things to keep in mind. The number of noise levels during training and generation time should be the same. Also, it is best if the number of noise levels is relatively large. But how many? *Can we have infinite noise levels?*

Infinite noise levels. As illustrated in Fig. 6, we can actually have infinite noise levels, with discrete finite noise levels, as discussed above, being seen as slices over the infinite alternative. We can use stochastic processes to sample many data points along the x-axis (different slices indexed by t) and get $\{x_t\}_{t\in[0,T]}$, each with an associated probability density $\{p_t(x)\}_{t\in[0,T]}$, representing data density plus noise. We can now describe the evolution of these random variables over time with stochastic differential equations (SDEs).

Building on the denoising approach, we recognize that injecting noise naturally leads to a continuous description of the noising process. This insight motivates the formulation of stochastic differential equations (SDEs) to capture the evolution from data to pure noise. The SDE framework provides a rigorous mathematical tool to describe and reverse this process, thereby linking score estimation with stochastic sampling dynamics.

4 Stochastic and Ordinary Differential Equations

Stochastic differential equations (SDEs) [9] offer a continuous-time framework for modeling the evolution of a data point under both deterministic and stochastic influences. In this context, SDEs describe how a clean data sample is progressively transformed into pure noise [10]. More importantly, SDEs also provide a means of reversing this process to generate new samples.

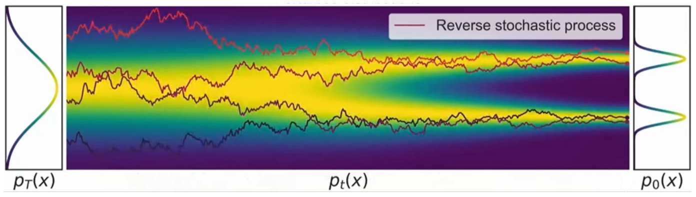


Figure 7: Reverse stochastic process.

4.1 Stochastic Differential Equation

Given a set of random variables $\{x_t\}_{t\in[0,T]}$, the changes in these variables can be described by stochastic differential equations (SDEs):

$$\mathrm{d}x_t = f(x_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}w_t \tag{36}$$

In this equation, the term $f(x_t, t)$ is called the deterministic drift, while $\mathrm{d}w_t$ represents the infinitesimal noise at each step, making the transition stochastic. This describes the process of transforming data into pure noise.

A common simplification of this process is given by

$$\mathrm{d}x_t = \sigma(t)\mathrm{d}w_t \tag{37}$$

which represents a purely stochastic evolution that gradually injects noise into the data.

4.2 Reverse Stochastic Processes

In this process, we reverse the direction of time in SDEs, moving from pure noise back to data, which is also known as the generation process.

The reverse SDE, for $t : T \to 0$, is given by

$$\mathrm{d}x_t = -\sigma(t)^2 \underbrace{\nabla_x \log p_t(x_t)}_{\text{the score function}} \mathrm{d}t + \sigma(t)\,\mathrm{d}\bar{w}_t \tag{38}$$

where $\nabla_x \log p_t(x_t)$ is the time-dependent score function and $\mathrm{d}\bar{w}_t$ represents a reverse-time Wiener process. By utilizing the score function, we can use the process in Eq. 38 to generate data from noise. To this end, we can train a neural network to learn a time-dependent score-based model:

$$s_\theta(x, t) \approx \nabla_x \log p_t(x) \tag{39}$$

This is exactly similar to the previous explanation, but instead of using a finite number of steps (e.g., 1000 noise levels), we now have an infinite number of steps. Consequently, the training objective would be:

$$\mathbb{E}_{t\in\mathcal{U}(0,T)}\left[\lambda(t)\mathbb{E}_{p_t(x)}\left[||\nabla_x \log p_t(x) - s_\theta(x,t)||_2^2\right]\right] \tag{40}$$

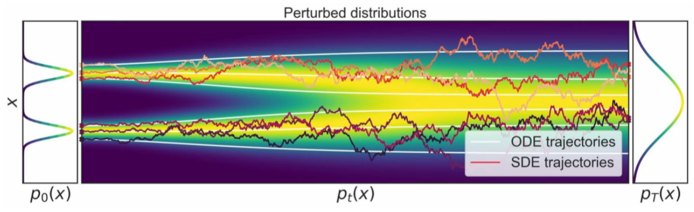


Figure 8: Casting the SDE to an ODE.

Once trained, we can use this model to represent the time-dependent score function and employ it in the reverse SDE process to generate data samples:

$$\mathrm{d}x_t = -\sigma(t)^2 s_\theta(x,t)\mathrm{d}t + \sigma(t)\mathrm{d}\bar{w}_t \tag{41}$$

This is a correct SDE, and the only step left is to solve it! Luckily, this is a well-established problem, and there are numerical methods to solve SDEs. One simple solution, called Euler-Maruyama, is to discretize time and solve:

$$x \leftarrow x - \sigma(t)^2 s_\theta(x,t)\Delta t + \sigma(t) z \tag{42}$$

where $z \sim \mathcal{N}(0, |\Delta t|\mathbf{I})$ and $t \leftarrow t + \Delta t$. This is very similar to Langevin dynamics, where we follow the scores $s_\theta(x,t)$ and add a little bit of noise, $\sigma(t)z$, at every step. The process is illustrated in Fig. 7. The advantage of this approach is that we don't have a fixed number of steps that must match during both training and generation. We can choose any number of steps during generation.

After establishing the role of SDEs in modeling the noising process, we note that under certain conditions, the stochastic evolution can be equivalently captured by a deterministic ordinary differential equation (ODE). This probability flow ODE not only provides insights into the underlying dynamics but also enables efficient and invertible mappings, similar to continuous normalizing flows.

4.3 Ordinary Differential Equation

With the mechanism of SDEs for transitioning from data to noise distribution, it turns out that we can describe the same stochastic process with exactly the same marginals using a purely *deterministic* mapping. This is achieved by ordinary differential equations (ODEs) [11]. An equivalent ODE for a simple SDE of $\mathrm{d}x_t = \sigma(t)\mathrm{d}w_t$ is

$$\frac{\mathrm{d}x_t}{\mathrm{d}t} = -\frac{1}{2}\sigma(t)^2 \nabla_x \log p_t(x_t) \tag{43}$$

which depends only on the score function. An illustration is provided in Fig. 8. The ODE provides a unique trajectory for each initial condition, yielding an invertible mapping similar to continuous normalizing flows. In other words, this machinery defines a continuous-time normalizing flow, where the invertible mapping is obtained by solving the ODE. A notable property is that the mapping from the data distribution to the noise distribution provides us with a latent representation of the data, which follows a simple distribution.

The deterministic formulation via ODEs paves the way for a novel interpretation of sampling methods. By drawing parallels between the iterative nature of Langevin dynamics and the encoder-decoder structure of variational autoencoders (VAEs), we uncover a unifying perspective. This connection emphasizes how sequential denoising can be seen as a hierarchical generative process, enriching our understanding of both frameworks.

5 Langevin Dynamics as VAEs

As discussed earlier and illustrated in Fig. 5, annealed Langevin dynamics [8] involves a sequential denoising process, where one starts from pure noise and progressively refines the sample using a series of noise scales $\{\sigma_1, \sigma_2, \ldots, \sigma_L\}$. At each stage, the data is perturbed with a specific noise level, and an iterative denoising process gradually refines the sample. Now, let us examine this process from the perspective of variational autoencoders (VAEs) [12].

5.1 The VAE Analogy

In annealed Langevin dynamics, the goal is to model the distribution of data perturbed with different levels of noise. Considering the entire Langevin chain, sampling involves going from pure noise to clean data. To train such a model, however, we use the inverse of this process by iteratively adding Gaussian noise, enabling us to train with the denoising score matching loss from Eq. 28. Looking at both processes, this closely resembles a variational autoencoder (VAE), where the inverse process (going from data to noise) is analogous to the VAE encoder, and the denoising process is analogous to the VAE decoder. However, instead of a one-step encoding and decoding process, this procedure is sequential.

Encoder (Forward Process). More formally, the forward process adds Gaussian noise to clean data x_0 over T steps, gradually converting it into nearly pure noise x_T. The noise-perturbed densities are obtained by adding Gaussian noise of the form

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \tag{44}$$

This process defines a joint distribution $q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$, which all are Gaussians. This process can also be expressed in closed form as

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_{t-1}, (1-\alpha_t)\mathbf{I}) \tag{45}$$

where $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$. It is important to note that $q(x_t|x_0)$ is also a Gaussian distribution. This is important because it is efficient to compute with the Gaussian assumption. This forward process is analogous to the encoder in a VAE, mapping data to a latent (noisy) representation.

Decoder (Reverse Process). The reverse process, which converts noise back to data, parallels the decoder in a VAE. Here, sample x_T from the noise distribution, $\pi(x_T)$ (i.e., the pure noise), and then iteratively sample from the *true* denoising distribution $q(x_{t-1}|x_t)$. This procedure would perfectly work if the denoising distribution $q(x_{t-1}|x_t)$ is known, which is not, unfortunately! The solution is to learn a *variational approximation* of the denoising distribution.

Since the true reverse conditional $q(x_{t-1} \mid x_t)$ is intractable, we learn a variational approximation:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2\mathbf{I}) \tag{46}$$

where $\mu_\theta(x_t, t)$ is predicted by a neural network that denoises x_t to produce x_{t-1}.

The problem is then to approximate the true denoising distribution $q(x_{t-1}|x_t)$ with $p_\theta(x_{t-1}|x_t)$. Once achieved, the joint distribution can be defined as $p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$. The key point here is that we choose the parameter $\bar{\alpha}_t$ of Eq. 45 such that at step T there is no signal-to-noise at the end (ending up in a known distribution) and we end up with pure noise, i.e., $\bar{\alpha} \to 0$ after a sufficiently large number of steps.

Recall, in order to train the VAE, we optimize the variational lower bound (aka ELBO) [12], that is

$$\mathbb{E}_{q_\phi(z|x)}\Big[\log p(z, x; \theta) - \log q_\phi(z|x)\Big] \tag{47}$$

5.2 Hierarchical Structure and the ELBO

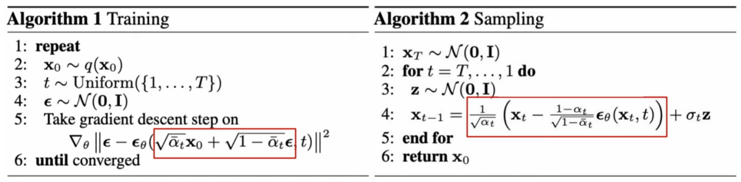
The analogy to the *iterative* denoising process is the *hierarchical* VAE [13], wherein instead of generating a data point directly from the latent code z, we generate it iteratively, $p(x, z_1, z_2) = p(z_2)p(z_1|z_2)p(x|z_1)$, in which we start by $z_2 \sim \mathcal{N}(0, \mathbf{I})$ (i.e., the prior), then sample z_1 from the decoder $p(z_1|z_2)$ and similarly sample x from the decoder $p(x|z_1)$. In this case, we can define the ELBO as

$$\mathbb{E}_{\underbrace{q(z_1, z_2|x)}_{\text{the encoder}}}\left[\log\left(\frac{p(x, z_1, z_2)}{q(z_1, z_2|x)}\right)\right] \tag{48}$$

With the VAE analogy providing an intuitive bridge between iterative sampling and latent variable models, we naturally extend these ideas to score-based diffusion models. In this section, we bring together the concepts of noise injection, score estimation, and variational inference to form a comprehensive generative framework that leverages the strengths of both denoising and diffusion processes.

6 Score-based Diffusion Models

Diffusion models [14] extend ideas from denoising score matching and hierarchical VAEs into a framework with many latent variables. Instead of using a few



```
Algorithm 1 Training
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
       $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boxed{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}, t) \right\|^2$
6: until converged
```

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

Figure 9: Training and sampling from a diffusion model (DDPM).

discrete noise levels, these models employ a long sequence—or a continuum—of latent variables that transform pure noise into data, unifying previous methods while leveraging the strengths of iterative denoising.

6.1 The Hierarchical VAE Analogy

What we discussed above on the hierarchical VAEs is basically what happens in diffusion models. Looking at the two latent variable scenario in Eq. 48, this can be extended to T latent variables, wherein we have the joint decoding distribution $p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$, i.e., going from pure noise to clean data. We also have the encoder $q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$, which is all fixed (just adding noise to the data). Similar to the hierarchical VAE loss of Eq. 48, we minimize an ELBO loss

$$\mathbb{E}_q(x_0)[-\log p_\theta(x_0)] \le \mathbb{E}_{q(x_0)q(x_{1:T}|x_0)}\left[-\log\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] \tag{49}$$

Since the encoder is fixed, unlike a VAE, computing the ELBO loss of Eq. 49 is simpler than that of the VAE. The decoder is $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}|\mu_\theta(x_t, t), \sigma_t^2\mathbf{I})$. Basically, the decoder is a neural network that computes the means of the Gaussians, which can be written as

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) \tag{50}$$

which contains a neural network $\epsilon_\theta(x_t, t)$ that aims at predicting the noise at step t which we use to subtract it from the actual x_t. Using this decoder parameterization in Eq. 49, we can simplify the ELBO loss as

$$L = \mathbb{E}_{x_0\sim q(x_0), t\sim\mathcal{U}\{1,T\}, \epsilon\sim\mathcal{N}(0,\mathbf{I})}\left[\lambda(t)\left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\,x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon,\, t\right)\right\|_2^2\right], \tag{51}$$

Eq. 51 is the denoising score matching loss. Although we started from the perspective of a VAE and thus wrote the loss function with ELBO, given the design choice of the decoder, we ended up with a score matching loss function. The difference between a diffusion model and score matching lies in the sampling. In score matching, we generate a sample using Langevin dynamics, whereas here we go through the decoding process. The pseudo-code for training and sampling of such a model is shown in Fig. 9.

6.2 Infinite Noise Levels

The actual heat diffusion process is a continuous-time process rather than discrete [2]. In our modeling of the diffusion process, we can use an SDE to explain the noising process (as in Eq. 36). To generate the samples, we can use the reverse SDE and go in the reverse direction, as in Eq. 38. Similarly to what was discussed before regarding the reverse SDE, there is a closed-form solution for Eq. 38 given a known score function.

Again, similar to what we discussed earlier in this note, we can have an ODE-equivalent of an SDE (see Eq. 43). Converting the SDE to ODE, we can consider the machinery as a continuous-time, infinite depth normalizing flow, wherein ODE solutions are unique (invertible mapping). To invert the mapping, we can solve the reverse ODE (the generation process).

Casting the ODE to a normalizing flow has several benefits. First, we can now evaluate the likelihood of a data point via the change-of-variable formula. To this end, starting from a data point, we solve the inverse ODE to compute the corresponding point in the latent space, evaluating its likelihood in the prior distribution, and then use the change-of-variable formula to get the data likelihood, as in

$$\log p_\theta(x_0) = \log \pi(x_T) - \frac{1}{2}\int_0^T \sigma(t)^2 \mathrm{tr}(\nabla_x s_\theta(x,t))\mathrm{d}t \tag{52}$$

Having detailed the construction of score-based diffusion models, let's focus on practical considerations.

6.3 The Diffusion Process

Forward (Noising) Process: The forward process corrupts the data over T steps:

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}), \tag{53}$$

where each transition is given by:

$$q(x_t \mid x_{t-1}) = \mathcal{N}\Big(x_t; \sqrt{1-\beta_t}\, x_{t-1},\, \beta_t \mathbf{I}\Big). \tag{54}$$

In closed form, this process can be expressed as:

$$q(x_t \mid x_0) = \mathcal{N}\Big(x_t; \sqrt{\bar{\alpha}_t}\, x_0,\, (1-\bar{\alpha}_t)\mathbf{I}\Big), \tag{55}$$

with $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$.

[2] Something like the one illustrated in Fig. 6.

Reverse (Denoising) Process: To generate new samples, we reverse the forward process. Starting from a simple prior $p(x_T)$ (e.g., a standard Gaussian), we define:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \tag{56}$$

with the reverse transitions modeled as:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\Big(x_{t-1}; \mu_\theta(x_t, t),\, \sigma_t^2 \mathbf{I}\Big). \tag{57}$$

Here, $\mu_\theta(x_t, t)$ is parameterized by a neural network that effectively denoises x_t to produce x_{t-1}.

Training Objective: The training objective resembles the denoising score matching loss. Typically, the network is reparameterized to predict the noise $\epsilon_\theta(x_t, t)$ rather than directly predicting the mean. This results in a loss function:

$$\mathcal{L} = \mathbb{E}_{x_0, t, \epsilon}\left[\lambda(t)\, \Big\|\epsilon - \epsilon_\theta\Big(\sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon,\, t\Big)\Big\|^2\right], \tag{58}$$

where $\lambda(t)$ is a weighting term balancing contributions across time steps.

6.4 Accelerated Sampling

When we convert the problem to solving an ODE, we can explore efficient sampling strategies [3].

For instance, DDIM [15] is often used as a sampling strategy in which, instead of going through, say, 1000 steps of denoising (as in DDPM [14]), we coarsely discretize the time axis (taking very big steps in time), e.g., taking only 20 steps. Although there is likely to be numerical errors as we take bigger steps, it is much faster, and the quality of samples is still good (speed-quality trade-off). Note that the score function (the marginals) is fixed once trained, and only the sampling strategy is different when we solve the SDE or the ODE with the same score function.

Another nice technique is distillation. The trick is to use DDIM as a teacher model to solve the ODE at some discrete time steps. Then, we can train a student model (another score-based model) to estimate, in a single step, what the DDIM would do in two steps. Basically, the student learns a new score function that in a single step represents what the teacher's score would do in two steps. We can do this iteratively and use such a student as a teacher for another student that learns to skip even further. This technique is called progressive distillation [16].

Another extreme example of distillation is consistency models [17], in which the authors proposed a solution to solve the ODE, going from the noise distribution $\pi(x_T)$ to the data distribution $p_\theta(x_0)$ in a single step.

[3]Typically, while with an SDE it may lead to higher quality samples, with an ODE, there are more efficient sampling solutions.

7 Conclusion

This tutorial has presented a comprehensive framework unifying several advanced concepts in generative modeling:

- **Energy-based Models:** Represent data distributions using unnormalized functions and address the challenge of computing the partition function.
- **Score Matching:** Bypass the intractable normalization constant by matching gradients (scores) instead of densities directly.
- **Denoising Score Matching:** Introduce controlled noise to simplify computation and improve efficiency.
- **SDEs and ODEs:** Provide a continuous-time perspective on the noising process and derive deterministic mappings for efficient sampling.
- **Langevin Dynamics as VAEs:** Interpret sequential denoising through the lens of variational autoencoders, connecting encoding and decoding processes.
- **Score-based Diffusion Models:** Unify the above ideas into a framework with a forward noising process and a learned reverse denoising process, augmented by accelerated sampling techniques.

These insights not only deepen our theoretical understanding of generative modeling but also pave the way for practical, scalable approaches to generating high-quality samples. Future work may explore improved sampling algorithms, more efficient training procedures, and novel applications of this unified framework.

References

[1] Stefano Ermon. Deep generative models, 2023.

[2] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, Fujie Huang, et al. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[3] Miguel A Carreira-Perpinan and Geoffrey Hinton. On contrastive divergence learning. In *International workshop on artificial intelligence and statistics*, pages 33–40. PMLR, 2005.

[4] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[5] Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.

[6] Giorgio Parisi. Correlation functions and computer simulations. *Nuclear Physics B*, 180(3):378–384, 1981.

[7] Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

[8] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[9] Peter E Kloeden, Eckhard Platen, Peter E Kloeden, and Eckhard Platen. *Stochastic differential equations*. Springer, 1992.

[10] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[11] Philip Hartman. *Ordinary differential equations*. SIAM, 2002.

[12] Diederik P Kingma, Max Welling, et al. Auto-encoding variational bayes, 2013.

[13] Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in neural information processing systems*, 33:19667–19679, 2020.

[14] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[15] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[16] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[17] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. 2023.