Diffusion, scores, flows, and all that

Diffusion models turn pure noise into stunning data, and now power some of the most advanced AI tools in the world. These notes break down the math so you can build, train, and truly understand them.

Prof. Emanuele Rodolà <rodola@di.uniroma1.it>

DLAI @ Sapienza, a.y. 2024/2025

https://github.com/erodola/DLAI-s2-2025

1 Background

Diffusion models went from obscure idea to powering some of the most jaw-dropping AI image and video generators in just a few years. Their secret? Start with pure noise and *run time backwards*. And suddenly, generating complex data becomes not just possible, but astonishingly effective.

The goal: Our ultimate goal is to recover an unknown data distribution $p_{\text{true}}(\mathbf{x})$ so that we can *sample* it to generate new, realistic samples.

The challenge: Learning this distribution directly using a parametric model $p_\theta(\mathbf{x})$ is often *intractable* due to the high dimensionality and complexity of real-world data.

Before diving in, let us review some of the key concepts that will accompany us throughout this lecture.

1.1 Energy and Partition Function

To understand the challenges in modeling $p_\theta(\mathbf{x})$, recall that any probability distribution can be written in the energy-based form:

$$p_\theta(\mathbf{x}) = \frac{1}{Z_\theta} \exp(-E_\theta(\mathbf{x}))\,. \tag{1}$$

This is known as a Boltzmann distribution, a concept borrowed from statistical physics.

An often useful expression, that we'll see often, is the log-likelihood:

$$\log p_\theta(\mathbf{x}) = -E_\theta(\mathbf{x}) - \log Z_\theta\,. \tag{2}$$

Exercise 1.1. Prove Eq. (2).

Interpretation. The function $E_\theta(\mathbf{x})$ is called the **energy function**. It assigns *low energy* to in-distribution data and *high energy* elsewhere. This energy function implicitly defines a probability distribution over the data space.

Think about it: What is the probability of a data sample (e.g., a photo) to be part of your distribution (e.g., photos of cats), when its energy is *high*? In other words, what is the value of $p_\theta(\mathbf{x})$ when $E_\theta(\mathbf{x})$ is very large?

The other quantity of interest is the term Z_θ, known as the **partition function**:

$$Z_\theta = \int \exp(-E_\theta(\mathbf{x}))\, d\mathbf{x}\,. \tag{3}$$

It ensures the distribution is properly normalized, i.e., $\int p_\theta(\mathbf{x})\, d\mathbf{x} = 1$. Once again, the name comes from physics, where it plays a central role in thermodynamics.

Key difficulty. Alas, computing Z_θ is typically *intractable*. If you look at Eq. (3) closely, you'll see it requires integration over the *entire data space*.

You might be tempted to estimate Z_θ by approximating the integral using only training data, but this often leads to incorrect results. Robust approximation methods like MCMC exist, but are often too expensive in practice.

Why we care about Z_θ. This number is indeed just a global rescaling factor, but we can't do without it. This constant directly affects both training and inference (sampling).

For example, if we want to maximize the log-likelihood, we need its gradient with respect to the parameters θ:

$$\nabla_\theta \log p_\theta(\mathbf{x}) = -\nabla_\theta E_\theta(\mathbf{x}) - \nabla_\theta \log Z_\theta. \tag{4}$$

At inference time, algorithms that explicitly sample from $p_\theta(\mathbf{x})$ also depend on Z_θ.

Fortunately, some approaches (like diffusion models) allow us to *sidestep the need to compute Z_θ directly*, while still sampling effectively from $p_\theta(\mathbf{x})$.

1.2 Bayes

Suppose you start with a known **prior** distribution $p(\mathbf{x})$, representing your belief about $\mathbf{x}$ before seeing any data. Now, after observing some data $\mathbf{y}$, you want to update your belief about $\mathbf{x}$. This is done via the **posterior** distribution:

$$p(\mathbf{x} \mid \mathbf{y}) = p(\mathbf{x}) \frac{p(\mathbf{y} \mid \mathbf{x})}{p(\mathbf{y})}\,. \tag{5}$$

This is known as the *Bayes update rule*. You can interpret it as a multiplicative correction to the prior $p(\mathbf{x})$, guided by the new information in $\mathbf{y}$.

Some more important terminology:

- $p(\mathbf{y} \mid \mathbf{x})$ is called the *likelihood*: it measures how compatible the observation $\mathbf{y}$ is with a specific value of $\mathbf{x}$.
- $p(\mathbf{y})$ is the *marginal likelihood* or *evidence*, defined as:

  $$p(\mathbf{y}) = \int p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \tag{6}$$

  It acts as a normalization factor ensuring the posterior is a valid probability distribution, i.e. $\int p(\mathbf{x} \mid \mathbf{y})\, d\mathbf{x} = 1$.

Note that the ratio $\frac{p(\mathbf{y}|\mathbf{x})}{p(\mathbf{y})}$ in Eq. (5) can be greater than or less than 1, depending on how surprising the observation $\mathbf{y}$ is under the assumption of $\mathbf{x}$.

2 Score matching

Back in 2005 (!), Hyvärinen [4] offered a way to sidestep the difficulty of modeling the full distribution $p_\theta(\mathbf{x})$. The idea hinges on a deep intuition: Instead of learning the distribution directly, we learn its **score function**:

$$\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) \,, \tag{7}$$

which is nothing but the gradient of the log-likelihood *with respect to the data* $\mathbf{x}$, not the parameters θ.

Exercise 2.1. What are the domain and codomain of $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$?

Exercise 2.2. Is the score function always well defined?

Why the score function? For a few reasons! First, the score captures the *local structure* of the data distribution, which should be much easier than modeling global density.

Furthermore, the partition function Z_θ disappears completely when taking the gradient:

$$\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}) \,. \tag{8}$$

This is *not* the gradient used in optimization (like SGD). It's a gradient with respect to *data*, that will be used at sampling time.

Finally, my favorite insight: The score function tells us *the direction in which the probability density increases* the fastest. So if we follow it, we're effectively climbing toward high-probability regions of the data distribution. This makes it incredibly useful for sampling! Even if we start from noise, we can iteratively adjust our sample using the score to move toward more realistic data.

Estimating the score. The authors defined a **score matching** objective, to find the parameters θ that make the *model score* match the *true score*:

$$\min_\theta \frac{1}{2} \left\| \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{true}}(\mathbf{x}) \right\|^2 \,. \tag{9}$$

Clearly, we don't really have access to $p_{\text{true}}(\mathbf{x})$, so we can't compute its score.

However, Hyvärinen showed that Eq. (9) is equivalent (under some regularity conditions) to the following:

$$\min_\theta \, \mathbb{E}_{\mathbf{x}\sim p_{\text{true}}} \left[\frac{1}{2} \left\| \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) \right\|^2 + \sum_{i=1}^{d} \frac{\partial^2}{\partial \mathbf{x}_i^2} \log p_\theta(\mathbf{x}) \right] , \tag{10}$$

where d is the dimension of the data space.

In practice, given a dataset $\{\mathbf{x}^s\}_{s=1}^n$ sampled from p_{true}, we minimize the empirical objective:

$$\sum_{s=1}^{n} \left[\frac{1}{2} \left\| \nabla_{\mathbf{x}^s} \log p_\theta(\mathbf{x}^s) \right\|^2 + \sum_{i=1}^{d} \frac{\partial^2}{\partial \mathbf{x}_i^2} \log p_\theta(\mathbf{x}^s) \right] . \tag{11}$$

However, no encouraging results were given. Sadly, neural networks weren't expressive or scalable enough to accurately learn the score on complex data like images. It took nearly two decades (and a few key innovations) before this idea was ready for the spotlight.

3 Denoising Score Matching

Denoising score matching (DSM), introduced by Vincent [8] in 2011, was a key insight that advanced Hyvärinen's original score matching framework and laid the foundation for modern diffusion models.

The core idea is that, instead of learning the score function directly on clean data samples $\mathbf{x}$, we first **corrupt** them with **Gaussian noise**:

$$\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}) \,. \tag{12}$$

The next step is to define a neural network $\mathbf{s}_\theta(\tilde{\mathbf{x}})$ to approximate the score of the *perturbed* data distribution:

$$\mathbf{s}_\theta(\tilde{\mathbf{x}}) \approx \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}) \,, \tag{13}$$

where:

$$q_\sigma(\tilde{\mathbf{x}}) = \int q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \, p_{\text{true}}(\mathbf{x}) \, d\mathbf{x} \tag{14}$$

is the distribution induced by the perturbation carried out in Eq. (12), and

$$q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{1}{2\sigma^2} \|\tilde{\mathbf{x}} - \mathbf{x}\|^2 \right) \,. \tag{15}$$

You can interpret the integral simply as a convolution of the true data density and the Gaussian noise. In other words, a continuous mixture of Gaussians centered at the data points.

To actually train $\mathbf{s}_\theta(\tilde{\mathbf{x}})$, directly minimize the **denoising score matching** objective:

$$\min_\theta \; \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p_{\text{true}}(\mathbf{x})} \left[\frac{1}{2} \left\| \mathbf{s}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \right\|^2 \right] \,. \tag{16}$$

This means the model learns to match the score of the corrupted conditional. Surprisingly (theorem alert!), minimizing this loss is *equivalent* to matching the true data distribution $p_{\text{true}}(\mathbf{x})$, under suitable conditions.

What's really happening: In practice, the model learns to *denoise* corrupted inputs. The score of the noise distribution is:

$$\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \frac{1}{\sigma^2} (\mathbf{x} - \tilde{\mathbf{x}}) \,, \tag{17}$$

which is a vector field that simply points from the noisy input back toward the clean data point. So the network learns to point in the direction of denoising.

Exercise 3.1. Prove Eq. (17).

Connection with autoencoders. Interestingly, Vincent noticed a key connection: training a **denoising autoencoder** (DAE) with a single layer and tied encoder-decoder weights is equivalent to denoising score matching:

- The DAE tries to reconstruct $\mathbf{x}$ from $\tilde{\mathbf{x}}$.
- The difference between the output and $\tilde{\mathbf{x}}$ approximates the score function.

But still no magic yet: Even though DSM gives us a tractable way to learn scores, it didn't lead to good sample generation in practice. Simply denoising noisy data isn't enough to generate from pure noise. We still lacked an effective way to turn score estimates into full generative algorithms.

4 Energy-Based Models (EBMs)

Fast-forward to 2019, when Du & Mordatch [1] showed that energy-based models (EBMs) could finally compete with GANs in high-quality sample generation. This was a major leap for score-based and energy-based methods.

EBMs start from a definition we know well:

$$p_\theta(\mathbf{x}) = \frac{1}{Z_\theta} \exp(-E_\theta(\mathbf{x}))\,, \tag{18}$$

where $E_\theta(\mathbf{x})$ is now a *neural network* that assigns low energy to likely data and high energy elsewhere. Once E_θ is learned, we can generate new samples from this model.

Sampling. Assume you were able to train an EBM. You now have *implicit* access (via $E_\theta(\mathbf{x})$) to a data distribution $p_\theta(\mathbf{x})$ you can sample from, to generate new data!

The sampling itself is done via an iterative procedure known as **Langevin dynamics**. The update rule is:

$$\tilde{\mathbf{x}}^k = \tilde{\mathbf{x}}^{k-1} - \frac{\lambda}{2}\nabla_{\mathbf{x}} E_\theta(\tilde{\mathbf{x}}^{k-1}) + \mathbf{z}^k, \quad \mathbf{z}^k \sim \mathcal{N}(0, \lambda\mathbf{I}). \tag{19}$$

Observe the following:

- As in score-based models, the gradient is with respect to data $\mathbf{x}$, not parameters.
- The noise $\mathbf{z}^k$ adds randomness, so the samples explore the space rather than deterministically converge.
- The number of steps K is called the **mixing time**.
- This is a form of gradient-based MCMC, and defines a distribution $q_\theta(\mathbf{x}) \approx p_\theta(\mathbf{x})$ as $K \to \infty$, $\lambda \to 0$.

Exercise 4.1. Prove that the update rule of Eq. (19) is equivalent to sampling using the score function $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$.

Note that unlike GANs or VAEs, which use an explicit mapping $\mathbf{x} = D(\mathbf{z})$ using some decoder D, EBMs define an *implicit* generator via optimization:

$$\mathbf{x} = \arg\min_{\mathbf{x}} E_\theta(\mathbf{x})\,. \tag{20}$$

Training. Say we are given some training data, defining an empirical distribution p_D. We want our model p_θ to match p_D. This is done by maximizing the log-likelihood:

$$\max_\theta \; \mathbb{E}_{\mathbf{x}\sim p_D}[\log p_\theta(\mathbf{x})]\,, \tag{21}$$

which measures how well the model explains the given data points.

However, the true gradient $\nabla_\theta \log p_\theta(\mathbf{x})$ is intractable due to the partition function. Instead, a **contrastive** approximation is used:

$$\nabla_\theta \log p_\theta(\mathbf{x}) \approx \mathbb{E}_{\mathbf{x}^+}[\nabla_\theta E_\theta(\mathbf{x}^+)] - \mathbb{E}_{\mathbf{x}^-}[\nabla_\theta E_\theta(\mathbf{x}^-)], \tag{22}$$

where:

- $\mathbf{x}^+ \sim p_D$ are real (positive) data samples,
- $\mathbf{x}^- \sim q_\theta$ are model-generated (negative) samples obtained via Langevin sampling *during training*.

This contrastive training ensures that positive samples have lower energy, negative samples have higher energy, while spurious low-energy modes are penalized. [1]

To regularize E_θ, the model uses an L_2 penalty on energy values:

$$\lambda_{\text{reg}}\left(\|E_\theta(\mathbf{x}^+)\|^2 + \|E_\theta(\mathbf{x}^-)\|^2\right). \tag{23}$$

Initial samples can be drawn from uniform noise or a replay buffer of past generations.

Remark: Unlike score matching or denoising score matching, there is **no explicit use of noise corruption** or reconstruction loss. The EBM is trained purely via contrastive likelihood and gradient-based sampling.

[1] It is worth noting that a similar idea was proposed with the concept of *contrastive divergence* by Nobel prize Hinton back in 2002 [2].

5 Noise Conditional Score Networks (NCSNs)

Notably, concurrently with the developments of EBMs in 2019, Song & Ermon [7] introduced the Noise Conditional Score Network (NCSN), a major step forward in score-based generative modeling. It draws inspiration from EBMs but instead **learns score functions directly** and **conditions them on noise levels**.

We are getting closer to the celebrated formulations of diffusion models that marked a new era of generative AI!

Key ideas:

1. Model the score function $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$ directly, rather than an energy function.
2. Condition the score estimation on a set of noise levels $\{\sigma_i\}$, using a single neural network $\mathbf{s}_\theta(\mathbf{x}, \sigma)$.

This is motivated by a fundamental challenge: score estimation is unreliable in low-density regions (e.g., near the boundaries of data). Adding noise spreads the distribution's support throughout the space, allowing the model to learn from everywhere.

Training. For each noise level σ_i, define the perturbed data distribution:

$$q_{\sigma_i}(\mathbf{x}) = \int q_{\sigma_i}(\mathbf{x} \mid \mathbf{x}_{\text{clean}})\, p_{\text{true}}(\mathbf{x}_{\text{clean}})\, d\mathbf{x}_{\text{clean}}\,, \tag{24}$$

where $q_{\sigma_i}(\mathbf{x} \mid \mathbf{x}_{\text{clean}}) = \mathcal{N}(\mathbf{x} \mid \mathbf{x}_{\text{clean}}, \sigma_i^2 \mathbf{I})$.

The idea is to train a single **conditional score network** $\mathbf{s}_\theta(\mathbf{x}, \sigma)$ to approximate the score of the perturbed distribution:

$$\mathbf{s}_\theta(\mathbf{x}, \sigma_i) \approx \nabla_{\mathbf{x}} \log q_{\sigma_i}(\mathbf{x}). \tag{25}$$

The training objective is the denoising score matching loss, averaged across noise levels – the score is trained at all noise levels simultaneously:

$$\mathcal{L}(\theta) = \sum_i \sigma_i^2\, \mathbb{E}_{\mathbf{x}_{\text{clean}}, \tilde{\mathbf{x}} \sim q_{\sigma_i}} \left[\frac{1}{2} \left\| \mathbf{s}_\theta(\tilde{\mathbf{x}}, \sigma_i) - \frac{1}{\sigma_i^2}(\mathbf{x}_{\text{clean}} - \tilde{\mathbf{x}}) \right\|^2 \right]. \tag{26}$$

Each score estimate $\mathbf{s}_\theta(\tilde{\mathbf{x}}, \sigma_i)$ corresponds to denoising data corrupted at a different noise scale.You can think of this as training multiple DAEs (one per noise level) but using a shared, noise-conditional network. Keep in mind that the output to the network has the same dimensions as the input data sample, but encodes a *gradient*.

The architecture of choice is a U-Net [5], well-suited for pixelwise predictions, with conditioning on σ_i implemented via *conditional instance normalization* (a feature normalization step where mean and variance are different for each σ_i).

Sampling. At inference time, the model performs a sequence of denoising steps (from high to low noise) using **annealed Langevin dynamics**. For each noise level σ_i, apply:

$$\tilde{\mathbf{x}}^k = \tilde{\mathbf{x}}^{k-1} + \frac{\lambda_i}{2}\, \mathbf{s}_\theta(\tilde{\mathbf{x}}^{k-1}, \sigma_i) + \sqrt{\lambda_i}\, \mathbf{z}^k, \quad \mathbf{z}^k \sim \mathcal{N}(0, \mathbf{I})\,, \tag{27}$$

where $\lambda_i \propto \sigma_i^2$.

Exercise 5.1. Prove that sampling $\mathbf{z} \sim \sqrt{\lambda} \cdot \mathcal{N}(0, \mathbf{I})$ is equivalent to $\mathbf{z} \sim \mathcal{N}(0, \lambda \mathbf{I})$.

During the sampling process, the noise level decreases exponentially: $\sigma_1 > \sigma_2 > \cdots > \sigma_L$. Here, each step gradually denoises the sample toward the data manifold. This process traces a smooth trajectory through data space, ending in a clean sample.

It is remarkable that the model does not define a normalized density (no partition function Z_θ in sight!) but provides a way to *sample* from the true data distribution.

This approach **laid the groundwork for modern diffusion models**, which replace discrete noise levels with continuous time and use stochastic differential equations for sampling.

Exercise 5.2. Reproduce Figure 2 and Figure 3 from the paper. See appendix for details.

6 Diffusion Probabilistic Models

Before reaching the modern version of diffusion models (Section 7), let's step back by a couple of years to 2015, when Sohl-Dickstein and coauthors [6] proposed Diffusion Probabilistic Models, a framework that defines a **forward noising process** and a **learned reverse denoising process** to model complex data distributions.

The core idea: transform data into noise in small, tractable steps, then learn to reverse the transformation.

Forward process (Diffusion). We start by defining a *Markov chain* $\mathbf{x}^{(0)} \to \mathbf{x}^{(1)} \to \cdots \to \mathbf{x}^{(T)}$, where:

$$q(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}) = \mathcal{N}\left(\mathbf{x}^{(t)}; \sqrt{1-\beta_t}\,\mathbf{x}^{(t-1)}, \beta_t \mathbf{I}\right). \tag{28}$$

Simply put, start from some real data $\mathbf{x}^{(0)} \sim q(\mathbf{x}^{(0)})$, and iteratively add Gaussian noise.

The parameters β_t control the **noise schedule**. Small noise is added at each step, accumulating over time until the signal is destroyed. The exact choice of β_t affects both the forward and reverse process and can be learned.

After many steps, the data is transformed into a standard Gaussian:

$$\pi(\mathbf{x}^{(T)}) = \mathcal{N}(\mathbf{x}^{(T)}; \mathbf{0}, \mathbf{I}). \tag{29}$$

If we were to write down the full forward trajectory distribution, this would be:

$$q(\mathbf{x}^{(0:T)}) = q(\mathbf{x}^{(0)}) \prod_{t=1}^{T} q(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}). \tag{30}$$

Reverse process (Generative model). Our initial state is now pure noise:

$$p(\mathbf{x}^{(T)}) = \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{31}$$

Each backward transition is modeled as:

$$p_\theta(\mathbf{x}^{(t-1)} \mid \mathbf{x}^{(t)}) = \mathcal{N}(\mathbf{x}^{(t-1)}; \boldsymbol{\mu}_\theta(\mathbf{x}^{(t)}, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}^{(t)}, t)). \tag{32}$$

Even though we're sampling from a Gaussian also in the reverse process, that Gaussian is centered not around $\mathbf{x}^{(t)}$, but around a *denoised* estimate $\boldsymbol{\mu}_\theta(\mathbf{x}^{(t)}, t)$. We're sampling from a distribution that (if trained well) is concentrated around the true denoised value.

The distribution parameters $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ are predicted by a neural network, often a CNN, taking $\mathbf{x}^{(t)}$ and t as input.

Exercise 6.1. How are the dimensions of $\mathbf{x}$, $\boldsymbol{\mu}_\theta$, and $\boldsymbol{\Sigma}_\theta$ related?

Training. We train the model p_θ by maximizing the data likelihood:

$$\max_\theta \; \mathbb{E}_{\mathbf{x}^{(0)} \sim q}\left[\log p_\theta(\mathbf{x}^{(0)})\right]. \tag{33}$$

In practice, maximize a variational lower bound that is easier to compute (similar to ELBO in VAEs):

$$\mathcal{L}_{\text{DPM}} = \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}^{(0:T)})}\left[D_{\text{KL}}\left(q(\mathbf{x}^{(t-1)} \mid \mathbf{x}^{(t)}, \mathbf{x}^{(0)}) \,\|\, p_\theta(\mathbf{x}^{(t-1)} \mid \mathbf{x}^{(t)})\right)\right], \tag{34}$$

where $q(\mathbf{x}^{(t-1)} \mid \mathbf{x}^{(t)}, \mathbf{x}^{(0)})$ has a closed-form expression because the forward process is Gaussian.

Intuitively, this loss simply compares forward process posteriors with reverse process posteriors. The objective encourages the model to match the reverse conditional at each step, effectively **learning to undo the noising** process one step at a time.

Sampling. Once the model is trained, we have access to a distribution p_θ (via $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$) we can sample from. Sampling starts from noise $\mathbf{x}^{(T)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, then iteratively applies:

$$\mathbf{x}^{(t-1)} \sim \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}^{(t)}, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}^{(t)}, t))$$

for $t = T, T-1, \ldots, 1$. At each step, the CNN predicts the mean and variance used to sample the next image. After T steps, a clean sample $\mathbf{x}^{(0)}$ is obtained.

Remark: While the sampling process resembles Langevin dynamics (due to step-wise updates with noise), it is not derived or interpreted as such in this work.

7 Denoising Diffusion Probabilistic Models (DDPM)

Exercise 7.1. Carefully read and study this Section; focus in particular on the **reparametrization** in the training loss.

We finally reach 2020 with the DDPM model by Ho et al. [3]. This has been the most widely adopted formulation of diffusion models. It simplified and stabilized earlier methods while clearly **connecting to denoising score matching and Langevin dynamics**.

Similarly to DPM, their goal is to learn to reverse a fixed forward diffusion process, defined by Gaussian noise with fixed variance schedule $\{\beta_t\}$.

Their key innovations include:

1. Simplified variational bound for training.
2. Direct connection to DSM via a new loss formulation.
3. Demonstrated high-quality image generation, rivaling GANs.

Forward process. Identical to DPM (Section 6), with pre-defined noise schedule $\beta_t \in [10^{-4}, 0.02]$, linearly increasing over $T = 1000$ steps.

Reverse process. Modeled as:

$$p_\theta(\mathbf{x}^{(t-1)} \mid \mathbf{x}^{(t)}) = \mathcal{N}(\mathbf{x}^{(t-1)}; \boldsymbol{\mu}_\theta(\mathbf{x}^{(t)}, t), \sigma_t^2 \mathbf{I}), \tag{35}$$

where the variance σ_t^2 is fixed and only the mean $\boldsymbol{\mu}_\theta$ is predicted by the network.

The mean is reparameterized as:

$$\boldsymbol{\mu}_\theta(\mathbf{x}^{(t)}, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}^{(t)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}^{(t)}, t) \right), \tag{36}$$

where $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

Here, $\boldsymbol{\epsilon}_\theta(\mathbf{x}^{(t)}, t)$ is the network (a U-Net) output, **trained to predict the noise** that was added to the clean data. This is conceptually similar to DSM, where the model learns to denoise a noisy input by estimating the gradient of the log-density.

Conditioning on t is implemented using sinusoidal positional embeddings.

Training. Use a reparametrized objective that resembles denoising score matching:

$$\mathbb{E}_{t, \mathbf{x}^{(0)}, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta \left(\sqrt{\bar{\alpha}_t} \mathbf{x}^{(0)} + \sqrt{1-\bar{\alpha}_t} \boldsymbol{\epsilon}, t \right) \right\|^2 \right], \tag{37}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the raw noise injected at all timesteps t, and $t \sim \text{Uniform}\{1, \ldots, T\}$.

We're simply training the network to recover the specific noise that was used in generating $\mathbf{x}^{(t)}$; that remains a valid target across all t, and it is easier to optimize than the full ELBO while encouraging the network to learn score-like behavior.

Sampling. Start from pure Gaussian noise $\mathbf{x}^{(T)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, then iterate:

$$\mathbf{x}^{(t-1)} = \boldsymbol{\mu}_\theta(\mathbf{x}^{(t)}, t) + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{38}$$

At each step, we are sampling $\mathbf{x}^{(t-1)}$ from a Gaussian distribution with mean $\boldsymbol{\mu}_\theta$ and standard deviation σ_t. This process resembles Langevin dynamics, because it alternates between "moving toward a denoised sample" and adding noise; however, the similarity is more conceptual than mathematical.

For the final step $p_\theta(\mathbf{x}^{(0)} \mid \mathbf{x}^{(1)})$, a quantizer is used to produce sharp outputs.

8 Conclusions

You've now seen the key historical steps that led to modern diffusion models, from score matching and EBMs to DDPMs. But there's a lot more out there! Here's what we haven't had time to cover:

Variants and Improvements of DDPM

- DDIM (Denoising Diffusion Implicit Models)
  Deterministic version of DDPM — removes noise during sampling for faster inference without retraining.
- Improved DDPM Sampling Schedules
  Fewer diffusion steps (e.g. 50 or 100 instead of 1000) using techniques like variance adjustment or classifier-free guidance.
- Classifier-Free Guidance
  Improves conditional generation by combining unconditioned and conditioned score estimates — used in models like Imagen and Stable Diffusion.
- Latent Diffusion Models (LDMs)
  Apply diffusion in a lower-dimensional latent space (e.g., VQGAN-encoded), making training and sampling faster and more scalable.
- Score Distillation Sampling (SDS)
  Used in text-to-image models like DreamFusion — distills the gradient of a diffusion model into other models (e.g., 3D).

Mathematical Extensions

- Continuous-Time Diffusion Models (SDE Formulation)
  Reformulate DDPMs as solutions to stochastic differential equations (SDEs). Enables more flexible samplers like reverse-time SDEs and ODE solvers.
- Probability Flow ODEs
  Deterministic version of reverse-time diffusion — no noise, enables exact likelihood computation and faster sampling.
- Rectified Flow
  A non-SDE-based flow model that trains directly to map noise to data in one step (or a few), bridging diffusion and optimal transport.

Flow Matching and Beyond

- Flow Matching (FM)
  Train a vector field (velocity) to map one distribution to another — a deterministic alternative to diffusion models.
- Gaussian Mixture Flow Matching (GMFM)
  Enhances FM by better modeling multimodal data distributions.
- Flow Map Matching / Two-Time Flow Models
  Learn the full mapping between noise and data at two different time points — generalizes many flow-based and diffusion-based techniques.

Architecture & Training Tricks

- Memory-efficient U-Nets and attention variants.
  Used in recent Stable Diffusion versions (e.g., SDXL, SD3) to scale to very high resolution.
- Conditional generation tricks
  Cross-attention for text conditioning, inpainting masking techniques, image-to-image conditioning, and temporal diffusion for video.

8.1 Suggested Reading Paths

1. Foundations (Start Here)
    - Hyvärinen, 2005 — Score Matching
    - Sohl-Dickstein et al., 2015 — Diffusion Probabilistic Models
    - Song & Ermon, 2019 — Noise Conditional Score Networks
    - Ho et al., 2020 — Denoising Diffusion Probabilistic Models (DDPM)
2. Refinements & Efficiency
    - Song et al., 2021 — DDIM: Denoising Diffusion Implicit Models
    - Nichol & Dhariwal, 2021 — Improved DDPM Sampling
    - Dhariwal & Nichol, 2021 — Classifier-Free Guidance
    - Rombach et al., 2022 — Latent Diffusion Models
3. Mathematical Extensions
    - Song et al., 2021 — Score-Based Generative Modeling via SDEs
    - Probability Flow ODEs — Deterministic sampling & likelihood
    - Kingma et al., 2021 — Variational Diffusion Models
4. Modern Flow-Based Approaches
    - Lipman et al., 2022 — Flow Matching
    - De Bortoli et al., 2023 — Rectified Flow
    - GMFlow, Discrete Flow Matching — Latest extensions
5. Architectures & Applications
    - Rombach et al., 2022 — Stable Diffusion / LDM
    - Google Brain — Imagen, DreamFusion
    - OpenAI — Sora, Consistency Models (2023)
    - Audio/Graph Diffusion — WaveGrad, GraphDenoise

References

[1] Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[2] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 08 2002.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[4] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[6] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[7] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2020.

[8] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.