Lecture Notes in Probabilistic Diffusion Models

Inga Strümke* and Helge Langseth
Norwegian University of Science and Technology

Abstract

Diffusion models are loosely modelled based on non-equilibrium thermodynamics, where *diffusion* refers to particles flowing from high-concentration regions towards low-concentration regions. In statistics, the meaning is quite similar, namely the process of transforming a complex distribution p_{complex} on $\mathbb{R}^d$ to a simple distribution p_{prior} on the same domain. This constitutes a Markov chain of diffusion steps of slowly adding random noise to data, followed by a reverse diffusion process in which the data is reconstructed from the noise. The diffusion model learns the data manifold to which the original and thus the reconstructed data samples belong, by training on a large number of data points. While the diffusion process pushes a data sample off the data manifold, the reverse process finds a trajectory back to the data manifold. Diffusion models have – unlike variational autoencoder and flow models – latent variables with the same dimensionality as the original data, and they are currently[1] outperforming other approaches – including Generative Adversarial Networks (GANs) – to modelling the distribution of, e.g., natural images.

1 Introduction

This document aims at being a coherent description of the mathematical foundation relevant for diffusion models. The body of literature in this area is growing very quickly, but the underlying mathematics of the diffusion process remains largely unchanged. In this document we give a self-contained presentation of this foundation, using coherent notation. We will whenever possible avoid discussing issues related to implementation, and rather focus on the fundamental properties of the diffusion models. This document was initially prepared as lecture notes for the course "IT3030: Deep Learning" at The Norwegian University of Science and Technology.

2 The diffusion process

2.1 Forward diffusion

Assume that we have a distribution that we can draw data samples $\mathbf{x}_0$ from. The subscript 0 indicates that this is an original data sample (for instance an image) without any noise added. In the *forward diffusion process*, noise is gradually added to the sample in T steps, generating increasingly noisy samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$, with $\mathbf{x}_T \sim p_{\text{prior}}$ (meaning that the $\mathbf{x}_T$ - samples follow a predefined distribution p_{prior} for sufficiently large T). The noising procedure must be scheduled to add noise ("destroy" the data sample) at the right pace. To this end, the variance β of the added noise increases following a schedule, i.e. the diffusion steps are parameterised by a *variance schedule* $\{\beta_t\}_{t=1}^T$. The data distribution is gradually converted into another distribution by repeatedly applying a Markov diffusion kernel K, i.e. the data sample $\mathbf{x}_t$ at step t is generated from $\mathbf{x}_{t-1}$ using

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = K\left(\mathbf{x}_t|\mathbf{x}_{t-1}; \beta_t\right) , \tag{1}$$

where β_t is the diffusion rate. This makes it clear that the process is Markovian, as each step depends only on the immediately preceding sample. The joint probability of the entire process from the original

---

*inga.strumke@ntnu.no
[1]At the time of writing, 2023.

Figure 1: The forward diffusion process from p_{complex} to p_{prior}.

data sample $\mathbf{x}_0$ to the final sample at step T can be written as

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = q(\mathbf{x}_1|\mathbf{x}_0) \prod_{t=2}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \,. \tag{2}$$

Using a Gaussian Markov diffusion kernel K, we have so-called Gaussian diffusion, as introduced in [Sohl-Dickstein et al., 2015]. Then, Equation (1) becomes

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \,, \tag{3}$$

where $\mathcal{N}$ denotes a Gaussian distribution and we have parameterized the distribution with mean $\boldsymbol{\mu} = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}$ and covariance matrix $\boldsymbol{\Sigma} = \beta_t\mathbf{I}$. The diffusion rate β_t typically starts close to 0 for small t, meaning that the variance of the conditional distribution in Equation (3) is small, while $\sqrt{1-\beta_t}$ is close to 1, and thus the conditional mean is close to $\mathbf{x}_{t-1}$, the sample at the previous step. It also follows that

$$q(\mathbf{x}_T) \approx p_{\text{prior}}(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}) \,,$$

with $q(\mathbf{x}_T) = p_{\text{prior}}(\mathbf{x}_T)$ in the limit as $T \to \infty$. A schematic visualisation of the diffusion process depicted as moving between distributions is shown in Figure 1.

The Gaussian distribution-function in vector notation is

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k \det\boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right) \right) \,.$$

We can draw samples from this distribution using that if $\mathbf{X} \sim \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, then we can reformulate $\mathbf{X}$ as

$$\mathbf{X} = \boldsymbol{\mu} + \mathbf{A}\mathbf{Z}, \quad \mathbf{Z} \sim \mathcal{N}(0, \mathbf{I}) \,, \tag{4}$$

where $\mathbf{A}$ is choses to ensure that $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^T$. Thus, defining $\alpha_t = 1 - \beta_t$, we can use Equation (3) to write

$$\mathbf{X}_t = \sqrt{\alpha_t}\,\mathbf{X}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{Z}_t \,,$$

if we let $\sqrt{1-\alpha_t}\,\mathbf{I}$ play the role of $\mathbf{A}$ in Equation (4) and choose $\mathbf{Z}_t$ to follow the standard Gaussian distribution. We can use this recursive relation to express $\mathbf{X}_t$ in terms of $\mathbf{X}_0$ directly as follows:

$$\mathbf{X}_t = \sqrt{\alpha_t}\,\mathbf{X}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{Z}_t \tag{5}$$

$$= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{X}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\,\mathbf{Z}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{Z}_t \tag{6}$$

$$= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{X}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\,\mathbf{Z}_{t-1:t} \tag{7}$$

$$= \ldots \tag{8}$$

$$= \sqrt{\bar{\alpha}_t}\,\mathbf{X}_0 + \sqrt{1-\bar{\alpha}_t}\,\mathbf{Z}_{0:t} \,. \tag{9}$$

2

Figure 2: Parameter values for $\beta = [10^{-4}, 0.02]$ over 1000 time steps t using a linear schedule. The information in the two figures are the same, but the right-hand side uses log-scale on the y-axis to show the speed of which $\bar{\alpha}_t$ goes towards zero.

In Equation (6) we have used Equation (5) to represent $\mathbf{X}_{t-1}$ via $\mathbf{X}_{t-2}$ and $\mathbf{Z}_{t-1}$. To get to Equation (7) we used two facts about random variables: First, the sum of two Gaussian variables with parameters $(\boldsymbol{\mu}_1 = \mathbf{0}, \boldsymbol{\Sigma}_1 = \sigma_1^2 \mathbf{I})$ and $(\boldsymbol{\mu}_2 = \mathbf{0}, \boldsymbol{\Sigma}_2 = \sigma_2^2 \mathbf{I})$ is a new Gaussian variable with mean $\mathbf{0}$ and covariance $(\sigma_1^2 + \sigma_1^2) \cdot \mathbf{I}$. Second, for a scalar γ and a random variable $\mathbf{W}$, we have $\text{Var}(\gamma \mathbf{W}) = \gamma^2 \text{Var}(\mathbf{W})$. Now, the transition from Equation (6) to Equation (7) holds because

$$\sqrt{\alpha_t(1-\alpha_{t-1})}\mathbf{Z}_{t-1} + \sqrt{1-\alpha_t}\mathbf{Z}_t = \mathcal{N}(\mathbf{0}, \alpha_t(1-\alpha_{t-1})\mathbf{I}) + \mathcal{N}\mathbf{0}, (1-\alpha_t)\mathbf{I})$$
$$= \mathcal{N}(\mathbf{0}, (1-\alpha_t\alpha_{t-1})\mathbf{I})$$
$$= \sqrt{1-\alpha_t\alpha_{t-1}}\mathcal{N}(\mathbf{0}, \mathbf{I}).$$

We have given the $\mathbf{Z}$-variables explicit indices in the development above to keep track of the time step(s), but remember that all of them follow the standard Gaussian distribution with mean $\mathbf{0}$ and covariance-matrix $\mathbf{I}$. The dots in (8) indicate repeating this procedure for all steps until $\mathbf{X}_0$, and in Equation (9) we have introduced the compact notation $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. We have thus arrived at an expression for generating a single noisy sample $\mathbf{x}_t$ given an initial sample $\mathbf{x}_0$:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}\right). \tag{10}$$

Note that the covariance matrix of the added noise is diagonal throughout. The samples $\mathbf{x}_t$ gradually become more noisy, and as $T \to \infty$, $\mathbf{x}_T$ is drawn from an isotropic Gaussian distribution, $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(0, \mathbf{I}) = p_{\text{prior}}$. [Ho et al., 2020] use $T = 1000$. Regarding the variance schedule, values of β_t are typically in the range $[10^{-4}, 0.02]$. The schedule used in [Sohl-Dickstein et al., 2015] defines a linear relationship between t and β_t, while the one used in [Nichol and Dhariwal, 2021] is a cosine. The parameter values for this range, using a linear scale for simplicity, are shown in Figure 2.

To summarise, the forward diffusion process consists of a step-wise transformation of a sampled $\mathbf{x}_0$ to a random Gaussian noise-variable $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$. We start from $\mathbf{x}_0$, and at each time-step $t = 1, \ldots, T$, make adjustments so that $\mathbf{x}_t$ has a mean that is gradually moved towards zero and a variance that gradually increases towards unity as t increases. This is most easily seen by studying the numerical values of the parameters in Equation (10) using Figure 2.

Finally, we can also use Equation (10) to generate a sample $\mathbf{x}_t$ directly from an original sample $\mathbf{x}_0$ via

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{11}$$

Figure 3 shows example trajectories from a univariate diffusion process. The top part of the figure shows ten trajectories all starting from the same $\mathbf{x}_0$. The data is generated by iteratively sample using $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ following Equation (3). Eventually, at $t = T$, the samples approximately follow the targeted Gaussian distribution shown on the right-hand side. The bottom part of Figure 3 gives the estimated density $q(\mathbf{x}_t|\mathbf{x}_0)$ for each t based on a high number of sampled trajectories, see also Equation (10), again using the same starting-point $\mathbf{x}_0$ as in the top part of the figure.

3

Figure 3: The forward process moves in the direction of increasing time (left to right). The marginal distribution on the right-hand side of each plot gives the distribution $q(\mathbf{x}_T|\mathbf{x}_0)$. Top: 10 trajectories sampled from the same starting-point $\mathbf{x}_0$. Bottom: For each t the figure shows the estimated conditional distribution $q(\mathbf{x}_t|\mathbf{x}_0)$ based on 25.000 trajectories sampled from the same starting-point $\mathbf{x}_0$.

The last point to make about the forward diffusion process is that we do not need to train a machine learning model to do any of this. We are simply following Equation (3) mechanically to transform a sample $\mathbf{x}_0$ from the data distribution p_{data} until we obtain a transformed sample $\mathbf{x}_T$ that (approximately) comes from the much simpler p_{prior}. One could ask what we have gained by this. That will hopefully become clear in the next subsection when we reverse the process: Starting from a sample $\mathbf{x}_T \sim p_{\text{prior}}$, we will gradually change that until we obtain an $\mathbf{x}_0$ that (approximately) is from p_{complex}.

2.2 Generative modelling = undoing diffusion

Having completed the forward diffusion process, we would now like to be able to undo that again, that is, gradually remove noise from a sample $\mathbf{x}_T$, ending up with a data sample $\mathbf{x}_0$ from the original distribution. An important result in this context is by [Feller, 1949], showing that in the limit of infinitesimal step size, the true reverse process will have the same distributional form as the forward process. In our case, we should thus expect the reversed process to be a series of random variables

4

following the Gaussian distribution (since the forward-process only involved Gaussians). In essence it should thus be *possible* to model the reverse process – we only need to calculate the parameters of the Gaussians. Nevertheless, while the forward diffusion process is fixed, finding the reverse process requires quite some effort.

Ideally, we would just calculate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and sample step-wise from the distribution guiding us from a sample $\mathbf{x}_T \sim p_{\text{prior}}$ to a sample $\mathbf{x}_0 \sim p_{\text{complex}}$. However, using Bayes' rule to calculate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ from $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ means that we will need to not only *represent* the distribution $q(\mathbf{x}_0)$, but also be able to *integrate* over it. This follows because the only way to use Bayes rule when we do not have access to the marginals $q(\mathbf{x}_{t-1})$ and $q(\mathbf{x}_t)$ would be as follows:

$$
\begin{aligned}
q(\mathbf{x}_{t-1}|\mathbf{x}_t) &= \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}) \cdot q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t)} \\
&= \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}) \cdot \int_{\mathbf{x}_0} q(\mathbf{x}_{t-1}|\mathbf{x}_0)q(\mathbf{x}_0)d\mathbf{x}_0}{\int_{\mathbf{x}_0} q(\mathbf{x}_t|\mathbf{x}_0)q(\mathbf{x}_0)d\mathbf{x}_0} .
\end{aligned}
$$

Remember that we previously referred to $q(\mathbf{x}_0)$ as p_{complex} because we have no reason to believe that it is a Gaussian (or that it comes from any other mathematically convenient distributional family, for that matter): It is only for $t > 0$ and conditionally on the starting-point $\mathbf{x}_0$ that all variables are Gaussians. In conclusion, we will not be able to calculate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ directly, but need to do something else. The idea is instead to find a suitable approximation to $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and let that approximation define the reverse process. Fortunately, we can define such a reverse diffusion process even if we limit our use of p_{complex}. In the following we will only be accessing *samples* from the distribution. We will generate those samples by randomly selecting examples from the training-data, which represents the data-distribution p_{data}, and therefore approximates p_{complex}. This is the core idea behind diffusion models, and how all of this fit together will be described next.

As an alternative to reversing the diffusion process with Bayes rule, we will define a new distribution that is meant to be representing the reverse process directly. That is, we will create a step-wise noise reduction process $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$ that at least approximates $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Leveraging the observation that the reverse process must have the same functional form as the forward process (given that each β_t is sufficiently small in the forward process), each reverse step can be parameterised as a Gaussian, and the parameters can be learned by fitting a neural network, as observed by [Sohl-Dickstein et al., 2015]. This means that we only have to estimate the mean and variance of the distribution $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$ in order to draw samples from it. Letting a neural network parameterised by $\boldsymbol{\theta}$ represent this distribution, i.e., produce the distributional parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, we can thus write

$$
p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right) . \tag{12}
$$

When learning the network that generates the parameters in Equation (12), the deep neural network takes as input the sample at time t, $\mathbf{x}_t$, in addition to the time step t itself, in order to account for the variance schedule of the forward process; as different time steps are associated with different noise levels, the model must learn to undo these individually.

Like the forward process, the reverse process is a Markov chain, and we can write the joint probability of a sequence of samples as a product of conditionals and the marginal probability of $\mathbf{x}_T$,

$$
p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t) . \tag{13}
$$

Here, the distribution $p(\mathbf{x}_T)$ is the same as $q(\mathbf{x}_T)$, namely the pure noise distribution $p_{\text{prior}} = \mathcal{N}(\mathbf{x}_T; 0, \mathbf{I})$. This means that the generation process starts with Gaussian noise, followed by sampling from the learned individual steps of the reverse process.

Note that while the original data-samples (e.g., images) do not have zero off-diagonal covariance (neighbouring pixels contain information about each other), the noise added to the original samples was diagonal, meaning that we can assume that the variance of the removed noise is also diagonal, i.e. $\boldsymbol{\Sigma} = \sigma \cdot \mathbf{I}$ for some scalar value σ. In the case of a diagonal covariance matrix $\boldsymbol{\Sigma}$, the mean and the variance in each dimension can be estimated separately, and the multivariate density function can be described in terms of a product of univariate Gaussians. If the variance is given, we merely have to predict the mean. In [Nichol and Dhariwal, 2021], it is shown empirically that learning the

$$p_{\boldsymbol{\theta}}\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\mathbf{x}_t, t\right), \sigma_t \mathbf{I}\right)$$

Figure 4: The general idea of the reverse diffusion process is to go backwards in time, starting with a sample $\mathbf{x}_T \sim p_{\text{prior}}$ and end up with $\mathbf{x}_0$ that (approximately) comes from p_{complex}. Compare this to the forward process in Figure 1 to realise that the major difference between the forward and the backward process is that when the forward process was *defined* by Equation (2), the reverse needs to be *learned*, cf. Equation (12).

covariance can improve the quality of the generated samples, but this is not something we will focus on here. We will rather predict only the mean of the reversed diffusion process-distribution, while the variance follows a schedule parameterised by t. The general process of the reverse process is thus as shown in Figure 4. We start by a sample from p_{prior}, use Equation (12) powered by the neural network with parameters $\boldsymbol{\theta}$, and eventually obtain $\mathbf{x}_0$. We will discuss two different reverse processes, namely the *Denoising Diffusion Probabilistic Model* [Ho et al., 2020, Nichol and Dhariwal, 2021] and *Denoising Diffusion Implicit Models* [Song et al., 2020]. However, we will first discuss how to define the loss function of the neural network.

2.3 The loss

What objective are we optimising when training the neural network to learn Equation (12)? All generative models attempt to learn the distribution of their training data, so it would make sense to maximise the likelihood assigned to $\mathbf{x}_0$ by the model. Calculating this would require us to marginalise over all steps from $t = T$ down to $t = 1$,

$$p_{\boldsymbol{\theta}}(\mathbf{x}_0) = \int p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \,. \tag{14}$$

Maximising Equation (14) gives the process $p_{\boldsymbol{\theta}}$ over $\mathbf{x}_T \rightarrow \mathbf{x}_{T-1} \ldots \mathbf{x}_1 \rightarrow \mathbf{x}_0$ that has the largest log likelihood of producing the observed $\mathbf{x}_0$ from the noise $\mathbf{x}_T$. However, evaluating the above expression involves integrating over all possible trajectories from noise to the data manifold, which is intractable. Instead we can maximise a lower bound of the log likelihood, taking a page out of the book of variational autoencoders.

To get to these results we will first discuss some results regarding variational inference (Section 2.3.1) and the VAE (Section 2.3.2). Unfortunately, some of the syntax used by the community behind these results differ from what is used elsewhere in this document. Nevertheless, we have been true to the original lingo in our description, and then try to "translate" the core concepts and ideas back to our language in Section 2.3.3.

2.3.1 Variational Lower Bound

Imagine that we are given a process $\mathbf{z} \rightarrow \mathbf{x}$ that can generate data samples $\mathbf{x}$ from latent variables $\mathbf{z}$. The latent variable can e.g. contain information about the properties of an image, and through the process, denoted $p(\mathbf{x}|\mathbf{z})$, the properties manifest into an actual image $\mathbf{x}$. We would like to know the reverse process, i.e. how to obtain $\mathbf{z}$ from $\mathbf{x}$. Knowing the distribution of $\mathbf{z}$, we could try to use Bayes'

rule,

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z})p(\mathbf{x}|\mathbf{z})}{\int p(\mathbf{z}, \mathbf{x})d\mathbf{z}}.$$

However, in general we do not know how to evaluate the integral $\int p(\mathbf{z}, \mathbf{x}) \, d\mathbf{z}$. Therefore, we cannot calculate the denominator in the above expression, and instead choose to approximate $p(\mathbf{z}|\mathbf{x})$ by a function $q_\phi(\mathbf{z}|\mathbf{x})$. This function can come from an approximation family $\mathcal{Q}$, where we seek to select the one that minimises some distance measure Δ between q_ϕ and the true distribution p. Formally,

$$\hat{q}_\phi(\mathbf{z}|\mathbf{x}) = \arg\min_{q \in \mathcal{Q}} \Delta(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})). \tag{15}$$

So, if for instance we let $\mathcal{Q}$ be the set of all Gaussian distributions then $\phi = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ is the parameterisation that identifies each member of $\mathcal{Q}$, and Equation (15) abstractly describes how the best parameterisation is to be chosen. For reasons beyond our control, the Kullbach-Leibler (KL) divergence $\mathcal{D}_{\mathrm{KL}}$ has been deemed a favourable distance measure, since it has some nice mathematical properties. It is calculated as

$$\mathcal{D}_{\mathrm{KL}}(q||p) = \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \, d\mathbf{x} = \mathbb{E}_{\mathbf{x} \sim q} \left[\log \frac{q(\mathbf{x})}{p(\mathbf{x})} \right],$$

and quantifies how different the probability distribution p is from another distribution q. The shorthand $\mathbb{E}_{\mathbf{x} \sim q}$ indicates that the expectation is calculated using values of $\mathbf{x}$ coming from q. The KL divergence is always positive, but failing to be symmetrical under the interchange of p and q is one of its less desirable mathematical properties. Figure 5 gives an example using two Gaussians, where the KL divergence is the area under the red curve. Note that the KL divergence cannot be negative, even if the red curve itself should sometimes take negative values.

Figure 5: The KL divergence between two Gaussians, here from $q \sim \mathcal{N}(x; \mu = 1, \sigma^2 = 1)$ (green) to $p \sim \mathcal{N}(x; \mu = 0, \sigma^2 = 4)$ (blue) is given by the area under the red curve.

Armed with the KL divergence, and using that the probability distribution of $q_\phi(\mathbf{z}|\mathbf{x})$ is normalised,

Figure 6: In a variational autoencoder, the encoder $q(\mathbf{z}|\mathbf{x})$ defines a distribution over latent variables $\mathbf{z}$ given observations $\mathbf{x}$, and the decoder $p(\mathbf{x}|\mathbf{z})$ defines the distribution over observations given the latent variables.

i.e. $\int q_\phi(\mathbf{z}|\mathbf{x})d\mathbf{z} = 1$, we can now rewrite the log likelihood over the observed data, $\log p(\mathbf{x})$, as

$$
\begin{aligned}
\log p(\mathbf{x}) &= \log p(\mathbf{x}) \int q_\phi(\mathbf{z}|\mathbf{x})d\mathbf{z} && \text{Multiply by } \int q_\phi(\mathbf{z}|\mathbf{x})d\mathbf{z} = 1 \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x})d\mathbf{z} && \text{Bring } p \text{ into integral} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x})\right] && \text{Expectation} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}\right] && \text{Multiply by } \frac{p(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} = 1; \text{ use chain rule} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x})}\right] && \text{Multiply by } \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} = 1 \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right] && \text{Split up} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] + \mathcal{D}_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) && \text{Definition of KL divergence} \\
&\geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] && \text{KL divergence always positive}.
\end{aligned}
$$

The quantity on the last line is known as the Evidence Lower Bound (ELBO), and it is a *lower bound* on $\log p(\mathbf{x})$ because $\mathcal{D}_{\mathrm{KL}}$ is always positive. This step is often skipped in lecture notes and papers, referring to "Jensen's inequality", which generalises that the secant line of a convex function lies above the graph of the function itself. In the standard formulation of variational autoencoders (VAEs), the objective is to maximise the ELBO. The term *variational* refers to the optimisation of $q_\phi(\mathbf{z}|\mathbf{x})$ from the family $\mathcal{Q}$ of potential approximation functions.

Note that the true data distribution $p(\mathbf{x})$ is constant with respect to q_ϕ (the real world doesn't care which function we choose for approximating it), meaning that the ELBO and the KL divergence sum to a constant under optimisation of ϕ. Therefore, maximising the ELBO with respect to ϕ is identical to a minimisation of $\mathcal{D}_{\mathrm{KL}}$, corresponding to finding an optimal model for approximating the true latent distribution.

2.3.2 Variational Autoencoders

An autoencoder is a neural network consisting of an encoder q that transforms observed data $\mathbf{x}$ to a latent representation $\mathbf{z}$, and a decoder p that transforms the data from the latent representation back to the original, see Figure 6. We do not know the ground truth encoder $q(\mathbf{z}|\mathbf{x})$ or the decoder $p(\mathbf{x}|\mathbf{z})$, but we can estimate them using parameterised models $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{x}|\mathbf{z})$, where the parameters ϕ and θ have to be optimised.

In order to optimise using the ELBO as loss function, we need to write it in terms we can calculate,

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{p(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \qquad\qquad \text{Definition of ELBO.}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \qquad\qquad \text{Chain rule.}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \qquad \text{Split up.}$$

$$= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right]}_{\text{reconstruction term}} - \underbrace{\mathcal{D}_{\mathrm{KL}} \left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}) \right)}_{\text{prior matching term}} \qquad \text{Definition of KL divergence}\,.$$

When optimising the ELBO we thus learn two distributions; the prior matching term optimises the parameters ϕ of the encoding function $q_\phi(\mathbf{z}|\mathbf{x})$ to match the true latent distribution $p(\mathbf{z})$, i.e. it encourages the approximate posterior to be similar to the prior on the latent variable, while the reconstruction term optimises the parameters θ of the decoding function $p_\theta(\mathbf{x}|\mathbf{z})$, maximising the expected probability density assigned to the data given the latent variables. Optimising the parameters ϕ and θ jointly is a defining feature of VAEs.

In order to calculate the loss for each choice of parameters, we have to specify a prior on the latent variables, and for VAEs (as for Gaussian diffusion), the common choice is a standard multivariate Gaussian,

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})\,.$$

During training, we get data samples $\mathbf{x}$ by drawing from our training data. Modelling the encoder as a multivariate Gaussian with unknown mean $\boldsymbol{\mu}_\phi$ and covariance σ_ϕ, assuming diagonal covariance,

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}, \boldsymbol{\mu}_\phi(\mathbf{x}), \sigma_\phi^2(\mathbf{x})\mathbf{I})\,,$$

the KL divergence term can be computed analytically. The reconstruction term can be approximated using a Monte Carlo (MC) estimate (which is a fancy way of saying "random sampling"), see Appendix A.1. We thus rewrite our objective as

$$\arg\max_{\phi,\theta} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right] - \mathcal{D}_{\mathrm{KL}} \left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}) \right) \approx \arg\max_{\phi,\theta} \sum_{l=1}^{L} \log p_\theta(\mathbf{x}|\mathbf{z}^l) - \mathcal{D}_{\mathrm{KL}} \left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}) \right)\,,$$

where the Monte Carlo action happens as we sample the $\mathbf{z}^l$ from $q_\phi(\mathbf{z}|\mathbf{x})$ for every $\mathbf{x}$ in the training data. After training, new data can be generated by sampling directly from the latent space via $p(\mathbf{z})$ and running the decoder on these samples.

2.3.3 Back to the diffusion

The diffusion model story is the reverse of the variational autoencoder story from Section 2.3.2: We start with an object $\mathbf{x}_0$ that we gradually convert to noise through the known process $q(\mathbf{x}_t|\mathbf{x}_{t-1})$. This is in contrast to the VAEs setting, as our forward process – corresponding to the encoding to latent variables – is not learned, but fixed. We therefore do not have a parameterised q_ϕ. However, the reverse process has to be learned, so we still have the parameterised p_θ. We thus only need one neural network model (that learns the reverse process), analogous to the decoder part of the VAE. Nevertheless, we can make use of the training objective used by VAEs.

We have the conditional distribution from Equation (2), repeated here for convenience:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})\,, \tag{16}$$

and the joint distribution from Equation (13),

$$p(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\,, \tag{17}$$

9

where, as before, $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. Let us now derive the ELBO again, using the diffusion model notation and these two distributions:

$$
\begin{aligned}
\log p(\mathbf{x}_0) &= \log \int p(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \\
&= \log \int \frac{p(\mathbf{x}_{0:T}) q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \\
&= \log \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \\
&\geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \qquad\qquad \text{Jensen's inequality} \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \quad \text{Use Equation (16) and Equation (17).} \quad (18)
\end{aligned}
$$

We will later promote $\log p(\mathbf{x}_0)$ to be the (negative) loss function, so let us begin by discussing why this is reasonable. Obviously, $\log p(\mathbf{x}_0)$ evaluated at an element from the training data set gives the log likelihood of that specific example, i.e., the log likelihood that the reverse process starting from a random initialisation should end up at that specific data point. If this number is high, we have a reverse process that makes generating the data point likely. In other words, the reverse process has a good chance of producing $\mathbf{x}_0$. If the reverse process is able to give a reasonably sized log likelihood to all the examples in our training data, this indicates that the reverse process is well suited to generate our data. Overall, using the negative log likelihood summed over all data points in the training data set seems reasonable, and is therefore the strategy we would like to follow. Unfortunately, this quantity is not available to us, which is why we instead use the ELBO discussed in Section 2.3.1 and defined for diffusion models in Equation (18).

Next, we look at how the ELBO can be calculated efficiently. In the following, we use that for positive real numbers a and b, $\log a \cdot b = \log a + \log b$, and $\log a/b = \log a - \log b$. Using this in Equation (18), we obtain

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}_0) &\geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_1) \right] + \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_1) \right] + \mathbb{E}_{q(\mathbf{x}_{T-1:T}|\mathbf{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_{T-1})} \right] \\
&\quad + \sum_{t=1}^{T-1} \mathbb{E}_{q(\mathbf{x}_{t-1:t+1}|\mathbf{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right]. \qquad\qquad (19)
\end{aligned}
$$

The last equality follows from that the expectation of a function of the subset of variables $\mathbf{x}_{a:b}$ with $1 \leq a \leq b \leq T$, say $f(\mathbf{x}_{a:b})$ wrt. a distribution $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$, is given by taking the expectation only over the variables $\mathbf{x}_{a:b}$:

$$
\begin{aligned}
\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[f(\mathbf{X}_{a:b}) \right] &= \int_{\mathbf{x}_{1:T}} q(\mathbf{x}_{1:T}|\mathbf{x}_0) \cdot f(\mathbf{x}_{a:b}) \, d\mathbf{x}_{1:T} \\
&= \int_{\mathbf{x}_{1:T}} q(\mathbf{x}_{a:b}|\mathbf{x}_0) \cdot q(\mathbf{x}_{1:a-1,b+1:T}|\mathbf{x}_0, \mathbf{x}_{a:b}) \cdot f(\mathbf{x}_{a:b}) \, d\mathbf{x}_{1:T} \\
&= \int_{\mathbf{x}_{a:b}} q(\mathbf{x}_{a:b}|\mathbf{x}_0) f(\mathbf{x}_{a:b}) \underbrace{\int_{\mathbf{x}_{1:a-1,b+1:T}} q(\mathbf{x}_{1:a-1,b+1:T}|\mathbf{x}_0, \mathbf{x}_{a:b}) \, d\mathbf{x}_{1:a-1,b+1:T}}_{\text{The inner integral equals 1.}} d\mathbf{x}_{a:b} \\
&= \int_{\mathbf{x}_{a:b}} q(\mathbf{x}_{a:b}|\mathbf{x}_0) f(\mathbf{x}_{a:b}) \, d\mathbf{x}_{a:b} \\
&= \mathbb{E}_{q(\mathbf{x}_{a:b}|\mathbf{x}_0)} \left[f(\mathbf{X}_{a:b}) \right]. \qquad\qquad (20)
\end{aligned}
$$

(a) Forward process: $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ (b) Consistency term in Equation (21)

Figure 7: Motivation for the consistency term in Equation (21).

With the help of Equation (20), and remembering that $\mathcal{D}_{\mathrm{KL}}\left(q||p_{\boldsymbol{\theta}}\right) = \mathbb{E}_{\mathbf{x}\sim q}\left[\log\frac{q(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})}\right]$, we can simplify Equation (19) as follows:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}_0) \geq \overbrace{\mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_1)\right]}^{\text{Reconstruction}} - \overbrace{\mathbb{E}_{q(\mathbf{x}_{T-1}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_T|\mathbf{x}_{T-1})||p_{\boldsymbol{\theta}}(\mathbf{x}_T)\right)\right]}^{\text{Prior matching}}$$
$$- \sum_{t=1}^{T}\underbrace{\mathbb{E}_{q(\mathbf{x}_{t-1,t+1}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_t|\mathbf{x}_{t-1})||p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})\right)\right]}_{\text{Consistency terms}}, \qquad (21)$$

which is finally the (negative) loss function for training our diffusion model.

Let us look at each term of Equation (21) in turn: The reconstruction term represents how successful the model is on average at reconstructing a data point $\mathbf{x}_0$ after one step in the diffusion process. The prior matching term compares the average result of the forward process at $\mathbf{x}_T$ to $p_{\boldsymbol{\theta}}(\mathbf{x}_T)$. Since $p_{\boldsymbol{\theta}}(\mathbf{x}_T)$ is fixed to be the standard Gaussian already, the prior matching term will be disregarded in the following. This leaves us with the consistency term, which we will consider in detail with the help of Figure 7.

Starting with Figure 7a, this diagram shows the (forward) diffusion process evaluated from an initial $\mathbf{x}_0$. We sample from $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ at each time-step t, and end up at $\mathbf{x}_T$. The figure indicates only a handful of trajectories using $q(\mathbf{x}_t|\mathbf{x}_{t-1}) \sim \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t\,\mathbf{I}\right)$ (see Equation (3)), but we can of course perform this as many times as we want. As defined in Equation (10), we also know that $q(\mathbf{x}_t|\mathbf{x}_0) \sim \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0, (1-\bar{\alpha}_t)\,\mathbf{I}\right)$.

In Figure 7b, we focus on a given t and can reason as follows:

> "If the reverse process $p(\mathbf{x}_t|\mathbf{x}_{t+1})$ works well at time t, we should be able to sample from $p(\mathbf{x}_t|\mathbf{x}_{t+1})$ and get values of $\mathbf{x}_t$ similar to what we got when sampling using the forward process $q(\mathbf{x}_t|\mathbf{x}_{t-1})$."

This means that the distribution for the reverse process at time t (red area in Figure 7b) should be as close as possible to the (green) forward-process at the same time.

As indicated in Figure 7b, one idea to operationalise this insight is to take the samples obtained by the forward process from $\mathbf{x}_0$ at time $t+1$, and use those to start off the sampling of $p(\mathbf{x}_t|\mathbf{x}_{t+1})$. The resulting distribution should be compared to the forward process that we can obtain by looking at samples from $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ initialized at time $t-1$ with samples obtained by starting from $\mathbf{x}_0$.

Armed with this intuition, we turn to defining $p(\mathbf{x}_t|\mathbf{x}_{t+1})$. First, and as already noted, we know that the *forward* process will be a Gaussian distribution. Since the *reverse* is supposed to mimic that distribution over $\mathbf{x}_t$, it is reasonable to enforce that the reverse process should be Gaussian, too. Finally, we will use the KL divergence as a means to compare distributions, and we now aim to formalise the general idea. To minimise $\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_t|\mathbf{X}_{t-1})||p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{X}_{t+1})\right)$ we must let the two conditioning variables $\mathbf{X}_{t-1}$ and $\mathbf{X}_{t+1}$ be related to our initial $\mathbf{x}_0$. In principle we can think that we are sampling the forward process starting from $\mathbf{x}_0$ a number of times and average $\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_t|\mathbf{X}_{t-1})||p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{X}_{t+1})\right)$ over these samples. Mathematically we here use that $\mathbf{X}_{t-1} \sim q(\mathbf{x}_{t-1}|\mathbf{x}_0)$ and $\mathbf{X}_{t+1} \sim q(\mathbf{x}_{t+1}|\mathbf{x}_0)$, and we end up with $\mathbb{E}_{q(\mathbf{X}_{t-1,t+1}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{X}_t|\mathbf{X}_{t-1})||p_{\boldsymbol{\theta}}(\mathbf{X}_t|\mathbf{X}_{t+1})\right)\right]$ as our measure of the quality of $p_{\boldsymbol{\theta}}$ at time t. This is exactly the consistency term at time t in Equation (21). Notice that since both $p_{\boldsymbol{\theta}}$ and q are Gaussian distributions, we can calculate the KL divergence analytically (see Equation (26) below).

2.3.4 Only one expectation

As briefly eluded to above, it is natural to define our loss function as $\mathcal{L}(\boldsymbol{\theta}) = -\log p_{\boldsymbol{\theta}}(\mathbf{x}_0)$ because a low log likelihood of an observed data-point $\mathbf{x}_0$, $\log p_{\boldsymbol{\theta}}(\mathbf{x}_0)$, implies poor predictive performance for that $\mathbf{x}_0$. Since we are unable to calculate this log likelihood analytically, we can instead utilise Equation (21), repeated here for ease of reference:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}_0) \geq \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_1)\right] - \mathbb{E}_{q(\mathbf{x}_{T-1}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_T|\mathbf{x}_{T-1})||p_{\boldsymbol{\theta}}(\mathbf{x}_T)\right)\right]$$

$$- \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_{t-1,t+1}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_t|\mathbf{x}_{t-1})||p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})\right)\right]$$

Notice here the term $\mathbb{E}_{q(\mathbf{x}_{t-1,t+1}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_t|\mathbf{x}_{t-1})||p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})\right)\right]$, where we for each t need to calculate the expectation over $\mathbf{X}_{t-1,t+1}|\mathbf{x}_0$. The source for the double expectation is that while the q-process works in the forward direction (from $t-1$ to t), the $p_{\boldsymbol{\theta}}$-process works in reverse (from $t+1$ to t). The idea now is to use Bayes rule to change the direction of the q-process in our definition of the loss function, meaning that we try to express

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

through the alternative representation where we now reverse the process, i.e., focus on the transition from t to $t-1$ instead of the forward process:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = q(\mathbf{x}_T|\mathbf{x}_0)\prod_{t=2}^{T} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0).$$

Ensuring that both processes $p_{\boldsymbol{\theta}}$ and q are reversed in time makes them easier to compare. Doing so, we find that the loss can be expressed as $-\log p_{\boldsymbol{\theta}}(\mathbf{x}_0) \leq \mathcal{L}_{\mathrm{vlb}} := \sum_{t=0}^{T} \mathcal{L}_t$ with

$$\begin{aligned} \mathcal{L}_0 &= -\mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_1)\right] \\ \mathcal{L}_{t-1} &= \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)\right)\right], \quad \text{for } 2 \leq t \leq T \\ \mathcal{L}_T &= \mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T)\right), \end{aligned} \tag{22}$$

where you should notice that the $p(\mathbf{x}_T)$ used in $\mathcal{L}_T$ is parameter-free (no $\boldsymbol{\theta}$) as it is simply assumed to be a standard Gaussian.

Calculating $\mathcal{L}_0$ and $\mathcal{L}_T$ is straightforward, and – as above – we argue that $\mathcal{L}_T$ can be neglected. To evaluate Equation (22), we calculate $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ using Bayes' rule:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)}. \tag{23}$$

We already know how to calculate two of the terms in Equation (23) (see Equation (10)), which are

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}),$$
$$q(\mathbf{x}_{t-1}|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1 - \bar{\alpha}_{t-1})\mathbf{I}).$$

For the final term, we utilise that $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is Markovian so that $q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1})$, defined in Equation (3), so that

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}).$$

Now, we have defined all the terms used to calculate $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ in Equation (23), and some pencil pushing reveals that

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}\right), \tag{24}$$

with

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\bar{\alpha}_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t; \quad \tilde{\beta}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t. \tag{25}$$

To summarise, the loss can be decomposed as $\mathcal{L} = \sum_{t=0}^{T}\mathcal{L}_t$, and if we zoom in on the contributions $2 \leq t \leq T$, then $\mathcal{L}_{t-1}$ is determined by a KL divergence term between two Gaussian distributions: $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$, that we now know how to calculate, and $p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)$, that we will choose ourselves, i.e., optimise with respect to the parameters $\boldsymbol{\theta}$.

Moving on, we note that the KL divergence between two multivariate Gaussian distributions, $\mathcal{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q)||\mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)\right)$, can be calculated as

$$\mathcal{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q)||\mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)\right) = \frac{1}{2}\left[\log\frac{|\boldsymbol{\Sigma}_p|}{|\boldsymbol{\Sigma}_q|} - d + \mathrm{tr}(\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\Sigma}_q) + \left(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right)^{\top}\boldsymbol{\Sigma}_p^{-1}\left(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right)\right]. \tag{26}$$

We want to minimise this KL divergence, and can choose both $\boldsymbol{\mu}_p$ and $\boldsymbol{\Sigma}_p$ freely when doing so. First, note that we know $\boldsymbol{\Sigma}_q = \tilde{\beta}_t\mathbf{I}$, and choose $\boldsymbol{\Sigma}_p$ to be the same. This gives us

$$\mathcal{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q)||\mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)\right) = \frac{1}{2}\left[\left(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right)^{\top}\boldsymbol{\Sigma}_q^{-1}\left(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right)\right]$$
$$= \frac{1}{2\tilde{\beta}_t}\left|\left|\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\right|\right|_2^2. \tag{27}$$

When minimising $\mathcal{L}_{t-1}$, we would therefore prefer to choose $p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)$ as a Gaussian with mean $\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0)$ to perfectly match $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ in Equation (24). However, we are not able to calculate $\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0)$, since the p-distribution is only conditioned on $\mathbf{x}_t$, while $\mathbf{x}_0$ is unknown. Basically, what we try to do is force the $p_{\boldsymbol{\theta}}$ - distribution that *is not* conditioned on $\mathbf{x}_0$ to fit with the q-distribution that *is* conditioned on $\mathbf{x}_0$ as we minimise $\mathcal{L}_{t-1}$.

The solution is to train a neural network to guess what $\mathbf{x}_0$ is. Basically, we create a neural network with parameters $\boldsymbol{\theta}$ that takes as input the noisy object $\mathbf{x}_t$ and the time-index t, and outputs a best guess on the final $\mathbf{x}_0$. We use $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ to denote this "predicted $\mathbf{x}_0$".

Using

$$\boldsymbol{\mu}_q = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\bar{\alpha}_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t \qquad \text{(From Equation (25))}$$

$$\boldsymbol{\mu}_p = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \frac{\sqrt{\bar{\alpha}_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t \qquad \text{(Substituting } \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \text{ for the unknown } \mathbf{x}_0)$$

in Equation (27), yields

$$\mathcal{L}_{t-1} = \frac{1}{2\tilde{\beta}_t}\frac{\bar{\alpha}_{t-1} \cdot \beta_t^2}{(1 - \bar{\alpha}_t)^2} \cdot \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[||\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \mathbf{x}_0||_2^2\right]. \tag{28}$$

Optimizing our neural network to minimise a loss similar to Equation (28) will help us define a generative process that allocates as high a likelihood to the training data as possible. We will soon discuss how to use this to generate new samples that (approximately) are sampled from p_{complex}, but first we will summarise what we have found so far.

2.3.5 Summary of the diffusion process and model training loss

We will consider how to efficiently optimise this loss wrt. $\boldsymbol{\theta}$ in the following section, but let us first summarise where we are and how we got here:

- Starting from a predetermined and fixed diffusion model $q(\mathbf{x}_t|\mathbf{x}_{t-1})$, we gradually diffuse a data-point $\mathbf{x}_0$ from the data distribution p_{complex} into a point $\mathbf{x}_T$ from a simple distribution p_{prior}, typically assumed to be a standard Gaussian.

- We now aim to reverse the diffusion process, that is, find the "inverse" of $q(\mathbf{x}_t|\mathbf{x}_{t-1})$, which we denote $p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)$. We have the benefit of knowing where the reverse process is supposed to end up, i.e., we know that the marginal distribution for $t = 0$, $p_{\boldsymbol{\theta}}(\mathbf{x}_0)$, must be equal to p_{data}, which again approximates p_{complex}.

- Starting out with the goal of maximising the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}_0) = \int_{\mathbf{x}_{1:T}} p(\mathbf{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t) \, d\mathbf{x}_{1:T}$, we ended up with the goal of minimising terms like

$$\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[\mathcal{D}_{\text{KL}} \left(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) || p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t) \right) \right] .$$

We argued that this is obtained by choosing $p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to be a Gaussian with variance depending only on the scheduling parameters $\{\beta_t\}_{t=0}^{T}$. The mean, on the other hand, should be determined as a combination of $\mathbf{x}_t$ and $\mathbf{x}_0$, because we are starting from $\mathbf{x}_t$ and taking a step intended to be in the direction of $\mathbf{x}_0$. Unfortunately, though, $\mathbf{x}_0$ is not known. Therefore, we create a neural network with weights $\boldsymbol{\theta}$ that uses the current value $\mathbf{x}_t$ and the time-index t as inputs, and outputs an estimate for $\mathbf{x}_0$, denoted $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$. Loss terms like Equation (28) help optimise $\boldsymbol{\theta}$.

- Before continuing on to actually optimise the loss, consider first why one would expect that it is even possible to train a good model $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$. Let us first consider $t = 1$. Now, as the network is asked to generate $\hat{\mathbf{x}}_0 := \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_1, t = 1)$, it has two pieces of information: First, if $\mathbf{x}_1$ is meaningful, then the generated $\hat{\mathbf{x}}_0$ must be "close" to $\mathbf{x}_1$ – because the pair $(\hat{\mathbf{x}}_0, \mathbf{x}_1)$ should be likely under the known model $q(\mathbf{x}_1|\mathbf{x}_0)$, and this process typically has a very small variance $(1 - \alpha_1)\,\mathbf{I}$. Second, $\hat{\mathbf{x}}_0$ should be likely under p_{complex}. For small values of t one would thus expect to have a reasonable learning signal for learning $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$. As t grows larger, the strength of the signal decreases. While it is still true that the predicted $\hat{\mathbf{x}}_0$ should be likely under p_{complex}, the guiding from $\mathbf{x}_t$ decreases in t (remember that the variance in $q(\mathbf{x}_t|\mathbf{x}_0)$ is $(1 - \bar{\alpha}_t)\mathbf{I}$, see Equation (10), and consider how quickly $\bar{\alpha}_t$ drops in Figure 2). To accept that it still works, though, the argument can be made that if $\hat{\mathbf{x}}_{t-1} := \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ is sufficiently precise for all $t \leq t'$ it is possible to utilise this for $t = t' + 1$ too: $\hat{\mathbf{x}}_{t'}$ must be fairly close to $\mathbf{x}_{t'+1}$ to ensure that the continuation is likely under $q(\mathbf{x}_{t+1}|\mathbf{x}_{t'})$ (notice how α_t drops off much slower than $\bar{\alpha}_t$ in Figure 2). Step by step, this argument can be used to argue that it should be possible to learn $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ for all $t \leq T$.

2.4 Reverse samplers

Now that we have the general principles for defining the loss, we turn back to how to define the reverse diffusion process.

2.4.1 The DDPM sampler

The Denoising Diffusion Probabilistic Model (DDPM) [Ho et al., 2020, Nichol and Dhariwal, 2021], learns the following parameterised Gaussian transitions:

$$p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t), \tilde{\beta}_t \, \mathbf{I}) , \tag{29}$$

with $\tilde{\beta}_t$ defined as in Equation (25). This leaves us with only having to learn $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$; compare this to Equation (12). Furthermore, [Ho et al., 2020] showed that rather than training a neural network to predict $\mathbf{x}_0$ (via $\tilde{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$) using the loss we found in Equation (28), it is beneficial to take one more look at this and try to reformulate the objective. Starting from Equation (11), we can represent $\mathbf{x}_0$ as

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}_t \right) , \tag{30}$$

where $\boldsymbol{\epsilon}_t$ is a multivariate standard Gaussian.

If we use the $\mathbf{x}_0$ from Equation (30) when calculating $\tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0)$ in Equation (25), we find that $\tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0)$, the expectation of the distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$, and therefore our desired choice for the expectation of $p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)$, too, can be re-written as

$$\tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_t \right) . \tag{31}$$

A main idea is now to use the representation Equation (31) for learning. This means, instead of learning the estimator $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$, we will approximate $\boldsymbol{\epsilon}_t$, and call the output from the learned model $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$. Using this idea, we can (after some pencil-pushing) reformulate the KL-loss in Equation (27) as

$$\mathcal{L}_{t-1} = \frac{1}{2\tilde{\beta}_t} \frac{(1-\alpha_t)^2}{\alpha_t(1-\bar{\alpha}_t)} \cdot \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[||\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \boldsymbol{\epsilon}_t||_2^2 \right] . \tag{32}$$

It was later shown empirically by [Ho et al., 2020] that one obtained better results ignoring the weighing-term in the loss, so that

$$\mathcal{L}_{t-1} \leftarrow \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[||\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \boldsymbol{\epsilon}_t||_2^2 \right] . \tag{33}$$

This gives us the training algorithm given in Algorithm 1.

Algorithm 1 Training using learning-rate η

1: **repeat**
2:     $\mathbf{x}_0 \sim p_{\text{data}}$
3:     $t \sim \text{Uniform}(\{1, \ldots, T\})$
4:     $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:     $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t} \cdot \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \cdot \boldsymbol{\epsilon}$
6:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}} ||\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \boldsymbol{\epsilon}||_2^2$
7: **until** converged

Given the DDPM's approach to finding $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$, one step of the denoising process from t to $t-1$ is done via

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \right) + \sqrt{\tilde{\beta}_t} \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{34}$$

Therefore, the reverse DDPM progresses as shown in Algorithm 2.

Algorithm 2 Reverse DDPM

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3:     **if** $t > 1$ **then**
4:         $\lambda \leftarrow 1$
5:     **else**
6:         $\lambda \leftarrow 0$
7:     **end if**
8:     $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
9:     $\mathbf{x}_{t-1} \leftarrow \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \right) + \lambda \cdot \sqrt{\tilde{\beta}_t} \cdot \mathbf{z}$
10: **end for**
11: **return** $\mathbf{x}_0$

Figure 8 shows the DDPM-sampler at work for a simple univariate problem. The figure shows a number of trajectories all starting from the same sampled $\mathbf{x}_T$, and as the value of t is reduced from $t = T$ to $t = 0$ (i.e., moving from right to left) the end-result produces samples from a bimodal distribution quite similar to the target-distribution used to generate the training-data.

Figure 8: Reverse process: Samples from DDPM, all starting from the same $\mathbf{x}_T$. The reverse process starts from $t = T$ and moves towards $t = 0$, so from right to left in the figure. The marginal drawn with a solid line at the far left shows the empirical distribution when repeating the samples several times, whereas the shaded area is the actual distribution p_{complex} used to generate the training-data in this example. Notice how trajectories starting from a *fixed* $\mathbf{x}_T$ still can cover the full distribution p_{complex}.

2.4.2 DDIM: Deterministic denoising

Referring again to the DDPM sampler in Equation (34), also depicted in Figure 8, we see that the denoising procedure is stochastic, and we cannot determine which latent location $\mathbf{x}_T$ is the origin of a sampled $\mathbf{x}_0$. However, given the noise prediction model $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$, we can alternatively create a *deterministic* denoising process to create samples from the target distribution. This is called Denoising Diffusion Implicit Models (DDIMs), introduced in [Song et al., 2020].

A key observation is that the DDPM objective in Equation (28) and Equation (33), the latter repeated here for convenience,

$$\mathcal{L}_{t-1} \; = \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[||\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \boldsymbol{\epsilon}_t||_2^2 \right] \,,$$

only depends on $q(\mathbf{x}_t|\mathbf{x}_0)$ (this is the distribution used in the expectation), and not explicitly on the joint distribution $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ or each of the noising-steps $q(\mathbf{x}_t|\mathbf{x}_{t-1})$. Since many joint distributions $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ can have the same marginals $q(\mathbf{x}_t|\mathbf{x}_0)$, [Song et al., 2020] explore an alternative, non-Markovian inference processes leading to new generative processes, but with the same surrogate objective function as DDPM: They consider a family of distributions defined as

$$q_{\sigma}(\mathbf{x}_{1:T}|\mathbf{x}_0) := q_{\sigma}(\mathbf{x}_T|\mathbf{x}_0) \prod_{t=2}^{T} q_{\sigma}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,.$$

Notice how the joint distribution $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is defined in the direction of reversed time; where we previously had $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ as our building-blocks, the joint of the forward process is now defined using the reverse view, and focus is on $q_{\sigma}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$. In order to arrive at an operational expression for the forward, i.e. the noising process, we can use Bayes' rule:

$$q_{\sigma}(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = \frac{q_{\sigma}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \cdot q_{\sigma}(\mathbf{x}_t|\mathbf{x}_0)}{q_{\sigma}(\mathbf{x}_{t-1}|\mathbf{x}_0)} \,. \tag{35}$$

Notice that this forward process, $q_{\sigma}(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)$, is *not* Markovian, as the distribution of $\mathbf{x}_t$ is conditioned on *both* $\mathbf{x}_{t-1}$ *and* $\mathbf{x}_0$. Please compare Equation (35) to Equation (3) and note this important difference.

Next, [Song et al., 2020] introduce the following functional representation[2] of the denoising process

---

[2]Note the subtle difference between our notation and what was used in [Song et al., 2020]: Our $\bar{\alpha}_t$ corresponds to α_t in [Song et al., 2020]. We have chosen to use this notation to be consistent with [Ho et al., 2020, Nichol and Dhariwal, 2021].

for all $t > 1$,

$$q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2} \cdot \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}, \sigma_t^2 \cdot \mathbf{I}\right). \qquad (36)$$

This is selected to ensure that $q_\sigma(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$ for all t, i.e. exactly our Equation (10). The expression in Equation (36) is similar to Equation (24), but notice that the variance term has been replaced by the parameter σ_t that can take any value, including zero. Since $q_\sigma(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$, just as is was for DDPM, and this term is the only one that played a role in the loss (see the expectation in Equation (33)), we see that training for the two approaches will be identical: A neural network trained as a DDPM model to estimate $\boldsymbol{\epsilon_\theta}(\mathbf{x}_t, t)$ can be directly used by the DDIM reverse process. This is really the exciting part: We do not have to train a different model for every choice of σ_t, but can use our DDPM objective to learn a generative process also for the *non-Markovian forward processes* parameterised by σ_t!

The difference between DDPM and DDIM instead lies in how we define $p_{\boldsymbol\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)$, and thus how to "translate" the estimate of $\boldsymbol{\epsilon_\theta}(\mathbf{x}_t, t)$ into a new sample $\mathbf{x}_{t-1}$. The DDPM lead us to investigate the distribution in Equation (24), and obtaining the sampling-step in Equation (34), repeated here for reference:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\boldsymbol{\epsilon_\theta}(\mathbf{x}_t, t)\right) + \sqrt{\tilde{\beta}_t} \cdot \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

Similarly, DDIM starts from Equation (36), and results in the sampling procedure

$$\mathbf{x}_{t-1} = \underbrace{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \boldsymbol{\epsilon_\theta}(\mathbf{x}_t, t)\right)}_{\text{Predicted } \mathbf{x}_0} + \underbrace{\sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2} \cdot \boldsymbol{\epsilon_\theta}(\mathbf{x}_t, t)}_{\text{Direction of } \mathbf{x}_t} + \underbrace{\sigma_t \cdot \mathbf{z}}_{\text{Noise}}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \qquad (37)$$

The DDIM-sampling of $\mathbf{x}_{t-1}$ consists of three parts: We first estimate $\mathbf{x}_0$ from $\boldsymbol{\epsilon_\theta}(\mathbf{x}_t, t)$, then (since $\mathbf{x}_{t-1}$ is not supposed to be $\mathbf{x}_0$ if $t > 0$, but rather a step on a path between $\mathbf{x}_0$ and $\mathbf{x}_t$), a move is made in the direction from $\mathbf{x}_0$ towards $\mathbf{x}_t$. Finally, we add Gaussian noise, scaled by the constant σ_t. The last part, regarding the level of noise, gives us extra flexibility here as compared to the DDPM. For DDPM, noise had to be scheduled according to $\tilde{\beta}_t$ as defined in Equation (25). However, in DDIM, σ_t has been introduced as an extra constant that we can choose ourselves. If we were to select

$$\sigma_t = \sqrt{\frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}}\sqrt{1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}},$$

the forward process becomes Markovian and we re-obtain the DDPM generative process. On the other hand, it has become customary to rather select $\sigma_t = 0$, in which case the generative process is deterministic. This model is a so-called *implicit* probabilistic model [Mohamed and Lakshminarayanan, 2017]. The generative process performed by such a model is coined a *denoising diffusion implicit model*, short DDIM, by [Song et al., 2020]. The important insight to remember is that the training objectives are equivalent for any value of σ_t, meaning that a model trained using the general DDPM process can be used for any generative process in the family, including DDIM, and thus be used to generate samples deterministically. This leads us to the sampling procedure in Algorithm 3.

Algorithm 3 Reverse DDPM

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3:     **if** $t > 1$ **then**
4:         $\lambda \leftarrow 1$
5:     **else**
6:         $\lambda \leftarrow 0$
7:     **end if**
8:     $\mathbf{x}_{t-1} \leftarrow \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \boldsymbol{\epsilon_\theta}(\mathbf{x}_t, t)\right) + \lambda \cdot \sqrt{1 - \bar{\alpha}_{t-1}} \cdot \boldsymbol{\epsilon_\theta}(\mathbf{x}_t, t).$
9: **end for**
10: **return** $\mathbf{x}_0$

Figure 9: Reverse process using DDIM: $\mathbf{x}_T$ is sampled from a isotropic Gaussian distribution, while the following steps are deterministic. The reverse process is shown, hence moving from right to left. The solid-line marginal at the far left shows the empirical distribution when repeating the samples several times, whereas the shaded area is the actual distribution p_{complex}. Notice how the deterministic relationship between the starting $\mathbf{x}_T$ and the resulting $\mathbf{x}_0$ makes it necessary to sample many $\mathbf{x}_T$ to get a decent approximation to p_{complex}.

Having removed the stochastic element from the denoising process, this is fully deterministic, and we obtain the concept of a unique latent space for the model. To concretise: given the same initial sample $\mathbf{x}_T$, the DDIM process, Equation (37), always generates the same final sample, while the DDPM process, Equation (34), generates different final samples for the same initial sample. This can be observed in Figure 9.

The number of steps, T, in the forward, i.e. noising, process is an important hyper-parameter in diffusion models. As mentioned earlier, a small step size lets the denoising process approach a Gaussian, so that the generative process modelled with a Gaussian conditional distributions is a good approximation. This motivates large values of T, as in [Ho et al., 2020]. However, in contrast to other generative models where the processes can be parallellised, all steps in the denoising process of DDPM must be performed sequentially. This gives the process of generation by denoising a significant disadvantage compared to other generative models. An important result related to the developments of DDIM is therefore that one can in principle only evaluate the reverse process on a subset of points instead of doing it at each and every t, often without significant drops in quality of the generated objects. As this finding is not at the core of the definition of the model itself, we do not discuss the result here, but refer the interested reader to [Song et al., 2020].

3 Text-prompting

3.1 Text-conditioning

The most basic attempt at text-conditioning is described in [Nichol et al., 2022], and is tightly connected to the structure of the model: The U-Net [Ronneberger et al., 2015], that is used for predicting $\hat{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$. The key is, for each text prompt y, to first encode the text into a sequence of k tokens, and feed that into a transformer model. The last layer of the transformer's token embeddings (a sequence of k feature vectors) is concatenated to the attention context at each layer in the U-Net. However, this approach reportedly leads to unstable image generation, and a need for stronger signal from the text-prompting. This is discussed next.

3.2 Classifier guidance

Text-conditioning (as presented in Section 3.1) gives us the opportunity to be gently steer the model towards objects that relate to the text-prompt. However, there is ample evidence showing that this

can be improved even further. In this subsection we describe *classifier guidance*, a way to guide the generation using an external classifier. We first describe the main idea and its mathematical foundation, then motivate how this can be used for text-guiding.

3.2.1 The main idea

Classifier guidance was introduced to diffusion models in [Dhariwal and Nichol, 2021]. The core idea is inspired by work on GANs, but we will develop it independently of GAN-lingo here. Our starting-point is that we have learned the reverse (unconditional) process $p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)$, but now want to extend that using information from a separately trained probabilistic classifier. The classifier takes an input-point, e.g. an image $\mathbf{x}$, and produce a distribution over the random variable Y that signifies the classes of the object (e.g., images can be of cars, horses, or people). This classifier is trained with weights $\boldsymbol{\xi}$ different from the diffusion-model, so we will use $p_{\boldsymbol{\xi}}(y|\mathbf{x})$ to signify the output of the classifier. Now, the idea is to define the reverse diffusion $p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_t|\mathbf{x}_{t+1}, y)$. This means we are making the reverse *conditional on the class*. This will make us say, e.g., "generate an image of a horse" instead of just asking for a general image (that could come from any class represented in the training data, of course including horses). [Dhariwal and Nichol, 2021] showed how this could be done using some simple tricks. First, assume that $Y \perp\!\!\!\perp \mathbf{X}_{t+1}|\mathbf{X}_t$. This means that any trace of the class variable that is present in $\mathbf{X}_{t+1}$ is also available after "cleaning up" to get $\mathbf{X}_t$. Therefore, if we want to guess on the class and have both $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ available, then using only the former will suffice. Then

$$
\begin{aligned}
p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_t|\mathbf{x}_{t+1}, y) &= \frac{p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_t, \mathbf{x}_{t+1}, y)}{p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_{t+1}, y)} \\
&\propto p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_t, \mathbf{x}_{t+1}, y) \\
&= p_{\boldsymbol{\theta}}(\mathbf{x}_{t+1}) \cdot p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1}) \cdot p_{\boldsymbol{\xi}}(y|\mathbf{x}_t) \\
&\propto p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1}) \cdot p_{\boldsymbol{\xi}}(y|\mathbf{x}_t),
\end{aligned}
\tag{38}
$$

where both proportionality statements follow from our interest on a distribution over only $\mathbf{x}_t$ so any distribution over the other variables are collected in proportionality constants. Further, the second equality follows from the conditional independence assumption: $p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(y|\mathbf{x}_t, \mathbf{x}_{t+1}) = p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)$. The consequence of Equation (38) is that we can utilise both the externally learned classifier $p_{\boldsymbol{\xi}}(y|\mathbf{x})$ and the reverse diffusion process $p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})$ as they are, and incorporate the classifier's output to guide the reverse diffusion towards creating objects seen as representatives from a particular class y. What is still problematic, though, is that we do not really know what distribution family $p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_t|\mathbf{x}_{t+1}, y)$ will belong to. Without knowing this, it will be difficult to sample from the distribution. $p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})$ has been assumed to be a Gaussian, but we have no reason to make strong assumptions about the structure of the likelihood $p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)$ (that is, as a function of $\mathbf{x}_t$, and not y). To simplify matters, we will use a one-step Taylor approximation of $\log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)$ around some value $\boldsymbol{\nu}$:

$$
\log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t) \approx \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t=\boldsymbol{\nu}} + (\mathbf{x}_t - \boldsymbol{\nu})^{\top} \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t=\boldsymbol{\nu}}.
$$

Notice that since $\log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t=\boldsymbol{\nu}} = \log p_{\boldsymbol{\xi}}(y|\boldsymbol{\nu})$ is constant in $\mathbf{x}_t$, we will simplify an write that $\log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t) \approx (\mathbf{x}_t - \boldsymbol{\nu})^{\top} \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t=\boldsymbol{\nu}} + C'$ where C' collects terms not related to $\mathbf{x}_t$. We can simplify further by utilising that also $\boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t=\boldsymbol{\nu}}$ is constant in $\mathbf{x}_t$ after inserting that $\mathbf{x}_t = \boldsymbol{\nu}$, and we get

$$
\log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t) \approx \mathbf{x}_t^{\top} \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t=\boldsymbol{\nu}} + C.
\tag{39}
$$

If we look back at Equation (38) and use Equation (39) to approximate the effect of the classifier, we see that we can express the classifier-guided reverse process as

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_t|\mathbf{x}_{t+1}, y) &= \log p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1}) + \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t) + C_1 \\
&\approx \log p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1}) + \mathbf{x}_t^{\top} \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t=\boldsymbol{\nu}} + C_2.
\end{aligned}
\tag{40}
$$

Remember that we previously decided that $\log p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})$ should be a Gaussian with given mean and covariance (see Equation (12)). Let's call these parameters $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ for simplicity. Further, note that a random variable $\mathbf{Z}$ follows the multivariate Gaussian with these parameters if and only if its log-density can be expressed as

$$
\log p(\mathbf{z}|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) = -\frac{1}{2} \mathbf{z}^{\top} \boldsymbol{\Sigma}_t^{-1} \mathbf{z} + \mathbf{z}^{\top} \boldsymbol{\Sigma}_t^{-1} \boldsymbol{\mu}_t + \text{constant},
\tag{41}
$$

where the term "constant" refers to terms independent of $\mathbf{z}$. Using this representation for $\log p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})$ in Equation (40) and deciding to do the Taylor approximation around $\boldsymbol{\nu} = \boldsymbol{\mu}_t$, we get

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_t|\mathbf{x}_{t+1}, y) &\approx \log p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1}) + \mathbf{x}_t^\top \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t = \boldsymbol{\mu}_t} + C_2 \\
&= -\frac{1}{2}\mathbf{x}_t^\top \boldsymbol{\Sigma}_t^{-1}\mathbf{x}_t + \mathbf{x}_t^\top \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \mathbf{x}_t^\top \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t = \boldsymbol{\mu}_t} + C \\
&= -\frac{1}{2}\mathbf{x}_t^\top \boldsymbol{\Sigma}_t^{-1}\mathbf{x}_t + \mathbf{x}_t^\top \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \mathbf{x}_t^\top \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}_t \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t = \boldsymbol{\mu}_t} + C \\
&= -\frac{1}{2}\mathbf{x}_t^\top \boldsymbol{\Sigma}_t^{-1}\mathbf{x}_t + \mathbf{x}_t^\top \boldsymbol{\Sigma}_t^{-1} \left[\boldsymbol{\mu}_t + \boldsymbol{\Sigma}_t \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t = \boldsymbol{\mu}_t} \right] + C \quad (42)
\end{aligned}
$$

Comparing Equation (42) to Equation (41) we realise that $\log p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_t|\mathbf{x}_{t+1}, y)$ is (approximately) the log-pdf of a Gaussian with mean $\boldsymbol{\mu}_t + \boldsymbol{\Sigma}_t \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t = \boldsymbol{\mu}_t}$ and covariance $\boldsymbol{\Sigma}_t$. Therefore the classifier-corrected reverse diffusion process is also (approximately) Gaussian, with known parameters. This result is quite fascinating: At each point in time, the classifier-guided reverse will sample from a distribution which has its mean nudged a little bit off the mean that we would have had without the classifier. The offset is $\boldsymbol{\Sigma}_t \ \boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t = \boldsymbol{\mu}_t}$. As $\boldsymbol{\Sigma}_t$ is simply a constant times the identity matrix, the contribution from the classifier guidance is in $\boldsymbol{\nabla}_{\mathbf{x}_t} \log p_{\boldsymbol{\xi}}(y|\mathbf{x}_t)_{|\mathbf{x}_t = \boldsymbol{\mu}_t}$. Remember that the vector $\boldsymbol{\nabla}_{\mathbf{x}} \log p_{\boldsymbol{\xi}}(y|\mathbf{x})$ points in the direction (in $\mathbf{x}$-space) for which $\log p_{\boldsymbol{\xi}}(y|\mathbf{x})$ increases the most from a starting-point $\mathbf{x}$. This means that when starting from an object $\mathbf{x}$, the clever thing to do to make it more like "something from class y" is to make an adjustment in the direction of $\boldsymbol{\nabla}_{\mathbf{x}} \log p_{\boldsymbol{\xi}}(y|\mathbf{x})$. This is also exactly what goes on in Equation (42). So, while the developments leading to Equation (42) were based on careful examination of the independence structure (Equation (38)) and a first-order Taylor approximation to the classifier's likelihood (Equation (39)), the end-result is also intuitive and easily acceptable. Finally, [Dhariwal and Nichol, 2021] argue that one may want more or less focus on the classifier guidance, and they therefore introduce a hyper-parameter s to determine this part's importance. Their final result is thus

$$
p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_{t-1}|\mathbf{x}_t, y) \sim \mathcal{N}\left(\hat{\boldsymbol{\mu}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + s\, \tilde{\beta}_t \boldsymbol{\nabla}_{\mathbf{x}} \log p_{\boldsymbol{\xi}}(y|\mathbf{x})_{|\mathbf{x} = \hat{\boldsymbol{\mu}}(\mathbf{x}_t, t)}; \tilde{\beta}_t \mathbf{I} \right) \ . \quad (43)
$$

3.2.2   Using classifier guidance for text

The description above is general, in the sense that we only require a probabilistic classifier $p_{\boldsymbol{\xi}}(y|\mathbf{x})$ and did not make hard requirements about how the classifier is learned, or what the classes actually mean. In this subsection we will briefly discuss how to use the classifier-guided setup for text prompting. Of course, if we let the classifier denote captions, and think that there for each image is one and only one caption that is correct, then we can in principle learn a classifier over this (extremely huge) set of classes, and use Equation (43) to guide the object generation towards an image that is relevant for the desired text. This is the goal of [Nichol et al., 2022]. However, to avoid the challenges of learning the classifier, the authors use CLIP [Radford et al., 2021].

Very briefly, CLIP utilised a data-set of 400 million (image, text)-pairs to learn two learn a multi-modal embedding space by jointly training an image encoder $\mathbf{f}_{\text{img}}(\cdot)$ and a text encoder $\mathbf{f}_{\text{txt}}(\cdot)$. These two models are learned so that the cosine similarity between an image $\mathbf{x}$ and caption y, $\mathbf{f}_{\text{img}}(\mathbf{x})^\top \mathbf{f}_{\text{txt}}(y)$ is maximised if y is the caption for the image $\mathbf{x}$.

With this in hand, [Nichol et al., 2022] propose to adapt the classifier-guidance in Equation (43) using $\mathbf{f}_{\text{img}}(\mathbf{x}_t)^\top \mathbf{f}_{\text{txt}}(y)$ as a proxy for $\log p_{\boldsymbol{\xi}}(y|\mathbf{x})$, giving the mean of the distribution $p_{\boldsymbol{\theta}, \boldsymbol{\xi}}(\mathbf{x}_{t-1}|\mathbf{x}_t, y)$ as

$$
\hat{\boldsymbol{\mu}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + s\, \tilde{\beta}_t \boldsymbol{\nabla}_{\mathbf{x}} \left(\mathbf{f}_{\text{img}}(\mathbf{x})^\top \mathbf{f}_{\text{txt}}(y) \right)_{|\mathbf{x} = \mathbf{x}_t} .
$$

A final comment needed here is that the original clip model trained $\mathbf{f}_{\text{img}}(\cdot)$ on *actual* images, while our use-case is to analyse *noisy* images $\mathbf{x}_t$, $t > 0$. Naturally, [Nichol et al., 2022] were therefore able to improve their results by retraining the image embeddings using noisy images, i.e., learn an image embedder $\mathbf{f}_{\text{img}}(\mathbf{x}_t, t)$ that receives a noisy image ($\mathbf{x}_t$) and the noise level (indirectly, via t) and produces an embedding in the same multi-modal embedding space.

3.3   Classifier-free guidance

In an attempt to do text-guiding without requiring the "external" classifier $p_{\boldsymbol{\xi}}(y|\mathbf{x})$ (or the CLIP embedding models $\mathbf{f}_{\text{img}}(\cdot)$ and $\mathbf{f}_{\text{txt}}(\cdot)$), [Ho and Salimans, 2021] proposed a way to train a diffusion-

model without the reliance on external models.

The starting-point is the estimator for $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ as learned in Equation (32). The idea is now to instead learn a *conditional model*, $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, y, t)$, that now also receives an image caption (or in practice a semantic embedding of the captions; it has been shown by [Saharia et al., 2022] that pre-trained language-model can be used to embed the textual content without loss in image generation quality) as its an additional input $\mathbf{y}$. This means that the model is learned from (image, text)-pairs. Every now and then, i.e., randomly and with a fixed probability, the textual information is suppressed in the input, leading the model to also learn $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{y} \leftarrow \emptyset, t)$.

At the time of image generation, the system starts by generating the embedding of the prompt, resulting in the embedding $\mathbf{y}$. Then, the guided $\boldsymbol{\epsilon}$-estimator is defined as

$$\tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{y}, t) = \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{y}, t) + s \cdot (\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{y}, t) - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \emptyset, t)),$$

where s is a hyper-parameter used to determine the strength of the guiding. [Ho and Salimans, 2021] show that if the $\hat{\boldsymbol{\epsilon}}$-model is exact, then

$$\boldsymbol{\nabla}_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t) \propto \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{y}, t) - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \emptyset, t). \tag{44}$$

Looking at the terms in Equation (44), we can thus see that the idea is to approximate the gradient of the score function by $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{y}, t) - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \emptyset, t)$, and add a correction to the estimated $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{y}, t)$ in that direction. Finally, $\tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{y}, t)$ can now take the role of $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ in the reversed sampler (being either DDMP and DDIM).

4   Current trends

We end these lecture notes by making some quick comments about directions of ongoing research, with examples of relevant works. A major trend in the use of diffusion models at the time of writing is to further improve text-based image-generation, also extending to both 3D synthesis [Poole et al., 2022] as well as generation of video [Xing et al., 2023]. Diffusion models are also used to generate structured datatypes, like text sequences [Gong et al., 2023] and protein folding [Wu et al., 2022], and has also been used for solving reinforcement learning problems [Ajay et al., 2023]. Furthermore, there is an ongoing development of new theoretical results. One interesting strand of work is related to Bayesian Flow Networks [Graves et al., 2023], that are able to generate samples both form from continuous *and* discrete variables.

A   Mathematical tricks

A.1   Monte Carlo estimation

In this subsection we will briefly discuss Monte Carlo methods for estimating an expectation via sampling. This is an often used trick to get away from calculating difficult integrals to find expected values. Assume we have a function $f_{\boldsymbol{\theta}}(\mathbf{x})$ for which we want to calculate the expected value, over a random variable $\mathbf{X}$ following some distribution $p_{\boldsymbol{\theta}}$. When the required calculations cannot be done analytically, the standard approach is to use a *sample average*, also known as the Monte Carlo (or simply MC) estimator:

$$\mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})] = \int_{\mathbf{x}} f_{\boldsymbol{\theta}}(\mathbf{x}) \cdot p_{\boldsymbol{\theta}}(\mathbf{x}) \, d\mathbf{x} \approx \frac{1}{M} \sum_{m=1}^{M} f_{\boldsymbol{\theta}} \left(\mathbf{x}_{(m)} \right), \tag{45}$$

where $\{\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(M)}\}$ are M independent samples we have drawn from the distribution $p_{\boldsymbol{\theta}}$. This estimator is powerful because we only need to be able to generate the samples $\{\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(M)}\}$, and we can use the MC technique almost[3] without any assumptions about the function $f_{\boldsymbol{\theta}}$ or the distribution $p_{\boldsymbol{\theta}}$. Let us use $\hat{\boldsymbol{\mu}}_M = \frac{1}{M} \sum_{m=1}^{M} f_{\boldsymbol{\theta}} \left(\mathbf{x}_{(m)} \right)$ as a notational shortcut for the sample mean based on M

---

[3]None of the functions considered here are problematic, so we will not discuss the assumptions underlying the MC estimator.

samples. Since this estimator is generated by sampling, it is a random variable. It can easily be shown that

$$\mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [\hat{\boldsymbol{\mu}}_M] = \mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})], \ \mathbb{V}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [\hat{\boldsymbol{\mu}}_M] = \frac{1}{M} \mathbb{V}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})].$$

Hence, the Monte Carlo estimator is unbiased (meaning that it has the correct value in expectation). Furthermore, if the variance $\mathbb{V}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})]$ is finite, it follows that the estimator's variance monotonically decreases towards zero when we increase the sampling effort (that is, as $M \to \infty$).

A.2   The reparameterisation trick

A key concept in probabilistic AI, which will be useful for evaluating the loss function when training our diffusion model, relies on the so-called *reparameterisation trick* [Kingma and Welling, 2014]. The starting point for understanding this very useful idea is the approximation of an expected value, see Appendix A.1. Next, assume that the expected value is part of a loss function and that we want to optimise the loss with respect to the parameters $\boldsymbol{\theta}$. We then need the gradient of the expectation we just approximated:

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})] = \boldsymbol{\nabla}_{\boldsymbol{\theta}} \int_{\mathbf{x}} p_{\boldsymbol{\theta}}(\mathbf{x}) \cdot f_{\boldsymbol{\theta}}(\mathbf{x}) \, \mathrm{d}\mathbf{x} = \underbrace{\int_{\mathbf{x}} p_{\boldsymbol{\theta}}(\mathbf{x}) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}) \, \mathrm{d}\mathbf{x}}_{\text{Suitable for MC}} + \underbrace{\int_{\mathbf{x}} f_{\boldsymbol{\theta}}(\mathbf{x}) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{x}) \, \mathrm{d}\mathbf{x}}_{\text{NOT suitable for MC}}. \quad (46)$$

Notice that while we can use the MC estimate from Equation (45) to approximate the first integral on the right-hand-side in Equation (46) by simply evaluating $\boldsymbol{\nabla}_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}_{(m)})$ for each sample $\mathbf{x}_{(m)}$, the MC estimate cannot be used to approximate the second integral. This is because neither $\boldsymbol{\nabla}_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{x})$ nor $f_{\boldsymbol{\theta}}(\mathbf{x})$ will in general be probability densities. The second integral is therefore not an expectation, and thus it makes no sense to approximate it by a sample average.

Here comes the trick: Assume that the distribution for $\mathbf{X}$ has a special form, namely that there exists a random variable $\mathbf{Y}$ following some distribution p, and a differentiable function $g(\mathbf{y}, \boldsymbol{\theta})$ so that $g(\mathbf{Y}, \boldsymbol{\theta})$ is distributed as $p_{\boldsymbol{\theta}}$. This is a mouthful, but the requirement is that while $\mathbf{Y} \sim p$ comes from a distribution that is *not* parameterised by $\boldsymbol{\theta}$, we can still find the function g so that $g(\mathbf{Y}, \boldsymbol{\theta})$ has the same distribution as $\mathbf{X}$, i.e., $g(\mathbf{Y}, \boldsymbol{\theta}) \sim p_{\boldsymbol{\theta}}$. This means that we can replace the expectation over $\mathbf{X} \sim p_{\boldsymbol{\theta}}$ with an expectation over $\mathbf{Y} \sim p$:

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})] = \boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{Y} \sim p} [f_{\boldsymbol{\theta}}(g(\mathbf{Y}, \boldsymbol{\theta}))] \quad (47)$$

$$= \mathbb{E}_{\mathbf{Y} \sim p} [\boldsymbol{\nabla}_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(g(\mathbf{Y}, \boldsymbol{\theta}))] \quad (48)$$

$$\approx \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{\nabla}_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(g(\mathbf{y}_{(m)}, \boldsymbol{\theta})). \quad (49)$$

Here, Equation (47) is a simple change of variables, Equation (48) holds because the distribution of $\mathbf{Y}$ does not depend on $\boldsymbol{\theta}$ and we can therefore interchange the expectation and the gradient, and Equation (49) simply uses Monte Carlo sampling to approximate the expectation in Equation (48), as we did in Equation (45). Note that we need to use the chain rule for derivatives to calculate $\boldsymbol{\nabla}_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(g(\mathbf{y}_{(m)}, \boldsymbol{\theta}))$. In applications, we typically select a low value of M for doing these calculations, sometimes as low as $M = 10$ or even $M = 1$, so that the approximation of the gradient can be calculated quickly.

Example: To see how this works, consider a situation where $p_{\boldsymbol{\theta}}(x) = \mathcal{N}(\theta_1, \theta_2^2)$, and let $f(x) = x^2/2$ so that $f'(x) = x$. We are interested in $\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})]$. Due to the simple structure of this problem, we can in fact find the solution analytically: $\mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [\frac{1}{2} \mathbf{X}^2] = \frac{1}{2} \cdot (\theta_1^2 + \theta_2^2)$, hence $\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})] = \boldsymbol{\theta}$. Nevertheless, let us also consider how the result can be obtained using the reparameterization trick. Define $g(y, \boldsymbol{\theta}) = \theta_1 + \theta_2 \cdot y$, and let Y be a random variable with $p(y) = \mathcal{N}(0, 1)$. The properties of the Gaussian distribution ensures that $g(Y, \boldsymbol{\theta}) \sim \mathcal{N}(\theta_1, \theta_2^2)$, which is equal to $p_{\boldsymbol{\theta}}$. We can now use the reparameterisation trick to approximate $\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})]$. Explicitly, we find

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{X} \sim p_{\boldsymbol{\theta}}} [f_{\boldsymbol{\theta}}(\mathbf{X})] \approx \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{\nabla}_{\boldsymbol{\theta}} f(\theta_1 + \theta_2 \cdot y) = \frac{1}{M} \sum_{m=1}^{M} \begin{bmatrix} \theta_1 + \theta_2 \cdot y_{(m)} \\ y_{(m)} \cdot (\theta_1 + \theta_2 \cdot y_{(m)}) \end{bmatrix},$$

where we used that $f(\theta_1 + \theta_2 \cdot y) = \frac{1}{2}(\theta_1 + \theta_2 \cdot y)^2$, and therefore

$$\boldsymbol{\nabla_\theta}\, f(\theta_1 + \theta_2 \cdot y) = \begin{bmatrix} \frac{\partial}{\partial \theta_1} f(\theta_1 + \theta_2 \cdot y) \\ \frac{\partial}{\partial \theta_2} f(\theta_1 + \theta_2 \cdot y) \end{bmatrix} = \begin{bmatrix} 2 \cdot \frac{1}{2}(\theta_1 + \theta_2 \cdot y) \cdot \frac{\partial}{\partial \theta_1}(\theta_1 + \theta_2 \cdot y) \\ 2 \cdot \frac{1}{2}(\theta_1 + \theta_2 \cdot y) \cdot \frac{\partial}{\partial \theta_2}(\theta_1 + \theta_2 \cdot y) \end{bmatrix} = \begin{bmatrix} (\theta_1 + \theta_2 \cdot y) \cdot 1 \\ (\theta_1 + \theta_2 \cdot y) \cdot y \end{bmatrix}.$$

Since $Y \sim \mathcal{N}(0,1)$ we know that $\mathbb{E}[Y] = 0$ and $\mathbb{E}[Y^2] = 1$, hence the reparameterization-trick will produce the right result in the limit:

$$\frac{1}{M} \sum_{m=1}^{M} \begin{bmatrix} \theta_1 + \theta_2 \cdot y_{(m)} \\ \theta_1 \cdot y_{(m)} + \theta_2 \cdot y_{(m)}^2 \end{bmatrix} \xrightarrow{M \to \infty} \begin{bmatrix} \theta_1 + \theta_2 \cdot \mathbb{E}[Y] \\ \theta_1 \cdot \mathbb{E}[Y] + \theta_2 \cdot \mathbb{E}[Y^2] \end{bmatrix} = \begin{bmatrix} \theta_1 + 0 \cdot \theta_2 \\ 0 \cdot \theta_1 + 1 \cdot \theta_2 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \boldsymbol{\theta}.$$

In this simple example we could thus calculate the exact value, and found that $\boldsymbol{\nabla_\theta}\, \mathbb{E}_{\mathbf{X} \sim p_\theta}\left[f_\theta(\mathbf{X}) \right] = \boldsymbol{\theta}$. We were also able to show that the same result was obtained by the reparameterisation trick in the limit when $M \to \infty$. This concludes the demonstration.

References

[Ajay et al., 2023] Ajay, A., Du, E., Gupta, A., Tenenbaum, J. B., Jaakkola, T. S., and Agrawal, P. (2023). Is conditional generative modeling all you need for decision making? In *The Eleventh International Conference on Learning Representations*.

[Dhariwal and Nichol, 2021] Dhariwal, P. and Nichol, A. (2021). Diffusion models beat GANs on image synthesis. *ArXiv*, abs/.2105.05233.

[Feller, 1949] Feller, W. (1949). On the theory of stochastic processes, with particular reference to applications. In *Proceedings of the [First] Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 403–433. University of California Press.

[Gong et al., 2023] Gong, S., Li, M., Feng, J., Wu, Z., and Kong, L. (2023). DiffuSeq: Sequence to sequence text generation with diffusion models. *ArXiv*, abs/2210.08933.

[Graves et al., 2023] Graves, A., Srivastava, R. K., Atkinson, T., and Gomez, F. (2023). Bayesian flow networks. *ArXiv*, abs/2308.07037.

[Ho et al., 2020] Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc.

[Ho and Salimans, 2021] Ho, J. and Salimans, T. (2021). Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. *ArXiv*, abs/1312.6114.

[Mohamed and Lakshminarayanan, 2017] Mohamed, S. and Lakshminarayanan, B. (2017). Learning in implicit generative models. *ArXiv*, abs/1610.03483.

[Nichol et al., 2022] Nichol, A., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., McGrew, B., Sutskever, I., and Chen, M. (2022). GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. *ArXiv*, abs/2112.10741.

[Nichol and Dhariwal, 2021] Nichol, A. Q. and Dhariwal, P. (2021). Improved denoising diffusion probabilistic models. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171. PMLR.

[Poole et al., 2022] Poole, B., Jain, A., Barron, J. T., and Mildenhall, B. (2022). DreamFusion: Text-to-3D using 2D diffusion. *ArXiv*, 2209.14988.

[Radford et al., 2021] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. *ArXiv*, abs/1505.04597.

[Saharia et al., 2022] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Gontijo-Lopes, R., Ayan, B. K., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. (2022). Photorealistic text-to-image diffusion models with deep language understanding. *ArXiv*, abs/2205.11487.

[Sohl-Dickstein et al., 2015] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France. PMLR.

[Song et al., 2020] Song, J., Meng, C., and Ermon, S. (2020). Denoising diffusion implicit models. *ArXiv*, abs/2010.02502.

[Wu et al., 2022] Wu, K. E., Yang, K. K., van den Berg, R., Zou, J. Y., Lu, A. X., and Amini, A. P. (2022). Protein structure generation via folding diffusion. *ArXiv*, abs/2209.15611.

[Xing et al., 2023] Xing, Z., Feng, Q., Chen, H., Dai, Q., Hu, H., Xu, H., Wu, Z., and Jiang, Y.-G. (2023). A survey on video diffusion models. *ArXiv*, abs/2310.10647.