Tutorial on Diffusion Models for Imaging and Vision

Tutorial on Diffusion Models for Imaging and Vision

Stanley Chan
Purdue University
stanchan@purdue.edu

now
the essence of knowledge
Boston — Delft

Foundations and Trends® in Computer Graphics and Vision

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

S. Chan. *Tutorial on Diffusion Models for Imaging and Vision*. Foundations and Trends® in Computer Graphics and Vision, vol. 16, no. 4, pp. 322–471, 2024.

ISBN: 978-1-63828-433-8
© 2025 S. Chan

Foundations and Trends® in Computer Graphics and Vision
Volume 16, Issue 4, 2024

Editorial Scope

Foundations and Trends® in Computer Graphics and Vision publishes survey and tutorial articles in the following topics:

- Rendering
- Shape
- Mesh simplification
- Animation
- Sensors and sensing
- Image restoration and enhancement
- Segmentation and grouping
- Feature detection and selection
- Color processing
- Texture analysis and synthesis
- Illumination and reflectance modeling
- Shape representation
- Tracking
- Calibration
- Structure from motion
- Motion estimation and registration
- Stereo matching and reconstruction
- 3D reconstruction and image-based modeling
- Learning and statistical methods
- Appearance-based matching
- Object and scene recognition
- Face detection and recognition
- Activity and gesture recognition
- Image and video retrieval
- Video analysis and event recognition
- Medical image analysis
- Robot localization and navigation

Information for Librarians

Foundations and Trends® in Computer Graphics and Vision, 2024, Volume 16, 4 issues. ISSN paper version 1572-2740. ISSN online version 1572-2759. Also available as a combined paper and online subscription.

Contents

Tutorial on Diffusion Models for Imaging and Vision

Stanley Chan

School of Electrical and Computer Engineering, Purdue University, USA; stanchan@purdue.edu

ABSTRACT

The astonishing growth of generative tools in recent years has empowered many exciting applications in text-to-image generation and text-to-video generation. The underlying principle behind these generative tools is the concept of *diffusion*, a particular sampling mechanism that has overcome some longstanding shortcomings in previous approaches. The goal of this tutorial is to discuss the essential ideas underlying these diffusion models. The target audience of this tutorial includes undergraduate and graduate students who are interested in doing research on diffusion models or applying these tools to solve other problems.

Stanley Chan (2024), "Tutorial on Diffusion Models for Imaging and Vision", Foundations and Trends® in Computer Graphics and Vision: Vol. 16, No. 4, pp 322–471. DOI: 10.1561/0600000112.

1

Variational Auto-Encoder (VAE)

A long time ago, in a galaxy far far away, we wanted to build a generator—a generator that generates texts, speeches, or images from some inputs with which we give to the computer. While this may sound magical at first, the problem has actually been studied for a long time. To kick off the discussion of this tutorial, we shall first consider the **variational autoencoder** (VAE). VAE was proposed by Kingma and Welling in 2014 [24]. According to their 2019 tutorial [23], the VAE was inspired by the Helmholtz Machine [10] as the marriage of graphical models and deep learning. In what follows, we will discuss VAE's problem setting, its building blocks, and the optimization tools associated with the training.

1.1 Building Blocks of VAE

We start by discussing the schematic diagram of a VAE. As shown in the figure below, the VAE consists of a pair of models (often realized by deep neural networks). The one located near the input is called an **encoder** whereas the one located near the output is called a **decoder**. We denote the input (typically an image) as a vector $\mathbf{x}$, and the output (typically another image) as a vector $\widehat{\mathbf{x}}$. The vector located in the middle between the encoder and the decoder is called a **latent variable**, denoted as $\mathbf{z}$. The job of the encoder is to extract a meaningful representation for $\mathbf{x}$, whereas the job of the decoder is to generate a new image from the latent variable $\mathbf{z}$ (Figure 1.1).

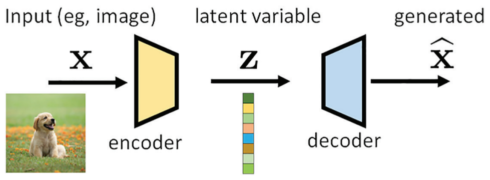


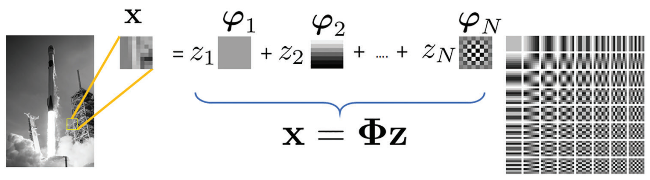
Figure 1.1: A variational autoencoder consists of an encoder that converts an input $\mathbf{x}$ to a latent variable $\mathbf{z}$, and a decoder that synthesizes an output $\widehat{\mathbf{x}}$ from the latent variable.

The latent variable $\mathbf{z}$ has two special roles in this setup. With respect to the input, the latent variable encapsulates the information that can be used to describe $\mathbf{x}$. The encoding procedure could be a lossy process, but our goal is to preserve the important content of $\mathbf{x}$ as much as we can. With respect to the output, the latent variable serves as the "seed" from which an image $\widehat{\mathbf{x}}$ can be generated. Two different $\mathbf{z}$'s should in theory give us two different generated images.

A slightly more formal definition of a latent variable is given below.

Definition 1.1 (Latent Variables [23]). In a probabilistic model, latent variables $\mathbf{z}$ are variables that we do not observe and hence are not part of the training dataset, although they are part of the model.

Example 1.1. Getting a latent representation of an image is not an alien thing. Back in the time of JPEG compression (which is arguably a dinosaur), we used discrete cosine transform (DCT) basis functions $\boldsymbol{\varphi}_n$ to encode the underlying image/patches of an image. The coefficient vector $\mathbf{z} = [z_1, \ldots, z_N]^T$ is obtained by projecting the image $\mathbf{x}$ onto the space spanned by the basis, via $z_n = \langle \boldsymbol{\varphi}_n, \mathbf{x} \rangle$. So, given an image $\mathbf{x}$, we can produce a coefficient vector $\mathbf{z}$. From $\mathbf{z}$, we can use the inverse transform to recover (i.e., decode) the image (Figure 1.2).



$\mathbf{x}$
$= z_1 \varphi_1 + z_2 \varphi_2 + \dots + z_N \varphi_N$
$\mathbf{x} = \mathbf{\Phi z}$

Figure 1.2: In discrete cosine transform (DCT), we can think of the encoder as taking an image $\mathbf{x}$ and generating a latent variable $\mathbf{z}$ by projecting $\mathbf{x}$ onto the basis functions.

In this example, the coefficient vector $\mathbf{z}$ is the latent variable. The encoder is the DCT transform, and the decoder is the inverse DCT transform.

The term "variational" in VAE is related to the subject of calculus of variations which studies optimization over functions. In VAE, we are interested in searching for the optimal probability distributions to describe $\mathbf{x}$ and $\mathbf{z}$. In light of this, we need to consider a few distributions:

- $p(\mathbf{x})$: The true distribution of $\mathbf{x}$. It is never known. The whole universe of diffusion models is to find ways to draw samples from $p(\mathbf{x})$. If we knew $p(\mathbf{x})$ (say, we have a formula that describes $p(\mathbf{x})$), we can just draw a sample $\mathbf{x}$ that maximizes $\log p(\mathbf{x})$.

- $p(\mathbf{z})$: The distribution of the latent variable. Typically, we make it a zero-mean unit-variance Gaussian $\mathcal{N}(0, \mathbf{I})$. One reason is that linear transformation of a Gaussian remains a Gaussian, and so this makes the data processing easier. Doersch [12] also has an excellent explanation. It was mentioned that any distribution can be generated by mapping a Gaussian through a sufficiently complicated function. For example, in a one-variable setting, the inverse cumulative distribution function (CDF) technique [7, Chapter 4] can be used for any continuous distribution with an invertible CDF. In general, as long as we have a sufficiently powerful function (e.g., a neural network), we can learn it and map the i.i.d. Gaussian to whatever latent variable needed for our problem.

- $p(\mathbf{z}|\mathbf{x})$: The conditional distribution associated with the **encoder**, which tells us the likelihood of $\mathbf{z}$ when given $\mathbf{x}$. We have no access to it. $p(\mathbf{z}|\mathbf{x})$ itself is *not* the encoder, but the encoder has to do something so that it will behave consistently with $p(\mathbf{z}|\mathbf{x})$.

- $p(\mathbf{x}|\mathbf{z})$: The conditional distribution associated with the **decoder**, which tells us the posterior probability of getting $\mathbf{x}$ given $\mathbf{z}$. Again, we have no access to it.

When we switch from the classical parameteric models to deep neural networks, the notion of latent variables is changed to *deep* latent variables. Kingma and Welling [23] gave a good definition below.

Definition 1.2 (Deep Latent Variables [23])**.** Deep Latent Variables are latent variables whose distributions $p(\mathbf{z})$, $p(\mathbf{x}|\mathbf{z})$, or $p(\mathbf{z}|\mathbf{x})$ are parameterized by a neural network.

The advantage of deep latent variables is that they can model very complex data distributions $p(\mathbf{x})$ even though the structures of the prior distributions and the conditional distributions are relatively simple (e.g., Gaussian). One way to think about this is that the neural networks can be used to estimate the mean of a Gaussian. Although the Gaussian itself is simple, the mean is a function of the input data, which passes through a neural network to generate a data-dependent mean. So the expressiveness of the Gaussian is significantly improved.

Let's go back to the four distributions above. Here is a somewhat trivial but educational example that can illustrate the idea:

Example 1.2. Consider a random variable $\mathbf{X}$ distributed according to a Gaussian mixture model with a latent variable $z \in \{1, \ldots, K\}$ denoting the cluster identity such that $p_Z(k) = \mathbb{P}[Z = k] = \pi_k$ for $k = 1, \ldots, K$. We assume $\sum_{k=1}^{K} \pi_k = 1$. Then, if we are told that we need to look at the k-th cluster only, the conditional distribution of $\mathbf{X}$ given Z is

$$p_{\mathbf{X}|Z}(\mathbf{x}|k) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \sigma_k^2\mathbf{I}).$$

The marginal distribution of $\mathbf{x}$ can be found using the law of total probability, giving us

$$p_{\mathbf{X}}(\mathbf{x}) = \sum_{k=1}^{K} p_{\mathbf{X}|Z}(\mathbf{x}|k) p_Z(k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \sigma_k^2\mathbf{I}). \tag{1.1}$$

Therefore, if we start with $p_{\mathbf{X}}(\mathbf{x})$, the design question for the encoder is to build a magical encoder such that for every sample $\mathbf{x} \sim p_{\mathbf{X}}(\mathbf{x})$, the latent code will be $z \in \{1, \ldots, K\}$ with a distribution $z \sim p_Z(k)$.

To illustrate how the encoder and decoder work, let's assume that the mean and variance are known and are fixed. Otherwise we will need to estimate the mean and variance through an expectation-maximization (EM) algorithm. It is doable, but the tedious equations will defeat the educational purpose of this illustration.

Encoder: How do we obtain z from $\mathbf{x}$? This is easy because at the encoder, we know $p_{\mathbf{X}}(\mathbf{x})$ and $p_Z(k)$. Imagine that you only have two class $z \in \{1, 2\}$. Effectively you are just making a binary decision of where the sample $\mathbf{x}$ should belong to. There are many ways you can do the binary decision. If you like the maximum-a-posteriori decision rule, you can check

$$p_{Z|\mathbf{X}}(1|\mathbf{x}) \gtrless_{\text{class } 2}^{\text{class } 1} p_{Z|\mathbf{X}}(2|\mathbf{x}),$$

and this will return you a simple decision: You give us $\mathbf{x}$, we tell you $z \in \{1, 2\}$.

Decoder: On the decoder side, if we are given a latent code $z \in \{1, \ldots, K\}$, the magical decoder just needs to return us a sample $\mathbf{x}$ which is drawn from $p_{\mathbf{X}|Z}(\mathbf{x}|k) = \mathcal{N}(\mathbf{x} \,|\, \boldsymbol{\mu}_k, \sigma_k^2 \mathbf{I})$. A different z will give us one of the K mixture components. If we have enough samples, the overall distribution will follow the Gaussian mixture.

This example is certainly oversimplified because real-world problems can be much harder than a Gaussian mixture model with known means and known variances. But one thing we realize is that if we want to find the magical encoder and decoder, we must have a way to find the two conditional distributions $p(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$. However, they are both high-dimensional.

In order for us to say something more meaningful, we need to impose additional structures so that we can generalize the concept to harder problems. To this end, we consider the following two proxy distributions:

- $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: The proxy for $p(\mathbf{z}|\mathbf{x})$, which is also the distribution associated with the *encoder*. $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ can be any directed graphical model and it can be parameterized using deep neural networks

[23, Section 2.1]. For example, we can define

$$(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) = \text{EncoderNetwork}_{\boldsymbol{\phi}}(\mathbf{x}),$$
$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z} \mid \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)). \tag{1.2}$$

This model is a widely used because of its tractability and computational efficiency.

- $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$: The proxy for $p(\mathbf{x}|\mathbf{z})$, which is also the distribution associated with the *decoder*. Like the encoder, the decoder can be parameterized by a deep neural network. For example, we can define

$$f_{\boldsymbol{\theta}}(\mathbf{z}) = \text{DecoderNetwork}_{\boldsymbol{\theta}}(\mathbf{z}),$$
$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x} \mid f_{\boldsymbol{\theta}}(\mathbf{z}), \sigma_{\text{dec}}^2\mathbf{I}), \tag{1.3}$$

where σ_{dec} is a hyperparameter that can be pre-determined or it can be learned.

The relationship between the input $\mathbf{x}$ and the latent $\mathbf{z}$, as well as the conditional distributions, are summarized in Figure 1.3. There are two nodes $\mathbf{x}$ and $\mathbf{z}$. The "forward" relationship is specified by $p(\mathbf{z}|\mathbf{x})$ (and approximated by $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$), whereas the "reverse" relationship is specified by $p(\mathbf{x}|\mathbf{z})$ (and approximated by $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$).

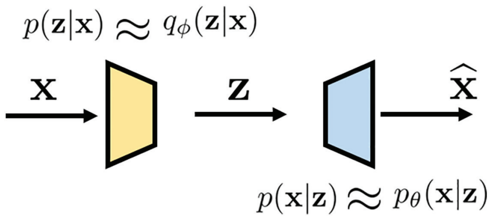

$p(\mathbf{z}|\mathbf{x}) \approx q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
$\mathbf{z}$
$\widehat{\mathbf{x}}$
$p(\mathbf{x}|\mathbf{z}) \approx p_\theta(\mathbf{x}|\mathbf{z})$

Figure 1.3: In a variational autoencoder, the variables $\mathbf{x}$ and $\mathbf{z}$ are connected by the conditional distributions $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$. To make things work, we introduce proxy distributions $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ and $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

Example 1.3. Suppose that we have a random variable $\mathbf{x} \in \mathbb{R}^d$ and a latent variable $\mathbf{z} \in \mathbb{R}^d$ such that

$$\mathbf{x} \sim p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \sigma^2\mathbf{I}),$$
$$\mathbf{z} \sim p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid 0, \mathbf{I}).$$

We want to construct a VAE. By this, we mean that we want to build two mappings Encoder$(\cdot)$ and Decoder$(\cdot)$. The encoder will take a sample $\mathbf{x}$ and map it to the latent variable $\mathbf{z}$, whereas the decoder will take the latent variable $\mathbf{z}$ and map it to the generated variable $\widehat{\mathbf{x}}$. If we *knew* what $p(\mathbf{x})$ is, then there is a trivial solution where $\mathbf{z} = (\mathbf{x} - \boldsymbol{\mu})/\sigma$ and $\widehat{\mathbf{x}} = \boldsymbol{\mu} + \sigma\mathbf{z}$. In this case, the true distributions can be determined and they can be expressed in terms of delta functions:

$$\begin{aligned} p(\mathbf{x}|\mathbf{z}) &= \delta(\mathbf{x} - (\sigma\mathbf{z} + \boldsymbol{\mu})), \\ p(\mathbf{z}|\mathbf{x}) &= \delta(\mathbf{z} - (\mathbf{x} - \boldsymbol{\mu})/\sigma). \end{aligned}$$

Suppose now that we do not know $p(\mathbf{x})$ so we need to build an encoder and a decoder to estimate $\mathbf{z}$ and $\widehat{\mathbf{x}}$. Let's first define the encoder. Our encoder in this example takes the input $\mathbf{x}$ and generates a pair of parameters $\widehat{\boldsymbol{\mu}}(\mathbf{x})$ and $\widehat{\sigma}(\mathbf{x})^2$, denoting the parameters of a Gaussian. Then, we define $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ as a Gaussian:

$$\begin{aligned} (\widehat{\boldsymbol{\mu}}(\mathbf{x}),\ \widehat{\sigma}(\mathbf{x})^2) &= \text{Encoder}_{\boldsymbol{\phi}}(\mathbf{x}), \\ q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\mathbf{z} \mid \widehat{\boldsymbol{\mu}}(\mathbf{x}),\ \widehat{\sigma}(\mathbf{x})^2\mathbf{I}). \end{aligned}$$

For the purpose of discussion, we assume that $\widehat{\boldsymbol{\mu}}$ is an affine function of $\mathbf{x}$ such that $\widehat{\boldsymbol{\mu}}(\mathbf{x}) = a\mathbf{x} + \mathbf{b}$ for some parameters a and $\mathbf{b}$. Similarly, we assume that $\widehat{\sigma}(\mathbf{x})^2 = t^2$ for some scalar t. This will give us

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z} \mid a\mathbf{x} + \mathbf{b}, t^2\mathbf{I}).$$

For the decoder, we deploy a similar structure by considering

$$\begin{aligned} (\widetilde{\boldsymbol{\mu}}(\mathbf{z}),\ \widetilde{\sigma}(\mathbf{z})^2) &= \text{Decoder}_{\boldsymbol{\theta}}(\mathbf{z}), \\ p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(\mathbf{x} \mid \widetilde{\boldsymbol{\mu}}(\mathbf{z}),\ \widetilde{\sigma}(\mathbf{z})^2\mathbf{I}). \end{aligned}$$

Again, for the purpose of discussion, we assume that $\widetilde{\boldsymbol{\mu}}$ is affine so that $\widetilde{\boldsymbol{\mu}}(\mathbf{z}) = c\mathbf{z} + \mathbf{v}$ for some parameters c and $\mathbf{v}$ and $\widetilde{\sigma}(\mathbf{z})^2 = s^2$ for some scalar s. Therefore, $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ takes the form of:

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid c\mathbf{x} + \mathbf{v}, s^2\mathbf{I}).$$

We will discuss how to determine the parameters later.

1.2 Evidence Lower Bound

How do we use these two proxy distributions to achieve our goal of determining the encoder and the decoder? If we treat $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ as optimization variables, then we need an objective function (or the loss function) so that we can optimize $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ through training samples. The loss function we use here is called the Evidence Lower BOund (ELBO) [23]:

Definition 1.3 (Evidence Lower Bound). The Evidence Lower Bound is defined as

$$\text{ELBO}(\mathbf{x}) \overset{\text{def}}{=} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right]. \tag{1.4}$$

You are certainly puzzled how on the Earth people can come up with this loss function!? Let's see what ELBO means and how it is derived.

In a nutshell, ELBO is a **lower bound** for the prior distribution $\log p(\mathbf{x})$ because we can show that

$$\begin{aligned}
\log p(\mathbf{x}) &= \text{some magical steps to be derived} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right] + \mathbb{D}_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}|\mathbf{x})) \qquad (1.5) \\
&\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right] \\
&\overset{\text{def}}{=} \text{ELBO}(\mathbf{x}),
\end{aligned}$$

where the inequality follows from the fact that the KL divergence is always non-negative. Therefore, ELBO is a valid lower bound for $\log p(\mathbf{x})$. Since we never have access to $\log p(\mathbf{x})$, if we somehow have access to ELBO and if ELBO is a good lower bound, then we can effectively maximize ELBO to achieve the goal of maximizing $\log p(\mathbf{x})$ which is the gold standard. Now, the question is how good the lower bound is. As you can see from the equation and also Figure 1.4, the inequality will become an equality when our proxy $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ can match

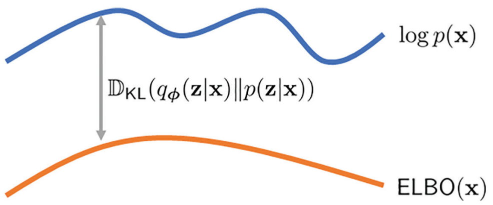

$\log p(\mathbf{x})$
$\mathbb{D}_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x}))$
$\text{ELBO}(\mathbf{x})$

Figure 1.4: Visualization of $\log p(\mathbf{x})$ and ELBO. The gap between the two is determined by the KL divergence $\mathbb{D}_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x}))$.

the true distribution $p(\mathbf{z}|\mathbf{x})$ exactly. So, part of the game is to ensure $q_\phi(\mathbf{z}|\mathbf{x})$ is close to $p(\mathbf{z}|\mathbf{x})$.

The derivation of Equation (1.5) is as follows.

Theorem 1.1 (Decomposition of Log-Likelihood)**.** The log likelihood $\log p(\mathbf{x})$ can be decomposed as

$$\log p(\mathbf{x}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\overset{\text{def}}{=}\text{ELBO}(\mathbf{x})} + \mathbb{D}_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})). \quad (1.6)$$

Proof. The trick is to use our magical proxy $q_\phi(\mathbf{z}|\mathbf{x})$ to poke around $p(\mathbf{x})$ and derive the bound.

$$\begin{aligned}
\log p(\mathbf{x}) &= \log p(\mathbf{x}) \times \underbrace{\int q_\phi(\mathbf{z}|\mathbf{x})d\mathbf{z}}_{=1} \quad \text{multiply 1} \\
&= \int \underbrace{\log p(\mathbf{x})}_{\text{some constant wrt } \mathbf{z}} \times \underbrace{q_\phi(\mathbf{z}|\mathbf{x})}_{\text{distribution in } \mathbf{z}} d\mathbf{z} \\
&\qquad\qquad \text{move } \log p(\mathbf{x}) \text{ into integral} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x})], \qquad (1.7)
\end{aligned}$$

where the last equality is the fact that $\int a \times p_Z(z)dz = \mathbb{E}[a] = a$ for any random variable Z and a scalar a.

See, we have already got $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\cdot]$. Just a few more steps. Let's use Bayes theorem which states that $p(\mathbf{x},\mathbf{z}) = p(\mathbf{z}|\mathbf{x})p(\mathbf{x})$:

$$\begin{aligned}
\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x})] &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}\right] && \text{(Bayes Theorem)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} \times \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
& && \text{(Multiply and divide } q_\phi(\mathbf{z}|\mathbf{x})\text{)} \\
&= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\text{ELBO}} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right]}_{\mathbb{D}_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x}))},
\end{aligned} \tag{1.8}$$

where we recognize that the first term is exactly ELBO, whereas the second term is exactly the KL divergence. Comparing Equation (1.8) with Equation (1.5), we complete the proof. □

Example 1.4. Using the previous example, we can minimize the gap between $\log p(\mathbf{x})$ and $\text{ELBO}(\mathbf{x})$ if we *knew* $p(\mathbf{z}|\mathbf{x})$. To see that, we note that $\log p(\mathbf{x})$ is

$$\log p(\mathbf{x}) = \text{ELBO}(\mathbf{x}) + \mathbb{D}_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})) \geq \text{ELBO}(\mathbf{x}).$$

The equality holds if and only if the KL-divergence term is zero. For the KL divergence to be zero, it is necessary that $q_\phi(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x})$. However, since $p(\mathbf{z}|\mathbf{x})$ is a delta function, the only possibility is to have

$$\begin{aligned}
q_\phi(\mathbf{z}|\mathbf{x}) &= \mathcal{N}\left(\mathbf{z} \,\middle|\, \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma},\ 0\right) \\
&= \delta\left(\mathbf{z} - \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}\right),
\end{aligned} \tag{1.9}$$

i.e., we set the standard deviation to be $t = 0$. To determine $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, we need some additional steps to simplify ELBO.

We now have ELBO. But this ELBO is still not too useful because it involves $p(\mathbf{x},\mathbf{z})$, something we have no access to. So, we need to do a little more work.

Theorem 1.2 (Interpretation of ELBO)**.** ELBO can be decomposed as

$$\text{ELBO}(\mathbf{x}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log \overbrace{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})}^{\text{a Gaussian}}]}_{\text{how good your decoder is}} - \underbrace{\mathbb{D}_{\text{KL}}\Big(\overbrace{q_\phi(\mathbf{z}|\mathbf{x})}^{\text{a Gaussian}} \,\|\, \overbrace{p(\mathbf{z})}^{\text{a Gaussian}}\Big)}_{\text{how good your encoder is}}. \tag{1.10}$$

Proof. Let's take a closer look at ELBO

$$\begin{aligned}
\text{ELBO}(\mathbf{x}) &\overset{\text{def}}{=} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] && \text{(definition)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \quad p(\mathbf{x},\mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}) \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z})\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
& && \text{(split expectation)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - \mathbb{D}_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})), \\
& && \text{(definition of KL)}
\end{aligned}$$

where we replaced the inaccessible $p(\mathbf{x}|\mathbf{z})$ by its proxy $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. □

This is a *beautiful* result. We just showed something very easy to understand. Let's look at the two terms in Equation (1.10):

- **Reconstruction**. The first term is about the *decoder*. We want the decoder to produce a good image $\mathbf{x}$ if we feed a latent $\mathbf{z}$ into the decoder (of course!!). So, we want to *maximize* $\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. It is similar to maximum likelihood where we want to find the model parameter to maximize the likelihood of observing the image. The expectation here is taken with respect to the samples $\mathbf{z}$ (conditioned on $\mathbf{x}$). This shouldn't be a surprise because the samples $\mathbf{z}$ are used to assess the quality of the decoder. It cannot be an arbitrary noise vector but a meaningful latent vector. So, $\mathbf{z}$ needs to be sampled from $q_\phi(\mathbf{z}|\mathbf{x})$.

- **Prior Matching**. The second term is the KL divergence for the *encoder*. We want the encoder to turn $\mathbf{x}$ into a latent vector $\mathbf{z}$ such that the latent vector will follow our choice of distribution, e.g., $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$. To be slightly more general, we write $p(\mathbf{z})$ as the target distribution. Because the KL divergence is a distance (which increases when the two distributions become more dissimilar), we need to put a negative sign in front so that it increases when the two distributions become more similar.

Example 1.5. Following up on the previous example, we continue to assume that we *knew* $p(\mathbf{z}|\mathbf{x})$. Then the reconstruction term in ELBO will give us

$$\begin{aligned}
\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log \mathcal{N}(\mathbf{x} \mid c\mathbf{z} + \mathbf{v},\ s^2\mathbf{I})] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[-\frac{1}{2}\log 2\pi - \log s - \frac{\|\mathbf{x} - (c\mathbf{z} + \mathbf{v})\|^2}{2s^2}\right] \\
&= -\frac{1}{2}\log 2\pi - \log s - \frac{c^2}{2s^2}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\left\|\mathbf{z} - \frac{\mathbf{x} - \mathbf{v}}{c}\right\|^2\right] \\
&= -\frac{1}{2}\log 2\pi - \log s - \frac{c^2}{2s^2}\mathbb{E}_{\delta(\mathbf{z} - \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma})}\left[\left\|\mathbf{z} - \frac{\mathbf{x} - \mathbf{v}}{c}\right\|^2\right] \\
&= -\frac{1}{2}\log 2\pi - \log s - \frac{c^2}{2s^2}\left[\left\|\frac{\mathbf{x} - \boldsymbol{\mu}}{\sigma} - \frac{\mathbf{x} - \mathbf{v}}{c}\right\|^2\right] \\
&\leq -\frac{1}{2}\log 2\pi - \log s,
\end{aligned}$$

where the upper bound is tight if and only if the norm-square term is zero, which holds when $\mathbf{v} = \boldsymbol{\mu}$ and $c = \sigma$. For the remaining terms, it is clear that $-\log s$ is a monotonically decreasing function in s with $-\log s \to \infty$ as $s \to 0$. Therefore, when $\mathbf{v} = \boldsymbol{\mu}$ and $c = \sigma$, it follows that $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$ is maximized when $s = 0$. This implies that

$$\begin{aligned}
p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(\mathbf{x} \mid \sigma\mathbf{z} + \boldsymbol{\mu},\ 0) \\
&= \delta(\mathbf{x} - (\sigma\mathbf{z} + \boldsymbol{\mu})). \qquad (1.11)
\end{aligned}$$

Limitation of ELBO. ELBO is practically useful, but it is *not* the same as the true likelihood $\log p(\mathbf{x})$. As we mentioned, ELBO is exactly equal to $\log p(\mathbf{x})$ if and only if $\mathbb{D}_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})) = 0$ which happens when $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x})$. In the following example, we will show a case where the $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ obtained from maximizing ELBO is not the same as $p(\mathbf{z}|\mathbf{x})$.

Example 1.6 (Limitation of ELBO)**.** In the previous example, if we have no idea about $p(\mathbf{z}|\mathbf{x})$, we need to train the VAE by maximizing ELBO. However, since ELBO is only a lower bound of the true distribution $\log p(\mathbf{x})$, maximizing ELBO will not return us the delta functions as we hope. Instead, we will obtain something that is quite meaningful but not exactly the delta functions.

For simplicity, let's consider the distributions that will return us unbiased estimates of the mean but with unknown variances:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z} \,\Big|\, \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}, t^2\mathbf{I}\right),$$
$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}\left(\mathbf{x} \,\Big|\, \sigma\mathbf{z}+\boldsymbol{\mu}, s^2\mathbf{I}\right).$$

This is partially "cheating" because in theory we should not assume anything about the estimates of the means. But from an intuitive angle, since $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ are proxies to $p(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$, they must resemble some properties of the delta functions. The closest choice is to define $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ as Gaussians with means consistent with those of the two delta functions. The variances are unknown, and they are the subject of interest in this example.

Our focus here is to maximize ELBO which consists of the prior matching term and the reconstruction term. For the prior matching error, we want to minimize the KL-divergence:

$$\mathbb{D}_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) = \mathbb{D}_{\text{KL}}\left(\mathcal{N}\left(\mathbf{z} \,\Big|\, \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}, t^2\mathbf{I}\right) \Big\| \,\mathcal{N}(\mathbf{z} \mid 0, \mathbf{I})\right).$$

The KL-divergence of two multivariate Gaussians $\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ and $\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ has a closed form expression which can be found in Wikipedia:

$$\begin{aligned}&\mathbb{D}_{\text{KL}}(\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)\|\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1))\\ &\quad= \frac{1}{2}\left(\text{Tr}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0) - d + (\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0) + \log\frac{\det\boldsymbol{\Sigma}_1}{\det\boldsymbol{\Sigma}_0}\right).\end{aligned}$$

Using this result (and with some algebra), we can show that

$$\mathbb{D}_{\text{KL}}\left(\mathcal{N}\left(\mathbf{z} \,\middle|\, \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}, t^2\mathbf{I}\right) \,\middle\|\, \mathcal{N}(\mathbf{z} \mid 0, \mathbf{I})\right) = \frac{1}{2}\left[t^2 d - d + \left\|\frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}\right\|^2 - 2\,d\log t\right],$$

where d is the dimension of $\mathbf{x}$ and $\mathbf{z}$. To minimize the KL-divergence, we take derivative with respect to t and show that

$$\frac{\partial}{\partial t}\left\{\frac{1}{2}\left[t^2 d - d + \left\|\frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}\right\|^2 - 2\log t\right]\right\} = t \cdot d - \frac{d}{t}.$$

Setting this to zero will give us $t = \frac{1}{\sqrt{d}}$. Therefore, we can show that

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z} \,\middle|\, \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}, \mathbf{I}\right).$$

For the reconstruction term, we can show that

$$\begin{aligned}
\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{1}{(\sqrt{2\pi s^2})^d}\exp\left\{-\frac{\|\mathbf{x}-(\sigma\mathbf{z}+\boldsymbol{\mu})\|^2}{2s^2}\right\}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[-\frac{d}{2}\log 2\pi - d\log s - \frac{\|\mathbf{x}-(\sigma\mathbf{z}+\boldsymbol{\mu})\|^2}{2s^2}\right] \\
&= -\frac{d}{2}\log 2\pi - d\log s - \frac{\sigma^2}{2s^2}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\left\|\mathbf{z} - \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}\right\|^2\right] \\
&= -\frac{d}{2}\log 2\pi - d\log s - \frac{\sigma^2}{2s^2} \\
&\quad \times \text{Trace}\left\{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\left(\mathbf{z} - \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}\right)\left(\mathbf{z} - \frac{\mathbf{x}-\boldsymbol{\mu}}{\sigma}\right)^T\right]\right\} \\
&= -\frac{d}{2}\log 2\pi - d\log s - \frac{\sigma^2}{2s^2}\cdot d,
\end{aligned}$$

because the covariance of $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is $\mathbf{I}$ and so the trace will give us 1. Taking derivatives with respect to s will give us

$$\frac{d}{ds}\left\{-\frac{d}{2}\log 2\pi - d\log s - \frac{\sigma^2}{2s^2}\right\} = -\frac{d}{s} + \frac{d\sigma^2}{s^3} = 0.$$

Equating this to zero will give us $s = \sigma$. Therefore,

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}\left(\mathbf{x} \mid \sigma\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I}\right).$$

As we can see in this example and the previous example, while the ideal distributions are delta functions, the proxy distributions we obtain have a finite variance. This finite variance adds additional randomness to the samples generated by the VAE. There is nothing wrong with this VAE—we do it correctly by maximizing ELBO. It is just that maximizing the ELBO is not the same as maximizing $\log p(\mathbf{x})$.

1.3 Optimization in VAE

In the previous two subsections we introduced the building blocks of VAE and ELBO. The goal of this subsection is to discuss how to train a VAE and how to do inference.

VAE is a model that aims to approximate the true distribution $p(\mathbf{x})$ so that we can draw samples. A VAE is parameterized by $(\boldsymbol{\phi}, \boldsymbol{\theta})$. Therefore, training a VAE is equivalent to solving an optimization problem that encapsulates the essence of $p(\mathbf{x})$ while being tractable. However, since $p(\mathbf{x})$ is not accessible, the natural alternative is to optimize the ELBO which is the lower bound of $\log p(\mathbf{x})$. That means, the learning goal of VAE is to solve the following problem.

Definition 1.4. The optimization objective of VAE is to maximize the ELBO:

$$(\boldsymbol{\phi}, \boldsymbol{\theta}) = \underset{\boldsymbol{\phi}, \boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\mathbf{x} \in \mathcal{X}} \text{ELBO}(\mathbf{x}), \tag{1.12}$$

where $\mathcal{X} = \{\mathbf{x}^{(\ell)} \mid \ell = 1, \dots, L\}$ is the training dataset.

Intractability of ELBO's Gradient. The challenge associated with the above optimization is that the gradient of ELBO with respect to $(\boldsymbol{\phi}, \boldsymbol{\theta})$ is intractable. Since the majority of today's neural network optimizers use first order methods and backpropagate the gradient to update the network weights, an intractable gradient will pose difficulties in training the VAE.

Let's elaborate more about the intractability of the gradient. We first substitute Definition 1.3 into the above objective function. The gradient of ELBO is:[1]

$$\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \ \text{ELBO}(\mathbf{x}) = \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \left\{ \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right] \right\}$$
$$= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \Big\{ \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \Big[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big] \Big\}. \quad (1.13)$$

The gradient contains two parameters. Let's first look at $\boldsymbol{\theta}$. We can show that

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \ \text{ELBO}(\mathbf{x}) &= \nabla_{\boldsymbol{\theta}} \Big\{ \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \Big[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big] \Big\} \\
&= \nabla_{\boldsymbol{\theta}} \left\{ \int \Big[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big] \cdot q_{\phi}(\mathbf{z}|\mathbf{x}) d\mathbf{z} \right\} \\
&= \int \nabla_{\boldsymbol{\theta}} \Big\{ \log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big\} \cdot q_{\phi}(\mathbf{z}|\mathbf{x}) \ d\mathbf{z} \\
&= \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \Big[\nabla_{\boldsymbol{\theta}} \Big\{ \log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big\} \Big] \\
&= \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \Big[\nabla_{\boldsymbol{\theta}} \Big\{ \log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) \Big\} \Big] \\
&\approx \frac{1}{L} \sum_{\ell=1}^{L} \nabla_{\boldsymbol{\theta}} \Big\{ \log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}^{(\ell)}) \Big\}, \qquad \text{where} \quad \mathbf{z}^{(\ell)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}),
\end{aligned} \quad (1.14)$$

where the last equality is the Monte Carlo approximation of the expectation.

In the above equation, if $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$ is realized by a computable model such as a neural network, then its gradient $\nabla_{\boldsymbol{\theta}}\{\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})\}$ can be computed via automatic differentiation. Thus, the maximization can be achieved by backpropagating the gradient.

The gradient with respect to $\boldsymbol{\phi}$ is more difficult. We can show that

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \ \text{ELBO}(\mathbf{x}) &= \nabla_{\boldsymbol{\phi}} \Big\{ \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \Big[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big] \Big\} \\
&= \nabla_{\boldsymbol{\phi}} \left\{ \int \Big[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big] \cdot q_{\phi}(\mathbf{z}|\mathbf{x}) d\mathbf{z} \right\} \\
&= \int \nabla_{\boldsymbol{\phi}} \Big\{ [\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})] \cdot q_{\phi}(\mathbf{z}|\mathbf{x}) \Big\} \ d\mathbf{z}
\end{aligned}$$

[1]The original definition of ELBO uses the true joint distribution $p(\mathbf{x},\mathbf{z})$. In practice, since $p(\mathbf{x},\mathbf{z})$ is not accessible, we replace it with its proxy $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$ which is a computable distribution.

$$\begin{aligned}
&\neq \int \nabla_{\boldsymbol{\phi}} \Big\{ \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big\} \cdot q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \, d\mathbf{z} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \Big[\nabla_{\boldsymbol{\phi}} \Big\{ \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big\} \Big] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \Big[\nabla_{\boldsymbol{\phi}} \Big\{ - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big\} \Big] \\
&\approx \frac{1}{L} \sum_{\ell=1}^{L} \nabla_{\boldsymbol{\phi}} \Big\{ - \log q_{\boldsymbol{\phi}}(\mathbf{z}^{(\ell)}|\mathbf{x}) \Big\}, \quad \text{where } \mathbf{z}^{(\ell)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}).
\end{aligned} \tag{1.15}$$

As we can see, even though we *wish* to maintain a similar structure as we did for $\boldsymbol{\theta}$, the expectation and the gradient operators in the above derivations cannot be switched. This forbids us from doing any backpropagation of the gradient to maximize ELBO.

Reparameterization Trick. The intractability of ELBO's gradient is inherited from the fact that we need to draw samples $\mathbf{z}$ from a distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ which itself is a function of $\boldsymbol{\phi}$. As noted by Kingma and Welling [24], for continuous latent variables, it is possible to compute an unbiased estimate of $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}$ ELBO$(\mathbf{x})$ so that we can approximately calculate the gradient and hence maximize ELBO. The idea is employ an technique known as the *reparameterization trick* [24].

Recall that the latent variable $\mathbf{z}$ is a sample drawn from the distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. The idea of reparameterization trick is to express $\mathbf{z}$ as some differentiable and invertible transformation of another random variable $\boldsymbol{\epsilon}$ whose distribution is independent of $\mathbf{x}$ and $\boldsymbol{\phi}$. That is, we define a differentiable and invertible function $\mathbf{g}$ such that

$$\mathbf{z} = \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}), \tag{1.16}$$

for some random variable $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. To make our discussions easier, we pose an additional requirement that

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \cdot \left| \det \left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}} \right) \right| = p(\boldsymbol{\epsilon}), \tag{1.17}$$

where $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}}$ is the Jacobian, and $\det(\cdot)$ is the matrix determinant. This requirement is related to change of variables in multivariate calculus. The following example will make it clear.

Example 1.7. Suppose $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \overset{\text{def}}{=} \mathcal{N}(\mathbf{z} \mid \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$. We can define

$$\mathbf{z} = \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \overset{\text{def}}{=} \boldsymbol{\epsilon} \odot \boldsymbol{\sigma} + \boldsymbol{\mu}, \tag{1.18}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ and "$\odot$" means elementwise multiplication. The parameter $\boldsymbol{\phi}$ is $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$. For this choice of the distribution, we can show that by letting $\boldsymbol{\epsilon} = \frac{\mathbf{z}-\boldsymbol{\mu}}{\boldsymbol{\sigma}}$:

$$\begin{aligned} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \cdot \left|\det\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}}\right)\right| &= \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left\{-\frac{(z_i - \mu_i)^2}{2\sigma_i^2}\right\} \cdot \prod_{i=1}^{d} \sigma_i \\ &= \frac{1}{(\sqrt{2\pi})^d} \exp\left\{-\frac{\|\boldsymbol{\epsilon}\|^2}{2}\right\} = \mathcal{N}(0, \mathbf{I}) = p(\boldsymbol{\epsilon}). \end{aligned}$$

With this re-parameterization of $\mathbf{z}$ by expressing it in terms of $\boldsymbol{\epsilon}$, we can look at $\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})]$ for some general function $f(\mathbf{z})$. (Later we will consider $f(\mathbf{z}) = -\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.) For notational simplicity, we write $\mathbf{g}(\boldsymbol{\epsilon})$ instead of $\mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$ although we understand that $\mathbf{g}$ has three inputs. By change of variables, we can show that

$$\begin{aligned} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] &= \int f(\mathbf{z}) \cdot q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\, d\mathbf{z} \\ &= \int f(\mathbf{g}(\boldsymbol{\epsilon})) \cdot q_{\boldsymbol{\phi}}(\mathbf{g}(\boldsymbol{\epsilon})|\mathbf{x})\, d\mathbf{g}(\boldsymbol{\epsilon}), \quad \text{let} \quad \mathbf{z} = \mathbf{g}(\boldsymbol{\epsilon}) \\ &= \int f\Big(\mathbf{g}(\boldsymbol{\epsilon})\Big) \cdot q_{\boldsymbol{\phi}}(\mathbf{g}(\boldsymbol{\epsilon})|\mathbf{x}) \cdot \left|\det\left(\frac{\partial \mathbf{g}(\boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}}\right)\right| \, d\boldsymbol{\epsilon} \\ &\qquad \text{(Jacobian due to change of variable)} \\ &= \int f(\mathbf{z}) \cdot p(\boldsymbol{\epsilon})\, d\boldsymbol{\epsilon} \qquad \text{(use Equation (1.17))} \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(\mathbf{z})\right]. \end{aligned} \tag{1.19}$$

So, if we want to take the gradient with respect to $\boldsymbol{\phi}$, we can show that

$$\begin{aligned} \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(\mathbf{z})\right] &= \nabla_{\boldsymbol{\phi}} \left\{\int f(\mathbf{z}) \cdot p(\boldsymbol{\epsilon})\, d\boldsymbol{\epsilon}\right\} \\ &= \int \nabla_{\boldsymbol{\phi}} \left\{f(\mathbf{z}) \cdot p(\boldsymbol{\epsilon})\right\}\, d\boldsymbol{\epsilon} \\ &= \int \left\{\nabla_{\boldsymbol{\phi}} f(\mathbf{z})\right\} \cdot p(\boldsymbol{\epsilon})\, d\boldsymbol{\epsilon} \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\phi}} f(\mathbf{z})\right], \end{aligned} \tag{1.20}$$

which can be approximated by Monte Carlo. Substituting $f(\mathbf{z}) = -\log q_\phi(\mathbf{z}|\mathbf{x})$, we can show that

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[-\log q_\phi(\mathbf{z}|\mathbf{x})] &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[-\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&\approx -\frac{1}{L}\sum_{\ell=1}^{L} \nabla_\phi \log q_\phi(\mathbf{z}^{(\ell)}|\mathbf{x}), \\
&\qquad\qquad \text{where} \quad \mathbf{z}^{(\ell)} = \mathbf{g}(\boldsymbol{\epsilon}^{(\ell)}, \boldsymbol{\phi}, \mathbf{x}) \\
&= -\frac{1}{L}\sum_{\ell=1}^{L} \nabla_\phi \left[\log p(\boldsymbol{\epsilon}^{(\ell)}) - \log\left|\det \frac{\partial \mathbf{z}^{(\ell)}}{\partial \boldsymbol{\epsilon}^{(\ell)}}\right|\right] \\
&= \frac{1}{L}\sum_{\ell=1}^{L} \nabla_\phi \left[\log\left|\det \frac{\partial \mathbf{z}^{(\ell)}}{\partial \boldsymbol{\epsilon}^{(\ell)}}\right|\right].
\end{aligned}$$

So, as long as the determinant is differentiable with respect to $\boldsymbol{\phi}$, the Monte Carlo approximation can be numerically computed.

Example 1.8. Suppose that the parameters and the distribution q_ϕ are defined as follows:

$$\begin{aligned}
(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) &= \text{EncoderNetwork}_\phi(\mathbf{x}) \\
q_\phi(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\mathbf{z} \mid \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)).
\end{aligned}$$

We can define $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, with $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$. Then, we can show that

$$\begin{aligned}
\log\left|\det \frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}}\right| &= \log\left|\det\left(\frac{\partial(\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}}\right)\right| \\
&= \log\left|\det(\text{diag}\,\{\boldsymbol{\sigma}\})\right| \\
&= \log \prod_{i=1}^{d} \sigma_i = \sum_{i=1}^{d} \log \sigma_i.
\end{aligned}$$

Therefore, we can show that

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[-\log q_\phi(\mathbf{z}|\mathbf{x})] &\approx \frac{1}{L}\sum_{\ell=1}^{L} \nabla_\phi \left[\log\left|\det \frac{\partial \mathbf{z}^{(\ell)}}{\partial \boldsymbol{\epsilon}^{(\ell)}}\right|\right] \\
&= \frac{1}{L}\sum_{\ell=1}^{L} \nabla_\phi \left[\sum_{i=1}^{d} \log \sigma_i\right]
\end{aligned}$$

$$
\begin{aligned}
&= \nabla_\phi \left[\sum_{i=1}^{d} \log \sigma_i \right] \\
&= \frac{1}{\boldsymbol{\sigma}} \odot \nabla_\phi \Big\{ \boldsymbol{\sigma}_\phi(\mathbf{x}) \Big\},
\end{aligned}
$$

where we emphasize that $\boldsymbol{\sigma}_\phi(\mathbf{x})$ is the output of the encoder which is a neural network.

As we can see in the above example, for some specific choices of the distributions (e.g., Gaussian), the gradient of ELBO can be significantly easier to derive.

VAE Encoder. After discussing the reparameterizing trick, we can now discuss the specific structure of the encoder in VAE. To make our discussions focused, we assume a relatively common choice of the encoder:

$$
\begin{aligned}
(\boldsymbol{\mu}, \sigma^2) &= \text{EncoderNetwork}_\phi(\mathbf{x}) \\
q_\phi(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\mathbf{z} \mid \boldsymbol{\mu}, \sigma^2 \mathbf{I}).
\end{aligned}
$$

The parameters $\boldsymbol{\mu}$ and σ are technically *neural networks* because they are the outputs of $\text{EncoderNetwork}_\phi(\cdot)$. Therefore, it will be helpful if we denote them as

$$
\begin{aligned}
\boldsymbol{\mu} &= \underbrace{\boldsymbol{\mu}_\phi}_{\text{neural network}}(\mathbf{x}), \\
\sigma^2 &= \underbrace{\sigma_\phi^2}_{\text{neural network}}(\mathbf{x}),
\end{aligned}
$$

Our notation is slightly more complicated because we want to emphasize that $\boldsymbol{\mu}$ is a function of $\mathbf{x}$; You give us an image $\mathbf{x}$, our job is to return you the parameters of the Gaussian (i.e., mean and variance). If you give us a different $\mathbf{x}$, then the parameters of the Gaussian should also be different. The parameter $\boldsymbol{\phi}$ specifies that $\boldsymbol{\mu}$ is controlled (or parameterized) by $\boldsymbol{\phi}$.

Suppose that we are given the ℓ-th training sample $\mathbf{x}^{(\ell)}$. From this $\mathbf{x}^{(\ell)}$ we want to generate a latent variable $\mathbf{z}^{(\ell)}$ which is a sample from $q_\phi(\mathbf{z}|\mathbf{x})$. Because of the Gaussian structure, it is equivalent to say that

$$
\mathbf{z}^{(\ell)} \sim \mathcal{N}\Big(\mathbf{z} \,\Big|\, \boldsymbol{\mu}_\phi(\mathbf{x}^{(\ell)}), \quad \sigma_\phi^2(\mathbf{x}^{(\ell)})\mathbf{I}\Big). \tag{1.21}
$$

The interesting thing about this equation is that we use a neural network $\text{EncoderNetwork}_\phi(\cdot)$ to estimate the mean and variance of the Gaussian. Then, from this Gaussian we draw a sample $\mathbf{z}^{(\ell)}$, as illustrated in Figure 1.5.

A more convenient way of expressing Equation (1.21) is to realize that the sampling operation $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{I})$ can be done using the reparameterization trick.

Reparameterization Trick for High-dimensional Gaussian:

$$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{I}) \qquad \Longleftrightarrow \qquad \mathbf{z} = \boldsymbol{\mu} + \sigma\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}). \tag{1.22}$$

Using the reparameterization trick, Equation (1.21) can be written as

$$\mathbf{z}^{(\ell)} = \boldsymbol{\mu}_\phi(\mathbf{x}^{(\ell)}) + \sigma_\phi(\mathbf{x}^{(\ell)})\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}).$$

Proof. We will prove a general case for an arbitrary covariance matrix $\boldsymbol{\Sigma}$ instead of a diagonal matrix $\sigma^2\mathbf{I}$.

For any high-dimensional Gaussian $\mathbf{z} \sim \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the sampling process can be done via the transformation of white noise

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{\epsilon}, \tag{1.23}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$. The half matrix $\boldsymbol{\Sigma}^{\frac{1}{2}}$ can be obtained through eigen-decomposition or Cholesky factorization. If $\boldsymbol{\Sigma}$ has an eigen-decomposition $\boldsymbol{\Sigma} = \mathbf{U}\mathbf{S}\mathbf{U}^T$, then $\boldsymbol{\Sigma}^{\frac{1}{2}} = \mathbf{U}\mathbf{S}^{\frac{1}{2}}\mathbf{U}^T$. The square root of the eigenvalue matrix $\mathbf{S}$ is well-defined because $\boldsymbol{\Sigma}$ is a positive semi-definite matrix.

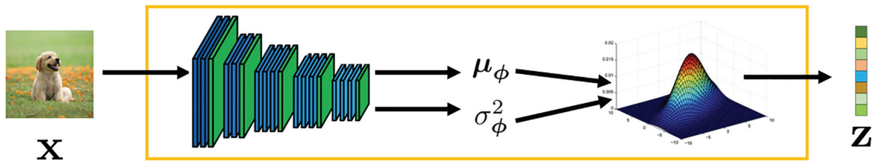


Figure 1.5: Implementation of a VAE encoder. We use a neural network to take the image $\mathbf{x}$ and estimate the mean $\boldsymbol{\mu}_\phi$ and variance σ^2_ϕ of the Gaussian distribution.

We can calculate the expectation and covariance of $\mathbf{x}$:

$$\mathbb{E}[\mathbf{z}] = \mathbb{E}[\boldsymbol{\mu} + \boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{\epsilon}] = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{\frac{1}{2}}\underbrace{\mathbb{E}[\boldsymbol{\epsilon}]}_{=0} = \boldsymbol{\mu},$$

$$\mathrm{Cov}(\mathbf{z}) = \mathbb{E}[(\mathbf{z}-\boldsymbol{\mu})(\mathbf{z}-\boldsymbol{\mu})^T] = \mathbb{E}\left[\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T(\boldsymbol{\Sigma}^{\frac{1}{2}})^T\right] = \boldsymbol{\Sigma}^{\frac{1}{2}}\underbrace{\mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T]}_{=\mathbf{I}}(\boldsymbol{\Sigma}^{\frac{1}{2}})^T = \boldsymbol{\Sigma}.$$

Therefore, for diagonal matrices $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$, the above is reduced to

$$\mathbf{z} = \boldsymbol{\mu} + \sigma\boldsymbol{\epsilon}, \qquad \text{where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}). \tag{1.24}$$

Given the VAE encoder structure and $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, we can go back to ELBO. Recall that ELBO consists of the prior matching term and the reconstruction term. The prior matching term is measured in terms of the KL divergence $\mathbb{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right)$. Let's evaluate this KL divergence.

To evaluate the KL divergence, we (re)use a result which we summarize below:

Theorem 1.3 (KL-Divergence of Two Gaussian)**.** The KL divergence for two d-dimensional Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ and $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ is

$$\begin{aligned}&\mathbb{D}_{\mathrm{KL}}\Big(\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \parallel \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\Big)\\ &\quad= \frac{1}{2}\Big(\mathrm{Tr}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0) - d + (\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)\\ &\quad quad + \log\frac{\det\boldsymbol{\Sigma}_1}{\det\boldsymbol{\Sigma}_0}\Big).\end{aligned} \tag{1.25}$$

Substituting our distributions by considering

$$\begin{aligned}\boldsymbol{\mu}_0 &= \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), & \boldsymbol{\Sigma}_0 &= \sigma_{\boldsymbol{\phi}}^2(\mathbf{x})\mathbf{I}\\ \boldsymbol{\mu}_1 &= 0, & \boldsymbol{\Sigma}_1 &= \mathbf{I},\end{aligned}$$

we can show that the KL divergence has an analytic expression

$$\mathbb{D}_{\mathrm{KL}}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\Big) = \frac{1}{2}\Big(\sigma_{\boldsymbol{\phi}}^2(\mathbf{x})d - d + \|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})\|^2 - 2d\log\sigma_{\boldsymbol{\phi}}(\mathbf{x})\Big), \tag{1.26}$$

where d is the dimension of the vector $\mathbf{z}$. The gradient of the KL-divergence with respect to $\boldsymbol{\phi}$ does not have a closed form but they can be calculated numerically:

$$\nabla_{\boldsymbol{\phi}} \mathbb{D}_{\text{KL}}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\Big) = \frac{1}{2}\nabla_{\boldsymbol{\phi}}\Big(\sigma_{\boldsymbol{\phi}}^2(\mathbf{x})d - d + \|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})\|^2 - 2d\log\sigma_{\boldsymbol{\phi}}(\mathbf{x})\Big). \tag{1.27}$$

The gradient with respect to $\boldsymbol{\theta}$ is zero because there is nothing dependent on $\boldsymbol{\theta}$.

VAE Decoder. The decoder is implemented through a neural network. For notation simplicity, let's define it as $\text{DecoderNetwork}_{\boldsymbol{\theta}}(\cdot)$ where $\boldsymbol{\theta}$ denotes the network parameters. The job of the decoder network is to take a latent variable $\mathbf{z}$ and generate an image $f_{\boldsymbol{\theta}}(\mathbf{z})$:

$$f_{\boldsymbol{\theta}}(\mathbf{z}) = \text{DecoderNetwork}_{\boldsymbol{\theta}}(\mathbf{z}). \tag{1.28}$$

The distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ can be defined as

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x} \mid f_{\boldsymbol{\theta}}(\mathbf{z}), \sigma_{\text{dec}}^2\mathbf{I}), \qquad \text{for some hyperparameter } \sigma_{\text{dec}}. \tag{1.29}$$

The interpretation of $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is that we estimate $f_{\boldsymbol{\theta}}(\mathbf{z})$ through a network and put it as the mean of the Gaussian. If we draw a sample $\mathbf{x}$ from $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, then by the reparameterization trick we can write the generated image $\widehat{\mathbf{x}}$ as

$$\widehat{\mathbf{x}} = f_{\boldsymbol{\theta}}(\mathbf{z}) + \sigma_{\text{dec}}\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}).$$

Moreover, if we take the log of the likelihood, we can show that

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x} \mid f_{\boldsymbol{\theta}}(\mathbf{z}), \sigma_{\text{dec}}^2\mathbf{I}) \\ &= \log \frac{1}{\sqrt{(2\pi\sigma_{\text{dec}}^2)^d}} \exp\left\{-\frac{\|\mathbf{x} - f_{\boldsymbol{\theta}}(\mathbf{z})\|^2}{2\sigma_{\text{dec}}^2}\right\} \\ &= -\frac{\|\mathbf{x} - f_{\boldsymbol{\theta}}(\mathbf{z})\|^2}{2\sigma_{\text{dec}}^2} - \underbrace{\log\sqrt{(2\pi\sigma_{\text{dec}}^2)^d}}_{\text{independent of } \boldsymbol{\theta} \text{ so we can drop it}}.\end{aligned} \tag{1.30}$$

Going back to ELBO, we want to compute $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$. If we straightly calculate the expectation, we will need to compute an integration

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] = \int \log\left[\mathcal{N}(\mathbf{x} \mid f_{\boldsymbol{\theta}}(\mathbf{z}), \sigma_{\text{dec}}^2\mathbf{I})\right] \cdot \mathcal{N}(\mathbf{z} \mid \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), \sigma_{\boldsymbol{\phi}}^2(\mathbf{x}))d\mathbf{z}$$

$$= -\int \frac{\|\mathbf{x} - f_{\boldsymbol{\theta}}(\mathbf{z})\|^2}{2\sigma_{\text{dec}}^2} \cdot \mathcal{N}\Big(\mathbf{z} \,\Big|\, \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), \sigma_{\boldsymbol{\phi}}^2(\mathbf{x})\Big) d\mathbf{z} + C,$$

where the constant C coming out of the log of the Gaussian can be dropped. By using the reparameterization trick, we write $\mathbf{z} = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}) + \sigma_{\boldsymbol{\phi}}(\mathbf{x})\boldsymbol{\epsilon}$ and substitute it into the above equation. This will give us[2]

$$\begin{aligned}
\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] &= -\int \frac{\|\mathbf{x} - f_{\boldsymbol{\theta}}(\mathbf{z})\|^2}{2\sigma_{\text{dec}}^2} \cdot \mathcal{N}\Big(\mathbf{z} \,\Big|\, \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), \sigma_{\boldsymbol{\phi}}^2(\mathbf{x})\Big) d\mathbf{z} \\
&\approx -\frac{1}{M}\sum_{m=1}^{M} \frac{\|\mathbf{x} - f_{\boldsymbol{\theta}}(\mathbf{z}^{(m)})\|^2}{2\sigma_{\text{dec}}^2} \qquad (1.31) \\
&= -\frac{1}{M}\sum_{m=1}^{M} \frac{\|\mathbf{x} - f_{\boldsymbol{\theta}}\Big(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}) + \sigma_{\boldsymbol{\phi}}(\mathbf{x})\boldsymbol{\epsilon}^{(m)}\Big)\|^2}{2\sigma_{\text{dec}}^2}.
\end{aligned}$$

The approximation above is due to Monte Carlo where the randomness is based on the sampling of the $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon} \mid 0, \mathbf{I})$. The index M specifies the number of Monte Carlo samples we want to use to approximate the expectation. Note that the input image $\mathbf{x}$ is fixed because $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$ is a function of $\mathbf{x}$.

The gradient of $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$ with respect to $\boldsymbol{\theta}$ is relatively easy to compute. Since only $f_{\boldsymbol{\theta}}$ depends on $\boldsymbol{\theta}$, we can do automatic differentiation. The gradient with respect to $\boldsymbol{\phi}$ is slightly harder, but it is still computable because we use chain rule and go into $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})$ and $\boldsymbol{\phi}_{\boldsymbol{\phi}}(\mathbf{x})$.

Inspecting Equation (1.31), we notice one interesting thing that the loss function is simply the ℓ_2 norm between the reconstructed image $f_{\boldsymbol{\theta}}(\mathbf{z})$ and the ground truth image $\mathbf{x}$. This means that if we have the generated image $f_{\boldsymbol{\theta}}(\mathbf{z})$, we can do a direct comparison with the ground truth $\mathbf{x}$ via the usual ℓ_2 loss as illustrated in Figure 1.6.

Training the VAE. Given a training dataset $\mathcal{X} = \{(\mathbf{x}^{(\ell)})\}_{\ell=1}^{L}$ of clean images, the training objective of VAE is to maximize the ELBO

$$\underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\text{argmax}} \sum_{\mathbf{x}\in\mathcal{X}} \text{ELBO}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}),$$

[2]The negative sign here is not a mistake. We want to *maximize* $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$, which is equivalent to *minimize* the negative of the ℓ_2 norm.

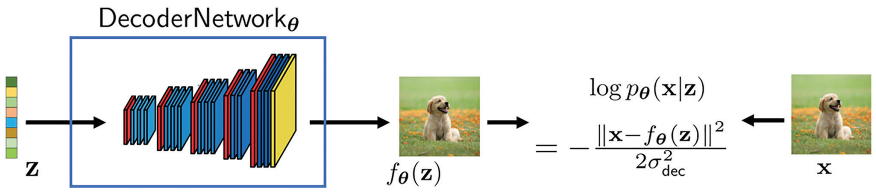

DecoderNetwork$_{\boldsymbol{\theta}}$
$\mathbf{z}$
$f_{\boldsymbol{\theta}}(\mathbf{z})$
$\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
$= -\frac{\|\mathbf{x}-f_{\boldsymbol{\theta}}(\mathbf{z})\|^2}{2\sigma_{\text{dec}}^2}$
$\mathbf{x}$

Figure 1.6: Implementation of a VAE decoder. We use a neural network to take the latent vector $\mathbf{z}$ and generate an image $f_{\boldsymbol{\theta}}(\mathbf{z})$. The log likelihood will give us a quadratic equation if we assume a Gaussian distribution.

where the summation is taken with respect to the entire training dataset. The individual ELBO is based on the sum of the terms we derived above

$$\text{ELBO}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - \mathbb{D}_{\text{KL}}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\Big). \tag{1.32}$$

Here, the reconstruction term is:

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] \approx -\frac{1}{M}\sum_{m=1}^{M} \frac{\|\mathbf{x} - f_{\boldsymbol{\theta}}\Big(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}) + \sigma_{\boldsymbol{\phi}}(\mathbf{x})\boldsymbol{\epsilon}^{(m)}\Big)\|^2}{2\sigma_{\text{dec}}^2}, \tag{1.33}$$

whereas the prior matching term is

$$\mathbb{D}_{\text{KL}}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\Big) = \frac{1}{2}\Big(\sigma_{\boldsymbol{\phi}}^2(\mathbf{x})d - d + \|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})\|^2 - 2d\log\sigma_{\boldsymbol{\phi}}(\mathbf{x})\Big). \tag{1.34}$$

To optimize for $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, we can run stochastic gradient descent. The gradients can be taken based on the tensor graphs of the neural networks. On computers, this is done automatically by the automatic differentiation.

Let's summarize these.

Theorem 1.4 (VAE Training)**.** To train an VAE, we need to solve the optimization problem

$$\underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\text{argmax}} \sum_{\mathbf{x}\in\mathcal{X}} \text{ELBO}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}),$$

where

$$\text{ELBO}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = -\frac{1}{M}\sum_{m=1}^{M} \frac{\|\mathbf{x} - f_{\boldsymbol{\theta}}\Big(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}) + \sigma_{\boldsymbol{\phi}}(\mathbf{x})\boldsymbol{\epsilon}^{(m)}\Big)\|^2}{2\sigma_{\text{dec}}^2} + \frac{1}{2}\Big(\sigma_{\boldsymbol{\phi}}^2(\mathbf{x})d - d + \|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})\|^2 - 2d\log\sigma_{\boldsymbol{\phi}}(\mathbf{x})\Big). \tag{1.35}$$

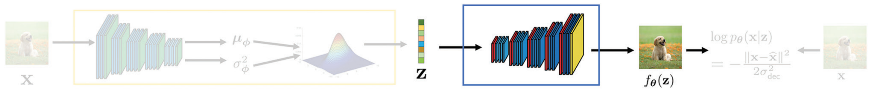

x
μ_ϕ
σ_ϕ^2
z
$f_\theta(\mathbf{z})$
$\log p_\theta(\mathbf{x}|\mathbf{z})$
$= -\frac{\|\mathbf{x}-\hat{\mathbf{x}}\|^2}{2\sigma_{\text{dec}}^2}$
x

Figure 1.7: Using VAE to generate image is as simple as sending a latent noise code $\mathbf{z}$ through the decoder.

VAE Inference. The inference of an VAE is relatively simple. Once the VAE is trained, we can drop the encoder and only keep the decoder, as shown in Figure 1.7. To generate a new image from the model, we pick a random latent vector $\mathbf{z} \in \mathbb{R}^d$. By sending this $\mathbf{z}$ through the decoder $f_{\boldsymbol{\theta}}$, we will be able to generate a new image $\widehat{\mathbf{x}} = f_{\boldsymbol{\theta}}(\mathbf{z})$.

1.4 Concluding Remark

For readers who are looking for additional references, we highly recommend the tutorial by Kingma and Welling [23] which is based on their original VAE paper [24]. A shorter tutorial by Doersch *et al.* [12] can also be helpful. [23] includes a long list of good papers including a paper by Rezende and Mohamed [32] on normalizing flow which was published around the same time as Kingma and Welling's VAE paper.

VAE has many linkages to the classical variational inference and graphical models [45]. VAE is also relevant to the generative adversarial networks (GAN) by Goodfellow *et al.* [15]. Kingma and Welling commented in [23] that VAE and GAN have complementary properties; while GAN produces better perceptual quality images, there is a weaker linkage with the data likelihood. VAE can meet the data likelihood criterion better but the samples are at times not perceptually as good.

References

[1] M. S. Albergo, N. M. Boffi, and E. Vanden-Eijnden, *Stochastic interpolants: A unifying framework for flows and diffusions*, 2023, URL: https://arxiv.org/abs/2303.08797.

[2] B. Anderson, "Reverse-time diffusion equation models," *Stochastic Process. Appl.*, vol. 12, no. 3, May pp. 313–326, 1982, URL: https://www.sciencedirect.com/science/article/pii/0304414982900515.

[3] K. Atkinson, W. Han, and D. Stewart, *Numerical Solution of Ordinary Differential Equations*. Wiley, 2009, URL: https://homepage.math.uiowa.edu/~atkinson/papers/NAODE_Book.pdf.

[4] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] C. A. Bouman and G. T. Buzzard, "Generative plug and play: Posterior sampling for inverse problems," in *2023 59th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1–7, 2023, URL: https://arxiv.org/abs/2306.07233.

[6] R. Brown, "A brief account of microscopical observations on the particles contained in the pollen of plants and the general existence of active molecules in organic and inorganic bodies," *Edinburgh New Philosophical Journal*, pp. 358–371, 1828.

[7] S. H. Chan, *Introduction to Probability for Data Science*. Michigan Publishing, 2021, URL: https://probability4datascience.com/.

[8] S. H. Chan, X. Wang, and O. Elgendy, "Plug-and-Play ADMM for image restoration: Fixed point convergence and applications," *IEEE Trans. Computational Imaging*, vol. 3, no. 5, pp. 84–98, 2017, URL: https://arxiv.org/abs/1605.01710.

[9] H. Chung and J. C. Ye, "Score-based diffusion models for accelerated mri," *Medical Image Analysis*, vol. 80, p. 102479, 2022, URL: https://arxiv.org/abs/2110.05243.

[10] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel, "The Helmholtz machine," *Neural Computation*, vol. 7, no. 5, pp.889–904, 1995.

[11] M. Delbracio and P. Milanfar, "Inversion by direct iteration: An alternative to denoising diffusion for image restoration," *Transactions on Machine Learning Research (TMLR)*, 2023, URL: https://openreview.net/forum?id=VmyFF5lL3F.

[12] C. Doersch, *Tutorial on variational autoencoders*, 2016, URL: https://arxiv.org/abs/1606.05908.

[13] L. Donati, *From Chapman-Kolmogorov equation to Master equation and Fokker-Planck equation*, URL: https://www.zib.de/userpage/donati/stochastics2023/03/lecture_notes/L03_dCKeq.pdf.

[14] A. Einstein, "On the movement of small particles suspended in stationary liquids required by the molecular-kinetic theory of heat," *Annalen der Physik*, pp. 549–560, 1905.

[15] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 27, 2014, URL: https://arxiv.org/abs/1406.2661.

[16] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, URL: https://arxiv.org/abs/2006.11239.

[17] J. Hu, B. Song, X. Xu, L. Shen, and J. A. Fessler, *Learning image priors through patch-based diffusion models for solving inverse problems*, 2024, URL: https://arxiv.org/abs/2406.02462.

[18] A. Hyvärinen, "Estimation of non-normalized statistical models by score matching," *Journal of Machine Learning Research (JMLR)*, vol. 6, no. 24, pp. 695–709, 2005, URL: https://jmlr.org/papers/volume6/hyvarinen05a/hyvarinen05a.pdf.

[19] Z. Kadkhodaie and E. P. Simoncelli, *Solving linear inverse problems using the prior implicit in a denoiser*, 2020, URL: https://arxiv.org/abs/2007.13640.

[20] T. Karras, M. Aittala, T. Aila, and S. Laine, "Elucidating the design space of diffusion-based generative models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022, URL: https://arxiv.org/abs/2206.00364.

[21] B. Kawar, M. Elad, S. Ermon, and J. Song, "Denoising diffusion restoration models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022, URL: https://arxiv.org/abs/2201.11793.

[22] D. P. Kingma, T. Salimans, B. Poole, and J. Ho, "Variational diffusion models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021, URL: https://arxiv.org/abs/2107.00630,

[23] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019, URL: https://arxiv.org/abs/1906.02691.

[24] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *International Conference on Learning Representations (ICLR)*, 2014, URL: https://openreview.net/forum?id=33X9fd2-9FyZd.

[25] A. Kolmogorov, *Foundations of the Theory of Probability.* Dover, 2018, The original version was published in 1933 in German. URL: https://dn790007.ca.archive.org/0/items/foundationsofthe00kolm/foundationsofthe00kolm.pdf.

[26] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022, URL: https://arxiv.org/abs/2206.00927.

[27] C. Luo, *Understanding diffusion models: A unified perspective*, 2022, URL: https://arxiv.org/abs/2208.11970.

[28] C. Meng, R. Rombach, R. Gao, D. Kingma, S. Ermon, J. Ho, and T. Salimans, "On distillation of guided diffusion models," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14297–14306, 2023, URL: https://arxiv.org/abs/2210.03142.

[29] G. Nagy, *MTH 235 differential equations*, 2024, URL: https://users.math.msu.edu/users/gnagy/teaching/ade.pdf.

[30] R. Pawula, "Generalizations and extensions of the Fokker-Planck-Kolmogorov equations," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 33–41, 1967.

[31] L. E. Reichl, *A Modern Course in Statistical Physics*, 2nd ed. John Wiley and Sons, Inc, 1998.

[32] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proceedings of International Conference on Machine Learning (ICML)*, pp. 1530–1538, 2015, URL: https://arxiv.org/abs/1505.05770.

[33] H. Risken, *The Fokker-Planck Equations: Methods of solutions and applications*, 2nd ed. Springer, 1989.

[34] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022, URL: https://arxiv.org/abs/2112.10752.

[35] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi, "Photorealistic text-to-image diffusion models with deep language understanding," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 36479–36494, 2022, URL: https://arxiv.org/abs/2205.11487.

[36] T. Salimans and J. Ho, "Progressive distillation for fast sampling of diffusion models," in *International Conference on Learning Representations (ICLR)*, 2022, URL: https://arxiv.org/abs/2202.00512.

[37] Y. Sanghvi, Y. Chi, and S. H. Chan, "Kernel diffusion: An alternate approach to blind deconvolution," in *European Conference on Computer Vision (ECCV)*, 2024, URL: https://arxiv.org/abs/2312.02319.

[38] M. von Smoluchowski, “Zur kinetischen theorie der brownschen molekularbewegung und der suspensionen,” *Annalen der Physik*, 1906, pp. 756–780.

[39] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep unsupervised learning using nonequilibrium thermodynamics,” in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 37, pp. 2256–2265, 2015, URL: https://arxiv.org/abs/1503.03585.

[40] J. Song, C. Meng, and S. Ermon, “Denoising diffusion implicit models,” in *International Conference on Learning Representations (ICLR)*, 2023, URL: https://openreview.net/forum?id=St1giarCHLP.

[41] Y. Song and S. Ermon, “Generative modeling by estimating gradients of the data distribution,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, URL: https://arxiv.org/abs/1907.05600.

[42] Y. Song and S. Ermon, “Improved techniques for training score-based generative models,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, URL: https://arxiv.org/abs/2006.09011.

[43] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-based generative modeling through stochastic differential equations,” in *International Conference on Learning Representations (ICLR)*, 2021, URL: https://openreview.net/forum?id=PxTIG12RRHS.

[44] P. Vincent, “A connection between score matching and denoising autoencoders,” *Neural Computation*, vol. 23, no. 7, pp. 1661–1674, 2011, URL: https://www.iro.umontreal.ca/~vincentp/Publications/smdae_techreport.pdf.

[45] M. J. Wainwright and M. I. Jordan, “Graphical models, exponential families, and variational inference,” *Foundations and Trends in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008, URL: https://cba.mit.edu/events/03.11.ASE/docs/Wainwright.1.pdf.

[46] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient Langevin dynamics," in *Proceedings of International Conference on Machine Learning (ICML)*, pp. 681–686, 2011, URL: https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf.

[47] Y. Zhu, K. Zhang, J. Liang, J. Cao, B. Wen, R. Timofte, and L. V. Gool, "Denoising diffusion models for plug-and-play image restoration," in *IEEE Conference on Computer Vision and Pattern Recognition Workhsop (CVPRW)*, pp. 1219–1229, 2023, URL: https://arxiv.org/abs/2306.07233.