Universitat de Barcelona

Advanced Mathematics Master's Thesis

Fundamental Principles of Binary Latent Diffusion

Author:
Àlex Pujol

Supervisors:
Dr. Carles Casacuberta
Dr. Sergio Escalera

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Advanced Mathematics

in the

Facultat de Matemàtiques i Informàtica

September 2, 2024

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Fundamental Principles of Binary Latent Diffusion

by Àlex Pujol

In this thesis we explore the fundamental principles of Binary Latent Diffusion Models (BLDM), a novel class of generative models that leverage probabilistic deep latent variable models and diffusion processes to approximate complex data distributions. The research delves into probability theory, generative models, and latent space representations, with a focus on Variational Autoencoders (VAE) that lead to Bernoulli Variational Autoencoders (BVAE). The study provides a comprehensive overview of the foundations of Diffusion Models, leading to the formal definition of Discrete Bernoulli Diffusion Models (DBDM) and its training objective. Both, BVAE and DBDM, are the building blocks of the BLDM. Additionally, a practical application is presented. This exploration highlights the mathematical formalization and implementation strategies for BLDMs, paving the way for future advancements in generative modeling.

Acknowledgements

I extend my sincere gratitude to my thesis supervisor, Carles Casacuberta and Sergio Escalera, for their invaluable guidance and support throughout the research process. I also thank Christos Kantas and Anders Skaarup for their insightful comments, and fruitful advices. Appreciation goes to Paula, Marco and Maren for enriching discussions and support, and to my friends and family for their continuous encouragement. This thesis reflects not only my effort but also the collaborative contributions of a supportive academic community, for which I am truly thankful.

Contents

1 Introduction and Motivation

Machine Learning (ML) is a field of computer science that is concerned with developing algorithms and models that can learn from data and make predictions or decisions based on that data, aiming to reproduce human like cognition capabilities. The subfield of ML that is concerned with the generation of creative outputs is known as Generative Modeling. Here, by *creative generation* we mean the ability to generate new data samples that are similar to a given dataset, but do not belong to it. Thus, a *generative model*, has the capability to create never-seen-before data samples that are plausible and realistic. This can be useful for a variety of tasks, such as generating new images, music, or text, or for building a synthetic dataset for training other machine learning models. Generative models are also used in unsupervised learning, where the goal is to learn the underlying structure of a dataset without any labels or annotations. For example, consider the task of generating new images of handwritten digits. One way to do this is to ask for group of people to draw a diversity of digits, and then train a generative model on this dataset. Once the process finishes, the model could then generate new images that are similar to the ones in the dataset. Furthermore, the model could be provided with a token representing a digit, conditioning it to generate images of a the specific digit.

In this thesis, we explore the field of Latent Diffusion Models (LDM), which are a class of generative models based on probabilistic Deep Latent Variable Models that take ideas from Variational Inference and Thermodynamics and which have been shown to be effective at approximating very complex data distributions, generating high-quality images, audio or video (Yang et al., 2023). In particular, we focus in a novel type of LDM called Binary Latent Diffusion Model (BLDM), which forces the latent spaces to be binary, and can be used to represent data in a more compact and efficient way (Wang et al., 2023). The goal of this research is to provide the mathematical formalization and the fundamental concepts behind such models. In the forthcoming chapter we introduce basic concepts of probability theory and generative models. Then we relate the concept of the latent space of a generative model with a dimensionality reduction process, introducing the concept of Autoencoders and Variational Autoencoders (VAEs). VAEs draw the correlation between the minimization of the divergence of the data distribution and the generative model, and the maximization of the likelihood function, by means of a Deep Latent Variable Model. In the third chapter we formalize the concept of Diffusion Models and introduce the LDM and BLDM. In the fourth chapter, we discuss the algorithm and implementation of the BLDM, and apply it to a novel practical example, that is text-to-motion generation. Finally, in the last chapter we draw conclusions and outline future steps for enhancing the proposed methods. The main references for this thesis are the papers Kingma and Welling, 2014, Sohl-Dickstein et al., 2015, Ho, Jain, and Abbeel, 2020 and Wang et al., 2023, as well as the books Bishop, 2006, Barber, 2012, Murphy, 2022 and Murphy, 2023.

1.1 Probabilistic Modelling

Probabilistic modeling is a fundamental approach in machine learning, particularly for generative models, where the goal is to model the underlying distribution of data. Generative models leverage probability theory to capture the complexities of data distributions and can generate new data points that resemble the original data. This chapter introduces the basic concepts of probability theory, which are essential for understanding how these models work. By building a strong foundation in probability, we can better grasp how generative models make predictions and generate data based on learned distributions. As we explore these concepts, we will see how probability provides the framework for modeling data distributions.

1.1.1 Probability Space

Probability theory is a branch of mathematics dedicated to studying random phenomena and modeling uncertainty. It provides a formal framework for reasoning under uncertainty and making predictions with incomplete information. Central to this framework is the concept of a *probability measure*, which assigns a numerical value to each event in a sample space.

Definition 1 (Probability Measure). *A **measure** on a set Ω is a function $\mu : \mathcal{F} \to [0, \infty]$ that assigns a nonnegative real number to each subset of Ω, where $\mathcal{F}$ is a σ-algebra over Ω. It satisfies the following properties:*

- ***Non-negativity:*** *$\mu(A) \geq 0$ for all $A \in \mathcal{F}$.*
- ***Measure 0:*** *$\mu(\emptyset) = 0$, where $\emptyset$ is the empty set.*
- ***Countable additivity:*** *If $A_1, A_2, \ldots \in \mathcal{F}$ are pairwise disjoint sets, then $\mu\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i)$.*

*If μ additionally satisfies the **Normalization** property, $\mu(\Omega) = 1$, then μ is a **probability measure**.*

Definition 2 (σ-algebra). *Let Ω be a set. A **σ-algebra** on Ω is a nonempty collection $\mathcal{F}$ of subsets of Ω for which:*

- *$\Omega \in \mathcal{F}$.*
- ***Closed under complements**: If $A \in \mathcal{F}$, then its complement $A^c \in \mathcal{F}$.*
- ***Closed under countable unions**: If $A_1, A_2, \ldots \in \mathcal{F}$, then their union $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$.*

A common measure is the *Lebesgue measure*, which assigns the length of an interval to the interval itself. Given an open set $A = \cup_k (a_k, b_k)$, containing a countable number of disjoint intervals, the Lebesgue measure of A is $\mu(A) = \sum_k (b_k - a_k)$.

Definition 3 (Probability Space). *A **probability space** is a triple $(\Omega, \mathcal{F}, P)$, where:*

- *$(\Omega, \mathcal{F})$ is a measurable space, meaning Ω is a set and $\mathcal{F}$ is a σ-algebra of subsets of Ω.*
- *P is a probability measure on $\mathcal{F}$.*

A probability space serves as a mathematical model of a random experiment. Here, Ω represents the sample space, which is the set of all possible outcomes of the experiment. The σ-algebra $\mathcal{F}$ consists of events that can be measured, while the probability measure P assigns a probability to each event in $\mathcal{F}$.

Note that the second property of σ-algebras implies that if we can compute the probability of an event $A \in \mathcal{F}$ occurring, we must also be able to compute the probability of its complement $A^c \in \mathcal{F}$ occurring. The third property ensures that if we can compute the probability of individual events occurring, we can also compute the probability of any countable combination of these events occurring.

Proposition 1 (Monotonicity of Probability)**.** *Let $A, B \in \mathcal{F}$ be events in a probability space $(\Omega, \mathcal{F}, P)$. If $A \subset B$, then $P(A) \leq P(B)$.*

Proof. Since $A \subset B$, we have $B = A \cup (B \setminus A)$. Since A and $B \setminus A$ are disjoint, $P(B) = P(A) + P(B \setminus A)$. Since $P(B \setminus A) \geq 0$, we have $P(A) \leq P(B)$. □

A further consequence is that for any $A, B \in \mathcal{F}$ we have $P(A \cap B) \leq P(A)$ and $P(A \cap B) \leq P(B)$.

1.1.2 Conditional Probability

When we have information about the occurrence of an event, we can update the probabilities of other events accordingly.

Definition 4 (Conditional Probability)**.** *Let $A, B \in \mathcal{F}$ be events in a probability space $(\Omega, \mathcal{F}, P)$. The **conditional probability** of A given B is defined as*

$$P(A|B) = \frac{P(A \cap B)}{P(B)},$$

provided that $P(B) > 0$.

Note that the conditional probability is not defined when $P(B) = 0$, which is intuitive since we cannot condition on an event that has zero probability of occurring. Given B, we can think of $P(\cdot|B)$ as a new probability measure P' that is normalized to B, taking non-zero values only on subsets of B and assigning $P'(B) = 1$.

Observe that the conditional probability of B given A is defined as

$$P(B|A) = \frac{P(A \cap B)}{P(A)},$$

which differs from $P(A|B)$ only by the normalization factor $P(A) > 0$. From this definition, we can derive the *product rule* of probability:

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A).$$

Here, if $P(A) = 0$ or $P(B) = 0$, the product rule still holds due to the monotonicity of probability, even though the conditional probabilities are not defined. However, if $P(A) > 0$, we obtain *Bayes' rule*:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}.$$

This result can be extended to a partition of the sample space.

Proposition 2 (Law of Total Probability)**.** *If Ω can be partitioned into $B_1, \ldots, B_k$ such that $B_i \cap B_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^k B_i = \Omega$, with $P(B_i) > 0$, then for any event A we have:*

$$P(A) = \sum_{i=1}^{k} P(A|B_i)P(B_i).$$

Proof. Suppose that Ω can be partitioned into $B_1, \dots, B_k$ as above. Then, from the properties of probability measures and conditional probability, we have:

$$\begin{aligned} P(A) &= P(A \cap \bigcup_{i=1}^{k} B_i) = P\left(\bigcup_{i=1}^{k} (A \cap B_i)\right) \\ &= \sum_{i=1}^{k} P(A \cap B_i) = \sum_{i=1}^{k} P(A|B_i)P(B_i). \end{aligned}$$

□

Therefore, given a partition $B_1, \dots, B_k$ of Ω, we have the following extension of Bayes' rule:

$$P(B_i|A) = \frac{P(A|B_i)P(B_i)}{\sum_{j=1}^{k} P(A|B_j)P(B_j)}.$$

This is a fundamental result in probability theory, used in many machine learning algorithms, such as Bayesian inference and probabilistic graphical models, as it allows us to "reverse the conditioning".

Two events are said to be independent if the occurrence of one does not affect the probability of the other. More formally:

Definition 5 (Independence)**.** *Let $(\Omega, \mathcal{F}, P)$ be a probability space.*

- *We say that $A, B \in \mathcal{F}$ are* ***independent*** *if $P(A \cap B) = P(A)P(B)$, denoted as $A \perp B$. If $P(B) > 0$, an equivalent definition is $P(A|B) = P(A)$.*
- *A collection of events $A_1, \dots, A_n$ is* ***mutually independent*** *if for any subset $I \subset \{1, \dots, n\}$, we have*

$$P\left(\bigcap_{i \in I} A_i\right) = \prod_{i \in I} P(A_i).$$

1.1.3 Random Variables

If the probability measure is the tool to quantify uncertainty, random variables are the tools to model it. A random variable provides a numerical value depending on the outcome of a random experiment. More precisely, it can be viewed as a function that maps the sample space Ω to the real numbers $\mathbb{R}$.

Definition 6 (Random Variable)**.** *Let $(\Omega, \mathcal{F}, P)$ be a probability space. A* ***random variable*** *on Ω is a measurable function $X : \Omega \to \mathbb{R}$. That is, for any $a \in \mathbb{R}$, the set $A = \{x \in \Omega : X(x) \leq a\}$ is measurable, i.e., $A \in \mathcal{F}$.*

For a random variable X, the event $\{x \in \Omega : X(x) \leq a\}$ is often written as $\{X \leq a\}$ and is called "the event that X is less than or equal to a." The probability of this event is well-defined since it belongs to $\mathcal{F}$. More generally, if B is a subset of the real line, we use the notation $X^{-1}(B)$ or $\{X \in B\}$ to denote the set $\{x \in \Omega : X(x) \in B\}$. The probability $P_X(B) = P(X \in B) = P(X^{-1}(B)) = P(\{x \in \Omega : X(x) \in B\})$ is well-defined since $X^{-1}(B) \in \mathcal{F}$. This follows from a fundamental result in measure theory, which states that the collection of intervals of the form $(-\infty, a]$ generates the Borel σ-algebra $\mathcal{B}$, which is the smallest σ-algebra that contains all open sets in $\mathbb{R}$. Therefore, if $B \in \mathcal{B}$, then B is a countable union, intersection, or complement of intervals of the form $(-\infty, a]$. Since X is measurable, $X^{-1}(B) \in \mathcal{F}$. We call the function $P_X : \mathcal{B} \to [0, 1]$ the *probability law* of X. In fact, one can prove that P_X is a probability measure on $(\mathbb{R}, \mathcal{B})$. Typically, $(\Omega, \mathcal{F}, P)$ remains implicit, and we work directly with the more tangible probability space $((\mathbb{R}, \mathcal{B}), P_X)$.

Proposition 3. *Let $(\Omega, \mathcal{F}, P)$ be a probability space, and let X be a random variable. Then, the probability law P_X is a probability measure on $(\mathbb{R}, \mathcal{B})$.*

Proof. To prove that P_X is a probability measure on $(\mathbb{R}, \mathcal{B})$, we need to verify three properties:

- **Non-negativity:** For any $B \in \mathcal{B}$, $P_X(B) \geq 0$. This follows directly from the definition of $P_X(B)$ as the probability of the event $X^{-1}(B) \in \mathcal{F}$, and since P is a probability measure on $\mathcal{F}$, $P(X^{-1}(B)) \geq 0$.
- **Measure 0:** $P_X(\emptyset) = 0$. Note that $P_X(\emptyset) = P(X^{-1}(\emptyset)) = P(\emptyset) = 0$.
- **Countable additivity:** If $\{B_i\}_{i=1}^{\infty}$ is a countable collection of disjoint sets in $\mathcal{B}$, then $P_X\left(\bigcup_{i=1}^{\infty} B_i\right) = \sum_{i=1}^{\infty} P_X(B_i)$. This follows because
$$P_X\left(\bigcup_{i=1}^{\infty} B_i\right) = P\left(X^{-1}\left(\bigcup_{i=1}^{\infty} B_i\right)\right) = P\left(\bigcup_{i=1}^{\infty} X^{-1}(B_i)\right) = \sum_{i=1}^{\infty} P(X^{-1}(B_i)) = \sum_{i=1}^{\infty} P_X(B_i),$$
where the third equality follows from the countable additivity of the probability measure P on $\mathcal{F}$.
- **Normalization:** $P_X(\mathbb{R}) = 1$. Note that $P_X(\mathbb{R}) = P(X \in \mathbb{R}) = P(X^{-1}(\mathbb{R}))$. Since $X^{-1}(\mathbb{R}) = \Omega$ and $P(\Omega) = 1$, we have $P_X(\mathbb{R}) = 1$.

Since P_X satisfies all four properties of a probability measure, it follows that P_X is a probability measure on $(\mathbb{R}, \mathcal{B})$. □

A sequence of random variables $X_1, X_2, \ldots, X_n$ that take values in $I \subseteq \mathbb{R}$, is said to be *Independently and Identically Distributed (i.i.d.)* if:

- All random variables are identically distributed, i.e., $P_{X_1}(x) = \cdots = P_{X_n}(x)$, for all $x \in I$.
- All random variables are independent to one another, i.e., the joint distribution can be factorized as $P_{X_1,\ldots,X_n}(x_1, \ldots, x_n) = P_{X_1}(x_1) \cdots P_{X_n}(x_n)$, for all $x_1, \ldots, x_n \in I$.

The i.i.d. assumption is foundational in many statistical methods and machine learning algorithms, as it simplifies analysis and often enables the application of central limit theorems and other key results in probability theory.

In many situations, random variables model very complex scenarios that can be intractable. However, it is possible to transform a given random variable to another one via a continuous transformation. Thus, one can extract knowledge from one random variable and transfer it to another. Additionally, the continuous combination of multiple random variables also leads to a random variable.

Theorem 1 (Functions of random variables)**.** *Let $(\Omega, \mathcal{F}, P)$ be a probability space.*

1. *Let $X : \Omega \to \mathbb{R}$ be a random variable, and suppose that $f : \mathbb{R} \to \mathbb{R}$ is a Borel measurable function. Then, $f(X)$ is a random variable.*
2. *Let $X_1, \ldots, X_n : \Omega \to \mathbb{R}$ be random variables, and suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a Borel measurable function. Then, $f(X_1, \ldots, X_n)$ is a random variable.*

Proof. We will prove both parts of the theorem:

1. Let $B \in \mathcal{B}(\mathbb{R})$ be a Borel set. We need to show that $\{f(X) \in B\} \in \mathcal{F}$. Since f is Borel measurable, $f^{-1}(B) \in \mathcal{B}(\mathbb{R})$. Now,

$$\{f(X) \in B\} = \{\omega \in \Omega : f(X(\omega)) \in B\} = \{\omega \in \Omega : X(\omega) \in f^{-1}(B)\} = X^{-1}(f^{-1}(B)).$$

Since X is a random variable, $X^{-1}(f^{-1}(B)) \in \mathcal{F}$. Therefore, $f(X)$ is a random variable.

2. Similarly, let $\mathbf{X} = (X_1, \dots, X_n) : \Omega \to \mathbb{R}^n$ be the vector-valued random variable formed by $X_1, \dots, X_n$. Let $B \in \mathcal{B}(\mathbb{R})$ be a Borel set. We need to show that $\{f(\mathbf{X}) \in B\} \in \mathcal{F}$. Since f is Borel measurable, $f^{-1}(B) \in \mathcal{B}(\mathbb{R}^n)$. Since $\mathbf{X}$ is a random vector (as each component is a random variable), the same reasoning as before leads to $\mathbf{X}^{-1}(f^{-1}(B)) \in \mathcal{F}$. Therefore, $f(X_1, \dots, X_n)$ is a random variable.

□

Definition 7 (Change of random variables)**.** *Let $(\Omega, \mathcal{F}, P)$ be a probability space, $X : \Omega \to \mathbb{R}$ be a random variable, and $f : \mathbb{R} \to \mathbb{R}$ be a continuous function. The change of variables from X to $Y = f(X)$ is the transformation that induces a new probability measure P_Y on $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ defined by:*

$$P_Y(B) = P(f(X) \in B) = P(X \in f^{-1}(B))$$

for all Borel sets $B \in \mathcal{B}(\mathbb{R})$.

Random variables can be classified into two main categories: discrete and continuous. Discrete random variables take on a countable number of distinct values, while continuous random variables can take on any value within a range. In the following subsections we will state some of the basic definitions and properties of discrete and continuous random variables, that are relevant for the mathematical formalization of generative models.

Discrete Random Variables

Discrete random variables are characterized by their Probability Mass Function (PMF), which assigns probabilities to each possible value the random variable can take.

Definition 8 (PMF)**.** *Let X be a discrete random variable taking values in a countable set S. The **Probability Mass Function (PMF)** of X, denoted $p_X(x)$, is defined as:*

$$p_X(x) = P(X = x), \quad x \in S.$$

The PMF satisfies the following properties:

1. $p_X(x) \geq 0$ for all $x \in S$.

2. $\sum_{x \in S} p_X(x) = 1$.

For discrete random variables, we can define joint, marginal, and conditional PMFs for multiple random variables.

Proposition 4 (Discrete Random Variables)**.** *For discrete random variables X and Y, we have the following definitions that extend from the definition of PMFs and the basic properties of probability measures stated above:*

*1. **Joint PMF:** $p_{X,Y}(x, y) = P(X = x, Y = y)$.*

*2. **Marginal PMF:** $p_X(x) = \sum_y p_{X,Y}(x, y)$.*

*3. **Conditional PMF:** $p_{X|Y}(x|y) = \frac{p_{X,Y}(x,y)}{p_Y(y)}$ if $p_Y(y) > 0$.*

Independence of discrete random variables can be defined in terms of their joint and marginal PMFs.

Proposition 5. *Discrete random variables X and Y are independent if and only if:*

$$p_{X,Y}(x,y) = p_X(x)p_Y(y) \quad \textit{for all } x \in S_X, y \in S_Y$$

where S_X and S_Y are the support sets of X and Y respectively.

Lastly, the expectation and conditional expectation, that provide a measure of the average value of a random variable, will be crucial for understanding generative models, since they usually aim to generate data samples that are representative of the underlying distribution, thus having similar expected values.

Definition 9 (Expected Value)**.** *The expected value of a discrete random variable X is defined as:*

$$E[X] = \sum_{x \in S} x \cdot p_X(x).$$

Definition 10 (Conditional Expectation)**.** *The conditional expectation of X given $Y = y$ is defined as:*

$$E[X|Y = y] = \sum_{x} x \cdot p_{X|Y}(x|y).$$

The simplest example of a discrete random variable is the Bernoulli distribution. In our case, such distribution will be used to encode the binary latent space of our generative model, as well as the main model to generate binary data samples.

Example 1 (Bernoulli distribution)**.** *The Bernoulli distribution is a discrete probability distribution of a random variable which takes the value 1 with probability p and the value 0 with probability $q = 1 - p$. A random variable X follows a Bernoulli distribution with parameter $p \in [0,1]$, denoted $X \sim \mathcal{B}(X; p)$, if its PMF is given by:*

$$p_X(x) = \begin{cases} p & \textit{if } x = 1 \\ 1 - p & \textit{if } x = 0. \end{cases}$$

The expected value of a Bernoulli random variable is:

$$E[X] = 1 \cdot p + 0 \cdot (1 - p) = p.$$

Continuous Random Variables

Continuous random variables are characterized by their ability to take on any value within a continuous range. Unlike discrete random variables, continuous random variables are described by probability density functions rather than probability mass functions.

Definition 11 (PDF)**.** *Let X be a continuous random variable. The probability density function (PDF) of X, denoted by $f_X : \mathbb{R} \to [0, \infty)$, is a function that satisfies:*

$$P(a \leq X \leq b) = \int_a^b f_X(x)\,dx$$

for all $a, b \in \mathbb{R}$ with $a \leq b$. The PDF must also satisfy:

$$\int_{-\infty}^{\infty} f_X(x)\,dx = 1.$$

The PDF is related to the Cumulative Distribution Function (CDF) as follows:

Definition 12 (CDF)**.** *Let X be a continuous random variable with PDF f_X. The cumulative distribution function (CDF) of X, denoted by $F_X : \mathbb{R} \to [0,1]$, is defined as:*

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} f_X(t)\, dt$$

for all $x \in \mathbb{R}$.

Theorem 2 (Fundamental Theorem of Calculus for CDFs)**.** *If X is a continuous random variable with PDF f_X and CDF F_X, then:*

$$f_X(x) = \frac{d}{dx} F_X(x)$$

at all points x where F_X is differentiable.

This correspondence allows us to work interchangeably with CDFs and PDFs, choosing whichever is more convenient for a given problem. The CDF has several important properties:

Proposition 6 (CDF Properties)**.** *Let F_X be the CDF of a continuous random variable X. Then F_X satisfies:*

1. *Monotonicity: If $x_1 < x_2$, then $F_X(x_1) \leq F_X(x_2)$ for all $x_1, x_2 \in \mathbb{R}$.*
2. *Continuity: F_X is continuous on $\mathbb{R}$.*
3. *Limiting behavior:*
   - $\lim_{x \to -\infty} F_X(x) = 0$
   - $\lim_{x \to +\infty} F_X(x) = 1$
4. *Probability of an interval: For any $a < b$,*

$$P(a < X \leq b) = F_X(b) - F_X(a).$$

Similar to discrete random variables, from the definition of continuous random variables and PDFs, we define joint, marginal, and conditional PDFs:

Proposition 7 (Continuous Random Variables)**.** *For continuous random variables X and Y:*

- ***Joint PDF:*** *$f_{X,Y}(x,y)$ is the joint PDF of X and Y, and satisfies:*

$$P((X,Y) \in A) = \int\int_A f_{X,Y}(x,y)\, dxdy$$

  for any measurable set $A \subseteq \mathbb{R}^2$.

- ***Marginal PDF:*** *$f_X(x)$ is the marginal PDF of X, and is obtained by integrating over all possible values of Y:*

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)\, dy.$$

- ***Conditional PDF:*** *$f_{X|Y}(x|y)$ is the conditional PDF of X given $Y = y$, and is defined as:*

$$f_{X|Y}(x|y) = \frac{f_{X,Y}(x,y)}{f_Y(y)}$$

  when $f_Y(y) > 0$.

These definitions lead to the chain rule of probability:

Theorem 3 (Chain Rule of Probability)**.** *For continuous random variables X and Y:*

$$f_{X,Y}(x,y) = f_{X|Y}(x|y)f_Y(y) = f_{Y|X}(y|x)f_X(x).$$

Independence for continuous random variables can also be defined in terms of their joint and marginal PDFs:

Proposition 8. *Continuous random variables X and Y are independent if and only if:*

$$f_{X,Y}(x,y) = f_X(x)f_Y(y) \quad \textit{for all } x, y \in \mathbb{R}.$$

Expected values for continuous random variables are defined using integrals, as to average over all possible values of the random variable:

Definition 13 (Expected Value)**.** *The expected value of a continuous random variable X with PDF f_X is defined as:*

$$E[X] = \int_{-\infty}^{\infty} x f_X(x)\, dx$$

provided the integral exists.

Definition 14 (Conditional Expectation)**.** *The conditional expectation of X given $Y = y$ is defined as:*

$$E[X|Y = y] = \int_{-\infty}^{\infty} x f_{X|Y}(x|y)\, dx$$

provided the integral exists.

Proposition 9 (Linearity of Expectation)**.** *For continuous random variables X and Y, and constants $a, b \in \mathbb{R}$:*

$$E[aX + bY + c] = aE[X] + bE[Y] + c.$$

A fundamental theorem for computing expectations of functions of random variables is the Law of the Unconscious Statistician (LOTUS):

Theorem 4 (LOTUS)**.** *Let X be a continuous random variable with PDF f_X, and let $g : \mathbb{R} \to \mathbb{R}$ be a measurable function. Then:*

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x)\, dx$$

provided the integral exists.

Bayes' rule can also be written in terms of PDFs:

Theorem 5 (Bayes' Rule)**.** *For continuous random variables X and Y:*

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)f_X(x)}{f_Y(y)} = \frac{f_{Y|X}(y|x)f_X(x)}{\int_{-\infty}^{\infty} f_{Y|X}(y|t)f_X(t)\, dt}.$$

Another important property that will be useful in forthcoming chapters is the PDF under a change of continuous random variables. If $Y = g(X)$, where g is a monotonous change of random variables with continuous derivative, then the PDF of Y can be obtained from the PDF of X as

$$f_Y(y) = f_X(g^{-1}(y)) \left| \frac{d}{dy} g^{-1}(y) \right|. \tag{1.1}$$

Note that if g is increasing, then

$$P(Y \leq y) = P(g(X) \leq y) = P(X \leq g^{-1}(y)) = F_X(g^{-1}(y)),$$
$$f_Y(y) = \frac{d}{dy}F_X(g^{-1}(y)) = f_X(g^{-1}(y))\frac{d}{dy}g^{-1}(y).$$

The Gaussian distribution is a fundamental example of a continuous random variable, and is widely used in statistics and machine learning. In fact, most advanced generative models are based on the Gaussian distribution, as it is a simple yet powerful model for continuous data. We will introduce them since they will be used in the generative models we will study, and we will try to understand their properties to translate them into a discrete space where we can work with Bernoulli distributions.

Example 2 (Gaussian Distribution)**.** *A random variable X follows a* ***Gaussian distribution*** *with parameters $\mu \in \mathbb{R}$ and $\sigma^2 > 0$, denoted $X \sim \mathcal{N}(X, \mu, \sigma^2)$, if its PDF is given by:*

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad x \in \mathbb{R}.$$

Expected value: $E[X] = \mu$
Variance: $Var(X) = \sigma^2$

Example 3 (Multivariate Gaussian Distribution)**.** *The multivariate Gaussian distribution is a generalization of the one-dimensional Gaussian distribution to higher dimensions. A random vector $\mathbf{X} = (X_1, \dots, X_n)^T$ follows a* ***multivariate Gaussian distribution*** *with mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$ (positive definite), denoted $\mathbf{X} \sim \mathcal{N}(\mathbf{X}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, if its PDF is given by:*

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \quad \mathbf{x} \in \mathbb{R}^n.$$

Expected value: $E[\mathbf{X}] = \boldsymbol{\mu}$
Covariance matrix: $Cov(\mathbf{X}) = \boldsymbol{\Sigma}$

These definitions, theorems, and examples provide a rigorous foundation for working with continuous and discrete random variables. They are essential for understanding and developing probabilistic models, including generative models in machine learning and statistics. For the sake of brevity, we have omitted many important results and concepts in basic probability theory, as well as some proofs. Further details can be found in the references provided, especially in Nualart and Sanz-Solé, 1990 and Bishop, 2006.

1.1.4 Modelling

Probabilistic modelling, it is often confounded with other types of modelling. We briefly discuss probabilistic, statistical and Bayesian models to provide a formal definition and clearly differentiate between the methods.

Probabilistic Models

We call probability, or *probabilistic model*, to the probability space $(\Omega, \mathcal{F}, P)$ corresponding to a random phenomenon. Often one works with the random variables and their respective PDFs or PMFs accordingly to P.

For example, the probabilistic model that models the random phenomenon of tossing a perfect coin corresponds to a Bernoulli random variable with probability of success $1/2$. The probability space is $(\{0,1\}, 2^{\{0,1\}}, P)$, where $P(0) = P(1) = 1/2$, and can be specified by the probability law

$$p(x) = \left(\frac{1}{2}\right)^x \left(\frac{1}{2}\right)^{1-x}, \quad x \in \{0,1\}.$$

Statistical Models

A *statistical model* is a set of probability measures defined on the same sample space Ω, i.e., a set of random variables that are defined on the same probability space. Usually, we refer to *parametric statistical models*, where the random variables X_θ are parameterised by $\theta \in \Theta \in \mathbb{R}^d$. The probability measure P_θ is defined by the probability law of X_θ, and the set of all probability measures $\{P_\theta : \theta \in \Theta\}$ is the statistical model. The PDFs or PMFs of the random variables are denoted by $p_\theta(x) = p(x|\theta)$. The goal of statistical modelling is to learn the parameter θ that best describes the data. The outcome of this process is a probabilistic model that approximates the data distribution.

For example, the collection of Bernoulli random variables parameterised by the probability of success θ is a statistical model for coin tossing. After collecting enough data of a perfect coin, one can estimate that the probability of success is close to $\theta = 1/2$. Arriving to the probability model from above. This statistical model is specified by

$$p(x|\theta) = \theta^x (1-\theta)^{1-x}, \quad x \in \{0,1\},\ \theta \in [0,1].$$

Bayesian Models

A *Bayesian model* is a statistical model where the parameters θ are considered random variables. The probability measure $P(\theta)$ is called the *prior distribution.* Each PDF or PMF $p_\theta(x)$ is subject to the prior PDF or PMF $p(\theta)$, which defines the joint distribution $p(x,\theta) = p(x|\theta)p(\theta)$. Assuming that the conditional distribution $p(x|\theta)$ is defined on Ω_x, a Bayesian model formally corresponds to a probabilistic model on the product space $\Omega_x \times \Theta$. The goal of Bayesian modelling is to learn the *posterior distribution* $p(\theta|x)$, which is the conditional distribution of the parameters given the data. This is achieved by applying Bayes' rule. Then, the posterior distribution can be used to make predictions or decisions.

For example, following the previous example of a Bernoulli random variable, one can consider a Bayesian model by assuming a prior distribution to θ. A common choice is the Beta distribution; thus we assume that $\theta \sim \text{Beta}(\alpha,\beta)$,

$$p(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{Z(\alpha,\beta)}, \quad \theta \in [0,1],$$

where

$$Z(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

is the normalizing constant. It ensures that the integral of $p(\theta)$ over $[0,1]$ is equal to 1. The parameters α and β are sometimes called *hyperparameters* and are assumed to be either fixed, or unknown. Then, the joint distribution is

$$\begin{aligned} p(x,\theta) &= p(x|\theta)p(\theta) \\ &= \theta^x(1-\theta)^{1-x}\frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{Z(\alpha,\beta)} \\ &= \frac{\theta^{x+\alpha-1}(1-\theta)^{\beta-x}}{Z(\alpha,\beta)}. \end{aligned}$$

and is defined on $\Omega_x \times \Theta = \{0, 1\} \times [0, 1]$. Note that if α and β are unknown, then we can formulate a statistical model $p(x, \theta|\alpha, \beta)$, which once again can be turned into a Bayesian model by assuming a prior distribution to α and β.

1.2 Generative Models

As discussed in the beginning of this chapter, in generative modelling, we are interested in learning a model that can generate new data samples from a data space. The essential idea behind the generation problem is to treat data observations x as samples from a distribution $p(x)$ that we would like to find, and use them to approximate it.

In this context, we are interested in the real valued case, where $\Omega = \mathbb{R}^D$ for some $D \in \mathbb{N}$. The probability space $(\mathbb{R}^D, \mathcal{B}(\mathbb{R}^D), P)$ is defined by the Borel σ-algebra $\mathcal{B}(\mathbb{R}^D)$, which is the smallest σ-algebra that contains all open sets in $\mathbb{R}^D$. And the probability measure P is induced by the Lebesgue measure, $P(A) = \int_A p(\mathbf{x})\, d\mathbf{x}$. If $D = 1$, recall that one can define the cumulative distribution function $F(x)$ as

$$F(x) := P(\{X \leq x\}) = \int_{-\infty}^{x} p(\mathbf{x})\, d\mathbf{x}.$$

Note that the probability density function $p(x)$ is the derivative of $F(x)$, $p(x) = \frac{dF(x)}{dx}$. Since the probability distribution of a random variable is defined by its density function, we will use these terms interchangeably.

Assume we have a *dataset* $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$ of N samples, where each *data sample* $\mathbf{x}_i$ is a vector of D dimensions. We call the vector space $\mathbb{R}^D$ *data space*. The *data distribution* $p_{\mathcal{D}}(\mathbf{x})$ is a probability distribution in $\mathbb{R}^D$ that describes the likelihood of a particular sample $\mathbf{x} \in \mathcal{D}$.

For instance, in the case of 64×64 RGB images, the data space is $\mathbb{R}^{64\times64\times3}$. If we have a dataset $\mathcal{D}$ of images of cats in this data space, each sample $\mathbf{x}$ is a vector of shape $64 \times 64 \times 3$, and the data distribution $p_{\mathcal{D}}(\mathbf{x})$ describes the likelihood a particular image of a cat within the image data space.

Definition 15. *A* ***generative model*** $q_{\mathcal{D},\theta}$ *is a probabilistic model that approximates the data distribution* $p_{\mathcal{D}}$*, where* $\mathcal{D}$ *is a dataset, and* θ *are the prarameters of the model.*

To simplify the notation, we will assume that the dataset $\mathcal{D}$ is fixed, and we will denote the generative model as q_θ. Usually the data distribution p is unknown, and the goal of generative modeling is to find the parameters θ for which q_θ best approximates $p_{\mathcal{D}}$. To do so, we need a way to measure the *similarity* between two probability distributions. A *divergence* measure is a function $D(p||q) : S \times S \to \mathbb{R}$ that takes two probability distributions p and q over the a space of distributions S, with the properties:

- $D(p||q) \geq 0$ for all $p, q \in S$.
- $D(p||q) = 0$ if and only if $p = q$.

Divergence generalizes the concept of distance, since it is not required to be symmetric, and does not satisfy the triangle inequality. Therefore, the learning problem for generative models can be formulated as an optimization problem, where the goal is to minimize the divergence:

$$\arg\min_{\theta\in\Theta} D(p_{\mathcal{D}}(\mathbf{x})||q_\theta(\mathbf{x})). \tag{1.2}$$

Note that given a model q_θ, with $\theta \in \Theta \subset \mathbb{R}^m$, for some $m \in \mathbb{N}$, the generation problem resembles the statistical modelling problem, and if assuming a prior to the parameters θ, we can turn the generative model into a Bayesian model. Thus, many techniques from variational inference and Bayesian statistics can be applied to find efficient methods to solve the task. The rise of deep learning and the development of new optimization algorithms have made it possible to train very complex generative models, which are defined by large deep neural networks. Together with a huge amount of high-dimensional data, these models have been able to generate very realistic samples of images, audio, and text.

There are several approaches to generative modeling. One of the most common is to define a density function for a generative model and maximize the likelihood of observing the data samples. This approach is known as the *likelihood maximization* approach. Another approach, known as the *adversarial* approach, involves estimating the distance between a given generative model's output and the real data, and then minimizing this difference by an adversarial auxiliar model. The later approach is outside the scope of this thesis, and we will focus on the likelihood maximization approach.

1.2.1 The Likelihood Maximization Approach

Given a generative model q_θ and a dataset $\mathcal{D}$ of i.i.d. samples, the likelihood maximization approach consists of finding the parameters θ that maximize the likelihood of observing the data samples in the dataset $\mathcal{D}$. This is defined as the probability of observing the data samples in $\mathcal{D}$ given the model q_θ, and since samples are i.i.d., this is the product of the likelihood of each data sample in $\mathcal{D}$:

$$\arg\max_{\theta\in\Theta} q_\theta(\mathbf{x}\in\mathcal{D}) = \arg\max_{\theta\in\Theta} \prod_{\mathbf{x}\in\mathcal{D}} q_\theta(\mathbf{x}) = \arg\max_{\theta\in\Theta} L(\theta),$$

where $L(\theta) = q_\theta(\mathbf{x}\in\mathcal{D})$ is the joint distribution of the data samples in $\mathcal{D}$, and is referred as the *likelihood function*.

Definition 16. *We call* ***maximum likelihood estimation (MLE)*** *of the parameters θ to the maximizer of the likelihood function:*

$$\theta_{MLE} = \arg\max_{\theta\in\Theta} L(\theta).$$

Since L is a product of probabilities, it is often more convenient to work with the *log-likelihood function* $\log L(\theta)$, that uses the logarithm properties to transform the product into a more manageable optimization objective.

Proposition 10. *The MLE of the parameters θ is equivalent to the maximizer of the log-likelihood function:*

$$\theta_{MLE} = \arg\max_{\theta\in\Theta} \sum_{\boldsymbol{x}\in\mathcal{D}} \log q_\theta(\boldsymbol{x}).$$

Proof. Since the logarithm is a monotonically increasing function,

$$\arg\max_x f(x) = \arg\max_x \log f(x).$$

Therefore, the maximizer of the log-likelihood function is the same as the maximizer of the likelihood function. □

We can also see that for any positive constant c,

$$\arg\max_x \log f(x) = \arg\max_x (c\log f(x)).$$

Thus, the MLE is invariant to scaling by a positive constant. This is why the log-likelihood is often written as the average log-likelihood, which is the average of the log-likelihood of each data sample in the dataset:

$$\theta_{\text{MLE}} = \arg\max_{\theta\in\Theta} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}\in\mathcal{D}} \log q_\theta(\mathbf{x}) \approx \arg\max_{\theta\in\Theta} \mathbb{E}_{p_\mathcal{D}}[\log q_\theta(\mathbf{x})].$$

Later, we will see that specific choice of divergence measure can make minimizing the divergence and maximizing the likelihood equivalent. Whenever the likelihood function is differentiable, simple and the dataset is small, the MLE can be computed analytically. However, in practice, the likelihood function is often complex and intractable, and the dataset is large, which makes the optimization problem infeasible to solve analytically. In such cases, one can use optimization algorithms to approximate the MLE. One of the most common is the *stochastic gradient descent* (SGD) algorithm, which uses a random subset of data samples to compute the gradient of the log-likelihood function, and then updates the parameters in the opposite direction of the gradient. This process is repeated until the algorithm converges to a local maximum.

A common way to model $p_\mathcal{D}$ is by means of a probabilistic model with a hidden latent variable, which is a random variable that is not observed in the data space. The latent variable can be used to represent the data in a more compact and efficient way, and can be used to generate new data samples.

Definition 17 (Latent Variable Model)**.** *A **latent variable model (LVM)** is a probabilistic model that includes a hidden latent variable $\boldsymbol{z}$ from a latent data space $\mathbb{R}^{D'}$. The model is defined by the joint distribution $p(\boldsymbol{x}, \boldsymbol{z})$, where $\boldsymbol{x}$ is the data sample in the data space $\mathbb{R}^D$. A parameterized latent variable model is defined by the joint distribution $p_\theta(\boldsymbol{x}, \boldsymbol{z})$, where θ are the parameters of the model. Whenever the probabilistic model p_θ is defined by a deep neural network, we call it a **deep latent variable model (DLVM)**.*

2 Latent Space

The growth of Machine Learning (ML) and Artificial Intelligence has been exponential in the last decade. Models are increasing in complexity and size and datasets are growing at an unprecedented rate. In particular, the outstanding capabilities of generative models have been a key factor in the development of new applications. This success is due in part to the advances in data representation. Since the early stages of information theory, the problem of dimensionality reduction and data compression has been a central topic. As the amount of data grows, the need for efficient data representation becomes a key challenge. The goal is to find a lower-dimensional representation of the data that retains the *relevant* information of the original data. This is particularly important for ML, where large datasets from large-dimensional spaces are used to train huge models. A data compression process can be defined as a two step process:

Definition 18 (Data compression process)**.** *Given a dataset $\mathcal{D} \subset \mathbb{R}^n$, the tuple $(\mathcal{D}, E, D)$ is a* ***data compression process*** *if:*

- *$E : \mathcal{D} \to \mathcal{Z} \subset \mathbb{R}^d$ is the* ***encoder****, which maps the data to a lower-dimensional space $\mathcal{Z}$, i.e., any $\boldsymbol{x} \in \mathcal{D}$, which has dimension n, is mapped to $E(\boldsymbol{x}) = \boldsymbol{z}$, which has dimension $d < n$.*
- *$D : \mathcal{Z} \to \mathcal{D}$ is the* ***decoder****, which maps the lower-dimensional representation back to the original space.*

Note that, in practice, real numbers are stored in computers following the 754-IEEE standard, stablished in mid 1985 ("IEEE Standard for Binary Floating-Point Arithmetic" 1985). In this regard, numbers are stored in binary format, occupying a fixed number of bits in memory. Each bit corresponds to a binary digit. The usual representation for storing real numbers is the *32-bit floating-point representation*, which consists of 32 bits, divided into three parts: the sign bit (s), the exponent (e), and the mantissa (m); with 1, 8, and 23 bits respectively. The value of the number is given by the formula:

$$(-1)^s \times 2^{e-127} \times 1.m.$$

Observe that the number of bits used to store a real number is fixed, and the precision of the number is limited by the number of bits used. The precision can be increased by using more bits, i.e., using `float64` or `float128` representations. However, this comes at the cost of memory, which is a limited resource. Integers are stored in a similar way, but they just require the sign bit and the mantissa, a common representation is the `int32` representation, which uses 32 bits to store an integer, 1 for the sign and 32 for the number. Similarly, we can define `int8`, `int16`, `int64`, etc. representations, which use 8, 16, and 64 bits, respectively.

With this in mind, we can redefine the data compression process in terms of the number of bits used to store the data. Ideally, we would like to find an encoder-decoder pair that maps the data to a lower-dimensional space, that is, using less bits than the original data, without losing *relevant* information. We will abuse the notation and when we refer to a lower-dimensional representation, we will mean a representation that uses less bits to store the data. In our context, the relevant information consists of the patterns and structures of the data that are useful for the model to learn the underlying generative factors of the data. As long as the chapter progresses, the term relevant will gain more meaning.

A classical method for data compression and dimensionality reduction is the so called Principal Component Analysis (PCA), which is a linear transformation that finds the directions of maximum variance in the data. The idea is to project the data into a lower-dimensional space, such that the variance of the projected data is maximized. The directions of maximum variance where the projections are done are called the principal components. However, PCA is a linear method, and it is not able to capture the non-linear structure of the data, and fails to provide a meaningful representation for complex datasets (Lever, Krzywinski, and Altman, 2017). In the last years, the advances in deep learning have provided new tools for data compression and representation learning. In particular, the use of Neural Networks has been a key factor in the development of new models for data compression. In this chapter, we will provide the foundation for obtaining a Neural Network which is able to compress data into a binary space. First, we introduce the concept of Autoencoder. Then, inspired by Variational Inference, we give a formal definition of Variational Autoencoders, which are a type of Autoencoder that learns a stochastic mapping between the observed and latent spaces. Finally, we will introduce Vector quantized Variational Autoencoders, which are a type of Variational Autoencoder that uses vector quantization to discretize the latent space.

An efficient data representation is crucial for the training of ML models, and an encoder-decoder pair can be used to map the original data to a more expressive and compressed representation. In ML, the output space of the encoder is called the *latent space* or *feature space*, where we have a *latent representation* $E(\mathbf{x}) = \mathbf{z}$ for any given data sample $\mathbf{x} \in \mathcal{D}$.

2.1 Autoencoders

An autoencoder, first introduced in Rumelhart and McClelland, 1987, is essentially a neural network with a bottleneck that learns the identity function in an unsupervised way such that it can compress and reconstruct an original input. Such low-dimensional representation can be used as embedding vector in various applications, help data compression, or reveal the underlying data generative factors.

We call a subfield of ML that deals with finding hidden patterns in unlabeled data, without the need of human intervention, *unsupervised learning* (Murphy, 2023). An autoencoder is a type of unsupervised learning model that learns to compress and decompress data without explicitly being told how to do it (Baldi, 2012).

Definition 19 (Autoencoder)**.** *Given a dataset $\mathcal{D} \subset \mathbb{R}^n$, the encoder-decorder pair $(E_{\theta'}, D_{\theta''})$ is an **autoencoder** if $(\mathcal{D}, E_{\theta'}, D_{\theta''})$ is a data compression process, and $E_{\theta'}$ and $D_{\theta''}$ are neural networks, with parameters θ' and θ'', respectively.*

The encoder $E_{\theta'} : \mathbb{R}^n \to \mathbb{R}^d$, which is the first part of the model and maps the data $\mathbf{x} \in \mathcal{D} \subset \mathbb{R}^n$ to a latent representation $E_{\theta'}(\mathbf{x}) = \mathbf{z} \in \mathcal{Z} \subset \mathbb{R}^d$, with $d < n$. Then, a decoder $D_{\theta''} : \mathbb{R}^d \to \mathbb{R}^n$ reconstructs the original data sample from $\mathbf{z}$ to $\mathbf{x}$. The output of the decoder is $\hat{\mathbf{x}} = D_{\theta''}(\mathbf{z}) = D_{\theta''}(E_{\theta'}(\mathbf{x}))$, which is of the same size as the original data.

We can simplify the notation of the autoencoder to be the composition $f_\theta = D_{\theta''} \circ E_{\theta'}$. In practice, the autoencoder is obtained by training the neural networks that compose it. That is, by solving the optimization problem which consists of minimizing the *reconstruction loss*, which can be any measure Δ that quantifies how the output of the decoder differs from the original data. Therefore, the goal is to look for the parameters θ such that

$$\arg\min_\theta \mathbb{E}_{\mathbf{x}\in\mathcal{D}}[\Delta(\mathbf{x}, f_\theta(\mathbf{x}))].$$

The training objective of the model can be further extended to include a *regularization term* R, which consists of another measure based on the complexity of the model, added to the reconstruction loss:

$$\arg\min_{\theta} \left(\mathbb{E}_{\mathbf{x}\in\mathcal{D}}[\Delta(\mathbf{x}, f_\theta(\mathbf{x}))] + \lambda R(\theta) \right),$$

where $\lambda \in \mathbb{R}$ is a hyperparameter that controls the strength of the regularization term. R can be any measure; the most common are the L_1 and L_2 norms of the vector of parameters θ.

Definition 20 (Loss function)**.** *We call the* ***Loss Function*** *of a machine learning model the function* $L : \Theta \times \mathbb{R}^n \to \mathbb{R}$ *that defines the training objective of the model, where* Θ *is the parameter space of the model, and* $\mathbb{R}^n$ *the data space. The function* L *returns a scalar value that quantifies how well the model is performing.*

In the simple case on the non-regularized autoencoder, the loss function is $L(\theta, \mathcal{D}) = \mathbb{E}_{\mathbf{x}\in\mathcal{D}}[\Delta(\mathbf{x}, f_\theta(\mathbf{x}))]$. The choice of the loss function is crucial for the training of the model. Ideally, the loss function should be differentiable, easy to compute, and have an achievable minimum. Whenever this is the case, the optimal parameters θ of the model can be obtained by either solving the optimization problem analytically, or by using optimization algorithms such as Stochastic Gradient Descent (SGD). Usually, in ML, both model and data are huge, so the latter option is chosen. The most common loss functions for autoencoders are the Mean Squared Error (MSE) and the Binary Cross-Entropy (BCE).

Definition 21 (MSE)**.** *Given a dataset* $\mathcal{D} \subset \mathbb{R}^n$ *with* $M = |\mathcal{D}|$*, the* ***Mean Squared Error*** *is defined as*

$$L_{MSE}(\theta, \mathcal{D}) := \frac{1}{M} \sum_{\boldsymbol{x}\in\mathcal{D}} |\boldsymbol{x} - \hat{\boldsymbol{x}}|^2,$$

where $\hat{\boldsymbol{x}} = f_\theta(\boldsymbol{x})$ *is the output of the model, and* $|\cdot|$ *is the Euclidean norm.*

Definition 22 (BCE)**.** *Given a dataset* $\mathcal{D} \subset \mathbb{R}^n$ *with* $M = |\mathcal{D}|$*, the* ***Binary Cross-Entropy*** *is defined as*

$$L_{BCE}(\theta, \mathcal{D}) := -\frac{1}{M} \sum_{\boldsymbol{x}\in\mathcal{D}} \sum_{i=1}^{n} [x_i \log(\hat{x}_i) + (1 - x_i)\log(1 - \hat{x}_i)],$$

where $\hat{\boldsymbol{x}} = f_\theta(\boldsymbol{x})$ *is the output of the model, and* log *is the natural logarithm.*

Observe that both loss functions have a minimum at $\hat{\mathbf{x}} = \mathbf{x}$, which is the identity function. Although having an Autoencoder close to the identity function may seem desirable, in practice it is not for the generative case. An overfitted model exhibits an unstable behaviour and fails to generalize to unseen data. Such model would excel at compressing and reconstructing a sample from the training dataset, but it would fail when inputed with even a slightly different data sample. The theorem suggests that Autoencoders suffer from a lack on generalizability and provide a *narrow* latent space, due to their deterministic nature. They fail especially for generation related problems, where we not only want to represent an element of the dataset, but a distribution of its features in the latent space from which we can sample novel datapoints, which in furthermore has been largely demonstrated empirically (Steck, 2020). Thus, the goal is to estimate a *latent distribution* were we can sample a latent element and then *reconstruct* a never-seen data sample. A major breakthrough in this regard comes from the hand of Variational Autoencoders (Kingma and Welling, 2014).

2.2 Variational Autoencoders

The framework for *Variational Autoencoders* (VAEs) was introduced in Kingma and Welling, 2014. They differ from regular Autoencoders in that they learn a stochastic mapping between the observed and latent spaces. The motivation relies on the assumption that the stochastic mapping allows the model to learn a more diverse and meaningful latent space, that is, a space that captures the underlying generative factors of the data. The latent space is a distribution, rather than a single point, which allows the model to generate new samples by sampling from the latent distribution. As Autoencoders, VAEs are composed of two coupled independently parameterized models, an encoder and a decoder. To emphasize its generative capabilities, the two parts are also called the *recognition model* and the *generative model*, respectively. Despite the similarities with Autoencoders, VAEs emerge from a completely different theoretical framework. As generative models, the goal is to find a probabilistic model that approximates the data distribution with the density $p_{\mathcal{D}}$ for a given dataset $\mathcal{D}$, by means of a latent variable model. Approximating difficult to compute probability densities is one of the main problems in core statistics. We will introduce the theoretical framework behind VAEs, Variational Inference, a technique from Bayesian statistics that allows us to approximate intractable posterior distributions. Then, we will formalize the definition of VAEs from the probabilistic perspective, and draw the connection with Autoencoders.

2.2.1 Variational Inference

In the forthcoming discussion, we assume that $\mathbf{x} \in \mathcal{D}$ are i.i.d. samples from the data space. From the generative perspective, the goal is to estimate the data distribution $p_{\mathcal{D}}$ from a dataset $\mathcal{D}$. To do so, we define a family of parameterized densities p_θ, where θ are the parameters of the probabilistic model, and maximize the likelihood of the data under the model 16:

$$\arg\max_{\theta} \prod_{\mathbf{x}\in\mathcal{D}} p_\theta(\mathbf{x}). \tag{2.1}$$

One way to estimate a probability density $p_{\mathcal{D}}$ is by introducing latent variables $\mathbf{z}$, which are unobserved variables over an auxiliary space $\mathcal{Z}$, and defining the latent variable model $p_{\mathcal{D},\mathcal{Z}}(\mathbf{x},\mathbf{z}) = p_{\mathcal{D}|\mathcal{Z}}(\mathbf{x}|\mathbf{z})p_{\mathcal{Z}}(\mathbf{z})$. Therefore, the marginal distribution of the data is given by

$$p_{\mathcal{D}}(\mathbf{x}) = \int_{\mathcal{Z}} p_{\mathcal{D},\mathcal{Z}}(\mathbf{x},\mathbf{z})\, d\mathbf{z} = \int_{\mathcal{Z}} p_{\mathcal{D}|\mathcal{Z}}(\mathbf{x}|\mathbf{z})p_{\mathcal{Z}}(\mathbf{z})\, d\mathbf{z}. \tag{2.2}$$

To simplify the notation, we will denote with p each of the densities, and distinguish them by the arguments, e.g. $p_{\mathcal{D},\mathcal{Z}}(\mathbf{x},\mathbf{z}) = p(\mathbf{x},\mathbf{z})$. To model such unknown densities from the observed data, the following assumptions are made:

- The term $p(\mathbf{x}|\mathbf{z})$ corresponds with the *likelihood* of the data given the latent variables, which is a distribution over the observed space and is easy to compute.
- The latent variables are generated by a known *prior* distribution $p(\mathbf{z})$, which is a distribution over the latent space.

Given the prior and the likelihood, computing samples from the joint distribution is straightforward. However, usually computing the integral in 2.2 is intractable since requires the evaluation on every latent variable. Note that with the chain rule of probability:

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

Therefore, the intractability of computing the marginal distribution $p(\mathbf{x})$ is related to the intractability of the posterior distribution $p(\mathbf{z}|\mathbf{x})$. Thus, to estimate the data distribution, we can estimate first the posterior distribution $p(\mathbf{z}|\mathbf{x})$. This is a main problem in Bayesian statistics, and it is known as the *inference problem*. The term inference refers to estimating the unknown latent variables given the observed data. Variational Inference (VI) is one method for approximating intractable posterior distributions as such, by transforming a probabilistic problem into an optimization problem. Although there are other methods that tackle this problem such as Markov-Chain Monte Carlo (MCMC) (Barber, 2012), VI is the preferred method when the dataset is large and the probabilistic models are complex, especially in the case of Machine Learning, where functions are modeled by neural networks and stochastic gradient descent allows efficiently solving optimization problems with them.

To this end, we define a parameterized statistical model $\{q_\phi : \phi \in \Phi\}$ over the latent space $\mathcal{Z}$, and we look for the parameters ϕ that best approximate the posterior distribution $p(\mathbf{z}|\mathbf{x})$. Thus, the optimization problem is defined as:

$$\arg\min_{\phi\in\Phi} D(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})), \tag{2.3}$$

where D is a divergence measure between the two distributions. A key point in VI is the choice of the divergence measure and the statistical model, which need to be flexible enough to approximate the posterior distribution but also simple enough for an efficient optimization. A common choice for the divergence measure is the Kullback-Leibler (KL) divergence:

Definition 23 (KL-Divergence)**.** *Given two probability distributions p and q, the **Kullback-Leibler (KL) divergence** is defined as*

$$D_{KL}(p||q) = \int p(z) \log\left(\frac{p(z)}{q(z)}\right) dz.$$

Proposition 11. *The KL divergence can equivalently be written as*

$$D_{KL}(p||q) = \mathbb{E}_{p(z)}\left[\log p(z) - \log q(z)\right].$$

Proof.

$$\begin{aligned} D_{KL}(p||q) &= \int p(\mathbf{z}) \log\left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right) d\mathbf{z} \\ &= \int p(\mathbf{z}) \left(\log(p(\mathbf{z})) - \log(q(\mathbf{z}))\right) d\mathbf{z} \\ &= \mathbb{E}_{p(\mathbf{z})}\left[\log p(\mathbf{z}) - \log q(\mathbf{z})\right]. \end{aligned}$$

□

Proposition 12. *The KL divergence satisfies the following properties:*

- $D_{KL}(p||q) \geq 0$.
- $D_{KL}(p||q) = 0$ *if and only if* $p = q$.
- $D_{KL}(p||q) \neq D_{KL}(q||p)$.

Proof.

- First we have that $\log(x) \leq x - 1$, which can be proved by taking the derivative of the function $f(x) = x - \log(x) - 1$ and showing that $x = 1$ is a global minimum of its domain and $f(1) = 0$. Therefore, we have that

$$\log\left(\frac{q(\mathbf{z})}{p(\mathbf{z})}\right) \leq \frac{q(\mathbf{z})}{p(\mathbf{z})} - 1.$$

Then, since $p(\mathbf{z})$ is a probability distribution, we can multiply by $p(\mathbf{z})$ and integrate over $\mathbf{z}$,

$$\int p(\mathbf{z})\log\left(\frac{q(\mathbf{z})}{p(\mathbf{z})}\right)d\mathbf{z} \leq \int p(\mathbf{z})\left(\frac{q(\mathbf{z})}{p(\mathbf{z})}-1\right)d\mathbf{z} = \int q(\mathbf{z})\,d\mathbf{z} - \int p(\mathbf{z})\,d\mathbf{z} = 1-1=0.$$

Therefore, rearranging terms,

$$\int p(\mathbf{z})\log\left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right)d\mathbf{z} = D_{KL}(p||q) \geq 0.$$

- If $p=q$, then $\log\left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right)=0$ and $D_{KL}(p||q)=0$. Conversely, if $D_{KL}(p||q)=0$, then $\log\left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right)=0$ and $p(\mathbf{z})=q(\mathbf{z})$.
- Assume that $D_{KL}(p||q)=D_{KL}(q||p)$. Then, we have that

$$\int p(\mathbf{z})\log\left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right)d\mathbf{z} = \int q(\mathbf{z})\log\left(\frac{q(\mathbf{z})}{p(\mathbf{z})}\right)d\mathbf{z}.$$

Then, we can rewrite the right-hand side as

$$\int q(\mathbf{z})\log\left(\frac{q(\mathbf{z})}{p(\mathbf{z})}\right)d\mathbf{z} = \int q(\mathbf{z})(\log(q(\mathbf{z}))-\log(p(\mathbf{z})))\,d\mathbf{z},$$

and the left-hand side as

$$\int p(\mathbf{z})\log\left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right)d\mathbf{z} = \int p(\mathbf{z})(\log(p(\mathbf{z}))-\log(q(\mathbf{z})))\,d\mathbf{z}.$$

Therefore, we have that

$$\int p(\mathbf{z})(\log(p(\mathbf{z}))-\log(q(\mathbf{z})))\,d\mathbf{z} = \int q(\mathbf{z})(\log(q(\mathbf{z}))-\log(p(\mathbf{z})))\,d\mathbf{z}.$$

Then,

$$\int (p(\mathbf{z})-q(\mathbf{z}))(\log(p(\mathbf{z}))-\log(q(\mathbf{z})))\,d\mathbf{z}=0.$$

Since p and q are density functions, we have that $p\equiv q$.

$\square$

Note that the KL divergence is not symmetric; therefore, the optimization problem in 2.3 is not symmetric either. Since, if p is the true distribution and q is the variational distribution, then,

$$D_{KL}(p||q) = \mathbb{E}_{p(\mathbf{z})}\left[\log\frac{p(\mathbf{z})}{q(\mathbf{z})}\right], \text{ and}$$
$$D_{KL}(q||p) = \mathbb{E}_{q(\mathbf{z})}\left[\log\frac{q(\mathbf{z})}{p(\mathbf{z})}\right].$$

Observe that the first equation tends to ∞ when $q(\mathbf{z})\to 0$ and $p(\mathbf{z})>0$, thus punishing the variational distribution for assigning low probability to regions where the true distribution is positive. On the other hand, the second equation forces $q(\mathbf{z})$ to be zero in regions where $p(\mathbf{z})=0$. Intuitively, this means that the former "stretches" q to cover over the entire distribution p, while the latter "shrinks" q to regions where p is positive.

The KL divergence turns out to be a good choice for the optimization problem since it can be computed analytically for many distributions, and can be related to the concept of entropy in information theory.

Definition 24 (Entropy)**.** *Given a probability distribution p, the **entropy** is defined as*

$$H(p) = -\int p(z)\log(p(z))\,dz.$$

Definition 25 (Cross Entropy)**.** *Given two probability distributions p and q, the **cross entropy** is defined as*

$$H(p,q) = -\int p(z)\log(q(z))\,dz.$$

Note that in the case of binary distributions, the cross entropy is equivalent to the binary cross entropy as defined in the previous section, 22. In information theoty, entropy is a measure of the uncertainty of a distribution, and the cross entropy is a measure of the uncertainty of one distribution relative to another. The KL divergence can be expressed in terms of the entropy as follows:

Proposition 13. *The KL divergence can be expressed in terms of the entropy as*

$$D_{KL}(p||q) = H(p,q) - H(p).$$

Proof.

$$\begin{aligned} D_{KL}(p||q) &= \int p(\mathbf{z})\log\left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right)d\mathbf{z} \\ &= \int p(\mathbf{z})\log(p(\mathbf{z}))\,d\mathbf{z} - \int p(\mathbf{z})\log(q(\mathbf{z}))\,d\mathbf{z} \\ &= -H(p) + H(p,q). \end{aligned}$$

□

Thus, minimizing the KL divergence is equivalent to maximizing the entropy of the posterior distribution, while minimizing the cross entropy between the true posterior and the variational distribution. Furthermore, there is a connection between the KL divergence and the maximum likelihood estimation, as we will see in the forthcoming sections.

Variational Objective

Given a joint distribution $p(\mathbf{x},\mathbf{z})$, whose intractable posterior $p(\mathbf{z}|\mathbf{x})$ we want to approximate with a variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$, where ϕ are the parameters of the model, the optimization problem in VI 2.3 with the KL divergence 23 is defined as

$$\arg\min_{\phi\in\Phi} D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \arg\min_{\phi\in\Phi} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z}|\mathbf{x})\right]. \tag{2.4}$$

To solve this optimization problem, we must first rewrite the optimization objective in terms of known tractable distributions. A common approach to tackle this issue consists in proving that minimizing the KL divergence is equivalent to maximizing the Evidence Lower Bound (ELBO) of the term $\log p(\mathbf{x})$. This result is a simple application of Jensen's inequality Jensen, 1906.

Theorem 6 (Jensen's Inequality)**.** *Let X be and integrable random variable. Given a convex function $f : \mathbb{R} \to \mathbb{R}$ such that $f \in \mathcal{C}^1(\mathbb{R})$ and $Y = f(X)$ is also integrable, we have that*

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)].$$

Proof. Recall that a function $f \in \mathcal{C}^1(\mathbb{R})$ is convex if and only if, for all $x_0 \in \mathbb{R}$, the graph of f lies above its tangent at point x_0:

$$f(x) \geq f(x_0) + f'(x_0)(x - x_0) \quad \forall x \in \mathbb{R}.$$

Then, by setting $x_0 = \mathbb{E}[X]$ and $x = X$, we have that

$$f(X) \geq f(\mathbb{E}[X]) + f'(\mathbb{E}[X])(X - \mathbb{E}[X]).$$

Taking the expectation of both sides, and by linearity of the expectation, we have that

$$\begin{aligned}
\mathbb{E}[f(X)] &\geq \mathbb{E}\big[f(\mathbb{E}[X]) + f'(\mathbb{E}[X])(X - \mathbb{E}[X])\big] \\
&= \mathbb{E}\big[f(\mathbb{E}[X])\big] + f'(\mathbb{E}[X])(\mathbb{E}[X] - \mathbb{E}\big[\mathbb{E}[X]\big]) \\
&= f(\mathbb{E}[X]) + f'(\mathbb{E}[X])(\mathbb{E}[X] - \mathbb{E}[X]) \\
&= f(\mathbb{E}[X]).
\end{aligned}$$

□

Following the same proof, we can show that, for a concave function, the inequality is reversed. Since the logarithm is a concave function in the positive real numbers, we can apply Jensen's inequality to bound the log-likelihood of the data distribution.

Corollary 1 (Evidence Lower Bound)**.** *Let $p(\boldsymbol{x}, \boldsymbol{z})$ be a joint distribution, and $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ be a variational distribution. Then, the log-likelihood of the data distribution can be bounded by $\mathcal{L}_\phi(\boldsymbol{x})$, which is called the* ***Evidence Lower Bound (ELBO)****:*

$$\log p(\boldsymbol{x}) \geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}, \boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})\right] \equiv \mathcal{L}_\phi(\boldsymbol{x}). \tag{2.5}$$

Proof.

$$\begin{aligned}
\log p(\mathbf{x}) = \log \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z} &= \log \int q_\phi(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \log \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
&\geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \equiv \mathcal{L}_\phi(\mathbf{x}). && \text{(Concave Jensen's Inequality)}
\end{aligned}$$

□

Futhermore, the ELBO of a MLE problem 16 is often directly denoted as $\mathcal{L}_\phi(\mathcal{D}) := \mathbb{E}_{p_\mathcal{D}}[\mathcal{L}_\phi(\mathbf{x})] \leq \mathbb{E}_{p_\mathcal{D}}[\log p(\mathbf{x})]$. Lastly, we can show the equivalence between the ELBO and the KL divergence, by expanding the terms in the ELBO and rearranging them.

Theorem 7. *Minimizing the KL divergence is equivalent to maximizing the Evidence Lower Bound. Furthermore, for any $\boldsymbol{x} \in \mathcal{D}$, we have that*

$$\log p(\boldsymbol{x}) = \mathcal{L}_\phi(\boldsymbol{x}) + D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})).$$

Proof. First, observe that $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x})] = \log p(\mathbf{x})$, because $\int q_\phi(\mathbf{z}|\mathbf{x})\, d\mathbf{z} = 1, \forall \phi \in \Phi$,

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x})] = \int q_\phi(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x})\, d\mathbf{z} = \log p(\mathbf{x}) \int q_\phi(\mathbf{z}|\mathbf{x})\, d\mathbf{z} = \log p(\mathbf{x}).$$

Then, we have that

$$\begin{aligned}
\log p(\mathbf{x}) - \mathcal{L}_\phi(\mathbf{x}) &= \log p(\mathbf{x}) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}) - \log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
&= -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} - \log p(\mathbf{x})\right] \\
&= -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})} - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{z}|\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z}|\mathbf{x})\right] \equiv D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})).
\end{aligned}$$

This provides the equivalence between the ELBO and the KL divergence we were looking for. Now, we can rewrite the optimization problem in VI as

$$\arg\min_{\phi\in\Phi} D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \arg\min_{\phi\in\Phi} \mathbb{E}_{p_\mathcal{D}}\left(\log p(\mathbf{x}) - \mathcal{L}_\phi(\mathbf{x})\right).$$

Since the log-likelihood of the data distribution does not depend on ϕ, we can ignore the term and conclude the proof:

$$\arg\min_{\phi\in\Phi} D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \arg\max_{\phi\in\Phi} \mathcal{L}_\phi(\mathcal{D}).$$

□

Note that due to the non-negativity of the KL divergence, the above theorem implies that $\mathcal{L}_\phi(\mathbf{x})$ is a lower bound on $\log p(\mathbf{x})$, without the need of Jensen's inequality. By rewritting the ELBO, we can get to another form that is more convenient for optimization, that is known as the Variational Lower Bound (VLB).

Corollary 2 (Variational Lower Bound)**.** *For $\boldsymbol{x} \in \mathcal{D}$, the Evidence Lower Bound can be written as*

$$\mathcal{L}_\phi(\boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{x}|\boldsymbol{z})] - D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})).$$

which is known as the ***Variational Lower Bound (VLB)****.*

Proof.

$$\begin{aligned}
\mathcal{L}_\phi(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})).
\end{aligned}$$

□

Amortized Variational Inference

When solving the VI problem, note that each data point is governed by a separate variational distribution, that is a latent variable z_i with variational parameters θ_i. For a large dataset, this can be computationally expensive. To address this issue, amortized variational inference uses a neural network as inference model. Thus, instead of optimizing the variational parameters for each data point, we optimize the parameters of the neural network, which are shared across all data points. Therefore, we will refer to the parameters of the variational distribution and the parameters of the neural network interchangeably.

Classical mean field variational inference has historically played an important role, however it is limited in multiple ways when it comes to modern application. Despite Amortized VI, one of the main challenges is to better scale it to growing datasets. To address this issue, several lines of research emerged, such as stochastic variational inference, that uses stochastic optimization such as SGD to scale the method to large datasets, and more complex variational distributions such as structured variational inference, that allows to capture the dependencies between the latent variables. This improvements led to the adoption of neural networks and machine learning techniques to solve the optimization problem in VI, which formalizes the development of Variational Autoencoders.

Furthermore, emerging lines of research are exploring the use of alternative divergence measures, that can be more robust to find the optimal parameters of the model. Some of the most relevant alternatives (Zhang et al., 2019) are the α-divergence, which is a family of divergence measures that generalizes the KL divergence, and the Stein Discrepancy, which is a measure of discrepancy between two probability distributions that is based on the Stein operator. Although, the choice of the divergence measure is still an open question in the field, in the next section we will focus in the classical introduction of Variational Autoencoders, which uses the KL divergence.

2.2.2 Variational Autoencoders

We have seen how Variational Inference is used to approximate intractable posterior distributions of a latent variable model. In the context of Machine Learning, the data distribution is often approximated by a deep latent variable model. Thus, given the parameters $\theta \in \Theta$, the model is defined as $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})$, where the latent variables are unobserved. The goal is to find the optimal parameters θ^* that maximize the likelihood of the data distribution. We have seen that the optimization problem is intractable, and the ELBO provides a lower bound that can be maximized using the KL divergence and an approximation $q_\phi(\mathbf{z}|\mathbf{x})$ of the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$. A Variational Autoencoder provides a computationally efficent way for optimizing DLVMs jointly with the corresponding inference model. For now, we will focus on the classical introduction of VAEs, and in the next section we will see how they are used in the context of discrete latent variables.

Definition 26 (Variational Autoencoder)**.** *A **Variational Autoencoder** is a tuple $(\mathcal{D}, q_\phi, p_\theta)$, where $q_\theta : \mathcal{D} \to P(\mathcal{Z})$ is the inference model or **recognition model**, and $p_\theta : \mathcal{Z} \to \mathcal{D}$ is the **generative model**.*

Following the amortized variational inference approach, in a VAE, the recognition model q_θ is a neural network that takes the data $\mathbf{x}$ as input and outputs the parameters of the variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$. The generative model p_θ is also a neural network that takes the latent variables $\mathbf{z}$ as input and outputs the parameters of the data distribution $p_\theta(\mathbf{x}|\mathbf{z})$.

ELBO for Variational Autoencoders

Substituting $p(\mathbf{x}, \mathbf{z})$ by $p_\theta(\mathbf{x}, \mathbf{z})$ in theorem 7, we have that the ELBO can be written as

$$\mathcal{L}_{\phi,\theta}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x}). \tag{2.6}$$

By maximizing this equation over θ and ϕ, we approximately maximize the log-likelihood of the data distribution, as well as minimize the KL divergence of the posteriors. The optimization problem in VAEs is then defined as

$$\arg\max_{\theta,\phi} \mathcal{L}_{\phi,\theta}(\mathbf{x}). \tag{2.7}$$

This optimization problem is usually solved by means of a gradient descent method such as SGD. However, computing the gradients of the ELBO with respect to the parameters of the model, $\nabla_{\theta,\phi}\mathcal{L}_{\phi,\theta}(\mathbf{x})$, is usually intractable. And Monte Carlo methods are not applicable since

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_{\theta,\phi} \log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \neq \nabla_{\theta,\phi}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right].$$

VAEs solve this issue by using the *reparameterization trick.*

Reparameterization Trick

Let g be a differentiable and invertible function, and consider a sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Then, we can rewrite the sample as $\mathbf{z} = g(\epsilon, \phi, \mathbf{x})$, where $\epsilon \sim p(\epsilon)$ is a random variable on $\mathcal{E} \subset \mathbb{R}^D$ with a known distribution independent of ϕ and $\mathbf{x}$. Given such a change of variables, we can use the LOTUS Theorem 4, and the equation of change of variables 1.1 to rewrite the expectations in terms of the random variable ϵ,

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \mathbb{E}_{p(\epsilon)}[f(g(\epsilon, \phi, \mathbf{x}))]. \tag{2.8}$$

Another tool that is useful for computing the gradients of the ELBO is the differentiation under the integral sign theorem, which allows to interchange the differentiation and the integral sign under certain conditions. The proof of this theorem is based on the dominated convergence theorem, we will refer to the book Folland, 2013 for a detailed proof, and state the theorem here.

Theorem 8 (Differentiation under the integral sign)**.** *Let U be an open subset of $\mathbb{R}$ and let E be a measure space. Let $f : U \times E \to \mathbb{R}$ be a function such that*

1. *For each $x \in U$, $f(x, \cdot)$ is integrable on E.*
2. *For each $y \in E$, $f(\cdot, y)$ is differentiable on U.*
3. *There exists an integrable function $g : E \to \mathbb{R}$ such that*

$$\left|\frac{\partial f}{\partial x}(x, y)\right| \leq g(y)$$

for all $x \in U$ and $y \in E$.

Then the function $F : U \to \mathbb{R}$ defined by

$$F(x) = \int_E f(x, y)\, dy$$

is differentiable on U, and

$$F'(x) = \int_E \frac{\partial f}{\partial x}(x,y)\,dy.$$

If U is a subset of $\mathbb{R}^n$ and E is a subset of $\mathbb{R}^m$, by applying the theorem to each variable, the result can be generalized to the case where $f : U \times E \to \mathbb{R}^k$ and $F : U \to \mathbb{R}^k$. In our case, substituting $f(x,y)$ from the theorem by $f(\phi,\epsilon) = p(\epsilon)\left[\log p_\theta(\mathbf{x}, g(\epsilon,\phi,\mathbf{x})) - \log q_\phi(g(\epsilon,\phi,\mathbf{x})|\mathbf{x})\right]$, we can compute the gradients of the ELBO as follows,

$$\begin{aligned}
\nabla_\phi \mathcal{L}_{\phi,\theta}(\mathbf{x}) &= \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z})\right] = \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] && (2.9)\\
&= \nabla_\phi \mathbb{E}_{p(\epsilon)}\left[\log p_\theta(\mathbf{x}, g(\epsilon,\phi,\mathbf{x})) - \log q_\phi(g(\epsilon,\phi,\mathbf{x})|\mathbf{x})\right] && (2.10)\\
&= \nabla_\phi \int_{\mathcal{E}} p(\epsilon)\left(\log p_\theta(\mathbf{x}, g(\epsilon,\phi,\mathbf{x})) - \log q_\phi(g(\epsilon,\phi,\mathbf{x})|\mathbf{x})\right) d\epsilon && (2.11)\\
&= \int_{\mathcal{E}} p(\epsilon)\nabla_\phi \left[\log p_\theta(\mathbf{x}, g(\epsilon,\phi,\mathbf{x})) - \log q_\phi(g(\epsilon,\phi,\mathbf{x})|\mathbf{x})\right] d\epsilon && (2.12)\\
&= \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(g(\epsilon,\phi,\mathbf{x}))\right]. && (2.13)
\end{aligned}$$

Similarly,

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\phi,\theta}(\mathbf{x}) &= \nabla_\theta \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z})\right] = \nabla_\theta \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] && (2.14)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_\theta\left(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right)\right]. && (2.15)
\end{aligned}$$

With this setting we can estimate the gradients of the ELBO by a Monte Carlo estimator. In the case of 2.9, we obtain the following estimator by sampling $\epsilon \sim p(\epsilon)$,

$$\begin{aligned}
\nabla_\phi \mathcal{L}_{\phi,\theta}(\mathbf{x}) &= \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(g(\epsilon,\phi,\mathbf{x}))\right] && (2.16)\\
&\approx \nabla_\phi f(g(\epsilon,\phi,\mathbf{x})) \equiv \nabla_\phi \tilde{\mathcal{L}}_{\phi,\theta}(\mathbf{x}), && (2.17)
\end{aligned}$$

and similarly for 2.14, sampling $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$,

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\phi,\theta}(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_\theta\left(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right)\right] && (2.18)\\
&\approx \nabla_\theta\left(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right) = \nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z}) \equiv \nabla_\theta \tilde{\mathcal{L}}_{\phi,\theta}(\mathbf{x}). && (2.19)
\end{aligned}$$

Thus,

$$\begin{aligned}
\nabla_{\theta,\phi} \tilde{\mathcal{L}}_{\phi,\theta}(\mathbf{x}) &= \nabla_{\theta,\phi}\left(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right)\\
&= \left(\nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z}), \nabla_\phi(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\right) = \left(\nabla_\theta \tilde{\mathcal{L}}_{\phi,\theta}(\mathbf{x}), \nabla_\phi \tilde{\mathcal{L}}_{\phi,\theta}(\mathbf{x})\right).
\end{aligned}$$

Note that given the change of variables, a Monte Carlo estimator of a point-wise evaluation of the ELBO, can also be obtained by sampling $\epsilon \sim p(\epsilon)$ and $\mathbf{z} = g(\epsilon,\phi,\mathbf{x})$, then the estimator is

$$\tilde{\mathcal{L}}_{\phi,\theta}(\mathbf{x}) = \log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}).$$

Theorem 9. *When averaged over the variables $\epsilon \sim p(\epsilon)$, the Monte Carlo estimator of the gradients of the ELBO is equal to the gradients of the point-wise evaluation of the ELBO.*

Proof.

$$\begin{aligned}\mathbb{E}_{p(\epsilon)}\left[\nabla_{\theta,\phi}\tilde{\mathcal{L}}_{\phi,\theta}(\mathbf{x})\right] &= \mathbb{E}_{p(\epsilon)}\left[\nabla_{\theta,\phi}\left(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right)\right] \\ &= \nabla_{\theta,\phi}\mathbb{E}_{p(\epsilon)}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\ &= \nabla_{\theta,\phi}\mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(g(\epsilon,\phi,\mathbf{x}))\right] \\ &= \nabla_{\theta,\phi}\mathcal{L}_{\phi,\theta}(\mathbf{x}).\end{aligned}$$

□

The theorem makes it possible to estimate the gradients of the ELBO by a simple sampling procedure of $\epsilon \sim p(\epsilon)$. Finally, the computation of this estimator requires the computation of $\log q_\phi(\mathbf{z}|\mathbf{x})$, which can be computed from the change of variables g, since it is differentiable and invertible. Note that applying equation 1.1, $q_\phi(\mathbf{z}|\mathbf{x}) = \frac{p(\epsilon)}{J_g}$, where $J_g = |\det(\frac{\partial \mathbf{z}}{\partial \epsilon})| = |\det(\frac{\partial g(\epsilon,\phi,\mathbf{x})}{\partial \epsilon})|$ is the Jacobian of the change of variables. Therefore,

$$\log q_\phi(\mathbf{z}|\mathbf{x}) = \log p(\epsilon) - \log J_g.$$

Ideally, we would like to find a function g such that the Jacobian is simple to compute, as well as it leads to a simple and known distribution $p(\epsilon)$ where we can sample from. With this setting, we can follow algorithm 1 and train the VAE model by means of stochastic gradient descent.

Algorithm 1 Stochastic optimization of the ELBO for VAEs

```
 1: Input:
 2:   $\mathcal{D}$: Dataset
 3:   $q_\phi(z|x)$: Inference model
 4:   $p_\theta(x,z)$: Generative model
 5: Output:
 6:   $\theta, \phi$: Learned parameters
 7: Algorithm:
 8: $(\theta, \phi) \leftarrow$ Initialize parameters
 9: while SGD not converged do
10:     $M \sim \mathcal{D}$                                ▷ Random minibatch of data
11:     $\epsilon \sim p(\epsilon)$                        ▷ Random noise for every datapoint in $M$
12:     Compute $\tilde{L}_{\theta,\phi}$ and its gradients $\nabla_{\theta,\phi}\tilde{L}_{\theta,\phi}$ for each datapoint in $M$ and each noise sample $\epsilon$
13:     Update $\theta$ and $\phi$ using SGD optimizer
14: end while
```

Factorized Gaussian Variational Autoencoders

The classical VAE introduced in Kingma and Welling, 2014, uses an amortized mean-field Gaussian variational distribution as recognition model,

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^{d} q_{\phi_i}(z_i|\mathbf{x}) = \prod_{i=1}^{d} \mathcal{N}(z_i; \mu_i, \sigma_i^2) = \mathcal{N}(\mathbf{z}; \mu, \sigma),$$

where $\mu_i(\mathbf{x})$ and $\sigma_i^2(\mathbf{x})$ are the outputs of a neural network E_ϕ, $(\mu, \sigma^2) = E_\phi(\mathbf{x})$. Consider $\mathbf{z} = g(\epsilon, \phi, \mathbf{x}) = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$, and $\odot$ denotes the element-wise product.

Note that g is a linear function, thus it is differentiable and invertible. Its Jacobian is given by

$$J_g = \left|\det\left(\frac{\partial g(\epsilon, \phi, \mathbf{x})}{\partial \epsilon}\right)\right| = \left|\det\left(\frac{\partial(\mu(\mathbf{x}) + \sigma(\mathbf{x}) \odot \epsilon)}{\partial \epsilon}\right)\right| = |\det(\sigma(\mathbf{x}))| = \prod_{i=1}^{d} \sigma_i(\mathbf{x}).$$

Therefore, the logarithm of the variational distribution is

$$\log q_\phi(\mathbf{z}|\mathbf{x}) = \log p(\epsilon) - \log J_g = \sum_{i=1}^{d} \left[\log \mathcal{N}(\epsilon_i|0,1) - \log \sigma_i(\mathbf{x})\right].$$

The generative model $p_\theta(\mathbf{x}|\mathbf{z})$ is usually a product of Gaussian distributions whose parameters are modelled by a neural network D_θ, that is,

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{j=1}^{|D|} p_{\theta_j}(x_j|\mathbf{z}) = \prod_{j=1}^{|D|} \mathcal{N}(x_j|\mu_j, \sigma_j^2),$$

where $\mu_j(\mathbf{z})$ and $\sigma_j^2(\mathbf{z})$ are the outputs of a neural network, $(\mu, \sigma^2) = D_\theta(\mathbf{z})$. In the special case where $\sigma = I$, we obtain a relationship with the reconstruction loss of an autoencoder that uses MSE as the loss function, given by

$$\begin{aligned} -\log p_\theta(\mathbf{x}|\mathbf{z}) &= -\log \mathcal{N}(\mathbf{x}|\mu(\mathbf{z}), I) = -\log\left(\frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu(\mathbf{z}))^T(\mathbf{x} - \mu(\mathbf{z}))\right)\right) \\ &= \frac{1}{2}||\mathbf{x} - \mu(\mathbf{z})||^2 + C = \text{MSE}(\mathbf{x}, \mu(\mathbf{z})) + C, \end{aligned}$$

where C is a constant that does not depend on θ nor ϕ. This is the main reason why VAEs are often seen as regularized autoencoders and the training objective is often implemented as a combination of the MSE reconstruction loss and the KL divergence.

The neural networks E_ϕ and D_θ are trained jointly by maximizing the ELBO following the algorithm 1. In relation to autoencoders, both networks are often seen as the encoder and decoder of the VAE, respectively.

VAEs offer a large flexibility in the choice of the generative model, and the recognition model, which allows to model complex data distributions both in discrete and continuous latent spaces. Here we have considered the simplest case of a Gaussian VAE; however, more complex models can be built by using different distributions. In the next section, we will introduce the VAE variant for discrete latent spaces, and later introduce the Bernoulli VAE, which is one of the key components of the thesis.

2.3 Discrete Variational Autoencoders

We have seen how Variational Autoencoders can be used to model complex data distributions by means of a neural network. The latent space of a VAE is usually continuous, which allows us to generate highly diverse and feasible samples. However, continuous latent spaces suffer from large variance, which makes them hard to train and unstable. Furthermore, a continuous latent space offers a continuous representation of the data which can be hard to interpret. In contrast, in a discrete latent space $\mathcal{Z}$ with a finite number of categories $|\mathcal{Z}| = K$, we have a precise notion of compression, since the bits of information represented by $\mathcal{Z}$ is upper bounded by $\log_2(K)$.

Definition 27 (Shanon Entropy)**.** *Let p be a discrete probability distribution over a set $\mathcal{Z}$. The* ***Shanon Entropy*** *of p is defined as*

$$H_2(p) = -\sum_{z\in\mathcal{Z}} p(z)\log_2 p(z) = -\mathbb{E}_{p(z)}[\log_2 p(z)].$$

In the field of information theory, as the regular entropy, the Shanon Entropy is a measure of the uncertainty of a random variable, and since it uses the base 2 logarithm the measure is given in bits. This is often referred to as the *information content* of the random variable.

Proposition 14. *The Shanon Entropy of a discrete random variable z with K categories is upper bounded by $\log_2(K)$,*

$$H(p) \leq \log_2(K).$$

Proof. Using Jensen's inequality, we have that

$$\begin{aligned} H(p) &= -\mathbb{E}_{p(z)}[\log_2 p(z)] = \mathbb{E}_{p(z)}[\log_2 \frac{1}{p(z)}] \\ &\leq \log_2 \mathbb{E}_{p(z)}\left[\frac{1}{p(z)}\right] = \log_2 K. \end{aligned}$$

□

Assuming a discrete latent space, the model can be trained as a regular VAE, by assuming a prior discrete distribution $p(\mathbf{z})$ and maximizing the ELBO. Recall from 2 that the ELBO for VAEs can be given by

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

Observe that the KL divergence is now between two discrete distributions, which can be computed as

Definition 28 (Discrete KL-Divergence)**.** *Let p and q be two discrete distributions over a set $\mathcal{Z}$. The* ***Discrete KL-Divergence*** *between p and q is defined as*

$$D_{KL}(q||p) = \sum_{z\in\mathcal{Z}} q(z)\log\frac{q(z)}{p(z)}.$$

It is easy to see the equivalence between continuous and discrete KL divergences, by considering the discrete distributions as Dirac deltas.

Usually, the prior distribution is chosen to be a uniform distribution over the categories, $p(\mathbf{z}) = \text{Uniform}(\mathcal{Z})$. In this case, the KL divergence can be computed as

$$\begin{aligned} D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) &= \sum_{z\in\mathcal{Z}} q_\phi(z|\mathbf{x})\log\frac{q_\phi(z|\mathbf{x})}{p(z)} \\ &= \sum_{z\in\mathcal{Z}} q_\phi(z|\mathbf{x})\log\frac{q_\phi(z|\mathbf{x})}{1/K} \\ &= \log K + \sum_{z\in\mathcal{Z}} q_\phi(z|\mathbf{x})\log q_\phi(z|\mathbf{x}) \\ &= \log K - H(q_\phi(\mathbf{z}|\mathbf{x})). \end{aligned}$$

Therefore, the optimization problem can be rewritten as

$$\arg\max_{\theta,\phi} \mathcal{L}_{\phi,\theta}(\mathbf{x}) = \arg\max_{\theta,\phi} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - \log K + H(q_\phi(\mathbf{z}|\mathbf{x})) \tag{2.20}$$

$$= \arg\max_{\theta,\phi} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log q_\phi(\mathbf{z}|\mathbf{x})] . \tag{2.21}$$

The problem arises when computing the gradients of the ELBO, since the discrete nature of the latent space makes the optimization problem non-differentiable. There are several techniques that tackle this issue. A heuristic approach is to use the Straight-Through Estimator, which allows to backpropagate the gradients through the discrete latent space. Another approach is to use the Gumbel-Softmax trick, which allows to sample from a categorical distribution by means of a continuous relaxation.

2.3.1 The Straight-Through Estimator

Proposed by Bengio, 2013, the Straight-Through Estimator (STE) is a trick for differentiating through thresholding operations. Given a step function $s : \mathbb{R} \to \mathbb{R}$, e.g., $s(x) = 1$ if $x > 0$ and $s(x) = 0$ otherwise, and considering $f = g \circ s \circ h$, where g and h are differentiable functions, we have that $\nabla_x f(x) = 0$ since $\nabla_x s \equiv 0$,

$$\nabla_x f(x) = \nabla_x g(s(h(x))) = \nabla_x g(s(h(x))) \nabla_x s(h(x)) \nabla_x h(x) = 0.$$

The STE allows to bypass this issue by replacing $\nabla_x s$ with the constant function 1, which allows to backpropagate the gradients through the thresholding operation. Thus,

$$\tilde{\nabla}_x f(x) = \nabla_x g(s(h(x))) \nabla_x h(x).$$

This can be seen as copying the gradients from the output of the step function to its input. With this, the gradients of the ELBO can be computed, and the model can be trained by means of stochastic gradient descent. Although the STE is not theoretically justified, it has been shown to work well in practice, and it is widely used in the context of discrete VAEs (Oord, Vinyals, and Kavukcuoglu, 2017). However, in complex cases, it can lead to biased gradients and fail to be a good estimator for the gradient (Liu and Mattina, 2019).

Vector Quantized Variational Autoencoders

Introduced by Oord, Vinyals, and Kavukcuoglu, 2017, Vector Quantized Variational Autoencoders (VQ-VAEs) are a type of VAE that uses Vector Quantization (VQ) to discretize the latent space. Then assuming a discrete prior on the latent space, the model is able to learn a discrete representation of the data. The VQ process is done by projecting the continuous latent variables to the nearest vector in a *codebook*, which is a set of vectors of the same dimension as the continuous latent space.

Definition 29 (Vector Quantization)**.** *Let $\mathcal{D}$ be a dataset, let $\boldsymbol{z} \in \mathbb{R}^D$ be a continuous latent representation of $\boldsymbol{x} \in \mathcal{D}$ and $\mathcal{V}_K = \{e_i\}_{i \in [1,\dots,K]} \subset \mathbb{R}^D$ be a set of K vectors of dimension D. The **Vector Quantization** of a vector $\boldsymbol{x} \in \mathcal{D}$ to the codebook $\mathcal{V}_K$ is defined as*

$$\boldsymbol{z}_q(\boldsymbol{x}) = \min_{e \in \mathcal{V}_K} ||\boldsymbol{z} - e||_2^2.$$

The VQ is incorporated into the VAE by adding the quantization after the continuous latent representation computed by the encoder $\mathrm{NN}_\phi^{\mathrm{enc}} : \mathcal{D} \to \mathbb{R}^D$; see Figure 2.1. This

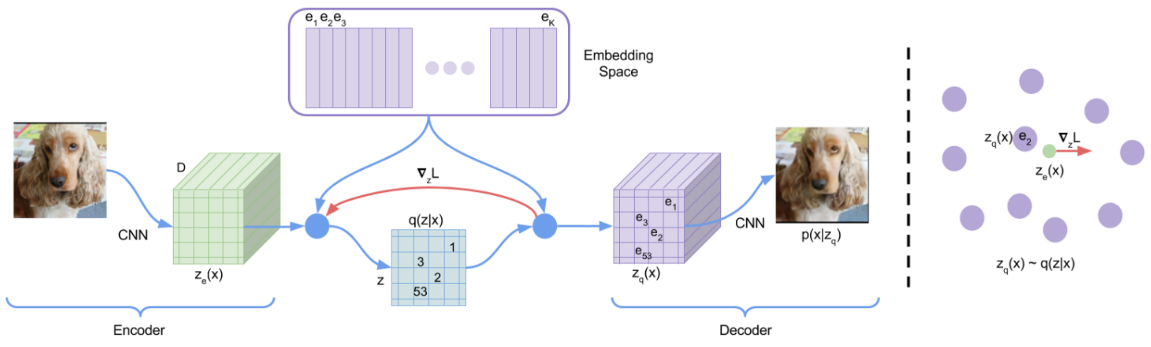


FIGURE 2.1: VQVAE architecture. A sample is encoded into a sequence of tokens, which are then decoded back to the original.

procedure discretizes the latent space into a K categorical discrete distribution defined as

$$q_\phi(z = k|\mathbf{x}) = \text{one\_hot}(\mathbf{z}_q(\mathbf{x}); k) = \begin{cases} 1 & \text{if } \mathbf{z}_q(\mathbf{x}) = e_k \\ 0 & \text{otherwise.} \end{cases} \tag{2.22}$$

The classic VQ-VAE model defines the prior distribution as a uniform distribution over the codebook, $p(\mathbf{z}) = \text{Uniform}(\mathcal{V}_K)$. Thus, the KL divergence can be computed as

$$\begin{aligned} D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) &= \sum_{k=1}^{K} q_\phi(z = k|\mathbf{x}) \log \frac{q_\phi(z = k|\mathbf{x})}{p(z = k)} \\ &= \log \frac{1}{1/K} = \log K. \end{aligned}$$

This constant term can be ignored from the optimization problem, and the ELBO can be maximized by

$$\arg\max_{\theta,\phi} \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \arg\max_{\theta,\phi} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z})\right].$$

This can be further simplified by computing the expectation over the discrete distribution,

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] = \sum_{k=1}^{K} q_\phi(z = k|\mathbf{x}) \log p_\theta(\mathbf{x}|z = k) = \log p_\theta(\mathbf{x}|\mathbf{z}_q(\mathbf{x})).$$

Therefore,

$$\arg\max_{\theta,\phi} \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \arg\max_{\theta,\phi} \log p_\theta(\mathbf{x}|\mathbf{z}_q(\mathbf{x})).$$

In this setting, STE is usually implemented using the stop gradient operator, which is the identity during the forward pass and has partial derivatives of zero during the backward pass to avoid updating the gradient.

Definition 30 (Stop Gradient Operator)**.** *The* ***Stop Gradient Operator*** *is an operator* $sg : \mathbb{R}^D \to \mathbb{R}^D$ *defined as the identity function*

$$sg[\boldsymbol{z}] = \boldsymbol{z},$$

and such that its partial derivatives are zero, i.e., $\frac{\partial sg[\boldsymbol{z}]}{\partial \boldsymbol{z}} = 0$.

Thus, $\tilde{z}_q(\mathbf{x}) = \text{sg}[\mathbf{z}_q(\mathbf{x})] + \mathbf{z} - \text{sg}[\mathbf{z}]$. Note that, with this operation, the codebook vectors $e \in \mathcal{V}_K$ receive no gradients, requiring only to add an extra term to the loss function, namely the VQ objective.

Definition 31 (VQ Objective)**.** *The **VQ Objective** is a term added to the loss function of a VQ-VAE that penalizes the distance between the encoder output and the codebook vectors. It is defined as*

$$VQ = ||\boldsymbol{z} - e||_2^2,$$

where e is the codebook vector that is closest to the encoder output.

Finally, to avoid the encoder output drifting away from the codebook, the authors propose to split the VQ objective into two terms, the codebook loss and the commitment loss. This is $||\text{sg}[\mathbf{z}] - e||_2^2$ and $||\mathbf{z} - \text{sg}[e]||_2^2$, respectively. Then considering the simplification of the ELBO, and the addition of the VQ objective, the training objective of the VQ-VAE is given by

$$\arg\min_{\theta,\phi} L_{\text{VQVAE}} = -\log p_\theta(\mathbf{x}|\mathbf{z}_q(\mathbf{x})) + ||\text{sg}[\mathbf{z}] - e||_2^2 + \beta||\mathbf{z} - \text{sg}[e]||_2^2, \tag{2.23}$$

where β is a hyperparameter that controls the strength of the commitment loss. Heuristically, with $\beta < 1$, the vectors of the codebook learn faster than the encoder, which empirically has shown to be beneficial for the training of the model.

2.3.2 Gumbel Softmax

The Gumbel Softmax estimation, proposed by Jang, Gu, and Poole, 2017 and Maddison, Mnih, and Teh, 2017, consists of a continuous relaxation of the categorical distribution by "reparameterizing" the problem through a Gumbel distribution, which is a continuous distribution over the simplex that can approximate a categorical distribution.

Definition 32 (Gumbel Distribution)**.** *Let $\epsilon \in \mathbb{R}$ be a random variable. Then ϵ is distributed according to a **Gumbel Distribution**, i.e., $\epsilon \sim Gumbel(\mu, \beta)$, with parameters $\mu \in \mathbb{R}$ and $\beta \in \mathbb{R}^+$, if its probability density function is given by*

$$p(\epsilon) = \frac{1}{\beta} \exp\left[\frac{\epsilon - \mu}{\beta} - \exp\left[\frac{\epsilon - \mu}{\beta}\right]\right].$$

The standard Gumbel distribution is defined as Gumbel$(0, 1)$.

One of the main properties of the Gumbel distribution is its connection to Uniform distributions, which allows for an easy sampling procedure. Given a random variable $u \sim \text{Uniform}(0, 1)$, $\text{Gumbel}(\mu, \beta)$ can be sampled as $\epsilon = g(u) = \mu - \beta \log(-\log(u))$. Note that if $u \in (0, 1)$ then g is well defined; in fact it is differentiable and invertible, and

$$\begin{aligned}\int_{\mathbb{R}} p(\epsilon) d\epsilon &= \int_{\mathbb{R}} \frac{1}{\beta} \exp\left[\frac{\epsilon - \mu}{\beta} - \exp\left[\frac{\epsilon - \mu}{\beta}\right]\right] d\epsilon \\ &= \int_0^1 \frac{1}{\beta} \exp\left[\frac{g(u) - \mu}{\beta} - \exp\left[\frac{g(u) - \mu}{\beta}\right]\right] g'(u)\, du = \int_0^1 du\end{aligned}$$

is a change of variables between $\text{Gumbel}(\mu, \beta)$ and $\text{Uniform}(0, 1)$.

Given a discrete random variable $z \in \{1, \ldots, K\}$, defined by $p(z = k) = \theta_k$, with $\sum_{k=1}^K \theta_k = 1$, we can denote z as a one-hot encoded vector $\text{one\_hot}(z; k) \in \{0, 1\}^K$, where $\text{one\_hot}(z; k)_k = 1$ if $z = k$ and $\text{one\_hot}(z; k)_k = 0$ otherwise. We will abuse of notation and consider z as a one-hot encoded vector, and denote $z \sim \text{Dis}(\theta)$, where $\theta = (\theta_1, \ldots, \theta_K)$. The standard Gumbel distribution can be used to sample from a categorical distribution by means of the Gumbel-Max trick, which consists of sampling from the Gumbel distribution and taking the argmax of the samples. Refer to Gumbel's original paper for a detailed proof of this result Gumbel, 1954.

Proposition 15 (Gumbel-Max Trick)**.** *If $z \sim Dis(\theta)$ is a discrete random variable, then z can be sampled by means of the Gumbel-Max trick as*

$$k = \arg\max_k \left[\log \theta_k + \epsilon_k\right],$$

where $\epsilon_k \sim Gumbel(0,1)$. Then $z_k = 1$ and $z_{k'} = 0$ for $k' \neq k$.

Now, the Gumbel Softmax trick consists of a continuous relaxation of the arg max operation by means of the softmax function. Given $g_1, \ldots, g_K \sim \text{Gumbel}(0,1)$, we can generate K-dimensional vectors $z \in \mathbb{R}^K$ by means of the softmax function,

$$z_i = \frac{\exp[(g_i + \log \theta_i)/\tau]}{\sum_{j=1}^K \exp[(g_j + \log \theta_j)/\tau]}, \tag{2.24}$$

where τ is a temperature parameter that controls the smoothness of the distribution. Observe that, as $\tau \to 0$, the distribution becomes more peaked around the argmax, and as $\tau \to \infty$, the distribution becomes uniform.

With this reparameterization, the ELBO for the discrete case 2.20 can be optimized by SGD, and considering

$$\mathbb{E}_{q_\phi(z|\mathbf{x})}\left[p_\theta(\mathbf{x}|z)\right] = \mathbb{E}_{q_\phi(z|\mathbf{x})}\left[f(z)\right] = \mathbb{E}_{g_i \sim \text{Gumbel}(0,1)}\left[f\left(\frac{\exp[(g_i + \log \theta_i)/\tau]}{\sum_{j=1}^K \exp[(g_j + \log \theta_j)/\tau]}\right)\right].$$

2.3.3 Binary Representations

Given a data point $\mathbf{x} \in \mathcal{D} \subset \mathbb{R}^D$, the goal is to learn a bidirectional stochastic mapping between itself and its binary representation, that is an Bernoulli Variational Autoencoder (BVAE). Consider $\mathbf{x} \in \mathcal{D}$. The first step is to train an image encoder E_θ, to encode the image into a real-valued latent tensor $\mathbf{y} = E_\theta(\mathbf{x}) \in \mathbb{R}^d$, where d is the latent dimension. Then, before quantization, a sigmoid σ is applied to normalize the output of the encoder to the range $[0,1]$ to makes sure it is a probability.

The quantization into binary vectors can be done in two ways, deterministically or stochastically. In the former, the encoder output is thresholded

$$\mathbf{z} = (\sigma(E_\theta(\mathbf{x})) > 0.5).$$

In the latter, we sample from a Bernoulli distribution and get

$$\mathbf{z} \sim \mathcal{B}(\mathbf{z}; \sigma(E_\theta(\mathbf{x}))).$$

Note that in both cases the method does not permit gradient propagation so the STE technique is used to copy the gradients from the decoder input to the encoder output,

$$\tilde{\mathbf{z}} = \text{STE}(\mathbf{z}) = \text{sg}[\mathbf{z}] + \mathbf{y} - \text{sg}[\mathbf{y}].$$

Finally, the decoder $D_{\theta'}$ is given $\tilde{\mathbf{z}}$, and is trained to reconstruct the original sample, $\hat{\mathbf{x}} = D_{\theta'}(\tilde{\mathbf{z}})$.

With this setting the training objective is identical as a VAE with stop-through estimator. Therefore, we aim to maximize the ELBO, 2.20.

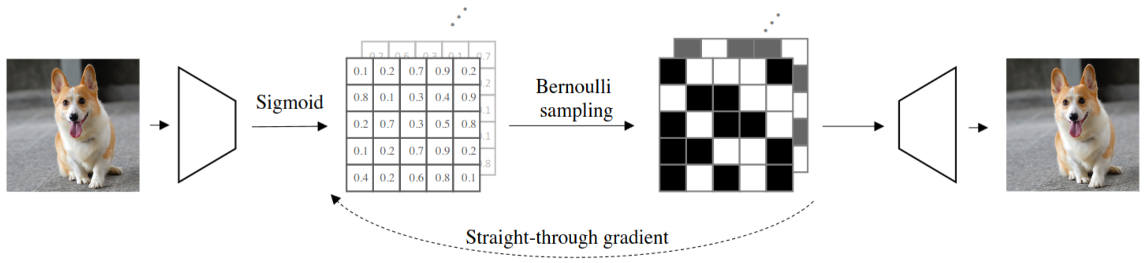


FIGURE 2.2: Binary VAE. The encoder maps the image to a latent space where Bernoulli vectors are sampled. Then the decoder reconstructs the image from the binary representation.

Gumbel Softmax for Bernoulli VAEs

Alternatively, the BVAE can be constructed by means of the Gumbel softmax trick. In this case the generative model is a product of Bernoulli distributions,

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{j=1}^{|D|} p_{\theta_j}(x_j|\mathbf{z}) = \prod_{j=1}^{|D|} \mathcal{B}(x_j; \theta_j(\mathbf{z})),$$

where $\theta_j(\mathbf{z})$ is the output of a neural network. The recognition model is a product of Bernoulli distributions as well,

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^{d} q_{\phi_i}(z_i|\mathbf{x}) = \prod_{i=1}^{d} \mathcal{B}(z_i; \phi_i(\mathbf{x})),$$

where $\phi_i(\mathbf{x})$ is the output of a neural network. Let us define the prior distribution for the binary latent variables as the simplest Bernoulli distribution,

$$p(\mathbf{z}) = \prod_{i=1}^{d} \mathcal{B}(z_i; 0.5).$$

Then, for each element i of the vectors, the KL divergence between the recognition model and the prior can be computed as

$$\begin{aligned} D_{\mathrm{KL}}(q_{\phi_i}(z_i|\mathbf{x})||p(z_i)) &= D_{\mathrm{KL}}(\mathcal{B}(z_i; \phi_i(\mathbf{x}))||\mathcal{B}(z_i; 0.5)) \\ &= \sum_{k\in\{0,1\}} \log\left(\phi_i(\mathbf{x})^k (1-\phi_i(\mathbf{x}))^{1-k}\right) - \log(0.5) \end{aligned}$$

Also, given $g_0, g_1 \sim \mathrm{Gumbel}(0,1)$, and for any j-th element of $\mathcal{D}$, equation 2.24 becomes

$$z_i = \frac{\exp[\left(g_i + \log\left(\theta_j(\mathbf{z})^i (1-\theta_j(\mathbf{z}))^{1-i}\right)\right)/\tau]}{\exp[(g_1 + \log(\theta_j(\mathbf{z})))/\tau] + \exp[(g_0 + \log(1-\theta_j(\mathbf{z})))/\tau]}.$$

With this setting, the Bernoulli distribution is relaxed to be continuous, and the ELBO 2.20 can be optimized by SGD.

Once trained, on the one hand, the BVAE is able to encode any image in a much expressive way than a one-hot vector, for instance, a 32-bit binary vector can represent 2^{32} different values, which is much more than the 32 values that a one-hot vector can represent. On the other hand, the image information is compacted over less bits compared to a real-valued. Wang et al., 2023 empirically showed that 8k-bit binary representation compares to a 131k-bit real-valued representation, in terms of reconstruction quality; see 2.3.

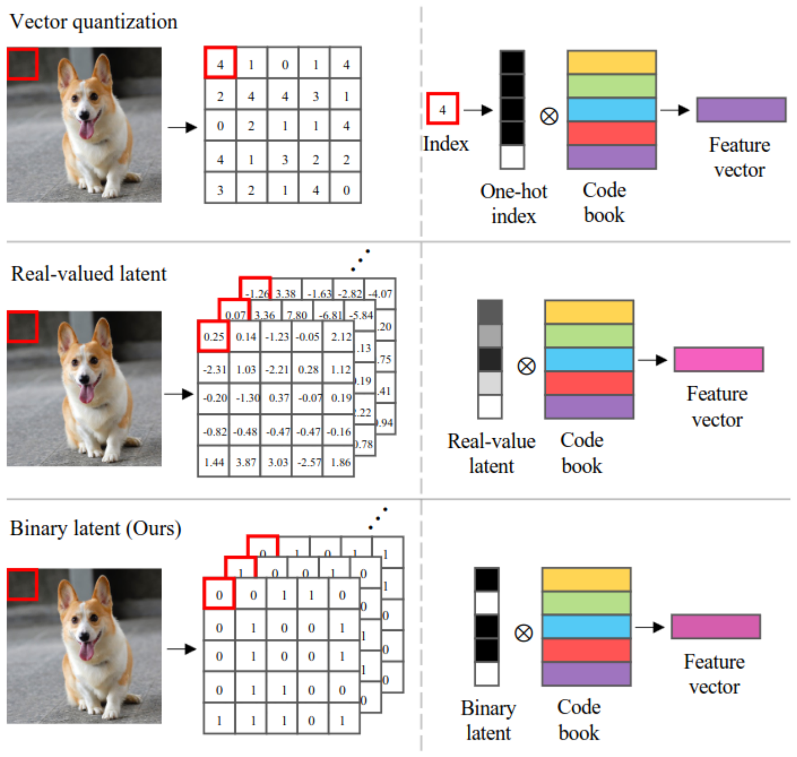


FIGURE 2.3: Heuristic comparison of model expressivity between the binary, continuous and quantized latent spaces. A one-hot vector can only represent a single element of a codebook. A real-valued vector can be any continuous combination of vectors in a codebook. A binary vector lies in between, balancing expressivity and compression.

3 Diffusion Models

In previous chapters, we introduced the concept of generative models, focusing on Variational Autoencoders (VAEs) and their ability not only to reduce the dimensionality of data but also to generate new samples. In this chapter, we will introduce Diffusion Models (DMs), a family of generative models that have gained popularity in recent years due to their impressive results in image (Podell et al., 2023), audio (Zhang et al., 2023a), video (Yang, Srivastava, and Mandt, 2022), and 3D (Xu et al., 2023) generation. The primary distinction between diffusion models and VAEs is that DMs are autoregressive and do not reduce the dimensionality of data. However, the mathematical formulation of their training objective is similar to that of VAEs, as both are generative models trained by maximizing the likelihood of the data.

Given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, recall that a generative model is a probabilistic model p_θ that approximates the data distribution $p_\mathcal{D}$, as shown in 15.

Definition 33 (Diffusion Model)**.** *A **Diffusion Model (DM)** is a generative model $p_\theta(\mathbf{x})$, with parameters θ, that approximates the data distribution $p_\mathcal{D}(\mathbf{x})$ by learning the reverse process of a stochastic diffusion process that transforms the complex distribution $p_\mathcal{D}(\mathbf{x})$ into a simpler, more tractable one.*

Since their introduction, DMs have undergone significant evolution. Various approaches have been proposed to define these models, including energy-based models, continuous-time DMs, and discrete-time DMs. The foundational concept of DMs can be traced back to the principles of stochastic processes and stochastic differential equations, where the diffusion problem has been extensively studied.

Although we will not delve deeply into the theory of stochastic processes, we will introduce some basic concepts related to discrete Markov chains and present relevant results for understanding DMs. We will then thoroughly discuss the training objective for discrete-time DMs and briefly mention other types of DMs, highlighting their connections to discrete-time DMs.

3.1 Discrete Markov Chains

A *stochastic process* is a collection of random variables $\{X_t,\, t \in T\}$, where the index t is often interpreted as time, and X_t represents the *state* of the process at time t. The set T is referred to as the *index set* of the process. If T is discrete, the process is termed a *discrete-time process*, whereas if T is continuous, it is called a *continuous-time process*. The set of possible values that X_t can take for all $t \in T$ is known as the *state space* of the stochastic process. A *Discrete Markov Chain* is a specific type of discrete-time stochastic process that satisfies the Markov property.

Definition 34 (Markov Property)**.** *A stochastic process $\{X_t,\, t \in T\}$ satisfies the **Markov property** if the conditional probability of the next state depends only on the current state and not on the sequence of events that preceded it. Formally, this can be written as:*

$$P(X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0) = P(X_{t+1} = x_{t+1} \mid X_t = x_t),$$

for all $t \in T$ and all states $x_0, x_1, \ldots, x_{t+1}$.

A key concept in discrete Markov chains is the *transition matrix*, which describes the probabilities of moving from one state to another.

The *transition matrix* of a discrete Markov chain is a matrix $P = [p_{ij}]$ where each element p_{ij} represents the probability of transitioning from state i to state j in one time step. Formally, if the states are indexed by integers, then:

$$p_{ij} = P(X_{t+1} = j \mid X_t = i).$$

The transition matrix must satisfy two properties:

1. All elements must be non-negative: $p_{ij} \geq 0$ for all i, j.
2. The sum of the probabilities for each row must equal 1: $\sum_j p_{ij} = 1$ for all i.

These properties ensure that the matrix correctly represents the probabilities of transitioning between states within the Markov chain.

Definition 35 (Irreducible Markov Chain)**.** *A* ***Markov chain*** *is said to be irreducible if, for any two states i and j in the state space, there exists a positive integer n such that the probability of transitioning from state i to state j in n steps is greater than zero. Formally, this can be expressed as:*

$$P(X_n = j \mid X_0 = i) > 0,$$

for some $n > 0$ and all states i and j in the state space.

An *invariant distribution* (or stationary distribution) π for a Markov chain is a probability distribution over the states that remains unchanged as the system evolves over time. If $\pi = [\pi_1, \pi_2, \dots, \pi_N]$ is the invariant distribution, it satisfies the following condition:

$$\pi_j = \sum_i \pi_i p_{ij} \quad \text{for all } j,$$

or, in matrix form:

$$\pi P = \pi.$$

This means that if the system starts in the invariant distribution, it will remain in that distribution at all future time steps. The significance of the invariant distribution lies in its role in determining the long-term behavior of the Markov chain, as it describes the proportion of time the process spends in each state.

Definition 36 (Reversibility)**.** *A Markov chain with an invariant distribution π is said to be* ***reversible*** *if it satisfies the detailed balance condition, which ensures that the process behaves the same in forward and reverse time. Mathematically, the chain is reversible if, for all states i and j, the following condition holds:*

$$\pi_i p_{ij} = \pi_j p_{ji}.$$

This condition implies that the flow of probability from state i to state j is balanced by the flow from state j to state i. Reversibility is an important property because it often simplifies the analysis of the system, particularly Discrete-time Bernoulli DMs take profit of the reversibility property to simplify learning a model that approximates the reverse step of diffusion, see 3.2.5.

3.2 Discrete-time Diffusion Models

Discrete-time DMs, are a subset of DMs that aim to approximate the data distribution by means of reversing a discrete Markov chain that progressively transforms a complex distribution into a simple and tractable one. They were first introduced by Sohl-Dickstein et al., 2015 and later gained popularity through the work of Ho, Jain, and Abbeel, 2020.

Therefore, the goal is first to define a forward (or inference) discrete Markov chain that converts the data distribution $p_{\mathcal{D}}$ into $\pi(\mathbf{y})$, which we assume to be known and easy to compute and sample from. Then a model $p_\theta(\mathbf{x})$ is trained to learn the reverse process, bringing the simple distribution to an approximation of the complex original one. Thus, the diffusion process is divided into two steps.

3.2.1 Forward Process

Let $\mathcal{D} \subset \mathbb{R}^n$ be a dataset and $\mathbf{x} \in \mathcal{D}$ a datapoint. Let $p_{\mathcal{D}}(\mathbf{x})$ be the data distribution of the elements in $\mathcal{D}$. Consider the discrete Markov chain $X_0, X_1, \dots, X_T$ such that the random variable X_t is the state of the system at time t, and X_0 is the initial state that follows the data distribution $p_{\mathcal{D}}(\mathbf{x})$. The element $\mathbf{x}$ will be denoted as $\mathbf{x}^{(0)}$, $q(\mathbf{x}^{(0)}) = p_{\mathcal{D}}(\mathbf{x})$ and the values of X_t will be denoted as $\mathbf{x}^{(t)}$. The transition probability between states is given by the Markov kernel $q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) = T_\pi(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}; \beta_t)$, where β_t, called the *noise scales*, are the parameters of the distribution. The kernels and the noise scales are chosen such that the distribution of the states $\mathbf{x}^{(T)}$ resembles to $\pi(\mathbf{y})$ as $T \to \infty$. Such a process is called a *forward trajectory.*

Definition 37 (Forward Trajectory)**.** *Given a dataset $\mathcal{D}$ and a data distribution $p_{\mathcal{D}}(\boldsymbol{x})$, the forward trajectory of a diffusion model is defined by the kernel*

$$q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}) = T_\pi(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}; \beta_t), \tag{3.1}$$

where $\{\beta_t \in (0,1)\}_{t=1}^T$ are the noise scales, and such that,

$$\lim_{T\to\infty} q(\boldsymbol{x}^{(T)}|\boldsymbol{x}^{(0)}) = \pi(\boldsymbol{y}).$$

From the definition, and the Markov property, the probability density function of a forward trajectory after T steps of diffusion is given by the joint probability of the states $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$,

$$q(\mathbf{x}^{(0\dots T)}) = q(\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}) = q(\mathbf{x}^{(0)}) \prod_{t=1}^{T} q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}).$$

Also, we have that

$$q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)}) = \frac{q(\mathbf{x}^{(0\dots T)})}{q(\mathbf{x}^{(0)})} = \prod_{t=1}^{T} q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) \tag{3.2}$$

Furthermore, by means of Bayes' rule, we can compute the forward trajectory $q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$ when conditioned to the data $\mathbf{x}^{(0)}$:

$$\begin{aligned} q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) &= q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}, \mathbf{x}^{(0)}) \\ &= \frac{q(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \mathbf{x}^{(0)})}{q(\mathbf{x}^{(t-1)}, \mathbf{x}^{(0)})} \\ &= \frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)})q(\mathbf{x}^{(t)}, \mathbf{x}^{(0)})}{q(\mathbf{x}^{(t-1)}, \mathbf{x}^{(0)})} \\ &= \frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)})q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})q(\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})q(\mathbf{x}^{(0)})} \\ &= \frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)})q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}. \end{aligned}$$

We will focus on the Gaussian and Bernoulli forward trajectories, despite that, extending on the theory of stochastic processes, it is possible to proof that this works in a more general setting (Pavliotis, 2014). The nice properties of such simple trajectories are mainly two-fold. First is that they can be easily computed in a closed form given $\mathbf{x}^{(0)}$ and $\{\beta_t\}_{t=1}^{T}$, as we will see in 3.2.4 and 3.2.5. Second, is that according, the reverse diffusion process will also be Gaussian or Bernoulli, respectively, and as long as $\{\beta_t\}_{t=1}^{T}$ are small enough. Feller, 1949 proves this property using the fact that a discrete-time diffusion process can be seen as a process described by a stochastic differential equation when the time step is small enough. In other words, a discrete-time diffusion process can be seen as a discretization of a continuous-time diffusion process, as we will discuss in 3.4.

3.2.2 Reverse Process

Given the above procedure, we would like to compute the reverse steps, i.e. the posterior distribution $q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})$. This is a similar setting to the Variational Inference problem discussed in the previous chapter, which requires computing an intractable integral. Therefore, instead of computing each reverse step $q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})$, it is approximated by a model $p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})$.

Definition 38 (Reverse Trajectory). *Given a dataset $\mathcal{D}$ with data distribution $p_{\mathcal{D}}(\boldsymbol{x})$, and a forward trajectory defined by the kernel $q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)})$, the reverse trajectory is defined by the model $p_\theta(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)})$ that approximates the reverse distribution $q(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)})$.*

The probability density function of the reverse trajectory is given by

$$p_\theta(\mathbf{x}^{(T\dots 0)}) = p_\theta(\mathbf{x}^{(T)}) \prod_{t=1}^{T} p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}),$$
$$p_\theta(\mathbf{x}^{(T-1\dots 0)}|\mathbf{x}^{(T)}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}).$$

If the chosen forward trajectory is reversible, the space of the variational distribution for the reverse trajectory is reduced to the distribution space of the forward trajectory. Therefore, $q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})$, $q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$ and $p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})$ are all the same type of distribution, and we only need to learn the parameters θ that best approximate the reverse distribution, drastically simplifying the problem. As we will see later, both Gaussian 3.2.4 and Bernoulli 3.2.5 forward trajectories have this property.

The forward trajectory is also called inference diffusion process, since the variable $\mathbf{x}^{(0)}$ is observed and the goal is to infer the distribution of the variable $\mathbf{x}^{(T)}$ given $\mathbf{x}^{(0)}$. The reverse trajectory, on the other hand, is called generative diffusion process, since the variable $\mathbf{x}^{(T)}$ is easily sampled and the goal is to generate samples from the distribution of the variable $\mathbf{x}^{(0)}$ by transforming $\mathbf{x}^{(T)}$.

Note that the probability density fuction the generative model assigns to the data is given by

$$p_\theta(\mathbf{x}^{(0)}) = \int p_\theta(\mathbf{x}^{(0\dots T)})\, d\mathbf{x}^{(1\dots T)},$$

which is intractable. However, by means of the forward and reverse trajectories, we have that

$$\begin{aligned} p_\theta(\mathbf{x}^{(0)}) &= \int p_\theta(\mathbf{x}^{(0\dots T)}) \frac{q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)})}\, d\mathbf{x}^{(1\dots T)} \\ &= \int q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)}) \frac{p_\theta(\mathbf{x}^{(0\dots T)})}{q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)})}\, d\mathbf{x}^{(1\dots T)} \\ &= \int q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)}) p_\theta(\mathbf{x}^{(T)}) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\, d\mathbf{x}^{(1\dots T)} \end{aligned}$$

To sample from this distribution, we need first to compute the integral, or at least approximate it. Is possible to show that for β_t small enough and for T large enough, this process is equivalent to a quasi-static process from statistical physics, where the system is in equilibrium at each step of the diffusion. In this case, the forward and reverse trajectories are identical and the above integral can be approximated. More details on non-equilibrium thermodynamics can be found in Jarzynski, 2011 and Spinney and Ford, 2013. Furthermore, with this approximation sampling can be done by means of Annealed Importance Sampling (Neal, 2001). In this thesis, we will not go into further detail on this line since, as we will later see in the chapter, there are simpler ways to compute the integral and sampling, for Gaussian and Bernoulli diffusion processes.

3.2.3 Training Objective

Observe that if we treat $\mathbf{x}^{(0)}$ as an observed variable, and $\mathbf{x}^{(1\dots T)}$ as latent variables, then the diffusion process can be seen as a VAE (Bank, Koenigstein, and Giryes, 2021). The forward and reverse trajectories would be equivalent to the encoder and decoder respectively, going from data to "latent space" and back.

As we have discussed in previous chapters, the training objective of a generative model amounts to maximizing the log-likelihood of the model p_θ that approximates the data distribution $q(\mathbf{x}^{(0)}) := p_{\mathcal{D}}(\mathbf{x})$,

$$\arg\max_\theta \mathbb{E}_{q(\mathbf{x}^{(0)})}[\log p_\theta(\mathbf{x}^{(0)})] = \arg\max_\theta \int q(\mathbf{x}^{(0)}) \log p_\theta(\mathbf{x}^{(0)})\, d\mathbf{x}^{(0)}.$$

As solving this integral is intractable, we are interested in finding ELBO of the log-likelihood and maximizing it instead.

Theorem 10 (ELBO for Discrete-time DMs)**.** *Let $q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)})$ define the forward trajectory of a DM and $p_\theta(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)})$ define the reverse trajectory, for $t \in \{1, ..., T\}$. Then the ELBO of the log-likelihood of the data distribution is given by*

$$\mathcal{L}_\theta(\mathcal{D}) := -\mathbb{E}_{q(\boldsymbol{x}^{(0\dots T)})}\left[\mathcal{L}_\theta^1(\boldsymbol{x}^{(0)}) + \sum_{t=2}^{T-1} \mathcal{L}_\theta^t(\boldsymbol{x}^{(0)}) + \mathcal{L}_\theta^T(\boldsymbol{x}^{(0)})\right] \tag{3.3}$$

where

$$\begin{aligned}
\mathcal{L}_\theta^1(\boldsymbol{x}^{(0)}) &:= -\log p_\theta(\mathbf{x}^{(0)}|\mathbf{x}^{(1)}) \\
\mathcal{L}_\theta^t(\boldsymbol{x}^{(0)}) &:= D_{KL}(q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})||p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})) \\
\mathcal{L}_\theta^T(\boldsymbol{x}^{(0)}) &:= D_{KL}(q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)})||p(\mathbf{x}^{(T)})).
\end{aligned}$$

Proof. By assigning $\mathbf{x} = \mathbf{x}^{(0)}$ and $\mathbf{z} = \mathbf{x}^{(1\ldots T)}$ in the ELBO from 2.5 we have that,

$$\begin{aligned}
-\mathcal{L}_\theta(\mathcal{D}) &= \mathbb{E}_{q(\mathbf{x}^{(0)})}\left[\mathbb{E}_{q_\phi(\mathbf{x}^{(1\ldots T)}|\mathbf{x}^{(0)})}\left[\log\left(\frac{q_\phi(\mathbf{x}^{(1\ldots T)}|\mathbf{x}^{(0)})}{p(\mathbf{x}^{(0)},\mathbf{x}^{(1\ldots T)})}\right)\right]\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{x}^{(0\ldots T)})}\left[\log\left(\frac{q_\phi(\mathbf{x}^{(1\ldots T)}|\mathbf{x}^{(0)})}{p(\mathbf{x}^{(0\ldots T)})}\right)\right] && 3.2 \\
&= \mathbb{E}_q\left[\log\left(\frac{\prod_{t=1}^T q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})}{p_\theta(\mathbf{x}^{(T)})\prod_{t=1}^T p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right)\right] && (q = q(\mathbf{x}^{(0\ldots T)})) \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}^{(T)}) + \sum_{t=1}^T \log\left(\frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right)\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}^{(T)}) + \sum_{t=2}^T \log\left(\frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right) + \log\left(\frac{q(\mathbf{x}^{(1)}|\mathbf{x}^{(0)})}{p(\mathbf{x}^{(0)}|\mathbf{x}^{(1)})}\right)\right] \\
&= (*).
\end{aligned}$$

Now, we have that the middle term can be simplified as follows:

$$\begin{aligned}
\sum_{t=2}^T \log\left(\frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right) &= \sum_{t=2}^T \log\left(\frac{1}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\cdot\frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}\right) \\
&= \sum_{t=2}^T \log\left(\frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right) + \sum_{t=2}^T \log\left(\frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}\right),
\end{aligned}$$

where

$$\sum_{t=2}^T \log\left(\frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}\right) = \log\left(\prod_{t=2}^T \frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}\right) = \log\left(\frac{q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(1)}|\mathbf{x}^{(0)})}\right).$$

Then, with

$$\log\left(\frac{q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(1)}|\mathbf{x}^{(0)})}\right) + \log\left(\frac{q(\mathbf{x}^{(1)}|\mathbf{x}^{(0)})}{p(\mathbf{x}^{(0)}|\mathbf{x}^{(1)})}\right) = \log q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)}) - \log p(\mathbf{x}^{(0)}|\mathbf{x}^{(1)}),$$

and

$$-\log p_\theta(\mathbf{x}^{(T)}) + \log q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)}) = \log\left(\frac{q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(T)})}\right).$$

We have that

$$(*) = \mathbb{E}_q\left[\log\left(\frac{q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(T)})}\right) + \sum_{t=2}^T \log\left(\frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right) - \log p(\mathbf{x}^{(0)}|\mathbf{x}^{(1)})\right]. \tag{3.4}$$

Also note that, by the the definition of KL divergence, we have that

$$\begin{aligned}\mathbb{E}_q\left[\log\left(\frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right)\right] &= \int q(\mathbf{x}^{(0\dots T)})\log\left(\frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right)d\mathbf{x}^{(0\dots T)}\\ &= \int q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})\log\left(\frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right)d\mathbf{x}^{(t-1)}\\ &= D_{\mathrm{KL}}(q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})||p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})).\end{aligned}$$

Finally, by taking the expectation of terms with fractions in 3.4 and using 9, we can write the expresion in terms of KL divergence,

$$\begin{aligned}(*) &= \mathbb{E}_q\left[\mathbb{E}_q\left[\log\left(\frac{q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(T)})}\right)\right]+\sum_{t=2}^{T}\mathbb{E}_q\left[\log\left(\frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})}{p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}\right)\right]-\log p(\mathbf{x}^{(0)}|\mathbf{x}^{(1)})\right]\\ &= \mathbb{E}_q\left[D_{\mathrm{KL}}\left(q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)})||p_\theta(\mathbf{x}^{(T)})\right)+\sum_{t=2}^{T}D_{\mathrm{KL}}(q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})||p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}))-\log p(\mathbf{x}^{(0)}|\mathbf{x}^{(1)})\right].\end{aligned}$$

□

With this, the task of maximizing the log-likelihood of the model p_θ is reduced to minimizing the KL divergence of each of the reverse steps of the diffusion process given an observed data point $\mathbf{x}^{(0)}$. Note that, ideally, the forward distribution chosen should facilitate the computation of $q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)},\mathbf{x}^{(0)})$ and the KL divergences. In the next sections, we will see how this is achieved for Gaussian and Bernoulli forward trajectories.

3.2.4 Discrete-time Gaussian Diffusion Models

Assuming that the forward trajectory of the diffusion process is defined by a Gaussian kernel, was first introduced in DM by Sohl-Dickstein et al., 2015 and later improved by Ho, Jain, and Abbeel, 2020, that added the reparameterization trick for such models. This is the natural choice for the forward trajectory in $\mathbb{R}$, since DMs are inspired by the stochastic process defined by the diffusion equation, which assumes Gaussian noise. We call this model Discrete-time Gaussian Diffusion Model (DGDM), although it is better known as Denoising Diffusion Probabilistic Models (DDPM) in the literature. In short, whener we say Gaussian diffusion model we will refer to DGDMs.

Definition 39 (Gaussian Forward Trajectory). *The forward trajectory of a Gaussian diffusion model of T steps is defined by the kernel*

$$q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}) = \mathcal{N}(\boldsymbol{x}^{(t)};\boldsymbol{x}^{(t-1)}\sqrt{1-\beta_t},\boldsymbol{I}\beta_t). \tag{3.5}$$

Recall that a Gaussian distribution $\mathcal{N}(\mathbf{x};\mu,\sigma)$ can be reparameterized as $\mathbf{x}=\mu+\sigma\cdot\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}\sim\mathcal{N}(0,\boldsymbol{I})$. Therefore, the kernel 3.5 can be reparameterized as

$$\mathbf{x}^{(t)} = \mathbf{x}^{(t-1)}\sqrt{1-\beta_t}+\sqrt{\beta_t}\cdot\boldsymbol{\epsilon}_{t-1}. \tag{3.6}$$

The previous steps can also be reparameterized,

$$\begin{aligned}\mathbf{x}^{(t-1)} &= \mathbf{x}^{(t-2)}\sqrt{1-\beta_{t-1}}+\sqrt{\beta_{t-1}}\cdot\boldsymbol{\epsilon}_{t-2},\\ &\dots\end{aligned}$$

Setting $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, and joining the reparameterizations, we have that

$$\mathbf{x}^{(t)} = \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}^{(t-2)} + \sqrt{\alpha_t(1-\alpha_{t-1})} \boldsymbol{\epsilon}_{t-2} + \sqrt{1-\alpha_t} \boldsymbol{\epsilon}_{t-1} = \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}^{(t-2)} + \hat{\boldsymbol{\epsilon}},$$

Where $\hat{\boldsymbol{\epsilon}}$ is obtained from the merged distribution $\mathcal{N}(\hat{\boldsymbol{\epsilon}}; \mathbf{0}, (1-\alpha_t\alpha_{t-1})\mathbf{I})$. Repeating this with every step, we have that

$$\mathbf{x}^{(t)} = \sqrt{\bar{\alpha}_t} \mathbf{x}^{(0)} + \sqrt{1-\bar{\alpha}_t} \boldsymbol{\epsilon},$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Therefore, the distribution of the forward trajectory can be computed in a closed form given $\mathbf{x}^{(0)}$ and $\{\beta_t\}_{t=1}^{T}$ and sampled from

$$q(\mathbf{x}^{(t)} | \mathbf{x}^{(0)}) = \mathcal{N}(\mathbf{x}^{(t)}; \mathbf{x}^{(0)} \sqrt{\bar{\alpha}_t}, (1-\bar{\alpha}_t)\mathbf{I}). \tag{3.7}$$

From this parameterization it is straight forward to see that, since $\beta_t \in (0, 1)$ for all t,

$$\lim_{t\to\infty} q(\mathbf{x}^{(t)} | \mathbf{x}^{(0)}) = \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

With this reparameterization, it is possible to see that the Gaussian diffusion process is reversible, as long as the noise scales β_t are small enough, without relaying on stochastic differential equations. For a detailed proof of this, we refer to the recent tutorial Nakkiran et al., 2024.

Provided that the Gaussian case is reversible, we have that

$$q(\mathbf{x}^{(t-1)} | \mathbf{x}^{(t)}) = \mathcal{N}(\mathbf{x}^{(t-1)}; \tilde{\mu}(\mathbf{x}^{(t)}, t), \tilde{\Sigma}(\mathbf{x}^{(t)}, t)).$$

Therefore, the model p_θ will be trained to learn the mean and covariance matrix of the reverse trajectory,

$$p_\theta(\mathbf{x}^{(t-1)} | \mathbf{x}^{(t)}) = \mathcal{N}(\mathbf{x}^{(t-1)}; \mu_\theta(\mathbf{x}^{(t)}, t), \Sigma_\theta(\mathbf{x}^{(t)}, t)).$$

Another important property of Gaussian forward trajectories, is that the reverse trajectory can be computed in a closed form when conditioned to the data.

Proposition 16. *For $\boldsymbol{x}^{(0)}$ observed, the reverse trajectory of a Gaussian diffusion model is given by*

$$q(\mathbf{x}^{(t-1)} | \mathbf{x}^{(t)}, \boldsymbol{x}^{(0)}) = \mathcal{N}(\mathbf{x}^{(t-1)}; \tilde{\mu}(\mathbf{x}^{(t)}, \mathbf{x}^{(0)}), \tilde{\beta}_t \mathbf{I}), \tag{3.8}$$

where

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$$

$$\tilde{\mu}_t(\mathbf{x}^{(t)}, \mathbf{x}^{(0)}) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} \mathbf{x}^{(t)} + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} \mathbf{x}^{(0)}$$

Proof. By the Bayes' rule and 3.7, we have that

$$
\begin{aligned}
&q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)}) = \\
&= q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}, \mathbf{x}^{(0)}) \frac{q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})} \\
&= \frac{1}{\sqrt{2\pi(1-\alpha_t)}} \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}^{(t)} - \sqrt{\alpha_t}\mathbf{x}^{(t-1)}}{\sqrt{(1-\alpha_t)}}\right)^2\right) \frac{\frac{1}{\sqrt{2\pi(1-\bar{\alpha}_t)}} \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}^{(t-1)} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}^{(0)}}{\sqrt{(1-\bar{\alpha}_{t-1})}}\right)^2\right)}{\frac{1}{\sqrt{2\pi(1-\bar{\alpha}_t)}} \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}^{(t)} - \sqrt{\bar{\alpha}_t}\mathbf{x}^{(0)}}{\sqrt{(1-\bar{\alpha}_t)}}\right)^2\right)} \\
&\propto \exp\left(-\frac{1}{2}\left(\frac{(\mathbf{x}^{(t)} - \sqrt{\alpha_t}\mathbf{x}^{(t-1)})^2}{\beta_t} + \frac{(\mathbf{x}^{(t-1)} - \sqrt{\alpha_{\bar{t}-1}}\mathbf{x}^{(0)})^2}{(1-\bar{\alpha}_{t-1})} - \frac{(\mathbf{x}^{(t)} - \sqrt{\bar{\alpha}_t}\mathbf{x}^{(0)})^2}{(1-\bar{\alpha}_t)}\right)\right) \\
&= \exp\Bigg(-\frac{1}{2}\Bigg(\frac{\mathbf{x}^{(t)2} - 2\mathbf{x}^{(t)}\sqrt{\alpha_t}\mathbf{x}^{(t-1)} + \alpha_t\mathbf{x}^{(t-1)2}}{\beta_t} + \frac{\mathbf{x}^{(t-1)2} - 2\mathbf{x}^{(t-1)}\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}^{(0)} + \bar{\alpha}_{t-1}\mathbf{x}^{(0)2}}{(1-\bar{\alpha}_{t-1})} - \\
&\quad - \frac{(\mathbf{x}^{(t)} - \sqrt{\bar{\alpha}_t}\mathbf{x}^{(0)})^2}{(1-\bar{\alpha}_t)}\Bigg)\Bigg) \\
&= \exp\left(-\frac{1}{2}\left(\frac{\alpha_t\mathbf{x}^{(t-1)2}}{\beta_t} + \frac{\mathbf{x}^{(t-1)2}}{(1-\bar{\alpha}_{t-1})} - \frac{2\mathbf{x}^{(t)}\sqrt{\alpha_t}\mathbf{x}^{(t-1)}}{\beta_t} - \frac{2\mathbf{x}^{(t-1)}\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}^{(0)}}{(1-\bar{\alpha}_{t-1})} + C(\mathbf{x}^{(t)}, x^{(0)})\right)\right) \\
&= \exp\left(-\frac{1}{2}\left(\left(\frac{\alpha_t}{\beta_t} + \frac{1}{(1-\bar{\alpha}_{t-1})}\right)\mathbf{x}^{(t-1)2} - 2\left(\frac{\mathbf{x}^{(t)}\sqrt{\alpha_t}}{\beta_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}^{(0)}}{(1-\bar{\alpha}_{t-1})}\right)\mathbf{x}^{(t-1)} + C(\mathbf{x}^{(t)}, \mathbf{x}^{(0)})\right)\right).
\end{aligned}
$$

We want to correspond this expression with a Gaussian distribution. To do so, observe that

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \propto \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) = \exp\left(-\frac{1}{2}\left(\frac{1}{\sigma^2}x^2 - \frac{2\mu}{\sigma^2}x + \frac{\mu^2}{\sigma^2}\right)\right).$$

Similarly, we can obtain the same for the multivariate case. Corresponding this result with q and denoting the matrix correlation by the diagonal vector $\tilde{\beta}_t$, we obtain what we were looking for:

$$
\begin{aligned}
\tilde{\beta}_t &= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \\
\tilde{\mu}_t(\mathbf{x}^{(t)}, \mathbf{x}^{(0)}) &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}^{(t)} + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}^{(0)}
\end{aligned}
$$

□

Observe that from the reparameterization of the Gaussian, since $\mathbf{x}^{(t)} = \sqrt{\bar{\alpha}_t}\mathbf{x}^{(0)} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$, we have that $\mathbf{x}^{(0)} = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}^{(t)} - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon})$, which leads to a further simplification of the mean $\tilde{\mu}_t$ as

$$
\begin{aligned}
\tilde{\mu}_t &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t) \\
&= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)
\end{aligned}
$$

In Ho, Jain, and Abbeel, 2020, the authors provided empirical evidence for an improvement in the performance of the diffusion model when reducing the amount of variables to be learned by fixing the covariance matrix according to the noise schedule. For the reverse process $p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) = \mathcal{N}(\mathbf{x}^{(t-1)}; \mu_\theta(\mathbf{x}^{(t)}, t), \Sigma_\theta(\mathbf{x}^{(t)}, t))$, the covariance matrix is predefined as a diagonal matrix $\sigma_\theta(\mathbf{x}^{(t)}, t) = \sigma_t^2\mathbf{I}$. Although this simplification reduces the flexibility of the model, it has been shown to improve the performance of the model in practice when choosing $\sigma_t^2 = \beta_t$ or $\sigma_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. Such choice of σ_t^2 comes from the upper and lower bounds of the entropy of the reverse process, $H_q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})$.

Proposition 17 (Entropy of the reverse process)**.** *The entropy of the reverse process* $q(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)})$ *is bounded as*

$$\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \leq H_q(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)}) \leq \beta_t.$$

The reparameterization trick, together with the fixed covariance matrix, allows the model to be trained to learn the denoising function between steps, with a simpler training objective given by the following theorem. For a detailed calculations we refer to Ho, Jain, and Abbeel, 2020.

Corollary 3 (Simplified Training Objective)**.** *The training objective of the DGDM model is given by*

$$\arg\min_\theta \mathcal{L}_\theta(\mathcal{D}) = \arg\min_\theta L_{simple}(\mathcal{D}, \theta) := \arg\min_\theta \mathbb{E}_{t,\boldsymbol{x}^{(0)},\epsilon}\left[||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\boldsymbol{x}^{(0)} + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||_2^2\right]. \tag{3.9}$$

The reparameterization trick and makes sampling from $p(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})$ much easier, we can obtain samples from the reverse process by iterating over the following steps:

$$\mathbf{x}^{(t-1)} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}^{(t)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}^{(t)}) + \sigma_t\mathbf{z}\right),$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

With this setting algorithms 2 and 3 can be used to train and sample from the DGDM model.

Algorithm 2 DGDM Training

```
1: repeat
2:     $x_0 \sim q(x_0)$
3:     $t \sim \text{Uniform}(\{1, \dots, T\})$
4:     $\epsilon \sim \mathcal{N}(0, I)$
5:     Take gradient descent step on $\nabla_\theta \left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t\right)\right\|^2$
6: until converged
```

Algorithm 3 DGDM Sampling

```
1: $x_T \sim \mathcal{N}(0, I)$
2: for $t = T, \dots, 1$ do
3:     $z \sim \mathcal{N}(0, I)$ if $t > 1$, else $z = 0$
4:     $x_{t-1} = \sqrt{\frac{1}{\alpha_t}}\left(x_t - \sqrt{\frac{1-\alpha_t}{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$
5: end for
6: return $x_0$
```

From this point on, several improvements have been made to the DGDM model. Nichol and Dhariwal, 2021 removes the assumption of the fixed covariance matrix and learn the full covariance matrix of the reverse process, subjected to more flexible constraints. Since the simple loss function does not depend on $\Sigma_\theta(\mathbf{x}^{(t)}, t)$, the authors propose minimizing instead the Hybrid Loss Function,

$$\arg\min_\theta L_{\text{hybrid}}(\theta) := \arg\min_\theta L_{\text{simple}}(\mathcal{D}, \theta) - \lambda \mathcal{L}_\theta, \tag{3.10}$$

where $\mathcal{L}_\theta$ is the ELBO of the Gaussian forward trajectory from 10, and λ is a hyperparameter that balances the two terms. The gradients of $\mathcal{L}_\theta$ can be computed in a closed form, and the gradients of $L_{\text{simple}}(\mathcal{D}, \theta)$ can be computed using the reparameterization trick. However, we only have empirical evidence that 3.10 performs better than 3.9.

The simple training objective can be shown to be equivalent to the loss weighting used Noise-Conditioned Score Networks (NCSN), an energy-based generative model (Song and Ermon, 2019a), which estimates the gradients of the data distributions at different levels of added noise. This connects the discrete-time diffusion models with another class of generative models, the energy-based models, which instead of estimating the data distribution, estimate the gradients of the data distribution and follow Langevin dynamics to generate samples. A definition of energy-based models will be given in the next section, but its properties are not discussed in this thesis and the reader is referred to Song and Ermon, 2019a and Song and Ermon, 2019b for more details.

3.2.5 Discrete-time Bernoulli Diffusion Models

In section 3.2 we have seen the classical definition of DGDM, where the forward and reverse trajectories are defined by Gaussian distributions. We have seen that by means of the reparameterization trick for Gaussian distributions, we can obtain an easier version of the ELBO. Our interest now is the case of binary state spaces. We will assume a forward trajectory defined by a Bernoulli distribution, and find a reparameterization trick for the reverse trajectory similar to the one used in DGDM that leads to a simple loss function. We call this the Discrete-time Bernoulli Diffusion Model (DBDM).

Definition 40 (Bernoulli Forward Trajectory)**.** *The forward trajectory of a Bernoulli diffusion model of T steps is defined by the kernel*

$$q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}) = \mathcal{B}(\boldsymbol{x}^{(t)}; \boldsymbol{x}^{(t-1)}(1-\beta_t) + 0.5\beta_t), \tag{3.11}$$

where $\{\beta_t\}_{t=1}^T$ is the noise schedule, and $\beta_t \in (0,1)$, for all t.

By recursively applying $q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) = \mathcal{B}(\mathbf{x}^{(t)}; \mathbf{x}^{(t-1)}(1-\beta_1) + 0.5\beta_1)$, we obtain that,

$$\begin{aligned}
&p_0 = P(\mathbf{x}^{(0)} = 1),\\
&p_1 = P(\mathbf{x}^{(1)} = 1|\mathbf{x}^{(0)}) = (1-\beta_1)p_0 + 0.5\beta_1,\\
&p_2 = P(\mathbf{x}^{(2)} = 1|\mathbf{x}^{(1)}) = (1-\beta_2)p_1 + 0.5\beta_2,\\
\dots&\\
&p_t = P(\mathbf{x}^{(t)} = 1|\mathbf{x}^{(t-1)}) = (1-\beta_t)p_{t-1} + 0.5\beta_t,
\end{aligned}$$

which leads to

$$p_t = p_0 \prod_{i=1}^t (1-\beta_i) + 0.5 \sum_{i=1}^t \beta_i \prod_{j=i+1}^t (1-\beta_j).$$

Writing $\bar{\alpha}_t = \prod_{i=1}^t (1-\beta_i)$, and $b_t = (1-\beta_t)b_{t-1} + 0.5\beta_t$, were $b_1 = 0.5\beta_1$, we have that $p_t = \bar{\alpha}_t p_0 + b_t$. Therefore, the forward trajectory can be computed in a closed form given $\mathbf{x}^{(0)}$

and $\{\beta_t\}_{t=1}^T$ and its distribution is given by

$$q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)}) = \mathcal{B}(\mathbf{x}^{(t)}; \bar{\alpha}_t\mathbf{x}^{(0)} + b_t). \tag{3.12}$$

For each bit $x_i^{(t)} \in \{0,1\}$ of the vector $\mathbf{x}^{(t)}$, can rewrite $q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})$ in terms of the transition matrix of the Markov chain by means of

$$q(x_i^{(t)}|x_i^{(t-1)}) = v(x_i^{(t)})^T Q_t v(x_i^{(t-1)}), \text{ where } Q_t = \begin{bmatrix} 1-\beta_t & 0.5\beta_t \\ 0.5\beta_t & 1-\beta_t \end{bmatrix}, \tag{3.13}$$

and $v(x_i^{(t)}) = \text{one\_hot}(x_i^{(t)})$. Also, we have that,

$$q(x_i^{(t)}|x_i^{(0)}) = v(x_i^{(t)})^T \bar{Q} v(x_i^{(0)}), \text{ where } \bar{Q} = \begin{bmatrix} 1+b_t & 1-(\bar{\alpha}_t+b_t) \\ b_t & \bar{\alpha}_t+b_t \end{bmatrix}.$$

We can work with vector notation and have the same formulation for the whole vector $\mathbf{x}^{(t)}$, and get the tensor $\mathbf{Q}$. Observe that it is possible to define a noise schedule for which the forward trajectory will transform the complex distribution into $\mathcal{B}(\mathbf{y}; 0.5)$, by setting a noise schedule such that $\prod_{i=1}^T(1-\beta_i) \approx 0$ and $\sum_{i=1}^T \beta_i \prod_{j=i+1}^T (1-\beta_j) \approx 1$.

Proposition 18. *For any $\{\beta_t \in (0,1)\}$, the following identity holds*

$$\sum_{i=1}^T \beta_i \prod_{j=i+1}^T (1-\beta_j) = 1 - \prod_{i=1}^T (1-\beta_i).$$

Proof. Let's prove it by induction. For $T = 1$, we have that

$$\sum_{i=1}^1 \beta_i \prod_{j=i+1}^1 (1-\beta_j) = \beta_1 = 1-(1-\beta_1).$$

For $T = 2$, also holds,

$$\sum_{i=1}^2 \beta_i \prod_{j=i+1}^2 (1-\beta_j) = \beta_1(1-\beta_2) + \beta_2 = 1-(1-\beta_1)(1-\beta_2).$$

Assume that the proposition holds for $T = k$, then for $T = k+1$ we have that

$$\begin{aligned}\sum_{i=1}^{k+1} \beta_i \prod_{j=i+1}^{k+1} (1-\beta_j) &= \sum_{i=1}^{k} \beta_i \prod_{j=i+1}^{k+1} (1-\beta_j) + \beta_{k+1} \prod_{j=k+2}^{k+1} (1-\beta_j) \\ &= \left[\sum_{i=1}^{k} \beta_i \prod_{j=i+1}^{k} (1-\beta_j)\right](1-\beta_{k+1}) + \beta_{k+1} \\ &= \left[1 - \prod_{i=1}^{k}(1-\beta_i)\right](1-\beta_{k+1}) + \beta_{k+1} \\ &= 1 - \prod_{i=1}^{k+1}(1-\beta_i).\end{aligned}$$

□

Therefore, we want $\sum_{i=1}^{T} \beta_i \prod_{j=i+1}^{T}(1-\beta_j) = 1 - \prod_{i=1}^{T}(1-\beta_i) \approx 1$, which is the same as $\prod_{i=1}^{T}(1-\beta_i) \approx 0$. Consider the infinite product,

$$\prod_{i=1}^{\infty}(1-\beta_i).$$

This converges to 0 if and only if the series $\sum_{i=1}^{\infty} \log(1-\beta_i)$ diverges to $-\infty$. It is possible to refine this even more, since it is possible to see that this series diverges to $-\infty$ if and only if the series

$$\sum_{i=1}^{\infty} -\beta_i \text{ diverges to } -\infty. \tag{3.14}$$

Therefore, for a large enough T, we can choose a β_t noise schedule such that at step T, $q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)}) \approx \mathcal{B}(\mathbf{y}; 0.5)$. Thus, the Bernoulli forward trajectory is well defined.

Looking at the transition matrix Q_t, we can see that the Bernoulli forward trajectory is reversible. Note that the process is irreducible, and that $\pi = [0.5, 0.5]$ is the invariant distribution, since $\pi Q_t = \pi$ for all t. With this, the reversibility condition is satisfied, since

$$\pi_0 Q_{0,1} = \pi_1 Q_{1,0} \text{ where } Q = Q_t, \forall t.$$

Doing the same for the entire tensor $\mathbf{Q}$, we have that the Bernoulli forward trajectory is reversible.

Therefore, we have that there exists some $\tilde{\mathbf{b}}(\mathbf{x}^{(t)})$ for which

$$q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) = \mathcal{B}(\mathbf{x}^{(t-1)}; \tilde{\mathbf{b}}(\mathbf{x}^{(t)})).$$

Therefore, the model p_θ will be trained to learn the bit flip probability of the reverse trajectory,

$$p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) = \mathcal{B}(\mathbf{x}^{(t-1)}; \tilde{\mathbf{b}}_\theta(\mathbf{x}^{(t)})).$$

Note that from 3.12, the reverse trajectory can be computed in a closed form when conditioned to the data,

$$\begin{aligned} q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)}) &= \frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}, \mathbf{x}^{(0)})q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})} \\ &= \frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}. \end{aligned}$$

Also, from the definition, we have that the KL divergence between two Bernoulli distributions $\mathcal{B}(p)$ and $\mathcal{B}(q)$ is given by

$$D_{KL}(\mathcal{B}(p)||\mathcal{B}(q)) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}. \tag{3.15}$$

Therefore, the KL divergences $D_{\text{KL}}(q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)})||p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}))$ from ELBO 10, can be computed in a closed form and minimized by SGD. However, optimizing $\tilde{\mathbf{b}}_\theta$ to fit the probabilities of the reverse kernel can be a hard problem and has empirically shown to be an unstable process and fail to converge, as it needs to accurately regress the complex interpolations between $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(T)}$. Therefore, Wang et al., 2023 propose two different approaches, either to predict $\mathbf{x}^{(0)}$ directly at each step, or to predict the residual similar to the DGDM model. It is the main goal of this thesis to provide all the necessary details to understand the training and sampling procedures of the Bernoulli diffusion as proposed in Wang et al., 2023, and ilustrated in figure 3.1.

Direct Prediction

Since defining an analog reparameterization as in the Gaussian case to predict the "noise" between steps, is not straightforward, we directly set the prediction target to $p_\theta(\mathbf{x}^{(0)}|\mathbf{x}^{(t)}) = \mathcal{B}(\mathbf{x}^{(0)}; f_\theta(\mathbf{x}^{(t)}, t))$. Here, f_θ is a model that predicts the logits of $\mathbf{x}^{(0)}$ at each step. This is inspired from the fact that

$$\begin{aligned} q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) &= q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)} = \mathbf{0})q(\mathbf{x}^{(0)} = \mathbf{0}|\mathbf{x}^{(t)}) \\ &+ q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)} = \mathbf{1})q(\mathbf{x}^{(0)} = \mathbf{1}|\mathbf{x}^{(t)}). \end{aligned}$$

Thus, once trained, $p_\theta(\mathbf{x}^{(0)}|\mathbf{x}^{(t)})$ can be used to compute the approximation of the denoising kernel as

$$\begin{aligned} p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) &= q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)} = \mathbf{0})p_\theta(\mathbf{x}^{(0)} = \mathbf{0}|\mathbf{x}^{(t)}) & (3.16) \\ &+ q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)} = \mathbf{1})p_\theta(\mathbf{x}^{(0)} = \mathbf{1}|\mathbf{x}^{(t)}). & (3.17) \end{aligned}$$

If we assume to have obtained f_θ and given $\mathbf{x}^{(t)}$, the reversed kernel can be computed in closed form. By substituting 3.12 into 3.16 we obtain the following proposition

Proposition 19. *Let $q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)})$ define the forward trajectory of a Bernoulli diffusion model and $p_\theta(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)})$ define the reverse trajectory, for $t \in \{1, ..., T\}$. Let f_θ be a model that predicts the logits of $\boldsymbol{x}^{(0)}$ at each step, such that $p_\theta(\boldsymbol{x}^{(0)}|\boldsymbol{x}^{(t)}) = \mathcal{B}(\boldsymbol{x}^{(0)}; f_\theta(\boldsymbol{x}^{(t)}, t))$. Then, the denoising kernel for inference is given by*

$$p_\theta(\boldsymbol{x}^{(t-1)}|\boldsymbol{x}^{(t)}) = \mathcal{B}(\mathbf{x}^{(t-1)}; \tilde{\mathbf{b}}_\theta(\mathbf{x}^{(t)}, \mathbf{x}^{(0)})) = \mathcal{B}\left(\boldsymbol{x}^{(t-1)}; \frac{[(1-\beta_t)\boldsymbol{x}^{(t)} + 0.5\beta_t] \odot [\bar{\alpha} f_\theta(\boldsymbol{x}^{(t)}, t) + 0.5 b_t]}{\boldsymbol{Z}}\right), \tag{3.18}$$

with

$$\boldsymbol{Z} = [(1-\beta_t)\boldsymbol{x}^{(t)} + 0.5\beta_t] \odot [\bar{\alpha} f_\theta(\boldsymbol{x}^{(t)}, t) + 0.5 b_t] + [(1-\beta_t)(1-\boldsymbol{x}^{(t)}) + 0.5\beta_t] \odot [\bar{\alpha}(1 - f_\theta(\boldsymbol{x}^{(t)}, t)) + 0.5 b_t],$$

where $\odot$ is the element-wise product.

Austin et al., 2021 and Gu et al., 2022 provide empirical evidence to show that this retarget in training lead to a more stable optimization process where is easier to learn the denoising kernel for inference, for more general discrete state spaces.

With this approach, the loss function 3.3 can be minimized, and a further regularization term enhancing the reparameterization can be added as in 3.10, to improve the performance of the model. This leads to the following training objective:

$$\arg\min_\theta L^1_{\text{hybrid}}(\theta) := L_{\text{dir}}(\theta) - \lambda \mathcal{L}_\theta, \tag{3.19}$$

where $L_{\text{dir}}(\theta) = \mathbb{E}_{q(\mathbf{x}^{(0)})}\mathbb{E}_{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}\left[-\log p_\theta(\mathbf{x}^{(0)}|\mathbf{x}^{(t)})\right]$, $\mathcal{L}^B_\theta$ is the ELBO of the Bernoulli reverse trajectory, and λ is a hyperparameter that balances the two terms.

Residual Prediction:

Another approach is based on the following observation. Let $\mathbf{x}^{(0)}$ be a binary vector, then the residual between $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(t)}$ can be computed as $\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)}$, where $\oplus$ is the XOR operator, since we have that

$$\mathbf{x}^{(0)} = \mathbf{x}^{(t)} \oplus (\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)}).$$

Recall that XOR outputs 1 if the inputs are different and 0 otherwise, and from its definition the following properties are easily obtained:

- Commutative: $A \oplus B = B \oplus A$
- Associative: $(A \oplus B) \oplus C = A \oplus (B \oplus C)$
- Identity: $A \oplus 0 = A$
- Self-inverse: $A \oplus A = 0$

Furthermore, the following propositi on shows $\mathbf{x}^{(0)}$ can be easily obtained from $\mathbf{x}^{(t)}$ and the residual $\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)}$.

Proposition 20. *Let A and B be binary vectors, Then*

$$B = [(1 - A) \odot (A \oplus B)] + [A \odot (1 - (A \oplus B))], \tag{3.20}$$

where $\odot$ is the element-wise product.

Proof. Note that the element-wise product in the binary space corresponds with the logical AND operator and $(1-A) = \text{NOT}(A)$, then we can apply basic properties of logical operators such as the distributive between AND and XOR operators, to obtain the result:

$$\begin{aligned}
[(1 - A) \odot (A \oplus B)] + [A \odot (1 - (A \oplus B))] &= [\text{NOT}(A) \odot (A \oplus B)] + [A \odot \text{NOT}(A \oplus B)] \\
&= [(\text{NOT}(A) \odot A) \oplus (\text{NOT}(A) \odot B)] \\
&\quad + [(A \odot \text{NOT}(A)) \oplus (A \odot B)] \\
&= [0 \oplus (\text{NOT}(A) \odot B)] + [0 \oplus (A \odot B)] \\
&= (\text{NOT}(A) \odot B) + (A \odot B).
\end{aligned}$$

Finally, with the following truth table

A	B	$(A \odot B)$	$(\text{NOT}A \odot B)$
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

We obtain that $(\text{NOT}(A) \odot B) + (A \odot B) = B$, as we wanted. □

By setting $A = \mathbf{x}^{(t)}$ and $B = \mathbf{x}^{(0)}$, the above proposition allows us to recover $\mathbf{x}^{(0)}$ from $\mathbf{x}^{(t)} \oplus \mathbf{x}^{(0)}$ and $\mathbf{x}^{(t)}$ with basic differentiable operations. Therefore, we want to learn the model $p_\theta(\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)} | \mathbf{x}^{(t)}) = \mathcal{B}(\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)}; f_\theta(\mathbf{x}^{(t)}, t))$, which can be found by directly setting the target variable of f_θ to be $\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)}$. Then, the original sample can be recovered by following the proposition 3.20 and sampling $(\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)})'$ from the approximated distribution p_θ. Once we have $\mathbf{x}^{(0)}$ we can compute the denoising kernel for inference in a closed form as in 3.18.

With this reparameterization, the training objective 3.10 for the Bernoulli diffusion process by predicting the residual is given by

$$\arg\min_\theta L_{\text{hybrid}}(\theta) := L_{\text{residual}}(\theta) - \lambda \mathcal{L}_\theta, \tag{3.21}$$

where $L_{\text{residual}}(\theta) = \mathbb{E}_{q(\mathbf{x}^{(0)})} \mathbb{E}_{q(\mathbf{x}^{(t)} | \mathbf{x}^{(0)})} \left[- \log p_\theta(\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)} | \mathbf{x}^{(t)}) \right]$, and $\mathcal{L}_\theta^B$ is the ELBO of the Bernoulli reverse trajectory. Is easy to see that the training objective 3.21 is equivalent to the one in 3.19, as the two approaches are just different reparameterizations of the same model by means of the XOR operator. Furthermore, because minimizing the negative log-likelihood is equivalent to minimizing the KL divergence, which in turn is equivalent to minimizing the

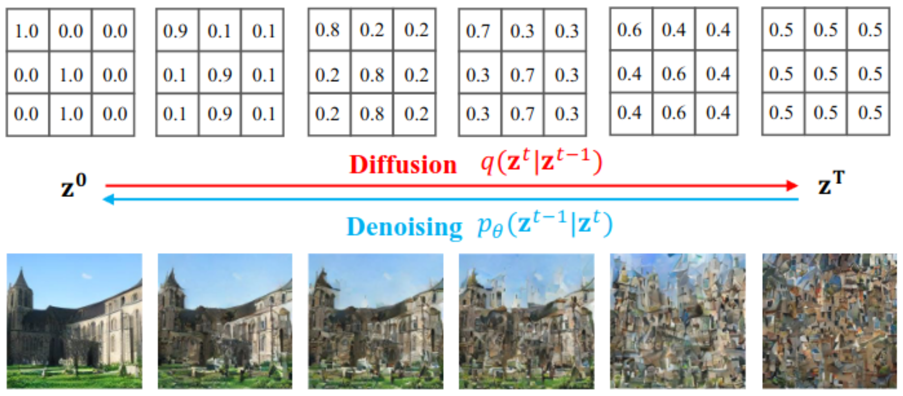


FIGURE 3.1: Bernoulli Diffusion Process.

cross-entropy between $p(\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)}|\mathbf{x}^{(t)})$ and $p_\theta(\mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)}|\mathbf{x}^{(t)})$, the final objective can be simplified to

$$\bar{L}_{\text{hybrid}}(\theta) = \mathbb{E}_{q(\mathbf{x}^{(0)})}\mathbb{E}_{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}\left[H(f_\theta(\mathbf{x}^{(t)}, t), \mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)})\right] - \lambda\mathcal{L}_\theta. \tag{3.22}$$

With this, the discussion on diffusion models for binary state spaces is concluded. The algorithms for training and sampling can be found in **??**. We have seen that the reparameterization trick for Bernoulli diffusion models can be achieved by predicting the residual between the original sample and the sample at time t. Furthermore, we have seen that the training objective for Bernoulli diffusion models can be simplified to a cross-entropy loss between the target and the prediction with the ELBO. This approach has been shown to empirically be more stable and efficient than the direct prediction of the kernel, and it is the main component of the Bernoulli diffusion model proposed in Wang et al., 2023.

Before concluding this chapter, we will briefly discuss the connection between diffusion models, energy-based models, and continuous-time diffusion models.

3.3 Energy-Based Models

The concept of an energy-based model Hinton, 1999 centers on learning an energy functional $E_\theta : \mathcal{D} \to \mathbb{R}$, instead of directly modeling a probabilistic distribution $p_\theta(\mathbf{x})$ over a space $\mathcal{D}$. This energy functional implicitly defines a probability distribution via the Boltzmann distribution, expressed as

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

where $Z_\theta = \int_{\mathcal{D}} \exp(-E_\theta(\mathbf{y}))d\mathbf{y}$ is a normalization constant, known as the partition function. While constructing a function $E_\theta(\mathbf{x})$ is straightforward, enforcing the constraint $\int_D p_\theta(\mathbf{x})d\mathbf{x} = 1$, or calculating the partition function Z_θ, can be quite challenging.

To utilize an energy-based model as a generative model, we must address two key challenges. On one hand, we need a training procedure to optimize the parameters of the energy function E_θ, so that the implicit distribution $p_\theta(\mathbf{x})$ approximates the true data-generating distribution $p(\mathbf{x})$. On the other hand, we need a sampling procedure to generate samples $\mathbf{x} \sim p_\theta$. Crucially, both solutions must avoid the intractable computation of the integral Z_θ.

3.3.1 Langevin Dynamics

Given a trained model E_θ, we aim to sample from the corresponding distribution $\mathbf{x} \sim p_\theta$. Although direct sampling from p_θ is challenging, we can approximate these samples using a Markov chain with p_θ as its stationary distribution. Langevin dynamics (Welling and Teh, 2011) provides a convenient framework for this in the domain $\mathcal{D} \subset \mathbb{R}^n$. This method is based on a continuous Markov process governed by the stochastic differential equation (SDE)

$$\frac{\partial \mathbf{x}}{\partial t} = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t)\, dt + \sqrt{2}\, d\mathbf{W}_t, \tag{3.23}$$

where $d\mathbf{W}_t$ represents white noise, which is the derivative of standard Brownian motion W_t. According to the Fokker-Planck equation (Feller, 1949), diffusion following these dynamics asymptotically converges to samples $\mathbf{x}_t \sim p_\theta$, in the sense that $D_{\mathrm{KL}}(\mathbf{x}_t || p_\theta)$ approaches zero as $t \to \infty$.

In practice, we cannot simulate the continuous dynamics exactly as given in Equation 3.23. Instead, we discretize the diffusion process and use a discrete Markov chain driven by i.i.d. Gaussian noise $\epsilon_t \sim \mathcal{N}(0, I)$:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}_t) + \sqrt{2\eta}\epsilon_t. \tag{3.24}$$

This equation can be interpreted as the stochastic analog of an Euler discretization of a deterministic differential equation. As the step size $\eta \to 0$, the approximation of the continuous dynamics in Equation 3.23 becomes more accurate, but the sampling process may become slower. An effective acceleration technique for sampling, based on simulated annealing (Neal, 2001), is detailed in Song and Ermon, 2019b.

3.3.2 Score Matching

Langevin dynamics enables sampling from an energy-based model because the gradient of the log-density is directly related to the energy function:

$$\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log Z_\theta = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}). \tag{3.25}$$

In fact, rather than modeling the energy function directly, we can estimate the gradient field of the log-density, also known as the score function $s : \mathbb{R}^n \to \mathbb{R}^n$, defined by $s(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$. This score function can be modeled using a neural network parameterized by θ, where θ are the parameters of the neural network. This approach implicitly defines an energy function $E_\theta(\mathbf{x})$ (up to an additive constant) and a corresponding density $p_\theta(\mathbf{x})$.

The score function s_θ is learned by minimizing the score matching objective:

$$\arg\min_\theta \mathbb{E}_{p(\mathbf{x})} \left[\frac{1}{2} || s_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p(\mathbf{x}) ||_2^2 \right]. \tag{3.26}$$

However, directly computing the gradient of the log-density $\log p(\mathbf{x})$ is typically infeasible. Fortunately, it has been shown that the score matching objective can be minimized indirectly by considering

$$\arg\min_\theta \mathbb{E}_{p(\mathbf{x})} \left[\frac{1}{2} || s_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log q(\mathbf{x}) ||_2^2 \right] = \arg\min_\theta \mathbb{E}_{p(\mathbf{x})} \left[\mathrm{tr}(\nabla_{\mathbf{x}}^2 \log s_\theta(\mathbf{x})) + \frac{1}{2} || s_\theta(\mathbf{x}) ||_2^2 \right]. \tag{3.27}$$

This approach, known as Implicit Score Matching, is rigorously derived in Hyvärinen, 2005. Furthermore, Song and Ermon, 2019b propose an efficient minimization method by projecting along random directions, while Song and Ermon, 2019a improve the model's performance by adding noise to the data distribution. These advancements culminate in the development of the NCSN model, that includes a kernel perturbation $q_\beta(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \beta^2 I)$ to the data distribution $p_\mathcal{D}(\mathbf{x})$, and $q_\beta(\tilde{\mathbf{x}}) = \int_\mathcal{D} q_\beta(\tilde{\mathbf{x}}|\mathbf{x}) p_\mathcal{D}(\mathbf{x})\, d\mathbf{x}$. Then, consider a sequence of positive noise scales $\beta_{\text{min}} = \beta_1 \leq \beta_2 \leq \ldots \leq \beta_N = \beta_{\text{max}}$, where β_{min} is small enough such that $q_{\text{min}}(\mathbf{x}) \approx p_\mathcal{D}(\mathbf{x})$, and β_{max} is large enough such that $q_{\text{max}} \approx \mathcal{N}(\mathbf{x}; 0, \beta^2_{\text{max}} I)$. With this setting, Song and Ermon, 2019b optimizes the training objective of NCSN model by minimizing a weighted sum of denoising score matching objectives:

$$\arg\min_\theta \sum_{t=1}^{N} \beta_t^2 \mathbb{E}_{q_{\text{min}}(\mathbf{x})} \mathbb{E}_{q_t(\tilde{\mathbf{x}}|\mathbf{x})} \left[||s_\theta(\tilde{\mathbf{x}}, \beta_t) - \nabla_\mathbf{x} \log q_t(\tilde{\mathbf{x}}|\mathbf{x})||_2^2 \right]. \tag{3.28}$$

It turns out that this formulation is equivalent to 3.9 DGDM model when the diffusion process is reinterpreted in terms of perturbed samples and the score function. A more detailed discussion on this equivalence can be found in Vincent, 2011 and Lim et al., 2023.

3.4 Continuous-Time Diffusion Models

We have observed that Langevin dynamics utilize a SDE to define a continuous Markov chain for sampling. However, score-based models discretize this process similarly to DGDMs. Here, we briefly mention that the discrete-time diffusion models have a continuous analog, specifically continuous-time diffusion generative models, we refer to Lim et al., 2023 for a deeper discusion.

Consider a forward diffusion process $\{\mathbf{x}_t\}_{t\in[0,1]}$, where $\mathbf{z}_0$ is the initial state and $\mathbf{x}_t$ represents its perturbation at time t. This diffusion process is governed by the following Itô SDE:

$$d\mathbf{x} = f(t)\mathbf{x}\, dt + g(t)\, d\mathbf{W}, \tag{3.29}$$

where $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ are scalar functions representing the drift and diffusion coefficients, respectively, and $\mathbf{W}$ is a standard Wiener process. The functions $f(t)$ and $g(t)$ can be chosen such that the terminal state $\mathbf{x}_1$ follows a standard normal distribution, i.e., $\mathbf{x}_1 \sim \mathcal{N}(0, I)$, at the end of the diffusion process. As shown in Lim et al., 2023, the SDE in Eq. 3.29 can be transformed into a generative model by first sampling $\mathbf{x}_1 \sim \mathcal{N}(0, I)$ and then solving the reverse-time SDE:

$$d\mathbf{x} = \left[f(t)\mathbf{x} - g(t)^2 \nabla_\mathbf{x} \log q_t(\mathbf{x}) \right] dt + g(t)\, d\bar{\mathbf{W}}, \tag{3.30}$$

where $\bar{\mathbf{W}}$ is a reverse-time Wiener process and dt is an infinitesimal negative time increment. The reverse SDE requires knowledge of the score function $\nabla_\mathbf{x} \log q_t(\mathbf{x}_t)$, which corresponds to the gradient of the log-density of the marginal distribution at time t. One approach to estimate this score function is by minimizing a continuous extension of the score matching objective given by:

$$\arg\min_\theta \mathbb{E}_{U[0,1]} \left[\lambda(t) \mathbb{E}_{q_0(\mathbf{x})} \mathbb{E}_{q_t(\tilde{\mathbf{x}}|\mathbf{x})} \left[||s_\theta(\tilde{\mathbf{x}}, t) - \nabla_\mathbf{x} \log q_t(\tilde{\mathbf{x}}|\mathbf{x})||_2^2 \right] \right]. \tag{3.31}$$

This objective trains the parametric score function $s_\theta(\tilde{\mathbf{x}}, t)$ at time $t \sim U[0,1]$ using a weighting coefficient $\lambda(t)$, where $q_t(\tilde{\mathbf{x}}|\mathbf{x})$ is the diffusion kernel, available in closed form for specific choices of $f(t)$ and $g(t)$.

Note that there is a close connection between continuous-time diffusion models and DGDMs. Recall the reparameterization 3.6, we can rewrite DGDM in terms of jumps of $\Delta_t = 1$, and obtain

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)}\sqrt{1-\beta_{t+1}} + \sqrt{\beta_{t+1}} \cdot \boldsymbol{\epsilon}_t$$
$$\mathbf{x}^{(t+\Delta_t)} = \mathbf{x}^{(t)}\sqrt{1-\beta_{t+1}\Delta_t} + \sqrt{\beta_{t+1}} \cdot \sqrt{\Delta_t}\boldsymbol{\epsilon}_t.$$

In this limit, as $\Delta_t \to 0$, the DGDM converges to a continuous-time diffusion model with equation as in 3.29. This connection allows to translate the properties from SDE, such as the Kolmogorov forward and backward equations, to the discrete-time diffusion models (Shiryayev, 1992).

Recent advances in training Gaussian-based diffusion models include leveraging well-established results from SDE theory to reduce the number of sampling steps and accelerate convergence. However, diffusion models suitable for discrete state spaces, such as binary data, remain relatively underexplored and represent an important area for future research.

Although diffusion models as stated above are powerful generative models, their capabilities can be further extend by conditioning the model on some input data. This allows the model to generate samples given a particular piece of information and paves the way to controlling the synthesis process through inputs y such as text, semantic maps or other image-to-image translation tasks Yang et al., 2023. Similar to other types of generative models, diffusion models are in principle capable of modeling conditional distributions of the form $p(\mathbf{x}|\mathbf{c})$. This can be implemented with a neural network that takes an extra input $f_\theta(\mathbf{x}^{(t)}, \mathbf{c}, t)$, where $\mathbf{c}$ is the conditioning information. This approach is known as explicit conditioning, and it is the most straightforward way to condition a diffusion model. However, there are other ways to condition the model, such as classifier guidance and classifier-free guidance, which are discussed briefly introduced in the next section.

3.5 Conditioning and Guidance Techniques

The wide flexiblity of diffusion models allows them to be used in several different scenarios and adopt different properties. One of which is conditional generation. It is possible to condition the model to generate samples given a particular piece of information. For instance, text-to-image generation using diffusion models has been shown to be even more powerful than other methods such as GANs (Dhariwal and Nichol, 2021). There are different ways of approaching this can be summarized into three techniques:

- **Explicit Conditioning:** Conditinal sampling can be considered as training a model to learn the conditional distribution $p_\theta(\mathbf{x}^{(0)}|\mathbf{c})$. Here $\mathbf{c}$ is the conditioning information, which can be a text prompt, an image, or any other source of information. This approach is the most straightforward one, usually implemented by concatenating the conditioning information to the input of the model or by using an attention mechanism.

- **Classifier Guidance:** Another approach is to train a classifier to predict the conditioning information from the generated samples. Baye's rule allows us to use the gradient of the conditional distribution $p_\theta(\mathbf{c}|\mathbf{x}^{(t)})$, which comes from the classifier, to guide the model towards the desired output $p_\theta(\mathbf{x}^{(0)}|\mathbf{c})$.

- **Classifier-free Guidance:** By using the Baye's rule again we can get an implicit classifier by jointly training a conditional and an unconditional diffusion model (Nichol et al., 2022). In practice, both models are trained together by randomly sampling the condition of the diffusion model at a certain chance.Hence, we want to compute

$$(1+\omega)\log p_\theta(\mathbf{x}_t|\mathbf{c}) - \omega \log p_\theta(\mathbf{x}_t).$$

The difference to explicit conditioning is that this approach additionally accentuates the distance between the conditional and unconditional distributions, by means of the implicit classifier.

Usually, $\mathbf{c}$ is encoded into a latent space by means of the encoder $E_{\theta'}$ of a VAE, and is the latent representation $E_{\theta'}(\mathbf{c})$ that is used as the conditioning information. For example, the Bernoulli diffusion model in 3.2.5 can be explicitly conditioned to $\mathbf{c}$ by

$$\bar{L}^2_{\text{hybrid}}(\theta) = \mathbb{E}_{\mathbf{c}}\mathbb{E}_{q(\mathbf{x}^{(0)})}\mathbb{E}_{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}\left[H(f_\theta(\mathbf{x}^{(t)}, t, E_{\theta'}(\mathbf{c})), \mathbf{x}^{(0)} \oplus \mathbf{x}^{(t)})\right] - \lambda \mathcal{L}^B_\theta. \tag{3.32}$$

Note that in general, the parameters θ' are fixed, although is not a required condition and can potentially be optimized jointly with θ.

The theory dicussed both in this and the previous chapters culminates by merging all together in a single model. In the next chapter we will introduce the Latent Diffusion Model, which combines the power of VAEs and diffusion models to generate high quality samples in a computationally efficient way. Furthermore, we will introduce the Binary Latent Diffusion Model, which extends the LDM to binary state spaces and provides an empirically more stable and efficient training process.

3.6 Latent Diffusion Models

The first work on Latent Difusion Models was introduced by Rombach et al., 2022. Similar to Ho, Jain, and Abbeel, 2020, they proposed to use a diffusion model to generate images, but running the diffusion process after a VAE, in the latent space, instead of the raw data space.

Definition 41 (LDM)**.** *Given a dataset $\mathcal{D}$ with data distribution $p_{\mathcal{D}}$. A **Latent Diffusion Model (LDM)** is a generative model that approximates the distribution $p_{\mathcal{D}}$ and is comprised of two components:*

- *The first component consists of a VAE $(\mathcal{D}, E_\phi, D_\theta)$, that encodes $\boldsymbol{x} \in \mathcal{D} \subset R^n$, into a latent space $\boldsymbol{z} \in \mathcal{Z} \subset \mathbb{R}^d$, of lower dimension, $d < n$.*

- *The second component is a DM that $p_{\theta'}$ that approximates the reverse step of a diffusion process in the latent space. That is, $p_{\theta'}(\boldsymbol{z}^{(t-1)}|\boldsymbol{z}^{(t)}) \approx q(\boldsymbol{z}^{(t-1)}|\boldsymbol{z}^{(t)})$, where q is the forward trajectory of a diffusion process.*

This approach cuts down dramatically the training cost of high resolution generators and makes the inference speed faster. It is motivated by the fact that the semantic information of the data remains after its compression, and performs there the costly computations of the DM. There are two important things to notice from this method.

First, the VAE is specifically designed in such a way that the latent space has a perceptual correlation with the data space. This is done by choosing a well suited architecture of the neural network for the encoder and the decoder. For instance, in the case of images, the common choice involves choosing 2D Convolutional Neural Networks (CNN), that have 2D convolutional layers to capture 2-dimensional local correlations. Second, by reducing to a perceptually simplified space, LDM exploit a property of the inductive bias in diffusion models which makes them particularly well suited for long-range signal generation. That is,the diffusion process deals with the compostition and semantic structure of the data, and leave the details to the decoder.

The training phase can be divided into two parts, where the VAE is trained to encode the data and then the diffusion model is trained over the latent space, or both models can be trained jointly. Also, any of the already discussed guiding techniques can be added to

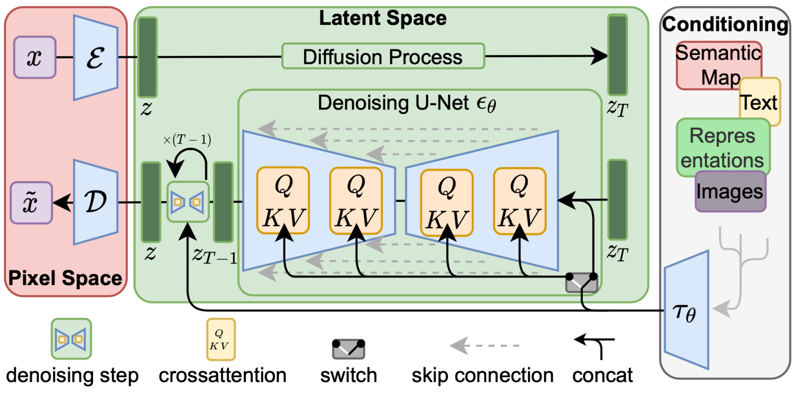


FIGURE 3.2: Architecture of the latent diffusion model. A sample is encoded into latent space, and a U-Net ϵ_θ is used to learn the reverse process of the diffusion model, given a condition encoded with τ_θ. Then, the denoised sample is decoded back to the pixel space of images.

generate samples conditioned to a specific input. In particular, in the case of text-to-image generation, i.e., generating a new image according to a given text, LDM uses a special CNN called U-Net, with classifier-free conditioning and a text encoder, as a denoising function.

3.7 Binary Latent Diffusion Models

Inspired from LDMs, Binary Latent Diffusion Models make a step further on data compression an model efficiency by compressing data into a binary space.

Definition 42 (BLDM)**.** *A **Binary Latent Diffusion Model (BLDM)** is a LDM with a binary latent dimension. That is, the VAE of is a Bernoulli VAE, and the DM is a Discrete Bernoulli DM.*

Since their introduction, LDMs have been incorporating all the advances in VAEs and diffusion models, and improved in several ways until achieving outstanding results in diverse fields and increasing the inference speed to the point of being able to generate high resolution images in almost real time (Sauer et al., 2023). However, although the leading research path has been on improving the Gaussian side of the diffusion model, we are interested in implementing the Bernoulli diffusion model within a LDM with a binary latent space obtained by means of a Bernoulli VAE. This can be beneficial for several reasons. For instance, the nature of the data might be better represented by a binary latent space, or the Bernoulli diffusion process might be more efficient and generate higher quality samples.

This approach has been empirically shown to provide the diffusion model with a compact yet expressive representations of the data, leading to remarkable improvements in performance. Crucially, Wang et al., 2023 provide a series of experiments to see that by means of a Bernoulli diffusion process it is empirically more efficient to train the model, and it is possible to achieve similar results to the ones obtained by LDM, with 16 times less inference steps, without using any test-time acceleration techniques. This puts the BLDM model in a promising position to be used in real applications and test it in different data modalities. With this insights, we have applied, for the first time, the BLDM to the specific case of text-to-motion generation, the results and implementation of which can be found in the appendix.

Finally, all the components discussed in the thesis merge up into algorithms 4 and 5, which display how to train and sample from a BLDM, respectively. Here we describe with the detail how to train a DBDM, with the Residual Prediction as target, in the latent space of a BVAE. The training for the Direct Prediction target is analogous.

Algorithm 4 Training DBDM in BLDM with residual target.

```
1: Input:
2:   $\mathcal{D}$: Dataset with conditioning.
3:   $\tau$: Conditioning encoder.
4:   $(\mathcal{D}, E_{\theta'}, D_{\theta''})$: BVAE pretrained on $\mathcal{D}$ with STE (or Gumbel-Softmax) and 1.
5:   $T$: Diffusion steps.
6:   $q, \{\beta_t\}_{t=1}^{T}$: Bernoulli forward trajectory, and noise schedule satisfying 3.14.
7:   $f_\theta$: Neural Network parameterized by $\theta$.
8: Output:
9:   $\theta$: Learned parameters.
10: Algorithm:
11: $\theta \leftarrow$ Initialize parameters
12: while SGD not converged do
13:     $(M, \mathbb{C}) \sim \mathcal{D}$.    ▷ Random minibatch of $M$ data samples and $C$ conditions.
14:     $L \leftarrow 0$    ▷ Initialize loss.
15:     for $(\mathbf{x}, \mathbf{c}) \in (M, C)$ do
16:         $t \sim \text{Uniform}\{1, \dots, T\}$    ▷ Taking a random diffusion step.
17:         $\mathbf{z}^{(0)} = E_{\theta'}(\mathbf{x})$    ▷ Obtaining binary representation for the data sample.
18:         $\mathbf{z}^{(t)} \sim q(\mathbf{z}^{(t)}|\mathbf{z}^{(0)})$    ▷ Sample from forward trajecotry.
19:         $f \leftarrow f_\theta(\mathbf{z}^{(t)}, t, \tau(\mathbf{c}))$    ▷ Obtaining the Residual Prediction.
20:         $\mathbf{z}^{(0)} \sim \mathbb{B}\left(\mathbf{z}^{(0)}; (1-\mathbf{z}^{(t)}) \odot f + \mathbf{z}^{(t)} \odot (1-f)\right)$ ▷ Recover original sample using 3.20
21:         $p_\theta(\mathbf{z}^{(t-1)}|\mathbf{z}^{(t)})$    ▷ Compute using $z^{(0)}$, $f$, and $z^{(t)}$, from 3.18.
22:         $L += L_{hybrid}$    ▷ Compute loss and its gradients.
23:     end for
24:     Average the aggregation of losses and gradients to obtain a Monte Carlo estimator.
25:     Update $\theta$ using SGD optimizer.
26: end while
```

Algorithm 5 Sampling from BLDM trained with residual target.

```
1: Input:
2:   $\mathbf{c}$: Condition.
3:   $\tau$: Conditioning encoder.
4:   $(\mathcal{D}, E_{\theta'}, D_{\theta''})$: BVAE pretrained on $\mathcal{D}$ with STE (or Gumbel-Softmax) and 1.
5:   $(\mathcal{D}, T, q, \{\beta_t\}_{t=1}^{T}, f_\theta)$: DBDM pretrained on $\mathcal{D}$ with Residual Prediction and 4.
6: Output:
7:   $\mathbf{x}$: New sampled datapoint.
8: Algorithm:
9: $\mathbf{z}^{(T)} \sim \mathcal{B}(\mathbf{z}^{(T)}; 0.5)$
10: for $\{t = T; t \in \{T, \dots, 1\}; t \leftarrow t-1\}$ do
11:     $f \leftarrow f_\theta(\mathbf{z}^{(t)}, t, \tau(\mathbf{c}))$
12:     $\mathbf{z}^{(0)} \sim \mathbb{B}\left(\mathbf{z}^{(0)}; (1-\mathbf{z}^{(t)}) \odot f + \mathbf{z}^{(t)} \odot (1-f)\right)$    ▷ Denoise sample using 3.20
13:     $p_\theta(\mathbf{z}^{(t-1)}|\mathbf{z}^{(t)})$    ▷ Compute using $\mathbf{z}^{(0)}$, $f$, and $\mathbf{z}^{(t)}$, from 3.18.
14:     $\mathbf{z}^{(t-1)} \sim p_\theta(\mathbf{z}^{(t-1)}|\mathbf{z}^{(t)})$    ▷ Sample from reverse trajectory.
15: end for
16: $\mathbf{x} = D_{\theta''}(\mathbf{z}^{(0)})$
```

4 Conclusion

In this thesis we have formalized the definition of Bernoulli diffusion and Bernoulli Variational Autoencoders. To this end, we first reviewed some basic concepts of probability theory and probabilistic models. We then stated the definition of generative models and the generative problem. In the following chapter, we tackled the first component of a Binary Latent Diffusion model. It started with the foundation BVAEs, alongside the data compression problem, the first deterministic Autoencoders, as well as Variational Autoencoders and is discrete counterpart. During that chapter, Variational Inference was the main branch of study, we established the strong relationship between the inference problem from classical VI and the generative problem from VAEs. There are several ways to adapt VAEs into a discrete space. We have reviewed two prominent methods, namely Vector Quantization with Straight-Trough-Estimator and Gumbel Softmax trick. These ideas were shaped into BVAEs. The third chapter introduced the concept of Diffusion Models. We went through a detailed study on a particular type of DMs, namely the Discrete-time DMs. We have seen the details on Discrete-time Gaussian DMs, widely studied in literature and with application to several fields in Machine Learning. Then we have translated the properties from Gaussian diffusion that make DGDMs suitable as generative models, are also applicable to define Discrete-time Bernoulli DMs, the second component of the Binary Latent Diffusion model. Then, we briefly related Discrete DMs to other approaches to diffusion processes for generative modelling, and introduced the tools used in ML for guided generation. Finally, we concluded the chapter by merging all concepts into a single model, the Binary Latent Diffusion Model. The last chapter discusses an application of the BLDM to the specific case of Text-to-Motion generation, providing the insights on how this model relates to the state-of-the-art and how it can be adapted to the specific needs of motion data. The detailed implementation and results of the experiments can be found in the appendix. Despite the recent advancements in DMs and the growing interest in the field, the focus in mainly targeted to Gaussian DMs. This thesis shares light on the path of exploring other types of diffusion models by providing a self-contained foundation for BLDMs.

With this project we conclude that the solid foundation for VAEs and DGDM can be formulated also for a binary alternative. We conclude that assuming a Bernoulli distribution in a latent space and learn a stochastic mapping between data samples in $\mathbb{R}^n$ and latent variables in $0, 1^d$, not only is an alternative for a data compression process, but also can take profit of the powerful capabilities of neural networks and optimization algorithms, e.g., by means of continuous relaxation. Also we conclude that Discrete Bernoulli DM is a promising generative model with a strong theoretical foundation. The fact that we are working with maybe the most simple distribution, the Bernoulli distribution, allows us to simplify many of the results using basic properties of discrete Markov chains, without the need of harder machinery from stochastic differential equations. Despite this theoretical evidence, few practical experimentation has been done applying DBDMs. We encourage researchers to try this approach in their use cases.

As a prospective course of action, there are many open problems yet to be solved. We have cited many works that prove their improvements empirically, each of them modifying the training objective in order to simplify it. Therefore, more efforts are required to properly prove why such changes improve the performance of the models. For example, Wang et al., 2023 argue that a binary latent space is much more efficient, i.e., "encodes better the information" of data into the latent variable, but this must be formalized and properly investigated. Furthermore, as we have defined VAEs and DMs, they are largely based on Bayesian Inference, where a distribution for the prior and posterior have to be assumed. Although this approach has been proven to be viable for high quality generation, other branches of statistics such as Conformal Inference, which gives an uncertainty interval rather than an output guessed by a prior, may be worth exploring for defining more comprehensible DMs. These could share light into one of the main problems in generative modelling, namely the "Hallucination problem", where a model generates fallacious sampled that not match with reality and were not present in the dataset. Finally, it is still an open problem to understand the inner workings of neural networks, which we are using to approximate the variational distributions. But more research is required to understand how the information is encoded in the parameters of the model, on how this is embedded into a meaningful latent space.

A Application and Implementation

The recent advancements in generative models and conditioning techniques, enables addressing the multimodality problem, involving the generation of samples from different data modalities (e.g., images, text, and audio) within a single model, by conditioning the data inputs from one to another. Multimodal models typically aim to estimate partial conditioned distributions $q_{\text{data}}(\mathbf{x}|\mathbf{c})$, where $\mathbf{c}$ represents the conditioning data. When $\mathbf{x}$ and $\mathbf{c}$ pertain to different data modalities, the model deals with multimodality. This opens avenues for addressing more complex tasks, such as text-to-image (Liu et al., 2023, Zhang, Rao, and Agrawala, 2023), text-to-3D (Xu et al., 2023), or conditioned video generation, where the goal is to generate temporally coherent images, often accompanied by audio, in response to a given text prompt (Ho et al., 2022, Qi et al., 2023).

One of the most challenging multimodal tasks is text-to-motion generation, which involves synthesizing human motion sequences from natural language descriptions. This task is particularly difficult due to the complexity of human motion, which is non-linear and articulated, see figure A.1 and A.2.

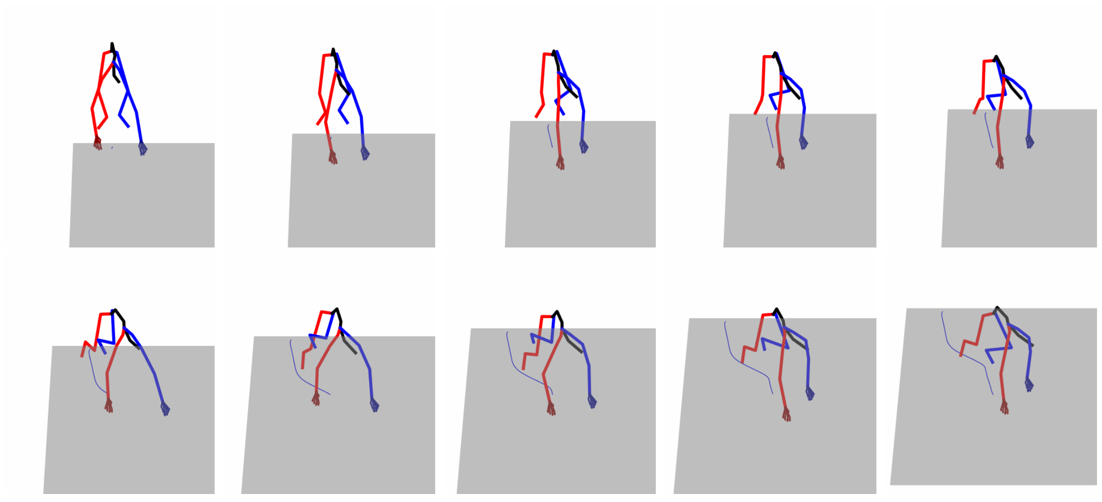


FIGURE A.1: Extract of human motion with description "on hands and feet a person crawls four paces on an angle to the left, turns and crawls back, and then stands".

In this chapter we explore the use of binary latent diffusion models to generate human motion sequences from text descriptions, introducing Motion Binary Latent Diffusion (MBLD). We evaluate the model on the Human3.6M dataset, a large-scale dataset of human motion sequences, and commonly used metrics for human motion. The results show that MBLD is able to encode motion in a binary latent space which is semantically rich enough to generate human motion sequences aligned with the conditioning text.

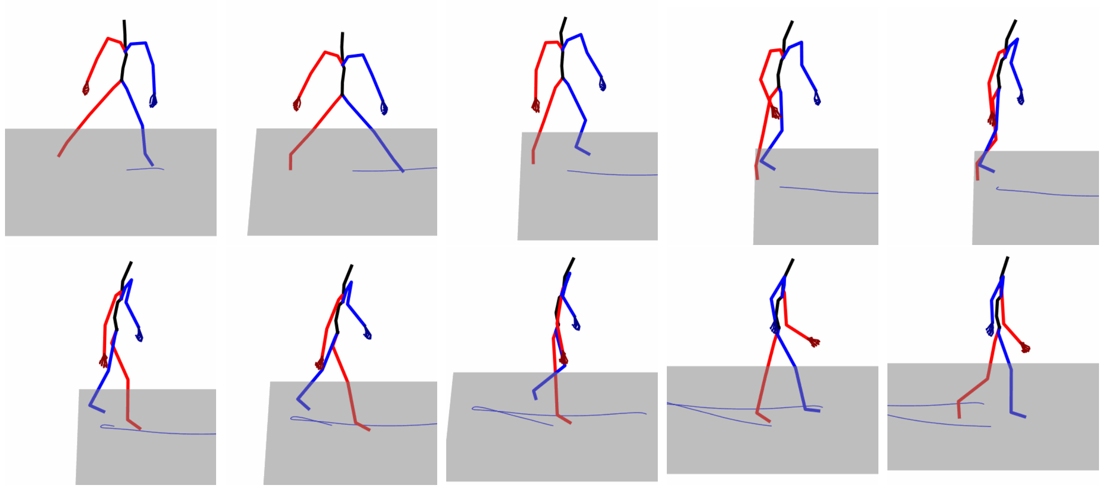


FIGURE A.2: Extract of human motion with description "a person walks forward, turns and walks back".

A.1 Problem Statement: Text-to-Motion Generation

Consider $\mathcal{D}$ a dataset of pairs $(\mathbf{x}, \mathbf{c})$, where $\mathbf{x} \in \mathbb{R}^{DxL}$ is a sequence of L frames of dimension D, and $\mathbf{c}$ is a text prompt describing the movement represented in $\mathbf{x}$. The goal of text-to-motion generation is to estimate the conditional distribution $q(\mathbf{x}|\mathbf{c})$, by tunning the parameters $\theta \in \mathbb{R}^N$ of the model $f_\theta(\mathbf{x}, \mathbf{c})$.

Such problem encounters several critical challenges. Firstly, the scarcity of large-scale motion datasets and the inherent complexity of non-linear and articulated human motion, pose common obstacles in the realm of Human Motion Generation. Secondly, the absence of a clear and well-defined mapping between text and motion requires models to learn intricate mappings capable of capturing the semantic nuances of text and generating diverse, plausible, and realistic sequences of movements. Thirdly, the absence of a clear and well-defined evaluation metric complicates the comparison of different models' performance.

Despite these challenges, several models have been proposed. The raise of diffusion-based models and VQ-VAEs has been a key factor in the development of the this models, dividing the field into two main approaches: Gaussian diffusion models on one side, and discrete representations with transformers[1] on the other side.

On one hand, inspired by the success of diffusion models in the field of image generation, efforts have been made to apply these models to human motion generation. Two of the worth mentioning attempts are Zhu et al., 2023 and Chen et al., 2023a, presenting Motion Diffusion Models (MDM) and Motion Latent Diffusion (MLD) respectively. Both approches, MDM and MLD, implement a Gaussian Diffusion model with classifier-free guidance 3.5, to align the text and motion representations. The main difference between the two models is the way they encode the motion data. MDM directly apply the diffusion process in the motion data space, while MLD first trains a VAE that encodes the motion into a discrete latent space, and then apply diffusion.

Within the two approaches, MLD present better results than MDM. However, both models demonstrate that diffusion-based techniques are able to properly harness the low-frequency signals of human motion and generate realistic and diverse motions. Despite of this, there are still some limitations that need to be addressed. First, the diffusion process is still slow and resource demanding. Second, the models are not able to generate long sequences of motion.

[1]A type of autoregressive model based on attention mechanisms

On the other hand, auto regressive models have been the natural candidates to handle sequence to sequence problems, and T2M can be seen as such. By factorizing distributions over the time dimension, predictions can be conditioned on past sequences of arbitrary length. However, the main drawback of these models is that they are costly and inefficient to train. Furthermore, during inference, the models accumulate errors over time, which leads to drifts and artifacts (Fragkiadaki et al., 2015, Martinez, Black, and Romero, 2017). Inspired by the success of quantization techniques in the field of image generation, several works have proposed the use of discrete representations for motion generation: PoseGPT (Lucas et al., 2022), T2M-GPT (Zhang et al., 2023b), and MotionGPT (Jiang et al., 2023).

Using a VQ-VAE to encode the motion allows to train a model in a lower-dimensional discrete space. Thus, an auto regressive learns to generate the motion by predicting a discrete sequence of tokens, rather than regressing the motion directly. Heuristically, quantizing motion data into discrete tokens can be seen as a way to reduce a complex and high-dimensional problem into a sequence of "words", which can be handled by a language model, i.e. a transformer. We will review in detail these models, since they are closely related to the work of this thesis.

The method achieved state-of-the-art results, proving to be a competitive alternative to the diffusion-based models. However it has a main drawback. Authors observed a slight jitter on the legs and hands movement, they claim that this is due to the VQ-VAE architecture, and proposed that with a better design could it be solved.

Furthermore, common to all the models seen until now, the models might miss some details of the motion when given a long description. This is due to the problems inherited from the text embedding layer, that is not able to capture the entire semantics of a long text and encode it into a fixed length vector.

Despite the promising results obtained by MotionGPT, one limitation of the model is that they assume that motion can be represented as a sequence of words. While this assumption seems plausible, in practice the codebook of the VQ-VAE is very limited to K vectors, which might not be enough to represent the entire "motion vocabulary". Based on this, we propose using a Binary Latent Diffusion model, with a quantization strategy that permits encoding motion token in a wider vocabulary using a binary space. Recall that in the previous chapter 2.3, we saw that the integers of the codebook can be seen as one-hot vectors of size K, allowing only K different tokens. However, a binary latent vector of the same size can represent 2^K different tokens, a much higher degree of information.

These methods claim that motion data is highly redundant, specially when recorded at high FPS. This redundancy can be exploited by a VAE and further compressed into a discrete latent space with a VQ-VAE. We propose a similar approach, but using a compact yet expressive latent space, given by a binary VAE 2.2. In other to achieve T2M, we propose adapting the Bernoulli diffusion process to the binarized motion data, exploiting the diffusion model's generation capabilities.

A.2 Motion Binary Variational Autoencoder

Let the data domain $\mathcal{D}$ be a human motion dataset, $\mathbf{x}_{1:L} \in \mathcal{D}$ a motion sequence of length L, and $\mathbf{c}_{\mathbf{x}}$ a text prompt describing the movement. Each frame of the movement consists of a human pose represented by a vector of rotations of dimension J, thus $\mathbf{x}_i \in \mathbb{R}^J$, for $i = 1, ..., L$. Our goal is to define a Motion Binary VAE (MBVAE) similar to 2.2, that encodes the motion sequence into a binary latent and reconstructs the original motion from it. Within this section we provide a discussion on how to define the MBVAE and test different versions inspired from the VQ-VAEs of the models from chapter 3. We will experiment with three different types of latent space. First, we encode the motion frame-wise, that is, each frame is encoded into a binary vector. Second, chunks of frames are encoded into a single binary vector, reducing the length of the latent motion. Third, the whole sequence is encoded into a single binary vector. We will refer to these three types of latent space as *frame-wise*, *some-frames* and *full-sequence* respectively.

A.2.1 Frame-wise binary quantization

Directly applying the binary VAE from 2.2 to the motion sequence requires to encode each frame into a binary vector. To this purpose, the convolutional layers of the original model are replaced by 1-dimensional convolutional layers. Thus, if $\mathbf{x}_i \in \mathbb{R}^J$ is the i-th frame of the sequence, the encoder extracts the local relation between the rotations of the pose and encodes it into a binary vector of reduced dimension. To be more precise, a linear embedding layer is applied to the input pose, reducing its dimension to $J' < J$. Then, a 1D convolutional layer with kernel size 3 and stride 1 is applied to the embedded pose. We use c' of such filters, thus the output after the convolutional step is a tensor of dimension $c' \times J'$. Then a block of 2 residual layers, an attention layer and a final convolutional layer with kernel size 3 and stride 2 are applied to the output tensor. The stride of the last CNN reduces the dimension of the latent joints by a factor of 2, performing as a downsampling layer. Repeating this process l times leads to a latent representation of dimension $c \times J'/2^l$. Finally, another convolution with c filters kernel size 3 and stride 1 outputs a real-valued tensor of dimension $c \times J'/2^l$. The latent space is then obtained by applying a sigmoid function to the output tensor, and then a binary quantization. The decoder is the inverse of the encoder, with nearest neighbor interpolation layers as upsamplers.

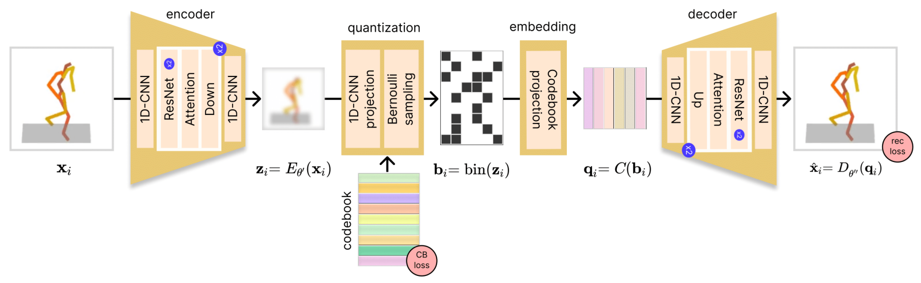
We propose two different ways of quantizing the motion latent space.

- **Binary:** As a first approach, we apply the binary quantization straightforwardly to the latent space. This is, sampling from a Bernoulli distribution with probability of success equal to the latent value, as in 2.3.3. Therefore, for any $\mathbf{x}_{1:L}$, and given the encoder $E_{\theta'}$, for $i = 1, ..., L$, the latent representation of the i-th frame is $\mathbf{z}_i = E_{\theta'}(\mathbf{x}_i)$, and the binary latent representation is $\mathbf{b}_i \sim \text{Bernoulli}(\mathbf{z}_i)$. The latent motion sequence is then $\mathbf{b}_{1:L} = (\mathbf{b}_1, ..., \mathbf{b}_L)$. The decoder $D_{\theta''}$ reconstructs the original motion from $\mathbf{b}_{1:L}$.

- **BinaryVQ:** The second approach is to apply the binary quantization prior to the VQ-VAE. That is, we define a VQ-VAE as in 2.1, with codebook of size K, and then apply the binary quantization to the latent space. Thus, for any $\mathbf{x}_{1:L}$, given the encoder $E_{\theta'}$, and the codebook C_K, for $i = 1, ..., L$, the latent representation of the i-th frame is $\mathbf{z}_i = E_{\theta'}(\mathbf{x}_i)$, and the binary latent representation is $\mathbf{b}_i = \text{bin}(\mathbf{z}_i)$. The binary quantization function $\text{bin}(\cdot)$ is composed by a projection layer that maps $\mathbf{z}_i \in \mathbb{R}^{c \times J'/2^l}$ to $\mathbf{b}_i \in \{0,1\}^{K \times J'/2^l}$, and a Bernoulli sampling layer. The resulting binary tensor $\mathbf{b}_i$ is then passed to the vector quantization layer, where the binary tensor is multiplied by the codebook C_K to obtain the quantized tensor $\mathbf{q}_i$. Heuristically, in this approach, each column of the tensor $\mathbf{b}_i$ is a binary vector of length K, with ones in the positions of the vectors of the codebook that best represent the latent joint of the pose. The decoder $D_{\theta''}$ reconstructs the original motion from $\mathbf{q}_{1:L}$. Observe that both approaches are equivalent, since the decoder $D_{\theta''}$ is the same in both cases, and in the first method it could learn an implicit codebook inside its convolutional and residual layers. Figure A.3 shows a comparison between both of them.

A.2.2 Some-frames binary quantization

Although the frame-wise representation successfully encodes the motion sequence into a robust binary latent space, it has a drawback. Poses in movement data are highly correlated and



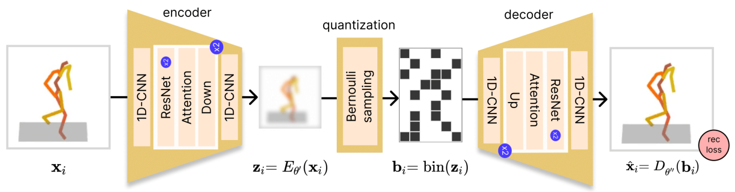

encoder
1D-CNN
ResNet
Attention
Down
1D-CNN
x2
quantization
Bernoulli sampling
decoder
1D-CNN
Up
Attention
ResNet
1D-CNN
rec loss
$\mathbf{x}_i$
$\mathbf{z}_i = E_{\theta'}(\mathbf{x}_i)$
$\mathbf{b}_i = \text{bin}(\mathbf{z}_i)$
$\hat{\mathbf{x}}_i = D_{\theta''}(\mathbf{b}_i)$

FIGURE A.3: Top: MBVAE with BinaryVQ. Bottom: MBVAE with Binary. The latter is equivalent to the former, since the decoder could learn an implicit codebook inside its convolutional and residual layers. As input, a pose in your favourite motion representation format. As output, its reconstruction.

redundant, specially when recorded at high frame rates. Therefore, encoding each frame into a binary vector can be inefficient. To overcome this problem, in spirit of the VQ-VAE from Zhang et al., 2023b and Jiang et al., 2023, we propose applying the 1D convolutional layers along the temporal dimension, and then binarize the latent space. Therefore, model is the same as the frame-wise binary quantization, but $\mathbf{x}_{1:L}$ is processed sequentially, that is, the kernel of the 1D CNNs moves along the L-dimensional temporal sequence, combining the frames the kernel size permits. With the same notation as before, and the new direction of the convolutional layers, the encoder extracts the local temporal relation between the rotations of the pose and outputs $\mathbf{z} \in \mathbb{R}^{c' \times L/2^l}$. Here, c' can be seen as the latent joints and $L/2^l$ as the number of latent frames. Binary quantization and decoding also remain the same, only considering the new dimensions of the latent space.

When applying the binary quantization, each of the binary columns of length K of $\mathbf{b} = \text{bin}(\mathbf{z})$ represents a latent frame. Therefore, the ones in the binary vector indicate which are the elements of the codebook that best represent the entire latent frame. Observe that this representation is more compact than the frame-wise one, since it encodes the whole latent frame into a single binary vector A.4.

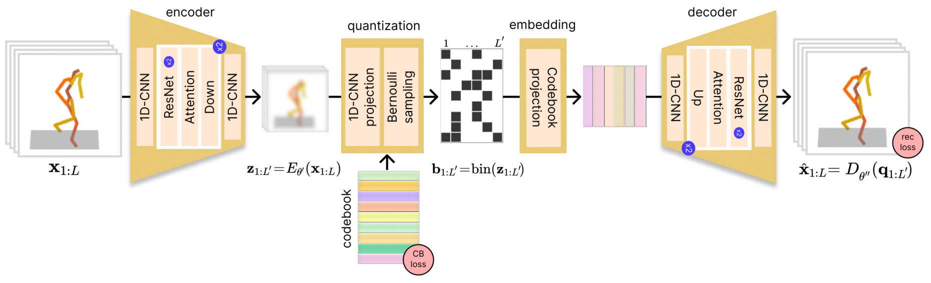


FIGURE A.4: MBVAE with BinaryVQ and some-frames binary quantization. As input, a sequence of the entire motion.

A.2.3 Full-sequence binary quantization

The semantic meaning of a movement is not only encoded in the poses, but also in the temporal relation between them. In this sense, the frame-wise and some-frames binary quantization methods may not be able to capture the whole significance of the motion. Therefore, we propose using the full-sequence binary quantization, that is, the encoder processes the whole sequence at once, and then binarizes the latent space. Then a transformer decoder manages the latent representation as cross-attention context, and reconstructs the original motion given a sequence of zero-motion tokens $\mathbf{0}$, as in MLD. Thus, $\mathbf{x}_{1:L}$ is processed all at once by the transformer encoder $E_{\theta'}$, and outputs the mean $\mu_{\mathbf{x}}$ and variance $\sigma_{\mathbf{x}}$ of a Gaussian distribution $\mathcal{N}(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}}) = \mathcal{N}(E_{\theta'}(\mathbf{x}))$. The binary quantization is done after the sampling step, $\mathbf{z} \sim \mathcal{N}(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}})$, thus $\mathbf{b} = \text{bin}(\mathbf{z})$. The decoder $D_{\theta''}$ reconstructs the original motion from $\mathbf{b}$ and $\mathbf{0}$, leading to a sequence of poses $\hat{\mathbf{x}}_{1:L} = D_{\theta''}(\mathbf{b}, \mathbf{0})$. From the binary vector quantized perspective this can be seen as choosing the best words from the codebook that better represent the entire sequence, see Figure A.5.

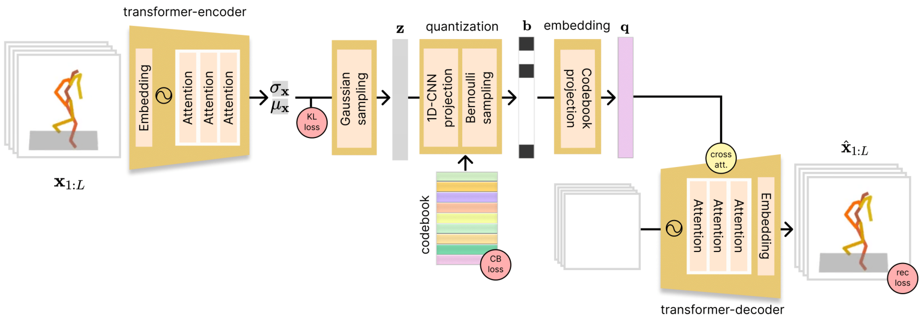


FIGURE A.5: MBVAE with BinaryVQ and full-sequence binary quantization. Input is also a sequence of the entire motion.

A.3 Motion Binary Latent Diffusion Model

With a learned binary latent space, we propose adapting the Binary Latent Diffusion model to motion data, introducing the Motion Binary Latent Diffusion (MBLD) model. The main difference with the original model relies within the architecture of the denoising function. In the original model, they use a transformer decoder with 2-dimensional attention layers. However, in our case, the latent space is a binary vector, hence we propose using a transformer decoder with 1-dimensional self-attention layers along the temporal axis that we denote by f_θ. The reason why we use the same architecture as in the MDM model is because it already has been proven to work well for motion data in the raw motion domain, and we expect it to work well in the binary latent domain, since the binary latent representation $\mathbf{b}_{1:L'}$ can be seen as a motion sequence of length L'. Notice that $\mathbf{b}_{1:L'}$ has latent frames and latent joints or rotations, in concordance with the input from the original model of the MDM.

We explore the performance of the model with the frame-wise binary quantization. In this case, the sequence $\mathbf{x}_{1:L}$ with textual description $\mathbf{c}$ is encoded to $\mathbf{b}_{1:L}$, and the Bernoulli Diffusion Process **??** is applied. The denoising function f_θ is trained to predict the flip probability of each bit at each step of the diffusion process, given the text contidion. Recall that the procedure begins with $\mathbf{b}_{1:L}^{(0)}$, where the superindex indicates the step of the diffusion Markov chain. Then, at each step t, f_θ estimates the binary tensor $\mathbf{b}_{1:L}^{(0)} \oplus \mathbf{b}_{1:L}^{(t)}$, where $\oplus$ is the element-wise XOR operation, or the original sample $\mathbf{b}_{1:L}^{(0)}$, as in BLD. When training, a random step t is sampled from a uniform distribution $t \sim \mathcal{U}(0, T)$, where T is the number of

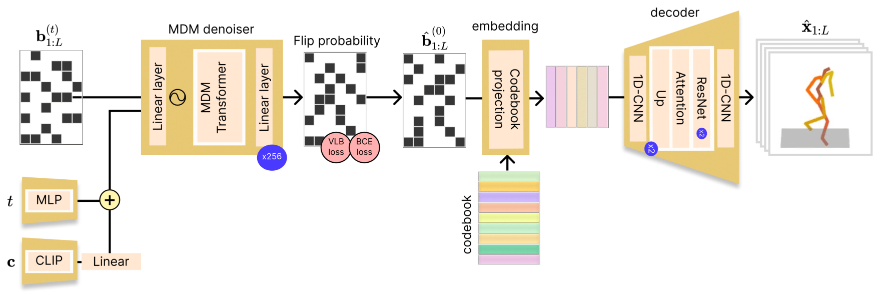


FIGURE A.6: MBLD with frame-wise binary quantization. When training $\mathbf{b}_{1:L}^{(0)}$ is a sequence of binarized poses, encoded from the target motion. When sampling, input is $\mathbf{b}_{1:L}^{(0)}$, a random binary tensor, and the models denoise it in $T = 256$ steps.

diffusion steps. Adding noise to the binary latent space is done by the schedule **??** defined in chapter 2, with a linear distribution of the β_t. Additionally, the text $\mathbf{c}$ is embedded into a vector of dimension d, projected to fit the dimension of the latent space by means of a linear layer, and concatenated to the binary tensor. Classifier-free guidance is used to condition the diffusion process with the text prompt 3.5, that is, at inference time, the denoising step is computed by the following equation,

$$(1+\omega)f_\theta(\mathbf{x}_t, \mathbf{c}, t) - \omega f_\theta(\mathbf{x}_t, \emptyset, t).$$

Therefore, f_θ must learn both, the conditional and unconditional denoising steps. Figure A.6 illustrates the diffusion process and sampling procedure.

A.3.1 Sampling

In order to generate new motion sequences from $\mathbf{c}$, we start from a random binary tensor $\mathbf{b}_{1:L}^{(t)}$ sampled from a Bernoulli distribution with probability of success 0.5. Then, we run the denoising process. At each step t we estimate $f_\theta\big(\mathbf{b}_{1:L}^{(t)}, \mathbf{c}, t\big)$ and $f_\theta\big(\mathbf{b}_{1:L}^{(t)}, \mathbf{0}, t\big)$, to compute the classifier-free guidance. Then, from 3.18 we can compute the transition kernel $p_\theta(\mathbf{b}_{1:L}^{(t-1)} | \mathbf{b}_{1:L}^{(t)})$ and sample the next binary tensor $\mathbf{b}_{1:L}^{(t-1)}$ from it. Repeating this process $T = 256$ times, we obtain a sequence of binary tensors $\mathbf{b}_{1:L}^{(0)}$. Finally, we reconstruct the original motion sequence by means of the decoder $D_{\theta''}$, that is, $\hat{\mathbf{x}}_{1:L} = D_{\theta''}(\mathbf{b}_{1:L}^{(0)})$. 4 and 5 illustrate the training and sampling algorithms.

B Experimental Setup and Results

B.1 Text-to-Motion Datasets

Common human motion datasets that include text descriptions are scarce. The most relevants are include KIT (Plappert, Mandery, and Asfour, 2016) and the recently released HumanML3D (Guo et al., 2022). Since the latter is bigger and common to all state-of-the-art models, we have decided to use HumanML3D. It is a large-scale dataset that contains $14,616$ motion sequences, along with $44,970$ sentences describing the motion. The motion data comes from HumanAct12 and AMASS (Mahmood et al., 2019), which are collections of several smaller motion captured datasets. Authors of HumanML3D have standarized the motion sequences to 20FPS and to a default human skeletal template. Motion sequences longer than 200 frames have been randomly cropped to fit this restriction. As a result each motion clip is of minimum and maximum length of 40 and 200 frames, respectively. Each pose $\mathbf{x}_i$ of the motion $\mathbf{x}_{1:L}$, is defined as a 263-dimensional vector, which includes positions, rotations and orientations of 22 joints, and a global root position.

B.2 Evaluation Metrics

Although assessing the quality of synthesized human movement is not a trivial task, several metrics have been proposed to measure the performance of the models from different perspectives. They can be summarized in three categories fidelity, diversity and condition consistency (Zhu et al., 2023, Chen et al., 2023b). Following the autors of MLD and T2M-GPT, we will focus on the following metrics; MSE, FID, DIV, MM, R-Precision and MMD, each of them measuring a different aspect of the model's performance.

Fidelity metrics aim to evaluate the overall quality of the generated motion.

1. **Mean Squared Error (MSE)**: measures the average of the squared differences between the prediction and the real motion. Relying solely on comparison to the ground truth joints and rotations is not enough to assess the quality of the generated motion, since countless of alternative sequences could be equally valid, but not similar to the ground truth.

2. **Fréchet Inception Distance (FID)**: estimates the distance between the distribution of a feature space of the generated motion and the ground truth. The metric leverages well-designed motion feature extractors, and uses the Fréchet distance between two multivariate Gaussians. To this end, given a generative model f_θ and real data $\mathcal{D}$, one can synthesize an artificial dataset $\mathcal{D}'$, to then fit $\mathcal{N}(\mu, \sigma)$ and $\mathcal{N}(\mu', \sigma')$, respectively. Then, the FID is computed as follows,

$$\mathrm{FID} = d_F(\mathcal{N}(\mu, \sigma), \mathcal{N}(\mu', \sigma'))^2 = \|\mu - \mu'\|_2^2 + \mathrm{tr}\big(\sigma + \sigma' - 2(\sigma\sigma')^{\frac{1}{2}}\big).$$

Diversity metrics aim to measure the model's ability to generate different motions.

1. **Diversity (DIV)**: measures the global diversity of the model by randomly splitting the generated dataset into two sets of the same size S. Then

$$\text{DIV} = \frac{1}{S}\sum_{i=1}^{S} \|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|_2^2,$$

where $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$ are two random samples from the two sets, respectively. In our experiments we set $S = 300$.

2. **MultiModality (MM)**: measures the diversity but conditioning to a set $\mathcal{C}$ of size C, of different text prompts. To this end, consider the generated motion sequences that satisfy a condition $c \in \mathcal{C}$, i.e. $\mathcal{D}'_{\mathbf{c}} := \{\hat{\mathbf{x}} \in \mathcal{D}' \,|\, \mathbf{c} \text{ is a description of } \hat{\mathbf{x}}\}$. Then, split $\mathcal{D}'_{\mathbf{c}}$ into two sets of the same size S', and compute the diversity as before,

$$\text{MM} = \frac{1}{C \cdot S'}\sum_{c\in\mathcal{C}}\sum_{i=1}^{S'} \|\hat{\mathbf{x}}_{\mathbf{c},i} - \hat{\mathbf{x}}_{\mathbf{c},j}\|_2^2,$$

where $\hat{\mathbf{x}}_{\mathbf{c},i}$ and $\hat{\mathbf{x}}_{\mathbf{c},j}$ are two random samples from the splits of $\mathcal{D}'_{\mathbf{c}}$, respectively.

Condition Consistency metrics aim to measure the accuracy of the model to generate motions that satisfy a given condition.

1. **R-Precision**: measures the accuracy of the model to generate motions that satisfy a given condition. It ranks the Euclidean distances among the features of the generated motion and the features of the ground truth, and averages the accuracy of the top-k results.

B.3 Implementation details

Throughout the training process, the models leverage the HumanML3D dataset, and akin to other methodologies, we employ CLIP ViT-B/16 as a text encoding prior. Such CLIP version, encodes text into a vector of dimension $d = 512$. This standardized approach ensures consistency and facilitates seamless integration with existing frameworks and datasets, contributing to the reproducibility and transparency of the research outcomes. The models are trained with the Adam optimizer and a learning rate of 10^{-4}. The hyperparameters of the training objective are set to $\omega_{\text{MSE}} = 1$, $\omega_1 = 1$, $\omega_{\text{KL}} = 0.1$. All versions run for 400 over the training set of 1528 motion samples from HumanML3D, which are no longer than $L = 196$, to avoid selecting randomly cropped clips, and no shorter than 150. We remain with 1528 different motion samples, with 4 descriptions each. Sequences are padded with zeros until reaching the maximum length. The linear projection fits the $J = 263$ joints and rotations of the motion poses into a $J' = 256$-dimensional vector. The codebook size is set to $K = 32$ for the binary vector quantized models. For frame-wise models, the downsampling factor is set to $l = 4$ and the inner channels of the encoder are $c' = 16$, which leads to a latent space of size 16×16. The some-frames models have downsampling factor $l = 2$ and a latens dimension $16 \times L'$, where $L' = 196/2^2 = 49$. This is different in the case of the full-sequence models, where $K = 256$ and the latent space for the entire motion is reduced to size 256, as MLD authors suggest (Chen et al., 2023b). The guidance weight for the Classifier-free Guidance is set to $\omega = 0.5$. Finally, the steps for the diffusion process and the sampling process are set to $T = 256$, as in the BLD (Wang et al., 2023).

All the implemented models are developed using PyTorch and trained efficiently on a single NVIDIA GeForce RTX 3090 GPU. The complete source code is accessible on GitHub at motion-binary-latent-diffusion.

B.4 Results

B.4.1 Comparisons on MBVAE

Experiments were undertaken to evaluate the performance of the MBVAE across various models. The comparison involved assessing the capabilities of straightforward binary quantization, binary vector quantization, and models without quantization. The goal of this comparison was to determine the most effective approach for representing motion in a binary latent space. Noticeably, the latent space dimensions for frame-wise are the largest, at $16 \times 16 \times 196$ for the complete motion, while the some-frames models have a latent space of size 16×49, and the full-sequence models have a latent space of size 256. Therefore, the frame-wise models have the most information about the motion at pose level. Consider also that the binary latent space of the vector quantized models is a bit larger, since it is prior to the codebook projection, thus, the binary latent spaces are of size $16 \times 32 \times 196$, 32×49 and 256, respectively. The latent space of no quantized models is the default 32-bit float, therefore we achieve up to a factor of 32 higher compresion rate. The table B.1 summarizes the memory footprint of a single motion sequence for each model embedded into the latent space.

Model	Binary	Binary-VQ	No-Quant
Frame-wise	50,176bits $\simeq$ 6.3kB	100,352bits $\simeq$ 12.6kB	1,605,632bits $\simeq$ 200.7kB
Some-frames	784bits $\simeq$ 0.1kB	1,568bits $\simeq$ 0.2kB	25,088bits $\simeq$ 3.1kB
Full-sequence	256bits $\simeq$ 0.03kB	256bits $\simeq$ 0.03kB	8,192bits $\simeq$ 1.02kB

Table B.1: Memory footprint of the different models for a latent representation of a motion sequence of 196 frames and 263 joints and rotations per pose.

The plots B.1 indicate that frame-wise models exhibit superior fitting to the reconstruction loss, primarily owing to their richer information content about motion at the pose level. Closely following are the some-frame models, which achieve a lower reconstruction loss than the full-sequence models. In terms of MSE and L1 losses, the Binary and Binary-VQ models perform similarly, while the no-quantized models achieve a lower loss. This difference is accentuated in the total loss, in the case of the VQ models, due to the codebook loss B.2. However, we can observe that if the binary space is large enough, i.e two times bigger in our case, the frame-wise Binary can overcome the No-Quant some-frames model. This is due to the fact that the binary latent space is able to capture the motion information. Therefore, with this simple training objective, the binary latent space is able to achive comparatively similar results to the No-Quant models, despite the need of a similar memory footprint in the latent space. This is a remarkable result, since it indicates that the binary latent space is able to represent motion information, but it needs further regularization to reduce the memory footprint effectively.

Regarding the codebook loss B.2, we can observe that all models with Biinary-VQ achieved a similar codebook even with different latent spaces and codebook sizes, 32 (frame-wise and some-frames) and 256 (full-sequence). The codebook loss ensures the binary latent space is as sparse as possible, choosing the minimum number of codewords to represent a binary vector. Therefore, it will never drop to zero, unless the model collapses. Thus, the fact that all models achieve similar codebook losses indicates that the proportion of codewords used to represent the binary latent space is comparable.

Full-sequence models face a trade-off between reconstruction loss and KL divergence, which hampers the training process. Interestingly, even when removing KL regularization, full-sequence models fail to achieve a lower reconstruction loss compared to frame-wise and some-frames models B.3. This discrepancy arises from their reduced ability to capture pose information, resulting in a latent space that inadequately represents motion.

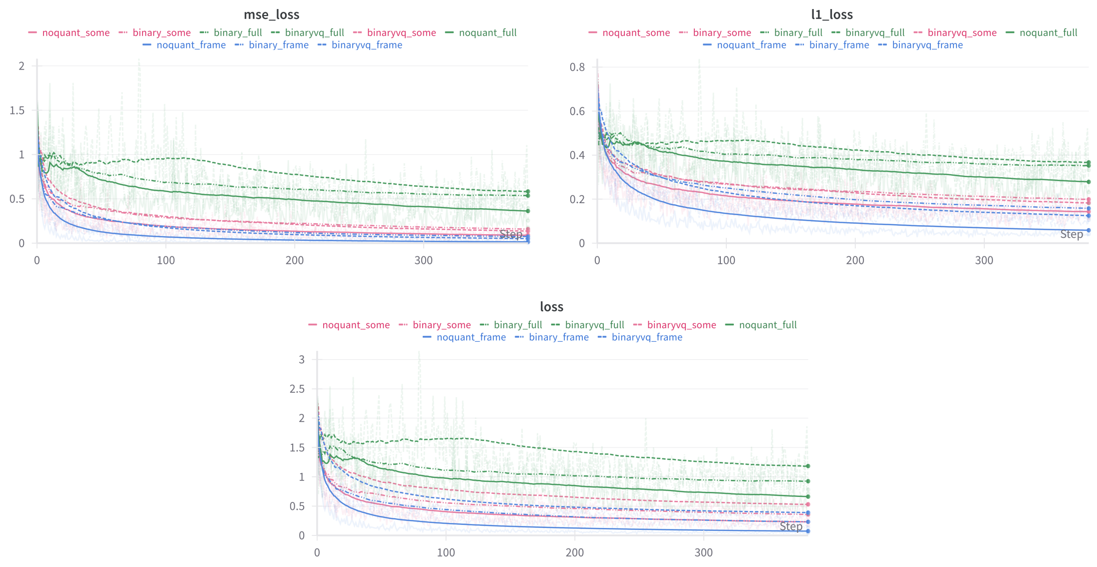


FIGURE B.1: Comparison of the training losses between **frame-wise**, **some-frames** and **full-sequence**.

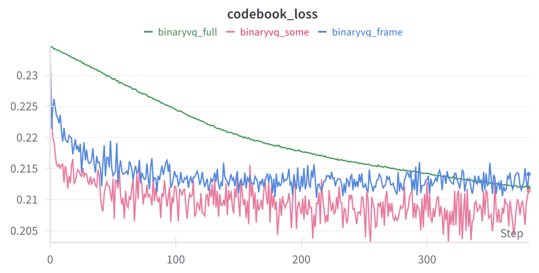


FIGURE B.2: Codebook loss of Binary-VQ versions of the models at training stage.

B.4.2 Exploration on MBLD

Since we observe that the binary latent space is able to represent motion information, we proceed to test the performance of the MBLD through a two-pronged approach.

Experiment 1: Frame-wise diffusion. Initially, a diffusion model is trained on the latent space of the frame-wise MBVAE, utilizing a codebook of size $K = 32$ for the BinaryVQ versions. Notably, we observed a pronounced sensitivity to hyperparameter tuning, particularly concerning the balancing weights of the losses. Remarkably, the model exhibits greater stability when estimating the original poses $\mathbf{b}_{1:L}^{(0)}$ instead of the flip probability—contrary to the suggestion of the BLD authors for image generation (Wang et al., 2023), see figure B.4. This is not surprising, since it is equivalent to predicting the original motion $\mathbf{x}_{1:L}^{(0)}$, which is the aim of the diffusion process in other relevant works, such as MDM (Zhu et al., 2023) or BeLfusion (Barquero, Escalera, and Palmero, 2023). As we can see in figure B.4, the model struggles to minimize the BCE loss when predicting the flip probability. We can also observe that the VLB regularization term is not able to minimize and bounces back an forth during the training process, depending on the level of noise of the sampled noise $\mathbf{b}_{1:L}^{(t)}$ of the Bernoulli diffusion process. Furthermore, if we take a closer look at the accuracy of the model, we can see that the overall trend is to increase, although also done in a very unstable manner. These results indicate that the model is able to recover the motion for the very first steps of

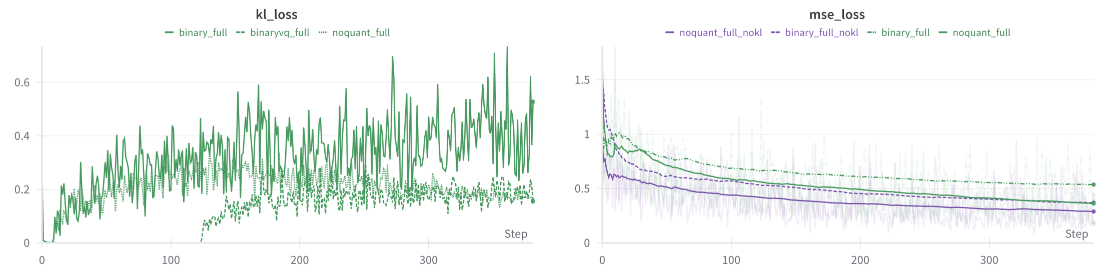


FIGURE B.3: Left: KL regularization of full-sequence models. Right: Comparison between models with and without KL regularization.

the diffusion process, but encounters challenges when denoising the motion for the inherently noisy steps of the diffusion process.

Despite struggling to achieve complete denoising and resulting in a somewhat shaky movement at inference time, the model successfully recovers the pose of the motion. This implies that pose information encoded in the binary latent space can be effectively reconstructed given a text prompt through a conditioned Bernoulli diffusion process, as depicted in Figure B.5.

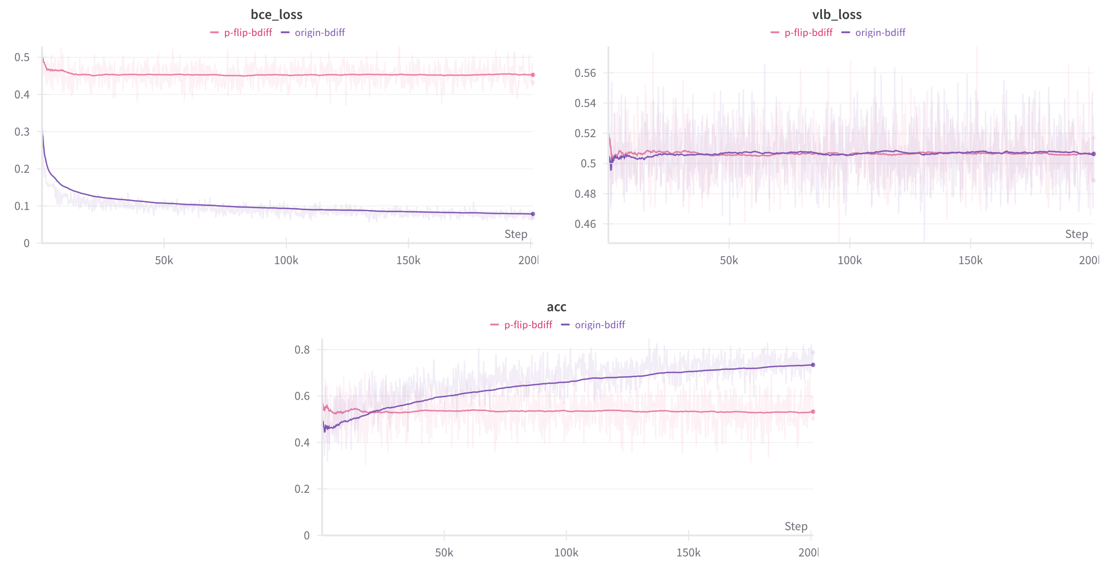


FIGURE B.4: Comparison between models with different prediction target: flip probability and original pose.

Experiment 2: Full-sequence diffusion. The initial experiment led us to the realization that the binary latent space derived from the frame-wise MBVAE lacks sufficient regularization for a denoising model to effectively approximate the reverse process of a Bernoulli diffusion. Consequently, in a subsequent iteration, a diffusion model was trained on the latent space of the full-sequence MBVAE. Despite yielding suboptimal results when compared with the other autoencoders, this model is subject to additional regularization through the inclusion of the KL loss, inherited from MLD (A.2.3).

In this second run, both the diffusion model and the MBLD model undergo training on identical datasets, employing identical hyperparameters and completing the same number of epochs (2000). Notably, both latent spaces maintain a consistent size, constituting a 256-dimensional vector —one continuous (8,192bits) and the other binary (256bits). Evaluation of the joint-level reconstruction loss, as depicted in Figure B.6, reveals that the MLD model

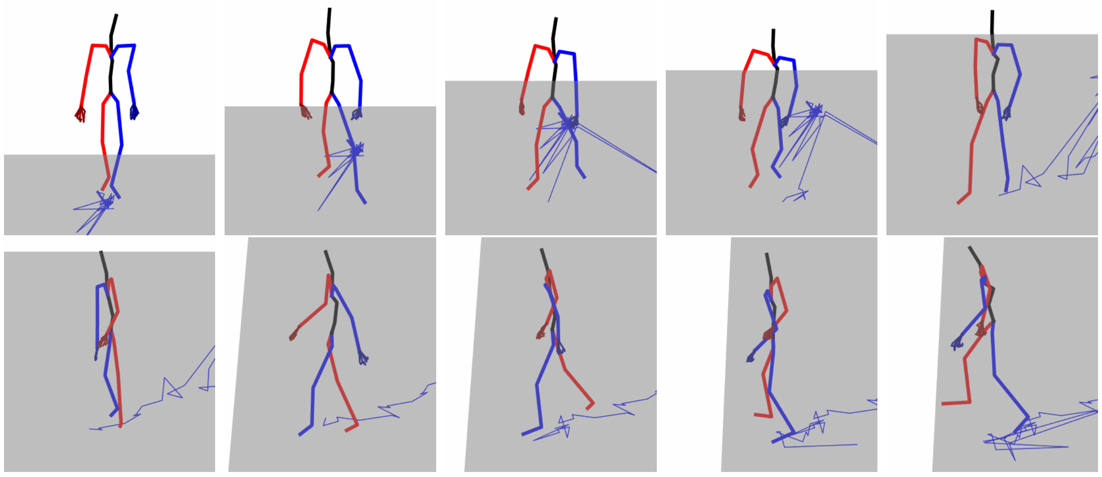


FIGURE B.5: Sampled motion: "A person walks forward and slightly to the left".

achieves a lower reconstruction loss compared to the Motion Binary Latent Diffusion (MBLD) model. It is noteworthy that the MBLD model exhibits signs of overfitting, as evidenced by the increasing reconstruction loss on the validation set. This indicates that a 256-bit binary latent space rapidly harnesses the information of the training set, and begins to memorize the training data without generalizing to the validation set. This may indicate that the training set is not sufficiently large to achieve a more robust binary latent space.

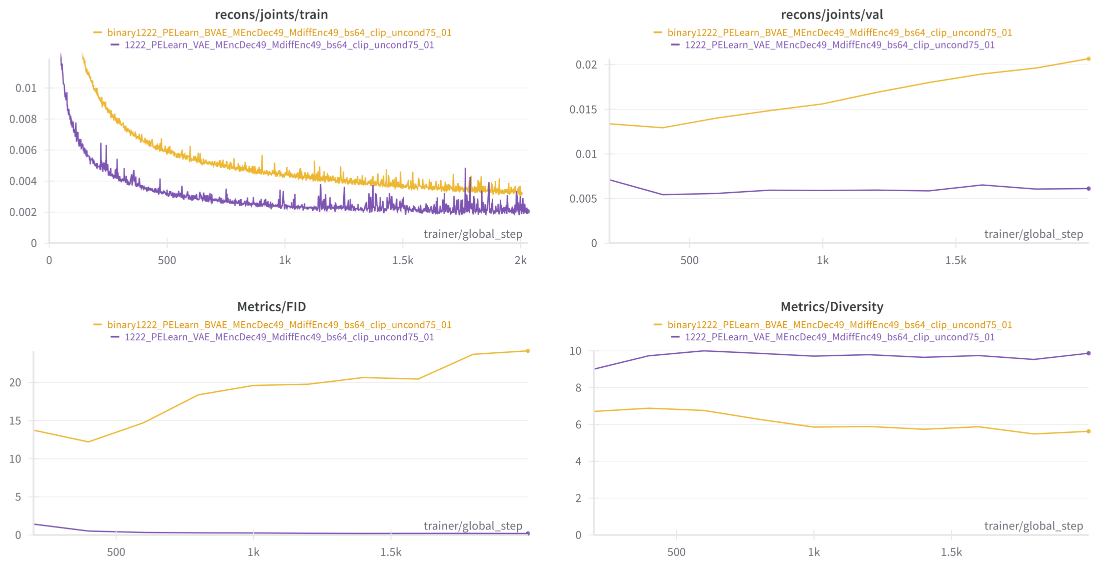


FIGURE B.6: Reconstruction loss at joint level obtained by the MLD and MBLD models. The MLD model is able to achieve a lower reconstruction loss than the MBLD model.

For a comprehensive comparison of metrics, refer to Table B.2. Firstly, it is important to note that the MLD model does not achive the good results reported by the authors (Chen et al., 2023b). In order to fit the computational demands, we have reduced the number of layers of the diffusion model, used a smaller version on CLIP, used a smaller subset of HumanML3D, and reduced the number of epochs. However, we are interested in comparing the results of the MLD and MBLD models, which are trained on the same conditions. The FID metric indicates what we have already seen in the plots; the MBLD, despite being able to recover the pose of the motion during training, does not generalize well to the test set increasing higher the FID. The other metrics, MM and DIV, indicate that both models achieve a similar degree of diversity, which is remarable since the MBLD latent is 32 times smaller. This suggests that the binary latent space is able to capture the motion information into a compact yet expressive representation. Furthermore, regarding the top-k accuracies of R-precision, both models behave similarly in terms of conditioning to a text prompt. The reason why there is a large gap between in FID, but not in the other metrics, is due to the fact that the FID, as well as the MSE and L1 losses, are fidelity metrics. Therefore, it is not surprising that if MSE and L1 overfit, the FID will also overfit. However, the other metrics measure different aspects of the motion, and remain similar. In light of these results, we can conclude that the binary latent space is able to harness the motion information, rapidly overfitting the training set, but not generalizing well to the test set. Hence, the binary latent space may still require further regularization and a larger training set to achieve a more robust representation of motion.

Model	**FID**	**MM**	**DIV**	**Top-1**	**Top-2**	**Top-3**
GT	-	-	9.5	0.514	0.705	0.797
MLD	17.504	4.744	5.284	0.03	0.062	0.093
MBLD	31.49	4.041	4.292	0.034	0.067	0.1

Table B.2: Metrics obtained by the MLD and MBLD models on the test set.

Bibliography

Austin, Jacob et al. (2021). “Structured Denoising Diffusion Models in Discrete State-Spaces”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., pp. 17981–17993. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

Baldi, Pierre (2012). “Autoencoders, Unsupervised Learning, and Deep Architectures”. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Ed. by Isabelle Guyon et al. Vol. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, pp. 37–49. URL: https://proceedings.mlr.press/v27/baldi12a.html.

Bank, Dor, Noam Koenigstein, and Raja Giryes (2021). *Autoencoders*. arXiv: 2003.05991 [cs.LG].

Barber, David (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Barquero, German, Sergio Escalera, and Cristina Palmero (2023). “BeLFusion: Latent Diffusion for Behavior-Driven Human Motion Prediction”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2317–2327.

Bengio, Yoshua (2013). “Estimating or Propagating Gradients Through Stochastic Neurons”. In: *ArXiv* abs/1305.2982. URL: https://api.semanticscholar.org/CorpusID:13985426.

Bishop, Christopher (2006). *Pattern Recognition and Machine Learning*. Springer. URL: https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/.

Chen, Xin et al. (2023a). *Executing your Commands via Motion Diffusion in Latent Space*. arXiv: 2212.04048 [cs.CV].

— (2023b). *Executing your Commands via Motion Diffusion in Latent Space*. arXiv: 2212.04048 [cs.CV].

Dhariwal, Prafulla and Alex Nichol (2021). *Diffusion Models Beat GANs on Image Synthesis*. arXiv: 2105.05233 [cs.LG].

Feller, William (1949). “On the Theory of Stochastic Processes, with Particular Reference to Applications”. In: URL: https://api.semanticscholar.org/CorpusID:121027442.

Folland, G.B. (2013). *Real Analysis: Modern Techniques and Their Applications*. Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Wiley. ISBN: 9781118626399. URL: https://books.google.dk/books?id=wI4fAwAAQBAJ.

Fragkiadaki, Katerina et al. (2015). *Recurrent Network Models for Human Dynamics*. arXiv: 1508.00271 [cs.CV].

Gu, Shuyang et al. (2022). “Vector Quantized Diffusion Model for Text-to-Image Synthesis”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10696–10706.

Gumbel, E.J. (1954). *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*. Applied mathematics series. U.S. Government Printing Office. URL: https://books.google.dk/books?id=l2sLIbIqjK4C.

Guo, Chuan et al. (2022). “Generating Diverse and Natural 3D Human Motions From Text”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5152–5161.

Hinton, Geoffrey E. (1999). "Products of experts". In: URL: https://api.semanticscholar.org/CorpusID:15059668.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 6840–6851. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Ho, Jonathan et al. (2022). *Imagen Video: High Definition Video Generation with Diffusion Models*. arXiv: 2210.02303 [cs.CV].

Hyvärinen, Aapo (2005). "Estimation of Non-Normalized Statistical Models by Score Matching". In: *Journal of Machine Learning Research* 6.24, pp. 695–709. URL: http://jmlr.org/papers/v6/hyvarinen05a.html.

"IEEE Standard for Binary Floating-Point Arithmetic" (1985). In: *ANSI/IEEE Std 754-1985*, pp. 1–20. DOI: 10.1109/IEEESTD.1985.82928.

Jang, Eric, Shixiang Gu, and Ben Poole (2017). "Categorical Reparameterization with Gumbel-Softmax". In: URL: https://arxiv.org/abs/1611.01144.

Jarzynski, Christopher (Mar. 2011). "Equalities and Inequalities: Irreversibility and the Second Law of Thermodynamics at the Nanoscale". In: *Annual Review of Condensed Matter Physics* 2.1, 329–351. ISSN: 1947-5462. DOI: 10.1146/annurev-conmatphys-062910-140506. URL: http://dx.doi.org/10.1146/annurev-conmatphys-062910-140506.

Jensen, J L W V (1906). "Sur les fonctions convexes et les inégalités entre les valeurs moyennes". In: *Acta Math.* 30.0, pp. 175–193.

Jiang, Biao et al. (2023). *MotionGPT: Human Motion as a Foreign Language*. arXiv: 2306.14795 [cs.CV].

Kingma, Diederik P and Max Welling (2014). *Auto-Encoding Variational Bayes*. arXiv: 1312.6114 [stat.ML].

Lever, Jake, Martin Krzywinski, and Naomi Altman (2017). "Principal component analysis". In: *Nature Methods* 14.7, pp. 641–642. ISSN: 1548-7105. DOI: 10.1038/nmeth.4346. URL: https://doi.org/10.1038/nmeth.4346.

Lim, Sungbin et al. (2023). "Score-based Generative Modeling through Stochastic Evolution Equations in Hilbert Spaces". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., pp. 37799–37812. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/76c6f9f2475b275b92d03a83ea270af4-Paper-Conference.pdf.

Liu, Haotian et al. (2023). *Visual Instruction Tuning*. arXiv: 2304.08485 [cs.CV].

Liu, Zhi-Gang and Matthew Mattina (July 2019). "Learning Low-precision Neural Networks without Straight-Through Estimator (STE)". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, pp. 3066–3072. DOI: 10.24963/ijcai.2019/425. URL: https://doi.org/10.24963/ijcai.2019/425.

Lucas, Thomas et al. (2022). "PoseGPT: Quantization-based 3D Human Motion Generation and Forecasting". In: *European Conference on Computer Vision (ECCV)*.

Maddison, C, A Mnih, and Y Teh (2017). "The concrete distribution: A continuous relaxation of discrete random variables". In: International Conference on Learning Representations, pp. 1–20.

Mahmood, Naureen et al. (Oct. 2019). "AMASS: Archive of Motion Capture as Surface Shapes". In: *International Conference on Computer Vision*, pp. 5442–5451.

Martinez, Julieta, Michael J. Black, and Javier Romero (2017). *On human motion prediction using recurrent neural networks*. arXiv: 1705.02445 [cs.CV].

Murphy, Kevin P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press. URL: probml.ai.

Murphy, Kevin P. (2023). *Probabilistic Machine Learning: Advanced Topics*. MIT Press. URL: http://probml.github.io/book2.

Nakkiran, Preetum et al. (2024). "Step-by-Step Diffusion: An Elementary Tutorial". In: *ArXiv* abs/2406.08929. URL: https://api.semanticscholar.org/CorpusID:270440143.

Neal, R.M. (2001). "Statistics and Computing". In: vol. 11. Springer. Chap. Annealed importance sampling, 125–139. URL: https://doi.org/10.1023/A:1008923215028.

Nichol, Alex et al. (2022). *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models*. arXiv: 2112.10741 [cs.CV].

Nichol, Alexander Quinn and Prafulla Dhariwal (2021). "Improved Denoising Diffusion Probabilistic Models". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 8162–8171. URL: https://proceedings.mlr.press/v139/nichol21a.html.

Nualart, David and Marta. Sanz-Solé (1990). *Curs de probabilitats*. cat. Estadística y análisis de datos ; 5. Barcelona: PPU. ISBN: 8476657188.

Oord, Aaron van den, Oriol Vinyals, and Koray Kavukcuoglu (2017). "Neural Discrete Representation Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

Pavliotis, Grigorios A. (2014). *Stochastic Processes and Applications: Diffusion Processes, the Fokker-Planck and Langevin Equations*. Vol. 60. Texts in Applied Mathematics. New York, NY: Springer. ISBN: 978-1-4939-1322-0. DOI: 10.1007/978-1-4939-1323-7.

Plappert, Matthias, Christian Mandery, and Tamim Asfour (Dec. 2016). "The KIT Motion-Language Dataset". In: *Big Data* 4.4, 236–252. ISSN: 2167-647X. DOI: 10.1089/big.2016.0028. URL: http://dx.doi.org/10.1089/big.2016.0028.

Podell, Dustin et al. (2023). *SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis*. arXiv: 2307.01952 [cs.CV].

Qi, Chenyang et al. (2023). *FateZero: Fusing Attentions for Zero-shot Text-based Video Editing*. arXiv: 2303.09535 [cs.CV].

Rombach, Robin et al. (2022). "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695.

Rumelhart, David E. and James L. McClelland (1987). "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pp. 318–362.

Sauer, Axel et al. (2023). *Adversarial Diffusion Distillation*. arXiv: 2311.17042 [cs.CV].

Shiryayev, A. N. (1992). "On Analytical Methods In Probability Theory". In: *Selected Works of A. N. Kolmogorov: Volume II Probability Theory and Mathematical Statistics*. Ed. by A. N. Shiryayev. Dordrecht: Springer Netherlands, pp. 62–108. ISBN: 978-94-011-2260-3. DOI: 10.1007/978-94-011-2260-3_9. URL: https://doi.org/10.1007/978-94-011-2260-3_9.

Sohl-Dickstein, Jascha et al. (2015). "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 2256–2265. URL: https://proceedings.mlr.press/v37/sohl-dickstein15.html.

Song, Yang and Stefano Ermon (2019a). "Generative Modeling by Estimating Gradients of the Data Distribution". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/3001ef257407d5a371a96dcd947c7d93-Paper.pdf.

Song, Yang and Stefano Ermon (2019b). “Generative Modeling by Estimating Gradients of the Data Distribution”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/3001ef257407d5a371a96dcd947c7d93-Paper.pdf.

Spinney, Richard and Ian Ford (Feb. 2013). *Fluctuation Relations: A Pedagogical Overview*. DOI: 10.1002/9783527658701.ch1. URL: http://dx.doi.org/10.1002/9783527658701.ch1.

Steck, Harald (2020). “Autoencoders that don't overfit towards the Identity”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 19598–19608. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/e33d974aae13e4d877477d51d8bafdc4-Paper.pdf.

Vincent, Pascal (2011). “A Connection Between Score Matching and Denoising Autoencoders”. In: *Neural Computation* 23.7, pp. 1661–1674. DOI: 10.1162/NECO_a_00142.

Wang, Ze et al. (2023). “Binary Latent Diffusion”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22576–22585.

Welling, Max and Yee Whye Teh (2011). “Bayesian Learning via Stochastic Gradient Langevin Dynamics”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. Ed. by Lise Getoor and Tobias Scheffer. ICML ’11. Bellevue, Washington, USA: ACM, pp. 681–688. ISBN: 978-1-4503-0619-5.

Xu, Jiale et al. (2023). *Dream3D: Zero-Shot Text-to-3D Synthesis Using 3D Shape Prior and Text-to-Image Diffusion Models*. arXiv: 2212.14704 [cs.CV].

Yang, Ling et al. (2023). “Diffusion Models: A Comprehensive Survey of Methods and Applications”. In: *ACM Comput. Surv.* 56.4. ISSN: 0360-0300. DOI: 10.1145/3626235. URL: https://doi.org/10.1145/3626235.

Yang, Ruihan, Prakhar Srivastava, and Stephan Mandt (2022). *Diffusion Probabilistic Modeling for Video Generation*. arXiv: 2203.09481 [cs.CV].

Zhang, Cheng et al. (2019). “Advances in Variational Inference”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.8, pp. 2008–2026. DOI: 10.1109/TPAMI.2018.2889774.

Zhang, Chenshuang et al. (2023a). *A Survey on Audio Diffusion Models: Text To Speech Synthesis and Enhancement in Generative AI*. arXiv: 2303.13336 [cs.SD].

Zhang, Jianrong et al. (2023b). *T2M-GPT: Generating Human Motion from Textual Descriptions with Discrete Representations*. arXiv: 2301.06052 [cs.CV].

Zhang, Lvmin, Anyi Rao, and Maneesh Agrawala (2023). *Adding Conditional Control to Text-to-Image Diffusion Models*. arXiv: 2302.05543 [cs.CV].

Zhu, Wentao et al. (2023). *Human Motion Generation: A Survey*. arXiv: 2307.10894 [cs.CV].