Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Alexander Johansen Ohrt

Probabilistic Tabular Diffusion for Counterfactual Explanation Synthesis

Master's thesis in Applied Physics and Mathematics
Supervisor: Kjersti Aas
June 2023

NTNU
Norwegian University of
Science and Technology

Alexander Johansen Ohrt

Probabilistic Tabular Diffusion for Counterfactual Explanation Synthesis

Master's thesis in Applied Physics and Mathematics
Supervisor: Kjersti Aas
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

Recent mainstream popularization of *artificial intelligence* (AI) has led to both positive and negative sentiments concerning the future of the technology. Several of the current most notable AI systems can be categorized as *deep generative AI*, a term that encompasses highly complex models capable of generating data from different modalities. Another subfield called *explainable AI* (XAI) aims to develop methods to increase understanding of opaque prediction models, an objective that both researchers and legislators continue to direct considerable efforts towards. An emerging, especially human-friendly technique from XAI corresponds to *counterfactual explanations*, which are valuable explanations for individual predictions. In this thesis, we combine these two seemingly contradictory subfields of AI, by applying deep generative models to synthesize counterfactual explanations.

Our main contributions are threefold. First, we develop an accessible and self-contained exposition of *diffusion probabilistic models*, the generative models that underpin several of the most successful technologies for generating data, for example, in art. Second, we add to the literature on diffusion models applied to *tabular data*, by dissecting and thoroughly explaining the key components of one such model. Third, we utilize the tabular diffusion model to generate counterfactual explanations, by altering one specific *model-agnostic* algorithm. The generative performance of the tabular diffusion model is evaluated on three publicly available, real datasets against two previously demonstrated models — one deep *variational autoencoder* model and one shallow *decision tree* model. Moreover, counterfactual explanations are computed using the three models as foundations, in order to evaluate their usefulness for explaining an arbitrary binary classifier.

In our experiments, we observe that all three models are able to generate tabular data and counterfactual explanations, but with differing levels of faithfulness and reliability. In fact, we do not find sufficient evidence to conclude that the considered diffusion model is superior to the baselines, neither at generating data from an approximated unknown joint distribution nor at generating counterfactual explanations for clarifying binary predictions on test observations. Due to promising results, we urge researchers to consider the out-of-the-box tree-based model as a reference during evaluation in further work on deep generative modelling for tabular data. Finally, we provide possible directions for future research on diffusion models for tabular data and counterfactual explanations.

Sammendrag

Nylig popularisering av *kunstig intelligens* (KI) har ført til både positive og negative utbrudd angående fremtiden til teknologien. Flere av den siste tidens mest omdiskuterte KI-systemer tilhører et område kalt *dyp generativ KI*, som er en samlebetegnelse for høyst kompliserte modeller i stand til å generere syntetiske data av ulike sorter. Et annet felt ved navn *forklarbar KI* har som mål å utvikle metoder for å øke forståelsen av ugjennomsiktige prediksjonsmodeller. Dette er en problemstilling som en stadig økende andel av forskere, lovgivere og brukere vier oppmerksomhet til. *Kontrafaktiske forklaringer* utgjør en spesielt spennende forklaringsmetode, fordi de er verdifulle forklaringer av individuelle prediksjoner, på samme tid som de er enkle for mennesker å forstå. I denne avhandlingen kombinerer vi de to nevnte, og tilsynelatende motstridende, underområdene av KI ved å bruke dype generative modeller til å generere kontrafaktiske forklaringer.

Vårt bidrag er tredelt. For det første gir vi en grundig og selvstendig, men likevel tilgjengelig, innføring i teorien som omhandler *diffusjonsmodeller*. Disse modellene er fundamentale i flere av de hittil mest vellykkede rammeverkene for å fremstille syntetiske data, for eksempel innenfor billedkunst. Deretter bidrar vi til forskningen innenfor diffusjonsmodeller med fokus på modellering av *tabulære data*. Dette gjør vi ved å gi en omfattende redegjørelse av én spesifikk nylig etablert modell, der vi fyller inn vesentlige detaljer som virker å mangle i den opprinnelige fremstillingen. Til slutt anvender vi den tabulære diffusjonsmodellen til å generere kontrafaktiske forklaringer, ved å modifisere en gitt *modellagnostisk* algoritme. Vi evaluerer den generative ytelsen til den tabulære diffusjonsmodellen på tre reelle og åpent tilgjengelige datasett, relativt til to tidligere undersøkte modeller med veldokumenterte prestasjoner — én dyp modell basert på *variational autoencoders* og én grunn modell basert på *beslutningstrær*. I tillegg beregner vi kontrafaktiske forklaringer ved hjelp av de tre nevnte modellene, før vi vurderer deres evne til å forklare prediksjoner fra en arbitrær binær klassifiseringsmodell.

Våre eksperimenter viser at alle tre modellene er i stand til å generere både syntetiske tabelldata og kontrafaktiske forklaringer, men med ulik grad av tillit og pålitelighet. Vi ser ingen tydelige tegn på at den tabulære diffusionsmodellen yter bedre enn referansemodellene, verken når det kommer til å generere tabelldata fra en estimert underliggende simultanfordeling eller når det kommer til å generere kontrafaktiske forklaringer for å belyse binære prediksjoner av test-observasjoner. Grunnet lovende resultater, uten omfattende justering av hyperparametre, oppfordrer vi til å benytte den tre-baserte modellen som referanse i enhver evalueringsprosess i videre forskning innenfor generativ modellering av tabulære data. Helt til slutt foreslår vi noen mulige retninger for fremtidig forskning på diffusjonsmodeller anvendt på tabelldata, med hovedvekt på generering av syntetiske data eller kontrafaktiske forklaringer.

Preface

I am honoured to present this thesis as my concluding work at the Norwegian University of Science and Technology (NTNU), granting me a Master of Science (M.Sc.) in Applied Physics and Mathematics with emphasis on *Industrial Mathematics*. My time as a student has been exciting, challenging and, at times, highly frustrating, but I cannot overstate how rewarding it has been, both academically and personally.

I would like to thank my supervisor, Kjersti Aas, for allowing me the freedom to pursue the thesis topic of my desire. Thank you for many stimulating discussions, for providing plenty of constructive feedback and for your general encouragement. Thank you to my closest friends and my dear girlfriend for your unwavering support. Last, but not least, I want to dedicate this thesis to my brother, Christian, and my mother, Heidi. Thank you for making this possible. You mean the world to me. Jeg elsker dere.

Lloret de Mar, Catalonia, 19 June 2023
Alexander Johansen Ohrt

Contents

Chapter 1

Introduction

1.1 Context and Motivation

Technology constructed by use of *artificial intelligence* (AI) and its subdomain *machine learning* (ML) is becoming increasingly prevalent and influential — for good reason. World renowned content providers like Spotify and Netflix use developments from these fields to power recommendation algorithms, increase personalization and optimize content encoding [90, 132]. Tesla uses AI and ML to develop autonomy in robots, cars and other vehicles [134]. Formula One, the highest class of international racing sanctioned by the Fédération Internationale de l'Automobile (FIA), employs a large repertoire of developments from these fields to improve on-track performance and fan engagement in the sport [4]. IBM's Watson provides a portfolio of tools for AI adoption in a wide range of sectors, for instance healthcare, travel, retail and security [52]. Some of the largest technology companies in the world have large research facilities that are important to the advancement of these fields, for example Meta AI, Google AI, Amazon Web Services (AWS) and Azure AI by Microsoft. Thus, there is no denying that AI and ML are important in today's technology-landscape, disciplines that likely will grow with increased speed in the future.

Of course, the omnipresence of AI does not only bring value, but plenty of challenges as well. In the wake of Microsoft backed [82] OpenAI's release of GPT-4 [96], a letter titled "Pause Giant AI Experiments: An Open Letter" [37] was published, calling on all developers to pause training of AI systems more powerful than GPT-4 for at least 6 months. The letter, signed by personalities like Yoshua Bengio, Elon Musk and Steve Wozniak, states, among other profound arguments, that "powerful AI systems should be developed only once we are confident that their effects will be positive and their risks will be manageable" [37]. As of June 2023, the letter has collected over 30000 signatures, showing that many people agree. However, there are plenty of people that disagree as well, for instance some of the cited researchers in the letter, who accuse the publishers of "prioritising imagined apocalyptic scenarios over more immediate concerns about AI, such as racist or sexist biases" [16]. No matter how you feel about the letter itself, there is no doubt that many of us are, and should be, concerned about AI and its implications on society.

Many examples of lack of understanding and legislation of AI technology exist. As a result, there is a growing unease among researchers, legislators and users about potential severe consequences. For example, in 2018 a pedestrian was killed by an autonomous Uber-car in Tempe, Arizona, after being hit by the vehicle while wheeling a bike across a road. A human safety driver was present inside the car, an individual who in 2020 was charged with negligent homicide. The incident lead to Uber ending its testing of autonomous vehicles in Arizona and resulted in a greater deal of caution among their competitors [6]. A press release from the National Transportation Safety Board (NTSB) stated that the

automated driving system detected the pedestrian 5.6 seconds before impact, but was not able to accurately predict its path or recognize that the detected object was a pedestrian [88].

Another example, told by Rich Caruana during the first ever debate at a Neural Information Processing Systems (NeurIPS) conference in 2017, stems from healthcare. For context, the proposition under discussion was: "interpetability is necessary in machine learning" [91]. Caruana gives an example regarding a pneumonia risk prediction problem. In this problem, researchers were interested in predicting high-risk pneumonia patients, in need for rapid hospitalization. He talks about how they were able to train a neural network with very good predictive performance on real patient data. However, after further investigation, the researchers saw that the model took its decisions based on some unusual patterns: patients previously diagnosed with asthma were predicted as having lower probabilities of death from pneumonia, compared to the rest of the patients in the dataset. In practice, one could say the model believed asthma was a protective factor against pneumonia, which obviously is not well-aligned with medical research. This is of course an example in which it is easy to conclude that the patterns in the data are misaligned with reality — but what about if such a model is used in a different domain, where such errors are not as easily recognizable? Caruana points to the fact that, in this case, since asthma patients typically are more aware of their breathing patterns, they usually notice pneumonia-symptoms earlier and receive treatment earlier, which explains why these patterns exist in the data. Thus, the network actually found correct patterns in the data, but made the wrong conclusions about them. Blind use of such a model, which seems to yield great predictive performance, can hence be very dangerous — in this case all asthmatics would have received treatment last!

In hindsight, it is easy to conclude that the death of the pedestrian could have been avoided by increased understanding and transparency of the algorithms powering the autonomous device. As demonstrated in Caruana's example, taking time to reason about what techniques the ML model is using might uncover unfortunate, but very important issues. The discipline called *explainable artificial intelligence* (XAI) has grown out of a need for understanding and securing AI systems [102]. Researchers in this field develop interpretable or explainable models, with the objective of facilitating a deeper understanding of how each model works internally [86].

The problems that XAI researchers are trying to solve can often be compressed into one fundamental problem called the *black box problem*. Simply, picture an algorithm or a model from ML, AI or statistical modelling as a black box. The black box is given some input and returns some output. Since it is a black box, the user cannot see what is happening inside. How does it make its decisions? For example, in image classification, the black box is fed an image and outputs a prediction of what the image depicts. In text-to-speech applications, from the field *natural language processing* (NLP), the black box is given a textual input and returns a rendition of the input in speech. Should the user care about what really happens inside the black box, as long as the outputs are satisfactory? The previously discussed examples hopefully motivate why investigating the black box is important, alongside reasons like scientific progress, quality control, and increased knowledge and trust. Thus, the main objective of XAI can be shortened to the simple task of transforming black boxes into white boxes, where the models' inner workings are known.

There exists many different strategies for "opening" the black box, but our focus lies on a particularly interesting method that deals with *counterfactual explanations*. These are human-friendly explanations, easily understood by practitioners and laymen alike, that suggest actionable relations between an example and a consequence. More specifically, according to Guidotti [43], "a counterfactual explanation reveals what should have been

different in an instance to observe a diverse outcome". Such explanations are *post-hoc*, and they are *model-agnostic*, meaning that they can be applied to any previously trained model [86]. Moreover, counterfactual explanations are *local explanations*, meaning that they explain individual instances. These explanations can be calculated following a plethora of different approaches, but perhaps one of the most exciting groups of synthesizers are called *on-manifold* methods [107][1].

The commonality between on-manifold methods for generating counterfactual explanations is that they generally work by modelling the underlying data distribution, the *data-manifold*, before traversing it in search of explanations. Many of the most popular methods from this group rely on *generative models* to fulfill their promise. In short, generative models work by estimating an unknown joint distribution based on some available data, making them capable of, for example, making decisions or generating new sets of data with certain characteristics.

Stepping aside from the question of explainability for a moment, synthetic data, which can be constructed by generative models, is highly interesting in itself, for many reasons [29]. According to Gartner [143], an estimated 60% of data used for analytics and AI development will be synthetically generated by 2024. This estimate speaks in volumes about the expected importance of truthfully generated synthetic data. Specifically, an ideal synthetic dataset has identical mathematical properties to its corresponding real dataset, but contains different *information*. For example, this means that no single instance from the real dataset can be found in the synthetic dataset, while the overall set of instances share the same traits in both datasets, on average. Why is synthetic data valuable? First of all, synthetic data can be used to expand the size of databases in cases where availability of data is low. Second, in cases where privacy is critical, we can create privacy-preserving synthetic data, that then can be used and shared without compromising any individuals. These are only two examples of situations where synthetic data is interesting — more wide-spread application is probably yet to come.

In fact, generative models have recently reached mainstream attention, through the introduction of models like DALL-E 2 [95, 106], Imagen [42, 114] and the aforementioned GPT-4 [96, 97]. Figure 1.1 shows some examples of photorealistic images created with Imagen. The two first models we mention are based on *diffusion probabilistic models*, which are of special interest to us. More specifically, we are inspired by the unprecedented progress that has taken place within this group of generative models in recent years, only recently surpassing previous state-of-the-art (SOTA) models in fields like image generation [20]. Our research topic is formed out of curiosity concerning if diffusion models can be used to construct high-performing on-manifold methods for generating counterfactual explanations.

More precisely, we concentrate on explaining models that deal with *tabular data*, which is ubiquitous. Essentially, most phenomena that can be measured or recorded are typically stored as tabular data. This makes it a highly interesting data modality to investigate, as advancements in handling such data can be rapidly rolled out to most industries, with large potential influence. However, this type of data is notoriously hard to model because of its inherent heterogeneity, leaving much to be desired with regards to performance compared to other data modalities.

Use of diffusion models to synthesize *visual counterfactual explanations*, i.e. counterfactual explanations for explaining image classifiers, has been investigated previously. For instance, some general research on this topic has been conducted [2, 55, 116, 142]. Moreover, some research on visual counterfactual explanations specific to medical imaging can

[1]Note that when we cite Redelmeier et al. [107] we are referring to a modified version of the paper, which we received from Kjersti Aas. Since this version is yet to be published, we add the arXiv preprint with the same name, but with slightly different content, to the reference list.

Figure 1.1: A few examples of photorealistic images created with the text-to-image diffusion model Imagen. The images are borrowed with permission from Saharia et al. [114].

be found in Sanchez et al. [115]. However, to the extent of our knowledge, no research has been done on generating counterfactual explanations for classifiers on tabular data with diffusion models. This is likely connected to the seemingly minimal research conducted on diffusion models applied to tabular data specifically. During the completing stages of this thesis, we discovered some more research on the topic [61, 73, 147], which we have not addressed in detail due to lack of remaining time. Nevertheless, a recently submitted preprint by Kotelnikov et al. [67] treats diffusion models specifically designed for synthesizing tabular data, a paper that is an important source of inspiration for our work.[2]

[2]Parts of this introduction were also present in the author's specialization project [93], either directly or indirectly.

1.2 Objectives and Thesis Structure

We aim to provide a self-contained and accessible introduction to diffusion probabilistic models. We aim to present this with a certain level of mathematical rigor, although intuition and applicability is of primary concern. Further, inspired by the recent surge of powerful diffusion models in disciplines like image generation and natural language processing, we seek to combine these advances with on-manifold methods for generating counterfactual explanations for tabular data. In order to investigate this possibility, we devise theory specific to tabular data, as well as perform informative experiments on a diverse set of real-world data. Finally, we provide several ideas for further work, with the objective of reducing friction in initiation of iterative improvement upon this work. In general, we strive for high levels of transparency, facilitating greater understanding, accessibility and potential for further development.

The structure of the thesis is:

Chapter 2 covers all necessary preliminaries. This includes a short reminder of concepts from probability theory, as well as an overview of our notation and delimitation of the scope. We proceed by presenting accessible introductions to explainable AI, generative modelling, decision trees, deep learning, autoencoders and variational autoencoders (VAEs), before ending the chapter with some important evaluation metrics. We recommend the familiar reader to skim or even skip topics that are known, but make sure to familiarize yourself with our notation and assumptions.

Chapter 3 contains an exposition of diffusion models. We show how diffusion models can be interpreted as an extension of VAEs. We proceed by treating two specific diffusion models that are based on different sets of assumptions. Finally, we combine these two models into a relatively simple model meant to be appropriate for all types of tabular data.

Chapter 4 introduces a specific method for generating counterfactual explanations, followed by a modification of this method which employs diffusion models.

Chapter 5 contains a description of two experiments we perform to understand how the models perform in practice. The chapter describes the datasets we utilize and outlines the methodology we follow in a detailed manner, including the software we employ. The first experiment is designed to evaluate the performance of the final model from Chapter 3, on the task of synthesizing tabular data from an unknown probability distribution, estimated from input data. The second experiment is designed to evaluate the performance of the final model from Chapter 4, on generating counterfactual explanations for tabular data. For context, we compare the diffusion-based models with two carefully selected reference models.

Chapter 6 presents some results from the first experiment, on generating synthetic data. We also provide some thoughts and comments on the results.

Chapter 7 lays out some results from the second experiment, on generating counterfactual explanations. We also provide some thoughts and comments on the results.

Chapter 8 summarizes the thesis, highlights some main points of discussion and proposes several possible directions for future research.

Chapter 2

Background Material

This chapter covers all necessary background material for understanding the contributions of this thesis. The level of difficulty is chosen such that students with a fundamental understanding of mathematics, optimization, probability, statistics and ML should be able to understand the concepts. Some sections are modified versions of sections from the author's specialization project [93], with the notation changed accordingly. The reader which does not have introductory knowledge in AI or ML is referred to Chapter 2 in the project paper.

The chapter is organized as follows. First of all, Section 2.1 briefly covers some important concepts from probability theory, as well as defines our notation. Second, Section 2.2 presents some important assumptions, setting the stage for the rest of our work. Next, Section 2.3 introduces explainable AI and counterfactual explanations, using mostly recycled material from the author's specialization project [93]. In continuation, Section 2.4 gives a concise introduction to generative AI, with emphasis on *deep* generative AI. The subsequent discussion on decision trees in Section 2.5 is also strongly inspired by content in the author's specialization project, with a new addition on gradient boosting. Then, Section 2.6 on deep learning and Section 2.7 on autoencoders are updated versions of elements from the introduction to deep learning in the project. Specifically, the former section introduces deep learning from a historical perspective, in addition to feedforward neural networks, as well as our notation for designing and describing neural network architectures. Moreover, the latter section introduces the encoder-decoder architecture, as well as some special types of autoencoders. Later, Section 2.8 gives a proper introduction to variational autoencoders. Note that we described these models in the specialization project as well, but this section introduces them from a new and updated perspective, yielding a more complete introduction to the topic. Furthermore, a VAE especially designed for tabular data, the *TVAE* [146], is explained in Section 2.9. Finally, in Section 2.10, we discuss some important metrics for evaluating our models later in the thesis.

2.1 Probability Theory

We assume that the reader is familiar with the essential parts of probability theory, but we reiterate some important concepts for clarity. A *probability distribution* is a collection of probabilities describing all possible events in a *probability model* [31]. Such a distribution may be represented by different mathematical quantities, depending on the nature of the *random variables* that map the sample space, i.e. the set of possible outcomes, to $\mathbb{R}$. Commonly, statisticians consider either *discrete* or *continuous* random variables, i.e. random variables that are valued on discrete sets or intervals of real numbers, respectively. When considering only the discrete variants, a probability distribution may be sufficiently represented by a *probability mass function* (PMF), which assigns a probability to each

discrete outcome in the sample space. Similarly, when considering continuous random variables, the probability distribution may be represented by a *probability density function* (PDF), which essentially describes the infinitesimal probability of each continuous value in the sample space [31]. We refer to both PMFs and PDFs as *densities*, with the implicit understanding that these are mass functions in cases where we are working with discrete random variables. In addition, we use the terms *distribution* and *density* interchangeably, which is common in the literature. The interested reader is referred to Evans and Rosenthal [31], or Jacod and Protter [53], depending on the level of sophistication that is desired, for more details on probability theory.

Notation

The material studied in this thesis comes from a variety of research fields, each with their own conventions in notation. There is seemingly little to no standardization in notation across disciplines that deal with concepts from probability and statistics. For clarity and transparency, we thoroughly explain the notation we use throughout the thesis.

Capital, italic letters like X denote scalar random variables, while lower case, italic letters like x denote their realizations. Bold letters denote vectors or matrices. More specifically, italic letters like $\boldsymbol{X}$ and $\boldsymbol{x}$ denote random vectors and their realizations, while roman letters like $\mathbf{X}$ denote matrices. Notice that bold, italic letters like $\boldsymbol{x}$ (or $\boldsymbol{\theta}$) also denote vectors of scalars when discussing parameters or other non-random quantities. This notation also applies to random and deterministic vector-valued functions, i.e. functions whose range consists of either random or deterministic vectors. The reader should be aware that, in most advanced textbooks and applied research, both random variables and their realizations are typically denoted by lower case letters, leaving each symbol's precise meaning to be understood from context. In contrast, we practice different notations for different quantities, to avoid unnecessary confusion in certain situations.

The probability of an event is denoted by capital P. In most cases, marginal densities are denoted by $p(\cdot)$, conditional densities are denoted by $p(\cdot \mid \cdot)$ and joint densities are denoted by $p(\cdot\,, \cdot)$. We use the same notation no matter the types of random variables we are working with. It should be clear from the context if all random variables are continuous or discrete, or if some random variables are continuous and others are discrete. In these notations, we suppress any explicit dependence on random variables and leave them to be understood from the context. For example, in order to be exact, one should denote a marginal density as $p_X(x)$, explicitly stating the random variable, X, it carries information about. This suppression is commonly used in statistics for brevity, in cases where this information is clearly understood from the circumstances.

We let $X \sim p(x)$ denote that a random variable X is distributed according to a density $p(x)$. In addition, when X follows a well-known distribution, we use a notation specifying the name of the distribution. For example, for the Gaussian distribution with parameters μ and σ^2, we simply write $X \sim \mathcal{N}(\mu, \sigma^2)$. In this case, the notation means that X is distributed as the specified Gaussian. Finally, in cases where we want to specifically represent the density function of a well-known distribution, we add an argument, like for example $\mathcal{N}(x; \mu, \sigma^2)$. For consistency, we let these considerations hold for vectors as well.

2.2 Assumptions

The work in this thesis is necessarily restricted because of limitations in time and prerequisites. Some of the assumptions we make are highlighted here.

Let $\mathcal{D}$ denote a dataset, a set of observations, from a larger population. This data could come from many different modalities, but our main focus is on *tabular* data, i.e. data that

contains samples (rows) which consist of values of certain features (columns). Oftentimes, the data is heterogeneous, meaning that continuous, discrete, binary, nominal and ordinal data types may be present in $\mathcal{D}$. Unless otherwise specified, we let the term *categorical* encompass all non-continuous data types. In addition, without loss of generality, we only consider datasets originally meant for binary classification, meaning that they have a binary *response, label* or *dependent variable*. As a consequence, unless otherwise specified, we discuss supervised models in the context of binary classification. To facilitate this, the dataset $\mathcal{D}$ is assumed to consist of n observations,

$$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\},$$

where each observation consists of $p+1$ values. Precisely, the vectors are defined as

$$(\boldsymbol{x}_i, y_i) \coloneqq \{x_i^1, x_i^2, \ldots, x_i^p, y_i\}, \quad i \in \{1, \ldots, n\},$$

where x_i^j, $j \in \{1, \ldots, p\}$, represents the values of the *features*, *covariates* or *independent variables* in observation $(\boldsymbol{x}_i, y_i)$. Additionally, $y_i \in \{0, 1\}$ represents the binary response value corresponding to the feature values in $\boldsymbol{x}_i$. The sub-indices are suppressed in cases where we are clearly discussing one observation. Even though we do not show this explicitly in the set-notation used above, vectors are always treated as column vectors. For simplicity, it is assumed that all n points are independent samples from some underlying, or latent, distribution. We represent this distribution with the density $p^*(\boldsymbol{x}, y)$. Moreover, we assume that this distribution is fixed. This means that if we could collect another dataset $\mathcal{D}'$, recording the same phenomena, it would be from exactly the same underlying distribution as $\mathcal{D}$. Of course, this is an assumption that oversimplifies reality, but it is commonly made in statistics. In such a case, we say that the observations $(\boldsymbol{x}_i, y_i)$, $i \in \{1, \ldots, n\}$, are *independently and identically distributed* (i.i.d.).

The dataset $\mathcal{D}$ can also be represented in a different, more abstract way. Let $(\boldsymbol{X}, Y) \coloneqq \{X^1, \ldots, X^p, Y\}$ denote a set of random variables. These variables follow an unknown joint distribution described by a collection of probabilities,

$$P(\{X^1, \ldots, X^p, Y\} \in A),$$

for all subsets $A \subseteq \mathbb{R}^{p+1}$. We stress that the superscript indices are used to index the random variables in the set and do not refer to exponentiation. To simplify the representation of the joint distribution, we assume that the set has a density denoted by $p^*(\boldsymbol{x}, y)$. We do not go into detail about when such an assumption can be made, but note that it is a fair assumption to make for most practical purposes. Details can be found in Jacod and Protter [53]. Then, let $(\boldsymbol{x}, y) \coloneqq \{x^1, x^2, \ldots, x^p, y\}$ represent a single sample from this joint distribution. These constructs can be tied back to our dataset $\mathcal{D}$ quite nicely, by noticing that the columns of $\mathcal{D}$ can be labelled by $(\boldsymbol{X}, Y)$. As a consequence, each row of $\mathcal{D}$ can be interpreted as a single sample from $p^*(\boldsymbol{x}, y)$. In other words, by letting the random vector $(\boldsymbol{X}, Y)$ represent the columns, each observation in $\mathcal{D}$ is a realization of $(\boldsymbol{X}, Y) \sim p^*(\boldsymbol{x}, y)$. Moreover, every value in a specific column $j \in \{1, \ldots, p, p+1\}$ in $\mathcal{D}$, labelled by its corresponding random variable from the set $\{X^1, \ldots, X^p, Y\}$, can be interpreted as a sample from an underlying marginal distribution, represented by $p^*(x^j)$ for $j \in \{1, \ldots, p\}$ or $p^*(y)$ for $j = p+1$. This way of interpreting each value in a tabular dataset is useful in several contexts, for example when discussing vector spaces corresponding to the domain of a function or model. Conclusively, we have described two quite different ways of intuitively arguing about what a given dataset $\mathcal{D}$ represents. Each representation will come in handy throughout this thesis.

2.3 Explainable Artificial Intelligence

The necessity of tangible explanations has increased alongside the rise in use of complex models from AI and ML in high-risk areas like medicine, banking and anti-money laundering [94]. *Explainable artificial intelligence* (XAI) is a branch of AI that treats explaining such models. Another term that is often used for XAI in the context of ML is *interpretable machine learning* (IML). Note that exactly what an *explanation* should include, or how *interpretability* should be defined, is a great discussion in itself. These questions are subject to research, not only specific to XAI or IML, but in broader fields like psychology, cognitive science and philosophy. We do not cover these questions in detail in this work, but simply use the terms *explanation* and *interpretation* interchangeably, according to our casual, imprecise understanding of them. The interested reader is referred to, e.g., Miller [83] for a thorough review of the research, in conjunction with work on explanations in AI.

Explainability vs performance. A tradeoff between explainability and performance of models can be observed in a wide range of applications. Figure 2.1 illustrates how some common ML methods generally compare to each other when it comes to predictive accuracy and interpretability. We observe that simpler models like linear regression and decision trees are highly interpretable. They perform well in some applications, but because of restrictive assumptions and non-robustness, there are other methods that yield higher accuracy in many applications. Deep learning is on the other side of the spectrum. Cutting edge algorithms in many fields — for example in generative modelling, image classification and natural language processing — are based on deep learning. However, these models are rarely innately interpretable.

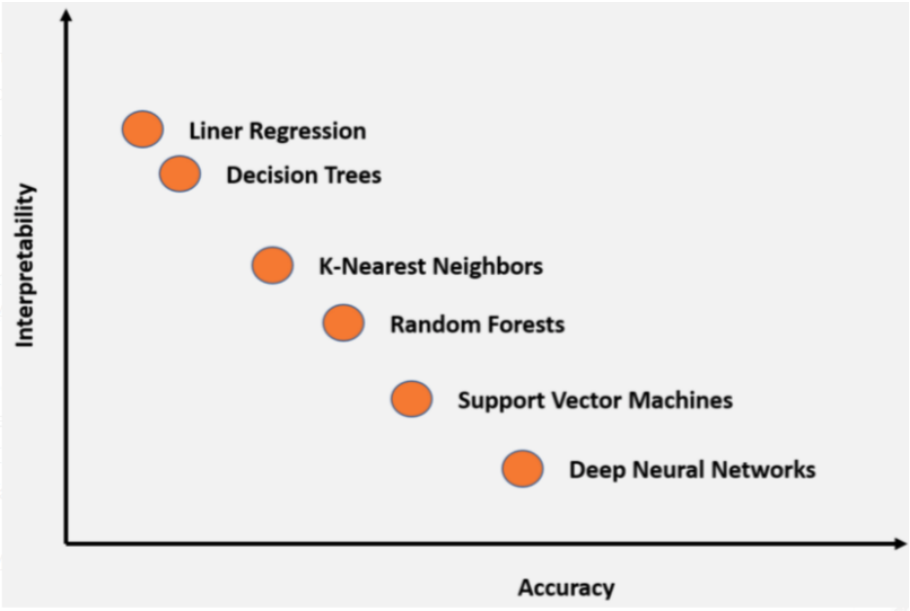


Figure 2.1: Qualitative illustration of the tradeoff between accuracy and interpretability in a set of common ML models. The image is borrowed with permission from Sivamani [123].

Motivation. Frequent use of black boxes for high stakes decision-making yields increased demand for explainability when deploying algorithms. Molnar [86] states that "interpretability helps the developer to debug and improve the model, build trust in the model, justify model predictions and gain insights". Interpretability is of course not only important for the developer — it is essential for users and individuals affected by deci-

sions made by such models. For example, imagine that a patient is predicted to have life-threatening cancer by a supervised learning model. This setting is not completely unrealistic, because, in recent years, several ML models have been applied to cancer prognosis and prediction [68]. Both for the patient and for the medical doctor in charge, it is vital that they understand *why* the model has returned this prediction. They need to know on what basis the prediction has been made and if the model's conclusions can be trusted. In addition to usefulness and necessity for trust, explanations are required by legislators and regulators. The General Data Protection Regulation (GDPR), developed by the EU, gives an individual subjected to automatic decisions that significantly affects him or her, among other rights, the right to obtain an explanation. In addition, decisions based solely on algorithms, without human intervention, having a significant effect on a subject, should be prohibited unless authorized, and the subject should have the right not to be subject to such decisions [30]. All these reasons make XAI attractive to researchers, practitioners and the general public. Before moving on to some theoretical concepts in the field, it is worth mentioning that despite today's frequent use of natively non-interpretable models, some argue that one should always use interpretable models instead of striving to explain black boxes [111].

Global and local explanations. Explanations in XAI can at first be divided into two different types: *global* and *local* explanations. The objective of a global explanation is to describe the average behaviour of a model. As a consequence, such an explanation is often stated as an expectation over the distribution of the data. In contrast, the objective of a local explanation is to explain an individual prediction made by a model. Some of the local explanation methods are so-called *attribution* methods, meaning that they describe individual predictions using feature effects, while other methods are so-called *example-based*, meaning that they produce new data points as explanations [86].

Model-specific and -agnostic explanations. Another important distinction in the field can be made between *model-specific* and *model-agnostic* methods. As the name implies, model-specific methods use assumptions in a ML model itself to explain its predictions. Thus, the specific ML model used to obtain the outputs we want to explain needs to be well-known when implementing model-specific methods. On the other hand, model-agnostic methods try to explain model predictions without using the inner workings of the model itself. It is not obvious which of these approaches is best — both of them have advantages and disadvantages — meaning that the choice of regime should be evaluated case by case. An advantage of a model-agnostic method is that it can be used across many different ML models, facilitating effective comparative evaluation of performance. Such explanatory methods can become natural pieces of a typical software engineering workflow. This is more challenging with model-specific methods, because one needs an individual surrogate model to explain each ML model. An advantage of a model-specific surrogate model is that it can be developed to deliver more focused and detailed explanations, especially tailored to a specific ML model. For example, in the field of deep learning there are plenty of examples of surrogate explanatory models that fall under this category [86]. Notice the use of the term *surrogate model*. In practice, this is often what use of methods from XAI entails, i.e. constructing new models to interpret or explain previously trained, black-box models. Before moving on, we provide a hypothetical example to clarify the ideas mentioned above.

Example 2.3.1. Imagine that a medical doctor has access to some ML-powered software that predicts if a patient in the emergency room needs immediate medical attention. The model uses feature values that are when a patient enters the emergency room. We are interested in analyzing *how* this software works, in order to make sure it is fair. XAI could provide such analyses. A global explanation method, like for example a *partial dependence plot* (PDP), describes the average behaviour of the model; "the PDP shows the marginal

effect one or two features have on the predicted outcome" [86]. In this case, a PDP of the time of entry of a patient vs the predicted probability of the patient needing immediate medical attention could be interesting to construct. This might reveal if the time a patient has waited seems to be more important in explaining the model's behaviour compared to other patient characteristics. In contrast to this, a local explanation method, like for example *Shapley values* or *counterfactual explanations*, explains individual predictions made by the model. If the model predicts that a patient $\boldsymbol{x}$ in the emergency room does not need immediate medical attention, a counterfactual to this prediction indicates which feature values of the customer that *should* be changed for the prediction to *flip*, i.e. that patient $\boldsymbol{x}$ would need immediate medical attention. For example, it could indicate that patient $\boldsymbol{x}$ would have been predicted as in immediate need if the heartbeat was 40 instead of a measured 70. We focus on this type of human-friendly explanations in this thesis — counterfactual explanations are studied in detail in Section 2.3.1. A model-specific explanation method would in this example be limited to, and constructed specifically for, the given ML model the doctor uses. For example, if this is a deep learning model, *Grad-CAM* [120] is an example of this type of method (for *convolutional neural networks*). In contrast, a model-agnostic explanation method is not dependent on the underlying prediction model. PDP, Shapley values and counterfactual explanations are usually model-agnostic. For example, a local counterfactual explanation is determined solely based on a specific individual and its corresponding prediction from the model. △

2.3.1 Counterfactual Explanations

Within XAI, our main focus lies on *counterfactual explanations* (CEs). Note that we use the terms *counterfactual explanations* (CEs) and *counterfactuals* interchangeably in the following, in reference to a prediction-explanation of this type. As noted, a CE is a local explanation, since it can be used to explain predictions of individual instances in the data. Notice that there exists both model-agnostic and model-specific methods for generating CEs [86], but we focus on model-agnostic versions, where a counterfactual is produced based only on the input and output of a prediction model. The aim of these types of explanations is to find new instances that change the original prediction to one that is desired, while making as few changes as possible to the feature values in the original input instance. That is, a prediction is explained by a counterfactual by highlighting which feature values in the input instance need to be changed in order to change the predicted outcome. Specifically, in binary classification problems, this change in prediction amounts to a binary flip; we construct a counterfactual that yields a positive prediction (binary indicator 1) instead of the original negative prediction (binary indicator 0). According to Redelmeier et al. [107], there exists some general criteria that counterfactuals should meet:

1. *on-manifold*: The counterfactual should lie on the data-manifold. This means that counterfactuals should ideally come from the same underlying distribution as the data.

2. *actionable*: The counterfactual should not change feature values of the original instance that are *immutable* or *fixed*. The fixed features are predefined in each specific application.

3. *valid*: The counterfactual should represent an instance that gives a different predicted outcome. The desired different prediction is predefined in each specific application. In a binary classification setting, we are typically looking for counterfactuals that yield positive predictions (binary indicator 1).

4. *low cost*: The number of different feature values in the original instance and the counterfactual should be as small as possible. The *amount* of difference between values of the same feature should also be as small as possible. This amount can be measured in a variety of ways, depending on the application, as well as on if the feature is continuous or categorical.

To clarify the ideas, we construct a simple, hypothetical example of what a counterfactual may look like in practice.

Example 2.3.2. A person applies for a mortgage at his bank. The customer profile is shown in Table 2.1a. The bank uses a supervised learning method, trained on data from

Table 2.1: The customer that solicits a mortgage, alongside an example of a useful counterfactual and a useless counterfactual. The attributes of the customer are *age, sex, nationality, salary (yearly), work sector, marital status, years as customer at the bank* and *postal code (ZIP).*

(a) Customer that solicits a mortgage.

Age	Sex	Nat.	Sal.	Work Sect.	Mar. Stat.	Cust. Years.	ZIP
22	M	Norway	350K	Public	Single	2	7051

(b) An example of a useful counterfactual.

Age	Sex	Nat.	Sal.	Work Sect.	Mar. Stat.	Cust. Years.	ZIP
22	M	Norway	420K	Private	Single	2	7051

(c) An example of a useless counterfactual.

Age	Sex	Nat.	Sal.	Work Sect.	Mar. Stat.	Cust. Years.	ZIP
200	F	Sweden	2200K	Private	Single	10	7051

all previous applicants. We assume that this particular person is not granted a mortgage, based on a prediction from the model. A possible counterfactual explaining this decision is shown in Table 2.1b. This counterfactual does not change the fixed characteristics (here: age, sex, nationality and years as customer at the bank), but it changes other feature values that the customer might be inclined to modify in order to be granted a mortgage based on a future application. For example, the customer could change to a job in the private sector, which could perhaps also increase his salary. Given that the "new" customer profile in Table 2.1b changes the predicted outcome to "mortgage granted", that it changes the feature values of the customer the least amount possible and is on the data-manifold, this is a useful counterfactual explanation. To contrast this, Table 2.1c shows a customer profile which is not a useful counterfactual. It changes all the fixed values of the original customer, already deeming it useless. Notice that the age of the customer no longer is realistic, indicating that the generated individual is not on-manifold. In addition, the salary of the generated individual seems unreasonable, likely also yielding a generated individual that is not on the data-manifold of the bank's mortgage applicants. △

Calculating CEs. Because the research field on counterfactual explanations is emerging and rapidly developing, there is no universally agreed upon grouping of different strategies for *how* to create counterfactuals. For example, Guidotti [43] uses a rather coarse split between methods based on optimization strategies and methods based on heuristics for minimizing some cost, where each of these two groups contains many different variants. In our case, we choose to follow the dual grouping introduced by Redelmeier et al. [107], between *algorithmic-based* approaches and *on-manifold* methods.

Algorithmic-based. The first general strategy for synthesizing counterfactuals is based on solving an optimization problem,

$$\arg\min_{\boldsymbol{x}'}\{d_1(f(\boldsymbol{x}'), y') + \lambda d_2(\boldsymbol{x}, \boldsymbol{x}')\}, \tag{2.1}$$

where $f(\cdot)$ is a binary classifier, and d_1 and d_2 are appropriate metrics. These methods work by finding an instance $\boldsymbol{x}'$ that is close to the original instance, $\boldsymbol{x}$, according to the metric d_2, while ensuring that the new prediction, $f(\boldsymbol{x}')$, is close to the desired outcome, y', according to the metric d_1. The hyperparameter λ is used for changing the relative importance of the two terms. An advantage of these methods is that a problem formulated like Equation (2.1) can be solved with a wide variety of techniques, like gradient descent methods, random walks or integer programming, among others [107]. Some disadvantages of such methods are that they are susceptible to generating counterfactuals that are neither on-manifold nor actionable. Additionally, they often require continuous and independent features in the data, alongside certain assumptions on the model f, like differentiability. Finally, the choice of appropriate metrics is not trivial, because the resulting $\boldsymbol{x}'$ is highly dependent on d_1 and d_2 [107].

On-manifold. The second general strategy for generating counterfactuals consists of so-called *on-manifold* methods. In these methods, the objective is to model the underlying data distribution, before finding counterfactuals on this manifold. An advantage of these methods is that many of them can exploit advances in, for example, *generative modelling*, a research field that is developing at an exceptional speed, which is covered in Section 2.4. Some disadvantages of such methods are that they often are restricted to explaining gradient-based models, in which case they are not truly model-agnostic. Additionally, many of the methods cannot effectively handle categorical features with more than two levels, and they are not able to treat fixed features in a reasonable way, both of which are large flaws in practice [107].

Closing remarks. For more details, the interested reader is referred to Guidotti's literature review [43], a survey that categorizes the most recent methods for calculating counterfactuals based on their individual properties. In addition to developing a grouping system based on theoretical properties, the author constructs a benchmarking system in order to compare the methods in practice. Regarding this, he states that "the results make evident that the current state of the art does not provide a counterfactual explainer able to guarantee all these properties simultaneously" [43]. It is not crucial to state exactly what properties the author is referring to, but most of them are similar to the properties we stated previously. Thus, as of recently, a method for calculating all-purpose counterfactuals does not exist, according to Guidotti. Hence, it is highly important that each application is analyzed differently; the method for calculating CEs should be chosen based on a careful consideration of the advantages and disadvantages in each case.

2.4 Generative Artificial Intelligence

Generative AI, or *generative modelling*, is one of the current major talking points within AI. Owing to rapidly growing research and development, several highly complex, high-performing models in their respective domains, like *DALL-E 2* [106], *Stable Diffusion* [110] and *ChatGPT* [97], entered the mainstream in 2022. The latter reached 1 million users in only 5 days [85], with the public opinion ranging from dystopic to utopic. In this section, we provide a concise introduction to generative modelling.

Discriminative vs. generative learning. Many examples where ML typically is used can be interpreted as problems from *decision theory*, a subject concerned with decision-making under uncertainty [8]. For instance, a typical scenario is: based on a set

of training observations, the objective is to make an optimal decision concerning a new and unseen observation. Consistent with previous notation, let $(\boldsymbol{X}, Y) \coloneqq \{X^1, \ldots, X^p, Y\}$ denote the random variables that describe the phenomenon we are studying, where Y is the chosen random variable we want to reason about. Specifically, when reasoning about Y, the objective is to train a model for the true underlying distribution of $Y|\boldsymbol{X}$, which we represent with the density $p^*(y|\boldsymbol{x})$, before using this model to assign new observations, $\boldsymbol{x} = \{x^1, \ldots, x^p\}$, to the optimal response. Strategies for solving such decision problems are commonly divided into two different groups; *discriminative* and *generative* [56]. These two techniques are generally treated as opposites, although work has been done to combine the two methodologies [56, 70]. Starting with the generative approach, it essentially boils down to learning an approximation of the true joint distribution of $(\boldsymbol{X}, Y)$, represented by the density $p^*(\boldsymbol{x}, y)$. Access to the joint distribution facilitates answering a wide range of uncertain questions, by marginalizing, conditioning or applying other mathematical operations to the density. In addition, as the name suggests, we can generate new data by sampling from the learnt model of the underlying joint distribution. For instance, the decision problem we outlined above can be solved by determining approximations to the posterior density,

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, y)}{p(\boldsymbol{x})}, \tag{2.2}$$

where $p(\boldsymbol{x}, y)$ is the learnt estimation of the true joint density and $p(\boldsymbol{x})$ is a *normalization* factor, which represents a model of the marginal distribution of $\boldsymbol{X}$. Note that the joint density can either be learnt directly or indirectly. The latter technique is based on the relation $p^*(\boldsymbol{x}, y) = p^*(\boldsymbol{x}|y)p^*(y)$, meaning that we could estimate $p^*(\boldsymbol{x}|y)$ and $p^*(y)$ individually to indirectly estimate $p^*(\boldsymbol{x}, y)$. Notice that this technique essentially transforms Equation (2.2) into *Bayes' theorem*. Thus, generative learning is highly attractive, since access to a proper approximation of the true joint distribution acts more or less as an oracle, because, as stated by Bishop [8], the joint "provides a complete summary of the uncertainty associated with the variables". However, this technique has some downsides. Most importantly, calculating an approximation of $p^*(\boldsymbol{x}, y)$ is often computationally challenging. In many applications, $\boldsymbol{X}$ is high-dimensional, and adequate parameter estimation of generative models typically requires a lot of data. Consequently, finding a reasonable approximation to $p^*(\boldsymbol{x}, y)$, and normalizing it, which is necessary for decision-making of the form in Equation (2.2), can be highly demanding or even intractable.

The complexity of the generative approach inspires *discriminative* models, which generally solve a decision problem by modelling the true posterior distribution, represented by the density $p^*(y|\boldsymbol{x})$, directly [8]. Thus, discriminative models bypass the issues related to modelling the joint distribution by simply optimizing a prediction model directly [56]. In fact, as Bishop [8] states: "if we only wish to make classification decisions, then it can be wasteful of computational resources, and excessively demanding of data, to find the joint distribution". However, as Tomczak [138] argues, estimating the joint distribution is crucial not only for generating synthetic data, but also for evaluating the likelihood of observations, properly weighting decisions and uncertainty assessment. Thus, a disadvantage of the less complex discriminative approach is that it does not facilitate such analyses, which provide the practitioner with a more well-informed decision. For instance, a downside of the typical discriminative model is that it displays certainty no matter how probable an input observation is under $p^*(\boldsymbol{x})$, whereas a generative model accounts for uncertainty by weighting decisions with their probability under the approximated feature joint, $p(\boldsymbol{x})$, which can be observed from the relation $p(\boldsymbol{x}, y) = p(y|\boldsymbol{x})p(\boldsymbol{x})$. Actually, the most rudimentary discriminative models simply map any observation, $\boldsymbol{x}$, onto a response, without constructing a probability model at all, which does not enable reasoning under

uncertainty.

Deep generative models. Moving on to the group of models where the previously mentioned mainstream technologies belong, *deep* generative models are generative models that are parameterized using *deep learning*. We provide an exposition of deep learning in Section 2.6, but, succinctly said, the research area consists of flexible and powerful algorithms that have been highly influential over the years. Following the taxonomy of Tomczak [138], we divide deep generative models into four main groups: *autoregressive models* (ARMs), *flow-based models* (flows), *latent variable models* and *energy-based models* (EBMs). In addition to these four, we mention a fifth group called *score-based models* (SBMs), which are closely related to both energy-based models and latent variable models. Our focus is on latent variable models, which we further divide into *likelihood-based models* and *non-likelihood-based models*. As the names suggest, the former models $p^*(\boldsymbol{x}, y)$ directly using the likelihood, or a bound on the likelihood, of the dataset $\mathcal{D}$, while the latter models $p^*(\boldsymbol{x}, y)$ in other ways. Notice that likelihood-based models exist in the other four main groups of generative models as well, which is why, as Tomczak points out, there exists no unique, one-size-fits-all taxonomy [138]. In fact, ARMs, flows, EBMs and SBMs largely consist of likelihood-based models.

Generative adversarial networks. The most popular models in the non-likelihood based paradigm are *generative adversarial networks* (GANs) [41]. Many high-performing, state-of-the-art (SOTA) models for data modalities like images are based on variants of GANs. Such models approximate complex sampling procedures by *adversarial training*. Succinctly explained, a GAN is an unsupervised model that consists of a *generator* and a *discriminator*. These two components are trained in parallel, by teaching the generator to "fool" the discriminator. This implicitly teaches the generator to model $p^*(\boldsymbol{x}, y)$. Since this model is not likelihood-based, we cannot evaluate the likelihood of an arbitrary observation $(\boldsymbol{x}, y) \in \mathcal{D}$ using GANs. We do not discuss such *implicit* generative models [84] any further, but we mention GANs because they have pushed the envelope of deep generative modelling for almost a decade.

Variational autoencoders and diffusion models. Within the group of deep likelihood-based latent variable models, *variational autoencoders* (VAEs) [65, 109] have been the most important model for years, rivaling GANs in some aspects. We provide a detailed exposition of VAEs in Section 2.8. Moreover, in Chapter 3, we introduce *diffusion models* [47, 124], which are the main likelihood-based latent variable models we investigate in this thesis. These have only recently been shown as worthy competitors to GANs and VAEs, being used to develop SOTA models for several data modalities. Notice that diffusion models have score-based interpretations as well [76], extending the knowledge about diffusion models to a whole new body of literature, a detail we swiftly return to later.

2.5 Decision Trees

Decision trees are among the most popular supervised ML models for both classification and regression, because of their proven performance, alongside their simplicity and inherent interpretability. In fact, they play important roles in this work, which is why this section is added for completeness.

Let $\mathcal{D}$ be a multivariate dataset ($p > 1$) of n training observations. Recall that every value x^j, $j \in \{1, \dots, p\}$, in an arbitrary observation $\boldsymbol{x} = \{x^1, \dots, x^p\}$ can be interpreted as a realization of feature X^j, $j \in \{1, \dots, p\}$. Following such a philosophy, the input space that a decision tree works in is p-dimensional, where each dimension pertains to each feature. If Y is categorical, a decision tree trained on the covariate values, $\boldsymbol{x}_i$, $i \in \{1, \dots, n\}$, with y_i, $i \in \{1, \dots, n\}$, as labels is called a *classification tree*. On the contrary, if Y is continuous, the decision tree is called a *regression tree*. Decision trees are typical

examples of discriminative models, essentially providing the practitioner with an estimate of the distribution represented by the density $p^*(y|\boldsymbol{x})$. Intuitively, a decision tree splits the p-dimensional input space into p-dimensional rectangular segments. In each segment, a model is fitted in order to predict the response. The traditional choice of such a model in a regression tree is a density estimator, for instance a *kernel density estimator*, fitted on the responses of the training points in the segment. Similarly, for a classification tree, the prediction is typically decided based on the shares of the observed categories in the training points in the segment. However, note that these models can be arbitrarily complex, depending on the level of flexibility that is desired.

The popularity of decision trees is partly due to the simple manner in which they can be represented, which makes trees simple to explain and interpret for most demographics. An example of a decision tree in $\mathbb{R}^2$, i.e. $p = 2$, is shown in Figure 2.2. Notice the tree-like structure to the left, which is how a fitted decision tree commonly is depicted. Each horizontal line represents a split in the input space at a specific feature value. Each of these intermediate subsets of the space are called *internal nodes*. These splits are almost always constricted to being binary, defining *binary decision trees*, meaning that they split continuous features into two segments, and partition the categories of categorical features into two groups. At the bottom of the tree, the *leaf nodes* are denoted as R_i, in this example with $i \in \{1, \ldots, 5\}$. The panel to the right shows the space of features in $\mathbb{R}^2$

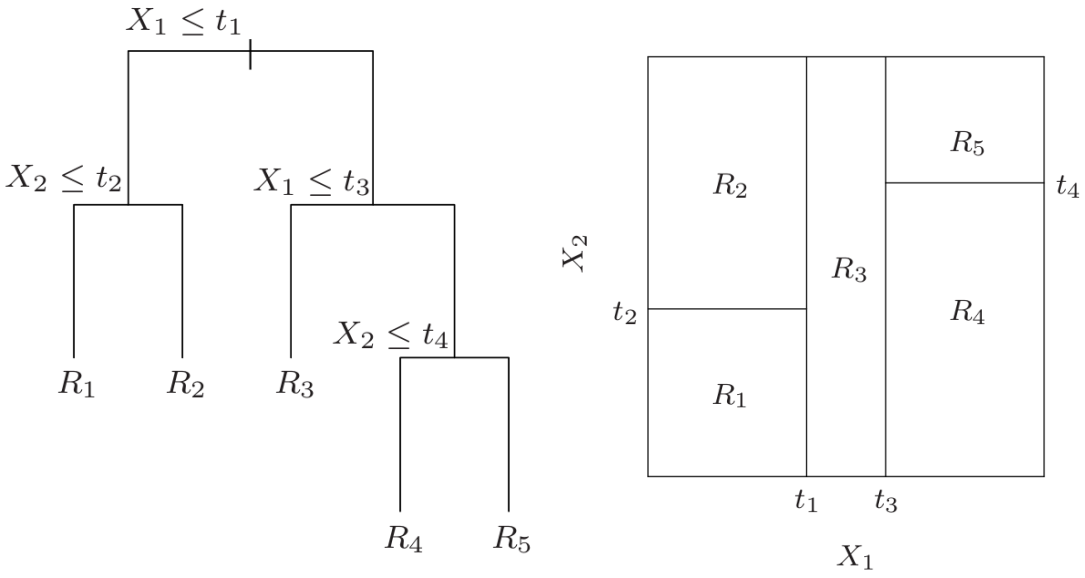


Figure 2.2: Example of a decision tree in $\mathbb{R}^2$. Please excuse the use of subscripts for features instead of superscripts. The images are borrowed with permission from Hastie et al. [44].

corresponding to the tree. For example, according to the tree splits, if a data point $\boldsymbol{x} = \{x^1, x^2\}$ has continuous feature values such that $x^1 \leq t_1$ and $x^2 \leq t_2$, it belongs to the segment represented by leaf node R_1. For instance, in a binary classification tree, R_1 could contain a PMF over the binary indicators 0 and 1, where the probability of each category is estimated from the share of the category among the training observations belonging to the leaf node. Thus, the conditional probabilities $P(Y = 0|X^1 \leq t_1, X^2 \leq t_2)$ and $P(Y = 1|X^1 \leq t_1, X^2 \leq t_2)$ are estimated based on the normalized ratios of response values in the training observations with $x^1 \leq t_1$ and $x^2 \leq t_2$. Similarly, the other leaf nodes contain estimations of other class-conditional PMFs, which illustrates why decision trees are discriminative models. In addition, decision trees can be used to design rudimentary discriminative models, by skipping the estimation of densities in the leaf nodes and simply

assigning any new observation to a deterministic value, most often dependent on the response values of the training observations belonging to each node. This value could for instance be the average value in regression trees or a majority vote in classification trees.

As mentioned, decision trees are highly attractive to many practitioners because of their simplicity in representation. Furthermore, they have certain theoretical properties that make them useful in many applications. Since they are non-parametric, they can handle interactions between features in a dataset automatically, to any order. Moreover, they are usually fast to build, scalable, require little to no extra data pre-processing, can handle mixed data types without issues and perform feature selection automatically [44, 107].

Parameter estimation. There exists several different algorithms for calculating decision trees. Some of the most important differences between the algorithms are the criteria used to find the internal nodes, how to estimate the response in each of the leaf nodes, when to stop splitting on a particular feature and the general structure of the fitted trees [86]. This overview does not give a detailed description of any of these algorithms, but will instead refer to relevant literature. One of the most popular algorithms for growing decision trees is *CART: Classification and Regression Trees* [11]. First introduced in the 1984 monograph with the same name, CART is a very important algorithm for many different applications, which explains why citations of the original source can be found in a wide variety of fields. CART builds trees by recursively splitting each feature in two, stopping when some pre-specified criterion is fulfilled. Another popular algorithm for calculating decision trees is called *C4.5* [105]. Both CART and C4.5 were recognized as two of the top ten most influential algorithms in the field of data mining, according to a survey paper by the IEEE International Conference on Data Mining (ICDM) in 2006 [144]. This indicates the influence these algorithms have had. In addition, the fact that two algorithms for building decision trees occupied this list further manifests the importance of the decision tree idea and implementation.

2.5.1 Gradient Tree Boosting

Gradient boosted decision trees (GBDTs), i.e. models that are constructed by using *gradient boosting* on decision trees, represent an improvement over regular decision trees. As noted, decision trees have plenty of highly attractive properties. However, the main problem with decision trees, which sometimes prevents them from being used in their standard form in practice, is that they rarely yield optimal predictive performance in applications. GBDTs tend to improve predictive performance, often in dramatic fashion [44]. In addition, there exists GBDTs which are able to maintain most of the attractive features of decision trees, with the exception of deteriorated interpretabilty and speed. In order to understand what gradient tree boosting is, we break down the term systematically in the following.

Boosting. The term *boosting* refers to the idea of improving a *weak learner* to yield comparable performance to a *strong learner*. Casually stated, a weak learner is a model that only is able to perform slightly better than random predictions, while a strong learner is a model that is able to perform very well, with high confidence over most of the instances in a dataset. Groundbreaking work was done on the concept of boosting in the late 1980's and 1990's, proving its abilities mathematically. In fact, in 1990, Schapire [117] proved equivalence between weak and strong learnability, which turned out being a highly important result in statistics and ML. Hence, in theory, one only needs to build a model that is able to classify a little over half of the available data points correctly, as its performance can be boosted a posteriori.

Boosting in this form is not constrained to any specific learning model, making it a very general idea. In common for all boosting algorithms is that they iteratively learn weak

classifiers on the data, before adding their outcomes to produce the final prediction. When added, each classifier is weighted according to its accuracy, giving larger weight to more accurate classifiers. Additionally, each data point is usually weighted and re-weighted, depending on how the classifier performed on the observation in the previous iteration. Thus, "each successive classifier is thereby forced to concentrate on those training observations that are missed by previous ones in the sequence" [44]. An adaptive weighting and re-weighting scheme, following similar ideas as those loosely described above, was introduced in a seminal paper by Freund and Schapire [35] in 1995, as part of the infamous method *AdaBoost*. The authors later showed that the boosting mechanism is feasible in practice, providing experiments to assess the method on real learning problems [36].

Gradient boosting. It turns out that the AdaBoost algorithm can be formulated as a simple form of gradient descent minimization, with a specific loss function and step size search [81]. In fact, AdaBoost can be interpreted as a special case of *gradient boosting*, a term that encompasses methods that minimize a variety of loss functions, depending on the application, with *functional gradient descent* [44, 118]. The mathematical details behind these discoveries are quite technical and are not covered because of lack of time. However, the connection between boosting and optimization in certain function spaces is very interesting to keep in mind, and is an example of how mathematical concepts often are found to be linked together, yielding fruitful results when discovered. In particular, a gradient boosting procedure builds an iterative sequence of predictors in a greedy fashion [104]. Recall that, without loss of generality, we only consider binary classifiers, even though the predictors can be classifiers on categorical responses or regression models on continuous responses, as well. Let f_t, $t \in \{0, 1, 2, \ldots\}$, denote a sequence of binary classifiers, where f_T is the final classifier, given that the procedure has converged at iteration T. These classifiers are defined sequentially as $f_t = f_{t-1} + \alpha h_t$, where α is a step size and $h_t \in H$, where H is a chosen family of functions. The step size α can either be set a priori or estimated via techniques like, e.g., line search [92]. The function $h_t \colon \mathbb{R}^p \longrightarrow \mathbb{R}$ is chosen according to

$$h_t = \underset{h \in H}{\arg\min}\, \mathbb{E}[L(y, f_{t-1}(\boldsymbol{x}) + h(\boldsymbol{x}))], \tag{2.3}$$

where L is some smooth loss function and the expectation is calculated over all observations, $(\boldsymbol{x}, y)$, in a held-out test dataset. In fact, the expected loss in Equation (2.3) is usually minimized with functional gradient descent, i.e. by choosing h_t such that it approximates the negative gradient of the loss. Mathematically, this can be stated as

$$h_t(\boldsymbol{x}) \approx -g_t(\boldsymbol{x}, y) \coloneqq -\frac{\partial L(y, s)}{\partial s}\Big|_{s=f_{t-1}(\boldsymbol{x})},$$

for all test observations, $(\boldsymbol{x}, y)$. In binary classification problems, L is usually defined as the *binary cross-entropy loss*,

$$L(y, f_{t-1}(\boldsymbol{x})) = -y \log f_{t-1}(\boldsymbol{x}) - (1 - y) \log(1 - f_{t-1}(\boldsymbol{x})),$$

where $(\boldsymbol{x}, y)$ is an arbitrary observation. Importantly, $y \in \{0, 1\}$ and $f_{t-1}(\boldsymbol{x}) \in [0, 1]$, i.e. the function f_{t-1} predicts the probability that $\boldsymbol{x}$ corresponds to a binary indicator 1 response. Commonly, this probability prediction is transformed into a either-or prediction by introducing a *discrimination threshold*, which we discuss in Section 2.10. Note that these are technical details we do not discuss exhaustively here, but the cross-entropy loss is discussed in conjunction with the *categorical distribution* later in this work.

Gradient boosted decision trees. Finally, GBDTs are models that, on a superficial level, follow the procedures outlined above, after defining H as a family of decision trees. Based on the previous discussion, this means that each decision tree, h_t, is trained to

approximate the negative gradient of the loss function, L [104]. Thus, GBDTs improve decision trees, which are the weak learners of choice, to yield strong learning capabilities, by means of gradient boosting. Note that the size of each tree that is included in the iterative boosting procedure is not trivial to determine. One simple option for specifying the tree-size is to define a fixed number of leaf nodes, J, for every single tree. This should be set a priori as a hyperparameter of the gradient boosting algorithm. A complete discussion about this strategy can be found in Hastie et al. [44].

GBDTs yield impressive performance on many data modalities, especially on tabular data. For instance, methods like *XGBoost* [13] are generally recommended for regression and classification tasks on tabular data. In fact, a study by Shwartz-Ziv and Armon [122] showed that XGBoost outperforms deep learning models across a plethora of datasets, while requiring significantly less tuning of hyperparameters. According to Kotelnikov et al. [67], *CatBoost* [104] is the leading GBDT implementation. Prokhorenkova et al. [104] introduce two new techniques to the GBDT literature, created to combat a statistical issue called *prediction shift*, which they show is present in all previously implemented GBDTs. Because empirical results from these two papers indicate that CatBoost outperforms previous SOTA implementations, we employ CatBoost as a classifier on our binary classification tabular datasets later in this thesis. Despite this choice, we note that the performance differences between several of the most used GBDT methods, like CatBoost [104], LightGBM [60] and XGBoost [13], are not dramatic in most cases, as remarked by Shwartz-Ziv and Armon [122].

2.6 Deep Learning

Deep learning is a subfield of ML that encompassees methods for both supervised and unsupervised learning. This topic is not new — it has been subject to research for many decades — but its relevance has increased alongside the rise in accessible computing power and large datasets. In fact, deep learning dates back to the 1940s, when some of the earliest models were intended to simulate how learning could happen in the human brain. Because of this, deep learning also goes by the name *artificial neural networks* (ANNs) [40]. We use the names *artificial neural networks*, or simply *neural networks*, and *deep learning*, interchangeably. Goodfellow et al. [40] state that a solution for "the tasks that are easy for people to perform but hard for people to describe formally" is to "allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts". When drawing a graph that shows how all these concepts are connected, the graph will be deep, with many layers. Hence the name *deep learning*. This class of methods exploits that many small, linear terms can model any mathematical function when added and transformed via a nonlinearity. This idea is summarized in the *Universal approximation theorem*. Theorem 1 shows Goodfellow et al.'s [40] precise statement of the theorem.

Theorem 1 (*Universal Approximation Theorem*)**.** A feedforward neural network with a linear output layer and at least one hidden layer with any "squashing" activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units.

Theorem 1 introduces a plethora of terms that need to be understood in order to recognize what it states. Instead of defining each term directly here, we let the meaning of each of the terms become clear gradually as we work our way through the material. Thus, after reading this entire section, the theorem should hopefully be quite easily understood. However, in order to keep the discussion relatively light, we do not cover the concept of

Borel measurability in detail. In this context, it is sufficient to state that the Universal approximation theorem is valid for every continuous function on a closed and bounded subset of the real line. All in all, deep learning solves incredibly complex, nonlinear problems — like for example mapping an image, or raw sensory input data, to the object the image shows — by breaking down the problem into many small and simple calculations.

2.6.1 Feedforward Neural Networks

The most important model in the field of deep learning is the *feedforward neural network* (FNN) or the *multilayer perceptron* (MLP) [40]. The MLP lays the foundation for the rest of the field, since many of its components are cornerstones in more complex models. This is analogous to how linear regression is often treated as the quintessential model in ML or statistical learning. In fact, linear regression is closely related to the MLP, as will become apparent. Neural networks are often depicted graphically for ease of comprehension, such as the simple version of a MLP shown in Figure 2.3. We use this illustration throughout our discussion to exemplify how different calculations in FNNs are performed.

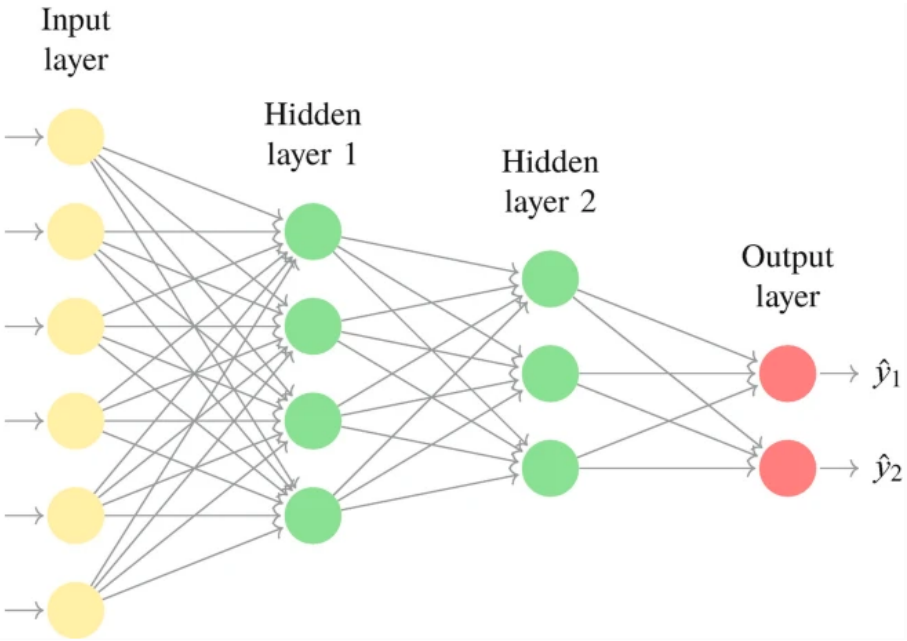


Figure 2.3: Example of a simple FNN for binary classification. The image is borrowed with permission from Dixon et al. [22].

In broad terms, MLPs consist of *layers*. There exists three types of layers; *input*, *hidden* and *output*. Most MLPs have one input layer, one output layer and one or more hidden layers. For instance, the FNN that is illustrated in Figure 2.3 has one *input layer*, one *output layer* and two *hidden layers* between the input and the output. Each layer consists of *nodes, neurons* or *units*. Each node represents a transformation, a mathematical operation, that is performed on its input. The number of nodes in a layer is often referred to as the *width* of the layer, while the number of hidden layers of the neural network is often referred to as the *depth* of the network itself. Notice that the nodes are connected by *weights*. In many cases, the way the nodes are connected decides what type of neural network it is. In the network in Figure 2.3, the nodes only connect to each other in the forward direction, from input to output, which is why it is a FNN. A more complicated neural network is the *recurrent neural network* (RNN), which allows for cyclic connections. RNNs are not treated here, but are mentioned to indicate that FNNs may be modified to

yield more complex models. Before explaining how the mathematical operations in MLPs are executed, we define some notation.

Notation. Recall our dataset, $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i = \{x_i^1, x_i^2, \ldots, x_i^p\}$, $i \in \{1, \ldots, n\}$, are feature values and y_i, $i \in \{1, \ldots, n\}$, are response values. When constructing a neural network for this dataset, the width of the input layer is set to p. That is, for a given observation $\boldsymbol{x}_i$, each unit j represents a x_i^j, $j \in \{1, \ldots, p\}$. For clarity, when we enumerate nodes in a figure we choose to start counting from the top. Thus, the FNN displayed in Figure 2.3 can be used for input data with $p = 6$ covariates, such that each node j represents the feature value x_i^j, $j \in \{1, 2, 3, 4, 5, 6\}$, for each observation $\boldsymbol{x}_i$, $i \in \{1, \ldots, n\}$. In addition, the hidden units are denoted by h_{ij}, $j \in \{1, \ldots, p_{h_i}\}$,, where the first index indicates hidden layer number i (enumerated from the left) and the second index indicates unit j (enumerated from the top) within hidden layer i. Note that p_{h_i} indicates the number of units in hidden layer i. For instance, in Figure 2.3, the hidden nodes are denoted as h_{1j}, $j \in \{1, 2, 3, 4\}$, and h_{2j}, $j \in \{1, 2, 3\}$. Furthermore, the output nodes are denoted by y_j, $j \in \{1, \ldots, p_{h_o}\}$, where the number of output nodes, p_{h_o}, depends on if the problem deals with classification or regression. Usually, $p_{h_o} = 1$ in a regression setting, while $p_{h_o} > 1$ in a classification setting. Finally, let w_{ijk} denote the weight that connects node j in the previous layer with node k in the current layer i. This means that in Figure 2.3, the weights are denoted as

$$w_{1jk}, \quad (j,k) \in \{(1,1), (2,1), \ldots, (6,1), (1,2), (2,2), \ldots, (6,2), (1,3), (2,3), \ldots, (6,3), (1,4), (2,4), \ldots, (6,4)\},$$

and

$$w_{2jk}, \quad (j,k) \in \{(1,1), \ldots, (4,1), (1,2), \ldots, (4,2), (1,3), \ldots, (4,3)\},$$

for the two hidden layers and

$$w_{3jk}, \quad (j,k) \in \{(1,1), (2,1), (3,1), (1,2), (2,2), (3,2)\},$$

for the output layer. Having defined our notation, we explain how an input observation is propagated through a FNN. In fact, doing all the mathematical operations through the graph from the input layer to the output layer is commonly referred to as *forward propagation* in the literature.

Forward propagation. The weights in a MLP can be interpreted as coefficients in a linear combination of the units they go from, feeding into a node in the subsequent layer. For example, given an arbitrary observation $\boldsymbol{x} \in \mathcal{D}$ and assuming known values for the weights w_{ijk}, $\forall(i,j,k)$, in the neural network, the value of unit h_{11} is calculated as

$$\text{val}_{\boldsymbol{x}}(h_{11}) = \sigma_{h_1}\left(b + \sum_{k=1}^{p} x^j w_{1k1}\right),$$

where $\text{val}_{\boldsymbol{x}}(\cdot)$ denotes the numerical value at node $\cdot$ based on observation $\boldsymbol{x}$, b is a so-called *bias* node and σ_{h_1} is called an *activation function*. Before explaining what these quantities represent, notice the analogy to linear regression; the number that is fed into the activation function is of the same form as a linear regression with x^j, $j \in \{1, \ldots, p\}$, as covariate values, given that the weights have already been estimated. As a side note, in linear regression, the weights w_{ijk} are normally called *coefficients* β_j. The bias node is not connected to any of the nodes in the previous layer. Analogously to the intercept in linear regression, this is often added to offset the linear combination of the weights and inputs, increasing its flexibility beyond linear models passing through the origin. For now,

we also assume that the value of the bias is known. Notice that this calculation, which was performed on node h_{11} as an example, can be done in a similar manner for all the other nodes, using the previous layer's values, commonly referred to as *activations*, as input. For example, for the output nodes in a FNN with two hidden layers, the calculations become nested,

$$a_{j_{\boldsymbol{x}}}^{(1)} = \text{val}_{\boldsymbol{x}}(h_{1j}) = \sigma_{h_1}\left(b_{h_1} + \sum_{k=1}^{p_{h_0}} x^k w_{1kj}\right), \quad j \in \{1, \ldots, p_{h_1}\}, \tag{2.4}$$

$$a_{j_{\boldsymbol{x}}}^{(2)} = \text{val}_{\boldsymbol{x}}(h_{2j}) = \sigma_{h_2}\left(b_{h_2} + \sum_{k=1}^{p_{h_1}} a_{k_{\boldsymbol{x}}}^{(1)} w_{2kj}\right), \quad j \in \{1, \ldots, p_{h_2}\}, \tag{2.5}$$

$$\hat{y}_{j_{\boldsymbol{x}}} = \text{val}_{\boldsymbol{x}}(y_j) = \sigma_{h_o}\left(b_{h_o} + \sum_{k=1}^{p_{h_2}} a_{k_{\boldsymbol{x}}}^{(2)} w_{3kj}\right), \quad j \in \{1, \ldots, p_{h_o}\}, \tag{2.6}$$

where $\sigma_{h_o}, \sigma_{h_2}, \sigma_{h_1}$ are the activation functions, and $b_{h_o}, b_{h_2}, b_{h_1}$ are the bias nodes, for the output layer, the second hidden layer and the first hidden layer, respectively. In addition, let there be $p_{h_o}, p_{h_2}, p_{h_1}, p_{h_0} = p$ nodes in the output layer, the second hidden layer, the first hidden layer and the input layer respectively. Notice that each layer may have a different activation function, which is why we explicitly added the sub-indices to σ.

Activation functions. The activation function, which we denote by σ, is a very important component of any neural network. As stated in Theorem 1, a "squashing" activation function is a prerequisite for the FNN to be able to approximate any (measurable) function. As the name indicates, the activation function essentially decides *how* each node should be "activated", i.e. what weight the value of the affine transformation that is used as input to the activation function should be given. An activation function is simply a fixed nonlinear function. For example, a FNN with σ defined as the identity function in all layers will behave like a linear regression and will never be able to learn any nonlinearities. Some commonly used activation functions are given in Table 2.2. Notice that the first four examples are functions of a single scalar x. In contrast, the softmax function is a vector-valued activation function, where the input and output are vectors in $\mathbb{R}^{p\times 1}$. In modern neural networks, the most used and recommended hidden activation function is ReLU [40]. Note that all the functions in Table 2.2 are "squashing", since their outputs are constrained to a certain area of $\mathbb{R}$. The activation function of the output layer has to be chosen depending on the problem at hand. Table 2.3 shows some common examples of supervised ML problems, alongside their corresponding output layer activation function and number of output nodes.

Table 2.2: Some commonly used activation functions. The first four functions are defined as $\sigma_\cdot : \mathbb{R} \longrightarrow \mathbb{R}$, while the softmax function is defined as $\boldsymbol{\sigma}_{\text{soft}} : \mathbb{R}^{p\times 1} \longrightarrow \mathbb{R}^{p\times 1}$, such that $\boldsymbol{\sigma}_{\text{soft}} = \{\sigma_{\text{soft}}(\boldsymbol{x})^1, \ldots, \sigma_{\text{soft}}(\boldsymbol{x})^p\}$.

Name	**Activation function**
Rectified linear unit (ReLU)	$\sigma_{\text{ReLU}}(x) = \max(0, x)$
Logistic sigmoid	$\sigma_{\text{logsig}}(x) = \frac{1}{1+\exp(-x)}$
Hyperbolic tangent (tanh)	$\sigma_{\tanh}(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$
Sigmoid-weighted linear unit (SiLU)	$\sigma_{\text{SiLU}}(x) = x\sigma_{\text{logsig}}(x)$
Softmax	$\sigma_{\text{soft}}(\boldsymbol{x})^j = \frac{\exp(x^j)}{\sum_{i=1}^{p}\exp(x^i)}$, $j \in \{1, \ldots, p\}$

Linear algebraic implementations. Since deep learning operations are linear, they are very well suited for highly efficient implementations using techniques from numerical

Table 2.3: Common problems solved with supervised deep learning models, alongside their corresponding output activation function and number of output nodes.

Problem	**Output activation**	**Output node(s)**
Binary classification	Logistic sigmoid or softmax	One or two
Multiclass classification	Softmax	Number of categories
Regression	Identity function	One

linear algebra. The equivalent formulations of equations (2.4), (2.5) and (2.6) for all units simultaneously are

$$\begin{aligned}\boldsymbol{a}^{(1)} &= \boldsymbol{\sigma}_{h_1}\left(\boldsymbol{b}^{(1)} + \mathbf{W}^{(1)T}\boldsymbol{x}\right),\\ \boldsymbol{a}^{(2)} &= \boldsymbol{\sigma}_{h_2}\left(\boldsymbol{b}^{(2)} + \mathbf{W}^{(2)T}\boldsymbol{a}^{(1)}\right),\\ \hat{\boldsymbol{y}} &= \boldsymbol{\sigma}_{h_o}\left(\boldsymbol{b}^{(o)} + \mathbf{W}^{(3)T}\boldsymbol{a}^{(2)}\right),\end{aligned}$$

where the activation functions are vector-valued. Moreover, the weight matrices $\mathbf{W}^{(j)}$, the activations $\boldsymbol{a}^{(j)}$, the input vector $\boldsymbol{x}$, the output vector $\hat{\boldsymbol{y}}$ and the bias vectors $\boldsymbol{b}^{(j)}$ are defined as

$$\mathbf{W}^{(j)} = \begin{pmatrix} w_{j11} & w_{j12} & w_{j13} & \cdots & w_{j1p_{h_j}} \\ w_{j21} & w_{j22} & w_{j23} & \cdots & w_{j2p_{h_j}} \\ \vdots & \vdots & \vdots & \cdots & \cdots \\ w_{jp_{h_{(j-1)}}1} & w_{jp_{h_{(j-1)}}2} & w_{jp_{h_{(j-1)}}3} & \cdots & w_{jp_{h_{(j-1)}}p_{h_j}} \end{pmatrix},$$

$$\boldsymbol{a}^{(j)} = \begin{pmatrix} a_1^{(j)} & a_2^{(j)} & \cdots & a_{p_{h_j}}^{(j)} \end{pmatrix}^T,$$

$$\boldsymbol{x} = \begin{pmatrix} x^1 & x^2 & \cdots & x^p \end{pmatrix}^T,$$

$$\hat{\boldsymbol{y}} = \begin{pmatrix} \hat{y}_1 & \hat{y}_2 & \cdots & \hat{y}_{p_{h_o}} \end{pmatrix}^T,$$

and

$$\boldsymbol{b}^{(j)} = \underbrace{\begin{pmatrix} b_{h_j} & b_{h_j} & b_{h_j} & \cdots \end{pmatrix}}_{p_{h_j} \text{ elements}}^T.$$

Notice that the explicit dependence on the observation $\boldsymbol{x}$ is dropped in the subscripts of the activations. Moreover, for clarity, let $p_{h_3} = p_{h_o}$ in this case, keeping the example in Figure 2.3 in mind. Furthermore, note that an arbitrary scalar activation function, σ, can be transformed into a vector-valued activation function, $\boldsymbol{\sigma}$, by applying it component-wise to its input vector. This yields the transformation

$$\boldsymbol{\sigma}(\boldsymbol{x}) = \{\sigma(x^1), \sigma(x^2), \ldots, \sigma(x^p)\},$$

when applied to a vector $\boldsymbol{x} = \{x^1, x^2, \ldots, x^p\}$. Thus, in the example above, we can define $\boldsymbol{\sigma}_{h_1}, \boldsymbol{\sigma}_{h_2}$ and $\boldsymbol{\sigma}_{h_o}$ based on any scalar- or vector-valued activation function. Finally, even though we only work with one arbitrary observation, $\boldsymbol{x} \in \mathcal{D}$, in this exposition, we usually work with groups of observations simultaneously in practice. In fact, we commonly forward propagate *batches* of observations, a concept we return to later.

Parameter estimation. Until now, we have assumed that the weights and biases, i.e. the parameters of the FNN, are known. However, in practice they are not known

a priori and need to be estimated before we can use the neural network for anything useful. Training, i.e. estimating the parameters of, deep learning models is an active area of research, where many different algorithms have been suggested for optimal predictive performance and computational efficiency. Because of lack of time, we do not dive deep into this vast field, but instead mention the two most frequently used algorithms for training deep learning models; *mini-batch stochastic gradient descent* and *backpropagation*. Backpropagation, popularized by Rumelhart et al. [112], is an algorithm for calculating the gradient of a loss function with respect to the parameters of a neural network. Said simply, it is an efficient implementation of the chain rule for differentiation applied to the loss function of a neural network. This loss function measures the error between the true response values and the predicted responses in the output layer, and its specific shape is defined based on the type of problem we are interested in solving. After calculating the gradient, mini-batch stochastic gradient descent is typically used. Specifically, this term refers to a group of methods that minimize a loss function iteratively by modifying the weights and biases in the direction of the *approximated* negative gradients. Notice that, in practice, the gradients are approximated, with, e.g., backpropagation, on stochastic batches of data, instead of calculated exactly on the entire dataset. This is the reason why the terms *mini-batch* and *stochastic* are used in conjunction with *gradient descent*, where the former explicitly states that the stochastic batches of data are smaller than the actual available dataset. As a side note, observe the analogy to the functional gradient descent techniques discussed in Section 2.5.1, which are similar deterministic techniques, applied to functional space instead of parameter space. The simplest mini-batch stochastic gradient descent algorithm works by performing the iterations

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \widehat{\boldsymbol{\nabla}}_{\boldsymbol{\theta}} L(y, f(\boldsymbol{x})),\ t \in \{1, 2, \ldots\}, \tag{2.7}$$

where $\boldsymbol{\theta}_t$, $t \in \{0, 1, 2, \ldots\}$, are the iteratively modified parameters of the neural network $f(\cdot)$, L is a smooth loss function of choice, η is a *learning rate* and $(\boldsymbol{x}, y) \in \mathcal{D}$ represents an arbitrary observation. Moreover, we let $\widehat{\boldsymbol{\nabla}}_{\boldsymbol{\theta}} L(y, f(\boldsymbol{x}))$ denote a mini-batch stochastic approximation of the true gradient, $\boldsymbol{\nabla}_{\boldsymbol{\theta}} L(y, f(\boldsymbol{x}))$. Consistent with our convention of working with an arbitrary observation $(\boldsymbol{x}, y) \in \mathcal{D}$, we do not explicitly specify in Equation (2.7) that the forward propagation in $f(\boldsymbol{x})$, as well as the approximation of the true gradient term, is commonly performed over batches of observations. The learning rate, η, determines the step size that is taken in the direction of the approximated gradient, which plays an important role during optimization. In continuation, there exists many extensions and improvements on the basic mini-batch stochastic gradient descent iteration scheme from Equation (2.7), like *AdaGrad* [27], *RMSProp* [46] and *Adam* [63]. We do not go into detail about any of these optimizers, but one of the main improvements in all these examples is the introduction of an *adaptive* learning rate, where the learning rate is modified in each iteration. The strategy for how the learning rate is changed between iterations differs in each of the optimizers. The interested reader is referred to Goodfellow et al. [40], for more details on parameter estimation in neural networks, including a thorough explanation of backpropagation, overview of how to choose the loss function and how to stochastically optimize it by tweaking the network parameters.

2.6.2 Neural Network Notation

The notation we have used thus far for neural networks was designed mainly for instructional purposes, explicitly highlighting the most significant details. However, our previous notation is slightly cumbersome in practice. For reference, we define some more practical notation for representing neural network architectures. Most of the notation given in the following list is inspired by Kotelnikov et al. [67] and Xu et al. [146]:

- $\texttt{FC}_{u \to v}(\boldsymbol{x})$: represents a fully connected linear layer. Applies an affine transformation to an input $\boldsymbol{x} \in \mathbb{R}^{u \times 1}$ resulting in a vector in $\mathbb{R}^{v \times 1}$.
- $\texttt{ReLU}(\boldsymbol{x})$: represents a ReLU activation function applied element-wise to an arbitrary input $\boldsymbol{x}$. This is equivalent to the application of the scalar function denoted by σ_{ReLU} in Table 2.2 to each element of $\boldsymbol{x}$.
- $\texttt{SiLU}(\boldsymbol{x})$: represents a SiLU activation function applied element-wise to an arbitrary input $\boldsymbol{x}$. This is equivalent to the application of the scalar function denoted by σ_{SiLU} in Table 2.2 to each element of $\boldsymbol{x}$.
- $\texttt{tanh}(\boldsymbol{x})$: represents a tanh activation function applied element-wise to an arbitrary input $\boldsymbol{x}$. This is equivalent to the application of the scalar function denoted by $\sigma_{\tanh}$ in Table 2.2 to each element of $\boldsymbol{x}$.
- $\texttt{softmax}(\boldsymbol{x})$: represents a softmax activation function applied to an arbitrary input $\boldsymbol{x}$. This is equivalent to the application of the vector-valued function denoted by $\boldsymbol{\sigma}_{\text{soft}}$ in Table 2.2 to $\boldsymbol{x}$. The output vector has elements $\sigma_{\text{soft}}(\boldsymbol{x})^j$, $j \in \{1, \ldots, p\}$, when $\boldsymbol{x} \in \mathbb{R}^{p \times 1}$.
- $\texttt{Dropout}(\boldsymbol{x})$: represents *dropout* in neural networks. The interested reader is referred to Srivastava et al. [133] for details on this technique used to prevent overfitting. This is a *regularization* method for neural networks, an idea we discuss in Section 2.7.

Closing remarks. Before moving on to some applications of deep learning, notice how the idea of deep learning itself reveals a need for interpretability and explainability. In many cases, it is difficult to monitor exactly what is going on with the data in each step during forward propagation and backpropagation, especially when using deeper and deeper networks. These methods are however still attractive, because they have achieved very high performance in many problems.

2.7 Autoencoders

An *autoencoder* is a deep learning architecture that is primarily used to reduce the dimension of a dataset by learning efficient encodings of the original features [69]. This task is often referred to as performing *representation learning*, which can be either supervised or unsupervised. The autoencoder belongs to the unsupervised learning paradigm, because its learning objective is essentially to reconstruct its input data. Why is this reconstruction interesting? The idea is that if a properly constrained autoencoder is able to reconstruct its input data, it has likely implicitly developed a useful and more compact representation of the input data, depending on its exact constraints. In the following, we describe more rigorously what type of constraints can be imposed and how the reconstruction of the data should be.

Intuition. The general autoencoder consists of two parts: an *encoder* and a *decoder*. These two parts cooperate in order to recreate the input. The encoder is a function $\boldsymbol{f}$ that learns a different representation of the inputs $(\boldsymbol{x}_i, y_i) = \{x_i^1, \ldots, x_i^p, y_i\}$, $i \in \{1, \ldots, n\}$. This representation is often referred to as the *latent* representation, which lives in a *latent* space. In the following, we work with an arbitrary observation $(\boldsymbol{x}, y) \in \mathcal{D}$, consistent with our previous practice. Thus, the encoder is a function such that $\boldsymbol{f}(\boldsymbol{x}, y) = \boldsymbol{z}$, where $\boldsymbol{z}$ is a new representation of $(\boldsymbol{x}, y)$. The decoder is another function, $\boldsymbol{h}$, that learns to reconstruct the input $(\boldsymbol{x}, y)$ from $\boldsymbol{z}$, $\boldsymbol{h}(\boldsymbol{z}) = \boldsymbol{r}$, where $\boldsymbol{r}$ is a reconstruction of $(\boldsymbol{x}, y)$. Ideally, the reconstruction $\boldsymbol{r}$ should be "similar" to $(\boldsymbol{x}, y)$, but not identical. For example,

an autoencoder which simply learns to copy or memorize the input, such that $(\boldsymbol{x}_i, y_i) = \boldsymbol{r}_i$, $\forall i \in \{1, \ldots, n\}$, is useless, because this memorization leads to an uninteresting latent representation. We return to *why* later. The encoder-decoder architecture can thus be interpreted as a strategy for producing a new representation of the input, $(\boldsymbol{x}, y)$, during training. After training, the model can be used for several tasks. The encoder can be used to build a more efficient representation of an observation $(\boldsymbol{x}, y)$, which subsequently can be used for some other task, such as classification or regression. This idea is closely related to *dimensionality reduction* and *feature extraction*. Furthermore, the decoder can be used to generate synthetic data, for example by randomly sampling a point in the latent space of the autoencoder, before feeding the point through the decoder. In this way, an autoencoder can be interpreted as a generative model, albeit a rudimentary variant. We return to these ideas later, as they are crucial in this work.

Mathematical formulation. The general problem an autoencoder seeks to solve may be formulated as an optimization problem,

$$\arg\min_{\boldsymbol{h},\boldsymbol{f}} \mathbb{E}[L((\boldsymbol{x}, y), \boldsymbol{h}(\boldsymbol{f}(\boldsymbol{x}, y)))], \tag{2.8}$$

where the expectation is calculated over all observations $(\boldsymbol{x}, y) \in \mathcal{D}$ and L is a loss function that penalizes $\boldsymbol{r} = \boldsymbol{h}(\boldsymbol{f}(\boldsymbol{x}, y))$ when it is "dissimilar" to an arbitrary $(\boldsymbol{x}, y) \in \mathcal{D}$. Thus, the "similarity" between input and reconstruction we have been referring to is defined via the loss function L. A typical loss function for autoencoders is the *mean squared error*, $L(\boldsymbol{x}, \boldsymbol{h}(\boldsymbol{f}(\boldsymbol{x}, y))) = \frac{1}{p}\|(\boldsymbol{x}, y) - \boldsymbol{h}(\boldsymbol{f}(\boldsymbol{x}, y))\|_2^2$, which is usually averaged over all observations in the dataset to estimate the expectation in Equation (2.8). Notice that we have not defined specifically what the functions $\boldsymbol{f}$ and $\boldsymbol{h}$ should look like. They can be calculated using any mathematical, statistical or ML model, but in the context of deep learning we naturally parameterize them as neural networks. The typical autoencoder employs relatively simple FNNs for $\boldsymbol{f}$ and $\boldsymbol{h}$, but they may be designed with arbitrary complexity depending on the application. An example of a simple autoencoder architecture is shown in Figure 2.4.

Under- and overcomplete variants. As mentioned, an important consideration when working with autoencoders is to avoid direct memorization of the input, i.e. such that $(\boldsymbol{x}, y) = \boldsymbol{r} = \boldsymbol{h}(\boldsymbol{z}) = \boldsymbol{h}(\boldsymbol{f}(\boldsymbol{x}, y))$. Oftentimes, this can happen in two different scenarios. The first scenario is when the encoder $\boldsymbol{f}$ is an excessively nonlinear function. For example, one could imagine a highly nonlinear encoder that learns to represent each training point $(\boldsymbol{x}_i, y_i) \in \mathcal{D}$ by its index i [40]. This would yield zero reconstruction loss, $L((\boldsymbol{x}_i, y_i), \boldsymbol{h}(\boldsymbol{f}(\boldsymbol{x}_i, y_i)))$, $\forall (\boldsymbol{x}_i, y_i)$, $i \in \{1, \ldots, n\}$, meaning that Equation (2.8) is optimized with respect to any loss function with image in the non-negative real numbers, for example a mean squared error loss. Despite this, such an encoder yields a useless model, because it is not *generalizable*. Specifically, the term *generalization* refers to the ability of a model to adjust its behaviour properly to previously unseen data from the same underlying distribution as the training data. Note that this example hardly occurs in practice, but it illustrates how encoders that are exceedingly nonlinear can yield uninteresting results. The second scenario occurs when the number of units in the output layer of the encoder is greater than or equal to the number of input nodes. Specifically, when the dimension of the latent representation is larger than the dimension of the input, the autoencoder is *overcomplete*. In this scenario, the network is generally able to copy the input data without learning an interesting latent representation, which yields a model that suffers from poor generalization.

An *undercomplete* autoencoder has an architecture where the dimension of the latent space is smaller than the number of input nodes [40]. We commonly refer to the internal layer of an undercomplete autoencoder as a "bottleneck" [69], which can be interpreted

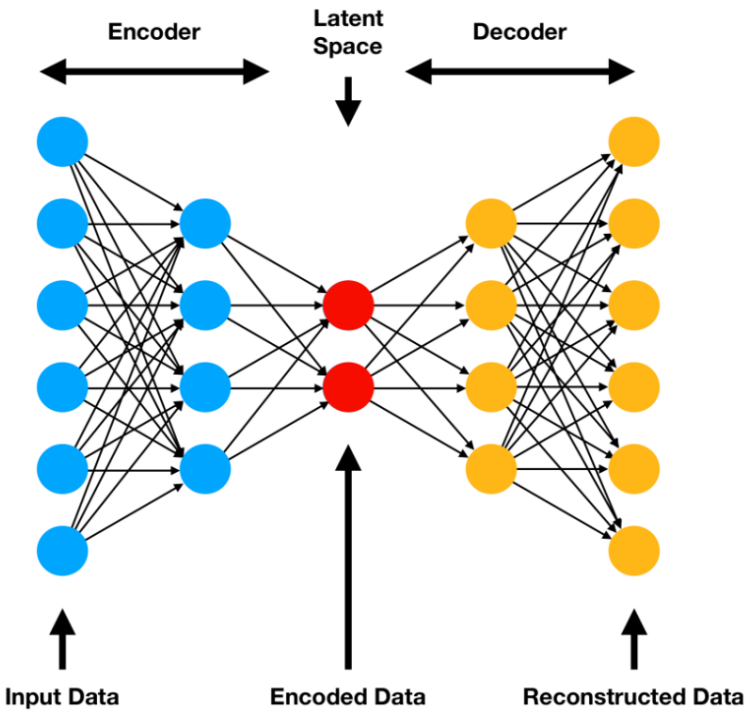


Figure 2.4: Example of an undercomplete autoencoder architecture. The image is borrowed with permission from Flores [34].

from Figure 2.4. This type of autoencoder is interesting in practice, because it avoids the second scenario mentioned previously. The lower latent dimension "forces" the encoder to learn the most prominent and important features in the data. An interesting note is that when $\boldsymbol{f}$ is linear and L is the mean squared error loss, an undercomplete autoencoder learns to span the same subspace as *principal component analysis* (PCA), but without guaranteeing orthogonality of the vectors [40]. In contrast, undercomplete autoencoders with nonlinear encoders have the potential of learning nonlinear dimensionality reductions. Yet, it is important to avoid giving the functions $\boldsymbol{f}$ and $\boldsymbol{h}$ too much capacity, as discussed in the first scenario above.

Regularization. Notice that both of the scenarios where mere memorization can occur, can be seen as special cases of *overparameterization*, which is a frequently occurring problem in ML and statistical modelling. When models are given too many parameters, they learn to overfit to the training data, which increases the generalization error. Because of this, most statistical or ML models are constrained, usually by a *regularization* procedure, which can be implemented either explicitly or implicitly. Implicit regularization can be applied in a plethora of ways, some ways we have already discussed, for example by choosing specific loss functions, choosing an undercomplete architecture or modifying the optimization algorithm for training the neural networks. These forms of regularization may seem more opaque, depending on the approach, but rest assured that most models include some sort of regularization technique. In contrast to their implicit counterparts, explicit regularization is more noticeable. In fact, a large group of problems can be represented in the form

$$\arg\min_{f \in \mathcal{F}} \mathbb{E}[L(y', f(\boldsymbol{x}')) + \lambda \Omega(\boldsymbol{x}', f)], \tag{2.9}$$

where $(\boldsymbol{x}', y')$ is an arbitrary observation in any dataset, $\mathcal{F}$ is a family of functions in which one is searching, L is a loss function measuring the difference between the model output, $f(\boldsymbol{x}') = \hat{y}'$, and the true label, y', and Ω is a regularization term that can depend on

the feature values and the function. The addition of the second term in the optimization problem makes this regularization technique explicit. This term introduces the hyperparameter λ, which is important in balancing the two terms in Equation (2.9). Since this is a hyperparameter, we must determine this value before optimizing the objective. When $\lambda \to 0$, the regularization term becomes unimportant and only the loss is minimized. When $\lambda \to \infty$, the importance of the loss function vanishes, yielding a highly regularized function f. This greatly limits its flexibility, within the constraints of the family of functions $\mathcal{F}$. In fact, the *unsupervised* autoencoder problem from Equation (2.8) fits nicely into this representation, by setting the observation $(\boldsymbol{x}, y)$ as the true label, $\boldsymbol{h}(\boldsymbol{f}(\boldsymbol{x}, y)) = \boldsymbol{r}$ as the model output, and adding a regularization term. Thus, memorization in an autoencoder can be avoided via explicit regularization, by solving the problem

$$\arg\min_{\boldsymbol{h},\boldsymbol{f}} \mathbb{E}[L((\boldsymbol{x}, y), \boldsymbol{h}(\boldsymbol{f}(\boldsymbol{x}, y))) + \lambda\Omega((\boldsymbol{x}, y), \boldsymbol{h}, \boldsymbol{f})],$$

where we let the general regularization term depend on both the encoder and decoder. In this manner, we are not limited to solely using undercomplete autoencoders, but can utilize overcomplete autoencoders with added regularization as well. The regularization term, Ω, may take several shapes, depending on how one wants to penalize model complexity. For instance, it can be defined in order to yield a *sparse* latent representation, such that the encoder cannot simply act as the identity function, but needs to respond to certain statistics in the input. Another example is to let

$$\Omega((\boldsymbol{x}, y), \boldsymbol{f}) \coloneqq \left\| \frac{\partial \boldsymbol{f}(\boldsymbol{x}, y)}{\partial(\boldsymbol{x}, y)} \right\|_F^2,$$

where $\|\cdot\|_F$ is the *Frobenius* norm, i.e. the sum of squared elements. This penalizes large derivatives of the latent representation $\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}, y)$, such that the encoder is forced to learn a more robust function, i.e. a function that exhibits minimal change when $(\boldsymbol{x}, y)$ changes slightly. Such autoencoders are called *contractive autoencoders* (CAEs) [40].

Closing remarks. As mentioned, the most important applications of autoencoders are dimensionality reduction, representation learning, feature extraction and generative modelling. If the optimization is successful, the latent representation should be of lower dimensionality in comparison to the input, without loss of the most critical information. For undercomplete autoencoders this dimensionality reduction is obvious, but overcomplete autoencoders may also be used for dimensionality reduction. For example, when learning a function $\boldsymbol{f}$ to encode the input, the family of functions $\mathcal{F}$ is typically parametric. Thus, a parametric function $\boldsymbol{f}$ learnt by the network can be seen as a dimensionality reduction from the number of features in the training data to the number of parameters of $\boldsymbol{f}$. This idea of learning a parametric function $\boldsymbol{f}$ is useful in the following section, where we discuss an extension of the autoencoder.

2.8 Variational Autoencoders

A *variational autoencoder* (VAE) can be regarded as an extension of the autoencoder architecture, although with a more intricate mathematical foundation and usually with different applications. Succinctly described, VAEs are autoencoders combined with *latent variable models* [40]. Recall from Section 2.4 that a VAE is a likelihood-based latent variable deep generative model, whose main objective is to estimate the true distribution of the population that our dataset, $\mathcal{D}$, is sampled from, which we represent with the density $p^*(\boldsymbol{x})$. Notice that we have changed the notation from Section 2.2 slightly. Instead of letting $(\boldsymbol{X}, Y) = \{X^1, \ldots, X^p, Y\}$ represent our random variables, we define $\boldsymbol{X} \coloneqq \{X^1, \ldots, X^p, Y\}$. The reason behind this change is that, in the general case, Y does not

have to be treated distinctly from the rest of the features. In fact, we already saw in the previous section that we can train autoencoders to reconstruct its input, where Y was interpreted as any of the other features in $(\boldsymbol{X}, Y)$. Thus, unless otherwise specified, if our objective is not to make decisions regarding a specific response, like in decision theory, we prefer to interpret all the $p+1$ random variables similarly. This also makes the notation slightly more refined, because we can refer to a general arbitrary observation vector $\boldsymbol{x}$, instead of the vector $(\boldsymbol{x}, y)$ we used throughout Section 2.7.

Latent variable. A latent variable model introduces one or more latent variables, which are assumed connected to the observable random variable $\boldsymbol{X}$ in some unobservable manner. Specifically, the VAE is developed under the assumption of *one* unobserved random variable $\boldsymbol{Z}$. As a side note, notice that $\boldsymbol{X}$ and $\boldsymbol{Z}$ are in fact random vectors, i.e. they contain random variables as elements. Despite this, we refer to $\boldsymbol{X}$ and $\boldsymbol{Z}$ as random variables as well. Returning from our digression, the joint distribution of $\boldsymbol{X}$ and $\boldsymbol{Z}$ is $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$, following the chain rule of probability. This factorization is not just a mathematical trick; it indicates the *generative process* that is assumed in a VAE. In detail, we assume that a single observation $\boldsymbol{x}$ is generated from $p^*(\boldsymbol{x})$ by following the process

$$\begin{aligned} &\text{Sample } \boldsymbol{z}, \text{ where } \boldsymbol{Z} \sim p(\boldsymbol{z}), \\ &\text{Sample } \boldsymbol{x}, \text{ where } \boldsymbol{X}|\boldsymbol{Z} = \boldsymbol{z} \sim p(\boldsymbol{x}|\boldsymbol{z}), \end{aligned}$$

which might seem unintuitive at first. However, a little philosophical exercise might highlight why this procedure is reasonable. Think about how you would draw a physical object, for example a car, on a piece of paper. Most likely, you would begin by sketching characteristics like its shape, size and viewing angle, before filling in details like wheels, door handles and brand. Thus, *your* generative process is to determine and sketch the high-level attributes of the car, before adding and perfecting the details. Transferring this observation to the generative process assumed in our simple latent variable model, the latent variable realization $\boldsymbol{z}$ represents the high-level attributes of your object, drawn from a collection (distribution) of plausible attributes $p(\boldsymbol{z})$, while the observation $\boldsymbol{x}$ represents the final product, which is completed based on a plausible collection (distribution) of details $p(\boldsymbol{x}|\boldsymbol{z})$ depending on the realization $\boldsymbol{z}$. This argument is not intended to replace a rigorous justification of the generative process, or to start a philosophical debate concerning how to draw objects, but hopefully we could agree that this way of thinking has some merit.

Likelihood maximization. As expressed many times already, the true data distribution is unknown, and the objective of generative models is to estimate this distribution via a set of observations $\mathcal{D}$. For likelihood-based models, the strategy is to maximize the likelihood of the observed data,

$$p(\boldsymbol{\theta}; \mathcal{D}) \coloneqq \prod_{\boldsymbol{x} \in \mathcal{D}} p_{\boldsymbol{\theta}}(\boldsymbol{x}),$$

where we let $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ denote the likelihood of one sample $\boldsymbol{x} \in \mathcal{D}$. We assume that $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ belongs to a parametric family of distributions, parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{Q \times 1}$, where Q is defined based on the chosen family. Notice that the likelihood is a function of the parameters and not the data. When the likelihood over all samples is maximized, the idea is that $p(\boldsymbol{\theta}; \mathcal{D}) \approx p^*(\boldsymbol{x})$, with the approximation becoming increasingly better with a larger number of observations in the dataset. In the following, we concentrate on only one observation $\boldsymbol{x} \in \mathcal{D}$. Then, each distribution in the product may be written as

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) \mathrm{d}\boldsymbol{z} = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} = \mathbb{E}_{p(\boldsymbol{z})}[p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})], \tag{2.10}$$

where the latent variable is marginalized out since it cannot be observed. The notation $\mathbb{E}_{p(\boldsymbol{z})}[p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})]$ is a shorthand, abuse of notation that represents $\mathbb{E}_{\boldsymbol{Z}\sim p(\boldsymbol{z})}[p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{Z})] = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})\mathrm{d}\boldsymbol{z}$, which is possible as a result of the *law of the unconscious statistician*. This type of notation is used in the rest of the thesis.

There are a few important things to notice concerning Equation (2.10). First, the *prior* $p(\boldsymbol{z})$ does not have any parameters $\boldsymbol{\theta}$. That is because this is usually set a priori, in analogy to any prior probability in Bayesian statistics, disregarding hierarchical priors [38]. Of course, this analogy is slightly problematic due to Bayesian priors usually being posed on parameters, but the idea is still illuminating. Second, a VAE is usually assumed to be working with a continuous latent variable, as it simplifies the computations. The case with a discrete latent variable is treated analogously, by simply replacing the integral with a sum over all the discrete levels of the variable. Keeping our objective in mind, the crucial question is: how can the integral in Equation (2.10) be calculated?

Intractability. Unfortunately, unless very strict assumptions are made on $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, the integral in Equation (2.10) is intractable, due to its high dimensionality [113]. For the interested reader, *Probabilistic principal component analysis* (pPCA) [137] is one example of a method that yields a tractable integral, by imposing, among other assumptions, linear dependence between $\boldsymbol{Z}$ and $\boldsymbol{X}$. In many applications, this linear assumption is limiting and excessively simple, meaning that this analytically computable model is not suitable. In the general case, when our models are more complex, the integral needs to be approximated. There exists several ways of computing such an approximation, for example by using *Markov Chain Monte Carlo* (MCMC) methods or *variational inference* (VI). In the remainder of this section, we dive deeper into the latter method, which is the technique used in VAEs. The interested reader may consult Speagle [131] for a conceptual introduction to MCMC.

Variational inference. Following Bayes' rule, the likelihood of each observation $\boldsymbol{x} \in \mathcal{D}$ can be rewritten as

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}.$$

Notice that the likelihood cannot be directly maximized using this expression either; $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is tractable to compute, but the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ is not [64, 113]. Since we do not have access to this posterior, the VI approach is used to approximate it with a simpler distribution from a family of tractable parameterized probability distributions. Hence, we define the approximate posterior

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}).$$

Observe that we introduce the *variational parameters* $\boldsymbol{\phi} \in \mathbb{R}^{V\times 1}$, where V is defined based on the chosen family. By replacing the true posterior with this approximation, we can maximize the likelihood of the marginal, $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, via a *surrogate* objective.

Before discussing the surrogate problem, and thus how the integral in Equation (2.10) may be approximated, we try to close the gap between autoencoders and VAEs. Instead of optimizing a function to deterministically regenerate its input, the objective of a VAE is to *approximately* maximize the likelihood of the data, $p(\boldsymbol{\theta};\mathcal{D})$. In other words, the deterministic encoder, $\boldsymbol{f}(\boldsymbol{x})$, and decoder, $\boldsymbol{h}(\boldsymbol{z})$, of an autoencoder are replaced by their stochastic counterparts, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, respectively. This modification "forces" the model to learn continuous and highly structured latent spaces, depending on the parametric family of distributions that each of these densities belong to. An interesting side note is that a VAE may also be viewed as a form of regularized autoencoder, where the regularization depends on the choice of encoder family. In practice, the objective of the encoder, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, is to learn a conditional probability distribution over the latent variables,

while the objective of the decoder, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, is to transform samples from $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ such that the conditional probability distribution $\forall \boldsymbol{x} \in \mathcal{D}$ is comparable to the corresponding distribution in the input data.

Evidence lower bound. Returning to our surrogate problem, we derive the objective function that is optimized in VAEs. For clarity, recall that we focus on one single observation $\boldsymbol{x} \in \mathcal{D}$. Following the standard procedure in maximum likelihood estimation, we are interested in maximizing the log-likelihood of the marginal distribution (the *marginal log-likelihood*) of $\boldsymbol{x}$, which can be rewritten as

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \mathrm{d}\boldsymbol{z} \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{z} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \right]}_{\mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \right]}_{D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))}.
\end{aligned} \tag{2.11}$$

The first term in Equation (2.11) is often called the *variational lower bound* or the *evidence lower bound* (ELBO), which we denote by $\mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$. The second term is the *Kullback-Leibler* (KL) divergence between $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, denoted by $D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))$, which is always non-negative [64]. In this notation, the expressions $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ refer to the probability density functions of the random variables they describe. Since we do not have access to the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, we replace direct maximization of Equation (2.11) with maximization of the ELBO. This idea gives rise to our surrogate problem, which is to maximize a lower bound of the marginal log-likelihood by using $\mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ as the objective function,

$$\begin{aligned}
\arg\max_{\boldsymbol{\theta},\boldsymbol{\phi}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \arg\max_{\boldsymbol{\theta},\boldsymbol{\phi}} \{\mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) + D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))\} \\
&\geq \arg\max_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}).
\end{aligned}$$

As a side note, this means that the ELBO also is a lower bound to the logarithm of the integral in Equation (2.10). This can be shown directly via *Jensen's inequality*, see, e.g., Luo [76]. We skip this calculation deliberately, as we think the one performed above is much more illustrative of the relationship between the likelihood and the ELBO. Nevertheless, this illustrates how VI is used in VAEs to solve the problem we initially posed, by reducing our original problem to optimizing a proxy objective, which contains integrals (expectations) that are tractable.

In consequence, we have developed an objective that can be optimized numerically. Before explaining *how* this optimization can be done, we discuss some consequences of optimizing a lower bound of the true objective. In fact, optimizing a lower bound might not be enough to yield an adequate VAE. Notice that Equation (2.11) does not guarantee that a numerically optimized $\mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ is close to the true marginal log-likelihood. The discrepancy between the two quantities is measured by $D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))$, which is zero only when $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, leading to a tight bound. Actually, a situation

where the ELBO is optimized and $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \not\approx p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ does exist, for example when the family of functions chosen for $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is overly simplistic. The problem is that we cannot deduce if the posterior should be assumed as more flexible or if the approximate posterior is close to the true posterior. However, there is some light at the end of the tunnel. Observe that the left-hand side of Equation (2.11), $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, is constant with respect to the variational parameters, $\boldsymbol{\phi}$. It follows that the sum on the right-hand side is constant with respect to $\boldsymbol{\phi}$ as well. Thus, a minimization of $D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))$ with respect to $\boldsymbol{\phi}$ necessarily follows from the maximization of $\mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ with respect to $\boldsymbol{\phi}$, and vice versa. Hence, optimizing the ELBO with respect to the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ will concurrently minimize $D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))$ and maximize $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, which are our two main objectives [64, 76]. This adds some further intuition to why the ELBO is used as objective function in VAEs, as well as in most VI methods. In general, however, we still cannot guarantee tightness of the lower bound, because this depends on our parametric choices for the encoder and decoder. *Inference suboptimality*, as Cremer et al. [18] call the mismatch between the true and approximate posterior, is a well-known nuisance in VAEs. We refer the interested reader to Chapter 4 in Tomczak [138], which contains a broad overview of possible suggested remedies to this problem, and other well-known problems, as well as references to further readings.

To gain additional insights into the objective function, we rewrite $\mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ as

$$\begin{aligned}
\mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log p(\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) - \log p(\boldsymbol{z})] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})). \qquad (2.12)
\end{aligned}$$

Equation (2.12) facilitates another interpretation of what is happening when the ELBO is maximized. The first term is commonly called the *negative reconstruction error*. The term *negative* is added because the loss function in ML and deep learning is typically defined to be minimized, meaning that the reconstruction error is $-\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})]$, which should be minimized. Nevertheless, increasing the first term in Equation (2.12) implies increasing the likelihood of the decoder producing an observation like $\boldsymbol{x}$ given samples from the encoder. Ideally, this is achieved by improving the approximate posterior and the decoder in conjunction [113]. The second term in Equation (2.12) can be understood as a regularizer, as it measures the discrepancy between the encoder and the prior belief on $\boldsymbol{Z}$. Decreasing this term "forces" the encoder to retain prior information over the latent variable, such that it learns a distribution instead of collapsing into deterministic points or Dirac delta functions [76]. Thus, in total, maximizing the ELBO boils down to maximizing the negative reconstruction error while minimizing the regularization term.

Minimization of the loss. One important question that remains unanswered is *how* the ELBO can be optimized. More specifically, how can the expectation in the ELBO be calculated? The technique that is typically used in VAEs is *Monte Carlo* (MC) estimation, which is a standard method for numerical integration. Thus, the two different expressions we have for the ELBO, as shown in Equations (2.11) and (2.12), can be estimated by

$$\hat{\mathcal{L}}^{I}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \coloneqq \frac{1}{L}\sum_{l=1}^{L}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}_l) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_l|\boldsymbol{x})], \qquad (2.13)$$

$$\hat{\mathcal{L}}^{II}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \coloneqq \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_l) - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})), \qquad (2.14)$$

respectively, where $\boldsymbol{z}_l$, $l \in \{1, \ldots, L\}$, are realizations of a random variable $\boldsymbol{Z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. Notice that the regularizer in Equation (2.14) has not been estimated by sampling, since it can be integrated analytically under certain restrictions. We return to this later. In continuation, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ are commonly parameterized by neural networks, even though, in theory, they may be calculated with any mathematical or statistical model. When using neural networks to estimate these densities, the derivatives with respect to the parameters need to be calculated, since deep learning models are typically trained using optimization techniques that require the derivatives. In general, the ELBO expressions in Equations (2.11) and (2.12) are differentiable with respect to $\boldsymbol{\theta}$, but not with respect to $\boldsymbol{\phi}$. This is true, because, for arbitrary functions $a_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}), b_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})$ and $b_{\boldsymbol{\phi}}(\boldsymbol{z})$,

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{b_{\boldsymbol{\phi}}(\boldsymbol{z})} \left[\log \frac{a_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{b_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})} \right] = \mathbb{E}_{b_{\boldsymbol{\phi}}(\boldsymbol{z})} \left[\nabla_{\boldsymbol{\theta}} \log \frac{a_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{b_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})} \right],$$

while

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{b_{\boldsymbol{\phi}}(\boldsymbol{z})} \left[\log \frac{a_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{b_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})} \right] \neq \mathbb{E}_{b_{\boldsymbol{\phi}}(\boldsymbol{z})} \left[\nabla_{\boldsymbol{\phi}} \log \frac{a_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{b_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})} \right].$$

The expectation and derivative cannot be interchanged in the second case since they both are performed with respect to the same parameters $\boldsymbol{\phi}$. The *reparameterization trick* [64, 65, 109] is used to rewrite the expectation in the ELBO in order to gain differentiability with respect to $\boldsymbol{\phi}$. Assuming that the latent variable is continuous, the reparameterization trick entails a relatively straightforward change of variables. Let $\boldsymbol{z}$ be a realization of a random variable $\boldsymbol{Z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and let $\boldsymbol{X} \sim p^*(\boldsymbol{x})$. $\boldsymbol{Z}$ may oftentimes be expressed as a transformation of another random variable, $\boldsymbol{\mathcal{E}} \sim p(\boldsymbol{\varepsilon})$, via a relation

$$\boldsymbol{Z} = \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{X}, \boldsymbol{\mathcal{E}}),$$

where $p(\boldsymbol{\varepsilon})$ is independent of $\boldsymbol{X}$ and $\boldsymbol{\phi}$. In addition, $\boldsymbol{g}_{\boldsymbol{\phi}}(\cdot, \cdot)$ is a vector-valued differentiable and invertible function with $\boldsymbol{\phi}$ as parameters [65]. We refer to Kingma and Welling [65] for an overview of approaches for choosing $\boldsymbol{g}_{\boldsymbol{\phi}}$ and $\boldsymbol{\mathcal{E}} \sim p(\boldsymbol{\varepsilon})$. Now, differentiation is possible with respect to the variational parameters. For instance, the derivative of the ELBO in Equation (2.11) can be calculated as

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \mathcal{L}^{VAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] \\
&= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\varepsilon})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon})) - \log q_{\boldsymbol{\phi}}(\boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon})|\boldsymbol{x})] \\
&= \nabla_{\boldsymbol{\phi}} \int [\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon})) - \log q_{\boldsymbol{\phi}}(\boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon})|\boldsymbol{x})] p(\boldsymbol{\varepsilon}) \mathrm{d}\boldsymbol{\varepsilon} \qquad (2.15) \\
&= \int \nabla_{\boldsymbol{\phi}} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon})) - \log q_{\boldsymbol{\phi}}(\boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon})|\boldsymbol{x})] p(\boldsymbol{\varepsilon}) \mathrm{d}\boldsymbol{\varepsilon} \qquad (2.16) \\
&= \mathbb{E}_{p(\boldsymbol{\varepsilon})}[\nabla_{\boldsymbol{\phi}} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon})) - \log q_{\boldsymbol{\phi}}(\boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon})|\boldsymbol{x})]].
\end{aligned}$$

The calculation of the gradient of the ELBO in Equation (2.12) is analogous and is therefore skipped. Notice that Equation (2.15) is equal to Equation (2.16) thanks to the fact that $\boldsymbol{\mathcal{E}} \sim p(\boldsymbol{\varepsilon})$ is independent of $\boldsymbol{\phi}$, such that both the density function and the variable of integration are considered as constants under differentiation with respect to the parameters. Thus, the reparameterization trick has essentially "externalized" the stochasticity in $\boldsymbol{Z}|\boldsymbol{X} = \boldsymbol{x}$ to $\boldsymbol{\mathcal{E}}$, such that it is deterministic given a realization of $\boldsymbol{\mathcal{E}}$, through the bijective function $\boldsymbol{g}_{\boldsymbol{\phi}}$. Hence, after reparameterization, MC estimates of the derivatives of the ELBO in Equations (2.11) and (2.12) can be expressed as

$$\begin{pmatrix} \frac{1}{L}\sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_l) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_l|\boldsymbol{x})] \\ \frac{1}{L}\sum_{l=1}^{L} \nabla_{\boldsymbol{\phi}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_l) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_l|\boldsymbol{x})] \end{pmatrix} = \begin{pmatrix} \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}^{I}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \\ \nabla_{\boldsymbol{\phi}} \hat{\mathcal{L}}^{I}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \end{pmatrix} = \nabla \hat{\mathcal{L}}^{I}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}),$$

and

$$\begin{pmatrix} \frac{1}{L}\sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_l) - \nabla_{\boldsymbol{\theta}} D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})) \\ \frac{1}{L}\sum_{l=1}^{L} \nabla_{\boldsymbol{\phi}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_l) - \nabla_{\boldsymbol{\phi}} D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})) \end{pmatrix} = \begin{pmatrix} \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}^{II}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \\ \nabla_{\boldsymbol{\phi}} \hat{\mathcal{L}}^{II}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \end{pmatrix} = \nabla \hat{\mathcal{L}}^{II}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}),$$

respectively, where $\boldsymbol{z}_l = \boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon}^l)$, and $\boldsymbol{\varepsilon}^l$, $l \in \{1, \ldots, L\}$, are realizations of a random variable $\boldsymbol{\mathcal{E}} \sim p(\boldsymbol{\varepsilon})$. Finally, Kingma and Welling [65] state that $\hat{\mathcal{L}}^{II}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ (Equation (2.14)), with its corresponding gradient, typically has less variance than $\hat{\mathcal{L}}^{I}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ (Equation (2.13)). Hence, $\hat{\mathcal{L}}^{II}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ is typically the preferred approximation of the ELBO for optimization.

Families of distributions. The only choices that remain to be made are *what* families of distributions should be used for $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z})$. First of all, the choice of distribution for the decoder is highly dependent on the application and the data that is available. For example, while working with images, like the frequently used MNIST database [19, 72], which are commonly represented as integer pixel values, $\boldsymbol{x} \in \{0, \ldots, 255\}^{1 \times p}$, where p is the number of pixels in each image, a possible decoder distribution could be the *categorical* distribution. It is represented by the density

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \text{Categorical}(\boldsymbol{x}; \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z})), \tag{2.17}$$

where the probability parameter is predicted by, e.g., a neural network $\boldsymbol{f}_{\boldsymbol{\theta}}$ with input $\boldsymbol{z}$ and a softmax output activation function [138]. For completeness, we define the categorical distribution in Appendix A. When working with tabular data, we might be inclined to use a different distribution for the decoder depending on the data type of each feature. For simplicity, we could use an individual categorical distribution for each categorical feature and a Gaussian distribution for the continuous features. We return to a similar idea when discussing *diffusion models* in Chapter 3. Beyond the decoder, the distributions corresponding to the latent variable need to be defined. In the quintessential VAE [65], the latent variable is assumed continuous, with densities

$$\begin{aligned} p(\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \mathbf{I}), \\ q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) &= \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})]). \end{aligned} \tag{2.18}$$

Here, the mean and standard deviation parameters in the encoder density are predicted by a neural network with input $\boldsymbol{x}$. Note that we use the notational convention $\text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})]$ for a diagonal matrix with the elements of $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})$ on its diagonal. Based on these assumptions, the KL term in $\hat{\mathcal{L}}^{II}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ can be integrated analytically, as swiftly remarked earlier. Let Λ be the latent space dimension and denote the elements of $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})$ and $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})$ by $\mu^j_{\boldsymbol{\phi},\boldsymbol{x}}$ and $(\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2$, for $j \in \{1, \ldots, \Lambda\}$, respectively. Then, the analytical solution is

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})) = -\frac{1}{2}\sum_{j=1}^{\Lambda}(1 + \log(\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2 - (\mu^j_{\boldsymbol{\phi},\boldsymbol{x}})^2 - (\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2).$$

The details are given in Appendix B, for completeness. Another consequence of the assumptions in Equation (2.18) is that $\boldsymbol{g}_{\boldsymbol{\phi}}$ in the reparameterization trick takes the shape

$$\boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon}) = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}) \odot \boldsymbol{\varepsilon} = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}) + \exp\left(\frac{1}{2}\log \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})\right) \odot \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon}$ is a realization of the random variable $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ denotes element-wise multiplication. Hence, we are able to transform realizations $\boldsymbol{\varepsilon}$ of a standard Gaussian random variable $\boldsymbol{\mathcal{E}}$ to realizations $\boldsymbol{z}$ of a random variable $\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})])$ via $\boldsymbol{g}_{\boldsymbol{\phi}}$, effectively transforming $\boldsymbol{Z}$ to a deterministic variable, when given a realization of $\boldsymbol{X} \sim p^*(\boldsymbol{x})$. This realization is necessary because the parameters $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})$ and $\text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})]$ depend on it. In closing, we have defined every piece of the puzzle necessary to apply VAEs in practice, except for the exact forms of the function approximators. More specifically, the functions that predict the parameters of the encoder and decoder densities, which in the case of the assumptions in Equation (2.18) (and our MNIST example) are $\boldsymbol{f}_{\boldsymbol{\theta}}, \boldsymbol{\mu}_{\boldsymbol{\phi}}$ and $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}$, can be defined as whatever model we like. However, as previously stated, neural networks are usually used because of their proven abilities for this task. Their architectures can be as simple or complex as desired, since we simply use their predictions in the encoder and decoder. Despite the fact that we have put little emphasis on explaining how these architectures should be defined thus far, the importance of their design should not be underestimated, as it is vital for the ability of a VAE to learn from data in practice. In fact, we highlight some different neural network architectures in Section 2.9, while discussing one particular VAE, specialized for tabular data.

Closing remarks. Recall from Section 2.7 that a model with the encoder-decoder architecture can be used for several tasks post training. The encoder can be used to produce a novel representation of the input data, which then can be used for other downstream tasks. For instance, the encoder of a VAE can be used to build *latent diffusion models* (LDMs) [110], a rather new application where a diffusion model, which we investigate in Chapter 3, works in the latent space that the encoder produces. Furthermore, the decoder can be used to generate synthetic data, which is a popular application for VAEs. Specifically, we can generate new samples from the inferred underlying joint density, $p(\boldsymbol{x}, \boldsymbol{z})$, by sampling in the latent space and decoding the samples. Besides, another important property of VAEs, which we have assumed implicitly in our discussion, is that the variational parameters, $\boldsymbol{\phi}$, in the encoder, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, are shared across all observations in $\mathcal{D}$. In contrast, more traditional methods in VI define distinct variational parameters for each $\boldsymbol{x} \in \mathcal{D}$, leading to much larger search spaces. This technique of sharing parameters is called *amortized variational inference*, and leads to more efficient implementations using common optimization techniques from deep learning like mini-batch stochastic gradient descent and backpropagation, facilitated by the reparameterization trick [64]. Thus, the VAE presents a possible solution to deeply-rooted problems in VI, marking its importance in practical applications.

2.9 The Tabular Variational Autoencoder

In this section, we outline a VAE-variant that is specifically designed to deal with heterogeneous tabular data. This is used as a reference model for our experiments in Chapter 5. Precisely, we discuss the *TVAE* [146]. In summary, the TVAE is a relatively simple VAE that is modified to better suit tabular data distribution modelling and generation compared to the standard VAE [65, 109]. This is enabled by three main innovations:

1. A so-called *mode-specific normalization* [146] scheme for the continuous features.
2. Specialized assumptions for the decoder, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, to accommodate the change in structure of the observations after pre-processing.
3. Special care when calculating the reconstruction term, $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})]$, in the ELBO in Equation (2.12).

In the following, we dive deeper into each of these devices. Notice that we alter and extend the theoretical introduction to these topics, especially for points 2 and 3, compared to the theory developed by Xu et al. [146]. This is done to adapt the theory to our notation, but also to provide a more detailed introduction, according to our understanding of it, that is hopefully more transparent and accessible than the original.

Assume a tabular dataset $\mathcal{D}$, which contains observations

$$\boldsymbol{x} \coloneqq \{(\boldsymbol{x}^{cont})^T, x^{cat_1}, \ldots, x^{cat_C}\}.$$

Each observation consists of values from N continuous features, comprising $\boldsymbol{x}^{cont} \in \mathbb{R}^{N \times 1}$ and C categorical features, x^{cat_j}, $j \in \{1, \ldots, C\}$, with K_j possible levels each. Thus, our notation is consistent, but we have simply split the features into two groups; one containing continuous random variables and the other containing categorical random variables. Notice that $p = N + C$ or $p + 1 = N + C$, depending on if the response, Y, is included as a categorical feature or not.

Mode-specific normalization. Xu et al. [146] observe that almost half of the continuous features they consider, distributed across 8 real-world datasets, empirically follow multimodal distributions. This frequently occurring non-Gaussian structure requires more specialized transformations than, e.g., a min-max transformation, which is commonly used in data modalities like images, to obtain an efficient data representation. Inspired by this observation, they design a mode-specific normalization method that is meant to better deal with continuous random variables that follow complicated multimodal distributions. Succinctly stated, they represent each continuous value in an observation $\boldsymbol{x} \in \mathcal{D}$ as a *one-hot encoded* (OHE) vector which indicates a mode the value pertains to, alongside a scalar that indicates its standardized value within the mode. More specifically, for each continuous feature $X^j \in \boldsymbol{X}^{cont}$, their method consists of the following three steps:

1. Let the density

$$p(x^j) = \sum_{k=1}^{M_j} \pi_k \mathcal{N}(x^j; \mu_k, \sigma_k^2),$$

represent a *Gaussian mixture model*, where π_k, $k \in \{1, \ldots, M_j\}$, are the *mixing coefficients*, constrained such that $\sum_{k=1}^{M_j} \pi_k = 1$. Assume that X^j follows this parametric model, where the parameters are unknown. Estimate the number of modes, M_j, as well as the other parameters, $\{\pi_k, \mu_k, \sigma_k^2\}$, $k \in \{1, \ldots, M_j\}$, of this mixture model with VI [8]. This is done using all the available realizations, x_i^j, $i \in \{1, \ldots, n\}$, of X^j in the dataset.

2. Compute the probability of each realization, x_i^j, coming from each of the modes, $\mu_1, \ldots, \mu_{M_j}$, by using the previously fitted Gaussian mixture. Precisely, the probability densities of x_i^j coming from each of the modes are

$$\rho^k(x_i^j) = \pi_k \mathcal{N}(x_i^j; \mu_k, \sigma_k^2), \quad k \in \{1, \ldots, M_j\},$$

which are used to compute the probabilities.

3. Sample a mode according to the computed probabilities in step 2. Use this mode to construct a OHE representation of each realization, x_i^j. Specifically, let $\boldsymbol{\beta}_i^j = \{\beta_i^{j,1}, \ldots, \beta_i^{j,M_j}\}$ be a OHE vector representing the sampled mode. Moreover, let α_i^j be a scalar representing the value x_i^j standardized within the mode. Thus, if the sampled mode for x_i^j is μ_k, then $\beta_i^{j,k} = 1$ and $\beta_i^{j,l} = 0$ for $l \neq k$, and $\alpha_i^j = \frac{x_i^j - \mu_k}{4\sigma_k}$.

For categorical features, Xu et al. [146] apply a standard OHE representation. Thus, the final representation of an arbitrary observation $\boldsymbol{x} \in \mathcal{D}$ after this pre-processing is

$$\boldsymbol{x} = \{\alpha^1, (\boldsymbol{\beta}^1)^T, \alpha^2, (\boldsymbol{\beta}^2)^T, \dots, \alpha^N, (\boldsymbol{\beta}^N)^T, (\boldsymbol{x}^1_{\text{OHE}})^T, \dots, (\boldsymbol{x}^C_{\text{OHE}})^T\}. \tag{2.19}$$

Notice that a pair $\{\alpha^j, \boldsymbol{\beta}^j\}$ denotes a representation of an element $j \in \{1, \dots, N\}$ of $\boldsymbol{x}^{cont}$ after mode-specific normalization. Furthermore, $\boldsymbol{x}^j_{\text{OHE}} = \{x^{j,1}_{\text{OHE}}, \dots, x^{j,K_j}_{\text{OHE}}\}$ denotes a OHE representation of x^{cat_j}, where each X^{cat_j} has K_j possible outcomes, for $j \in \{1, \dots, C\}$.

Encoder and decoder networks. As a consequence of the specialized pre-processing, the density $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ needs to facilitate use of observations in the shape shown in Equation (2.19). Before explaining the special structure Xu et al. [146] implement, we first note that the authors define the encoder similarly to the conventional VAE [65]. In fact, the encoder and the prior follow the assumptions of what we called the quintessential VAE in Section 2.8 (Equation (2.18)), defined as

$$\begin{aligned} p(\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{I}), \\ q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) &= \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})]), \end{aligned}$$

where the Gaussian parameters are predictions from a neural network. In order to define this neural network architecture, we follow the same idea for the structure as Xu et al. [146], but extend the notation to handle arbitrary linear hidden layers. More specifically, let $CD \coloneqq [cd_1, cd_2, \dots, cd_\xi]$ denote a list of hidden layer dimensions for the neural network in the encoder. These dimensions are appropriately called the *compression dimensions*, which explains why we call the list CD. Thus, ξ decides the number of hidden layers in the network. In addition, recall that we denote the latent space dimension by Λ. Then, the Gaussian parameters in the encoder can be calculated with a neural network architecture of the form

$$\begin{cases} \boldsymbol{h}^{(1)} &= \texttt{ReLU}(\texttt{FC}_{(N+\sum_{j=1}^{C} K_j) \to cd_1}(\boldsymbol{x})), \\ \boldsymbol{h}^{(2)} &= \texttt{ReLU}(\texttt{FC}_{cd_1 \to cd_2}(\boldsymbol{h}^{(1)})), \\ &\vdots \\ \boldsymbol{h}^{(\xi)} &= \texttt{ReLU}(\texttt{FC}_{cd_{\xi-1} \to cd_\xi}(\boldsymbol{h}^{(\xi-1)})), \\ \boldsymbol{\mu}_{\boldsymbol{\phi}} &= \texttt{FC}_{cd_\xi \to \Lambda}(\boldsymbol{h}^{(\xi)}), \\ \boldsymbol{\sigma}_{\boldsymbol{\phi}} &= \exp \tfrac{1}{2} \texttt{FC}_{cd_\xi \to \Lambda}(\boldsymbol{h}^{(\xi)}), \end{cases}$$

for each observation $\boldsymbol{x} \in \mathcal{D}$. Notice, in the last line, that exponentiation is performed after forward propagation to ensure that the standard deviation is positive.

The decoder is more involved. Xu et al. [146] make some key assumptions for the elements $\alpha^j_k, \boldsymbol{\beta}^j_k$, $j \in \{1, \dots, N\}$, and $\boldsymbol{x}^l_{\text{OHE},k}$, $l \in \{1, \dots, C\}$, of an arbitrary decoded point $\boldsymbol{x}_k$, given a latent realization $\boldsymbol{z}_i$. First, they assume that α^j_k, $j \in \{1, \dots, N\}$, are samples from Gaussian distributions with different means for each pair $\{i, j\}$, and different variances for each j, given $\boldsymbol{z}_i$ and $j \in \{1, \dots, N\}$. Mathematically, this means that α^j_k can be interpreted as a realization of $A^j_i \sim \mathcal{N}(\bar{\alpha}^j_i, \delta^j)$, where $\bar{\alpha}^j_i$ is predicted by a neural network and δ^j is a learnt parameter in the same network. Second, they assume that $\boldsymbol{\beta}^j_k$, $j \in \{1, \dots, N\}$, and $\boldsymbol{x}^l_{\text{OHE},k}$, $l \in \{1, \dots, C\}$, are samples from categorical distributions with different probability parameters for each pair $\{i, j\}$ and $\{i, l\}$, given $\boldsymbol{z}_i$, $j \in \{1, \dots, N\}$ and $l \in \{1, \dots, C\}$. Mathematically, this means that $\boldsymbol{\beta}^j_k$ can be interpreted as a realization of $\boldsymbol{B}^j_i \sim \text{Categorical}_{\text{OHE}}(\boldsymbol{\gamma}^j_i)$ and $\boldsymbol{x}^l_{\text{OHE},k}$ as a realization of $\boldsymbol{X}^l_{\text{OHE},i} \sim \text{Categorical}_{\text{OHE}}(\boldsymbol{\eta}^l_i)$, where $\boldsymbol{\gamma}^j_i$ and $\boldsymbol{\eta}^l_i$ are predicted by a neural network. Thus, these assumptions mean that the distribution of the random variable

$$(\boldsymbol{X}|\boldsymbol{Z}=\boldsymbol{z}_i)=\{A_i^1,(\boldsymbol{B}_i^1)^T,A_i^2,(\boldsymbol{B}_i^2)^T,\ldots,A_i^N,(\boldsymbol{B}_i^N)^T,(\boldsymbol{X}_{\text{OHE},i}^1)^T,\ldots,(\boldsymbol{X}_{\text{OHE},i}^C)^T\},$$

cannot easily be represented by a closed form density expression for $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. However, the probability of seeing a specific decoded observation

$$\boldsymbol{x}_k=\{\alpha_k^1,(\boldsymbol{\beta}_k^1)^T,\alpha_k^2,(\boldsymbol{\beta}_k^2)^T,\ldots,\alpha_k^N,(\boldsymbol{\beta}_k^N)^T,(\boldsymbol{x}_{\text{OHE},k}^1)^T,\ldots,(\boldsymbol{x}_{\text{OHE},k}^C)^T\},$$

as a result of decoding $\boldsymbol{z}_i$, can be calculated as

$$P(\boldsymbol{X}=\boldsymbol{x}_k|\boldsymbol{Z}=\boldsymbol{z}_i)=\prod_{j=1}^{N}P(A_i^j=\alpha_k^j)\prod_{j=1}^{N}P(\boldsymbol{B}_i^j=\boldsymbol{\beta}_k^j)\prod_{j=1}^{C}P(\boldsymbol{X}_{\text{OHE},i}^j=\boldsymbol{x}_{\text{OHE},k}^j),$$

as specified by Xu et al. [146]. In the following, we define an extended neural network architecture for calculating the mentioned quantities, following the authors' idea. In order to do this, let $DD\coloneqq[dd_1,dd_2,\ldots,dd_\kappa]$ be a list of hidden layer dimensions for the decoder in TVAE. These are called the *decompression dimensions*. Then, the neural network for calculating the parameters in the decoder has the architecture

$$\begin{cases}\boldsymbol{h}^{(1)}=\texttt{ReLU}(\texttt{FC}_{\Lambda\to dd_1}(\boldsymbol{z})),\\ \boldsymbol{h}^{(2)}=\texttt{ReLU}(\texttt{FC}_{dd_1\to dd_2}(\boldsymbol{h}^{(1)})),\\ \quad\vdots\\ \boldsymbol{h}^{(\kappa)}=\texttt{ReLU}(\texttt{FC}_{dd_{\kappa-1}\to dd_\kappa}(\boldsymbol{h}^{(\kappa-1)})),\\ \bar{\alpha}^j=\texttt{tanh}(\texttt{FC}_{dd_\kappa\to 1}(\boldsymbol{h}^{(\kappa)})), & 1\le j\le N,\\ \boldsymbol{\gamma}^j=\texttt{softmax}(\texttt{FC}_{dd_\kappa\to M_j}(\boldsymbol{h}^{(\kappa)})), & 1\le j\le N,\\ \boldsymbol{\eta}^j=\texttt{softmax}(\texttt{FC}_{dd_\kappa\to K_j}(\boldsymbol{h}^{(\kappa)})), & 1\le j\le C,\end{cases}\tag{2.20}$$

for one latent realization $\boldsymbol{z}$. Hence, to be clear, a latent point $\boldsymbol{z}$ can be decoded to yield a sample from $\boldsymbol{X}|\boldsymbol{Z}=\boldsymbol{z}\sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, after fitting the neural network given by the architecture above, by sampling the continuous feature values in mode-specific representations from distributions $\mathcal{N}(\bar{\alpha}^j,\delta^j)$ and $\text{Categorical}_{\text{OHE}}(\boldsymbol{\gamma}^j)$, for $j\in\{1,\ldots,N\}$. In addition, the categorical feature values in the decoded realization are sampled from $\text{Categorical}_{\text{OHE}}(\boldsymbol{\eta}^j)$, for $j\in\{1,\ldots,C\}$.

Modified calculation of ELBO. The ELBO in the TVAE follows the same expression as $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}^{VAE}$ in Equation (2.12). However, when calculating the reconstruction term, $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})]$, some extra precaution needs to be taken because of the decoder distribution. Recall that we typically approximate this term with a Monte Carlo (MC) estimator, such that

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})]\approx\frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_l),$$

where $\boldsymbol{z}_l$, $l\in\{1,\ldots,L\}$, are realizations of a random variable $\boldsymbol{Z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. This technique is also utilized in the TVAE. However, the calculation of the terms $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_l)$, $l\in\{1,\ldots,L\}$, depends on each distinct component of an arbitrary decoded sample $\boldsymbol{x}$. For the components α^j, $j\in\{1,\ldots,N\}$, we simply calculate $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_l)$ directly using the Gaussian density, $\mathcal{N}(\alpha^j;\bar{\alpha}_l^j,\delta^j)$. This implicitly teaches the TVAE to reconstruct the continuous values in the pre-processed input with a mean squared error loss. For the OHE components, $\boldsymbol{\beta}^j$, $j\in\{1,\ldots,N\}$, and $\boldsymbol{x}_{\text{OHE}}^j$, $j\in\{1,\ldots,C\}$, such a reconstruction is taught by using a *cross-entropy loss* between the reconstructed elements and the true elements. In the

following discussion, let the functions $\boldsymbol{f}_{\boldsymbol{\theta}}^{j} = \{f_{\boldsymbol{\theta}}^{j,1}, \ldots, f_{\boldsymbol{\theta}}^{j,M_j}\}$ and $\boldsymbol{g}_{\boldsymbol{\theta}}^{j} = \{g_{\boldsymbol{\theta}}^{j,1}, \ldots, g_{\boldsymbol{\theta}}^{j,K_j}\}$ represent the neural network output functions for $\boldsymbol{\gamma}_l^j$, $j \in \{1, \ldots, N\}$, and $\boldsymbol{\eta}_l^j$, $j \in \{1, \ldots, C\}$, by the neural network architecture in Equation (2.20), respectively. In this way, when feeding these functions with $\boldsymbol{z}_l$, they provide the necessary predictions for the probability parameters. For $\boldsymbol{\beta}^j$, $j \in \{1, \ldots, N\}$, the (negative) cross-entropy is defined as

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{\beta}^j | \boldsymbol{z}_l) = \sum_{k=1}^{M_j} \beta^{j,k} \log f_{\boldsymbol{\theta}}^{j,k}(\boldsymbol{z}_l), \quad j \in \{1, \ldots, N\}, \tag{2.21}$$

while for $\boldsymbol{x}_{\text{OHE}}^j$, $j \in \{1, \ldots, C\}$, the (negative) cross-entropy is defined as

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\text{OHE}}^j | \boldsymbol{z}_l) = \sum_{k=1}^{K_j} x_{\text{OHE}}^{j,k} \log g_{\boldsymbol{\theta}}^{j,k}(\boldsymbol{z}_l), \quad j \in \{1, \ldots, C\}. \tag{2.22}$$

These definitions follow directly from the PMFs of the categorical distributions $\boldsymbol{B}_l^j \sim \text{Categorical}_{\text{OHE}}(\boldsymbol{\gamma}_l^j)$, $j \in \{1, \ldots, N\}$, and $\boldsymbol{X}_{\text{OHE},l}^j \sim \text{Categorical}_{\text{OHE}}(\boldsymbol{\eta}_l^j)$, $j \in \{1, \ldots, C\}$. For instance, in the first case, the PMF is $p_{\boldsymbol{\theta}}(\boldsymbol{\beta}^j | \boldsymbol{z}_l) = \prod_{k=1}^{M_j} (f_{\boldsymbol{\theta}}(\boldsymbol{z}_l)^{j,k})^{\beta^{j,k}}$, as given in Equation (A.2) in Appendix A. The second case is analogous. Notice that the *negations* of Equations (2.21) and (2.22) are cross-entropy *losses*, since we are interested in maximizing the reconstruction term, $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})]$. However, this is just a question of terminology and making sure the signs are correct when either minimizing or maximizing the objective function. Conclusively, we have to beware of how the reconstruction term is calculated for each element in vectors in $\mathbb{R}^{(N+\sum_{j=1}^{C} K_j)\times 1}$, i.e. the space of pre-processed original data. As a final note, because of the assumptions on the latent prior and the encoder in the TVAE, the KL term in the ELBO, $D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}))$, is analytically integrated in the same way as explained in Section 2.8.

2.10 Evaluation Metrics

In order to evaluate the performance of any ML model, it is necessary to define some metrics. Inspired by Xu et al. [146] and Kotelnikov et al. [67], our main evaluation measure for generative models is *ML efficacy*. It is also referred to as the "train on synthetic, test on real (TSTR)" framework [61]. In short, this measure evaluates a classification or regression model on *real* test data, after fitting the model on *synthetic* data. For all practical purposes, a high-performing model trained on synthetic data implies a high-performing synthesizer. A more detailed description of this methodology is given later in this section.

However, to paint a more complete picture of a model's performance, we prefer to perform initial qualitative evaluation as well. Visualizations can be used to make qualitative comparisons between different models relatively simple, facilitating quick first impressions of their behaviour. In the following, we describe the qualitative metrics and methods we use, before describing the main quantitative metrics we use.

2.10.1 Qualitative Evaluation

Our initial procedure after producing results is to perform qualitative evaluation. For a generative model, we are especially interested in acquiring a first impression concerning the synthesizing performance of the model, because this quality is important not only for generating new data points, but also for generating counterfactuals. Precisely, this is the case for the specific on-manifold methods we discuss in Chapter 4. We qualitatively evaluate generative performance by visualizing a generated dataset, of identical size as

its corresponding real dataset, in conjunction with the real dataset. First, the marginal distributions of all features in both datasets are plotted. An advantage of such plots is that we can quickly spot if the synthetic data is very different from the real data. However, a disadvantage is that we cannot evaluate interactions between the features in the datasets by simply inspecting the marginal densities. Thus, in order to investigate connections between the features, we calculate what we call the *correlation matrix*. This is a slight abuse of language, since we cannot calculate correlation between non-numerical features, or numerical-non-numerical pairs of features, but we use the term for simplicity. To be clear, we use the term *numerical* to encompass continuous and discrete integer features, while the term *non-numerical* refers to all other categorical features. We return to these terms later. As its name suggests, the correlation matrix is a matrix that contains information about the strength of association between the features in a dataset. Precisely, the *Pearson correlation* coefficient [33] is calculated pairwise between the numerical features. This measures the strength of the linear relationship between the numerical features in each pair. Moreover, the strength of association between pairs of numerical and non-numerical features are measured by the *correlation ratio* [33]. This is a measure of the dispersion across the entire dataset in comparison to the dispersion within each category, and can be used to measure non-linear relationships. Finally, the interactions between pairs of non-numerical features are measured by *Theil's U* statistic [135]. Without going into details, this statistic, also known as the *uncertainty coefficient*, is an asymmetric measure of the association between two non-numerical features. Essentially, it measures the uncertainty of one feature explained by the other. Notice that the Pearson correlation and correlation ratio are symmetric measures. Additionally, all these measures are constrained to $[0, 1]$. After calculating a correlation matrix for a dataset, we visualize it as a heatmap, such that we can qualitatively inspect the magnitudes of the values.

2.10.2 Quantitative Evaluation

ML efficacy is a commonly used evaluation measure of synthetic data generators [146]. The idea behind this measure is that an ideal synthetic dataset should be able to yield (at least) as good results in applications as the corresponding real dataset. Specifically, we train a supervised model separately on both synthetic training data and the corresponding real training data. Then, both the trained models are evaluated on the real test set. If the performances of the two models are comparable, it is implied that the synthetic dataset is of high quality. Ultimately, this reflects on the quality of the generative model for synthesizing data. This method of evaluation is especially suitable for evaluating generative models, as their synthetic data often are used in similar downstream tasks, for example extending small datasets to train better ML models or constructing synthetic datasets that can be shared publicly, for facilitating ML model development, when the real datasets contain sensitive information. In such cases, we want the synthetic data to yield (at least) as good performance as the real data, which, in these examples, unfortunately is not sufficiently large or cannot be shared for protection of privacy.

Binary Classification Metrics

After generating synthetic datasets, we need to make some choices in order to use ML efficacy to evaluate the generators. Specifically, we need to decide which supervised models we want to train on our datasets, in addition to how we want to evaluate these surrogate models. Recall that we only consider binary classification datasets, meaning that only binary classifiers are interesting to train. As a consequence of this, we select three frequently used metrics for evaluating binary classifiers: *accuracy*, F_β*-score* and AUC. These choices are succinctly discussed in the following.

Confusion matrix. In order to facilitate a discussion concerning these metrics, we consider a special type of contingency table called a *confusion matrix*. Figure 2.5 shows a typical confusion matrix, which is used to visualize the performance of a binary classifier. Notice the definitions of the *true negatives* (TN), *false negatives* (FN), *false positives* (FP) and *true positives* (TP), which are used in the following.

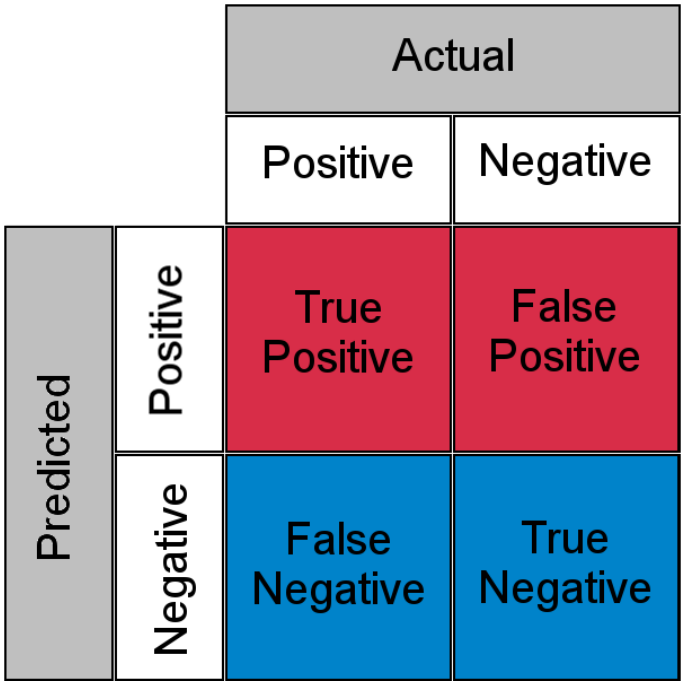


Figure 2.5: Illustration of a confusion matrix for a binary classifier. It indicates what groups of classified test observations are referred to as *true negatives* (TN), *false negatives* (FN), *false positives* (FP) and *true positives* (TP). The image is borrowed from ThresholdTom [136], and is licensed under the Creative Commons Attribution-Share Alike 4.0 International [17] license.

Accuracy. The first and most intuitive metric is the classification accuracy. For completeness, the accuracy is defined as

$$\text{Accuracy} := \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \in [0, 1].$$

An advantage of this metric is that it is simple and intuitive, yielding a quick overview of the share of correct classifications on a test dataset. However, in many cases, evaluation of the accuracy of a classifier is not sufficient, because it only takes into account the *number* of correct predictions that are made. In other words, the main disadvantage of this metric is that it does not take into account the *types* of correct or incorrect classifications that are produced by the model. In most applications, we also care about which of the categories the classifier is able to most accurately predict. The F_β-score is one example of a metric that takes this characteristic into account.

***F_β*-score.** The F_β-score is a commonly used metric for evaluating a classification algorithm in more detail than accuracy. Before stating the relatively simple formula for calculating this score, we define the quantities *precision* and *recall*,

$$\text{Precision} := \frac{\text{TP}}{\text{TP} + \text{FP}} \in [0, 1],$$

$$\text{Recall} := \frac{\text{TP}}{\text{TP} + \text{FN}} \in [0, 1].$$

Intuitively, "precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples" [119]. This means that the precision is equal to one when all the observations that the classifier predicts as true actually are true in the dataset, yielding zero FP. Furthermore, the recall is equal to one when all the true observations in the dataset are correctly retrieved by the model, yielding zero FN. Note that precision is often known as *positive predictive value* and recall is often known as *sensitivity* or *true positive rate*. Then, the F_β-score is defined as

$$F_\beta \coloneqq (\beta^2 + 1)\frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}} \in [0, 1], \quad \beta \geq 0. \tag{2.23}$$

Equation (2.23) was introduced by Chinchor [14], adapted from an earlier, slightly different definition by Van Rijsbergen [139]. The coefficient β sets the relative importance of precision and recall when calculating the score. If $\beta < 1$, the score gives more weight to the precision, while if $\beta > 1$, the score gives more weight to the recall. In practice, precision should be given most weight during evaluation if minimizing the number of FP should be prioritized, while recall should be given most weight during evaluation if minimizing the number of FN is more important. Of course, either precision or recall are more reasonable metrics to use for evaluation in cases where it is vital to avoid either type of misclassification. The choice of metric should not be underestimated for gaining the desired performance in specific applications.

For simplicity, in cases where we do not have a clear preference to minimizing either FP or FN, many researchers choose to give the two terms equal weight, by setting $\beta = 1$, which yields the frequently used F_1*-score*,

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \in [0, 1]. \tag{2.24}$$

In a multiclass setting, the F_1-score can be calculated for each class. For example, in binary classification, we can calculate one F_1-score for class 1 and one F_1-score for class 0, where the latter calculation is performed by simply switching the roles of the binary indicators, treating 0 as positive instead of negative. Essentially, this switches the roles of TN and TP, as well as the roles of FP and FN, in Figure 2.5. We could choose to evaluate either of these F_1-scores individually, depending on which of the classes we want to focus on. However, in general cases where we don't have a preference, it is common to aggregate the F_1-score of each class into one combined score. There are several ways of doing this. We choose to simply use the average of the two scores as an overall F_1-score in our experiments with binary classifiers. We refer to this as the *macro* F_1*-score*. Other alternatives are to calculate an average weighted by the support of each class, or to calculate a global F_1-score by summing all the FN, FP and TP, respectively, across all classes, before plugging into Equation (2.24).

In closing, an advantage of the F_β-score is that it is calculated from two quantities that take the types of misclassifications that a classifier makes into account. However, one of the main disadvantages of this metric is that it needs to be optimized for a certain choice of *discrimination threshold*. In binary classification, this threshold is the probability or score value at which we choose the positive class over the negative class. In general, this level is set to 0.5, but it can be adjusted based on the relative importance of TN, FN, FP and TP in each application. In order to overcome this issue, we choose to evaluate the binary classifiers using AUC as well.

AUC. The Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve is another commonly used metric for evaluating binary classifiers. The ROC curve is a plot that shows the performance of a binary classifier while changing the

discrimination threshold in $[0, 1]$. In order to understand the ROC curve, we define the *false positive rate* (FPR),

$$\text{FPR} := \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

In addition, remember that the recall is also known as the *true positive rate* (TPR), which is the second quantity that is used to plot a ROC curve. More specifically, a ROC curve is simply a plot of FPR against TPR, for differing levels of the discrimination threshold, ranging continuously from 0 to 1. A few examples of such curves are shown in Figure 2.6. An ideal, but unrealistic predictor should simply be a point in the upper left corner, representing $FPR = 0$ (no false positives) and $TPR = 1$ (no false negatives) for all discrimination thresholds. The AUC is, as the name suggests, the area under the ROC

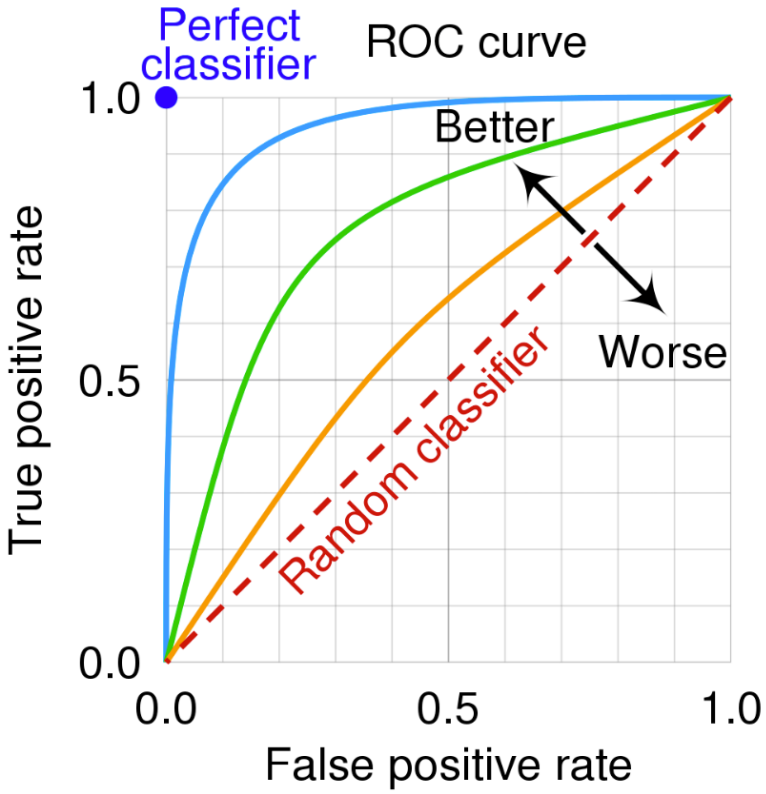


Figure 2.6: Examples of different ROC curves. The image is borrowed from cmglee & MartinThoma [15], and is licensed under the Creative Commons Attribution-Share Alike 4.0 International [17] license.

curve. A random binary classifier yields a linear ROC curve, as shown in Figure 2.6, meaning it has an AUC of 0.5, because both TPR and FPR increase linearly with a decrease in the decision threshold. An ideal classifier has an area close to 1, because this signals that the curve "hugs" the upper left corner, meaning that TPR is large, while FPR is small. Thus, instead of relying on visual inspection of ROC curves, we can equivalently rely on AUC values during evaluation. We do not dive deeper into the underlying theory of ROC, but simply assume that larger AUC is better. The interested reader is referred to, e.g., Fawcett [32] for a detailed introduction to ROC curves and their pitfalls.

ML Efficacy Protocols

After discussing the three metrics we chose for evaluating binary classifiers, we comment on two different ML efficacy protocols that appear in the literature. More specifically,

Kotelnikov et al. [67] highlight two different techniques, which we call the *average protocol* and the *optimal protocol*.

Average protocol. This is the most commonly used protocol [146], in which a set of different binary classifiers are trained on the real and synthetic datasets. Frequently selected classifiers are logistic regression, decision trees, GBDTs and neural networks. Post training, each model is evaluated, using the metrics of choice, on predictions on the test set, before each of the metrics are averaged across all the classifiers. Kotelnikov et al. [67] observe that this protocol often can be misleading, because it relies on simple and typically sub-optimal classifiers. They argue that this is not a realistic situation in any application, since a developer or researcher would not be inclined to use classifiers that are clearly not optimal.

Optimal protocol. Kotelnikov et al. [67] develop a protocol where they simply train a *CatBoost* [104] model on the real and synthetic datasets, stating that this model "is the leading GBDT implementation providing state-of-the-art performance on tabular tasks" [67]. They argue that this technique is more reliable and useful, since we always strive to use high-performing classifiers in practice, instead of averaging over mediocre predictors. Specifically, they argue that "this evaluation protocol demonstrates the practical value of synthetic data more reliably since in most real scenarios practitioners are not interested in using weak and suboptimal classifiers/regressors" [67]. Moreover, the authors observe that their selected ML efficacy metrics have better values for the models trained on the synthetic data compared to the models trained on the real data, while following the average protocol. From this, they conclude that the average protocol gives a false impression that the synthetic data is more valuable than the real data, which is never the case (unless, e.g., the real dataset is extremely small, denying it of any real statistical power). While using the optimal protocol, the authors do not observe the same phenomenon, which is another reason why they conclude that it is more realistic.

Chapter 3

Diffusion Models

This chapter is dedicated to developing an accessible exposition of *diffusion models*, especially for applications with tabular data. This is one of our main contributions in this thesis. Diffusion models were first introduced by Sohl-Dickstein et al. [124], inspired by non-equilibrium thermodynamics and statistical physics. A physical diffusion process describes progressive destruction of information. Inspired by this, generative diffusion models steadily destroy information in the input data through a so-called *forward* diffusion process, which commonly has a predefined structure. Subsequently, a so-called *reverse* diffusion process is learnt in order to restore the information in the data. After estimating the model parameters, it is possible to sample synthetic data points, calculate both unconditional and conditional probabilities, as well as compute likelihoods, in tractable ways. Several high-performing generative models have been trained with this technique as a backbone, e.g., [20, 106, 130].

Recall that we restrict our study to tabular data, which rarely contains homogeneous sets of data types. As a result, our aim is to develop models that can handle all possible data types in any tabular dataset. Because continuous and categorical data types are different in nature, they require different assumptions when modelling. Consequently, we discuss diffusion models that rely on different assumptions, depending on the data type they are intended to be applied to. After discussing them individually, they are joined together to build a model suitable for our data modality. This idea was actually recently implemented by Kotelnikov et al. [67], which is a great inspiration for our work.

The rest of the chapter is organized as follows. First, Section 3.1 introduces an extension of VAEs; *hierarchical* variational autoencoders (HVAEs). In the author's humble opinion, the simplest way of approaching diffusion models theoretically is to first introduce HVAEs, before extending their theory seamlessly into diffusion models. This is the reason why we chose to add a section on HVAEs in this chapter. The subsequent section, Section 3.2, gives a brief and easily digestible introduction to diffusion models. Next, Sections 3.3 and 3.4 introduce two important variations of diffusion models for our application; *Gaussian* diffusion models and *Multinomial* diffusion models. Furthermore, Section 3.5 exhibits algorithms for sampling from these two models. Finally, the main generative model we investigate in this thesis, which we call *Tabular diffusion*, is discussed in Section 3.6. This model combines the previously studied diffusion models into one model, compatible with any tabular dataset.

3.1 Hierarchical Variational Autoencoders

In the initial formulation of VAEs in Section 2.8, the number of latent variable *levels* is one. To be more specific, recall that in the original formulation we suppose that the generative model can be described by a joint density $p(\boldsymbol{x}, \boldsymbol{z})$, which relates an observable random

variable $\boldsymbol{X}$ and a latent random variable $\boldsymbol{Z}$. Thus, each observation, $\boldsymbol{x}$, is allowed to depend on one latent variable realization, $\boldsymbol{z}$. However, the restriction that each observation depends on only one level of latent variables need not be enforced, as we can introduce a deeper hierarchy of latent variables that depend on each other, using similar ideas as in the original VAE. Any Bayesian [5] would recognize this idea from *Bayesian hierarchical modelling*, where each model parameter is allowed to depend on other hyperparameters in a nested fashion through their hyperprior distributions [38].

Taking a step back from the mathematics, the following philosophical exercise demonstrates why several layers of latent variables might be reasonable to assume. In the classic VAE, we implement the idea that observable data depends on unobservable variables $\boldsymbol{Z}$ in a latent space. Luo [76] uses Plato's allegory of the cave to explain this concept. The prisoners of the cave only observe the two-dimensional projection of the three-dimensional world outside. In this situation, the projections are the observables, while the world outside the cave constitutes the latent variables in three dimensions. This leads us to the idea of a VAE with latent variables in one latent space. But, this idea might be reasonable to extend to more complex relationships. Imagine that the three-dimensional objects we observe in reality are projections of other latent variables of differing dimensionalities. Such variables could represent abstract quantities like shapes, colors, temperatures or other aspects of the objects we observe. With this idea in mind, the people of the cave could imagine a generative model that consists of (at least) two layers of latent variables: the first layer consisting of the three-dimensional objects outside the cave, which depends on other abstract latents in a second layer. These ideas can be extended an arbitrary amount of times, each extension increasing the hierarchical depth of the set of latent variables. As a side note, the allegory of the cave is lacking when it comes to illustrating the relationship between the dimensions of the latent variables and the observables in a VAE, since a bottleneck is used in order to learn latent representations of smaller dimension than the data. Because of this, recall that we introduced a different philosophical exercise in Section 2.8, to highlight a VAE's capabilities in representation learning, should it be desired.

More rigorously, let the set $\{\boldsymbol{Z}_1, \boldsymbol{Z}_2, \dots, \boldsymbol{Z}_T\}$ denote a hierarchy of latent variables. Then, the joint distribution with the observables $\boldsymbol{X}$ can be represented by the density $p(\boldsymbol{x}, \boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_T)$, analogously to in VAEs. In this notation, $\boldsymbol{z}_1$ represents a realization of the latent variable level closest to $\boldsymbol{X}$, while $\boldsymbol{z}_T$ represents a realization of the deepest or highest latent variable level, furthest from $\boldsymbol{X}$. In the rest of this thesis, we use the terms *lower*, *closer* and *shallower*, as well as *higher*, *further* and *deeper*, interchangeably. Notice that the introduction of more latent variables does not change the objective; recall that the goal of a generative latent variable model is to estimate the true data distribution by marginalization of the latent variables. For a likelihood-based model, this is achieved by maximizing

$$p(\boldsymbol{\theta}; \mathcal{D}) \coloneqq \prod_{\boldsymbol{x} \in \mathcal{D}} p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \prod_{\boldsymbol{x} \in \mathcal{D}} \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) \mathrm{d}\boldsymbol{z}_1 \cdots \mathrm{d}\boldsymbol{z}_T, \tag{3.1}$$

with respect to $\boldsymbol{\theta}$, where we let $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ denote the likelihood of one sample $\boldsymbol{x} \in \mathcal{D}$. Without loss of generality, we focus on an arbitrary observation $\boldsymbol{x} \in \mathcal{D}$ in the following. Similarly to in Equation (2.10), the joint distribution may be rewritten as a product of conditional distributions by using the chain rule for random variables, which in general reads

$$P(X_1, \dots, X_n) = P(X_1) \prod_{t=2}^{n} P(X_t | X_1, \dots, X_{t-1}), \tag{3.2}$$

for a collection of unordered random variables $\{X_1, \dots, X_n\}$. In our case, the joint distribution may be factorized as

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) = p(\boldsymbol{z}_T) \prod_{t=2}^{T} p_{\boldsymbol{\theta}_{t-1}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \dots, \boldsymbol{z}_T) p_{\boldsymbol{\theta}_0}(\boldsymbol{x}|\boldsymbol{z}_1, \dots, \boldsymbol{z}_T), \tag{3.3}$$

where we have added an additional index on $\boldsymbol{\theta}$ to explicitly highlight that the parameters in each of the densities typically are different. The reason why $p(\boldsymbol{z}_T)$ has no parameters is that it can be interpreted as a prior distribution, similar to the role of $p(\boldsymbol{z})$ in VAEs. From Equation (3.3) it is clear that each level in the hierarchy of latent variables is allowed to condition on all higher latents relative to itself; as $t \in \{T, T-1, \dots, 2\}$ decreases, each conditional distribution depends on an increasingly larger set of latent variables. This structure can quickly lead to very large and complex models, with many parameters. As discussed in Section 2.8, the marginal likelihood in Equation (3.1) is intractable in general. In any case, one possible solution to this problem is to approximate it via maximization of the ELBO. When parameterizing this latent variable model with neural networks, it is called a *hierarchical variational autoencoder* (HVAE) [76]. In its general formulation, the HVAE is a very flexible model, which can yield fruitful results, as well as some difficulties [66, 126]. In this work, we are particularly interested in a special case of the HVAE, where the hierarchy is assumed to follow a *Markov chain*. This assumption restricts the model, effectively limiting some of its flexibility, but it is a stepping stone in the derivation of diffusion models. We refer to this model as a *Markovian hierarchical variational autoencoder* (MHVAE) [76].

Decoding process. The Markov assumptions in MHVAEs yield a particularly simple *generative* (or *decoding*) *process*; each transition to a latent $\boldsymbol{Z}_t$ only conditions on the neighbouring higher latent $\boldsymbol{Z}_{t+1}$. Hence, Equation (3.3) can be simplified to

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) = p(\boldsymbol{z}_T) \prod_{t=2}^{T} p_{\boldsymbol{\theta}_{t-1}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) p_{\boldsymbol{\theta}_0}(\boldsymbol{x}|\boldsymbol{z}_1). \tag{3.4}$$

Generation from this model follows the process

$$\begin{aligned}
\text{Sample } \boldsymbol{z}_T, &\text{ where } \boldsymbol{Z}_T \sim p(\boldsymbol{z}_T), \\
\text{Sample } \boldsymbol{z}_{T-1}, &\text{ where } \boldsymbol{Z}_{T-1}|\boldsymbol{Z}_T = \boldsymbol{z}_T \sim p_{\boldsymbol{\theta}_{T-1}}(\boldsymbol{z}_{T-1}|\boldsymbol{z}_T), \\
&\vdots \\
\text{Sample } \boldsymbol{z}_1, &\text{ where } \boldsymbol{Z}_1|\boldsymbol{Z}_2 = \boldsymbol{z}_2 \sim p_{\boldsymbol{\theta}_1}(\boldsymbol{z}_1|\boldsymbol{z}_2), \\
\text{Sample } \boldsymbol{x}, &\text{ where } \boldsymbol{X}|\boldsymbol{Z}_1 = \boldsymbol{z}_1 \sim p_{\boldsymbol{\theta}_0}(\boldsymbol{x}|\boldsymbol{z}_1),
\end{aligned}$$

which is reminiscent of the generative process we commenced the discussion in Section 2.8 with, after nesting T levels of latent variables. Figure 3.1 further illustrates these ideas, demonstrating a chain of realizations of the random variables. Notice that the two leftmost nodes in the figure, representing the observables $\boldsymbol{X}$ and the most shallow latent $\boldsymbol{Z}_1$, gives a VAE when seen in isolation from all the other latents, with the encoder dictating the transformation from $\boldsymbol{X}$ to $\boldsymbol{Z}_1$ and the decoder dictating the transformation from $\boldsymbol{Z}_1$ to $\boldsymbol{X}$. Thus, another way of understanding a MHVAE is as a stack of VAEs, wherein each VAE the leftmost latent in the pair is treated as an observable, given the neighbouring higher VAE.

Encoding process. In order to complete the analogy to shallow VAEs, we should mention the *encoding process* as well. Recall from Section 2.8 that we introduce an approximate posterior, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, in order to develop a variational lower bound for the VAE. For HVAEs, the same idea is used. In this case, we are interested in inferring the distribution of the closest latent variable $\boldsymbol{Z}_1$ given a realization of $\boldsymbol{X}$, as well as the latent variable $\boldsymbol{Z}_t$

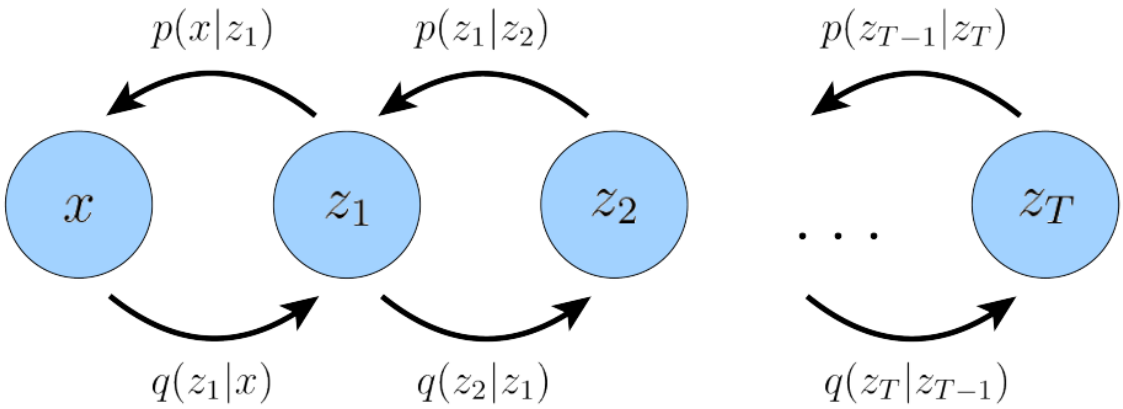


Figure 3.1: Example of a hierarchical latent structure with Markovian assumptions. The generative process moves from right to left, while the encoding process moves from left to right. The parameters of the encoders and decoders are not shown explicitly. The image is borrowed with permission from Luo [76].

conditioned on $\boldsymbol{Z}_{t-1}$ and the rest of the lower latents relative to itself, for $t \in \{2, \dots, T\}$. Mathematically, using the chain rule once again, the encoding process can be written as

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T|\boldsymbol{x}) = q_{\boldsymbol{\phi}_1}(\boldsymbol{z}_1|\boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}_t}(\boldsymbol{z}_t|\boldsymbol{x}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_{t-1}), \tag{3.5}$$

where we condition on an observation $\boldsymbol{x} \in \mathcal{D}$. Notice the complicated structure of this process when Markov assumptions are not made. After making such assumptions, Equation (3.5) can be expressed as

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T|\boldsymbol{x}) = q_{\boldsymbol{\phi}_1}(\boldsymbol{z}_1|\boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}_t}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}). \tag{3.6}$$

This process is also illustrated in Figure 3.1.

Evidence lower bound. Having introduced both the encoding and decoding processes, the marginal log-likelihood of an arbitrary observation $\boldsymbol{x} \in \mathcal{D}$ can be written as

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \int q_{\boldsymbol{\phi}}(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T|\boldsymbol{x}) \mathrm{d}\boldsymbol{z}_1 \cdots \mathrm{d}\boldsymbol{z}_T \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{z}_1 \cdots \mathrm{d}\boldsymbol{z}_T \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})} \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right]}_{\mathcal{L}^{HVAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right]}_{D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}) \,\|\, p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}))},
\end{aligned} \tag{3.7}$$

where we use the notational convention $\boldsymbol{z}_{1:T} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_T\}$. In addition, recall that the notation $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})]$ is a shorthand, abuse of notation that represents

$$\mathbb{E}_{\boldsymbol{Z}_{1:T}|\boldsymbol{X}=\boldsymbol{x}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})] = \int q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{z}_1 \cdots \mathrm{d}\boldsymbol{z}_T.$$

As expected, this calculation reveals that the ELBO, now with a different expression, is a lower bound for the marginal log-likelihood. In this case, the bound is tight when the entire approximate encoding process is equal to the true posterior, as expressed by $D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}))$. Moreover, analogously to in shallow VAEs, a minimization of the KL term with respect to $\boldsymbol{\phi}$ necessarily follows from maximization of the ELBO, and vice versa, since the marginal log-likelihood is constant with respect to the variational parameters. Issues concerning tightness of the bound and inference suboptimality can be discussed for HVAEs as well, similar to in VAEs. Finally, similar analyses of $\mathcal{L}^{HVAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ can be performed in this case, including expanding the expression for a better interpretation of what terms are involved during maximization. Notice that $\mathcal{L}^{HVAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ can be simplified when working with MHVAEs, by inserting Equations (3.4) and (3.6), which yields

$$\begin{aligned}
\mathcal{L}^{MHVAE}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_T)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)\prod_{t=2}^{T} p_{\boldsymbol{\theta}_{t-1}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)\, p_{\boldsymbol{\theta}_0}(\boldsymbol{x}|\boldsymbol{z}_1)}{q_{\boldsymbol{\phi}_1}(\boldsymbol{z}_1|\boldsymbol{x})\prod_{t=2}^{T} q_{\boldsymbol{\phi}_t}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right].
\end{aligned}$$

To conclude this section, we have seen how VAEs can be extended to models with arbitrarily deep latent variable structures, under very similar formulations. By imposing certain restrictions, like for instance Markovian, we can build tractable models that can be trained in similar fashion to VAEs, commonly through parameterization via neural networks.

3.2 A Brief Introduction to Diffusion Models

As stated, we believe the most straightforward way of explaining diffusion models is to use previously acquired knowledge on MHVAEs to segue into their quite vast body of literature. In fact, diffusion models can be derived as a special case of MHVAEs by imposing three key restrictions [76]. These simplifying assumptions are that

1. the latent variables, $\boldsymbol{Z}_t$, $t \in \{1, \ldots, T\}$, have the same dimensionality as the input data. For example, if $\boldsymbol{X} \in \mathbb{R}^{p+1}$, then $\boldsymbol{Z}_t \in \mathbb{R}^{p+1}$, $\forall t \in \{1, \ldots, T\}$.

2. the encoding process is fixed; the structure of the densities in the encoding process, $q_{\boldsymbol{\phi}_1}(\boldsymbol{z}_1|\boldsymbol{x})$ and $q_{\boldsymbol{\phi}_t}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$, $t \in \{2, \ldots, T\}$, is known a priori and is therefore not learnt. For example, in the most popular class of diffusion models, this structure is assumed Gaussian.

3. the encoding densities are defined in such a way that the distribution of the highest latent variable, when $T \to \infty$, is "standard" [130]. For example, the distribution of $\boldsymbol{Z}_T$, when $T \to \infty$, is standard Gaussian when these densities are assumed Gaussian.

Let us dive deeper into the significance of each of these assumptions. First of all, the fact that the latent variables have the same dimension as the input data means that the bottleneck structure of autoencoders is not present. Consequently, diffusion models are not suitable for representation learning, in contrast to VAEs. However, diffusion models are suitable in applications where representation learning is not a requirement, for example when the objective is purely generation of synthetic samples. Based on recent impressive results with respect to generative quality and diversity, especially prevalent in computer

vision [20, 106, 130], we anticipate that diffusion models can perform well on tabular data also. In fact, as noted, this was recently investigated by Kotelnikov et al. [67], demonstrating superiority over other alternatives. This paper is naturally an important inspiration and guide for our work in this thesis, as it demonstrates that the idea of using diffusion models for tabular data might have some merit.

The second assumption expresses that the structure of the encoding process is fixed a priori, as defined in Sohl-Dickstein et al.'s [124] inaugural formulation of generative diffusion models. For instance, the densities in Equation (3.6) can be assumed as Gaussian,

$$\begin{aligned} q(\boldsymbol{z}_1|\boldsymbol{x}) &= \mathcal{N}(\boldsymbol{z}_1; \sqrt{1-\beta_1}\boldsymbol{x}, \beta_1\mathbf{I}), \\ q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) &= \mathcal{N}(\boldsymbol{z}_t; \sqrt{1-\beta_t}\boldsymbol{z}_{t-1}, \beta_t\mathbf{I}), \quad t \in \{2, \dots, T\}, \end{aligned}$$

with coefficients β_t, $t \in \{1, \dots, T\}$. Notice that, contrary to in Equation (3.6), the encoding densities are not parameterized by $\boldsymbol{\phi}$, because they have no learnable parameters. This pre-defined structure has been kept in more recent work, e.g. in what is regarded by many as the set of seminal papers for diffusion models [20, 21, 47]. Obviously, the coefficients β_t, $t \in \{1, \dots, T\}$, play an important role, which we discuss in detail in Section 3.3.

Finally, the encoding distributions ensure that $\boldsymbol{Z}_T$ is "standard", when T is "sufficiently large". Extending our example above, the β_t-coefficients are defined such that $\boldsymbol{Z}_T$, $T \in \mathbb{Z}$, approximately follows a standard Gaussian distribution. This is an approximate statement because T has to be set to a finite scalar in practice, where some commonly found values in the literature are $T = 1000$ [47] or $T = 4000$ [21]. As we know, a simple recipe for generating synthetic data points can be followed once we have $p(\boldsymbol{z}_T)$; after parameter estimation, simply sample from $\boldsymbol{Z}_T \sim p(\boldsymbol{z}_T)$ and sequentially decode the sample, following Equation (3.4), in order to produce a new sample from $\{\boldsymbol{X}, \boldsymbol{Z}_1, \dots, \boldsymbol{Z}_T\} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T)$. Described in a hand-wavy way, if the model is trained to a satisfactory degree, this data point should be new and unseen, but should resemble the training data.

In the following, we describe two fundamental classes of diffusion models. In fact, they are the building blocks for the generative model that is mainly investigated in this thesis. To begin with, inspired by the terminology in Kotelnikov et al. [67], we introduce *Gaussian* diffusion models, which are quintessential [47, 124] for numerical data. Beware that these models are typically referred to as *diffusion models* in the literature. For clarity, the Gaussian assumptions we made in our reoccurring example above are from this branch of models. After discussing Gaussian diffusion models, we introduce *Multinomial* diffusion models [49], which make different assumptions in order to facilitate use in other applications, mainly for categorical data.

3.3 Gaussian Diffusion

After the high-level introduction to diffusion models in the previous section, we define some commonly used terminology and notation from the literature. Ho et al. [47] introduce a slight abuse of notation, letting $\boldsymbol{x}_0$ represent an observation from $\mathcal{D}$ and $\boldsymbol{x}_{1:T} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_T\}$ represent the set of latent variables. In this way, the entire set of variables in the hierarchy may be written as $\boldsymbol{x}_{0:T} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T\}$, which yields quite short equations with regards to notation. This notation is used in most of the literature on diffusion models. However, as statisticians, we prefer to keep our notation consistent with earlier; $\boldsymbol{x}$ represents an observation and $\{\boldsymbol{Z}_1, \dots, \boldsymbol{Z}_T\}$ represents the set of latent variables. We believe this leads to more easily understood equations.

Notice that diffusion models are usually said to consist of two different processes; the *forward* process and the *reverse* process. In fact, these concepts are already familiar to

us from Section 3.1, under different names. Bridging the gap between the terminologies, what we called the encoding and decoding processes in HVAEs are called the forward and reverse processes in diffusion models, respectively. In the following, we remind the reader of what these processes entail, while adding the defining assumptions of Gaussian diffusion models.

Forward Process

The forward, or *diffusion*, process is the common name for the approximate posterior distribution, represented by the density $q(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T|\boldsymbol{x})$. It consists of a fixed Markov chain that adds Gaussian noise to the input observation iteratively, following the equations

$$\begin{aligned} q(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T|\boldsymbol{x}) &= q(\boldsymbol{z}_1|\boldsymbol{x}) \prod_{t=2}^{T} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}), \\ q(\boldsymbol{z}_1|\boldsymbol{x}) &= \mathcal{N}(\boldsymbol{z}_1; \sqrt{1-\beta_1}\boldsymbol{x}, \beta_1 \mathbf{I}), \\ q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) &= \mathcal{N}(\boldsymbol{z}_t; \sqrt{1-\beta_t}\boldsymbol{z}_{t-1}, \beta_t \mathbf{I}), \quad t \in \{2, \dots, T\}. \end{aligned} \tag{3.8}$$

The coefficients $\beta_t \in (0, 1)$, $t \in \{1, \dots, T\}$, define a *variance schedule*. The values of these coefficients can be held constant as hyperparameters [47] or learnt [62]. For simplicity, we choose to fix their values prior to training. Again, this choice explains why we have removed the explicit dependence of the distributions on $\boldsymbol{\phi}$. In the literature, the two most prevalent a priori fixed variance schedules for the forward process are called the *linear* and *cosine* schedules.

Linear schedule. The linear schedule, first used by Ho et al. [47], defines the parameters as linearly increasing from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$. The authors justify this choice by noting that the constants are small relative to the image data they work with, scaled to $[-1, 1]$; "ensuring that the reverse and forward processes have approximately the same functional form while keeping the signal-to-noise ratio at $\boldsymbol{Z}_T$ as small as possible" [47].

Cosine schedule. The cosine schedule [21] was introduced as an improvement of the linear schedule. Precisely, Dhariwal and Nichol observed that the end of the forward process is too noisy when using the linear schedule and provided empirical evidence that the final parts of the process do not contribute a lot to sample quality. The cosine schedule is defined as

$$\begin{aligned} \beta_t &\coloneqq 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, \\ \bar{\alpha}_t \coloneqq \frac{f(t)}{f(0)}, \quad f(t) &\coloneqq \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2, \quad t \in \{1, \dots, T\}, \end{aligned} \tag{3.9}$$

where $s \coloneqq 0.008$ is a small offset designed to prevent β_t from being too small near $t = 0$, i.e. near the observation. The numerical value of this offset is justified based on pixel bin size, since the inventors of this schedule also evaluated diffusion models on image data. In fact, an interesting note is that progress in diffusion models has predominantly been made in image generation, especially early on, because this is the area where researchers first discovered comparable performance to GANs [20, 128]. Returning from our slight digression, note that when working with the cosine schedule in practice, the values of β_t are not allowed to grow larger than 0.999, to avoid singularities close to $t = T$ [21]. Additionally, notice the use of the symbol $\bar{\alpha}_t$ when defining the cosine schedule in Equations (3.9). There is a reason behind this choice of notation. Actually, $\bar{\alpha}_t$ represents a quantity that is very important in diffusion models, a quantity that is defined below.

Closed form forward density. An important property of the forward process is that it admits forward sampling to any latent, $\boldsymbol{Z}_t$, $t \in \{1, \ldots, T\}$, in closed form. At first glance, Equations (3.8) indicate that each conditional density needs to be calculated iteratively given realizations of all lower latents, as well as the input observation $\boldsymbol{x}$. Intuitively, remember that such a process goes from left to right in Figure 3.1. However, this recursive calculation is not necessary. The Gaussian assumptions in the diffusion process allow a closed form formulation,

$$q(\boldsymbol{z}_t|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}, (1-\bar{\alpha}_t)\mathbf{I}), \quad t \in \{1, \ldots, T\}, \tag{3.10}$$

where

$$\bar{\alpha}_t \coloneqq \prod_{i=1}^{t} \alpha_i, \quad \alpha_t \coloneqq 1 - \beta_t.$$

For completeness, the derivation of Equation (3.10) is provided in Appendix C.1. This equation makes it possible to calculate the conditional density of any latent in the forward chain, given only the variance schedule and the input observation. The main advantage of this is decreased computational burden while training the diffusion model, which is exploited in order to estimate the model parameters more efficiently. Estimation of parameters will be addressed in detail later. In addition to creating a computational advantage, Equation (3.10) illustrates the importance of $\bar{\alpha}_t$, facilitating simpler interpretation of the behaviour of the linear and cosine variance schedules. Figure 3.2 illustrates how $\bar{\alpha}_t$ evolves throughout the diffusion process, for both schedules. The cosine schedule was designed

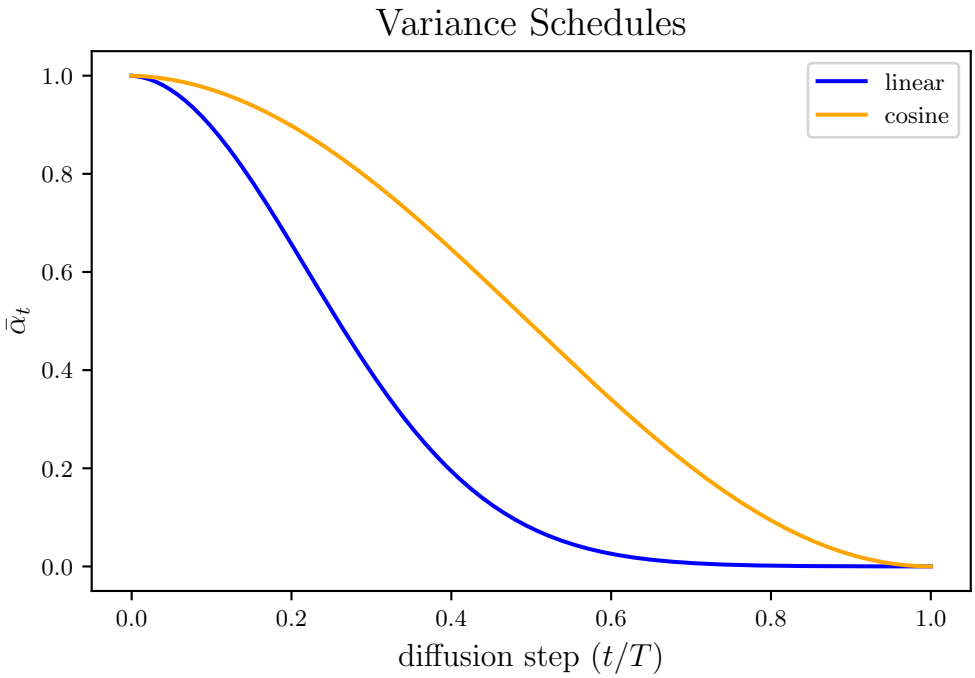


Figure 3.2: $\bar{\alpha}_t$ plotted against diffusion step for the linear and cosine schedules. This is a recreation of Figure 5 in Dhariwal and Nichol [21], with $T = 1000$.

to conserve more information further in the forward process, something it seems to do in comparison to the linear schedule, according to the figure. Furthermore, the equation qualitatively shows that $q(\boldsymbol{z}_T|\boldsymbol{x}) \approx \mathcal{N}(\boldsymbol{z}_T; \mathbf{0}, \mathbf{I})$, because $\bar{\alpha}_t \to 0$ when $t \to T$, which plays a role when defining the loss function for parameter estimation. Intuitively, the densities

that make up the forward process are transformed from a degenerate deterministic distribution at $t = 0$, "a Gaussian with zero variance and mean $\boldsymbol{x}$", to a standard Gaussian at $t = T$, following different paths based on the variance schedule.

Reverse Process

The reverse, or *generative*, process commonly refers to the joint distribution, represented by the density $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_T)$. Similarly to the forward process, this process is defined as a Markov chain with Gaussian transitions according to the equations

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_T) &= p(\boldsymbol{z}_T) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1), \\ p(\boldsymbol{z}_T) &= \mathcal{N}(\boldsymbol{z}_T; \mathbf{0}, \mathbf{I}), \\ p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) &= \mathcal{N}(\boldsymbol{z}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)), \quad t \in \{2, \ldots, T\}, \\ p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) &= \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)), \end{aligned} \tag{3.11}$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ represent any ML or mathematical model. Such a model is trained on inputs $\boldsymbol{z}_t$ and t, for $t \in \{1, \ldots, T\}$, in order to compute the mean and variance parameters in Equations (3.11). In other words, $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$ are predictions from the model, based on inputs $\boldsymbol{z}_t$ and t, for $t \in \{1, \ldots, T\}$. Commonly, neural networks are used for this task, as they have been used successfully by many researchers. We also use deep learning models in our experiments, whose architectures are outlined in Chapter 5. Notice that we do not explicitly highlight that the (hyper-)parameters in each of the densities can be different, as we did for HVAEs in Equation (3.4), i.e. we have removed the subindices on $\boldsymbol{\theta}$. This is intentional, because all the parameters $\boldsymbol{\theta}$, over all diffusion steps t, are estimated using the same model. Hence, $\boldsymbol{\theta}$ essentially represents neural network parameters, while the predictions from the neural network act as parameters in the Gaussian densities in Equations (3.11). We return to this idea when discussing parameter estimation in detail.

It is not obvious why the reverse process transitions in Equations (3.11) are defined as Gaussians. From a theoretical point of view, the ideal scenario would let us calculate the posteriors, $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$, $t \in \{2, \ldots, T\}$, and $q(\boldsymbol{x}|\boldsymbol{z}_1)$, such that we could compute

$$p(\boldsymbol{x}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_T) = p(\boldsymbol{z}_T) \prod_{t=2}^{T} q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) q(\boldsymbol{x}|\boldsymbol{z}_1),$$

directly. In this way, we could simply reverse the forward process by an iterative sampling procedure, enabling us to recreate the true observation $\boldsymbol{x}$ from standard Gaussian noise. However, the posterior distributions,

$$\begin{aligned} q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) &= \frac{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) p(\boldsymbol{z}_{t-1})}{p(\boldsymbol{z}_t)}, \quad t \in \{2, \ldots, T\}, \\ q(\boldsymbol{x}|\boldsymbol{z}_1) &= \frac{q(\boldsymbol{z}_1|\boldsymbol{x}) p^*(\boldsymbol{x})}{p(\boldsymbol{z}_1)}, \end{aligned} \tag{3.12}$$

are not tractable to compute. One reason why is that $p(\boldsymbol{z}_t), \forall t \in \{1, \ldots, T-1\}$, is intractable, especially when T is large, due to high-dimensional integrals while performing marginalization of the random variables. Another reason is that $q(\boldsymbol{x}|\boldsymbol{z}_1)$ depends on the true data density, $p^*(\boldsymbol{x})$, which is what we are trying to infer in generative models, meaning that the true posteriors depend on the entire data distribution [21]. Despite this intractability, it turns out that the posteriors are tractable when conditioned on $\boldsymbol{X}$.

Notice that this conditioning is superfluous because of the Markov assumptions, meaning that $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) = q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$, $\forall t \in \{2, \dots, T\}$, a property that is frequently used in derivations. More specifically, these distributions can be represented by the densities

$$\begin{aligned} q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) &= \mathcal{N}(\boldsymbol{z}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x}), \tilde{\Sigma}_t \mathbf{I}), \quad t \in \{2, \dots, T\}, \\ \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x}) &= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x} + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{z}_t, \\ \tilde{\Sigma}_t &= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \end{aligned} \tag{3.13}$$

For completeness, the derivation of these expressions is also given in Appendix C.1. Notice that $\tilde{\boldsymbol{\mu}}_1(\boldsymbol{z}_1, \boldsymbol{x}) = \frac{\sqrt{\bar{\alpha}_0}\beta_1}{1-\bar{\alpha}_1}\boldsymbol{x} + \frac{\sqrt{\alpha_1}(1-\bar{\alpha}_0)}{1-\bar{\alpha}_1}\boldsymbol{z}_1 = \boldsymbol{x}$ and $\tilde{\Sigma}_1 = \frac{1-\bar{\alpha}_0}{1-\bar{\alpha}_1}\beta_1 = 0$, because $\bar{\alpha}_0 = 1$. This means that $q(\boldsymbol{x}|\boldsymbol{z}_1, \boldsymbol{x})$ is a degenerate distribution, because the Gaussian density does not exist when the covariance matrix is not symmetric and positive-definite. This distribution can be represented by a Dirac delta at $\boldsymbol{x}$,

$$q(\boldsymbol{\xi}|\boldsymbol{z}_1, \boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{\xi} = \boldsymbol{x}, \\ 0, & \boldsymbol{\xi} \neq \boldsymbol{x}, \end{cases}$$

where we rename the argument to $\boldsymbol{\xi}$ for clarity. Intuitively, this result seems sensible, because we would deterministically choose $\boldsymbol{x}$ in the final forward posterior density if we have access to $\boldsymbol{x}$ — there is no point in defining a probability distribution over $\boldsymbol{x}$ in such a case. Unfortunately, we cannot use Equations (3.13) to reverse the forward process either. Recall that the philosophy of generative modelling is to estimate $p^*(\boldsymbol{x})$ using the realizations $\boldsymbol{x} \in \mathcal{D}$ of $\boldsymbol{X} \coloneqq \{X^1, \dots, X^p, Y\} \sim p^*(\boldsymbol{x})$, to enable inference on observations $\boldsymbol{x}' \notin \mathcal{D}$ from the same distribution. This explains why we cannot use these equations to reverse the forward process, since we do not have access to an arbitrary observation $\boldsymbol{x}' \notin \mathcal{D}$ of $\boldsymbol{X} \sim p^*(\boldsymbol{x})$ at time of inference, i.e. while simultaneously trying to generate $\boldsymbol{x}'$. Thus, since we cannot rely on the posteriors (Equations (3.12)) or the conditional posteriors (Equations (3.13)), the approximate densities in the reverse process, as stated in Equations (3.11), are introduced.

Interestingly, the reason why the reverse process transitions in Equations (3.11) are defined as Gaussians can be observed from Equations (3.13); in order to reverse the forward process, we seek to learn a reverse process where each transition $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$, $t \in \{2, \dots, T\}$, "matches" $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$, $t \in \{2, \dots, T\}$, as closely as possible. This "matching" procedure can be interpreted from the ELBO. Note that for $t = 1$, we cannot perform such a matching explicitly, since we defined $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ to be stochastic. Similarly to in a VAE, this is represented by a reconstruction term in the ELBO. This discussion is a little hand-wavy, with focus on intuition, but is made rigorous after deriving the loss function for estimating the parameters of the diffusion model, which is based on the ELBO. Before moving on to a detailed explanation of parameter estimation, Algorithm 1 shows how a Gaussian diffusion model can be trained in practice. We develop this algorithm with the training algorithm in Ho et al. [47] as inspiration. Steps 3 through 6 represent the forward process, where step 6 uses the closed form formulation from Equation (3.10) to diffuse an observation. Steps 7 and 8 represent the reverse process, where the parameters of a neural network $\boldsymbol{g}_{\boldsymbol{\theta}}$ are modified with backpropagation and a stochastic gradient descent method. Keep in mind that training is most commonly done with batches of observations, not singular observations, in each iteration, but we do not include this explicitly in the algorithm for simplicity. In the next subsection, we justify why this algorithm can be used.

Algorithm 1 Training a Gaussian Diffusion Model

Assume T and β_t, $t \in \{1, \dots, T\}$, set a priori.
while not converged **do**
  $\boldsymbol{x} \leftarrow$ one sample from $\mathcal{D}$ (equivalently from $\boldsymbol{X} \sim p^*(\boldsymbol{x})$).
  $t \leftarrow$ one sample from $\mathcal{T} \sim \text{Uniform}[1, T]$.
  $\boldsymbol{\varepsilon} \leftarrow$ one sample from $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
  $\boldsymbol{z}_t \leftarrow \sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}$. ▷ Diffuse $\boldsymbol{x}$ to step t.
  Calculate $\boldsymbol{\nabla}_{\boldsymbol{\theta}} \|\boldsymbol{\varepsilon} - \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)\|_2^2$ via backpropagation.
  Take gradient descent step.
end while

Details About Parameter Estimation

The main objective of diffusion models is to estimate the parameters of the reverse process, such that we can reverse the destruction of information in the forward process. This can be done in different ways. We have chosen to interpret diffusion models as likelihood-based models, meaning that our objective is to maximize the likelihood of the data, $p(\boldsymbol{\theta}; \mathcal{D})$, as shown in Equation (3.1). As a side note, diffusion models can be interpreted as score-based generative models as well, an interpretation we have chosen not to focus on. Precisely, the equivalence between denoising diffusion probabilistic models (DDPMs) and score-based generative models was shown by Ho et al. [47], in one of the key papers we rely on in this work. The interested and motivated reader should therefore also study diffusion models from a perspective of score-based generative models, *stochastic differential equations* (SDEs) and *Langevin dynamics* [128, 129, 130]. In fact, according to Tomczak [138], one of the reasons why diffusion probabilistic models have reached popularity is that they can be approached from several different perspectives, which grants an extensive theoretical foundation.

Returning from our slight digression, diffusion models are trained by optimizing a variational lower bound, analogously to VAEs. Without loss of generality, we focus on maximizing the likelihood of an arbitrary instance $\boldsymbol{x} \in \mathcal{D}$ in the following. The marginal likelihood, $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, is intractable, meaning that an approximate technique is necessary. More specifically, the loss function is the (negative) evidence lower bound (ELBO). The ELBO in diffusion models also appears as a lower bound to the marginal log-likelihood,

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq & \underbrace{\mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)\right]}_{L_0} - \underbrace{D_{KL}(q(\boldsymbol{z}_T|\boldsymbol{x}) \parallel p(\boldsymbol{z}_T))}_{L_T} \\ & - \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})}\left[D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t))\right]}_{L_{t-1}} =: \mathcal{L}_{\boldsymbol{\theta}}^{GD}(\boldsymbol{x}). \end{aligned} \tag{3.14}$$

The derivation of this lower bound is quite involved, but it is given in Appendix C.1 for completeness. Each of these terms can be given relatively clear interpretations [76]. The first term, L_0, can be interpreted as a reconstruction term, in the same way as in VAEs and HVAEs. The second term, L_T, can be interpreted as a prior matching term; it represents how close the final forward process density is to the standard Gaussian prior of the reverse process. This also appears in the ELBO in VAEs, as the KL divergence between the encoder, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, and the prior belief on $\boldsymbol{Z}$, $p(\boldsymbol{z})$. Each of the components in the third term, L_{t-1}, can be interpreted as a regularizer; at each diffusion step $t \in \{2, \dots, T\}$, the reverse process density, $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$, should be as similar as possible to the forward process posterior conditioned on the observation, $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$. This is an attractive property,

because $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$ describes how each latent $\boldsymbol{z}_t$ should be denoised into $\boldsymbol{z}_{t-1}$ when the initial observation $\boldsymbol{x}$ is known. Thus, during training, the density $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$ acts as a ground-truth signal for each reverse process density [76]. A closer look at $\mathcal{L}_{\boldsymbol{\theta}}^{GD}$ reveals some important properties of the loss that can be used during parameter estimation.

Optimizing L_T. First of all, under the assumption of a known and fixed variance schedule, where the coefficients β_t, $t \in \{1, \dots, T\}$, are defined such that $q(\boldsymbol{z}_T|\boldsymbol{x}) \approx \mathcal{N}(\boldsymbol{z}_T; \mathbf{0}, \mathbf{I})$, the term L_T can be ignored in the objective function. This is because $q(\boldsymbol{z}_T|\boldsymbol{x})$ has no learnable parameters, and $q(\boldsymbol{z}_T|\boldsymbol{x}) \approx p(\boldsymbol{z}_T)$, meaning that L_T can be treated as a small constant during optimization.

Optimizing L_{t-1}. Second, the main cost of maximization stems from minimizing L_{t-1}, $\forall t \in \{2, \dots, T\}$. Observe that $\sum_{t=2}^{T} L_{t-1}$ should be minimized in order to maximize $\mathcal{L}_{\boldsymbol{\theta}}^{GD}$. This explains why we previously stated that $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ should "match" $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$. More rigorously, the objective is to minimize their KL divergence,

$$D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)), \forall t \in \{2, \dots, T\}.$$

Importantly, notice that this quantity can be calculated in closed form, making optimization much more feasible than in regular HVAEs. This is because, according to Equations (3.11) and (3.13), both the terms are Gaussian, and the KL divergence between two Gaussians has a closed form expression, as derived in Appendix F. We return to this fact later.

Recall that the mean, $\boldsymbol{\mu}_{\boldsymbol{\theta}}$, and variance, $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$, of the reverse process densities, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$, $t \in \{2, \dots, T\}$, represent a ML model. However, since the variance schedule, β_t, $t \in \{1, \dots, T\}$, is known a priori, it is convenient to simply let $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) \coloneqq \tilde{\Sigma}_t \mathbf{I}$, $t \in \{2, \dots, T\}$, where $\tilde{\Sigma}_t$ is given in Equations (3.13). With this strategy, we do not need to train a ML model to predict the variance parameters of the reverse process densities, which makes mathematical derivations, as well as implementations, simpler. As a side note, Ho et al. [47] state that this choice is empirically optimal, among two different untrained time dependent constants, when the instance $\boldsymbol{x}$ is deterministically set to one point, which is precisely what we assume of our instance. Assuming that we make this choice, the parametric family of functions and the variances of $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ are already equal, for all $t \in \{2, \dots, T\}$. After this simplification, the only element remaining to match, in order to minimize the KL divergence between the two distributions at each step, is the mean. Ho et al. [47] develop three different techniques for matching the mean predictions of $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ in Equations (3.11), $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$, with the means of $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$ in Equations (3.13), $\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})$, for all $t \in \{2, \dots, T\}$. These three strategies are explained in the following.

Predict mean, alternative I. Since $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ does not depend on $\boldsymbol{x}$, we cannot simply set $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) = \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})$, $t \in \{2, \dots, T\}$, as we did with the variances. The first parameterization of $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ is based on the fact that we can write

$$L_{t-1} = \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} \left[\frac{1}{2\tilde{\Sigma}_t} \|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})\|_2^2 \right], \quad t \in \{2, \dots, T\}, \tag{3.15}$$

when the reverse transitions are defined as $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{z}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t), \tilde{\Sigma}_t \mathbf{I})$. This can be shown by direct calculation of $D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t))$, which is given in Appendix C.1 for completeness. Thus, the most obvious choice is to fit a ML model, $\boldsymbol{\mu}_{\boldsymbol{\theta}}$, whose predictions, $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$, are close to $\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})$, trained by minimization of Equation (3.15).

Predict mean, alternative II. Instead of predicting the mean directly, we can alternatively train a ML model to predict the input instance. Keeping Equations (3.13) in mind, this strategy can be derived after setting

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) + \frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{z}_t, \quad t \in \{2, \ldots, T\}. \tag{3.16}$$

Here, we approximate $\boldsymbol{x}$ by predictions from a model $\boldsymbol{f}_{\boldsymbol{\theta}}$, typically a neural network, based on inputs $\boldsymbol{z}_t$ and t for $t \in \{2, \ldots, T\}$. Defining $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$ like this yields a modified version of Equation (3.15),

$$L_{t-1} = \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})}\left[\frac{1}{2\tilde{\Sigma}_t}\frac{\bar{\alpha}_{t-1}\beta_t^2}{(1-\bar{\alpha}_t)^2}\|\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \boldsymbol{x}\|_2^2\right], \quad t \in \{2, \ldots, T\}, \tag{3.17}$$

which is attained by inserting Equations (3.13) and (3.16) into Equation (3.15). The details are provided in Appendix C.1. Thus, the ML model, $\boldsymbol{f}_{\boldsymbol{\theta}}$, can be trained to predict the observation $\boldsymbol{x}$ as closely as possible from the noised version of $\boldsymbol{x}$ at any diffusion step $t \in \{2, \ldots, T\}$, by using Equation (3.17) as loss function.

Predict mean, alternative III. However, Equation (3.16) is not the only possible way of parameterizing $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$ to "match" $\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})$. The reparameterization trick applied to Equation (3.10) can be used to obtain

$$\begin{aligned}\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x}) &= \tilde{\boldsymbol{\mu}}_t\left(\boldsymbol{z}_t, \frac{1}{\sqrt{\bar{\alpha}_t}}(\boldsymbol{z}_t - \sqrt{1-\bar{\alpha}_t})\boldsymbol{\varepsilon}\right) \\ &= \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}\right),\end{aligned} \tag{3.18}$$

where $\boldsymbol{\varepsilon}$ is a realization of the random variable $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. More specifically, reparameterization of $q(\boldsymbol{z}_t|\boldsymbol{x})$ reads $\boldsymbol{z}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}$, which can be rearranged as $\boldsymbol{x} = \frac{1}{\sqrt{\bar{\alpha}_t}}(\boldsymbol{z}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon})$. This can be plugged in for $\boldsymbol{x}$ in $\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})$ in Equations (3.13), which yields Equation (3.18). Insertion into Equation (3.15) now yields

$$L_{t-1} = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\frac{1}{2\tilde{\Sigma}_t}\left\|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t) - \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}\right)\right\|_2^2\right], \tag{3.19}$$

where the expectation is taken with respect to $\boldsymbol{\mathcal{E}} \sim p(\boldsymbol{\varepsilon}) = \mathcal{N}(\boldsymbol{\varepsilon}; \mathbf{0}, \mathbf{I})$ because of the reparameterization trick. Equation (3.19) reveals that $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ should be trained to predict $\frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}\right)$ based on inputs $\boldsymbol{z}_t$ and t, for $t \in \{2, \ldots, T\}$. Hence, the loss function can be stated as

$$L_{t-1} = \mathbb{E}_{p(\boldsymbol{\varepsilon})}\left[\frac{\beta_t^2}{2\tilde{\Sigma}_t\alpha_t(1-\bar{\alpha}_t)}\|\boldsymbol{\varepsilon} - \boldsymbol{g}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t)\|_2^2\right], \quad t \in \{2, \ldots, T\}, \tag{3.20}$$

were $\boldsymbol{g}_{\boldsymbol{\theta}}$ is a model that predicts a realization $\boldsymbol{\varepsilon}$ of the random variable $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ from $\boldsymbol{z}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}$ and t, trained with Equation (3.20) as loss function. For completeness, the arithmetic details from this discussion are given in Appendix C.1.

Thus, we are left with three different choices regarding what to train our ML model to predict; either we predict $\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})$, $\boldsymbol{x}$ or $\boldsymbol{\varepsilon}$, at each diffusion step $t \in \{2, \ldots, T\}$. How do we make this choice? Ho et al. [47] found that the best choice is to predict the noise, $\boldsymbol{\varepsilon}$, when combined with a reweighted loss function

$$L_{\text{simple}}^{GD} \coloneqq \mathbb{E}_{p(\boldsymbol{\varepsilon}),t}\left[\|\boldsymbol{\varepsilon} - g_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t)\|_2^2\right], \tag{3.21}$$

where the subscript t of the expectation is a shorthand notation that indicates that we also calculate the expectation over diffusion steps t that are realizations of a random

variable $\mathcal{T} \sim \text{Uniform}[1, T]$. More specifically, the authors found it beneficial for sample quality and simplicity of implementation to simply discard the factor $\frac{\beta_t^2}{2\tilde{\Sigma}_t \alpha_t (1-\bar{\alpha}_t)}$, and use the reweighted loss function in Equation (3.21), for each instance $\boldsymbol{x} \in \mathcal{D}$. Based on this conviction, we choose to use this technique for training a Gaussian diffusion model in the experiments outlined in Chapter 5 as well.

Optimizing L_0. Taking a step back, we have discussed the regularizers L_{t-1} to a great extent and shown how they can be calculated and minimized in several different ways. However, recall that the ELBO in Equation (3.14) contains a reconstruction term, L_0, as well. Above we state that Equation (3.21) is used as loss function, essentially replacing maximization of $\mathcal{L}_{\boldsymbol{\theta}}^{GD}$ with minimization of L_{simple}^{GD}, with stochastic sampling of diffusion steps uniformly between 1 and T. In a rather cryptic fashion, Ho et al. [47] state that "the $t = 1$ case corresponds to L_0 (...)", in reference to L_{simple}^{GD}, without clarifying exactly *how*. Theoretically, we find that defining the lowest reverse process density as

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) &= \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)), \\ \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) &= \frac{1}{\sqrt{1-\beta_1}} \left(\boldsymbol{z}_1 - \sqrt{\beta_1} \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) \right), \\ \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) &= a\mathbf{I}, \end{aligned} \tag{3.22}$$

where $a \in \mathbb{R}$ is arbitrary, enables inclusion of L_0 in L_{simple}^{GD}. Note that a can be arbitrary since we reweight the ELBO anyway, removing the coefficients in front of the Euclidean norm. Despite the simple calculations needed to show this, we include the computations in Appendix C.1 for completeness. Thus, the way we understand Ho et al.'s [47] statement is that implicit inclusion of L_0 in L_{simple}^{GD} at $t = 1$ is equivalent to defining $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ as in Equation (3.22), before explicitly computing and reweighting the reconstruction term, $L_0 = \mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{x})} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)]$, and including this term in the loss function alongside L_{simple}^{GD} for $t \in \{2, \dots, T\}$. It follows that $\boldsymbol{g}_{\boldsymbol{\theta}}$ is trained to predict $\boldsymbol{\varepsilon}$ in $\boldsymbol{z}_1 = \sqrt{\bar{\alpha}_1}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_1}\boldsymbol{\varepsilon}$, which is consistent with $\boldsymbol{g}_{\boldsymbol{\theta}}$ for the other diffusion steps. Thus, using L_{simple}^{GD} (Equation (3.21)) as objective function implicitly teaches the lowest reverse process distribution to reconstruct the input with a mean squared error loss, similar to the continuous components of the decoder in TVAE, as discussed in Section 2.9. To be clear, recall that L_T is not considered during optimization, since it has no learnable parameters. In addition, notice that when $t > 1$, L_{simple}^{GD} corresponds to a reweighted version of L_{t-1} in Equation (3.20). Hence, a minimization of L_{simple}^{GD} can replace a (reweighted) maximization of $\mathcal{L}_{\boldsymbol{\theta}}^{GD}$, because all the terms in the latter are included in the former.

Interpretation. Thus, we are now able to understand why Algorithm 1 can be used for training a Gaussian diffusion model, since we minimize

$$\mathbb{E}_{p(\boldsymbol{\varepsilon}),t} \left[\|\boldsymbol{\varepsilon} - \boldsymbol{g}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t)\|_2^2 \right],$$

iteratively over batches of inputs $\boldsymbol{x} \in \mathcal{D}$. The expectation is approximated with a Monte Carlo (MC) method, over stochastic samples of diffusion steps and noise. L_{simple}^{GD} can be interpreted as a weighted variational bound that gives different weight to reconstruction and regularization compared to directly optimizing $\mathcal{L}_{\boldsymbol{\theta}}^{GD}$. This idea is not new, and was for example introduced in the theory of VAEs with the β-VAE framework [45]. Ho et al. [47] state that this reweighting of the objective causes the objective to "down-weight loss terms corresponding to small t", leading to the network prioritizing more accurate denoising of latents at larger t. This lead to better sampling quality in their setup, according to the authors, probably because of the connection to score-based generative modelling, which uses analogous losses [128, 129, 130].

ML model design. Before moving on, we remind the reader about the vital role of the ML model trained to predict the relevant quantities. When choosing a deep learning model, the neural networks can be of any architecture we see fit depending on the application. The importance of its design should not be underestimated.

3.4 Multinomial Diffusion

Up until this point, we have mostly discussed diffusion models with Gaussian assumptions. However, the Gaussian assumptions in the forward and reverse processes represent only one way of restricting diffusion models to achieve tractability. In this section, we discuss *Multinomial* diffusion models, for handling categorical data, first introduced by Hoogeboom et al. [49].

Interestingly, note that Sohl-Dickstein et al. [124] experimented with diffusion models with Binomial assumptions, showing successful empirical results for binary sequence learning. Hoogeboom et al. [49] extend this idea by defining a diffusion process directly on categorical features with more than two levels. Recall the three key simplifying assumptions, effectively transforming a MHVAE to a diffusion model, stated in Section 3.2. These still hold for Multinomial diffusion models. The only assumption that differs in Gaussian and Multinomial diffusion is the assumption on the forward structure, otherwise known as the *diffusion kernel*, yielding a different distribution for $\boldsymbol{z}_T$ and modifying the assumed reverse process structure accordingly.

Precisely, let $\boldsymbol{x} \in \mathcal{D}$ represent an arbitrary realization of $\boldsymbol{X} \coloneqq \{X^1, \dots, X^p, Y\} \sim p^*(\boldsymbol{x})$. Assume that we are interested in modelling the jth element of $\boldsymbol{X}$, denoted by $X^j \sim p^*(x^j)$, which we assume is a categorical random variable with K different categories. Thus, element x^j of $\boldsymbol{x}$ contains a value representing one of the K categories. Suppose that this value is represented in *one-hot encoded* (OHE) format, as a vector $\boldsymbol{x}^j_{\text{OHE}} = \{x^{j,1}_{\text{OHE}}, \dots, x^{j,K}_{\text{OHE}}\}$. For clarity, this means that, assuming x^j pertains to category $k \in \{1, \dots, K\}$, then $x^{j,k}_{\text{OHE}} = 1$ and $x^{j,i}_{\text{OHE}} = 0$ for $i \neq k$. As a consequence, we let $\boldsymbol{X}^j_{\text{OHE}}$ represent X^j in $\mathbb{R}^{K\times 1}$, and introduce the latent variables $\boldsymbol{Z}^j_t$, $t \in \{1, \dots, T\}$, corresponding to $\boldsymbol{X}^j_{\text{OHE}}$, as vectors in $\mathbb{R}^{K\times 1}$. For ease of notation, in the following we drop the subscript text, in addition to the superscript specification of the a priori selected element j of $\boldsymbol{x} \in \mathcal{D}$, in all observable vectors, like $\boldsymbol{x} \coloneqq \boldsymbol{x}^j_{\text{OHE}}$, with the implicit understanding that $\boldsymbol{x}$ represents the value x^j in OHE format. Moreover, we drop the superscript j in the latent variables, like $\boldsymbol{Z}_t \coloneqq \boldsymbol{Z}^j_t$, $t \in \{1, \dots, T\}$.

Forward process. The forward process in Multinomial diffusion is characterized by

$$
\begin{aligned}
q(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T|\boldsymbol{x}) &= q(\boldsymbol{z}_1|\boldsymbol{x}) \prod_{t=2}^{T} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}), \\
q(\boldsymbol{z}_1|\boldsymbol{x}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_1; (1-\beta_1)\boldsymbol{x} + \frac{\beta_1}{K}\mathbb{1}\right), \\
q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_t; (1-\beta_t)\boldsymbol{z}_{t-1} + \frac{\beta_t}{K}\mathbb{1}\right), \quad t \in \{2, \dots, T\},
\end{aligned}
$$

where $\mathbb{1} \in \mathbb{R}^{K\times 1}$ is a vector of ones. For completeness, the categorical distribution is defined in Appendix A. From the forward process mass functions we can derive a closed form forward sampling formula,

$$
q(\boldsymbol{z}_t|\boldsymbol{x}) = \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_t; \bar{\alpha}_t\boldsymbol{x} + \frac{1-\bar{\alpha}_t}{K}\mathbb{1}\right), \tag{3.23}
$$

whose derivation is given in Appendix C.2. Moreover, from the development of $\bar{\alpha}_t$ with increasing diffusion step, illustrated in Figure 3.2, we deduce that the limit of the forward process reaches the distribution

$$p(\boldsymbol{z}_T) = \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_T; \frac{1}{K}\mathbb{1}\right),$$

because the first term, $\bar{\alpha}_T\boldsymbol{x}$, in the parameter of the categorical distribution in Equation (3.23) becomes vanishingly small, leaving $\frac{1-\bar{\alpha}_T}{K}\mathbb{1} \approx \frac{1}{K}\mathbb{1}$ as the dominating term.

Reverse process. Like in Gaussian diffusion, the forward posterior distributions are not tractable. However, the posteriors can be calculated analytically when conditioned on $\boldsymbol{X}^j_{\text{OHE}}$, which yields the conditional posteriors

$$\begin{aligned} q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_{t-1}; \boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{x})\right), \quad t \in \{2, \dots, T\}, \\ \boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{x}) &= \frac{\boldsymbol{\lambda}}{\|\boldsymbol{\lambda}\|_1}, \\ \boldsymbol{\lambda} &= \left[\alpha_t \boldsymbol{z}_t + \frac{1-\alpha_t}{K}\mathbb{1}\right] \odot \left[\bar{\alpha}_{t-1}\boldsymbol{x} + \frac{(1-\bar{\alpha}_{t-1})}{K}\mathbb{1}\right], \end{aligned} \tag{3.24}$$

where $\|\cdot\|_1$ is the L^1 norm or *Manhattan distance*, i.e. the sum of absolute elements. For completeness, these equations are also derived in Appendix C.2. Notice that $\boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_1, \boldsymbol{x}) = \frac{1}{\|\boldsymbol{\lambda}\|_1}((\alpha_1\boldsymbol{z}_1 + \frac{1-\alpha_1}{K}\mathbb{1}) \odot (\bar{\alpha}_0\boldsymbol{x} + \frac{1-\bar{\alpha}_0}{K}\mathbb{1}) = \frac{1}{\|\boldsymbol{\lambda}\|_1}((\alpha_1\boldsymbol{z}_1 + \frac{1-\alpha_1}{K}) \odot \boldsymbol{x}) = \boldsymbol{x}$, because $\boldsymbol{x}$ is OHE. This means that $q(\boldsymbol{x}|\boldsymbol{z}_1, \boldsymbol{x})$ follows a deterministic degenerate distribution at one point, analogously to in Gaussian diffusion.

As always in diffusion models, the objective is to learn a reverse process that is able to denoise the slowly destroyed input data. For this aim, the conditional forward posteriors $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$, $t \in \{2, \dots, T\}$, are "matched" as closely as possible by the reverse process densities $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$, $t \in \{2, \dots, T\}$. Hence, the reverse process is parameterized as

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) &= p(\boldsymbol{z}_T)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1), \\ p(\boldsymbol{z}_T) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_T; \frac{1}{K}\mathbb{1}\right), \\ p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_{t-1}; \boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t))\right), \quad t \in \{2, \dots, T\}, \\ p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) &= \text{Categorical}_{\text{OHE}}(\boldsymbol{x}; \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)), \end{aligned}$$

where $\boldsymbol{h}_{\boldsymbol{\theta}}$ represents a ML model of choice, commonly a neural network with a suitable architecture, that is trained to predict a probability vector for $\boldsymbol{x}$ from $\boldsymbol{z}_t$ and t for $t \in \{1, \dots, T\}$. Notice that, when $\boldsymbol{h}_{\boldsymbol{\theta}}$ is parameterized by a neural network, we ensure non-negative and correctly normalized probability vector predictions, $\boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$, $t \in \{1, \dots, T\}$, via a softmax function in the output layer of the network. As in Gaussian diffusion, we cannot use the matching principle when $t = 1$, since the final decoder is stochastic. This is handled via a reconstruction term in the ELBO, which assures that $\boldsymbol{h}_{\boldsymbol{\theta}}$ is trained to predict a probability parameter in $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ that maximizes the likelihood of each input observation, $\boldsymbol{x}$, in the training data.

Before moving on to a detailed discussion concerning parameter estimation, Algorithm 2 describes how a Multinomial diffusion model can be trained in practice. We develop this algorithm analogously to in Gaussian diffusion, hoping to show how similar the training procedures are. Steps 4 through 7 represent the forward process, where step 6 uses

Algorithm 2 Training a Multinomial Diffusion Model

1: Assume T, β_t, $t \in \{1, \dots, T\}$, set a priori and select a **categorical** feature.
2: $K \leftarrow$ number of levels in categorical feature.
3: **while** not converged **do**
4: $\quad \boldsymbol{x} \leftarrow$ one sample from $\mathcal{D}$ (equivalently from $\boldsymbol{X} \sim p^*(\boldsymbol{x})$).
5: $\quad \boldsymbol{x} \leftarrow$ select element corresponding to categorical feature and one-hot encode.
6: $\quad t \leftarrow$ one sample from $\mathcal{T} \sim \text{Uniform}[1, T]$.
7: $\quad \boldsymbol{z}_t \leftarrow$ one sample from $\boldsymbol{Z} \sim \text{Categorical}_{\text{OHE}}\left(\bar{\alpha}_t \boldsymbol{x} + \frac{1-\bar{\alpha}_t}{K}\mathbb{1}\right)$.
8: $\quad$ **if** $t = 1$ **then**
9: $\quad\quad$ Calculate $-\boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ via backpropagation.
10: $\quad$ **else**
11: $\quad\quad$ Calculate $\boldsymbol{\nabla}_{\boldsymbol{\theta}} D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t))$ via backpropagation.
12: $\quad$ **end if**
13: $\quad$ Take gradient descent step.
14: **end while**

the closed form formulation from Equation (3.23), enabling efficient training. In practice, sampling from a categorical distribution is performed via the *Gumbel-Max trick* [78], which we outline in Appendix A.1. Steps 8 through 13 represent the reverse process, where the parameters of the neural network $\boldsymbol{h}_{\boldsymbol{\theta}}$ are modified with backpropagation and a stochastic gradient descent method. Notice that this network is not explicitly stated in Algorithm 2 to promote a less cluttered algorithm, but it appears implicitly through $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$, $t \in \{2, \dots, T\}$, and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$. Before continuing, note that we derived the ELBO in Equation (3.14) without any assumptions specific to Gaussian diffusion, meaning that parameter estimation in Multinomial diffusion is based on the same ELBO. However, in this case, L_0 is not implicitly included in the expression for L_{t-1} when $t = 1$, which is why the backpropagation algorithm is applied slightly differently when $t = 1$ in Algorithm 2. As noted previously, keep in mind that training is most commonly performed with batches of observations, not singular observations, in each iteration, but we do not include this explicitly in the algorithm for simplicity. In the next paragraph, we explain why Algorithm 2 can be used for training, and state the expanded expressions of $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ and $D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t))$ that appear in lines 8 and 10 in the algorithm.

Details about parameter estimation. Estimation of the parameters follows a similar idea as discussed for Gaussian diffusion. The starting point is the ELBO, $\mathcal{L}_{\boldsymbol{\theta}}^{GD}$, as stated in Equation (3.14). L_T can still be neglected, since it has no learnable parameters. The KL divergences between the two sets of categorical distributions in L_{t-1}, $t \in \{2, \dots, T\}$, can be computed as

$$
\begin{aligned}
&D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)) \\
&= D_{KL}(\text{Categorical}_{\text{OHE}}(\boldsymbol{z}_{t-1}|\boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{x})) \parallel \text{Categorical}_{\text{OHE}}(\boldsymbol{z}_{t-1}|\boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)))) \\
&= \sum_{k=1}^{K} \Lambda_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{x})^k \cdot \log \frac{\Lambda_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{x})^k}{\Lambda_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t))^k},
\end{aligned} \tag{3.25}
$$

where $\Lambda_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{x})^k$ and $\Lambda_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t))^k$, for $k \in \{1, \dots, K\}$, are the elements of $\boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{x})$ and $\boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t))$, respectively. Moreover, L_0 can be calculated by

$$
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) = \sum_{k=1}^{K} x^k \log h_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)^k, \tag{3.26}
$$

where x^k and $h_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)^k$, for $k \in \{1, \ldots, K\}$, are the elements of the OHE vector $\boldsymbol{x}$ and the probability vector $\boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)$, respectively. Equation (3.26) follows directly from the PMF of a categorical distribution, as shown in Equation (A.2) in Appendix A, because $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) = \prod_{k=1}^{K}(h_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)^k)^{x^k}$. In fact, this is identical to the (negative) cross-entropy loss we defined for the OHE components of the decoder in TVAE, as discussed in Section 2.9. Thus, the loss function in Multinomial diffusion is

$$\begin{aligned} L_j^{MD} \coloneqq \sum_{t=2}^{T} \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} \left[D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)) \right] \\ - \mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) \right], \end{aligned} \tag{3.27}$$

where the KL divergences in L_{t-1}, $t \in \{2, \ldots, T\}$, as well as the conditional log-likelihood term in L_0, are calculated as shown in Equations (3.25) and (3.26). Notice the addition of the sub-index j in L_j^{MD}, which explicitly highlights that Multinomial diffusion models one categorical feature j at a time. This notation is necessary in Section 3.6, where we model several categorical features in the same dataset simultaneously. Conclusively, we are able to understand why Algorithm 2 can be used to train a Multinomial diffusion model; we essentially minimize Equation (3.27) iteratively over values from a categorical feature j from batches of inputs $\boldsymbol{x} \in \mathcal{D}$ and over stochastic samples of diffusion steps, using a MC method.

3.5 Sampling From Diffusion Models

In this section, we display algorithms for generating synthetic data from Gaussian and Multinomial diffusion models. We require that the parameters of the models are estimated before applying these algorithms. Sampling from a Gaussian diffusion model is shown in Algorithm 3. This algorithm is developed with the sampling algorithm by Ho et al. [47] as inspiration. Essentially, it generates each synthetic observation, $\boldsymbol{z}_0$, by sampling an observation from a standard Gaussian random variable, before running the sample iteratively through a denoising process, resembling Langevin dynamics with $\boldsymbol{g}_{\boldsymbol{\theta}}$ estimating the data density gradient [47]. Recall that the reverse process densities are defined as

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{z}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)), \quad t \in \{2, \ldots, T\}.$$

Furthermore, recall that, while deriving how to estimate the parameters of a Gaussian diffusion model, we let $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) \coloneqq \tilde{\Sigma}_t \mathbf{I}$, $t \in \{2, \ldots, T\}$. Additionally, as realized from Equation (3.19), we defined

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) \coloneqq \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{z}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) \right), \quad t \in \{2, \ldots, T\}.$$

Thus, the denoising process simply consists of a reparameterization of $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ for $t \in \{T, \ldots, 2\}$, with the predictions $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$ defined as above at each step in the loop. Finally, when $t = 1$ we do not apply reparameterization, because $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) = \frac{1}{\sqrt{1-\beta_1}} \left(\boldsymbol{z}_1 - \sqrt{\beta_1} \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) \right)$ predicts the mean of $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$, which is the mode, i.e. the point of largest probability density, in the Gaussian, no matter the variance $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) = a\mathbf{I}$.

Similarly, sampling from a Multinomial diffusion model is shown in Algorithm 4. We develop this algorithm analogously to Algorithm 3, hoping to show how similar the sampling procedures are. The idea is similar; the sampling process starts by sampling a realization $\boldsymbol{z}_T$ from $\boldsymbol{Z}_T \sim p(\boldsymbol{z}_T)$, which in this case is a categorical distribution with equal probability of sampling each of the K categories. Then, the sample is iteratively modified

Algorithm 3 Sampling from a Gaussian Diffusion Model

Assume T and β_t, $t \in \{1, \dots, T\}$, set a priori.
Assume $\boldsymbol{g_\theta}$ has been trained.
$\boldsymbol{z}_T \leftarrow$ one sample from $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
for $t \in \{T, \dots, 1\}$ **do**
    if $t > 1$ **then**
        $\boldsymbol{\varepsilon} \leftarrow$ one sample from $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
    else
        $\boldsymbol{\varepsilon} \leftarrow \mathbf{0}$. ▷ Don't add noise to last forward posterior.
    end if
    $\boldsymbol{z}_{t-1} \leftarrow \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{z}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{g_\theta}(\boldsymbol{z}_t, t) \right) + \tilde{\Sigma}_t \boldsymbol{\varepsilon}$.
end for
return $\boldsymbol{z}_0$

Algorithm 4 Sampling from a Multinomial Diffusion Model

Assume T and β_t, $t \in \{1, \dots, T\}$, set a priori.
Assume $\boldsymbol{h_\theta}$ has been trained for a selected **categorical** feature.
$K \leftarrow$ number of levels in categorical feature.
$\boldsymbol{z}_T \leftarrow$ one sample from $\boldsymbol{Z} \sim \text{Categorical}_{\text{OHE}}\left(\frac{1}{K}\mathbb{1}\right)$.
for $t \in \{T, \dots, 1\}$ **do**
    $\hat{\boldsymbol{z}} \leftarrow \boldsymbol{h_\theta}(\boldsymbol{z}_t, t)$.
    $\boldsymbol{z}_{t-1} \leftarrow$ one sample from $\boldsymbol{Z} \sim \text{Categorical}_{\text{OHE}}\left(\boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \hat{\boldsymbol{z}})\right)$.
end for
return $\boldsymbol{z}_0$ ▷ Returns one-hot encoded vector.

through a denoising process. In each iteration, we predict a probability parameter and use it to sample from a categorical distribution to get the new, slightly denoised synthetic sample. Finally, after T iterations, the result is a OHE vector representing a synthetic sample from a pre-selected categorical feature in the dataset $\mathcal{D}$, on which we can, for example, apply an arg max- or softmax-operation to find the most likely category. Notice that Algorithm 4 only produces samples from the same categorical feature in the dataset that the Multinomial diffusion model was originally trained on.

Keep in mind that we usually sample some larger positive number of synthetic data points, $\nu \in \mathbb{R}$, in parallel, and not only one, as shown in Algorithms 3 and 4. We only discuss one sample at a time in the algorithms, for simplicity, since discussions concerning *how* or *if* computations may be optimized or parallelized are outside our scope.

3.6 Tabular Diffusion

Finally, in this section we develop *Tabular diffusion*, a probabilistic diffusion-based model for tabular data. This is done by combining the previously discussed Gaussian and Multinomial diffusion models in a relatively simple way. In fact, this idea was recently implemented by Kotelnikov et al. [67], introducing a model called *TabDDPM*. The authors evaluate their model on a set of benchmarks, against other deep generative models from the literature, specifically TVAE and two GAN-variants. In addition, they use a simple interpolation technique called *SMOTE* [12] as a benchmark. However, two drawbacks of this paper come to mind. First, they do not evaluate their model against more "classical" ML algorithms. We design experiments involving such a model, specifically a decision tree-based model, in Chapter 5, using it as a baseline against Tabular diffusion. Second, they develop their model rather hastily — they skip details on several of its key constituents.

One of our contributions to the literature is thus to extend the background material and theoretical development of the model. Most of the previous material in this thesis creates the backbone for this objective. We dissect their paper and add detailed reasoning behind each specific choice, the way we understand it. Finally, we make some additions where we find it appropriate.

Recall the notation we developed in Section 2.9; we assume a tabular dataset $\mathcal{D}$, which contains observations $\boldsymbol{x} = \{(\boldsymbol{x}^{cont})^T, x^{cat_1}, \dots, x^{cat_C}\}$. Each observation consists of values from N continuous features, comprising $\boldsymbol{x}^{cont} \in \mathbb{R}^{N\times 1}$ and C categorical features, x^{cat_j}, $j \in \{1, \dots, C\}$, with K_j possible levels each. Keeping the two previously introduced variants of diffusion models in mind, a Gaussian diffusion model can plausibly model the continuous features, but not the categorical features. Similarly, each categorical feature can be modelled by Multinomial diffusion, but the continuous features cannot truthfully be modelled using categorical distributions. Thus, the question is if it is possible to combine the two models into one? In certain ways, this is what Tabular diffusion does. In the following, we explain how Tabular diffusion can be used to model $\boldsymbol{X} \coloneqq \{X^1, \dots, X^p, Y\} \sim p^*(\boldsymbol{x})$, where we focus on an arbitrary instance $\boldsymbol{x} \in \mathcal{D}$ for instructional purposes.

Forward process. First of all, we define a Gaussian diffusion process for the continuous features $\boldsymbol{X}^{cont} \in \mathbb{R}^{N\times 1}$. This process destroys the information in the continuous feature values slowly until reaching approximate standard Gaussian noise, as we have seen previously. Then, we define a separate Multinomial diffusion process for each categorical feature X^{cat_j}, since we have seen that only one categorical feature can be modelled at once. Each distinct process destroys the information in each separate set of categorical feature values gradually until reaching approximate uniform probabilities for each category. Thus, in total, Tabular diffusion consists of $C + 1$ forward processes. Our objective is to reverse all these forward processes.

Reverse process. Actually, the reverse process in Tabular diffusion is constructed similarly to in previously studied models. Let $\boldsymbol{f_\theta}$ denote a ML model, typically a neural network, that contains all the learnable parameters in Tabular diffusion. The objective is to train this model to predict the Gaussian noise added to $\boldsymbol{x}^{cont}$, as well as the probability parameters for x^{cat_j}, $j \in \{1, \dots, C\}$, from each diffusion step t and corresponding diffused observation $\boldsymbol{z}_t$, for $t \in \{1, \dots, T\}$.

Mathematical formulation. More precisely, let the hierarchical latent variables, $\{\boldsymbol{Z}_1, \dots, \boldsymbol{Z}_T\}$, consist of continuous and categorical components, i.e. each realization is represented by $\boldsymbol{z}_t = \{(\boldsymbol{z}_t^{cont})^T, \boldsymbol{z}_t^{cat_1}, \dots, \boldsymbol{z}_t^{cat_C}\}$, such that they have the same structure and dimensionality as $\boldsymbol{X}$. We assume that the categorical realizations, x^{cat_j}, are transformed to OHE vectors, $\boldsymbol{x}^{cat_j} \in \mathbb{R}^{K_j\times 1}$, for $j \in \{1, \dots, C\}$, before diffusion is performed. Hence, every categorical latent variable component, $Z_t^{cat_j}$, $t \in \{1, \dots, T\}$, is also represented as a vector in $\mathbb{R}^{K_j\times 1}$, for $j \in \{1, \dots, C\}$. Then, the forward process is given by

$$
\begin{aligned}
q(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T|\boldsymbol{x}) &= q(\boldsymbol{z}_1|\boldsymbol{x}) \prod_{t=2}^{T} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}), \\
q(\boldsymbol{z}_1^{cont}|\boldsymbol{x}^{cont}) &= \mathcal{N}(\boldsymbol{z}_1^{cont}; \sqrt{1-\beta_1}\boldsymbol{x}^{cont}, \beta_1\mathbf{I}), \\
q(\boldsymbol{z}_1^{cat_j}|\boldsymbol{x}^{cat_j}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_1^{cat_j}; (1-\beta_1)\boldsymbol{x}^{cat_j} + \frac{\beta_1}{K_j}\mathbb{1}\right), \quad j \in \{1, \dots, C\}, \\
q(\boldsymbol{z}_t^{cont}|\boldsymbol{z}_{t-1}^{cont}) &= \mathcal{N}(\boldsymbol{z}_t^{cont}; \sqrt{1-\beta_t}\boldsymbol{z}_{t-1}^{cont}, \beta_t\mathbf{I}), \\
q(\boldsymbol{z}_t^{cat_j}|\boldsymbol{z}_{t-1}^{cat_j}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_t^{cat_j}; (1-\beta_t)\boldsymbol{z}_{t-1}^{cat_j} + \frac{\beta_t}{K_j}\mathbb{1}\right), \quad j \in \{1, \dots, C\}.
\end{aligned}
$$

Furthermore, the reverse process is given by

$$
\begin{aligned}
p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_T) &= p(\boldsymbol{z}_T) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1), \\
p(\boldsymbol{z}_T^{cont}) &= \mathcal{N}(\boldsymbol{z}_T^{cont}; \mathbf{0}, \mathbf{I}), \\
p(\boldsymbol{z}_T^{cat_j}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_T^{cat_j}; \frac{1}{K_j}\mathbb{1}\right), && j \in \{1, \ldots, C\},
\end{aligned}
$$

$$
\begin{aligned}
p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}^{cont}|\boldsymbol{z}_t^{cont}) &= \mathcal{N}(\boldsymbol{z}_{t-1}^{cont}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t^{cont}, t), \tilde{\Sigma}_t \mathbf{I}), \\
p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}^{cat_j}|\boldsymbol{z}_t^{cat_j}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_{t-1}^{cat_j}; \boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t^{cat_j}, \boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t))\right), && j \in \{1, \ldots, C\},
\end{aligned}
$$

$$
\begin{aligned}
p_{\boldsymbol{\theta}}(\boldsymbol{x}^{cont}|\boldsymbol{z}_1^{cont}) &= \mathcal{N}(\boldsymbol{x}^{cont}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1^{cont}, 1), a\mathbf{I}), && a \in \mathbb{R}, \\
p_{\boldsymbol{\theta}}(\boldsymbol{x}^{cat_j}|\boldsymbol{z}_1^{cat_j}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{x}^{cat_j}; \boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_1^{cat_j}, 1)\right), && j \in \{1, \ldots, C\},
\end{aligned}
$$

where $\tilde{\Sigma}_t$ is given in Equation (3.13) and $a \in \mathbb{R}$ is arbitrary. Moreover, $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ is given by

$$
\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t^{cont}, t) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{z}_t^{cont} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_t^{cont}, t)\right), \quad t \in \{1, \ldots, T\},
$$

as established in Section 3.3, and $\boldsymbol{\Lambda}_{\text{post}}$ is given by

$$
\begin{aligned}
\boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t^{cat_j}, \boldsymbol{x}^{cat_j}) &= \frac{\boldsymbol{\lambda}}{\|\boldsymbol{\lambda}\|_1}, \quad t \in \{2, \ldots, T\}, \\
\boldsymbol{\lambda} &= \left[\alpha_t \boldsymbol{z}_t^{cat_j} + \frac{1-\alpha_t}{K_j}\mathbb{1}\right] \odot \left[\bar{\alpha}_{t-1}\boldsymbol{x}^{cat_j} + \frac{(1-\bar{\alpha}_{t-1})}{K_j}\mathbb{1}\right],
\end{aligned}
$$

as established in Section 3.4. Notice that we introduce the functions $\boldsymbol{g}_{\boldsymbol{\theta}}$ and $\boldsymbol{h}_{\boldsymbol{\theta}}$ in the equations above. This is done in order to explicitly highlight that different elements of the output of the model $\boldsymbol{f}_{\boldsymbol{\theta}}$ are used in each of the reverse process densities, depending on if they pertain to the continuous or the categorical features. In the following, for simplicity, we assume that $\boldsymbol{f}_{\boldsymbol{\theta}}$ is a neural network. In this context, $\boldsymbol{g}_{\boldsymbol{\theta}}$ represents the first N nodes in the output layer of $\boldsymbol{f}_{\boldsymbol{\theta}}$, which is trained to predict a realization, $\boldsymbol{\varepsilon}$, of $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ in $\boldsymbol{z}_t^{cont} = \sqrt{\bar{\alpha}_t}\boldsymbol{x}^{cont} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}$ at any diffusion step $t \in \{1, \ldots, T\}$. Additionally, $\boldsymbol{h}_{\boldsymbol{\theta}} = \{\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_1}, \ldots, \boldsymbol{h}_{\boldsymbol{\theta}}^{cat_C}\}$ represents the rest of the $\sum_{j=1}^{C} K_j$ nodes in the output layer of $\boldsymbol{f}_{\boldsymbol{\theta}}$. Each component, $\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}$, is trained to predict the probability parameter of x^{cat_j} by applying a softmax function to the corresponding K_j output nodes in $\boldsymbol{f}_{\boldsymbol{\theta}}$. Thus, a prediction can be split between the two functions like

$$
\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) = \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_t^{cont}, t) + \boldsymbol{h}_{\boldsymbol{\theta}}^{cat_1}(\boldsymbol{z}_t^{cat_1}, t) + \cdots + \boldsymbol{h}_{\boldsymbol{\theta}}^{cat_C}(\boldsymbol{z}_t^{cat_C}, t), \quad t \in \{1, \ldots, T\}.
$$

Finally, the loss function that is used in Tabular diffusion is a relatively simple sum,

$$
L^{TD} \coloneqq L_{\text{simple}}^{GD} + \frac{1}{C}\sum_{j=1}^{C} L_j^{MD},
$$

where we weight the sum of the multinomial losses by the reciprocal of the number of categorical features, in order to balance the two terms of the total loss, L^{TD} [67]. To

complete this chapter, Algorithms 5 and 6 show how to train, and sample from, a Tabular diffusion model, respectively.

Training algorithm. The forward process in Algorithm 5 is performed in steps 4 through 22, including extra data-processing steps in between. Then, the reverse process is performed in steps 23 through 33, including calculation of the total loss, L^{TD}, where the parameters of $\boldsymbol{f_\theta}$ are adjusted. The adjustments are performed with back-propagation and a stochastic gradient descent method. We stress that, in theory, the neural network should be able to learn the parameters in the reverse process densities of all the features at the same time, where the first N output nodes are taught to predict the noise, whose predictions are represented by $\boldsymbol{g_\theta}(\boldsymbol{z}_t^{cont}, t)$, and the rest of the output nodes learn to predict the probability parameters, whose predictions are represented by $\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t)$, $j \in \{1, \dots, C\}$. Notice that the predictions $\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t)$ from step 21 are used when calculating the losses in the if-else statement in steps 25 through 29, though not included explicitly to avoid excess clutter. Finally, note that in Sections 3.4 and 3.5, we assumed that a softmax activation function was applied directly in the output layer of $\boldsymbol{h_\theta}$, whereas here we need to apply it explicitly to $\boldsymbol{h_\theta}$ after making the prediction, since we manually arrange the output nodes in $\boldsymbol{f_\theta}$ depending on the features they are applied to. This is one way of solving this problem, but we specify this difference explicitly to avoid confusion between Algorithms 2 and 5.

Sampling algorithm. Sampling from a previously trained Tabular diffusion model can be performed according to Algorithm 6. In this case, similar comments to the ones added in Section 3.5 can be made. In general, we start generation by sampling from $\boldsymbol{Z} \sim p(\boldsymbol{z}_T)$. For the continuous features this amounts to sampling from a standard Gaussian and for each categorical feature j it amounts to sampling from a categorical distribution with equal probability of sampling each of the K_j categories. Then, these samples are run iteratively through the decoding process. In each step $t \in \{T, \dots, 1\}$, we first concatenate the samples into one single vector $\boldsymbol{z}_t$. Then, we make a prediction based on $\boldsymbol{z}_t$ and t from the estimated $\boldsymbol{f_\theta}$, and split the prediction into its corresponding continuous and categorical parts, before modifying the corresponding parts as derived in this chapter. In the end, after T iterations, we return a concatenation of $\boldsymbol{z}_0^{cont}$ and $\boldsymbol{z}_0^{cat_j}$, $j \in \{1, \dots, C\}$, which can then be further processed. Finally, we reiterate that training is usually done on batches of observations from $\mathcal{D}$ and we usually sample more than one observation at once.

Algorithm 5 Training a Tabular Diffusion Model

Assume T and β_t, $t \in \{1, \dots, T\}$, set a priori.
Assume K_j, $j \in \{1, \dots, C\}$, known.
while not converged **do**
  $\boldsymbol{x} \leftarrow$ one sample from $\mathcal{D}$ (equivalently from $\boldsymbol{X} \sim p^*(\boldsymbol{x})$).
  $\boldsymbol{x}^{cont} \leftarrow$ select N elements in $\boldsymbol{x}$ corresponding to the continuous features.
  for $j \in \{1, \dots, C\}$ **do**
    $x^{cat_j} \leftarrow$ select K_j elements in $\boldsymbol{x}$ corresponding to categorical feature j.
    $\boldsymbol{x}^{cat_j} \leftarrow$ one-hot encode x^{cat_j}.
  end for
  $t \leftarrow$ one sample from $\mathcal{T} \sim \text{Uniform}[1, T]$.
  $\boldsymbol{\varepsilon} \leftarrow$ one sample from $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
  $\boldsymbol{z}_t^{cont} \leftarrow \sqrt{\bar{\alpha}_t}\boldsymbol{x}^{cont} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}$.
  for $j \in \{1, \dots, C\}$ **do**
    $\boldsymbol{z}_t^{cat_j} \leftarrow$ one sample from $\boldsymbol{Z} \sim \text{Categorical}_{\text{OHE}}\left(\bar{\alpha}_t \boldsymbol{x}^{cat_j} + \frac{1-\bar{\alpha}_t}{K_j}\mathbb{1}\right)$.
  end for
  $\boldsymbol{z}_t \leftarrow \{(\boldsymbol{z}_t^{cont})^T, \boldsymbol{z}_t^{cat_1}, \dots, \boldsymbol{z}_t^{cat_C}\}$.
  Calculate prediction $\boldsymbol{f_\theta}(\boldsymbol{z}_t, t)$.
  $\boldsymbol{g_\theta}(\boldsymbol{z}_t^{cont}, t) \leftarrow$ select first N components of $\boldsymbol{f_\theta}(\boldsymbol{z}_t, t)$.
  for $j \in \{1, \dots, C\}$ **do**
    $\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t) \leftarrow$ select each successive, consecutive set of K_j values remaining in $\boldsymbol{f_\theta}(\boldsymbol{z}_t, t)$ after the first N.
    $\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t) \leftarrow \boldsymbol{\sigma}_{\text{soft}}(\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t))$.
  end for
  $L_{\text{simple}}^{GD} \leftarrow \|\boldsymbol{\varepsilon} - \boldsymbol{g_\theta}(\boldsymbol{z}_t^{cont}, t)\|_2^2$.
  for $j \in \{1, \dots, C\}$ **do**
    if $t = 1$ **then**
      $L_j^{MD} \leftarrow \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{cat_j} | \boldsymbol{z}_1^{cat_j})$.
    else
      $L_j^{MD} \leftarrow D_{KL}(q(\boldsymbol{z}_{t-1}^{cat_j} | \boldsymbol{z}_t^{cat_j}, \boldsymbol{x}^{cat_j}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}^{cat_j} | \boldsymbol{z}_t^{cat_j}))$.
    end if
  end for
  $L^{TD} \leftarrow L_{\text{simple}}^{GD} + \frac{1}{C}\sum_{j=1}^{C} L_j^{MD}$
  Calculate $\boldsymbol{\nabla_\theta} L^{TD}$ via backpropagation.
  Take gradient descent step.
end while

Algorithm 6 Sampling from a Tabular Diffusion Model

Assume T and β_t, $t \in \{1, \dots, T\}$, set a priori.
Assume $\boldsymbol{f_\theta}$ has been trained.
Assume K_j, $j \in \{1, \dots, C\}$, known.
$\boldsymbol{z}_T^{cont} \leftarrow$ one sample from $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
for $j \in \{1, \dots, C\}$ **do**
    $\boldsymbol{z}_T^{cat_j} \leftarrow$ one sample from $\boldsymbol{Z} \sim \text{Categorical}_{\text{OHE}}\left(\frac{1}{K_j}\mathbb{1}\right)$.
end for
for $t \in \{T, \dots, 1\}$ **do**
    $\boldsymbol{z}_t \leftarrow \{(\boldsymbol{z}_t^{cont})^T, \boldsymbol{z}_t^{cat_1}, \dots, \boldsymbol{z}_t^{cat_C}\}$.
    Calculate prediction $\boldsymbol{f_\theta}(\boldsymbol{z}_t, t)$.
    $\boldsymbol{g_\theta}(\boldsymbol{z}_t^{cont}, t) \leftarrow$ select first N components of $\boldsymbol{f_\theta}(\boldsymbol{z}_t, t)$.
    for $j \in \{1, \dots, C\}$ **do**
        $\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t) \leftarrow$ select each successive, consecutive set of K_j values remaining in $\boldsymbol{f_\theta}(\boldsymbol{z}_t, t)$ after the first N.
        $\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t) \leftarrow \boldsymbol{\sigma}_{\text{soft}}(\boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t))$.
    end for
    if $t > 1$ **then**
        $\boldsymbol{\varepsilon} \leftarrow$ one sample from $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
    else
        $\boldsymbol{\varepsilon} \leftarrow \mathbf{0}$.
    end if
    $\boldsymbol{z}_{t-1}^{cont} \leftarrow \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{z}_t^{cont} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{g_\theta}(\boldsymbol{z}_t^{cont}, t)\right) + \tilde{\Sigma}_t\boldsymbol{\varepsilon}$.
    for $j \in \{1, \dots, C\}$ **do**
        $\hat{\boldsymbol{z}}^{cat_j} \leftarrow \boldsymbol{h}_{\boldsymbol{\theta}}^{cat_j}(\boldsymbol{z}_t^{cat_j}, t)$.
        $\boldsymbol{z}_{t-1}^{cat_j} \leftarrow$ one sample from $\boldsymbol{Z} \sim \text{Categorical}_{\text{OHE}}\left(\boldsymbol{\Lambda}_{\text{post}}(\boldsymbol{z}_t, \hat{\boldsymbol{z}}^{cat_j})\right)$.
    end for
end for
return $\boldsymbol{z}_0 = \{(\boldsymbol{z}_0^{cont})^T, \boldsymbol{z}_0^{cat_1}, \dots, \boldsymbol{z}_0^{cat_C}\}$.

Chapter 4

Generating Counterfactuals

This chapter is devoted to discussing one particular strategy for generating counterfactual explanations. In general, recall from Section 2.3.1 that we group the techniques for calculating counterfactuals into two main groups; algorithmic-based methods and on-manifold methods. In this thesis, we focus on the latter approach. More specifically, we take inspiration from one particular recently introduced method from this paradigm, which facilitates modifications with generative models.

The chapter is structured as follows. Section 4.1 is devoted to explaining a novel on-manifold method for generating counterfactuals called *MCCE: Monte Carlo Sampling of Realistic Counterfactual Explanations*, introduced by Redelmeier et al. [107]. We note that this section is a largely rewritten version of a chapter contained in the author's specialization project [93]. Then, Section 4.2 outlines a modification of this method, where we introduce a different generative model, more specifically Tabular diffusion, as discussed in the previous chapter. Notice that we investigated a modification of MCCE in our project [93] as well, where we relied on a simple VAE as a generative model, but we have chosen not to address this specific variant here. However, as is explained in Chapter 5, we use a modification based on TVAE, as outlined in Section 2.9, as a baseline in our experiments.

4.1 MCCE

The main innovation in Redelmeier et al. [107] is two-fold. First, a simple process consisting of three independent steps is introduced, making the method highly modular. Second, MCCE relies on decision trees to model the underlying data distribution, which alleviates some of the disadvantages of other on-manifold methods. MCCE guarantees generation of actionable counterfactuals, which is one of the main reasons why the authors claim it is an improvement over competing methods, like, e.g., CRUDS [24], REVISE [58] or C-CHVAE [100]. Additionally, the method handles categorical features with more than two levels and is not restricted to explaining certain types of models.

Succinctly put, MCCE consists of three independent steps:

1. *Distribution modelling*: The underlying data distribution of the *mutable* features given the *fixed* features is modelled.

2. *Generation*: A large set of random samples is generated from the model in step 1.

3. *Post-processing*: Samples that do not obey the counterfactual criteria are removed from the set of samples in step 2.

In the following, we dive deeper into each of these steps.

Distribution Modelling

As stated, the authors make use of decision trees to model the underlying data distribution. However, recall from Section 2.5 that decision trees are common examples of discriminative models, which generally do not model the joint distribution of $\boldsymbol{X} \coloneqq \{X^1, \ldots, X^p\}$, represented by the density $p^*(\boldsymbol{x})$. Despite this, the authors model the joint distribution *indirectly*, by combining discriminative models, particularly decision trees, in a specific way. The chain rule for random variables, as stated in Equation (3.2), is a crucial part of this technique.

Assume a dataset $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, where each observation is a realization of $\boldsymbol{X} \sim p^*(\boldsymbol{x})$. Recall the *actionability* constraint for counterfactuals; the immutable feature values in the input instances should be preserved in their corresponding counterfactual explanations. To quantify this, we assume that the random variables are split into two groups. Let $\boldsymbol{X} = \boldsymbol{X}_m \cup \boldsymbol{X}_f$, such that $\boldsymbol{X}_m \cap \boldsymbol{X}_f = \emptyset$, where

$$\boldsymbol{X}_f \coloneqq \{X_f^1, \ldots, X_f^u\},$$

and

$$\boldsymbol{X}_m \coloneqq \{X_m^1, \ldots, X_m^q\},$$

denote fixed features and mutable features, respectively. Notice that $p = u + q$, where u denotes the number of fixed features and q denotes the number of mutable features. Since the fixed features are immutable, Redelmeier et al. [107] chose to model the underlying joint distribution of the q mutable features *conditional* to the u fixed features, a distribution that can be represented by the density $p^*(\boldsymbol{x}_m|\boldsymbol{x}_f)$. The chain rule for random variables lets us rewrite the density as

$$\begin{aligned} p^*(\boldsymbol{x}_m|\boldsymbol{x}_f) &= p^*(x_m^1, \ldots, x_m^q|\boldsymbol{x}_f) \\ &= p^*(x_m^1|\boldsymbol{x}_f) \prod_{j=2}^{q} p^*(x_m^j|\boldsymbol{x}_f, x_m^1, \ldots, x_m^{j-1}). \end{aligned}$$

In practice, the factorization into a product of conditional distributions enables modelling one dependent variable at a time, which is typically a much easier problem to solve compared to modelling multivariate distributions directly [107]. Notice the analogy to the generative processes we discussed in Section 2.8 and Chapter 3; we imagine a similar generative process in this case, where, instead of latent variables, we condition on fixed features, as well as previously modelled mutable features, in each step. However, it is important to notice that $p^*(\boldsymbol{x})$ is not modelled in MCCE, in contrast to the previously discussed generative models, precisely because of the actionability constraint. Despite this, note that the underlying joint distribution of all the features can be calculated by using the property $p^*(\boldsymbol{x}) = p^*(\boldsymbol{x}_m|\boldsymbol{x}_f)p^*(\boldsymbol{x}_f)$, where the underlying joint distribution of the fixed features is modelled separately. This is something we further explore in Chapter 5.

Specifically, each of the conditional distributions is learnt by a decision tree in MCCE. To clarify, let T_j, $j \in \{1, \ldots, q\}$, denote a set of decision trees that are trained to model $p^*(\boldsymbol{x}_m|\boldsymbol{x}_f)$. The trees are fitted like

$$
\begin{aligned}
T_1: &\ \boldsymbol{x}_m^1 \sim (\boldsymbol{x}_f^1, \ldots, \boldsymbol{x}_f^u), \\
T_2: &\ \boldsymbol{x}_m^2 \sim (\boldsymbol{x}_f^1, \ldots, \boldsymbol{x}_f^u, \boldsymbol{x}_m^1), \\
T_3: &\ \boldsymbol{x}_m^3 \sim (\boldsymbol{x}_f^1, \ldots, \boldsymbol{x}_f^u, \boldsymbol{x}_m^1, \boldsymbol{x}_m^2), \\
&\ \vdots \\
T_q: &\ \boldsymbol{x}_m^q \sim (\boldsymbol{x}_f^1, \ldots, \boldsymbol{x}_f^u, \boldsymbol{x}_m^1, \boldsymbol{x}_m^2, \boldsymbol{x}_m^3, \ldots, \boldsymbol{x}_m^{q-1}),
\end{aligned}
\tag{4.1}
$$

where $\boldsymbol{x}_i^j$, $i \in \{m, f\}, j \in \{1, \ldots, \max(q, u)\}$, denotes a vector of all the values in a specific column in the training data. Thus, we stress that the trees are fitted with all available observations at training time, like any other supervised learning algorithm. In addition, the abuse of notation $\boldsymbol{a} \sim (\boldsymbol{b}, \boldsymbol{c})$, inspired by notation in Redelmeier et al. [107], symbolizes that a tree is fitted to the response values $\boldsymbol{a}$ based on feature values $\boldsymbol{b}$ and $\boldsymbol{c}$. We reiterate that this process is a concrete example of how a discriminative model can be used to construct a generative model; each of the trees are discriminative models, for either classification or regression, which are combined to model a joint distribution represented by the density $p^*(\boldsymbol{x}_m|\boldsymbol{x}_f)$.

Before moving on, as mentioned in Section 2.5, there exists several different algorithms for fitting decision trees. Redelmeier et al. [107] chose CART for calculating the trees, based on a specific set of beneficial characteristics of this algorithm. Essentially, they chose CART over other non-parametric models because "CART is consistent and allows for high dimensionality" [107]. Moreover, the authors refer to Drechsler and Reiter [25], and Reiter [108], which are instances where CART has performed well on generative tasks.

Generation

Once the parameters have been estimated, the generation step is relatively simple. A large number, K, of data points is sampled from the tree-based model that represents the density $p^*(\boldsymbol{x}_m|\boldsymbol{x}_f)$, for each relevant set of fixed feature values. These sets of fixed feature values are based on the instances we want to calculate explanations for. This is formalized later. Given that the tree-model is able to approximate the underlying data distribution in a truthful manner, the likelihood of the empirical distribution of the mutable features given the fixed features, represented by $p(\boldsymbol{x}_m|\boldsymbol{x}_f)$, in the training data, should be large. K should be large in order to ensure that successful counterfactuals are produced after the post-processing step. The authors do not define exactly how large K should be, but we believe it should follow a heuristic along the lines of "the larger the better, while taking available computing resources and time into account".

Specifically, generation of synthetic samples is done by sampling recursively from the trees in order of increasing number of conditional features. Suppose q decision trees have been fitted, as illustrated in Equation (4.1). Let $\mathcal{H}$ be a set of observations with undesirable predictions. Recall that we restrict our discussion to binary classification, meaning that an undesirable prediction is a prediction with binary indicator 0. As is common in parts of the literature, we refer to $\mathcal{H}$ as a set of *factuals*. Furthermore, let $\mathbf{D}_{\boldsymbol{h}} \in \mathbb{R}^{K \times p}$ denote a set of generated observations for each $\boldsymbol{h} \in \mathcal{H}$. Precisely, recall that $p = u + q$, since we leave the response variable, Y, out of the modelling in step 1 when constructing explanations for prediction models. Circling back to Example 2.3.2, $\mathcal{H}$ would be the set of all clients getting their mortgage solicitation rejected, $\boldsymbol{h} \in \mathcal{H}$ would be a specific client vector among these (e.g., the one that was highlighted) and $\mathbf{D}_{\boldsymbol{h}}$ would be a generated set of K synthetic customer vectors, with fixed feature values identical to the ones in $\boldsymbol{h}$ and mutable feature values "close" to the ones in $\boldsymbol{h}$. Notice that this "closeness" is what the post-processing deals with ensuring.

Algorithm 7 outlines the procedure for producing $\mathbf{D}_{\boldsymbol{h}}$ for one $\boldsymbol{h} \in \mathcal{H}$ [107]. The idea is to first define the matrix $\mathbf{D}_{\boldsymbol{h}}$ with K copies of the factual $\boldsymbol{h}$, in such a way that the fixed feature columns are placed to the *left* of the mutable feature columns. This is the reason why the first step in Algorithm 7 explicitly states that $\mathbf{D}_{\boldsymbol{h}}$ should be a matrix of dimension $K \times (u+q)$, instead of simply writing $K \times p$. This structure is necessary in our version of the algorithm, in order for the indexing in the rest of the algorithm to make sense. Then, we proceed from row $i = 1$ to $i = K$ through one mutable feature column $j \in \{1, \ldots, q\}$, at a time, overwriting each cell iteratively. Precisely, each cell is filled by first finding the leaf node of tree T_j based on input data consisting of the fixed feature values and the previously filled values in $\mathbf{D}_{\boldsymbol{h}}$ from trees $T_{j-1}, \ldots, T_1$, for $j \geq 2$. Thus, the inputs to each tree are simply all the values in the preceding columns, in the same row i. Then, after finding the leaf node, each cell is filled by sampling from the set of response values of the training observations classified to the leaf node. Tying this back to Section 2.5, this is an example where the conditional density in each leaf node is roughly estimated by random sampling from the set of training response values belonging to the node. For extra clarity, notice that indexing is assumed to start at 1 in Algorithm 7, whereas many programming languages have a starting index of 0.

Algorithm 7 Generation (Step 2) in MCCE per $\boldsymbol{h} \in \mathcal{H}$

$\mathbf{D}_{\boldsymbol{h}} \leftarrow \text{matrix}(K \times (u+q))$ ▷ each row is a copy of the factual $\boldsymbol{h}$.
for $1 \leq j \leq q$ **do**
  for $1 \leq i \leq K$ **do**
    Find leaf node of tree T_j based on vector $\mathbf{D}_{\boldsymbol{h}}[i, 1 : (u+j-1)]$.
    $\mathbf{D}_{\boldsymbol{h}}[i, u+j] \leftarrow$ one sample from the set of response values
      of the training observations
      belonging to the leaf node.
  end for
end for
return $\mathbf{D}_{\boldsymbol{h}}$

Post-processing

The post-processing step is important for ensuring that the generated samples in $\mathbf{D}_{\boldsymbol{h}}$ fulfill the criteria of counterfactuals $\forall \boldsymbol{h} \in \mathcal{H}$. In this final step of MCCE, rows that do not obey the criteria from Section 2.3.1 are removed from each $\mathbf{D}_{\boldsymbol{h}}$, such that the remaining rows can be used as counterfactuals. The procedure varies slightly depending on if we want to generate several, or only one, counterfactual(s) per factual. Notice that criteria 1 (*on-manifold*) and 2 (*actionable*) are automatically fulfilled following the first step of MCCE, since "the observations come from an approximation to the data distribution conditioned on the fixed features" [107]. Thus, for the first criterion we simply assume that the generative model is able to truthfully model the true underlying data distribution, an assumption that is challenged in the experiments outlined in Chapter 5. In order to fulfill criterion 3 (*valid*), rows of $\mathbf{D}_{\boldsymbol{h}}$ which do not yield a desirable prediction, which necessarily is different from the prediction on $\boldsymbol{h}$, are removed. In binary classification, this means that the rows i of $\mathbf{D}_{\boldsymbol{h}}$ where $f(\mathbf{D}_{\boldsymbol{h}}[i,]) < c$, are removed, where $f(\cdot)$ is a binary classifier and c is a discrimination threshold at which binary indicator 0 is split from 1. Tying this to Example 2.3.2, every row in $\mathbf{D}_{\boldsymbol{h}}$ that does not yield a granted mortgage prediction is removed. Thus, after this filtering, we have ensured that the remaining rows in $\mathbf{D}_{\boldsymbol{h}}$ have the same fixed feature values as the factual, but with a positive predicted outcome. Criterion 4 (*low cost*) relates to the previously mentioned "closeness" between $\boldsymbol{h}$ and the generated observations in $\mathbf{D}_{\boldsymbol{h}}$. This is also where we decide if we want to produce one

or more counterfactuals per $\boldsymbol{h}$. The authors of MCCE chose to calculate the quantities *sparsity* and *Gower distance* between $\boldsymbol{h}$ and each remaining row of $\mathbf{D}_h$. These are used as tools to satisfy criterion 4. The sparsity of row $\boldsymbol{d}$ of $\mathbf{D}_{\boldsymbol{h}}$ is simply equal to the number of feature values that are different in $\boldsymbol{d}$ and $\boldsymbol{h}$. Furthermore, the Gower distance between factual $\boldsymbol{h} = \{h^1, \dots, h^p\}$ and row $\boldsymbol{d} = \{d^1, \dots, d^p\}$ of $\mathbf{D}_{\boldsymbol{h}}$ is defined as

$$G(\boldsymbol{h}, \boldsymbol{d}) \coloneqq \frac{1}{p} \sum_{j=1}^{p} \delta_G(h^j, d^j) \in [0, 1],$$

where

$$\delta_G(h^j, d^j) \coloneqq \begin{cases} \frac{1}{R_j}|h^j - d^j| & \text{if } h^j \text{ is continuous,} \\ \mathbb{1}_{h^j \neq d^j} & \text{if } h^j \text{ is categorical,} \end{cases}$$

where $\mathbb{1}_{h^j \neq d^j}$ is the indicator function,

$$\mathbb{1}_{x \neq y} \coloneqq \begin{cases} 1 & \text{if } x \neq y, \\ 0 & \text{otherwise,} \end{cases}$$

and R_j normalizes the jth continuous feature values such that they lie between 0 and 1. The value R_j, for any column j pertaining to a continuous feature, is not explicitly defined by Redelmeier et al. [107]. However, the pre-processing steps the authors follow in their open source implementation[1] indicate that they use the empirical *range* of each continuous feature in the entire dataset $\mathcal{D}$. Precisely, $R_j = |\max \boldsymbol{x}^j - \min \boldsymbol{x}^j|$, where $\boldsymbol{x}^j$ denotes a vector of all values in continuous feature column j in the entire dataset. In this manner, the distance in $G(\cdot, \cdot)$ between two values indexed by continuous feature j is constrained to $[0, 1]$. Again, an important assumption is that the distribution of the data is well-modelled in step 1 in MCCE, such that the distance $|h^j - d^j|$ does not exceed R_j for any continuous feature j.

If we choose to return only one counterfactual per $\boldsymbol{h} \in \mathcal{H}$, the process for ensuring fulfilment of criterion 4 has two steps. First, the minimum sparsity across the rows in $\mathbf{D}_{\boldsymbol{h}}$ is calculated, and the rows with larger sparsity than this minimum value are removed. Next, the row with the smallest Gower distance to $\boldsymbol{h}$ among the remaining rows is the final counterfactual for $\boldsymbol{h}$.

If we instead choose to return a set of counterfactuals per $\boldsymbol{h} \in \mathcal{H}$, the process for ensuring fulfilment of criterion 4 is slightly different. One suggestion by Redelmeier et al. [107] is to set upper bounds for sparsity and Gower distance across the rows in $\mathbf{D}_{\boldsymbol{h}}$, and remove rows with values exceeding these bounds. Another suggestion is to return a set of counterfactuals which are not close to each other. In this way, the recipient of the counterfactuals gets a certain variety in the explanations, which might be reasonable in some domains. The final suggestion they mention is to return the "Pareto Front of valid samples in $\mathbf{D}_{\boldsymbol{h}}$" [107], which we do not discuss further.

4.2 Diff-MCCE

In this section, we propose an alteration of MCCE. As previously covered, the first two steps of the three-step MCCE algorithm are solved using conditional probability modelling in combination with decision trees [107]. The three-step procedure is especially attractive due to its high degree of modularity, in particular between the two first steps and the third step. We propose a modification where Tabular diffusion, as developed in Chapter

[1]The open source implementation of MCCE can be found at `https://github.com/NorskRegnesentral/mccepy`. Note that we modify this implementation to fit our needs, as explained in Chapter 5.

3, is used for distribution modelling and generation. As a consequence, some slight, but necessary, modifications are introduced in the post-processing step, but the main ideas from MCCE remain unchanged. For simplicity, we call this modification *Diff-MCCE*. In the following, we discuss each of the steps this modification consists of.

Distribution Modelling

First of all, we use Tabular diffusion to estimate the joint distribution of $\boldsymbol{X} \coloneqq \{X^1, \ldots, X^p\} \sim p^*(\boldsymbol{x})$ directly. As in Section 4.1, we do not model Y alongside the rest of the features in the dataset, because it is not necessary when constructing explanations for prediction models with the three-step method. In particular, in reference to the notation from Sections 2.9 and 3.6, $p = N + C$. In contrast to MCCE, we cannot model any conditional distribution, like the one represented by the density $p^*(\boldsymbol{x}_m|\boldsymbol{x}_f)$, because our relatively simple Tabular diffusion model is not capable of modelling such a complex distribution in general. This means that the concept of fixed and mutable features is irrelevant in the first two steps of Diff-MCCE. Note that Tabular diffusion can be extended to allow for conditional modelling. For example, such conditioning may be implemented using a technique called *guidance*, in the form of classifier guidance [20] or classifier-free guidance [48]. Unfortunately, because of lack of time, we have not discussed this in detail, meaning that such an addition is left as further work.

Generation

Once the diffusion model is fitted, sampling is done following Algorithm 6. Recall from the previous section that we sample K realizations from the model of the conditional distribution, represented by $p^*(\boldsymbol{x}_m|\boldsymbol{x}_f)$, for each factual in $\mathcal{H}$. Thus, the total number of generated points are $K \cdot |\mathcal{H}|$, where $|\mathcal{H}|$ represents the number of factuals in $\mathcal{H}$. Hence, in order to stay consistent with the method from Section 4.1, we sample a total of $K \cdot |\mathcal{H}|$ synthetic data points from the trained Tabular diffusion model, using Algorithm 6. In practice, this can done with relative ease, because of efficient implementations of linear algebraic computations with matrices and tensors.

Post-processing

After generating data from the estimated underlying distribution, post-processing is performed. In Diff-MCCE, the post-processing is slightly more involved, because *actionability* is not inherently fulfilled after non-conditional sampling. First of all, the *on-manifold* criterion should be satisfied. Of course, we cannot be certain that our fitted Tabular diffusion model has modelled the underlying data distribution truthfully. As a consequence, we cannot be certain that samples from the model are samples from $p^*(\boldsymbol{x})$. However, if the likelihood of our data under the model is large, we assume that the approximation of $p^*(\boldsymbol{x})$ is good. Within this statement is a discussion about exactly *what* a *large* likelihood really is, but we skip those details. Thus, just as in MCCE, we simply assume that the trained generative model is able to truthfully model the underlying data distribution, an assumption that we challenge in the experiments outlined in Chapter 5. Next, we need to satisfy criterion 2 (*actionable*) of the counterfactuals. This is not very demanding — we simply remove every observation in $\mathbf{D}_{\boldsymbol{h}}$ which does not have the same fixed covariate values as h. Finally, criteria 3 (*valid*) and 4 (*low cost*) are satisfied by following the same procedure as in Section 4.1.

Thus, we have seen how easily MCCE can be modified to include any other generative model. In the next chapter, we design some experiments to evaluate these models in practice.

Chapter 5

Experiments

In this chapter, we design an evaluation procedure for Diff-MCCE. This facilitates extensive evaluation of the method, which is one of our main contributions. In order to provide a detailed account of its strengths and weaknesses, we construct two different experiments. The first experiment is devoted to assessing the first two steps; population modelling and generation of synthetic data. These two steps have to be evaluated in conjunction, as it is difficult to get an impression of the correctness of the approximated underlying distribution without sampling data from it. Succinctly stated, in this experiment we compare the performance of Diff-MCCE with respect to these two steps to the performance of two carefully selected baselines. The first baseline is the tree-based model used in MCCE, as outlined in Section 4.1. The second baseline is TVAE, as discussed in Section 2.9. To the best of our knowledge, this is the leading VAE-based deep generative model for tabular data, with open source code. Results from the first experiment are evaluated in Chapter 6. Subsequently, the second experiment is devoted to investigating the capabilities of Diff-MCCE with regards to generating counterfactuals, which is our main interest. Thus, this experiment evaluates the entire three-step process simultaneously. In order to gain a more nuanced picture of Diff-MCCE's performance, it is compared to its counterpart, MCCE, as well as another modification of MCCE where TVAE is used as generative model. Results from this second experiment are evaluated in Chapter 7.

The chapter is organized as follows. Section 5.1 describes the datasets that we base our experiments on. Then, Section 5.2 gives some general practical information about the experiments. Furthermore, Section 5.3 outlines our chosen model architectures and hyperparameters in each of the methods we evaluate, as well as explains the data pre-processing steps we follow. We discuss the choices we make thoroughly, including some implementation details where necessary. Finally, Sections 5.4 and 5.5 explain how evaluation of Diff-MCCE is performed relative to the baselines in each of the experiments.

5.1 Data

In this section, we briefly describe the datasets we consider during our experiments. For the sake of investigating the applicability of Tabular diffusion and Diff-MCCE in a variety of scenarios, we tried to choose a diverse set of datasets. For simplicity, our starting point was the list of datasets used by Kotelnikov et al. [67]. All these datasets are publicly available, and most have been used in several previous research papers on tabular data applications, which promotes easier and quicker comparison between works. From this list, we chose three datasets of differing sizes, distributions and compositions of features, while restricting our freedom of choice to binary classification datasets. This restriction constrains our study, making it easier to understand and implement, while still being extensible to more complex situations with some effort. Table 5.1 contains a list of the

chosen datasets, describing the number of observations in our train, test and validation sets. In addition, it contains the total number of features corresponding to each dataset, alongside the number of continuous, numerical and categorical features among them. To be precise, we count both integers and floats as numerical features, meaning that all the continuous features, in addition to discrete data types that innately describe numeric phenomena, are included in the number of numerical features. Also, recall that all non-continuous features are counted as categorical. We note that the binary response is not included while counting the number of categorical features. Furthermore, each dataset is given a two-letter abbreviation, which is used to refer to them in the rest of the work. Finally, Appendix D gives an overview of the different features in each dataset, as well as a description of how we pre-processed the data initially. At this point we do not present standard results from exploratory data analysis, because we take a deeper look at marginal distributions, correlations and other metrics in the results in Chapter 6. Notice that we use

Table 5.1: List of datasets we use during the experiments. We report the number of observations in the entire dataset after initial pre-processing, in addition to the number of observations in each of the three datasets after stochastic splitting. The total number of features, alongside the number of continuous, numerical and categorical features, corresponding to each dataset is also given, excluding the binary response.

Code	Name	# Total	# Train	# Test	# Validation	# Feat.	# Cont.	# Num.	# Cat.
AD	Adult Census	45222	36177	4522	4523	13	0	6	13
CH	Churn Modelling	10000	8000	1000	1000	10	2	6	8
DI	Diabetes	768	614	77	77	8	2	8	6

the splits $\{0.8, 0.1, 0.1\}$ for training, testing and validation for all datasets, respectively. This is a simple way of gaining a level of standardization across currently (and future) treated datasets, for reproducibility of the results.

5.2 Experiments — General Information

In order to account for some of the inherent stochasticity and variability when applying our models to data, we aggregate most of the results over five different trials, each initialized with a different (pseudo-)random number seed. The seeds we use in each trial are given in Table 5.2. In Chapters 6 and 7, we report results that summarize the performance of each of the models, on the quantitative metrics described in Section 2.10, over the five trials. Specifically, we report the mean $\pm$ standard error of each given metric over the five trials. Additionally, we produce *box plots*, which display a five number summary over the five trials; minimum, first quartile, median, third quartile and maximum. More on this later. For complete transparency, we use the term *trial* to encompass the entire experimental pipeline of ML; loading and pre-processing the data, estimating model parameters and testing or using the model in practice. Thus, each experiment has essentially been performed five times in its entirety. The reason why we chose five different seeds, is that we believe this number yields a decent trade-off between computational cost and bias of the reported results.

Speaking of trials, the experiments are implemented in Python [140] — a popular open source programming language which is used in many different applications. Specific details concerning libraries, methods and hyperparameters are given when relevant in each experiment, to ensure transparency and reproducibility. For full disclosure, all our implementations[1] are publicly available.

[1]The source code for our models and experiments is available at `https://github.com/alexaoh/tabular-diffusion-for-counterfactuals`

Table 5.2: Five different (pseudo-)random number seeds used for different trials.

	Seed
#1	1234
#2	4500
#3	2018
#4	1999
#5	2023

Before moving on, we comment on the use of testing and validation data while performing our experiments. Recall that a supervised ML model is usually trained on a training dataset, before it is validated and tested on two disjoint, held-out datasets. The validation is often performed iteratively, in order to help guide the training process, while testing is left out until the parameters are estimated. As expected, we follow this design when training supervised models, using the splits shown in Table 5.1. However, generative models are unsupervised models, which changes the training process slightly. Specifically, when estimating their parameters, we do not need a held-out testing dataset for evaluation after completion of model training. Intuitively, this is because they essentially are trained to be able to synthesize data that closely resembles the training data, not to, for instance, discriminate between a number of classes. Thus, we choose to concatenate the validation and testing datasets, before using the concatenation for validation of the generative model during parameter estimation. As a side note, we could have validated and tested each of the trees, T_i, $i \in \{1, \ldots, q\}$, in MCCE in a supervised manner, but we chose not to, because discriminative performance on each of the features is of secondary importance. Moreover, since we are interested in using MCCE as a reference for Diff-MCCE, this type of evaluation is useless, since we cannot calculate similar metrics in the diffusion model.

Another important note is that we do not perform systematic hyperparameter tuning in any of our experiments. Our hyperparameter choices are guided by previous results from other authors, where possible. Specifically, we use the hyperparameter values of the best performing models in Kotelnikov et al. [67] as a starting point for both Tabular diffusion and TVAE, for all three datasets. These values are available in their open source implementation[2]. Then, we compare the achieved performance with our starting point with a few selected sets of hyperparameter values chosen based on intuition, finally selecting the values that seem to perform the best. Thus, we are mindful when choosing the hyperparameter values in each model, but not highly systematic or modern in our approach. Despite our slight neglect, due to limitations in scope and research time, the value of performing proper hyperparameter tuning should not be underestimated for optimizing predictive performance.

5.3 Implementation Details

This section is devoted to explaining how we implement Tabular diffusion and the baseline models in practice, including necessary pre-processing steps on input data and hyperparameter choices. Keep in mind that, our initial interest lies in investigating the performance of Tabular diffusion on modelling, and generating synthetic realizations of, $(\boldsymbol{X}, Y) \coloneqq \{X^1, \ldots, X^p, Y\} \sim p^*(\boldsymbol{x}, y)$. Ultimately, this is interesting because we hypothesize that it is crucial for the counterfactual generating capabilities within the framework of Diff-MCCE. For instance, if the generative model is not capable of truthfully modelling the

[2] `https://github.com/rotot0/tab-ddpm`

underlying data distribution, and sampling from it, the quality of counterfactuals would likely suffer.

5.3.1 Data Pre-processing

The first step of any data analysis is data pre-processing. To what extent a dataset needs pre-processing is highly dependent on the data itself, alongside the statistical or ML model in question, but pre-processing is usually accepted as an integral part of any ML workflow. For instance, in Tabular diffusion and TVAE, we use neural networks as function approximators, which introduces certain requirements on the input data format. Most importantly, neural networks cannot inherently process categorical features when they are represented as strings. In addition, without going into detail, training data for fitting neural networks should be scaled in certain ways to increase convergence rate and performance. For instance, MLPs converge faster in practice if their numerical inputs are standardized, i.e. transformed to approximately zero means and unit variances [71].

In the following, we discuss how we pre-process the *numerical* and *non-numerical* feature values. The reason why we distinguish between numerical and non-numerical in this case, instead of continuous and categorical, is that, in practice, we treat all floats and integers as continuous, and all non-numerical data types as categorical. This is a rather common assumption to make, for simplicity, especially when the integer random variables represent phenomena with large numbers of possible outcomes. Intuitively, the larger the number of possible discrete outcomes, the closer an integer feature is to behave like a continuous random variable. In fact, we observe in Appendix D that the majority of the integer features corresponding to the data we want to model contain relatively large sets of possible outcomes. Another consequence of large discrete sample spaces is that the standard treatment of categorical features, i.e. one-hot encoding, becomes computationally troublesome. The practical implications, implementation-wise, of our assumptions are quite simple; any time we, for instance, fit a generative model to a tabular dataset or construct a supervised classifier, all non-continuous numerical features are treated as if they were continuous, alongside the continuous features, whereas the rest of the categorical features are treated as expected. Note that the basic pre-processing mentioned in Appendix D is performed before the pre-processing we discuss here.

Non-numerical features. As considered several times already, we simply one-hot encode (OHE) the non-numerical feature values in the data, representing them as numerical vectors that can be used as input to neural networks. To be specific, the OHE scheme we use throughout the thesis does not yield full-rank *design matrices*, like those we usually strive for in some standard statistical methods. For example, in linear regression, the design matrix is invertible only if the matrix is full rank, i.e. all row- and column-vectors are linearly independent. This property is necessary for calculating the closed form solution of the parameters in linear regression. Full rank is often accomplished by dropping one of the vectors in the OHE, for example in *dummy coding*, where one category is set as a reference level. However, in methods where we use approximate solvers, like neural networks, we do not care about linear dependence between OHE columns in the data matrix. Thus, we follow the convention in the ML literature of working with rank-deficient data matrices.

A method to OHE any feature is not difficult to implement manually. However, since there are open source options for performing this transformation, we use code that many researchers and practitioners have used before us. More specifically, we use the `OneHotEncoder` method from *Scikit-learn*'s [101] pre-processing module on the non-numerical feature values in each of our datasets. Precisely, we OHE the non-numerical feature values in the entire dataset, before splitting. Exactly this order of operations should be followed to ensure that all the different categories in each of the columns are represented in the OHE dataset.

Numerical features. The numerical feature values in the training data are transformed with a Gaussian *Quantile transformer*, when used to train Tabular diffusion and MCCE. We do not apply this transformation to the numerical feature values in the training data for TVAE, due to the introduction of the specialized mode-specific normalization scheme. The Gaussian Quantile transformer implements a non-linear transformation that uses quantile information in the dataset to put all feature values into a desired distribution. Here, our target distribution is a standard Gaussian. We argue that this is a good choice for two reasons. First, because the continuous features in Tabular diffusion are assumed to follow a Gaussian. Second, we aim at mapping to a *standard* Gaussian to improve convergence in MLP training [71]. Precisely, the transformation is applied to each feature, X^j, independently. It is based on the formula $G^{-1}(F(X^j))$, where F is the *cumulative distribution function* (CDF) of X^j and G^{-1} is the *quantile function* of the standard Gaussian. In practice, the CDF of X^j is estimated using all available realizations of X^j in the training dataset. The subsequent application of the approximate CDF on these values maps them to an approximate Uniform$[0, 1]$ distribution. This is possible based on the well-known result that $F(X)$ is uniformly distributed in $[0, 1]$, when F is the CDF of X, for any continuous random variable X. Finally, the standard Gaussian quantile function, G^{-1}, is applied to the approximately uniformly distributed values, to map them to a standard Gaussian distribution. Before moving on, because we assume that it is reasonable to treat integers as continuous values in practice, we apply the Gaussian Quantile transformer and the mode-specific normalization scheme (in TVAE) not only to floats, but also to integers.

We implement the Quantile transformer using the `QuantileTransformer` method from Scikit-learn's [101] pre-processing module. To be specific, we first perform this transformation on the training data. Then, the quantile information from the training data is used to perform the transformations on the held-out testing and validation datasets. Be wary of the fact that we perform the transformations on each of the three splits separately, using only the information from the training dataset in each case. This is very important to avoid *data leakage* [59] — a phenomenon where the real-world performance of a trained model is overestimated during testing due to illegitimate information in the training set, giving it unrealistic power. There are many manifestations of this phenomenon, but one instance occurs if we perform the Quantile transformation on the entire dataset at once. In such a case, we introduce information about the quantiles from more data than what we have available in the training dataset alone, leaving the purpose of splitting the data into several disjoint sets compromised. This is a research field in itself, which we do not dive deeper into, but we have tried to avoid such subtle mistakes in our experiments. The interested reader is referred to Kaufman et al. [59] for a more detailed overview of the topic.

5.3.2 Model Architectures and Hyperparameters

Some of the most important choices we make, are the model architectures and hyperparameters in Tabular diffusion, as well as in the baselines. For utmost clarity, we recall that $(\boldsymbol{X}, Y) = \{X^1, \ldots, X^p, Y\} \sim p^*(\boldsymbol{x}, y)$, meaning that we consider Y alongside the rest of the features, such that $p^*(\boldsymbol{x}, y)$ represents a distribution of dimension $p + 1$. In addition, we define $\tilde{\boldsymbol{X}} \coloneqq (\boldsymbol{X}, Y)$, which we use to simplify the notation in certain cases, to be consistent with previous expositions of VAEs and diffusion models in Chapters 2 and 3, respectively, without sacrificing clarity. We start by describing our choices in Tabular diffusion, before explaining TVAE and the tree-based model in MCCE.

5.3.3 Tabular Diffusion

Recall that the parameters in Tabular diffusion are all contained in the function $\boldsymbol{f_\theta}$. As already stated, we let this function represent a neural network. We hypothesize that the architecture of this neural network is vital for the performance of Tabular diffusion. Here, we explain our choice of architecture. In order to simplify its representation, we define

$$\texttt{MLPBlock}_{u\to v}(\boldsymbol{x}) = \texttt{Dropout}(\texttt{ReLU}(\texttt{FC}_{u\to v}(\boldsymbol{x})),$$

for an arbitrary vector $\boldsymbol{x} \in \mathbb{R}^{u\times 1}$. Actually, we evaluate the performance of Tabular diffusion based on two different techniques for modelling, and sampling from, $p^*(\boldsymbol{x}, y)$. The two techniques, which we refer to as *direct joint modelling* and *conditional modelling*, are explained in the following.

Direct Joint Modelling

In the first technique, we model the joint underlying data density, $p^*(\boldsymbol{x}, y)$, directly. Precisely, we learn the reverse process

$$p_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) = p(\boldsymbol{z}_T) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) p_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}|\boldsymbol{z}_1),$$

as given in Section 3.6, where $\boldsymbol{f_\theta}$ represents a neural network. In general, the architecture of this network is

$$\begin{aligned} \texttt{MLP}_{[\alpha,\dots,\psi,\omega]}(\boldsymbol{u}) &= \texttt{FC}_{\omega\to p+1}(\texttt{MLPBlock}_{\psi\to\omega}(\cdots(\texttt{MLPBlock}_{\tau\to\alpha}(\boldsymbol{u})))), \\ \boldsymbol{u} \coloneqq \boldsymbol{u}(\boldsymbol{z}_t, t) &= \texttt{FC}_{p+1\to\tau}(\boldsymbol{z}_t) + \texttt{t\_emb}(t), \\ \texttt{t\_emb}(t) &= \texttt{FC}_{\tau\to\tau}(\texttt{SiLU}(\texttt{FC}_{\tau\to\tau}(\texttt{SinTimeEmb}_\tau(t)))), \end{aligned} \tag{5.1}$$

where $\boldsymbol{z}_t \in \mathbb{R}^{(p+1)\times 1}$ is a forward diffused observation, $\tilde{\boldsymbol{x}} \in \mathbb{R}^{(p+1)\times 1}$, to a diffusion step $t \in \{1, \dots, T\}$. There are quite a few parts of this architecture, which is inspired by Kotelnikov et al.'s [67] MLP, that need an explanation. First of all, ignoring $\texttt{t\_emb}(t)$, notice that $\boldsymbol{z}_t$ is linearly transformed to a vector in $\mathbb{R}^{\tau\times 1}$, before it is used as input to the MLP. Moreover, the MLP is relatively simple, made up of $|[\alpha, \dots, \psi, \omega]|$ `MLPBlock`s, with dimensions given by $[\alpha, \dots, \psi, \omega]$. Notice that the output from the MLP is a vector in $\mathbb{R}^{(p+1)\times 1}$, which is defined to match the input data dimension.

Sinusoidal embedding. The term $\texttt{t\_emb}(t)$ is a crucial addition to the neural network that enables training *one* neural network across *all* diffusion steps. Notice the introduction of $\texttt{SinTimeEmb}_\tau(t)$ inside this term. This refers to a *sinusoidal embedding* [141], first introduced as a positional embedding for use in *Transformers*. The use of this embedding in diffusion models was first mentioned by Ho et al. [47], without a detailed account of why this is an attractive choice. However, while training diffusion models, it is essential that the neural network has good awareness of the noise level t, since the estimated function essentially should be different for different t's. Intuitively, a noise prediction from $\boldsymbol{z}_{T-1}$ should be very different from a noise prediction from $\boldsymbol{z}_{T/2}$ or $\boldsymbol{z}_2$. In fact, the predicted noise should be much larger in the former than the two latter cases. The sinusoidal positional embedding by Vaswani et al. [141] is one possible technique for providing the neural network with the necessary awareness, such that it can be used to learn the denoising process across all noise levels $t \in \{1, \dots, T\}$. Said succinctly, a sinusoidal embedding uses sine and cosine functions of different frequencies to encode a scalar input t in a space $\mathbb{R}^{\tau\times 1}$ [141].

For simplicity, notice that we keep the *embedding dimension* τ constant throughout the linear transformations that make up $\texttt{t\_emb}(t)$. This can be generalized to differing

dimensions in each transformation, similar to in the main MLP. The hyperparameters that define the exact diffusion model we use to produce results in Chapters 6 and 7 are provided in Table 5.3. This table contains all necessary hyperparameters for uniquely defining the diffusion process, as well as the reverse process, in Tabular diffusion.

Role in algorithms. Recall Algorithms 5 and 6, for training and sampling from our diffusion model, respectively. The role of the MLP in Equation (5.1) is to calculate the prediction $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$ in step 17 and 10 in Algorithm 5 and Algorithm 6, respectively.

Conditional Modelling

In the second technique, we model the underlying joint density, $p^*(\boldsymbol{x}, y)$, via the property

$$p^*(\boldsymbol{x}, y) = p^*(y)p^*(\boldsymbol{x}|y).$$

This technique is inspired by the observation that Kotelnikov et al. [67] solely define a class-conditional model for classification datasets, where they learn the reverse process densities conditional to the label, i.e. $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1, y)$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, y)$, $t \in \{2, \dots, T\}$. Thus, our second technique is a concretization of the strategy that Kotelnikov et al. [67] appear to follow, where we develop a corresponding equation for the reverse process and explain how the densities are calculated. Notice that we develop direct joint modelling, with the intention of contrasting Kotelnikov et al.'s [67] suggested strategy.

Precisely, we learn the reverse process

$$\begin{aligned} p_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) &= p_{\boldsymbol{\theta}}(\boldsymbol{x}, y, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) \\ &= p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_T|y)p^*(y) \\ &= p(\boldsymbol{z}_T) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, y)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1, y)p^*(y), \end{aligned} \tag{5.2}$$

where the distributions $p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, y)$, $t \in \{2, \dots, T\}$, and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1, y)$ are parameterized by a neural network $\boldsymbol{f}_{\boldsymbol{\theta}}$. In this case, the neural network uses y as input, in addition to $\boldsymbol{z}_t \in \mathbb{R}^{p \times 1}$ and $t \in \{1, \dots, T\}$, when making predictions. Thus, the architecture of $\boldsymbol{f}_{\boldsymbol{\theta}}$ is

$$\begin{aligned} \texttt{MLP}_{[\alpha,\dots,\psi,\omega]}(\boldsymbol{u}) &= \texttt{FC}_{\omega \to p}(\texttt{MLPBlock}_{\psi \to \omega}(\cdots(\texttt{MLPBlock}_{\tau \to \alpha}(\boldsymbol{u})))), \\ \boldsymbol{u} \coloneqq \boldsymbol{u}(\boldsymbol{z}_t, t, y) &= \texttt{FC}_{p \to \tau}(\boldsymbol{z}_t) + \texttt{t\_emb}(t) + \texttt{FC}_{1 \to \tau}(y), \\ \texttt{t\_emb}(t) &= \texttt{FC}_{\tau \to \tau}(\texttt{SiLU}(\texttt{FC}_{\tau \to \tau}(\texttt{SinTimeEmb}_{\tau}(t)))). \end{aligned} \tag{5.3}$$

This MLP is identical to the one shown in Equation (5.1), with the exception of two details. First, any observation, $\boldsymbol{x}$, belongs to $\mathbb{R}^{p \times 1}$, which implies that the latent variables also are defined in this space. Second, the linear transformation $\texttt{FC}_{1 \to \tau}(y)$ is added to the input, in order to learn a representation of the class label that can be used by the network. Figure 5.1 illustrates the Tabular diffusion model implemented by Kotelnikov et al. [67], where the role of the class-conditional MLP is highlighted. Please ignore the slightly different notation. The idea is to illustrate that a data point is first split into its numerical and non-numerical parts, which are pre-processed differently. For clarity, we reiterate that, because of our assumptions on the data types, we process all numerical features as continuous features, i.e. in the Gaussian "part" of Tabular diffusion, and all the non-numerical features as categorical features, i.e. in the Multinomial "part" of Tabular diffusion. After the pre-processing, the two sets of features are forward diffused to a level $t \in \{1, \dots, T\}$ (during training) or separately sampled (during generation) — a process that is not explicitly shown in the figure. Thereafter, the two parts are concatenated

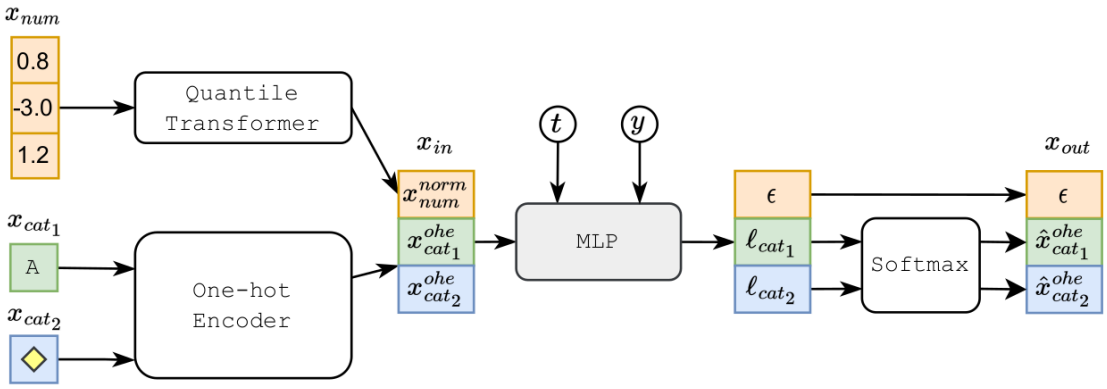


Figure 5.1: Schematic illustration of Tabular diffusion used for conditional modelling. Please ignore the slightly different notation. Specifically, Kotelnikov et al. [67] denote the latent variables by $\boldsymbol{x}$, as noted in the introduction of Section 3.3. In addition, t denotes a diffusion level, y denotes a class label, l denotes a logit and ϵ denotes a Gaussian noise prediction. The image is borrowed with permission from Kotelnikov et al. [67].

into $\boldsymbol{z}_t$, which then is used as input to $\boldsymbol{f_\theta}$, alongside t and y, to output a prediction, where different elements of the output are treated differently. Thus, the role of the MLP from Equation (5.3) during training and sampling is exactly the same as in direct joint modelling, with the exception that each input, $\boldsymbol{x}$, from the training dataset does not contain a value y, such that $\boldsymbol{z}_t \in \mathbb{R}^{p \times 1}$, and y is instead used as an extra input in steps 17 and 10 in Algorithms 5 and 6, respectively. Moreover, recall that we need to estimate $p^*(y)$ separately in order to sample using the reverse process in Equation (5.2). At sampling time, $p^*(y)$ is approximated by simply sampling realizations y from $Y \sim \text{Categorical}(\boldsymbol{p})$, where $\boldsymbol{p}$ is a vector of the empirical proportions of negative and positive predictions in the entire dataset. Finally, for simplicity, we use the same hyperparameters as in direct joint modelling, as shown in Table 5.3.

Technical details

The Tabular diffusion models are implemented using PyTorch [98]. This is a highly popular open source framework, used both in academia, research and industry, for implementing deep learning models in computer vision, natural language processing, generative modelling and many more fields. We use PyTorch for both the forward and reverse processes. For completeness, we actually first implement both Gaussian and Multinomial diffusion models separately, before combining them into the Tabular diffusion model. In this way, we facilitate modelling the fully numerical data in **DI**, when using the conditional modelling technique, with Gaussian diffusion, while the rest of the situations with **AD**, **CH** and **DI** can be modelled with the mixed Tabular diffusion model.

During implementation, we have taken inspiration from Kotelnikov et al.'s [67] diffusion model implementation[3]. While studying their code, it becomes clear that they have recycled several pieces of code from other authors that have used PyTorch as well. In fact, they make some explicit comments about this in some of their files. Some of the code that is recycled comes from the openly available implementations[4] by Dhariwal and Nichol [20, 21]. The first implementation among these two is attributed to a rewrite of Ho et al.'s [47] implementation[5] in Tensorflow [1], which is the main competitor of Py-

[3] `https://github.com/rotot0/tab-ddpm`

[4] `https://github.com/openai/improved-diffusion` and `https://github.com/openai/guided-diffusion`

[5] `https://github.com/hojonathanho/diffusion`

Table 5.3: Hyperparameters used for training **Tabular diffusion** for each dataset. We explicitly highlight when the hyperparameter value is default.

(a) Adult Census (AD)

Hyperparameter	Value
T	1000
`MLPBlock` dims.	[256, 1024, 1024, 1024, 1024, 256]
Emb. dim. (τ)	128
Var. schedule (β_t)	Linear
Batch size	256
Epochs	200
Early stop. tol.	10
Dropout	0.0
Optimizer	Adam
Learning rate	0.0001
Weight decay (L2 penalty)	0 (default)

(b) Churn Modelling (CH)

Hyperparameter	Value
T	1000
`MLPBlock` dims.	[512, 1024, 1024, 1024, 1024, 512]
Emb. dim. (τ)	128
Var. schedule (β_t)	Linear
Batch size	128
Epochs	200
Early stop. tol.	10
Dropout	0.0
Optimizer	Adam
Learning rate	0.0001
Weight decay (L2 penalty)	0 (default)

(c) Diabetes (DI)

Hyperparameter	Value
T	1000
`MLPBlock` dims.	[128, 512]
Emb. dim. (τ)	128
Var. schedule (β_t)	Linear
Batch size	64
Epochs	200
Early stop. tol.	10
Dropout	0.0
Optimizer	Adam
Learning rate	0.0001
Weight decay (L2 penalty)	0 (default)

Torch among deep learning platforms. The mentioned implementations are only for the Gaussian part, whereas the Multinomial part of Kotelnikov et al.'s [67] implementation was inspired by Hoogeboom et al.'s [49] implementation[6]. The point of this discussion is to show that researchers do not re-implement every part of the models in their individual research. In fact, many of the researchers more or less copy the main framework, before making modifications to suit their novel theories. This is one of the great assets of open research and open source code, which probably has increased the speed of research on diffusion models. Some of our code, for example a function defining the variance schedules in Gaussian diffusion, is mostly copied from one of the mentioned sources, whereas other parts are re-implemented by us. For instance, we have chosen to construct the main framework for Gaussian, Multinomial and Tabular diffusion differently than the rest of the works, but this is just an implementation detail. Thus, despite using the mentioned open source repositories as inspiration, we have copied little code directly, adding some originality to our implementation. An advantage of this strategy is that it has helped us deepen our knowledge on diffusion probabilistic models. For transparency, we strive to give explicit credit in the form of comments in our code at parts we have copied or parts that are highly recognizable from other implementations.

As a side note, the Multinomial diffusion implementation has some intricacies that we want to comment on. Drawing inspiration from Hoogeboom et al. [49], we implement Multinomial diffusion in *log-space*, to facilitate greater numerical stability. The interested reader should read Appendix A in Hoogeboom et al. [49] for a concise technical explanation of how this may be performed. In addition to this, some more in-depth considerations we make while developing are added in Appendix E.

Finally, we clarify some hyperparameter names from Table 5.3. "Early stop. tol." refers to the fact that we have added *early stopping* to our training loops. This is a simple regularization technique that stops training when the validation loss stops improving, according to some pre-specified tolerance. For example, setting the tolerance to 10 in our case means that training is stopped if the validation loss does not reach a new minimum for 10 epochs in a row. While training, the model parameters are updated and the parameters that yield the lowest validation loss are saved. Notice that the criterion we have implemented is very simple. The interested reader is referred to Prechelt [103] for a more detailed discussion on when and how to use early stopping. Moreover, note that the hyperparameters "Learning rate" and "Weight decay (L2 penalty)" in Table 5.3 all refer to hyperparameters of the Adam optimizer [63] implementation in PyTorch (from `torch.optim`). Notice that we do not use any weight decay during optimization, which is the default value in the PyTorch implementation, which we specify by "default" in Table 5.3.

5.3.4 TVAE

Before explaining how we use TVAE to model the joint underlying data distribution, we provide some more context on why we chose TVAE as a baseline model. First, to the best of our knowledge, TVAE [146] is the state-of-the-art (SOTA) VAE-based tabular data synthesizer. Moreover, it has open source code, which makes it practical to incorporate in experiments. For these reasons, it was also used as a baseline by Kotelnikov et al. [67], which we, as noted, exploit to guide our choices concerning hyperparameter values.

With TVAE, we model the joint underlying data density, $p^*(\boldsymbol{x}, y)$, directly, analogously to in direct joint modelling in Tabular diffusion. In reference to the neural networks used for the encoder and decoder described in Section 2.9, the hyperparameters we use for training and sampling from TVAE are shown in Table 5.4.

[6] `https://github.com/ehoogeboom/multinomial_diffusion`

Table 5.4: Hyperparameters used for training **TVAE** for each dataset. We explicitly highlight when the hyperparameter value is default.

(a) Adult Census (AD)

Hyperparameter	Value
CD	[128, 128] (default)
DD	[128, 128] (default)
Latent space dimension (Λ)	128 (default)
Batch size	256
Epochs	200
Optimizer	Adam (default)
Learning rate	0.001 (default)
Weight decay (L2 penalty)	1e-5
Loss factor	2 (default)

(b) Churn Modelling (CH)

Hyperparameter	Value
CD	[256, 512]
DD	[256, 512]
Latent space dimension (Λ)	256
Batch size	128
Epochs	200
Optimizer	Adam (default)
Learning rate	0.001 (default)
Weight decay (L2 penalty)	1e-5
Loss factor	2 (default)

(c) Diabetes (DI)

Hyperparameter	Value
CD	[128, 128] (default)
DD	[128, 128] (default)
Latent space dimension (Λ)	128 (default)
Batch size	64
Epochs	200
Optimizer	Adam (default)
Learning rate	0.001 (default)
Weight decay (L2 penalty)	1e-5
Loss factor	2 (default)

Technical details

We implement TVAE with open source code[7] under the project *the Synthetic Data Vault Project* (SDV) [99]. First created at MIT in 2016, it was designed to be the "one-stop shop for synthetic data", according to the inventors. SDV contains open source implementations for the entire synthetic data pipeline, from data pre-processing to measuring quality and privacy of different synthetic data generation models. Our initial idea was to use the API provided by SDV to train, and sample data, from TVAE [146]. At the time of developing our code, the developers of SDV were in the middle of transitioning into a 1.0 Beta version, with several improvements and new documentation sites. However, because of incompatibilities with our system, specifically the beta not yet being available for the *conda* package manager, we chose to use the pre-beta implementation of the code. In fact, since SDV contains a simple wrapper on top of the standalone open source implementation[8], we use the latter directly. The code we develop using this API is available in our final repository[9].

Again, note that the hyperparameters "Learning rate", "Weight decay (L2 penalty)" and "Loss factor" in Table 5.4 all refer to hyperparameters of the Adam optimizer [63] implementation in PyTorch (from `torch.optim`). Notice the values with an added "(default)" — these refer to default hyperparameter values in the standalone open source implementation of TVAE.

5.3.5 Trees

Analogously to in the conditional modelling technique for Tabular diffusion, we model the joint density $p^*(\boldsymbol{x}, y)$ via the property

$$p^*(\boldsymbol{x}, y) = p^*(y)p^*(\boldsymbol{x}|y),$$

where $\boldsymbol{X} \coloneqq \boldsymbol{X}_m \coloneqq \{X^1, \ldots, X^p\}$. The conditional density $p^*(\boldsymbol{x}|y)$ is approximated by use of the two first steps in MCCE after defining $\boldsymbol{X}_f \coloneqq Y$. Thus, we fit p trees in total, following the illustration in Equation (4.1). After estimating the parameters of the tree-based model, we use Algorithm 7 to generate data, where $\mathbf{D}_{\boldsymbol{h}}$ is initialized slightly differently, depending on the experiment. In the first experiment, where we do not have any factuals to explain, we define $\mathbf{D}_{\boldsymbol{h}}$ as a matrix of mostly zeroes in $\mathbb{R}^{K\times(p+1)}$, except for the values in the entire left-most column, which are random samples from the response column in the original dataset $\mathcal{D}$, i.e. the column corresponding to Y. Thus, at sampling time, $p^*(y)$ is essentially modelled by sampling y-values from $\mathcal{D}$, analogously to in the conditional modelling technique for Tabular diffusion, such that $p^*(\boldsymbol{x}, y)$ ultimately can be estimated. In the second experiment, where the goal is to synthesize counterfactuals, we define $\mathbf{D}_{\boldsymbol{h}}$ as described in Section 4.1, where the sets of mutable and fixed features are pre-defined. We return to this in Section 5.5.

Technical Details

When producing results from the trees in MCCE, we use the API[10] developed by Redelmeier at al. [107]. In fact, we adapt certain parts of the MCCE-implementation to fit our needs. For completeness, we added the entire open source implementation of MCCE, including our amendments, to our final code repository. For example, we have explicitly added the normalization values, R_j, as the empirical range, for all continuous (in

[7] All code is openly available at `https://github.com/sdv-dev/SDV`

[8] The standalone implementation of TVAE can be found at `https://github.com/sdv-dev/CTGAN`

[9] `https://github.com/alexaoh/tabular-diffusion-for-counterfactuals`

[10] The open source implementation of MCCE can be found at `https://github.com/NorskRegnesentral/mccepy`

Table 5.5: Hyperparameters used for training **MCCE** across all three datasets. The hyperparameters refer to the `DecisionTreeClassifier` implementation of CART in Scikit-learn [101]. We explicitly highlight when the hyperparameter value is default.

Hyperparameter	Value
`criterion`	gini (default)
`splitter`	best (default)
`max_depth`	None (default)
`min_samples_split`	2 (default)
`min_samples_leaf`	5 (default)
`min_weight_fraction_leaf`	0.0 (default)
`max_features`	None (default)
`max_leaf_nodes`	None (default)
`min_impurity_decrease`	0.0 (default)
`class_weight`	None (default)
`ccp_alpha`	0.0 (default)
`random_state`	seeds from Table 5.2.

our case, all numerical) features j, for calculating the Gower distance, since this was implicitly added through the pre-processing performed by Redelmeier et al. [107] in their original implementation. Moreover, notice that they implemented CART with Scikit-learn's [101] `DecisionTreeClassifier`, which uses an optimized version of the CART algorithm. As a consequence, we specify the hyperparameters we use to produce results in Chapters 6 and 7, with the tree-based model, in reference to the terms defined in the `DecisionTreeClassifier`. These hyperparameters are specified in Table 5.5. Notice that we only use the default hyperparameters from Redelmeier et al.'s [107] implementation or the defaults from `DecisionTreeClassifier`, except for the `random_state`. Either of these two situations are specified with "(default)" in Table 5.5.

5.4 Experiment I — Evaluation of Synthetic Data Generation

The first experiment is devoted to evaluating the performance of Diff-MCCE on its first two steps in isolation. The objective is to compare the performance of Tabular diffusion to our two baselines; TVAE and the tree-based model in MCCE. All the generative models are trained according to the techniques for modelling $p^*(\boldsymbol{x}, y)$ described in Section 5.3. Then, after estimating the parameters, we generate synthetic datasets of *identical* sizes to the full original datasets, as specified in the column "# Total" in Table 5.1. Precisely, recall that pre-processing, training and sampling is performed in five different random trials for each generative model and each real-world dataset. In the following, we describe precisely how we evaluate the models, post sampling.

Qualitative evaluation. Initial qualitative evaluation is performed, following the processes described in Section 2.10. Notice that we initially select a synthetic dataset from each generative model from only *one* of the five trials performed for each of the three real-world datasets. Then, the following procedures are performed for each set of five datasets; one real dataset and four corresponding synthetic datasets. We overlay all marginal distributions in each pair of real and synthetic datasets, in order to easily spot high-level similarities and differences. Moreover, we calculate the correlation matrices of all the datasets, both synthetic and real. After that, we perform element-wise subtraction of the correlation matrix of each synthetic dataset from the correlation matrix of the

corresponding real data, before applying the absolute value. These matrices of absolute differences in correlation are then visualized as heatmaps, arranged in a grid-like structure, enabling an easy qualitative understanding of where the synthetic data is suboptimal, both for each individual synthesizer and comparatively between the synthesizers.

Quantitative evaluation. ML efficacy is our quantitative evaluation measure of choice. Adding to the outline in Section 2.10, we explain specifically how we calculate ML efficacy in this experiment. The following procedure is performed in each trial. We assume the starting point is a synthetic dataset per generative model. The synthetic data is randomly split into training, validation and testing datasets, following the same split ratios as for the real data. We proceed by employing the optimal protocol for ML efficacy, meaning that we fit a CatBoost model on each of the training datasets, stemming from real and synthetic data. Thus, in total we fit five binary classifiers for each real-world dataset; one using the real training data and four using the synthetic training data. Specifically, we use the `CatBoostClassifier` method from the open source Python implementation[11] of CatBoost by Prokhorenkova et al. [104]. We leave all the hyperparameters of the model as default, except for the `random_seed`, which is changed according to the seeds in Table 5.2, to initiate each trial. Also, since the CatBoost model can handle non-numerical features, and because it is not as sensitive to differing scales in numerical feature values as, for example, neural networks, we simply skip pre-processing and train the CatBoost model on the original training datasets. After parameter estimation, every CatBoost classifier is evaluated, with respect to our chosen metrics, on the *real* test dataset. Finally, the results are presented as the mean $\pm$ standard error of each given metric over the five distinct trials. To complement this presentation, we make some box plots as well, which display the minimum, first quartile, median, third quartile and maximum, of each metric over the five distinct trials, in a visually pleasing way. Both the qualitative and quantitative results are presented and discussed in Chapter 6.

5.5 Experiment II — Evaluation of Counterfactual Generation

The second experiment is devoted to evaluating the performance of Diff-MCCE on generating counterfactuals for a set of test observations. We use the different generative models from the previous experiment to solve the first two steps of the MCCE-framework, before post-processing the samples, as explained in Chapter 4. As in the first experiment, we perform five different trials and report aggregated results across them. In the following, we explain the process that is performed in each trial, from how the set of factuals for each real-world dataset is found to how the final results are reported.

Find factuals. We base the experiment on a binary classifier we assume that we are interested in understanding to a greater extent. Specifically, we train a CatBoost model to predict the binary response in each real-world dataset, without pre-processing the data. We leave all hyperparameter values as default, except for the `random_seed`, just as in the previous experiment. After estimating the parameters, we use the models to make predictions on test data. Among the predicted individuals, we randomly select 100 negatively predicted individuals in **AD** and **CH**, while we select only 10 negatively predicted individuals in **DI**. This has to do with the different test dataset sizes in each case, as shown in Table 5.1. These sets of individuals constitute the sets of factuals, $\mathcal{H}$, we want to explain in each case. Notice that the sets of factuals are most likely different in each trial, since the process described above is stochastic and included in each distinct trial.

[11] `https://github.com/catboost/catboost`

Fixed features. Recall that the actionability constraint of CEs require a predefined notion of immutable features. The fixed features we assume in each of the datasets are given in Table 5.6, implicitly assuming that the rest of the features in each dataset are mutable. Please notice that we are not experts in any of the domains where these datasets were collected. This means that we risk naively overlooking or defining certain variables as fixed or mutable, but we have done our best given our limited domain knowledge. However, this should not take away from the main objectives of the work, so we do not spend much time making sure these choices are realistic.

Table 5.6: Predefined fixed features in each dataset.

Dataset	Fixed features
Adult Census (**AD**)	`age`, `sex`
Churn Modelling (**CH**)	`Age`, `Gender`
Diabetes (**DI**)	`age`

Counterfactual synthesis. In each trial, we simply use the three-step process described in Chapter 4 to generate CEs from each model. We set $K = 10000$, meaning that we generate $K = 10000$ samples for each counterfactual $\boldsymbol{h} \in \mathcal{H}$, each set representing the possible CEs to choose from, $\mathbf{D}_{\boldsymbol{h}}$. Moreover, we choose to return only *one* counterfactual per factual, instead of several. The first two steps in Diff-MCCE and the TVAE-based reference are implemented according to the descriptions in Section 5.3, with a slight alteration. Instead of modelling $(\boldsymbol{X}, Y) = \{X^1, \ldots, X^p, Y\} \sim p^*(\boldsymbol{x}, y)$, we model $\boldsymbol{X} = \{X^1, \ldots, X^p\} \sim p^*(\boldsymbol{x})$, because it is not necessary to include Y in the modelling when the objective is to generate counterfactuals. Moreover, the third step in Diff-MCCE and the TVAE-based reference is performed according to the post-processing outlined in Section 4.2. Finally, we utilize MCCE exactly as explained in Section 4.1, with the fixed features in Table 5.6.

Results. The results from this experiment are reported in Chapter 7. In every trial, we calculate the average values, across all factuals, of sparsity and Gower distance. Moreover, we count the number of factuals that are given CEs, which we denote by N_{CE}. As a side note, this means that if $N_{\text{CE}} < 100$ in **AD** and **CH**, and $N_{\text{CE}} < 10$ in **DI**, then some of the factuals have not obtained explanations from the method in question. After calculating these three metrics in each trial, we report the first two as the mean $\pm$ standard error over the five distinct trials. In addition, to complement this presentation, we display some box plots calculated from the first two metrics, similarly to in the first experiment. Furthermore, the third metric is reported as a list of N_{CE}s, containing one value of N_{CE} per trial. Finally, in addition to the aggregated results, we report some examples of explanations for a randomly selected factual $\boldsymbol{h} \in \mathcal{H}$, calculated from Diff-MCCE, as well as the baselines.

Chapter 6

Results — Generating Synthetic Data

This chapter is dedicated to demonstrating and discussing results from the first experiment described in Chapter 5, on generating synthetic data. Specifically, these results are used to evaluate the performance of Diff-MCCE on the task of modelling $(\boldsymbol{X}, Y) \sim p^*(\boldsymbol{x}, y)$ and generating synthetic realizations of the random variables.

6.1 Qualitative Evaluation

The marginal distributions in each synthetic and corresponding real dataset are plotted in Figures 6.2 to 6.13. Notice that we do not explicitly add the category names of the non-numerical features to make the plots more readable, since the specific categories are not of primary concern. In addition to the marginals, the matrices of absolute differences in correlation, as explained in Section 5.4, are plotted as heatmaps in Figure 6.1. We have deliberately placed this figure before the others, to facilitate easier comparison between the marginals from the different models when reading the thesis in two page view.

Naming. When presenting the results, the terms *TabDiffJoint* and *TabDiffCond* refer to results when using Tabular diffusion with direct joint modelling and conditional modelling, respectively. Moreover, the terms *TVAE* and *MCCE* refer to results when using TVAE and the tree-based model in MCCE for modelling and sampling, respectively.

Initial observation. Our initial observation is that the tree-based model in MCCE is clearly superior, on all datasets, according to all qualitative measures. The qualitative differences between the results from this model and the rest are quite striking. In the following, we dive deeper into a qualitative evaluation of the models on each of the datasets separately.

Adult Census (AD) dataset. Moving past the most striking initial observation, there is no clearly superior variant among the three other models. Studying the marginals from these three models in **AD**, in Figures 6.2 to 6.4, most marginal distributions in both the sets of results from Tabular diffusion look similar. The marginals of some of the features look more similar to the real marginals in direct joint modelling, and vice versa, but neither of the two models produce an overwhelmingly superior set of marginals across all features. However, most of the marginal distributions produced by both Tabular diffusion models look qualitatively better than the results from TVAE. This is not the case in the correlation matrices for **AD**, in Figure 6.1; TVAE is clearly superior to the Tabular diffusion variants on these measures. Diving deeper, they look quite similar in terms of quality of reconstruction of linear correlations between the numerical features, but the Tabular diffusion models perform much worse according to the correlation ratios in non-numerical and numerical pairs of features, as well as according to the Theil's U

statistics between non-numerical features. For instance, the two models struggle especially with truthfully modelling the relation between `marital_status` and `relationship`, in both asymmetric cases, as well as the relation `relationship` given `sex`. From this we might hypothesize that Tabular diffusion performs relatively better for datasets with only numerical features, i.e. that errors in correlation-reconstruction are introduced mainly because of issues with modelling interactions including non-numerical features. Further investigation in the other datasets will illuminate the sense of this hypothesis. Finally, note that the correlation matrices in Figure 6.1 for **AD** are almost identical for both Tabular diffusion models, meaning that we cannot immediately distinguish them with regards to correlation modelling capabilities either.

Churn Modelling (CH) dataset. A similar discussion is performed on **CH**. Looking at the marginals in Figures 6.6 to 6.8, the numerical features look relatively similar in all three, except for TVAE severely overestimating the number of customers with `Tenure` 1 and overestimating the number of customers around the mode in `CreditScore`. At the same time, both sets of results from Tabular diffusion show an overestimation of values close to the maximum in `CreditScore`, while this number is underestimated in TVAE. These observations might be consequences of choosing to treat the integers as continuous variables, especially for `Tenure`, which only has 11 levels. However, opposing this hypothesis, notice that the estimations of `NumOfProducts` are not very far off the true marginals, especially in TVAE, despite the fact that this integer only has four different recorded levels in the full real **CH** dataset. Thus, this assumption seems to yield adequate results in some cases and inadequate results in others, but we do not see enough evidence to make more specific conclusions. Moreover, keep in mind that we pre-process the data in the same way for the tree-based model, in which we do not observe any of these issues, which implies that there are other elements behind the performance discrepancies. Moving on from the numerical features, most of the non-numerical feature marginals look closer to the corresponding marginals in the real data in the results from Tabular diffusion. The sum of all these observations lead us to believe that the two, almost indistinguishable Tabular diffusion models have reproduced the real empirical marginals in **CH** to a greater extent than TVAE. Progressing to the correlation matrices in Figure 6.1, all three heatmaps look relatively similar. TVAE seems to struggle a little more with accurately reproducing the relation between some of the non-numerical and integer features, while both the Tabular diffusion correlation matrices look marginally better. Thus, for **CH**, we argue that Tabular diffusion seems to qualitatively perform better than TVAE, where the two diffusion-techniques are practically indistinguishable. In contrast to in **AD**, the reconstruction of correlation from Tabular diffusion does not seem to suffer from the existence of non-numerical features in **CH**, which does not support the hypothesis we introduced in the previous paragraph.

Diabetes (DI) dataset. Finally, we qualitatively reflect on the performance of the models on **DI**. First of all, despite MCCE still clearly performing the best, the marginal distributions and the correlation matrices are not as close to the corresponding quantities in the real dataset as observed in the two other datasets. This is likely because of the very limited sizes of the data. In continuation, the marginals in Figures 6.10 to 6.12 reveal that the Tabular diffusion models struggle with learning truthful marginals. The direct joint modelling technique performs especially bad, producing many outliers or values close to and beyond the minimum and maximum values in the real dataset. A similar phenomenon can be observed for the conditional modelling version as well, but this model seems to learn a larger part of the marginals somewhat more truthfully. TVAE does not suffer from such a problem. It clearly performs better, in some cases quite close to the performance of MCCE. The most notable unrealistic result from TVAE is that it has synthesized negative values in `dbp`, when the diastolic blood pressure should not be negative in practice. In

addition, it has produced values of `plasma` larger than 199, which has not occurred in the other generative models, and does not exist in the complete real dataset. However, we lack the domain knowledge to conclude if this is physiologically unrealistic. Moving on to the correlation matrices in Figure 6.1, our conclusions are similar to the ones we made based on the marginals. Tabular diffusion struggles to truthfully model the correlations in the real dataset, especially in the direct joint modelling version, but the conditional modelling version is also suboptimal. TVAE seems more comparable to MCCE, although still suboptimal, performing a lot better than Tabular diffusion. Thus, keeping our hypothesis raised in the paragraph on **AD** in mind, we do not find the statement reasonable in general, because **DI** only consists of numerical features, but Tabular diffusion clearly displays the worst qualitative performance among the set of models.

Closing remarks. A general observation is that the Tabular diffusion models seem to consistently overestimate the number of observations in the smallest and largest numerical value bins in the marginal distributions. As noted, this is spotted in **DI**, in Figures 6.10 and 6.11, where the overestimation is quite severe. However, we also observe the same phenomenon in **AD** and **CH**, although in a notably less critical fashion. It is not immediately apparent why this happens, but it seems to be a downfall for Tabular diffusion in comparison to the other generative models we investigate. Keeping our choice of treating integers as continuous values in mind, we do not find any obvious evidence to conclude that this assumption aggravates the behaviour, as the overestimation seems to appear with similar severity for both floats and integers. For **DI**, the problem may be exacerbated by the size of the dataset, because we only evaluate a synthetic dataset containing 768 points. Thus, the evaluations made on **DI** perhaps do not have great statistical power, but they indicate how the models could perform on small tabular, binary classification datasets with only numerical features. At the same time, TVAE seems to consistently overestimate the number of observations around the modes of the marginal distributions. It is not immediately clear why this happens either, but it would be reasonable to hypothesize that the Gaussian-based (mode-specific in TVAE) pre-processing of discrete numerical values could exacerbate this problem. With this idea in mind, we would also expect similar effects on the integer features in Tabular diffusion and MCCE, because of the continuous pre-processing, in addition to the Gaussian assumptions in the forward process of the former, but we do not observe this. This means that we, again, do not think the "continuous-integer" assumption is vital for any of these observed differences in performance. All in all, whatever the true explanations are, the observations perhaps point at some general practical disadvantages of Tabular diffusion and TVAE. However, a clear conclusion cannot be reached without further investigation.

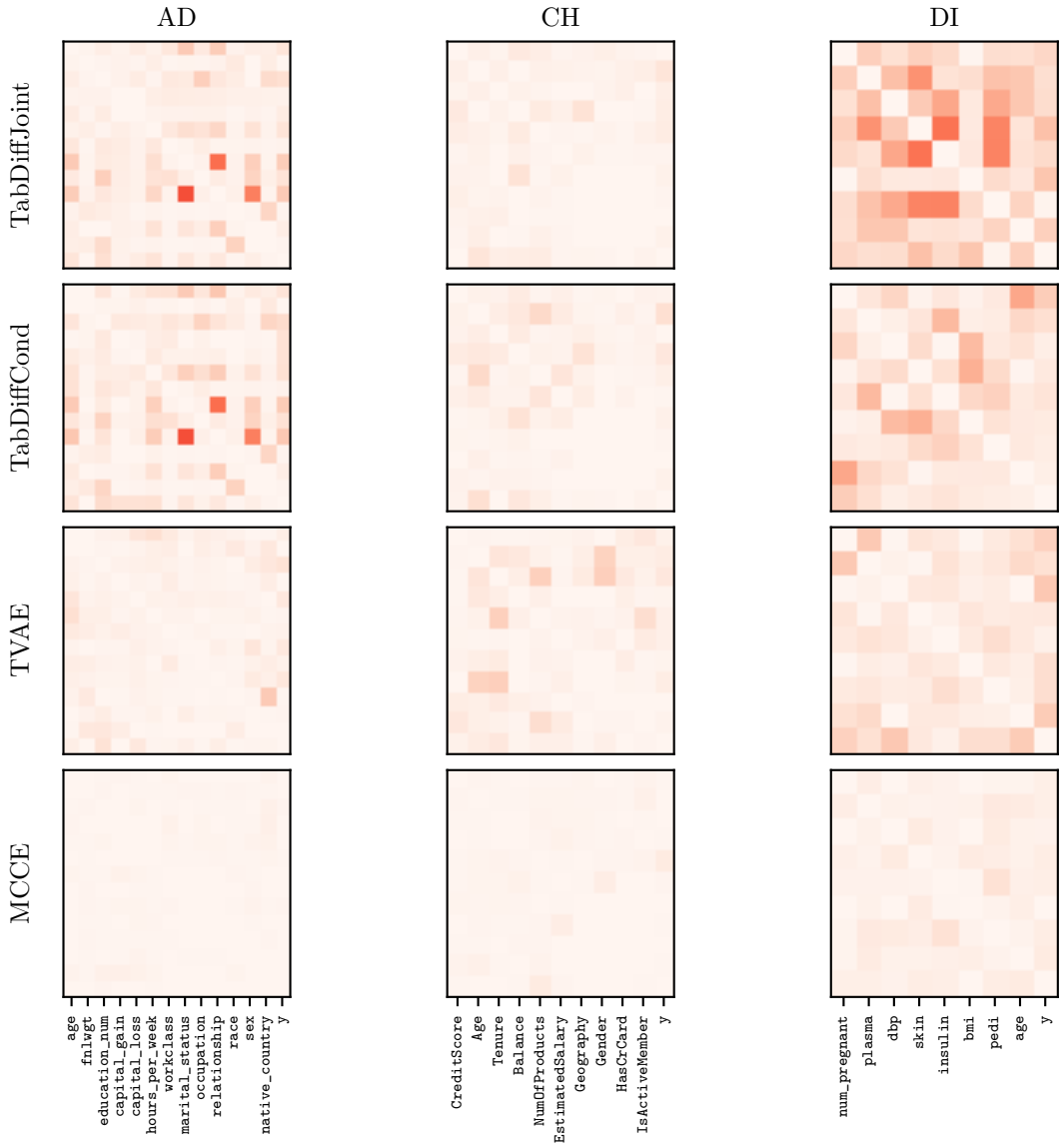


Figure 6.1: Absolute differences between correlations in synthetic and real datasets for the Adult Census (**AD**), Churn Modelling (**CH**) and Diabetes (**DI**) datasets, for both Tabular diffusion techniques, as well as TVAE and the decision tree model in MCCE. Deeper red colors indicate larger absolute differences in correlation between the synthetic and real data.

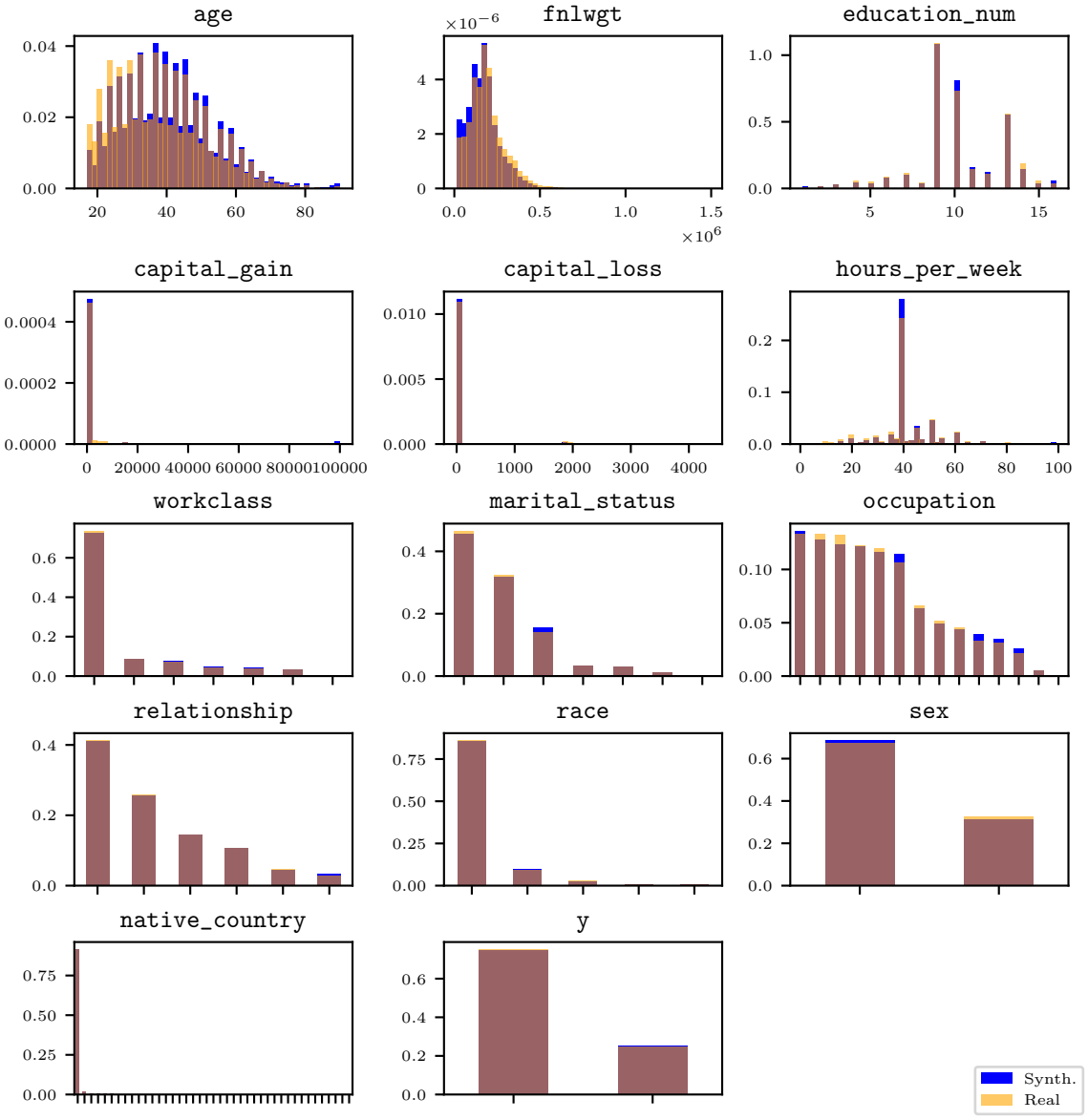


Figure 6.2: Histograms and bar plots of all the features in the **Adult Census** (**AD**) dataset, comparing the real dataset with a synthetic dataset generated by **Tabular diffusion** with **direct joint modelling**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

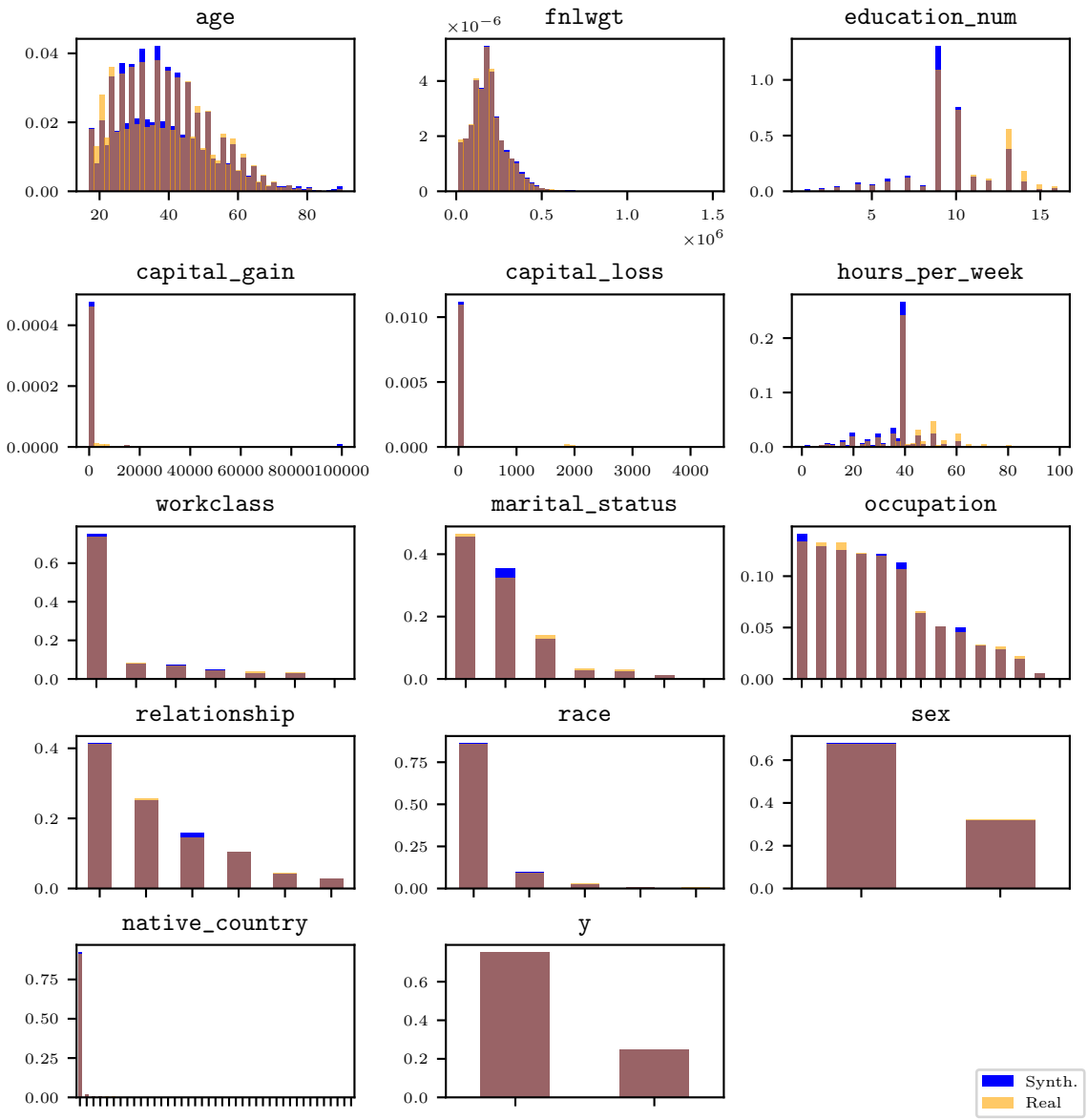


Figure 6.3: Histograms and bar plots of all the features in the **Adult Census** (**AD**) dataset, comparing the real dataset with a synthetic dataset generated by **Tabular Diffusion** with **conditional modelling**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

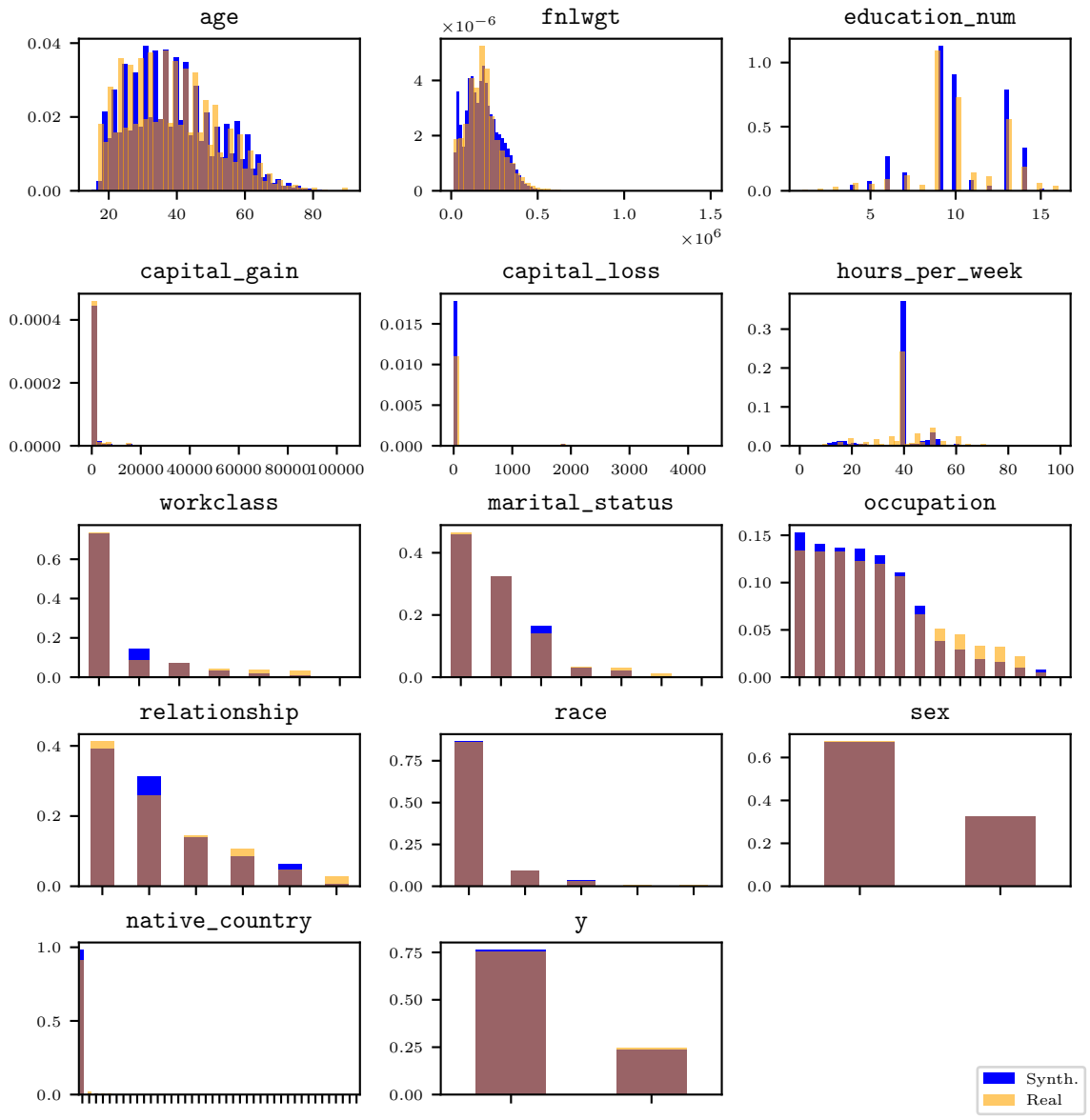


Figure 6.4: Histograms and bar plots of all the features in the **Adult Census** (**AD**) dataset, comparing the real dataset with a synthetic dataset generated by **TVAE**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

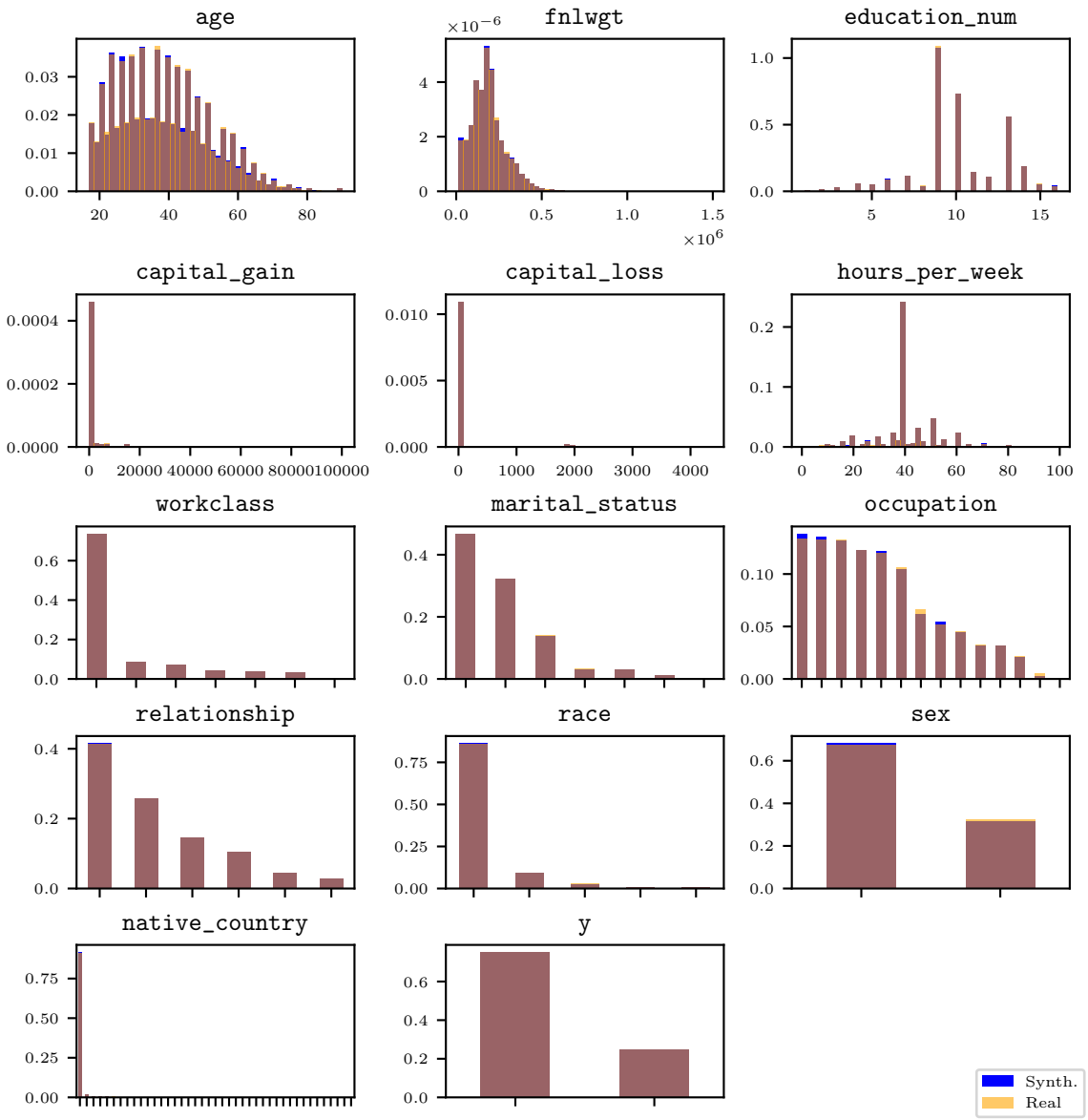


Figure 6.5: Histograms and bar plots of all the features in the **Adult Census** (**AD**) dataset, comparing the real dataset with a synthetic dataset generated by the two first steps of **MCCE**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

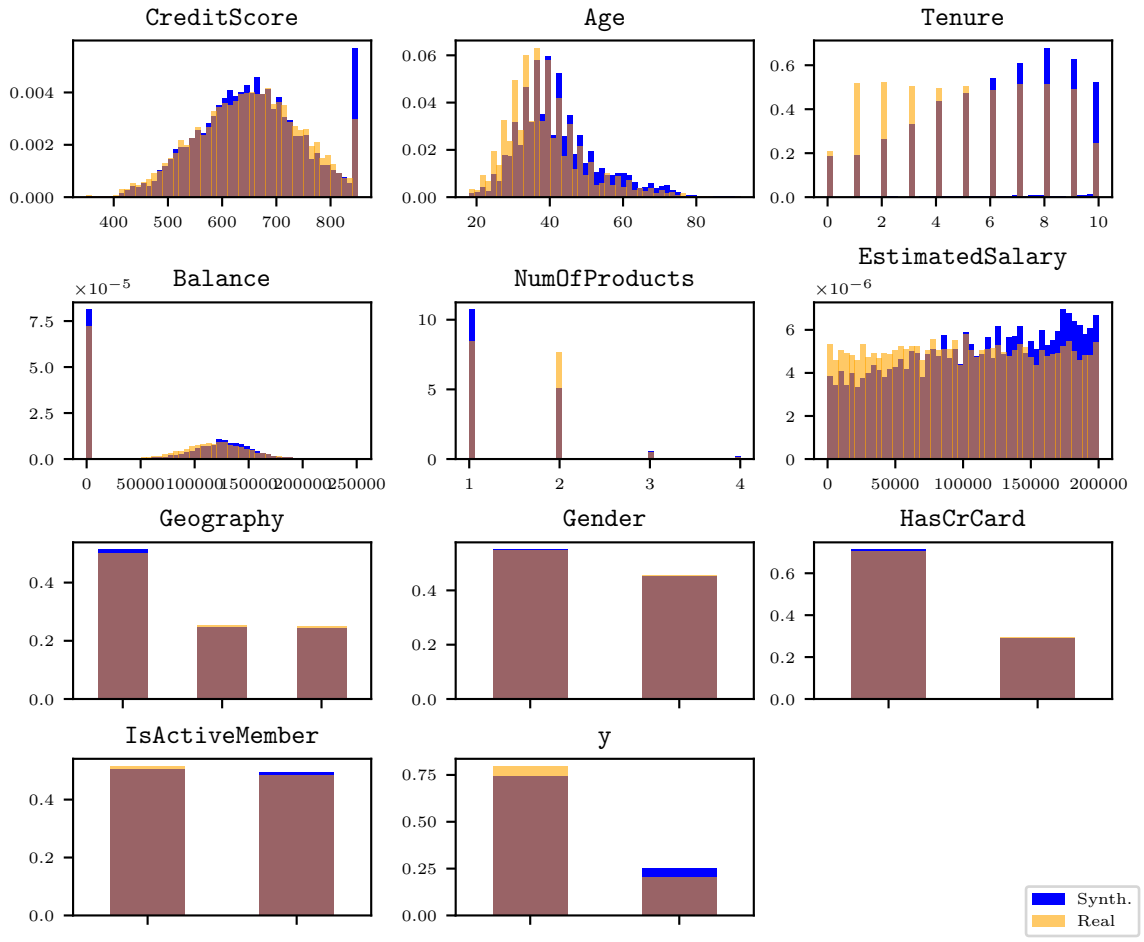


Figure 6.6: Histograms and bar plots of all the features in the **Churn Modelling** (**CH**) dataset, comparing the real dataset with a synthetic dataset generated by **Tabular Diffusion** with **direct joint modelling**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

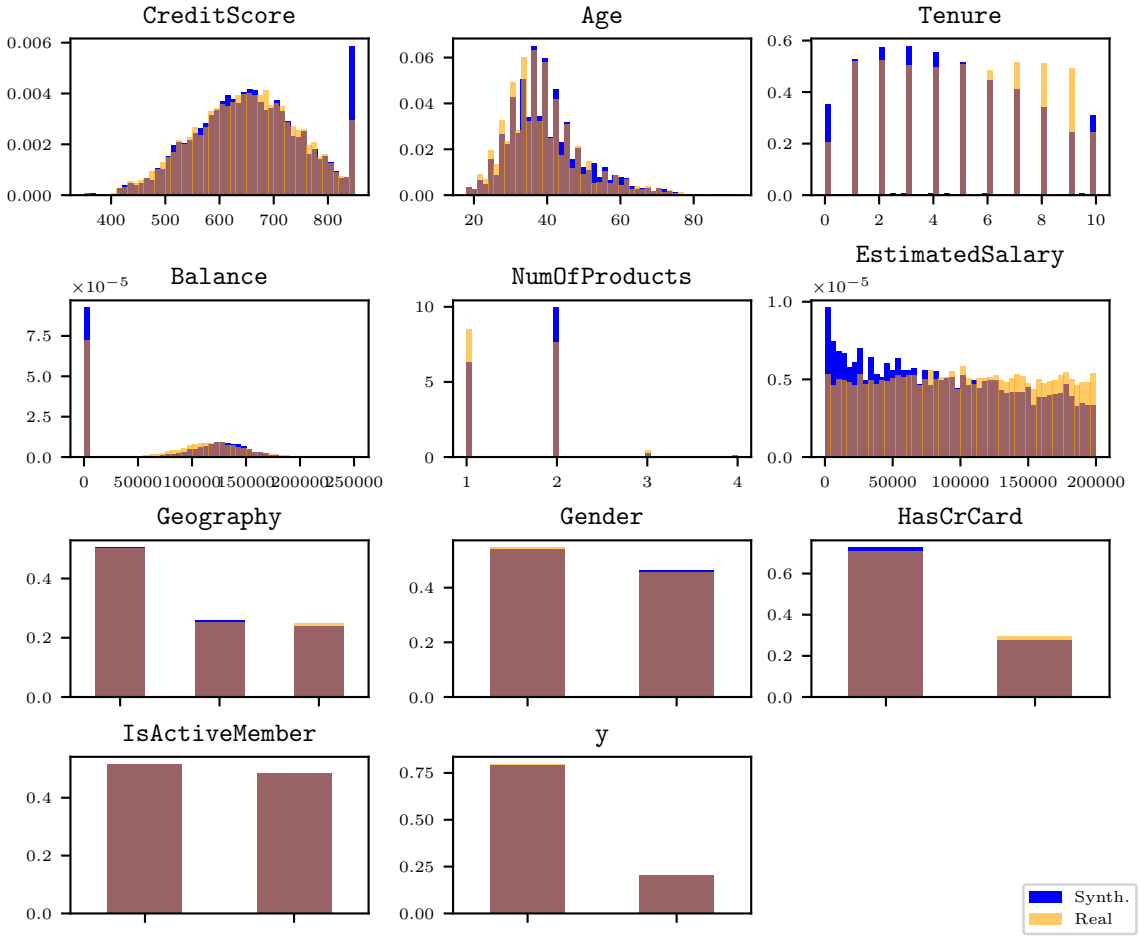


Figure 6.7: Histograms and bar plots of all the features in the **Churn Modelling** (**CH**) dataset, comparing the real dataset with a synthetic dataset generated by **Tabular Diffusion** with **conditional modelling**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

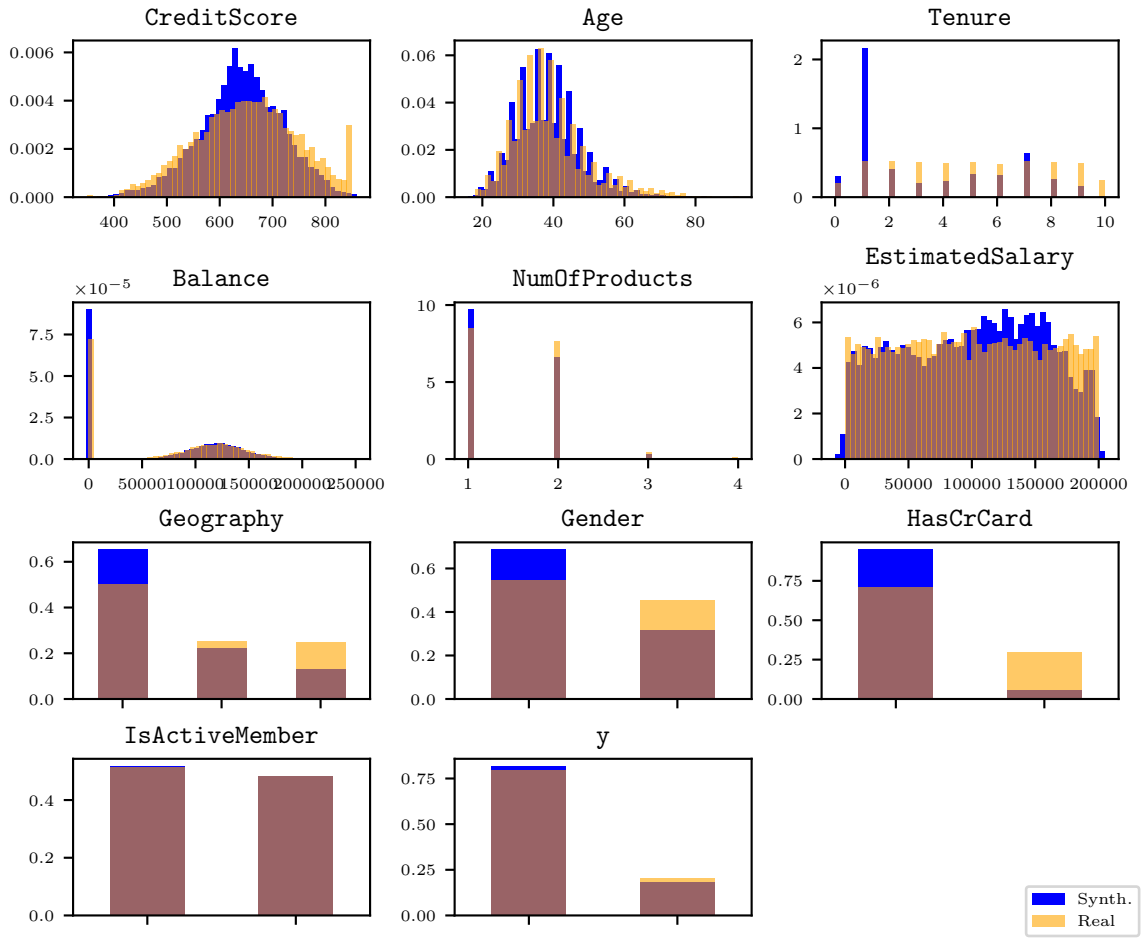


Figure 6.8: Histograms and bar plots of all the features in the **Churn Modelling** (**CH**) dataset, comparing the real dataset with a synthetic dataset generated by **TVAE**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

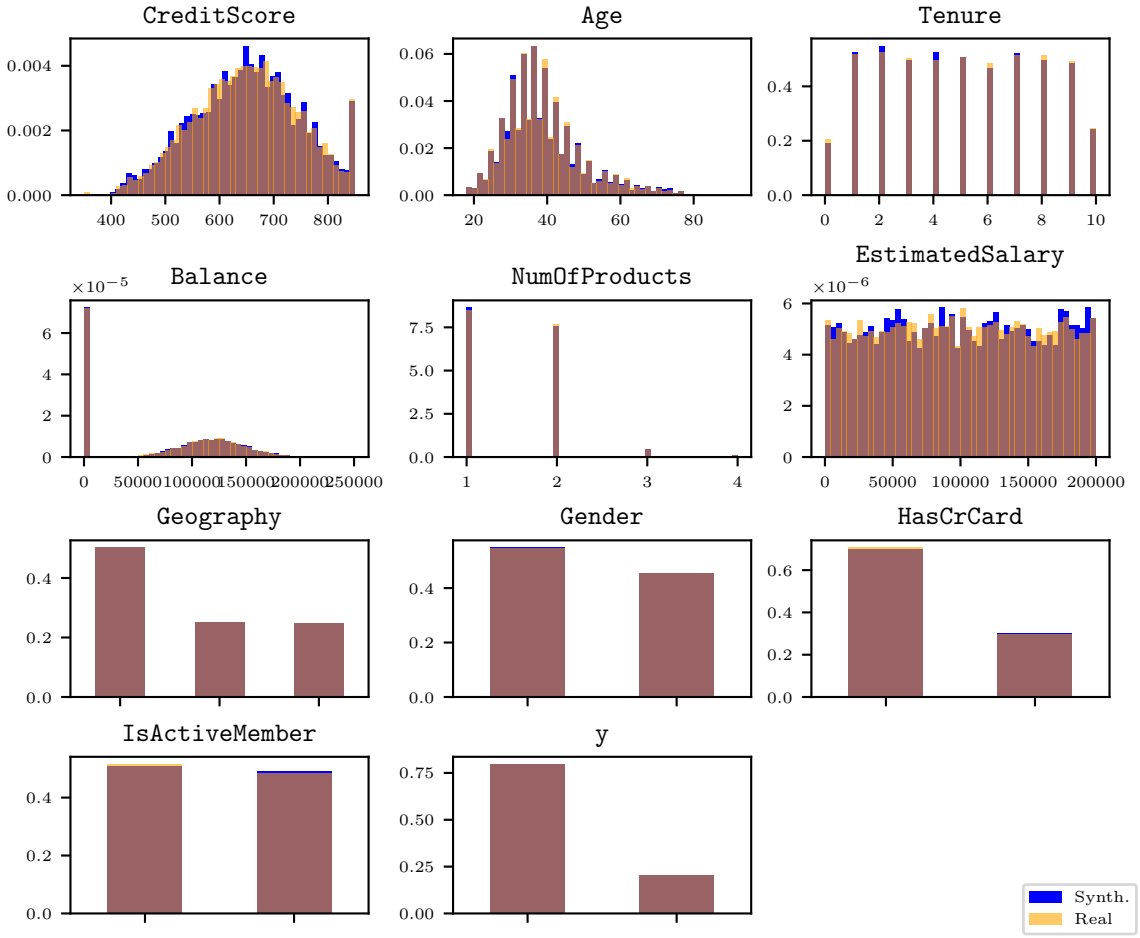


Figure 6.9: Histograms and bar plots of all the features in the **Churn Modelling** (**CH**) dataset, comparing the real dataset with a synthetic dataset generated by the two first steps of **MCCE**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

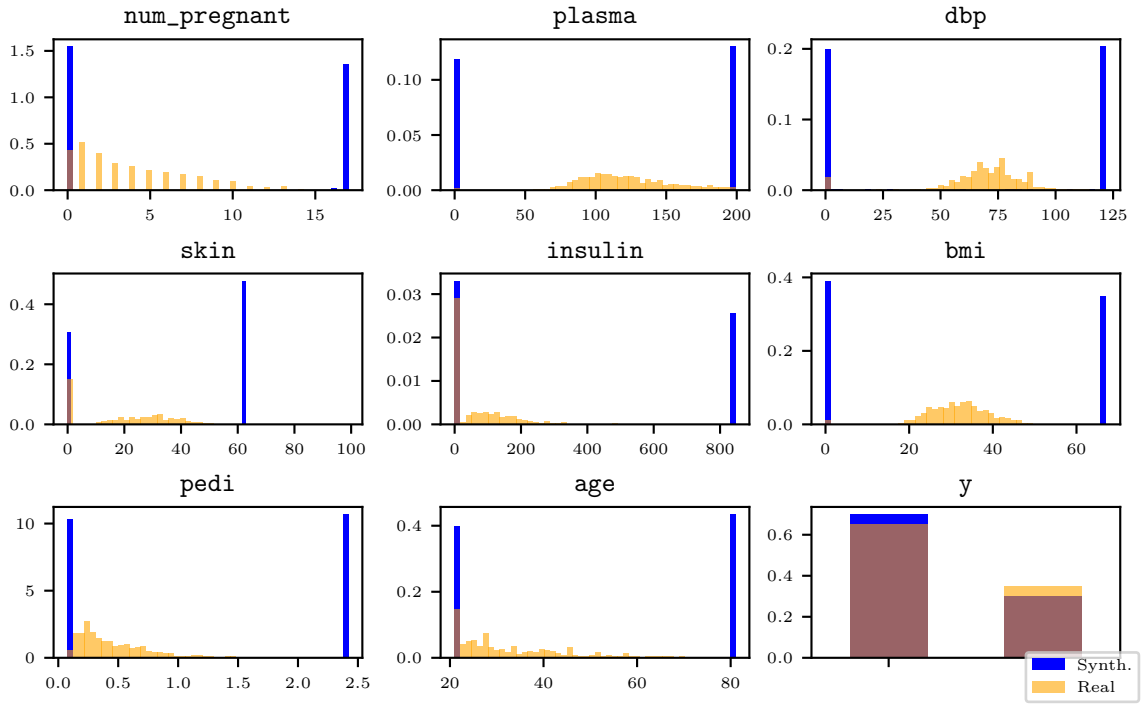


Figure 6.10: Histograms and bar plots of all the features in the **Diabetes** (**DI**) dataset, comparing the real dataset with a synthetic dataset generated by **Tabular diffusion** with **direct joint modelling**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

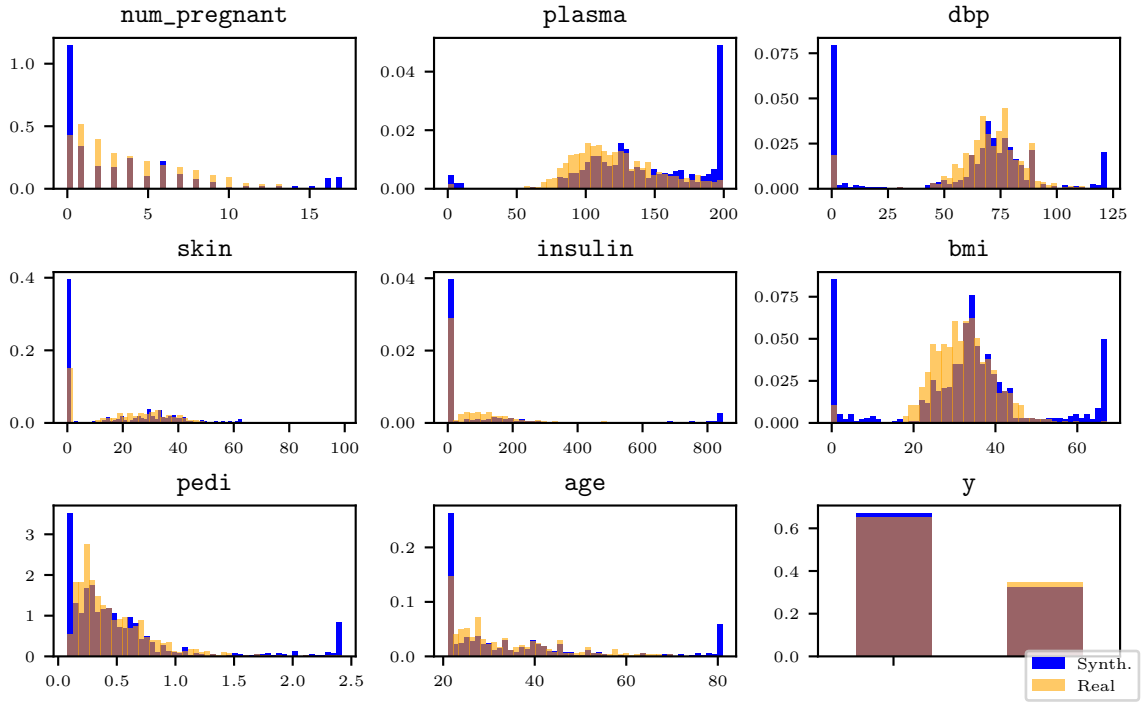


Figure 6.11: Histograms and bar plots of all the features in the **Diabetes** (**DI**) dataset, comparing the real dataset with a synthetic dataset generated by **Tabular Diffusion** with **conditional modelling**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

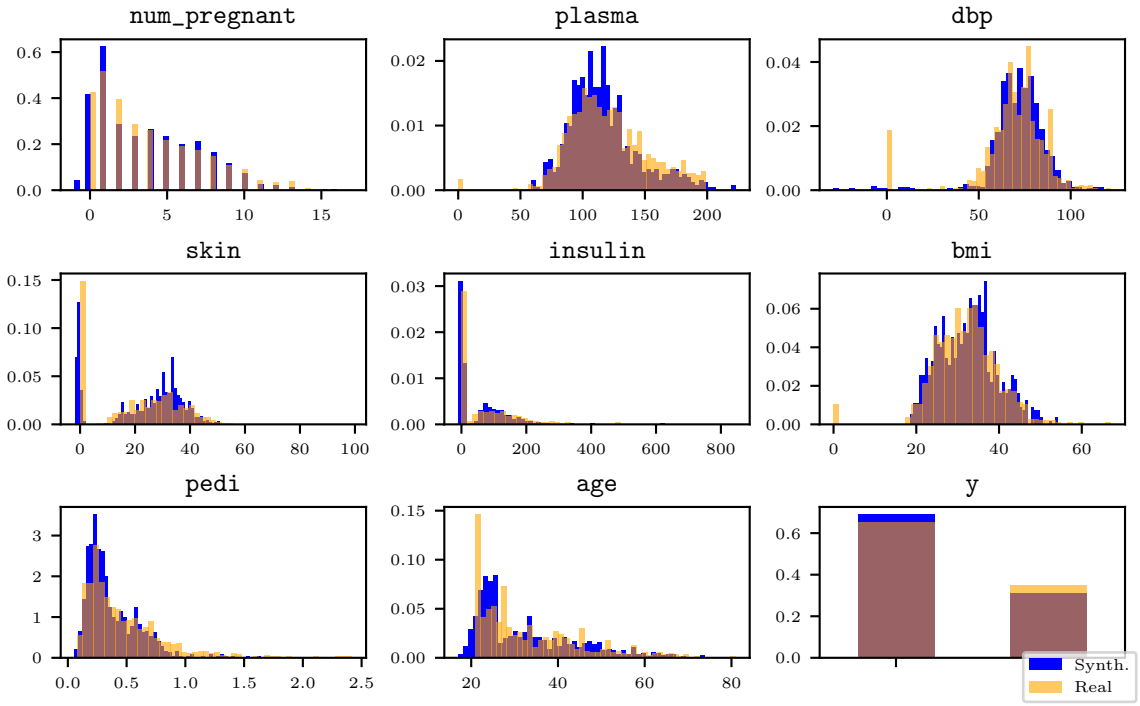


Figure 6.12: Histograms and bar plots of all the features in the **Diabetes** (**DI**) dataset, comparing the real dataset with a synthetic dataset generated by **TVAE**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

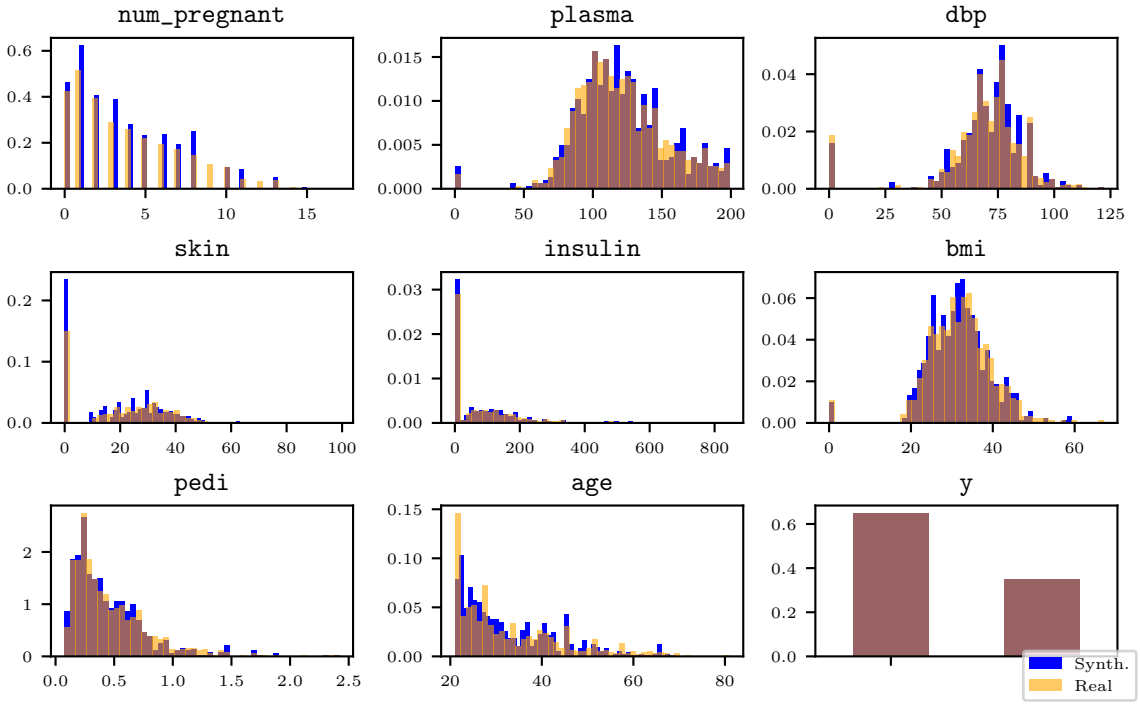


Figure 6.13: Histograms and bar plots of all the features in the **Diabetes** (**DI**) dataset, comparing the real dataset with a synthetic dataset generated by the two first steps of **MCCE**. The non-numerical feature values are normalized, such that the vertical axes show ratios out of 1.0.

6.2 Quantitative Evaluation

After the initial qualitative evaluation, we perform a quantitative evaluation based on ML efficacy, as described in Section 5.4. The ML efficacy metrics of each of the models, for each of the datasets, aggregated as empirical mean ± standard error over the five trials, are shown in Table 6.1. We highlight the best mean and standard error values in each column, among the models we evaluate, in bold.

Naming. We use the same naming conventions as in Section 6.1, and we introduce the term *Real*. This refers to a situation where we both train and test the classifier with the true real-world datasets. These values are added as another baseline for the performance of the models, representing a theoretical upper limit for the performance on synthetic data. Intuitively, we should not expect a model fitted with synthetic training data to perform better on the true test data compared to an identical model that is trained on real training data, when both these datasets are of identical size. This would indicate that the synthetic data is innately more valuable than the real data in a binary classification setting, which is not the case, as mentioned in Section 2.10.

Initial observation. Just as in the qualitative evaluation, our initial observation is that MCCE is superior to the rest of the models when it comes to all average ML efficacy metrics displayed in Table 6.1, except for the AUC in **DI**. MCCE not only attains the largest average values on all metrics across all datasets, excluding the previously mentioned exception, but it also achieves low standard errors across the five trials compared to the competing methods. This indicates that MCCE steadily outperforms its rivals, with less variable and better performance. However, notice that several of the results from the other models are within a few standard errors of the results from MCCE. Thus, the metrics in the table do not paint the full picture. To obtain a better idea of the metrics in each of the trials, we add some box plots in Figures 6.14 to 6.16. These are constructed with all the same metrics as used for Table 6.1. To be clear, the top and bottom limits of the *whiskers*, i.e. the vertical lines drawn through each box, denote the maximum and minimum attained metric values across the five trials, respectively. Moreover, the upper and lower limits of the boxes denote the third and first quartiles, respectively, while the orange, horizontal lines denote the median values. Note that the difference between the third and first quartile, i.e. the height of the box, is often referred to as the *interquartile range*. These plots still indicate that MCCE is superior in many cases, but not all, as the table might lead us to conclude at first sight. In the following, we dive deeper into a quantitative evaluation of the models' performance on each of the datasets, moving past our initial observations.

Adult Census (AD) dataset. When it comes to **AD**, the reported F_1 scores in Table 6.1 make it quite clear that the Tabular diffusion models, especially when trained using the conditional modelling technique, perform notably worse than the rest of the models. The other two metrics paint a similar picture, although the relative differences in performance between the models seem less notable. Overall it is clear from Table 6.1 that Tabular diffusion is inferior to both the baselines on this dataset, with the conditional modelling technique being the worst performer. This conclusion is also the most obvious one to make from the box plots in Figure 6.14, where any practitioner would prefer TVAE and MCCE over Tabular diffusion.

Churn Modelling (CH) dataset. A similar discussion can be performed for **CH**. First of all, notice that the means and standard errors of both the F_1 score and the accuracy are strikingly low for Tabular diffusion based on conditional modelling. The reason behind this is that, in all five trials, the CatBoost classifier that is trained on the synthetic data makes zero or close to zero positive predictions on the real test data. Thus, the classifier that is trained on this data is not able to discriminate between the positive

Table 6.1: ML efficacy for both techniques of Tabular diffusion, as well as TVAE and the two first steps of MCCE, for all three datasets. The reported numbers are aggregations over five different random seeds, given as empirical mean ± standard error. The *Real* rows show results when using true training and true test data. Upward arrows symbolize that higher is better.

(a) Adult Census (AD)

	Macro F_1 ↑	AUC ↑	Accuracy ↑
Real	0.7280 ± 0.0027	0.9320 ± 0.0003	0.8720 ± 0.0013
TabDiffJoint	0.4500 ± 0.0289	0.8210 ± 0.0123	0.8000 ± 0.0046
TabDiffCond	0.2130 ± 0.0379	0.7670 ± 0.0256	0.7630 ± 0.0072
TVAE	0.6230 ± 0.0344	0.8820 ± 0.0090	0.8150 ± 0.0134
MCCE	$\mathbf{0.7250 \pm 0.0031}$	$\mathbf{0.9300 \pm 0.0005}$	$\mathbf{0.8700 \pm 0.0017}$

(b) Churn Modelling (CH)

	Macro F_1 ↑	AUC ↑	Accuracy ↑
Real	0.9200 ± 0.0017	0.8630 ± 0.0006	0.8630 ± 0.0031
TabDiffJoint	0.9010 ± 0.0037	0.7360 ± 0.0346	0.8200 ± 0.0061
TabDiffCond	$0.0010 \pm \mathbf{0.0022}$	$0.7200 \pm \mathbf{0.0017}$	$0.1790 \pm \mathbf{0.0009}$
TVAE	0.9080 ± 0.0061	0.8030 ± 0.0111	0.8420 ± 0.0090
MCCE	$\mathbf{0.9120} \pm 0.0037$	$\mathbf{0.8290} \pm 0.0050$	$\mathbf{0.8490} \pm 0.0065$

(c) Diabetes (DI)

	Macro F_1 ↑	AUC ↑	Accuracy ↑
Real	0.8370 ± 0.0096	0.8150 ± 0.0042	0.7790 ± 0.0130
TabDiffJoint	0.7850 ± 0.0255	0.6460 ± 0.0545	0.6600 ± 0.0169
TabDiffCond	0.5710 ± 0.0491	0.7480 ± 0.0268	0.5710 ± 0.0450
TVAE	0.7890 ± 0.0266	$\mathbf{0.8050} \pm 0.0053$	0.7270 ± 0.0275
MCCE	$\mathbf{0.8270 \pm 0.0099}$	$0.8010 \pm \mathbf{0.0024}$	$\mathbf{0.7640 \pm 0.0142}$

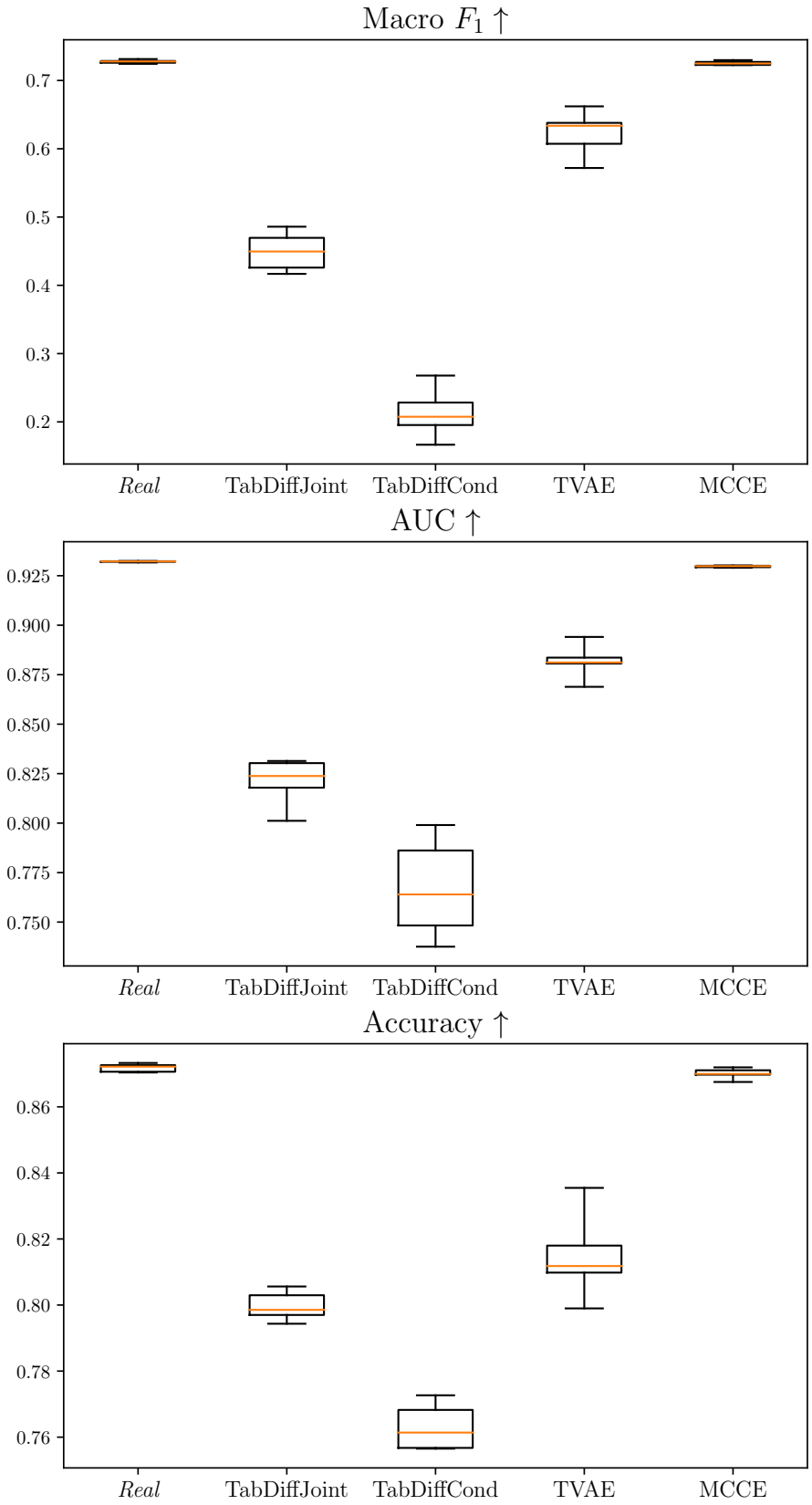


Figure 6.14: Box plots for the ML efficacy metrics across all five trials in the **Adult Census** (**AD**) dataset. The upward arrows in the titles symbolize that higher is better.

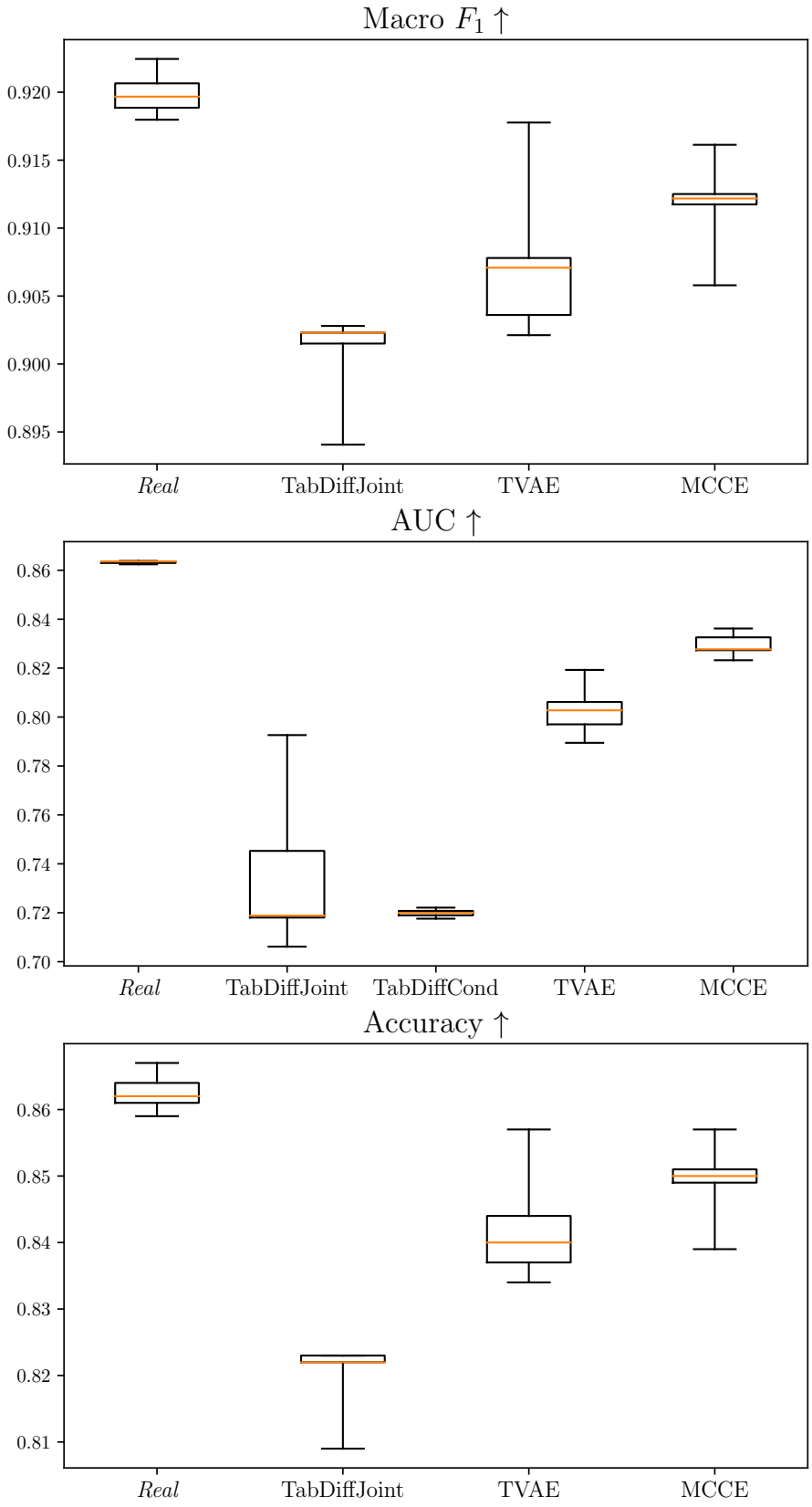


Figure 6.15: Box plots for the ML efficacy metrics across all five trials in the **Churn Modelling** (**CH**) dataset. The upward arrows in the titles symbolize that higher is better. Notice that we do not display box plots for Macro F_1 and Accuracy for Tabular diffusion with conditional modelling, because of the very low values in these cases.

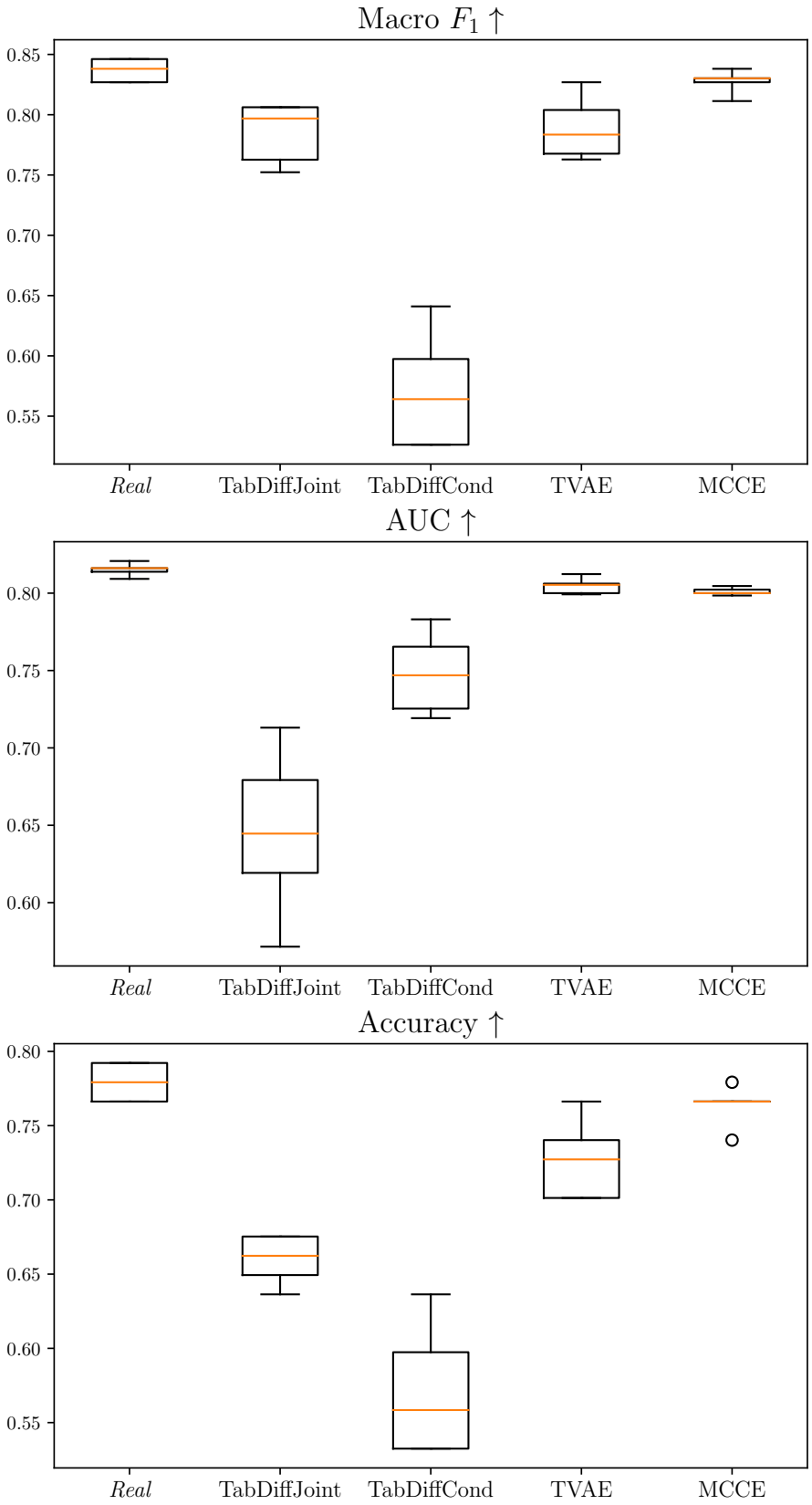


Figure 6.16: Box plots for the ML efficacy metrics across all five trials in the **Diabetes** (**DI**) dataset. The upward arrows in the titles symbolize that higher is better.

and negative binary responses in the real test data, signalling that the synthetic training data and the real test data are not well synchronized. Naturally, this reflects badly on the synthesizer, leading us to conclude that Tabular diffusion with conditional modelling is essentially useless for **CH**. Of course, this conclusion opposes our reasoning from the qualitative evaluation, where both Tabular diffusion models seem to perform decently. In contrast to the conditional variant, we conclude differently for the direct joint modelling variant of Tabular diffusion. This synthesizer seems to work quite well, relative to the baselines, according to Table 6.1. To complement the previous observations, the box plots in Figure 6.15 paint a more complete picture. In fact, the first quartiles in the results from TVAE and MCCE are consistently above the maximums in the results from Tabular diffusion, indicating that the baselines perform systematically better. Notice that we do not display box plots for F_1 and accuracy for Tabular diffusion with conditional modelling, because of the very low values in these cases. All in all, any practitioner would also prefer MCCE and TVAE over Tabular diffusion, based on these results.

Diabetes (DI) dataset. Finally, for **DI**, we make similar arguments as in the two other datasets. Tabular diffusion seems to perform worse than the baselines according to Table 6.1. This is substantiated by the box plots in Figure 6.16. Tabular diffusion with direct joint modelling seems to be better than its conditional counterpart according to F_1 score and accuracy, but worse on AUC. This is slightly curious and we are not sure why. Perhaps it is connected to the smaller number of observations in **DI**. In conclusion, we see no clear evidence that would endorse using Tabular diffusion over any of the baselines.

6.3 Final Evaluation

Finally, we provide a conclusive evaluation in this experiment. The most striking result from this experiment is that MCCE is overwhelmingly superior to the deep learning methods in most regards. It is superior in all qualitative investigations and on most quantitative metrics. Thus, we argue that MCCE is a highly competitive model. In the words of Kotelnikov et al. [67], admittedly used in a different context, this observation "raises the question if sophisticated deep generative models are needed". Beyond this conclusion, we discuss the overall performance of Tabular diffusion compared to TVAE, within the framework of the first experiment, in the following.

We do not observe any convincing results in favour of Tabular diffusion over TVAE, for any specific composition of data types, in contrast to what Kotelnikov et al. [67] report. For instance, in their paper, they provide fairly convincing absolute correlation matrices for **AD** and **CH**, in favour of Tabular diffusion over TVAE. Moreover, most of their ML efficacy values indicate that Tabular diffusion is superior to TVAE, with the exception of the F_1 score for **DI**. In our case, the qualitative analysis indicates that Tabular diffusion and TVAE perform relatively similarly overall, with the marginal distributions seemingly superior more often with Tabular diffusion, and vice versa for the correlation reconstructions. This is not true in all cases however, because the marginals in **DI** with Tabular diffusion are useless compared to TVAE, while the correlation reconstructions are marginally better with Tabular diffusion in **CH** compared to TVAE. Based on the quantitative analysis, TVAE is generally notably superior to Tabular diffusion. Hence, it seems like Tabular diffusion is better than TVAE when it comes to modelling the marginal distributions, but worse when it comes to modelling the joint probability, according to the qualitative evaluation of correlation and the ML efficacy values. One possible reason why the qualitative and quantitative analyses indicate different relative strengths and weaknesses of Tabular diffusion and TVAE, is the metrics we use when calculating the correlation matrices. Precisely, recall that we employ Pearson correlation, the correlation ratio and Theil's U statistic to measure the interactions between the features. Choosing

these metrics restricts the types of dependencies we are able to detect between the features, because each metric is designed to detect certain relationships. For example, Pearson correlation only takes linear dependence into account, which means that we cannot use the correlation matrices to detect non-linear relationships. As a side note, we could use, for example, *copulas* [80, 89] to investigate non-linear dependencies between continuous features. Consequently, we are not able to investigate all forms of dependencies between the features in a dataset in our qualitative analysis. In contrast, in a sense, ML efficacy implicitly investigates all types of relationships between the features in a dataset, since it uses the synthetic data on a downstream task, and compares the results to results from the same task solved with real data. Thus, the quantitative analysis is not inherently restricted, like the qualitative analysis, meaning that the relationships that are overlooked in the latter are ideally implicitly accounted for in the former. Moreover, we argue that high performance on the ML efficacy metrics should be regarded as more important than high performance in the qualitative analysis, as it indicates high performance on the task we are most interested in; modelling and sampling from $p^*(\boldsymbol{x}, y)$. Finally, we conclude that we find no evidence, in the first experiment, to promote the idea that Tabular diffusion is superior to any of the baselines, especially MCCE, on the task of modelling $p^*(\boldsymbol{x}, y)$ and synthesizing data from it.

Before moving on, we also discuss the relative performance of the two techniques we introduce for Tabular diffusion. We argue that the evidence is sufficient to conclude that direct joint modelling yields marginally better performance than conditional modelling. As stated, in the qualitative analysis, the two models seem hard to distinguish, but their relative performances in the quantitative analysis are quite different in many cases. In fact, in most cases, the direct joint modelling technique yields superior ML efficacy values. Thus, we argue that direct joint modelling is a better technique to use to construct a generative model with Tabular diffusion, compared to conditional modelling. However, this matter should be investigated further in order to gain complete confidence in the conclusion.

Chapter 7

Results — Generating Counterfactuals

In this chapter, we demonstrate and discuss results from the second experiment described in Chapter 5, on generating counterfactuals. Specifically, these results are used to evaluate the performance of Diff-MCCE on the entire three-step process outlined in Chapter 4.

Table 7.1 shows the obtained values of the counterfactual metrics described in Section 5.5, for each dataset and each model. In addition, notice that we do not perform a systematic qualitative evaluation, like in Chapter 6, but we add Table 7.2 to show some examples of factuals and their corresponding counterfactuals, generated from each of the synthesizers. In these examples, we observe that some of the counterfactuals are rather similar, while others are quite different. We cannot establish if some of them are more reasonable than others with the naked eye, but perhaps a domain expert could argue in either direction about each of the proposed counterfactuals. However, notice that the actionability constraint is fulfilled in all these examples, which we have highlighted with grey vertical bars in Table 7.2. Moreover, observe that the validity constraint is also fulfilled in all these examples, because the predictions on the CEs are positive, in contrast to the negative predictions on the factuals. We deduce this from the rows $f_{\mathbf{AD}}(\cdot), f_{\mathbf{CH}}(\cdot)$ and $f_{\mathbf{DI}}(\cdot)$, which represent the binary predictions from the CatBoost models, after applying a discrimination threshold of 0.5, on the observations in each column. For clarity, we add the subscripts to explicitly highlight that separate classifiers are trained on each of the datasets.

Naming. We use the naming conventions from Section 6.1, except that the terms refer to the use of the different generative models followed by the post-processing steps outlined in Chapter 4. Specifically, post-processing in MCCE is performed as explained in Section 4.1, while post-processing in the three other cases is performed as explained in Section 4.2.

Quantitative evaluation. Observe in Table 7.1 that MCCE has the lowest reported average sparsity and Gower distance in all three datasets. In addition, MCCE provides explanations for all the factuals in all cases, except for one case in the first trial in **AD**. This exception is a consequence of the fact that none of the $K = 10000$ synthetic points are valid for the given factual, according to the predictions from the CatBoost model. This could likely have been solved by increasing K. Overall, MCCE displays the lowest average metrics and lowest standard errors, except for a case where TVAE displays a lower standard error in sparsity in **CH**. Thus, at first sight we conclude, similarly to in the first experiment, that MCCE steadily outperforms its rivals, with less variable and better performance. However, as discussed previously, Table 7.1 does not paint the entire picture. We construct and display some box plots in Figures 7.1 to 7.3, in order to complement the information from the table. Specifically, these box plots are constructed based on the

average metric values of sparsity and Gower distance over all the counterfactuals in each trial.

Generally, the deep learning based models suffer from worse values across the metrics, indicating that the explanations they are able to find are of higher cost compared to the counterfactuals that are calculated with MCCE. This is clear from both Table 7.1 and the box plots. However, perhaps the biggest downfall of the rest of the methods is that they are not reliably able to calculate explanations for all factuals in $\mathcal{H}$, in all datasets. There might be several reasons behind this. The most obvious reason is that the deep learning models are less successful generative models overall, as we concluded in Chapter 6. This deteriorates the performance of the MCCE-framework, because the two first steps rely on high generative capacity. Next, all models, except for MCCE, are not able to sample conditional to the fixed features, meaning that is it likely that the majority of the total number of generated samples, $K \cdot |\mathcal{H}|$, are not actionable. Thus, this majority is discarded during post-processing, leaving only actionable points. Then, if none of these actionable points are valid, there are no possible explanations left in $\mathbf{D}_{\boldsymbol{h}}$, meaning that the model cannot calculate an explanation for the corresponding factual $\boldsymbol{h} \in \mathcal{H}$. Another reason is that K is too small, i.e. an increase in K would likely increase N_{CE}. However, we do not know how much of an increase would be necessary, which is an issue that merits further research.

Interestingly, TVAE exhibits worse performance than Tabular diffusion when it comes to N_{CE} in **AD**. However, neither of these models are clearly better in **AD** when it comes to the other metrics in Table 7.1 or the box plots in Figure 7.1. When it comes to **CH** and **DI**, the Tabular diffusion models demonstrate worse performance than the two baselines in reference to N_{CE}. In addition, the two other metrics display quite large standard errors compared to TVAE and MCCE, indicating that Tabular diffusion is less reliable. In the box plots, we observe that most of the interquartile ranges, as well as the distances between the maximums and minimums, are larger in Tabular diffusion than in the baselines, which is in agreement with our previously mentioned observations. We declare no clear winner in terms of performance among the deep learning methods, in reference to sparsity and Gower distance, but we argue that TVAE is more reliable overall, because it can provide counterfactuals for larger sets of factuals.

As a technical side note, we exclude the final trial from the averages, standard errors and box plot calculations when working with Tabular diffusion in **DI**, since none of the 10 factuals are provided a counterfactual in this trial. Thus, we cannot calculate any of the metrics in these cases.

Conclusively, we argue that MCCE is the superior method for generating counterfactuals among the four variants we have investigated, representing a reliable method across all three datasets. It would have been interesting to compare this method to other SOTA on-manifold or algorithmic-based methods for generating CEs, similar to the comparison performed by Redelmeier et al. [107]. We have not done this because of lack of time, but it should be investigated further. When it comes to the relative performance of the deep learning models, it is not clear which is better. Based on sparsity and Gower distance, the difference between the three models is not large, but we argue that there is no evidence that supports the idea that Tabular diffusion is a better generative model than TVAE for use in the three-step MCCE-process. In fact, the low numbers of N_{CE} in many cases in Tabular diffusion compared to TVAE indicates that the latter is a more reliable choice. Finally, we find no clear evidence that supports that either of the Tabular diffusion techniques is better for use in Diff-MCCE.

Table 7.1: Average counterfactual performance metrics for 100 test observations from the Adult Census (**AD**) and Churn Modelling (**CH**) datasets, as well as for 10 test observations from the Diabetes (**DI**) dataset. Results are produced with $\boldsymbol{K = 10000}$. The reported numbers are aggregations over five different random seeds, given as empirical mean ± standard error. Downward arrows symbolize that lower is better, while upward arrows symbolize that higher is better.

(a) Adult Census (AD)

	Sparsity ↓	Gower ↓	N_{CE} ↑
TabDiffJoint	4.76 ± 0.12	2.56 ± 0.08	$[91, 92, 90, 95, 94]$
TabDiffCond	4.81 ± 0.22	2.58 ± 0.15	$[93, 78, 95, 93, 90]$
TVAE	4.95 ± 0.13	2.46 ± 0.10	$[79, 72, 88, 64, 79]$
MCCE	$\mathbf{3.43 \pm 0.08}$	$\mathbf{1.23 \pm 0.05}$	$[99, 100, 100, 100, 100]$

(b) Churn Modelling (CH)

	Sparsity ↓	Gower ↓	N_{CE} ↑
TabDiffJoint	4.38 ± 0.16	1.60 ± 0.25	$[87, 90, 90, 91, 90]$
TabDiffCond	4.20 ± 0.11	1.34 ± 0.07	$[89, 91, 90, 91, 90]$
TVAE	$4.56 \pm \mathbf{0.04}$	1.57 ± 0.07	$[99, 100, 100, 99, 100]$
MCCE	$\mathbf{3.33} \pm 0.08$	$\mathbf{0.68 \pm 0.02}$	$[100, 100, 100, 100, 100]$

(c) Diabetes (DI)

	Sparsity ↓	Gower ↓	N_{CE} ↑
TabDiffJoint	6.00 ± 0.82	2.23 ± 1.50	$[1, 1, 1, 1, 0]$
TabDiffCond	5.50 ± 1.00	0.78 ± 0.14	$[1, 1, 1, 1, 0]$
TVAE	$5.34 \pm \mathbf{0.09}$	0.88 ± 0.10	$[10, 10, 10, 10, 10]$
MCCE	$\mathbf{4.56 \pm 0.09}$	$\mathbf{0.61 \pm 0.05}$	$[10, 10, 10, 10, 10]$

Table 7.2: Comparison of three different counterfactuals for the same factual $\boldsymbol{h}$, in each of the three datasets. The rows $f_{\mathbf{AD}}(\cdot), f_{\mathbf{CH}}(\cdot)$ and $f_{\mathbf{DI}}(\cdot)$ represent predictions from the CatBoost binary classifiers trained on each of the datasets. We have shortened some of the feature names and categories in the Adult Census (**AD**) datasets to make the table fit vertically.

(a) Adult Census (AD)

	$\boldsymbol{h}$	TabDiffJoint	TabDiffCond	TVAE	MCCE
age	25	25	25	25	25
fnlwgt	188767	270018	218210	96857	104097
ed_num	12	13	13	13	13
cap_gain	0	26013	0	99325	13550
cap_loss	0	0	1933	0	0
h_p_w	45	52	16	49	45
workcl.	Priv.	Priv.	Priv.	Self-emp-n.	Priv.
mar_stat	Nev-mar.	Mar-civ.	Mar-civ.	Nev-mar.	Nev-mar.
occup.	Exec.	Other-s.	Sales	Prof.	Exec.
rel.	Not-fam.	Not-fam.	Husband	Not-fam.	Not-fam.
race	White	White	White	White	White
sex	Male	Male	Male	Male	Male
country	US	US	US	US	US
$f_{\mathbf{AD}}(\cdot)$	0	1	1	1	1

(b) Churn Modelling (CH)

	$\boldsymbol{h}$	TabDiffJoint	TabDiffCond	TVAE	MCCE
CreditScore	509	850	470	539	511
Age	46	46	46	46	46
Tenure	1	1	0	1	1
Balance	0.0	0.0	0.0	96236.7	104947.7
NumOfProducts	1	1	2	1	1
EstimatedSalary	71244.6	53670.2	114890.7	48596.9	55072.3
Geography	France	France	France	France	France
Gender	Female	Female	Female	Female	Female
HasCrCard	1	1	1	1	1
IsActiveMember	0	1	0	0	0
$f_{\mathbf{CH}}(\cdot)$	0	1	1	1	1

(c) Diabetes (DI)

	$\boldsymbol{h}$	TabDiffJoint	TabDiffCond	TVAE	MCCE
num_pregnant	0	0	0	0	0
plasma	177	199	146	108	134
dbp	60	0	69	60	60
skin	29	31	30	29	30
insulin	478	846	742	90	158
bmi	34.6	0.0	30.9	31.3	34.6
pedi	1.1	0.1	0.9	0.4	0.5
age	21	21	21	21	21
$f_{\mathbf{DI}}(\cdot)$	0	1	1	1	1

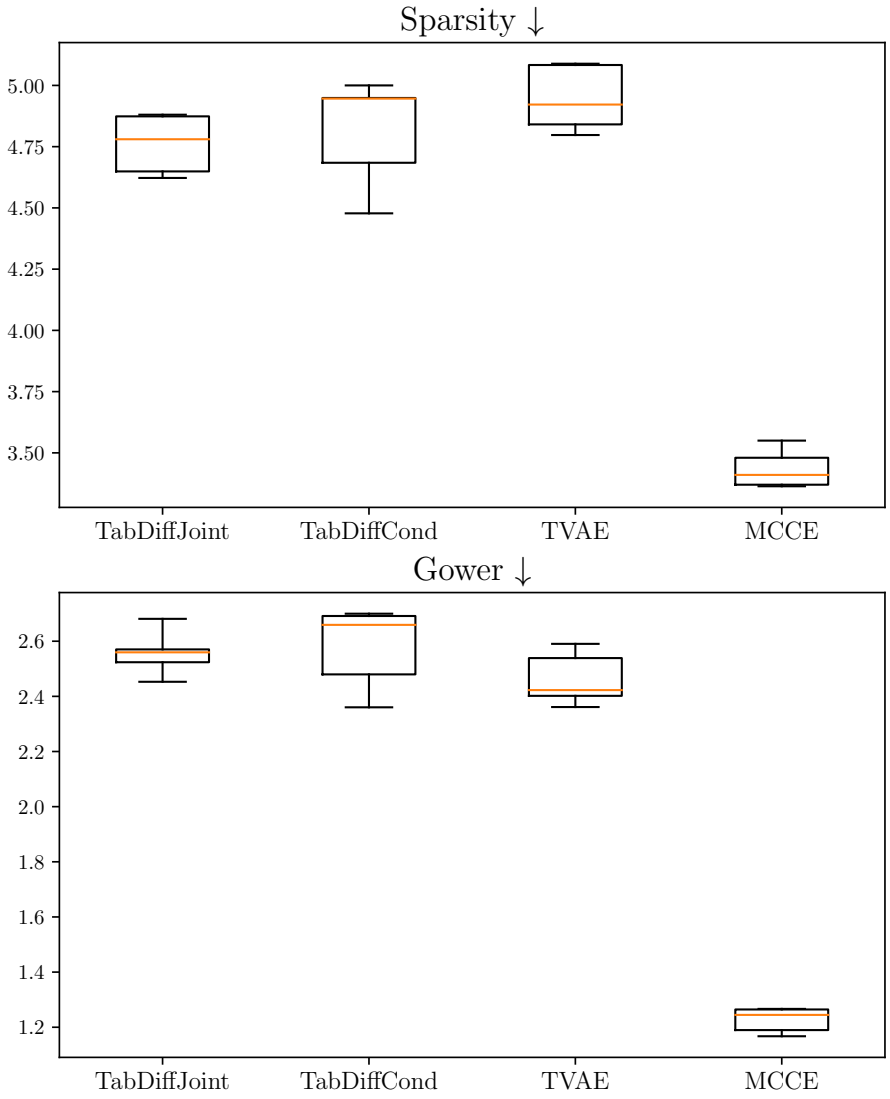


Figure 7.1: Box plots for the CE metrics across all five trials in the **Adult Census** (**AD**) dataset. The downward arrows in the titles symbolize that lower is better.

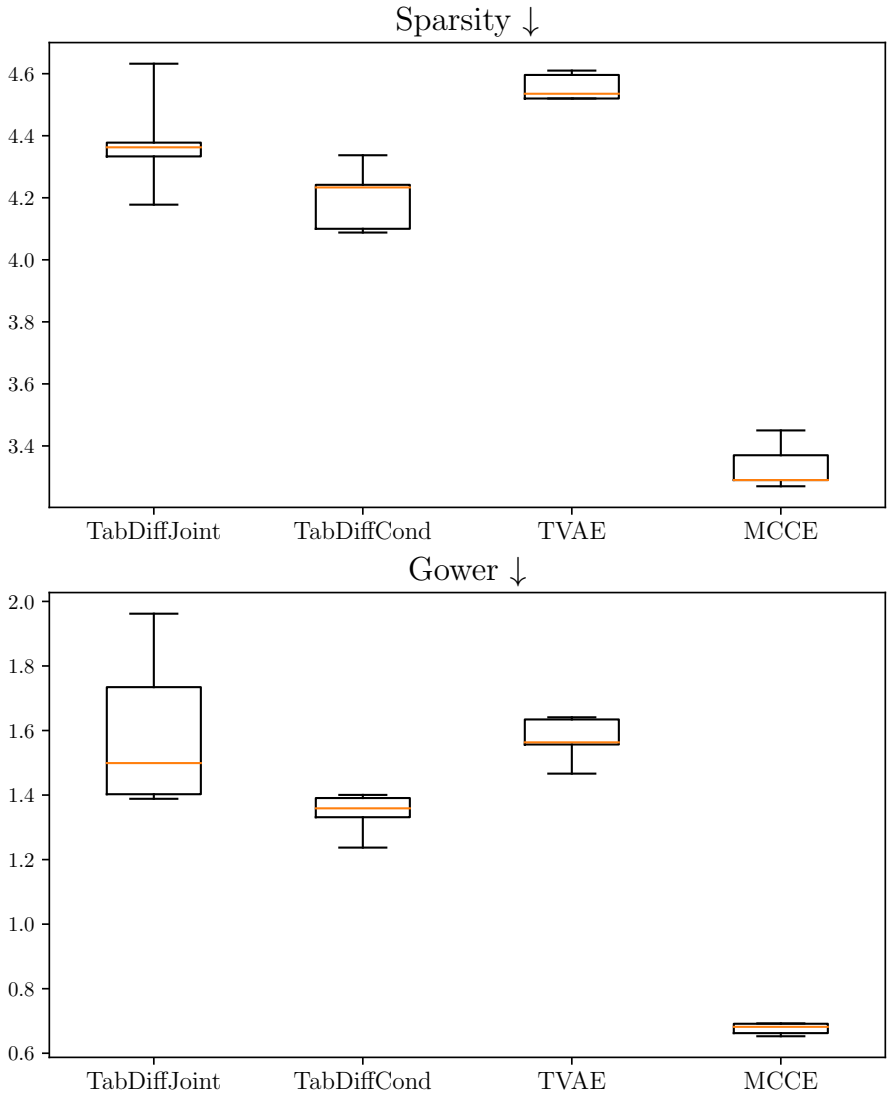


Figure 7.2: Box plots for the CE metrics across all five trials in the **Churn Modelling** (**CH**) dataset. The downward arrows in the titles symbolize that lower is better.

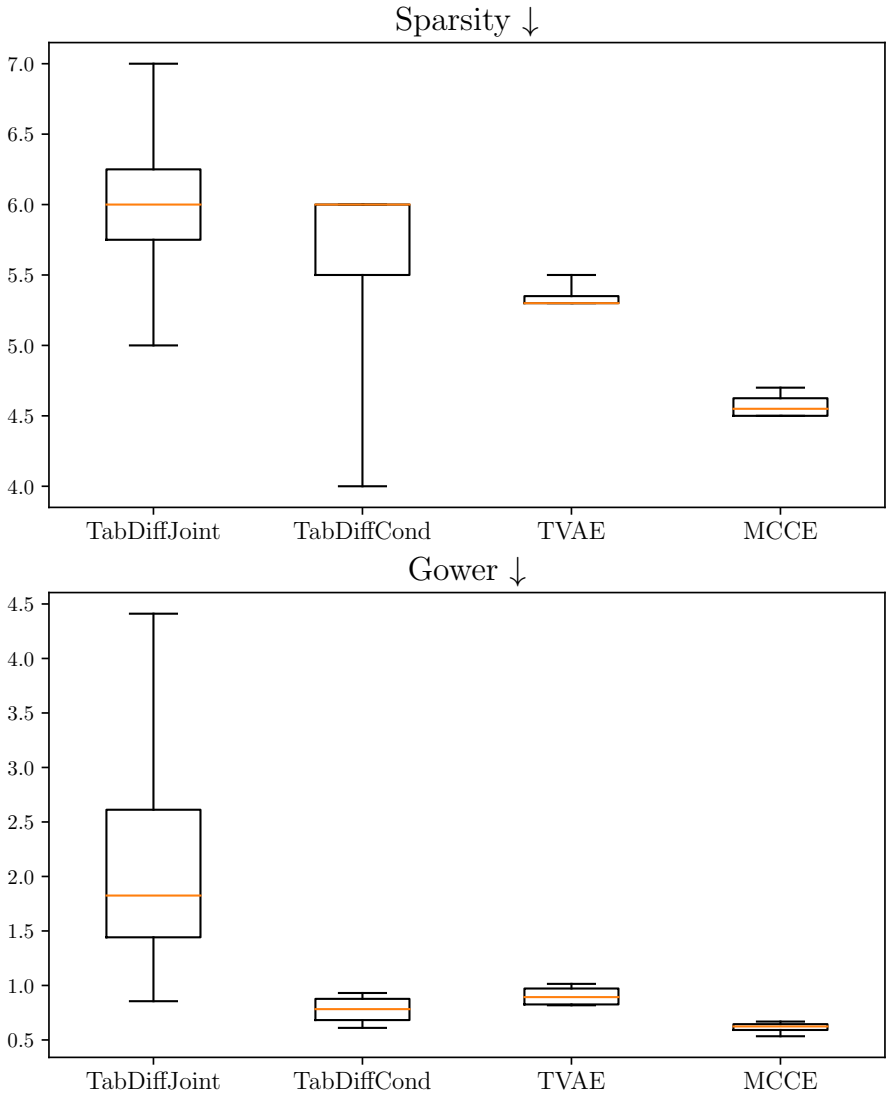


Figure 7.3: Box plots for the CE metrics across all five trials in the **Diabetes** (**DI**) dataset. The downward arrows in the titles symbolize that lower is better.

Chapter 8

Summary and Outlook

In this chapter, we summarize the work. Moreover, we acknowledge and discuss some limitations of our study, and provide a plethora of further research opportunities, which we have not been able to pursue due to lack of time.

Summary

In this thesis, we have developed a self-contained and accessible introduction to diffusion models. This was done by providing preliminaries on generative modelling, deep learning, autoencoders and their variational extensions ranging from a simple VAE to a more complex hierarchical variant. We have explained a specific diffusion model for tabular data, including assumptions and restrictions apt for applications within this data modality. In particular, we have elaborated a more detailed theoretical exposition of TabDDPM [67]. Finally, we have constructed a method for applying Tabular diffusion to generate counterfactual explanations, which are up-and-coming tools from XAI. In order to accomplish this, we first provided the unfamiliar reader with an introduction to XAI and counterfactual explanations, before adapting MCCE [107], which is our fundamental model of choice.

Beyond the theoretical work, we have performed two experiments to evaluate our diffusion model for tabular data. Precisely, we designed, implemented and evaluated two different strategies for applying this model in practice. For some perspective, we evaluated the performance of Tabular diffusion relative to two carefully selected baselines with previously proven outputs. Preliminaries on decision trees, gradient tree boosting, TVAE [146] and a selection of evaluation metrics has been provided, to facilitate evaluation of the diffusion model in relation to the baselines. The first experiment evaluated the generative capabilities of Tabular diffusion relative to TVAE and the tree-based model in MCCE. The second experiment evaluated the performance of Diff-MCCE on synthesizing counterfactuals relative to MCCE and an alternative variant of MCCE with TVAE as generative model.

The results from our experiments indicate that the shallow tree-based technique is superior to the deep generative frameworks we have investigated at modelling $p^*(\boldsymbol{x}, y)$, at sampling from it and at counterfactual synthesis using the modular three-step procedure introduced by Redelmeier et al. [107]. Moreover, we found no convincing evidence that Tabular diffusion performs better than any of the baselines at modelling the underlying joint data distribution and at sampling from it. In continuation, we argue that Tabular diffusion with direct joint modelling generally is superior to Tabular diffusion with conditional modelling at solving the first two steps in the three-step process. For generating tabular counterfactuals, which is the main application of diffusion models we are interested in, we found no evidence that Tabular diffusion performs better than the baselines.

The diffusion models seem to yield quite similar performance to TVAE in the second experiment, according to sparsity and Gower distance, but they appear unable to reliably calculate counterfactuals for many of the factuals. Finally, we found no compelling evidence that separate either of the Tabular diffusion models in performance on generating explanations.

The tree-based model introduced alongside the three-step method by Redelemeier et al. [107] is especially designed to tackle the problem of generating counterfactuals. Specifically, it implements a generator that synthesizes conditional to the fixed features, such that all K synthetic observations per factual are actionable before post-processing. With this in mind, none of the three deep learning models we investigated in Experiment II implement conditional generation, which seems to be the largest disadvantage of these methods. An addition of this feature to any of these models would likely notably improve upon the results we obtained in the second experiment. This can, for instance, be implemented in TVAE by combining it with CVAE [125], while it can be implemented in Tabular diffusion by adding guidance, either classifier guidance [20] or classifier-free guidance [48]. Moreover, we hypothesize that systematic hyperparameter tuning would yield marked performance increases in the deep learning methods. It seems probable that such tuning would not yield as pronounced increases in performance in MCCE, as the tree-based generator already performs quite closely to the real cases, as observed in the qualitative and quantitative analyses in Chapter 6.

Our systematic evaluation has not taken aspects like computational burden, implementation difficulty, theoretical complexity or ease of adoption into account. This is a clear limitation of our empirical study, because these are all critical aspects for utility in business, industry or academia. Despite this limitation, we argue that the tree-based baseline model is superior with regards to these characteristics as well. First of all, it is easier to understand, building on decision trees, which have been around for a long time. Moreover, it is easier to implement, making it easier to adopt, and we observed that the computational burden is clearly less than that of the deep learning variants, despite taking no exact measurements of metrics like training and sampling times. When it comes to the two deep learning models, TVAE seems superior to Tabular diffusion with respect to most, if not all, of these characteristics. Theoretically, Tabular diffusion is obviously more involved, as we observe from this work. In addition, it is more computationally demanding than TVAE, at least in our implementations. Also, we argue that it is more burdensome to implement. Thus, all in all, we found no evidence that indicates that the out-of-the-box performance of MCCE, in either of the three steps from Redelmeier et al. [107], is inferior to any of the deep generative frameworks, especially Tabular diffusion.

Based on our findings, we argue that MCCE represents a baseline that is hard to beat. This seems obvious from Experiment I, as the tree-based model is the superior generative model, even displaying results close to the real results in both the qualitative and quantitative analyses. Consequently, we urge researchers to use the tree-based model as a reference in future research on generative models for tabular data. When it comes to MCCE as a method for generating counterfactuals, we have not presented any evidence pointing at the level of performance of MCCE relative to other on-manifold or algorithmic-based methods. However, based on the performance reported by Redelmeier et al. [107], it is tempting to urge researchers to employ MCCE as a baseline also for this purpose.

Limitations

Keep in mind that our study has been under strict time constraints, and we need to acknowledge some additional important limitations of our work. Above all, we should interpret our results with caution. When it comes to our empirical study, our objective has been to provide an extensive and fair evaluation of Tabular diffusion, relative to the

reference models. However, we acknowledge that further evaluation is necessary in order to draw more certain and significant conclusions.

First, despite our efforts, we have focused on a rather limited set of real-world datasets in our assessment. Investigation of a larger group of datasets, containing, e.g., high-dimensional datasets, data with multivariate responses, multiclass classification datasets and regression datasets, would likely add more nuance to the discussion. Second, we have fixed certain neural network architectures, optimizers, batch sizes and other hyperparameters, in addition to certain classifiers. Implementation of a thorough procedure for optimizing these choices could yield more useful and realistic results for the practitioner. Third, we have chosen some popular metrics for evaluating generative models and counterfactual synthesizers. However, unsupervised evaluation of this sort remains challenging, as there exists no agreed upon best practice for this. For instance, *ablation* studies could be conducted to investigate Tabular diffusion with and without certain features. In addition, utilizing a larger set of metrics, for example for simultaneously measuring performance on the three essential conditions in the *generative learning trilemma* [145], would be beneficial for increased confidence in any conclusions. Fourth, the inherent stochasticity of the methods poses another challenge when it comes to drawing significant conclusions. We have aimed at controlling some of this variability during the quantitative evaluation by aggregating the results over five distinct trials, but the qualitative evaluation is lacking robustness to random seeds. Nevertheless, we urge researchers to implement evaluation procedures with meticulous care, in order to investigate the possible practical benefits of Tabular diffusion, and other deep generative models, for applications with tabular data.

Outlook

Conclusively, we provide some ideas for further research. We do not go into detail about each specific research direction, but focus on providing actionable ideas, with references if relevant. Naturally, there are plenty of extensions we have not been able to pursue, several of which we expect to improve the obtained results from the deep learning frameworks, especially Tabular diffusion.

Improving Tabular diffusion. To begin with, we consider some techniques aimed at improving the relatively simple Tabular diffusion model. The most obvious strategies for increasing the flexibility of the model are connected to the set of simplifying assumptions we made during our theoretical development in Chapter 3. For example, recall that we chose to define the variance schedule, β_t, $t \in \{1, \dots, T\}$, a priori, introducing the linear and cosine schedules. In contrast to this, we could either design a new a priori schedule or define the variance schedule as parameters that can be estimated during training [62]. Moreover, we could perhaps redefine the constant values β_1, β_T or s to better fit the tabular data modality, if we prefer working with the linear or cosine schedule, respectively. Subsequently, recall from Chapter 3 that we also assumed that the variance parameters in the reverse process densities, $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)$, $t \in \{2, \dots, T\}$, were known a priori. Precisely, we defined their values as $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \tilde{\Sigma}_t \mathbf{I}$, $t \in \{2, \dots, T\}$, where $\tilde{\Sigma}_t$ is a combination of the variance schedule parameters, and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) = a\mathbf{I}$, where $a \in \mathbb{R}$ is arbitrary. We imagine two possible strategies for redefining these parameters to increase flexibility. First, we could simply define them a priori, but combine it with learning the variance schedule. Estimation of the variance schedule would then, as a consequence, iteratively change the variance parameters, until training converges, which implicitly learns the variance parameters [62]. Second, they could be learnt explicitly, similarly to how we learn the mean parameters; by teaching a ML model to predict the parameter in each diffusion step. However, Nichol and Dhariwal [21] find that predicting the variances like this is difficult in practice, and instead defined the variance parameters as an interpolation between $\tilde{\Sigma}_t$ and

β_t at each diffusion step, where a vector combining the two quantities is output from a model. According to Nichol and Dhariwal [21] "learning variances of the reverse diffusion process allows sampling with an order of magnitude fewer forward passes with a negligible difference in sample quality, which is important for the practical deployment of these models". This leads us nicely into the next propositions, for improved sampling speed. For instance, we could reduce the number of reverse steps in Tabular diffusion [21, 75, 127], or we could introduce parallel computation of reverse process transitions, as recently proposed by Shih et al. [121].

Another simplifying assumption we made in Chapter 3 is to sample diffusion steps uniformly for training the reweighted loss function, L^{GD}_{simple}, in Gaussian diffusion. We could replace this uniform sampling with *importance sampling* [21], which might lead to better optimums. In continuation, in Chapter 5 we mentioned the sinusoidal embeddings [141], which have not been tuned. In the literature, it seems like most research after Ho et al. [47] simply employs this technique, without explicitly investigating other options. For instance, could we provide the neural network with the necessary diffusion step awareness by simply adding or concatenating fractions in the range $[1, T]$ to the features? We argue that the effects of the sinusoidal embeddings, as well as alternative methods, should be investigated in further research. In addition, it would be interesting to discuss why these embeddings, as well as the linear embedding of the label in the conditional modelling technique, are *added* to the diffused input transformation, making up the MLP in the reverse process. Why are they, for instance, not concatenated? Would this yield completely different performance? This should also be investigated. In connection to these ideas, one could investigate use of distinct diffusion models for different diffusion steps, which, in the limiting case, effectively removes the need for a diffusion step embedding [138]. As far as we know, this has not been investigated to a great extent, because of computational issues, but we expect a tractable implementation of this idea to notably increase performance in diffusion models.

Moving on, we suggest further research on designing new variants of diffusion models based on different distributional or parametric assumptions. For example, for the continuous features, it is not clear that a Gaussian assumption would work well as decoding distribution for all features. Defining a different decoding distribution, either throughout the entire reverse process, consequently having to change the forward process as well, or just by changing the final decoder distribution, could yield fruitful results for features with less regular distributions, even after applying a Quantile transformation. Naturally, since we decided to treat all numerical features as continuous, these reflections also hold for integer values. In addition, Multinomial diffusion can be extended into a more general framework for discrete state-space diffusion, moving beyond forward information destruction with uniform probabilities, as introduced by Austin et al. [3].

Data pre-processing. An ever-present matter in data analysis is data pre-processing. We suggest some extensions to the methods we applied, which could be worth pursuing in future research. First, the Gaussian Quantile transformer we applied to the numerical features is robust, but it may distort linear correlations in a dataset, because it is a non-linear transformation. This might limit the correctness of the investigated correlation matrices Figure 6.1, despite the fact that we used the same transformation on all datasets when modelling with Tabular diffusion and MCCE. However, in retrospect, we could have applied the Quantile transformation before using the mode-specific normalization scheme in TVAE as well, which one could argue would make the comparison between TVAE and the two other methods slightly more appropriate. This idea leads to a more general suggestion, where combinations of pre-processing transformations can be applied. For instance, the Gaussian Quantile transformation is approximate, meaning that its output is only approximately standard Gaussian. Thus, this transformation could, e.g., be followed by an

application of *standardization*, perhaps yielding data closer to standard Gaussian, before training the models. This could improve convergence of MLP-training [71] or improve the general performance of models with Gaussian, or other, regularity assumptions. Moreover, another interesting idea that came to mind is to apply the mode-specific normalization introduced by Xu et al. [146] to data that is modelled with Gaussian diffusion. There is no obvious way of doing this, but our idea is the following: pre-process the continuous data using the mode-specific scheme and define separate Gaussian diffusion processes for each Gaussian in the mixture model from the pre-processing scheme. Keeping the notation from Section 2.9 in mind, this essentially requires M_j forward processes for each continuous feature, $X^j \in \boldsymbol{X}^{cont}$, analogously to how we define C Multinomial diffusion processes in Tabular diffusion. After defining the forward processes, proceed by training the reverse processes. Notice that this is a mere idea at this stage — we do not know if it has any merit. Nevertheless, in further works, other continuous transformations should be investigated. Moving on from the continuous features, pre-processing categorical features is another issue that should be investigated. One-hot encoding (OHE) is regarded as the standard technique for this purpose. However, as Lee et al. [73] point out; treating discrete variables in continuous spaces, like is the case after OHE, may be suboptimal, essentially compromising what they call the "inter-column correlations" between the continuous and discrete variables. Moreover, recall that one of the reasons we chose to treat integer features as if they were continuous is related to compute and memory; when using OHE on discrete random variables with large sample spaces, the memory and compute requirements can be very large. Thus, perhaps alternative pre-processing methods for categorical data could facilitate treating integers as categorical. For instance, Zheng and Charoenphakdee [147] propose two other techniques for pre-processing categorical data; *analog bits encoding* and *feature tokenization*. Similar techniques should be investigated in conjunction with Tabular diffusion, in order to more effectively handle continuous and categorical features at once.

New design for tabular data. Beyond the tweaks in Tabular diffusion described previously, larger structural changes in the design of the diffusion model can be made. Lee et al. [73] recently introduced a novel technique for combining Gaussian and Multinomial diffusion models, which we argue is a more complicated extension of the relatively simple design we highlighted in Section 3.6. Essentially, they propose two *co-evolving* diffusion models, one Gaussian and one Multinomial, for modelling continuous and discrete columns simultaneously. The individual diffusion models work conditionally to each other, at each diffusion step, in both the forward and reverse processes — hence they are *co-evolving*. This exciting novel twist on diffusion models for tabular data should be compared to work in this thesis in future research, but we expect it to perform notably better than Tabular diffusion on both qualitative and quantitative metrics, based on initial impressions from Lee et al. [73].

Theoretical extensions. We argue that a great advantage of diffusion models is that they have a vast theoretical underpinning, as exemplified in Section 3.3, where we note that diffusion models can be equivalently discussed from a perspective of score-based generative models [128, 129, 130]. Further research should focus on bridging the gap between these two perspectives [47, 50], with tabular data as a focal point. In addition, we suggest further research on *latent diffusion models* (LDMs), for tabular data. To the best of our knowledge, research of this sort has not been conducted. LDMs have recently been used to build high-performing models in, e.g., image generation [110] and video synthesis [10], effectively solving some of the most pressing issues in diffusion models, like scalability and computational demands. We believe such models could be worth pursuing for tabular data applications, especially for high-dimensional data.

Privacy. Finally, we mention a research direction we have not considered in detail

previously in this work; protection of privacy. The societal impact of high-performing generative models is not solely positive, because an increase in realistic fake data might lead to more severe misuse, including abuse against individuals. As a consequence, the topic of privacy protection needs to be studied [73]. In fact, several deep generative models have been proposed for synthesizing tabular data with *differential privacy* [28] guarantees [51, 57, 74]. Some research on differentially private diffusion models has been conducted recently as well [23, 39, 77], but it has mostly been restricted to image data, consistent with the modality that diffusion models predominantly have been developed for. To the best of our knowledge, no thorough research has been conducted on diffusion models with inherent privacy protection for tabular data. As a consequence, further research should be conducted on this matter, as we predict that this is a domain where deep generative models could be highly beneficial over shallow methods like the tree-based model from MCCE. Such a result would give an affirmative answer to the previously mentioned question raised by Kotelnikov et al. [67]; "if sophisticated deep generative models are needed".

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://tensorflow.org/`.

[2] M. Augustin, V. Boreiko, F. Croce, and M. Hein. "Diffusion Visual Counterfactual Explanations". In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022.

[3] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg. *Structured Denoising Diffusion Models in Discrete State-Spaces*. Feb. 2023. arXiv: `2107.03006v3 [cs.LG]`.

[4] AWS. *Featured AWS Sports and Entertainment Partnerships — Why F1 Chooses AWS*. Accessed 1 May 2023. URL: `https://aws.amazon.com/sports/f1/`.

[5] M. J. Bayarri and J. O. Berger. "The Interplay of Bayesian and Frequentist Analysis". In: *Statistical Science* 19.1 (Feb. 2004), pp. 58–80.

[6] BBC. *Uber's Self-driving Operator Charged Over Fatal Crash*. Accessed 1 May 2023. Sept. 2020. URL: `https://bbc.com/news/technology-54175359`.

[7] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961. ISBN: 9780691079011.

[8] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 978-0387-31073-2.

[9] P. Blanchard, D. J. Higham, and N. J. Higham. "Accurately Computing the Log-sum-exp and Softmax Functions". In: *IMA Journal of Numerical Analysis* 41.4 (Oct. 2021), pp. 2311–2330.

[10] A. Blattmann, R. Rombach, H. Ling, T. Dockhorn, S. W. Kim, S. Fidler, and K. Kreis. "Align Your Latents: High-Resolution Video Synthesis With Latent Diffusion Models". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2023.

[11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Routledge, 1984.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. "SMOTE: Synthetic Minority Over-Sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357.

[13] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA, 2016, pp. 785–794.

[14] N. Chinchor. "MUC-4 Evaluation Metrics". In: *Fourth Message Understanding Conference (MUC-4): Proceedings of a Conference Held in McLean, Virginia, June 16-18*. 1992.

[15] M. cmglee. *File:Roc curve.svg*. Accessed 21 Sept. 2023. June 2018. URL: https://commons.wikimedia.org/wiki/File:Roc_curve.svg.

[16] M. Coulter. *AI Experts Disown Musk-backed Campaign Citing Their Research*. Accessed 1 May 2023. Apr. 2023. URL: https://reuters.com/technology/ai-experts-disown-musk-backed-campaign-citing-their-research-2023-03-31/.

[17] Creative Commons. *Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)*. Accessed 1 May 2023. URL: https://creativecommons.org/licenses/by-sa/4.0/.

[18] C. Cremer, X. Li, and D. Duvenaud. "Inference Suboptimality in Variational Autoencoders". In: *Proceedings of the 35th International Conference on Machine Learning, ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1078–1086.

[19] L. Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[20] P. Dhariwal and A. Nichol. "Diffusion Models Beat GANs on Image Synthesis". In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021.

[21] P. Dhariwal and A. Nichol. "Improved Denoising Diffusion Probabilistic Models". In: *Proceedings of the 38th International Conference on Machine Learning, ICML*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 8162–8171.

[22] M. F. Dixon, I. Halperin, and P. Bilokon. *Machine Learning in Finance — From Theory to Practice*. Vol. 1. Springer, 2021.

[23] T. Dockhorn, T. Cao, A. Vahdat, and K. Kreis. *Differentially Private Diffusion Models*. Oct. 2022. arXiv: 2210.09929 [stat.ML].

[24] M. Downs, J. L. Chu, Y. Yacoby, F. Doshi-Velez, and W. Pan. "CRUDS: Counterfactual Recourse Using Disentangled Subspaces". In: *ICML Workshop on Human Interpretability in Machine Learning*. 2020, pp. 1–23.

[25] J. Drechsler and J. P. Reiter. "An Empirical Evaluation of Easily Implemented, Nonparametric Methods for Generating Synthetic Datasets". In: *Computational Statistics & Data Analysis* 55.12 (2011), pp. 3232–3243.

[26] D. Dua and C. Graff. *UCI Machine Learning Repository*. 2017. URL: https://archive.ics.uci.edu.

[27] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159.

[28] C. Dwork, A. Roth, et al. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407.

[29] B. Eastwood. *What Is Synthetic Data — And How Can It Help You Competitively?* Accessed 1 May 2023. Jan. 2023. URL: https://mitsloan.mit.edu/ideas-made-to-matter/what-synthetic-data-and-how-can-it-help-you-competitively.

[30] European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. Vol. 59. Official Journal of the European Union, 2016.

[31] M. J. Evans and J. S. Rosenthal. *Probability and Statistics — The Science of Uncertainty*. Second edition. W. H. Feeman, 2009.

[32] T. Fawcett. "An Introduction to ROC Analysis". In: *Pattern Recognition Letters* 27.8 (2006). ROC Analysis in Pattern Recognition, pp. 861–874.

[33] R. A. Fisher. *Statistical Methods for Research Workers*. Olived & Boyd, 1925.

[34] S. Flores. *Variational Autoencoders are Beautiful*. Accessed 1 May 2023. Apr. 2019. URL: `https://www.compthree.com/blog/autoencoder/`.

[35] Y. Freund and R. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Computational Learning Theory. Lecture Notes in Computer Science*. Vol. 904. EuroCOLT'95. Springer, 1995, pp. 23–37.

[36] Y. Freund and R. Schapire. "Experiments With a New Boosting Algorithm". In: *Proceedings of the Thirteenth International Conference on Machine Learning, ICML*. 1996, pp. 148–156.

[37] Future of Life Institute. *Pause Giant AI Experiments: An Open Letter*. Accessed 1 June 2023. Mar. 2023. URL: `https://futureoflife.org/open-letter/pause-giant-ai-experiments/`.

[38] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. Third edition. Chapman and Hall/CRC, 2013.

[39] S. Ghalebikesabi, L. Berrada, S. Gowal, I. Ktena, R. Stanforth, J. Hayes, S. De, S. L. Smith, O. Wiles, and B. Balle. *Differentially Private Diffusion Models Generate Useful Synthetic Images*. Feb. 2023. arXiv: `2302.13861 [cs.LG]`.

[40] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014.

[42] Google Research, Brain Team. *Imagen*. Accessed 1 May 2023. URL: `https://imagen.research.google/`.

[43] R. Guidotti. "Counterfactual Explanations and How To Find Them: Literature Review and Benchmarking". In: *Data Mining and Knowledge Discovery* (2022).

[44] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Vol. 2. Corrected 12th printing - 13 Jan. 2017. Springer, 2009.

[45] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-VAE: Learning Basic Visual Concepts With a Constrained Variational Framework". In: *International Conference on Learning Representations, ICLR*. 2017.

[46] G. E. Hinton. *Neural Networks for Machine Learning*. Coursera, video lectures. 2012.

[47] J. Ho, A. Jain, and P. Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020.

[48] J. Ho and T. Salimans. "Classifier-Free Diffusion Guidance". In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. Dec. 2021.

[49] E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling. "Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions". In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021.

[50] C.-W. Huang, J. H. Lim, and A. C. Courville. "A Variational Perspective on Diffusion-Based Generative Models and Score Matching". In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021.

[51] J. Hyeong, J. Kim, N. Park, and S. Jajodia. "An Empirical Study on the Membership Inference Attack Against Tabular Data Synthesis Models". In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. CIKM '22. Atlanta, GA, USA, 2022, pp. 4064–4068.

[52] IBM. *IBM Watson is AI for Smarter Business*. Accessed 1 May 2023. URL: `https://ibm.com/watson`.

[53] J. Jacod and P. Protter. *Probability Essentials*. Second printing of the second edition. Springer Science & Business Media, 2004.

[54] E. Jang, S. Gu, and B. Poole. "Categorical Reparameterization with Gumbel-Softmax". In: *International Conference on Learning Representations, ICLR*. 2017.

[55] G. Jeanneret, L. Simon, and F. Jurie. "Diffusion Models for Counterfactual Explanations". In: *Proceedings of the Asian Conference on Computer Vision, ACCV*. Dec. 2022, pp. 858–876.

[56] T. Jebara. *Machine Learning: Discriminative and Generative*. Springer Science & Business Media, 2004. ISBN: 978-1-4613-4756-9.

[57] J. Jordon, J. Yoon, and M. Van Der Schaar. "PATE-GAN: Generating Synthetic Data With Differential Privacy Guarantees". In: *International Conference on Learning Representations, ICLR*. 2019.

[58] S. Joshi, O. Koyejo, W. Vijitbenjaronk, B. Kim, and J. Ghosh. *Towards Realistic Individual Recourse and Actionable Explanations in Black-Box Decision Making Systems*. July 2019. arXiv: `1907.09615 [cs.LG]`.

[59] S. Kaufman, S. Rosset, and C. Perlich. "Leakage in Data Mining: Formulation, Detection, and Avoidance". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. San Diego, California, USA, 2011, pp. 556–563.

[60] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[61] J. Kim, C. Lee, and N. Park. "STaSy: Score-based Tabular Data Synthesis". In: *International Conference on Learning Representations, ICLR*. 2023.

[62] D. Kingma, T. Salimans, B. Poole, and J. Ho. "Variational Diffusion Models". In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021.

[63] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations, ICLR*. 2015.

[64] D. P. Kingma and M. Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[65] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations, ICLR*. 2014.

[66] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. "Improved Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems*. Vol. 29. 2016.

[67] A. Kotelnikov, D. Baranchuk, I. Rubachev, and A. Babenko. *TabDDPM: Modelling Tabular Data With Diffusion Models*. Sept. 2022. arXiv: 2209.15421 [cs.LG].

[68] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis. "Machine Learning Applications in Cancer Prognosis and Prediction". In: *Computational and Structural Biotechnology Journal* 13 (2015), pp. 8–17.

[69] M. A. Kramer. "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks". In: *AIChE journal* 37.2 (1991), pp. 233–243.

[70] J. Lasserre, C. Bishop, and T. Minka. "Principled Hybrids of Generative and Discriminative Models". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 1. 2006, pp. 87–94.

[71] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. "Efficient BackProp". In: *Neural networks: Tricks of the Trade*. Springer, 2002, pp. 9–48.

[72] Y. Lecun, C. Cortes, and C. J. Burges. *THE MNIST DATABASE of Handwritten Digits*. Accessed 3 Mar. 2023. URL: http://yann.lecun.com/exdb/mnist.

[73] C. Lee, J. Kim, and N. Park. *CoDi: Co-evolving Contrastive Diffusion Models for Mixed-type Tabular Synthesis*. Apr. 2023. arXiv: 2304.12654 [cs.LG].

[74] J. Lee, J. Hyeong, J. Jeon, N. Park, and J. Cho. "Invertible Tabular GANs: Killing Two Birds with One Stone for Tabular Data Synthesis". In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021.

[75] C. Lu, Y. Zhou, F. Bao, J. Chen, C. LI, and J. Zhu. "DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps". In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022.

[76] C. Luo. *Understanding Diffusion Models: A Unified Perspective*. Aug. 2022. arXiv: 2208.11970 [cs.LG].

[77] S. Lyu, M. Vinaroz, M. F. Liu, and M. Park. *Differentially Private Latent Diffusion Models*. May 2023. arXiv: 2305.15759 [stat.ML].

[78] C. J. Maddison, D. Tarlow, and T. Minka. "A* Sampling". In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014.

[79] C. J. Maddison, A. Mnih, and Y. W. Teh. "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables". In: *International Conference on Learning Representations, ICLR*. 2017.

[80] G. Marti, S. Andler, F. Nielsen, and P. Donnat. "Exploring and Measuring Non-linear Correlations: Copulas, Lightspeed Transportation and Clustering". In: *Proceedings of the Time Series Workshop at NIPS 2016*. Vol. 55. Proceedings of Machine Learning Research. PMLR, 2017, pp. 59–69.

[81] L. Mason, J. Baxter, P. Bartlett, and M. Frean. "Boosting Algorithms as Gradient Descent". In: *Advances in Neural Information Processing Systems*. Vol. 12. 1999.

[82] Microsoft Corporate Blogs. *Microsoft and OpenAI Extend Partnership*. Accessed 5 June 2023. Jan. 2023. URL: https://blogs.microsoft.com/blog/2023/01/23/microsoftandopenaiextendpartnership/.

[83] T. Miller. "Explanation in Artificial Intelligence: Insights From the Social Sciences". In: *Artificial Intelligence* 267 (2019), pp. 1–38.

[84] S. Mohamed and B. Lakshminarayanan. *Learning in Implicit Generative Models*. Feb. 2017. arXiv: 1610.03483v4 [stat.ML].

[85] S. Mollman. *ChatGPT Gained 1 Million Users in Under a Week. Here's Why the AI Chatbot is Primed to Disrupt Search as We Know It*. Accessed 3 June 2023. Dec. 2022. URL: https://finance.yahoo.com/news/chatgpt-gained-1-million-followers-224523258.html.

[86] C. Molnar. *Interpretable Machine Learning*. Lulu.com, 2020.

[87] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[88] National Transportation Safety Board (NTSB). *'Inadequate Safety Culture' Contributed to Uber Automated Test Vehicle Crash - NTSB Calls for Federal Review Process for Automated Vehicle Testing on Public Roads*. Accessed 2 May 2023. Nov. 2019. URL: https://ntsb.gov/news/press-releases/Pages/NR20191119c.aspx.

[89] R. B. Nelsen. *An Introduction to Copulas*. Vol. 139. Springer Science & Business Media, 2013.

[90] Netflix. *Research Areas — Machine Learning*. Accessed 1 May 2023. URL: https://research.netflix.com/research-area/machine-learning.

[91] NeurIPS. *Interpretable Machine Learning*. Accessed 3 May 2023. Dec. 2017. URL: https://nips.cc/Conferences/2017/Schedule?showEvent=8744.

[92] J. Nocedal and S. J. Wright. *Numerical Optimization*. Vol. 2. Springer, 2006.

[93] A. J. Ohrt. *Explainable AI: A Comparison of Generative On-Manifold Methods for Counterfactual Explanations*. Project report in TMA4500 — Industrial Mathematics, Specialization Project. Department of Mathematical Sciences, NTNU – Norwegian University of Science and Technology, Dec. 2022.

[94] L. H. B. Olsen, I. K. Glad, M. Jullum, and K. Aas. "Using Shapley Values and Variational Autoencoders to Explain Predictive Models With Dependent Mixed Features". In: *Journal of Machine Learning Research* 23.213 (2022), pp. 1–51.

[95] OpenAI. *DALL·E 2 is an AI system that can create realistic images and art from a description in natural language.* Accessed 30 May 2023. URL: https://openai.com/dall-e-2/.

[96] OpenAI. *GPT-4 Technical Report*. Mar. 2023. arXiv: 2303.08774v3 [cs.CL].

[97] OpenAI. *Introducing ChatGPT*. Accessed 13 Mar. 2023. Nov. 2022. URL: https://openai.com/blog/chatgpt.

[98] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[99] N. Patki, R. Wedge, and K. Veeramachaneni. "The Synthetic Data Vault". In: *IEEE International Conference on Data Science and Advanced Analytics, DSAA*. Oct. 2016, pp. 399–410.

[100] M. Pawelczyk, K. Broelemann, and G. Kasneci. "Learning Model-Agnostic Counterfactual Explanations for Tabular Data". In: *Proceedings of The Web Conference*. WWW '20. ACM, Apr. 2020, pp. 3126–3132.

[101] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830.

[102] P. J. Phillips, C. Hahn, P. Fontana, A. Yates, K. K. Greene, D. Broniatowski, and M. A. Przybocki. *Four Principles of Explainable Artificial Intelligence*. Sept. 2021. DOI: https://doi.org/10.6028/NIST.IR.8312.

[103] L. Prechelt. "Early Stopping — But When?" In: *Neural Networks: Tricks of the Trade*. Springer, 2002, pp. 53–67.

[104] L. Prokhorenkova, A. V. Dorogush, A. Gulin, G. Gusev, and A. Vorobev. "CatBoost: Unbiased Boosting With Categorical Features". In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018.

[105] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[106] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. Apr. 2022. arXiv: 2204.06125 [cs.CV].

[107] A. Redelmeier, M. Jullum, K. Aas, and A. Løland. *MCCE: Monte Carlo Sampling of Realistic Counterfactual Explanations*. Nov. 2021. arXiv: 2111.09790 [stat.ML].

[108] J. P. Reiter. "Using CART to Generate Partially Synthetic Public Use Microdata". In: *Journal of Official Statistics* 21.3 (2005), p. 441.

[109] D. J. Rezende, S. Mohamed, and D. Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on Machine Learning, ICML*. Vol. 32. Proceedings of Machine Learning Research. PMLR, 2014, pp. 1278–1286.

[110] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 10684–10695.

[111] C. Rudin. "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead". In: *Nature Machine Intelligence* 1 (2019), pp. 206–215.

[112] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Representations by Back-Propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[113] L. Ruthotto and E. Haber. "An Introduction to Deep Generative Modeling". In: *GAMM-Mitteilungen* 44.2 (2021).

[114] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding". In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022.

[115] P. Sanchez, A. Kascenas, X. Liu, A. Q. O'Neil, and S. A. Tsaftaris. "What is Healthy? Generative Counterfactual Diffusion for Lesion Localization". In: *MICCAI Workshop on Deep Generative Models*. Springer. Sept. 2022, pp. 34–44.

[116] P. Sanchez and S. A. Tsaftaris. "Diffusion Causal Models for Counterfactual Estimation". In: *First Conference on Causal Learning and Reasoning*. Apr. 2022.

[117] R. Schapire. "The Strength of Weak Learnability". In: *Machine Learning* 5 (1990), pp. 197–227.

[118] R. Schapire and Y. Freund. *Boosting: Foundations and Algorithms*. 2012. ISBN: 9780262301183.

[119] Scikit-learn developers. *Metrics and Scoring: Quantifying the Quality of Predictions*. Accessed 14 Apr. 2023. URL: https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics.

[120] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision, ICVJ*. Vol. 128. 2020.

[121] A. Shih, S. Belkhale, S. Ermon, D. Sadigh, and N. Anari. *Parallel Sampling of Diffusion Models*. June 2023. arXiv: 2305.16317v2 [cs.LG].

[122] R. Shwartz-Ziv and A. Armon. "Tabular Data: Deep Learning is Not All You Need". In: *8th ICML Workshop on Automated Machine Learning (AutoML)*. 2021.

[123] K. S. Sivamani. *The great AI debate: Interpretability*. Accessed 1 May 2023. July 2019. URL: https://medium.com/swlh/the-great-ai-debate-interpretability-1d139167b55.

[124] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML*. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 2256–2265.

[125] K. Sohn, H. Lee, and X. Yan. "Learning Structured Output Representation Using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015.

[126] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. "Ladder Variational Autoencoders". In: *Advances in Neural Information Processing Systems*. Vol. 29. 2016.

[127] J. Song, C. Meng, and S. Ermon. "Denoising Diffusion Implicit Models". In: *International Conference on Learning Representations, ICLR*. 2021.

[128] Y. Song and S. Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[129] Y. Song and S. Ermon. "Improved Techniques for Training Score-Based Generative Models". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020.

[130] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. "Score-Based Generative Modeling Through Stochastic Differential Equations". In: *International Conference on Learning Representations, ICLR*. 2021.

[131] J. S. Speagle. *A Conceptual Introduction to Markov Chain Monte Carlo Methods*. Mar. 2020. arXiv: 1909.12313v2 [stat.OT].

[132] Spotify Engineering. *How Spotify Uses ML to Create the Future of Personalization*. Accessed 1 May 2023. Dec. 2021. URL: https://engineering.atspotify.com/2021/12/how-spotify-uses-ml-to-create-the-future-of-personalization.

[133] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks From Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[134] Tesla. *AI & Robotics*. Accessed 2 June 2023. URL: https://tesla.com/AI.

[135] H. Theil. *Statistical Decomposition Analysis*. North-Holland, 1972.

[136] ThresholdTom. *File:ConfusionMatrixRedBlue.png*. Accessed 21 Sept. 2023. May 2019. URL: https://commons.wikimedia.org/wiki/File:ConfusionMatrixRedBlue.png.

[137] M. E. Tipping and C. M. Bishop. "Probabilistic Principal Component Analysis". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 61.3 (1999), pp. 611–622.

[138] J. M. Tomczak. *Deep Generative Modeling*. Springer International Publishing AG, 2022. ISBN: 9783030931575.

[139] C. Van Rijsbergen. *Information Retrieval*. Second edition. Butterworths, 1979. ISBN: 9780408709293.

[140] G. Van Rossum. *Python Reference Manual*. Centrum voor Wiskunde en Informatica Amsterdam, Apr. 1995.

[141] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[142] J. Vendrow, S. Jain, L. Engstrom, and A. Madry. *Dataset Interfaces: Diagnosing Model Failures Using Controllable Counterfactual Generation*. Feb. 2023. arXiv: 2302.07865 [cs.LG].

[143] A. White. *By 2024, 60% of the Data Used for The Development of AI and Analytics Projects Will Be Synthetically Generated*. Accessed 1 May 2023. July 2021. URL: https://blogs.gartner.com/andrew_white/2021/07/24/by-2024-60-of-the-data-used-for-the-development-of-ai-and-analytics-projects-will-be-synthetically-generated/.

[144] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, et al. "Top 10 Algorithms in Data Mining". In: *Knowledge and Information Systems* 14 (2008), pp. 1–37.

[145] Z. Xiao, K. Kreis, and A. Vahdat. "Tackling the Generative Learning Trilemma With Denoising Diffusion GANs". In: *International Conference on Learning Representations, ICLR*. 2022.

[146] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. "Modeling Tabular data using Conditional GAN". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[147] S. Zheng and N. Charoenphakdee. "Diffusion Models for Missing Value Imputation in Tabular Data". In: *NeurIPS 2022 First Table Representation Workshop*. 2022.

Appendices

The appendices are organized as follows. Appendix A defines the *categorical distribution* and outlines how we can sample from it in practice. Appendix B shows how the KL divergence term in the ELBO in VAE can be integrated analytically under Gaussian assumptions. Appendix C provides complete derivations of several equations referenced when discussing diffusion models in Chapter 3. Appendix D contains further descriptions of the datasets that are used for the experiments we design in Chapter 5, including how we initially process them. Appendix E adds some thoughts on implementing Multinomial diffusion. Appendix F contains a proof of the closed form formula for calculating the KL divergence between two multivariate Gaussian distributions.

The code we have developed while working on this thesis can be found in the following repository on GitHub: `https://github.com/alexaoh/tabular-diffusion-for-counterfactuals`.

Appendix A

The Categorical Distribution

The categorical distribution plays an important role in our tabular diffusion model in Chapter 3 and appears in conjunction with the decoder distributions in VAEs in Sections 2.8 and 2.9. The term *categorical distribution* has no universally acknowledged definition across all disciplines, like, for example, the Gaussian distribution. For instance, in ML, the categorical and *multinomial* distributions are conflated, making it essential for us to define clearly what our terminology refers to mathematically. This appendix is devoted to this task.

The categorical distribution is a probability distribution that describes the outcome of a single draw from a discrete random variable X with Ψ possible categories, labelled or indexed, but not necessarily ordered, by $\{1, \dots, \Psi\}$. The probability of each of the categories is specified by a probability parameter, which we denote by $\boldsymbol{\pi} = \{\pi_1, \dots, \pi_\Psi\}$. Thus, $P(x = i) = \pi_i$, $i \in \{1, \dots, \Psi\}$, and $\|\boldsymbol{\pi}\|_1 = 1$. Based on this, the PMF of X is

$$\text{Categorical}(x; \boldsymbol{\pi}) \coloneqq \pi_1^{[x=1]} \cdots \pi_\Psi^{[x=\Psi]}, \tag{A.1}$$

where $[\cdot]$ is the *Iverson bracket*, which is defined as

$$[\xi] \coloneqq \begin{cases} 1, & \text{if } \xi \text{ is true,} \\ 0, & \text{otherwise,} \end{cases}$$

for a condition ξ. Notice that the support of the function in Equation (A.1) is a scalar value such that $x \in \{1, \dots, \Psi\}$, representing the index of the corresponding category of a realization x of X. When we denote the PMF with a vector argument, like $\text{Categorical}(\boldsymbol{x}; \boldsymbol{\pi}_j)$, we refer to the application of Equation (A.1) element-wise, where each realization x^j, $j \in \{1, \dots, p\}$, in $\boldsymbol{x} = \{x^1, \dots, x^p\}$ has its own probability parameter $\boldsymbol{\pi}_j$, each containing Ψ elements. For instance, when discussing the MNIST data mentioned in Section 2.8, where each image can be represented as $\boldsymbol{x} \in \{0, \dots, 255\}^{1 \times p}$, the notation in Equation (2.17) refers to a situation where the pixels follow a random vector $\boldsymbol{X} = \{X^1, \dots, X^p\}$, where each component X^i, $i \in \{1, \dots, p\}$, is distributed according to a categorical distribution with PMF

$$\text{Categorical}(x; \boldsymbol{\pi}_i) = \pi_{i,1}^{[x=1]} \cdots \pi_{i,256}^{[x=256]},$$

and separate probability parameters $\boldsymbol{\pi}_i = \{\pi_{i,1}, \dots, \pi_{i,256}\}$, $i \in \{1, \dots, p\}$. Thus, for correctness, $\boldsymbol{f_\theta}(\boldsymbol{z})$ in Equation (2.17) should actually be a matrix with softmax applied row-wise to the outputs of the neural network, where each row gives rise to one $\boldsymbol{\pi}_i$. This is a slight abuse of notation, but we skip this detail.

When analyzing data in practice, realizations of categorical variables are often *one-hot encoded* (OHE), meaning that the scalar x representing the index i of the corresponding

category that the realization has taken is represented by a vector $\boldsymbol{x} \in \mathbb{R}^{\Psi \times 1}$, where element i equals one and the rest of the elements are zero. In such a case, we write the the PMF of $\boldsymbol{X}$ as

$$\text{Categorical}_{\text{OHE}}(\boldsymbol{x}; \boldsymbol{\pi}) \coloneqq \pi_1^{x^1} \cdots \pi_\Psi^{x^\Psi}, \tag{A.2}$$

where $\boldsymbol{x} = \{x^1, \ldots, x^\Psi\}$ and we add the subscript *OHE* to specify that the support of this function is a OHE representation, $\boldsymbol{x}$, of x. This PMF was for instance adopted by Bishop [8], without using the term *categorical distribution*, and by Murphy [87], referring to it as the *categorical*, *discrete* and *multinoulli* distribution. To be clear, the PMFs (A.1) and (A.2) describe the same discrete random variable X, but have different supports, because they are based on slightly different representations.

A.1 Gumbel-Max Trick

We explain the *Gumbel-Max trick* [78], a method frequently used for sampling from a categorical distribution in practice. For instance, we use this trick to train, and sample from, Multinomial diffusion models, as noted in Sections 3.4 and 3.5.

Consistent with our previous notation in this appendix, we let X denote a categorical random variable with probability parameter $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_\Psi\}$. In addition, let

$$g_k = -\log(-\log u_k),\ u_k \sim \text{Uniform}[0, 1], \quad k \in \{1, \ldots, \Psi\},$$

be Ψ i.i.d. samples from a standard Gumbel distribution [54]. Then, a sample from the categorical distribution with PMF given in Equation (A.1) can be drawn as

$$x = \arg\max_k (\log \pi_k + g_k). \tag{A.3}$$

Moreover, a sample from the OHE variant of the PMF, as given in Equation (A.2), can be drawn by simply representing the resulting x from Equation (A.3) in OHE format [54].

As a side note, the arg max function is not differentiable, which is problematic when using this trick in certain situations. To avoid issues related to its non-differentiability, the *Gumbel-Softmax*, otherwise known as the *Concrete*, distribution was invented [54, 79]. Without going into detail, this distribution was derived by using the softmax function as an approximation of the arg max function in Equation (A.3). This distribution can, for instance, be used to implement a discrete version of the continuous reparameterization trick, which we have seen some vital applications of in VAEs and Gaussian diffusion models in this work. The interested reader is referred to Jang et al. [54] and Maddison et al. [79] for detailed expositions of this invention. Interestingly enough, these two sets of authors developed highly similar methods in parallel.

Appendix B

Derivation of KL Term in ELBO in VAE With Gaussian Assumptions

As stated in Section 2.8, the KL divergence term in $\hat{\mathcal{L}}^{II}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ can be integrated analytically under certain assumptions. While discussing VAEs, we simply stated the solution, assuming a specific Gaussian prior and encoder, as defined in Equation (2.18). The details were however omitted. Here, we provide a simple proof.

Recall that we let Λ be the latent space dimension and denote the elements of $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})$ and $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})$ by $\mu^j_{\boldsymbol{\phi},\boldsymbol{x}}$ and $(\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2$, for $j \in \{1, \ldots, \Lambda\}$, respectively. For clarity, the diagonal covariance structure of $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ implies that $\text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})] = \text{diag}((\sigma^1_{\boldsymbol{\phi},\boldsymbol{x}})^2, \ldots, (\sigma^\Lambda_{\boldsymbol{\phi},\boldsymbol{x}})^2)$ is a diagonal matrix with the elements of $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})$ on its diagonal. In general, the KL divergence between two multivariate Gaussian distributions with densities $p_1(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $p_2(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, in $\mathbb{R}^n$ is

$$D_{KL}(p_1(\boldsymbol{x}) \parallel p_2(\boldsymbol{x})) = \frac{1}{2}\left(\log\frac{\det\boldsymbol{\Sigma}_2}{\det\boldsymbol{\Sigma}_1} - n + \text{trace}\left(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1\right) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)\right),$$

which is proved in Appendix F. In our case, $n = \Lambda$, $p_1 = q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_2 = p(\boldsymbol{z})$, meaning that $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})$, $\boldsymbol{\Sigma}_1 = \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})\mathbf{I}$, $\boldsymbol{\mu}_2 = \mathbf{0}$ and $\boldsymbol{\Sigma}_2 = \mathbf{I}$, which yields

$$\begin{aligned}
&D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})) \\
&= \frac{1}{2}\left(\log\frac{\det\mathbf{I}}{\det\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})\mathbf{I}} - \Lambda + \text{trace}(\mathbf{I}^{-1}\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})\mathbf{I}) + (\mathbf{0} - \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}))^T\mathbf{I}^{-1}(\mathbf{0} - \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}))\right) \\
&= \frac{1}{2}\left(-\log\prod_{j=1}^{\Lambda}(\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2 - \Lambda + \sum_{j=1}^{\Lambda}(\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2 + \sum_{j=1}^{\Lambda}(\mu^j_{\boldsymbol{\phi},\boldsymbol{x}})^2\right) \\
&= \frac{1}{2}\left(-\sum_{j=1}^{\Lambda}(\log(\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2 + 1) + \sum_{j=1}^{\Lambda}((\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2 + (\mu^j_{\boldsymbol{\phi},\boldsymbol{x}})^2)\right) \\
&= -\frac{1}{2}\sum_{j=1}^{\Lambda}(1 + \log(\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2 - (\mu^j_{\boldsymbol{\phi},\boldsymbol{x}})^2 - (\sigma^j_{\boldsymbol{\phi},\boldsymbol{x}})^2).
\end{aligned}$$

Appendix C

Complete Derivations in Diffusion Models

This section is devoted to deriving the essential equations in diffusion models, as discussed in Chapter 3. Most of this material is gathered from papers [47, 49, 76, 124], modified and extended where we find it necessary. Section C.1 contains derivations for Gaussian diffusion models, while Section C.2 contains derivations for Multinomial diffusion models.

C.1 Gaussian Diffusion

Closed Form Forward Density Formula

The closed form formulation of any density in the forward process, as stated in Equation (3.10), is derived here. Specifically, the derivation is inspired by Luo [76]. Recall the Gaussian assumptions in the forward process,

$$\begin{aligned} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) &= \mathcal{N}(\boldsymbol{z}_t; \sqrt{1-\beta_t}\boldsymbol{z}_{t-1}, \beta_t\mathbf{I}), \quad t \in \{2,\dots,T\}, \\ q(\boldsymbol{z}_1|\boldsymbol{x}) &= \mathcal{N}(\boldsymbol{z}_1; \sqrt{1-\beta_1}\boldsymbol{x}, \beta_1\mathbf{I}). \end{aligned} \tag{C.1}$$

According to the reparameterization trick, sampling from the Gaussian distributions represented by the densities in Equation (C.1) may be achieved by

$$\boldsymbol{z}_t = \sqrt{1-\beta_t}\boldsymbol{z}_{t-1} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_{t-1}, \quad t \in \{1,\dots,T\}$$

where $\boldsymbol{z}_0 \coloneqq \boldsymbol{x}$ and $\boldsymbol{\varepsilon}_t$, $t \in \{0,\dots,T-1\}$, is i.i.d. standard Gaussian noise of the same dimension as $\boldsymbol{z}_t$, $t \in \{1,\dots,T\}$. Equation (3.10) is derived by recursive use of this trick. Before doing so, we let $\boldsymbol{\varepsilon}^*_t$, $t \in \{0,\dots,T-1\}$, denote a second set of i.i.d. standard Gaussian noise observations. Then, we proceed to a direct calculation,

$$\begin{aligned} \boldsymbol{z}_t &= \sqrt{1-\beta_t}\boldsymbol{z}_{t-1} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_{t-1} \\ &= \sqrt{1-\beta_t}(\sqrt{1-\beta_{t-1}}\boldsymbol{z}_{t-2} + \sqrt{\beta_{t-1}}\boldsymbol{\varepsilon}_{t-2}) + \sqrt{\beta_t}\boldsymbol{\varepsilon}_{t-1} \\ &= \sqrt{(1-\beta_t)(1-\beta_{t-1})}\boldsymbol{z}_{t-2} + \sqrt{\beta_{t-1}-\beta_t\beta_{t-1}}\boldsymbol{\varepsilon}_{t-2} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_{t-1} \qquad \text{(C.2)} \\ &= \sqrt{(1-\beta_t)(1-\beta_{t-1})}\boldsymbol{z}_{t-2} + \sqrt{\beta_{t-1}-\beta_t\beta_{t-1}+\beta_t}\boldsymbol{\varepsilon}^*_{t-2} \qquad \text{(C.3)} \\ &= \sqrt{(1-\beta_t)(1-\beta_{t-1})}\boldsymbol{z}_{t-2} + \sqrt{1-(1-\beta_t)(1-\beta_{t-1})}\boldsymbol{\varepsilon}^*_{t-2} \\ &= \sqrt{(1-\beta_t)(1-\beta_{t-1})}(\sqrt{1-\beta_{t-2}}\boldsymbol{z}_{t-3} + \sqrt{\beta_{t-2}}\boldsymbol{\varepsilon}_{t-3}) + \sqrt{1-(1-\beta_t)(1-\beta_{t-1})}\boldsymbol{\varepsilon}^*_{t-2} \\ &= \sqrt{(1-\beta_t)(1-\beta_{t-1})(1-\beta_{t-2})}\boldsymbol{z}_{t-3} + \sqrt{1-(1-\beta_t)(1-\beta_{t-1})(1-\beta_{t-2})}\boldsymbol{\varepsilon}^*_{t-3} \end{aligned}$$

$$
\begin{aligned}
&= \sqrt{\prod_{i=(t-2)}^{t}(1-\beta_i)}\boldsymbol{z}_{t-3} + \sqrt{1-\prod_{i=(t-2)}^{t}(1-\beta_i)}\boldsymbol{\varepsilon}^*_{t-3} \\
&\vdots \\
&= \sqrt{\prod_{i=2}^{t}(1-\beta_i)}\boldsymbol{z}_1 + \sqrt{1-\prod_{i=2}^{t}(1-\beta_i)}\boldsymbol{\varepsilon}^*_1 \\
&= \sqrt{\prod_{i=1}^{t}(1-\beta_i)}\boldsymbol{x} + \sqrt{1-\prod_{i=1}^{t}(1-\beta_i)}\boldsymbol{\varepsilon}^*_0 \\
&= \sqrt{\prod_{i=1}^{t}\alpha_i}\boldsymbol{x} + \sqrt{1-\prod_{i=1}^{t}\alpha_i}\boldsymbol{\varepsilon}^*_0 \\
&= \sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}^*_0, \qquad \text{(C.4)}
\end{aligned}
$$

where the equality of Equations (C.2) and (C.3) is achieved by using that the sum of two independent Gaussian random variables is Gaussian, with mean as the sum of means and variance as the sum of variances. More precisely, $\sqrt{\beta_{t-1}-\beta_t\beta_{t-1}}\boldsymbol{\varepsilon}_{t-2}$ may be interpreted as a sample from $\mathcal{N}(\mathbf{0},(\beta_{t-1}-\beta_t\beta_{t-1})\mathbf{I})$ and $\sqrt{\beta_t}\boldsymbol{\varepsilon}_{t-1}$ may be interpreted as a sample from $\mathcal{N}(\mathbf{0},\beta_t\mathbf{I})$, meaning that their sum may be interpreted as a sample from $\mathcal{N}(\mathbf{0},(\beta_{t-1}-\beta_t\beta_{t-1}+\beta_t)\mathbf{I})$. Notice that Equation (C.4) is the reparameterization trick applied to a Gaussian distribution described by the density $\mathcal{N}(\boldsymbol{z}_t;\sqrt{\bar{\alpha}_t}\boldsymbol{x},(1-\bar{\alpha}_t)\mathbf{I})$. Thus, we have derived Equation (3.10).

Forward Posterior Conditioned on Data

The forward posterior densities conditioned on an arbitrary observation $\boldsymbol{x}\in\mathcal{D}$, $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})$, $t\in\{2,\dots,T\}$, as given in Equation (3.13), are derived in this section. The derivation is inspired by Luo [76]. The starting point is Bayes' rule, which reads

$$
q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x}) = \frac{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x})q(\boldsymbol{z}_{t-1}|\boldsymbol{x})}{q(\boldsymbol{z}_t|\boldsymbol{x})}, \quad t\in\{2,\dots,T\}, \qquad \text{(C.5)}
$$

in this case. We know that $q(\boldsymbol{z}_t|\boldsymbol{x})=\mathcal{N}(\boldsymbol{z}_t;\sqrt{\bar{\alpha}_t}\boldsymbol{x},(1-\bar{\alpha}_t)\mathbf{I})$, $t\in\{1,\dots,T\}$, as proved in the previous section, which implies that $q(\boldsymbol{z}_{t-1}|\boldsymbol{x})=\mathcal{N}(\boldsymbol{z}_{t-1};\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x},(1-\bar{\alpha}_{t-1})\mathbf{I})$, $t\in\{2,\dots,T\}$. In addition, we know that $q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x})=\mathcal{N}(\boldsymbol{z}_t;\sqrt{1-\beta_t}\boldsymbol{z}_{t-1},\beta_t\mathbf{I})$ because $q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x})=q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ via the Markov property. Thus, insertion yields

$$
\begin{aligned}
q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x}) &= \frac{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x})q(\boldsymbol{z}_{t-1}|\boldsymbol{x})}{q(\boldsymbol{z}_t|\boldsymbol{x})} \\
&= \frac{(2\pi(1-\bar{\alpha}_t))^{(p+1)/2}}{(2\pi\beta_t\cdot 2\pi(1-\bar{\alpha}_{t-1}))^{(p+1)/2}}\frac{\exp\{-\frac{1}{2\beta_t}\|\boldsymbol{z}_t-\sqrt{\alpha_t}\boldsymbol{z}_{t-1}\|^2\}\exp\{-\frac{1}{2(1-\bar{\alpha}_{t-1})}\|\boldsymbol{z}_{t-1}-\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}\|^2\}}{\exp\{-\frac{1}{2(1-\bar{\alpha}_t)}\|\boldsymbol{z}_t-\sqrt{\bar{\alpha}_t}\boldsymbol{x}\|^2\}} \\
&\propto \exp\left(-\frac{1}{2}\left(\frac{1}{\beta_t}\|\boldsymbol{z}_t-\sqrt{\alpha_t}\boldsymbol{z}_{t-1}\|^2+\frac{1}{1-\bar{\alpha}_{t-1}}\|\boldsymbol{z}_{t-1}-\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}\|^2-\frac{1}{1-\bar{\alpha}_t}\|\boldsymbol{z}_t-\sqrt{\bar{\alpha}_t}\boldsymbol{x}\|^2\right)\right) \\
&= \exp\left(-\frac{1}{2}\left(\frac{1}{\beta_t}(-2\sqrt{\alpha_t}\boldsymbol{z}_t^T\boldsymbol{z}_{t-1}+\alpha_t\|\boldsymbol{z}_{t-1}\|^2)+\frac{1}{1-\bar{\alpha}_{t-1}}(\|\boldsymbol{z}_{t-1}\|^2-2\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}^T\boldsymbol{z}_{t-1})+C(\boldsymbol{z}_t,\boldsymbol{x})\right)\right)
\end{aligned}
$$

$$\propto \exp\left(-\frac{1}{2}\left(\|\boldsymbol{z}_{t-1}\|^2\left(\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}}\right)-2\boldsymbol{z}_{t-1}^T\left(\frac{\sqrt{\alpha_t}\boldsymbol{z}_t}{\beta_t}+\frac{\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}}{1-\bar{\alpha}_{t-1}}\right)\right)\right)$$
$$= \exp\left(-\frac{1}{2}\left(\|\boldsymbol{z}_{t-1}\|^2\left(\frac{\alpha_t(1-\bar{\alpha}_{t-1})+\beta_t}{\beta_t(1-\bar{\alpha}_{t-1})}\right)-2\boldsymbol{z}_{t-1}^T\left(\frac{\sqrt{\alpha_t}\boldsymbol{z}_t(1-\bar{\alpha}_{t-1})+\beta_t\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}}{\beta_t(1-\bar{\alpha}_{t-1})}\right)\right)\right)$$

$$= \exp\left(-\frac{1}{2}\left(\|\boldsymbol{z}_{t-1}\|^2\left(\frac{1-\bar{\alpha}_t}{\beta_t(1-\bar{\alpha}_{t-1})}\right)-2\boldsymbol{z}_{t-1}^T\left(\frac{\sqrt{\alpha_t}\boldsymbol{z}_t(1-\bar{\alpha}_{t-1})+\beta_t\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}}{\beta_t(1-\bar{\alpha}_{t-1})}\right)\right)\right)$$
$$= \exp\left(-\frac{1}{2}\left(\frac{1-\bar{\alpha}_t}{\beta_t(1-\bar{\alpha}_{t-1})}\right)\left(\|\boldsymbol{z}_{t-1}\|^2-2\boldsymbol{z}_{t-1}^T\left(\frac{\sqrt{\alpha_t}\boldsymbol{z}_t(1-\bar{\alpha}_{t-1})+\beta_t\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}}{1-\bar{\alpha}_t}\right)\right)\right)$$
$$= \exp\left(-\frac{1}{2}\left(\frac{1}{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}}\right)\left(\|\boldsymbol{z}_{t-1}\|^2-2\boldsymbol{z}_{t-1}^T\left(\frac{\sqrt{\alpha_t}\boldsymbol{z}_t(1-\bar{\alpha}_{t-1})+\beta_t\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}}{1-\bar{\alpha}_t}\right)\right)\right)$$
$$\propto \exp\left(-\frac{1}{2}\left(\frac{1}{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}}\right)\left\|\boldsymbol{z}_{t-1}-\frac{\sqrt{\alpha_t}\boldsymbol{z}_t(1-\bar{\alpha}_{t-1})+\beta_t\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}}{1-\bar{\alpha}_t}\right\|^2\right) \qquad \text{(Complete square)}$$

$$= \mathcal{N}\left(\boldsymbol{z}_{t-1};\left(\frac{\sqrt{\alpha_t}\boldsymbol{z}_t(1-\bar{\alpha}_{t-1})+\beta_t\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}}{1-\bar{\alpha}_t}\right),\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{I}\right)$$
$$=: \mathcal{N}(\boldsymbol{z}_{t-1};\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t,\boldsymbol{x}),\tilde{\Sigma}_t\mathbf{I})$$

where $C(\boldsymbol{z}_t,\boldsymbol{x})$ is a term not involving $\boldsymbol{z}_{t-1}$ and where $\|\cdot\|$ is the Euclidean norm. The completion of the square implicitly reintroduces $C(\boldsymbol{z}_t,\boldsymbol{x})$, such that equality in fact holds between the left and right hand sides of the equation. This can be verified with standard arithmetic. Hence

$$\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t,\boldsymbol{x}) \coloneqq \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}+\frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{z}_t,$$

and

$$\tilde{\Sigma}_t \coloneqq \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t.$$

Thus, we have derived the quantities in Equations (3.13).

Evidence Lower Bound

The ELBO, as given in Equation (3.14), is derived here. The derivation is inspired by the corresponding derivation by Luo [76], but we have added several more detailed calculations, as well as modified it to fit our notation. The initial expression of the ELBO of diffusion models is almost identical to the ELBO of HVAEs, which is shown in Equation (3.7). The expression reads

$$\mathcal{L}_{\boldsymbol{\theta}}^{GD}(\boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T)}{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\right],$$

where the variational parameters, $\boldsymbol{\phi}$, have been suppressed, because the forward process densities have no learnable parameters. Using Equations (3.8) and (3.11), it can be rewritten as

$$\mathcal{L}_{\boldsymbol{\theta}}^{GD}(\boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log\frac{p(\boldsymbol{z}_T)\prod_{t=2}^T p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)}{q(\boldsymbol{z}_1|\boldsymbol{x})\prod_{t=2}^T q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right]. \tag{C.6}$$

This equation is expanded further through the following calculations

$$
\begin{aligned}
\mathcal{L}_{\boldsymbol{\theta}}^{GD}(\boldsymbol{x}) &= \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)}{q(\boldsymbol{z}_1|\boldsymbol{x})\prod_{t=2}^{T} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})\prod_{t=2}^{T-1} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})q(\boldsymbol{z}_1|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)\prod_{t=3}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2)}{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})\prod_{t=3}^{T} q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_{t-2})q(\boldsymbol{z}_1|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)] + \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)}{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})}\right] \\
&\quad + \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2)}{q(\boldsymbol{z}_1|\boldsymbol{x})}\right] + \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \prod_{t=2}^{T-1} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})}{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)] + \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)}{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})}\right] \\
&\quad + \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2)}{q(\boldsymbol{z}_1|\boldsymbol{x})}\right] + \sum_{t=2}^{T-1} \mathbb{E}_{q(\boldsymbol{z}_1,\ldots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})}{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{x})}[\log p_{\theta}(\boldsymbol{x}|\boldsymbol{z}_1)] + \mathbb{E}_{q(\boldsymbol{z}_{T-1},\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)}{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})}\right] \\
&\quad + \mathbb{E}_{q(\boldsymbol{z}_1,\boldsymbol{z}_2|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2)}{q(\boldsymbol{z}_1|\boldsymbol{x})}\right] + \sum_{t=2}^{T-1} \mathbb{E}_{q(\boldsymbol{z}_{t-1},\boldsymbol{z}_t,\boldsymbol{z}_{t+1}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})}{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)] \underbrace{- \mathbb{E}_{q(\boldsymbol{z}_{T-1}|\boldsymbol{x})}[D_{KL}(q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1}) \parallel p(\boldsymbol{z}_T))]}_{(A)} \\
&\quad \underbrace{- \mathbb{E}_{q(\boldsymbol{z}_2|\boldsymbol{x})}[D_{KL}(q(\boldsymbol{z}_1|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2))]}_{(B)} \\
&\quad \underbrace{- \sum_{t=2}^{T-1} \mathbb{E}_{q(\boldsymbol{z}_{t-1},\boldsymbol{z}_{t+1}|\boldsymbol{x})}[D_{KL}(q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})]}_{(C)}, \qquad \text{(C.7)}
\end{aligned}
$$

where we show that (A), (B) and (C) hold in the following. In order to show that the second term can be written as (A) we use the property

$$
q(\boldsymbol{z}_T, \boldsymbol{z}_{T-1}|\boldsymbol{x}) = q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1}, \boldsymbol{x})q(\boldsymbol{z}_{T-1}|\boldsymbol{x}) = q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})q(\boldsymbol{z}_{T-1}|\boldsymbol{x}),
$$

where the first equality is a standard conditional probability calculation and the second equality holds because of the Markov assumptions. Direct calculation then gives

$$
\begin{aligned}
(A) &= \mathbb{E}_{q(\boldsymbol{z}_{T-1},\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)}{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})q(\boldsymbol{z}_{T-1}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)}{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})}\right] \\
&= \iint -\log \frac{q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})}{p(\boldsymbol{z}_T)} q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})q(\boldsymbol{z}_{T-1}|\boldsymbol{x})\mathrm{d}\boldsymbol{z}_T\mathrm{d}\boldsymbol{z}_{T-1} \\
&= -\int D_{KL}(q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1}) \parallel p(\boldsymbol{z}_T))q(\boldsymbol{z}_{T-1}|\boldsymbol{x})\mathrm{d}\boldsymbol{z}_{T-1} \\
&= -\mathbb{E}_{q(\boldsymbol{z}_{T-1}|\boldsymbol{x})}[D_{KL}(q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1}) \parallel p(\boldsymbol{z}_T))].
\end{aligned}
$$

Next, in order to show that the third term can be written as (B) we use the property

$$q(\boldsymbol{z}_2, \boldsymbol{z}_1|\boldsymbol{x}) = q(\boldsymbol{z}_1|\boldsymbol{z}_2, \boldsymbol{x})q(\boldsymbol{z}_2|\boldsymbol{x}) = q(\boldsymbol{z}_1|\boldsymbol{x})q(\boldsymbol{z}_2|\boldsymbol{x}),$$

which holds for similar reasons as above. Direct calculation then gives

$$\begin{aligned}
(B) &= \mathbb{E}_{q(\boldsymbol{z}_1,\boldsymbol{z}_2|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2)}{q(\boldsymbol{z}_1|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{x})q(\boldsymbol{z}_2|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2)}{q(\boldsymbol{z}_1|\boldsymbol{x})}\right] \\
&= \iint -\log \frac{q(\boldsymbol{z}_1|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2)} q(\boldsymbol{z}_1|\boldsymbol{x})q(\boldsymbol{z}_2|\boldsymbol{x})\mathrm{d}\boldsymbol{z}_1\mathrm{d}\boldsymbol{z}_2 \\
&= -\int D_{KL}(q(\boldsymbol{z}_1|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2))q(\boldsymbol{z}_2|\boldsymbol{x})\mathrm{d}\boldsymbol{z}_2 \\
&= -\mathbb{E}_{q(\boldsymbol{z}_2|\boldsymbol{x})}[D_{KL}(q(\boldsymbol{z}_1|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_1|\boldsymbol{z}_2))].
\end{aligned}$$

Finally, in order to show that the fourth term can be written as (C) we use the property

$$q(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{x}) = q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{z}_{t+1}, \boldsymbol{x})q(\boldsymbol{z}_{t-1}, \boldsymbol{z}_{t+1}|\boldsymbol{x}) = q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})q(\boldsymbol{z}_{t-1}, \boldsymbol{z}_{t+1}|\boldsymbol{x}),$$

which holds $\forall t \in \{2, \dots, T-1\}$. Then, direct calculation gives

$$\begin{aligned}
(C) &= \sum_{t=2}^{T-1} \mathbb{E}_{q(\boldsymbol{z}_{t-1},\boldsymbol{z}_t,\boldsymbol{z}_{t+1}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})}{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right] \\
&= \sum_{t=2}^{T-1} \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})q(\boldsymbol{z}_{t-1},\boldsymbol{z}_{t+1}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})}{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right] \\
&= \sum_{t=2}^{T-1} \iiint -\log \frac{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})q(\boldsymbol{z}_{t-1}, \boldsymbol{z}_{t+1}|\boldsymbol{x})\mathrm{d}\boldsymbol{z}_t\mathrm{d}\boldsymbol{z}_{t-1}\mathrm{d}\boldsymbol{z}_{t+1} \\
&= -\sum_{t=2}^{T-1} \iint D_{KL}(q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}))q(\boldsymbol{z}_{t-1}, \boldsymbol{z}_{t+1}|\boldsymbol{x})\mathrm{d}\boldsymbol{z}_{t-1}\mathrm{d}\boldsymbol{z}_{t+1} \\
&= -\sum_{t=2}^{T-1} \mathbb{E}_{q(\boldsymbol{z}_{t-1},\boldsymbol{z}_{t+1}|\boldsymbol{x})}[D_{KL}(q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})].
\end{aligned}$$

Thus, we have expanded the ELBO into separate expectations, which can be estimated. However, when estimating Equation (C.7), Monte Carlo (MC) estimators suffer from large variance, especially for large values of T, since the expectations are calculated based on several random variables at once [76]. This is one example of the phenomenon often referred to as the *curse of dimensionality* [7, 44]. This is a collective term for phenomena that often arise when dealing with data in high-dimensional spaces. Since MC estimators use direct sampling, the number of samples necessary to achieve a reasonable accuracy increases rapidly with an increase in dimension. Thus, they are usually highly attractive estimators in low-dimensional probability spaces, preferably of only one dimension, but suffer from large variance or reduced efficiency in larger dimensions. Thus, the question is: are we able to find an expression of the ELBO that enables approximation of only one-dimensional integrals?

In fact, in order to avoid this problem, Sohl-Dickstein et al. [124] and Ho et al. [47] use a different expression as their ELBO. We derive this expression using Bayes' rule, as given in Equation (C.5). Precisely, Bayes' equation is rewritten as

$$q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x}) = \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})q(\boldsymbol{z}_t|\boldsymbol{x})}{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})}, \quad t \in \{2,\dots,T\}. \tag{C.8}$$

In addition, we combine this with

$$q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x}), \quad t \in \{2,\dots,T\}, \tag{C.9}$$

which holds because of the Markov property. Keeping Equations (C.8) and (C.9) in mind, we expand Equation (C.6) through the following calculations

$$\begin{aligned}
&\mathcal{L}_{\boldsymbol{\theta}}^{GD}(\boldsymbol{x}) \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)}{q(\boldsymbol{z}_1|\boldsymbol{x})\prod_{t=2}^{T} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right]. \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q(\boldsymbol{z}_1|\boldsymbol{x})\prod_{t=2}^{T} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x})}\right] && \text{(Equation (C.9))} \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)}{q(\boldsymbol{z}_1|\boldsymbol{x})}\right] + \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\sum_{t=2}^{T}\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)}{q(\boldsymbol{z}_1|\boldsymbol{x})}\right] + \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\sum_{t=2}^{T}\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{\frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})q(\boldsymbol{z}_t|\boldsymbol{x})}{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})}}\right] && \text{(Equation (C.8))} \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)}{\cancel{q(\boldsymbol{z}_1|\boldsymbol{x})}}\right] + \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{\cancel{q(\boldsymbol{z}_1|\boldsymbol{x})}}{q(\boldsymbol{z}_T|\boldsymbol{x})}\right] \\
&\qquad + \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\sum_{t=2}^{T}\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})}\right] && \text{(Telescoping sum)} \\
&= \mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)}{q(\boldsymbol{z}_T|\boldsymbol{x})}\right] + \sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{z}_1,\dots,\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)\right] + \mathbb{E}_{q(\boldsymbol{z}_T|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z}_T)}{q(\boldsymbol{z}_T|\boldsymbol{x})}\right] \\
&\qquad + \sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{z}_t,\boldsymbol{z}_{t-1}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)\right] - D_{KL}(q(\boldsymbol{z}_T|\boldsymbol{x}) \parallel p(\boldsymbol{z}_T)) \\
&\qquad - \sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})}\left[D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t))\right],
\end{aligned}$$

where the second term follows from the definition of the KL divergence and the final term follows because

$$\begin{aligned}
&\sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{z}_t,\boldsymbol{z}_{t-1}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})}\right] \\
&= \sum_{t=2}^{T}\mathbb{E}_{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})q(\boldsymbol{z}_t|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})}\right]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{t=2}^{T} \iint -\log \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)} q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) q(\boldsymbol{z}_t|\boldsymbol{x}) \mathrm{d}\boldsymbol{z}_{t-1} \mathrm{d}\boldsymbol{z}_t \\
&= -\sum_{t=2}^{T} \int D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)) q(\boldsymbol{z}_t|\boldsymbol{x}) \mathrm{d}\boldsymbol{z}_t \\
&= -\sum_{t=2}^{T} \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} \left[D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)) \right],
\end{aligned}
$$

similarly to the previous calculations of $(A), (B)$ and (C). Moreover, the telescoping sum is

$$
\begin{aligned}
\prod_{t=2}^{T} q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{x}) &= \prod_{t=2}^{T} \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) q(\boldsymbol{z}_t|\boldsymbol{x})}{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})} \\
&= \frac{q(\boldsymbol{z}_1|\boldsymbol{z}_2, \boldsymbol{x}) \cancel{q(\boldsymbol{z}_2|\boldsymbol{x})}}{q(\boldsymbol{z}_1|\boldsymbol{x})} \frac{q(\boldsymbol{z}_2|\boldsymbol{z}_3, \boldsymbol{x}) q(\boldsymbol{z}_3|\boldsymbol{x})}{\cancel{q(\boldsymbol{z}_2|\boldsymbol{x})}} \prod_{t=4}^{T} \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) q(\boldsymbol{z}_t|\boldsymbol{x})}{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})} \\
&= q(\boldsymbol{z}_1|\boldsymbol{z}_2, \boldsymbol{x}) q(\boldsymbol{z}_2|\boldsymbol{z}_3, \boldsymbol{x}) \cdots q(\boldsymbol{z}_{T-1}|\boldsymbol{z}_T, \boldsymbol{x}) \frac{q(\boldsymbol{z}_T|\boldsymbol{x})}{q(\boldsymbol{z}_1|\boldsymbol{x})} \\
&= \frac{q(\boldsymbol{z}_T|\boldsymbol{x})}{q(\boldsymbol{z}_1|\boldsymbol{x})} \prod_{t=2}^{T} q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}),
\end{aligned}
$$

which means that

$$
\begin{aligned}
\sum_{t=2}^{T} \log q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{x}) &= \sum_{t=2}^{T} \log \frac{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) q(\boldsymbol{z}_t|\boldsymbol{x})}{q(\boldsymbol{z}_{t-1}|\boldsymbol{x})} \\
&= \log \frac{q(\boldsymbol{z}_T|\boldsymbol{x})}{q(\boldsymbol{z}_1|\boldsymbol{x})} + \sum_{t=2}^{T} \log q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}),
\end{aligned}
$$

which was used in the derivation above. Thus, we have derived an expression of the ELBO that facilitates approximation of only one-dimensional integrals, which is the expression we displayed in Equation (3.14).

Direct Calculation of L_{t-1}

As stated in the main text, L_{t-1} can be calculated directly using the formula for the KL divergence between two Gaussians that is proved in Appendix F. The direct calculation is

$$
\begin{aligned}
L_{t-1} &= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} \left[D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)) \right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} [D_{KL}(\mathcal{N}(\boldsymbol{z}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x}), \tilde{\Sigma}_t \mathbf{I}) \parallel \mathcal{N}(\boldsymbol{z}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t), \tilde{\Sigma}_t \mathbf{I}))] \\
&= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} \left[\frac{1}{2} \left(\log \frac{\det \tilde{\Sigma}_t \mathbf{I}}{\det \tilde{\Sigma}_t \mathbf{I}} - n + \operatorname{trace} \left((\tilde{\Sigma}_t \mathbf{I})^{-1} \tilde{\Sigma}_t \mathbf{I} \right) + (\boldsymbol{\mu}_{\boldsymbol{\theta}} - \tilde{\boldsymbol{\mu}}_t)^T (\tilde{\Sigma}_t \mathbf{I})^{-1} (\boldsymbol{\mu}_{\boldsymbol{\theta}} - \tilde{\boldsymbol{\mu}}_t) \right) \right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} \left[\frac{1}{2} \left((\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x}))^T (\tilde{\Sigma}_t \mathbf{I})^{-1} (\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})) \right) \right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} \left[\frac{1}{2\tilde{\Sigma}_t} \|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})\|_2^2 \right],
\end{aligned} \tag{C.10}
$$

where we suppress the arguments of the mean parameter predictions in the third line to avoid line overflow. This explains why we can train the ML model $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ according to the loss in Equation (3.15). After setting

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) \coloneqq \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) + \frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{z}_t,$$

this expression can be further simplified to

$$\begin{aligned}
L_{t-1} &= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})}\left[D_{KL}(q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t))\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})}\left[\frac{1}{2\tilde{\Sigma}_t}\|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})\|_2^2\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})}\left[\frac{1}{2\tilde{\Sigma}_t}\left\|\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) + \frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{z}_t - \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x} - \frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{z}_t\right\|_2^2\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})}\left[\frac{1}{2\tilde{\Sigma}_t}\left\|\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}\right\|_2^2\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})}\left[\frac{1}{2\tilde{\Sigma}_t}\frac{\bar{\alpha}_{t-1}\beta_t^2}{(1-\bar{\alpha}_t)^2}\|\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \boldsymbol{x}\|_2^2\right].
\end{aligned}$$

This explains why we can train a model, $\boldsymbol{f}_{\boldsymbol{\theta}}$, to predict the input instance $\boldsymbol{x} \in \mathcal{D}$, using the loss in Equation (3.17), which is the second strategy from Ho et al. [47] for learning the reverse process parameters in a Gaussian diffusion model. Finally. the third strategy can be derived by realizing that the rearrangement

$$\boldsymbol{x} = \frac{\boldsymbol{z}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}}{\sqrt{\bar{\alpha}_t}},$$

of $q(\boldsymbol{z}_t|\boldsymbol{x})$, where $\boldsymbol{\varepsilon}$ is a realization of $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, can be inserted into $\tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})$. This insertion yields

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

This means that we can match the two means by defining

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) \coloneqq \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) \right),$$

where $\boldsymbol{g}_{\boldsymbol{\theta}}$ is a predictor of $\boldsymbol{\varepsilon}$. Insertion into Equation (C.10) yields

$$\begin{aligned}
L_{t-1} &= \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x})} \left[\frac{1}{2\tilde{\Sigma}_t} \|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \tilde{\boldsymbol{\mu}}_t(\boldsymbol{z}_t, \boldsymbol{x})\|_2^2 \right] \\
&= \mathbb{E}_{p(\boldsymbol{\varepsilon})} \left[\frac{1}{2\tilde{\Sigma}_t} \left\| \frac{1}{\sqrt{\alpha_t}} \boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) - \frac{1}{\sqrt{\alpha_t}} \boldsymbol{z}_t + \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \boldsymbol{\varepsilon} \right\|_2^2 \right] \\
&= \mathbb{E}_{p(\boldsymbol{\varepsilon})} \left[\frac{1}{2\tilde{\Sigma}_t} \left\| - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t) + \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \boldsymbol{\varepsilon} \right\|_2^2 \right] \\
&= \mathbb{E}_{p(\boldsymbol{\varepsilon})} \left[\frac{1}{2\tilde{\Sigma}_t} \frac{\beta_t^2}{\alpha_t(1-\bar{\alpha}_t)} \|\boldsymbol{\varepsilon} - \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)\|_2^2 \right],
\end{aligned}$$

where the expectation is taken with respect to $\boldsymbol{\mathcal{E}} \sim p(\boldsymbol{\varepsilon}) = \mathcal{N}(\boldsymbol{\varepsilon}; \mathbf{0}, \mathbf{I})$ because of the reparameterization trick. Recall that $\boldsymbol{z}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}$. This explains why we can train a model, $\boldsymbol{g}_{\boldsymbol{\theta}}$, to predict the added noise during diffusion of the input instance $\boldsymbol{x} \in \mathcal{D}$, $\boldsymbol{\varepsilon}$, using the loss function in Equation (3.20). In conclusion, we have shown how L_{t-1}, $t \in \{2, \ldots, T\}$, can be calculated in three different ways, each case indicating what quantity we train our ML model to predict.

Inclusion of L_0 in L_{simple}^{GD}

In the main text we state that explicitly defining $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ as

$$\begin{aligned}
p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) &= \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)), \\
\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) &= \frac{1}{\sqrt{1-\beta_1}} \left(\boldsymbol{z}_1 - \sqrt{\beta_1} \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) \right), \\
\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) &= a\mathbf{I},
\end{aligned}$$

explains why L_0 is in fact included in L_{simple}^{GD} when $t = 1$. In this section, we show that minimizing L_{simple}^{GD} when $t = 1$ is equivalent to explicitly defining $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ according to the equations above, before reweighting $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ and including this term in the loss function alongside L_{simple}^{GD} for $t \in \{2, \ldots, T\}$. The calculations are simple, but we add them here for completeness.

Following the definition above,

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) &= \log \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)) \\
&= \log \left\{ (2\pi)^{-\frac{p+1}{2}} \det(a\mathbf{I})^{-\frac{1}{2}} \exp\left[-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{\theta}})^T (a\mathbf{I})^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{\theta}}) \right] \right\} \\
&= -\frac{1}{2a} \|\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{\theta}}\|_2^2 + C,
\end{aligned}$$

where we have added all constants irrelevant to the optimization problem in $C \in \mathbb{R}$. Recall that $q(\boldsymbol{z}_1|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}_1; \sqrt{1-\beta_1}\boldsymbol{x}, \beta_1\mathbf{I})$, meaning that the reparameterization trick can be applied to yield $\boldsymbol{z}_1 = \sqrt{1-\beta_1}\boldsymbol{x} + \sqrt{\beta_1}\boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is a realization of $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Rewriting with respect to $\boldsymbol{x}$ and inserting in $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)$ gives

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) = -\frac{1}{2a}\left\|\frac{1}{\sqrt{1-\beta_1}}\left(\boldsymbol{z}_1 - \sqrt{\beta_1}\boldsymbol{\varepsilon}\right) - \boldsymbol{\mu}_{\boldsymbol{\theta}}\right\|_2^2 + C.$$

From this relation, we can conclude that defining $\mu_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) \coloneqq \frac{1}{\sqrt{1-\beta_1}}\left(\boldsymbol{z}_1 - \sqrt{\beta_1}\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)\right)$, where $\boldsymbol{g}_{\boldsymbol{\theta}}$ is a neural network for predicting $\boldsymbol{\varepsilon}$, gives the relation we are looking for. Hence,

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) &= -\frac{1}{2a}\left\|\frac{1}{\sqrt{1-\beta_1}}\left(\boldsymbol{z}_1 - \sqrt{\beta_1}\boldsymbol{\varepsilon}\right) - \boldsymbol{\mu}_{\boldsymbol{\theta}}\right\|_2^2 + C \\
&= -\frac{1}{2a}\left\|\frac{1}{\sqrt{1-\beta_1}}\left(\boldsymbol{z}_1 - \sqrt{\beta_1}\boldsymbol{\varepsilon}\right) - \frac{1}{\sqrt{1-\beta_1}}\left(\boldsymbol{z}_1 - \sqrt{\beta_1}\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1)\right)\right\|_2^2 + C \\
&= -\frac{\beta_1}{2a(1-\beta_1)}\left\|\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1) - \boldsymbol{\varepsilon}\right\|_2^2 + C.
\end{aligned}$$

C.2 Multinomial Diffusion

Closed Form Forward Sampling Formula

The closed form formulation of any mass function in the forward process, as stated in Equation (3.23), is derived here. This formula was simply stated by Hoogeboom et al. [49] without proof, but we prove it in detail here. Recall the categorical assumptions in the forward process,

$$\begin{aligned}
q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_t; (1-\beta_t)\boldsymbol{z}_{t-1} + \frac{\beta_t}{K}\mathbb{1}\right), \quad t \in \{2, \dots, T\}, \\
q(\boldsymbol{z}_1|\boldsymbol{x}) &= \text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_1; (1-\beta_1)\boldsymbol{x} + \frac{\beta_1}{K}\mathbb{1}\right).
\end{aligned}$$

These assumptions are applied iteratively to calculate the probability parameter of $q(\boldsymbol{z}_t|\boldsymbol{x})$, which we conveniently denote by $\boldsymbol{p}_{q(\boldsymbol{z}_t|\boldsymbol{x})}$. From our assumptions we know that

$$\boldsymbol{p}_{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})} = (1-\beta_t)\boldsymbol{z}_{t-1} + \frac{\beta_t}{K}\mathbb{1} = \alpha_t\boldsymbol{z}_{t-1} + \frac{\beta_t}{K}\mathbb{1}.$$

Using this, we derive that

$$\begin{aligned}
\boldsymbol{p}_{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-2})} &= \alpha_t\boldsymbol{p}_{q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_{t-2})} + \frac{\beta_t}{K}\mathbb{1} \\
&= \alpha_t\left((\alpha_{t-1})\boldsymbol{z}_{t-2} + \frac{\beta_{t-1}}{K}\mathbb{1}\right) + \frac{\beta_t}{K}\mathbb{1} \\
&= \prod_{i=t-1}^{t}\alpha_i\boldsymbol{z}_{t-2} + \frac{1}{K}\left(1 - \prod_{i=t-1}^{t}(1-\beta_i)\right)\mathbb{1} \\
&= \prod_{i=t-1}^{t}\alpha_i\boldsymbol{z}_{t-2} + \frac{1}{K}\left(1 - \prod_{i=t-1}^{t}\alpha_i\right)\mathbb{1}.
\end{aligned}$$

Applying this recursively, without showing all the details, leads to the probability parameter of interest,

$$\boldsymbol{p}_{q(\boldsymbol{z}_t|\boldsymbol{x})} = \prod_{i=1}^{t}\alpha_i\boldsymbol{x} + \frac{1}{K}\left(1 - \prod_{i=1}^{t}\alpha_i\right)\mathbb{1}$$

$$= \bar{\alpha}_t \boldsymbol{x} + \frac{1 - \bar{\alpha}_t}{K} \mathbb{1},$$

where we have used that $\boldsymbol{z}_0 \coloneqq \boldsymbol{x}$. Finally, this means that the closed form forward formula is

$$q(\boldsymbol{z}_t|\boldsymbol{x}) = \text{Categorical}_{\text{OHE}} \left(\boldsymbol{z}_t; \bar{\alpha}_t \boldsymbol{x} + \frac{1 - \bar{\alpha}_t}{K} \mathbb{1} \right),$$

as stated in Equation (3.23). As a side note, Austin et al. [3] state this formula in matrix form, without proof, in their generalized framework for diffusion applied to discrete data.

Forward Posterior Conditioned on Data

The forward posterior mass functions conditioned on a OHE categorical feature from an arbitrary observation from $\mathcal{D}$, $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$, $t \in \{2, \dots, T\}$, as given in Equation (3.24), are derived in this section. Hoogeboom et al. [49] state Equation (3.24) without proof, but we prove it in detail here. The starting point is the same as in Gaussian diffusion; Bayes' rule, as stated in Equation (C.5). Then, we simply replace the mass functions with their corresponding expressions from Section 3.4. The direct calculation reads

$$\begin{aligned}
&q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x}) \\
&= \frac{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{x}) q(\boldsymbol{z}_{t-1}|\boldsymbol{x})}{q(\boldsymbol{z}_t|\boldsymbol{x})} \quad \text{(Bayes' rule)} \\
&= \frac{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) q(\boldsymbol{z}_{t-1}|\boldsymbol{x})}{q(\boldsymbol{z}_t|\boldsymbol{x})} \quad \text{(Markov)} \\
&= \frac{\text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_t; (1 - \beta_t)\boldsymbol{z}_{t-1} + \frac{\beta_t}{K} \mathbb{1} \right) \text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_{t-1}; \bar{\alpha}_{t-1} \boldsymbol{x} + \frac{1 - \bar{\alpha}_{t-1}}{K} \mathbb{1} \right)}{\text{Cat}_{\text{OHE}} \left(x_t; \bar{\alpha}_t \boldsymbol{x} + \frac{1 - \bar{\alpha}_t}{K} \mathbb{1} \right)} \\
&= \frac{\text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_t; \alpha_t \boldsymbol{z}_{t-1} + \frac{1 - \alpha_t}{K} \mathbb{1} \right) \text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_{t-1}; \bar{\alpha}_{t-1} \boldsymbol{x} + \frac{1 - \bar{\alpha}_{t-1}}{K} \mathbb{1} \right)}{\text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_t; \bar{\alpha}_t \boldsymbol{x} + \frac{1 - \bar{\alpha}_t}{K} \mathbb{1} \right)} \qquad \text{(C.11)} \\
&= \frac{\text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_{t-1}; \alpha_t \boldsymbol{z}_t + \frac{1 - \alpha_t}{K} \mathbb{1} \right) \text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_{t-1}; \bar{\alpha}_{t-1} \boldsymbol{x} + \frac{1 - \bar{\alpha}_{t-1}}{K} \mathbb{1} \right)}{\text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_t; \bar{\alpha}_t \boldsymbol{x} + \frac{1 - \bar{\alpha}_t}{K} \mathbb{1} \right)} \qquad \text{(C.12)} \\
&\propto \text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_{t-1}; \alpha_t \boldsymbol{z}_t + \frac{1 - \alpha_t}{K} \mathbb{1} \right) \text{Cat}_{\text{OHE}} \left(\boldsymbol{z}_{t-1}; \bar{\alpha}_{t-1} \boldsymbol{x} + \frac{1 - \bar{\alpha}_{t-1}}{K} \mathbb{1} \right) \\
&= \text{Cat}_{\text{OHE}}(\boldsymbol{z}_{t-1}; \boldsymbol{\lambda}), \\
&\qquad \boldsymbol{\lambda} = \left[\alpha_t \boldsymbol{z}_t + \frac{1 - \alpha_t}{K} \mathbb{1} \right] \odot \left[\bar{\alpha}_{t-1} \boldsymbol{x} + \frac{1 - \bar{\alpha}_{t-1}}{K} \mathbb{1} \right], \qquad \text{(C.13)}
\end{aligned}$$

where we use *Cat* as a shorthand for *Categorical* to avoid line overflow. The equality of Equations (C.11) and (C.12) needs to be explained. The function $\text{Categorical}_{\text{OHE}} \left(\boldsymbol{z}_t; \alpha_t \boldsymbol{z}_{t-1} + \frac{1 - \alpha_t}{K} \mathbb{1} \right)$ can be written as

$$\text{Categorical}_{\text{OHE}} \left(\boldsymbol{z}_t; \alpha_t \boldsymbol{z}_{t-1} + \frac{1 - \alpha_t}{K} \mathbb{1} \right) = \begin{cases} \alpha_t + \frac{1 - \alpha_t}{K}, & \boldsymbol{z}_{t-1} = \boldsymbol{z}_t, \\ \frac{1 - \alpha_t}{K}, & \boldsymbol{z}_{t-1} \neq \boldsymbol{z}_t. \end{cases}$$

Notice that this function is symmetric with respect to the function argument and the latent variable realization in the probability parameter, i.e. $\boldsymbol{z}_t$ and $\boldsymbol{z}_{t-1}$ [49]. This implies that we can switch them around without changing the function, which yields

$$\text{Categorical}_{\text{OHE}}\left(\boldsymbol{z}_{t-1}; \alpha_t \boldsymbol{z}_t + \frac{1-\alpha_t}{K}\mathbb{1}\right) = \begin{cases} \alpha_t + \frac{1-\alpha_t}{K}, & \boldsymbol{z}_t = \boldsymbol{z}_{t-1}, \\ \frac{1-\alpha_t}{K}, & \boldsymbol{z}_t \neq \boldsymbol{z}_{t-1}, \end{cases}$$

which explains why Equation (C.11) is equal to Equation (C.12). Furthermore, we assume that the product of two categorical random variables is categorical with probability parameter equal to the product of the probability parameters of each of the factors. The final step in the derivation of the forward posterior in Equation (3.24) is the realization that the parameter $\boldsymbol{\lambda}$ in Equation (C.13) needs to be normalized such that $\|\boldsymbol{\lambda}\|_1 = 1$, i.e. such that the sum of all individual probabilities sum to one. This leads to the forward posterior given in Equation (3.24). As a side note, Austin et al. [3] state this formula in matrix form, without proof, in their generalized framework for diffusion applied to discrete data.

Appendix D

Data

This appendix is devoted to describing each of the datasets we use to evaluate our models, as listed in Table 5.1. Moreover, we explain how we perform rudimentary pre-processing of each dataset.

Adult Census Data (AD)

This is data from the 1994 Census database, containing information on individuals mostly from the United States. The dependent variable is binary, measuring if each individual has a yearly income above (1) or below (0) $50000 USD. The initial processing we perform for this dataset follows the steps:

- Download the data from UCI Machine Learning repository [26].
- Check if there are any missing values. The missing values are signalled by a question mark by default in the downloaded files. Approximately 7.5% of the total number of observations contain at least one piece of missing information. We simply remove all observations with at least one missing measurement.
- Remove the `education` column, since this information is essentially a more coarse representation of the information in the column `education_num`.

The total number of observations after this pre-processing is stated in Table 5.1. Furthermore, Table D.1 shows the features in **AD**, their data types, as well as simple descriptions of what the different quantities measure. Moreover, we indicate the range $[\min, \max]$ of each numerical feature, as well as the shares of categories in each non-numerical feature.

Churn Modelling (CH)

This dataset contains information on individual customers from a bank. The dependent variable is binary, measuring if the customer has left the bank (1) or not (0). The initial processing we perform for this dataset follows the steps:

- Download the data from Kaggle: `https://kaggle.com/datasets/shrutimechlearn/churn-modelling`
- Check if there are any missing values. There are none.
- Remove `CustomerID`, as this is a unique key for each customer and not interesting for prediction or inference.

- Remove `Surname`. It contains 2932 different names, where 1558 exist only once and 2311 exist less than five times. Thus, most of the names are recorded few times, which makes them difficult to draw inferences from in the first place. Moreover, one-hot encoding yields 2932 columns for this feature. For these two reasons, alongside computational difficulties following from such an encoding, we remove this feature. As a side note, the most common names are Smith, Martin, Scott, Walker and Brown, all recorded over 25 times in the dataset.
- Change the response name from `Exited` to `y`.
- Flip the original response class labels. Instead we use binary flag 1 if the customer is retained and 0 if the customer closed their account with the bank. We do this because counterfactuals are commonly focused on explaining how we can change a negative result to a positive one. While modelling customer churn, retaining a customer should be regarded as positive.

The total number of observations after this pre-processing is stated in Table 5.1. Furthermore, Table D.2 shows the features in **CH**, their data types, as well simple descriptions of what the different quantities measure. Moreover, we indicate the range $[\min, \max]$ of each numerical feature, as well as the shares of categories in each non-numerical feature.

Diabetes (DI)

This dataset contains data from a database titled "Pima Indians Diabetes Database" from the National Institute of Diabetes and Digestive and Kidney Diseases. It contains records of patients, where all individuals are females above 20 years old of Pima Indian heritage. The dependent variable is binary, stating if the individual has tested positively to diabetes (1) or not (0). The initial processing we perform for this dataset follows the steps:

- Download the data from OpenML: `https://openml.org/search?type=data&sort=runs&id=37&status=active` [26]
- Check if there are any missing values. There are none.
- Change the response name from `class` to `y`. Also, we change some of the other feature names to make them more understandable.
- Flip the original response class labels. Instead we set the response class labels to 1 if patient tested negative to diabetes and 0 if patient tested positive to diabetes. We do this for the same reason as earlier; we want 1 to signal a positive result in our application. In this case, testing negative should be regarded as a positive outcome for each patient.

The total number of observations after this pre-processing is stated in Table 5.1. Furthermore, Table D.3 shows the features in **DI**, their data types, as well as a simple description of what the different quantities measure. Moreover, we indicate the range $[\min, \max]$ of each numerical feature.

Table D.1: The features in the **Adult Census** (**AD**) dataset, along with their data types and descriptions of the quantities they measure. The description contains the range [min, max] of each numerical feature. In addition, it contains the levels, and their relative shares, of each non-numerical feature.

Column	Data Type	Description
`age`	integer	Age of individual $[17, 90]$.
`fnlwgt`	integer	"Final Weight": proxy for the demographic background of the individual $[13492, 1490400]$.
`education_num`	integer	Number of years of schooling $[1, 16]$.
`capital_gain`	integer	Capital gain incurred by the individual $[0, 99999]$.
`capital_loss`	integer	Capital loss incurred by the individual $[0, 4356]$.
`hours_per_week`	integer	Hours worked per week $[1, 99]$.
`workclass`	non-numerical	Private (74%), Self-emp-not-inc (8%), Local-gov (7%), State-gov (4%), Self-emp-inc (4%), Federal-gov (3%) or Without-pay ($\sim 0\%$).
`marital_status`	non-numerical	Married-civ-spouse (47%), Never-married (32%), Divorced (14%), Separated (3%), Widowed (3%), Married-spouse-absent (1%) or Married-AF-spouse ($\sim 0\%$).
`occupation`	non-numerical	Craft-repair (13%), Prof-specialty (13%), Exec-managerial (13%), Adm-clerical (12%), Sales (12%), Other-service (11%), Machine-op-inspct (7%), Transport-moving (5%), Handlers-cleaners (5%), Farming-fishing (3%), Tech-support (3%), Protective-serv (2%), Priv-house-serv (1%) or Armed-Forces ($\sim 0\%$).
`relationship`	non-numerical	Husband (41%), Not-in-family (26%), Own-child (15%), Unmarried (11%), Wife (5%) or Other-relative (2%).
`race`	non-numerical	White (86%), Black (9%), Asian-Pac-Islander (3%), Amer-Indian-Eskimo (1%) or Other (1%).
`sex`	binary	Male (68%) or Female (32%).
`native_country`	non-numerical	The native country of individual (41 levels): USA (91%), Mexico (2%), Philippines (0.6%), etc.
`y`	binary	The response; 1 if $>$ 50K (25%), 0 if $\leq$ 50K (75%), USD.

Table D.2: The features in the **Churn Modelling** (**CH**) dataset, along with their data types and descriptions of the quantities they measure. The description contains the range [min, max] of each numerical feature. In addition, it contains the levels, and their relative shares, of each non-numerical feature.

Column	Data Type	Description
`CreditScore`	integer	Credit score of customer $[350, 850]$.
`Age`	integer	Age of customer $[18, 92]$.
`Tenure`	integer	Number of years as customer $[0, 10]$.
`Balance`	float	Bank balance of customer $[0.0, 250898.1]$.
`NumOfProducts`	integer	Number of bank products customer has $[1, 4]$.
`EstimatedSalary`	float	Estimated yearly salary of customer in USD $[11.6, 199992.5]$.
`Geography`	non-numerical	France (50%), Germany (25%) or Spain (25%).
`Gender`	binary	Male (55%) or Female (45%).
`HasCrCard`	binary	Customer has credit card (71%) or not (29%).
`IsActiveMember`	binary	Customer is an active member of the bank (52%) or not (48%).
`y`	binary	The response; 1 if customer is retained (80%), 0 if customer churns (20%).

Table D.3: The features in the **Diabetes** (**DI**) dataset, along with their data types and descriptions of the quantities they measure. The description contains the range [min, max] of each numerical feature.

Column	Data Type	Description
`num_pregnant`	integer	Number of times pregnant $[0, 17]$.
`plasma`	integer	Plasma glucose concentration $[0, 199]$.
`dbp`	integer	Diastolic blood pressure (mm Hg) $[0, 122]$.
`skin`	integer	Triceps skin fold thickness (mm) $[0, 99]$.
`insulin`	integer	2-hour serum insulin (mu U/ml) $[0, 846]$.
`bmi`	float	Body mass index (kgm^{-2}) $[0.00, 67.10]$.
`pedi`	float	Diabetes pedigree function $[0.08, 2.42]$.
`age`	integer	Age of individual $[21, 81]$.
`y`	binary	The response; 1 if patient tests negative to diabetes (65%), 0 if test is positive (35%).

Appendix E

Multinomial Diffusion Implementation Notes

This appendix collects some considerations we make while implementing Multinomial diffusion. We recommend that the interested reader keeps this appendix in mind while studying our open source implementations of Multinomial and Tabular diffusion, as it helps to clarify some vital techniques.

As noted in Section 5.3, we implement Multinomial diffusion in *log-space*, inspired by Hoogeboom et al. [49]. In practice, this means that we work with logarithms of probabilities, instead of probabilities directly. It might not be immediately clear why this is a good choice; why does it promote numerical stability? The inventors do not provide any explicit reasoning behind this choice, except stating that their implementation[1] is done in a "safe manner". In Appendix A of their paper, they define a few helper functions in Python, that are used to compute quantities in log-space, as well as transform quantities to and from log-space, but they do not explain how these functions were designed. Due to this lack of contextualization, we outline where they can be derived from, while clarifying why computations in log-space are preferred.

Precisely, two different "tricks" are employed to avoid problems with *underflow*, i.e. assigning absolute values that are too small and out of the range of the declared data type. First of all, in order to avoid underflow, when calculating the product of two (or more) small quantities, we use the fact that

$$\log ab = \log a + \log b. \tag{E.1}$$

For simplicity, and without loss of generality, we illustrate how arithmetic operations are performed in log-space with only two quantities. Moreover, to make this exposition even simpler, we focus on a single example, on calculating the probabilities of the forward process densities, $q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$, in Multinomial diffusion. Specifically, we define

$$\boldsymbol{p} \coloneqq (1-\beta_t)\boldsymbol{z}_{t-1} + \frac{\beta_t}{K}\mathbb{1}. \tag{E.2}$$

Equation (E.1) is used to calculate such quantities in a numerically safe manner. In log-space, each of the terms in $\boldsymbol{p}$ can be represented as

$$\log\left((1-\beta_t)\boldsymbol{z}_{t-1}\right) = \log\left((1-\beta_t)\mathbb{1}\right) + \log \boldsymbol{z}_{t-1}, \tag{E.3}$$

and

$$\log \frac{\beta_t}{K}\mathbb{1} = \log \beta_t \mathbb{1} - \log K \mathbb{1}, \tag{E.4}$$

[1] `https://github.com/ehoogeboom/multinomial_diffusion`

respectively, which follows from Equation (E.1). Calculating the terms via addition in log-space avoids underflow in $\boldsymbol{p}$ when either of the factors in either of the terms are very small, because the multiplication is no longer performed. Thus, in practice, to ensure what Hoogeboom et al. [49] call a "safe" implementation, we calculate all individual terms in the categorical probability parameters in log-space.

Second, notice that, as a consequence of the first trick, we encounter problems when calculating $\boldsymbol{p}$, because it is not straightforward to use $\log\left(1-\beta_t\right)\boldsymbol{z}_{t-1}$ and $\log \frac{\beta_t}{K}\mathbb{1}$ to calculate $\boldsymbol{p}$ in Equation (E.2). Transforming calculations like Equations (E.3) and (E.4) from log-space back to linear-space can easily lead to underflow, for reasons we have already discussed. For instance, if we naively apply the exponential function to log-space calculations, when probabilities are very small, the exponential can underflow because of very small logarithmic inputs to the function. In practice, this means that we cannot reliably calculate $\boldsymbol{p} = (1-\beta_t)\boldsymbol{z}_{t-1} + \frac{\beta_t}{K}\mathbb{1}$ by first using the logarithm rules above, then applying an exponential and finally adding the individual terms. In order to solve this problem, one possible trick is to use $\log\left(1-\beta_t\right)\boldsymbol{z}_{t-1}$ and $\log \frac{\beta_t}{K}\mathbb{1}$ to calculate $\log \boldsymbol{p}$ directly in log-space. But how can this be done, since we cannot do it naively? In fact, such a calculation is widely used in ML and statistics, precisely for the reasons we outline. Keeping our example in mind, we are interested in calculating $\log \boldsymbol{p} = \log\left\{(1-\beta_t)\boldsymbol{z}_{t-1} + \frac{\beta_t}{K}\mathbb{1}\right\}$ given $\log \boldsymbol{a} \coloneqq \log\left(1-\beta_t\right)\boldsymbol{z}_{t-1}$ and $\log \boldsymbol{b} \coloneqq \log \frac{\beta_t}{K}\mathbb{1}$, which we assume have been previously calculated. For this purpose, the *log-sum-exp* function can be used. In general, in its naive form, it takes a real vector $\boldsymbol{u} = \{u^1, \ldots, u^p\}$ as input and returns

$$\operatorname{lse}(\boldsymbol{u}) \coloneqq \log \sum_{j=1}^{p} e^{u^j}. \tag{E.5}$$

From Equation (E.5), we notice that $\operatorname{lse}(\{\log \boldsymbol{a}, \log \boldsymbol{b}\}) = \log\left(\boldsymbol{a}+\boldsymbol{b}\right)$, which is why it is useful in our case. The reason it is attractive in practice is because it has been shown that it can be calculated in a numerically stable manner [9], and because it has implementations in commonly used libraries in high-level languages like Python [140]. Notice that

$$\log(a+b) = \log(a(1+\frac{b}{a})) = \log a + \log(1+\frac{b}{a}) = \log a + \log(1+e^{\log b - \log a}),$$

for any scalars a, b, which can be extended to rewrite the log-sum-exp function like

$$\operatorname{lse}(\boldsymbol{u}) = \log \sum_{j=1}^{p} e^{u^j} = \log \sum_{j=1}^{p} e^{d} e^{u^j - d} = d + \log \sum_{j=1}^{p} e^{u^j - d}. \tag{E.6}$$

Numerical problems occur in the exponential in Equation (E.6) if u^j, for any $j \in \{1, \ldots, p\}$, is very large compared to d, essentially making it "blow up". To mitigate this, define $d \coloneqq \max(\boldsymbol{u})$, such that $u^j - d \leq 0$ and hence $1 \geq e^{u^j - d} \geq 0$ [9]. Thus, returning to our simple example,

$$\begin{aligned}
\log \boldsymbol{p} &= \log\left\{(1-\beta_t)\boldsymbol{z}_{t-1} + \frac{\beta_t}{K}\mathbb{1}\right\} \\
&= \operatorname{lse}(\{\log(1-\beta_t)\boldsymbol{z}_{t-1}, \log \frac{\beta_t}{K}\mathbb{1}\}) \\
&= \boldsymbol{d} + \log\left(e^{\log(1-\beta_t)\boldsymbol{z}_{t-1} - \boldsymbol{d}} + e^{\log(\frac{\beta_t}{K}\mathbb{1}) - \boldsymbol{d}}\right),
\end{aligned}$$

where $\boldsymbol{d} = \max\left\{\log(1-\beta_t)\boldsymbol{z}_{t-1}, \log \frac{\beta_t}{K}\mathbb{1}\right\}$, is mathematically equivalent to the naive computations, but numerically stable. For clarity, the logarithm and the exponential are

applied element-wise to the vectors, as we have assumed throughout this entire discussion. This is precisely the trick that Hoogeboom et al. [49] use in the helper functions `log_add_exp` and `log_sum_exp` in Appendix A of their paper. In our implementation, instead of using these manual implementations, we rely on the methods `logaddexp`, `logsumexp` and `logcumsumexp` from PyTorch [98], which perform numerically stable computations following strategies similar to the one we have outlined in this appendix.

Appendix F

KL Divergence Between Two Multivariate Gaussians

In this appendix, we provide a proof of the general formula for the KL divergence between two multivariate normally distributed random variables. Assume two random variables $\boldsymbol{X}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\boldsymbol{X}_2 \sim \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ in $\mathbb{R}^n$. For completeness, recall that the probability density function of a random variable $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is

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

Note that it exists when $\boldsymbol{\Sigma}$ is symmetric and positive-definite. For clarity, the densities of $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are $p_1(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $p_2(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, respectively. The proof consists of the following direct calculation:

$$\begin{aligned}
&D_{KL}(p_1(\boldsymbol{x}) \parallel p_2(\boldsymbol{x})) \\
&= \int p_1(\boldsymbol{x}) \log\left(\frac{p_1(\boldsymbol{x})}{p_2(\boldsymbol{x})}\right) \mathrm{d}\boldsymbol{x} \\
&= \mathbb{E}_{p_1(\boldsymbol{x})}\left[\log p_1(\boldsymbol{x}) - \log p_2(\boldsymbol{x})\right] \\
&= \frac{1}{2}\,\mathbb{E}_{p_1(\boldsymbol{x})}\left[-\log\det\boldsymbol{\Sigma}_1 - (\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) + \log\det\boldsymbol{\Sigma}_2 + (\boldsymbol{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)\right] \\
&= \frac{1}{2}\log\frac{\det\boldsymbol{\Sigma}_2}{\det\boldsymbol{\Sigma}_1} + \frac{1}{2}\,\mathbb{E}_{p_1(\boldsymbol{x})}\left[-(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) + (\boldsymbol{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)\right] \qquad \text{(F.1)} \\
&= \frac{1}{2}\log\frac{\det\boldsymbol{\Sigma}_2}{\det\boldsymbol{\Sigma}_1} + \frac{1}{2}\,\mathbb{E}_{p_1(\boldsymbol{x})}\Big[-\operatorname{trace}\left(\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\right) \\
&\qquad\qquad + \operatorname{trace}\left(\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)(\boldsymbol{x}-\boldsymbol{\mu}_2)^T\right)\Big] \qquad \text{(F.2)} \\
&= \frac{1}{2}\log\frac{\det\boldsymbol{\Sigma}_2}{\det\boldsymbol{\Sigma}_1} - \frac{1}{2}\operatorname{trace}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_1\right) \\
&\qquad\qquad + \frac{1}{2}\operatorname{trace}\left(\boldsymbol{\Sigma}_2^{-1}\,\mathbb{E}_{p_1(\boldsymbol{x})}\left[(\boldsymbol{x}-\boldsymbol{\mu}_2)(\boldsymbol{x}-\boldsymbol{\mu}_2)^T\right]\right) \qquad \text{(F.3)} \\
&= \frac{1}{2}\log\frac{\det\boldsymbol{\Sigma}_2}{\det\boldsymbol{\Sigma}_1} - \frac{1}{2}n + \frac{1}{2}\operatorname{trace}\left(\boldsymbol{\Sigma}_2^{-1}\left(\boldsymbol{\Sigma}_1 + (\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)^T\right)\right) \qquad \text{(F.4)} \\
&= \frac{1}{2}\left(\log\frac{\det\boldsymbol{\Sigma}_2}{\det\boldsymbol{\Sigma}_1} - n + \operatorname{trace}\left(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1\right) + \operatorname{trace}\left(\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)^T\right)\right) \qquad \text{(F.5)} \\
&= \frac{1}{2}\left(\log\frac{\det\boldsymbol{\Sigma}_2}{\det\boldsymbol{\Sigma}_1} - n + \operatorname{trace}\left(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1\right) + (\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)\right). \qquad \text{(F.6)}
\end{aligned}$$

In the following, we explain the most important steps above, whose details we skipped. The equality of Equations (F.1) and (F.2) is based on the property

$$\operatorname{trace}(\boldsymbol{a}\boldsymbol{b}^T) = \boldsymbol{b}^T\boldsymbol{a}, \tag{F.7}$$

applied to the pairs of vectors $\boldsymbol{a} = \boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_1)$ and $\boldsymbol{b}^T = (\boldsymbol{x} - \boldsymbol{\mu}_1)$, as well as $\boldsymbol{a} = \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_2)$ and $\boldsymbol{b}^T = (\boldsymbol{x} - \boldsymbol{\mu}_2)$, respectively. Then, equality of Equations (F.2) and (F.3) holds because of linearity of the trace, i.e. $\mathbb{E}(\operatorname{trace}(\mathbf{A})) = \operatorname{trace}(\mathbb{E}(\mathbf{A}))$ for an arbitrary matrix $\mathbf{A}$. We use this to show that

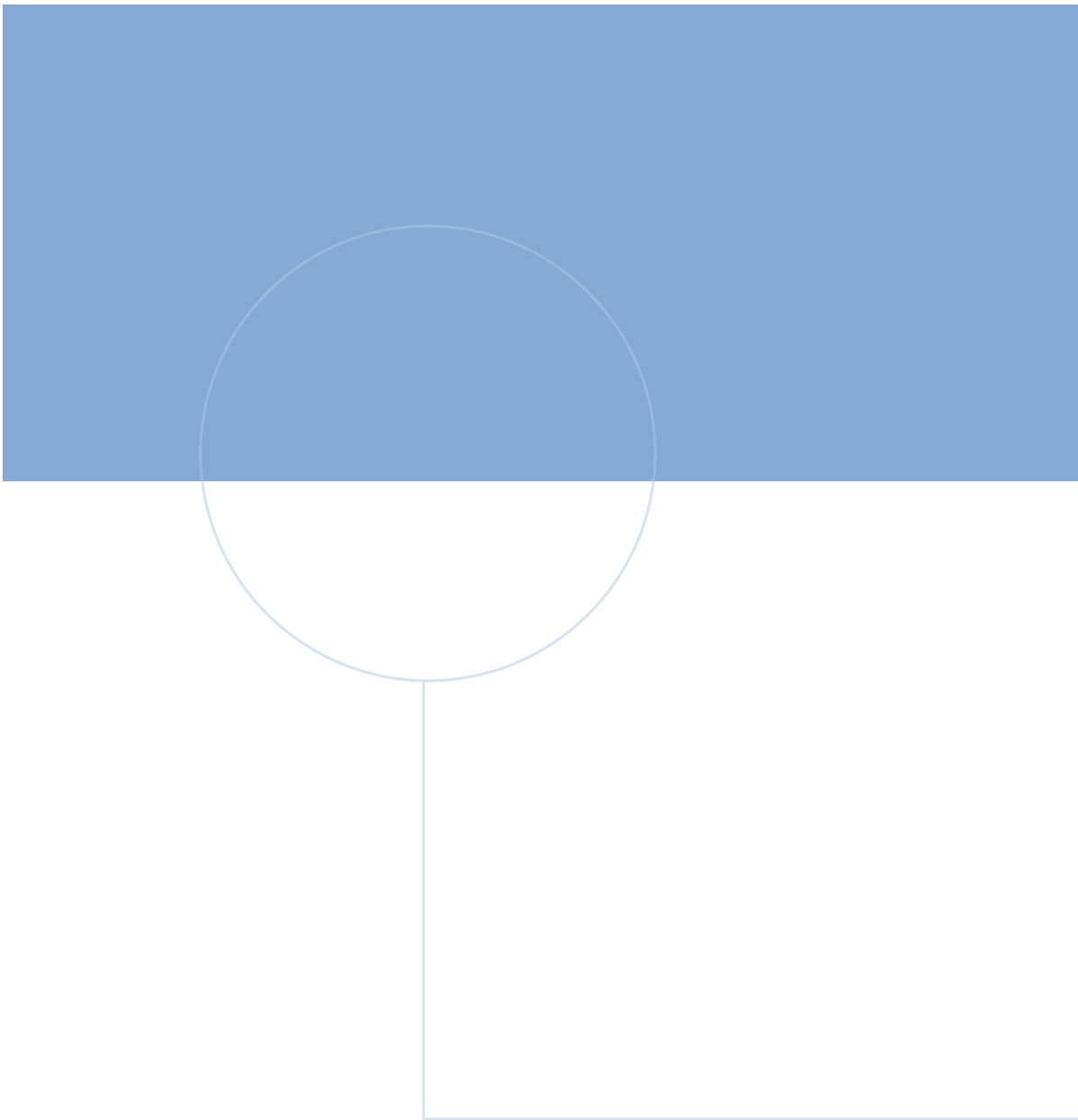
$$\begin{aligned}
\mathbb{E}_{p_1(\boldsymbol{x})}(\operatorname{trace}(\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_1)(\boldsymbol{x} - \boldsymbol{\mu}_1)^T) &= \operatorname{trace}(\mathbb{E}_{p_1(\boldsymbol{x})}(\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_1)(\boldsymbol{x} - \boldsymbol{\mu}_1)^T)) \\
&= \operatorname{trace}(\boldsymbol{\Sigma}_1^{-1}\,\mathbb{E}_{p_1(\boldsymbol{x})}((\boldsymbol{x} - \boldsymbol{\mu}_1)(\boldsymbol{x} - \boldsymbol{\mu}_1)^T)) \\
&= \operatorname{trace}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_1).
\end{aligned}$$

The linearity of the trace is also used to obtain the third term in Equation (F.3). For completeness, we mention that we also use the linearity of the expectation several places. Later, equality of Equations (F.3) and (F.4) is based on the result

$$\mathbb{E}_{p_1(\boldsymbol{x})}\left[(\boldsymbol{x} - \boldsymbol{\mu}_2)\,(\boldsymbol{x} - \boldsymbol{\mu}_2)^T\right] = \mathbb{E}_{p_1(\boldsymbol{x})}\left[((\boldsymbol{x} - \boldsymbol{\mu}_1) - (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1))\,((\boldsymbol{x} - \boldsymbol{\mu}_1) - (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1))^T\right]$$

$$\begin{aligned}
&= \mathbb{E}_{p_1(\boldsymbol{x})}\left[(\boldsymbol{x} - \boldsymbol{\mu}_1)(\boldsymbol{x} - \boldsymbol{\mu}_1)^T - (\boldsymbol{x} - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T\right. \\
&\qquad \left. -(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{x} - \boldsymbol{\mu}_1)^T + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T\right] \\
&= \boldsymbol{\Sigma}_1 - \underbrace{\mathbb{E}_{p_1(\boldsymbol{x})}\left[(\boldsymbol{x} - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T\right]}_{=0} \\
&\qquad - \underbrace{\mathbb{E}_{p_1(\boldsymbol{x})}\left[(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{x} - \boldsymbol{\mu}_1)^T\right]}_{=0} \\
&\qquad + \mathbb{E}_{p_1(\boldsymbol{x})}\left[(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T\right] \\
&= \boldsymbol{\Sigma}_1 + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T,
\end{aligned}$$

where both the middle terms are equal to zero because the random variable $\boldsymbol{X}_1 - \boldsymbol{\mu}_1$ is centered at $\mathbf{0}$. This can easily be demonstrated algebraically as well, by calculating the expectations. Moreover, the trivial property $\operatorname{trace}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_1) = \operatorname{trace}(\mathbf{I}) = n$ is used to obtain the first term in Equation (F.4). Finally, equality of Equations (F.5) and (F.6) is attributed to Property (F.7), with $\boldsymbol{a} = \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$ and $\boldsymbol{b}^T = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T$.



NTNU

Norwegian University of
Science and Technology