Advances in Deep Generative Models, Approximate Inference, and their Applications

Federico Bergamin

DTU

Advances in Deep Generative Models, Approximate Inference, and their Applications

Federico Bergamin

DTU

Ph.D. Thesis
Doctor of Philosophy

DTU Compute
Department of Applied Mathematics and Computer Science

Advances in Deep Generative Models, Approximate Inference, and their Applications

Federico Bergamin

Kongens Lyngby 2024

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersens Plads
Building 324
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

Applications of deep learning and deep generative models are becoming increasingly influential. With these models becoming an integral part of our lives, we must ensure that they are not only accurate, but also robust and safe. However, neural networks have been found to be overconfident both in misclassified examples and on test examples that do not belong to the distribution used to generate the training data. At the same time, deep generative models have been shown to sometimes assign a higher likelihood to out-of-distribution examples. This makes the safe deployment of deep learning models challenging. However, the success of deep generative models in image and text generation has paved the way for the application of these models in different fields and different tasks.

This thesis consists of four different self-contained research articles addressing some of the challenges and opportunities mentioned above. We start by focusing on the problem of deep generative models, sometimes assigning a higher likelihood to out-of-distribution examples. In this context, we proposed to combine different test statistics for anomaly detection using Fisher's method. We then present a method that borrow tools from Riemannian geometry to avoid that classic Laplace approximation spread mass on low-probability regions of the true posterior when performing approximate inference for neural networks. In the second part of this dissertation, we focus on two different applications of deep generative models. We start by analyzing how clustering is performed with variational autoencoders and then extend this framework to perform co-clustering, i.e. simultaneously clustering both rows and columns of a data matrix. The last article of this thesis applies diffusion models for modeling long-range rollouts of partial differential equations, highlighting the fact that training a joint model and performing conditioning via reconstruction guidance is a promising way to have a flexible and general way to use the same model to tackle different tasks.

List of publications

This doctoral thesis is based on the following contributions:

- Paper 1:
  Model-agnostic out-of-distribution detection using combined statistical tests
  Federico Bergamin*, Pierre-Alexandre Mattei*, Jakob D. Havtorn, Hugo Senetaire, Hugo Schmutz, Lars Maaløe, Søren Hauberg, Jes Frellsen
  AISTATS 2022 (International Conference on Artificial Intelligence and Statistics)
  *Equal contribution

- Paper 2:
  Riemannian Laplace approximations for Bayesian neural networks
  Federico Bergamin, Pablo Moreno-Muñoz, Søren Hauberg, Georgios Arvanitidis
  NeurIPS 2023 (Neural Information Processing Systems)

- Paper 3:
  Clustering and co-clustering incomplete data with deep latent variable models
  Federico Bergamin, Pierre-Alexandre Mattei*, Jes Frellsen*
  Under review at ICML 2024 (International Conference on Machine Learning)
  *Equal supervision

- Paper 4:
  Guided Autoregressive Diffusion Models with Applications to PDE Simulation
  Federico Bergamin*, Cristiana Diaconu*, Aliaksandra Shysheya*, Paris Perdikaris, José Miguel Hernández-Lobato, Richard E. Turner, Emile Mathieu
  Under review at ICML 2024 (International Conference on Machine Learning)
  Shorter version appearing at the ICLR 2024 Workshop on AI4DifferentialEquations in Science
  *Equal contribution

Preface

This thesis was prepared at the Section for Cognitive Systems at the Department of Applied Mathematics and Computer Science (DTU Compute), Technical University of Denmark. It constitutes a partial fulfillment of the requirements for acquiring a PhD at the Technical University of Denmark.

The PhD project was supervised by Associate Professor Jes Frellsen, Professor Søren Hauberg, and Research Scientist Pierre-Alexandre Mattei and was funded by the Innovation Fund Denmark (grant nr 0175-00014B). The PhD project was carried out at DTU during the period October 2020 - March 2024, except for a three-month external stay at the University of Cambridge under the supervision of Professor Richard Turner and Postdoctoral Research Associate Emile Mathieu.

The work I did during my PhD covers two different topics: deep generative models and approximate inference. This resulted in four different research papers (two peer-reviewed publications and two preprints currently under review) which constitute the main chapters of this dissertation.

Kongens Lyngby, June 17, 2024

Federico Bergamin

Acknowledgements

"If you see a turtle on the top of a fence post, it did not get there by itself."

I would like to start by thanking Ole Winther, Eric Nalisnick, and Matthias Bauer for accepting the invitation to be on my committee and for taking the time to read and provide feedback on this dissertation.

The past three years and a half have been quite a journey. When I think about it, my mind mostly brings back fond memories (Mitchell et al., 1997, cfr. rosy retrospection). I have learned a lot in this limited time, I have grown up both as a researcher and as a person and shared the road with amazing people that I am lucky enough to call friends. No journey is always smooth, and my PhD journey is no exception. I came to the conclusion that going through a PhD is in part taking a crash course on most emotions that characterized someone's life. Indeed, our lives are rarely one relentless conveyor belt of successes, but are full of ups and downs, bad luck and disappointment, frustration, and things we just cannot explain. And so was my PhD, with the drawback that everything is compressed in three years.

If I write about this in my thesis, it means that I made it through this roller coaster ride. And if I made it through, much of the credit goes to the fact that I was not alone in this. There are many people who helped me to swim in cold dark water, which represents research because that is where the most interesting swimming happens.

First and foremost, I will forever be grateful to my supervisors Jes Frellsen, Pierre-Alexandre Mattei, and Søren Hauberg. Jes, thanks for giving me the opportunity to do a PhD in machine learning in the first place, allowing me to learn how to do research, despite my knowledge of the field at the time was standing on shaking fundamentals. You always believed in me and trusted me in the past years and I appreciate this. Also, thanks for all the support and freedom you gave me. I appreciate the mutual honesty and transparency between us, allowing me to always be open with you and vice versa. Indeed, I tend to value human values on the same level as professional ones. Thanks also for engaging in battles that you could have easily avoided if I had been better at planning and decision making, such as fighting to allow me to go on a "too late" external stay. I am very grateful for that.

Thanks Pierre for joining the ride from the very beginning without even knowing me. I am always amazed by your impressive knowledge of fundamental topics. Your knowledge enhanced every project we collaborated on. Thanks for your patience and availability to explain to me sometimes even the simplest things. Thank you for taking the time to read my dissertation and for providing helpful feedback.

Thanks Søren for your valuable pieces of advice and for your availability whenever I wanted to discuss something. I am a huge fan of your open-door policy. I am also always amazed by your intuition and simple questions that always challenge my idea of having truly understood something.

I would also like to thank Georgios Arvanitidis, with whom I collaborated in the final part of my PhD. Our collaboration was one of the highlights of my PhD. Thank you for teaching me Riemannian geometry and for believing in me in a period when I was mostly failing. I will be forever grateful for that.

As part of my PhD, I was fortunate enough to be able to spend three months visiting the University of Cambridge. I am grateful for that to Emile Mathieu, Rich Turner, and Jose Miguel Hernández-Lobato. Emile, I cannot express my gratitude enough to you. By answering my cold email, you gave me the possibility to spend three months at the University of Cambridge. Thanks for all the honest discussions about research life, for teaching me how to be a better software engineer (I still have so much to improve on that aspect), and for all the chats about traveling by train or by bike. I really have learned a lot from you. Rich and Miguel, thanks for your trust and for investing your time in this project. Your feedback and observations have been invaluable. I have learned a lot from you during my brief time there, and I would have hoped I could have stayed longer to continue learning. I am still amazed by your dedication to research. In general, my experience in Cambridge is something that I will always carry with me.

I would like to thank all my colleagues and visitors in Jes' group: Anshuk, Hugo, Paul, Maxim, Jakob, Kristoffer, Marloes, Ignacio, Johannes, and Dennis. It was really nice having you in the past three years. Thanks for making an open and safe environment for discussions. I enjoyed all the conversations and discussions we had together and honestly learned a lot from each of you along the way. I also want to express my gratitude for the time we have shared at conferences, summer schools, and even during casual moments like lunch or coffee breaks. Anshuk and Maxim also thank you for always taking care of organizing the classic Christmas lunch at Barr or organizing spontaneous Friday beers. I also thank you Marloes for sharing your knowledge about proteins with me, I have learned a lot from you.

As I mentioned before, the PhD is full of hardships. However, I was able to always keep my motivation high because of the supportive and nice atmosphere I had in my office. I am sincerely grateful to all the people with whom I had the privilege to share 321/220. Manja, Dimitris, Simon, and Alison, thank you for the many interesting discussions and the long coffee breaks. Thanks for being always available for a chat whenever I felt like it and for listening to my rants and complaints.

To Pablo and Kilian: That I have now handed in my thesis is, in part, thanks to the two of you. There was quite a chance I would have quit without you. You are the ones with whom I have spent the most time in the last three years. I have seen you so often that you became my family away from home. I have the feeling that sometimes you were also genuinely happier than me for my successes. You really helped me grow as a person and as a friend and helped me have real discussions and defend my positions. I think I still have a lot to learn from you. While the idea of us going separate ways in a few months is worrying, I am choosing to focus on the present and I am grateful for your friendship on this journey.

I would also like to extend my appreciation to Mikkel and François for being part of this journey with me. Thanks for all the stimulating discussions in the office and for your friendship outside of its walls. Our memorable trip to Valencia for AISTATS and the enjoyable after-work training rides are something I will always carry with me.

I would also like to thank Lars Kai Hansen for making CogSys an amazing place to do a PhD. I also want to thank all the people who I met in the last three years in the section: Marco, Hrittik, Nico, Stas, Giulia, Pola, Cilie, and Frederik. Thank you for the pleasant chats we had during lunch and coffee breaks. It has been a pleasure pursuing my PhD alongside such smart people. Frederik, thank you for welcoming me into "Danish society" by inviting me to your farewell party before your departure to California. I will remember that party forever. I would also like to thank Anne for helping me with all the bureaucracy while she was a part of our section.

While in Cambridge I was extremely lucky to collaborate with Cristiana and Sasha. You got to work with me in a period where I was forced to manage multiple deadlines simultaneously. I could not have asked for better collaborators. I deeply appreciate your patience with me, I enjoyed working and fighting against reviewers side by side with you and learned a lot from these experiences. I also would like to extend my gratitude to all the people who made me feel welcome and with whom I had meaningful discussions during my time there. Thank you, Vincent, Lawrence, Isaac, Stratis, Luis, Xienda, and Matthew.

Two people inspired me and I never had the opportunity to say thank you. First, I would like to thank Andriy Mnih for being always available for a chat at the beginning of my PhD. I admire your modesty and you are one of the role models in research for me. I also thank Carl-Henrik Ek, with whom I share the passion for cycling, for showing me that research and hobby can go hand in hand.

There are phases in a PhD that seem like living inside a bubble–periods, where I ended up constantly worrying about experiments failing, writing papers, and I forgot about taking care of myself. Even more challenging was the tendency to let the results I was obtaining define me as a person, rather than recognizing them as a small part of who I am. During such times, I was incredibly fortunate to have people outside of this bubble who were there to support me and remind me to "recalibrate" this perspective. I am grateful to have Davide and Maja in my life. At the beginning of my PhD, before you moved back to Italy, you were my great companion of adventures. Thank you for organizing hiking and camping trips, as well as bikepacking trips, and for always being available to offer practical advice, whether it was related to science or everyday matters.

To Emanuele and Matteo: I have known you since I moved to Denmark eight years ago and, since then, I have always been able to count on you. Growing up abroad involves discovering new reference points, and I can honestly say that you are the ones I have always looked up to as I strive to become a better person. I am grateful for all the "therapy sessions" we had in the Lord Nelson pub. Thank you for always pushing me outside my comfort zone. Lele, also thanks for all the rides and adventures you organized, they truly helped distract me over the last three years. I would also like to thank all the other Italians in the group for always welcoming me with such warmth and enthusiasm in your activities.

To Andrea and Enrico: You inspired me to pursue a PhD in the first place while we were studying together for our Master's degree. Thanks for visiting me while I was in Cambridge, I have really good memories of those days. Although we are not all together in Copenhagen anymore and we see each other less frequently, I am still grateful for the time we spent together, the beers at Charlie's bar, and the valuable lessons I have learned from you.

I would also like to thank all the friends I have in Italy: Giacomo, Alessandro, Thomas, Nicola, Federico, Stefano, Andrea, Luigi, Noemi, Anna, and Simone. I may not be the one who often calls or sends messages, but I would like to express my sincere gratitude to you for always waiting for me to be home and making me feel like I never left. I am lucky to have you in my life.

Finally, I would like to express my gratitude to my parents and sisters for their unconditional love and unwavering support. I understand the challenges of having a family member who has only returned home sporadically over the past few years. I want to thank you for patiently bearing with me during those times when I called home feeling frustrated or even avoided calling altogether when my research wasn't progressing as expected or when I had too much to do before a deadline. Your support means the world to me, and I am deeply thankful for your understanding and encouragement over the past three years.

Contents

Summary i

List of publications iii

Preface v

Acknowledgements vii

Contents xi

1 Introduction 1
1.1 Research Questions and Contributions 2

2 Technical Background 7
2.1 Deep generative models 7
2.1.1 Variational Autoencoders 10
2.1.2 Normalizing Flows 17
2.1.3 Autoregressive Models 19
2.1.4 Score-based Models 20
2.1.5 On Likelihood and Typicality for out-of-distribution detection . 30
2.2 Laplace Approximation 33
2.3 A primer on differential geometry 43

3 Model-agnostic out-of-distribution detection using combined statistical tests 53

4 Riemannian Laplace approximations for Bayesian neural networks 85

5 Clustering and co-clustering incomplete data with deep latent variable models 117

6 Guided autoregressive diffusion models with applications to PDE simulation 145

7 Final Remarks 179

Bibliography of the Background 183

CHAPTER 1

Introduction

In the past decade, the adoption of machine learning-based solutions for decision-making has increased for different real-world applications. This trend is motivated by the impressive results that neural networks have achieved in various supervised learning tasks, ranging from image classification and object detection (Krizhevsky et al., 2012; Redmon et al., 2016) to natural language processing (Sutskever et al., 2014; Hinton et al., 2012). The goal of these tasks is to *learn* or *model* the relation between certain inputs, denoted by $\mathbf{x}$, and their respective targets y. From a probabilistic perspective, these tasks can be defined as approximating the conditional distribution $p(y|\mathbf{x})$ of the target y given the example $\mathbf{x}$.

The success achieved in these supervised learning tasks has fueled research for the application of machine learning models on different data modalities such as video, graphs, and molecules (Butler et al., 2018; Oprea et al., 2020; Noé et al., 2020; Bronstein et al., 2021; Duval et al., 2023). In addition to that, it has also driven research to explore the application of machine learning for unsupervised learning tasks, where only unlabeled data is required. One such task is generative modeling, where one of the goals is to learn a model that generates new examples similar to those used during training. From a probabilistic perspective, this corresponds to learning a distribution $p(\mathbf{x})$ from which we can easily generate new samples $\mathbf{x} \sim p(\mathbf{x})$. This also led to the great success of large deep generative models for both text (Achiam et al., 2023) and image generation (Vahdat & Kautz, 2020; Ho et al., 2020; Rombach et al., 2022) that we have witnessed since two years ago. These models today are not only used by researchers and industry professionals – they have become part of many tools used by everybody (Ramesh et al., 2021; Rombach et al., 2022; Achiam et al., 2023). This highlights how machine learning is becoming an integral part of our lives and will continue to play an important role in the future.

Due to the increasing applications and the influence that machine learning models will likely have on our daily lives, if we want to deploy and use these models safely, we must ensure that they are not only accurate but also robust to distribution shifts. Ideally, we would like these models to be aware of *what they do not know*. From a practical point of view, models, whether they are regression models or classifiers, should exhibit low confidence in their predictions for inputs that significantly deviate from the training data. Similarly, generative models should assign low likelihoods to such inputs. Consider, for example, the problem of making a supervised learning model $p(y|\mathbf{x})$ more robust to inputs that come from a different distribution than the

one it was trained on. A simple solution for this problem was already proposed by Bishop (1994). The idea is to add another layer of security by pre-processing the examples that are given as input to $p(y|\mathbf{x})$. In fact, we can make an assumption about the true process generating the data being defined as $p(\mathbf{x}, y) = p(y|\mathbf{x})p(\mathbf{x})$, meaning that in principle we first sample an input $\mathbf{x}$ and then get its corresponding label y by sampling from $p(y|\mathbf{x})$. The idea, therefore, is to train a density model $p(\mathbf{x})$, which can be given by a deep generative model, and use it as a filter. This way, if a certain input $\mathbf{x}$ has low density under $p(\mathbf{x})$, we can defer the decision to a human expert. Although this simple solution seems appealing, it does not work in practice. Indeed, neural networks have been found to be overconfident both on misclassified examples (Guo et al., 2017) and on test examples that do not belong to the distribution used to generate the training data (Nguyen et al., 2015). Furthermore, deep generative models have been shown to sometimes assign a higher likelihood to out-of-distribution examples (Nalisnick et al., 2018; Hendrycks et al., 2019). This behavior of deep generative models also prevents their application as density models to filter the inputs we want to evaluate using our $p(y|\mathbf{x})$ model based on their likelihood under $p(\mathbf{x})$.

In this dissertation we present four different self-contained research papers. The first two papers can be interpreted as new ways to tackle the two problems we have mentioned above. Indeed, Paper 1 focuses on the problem of deep generative models assigning a higher likelihood to out-of-distribution examples. Paper 2 tackles the problem of neural networks being overconfident away from the training data by using approximate Bayesian inference. The remaining two papers, as we are going to mention in the next section, deal with the application of deep generative models for two different tasks. Paper 3 will present an analysis of clustering using a specific class of deep generative models and it extends the framework to co-clustering, while Paper 4 analyzes and investigates the use of deep generative models for simulating partial differential equations.

1.1 Research Questions and Contributions

In contrast to some PhD dissertations, this work is not focused on a single application or a single method, but instead spans multiple methods and problems and also focuses on solving several different tasks. The field of deep learning is moving and evolving incredibly fast, with methods and techniques changing over time. This work reflects the change of interest during my PhD studies, as doing research on a topic is an effective way to learn about it. Therefore, there is no common red thread that connects all the works we will present in this thesis. In the previous section we mentioned that, from a higher level, we can interpret the first two papers as dealing with uncertainty and overconfidence and the last two as dealing with two different applications of deep generative models. In the following, we are going to outline the structure of this dissertation and we describe more in detail the research questions

that these papers are trying to answer.

In Chapter 2 we introduce the relevant background needed to understand the remaining chapters. This chapter will cover the basic concepts of deep generative models, mostly focusing on two classes, variational autoencoders, and diffusion models. After presenting the concept of typicality, showing why high-dimensional probability distributions are unintuitive, we will introduce the Laplace approximation as a standard and successful way to turn a pre-trained neural network into a Bayesian Neural Network. In the end, we will give a primer on a few basic concepts from differential geometry that are needed to understand Paper 2. The remaining chapters contain my main research contributions and answer the following, somewhat unrelated, research questions:

Research Question 1: *How can we make out-of-distribution detection using deep-generative models more robust?*

Nalisnick et al. (2018); Hendrycks et al. (2019) were the first to highlight the problem of deep generative models sometimes assigning a higher likelihood to out-of-distribution data than in-distribution data. This made people question their effectiveness as density estimators since the likelihood might fail. Therefore, since Nalisnick et al. (2018); Hendrycks et al. (2019), different test statistics or methodologies were proposed to show that deep generative models can actually distinguish between *in-distribution* and *out-of-distribution* data. However, Zhang et al. (2021) proved that, in the case of single sample out-of-distribution detection, where the goal is to classify a single sample as an inlier or an outlier, no test statistic is constantly better than all possible alternatives.

In Chapter 3, we investigate whether there are any benefits in combining multiple statistical tests to perform anomaly detection with deep generative models. We start by revisiting classical statistical tests for anomaly detection and focus our attention on Rao's score test (Rao, 1948). We then derive an alternative formulation for the typicality score (Nalisnick et al., 2019) and empirically show their complementarity, which leads us to consider their combination using Fisher's method (Fisher, 1925).

Research Question 2: *Can we define a flexible approximate posterior that adapts to the nonlinear structure of the true posterior while sampling remains efficient?*

As we have highlighted in the introduction, modern neural networks are overconfident both in misclassified examples and on inputs that are from a completely different distribution (Guo et al., 2017; Nguyen et al., 2015). The Bayesian formulation of deep learning, which aims to learn a distribution over the weights instead of a single set of parameters, has the potential to solve these issues by providing uncertainty estimates and by improving the model's robustness. However, due to the high-dimensional parameter space, the computation of the exact posterior distribution is challenging,

making these methods rely on simpler approximations, mostly in the form of a Gaussian distribution. In recent years, Laplace approximation has emerged as a practical way and one of the best-performing methods for inference in neural networks, thanks to advances in approximating the Hessian of a neural network in a scalable way. However, the classic Laplace approximation is well known to underfit (Lawrence, 2001, Chapter 5)(Ritter et al., 2018), as the Gaussian approximation spreads probability mass in low posterior regions. A solution to alleviate it was already proposed by (Mackay, 1992, Chapter 4) and later justified by Khan et al. (2019); Immer et al. (2021b) and consists of linearizing the neural network when estimating the predictive posterior.

In Chapter 4, we investigate whether it is possible to solve the underfitting problem of classic Laplace without having to consider the linearization of the neural network at prediction time. We propose a simple parametric approximate posterior that adapts to the shape of the true posterior through a Riemannian metric that is determined by the log-posterior gradient. We can then obtain samples from our proposed solution by solving a system of ordinary differential equations (ODEs), which can be done efficiently by leveraging the structure of the Riemannian metric and automatic differentiation.

Research Question 3: *Which is the correct way to perform clustering using variational autoencoders? Can we easily extend this framework to perform co-clustering?*

A classic probabilistic approach to clustering is to assume that the data is generated conditioned on a latent cluster distribution. A typical approach is then to train a variational autoencoder, a specific class of deep generative models, with a mixture of Gaussians prior and then cluster the data by computing the responsibility under the different prior components using the mean of the variational distribution. In addition to that, despite the success of deep learning methods in clustering, the use of deep learning in co-clustering, where the goal is to cluster observations and features simultaneously, is still limited.

In Chapter 5, we start by formalizing clustering using a variational autoencoder with a mixture of Gaussians prior by proposing a tractable estimate for genuine quantity we are interested in for clustering in this type of model. We analyze the benefit of using our proposed estimate compared to the standard way people perform clustering using VAEs. We then present how deep latent variable models can be extended to perform co-clustering both on fully observed datasets and datasets with missing values. We also present two different ways of using such models to impute those missing values.

Research Question 4: *How can we create a generative model to model and simulate accurately long partial-differential equations (PDEs) trajectories? Can diffusion models be used for such a task?*

Partial differential equations are a powerful mathematical framework for modeling physical phenomena such as fluid motion and thermodynamics. Some applications often require solving the same PDE but with different initial conditions. For this reason, in recent years, there has been an increased interest in using deep learning methods to learn neural approximations of PDE solutions (Raissi et al., 2017). One of the main challenges of using deep generative models on this task is that modeling the full joint distribution over the trajectory can become prohibitively expensive for long trajectories and large input spaces.

In Chapter 6, we present a joint diffusion model trained on short trajectory segments and conditioned a posteriori, allowing for autoregressive generation of accurate and stable predictions of long PDE trajectories. We tested our approach both for forecasting and data assimilation, showing that these types of models, while being more flexible as they can handle different tasks a posteriori, are also competitive with diffusion models that are trained instead for those specific tasks.

CHAPTER 2

Technical Background

In this chapter, we provide the relevant background for the contributions contained in this thesis. We start by introducing the basic concepts of *generative models* by focusing on the classes of models that we used in the remainder of the thesis by highlighting their different assumptions over the generative process of the data. This will form the basis for understanding the models used in Papers 1, 3, and 4. In the context of generative models, we will introduce the concept of typicality which will be used in Paper 1. We then briefly delve into approximate Bayesian inference by presenting the Laplace approximation and the techniques that made this approximation recently very successful in turning standard deep neural networks into Bayesian neural networks. To fully understand the idea we propose in Paper 2, we need to introduce some tools from differential geometry. Therefore, the last section of this background section will serve as a primer on differential geometry. We would also highlight that this chapter aims to provide the reader with the basic tools to understand the thesis, it is also written as personal lecture notes, and sometimes there will be pedantic step-by-step derivations. Gray boxes are used to present additional details and remarks. As with any textbook, there is a non-zero probability that there will be typos or errors, I apologize in advance for that.

2.1 Deep generative models

Deep generative models (DGMs) aim to combine the uncertainty coming from a probabilistic model with the flexibility given by neural networks to approximate complex, high-dimensional data distributions. The goal of a generative model is to fit a parametric probability distribution p_θ where $\theta \in \Theta$ are the learnable parameters such that it is able to approximate a certain data generating distribution p_{data}. Learning the optimal parameters θ is usually done by minimizing a divergence measure $\mathcal{D}$ between p_{data} and p_θ:

$$\arg\min_{\theta \in \Theta} \mathcal{D}(p_{\text{data}} || p_\theta). \tag{2.1}$$

The divergence $\mathcal{D}$ represents the notion of a "distance" between two distributions, and thus it has to satisfy the following "distance-like" properties: $\mathcal{D}(p||q) \geq 0$ for all p, q, where p and q are two probability distributions, and $\mathcal{D}(p||q) = 0$ if and only if $p = q$. Possible examples are the Kullback-Leibler (KL) divergence, the Wasserstein distance, the maximum mean discrepancy (MMD), etc. If we assume

that both probability distributions are defined on $\mathcal{X}$ and consider the Kullback-Leibler divergence, for example, which is defined[1] as

$$\begin{aligned}\mathrm{KL}(p_{\text{data}}||p_\theta) &= \int_{\mathcal{X}} \log\left(\frac{p_{\text{data}}(\mathbf{x})}{p_\theta(\mathbf{x})}\right) p_{\text{data}}(\mathbf{x})d\mathbf{x} \\ &= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log\left(\frac{p_{\text{data}}(\mathbf{x})}{p_\theta(\mathbf{x})}\right)\right]. \end{aligned} \tag{2.2}$$

We can rewrite the expectation by using the properties of the logarithm

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log\left(\frac{p_{\text{data}}(\mathbf{x})}{p_\theta(\mathbf{x})}\right)\right] = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_{\text{data}}(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_\theta(\mathbf{x})].$$

We can see that the first term on the right-hand side does not depend on $\theta \in \Theta$ and therefore we can avoid considering it when minimizing (2.1). Therefore, if our divergence measure is the KL divergence, optimizing (2.1) corresponds to maximizing the "population likelihood", as we get:

$$\arg\min_{\theta\in\Theta} \mathrm{KL}(p_{\text{data}}||p_\theta) = \arg\max_{\theta\in\Theta} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_\theta(\mathbf{x})]. \tag{2.3}$$

We do not have access to the true distribution p_{data}, making the maximization defined in (2.3) impossible. However, if we have access to samples from p_{data}, or a dataset, $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathcal{X}^n$, where the data instances are independent and identically distributed (IID), we can define the empirical distribution[2] $p_{\mathcal{D}} = \frac{1}{n}\sum_{i=1}^n \delta_{\mathbf{x}_i}$ as a mixture of Dirac distributions supported on $\mathbf{x}_i$, and we use these samples to approximate the expectation in (2.3). This will correspond to optimizing

$$\arg\max_{\theta\in\Theta} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_\theta(\mathbf{x})] \approx \arg\max_{\theta\in\Theta} \mathbb{E}_{\mathbf{x}\sim p_{\mathcal{D}}}[\log p_\theta(\mathbf{x})] = \arg\max_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^n \log p_\theta(\mathbf{x}_i), \tag{2.4}$$

which is equivalent to expressing the expectation with respect to p_{data} as a Monte Carlo estimate using the training examples. The maximum likelihood estimate (MLE) objective can be interpreted as finding the parameters θ that maximize the log-likelihood of the observed data. Since for deep generative models, we assume that p_θ is a simple distribution parameterized through a neural network with parameters θ, the optimization is usually performed by doing gradient descent or stochastic gradient descent.

[1] While the "distance"-like property is satisfied, the KL divergence is asymmetric, meaning that $\mathrm{KL}(p_{\text{data}}||p_\theta) \neq \mathrm{KL}(p_\theta||p_{\text{data}})$. Furthermore, we assume that p_{data} is *absolutely continuous* with respect to p_θ, which means that both admit density with respect to the same measure. Generally speaking, for the KL to be defined, $p_{\text{data}}(\mathbf{x})$ and $p_\theta(\mathbf{x})$ we assume they have the same support.

[2] In this case $p_{\mathcal{D}}$ is a measure that may not have a density. In this dissertation, we will use the terms probability distribution and probability measure interchangeably

Tasks. Deep generative models have several applications. Indeed, there are different fields where one is interested in approximating a high-dimensional distribution and sample from it. While initially generative models were considered for sampling new images (Van Den Oord et al., 2016; Van den Oord et al., 2016; Kingma & Dhariwal, 2018; Vahdat & Kautz, 2020; Child, 2021; Ho et al., 2020; Ramesh et al., 2021), videos (Kalchbrenner et al., 2017; Babaeizadeh et al., 2021; Saxena et al., 2021; Clark et al., 2019; Luc et al., 2020; Höppe et al., 2022; Yang et al., 2023), audio (Chung et al., 2015; Van Den Oord et al., 2016), in recent years there was an increased interest in using generative models in physical sciences, such as biology and physics. In these settings, possible applications involve sampling new proteins or molecules (Corso et al., 2023; Schneuing et al., 2022; Ingraham et al., 2019; Watson et al., 2023) and modeling the dynamics of partial differential equations (Kohl et al., 2023; Yang & Sommer, 2023; Rozet & Louppe, 2023b). In Chapter 6, for example, we will analyze whether diffusion models can be used to model the underlying dynamics of different PDEs and then use them to generate long trajectories conditioned on some initial observations.

A generative model can also be used for density estimation, which can be useful for computing $p_\theta(\hat{\mathbf{x}})$ for a new sample $\hat{\mathbf{x}} \sim q(\mathbf{x})$, where $q(\mathbf{x})$ can be any probability distribution on $\mathcal{X}$. This can be useful for out-of-distribution detection (Bishop, 1994), for example, although directly using the log-likelihood $\log p_\theta(\hat{\mathbf{x}})$ has been shown to be a flawed proxy (Nalisnick et al., 2018; Hendrycks et al., 2019). Our Paper 1 shed some light on and propose a new way to use a generative model exactly for this task.

Due to their probabilistic formulation, deep generative models can handle the presence of missing values in the dataset by performing the imputation of such values. In fact, there are different procedures to use the learned $p_\theta(\mathbf{x})$ for imputing a new test example $\hat{\mathbf{x}} = \{\hat{\mathbf{x}}^\text{m}, \hat{\mathbf{x}}^\text{o}\} \in \mathcal{X}$ (Rezende et al., 2014) or to fit a generative model directly on $\mathbf{X} \in \mathcal{X}^n$ (Nazabal et al., 2020; Mattei & Frellsen, 2019), which consists of some observed values $\mathbf{X}^\text{o}$ and missing entries $\mathbf{X}^\text{miss}$. Image in-painting or other inverse problems can be seen as missing values imputation problems. In Chapter 5 and Chapter 6, we will touch on this problem in two different settings.

In addition to the aforementioned applications, certain generative models can be used for representation learning (Bengio et al., 2013), with the goal of learning a meaningful representation of high-dimensional data that will be beneficial for downstream tasks (Finn et al., 2016; Ha & Schmidhuber, 2018; Kingma et al., 2014), or for discovering new structures in the data, such as possible clusters (Jiang et al., 2017). In Chapter 5, we will tackle the problem of clustering and co-clustering using deep latent variable models.

Types of deep generative models. Due to the challenges arising from modeling complex high-dimensional distributions, different types of generative models have

been proposed in recent years. All these approaches differ in terms of data generation process assumption and in terms of whether they allow for exact likelihood computation. The main types are variational autoencoders (VAE) (Rezende et al., 2014; Kingma & Welling, 2014), autoregressive models (ARM) (Neal, 1992; Bengio & Bengio, 1999; Larochelle & Murray, 2011; Van Den Oord et al., 2016), normalizing flows (Rezende & Mohamed, 2015; Papamakarios et al., 2021), diffusion- and score-based models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song & Ermon, 2019; Song et al., 2020a), energy-based models (Hinton, 2002; Teh et al., 2003; Hinton et al., 2006; Du & Mordatch, 2019; Grathwohl et al., 2019), generative adversarial networks (Goodfellow et al., 2014), and probabilistic circuits (Peharz et al., 2020b,a). In the next sections, we will briefly present VAEs, flows, ARMs, and score-based models, as these will provide sufficient background to understand the papers composing this dissertation. We will mostly focus on VAEs and score-based diffusion models since these are the basics of Chapter 5 and Chapter 6. Hierarchical VAEs and state-of-the-art implementation of flows and ARMs are used for out-of-distribution detection in Chapter 3.

2.1.1 Variational Autoencoders

In this section, we present variational autoencoders (VAEs) as a framework to model input data $\mathbf{x} \in \mathcal{X}$ using a *deep latent variable models* (DLVMs) parameterized by neural networks, whose parameters are optimized via amortized variational inference.

Deep latent variable models. Before jumping into VAEs, the reader should become familiar with latent variable models. These are probabilistic models that relate the observed variable $\mathbf{x} \in \mathcal{X}$, which in our case are actual data, with some unobserved variables $\mathbf{z} \in \mathcal{Z}$, referred to as a latent variable. Therefore, using the notation introduced above, a latent variable model uses the following parametric probability distribution p_θ

$$p_\theta(\mathbf{x}) = \int_{\mathcal{Z}} p_\theta(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z}, \tag{2.5}$$

to approximate the p_{data}, where $p_\theta(\mathbf{x}, \mathbf{z})$ is the joint distribution over observed and latent variables and p_θ is called the marginal density of $\mathbf{x}$. One of the most common factorizations considered for $p_\theta(\mathbf{x}, \mathbf{z})$ by latent variable models is

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}), \tag{2.6}$$

where $p_\theta(\mathbf{x}|\mathbf{z})$ is the likelihood or the observation model and $p_\theta(\mathbf{z})$ is the prior distribution over the latent variables. Generally, they assume that the dimensionality of $\mathcal{Z}$ is smaller than $\mathcal{X}$, although in more recent generative models that use the same factorization, this is not the case (Vahdat & Kautz, 2020; Ho et al., 2020). This corresponds to assuming that the input data $\mathbf{x}$ is generated from a latent, unobserved

and usually lower-dimensional variable $\mathbf{z} \in \mathcal{Z}$, or in a similar way, that the generative process works as follows: first, you sample a latent variable $\mathbf{z} \sim p_\theta(\mathbf{z})$ and then you generate a new sample $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$ where conditioning on the previously generated latent variable. This assumption can be motivated by the *manifold hypothesis*[3], which states that data lie near or on a lower-dimensional manifold embedded within the high-dimensional data space $\mathcal{X}$. The hope is that the latent variable $\mathbf{z}$ captures the main factors of variation in $\mathbf{x}$, providing a compressed but still meaningful representation. This also reflects the reason why latent variable models are used to perform representation learning and dimensionality reduction.

DLVMs parameterize the conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$ with a neural network, increasing thus the modeling capacity and flexibility of the generative model. Formally, DLVMs consider the observation model of the following form $p_\theta(\mathbf{x}|\mathbf{z}) = \Phi(\mathbf{x}|f_\theta(\mathbf{z}))$, where the function $f_\theta : \mathcal{Z} \to H$ is parameterized as a neural network with parameters θ and takes as input the latent variable $\mathbf{z}$. This is usually referred to as the *decoder*. $\{\Phi(\cdot|\eta)\}_{\eta \in H}$ is a parametric family of distributions, called the *output density*, e.g. the Gaussian distribution for continuous data, i.e. $p(\mathbf{x}|\mathbf{z}) = \mathrm{N}(\mathbf{x}|\mu_\theta(\mathbf{z}), \sigma_\theta(\mathbf{z})\mathbf{I})$ or the Bernoulli distribution for binary data, i.e. $p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{D} \mathrm{Ber}(x_i|\sigma(f_\theta(\mathbf{z}))$ where σ here indicates the sigmoid function. The decoder is therefore transforming the latent variable $\mathbf{z}$ into the parameters η of the output distribution. The prior distribution is usually assumed to be a standard Gaussian distribution, but, as we will see later in this section, there are different alternatives, as using a different prior will correspond to enforcing different structures on the latent space.

Inference. Assume we have now access to samples from p_{data}, or a dataset, $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}^T \in \mathcal{X}^n$. As we have seen in Section 2.1, by assuming the KL divergence as the statistical divergence between $p_{data}(\mathbf{x})$ and the parametric distribution $p_\theta(\mathbf{x})$, we can find the optimal θ parameters by performing maximum likelihood estimation where we approximate the expectation using the empirical distribution. In the DLVM setting, this would correspond to the following maximum likelihood estimate:

$$\begin{aligned} \arg\max_{\theta \in \Theta} \mathbb{E}_{\mathbf{x} \sim p_{\mathcal{D}}}[\log p_\theta(\mathbf{x})] &= \arg\max_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(\mathbf{x}_i) \\ &= \arg\max_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \log \int_{\mathcal{Z}} p_\theta(\mathbf{x}_i|\mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\mathbf{z}, \end{aligned} \tag{2.7}$$

where we used the the factorization in (2.6). However, directly optimizing (2.7) is often intractable since there is no closed form solution for the integral and naive

[3]This is particularly true in the case of continuous data $\mathbf{x} \in \mathbb{R}^d$ and the observation model $p(\mathbf{x}|\mathbf{z})$ is a Gaussian distribution with mean $f(\mathbf{z})$. In that case, the manifold is given by $f(\mathbf{Z}) = \{f(z), \mathbf{z} \in \mathbf{Z}\}$.

approximations have high variance due to the high-dimensionality of the latent space $\mathcal{Z}$[4]. One possible way to address this issue is to employ *variational inference*:

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log \int_{\mathcal{Z}} p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\mathbf{z} \\
&= \log \int_{\mathcal{Z}} \frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) q_\phi(\mathbf{z})}{q_\phi(\mathbf{z})} \mathrm{d}\mathbf{z} \qquad \text{(variational distribution } q_\phi(\mathbf{z})\text{)} \\
&= \log \mathbb{E}_{q_\phi(\mathbf{z})} \left[\frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z})} \right] \\
&\geq \mathbb{E}_{q_\phi(\mathbf{z})} \left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z})} \right] \qquad \text{(using Jensen's inequality)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z})} [\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z})] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z})||p_\theta(\mathbf{z})) = \mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x}),
\end{aligned} \tag{2.8}$$

where we introduce a variational distribution $q_\phi(\mathbf{z})$ on the second line, which corresponds to performing importance sampling (Andrieu et al., 2003), and use Jensen's inequality and the fact that the logarithm is a concave function on the fourth line. The objective given by $\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x})$ is called *evidence lower bound* (ELBO) and is a lower bound on $\log p_\theta(\mathbf{x})$.

The ELBO $\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x})$ can also be derived from a posterior point of view. Indeed, the posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is intractable because $p_\theta(\mathbf{x})$ appears in the denominator. While one approach is to use Markov Chain Monte Carlo (MCMC) (Neal, 1993; Hoffman, 2017) or Hamiltonian Monte Carlo (HMC) (Neal, 2011) to get samples from the exact posterior, it will result in a computationally expensive procedure. We can, instead, define a variational posterior $q_\phi(\mathbf{z})$ as above, which approximates the true intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$. We can then write the KL divergence between $q_\phi(\mathbf{z})$ and the intractable true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ and by using its definition we get:

$$\begin{aligned}
\mathrm{KL}(q_\phi(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})) &= \mathbb{E}_{q_\phi(\mathbf{z})} \left[\log \frac{q_\phi(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})} \right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})} [\log q_\phi(\mathbf{z}) - \log p_\theta(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})} [\log q_\phi(\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z})} [\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})}] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})} [\log q_\phi(\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z})} [\log p_\theta(\mathbf{x}, \mathbf{z})] + \log p_\theta(\mathbf{x}).
\end{aligned} \tag{2.9}$$

By rearranging it we get:

$$\log p_\theta(\mathbf{x}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z})} [\log p_\theta(\mathbf{x}, \mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z})} [\log q_\phi(\mathbf{z})]}_{\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x})} + \mathrm{KL}(q_\phi(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})). \tag{2.10}$$

[4] We can approximate the integral by using a naive Monte Carlo estimator $p(\mathbf{x}_i) = \mathbb{E}_{p(\mathbf{z})}[p_\theta(\mathbf{x}_n|\mathbf{z})] \approx \frac{1}{K} \sum_{k=1}^{K} p(\mathbf{x}_i|\mathbf{z}_k)$. However, this has high variance and therefore will result in unstable optimization.

Since $\mathrm{KL}(q_\phi(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})) \geq 0$, then we can see that $\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x})$ is a lower bound of $\log p_\theta(\mathbf{x})$. By rewriting $\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x})$, we can retrieve (2.8).

Variational autoencoder. In the standard variational inference setting, we need to optimize the variational distribution $q_\phi(\mathbf{z})$ for each input $\mathbf{x} \in \mathcal{X}$, that is, each data point $\mathbf{x}_i$ has its own variational parameters $q_{\phi_i}(\mathbf{z}_i)$. This results in a really expensive optimization for large datasets, even if the variational distribution family is simple enough as a diagonal Gaussian. A different approach is to use *amortized variational inference* (Gershman & Goodman, 2014), which learns a common inference function g_ϕ that maps each datapoint $\mathbf{x}_i$ to the parameters of the corresponding variational distribution $q_{\phi_i}(\mathbf{z}_i)$. Formally, we can write that the variational distribution has the following form $q_\phi(\mathbf{z}|\mathbf{x}) = \Psi(\mathbf{z}|g_\phi(\mathbf{x}))$, where $g_\phi : \mathcal{X} \to \mathcal{K}$ is a function that maps each datapoint to the parameters of the variational distribution family $\{\Psi(\cdot|\kappa)\}_{\kappa\in\mathcal{K}}$ defined in $\mathcal{Z}$. In this way, the variational parameters that now are the parameters of this function g_ϕ, are shared between data points. In the variational autoencoder setting, the function g_ϕ is parameterized by a neural network and it is called *encoder*. This is the setup described by Rezende et al. (2014); Kingma & Welling (2014). The ELBO $\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x})$ thus becomes:

$$\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})), \tag{2.11}$$

where the first term can be interpreted as a reconstruction error and the KL can be seen as a regularization term, which limits the information passing in the latent, such that the model can generalize and not only reconstruct training examples. The optimization of both the observation model θ and the variational parameters ϕ is done using stochastic gradient descent by minimizing the negative ELBO. Again using the notation we presented in Section 2.1, this corresponds to $\arg\min_{\theta,\phi} \mathbb{E}_{\mathbf{x}\sim p_\mathcal{D}}[-\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x})]$[5], where the expectation is estimated using samples from the empirical distribution $p_\mathcal{D}$. Since we need to sample from $q_\phi(\mathbf{z}|\mathbf{x})$ to obtain an unbiased Monte Carlo estimate of (2.11), we need to backpropagate through the sampling step. In order to do that, Kingma & Welling 2014 showed that if $q_\phi(\mathbf{z}|\mathbf{x})$ belongs to a certain distribution family, such as the location-scale family that includes the Gaussian distribution among others, a sample $z \sim q_\phi(\mathbf{z}|\mathbf{x})$ can be expressed as a deterministic and differentiable transformation of an auxiliary random variable that does not depend on $\mathbf{x}$. If $q_\phi(\mathbf{z}|\mathbf{x})$ is a Gaussian distribution $\mathrm{N}(\mathbf{z}|\mu_\phi(\mathbf{x}), \sigma_\phi(\mathbf{x}))$ with mean and variance parameterized with neural networks, then we can express a sample from the distribution as $\mathbf{z} = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x})\epsilon$ with $\epsilon \sim \mathrm{N}(0, 1)$. Thanks to this trick, commonly referred to as *reparameterization trick* or *pathwise gradient estimator*, we can compute the derivative of $\mathbf{z}$ with respect to ϕ using standard backpropagation. Alternatives to estimate this gradient exist, including score function estimators (Williams, 1992), but we refer to (Mohamed et al., 2020) for further details, as these are beyond the scope of this

[5]Note that the ELBO is also analytically intractable. However, we can obtain an unbiased estimate by a Monte Carlo approximation of the expectation over $q_\phi(\mathbf{z}|\mathbf{x})$.

dissertation. VAEs are also closely related to dimensionality reduction techniques. Indeed, if we defined the decoder as $p_\theta(\mathbf{x}|\mathbf{z}) = \Phi(\mathbf{x}|f_\theta(\mathbf{z})) = \mathrm{N}(\mathbf{x}; \mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$, i.e., a linear function used to parameterize a Gaussian likelihood, this is closely related to probabilistic principal components analysis (PPCA).

Different priors. The common choice for the prior over the latent variables $p_\theta(\mathbf{z})$ is a standard Gaussian distribution, i.e. $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$. This prior is simple, and it has no parameter to be learned. We can also understand the role of the prior by rewriting the KL term from (2.8) in a different way. By defining the aggregated posterior as $q_\phi(\mathbf{z}) = \mathbb{E}_{x\sim p_\mathcal{D}}[q_\phi(\mathbf{z}|\mathbf{x})] = \frac{1}{N}\sum_{n=1}^{N} q_\phi(\mathbf{z}|\mathbf{x}_n)$, Hoffman & Johnson (2016); Bauer & Mnih (2019) showed that we can rewrite $\mathbb{E}_{\mathbf{x}\sim p_\mathcal{D}}[\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))]$ as

$$
\begin{aligned}
&\mathbb{E}_{p_\mathcal{D}}[\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))] = \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})p_\mathcal{D}(\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})}\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log \frac{1}{p_\theta(\mathbf{z})}\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})p_\mathcal{D}(\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{x},\mathbf{z})}{q(\mathbf{x})}\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log \frac{1}{p_\theta(\mathbf{z})}\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})p_\mathcal{D}(\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{x},\mathbf{z})}{q(\mathbf{x})}\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})p_\mathcal{D}(\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{z})}{q_\phi(\mathbf{z})}\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log \frac{q_\phi(\mathbf{z})}{p_\theta(\mathbf{z})}\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})p_\mathcal{D}(\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{x},\mathbf{z})}{q(\mathbf{x})q_\phi(\mathbf{z})}\right] \\
&= \mathrm{KL}(q_\phi(\mathbf{z})||p_\theta(\mathbf{z})) + \mathbf{I}_{q_\phi(\mathbf{z}|\mathbf{x})p_\mathcal{D}(\mathbf{x})}[\mathbf{x};\mathbf{z}]
\end{aligned}
\tag{2.12}
$$

where $\mathbb{I}_{q_\phi(\mathbf{z}|\mathbf{x})p_\mathcal{D}}(\mathbf{z};\mathbf{x})$ is the mutual information between the latent representation and the example. The equation (2.12) tells us that by maximizing the ELBO, we are both minimizing the mutual information between $\mathbf{x}$ and $\mathbf{z}$ which goes against the intuition of VAEs as learning a representation of the data and encouraging the aggregate posterior to match the prior distribution. Indeed, the KL term will be minimized if $q_\phi(\mathbf{z}) = p_\theta(\mathbf{z})$. However, when the prior is fixed as is the case of the standard Gaussian, the aggregate posterior does not necessarily fully follow the prior distribution, resulting in the prior assigning probability mass in areas of the latent space where the aggregated posterior has zero mass. This is mainly because during the optimization of the VAE parameters θ and ϕ the decoder is forcing the encoder to have the smallest variance possible, while at the same time, the KL term in (2.11) is preventing this to happen. Therefore, it is unlikely for the aggregated posterior to fit perfectly the prior, leaving this *holes* where the prior has probability mass but the aggregated posterior does not. Consequently, at sampling time, if we sample from one of these holes then the samples usually have low quality. This is referred to as the prior-aggregated posterior mismatch problem (Rosca et al., 2018). A way to alleviate this problem is to learn the prior and to make it more flexible (Tomczak & Welling, 2018; Bauer & Mnih, 2019).

Additionally, from (2.12), we can see that the prior is one way to impose additional structure on the latent space. For example, if one wants the latent space to present clusters, one can substitute the standard normal with a Mixture of Gaussians prior. The prior is thus defined as $p_\theta(\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathrm{N}(\mathbf{z}|\mu_k, \sigma_k^2\mathbf{I})$, where π_k, μ_k, σ_k^2 are trainable parameters, and K is the number of components considered. We analyzed clustering using a VAE and how to extend a VAE for co-clustering in Chapter 5.

Hierarchical VAEs. As we mentioned above, a popular way to make a DLVM more flexible is to consider a hierarchy of latent variables $\mathbf{z} = \mathbf{z}_1, \ldots, \mathbf{z}_L$, where $\mathbf{z}_1$ is the latent closest to $\mathbf{x}$ in the hierarchy. Therefore, the generative process can be expressed as

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z}_1)p_\theta(\mathbf{z}_1|\mathbf{z}_2)\cdots p_\theta(\mathbf{z}_{L-1}|\mathbf{z}_L)p_\theta(\mathbf{z}_L), \tag{2.13}$$

There are two different ways to factorize the variational distribution. One is the so-called *bottom-up* inference (Burda et al., 2016), where the variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is given by

$$q_\phi(\mathbf{z}|\mathbf{x}) = q_\phi(\mathbf{z}_1|\mathbf{x})q_\phi(\mathbf{z}_2|\mathbf{z}_1)\cdots q_\phi(\mathbf{z}_L|\mathbf{z}_{L-1}) = q_\phi(\mathbf{z}_1|\mathbf{x})\prod_{l=2}^{L} q_\phi(\mathbf{z}_l|\mathbf{z}_{l-1}). \tag{2.14}$$

In this bottom-up inference model, latent variables in the upper stochastic layers of hierarchical VAEs tend to become inactive, meaning that they are not contributing to learning a meaningful representation. Sønderby et al. (2016) proposed to use a *top-down* inference model as a solution, usually referred to as Ladder VAE, defined as

$$q_\phi(\mathbf{z}|\mathbf{x}) = q_\phi(\mathbf{z}_L|\mathbf{x})q_\phi(\mathbf{z}_{L-1}|\mathbf{z}_L)\cdots q_\phi(\mathbf{z}_1|\mathbf{z}_2) = q_\phi(\mathbf{z}_L|\mathbf{x})\prod_{l=1}^{L-1} q_\phi(\mathbf{z}_l|\mathbf{z}_{l+1}). \tag{2.15}$$

The two different inference strategies also lead to two different ELBOs. For simplicity, we consider just two stochastic layers, that is, $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z}_1)p_\theta(\mathbf{z}_1|\mathbf{z}_2)p(\mathbf{z}_2)$. In the case of the bottom-up inference, the ELBO reads as

$$\begin{aligned}\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}_1|\mathbf{x})q_\phi(\mathbf{z}_2|\mathbf{z}_1)}[\log p_\theta(\mathbf{x}|\mathbf{z}_1)] + \mathbb{E}_{q_\phi(\mathbf{z}_1|\mathbf{x})}\left[\mathbb{E}_{q_\phi(\mathbf{z}_2|\mathbf{z}_1)}\left[\log \frac{p_\theta(\mathbf{z}_1|\mathbf{z}_2)}{q_\phi(\mathbf{z}_1|\mathbf{x})}\right]\right] \\ - \mathbb{E}_{q_\phi(\mathbf{z}_1|\mathbf{x})}[\mathrm{KL}(q_\phi(\mathbf{z}_2|\mathbf{z}_1)||p_\theta(\mathbf{z}_2))],\end{aligned} \tag{2.16}$$

where the second term cannot be rewritten as a KL divergence. In the top-down inference scheme, instead, the ELBO can be written as

$$\begin{aligned}\mathcal{L}^{\mathrm{ELBO}}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}_2|\mathbf{x})q_\phi(\mathbf{z}_1|\mathbf{z}_2)}[\log p_\theta(\mathbf{x}|\mathbf{z}_1)] - \mathrm{KL}(q_\phi(\mathbf{z}_2|\mathbf{x})||p_\theta(\mathbf{z}_2)) \\ - \mathbb{E}_{q_\phi(\mathbf{z}_2|\mathbf{x})}[\mathrm{KL}(q_\phi(\mathbf{z}_1|\mathbf{z}_2)||p_\theta(\mathbf{z}_1|\mathbf{z}_2))].\end{aligned} \tag{2.17}$$

Training a VAE on incomplete datasets. We present how we can train a VAE on incomplete datasets under the assumption that the missing mechanism is either *missing-completely-at-random* (MCAR) or *missing-at-random* (MAR). We will present this for a single stochastic-layer VAE, but this can be easily extended to hierarchical VAE. We will focus on the approach proposed by Mattei & Frellsen (2019) which we also have used in Paper 3 to train our model on incomplete datasets. Under the MCAR assumption, the missing mechanism is independent of the data. For example, the probability that any value is missing is given by constant. In the MAR assumption, instead, the missing mechanism depends on the observed values. Assume that we have a dataset $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T \in \mathcal{X}^n$, that is, we have n training example. In case of missing values, we can represent every training example $\mathbf{x}_i = \{\mathbf{x}_i^o, \mathbf{x}_i^m\}$, where $\mathbf{x}_i^o$ are the observed values and $\mathbf{x}_i^m$ the missing ones. Under the MCAR and MAR assumptions, if we want to fit a deep generative model $p_\theta(\mathbf{x})$ on the incomplete data, we can just maximize the log-likelihood of the observed data $\mathbf{x}^o$, i.e.:

$$\arg\max_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(\mathbf{x}_i^o) = \arg\max_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \log \int_{\mathcal{Z}} p_\theta(\mathbf{x}_i^o | \mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\mathbf{z}. \tag{2.18}$$

As before, we rely on amortized variational inference to define a lower-bound on the maximum-likelihood estimate. In this case, we can write the variational distribution in the following way $q_\phi(\mathbf{z}|\mathbf{x}^o) = \Psi(\mathbf{z}|g_\phi(\iota(\mathbf{x}^o))$, where $\iota(\mathbf{x}^o)$ is an imputation function such that $\iota(\mathbf{x}^o) \in \mathcal{X}$ and g_ϕ is the encoder as we have presented above. Therefore, the ELBO in this case can be expressed as

$$\log p_\theta(\mathbf{x}^o) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^o)}[\log p_\theta(\mathbf{x}^o|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}^o)||p_\theta(\mathbf{z})), \tag{2.19}$$

and we can use it to learn θ and ϕ. Alternative approaches to the one we have just presented exist (Nazabal et al., 2020; Ma et al., 2018, 2019; Ivanov et al., 2018), but they are beyond the scope of this dissertation.

Remarks on Equation (2.18). We have presented at a high level how to train a VAE on an incomplete dataset. Rubin (1976) was the first one investigating when the observed likelihood presented in (2.18) is the correct objective for training a model in the presence of missing values. We will present the notation and explain more in detail the MAR and MCAR assumptions and why we can fit a model by just maximizing the likelihood of the observed data. We start by formalizing the notation following Ghahramani & Jordan (1995): we consider to have a dataset containing datapoints $\mathbf{x} \in \mathcal{X}$, where in this case $\mathcal{X} = \mathcal{X}^1 \times \cdots \times \mathcal{X}^p$, i.e., each datapoint has p features. Therefore, the dataset is $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \in \mathcal{X}^n$. The missing mechanism is defined as a random variable $\mathbf{s} \in \{0, 1\}^p$, where 1 indicates that x is observed and 0 that it is missing. Therefore, given $\mathbf{s}$, we can split a single example $\mathbf{x} = \{\mathbf{x}^o, \mathbf{x}^m\}$. We assume that $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$ and, as we have done before, we use a model

$p_\theta(\mathbf{x})$ to approximate it. We also approximate the true missing mechanism $\mathbf{s} \sim p(\mathbf{s}|\mathbf{x})$ with a model $p_\phi(\mathbf{s}|\mathbf{x})$ where ϕ are the model parameters. The joint probability distribution is given by:

$$p_{\theta,\phi}(\mathbf{x}, \mathbf{s}) = p_\phi(\mathbf{s}|\mathbf{x})p_\theta(\mathbf{x}).$$

We can define the MCAR and MAR mechanisms in terms of the missing mechanism model:

- MCAR: $p_\phi(\mathbf{s}|\mathbf{x}) = p_\phi(\mathbf{s})$, that is, the probability of $\mathbf{x}_i$ missing is independent of its value.
- MAR: $p_\phi(\mathbf{s}|\mathbf{x}) = p_\phi(\mathbf{s}|\mathbf{x}^o, \mathbf{x}^m) = p_\phi(\mathbf{s}|\mathbf{x}^o)$, that is, the probability that $\mathbf{x}_i$ is missing may depend on the values of $\mathbf{x}_k$ for $k \neq i$, assuming that $\mathbf{x}_k$ is observed.

The learning of θ, ϕ is done by optimizing $p_{\theta,\phi}(\mathbf{x}^o, \mathbf{s})$. We can write this as follows:

$$p_{\theta,\phi}(\mathbf{x}^o, \mathbf{s}) = \int p_\phi(\mathbf{s}|\mathbf{x}^o, \mathbf{x}^m)p_\theta(\mathbf{x}^o, \mathbf{x}^m)\mathrm{d}\mathbf{x}^m.$$

In the MCAR and MAR settings, this can be rewritten as

$$p_{\theta,\phi}(\mathbf{x}^o, \mathbf{s}) = p_\phi(\mathbf{s}) \int p_\theta(\mathbf{x}^o, \mathbf{x}^m)\mathrm{d}\mathbf{x}^m = p_\phi(\mathbf{s})p_\theta(\mathbf{x}^o) \qquad \text{(MCAR setting)}$$

$$p_{\theta,\phi}(\mathbf{x}^o, \mathbf{s}) = p_\phi(\mathbf{s}|\mathbf{x}^o) \int p_\theta(\mathbf{x}^o, \mathbf{x}^m)\mathrm{d}\mathbf{x}^m = p_\phi(\mathbf{s}|\mathbf{x}^o)p_\theta(\mathbf{x}^o) \quad \text{(MAR setting)}$$

where in both cases the missing mechanism model depends only on ϕ. If we are only interested in having a generative model to generate new $\mathbf{x}$ and not generate new masks $\mathbf{s}$, we can see that in both MAR and MCAR settings we have that $\log p_{\theta,\phi}(\mathbf{x}^o, \mathbf{s}) = \log p_\theta(\mathbf{x}^o) + C$, where C is a constant that does not depend on θ. Therefore, maximizing the observed likelihood is enough to maximize the joint likelihood of the observed data and the mask.

2.1.2 Normalizing Flows

Normalizing flows and autoregressive models that we will present in the next section do not play as central a role in this dissertation as VAEs do. Therefore, we limit ourselves to introducing the basic concepts and notions about these two models and not a detailed treatment as we did for VAEs. Indeed, the main point is to show that different models that make different assumptions about the underlying data generation process share the same behavior when evaluated on data coming from a different distribution than the one on which they were trained.

Let us assume, as before, that we have access to samples $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}^T \in \mathcal{X}^n$ and $\mathcal{X} \subseteq \mathbb{R}^D$ from an underlined data distribution $p_{\text{data}}(\mathbf{x})$. A normalizing flow makes the generative assumption that data $\mathbf{x}$ can be generated from a random variable $\mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^D$ by a deterministic transformation $f_\theta : \mathcal{U} \to \mathcal{X}$ parameterized by θ. The random variable $\mathbf{u}$ can be seen as a "latent" variable, although in contrast to the VAE framework $\mathbf{u}$ has the same shape as the input $\mathbf{x}$ and given $\mathbf{x}$ and f_θ is a deterministic mapping, meaning that the distribution of the latent given the data is a Dirac distribution. It usually follows a simple distribution that is easy to sample, such as a standard Gaussian $p_U(\mathbf{u}) = \mathrm{N}(\mathbf{u}|\mathbf{0}, \mathbf{I}_D)$. The distribution $p_U(\mathbf{u})$ is also called the base distribution. Therefore, the generative process can be written as:

$$\begin{aligned} \mathbf{u} &\sim p_U(\mathbf{u}) \\ \mathbf{x} &= f_\theta(\mathbf{u}), \end{aligned} \tag{2.20}$$

where we first sample $\mathbf{u}$ from the base distribution and then we transform it through $f_\theta(\mathbf{u})$. The only constraint is that the map $f_\theta(\mathbf{x})$ is a *diffeomorphism*, i.e. it has to have an inverse $f_\theta^{-1}(\mathbf{u})$ and both $f_\theta(\mathbf{x})$ and $f_\theta^{-1}(\mathbf{u})$ has to be differentiable. This is the reason why $\mathbf{u}$ must have the same dimension of the input data $\mathbf{x}$. In this case, we can have $\mathbf{u} = f_\theta^{-1}(\mathbf{x})$ and the density of $\mathbf{x}$ can be computed using the change of variables formula:

$$p(\mathbf{x}) = p_U(\mathbf{u}) |\det J_f(\mathbf{u})|^{-1}, \tag{2.21}$$

where $J_f(\mathbf{u})$ is the $D \times D$ Jacobian matrix of the function f_θ with respect the input $\mathbf{u}$. We can express (2.21) only in terms of $\mathbf{x}$ by writing:

$$p(\mathbf{x}) = p_U(f_\theta^{-1}(\mathbf{x})) |\det J_{f^{-1}}(\mathbf{x})|, \tag{2.22}$$

and by taking the logarithm this can be written as:

$$\log p(\mathbf{x}) = \log p_U(f_\theta^{-1}(\mathbf{x})) + \log |\det J_{f^{-1}}(\mathbf{x})|, \tag{2.23}$$

showing that in this case, under the assumption that the determinant of the Jacobian is easy to compute, we have access to the exact likelihood computation. Therefore, to learn the parameters θ of the transformation f_θ, we can compute the maximum likelihood estimate

$$\arg\max_{\theta \in \Theta} \mathbb{E}_{\mathbf{x} \sim p_{\mathcal{D}}}[\log p(\mathbf{x})] = \arg\max_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} [\log p_U(f_\theta^{-1}(\mathbf{x}_i)) + \log |\det J_{f^{-1}}(\mathbf{x}_i)|], \tag{2.24}$$

where $p_{\mathcal{D}}$ is the empirical distribution as before.

A single transformation may not be flexible enough to approximate $p_{\text{data}}(\mathbf{x})$. However, an important property of invertible functions is that they are *composable*. Therefore, we can consider the following generative process:

$$\mathbf{x} = f_\theta(\mathbf{u}) = f_{\theta_L}(\dots f_{\theta_2}(f_{\theta_1}(\mathbf{u}))), \qquad \text{with } \mathbf{u} \sim p_U(\mathbf{u}), \tag{2.25}$$

then we can write the log-likelihood of $\mathbf{x}$ as

$$\log p(\mathbf{x}) = \log p_U(f_\theta^{-1}(\mathbf{x})) + \sum_{i=1}^{L} \log |\det J_{f_{\theta_i}^{-1}}(\mathbf{x})|. \tag{2.26}$$

A major challenge in this framework is the computation of the determinant of Jacobian in (2.24) and (2.26) whose computation is $\mathcal{O}(D^3)$. Therefore, the usual approach is to constrain the model to have a structure that allows us to reduce the cost of this computation from $\mathcal{O}(D^3)$ to $\mathcal{O}(D)$, for example.

2.1.3 Autoregressive Models

Let us assume that we have samples $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}^T \in \mathcal{X}^n$ with $\mathcal{X} \subseteq \mathbb{R}^D$ from an unknown distribution $p_{\text{data}}(\mathbf{x})$ as before. The joint distribution of a single sample $\mathbf{x} = [x_1, \dots, x_D]$ can be written by using the product rule as the product of the following conditionals:

$$\begin{aligned} p(\mathbf{x}) &= p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}), \\ &= \prod_{i=1}^{D} p(x_i|\mathbf{x}_{<i}), \end{aligned} \tag{2.27}$$

where $\mathbf{x}_{<i} = [x_1, \cdots, x_{i-1}]$ represents the set of variables with index less than i. Autoregressive models approximate the true $p(x_i|\mathbf{x}_{<i})$ with a learned parameterized distribution $p_\theta(x_i|\mathbf{x}_{<i}) = \Phi(x_i|f_\theta(\mathbf{x}_{<i}))$, where the function $f_\theta : \mathcal{X} \cup \mathcal{X}^2 \cup \cdot \cup \mathcal{X}^D \to H$ is parameterized by a neural network with parameters θ. $\{\Phi(\cdot|\eta)\}_{\eta \in H}$ is a parametric family of distributions.

The parameters θ are learned also in this case via maximum likelihood:

$$\arg\max_{\theta \in \Theta} \mathbb{E}_{\mathbf{x} \sim p_{\mathcal{D}}}[\log p_\theta(\mathbf{x})] = \arg\max_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i) \tag{2.28}$$

$$= \arg\max_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{D} \log p_\theta(x_j^{(i)}|\mathbf{x}_{<j}^{(i)}), \tag{2.29}$$

where we just used the definition of (2.27), but we considered the parameterized approximation. As in the case of flows, ARMs allow us to use the exact log-likelihood

instead of having to use a lower bound for the optimization of θ. The major drawback of these models is that the sampling is done sequentially by sampling each conditional at the time, resulting in a slow process. In addition to that, for certain modalities, it is not straightforward to define a specific ordering between the random variable of the input.

2.1.4 Score-based Models

In this section, we are going to introduce continuous time score-based diffusion models using the stochastic differential equation formulation presented by Song et al. (2021). In fact, this formulation allows us to see both denoising diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020) and denoising score matching with Langevin dynamics (Song & Ermon, 2019) as a specific discretization of continuous-time score-based models. In the following, we also use diffusion models and score-based diffusion models interchangeably. This type of deep generative model is the one used in Paper 4.

Diffusion models consist of a forward noising process and the corresponding backward denoising process. We denote the forward noising process by $\{\mathbf{x}(t)\}_0^T$, where t is a continuous-time variable. We assume to only have access to initial samples $\mathbf{x}(0) \sim p_0(\mathbf{x})$, that is, we do not know the true data distribution $p_{\text{data}}(\mathbf{x}) = p_0(\mathbf{x})$. In this setting, we expect that the distribution of the forward process at time T, $p_T(\mathbf{x})$, when T is large this is a simple distribution that we can easily sample from and that does not depend on $\mathbf{x}(0)$ anymore. For this reason, the distribution $p_T(\mathbf{x})$ can also be interpreted as the prior distribution. This process can be defined in the Euclidian space as the solution to the following Itô stochastic differential equation:

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}(t), \tag{2.30}$$

where $\mathbf{w}$ is an isotropic Wiener process or Brownian motion, $\mathbf{f}(\mathbf{x}, t) : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^d$ is called the *drift* coefficient and $g(t) : \mathbb{R}^+ \to \mathbb{R}$ is the *diffusion* coefficient. Starting from $\mathbf{x}_0$ the sample will thus be transformed by the drift term and noised by the diffusion term resulting in a sample $\mathbf{x}_T$ distributed accordingly $p_T(\mathbf{x})$ that we hope won't contain any information about the starting distribution at time $t = 0$. There are several choices of the drift and diffusion coefficient resulting in processes that satisfy these conditions, and we will present one of those processes later in this section.

Anderson (1982); Haussmann & Pardoux (1986); Cattiaux et al. (2022) have shown that the reverse or backward denoising process $\{\mathbf{x}(t)\}_T^0$ from $\mathbf{x}_T$ to $\mathbf{x}_0$ is also a diffusion process described by the following SDE:

$$\mathrm{d}\mathbf{x}(t) = \left\{\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))\right\} \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}(t), \tag{2.31}$$

where $\bar{\mathbf{w}}(t)$ is the Wiener process running backward and in this case, the drift coefficient contains the gradient of the log-likelihood with respect to the input $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$,

which is also referred to as the Stein score. This should not be confused with the Fisher score, which is instead the derivative with respect to the parameters. Therefore, if we have access to $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ for every t we can plug it in (2.31) to simulate the reverse SDE and get a new sample $\mathbf{x}_0$ starting from $\mathbf{x}_T \sim p_T(\mathbf{x})$. In practice, we do not have access to $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$, thus we approximate it by using a neural network $s_\theta(\mathbf{x}(t), t)$, referred to as the *score network*. Training of network parameters can be done by minimizing the denoising score matching (DSM) loss (Hyvärinen & Dayan, 2005; Vincent, 2011; Song et al., 2021):

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p_0,\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}[\|\mathbf{s}_\theta(\mathbf{x}(t), t) - \nabla_{x(t)} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))\|^2], \tag{2.32}$$

where $p_t(\mathbf{x}(t)|\mathbf{x}(0))$ is the noising kernel defined by the noising SDE of (2.30). In (2.32), one can notice that we are using $\nabla_{x(t)} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))$ instead of $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ during training, while the goal is to have $s_\theta(\mathbf{x}(t), t) \approx \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$. Vincent (2011) showed that the direction of the "denoising" score $\nabla_{x(t)} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))$ almost perfectly matches the direction of the true score $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ we are interested in learning while making the optimization easier since we have an easy way to compute the denoising score. We will discuss this more in detail in the remark we present in the grey box below. This noising kernel depends on the noising SDE definition. If the drift $f(\mathbf{x}(t), t)$ and the diffusion coefficient $g(t)$ are affine transformations of $\mathbf{x}(0)$, then the conditional marginal distribution or transition kernel is Gaussian with the following form:

$$p(\mathbf{x}(t)|\mathbf{x}(0)) = \mathrm{N}(\mathbf{x}(t)|\alpha(t)\mathbf{x}(0), \sigma^2(t)\mathbf{I}), \tag{2.33}$$

where $\alpha(t)$ and $\sigma(t)$ depends on the choice of the drift $f(\mathbf{x}(t), t)$ and the diffusion $g(t)$ coefficients. If we have an affine transformation of $\mathbf{x}(0)$, getting $\mathbf{x}(t)$ is easy and does not require simulating the forward SDE. In the case we do not have access to a closed-form definition of the conditional marginal distribution, one can rely on a sliced-score matching loss (Song et al., 2020b) which does not require the computation of $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))$ but only the score network $s_\theta(\mathbf{x}(t), t)$. The sliced-score matching loss (Song et al., 2020b) is defined as

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p_0,\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0)),\,\mathbf{v}\sim p_\mathbf{v}} \left[\frac{1}{2}\|s_\theta(\mathbf{x}(t), t)\|^2 + \mathbf{v}^T \nabla_{\mathbf{x}_t} s_\theta(\mathbf{x}(t), t)\mathbf{v}\right], \tag{2.34}$$

where $\mathbf{v} \sim p(\mathbf{v})$ satisfies $\mathbb{E}[\mathbf{v}] = 0$ and $\mathrm{Cov}[\mathbf{v}] = \mathbf{I}$, like a Gaussian or a Rademacher distribution[6], and it can be interpreted as a random projection direction. To obtain a sample from $\mathbf{x}(t) \sim p_t(\mathbf{x}(t)|\mathbf{x}(0))$ we need to simulate the SDE, and $\mathbf{v}^T \nabla_{\mathbf{x}_t} s_\theta(\mathbf{x}(t), t)\mathbf{v}$ can be computed using Jacobian vector product.

[6] A random variable following a Rademacher distribution is a discrete random variable that can take only two values $\{-1, +1\}$ with 50% chance.

Remark: Why $\nabla_{x(t)} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))$ when we want to approximate $\nabla_{x(t)} \log p_t(\mathbf{x}(t))$ in (2.32)? The general goal is to train a score network $s_\theta(\mathbf{x}(t), t)$ that approximates $\nabla_{x(t)} \log p_t(\mathbf{x}(t))$ for all t. However, in (2.32) we define the loss using $\nabla_{x(t)} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))$, which means that our network will approximate a different score. In addition to solving the problem of not being able to compute $p(\mathbf{x}(t))$, we will show by following Vincent (2011), that the minimum is the same for both losses. The loss we would have expected is referred to as the *explicit score matching* loss given by

$$
\begin{aligned}
&\mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(t)\sim p(\mathbf{x}(t))}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x}(t),t) - \nabla_{x(t)} \log p_t(\mathbf{x}(t))\|^2\right] \\
&= \mathbb{E}\left[\frac{1}{2}(\mathbf{s}_\theta(\mathbf{x}(t),t) - \nabla_{x(t)} \log p_t(\mathbf{x}(t)))^T(\mathbf{s}_\theta(\mathbf{x}(t),t) - \nabla_{x(t)} \log p_t(\mathbf{x}(t)))\right] \\
&= \mathbb{E}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x}(t),t)\|^2 - \mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{x(t)} \log p_t(\mathbf{x}(t)) + \frac{1}{2}\|\nabla_{x(t)} \log p_t(\mathbf{x}(t))\|^2\right] \\
&= \mathbb{E}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x}(t),t)\|^2 - \mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{x(t)} \log p_t(\mathbf{x}(t))\right] + C \qquad (2.35)
\end{aligned}
$$

where we consider the $\frac{1}{2}$ to simplify derivation and from the second line we just computed and rewrote the squared norm and then left out the terms that do not depend on θ we want to optimize. Now we will focus only on the term given by $\mathbb{E}_{\mathbf{x}(t)\sim p(\mathbf{x}(t))}[\mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{x(t)} \log p_t(\mathbf{x}(t))]$ by reminding the reader that we can express $p_t(\mathbf{x}(t)) = \int_{\mathcal{X}} p_t(\mathbf{x}(t)|\mathbf{x}(0))p(\mathbf{x}(0))\mathrm{d}\mathbf{x}(0)$ and that the logarithmic derivative trick is $\nabla_x \log p(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p(\mathbf{x})}{p(\mathbf{x})}$, which implies $\nabla_x p(\mathbf{x}) = p(\mathbf{x})\nabla_x \log p(\mathbf{x})$.

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x}(t)\sim p(\mathbf{x}(t))}[\mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{x(t)} \log p(\mathbf{x}(t))] = \\
&= \int p(\mathbf{x}(t))\mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{\mathbf{x}(t)} \log p(\mathbf{x}(t))\ \mathrm{d}\mathbf{x}(t) \\
&= \int \mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{\mathbf{x}(t)} p(\mathbf{x}(t))\ \mathrm{d}\mathbf{x}(t) \\
&= \int \mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{\mathbf{x}(t)} \int p(\mathbf{x}(t)|\mathbf{x}(0))p(\mathbf{x}(0))\ \mathrm{d}\mathbf{x}(0)\mathrm{d}\mathbf{x}(t) \\
&= \int \mathbf{s}_\theta(\mathbf{x}(t),t)^T \int \nabla_{\mathbf{x}(t)} p(\mathbf{x}(t)|\mathbf{x}(0))p(\mathbf{x}(0))\ \mathrm{d}\mathbf{x}(0)\mathrm{d}\mathbf{x}(t) \\
&= \int \mathbf{s}_\theta(\mathbf{x}(t),t)^T \int p(\mathbf{x}(0))p(\mathbf{x}(t)|\mathbf{x}(0))\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))\ \mathrm{d}\mathbf{x}(0)\mathrm{d}\mathbf{x}(t) \\
&= \mathbb{E}_{\mathbf{x}(0)\sim p(\mathbf{x}(0)),\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}[\mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))] \qquad (2.36)
\end{aligned}
$$

By substituting this in the equation above:

$$\mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(t)\sim p(\mathbf{x}(t))}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x}(t),t)\|^2 - \mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{x(t)}\log p(\mathbf{x}(t))\right] + C$$
$$= \mathbb{E}_{t,\mathbf{x}(0)\sim p(\mathbf{x}(0)),\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x}(t),t)\|^2 - \mathbf{s}_\theta(\mathbf{x}(t),t)^T\nabla_{\mathbf{x}(t)}\log p_t(\mathbf{x}(t)|\mathbf{x}(0))\right] + C$$
$$= \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p(\mathbf{x}(0)),\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x}(t),t) - \nabla_{\mathbf{x}(t)}\log p_t(\mathbf{x}(t)|\mathbf{x}(0))\|^2\right]$$

where we considered $C = \mathbb{E}[\frac{1}{2}\|\nabla_{x(t)}\log p_t(\mathbf{x}(t)|\mathbf{x}(0))\|^2]$ in this case to complete the square. This shows that optimizing

$$\mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(t)\sim p(\mathbf{x}(t))}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x}(t),t) - \nabla_{x(t)}\log p_t(\mathbf{x}(t))\|^2\right]$$

is exactly the same as optimizing

$$\mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p(\mathbf{x}(0)),\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x}(t),t) - \nabla_{\mathbf{x}(t)}\log p_t(\mathbf{x}(t)|\mathbf{x}(0))\|^2\right]$$

Example: Ornstein-Uhlenbeck process as a variance-preserving SDE The definition of the noising process presented in (2.30) is pretty general. We did not specify any drift and diffusion coefficient and, therefore, we do not know how the distribution $p_T(\mathbf{x})$ (2.30) will converge at equilibrium. In general, $\mathbf{f}(\mathbf{x},t)$ and $g(t)$ are chosen in a way that the distribution at time T is approximately standard normal $p_T(\mathbf{x}) \approx \mathrm{N}(\mathbf{0},\sigma^2\mathbf{I})$. One possible approach to achieve this is by defining the forward noising process as an Ornstein–Uhlenbeck (OU) process of form[7]:

$$\mathrm{d}\mathbf{x}(t) = -\frac{1}{2}\mathbf{x}(t)\mathrm{d}t + \sigma\mathrm{d}\mathbf{w}(t), \tag{2.37}$$

where the drift is $\mathbf{f}(\mathbf{x},t) = -\frac{1}{2}\mathbf{x}(t)$ and the diffusion coefficient is given by $g(t) = \sigma$ with $\sigma > 0$. As we are going to derive step-by-step in the grey box below, this process, for larger T, will asymptotically follow $p_T(\mathbf{x}) = \mathrm{N}(\mathbf{0},\sigma^2\mathbf{I})$.

[7]The Markov Chain defined by the perturbation kernel used in the DDPM (Ho et al., 2020) formulation converge to the following OU process as $N\to\infty$, i.e. the number of discretization steps goes to infinity: $\mathrm{d}\mathbf{x}(t) = -\frac{1}{2}\beta(t)\mathbf{x}(t)\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\mathbf{w}(t)$, where the drift is $\mathbf{f}(\mathbf{x},t) = -\frac{1}{2}\beta(t)\mathbf{x}(t)$ and the diffusion coefficient is given by $g(t) = \sqrt{\beta(t)}$, where $\beta(t)$ is a time-dependent function that we can choose and will correspond to different time schedules. The associated transition kernel is defined as follows: $p(\mathbf{x}(t)|\mathbf{x}(0)) = \mathrm{N}\left(\mathbf{x}(t)|\mathbf{x}(0)e^{-\frac{1}{2}\int_0^t\beta(s)\mathrm{d}s}, (1-e^{-\int_0^t\beta(s)\mathrm{d}s})\mathbf{I}\right)$ and in this case we see that $\mathbf{x}(1)$ is approximately distributed as $\mathrm{N}(\mathbf{0},\mathbf{I})$. This is also the reason why when using this OU process, we can optimize (2.32) by sampling $t\sim\mathcal{U}([0,1])$.

Derivation of the transition kernel and the distribution $p_T(\mathbf{x})$ for the forward process in (2.37). We want to show that the OU process defined in (2.37) converge to a distribution $p_T(\mathbf{x})$ that is independent on $\mathbf{x}$ but depends only on σ. We start by deriving the solution associated with (2.37):

$$
\begin{aligned}
&\mathrm{d}\mathbf{x}(t) = -\frac{1}{2}\mathbf{x}(t)\mathrm{d}t + \sigma\mathrm{d}\mathbf{w}(t) \\
&\mathrm{d}\mathbf{x}(t) + \frac{1}{2}\mathbf{x}(t)\mathrm{d}t = \sigma\mathrm{d}\mathbf{w}(t) && \text{(reordering the terms)} \\
&e^{\frac{1}{2}t}\mathrm{d}\mathbf{x}(t) + e^{\frac{1}{2}t}\frac{1}{2}\mathbf{x}(t)\mathrm{d}t = e^{\frac{1}{2}t}\sigma\mathrm{d}\mathbf{w}(t) && (e^{\frac{1}{2}t} \text{ integrating factor}) \\
&\mathrm{d}(e^{\frac{1}{2}t}\mathbf{x}(t)) = e^{\frac{1}{2}t}\sigma\mathrm{d}\mathbf{w}(t) && \text{(using Itô's product rule),}
\end{aligned}
$$

where one can read about Itô's product rule in Oksendal (2013). If we now integrate from 0 to T:

$$
\begin{aligned}
&\int_0^T \mathrm{d}(e^{\frac{1}{2}t}\mathbf{x}(t)) = \int_0^T e^{\frac{1}{2}t}\sigma\mathrm{d}\mathbf{w}(t) \\
&[e^{\frac{1}{2}T}\mathbf{x}(T) - e^{\frac{1}{2}0}\mathbf{x}(0)] = \sigma\int_0^T e^{\frac{1}{2}t}\mathrm{d}\mathbf{w}(t) \\
&\mathbf{x}(T) - e^{-\frac{1}{2}T}\mathbf{x}(0) = \sigma\int_0^T e^{-\frac{1}{2}(T-t)}\mathrm{d}\mathbf{w}(t) && \text{(multiply by } e^{-\frac{1}{2}T}) \\
&\mathbf{x}(T) = e^{-\frac{1}{2}T}\mathbf{x}(0) + \sigma\int_0^T e^{-\frac{1}{2}(T-t)}\mathrm{d}\mathbf{w}(t) && \text{(solution associated with (2.37))}
\end{aligned}
$$

The solution is a sum of a deterministic function and an integral of a deterministic function with respect to a Wiener process with normally distributed increments. Therefore, the resulting distribution $p(\mathbf{x}(T)|\mathbf{x}(0))$ is normally distributed. We can now compute the first moments:

$$
\begin{aligned}
\mathbb{E}[\mathbf{x}(T)|\mathbf{x}(0)] &= \mathbb{E}\left[e^{-\frac{1}{2}T}\mathbf{x}(0) + \sigma\int_0^T e^{-\frac{1}{2}(T-t)}\mathrm{d}\mathbf{w}(t)\right] \\
&= \mathbb{E}[e^{-\frac{1}{2}T}\mathbf{x}(0)] = e^{-\frac{1}{2}T}\mathbf{x}(0)
\end{aligned}
$$

where we have used the fact that the expected value of a deterministic function with respect to the Brownian motion is zero. The variance instead can be

computed as:

$$\begin{aligned}\mathrm{Var}(x(T)|\mathbf{x}(0)) &= \mathbb{E}[(\mathbf{x}(T) - \mathbb{E}[\mathbf{x}(T)])^2] = \mathbb{E}\left[\left(\sigma \int_0^T e^{-\frac{1}{2}(T-t)} \mathrm{d}\mathbf{w}(t)\right)^2\right] \\ &= \mathbb{E}\left[\sigma^2 \int_0^T e^{-(T-t)} \mathrm{d}\mathbf{w}(t)\right] \\ &= \sigma^2 \int_0^T e^{-(T-t)} \mathrm{d}t \qquad \text{(using Itô's isometry rule)} \\ &= \sigma^2(1 - e^{-T}),\end{aligned}$$

where one can read about Itô's isometry in Oksendal (2013). Therefore we have $p(\mathbf{x}(t)|\mathbf{x}(0)) = \mathrm{N}(\mathbf{x}(t)|e^{-\frac{1}{2}t}\mathbf{x}(0), \sigma^2(1 - e^{-t}))$. Now we can study the behavior of this distribution when T tends to infinity:

$$\begin{aligned}\lim_{T\to\infty} \mathbb{E}[\mathbf{x}(T)] &= 0 \\ \lim_{T\to\infty} \mathrm{Var}(\mathbf{x}(T)) &= \sigma^2.\end{aligned}$$

Therefore, the process defined in (2.37) is characterized by $\mathbf{x}(T)$ approximately following $\mathrm{N}(\mathbf{0}, \sigma^2\mathbf{I})$ when T goes to infinity.

The backward or denoising process in this case then becomes

$$\mathrm{d}\mathbf{x}(t) = \left\{\frac{1}{2}\mathbf{x}(t) - \sigma^2 \nabla_{x(t)} \log p_t(\mathbf{x}(t))\right\} \mathrm{d}t + \sigma \mathrm{d}\bar{\mathbf{w}}(t). \tag{2.38}$$

Since both $f(\mathbf{x}(t), t)$ and $g(t)$ in (2.37) are affine transformation of $\mathbf{x}(t)$[8], then the transition kernel is Gaussian and as we have seen in the grey box and it is given by

$$p(\mathbf{x}(t)|\mathbf{x}(0)) = \mathrm{N}(\mathbf{x}(t)|e^{-\frac{1}{2}t}\mathbf{x}(0), \sigma^2(1 - e^{-t})\mathbf{I}). \tag{2.39}$$

Since the transition kernel is Gaussian, we compute $\nabla_{x(t)} \log p(\mathbf{x}(t)|\mathbf{x}(0))$ in closed form. To make it more general, we are going to derive everything by considering again the general Gaussian transition kernel case defined as $p(\mathbf{x}(t)|\mathbf{x}(0)) = \mathrm{N}(\mathbf{x}(t)|\alpha(t)\mathbf{x}(0), \sigma^2(t)\mathbf{I})$.

[8] In the considered case we have that the diffusion coefficient $g(t)$ is independent of $\mathbf{x}(t)$, but depends only on t.

The closed form solution for $\nabla_{x(t)} \log p(\mathbf{x}(t)|\mathbf{x}(0))$ is given by:

$$\begin{aligned}
\frac{\mathrm{d} \log p_t(\mathbf{x}(t)|\mathbf{x}(0))}{\mathrm{d}\mathbf{x}(t)} &= \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}(t)} \log \left\{ \frac{1}{(2\pi)^n/2|\sigma^2(t)\mathbf{I}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}(t) - \alpha(t)\mathbf{x}(0))^T(\sigma^2(t)\mathbf{I})^{-1}(\mathbf{x}(t) - \alpha(t)\mathbf{x}(0))\right)\right\} \\
&= -\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}(t)}\left[(\mathbf{x}(t) - \alpha(t)\mathbf{x}(0))^T(\sigma^2(t)\mathbf{I})^{-1}(\mathbf{x}(t) - \alpha(t)\mathbf{x}(0))\right] \\
&= -\frac{1}{2}\left[2(\sigma^2(t)\mathbf{I})^{-1}(\mathbf{x}(t) - \alpha(t)\mathbf{x}(0))\right] \\
&= (\sigma^2(t)\mathbf{I})^{-1}(\alpha(t)\mathbf{x}(0) - \mathbf{x}(t))
\end{aligned}$$

We can substitute this inside the DSM loss of (2.32) to obtain:

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p_0,\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}[\|\mathbf{s}_\theta(\mathbf{x}(t),t) - (\sigma^2(t)\mathbf{I})^{-1}(\alpha(t)\mathbf{x}(0) - \mathbf{x}(t))\|^2],$$

and by using the fact that we can express $\mathbf{x}(t)$ using the reparameterization trick presented in Section 2.1.1 as $\mathbf{x}(t) = \alpha(t)\mathbf{x}(0) + \sigma\epsilon$ where $\epsilon \sim \mathrm{N}(\mathbf{0}, \mathbf{I})$ then we get a simplify DSM loss:

$$\begin{aligned}
\mathcal{L}(\theta) &= \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p_0,\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}[\|\mathbf{s}_\theta(\mathbf{x}(t),t) - (\sigma^2(t)\mathbf{I})^{-1}(\alpha(t)\mathbf{x}(0) - \alpha(t)\mathbf{x}(0) - \sigma(t)\epsilon)\|^2] \\
&= \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p_0,\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}[\|\mathbf{s}_\theta(\mathbf{x}(t),t) + (\sigma(t)^2\mathbf{I})^{-1}(\sigma(t)\epsilon)\|^2] \\
&= \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p_0,\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}[\|\mathbf{s}_\theta(\mathbf{x}(t),t)\cdot\sigma(t) + \epsilon\|^2].
\end{aligned}$$

We can therefore consider the following parameterization $\epsilon(\mathbf{x}(t),t) = -\mathbf{s}_\theta(\mathbf{x}(t),t)\cdot\sigma(t)$ and train the model by using the $\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p_0,\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}[\|\epsilon(\mathbf{x}(t),t) - \epsilon\|^2]$ as loss. We can notice that even if we use this last parameterization where we learn the noise instead of the score, we can get the approximate score function by computing $\mathbf{s}_\theta(\mathbf{x}(t),t) = -\epsilon(\mathbf{x}(t),t)/\sigma(t)$.

The one presented is an example of a Variance Preserving (VP) SDE, i.e.$\mathbf{x}(t)$ has a finite variance for all t if the variance of $\mathbf{x}(0)$ is also finite, which can be easily achieved by standardizing the training data.

Sampling from the prior. Once we trained a score network $s_\theta(\mathbf{x}(t),t)$ we can use it in the backward or denoising process defined by (2.31) to sample from $p_0(\mathbf{x})$. This can be done by employing a general-purpose numerical solver for SDE like Euler-Maruyama and stochastic Runge-Kutta methods. Having to simulate the SDE makes sampling slower compared to other deep generative models, such as VAEs. However, speeding up sampling in diffusion models is an active research area. One promising direction is to exploit the fact that there exists an ordinary differential equation (ODE) that admits the same marginals as the denoising reverse SDE shown in (2.31). This ODE is referred to as the *probability flow* (Song et al., 2021), and it's given by:

$$\mathrm{d}\mathbf{x}(t) = \left\{\mathbf{f}(\mathbf{x},t) - g(t)^2\nabla_{\mathbf{x}(t)}\log p_t(\mathbf{x}(t))\right\}\mathrm{d}t. \tag{2.40}$$

New samplers for diffusion models discretize (2.40) instead of the denoising SDE to speed up computations (Lu et al., 2023; Zhang & Chen, 2023). In addition to that, since $s_\theta(\mathbf{x}(t), t) \approx \nabla_{x(t)} \log p_t(\mathbf{x}(t))$, Song et al. (2021) proposed to use the trained score to improve the obtained solution by performing few steps of Markov Chain Monte Carlo (MCMC) method, such as Langevin MCMC (Parisi, 1981) or HMC (Neal, 2011). Therefore, at each step, the solver provides a first estimate of the next sample (*predictor*), and then the score-based MCMC method corrects the marginal distribution $p_t(\mathbf{x}(t))$ we are sampling from (*corrector*). This is the reason why this is referred to as predictor-corrector sampling. This helps prevent approximation errors from accumulating and leading to low-quality samples.

Conditional sampling. While sampling from $p_0(\mathbf{x})$ can be useful for certain tasks, like image generation, in several applications we are interested in sampling from $p_0(\mathbf{x}|y)$ where y is another random variable. Possible examples include y being a specific class label, textual information, or even a noised or unnoised part of the input as is the case for image inpainting. In this setting, we are thus interested in generating samples by simulating the following conditional denoising reverse SDE (Song et al., 2021; Chung et al., 2023; Didi et al., 2023):

$$\mathrm{d}\mathbf{x}(t) = \left\{\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)|y)\right\} \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}(t), \tag{2.41}$$

Here we are going to present two possible ways to tackle this problem. The first approach is to directly learn a conditional score $s_\theta(\mathbf{x}(t)|y) \approx \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)|y)$ (Song et al., 2021; Ho et al., 2020) by minimizing the following DSM loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}([0,T]),\mathbf{x}(0)\sim p_0,\mathbf{x}(t)\sim p(\mathbf{x}(t)|\mathbf{x}(0))}[\|\mathbf{s}_\theta(t, \mathbf{x}(t), y) - \nabla_{x(t)} \log p(\mathbf{x}(t)|x(0))\|^2], \tag{2.42}$$

where the conditioning in $\mathbf{s}_\theta(t, \mathbf{x}(t), y)$ is done in practice by feeding y to the score network along with the input as separate channels for example. Once we have a trained $\mathbf{s}_\theta(t, \mathbf{x}(t), y)$, we can use it to simulate the reverse SDE as we have explained before. The only drawback of this approach is that one needs to have a dataset of pairs $(\mathbf{x}(0), y) \sim p_{0\times y}$.

A different approach, instead, consists in training a prior score $s_\theta(\mathbf{x}, t)$[9] and then conditioning on y "a posteriori" during sampling. Indeed, using Bayes' rule, we can write the term we are interested in computing as follows $p_t(\mathbf{x}(t)|y) = \frac{p_t(y|\mathbf{x}(t))p_t(\mathbf{x}(t))}{p(y)}$.

[9]Note that $s_\theta(\mathbf{x}, t)$ is the usual score network we have introduce at the beginning of the section that can be trained using (2.32). In this setting, we refer to it as *prior score* because it is trained before being conditioned on y. This is mostly to differentiate it from the amortized conditional score which uses y also at training time.

If we now compute $\log p_t(\mathbf{x}(t)|y)$ and take the gradient we get

$$\begin{aligned}\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)|y) &= \nabla_{\mathbf{x}(t)} \log \frac{p_t(y|\mathbf{x}(t))p_t(\mathbf{x}(t))}{p(y)} \\ &= \nabla_{\mathbf{x}(t)} \log p_t(y|\mathbf{x}(t)) + \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)) - \underbrace{\nabla_{x(t)} \log p(y)}_{=0} \\ &= \underbrace{\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))}_{\text{prior score}} + \underbrace{\nabla_{x(t)} \log p_t(y|\mathbf{x}(t))}_{\text{guidance}}, \end{aligned} \tag{2.43}$$

where $\nabla_{\mathbf{x}(t)} \log p(\mathbf{x}(t))$ can be approximated by a score network trained only on the data $\mathbf{x}$ without having access to the conditioned observation y at training time. This is exactly the same procedure that we have introduced in the previous sections. The conditioning plays a role through $\nabla_{\mathbf{x}(t)} \log p(y|\mathbf{x}(t))$ which is referred to as *guidance* (Ho et al., 2022; Chung et al., 2022, 2023; Meng & Kabashima, 2023; Wu et al., 2022; Song et al., 2022, 2023). We can see that $p(y|\mathbf{x}(t))$ is exactly the distribution that we aim to learn when solving supervised learning tasks, such as regression and classification. Therefore, we can train a classification or regression model on the conditioning y in which we are interested and compute the gradient with respect to the input. This is the reason why this term is sometimes also called *classifier guidance*. The main difference from the usual supervised task is that in this setting the model should also be trained on a noisy version of the input $\mathbf{x}$ in order to get a meaningful signal to guide (2.43), especially when the noise level is high.

For certain tasks, such as when we are interested in solving inverse problems, we do not have to train a separate classifier. In fact, we can rewrite $p(y|\mathbf{x}(t))$ as

$$p(y|\mathbf{x}(t)) = \int p(y, \mathbf{x}(0)|\mathbf{x}(t))\, \mathrm{d}\mathbf{x}(0) = \int p(y|\mathbf{x}(0))p(\mathbf{x}(0)|\mathbf{x}(t))\, \mathrm{d}\mathbf{x}(0), \tag{2.44}$$

where we still do not have access to a closed-form solution for the distribution $p(\mathbf{x}(0)|\mathbf{x}(t))$, which given a noised $\mathbf{x}(t)$ gives us a distribution over the possible noiseless $\mathbf{x}(0)$. However, it is possible to approximate by moment matching (Chung et al., 2023; Finzi et al., 2023). The mean of $p(\mathbf{x}(0)|\mathbf{x}(t))$ can be obtained by using Tweedie's formula (Efron, 2011) as:

$$\hat{x}(x(t)) = \mathbb{E}[\mathbf{x}(0)|\mathbf{x}(t)] = \frac{\mathbf{x}(t) + \sigma^2(t)\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))}{\alpha(t)}. \tag{2.45}$$

The covariance function can also be approximated by using Tweedie's formula, as shown by Finzi et al. (2023):

$$\mathrm{Cov}[\mathbf{x}(0) \mid \mathbf{x}(t)] = \left[\frac{\sigma(t)^2}{\alpha(t)^2}(\mathrm{I} + \sigma(t)^2\nabla^2 \log p(\mathbf{x}(t))\right] = \hat{\Sigma}(\mathbf{x}(t)), \tag{2.46}$$

which requires the computation of the Hessian $\sigma(t)^2\nabla^2 \log p(\mathbf{x}(t))$. This motivates the approximation of $p(\mathbf{x}(0)|\mathbf{x}(t)) = \mathrm{N}(\mathbf{x}(0)|\hat{x}(x(t)), \hat{\Sigma}(\mathbf{x}(t)))$ as a Gaussian distribution

whose first two moments are the ones we just presented. We can then substitute this approximation in (2.44) and obtain the following:

$$p(y|\mathbf{x}(t)) = \int p(y|\mathbf{x}(0))p(\mathbf{x}(0)|\mathbf{x}(t))\ \mathrm{d}\mathbf{x}(0) \tag{2.47}$$

$$\approx \int p(y|\mathbf{x}(0))\mathrm{N}(x(0)|\hat{x}(x(t)), \hat{\Sigma}(\mathbf{x}(t)))\ \mathrm{d}\mathbf{x}(0). \tag{2.48}$$

Let us assume, for example, that the likelihood term is $p(y|\mathbf{x}(0))$ a Gaussian distribution $\mathrm{N}(y|\mathcal{A}(\mathbf{x}(0)), \sigma_y^2 I)$ whose mean depends on an observation model $\mathcal{A}(\mathbf{x}(0))$. For example, if we are interested in performing image inpainting or forecasting, we can interpret $A : \mathbf{x} \to \mathbf{x}[I]$ as an operator that takes an input and just returns the input at a set of indexes over space and/or time depending on the setting. Then, in the particular case where the observation model is a linear transformation of the form $\mathcal{A}(\mathbf{x}(0)) = A\mathbf{x}(0)$, we can rewrite $p(y|\mathbf{x}(t))$ as a Gaussian distribution:

$$\begin{aligned} p(y|\mathbf{x}(t)) &= \int p(y|\mathbf{x}(0))p(\mathbf{x}(0)|\mathbf{x}(t))\ \mathrm{d}\mathbf{x}(0) \\ &\approx \int p(y|\mathbf{x}(0))\mathrm{N}(x(0)|\hat{x}(\mathbf{x}(t)), \hat{\Sigma}(\mathbf{x}(t)))\ \mathrm{d}\mathbf{x}(0). \\ &= \mathrm{N}\left(y|\mathrm{A}\hat{x}(\mathbf{x}(t)), A\hat{\Sigma}(\mathbf{x}(t))A^T + \sigma_y^2\mathbf{I}\right), \end{aligned} \tag{2.49}$$

One of the main challenges in computing (2.49) is that we need to compute the Hessian in $\hat{\Sigma}(\mathbf{x}(t))$. For this reason, different papers propose different approaches to approximate it simply by considering $\mathrm{Cov}[\mathbf{x}(0)|\mathbf{x}(t)] \approx r^2(t)\mathbf{I}$ (Rozet & Louppe, 2023a; Finzi et al., 2023; Song et al., 2022; Pokle et al., 2023), where $r(t)$ is a monotonically increasing function. Therefore, we can rewrite $p(y|\mathbf{x}(t)) \approx \mathrm{N}\left(y|\mathrm{A}\hat{x}(\mathbf{x}(t)), (\sigma_y^2 + r^2(t))\mathbf{I}\right)$.

Additional details on Tweedie's formula. We will present Tweedie's formula in more detail. The formula was first introduced by Robbins (1992). Such a formula appears in several places in the diffusion models literature, as it can also be used to perform a last denoising step at sampling time. Let us assume that we have access to samples $\mathbf{z}$ from a certain Gaussian distribution $p(\mathbf{z}|\mu_\mathbf{z}) = \mathrm{N}(\mathbf{z}|\mu_\mathbf{z}, \Sigma)$, where Σ is observed, but $\mu_\mathbf{z}$ is a random variable from a distribution $p(\mu_\mathbf{z})$. The Tweedie's formula computes the posterior expectation of μ given $\mathbf{z}$ by just using the samples as:

$$\mathbb{E}[\mu_\mathbf{z}|\mathbf{z}] = \mathbf{z} + \Sigma\nabla_\mathbf{z} \log p(\mathbf{z}), \tag{2.50}$$

where $p(\mathbf{z})$ is the marginal distribution over $\mathbf{z}$.

In our setting, we assume to have sample $\mathbf{x}(0) \sim p(\mathbf{x}(0))$ is the data distribution in our case, we also know that the transition kernel can be defined as $p(\mathbf{x}(t)|\mathbf{x}(0)) = \mathrm{N}(\mathbf{x}(t)|\alpha(t)\mathbf{x}(0), \sigma^2(t)\mathrm{I})$, where the covariance is known and the

mean is a random variable depending on $\mathbf{x}(0)$ but we have access to samples $\mathbf{x}(t)$ from it. Therefore, by using Tweedie's formula we can write that the best estimate of the mean that computed $\mathbf{x}(t)$ is given by:

$$\mathbb{E}[\mu_{\mathbf{x}(t)}|\mathbf{x}(t)] = \mathbf{x}(t) + \sigma^2(t)\nabla_{\mathbf{x}(t)} \log p(\mathbf{x}(t)). \tag{2.51}$$

Since we know that the true mean is given by $\alpha(t)\mathbf{x}(0)$, then we can use it to get the following:

$$\alpha(t)\mathbf{x}(0) = \mathbf{x}(t) + \sigma^2(t)\nabla_{\mathbf{x}(t)} \log p(\mathbf{x}(t)) \tag{2.52}$$

$$\rightarrow \mathbf{x}(0) = \frac{\mathbf{x}(t) + \sigma^2(t)\nabla_{\mathbf{x}(t)} \log p(\mathbf{x}(t))}{\alpha(t)}. \tag{2.53}$$

2.1.5 On Likelihood and Typicality for out-of-distribution detection

In the previous section, we have presented four different types of deep generative models to approximate certain true data distribution $p_{\text{data}}(\mathbf{x})$ and generate samples from it. VAEs, flows, and ARMs are all trained by maximizing either the exact log-likelihood or a lower bound. Roughly speaking, we can think of the likelihood as a measure of surprise, where the higher the likelihood the less surprised the model is. Therefore, training the parameters of these models by optimizing the likelihood corresponds to making these models not surprised by the observed data, or equivalently being able to explain the training data as well as possible. Under this view of the likelihood as a measure of surprise, one can understand why the entropy, which is the expected value of the negative log-likelihood, is often referred to as the expected surprise. Consequently, if a model assigns a low likelihood to a certain datapoint, then it means that the model is highly surprised by it, hinting that it can be a datapoint that was not really similar to the ones in the training set. This motivated Bishop (1994) to propose the use of a threshold on the likelihood for anomaly or out-of-distribution detection, i.e. the task of identifying inputs that are not coming from the same distribution as the observed examples used to train the model. Given the recent successful results obtained by DGMs in modeling the distribution $p(\mathbf{x})$, one would think that using their likelihood for such a task is a promising way to follow.

However, Nalisnick et al. (2019) showed that deep generative models tend to assign a higher likelihood to examples on which they were not trained. In a follow-up work, they hypothesized that this behavior is due to a mismatch between the region of the model support where the model assigns high likelihood and the one from which the model samples. The area from which the model draws samples is commonly known as *typical set*. While we will discuss in detail the problem of out-of-distribution detection using deep generative models in Paper 1, we will briefly present the concept of typicality and the typical set in the following, as we feel that it was an underap-

preciated concept in the machine learning literature before Nalisnick et al. (2019)'s work. Indeed, issues related to typicality occur not only in trained deep generative models but also in standard distributions that we often use in probabilistic machine learning, such as a high-dimensional multivariate Gaussian distribution. The central take-home message is that probabilities in high dimensions are highly unintuitive, and one of the common mistakes one can make in this setting is to expect the mode of a distribution to look similar to the random samples generated by the distribution itself. To showcase how unintuitively high-dimensional distributions behave, let us consider, for example, a distribution over 28×28 grayscale images defined as factorized Bernoulli, where each pixel has a probability of 0.8 to be black. The most likely image is a completely black image, but it is highly improbable to get it when sampling from it. In the same way, if we consider a D-dimensional multivariate Gaussian $\mathrm{N}(0, \mathbf{I}_D)$, the mean has the highest likelihood, but most of the samples are concentrated in a shell of radius $\sqrt{D}$ from the mean, meaning that getting samples close to the mode is really unlikely. This is also known as the Gaussian annulus theorem.

Formally, typicality[10] for a specific distribution $p(\mathbf{x})$ with support on $\mathcal{X}$ is defined in terms of its (ϵ, N)-typical set denoted with $\mathcal{A}_\epsilon^N[p(\mathbf{x})] \subseteq \mathcal{X}^N$ which contains all N-length sequences that satisfy

$$\mathbb{H}[p(\mathbf{x})] - \epsilon \leq -\frac{1}{N} \log p(\mathbf{x}_1, \ldots, \mathbf{x}_N) \leq \mathbb{H}[p(\mathbf{x})] + \epsilon, \tag{2.54}$$

[10] Usually the concept of weak typicality and typical set are defined in an information-theoretic context. It was introduced by Claude Shannon while he wanted to efficiently encode a stream of symbols of a specific length from a finite alphabet, where each symbol is i.i.d. sampled from a certain distribution p. He noticed that there is a set of sequences that appears disproportionally compared to other sequences, and this set of sequences is referred to as *typical set*. Now we present a simple experiment to reproduce this behavior. This was presented in the following blog post by Bob Carpenter. The experiment is related to the grayscale image distribution we presented before. Consider an alphabet with only two symbols $\{0, 1\}$ sampled from $p(x) = \mathrm{Ber}(x|0.8)$. Then we can consider the sequences that contain 100 symbols. We can see that the most likely sequence is the one with all successes with a probability of $0.8^{100} \approx 2 \cdot 10^{-10}$. Instead, a sequence with only 80 successes has the probability of $0.8^{100} 0.2^{20} \approx 2 \cdot 10^{-22}$. Therefore, getting a sequence with all the successes is 10^{12} more probable. However, if we count how many sequences there are with 80 successes out of 100, these are $\binom{100}{80} \approx 10^{20}$, and therefore their overall probability mass is $10^{20} \cdot 2 \cdot 10^{-22}$ which is now bigger than $2 \cdot 10^{-10}$.

or if the $\log p(\mathbf{x}_1, \dots, \mathbf{x}_N)$ factorizes[11] we can write

$$\mathbb{H}[p(\mathbf{x})] - \epsilon \leq -\frac{1}{N}\sum_{i=1}^{N} \log p(\mathbf{x}_i) \leq \mathbb{H}[p(\mathbf{x})] + \epsilon, \tag{2.55}$$

where $\epsilon \in \mathbb{R}^+$ is a small positive constant and $\mathbb{H}[p(\mathbf{x})]$ is the differential entropy if $\mathbf{x}$ is continuous or the discrete entropy if $\mathbf{x}$ is discrete. The term $-\frac{1}{N}\sum_{i=1}^{N} \log p(\mathbf{x}_i)$, instead, can be seen as *empirical entropy*. The definition used in (2.54) is also called *weak typicality*. As the number of samples N increases then the probability of a sequence of samples to be typical is close to 1, meaning that the typical set covers almost the whole support of the distribution. Indeed, we have that $P(\mathcal{A}_\epsilon^N[p(\mathbf{x})]) > 1 - \epsilon$ [12]. In the case of the multivariate Gaussian example, it means that the shell of radius $\sqrt{D}$ represents the distribution and, surprisingly, the most likely sample is not contained.

A typicality test for out-of-distribution detection. Nalisnick et al. (2019) used the definition of typicality in (2.55) to define a statistical test for detecting out-of-distribution samples. Indeed. the concepts of typicality and typical set might explain why deep generative models assign a higher likelihood to out-of-distribution data, but are not able to sample them. Therefore, if an example is not in the typicality set, then it is likely to be out-of-distribution. Assuming that we have some example $\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_N$ we want to classify as either in-distribution or out-of-distribution, then the typicality test is given by:

$$\mathcal{T}_{\text{Typicality}} = \left| \frac{1}{N}\sum_{i=1}^{N} - \log p_\theta(\tilde{\mathbf{x}}_i) - \mathbb{H}[p_\theta(\mathbf{x})] \right|. \tag{2.56}$$

The differential entropy term:

$$\mathbb{H}[p_\theta(\mathbf{x})] = \int_{\mathcal{X}} p_\theta(\mathbf{x})[-\log p_\theta(\mathbf{x})]\mathrm{d}\mathbf{x} \approx \frac{1}{S}\sum_{i=1}^{S} - \log p_\theta(\hat{\mathbf{x}}_i), \tag{2.57}$$

can be computed by generating S samples using the deep generative model $p_\theta(\mathbf{x})$. However, Nalisnick et al. (2019) found that computing it using the training set leads

[11] Here we are assuming that N is the number of samples we get from a certain distribution $p(\mathbf{x})$. In case the high-dimensional distribution $p(\mathbf{x})$ factorizes across dimensions, i.e. $p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i)$, with $p(x_i) = p(x_j)$ for all i and j meaning that all factors are the same, then we can interpret typicality also in terms of dimensionality and not only in terms of number of samples. In fact, a sample $\mathbf{x} \in \mathcal{X}^D$ can be interpreted as a sequence of D samples from $p(x)$. For example, if we consider $\mathrm{N}(0, \mathbf{I}_D)$, then we find that each dimension can be sampled from $\mathrm{N}(x|0, 1)$, and therefore we can interpret each sample as a sequence of D samples.

[12] If we indicate with $\bar{\mathbf{x}}^N$ a sequence with N elements, then we can interpret $P(\mathcal{A}_\epsilon^N[p(\mathbf{x})]) > 1 - \epsilon$ as $\sum_{\bar{\mathbf{x}}^N \in \mathcal{A}_\epsilon^N[p(\mathbf{x})]} p(\bar{\mathbf{x}}^N) > 1 - \epsilon$. Alternatively, this means that a sequence of length N sampled from $p(\mathbf{x})$ has the probability $1 - \epsilon$ to be in the typical set.

to better results in terms of out-of-distribution detection.

In Paper 1, we will show that we can arrive at the same typicality test by starting from *two-sample test*, where a popular way to build statistics for this type of test is by using a measure of distance between two probabilities distributions P and Q defined over the same space $\mathcal{X}$. Choosing *maximum-mean-discrepancy* (MMD) (Gretton et al., 2012) as a distance measure between the two distributions with a kernel whose feature map is defined as $\Phi : \mathcal{X} \to \mathcal{H}$, results in the following:

$$\text{MMD}_\Phi(P, D) = \|\mathbb{E}_{X\sim P}[\Phi(X)] - \mathbb{E}_{X\sim Q}[\Phi(Y)]\|_\mathcal{H}. \tag{2.58}$$

In our context, we are interested in comparing the training set $\mathbf{x}_1, \ldots, \mathbf{x}_m$ and the examples we want to classify $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$. Then, our distributions are defined as the empirical distribution of these two sets. Therefore, we can write the MMD as:

$$\text{MMD}_\Phi\left(\frac{1}{m}\sum_{i=1}^m \delta(\mathbf{x}_i), \frac{1}{n}\sum_{i=1}^n \delta(\tilde{\mathbf{x}}_i)\right) = \left\|\frac{1}{m}\sum_{i=1}^m \Phi(\mathbf{x}_i) - \frac{1}{n}\sum_{i=1}^n \Phi(\tilde{\mathbf{x}}_i)\right\|.$$

If we now consider the following kernel $\Phi_{\text{Typiality}}(\mathbf{x}) = \log p_\theta(\mathbf{x})$, if we substitute this in the above equation, we get:

$$\text{MMD}_{\Phi_{\text{Typiality}}}\left(\frac{1}{m}\sum_{i=1}^m \delta(\mathbf{x}_i), \frac{1}{n}\sum_{i=1}^n \delta(\tilde{\mathbf{x}}_i)\right) = \left\|\frac{1}{m}\sum_{i=1}^m \log p_\theta(\mathbf{x}_i) - \frac{1}{n}\sum_{i=1}^n \log p_\theta(\tilde{\mathbf{x}}_i)\right\| \tag{2.59}$$

which is measuring the same quantity as the typicality test $\mathcal{T}_{\text{Typicality}}$ defined in (2.56).

2.2 Laplace Approximation

In the previous section, we have presented various classes of deep generative models as flexible ways of combining ideas from probabilistic modeling and deep learning to approximate the true data distribution $p_{\text{data}}(\mathbf{x})$ by using only samples from it. In this section, instead, we change our focus from unsupervised learning to supervised learning, where the task is to learn the relationship between inputs $\mathbf{x}$ and targets $\mathbf{y}$. We first present the way most practitioners use to train neural networks, and then we will move to explain the probabilistic approach, which is the main focus of this section. We introduce the two approaches to make the connection between them more clear for the reader.

Empirical Risk Minimization. Let us assume that we have access to a training dataset $\mathcal{D} = (\mathbf{x}_i, \mathbf{y}_i)_{i=1}^N$, consisting of pairs $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^D$ and $\mathcal{Y}$ can be continuous in the case of a regression task, i.e. $\mathcal{Y} = \mathbb{R}$, or discrete in the case of

having a classification task, i.e. $\mathcal{Y} = \{0, 1\}$ for binary classification or $\mathcal{Y} = \{0, \dots, k\}$ for multiclass scenarios. In this setting, the goal is to learn a function $f(\cdot) : \mathcal{X} \to \mathcal{Y}$[13] that infers the relationship between $\mathbf{x}$ and $\mathbf{y}$. In principle, to make the learning of this function f feasible, we have to restrict ourselves to a "small enough" class of functions $\mathcal{F}_n$. Once $\mathcal{F}_n$ is chosen, we can learn the best function $\hat{f}$ by performing *empirical risk minimization* (ERM) (Vapnik, 1991):

$$\hat{f} \in \arg\min_{f \in \mathcal{F}_n} \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}_i), \mathbf{y}_i), \tag{2.60}$$

where $L : \mathcal{Y}^2 \to \mathbb{R}^+$ is a loss function that depends on the task we are trying to solve. The loss function measures how close $f(\mathbf{x}_i)$ is to $\mathbf{y}_i$ and it usually satisfies $L(f(\mathbf{x}_i), \mathbf{y}_i) = 0$ if and only if $f(\mathbf{x}_i) = \mathbf{y}_i$.

To avoid choosing a specific class of function $\mathcal{F}_n$, one can parameterize the function and add a penalization term on its parameters such that it favors simple and smooth functions. We denote the parametric model as $f(\cdot, \theta) : \mathcal{X} \to \mathcal{Y}$, where $\theta \in \Theta$ are the parameters, for example, the weights of a neural network, and the empirical risk minimization of (2.60) becomes:

$$\theta^* \in \arg\min_{\theta} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}_i), \mathbf{y}_i) + \lambda R(\theta)}_{\mathcal{L}_{\text{ERM}}(\mathcal{D}, \theta)}, \tag{2.61}$$

where R is the regularization term and λ is the regularization strength. In the following, we will denote the combination of the loss function and a regularization term as $\mathcal{L}_{\text{ERM}}(\mathcal{D}, \theta)$. In a regression task, a typical choice for the loss L is the squared loss, while in the classification case, the loss is usually the cross-entropy loss. Instead, for the regularization term, a typical choice is a squared norm, i.e. $R = \|\theta\|_2^2$, which avoids the value of the weights from blow-up. This is the usual setup used to train a neural network. We will come back to (2.61) later when we show that this can also be interpreted from a probabilistic perspective.

A probabilistic approach. When approaching a supervised learning task in a probabilistic way, a common assumption is that the dataset is generated by the following generative process

$$(\mathbf{x}_i, \mathbf{y}_i) \sim p(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}), \tag{2.62}$$

[13] We are aware that this definition is not completely exact. Indeed, in general, the output set of the function $f(\cdot)$ is not the same as the label set. For example, for classification the output of $f(\cdot)$ is continuous while the label set is discrete. We could have defined the output set as $\hat{\mathcal{Y}}$, but we felt introducing too many notations.

which can be described as follows: first sample an input $\mathbf{x}_i$ from the distribution $p(\mathbf{x})$ and then the corresponding output $\mathbf{y}_i$ is sampled from the conditional distribution $p(\mathbf{y}|\mathbf{x})$. In this setting, as we did in the previous section, we use a parametric distribution $p(\mathbf{y}|\mathbf{x},\theta) = \Phi(\mathbf{y}|f(\mathbf{x},\theta))$[14] to approximate the true $p(\mathbf{y}|\mathbf{x})$, where $\{\Phi(\cdot|\eta)\}_{\eta\in\mathcal{H}}$ is a parametric family of distributions and $f(\mathbf{x},\theta)$ is a neural network. The distribution $p(\mathbf{y}|\mathbf{x},\theta)$ is also referred to as the *likelihood*. In the Bayesian approach, to represent the uncertainty we have about the parameters θ in the model before observing any data, we define a *prior* distribution $p(\theta)$. This distribution can encode information about the problem being studied or incorporate some regularization properties to control the model complexity. Since we are assuming the datapoints $(\mathbf{x}_i, \mathbf{y}_i)$ to be independently and identically distributed, then the likelihood of the dataset $\mathcal{D}$ under the parameters θ factorizes according to $p(\mathcal{D}|\theta) = \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x}_i,\theta)$. Thus, the goal is to use the dataset $\mathcal{D}$ to learn the parameters θ to predict the output $\mathbf{y}^*$ of a new unseen input $\mathbf{x}^*$. Importantly, we are also interested in computing the probability $p(\mathbf{y}^*|\mathbf{x}^*,\mathcal{D})$, i.e. the probability of the outputs $\mathbf{y}^*$ given the input $\mathbf{x}^*$, called *posterior predictive distribution*, which can be written as:

$$\begin{aligned} p(\mathbf{y}^*|\mathbf{x}^*,\mathcal{D}) &= \int p(\mathbf{y}^*,\theta|\mathbf{x}^*,\mathcal{D})\mathrm{d}\theta = \int p(\mathbf{y}^*|\mathbf{x}^*,\theta,\mathcal{D})p(\theta|\mathbf{x}^*,\mathcal{D})\mathrm{d}\theta \\ &= \int p(\mathbf{y}^*|\mathbf{x}^*,\theta)p(\theta|\mathcal{D})\mathrm{d}\theta, \end{aligned} \tag{2.63}$$

where $p(\theta|\mathcal{D})$ is the true posterior distribution of the weights given the data and this is the distribution we are interested in approximating. Moreover, we can write the posterior density using Bayes' rule, resulting in

$$p(\theta|\mathcal{D}) = \frac{\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}\overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}, \tag{2.64}$$

where $p(\theta)$ is the *prior* and it represents our beliefs about the parameters before observing the data, $p(\mathcal{D}|\theta)$ is the likelihood as we have defined it above, and $p(\mathcal{D})$ is the evidence also denoted as the *marginal likelihood* and can be written as $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)\mathrm{d}\theta$ where we are marginalizing over the parameters θ. This integral usually does not have a closed-form solution, making it challenging to compute the posterior exactly. Therefore, one has to rely on approximate inference techniques in order to approximate (2.64), and in this section, we will present the Laplace approximation, one of such methods that recently has regained popularity in the context of neural networks due to its simplicity. Alternative approaches involve either using sampling to get samples from $p(\theta|\mathcal{D})$, which is usually computationally expensive

[14] Here we will denote $p(\mathbf{y}|\mathbf{x},\theta)$ instead of $p_\theta(\mathbf{y}|\mathbf{x})$ to highlight the fact that we are treating θ also as random variables. In addition, this can also be written as $p(\mathbf{y}|f(\mathbf{x},\theta))$ to show the dependency of $p(\mathbf{y}|\mathbf{x},\theta)$ on the function $f(\cdot,\theta)$.

for large neural networks, or using variational inference and learning a variational distribution $q(\theta)$ that approximates $p(\theta|\mathcal{D})$. Since our focus is on the Laplace approximation, we will not cover these two other approaches here. Notice that θ are the parameters of a neural network and we have a way to approximate $p(\theta|\mathcal{D})$ and get samples from it. In the following, we will refer to f_θ as a Bayesian neural network.

In light of all the quantities we have presented above, the benefit of the Bayesian approach is that it allows us to solve two inference tasks of interest. First, given a model m with parameters θ, the posterior distribution $p(\theta|\mathcal{D}, m)$ gives us the parameters that are more plausible given the data. In addition to that, if we have two different models m_1 and m_2, with the corresponding parameters θ_1 and θ_2, we can use the marginal likelihood $p(\mathcal{D}|m) = \int p(\mathcal{D}|\theta, m)p(\theta|m)\mathrm{d}\theta$ for model selection (MacKay, 2003). This is done by using Bayes' rule at the model level, i.e. $p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{p(\mathcal{D})} \propto p(\mathcal{D}|m)p(m)$. These quantities are the same as the one we introduced above, with the only difference being that in this case, we are considering the model as an additional random variable. It has been shown that the marginal likelihood prefers simpler models over more complex ones (Jefferys & Berger, 1992; Rasmussen & Ghahramani, 2000; MacKay, 2003), naturally encoding a notion of Occam's razor principle. The connection between model selection using the marginal likelihood and better generalization is yet debated (MacKay, 1992; Fong & Holmes, 2020; Lotfi et al., 2022). As we are going to present in the following, Laplace approximation is a cheap and practical way to compute an approximation of both the posterior distribution $p(\theta|\mathcal{D})$ and the log-marginal likelihood or model evidence $p(\mathcal{D})$, for a specific model choice m.

MLE, MAP, and the connection to ERM. Since approximating the posterior is challenging, finding a point estimate is usually preferred in practice, i.e. assuming that the posterior is just a delta distribution on the most likely parameters. There are two common ways to obtain such an estimate: The first way is to maximize the likelihood distribution $p(\mathcal{D}|\theta)$ without an *a priori* assumption on the weights[15]. The estimate obtained is referred to as *maximum likelihood estimate*:

$$\theta^*_{\text{MLE}} = \arg\max_\theta p(\mathcal{D}|\theta) = \arg\max_\theta \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x}_i, \theta). \tag{2.65}$$

Denoting the posterior as a delta function, that is, $p(\theta|\mathcal{D}) = \delta(\theta - \theta^*_{\text{MLE}})$, we can insert it into (2.63) resulting in $p(\mathbf{y}^*|\mathbf{x}^*, \theta^*_{\text{MLE}})$.

The second approach takes into account the prior over the weights $p(\theta)$ and is called *maximum a-posteriori (MAP)* inference. From (2.64), we can see that the denominator does not depend on θ and therefore we have $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$, and therefore from an optimization perspective, both the true posterior and the right-hand side of

[15]This can be seen as having a flat prior over the weights.

the equation have the same maximum or mode. Additionally, the estimate θ^*_{MAP} can be found by maximizing the following expression:

$$\begin{aligned}\theta^*_{\text{MAP}} &= \arg\max_\theta p(\theta|\mathcal{D}) = \arg\max_\theta \log p(\theta|\mathcal{D}) \\ &= \arg\max_\theta \log\left[p(\mathcal{D}|\theta)p(\theta)\right] \\ &= \arg\max_\theta \left[\sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x}_i,\theta) + \log p(\theta)\right]. \end{aligned} \tag{2.66}$$

In this setting, we also assume that the posterior distribution is just a delta function centered at θ^*_{MAP}, i.e. $p(\theta|\mathcal{D}) = \delta(\theta - \theta^*_{\text{MAP}})$ and in this case the posterior predictive becomes $p(\mathbf{y}^*|\mathbf{x}^*, \theta^*_{\text{MAP}})$. We can see that, by comparing (2.66) and (2.61), both equations have a similar structure. For certain losses and certain regularization, we can show that performing ERM is equivalent to MAP inference. Indeed, performing ERM with the squared loss and the squared L2-norm as regularizer is equivalent to computing the MAP estimate by having a Gaussian likelihood up to a scaling factor and a Gaussian prior where the variance depends on the regularization strength. Using the cross-entropy loss instead is equivalent to a categorical likelihood. We will show this relationship in the gray box below. Therefore, we can retrieve the following mapping: $\mathcal{L}_{\text{ERM}}(\mathcal{D}, \theta)$ of (2.61) corresponds to the negative log posterior, i.e. $\mathcal{L}_{\text{ERM}}(\mathcal{D}, \theta) = -\log p(\theta|\mathcal{D})$, where the empirical loss is equivalent to the negative log-likelihood $L(\mathbf{y}_i, f(\mathbf{x}_i, \theta)) = -\log p(\mathbf{y}_i|\mathbf{x}_i, \theta)$ and the regularization term is the negative log-prior $R(\theta) = -\log p(\theta)$.

However, defining the posterior distribution as a delta function, or equivalently finding the best parameter estimate and using it to compute predictions, can be problematic. Instead, we would prefer to know how much we should trust this set of parameters. In this direction, standard neural networks have been shown to be poorly calibrated and especially overconfident outside the data region and in out-of-distribution examples (Guo et al., 2017; Nguyen et al., 2015). In addition to that, the mode and in this specific setting θ^*_{MAP} might not be representative of the entire posterior distribution. For this reason, having a way to approximate the posterior in a better way than a delta function can help in solving some of these issues.

ERM and MAP relationship in the regression setting. We have mentioned that there is a connection between the MAP estimate and the ERM approach. We will show that computing the MAP with a Gaussian likelihood and a standard normal prior is equivalent to ERM with a squared loss and a squared L2-norm regularization. For simplicity, we assume that both

$\mathbf{x} \in \mathcal{X} = \mathbb{R}^D$ and $y \in \mathcal{Y} = \mathbb{R}$. We can thus write (2.61) as

$$\mathcal{L}_{\text{ERM}}(\mathcal{D}, \theta) = \frac{1}{N} \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 + \lambda \|\theta\|_2^2.$$

Let us start by considering for simplicity a Gaussian likelihood with mean parameterized by a neural network and fixed variance $f(\mathbf{x}, \theta)$, i.e. $p(y|x, \theta) = \Phi(y|f(\mathbf{x}, \theta)) = \text{N}(y|f(\mathbf{x}, \theta), \sigma^2)$. We have

$$\begin{aligned}
\log p(\mathcal{D}|\theta) &= \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \theta) = \sum_{i=1}^{N} \log \text{N}(y_i|f(\mathbf{x}_i, \theta), \sigma^2) \\
&= \sum_{i=1}^{N} \log \left\{ \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(f(\mathbf{x}_i, \theta) - y_i)^2}{2\sigma^2} \right\} \\
&= -\frac{N}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (f(\mathbf{x}_i, \theta) - y_i)^2 \\
&= -\frac{1}{2\sigma^2} \sum_{i=1}^{N} (f(\mathbf{x}_i, \theta) - y_i)^2 + C.
\end{aligned}$$

If we assume to have the following zero mean prior over the weights $p(\theta) = \text{N}\left(\theta|\mathbf{0}, \frac{\lambda^{-1}}{2}\mathbf{I}\right)$, where the variance depends on a parameter λ that we can be freely chosen. Then we can write $\log p(\theta)$ in closed form as:

$$\begin{aligned}
\log p(\theta) &= \log \text{N}\left(\theta|\mathbf{0}, \frac{\lambda^{-1}}{2}\mathbf{I}\right) \\
&= \log \left\{ \frac{1}{(2\pi)^{k/2} |\frac{\lambda^{-1}}{2}\mathbf{I}|^{1/2}} \exp - \frac{1}{2} \theta^T \left(\frac{\lambda^{-1}}{2}\mathbf{I}\right)^{-1} \theta \right\} \\
&= \log \left\{ \frac{1}{(2\pi)^{k/2} \left(\frac{1}{2\lambda}\right)^{p/2}} \exp - \lambda \theta^T \theta \right\} \\
&= \log \left\{ \frac{1}{(2\pi)^{k/2} \left(\frac{1}{2\lambda}\right)^{k/2}} \exp - \lambda \|\theta\|_2^2 \right\} \\
&= -\frac{k}{2} \log 2\pi - \frac{k}{2} \log \left(\frac{1}{2\lambda}\right) - \lambda \|\theta\|_2^2 \\
&= -\lambda \|\theta\|_2^2 + C,
\end{aligned}$$

where we assume that $\theta \in \mathbb{R}^k$. By putting everything together:

$$\arg\max_{\theta} \log p(\theta|\mathcal{D}) = \arg\min_{\theta} \frac{1}{2\sigma^2} \sum_{i=1}^{N} (f(\mathbf{x}, \theta) - y)^2 + \lambda \|\theta\|_2^2 + C$$

where we show that ERM has the same minimum as the MAP inference. Indeed, if we have a finite training set, then $1/N$ is just a constant as $1/(2\sigma^2)$ that does not change the minimum. Therefore, the parameters found by performing the ERM can be interpreted as the mode of the posterior distribution $p(\theta|\mathcal{D})$.

Laplace approximation. Laplace approximation (LA) is a simple way to approximate any unnormalized distribution using a Gaussian distribution centered around its peak. In the context of neural networks, Mackay (1992) was the first one to propose its use to approximate the posterior $p(\theta|\mathcal{D})$ for small networks. Recently, its application for approximating the posterior distribution of larger neural networks has been made possible by new computationally feasible ways to approximate the Hessian based on the generalized Gauss-Newton (GGN) approximation (Foresee & Hagan, 1997; Schraudolph, 2002; Martens & Grosse, 2015; Botev et al., 2017; Ritter et al., 2018), easy-to-use software (Daxberger et al., 2021a), and new approaches to scale up computations (Antorán et al., 2022). In the gray box above we have highlighted that the standard training of neural networks by ERM corresponds to finding the MAP estimate, that is the mode of the posterior. The success of Laplace lies also in the fact that it can be applied to any pretrained neural network in a post-hoc procedure and turn it into a Bayesian neural network where predictions will then be computed using (2.63).

As we have seen above, we have no access to the true posterior, but only to its unnormalized version, i.e. $p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} = \frac{1}{Z}p(\mathcal{D}|\theta)p(\theta)$. The maximum of $p(\theta|\mathcal{D})$ can be found by just computing the MAP estimate as shown in (2.66)[16]

$$\theta^*_{\text{MAP}} = \arg\max_\theta \log p(\theta|\mathcal{D}) = \arg\max_\theta \underbrace{\left[\sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x}_i, \theta) + \log p(\theta)\right]}_{\mathcal{L}(\theta,\mathcal{D})}.$$

If we compute a second-order Taylor expansion of $\mathcal{L}(\theta, \mathcal{D})$ around its peak θ^*_{MAP} we get:

$$\hat{\mathcal{L}}(\theta, \mathcal{D}) \approx$$
$$\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D}) + \underbrace{(\theta - \theta^*_{\text{MAP}})\nabla_\theta \mathcal{L}(\theta, \mathcal{D})|_{\theta^*_{\text{MAP}}}}_{=0} + \frac{1}{2}(\theta - \theta^*_{\text{MAP}})^T \nabla^2_\theta \mathcal{L}(\theta, \mathcal{D})|_{\theta^*_{\text{MAP}}}(\theta - \theta^*_{\text{MAP}}). \tag{2.67}$$

[16] We want to remark that in this setting we have $p(\mathbf{y}|\mathbf{x}, \theta) = \Phi(\mathbf{y}|f(\mathbf{x}, \theta))$, where $f(\mathbf{x}, \theta)$ is a neural network. Sometimes this is also written as $p(\mathbf{y}|f(\mathbf{x}, \theta))$, which highlights the dependency on the function $f(\cdot, \theta)$.

Since, in principle, we are at the MAP, then we have that $\nabla_\theta \mathcal{L}(\theta, \mathcal{D})|_{\theta^*_{\text{MAP}}} = 0$, resulting in the first order term being equal to 0. Thus, by defining $\Sigma = -\nabla^2_\theta \mathcal{L}(\theta, \mathcal{D})|_{\theta^*_{\text{MAP}}}$, we can rewrite (2.67) as:

$$\hat{\mathcal{L}}(\theta, \mathcal{D}) \approx \mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D}) - \frac{1}{2}(\theta - \theta^*_{\text{MAP}})^T \Sigma (\theta - \theta^*_{\text{MAP}}). \tag{2.68}$$

We can recall that the right-hand side of (2.68) is given by $\log[p(\mathcal{D}|\theta)p(\theta)]$ which is the numerator of the unnormalized log-posterior. Therefore, by taking the exponential, we get:

$$\begin{aligned} p(\mathcal{D}|\theta)p(\theta) &\approx \exp\big\{\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D}) - \frac{1}{2}(\theta - \theta^*_{\text{MAP}})^T \Sigma (\theta - \theta^*_{\text{MAP}})\big\} \\ &= \exp\{\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D})\} \exp\big\{-\frac{1}{2}(\theta - \theta^*_{\text{MAP}})^T \Sigma (\theta - \theta^*_{\text{MAP}})\big\} \end{aligned} \tag{2.69}$$

The integral of $p(\mathcal{D}|\theta)p(\theta)$ is the normalizing constant Z. By integrating (2.69) we obtain:

$$\begin{aligned} Z &\approx \int \exp\{\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D})\} \exp\big\{-\frac{1}{2}(\theta - \theta^*_{\text{MAP}})^T \Sigma (\theta - \theta^*_{\text{MAP}})\big\} \mathrm{d}\theta \\ &= \exp\{\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D})\} \int \exp\big\{-\frac{1}{2}(\theta - \theta^*_{\text{MAP}})^T \Sigma (\theta - \theta^*_{\text{MAP}})\big\} \mathrm{d}\theta \\ &= \exp\{\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D})\} \frac{(2\pi)^{\frac{k}{2}}}{(\det \Sigma)^{\frac{1}{2}}}, \end{aligned} \tag{2.70}$$

where we used the fact the integral is the Gaussian integral and the fact that $\det \Sigma^{-1} = \frac{1}{\det \Sigma}$ and we assume $\theta \in \mathbb{R}^k$. We now have an approximation both for $p(\mathcal{D}|\theta)p(\theta)$ and Z, therefore, by putting everything together we can get an approximation for the posterior $p(\theta|\mathcal{D})$:

$$\begin{aligned} p(\theta|\mathcal{D}) &= \frac{1}{Z} p(\mathcal{D}|\theta)p(\theta) \\ &\approx \exp\{\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D})\} \exp\big\{-\frac{1}{2}(\theta - \theta^*_{\text{MAP}})^T \Sigma (\theta - \theta^*_{\text{MAP}})\big\} \frac{(\det \Sigma)^{\frac{1}{2}}}{(2\pi)^{\frac{k}{2}} \exp\{\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D})\}} \\ &= \frac{(\det \Sigma)^{\frac{1}{2}}}{(2\pi)^{\frac{k}{2}}} \exp\big\{-\frac{1}{2}(\theta - \theta^*_{\text{MAP}})^T \Sigma (\theta - \theta^*_{\text{MAP}})\big\} \\ &= \mathrm{N}(\theta|\theta^*_{\text{MAP}}, \Sigma^{-1}). \end{aligned} \tag{2.71}$$

So far, we have shown that we can approximate the posterior with a Gaussian distribution centered on θ^*_{MAP} and with covariance given by Σ^{-1}, where $\Sigma = -\nabla^2_\theta \mathcal{L}(\theta, \mathcal{D})|_{\theta^*_{\text{MAP}}}$.

The computation of Σ is the main computational bottleneck to apply LA. We can elaborate on Σ a bit more by plugging in the definition of $\mathcal{L}(\theta, \mathcal{D})$. This consists of a

contribution coming from the likelihood term and one from the prior term:

$$\Sigma = -\nabla^2_\theta \mathcal{L}(\theta, \mathcal{D})|_{\theta^*_{\text{MAP}}} = -\nabla^2_\theta \left[\sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x}_i, \theta) + \log p(\theta) \right] \Bigg|_{\theta^*_{\text{MAP}}}. \tag{2.72}$$

If we assume an isotropic Gaussian prior over the parameters, i.e $p(\theta) = \mathrm{N}(\theta; 0, \sigma^2\mathbf{I})$, then we have $-\nabla^2_\theta \log p(\theta)|_{\theta^*_{\text{MAP}}} = \frac{1}{\sigma^2}\mathbf{I}$, so we can rewrite Σ according to:

$$\Sigma = -\nabla^2_\theta \sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x}_i, \theta)\Big|_{\theta^*_{\text{MAP}}} + \frac{1}{\sigma^2}\mathbf{I} = \mathrm{H} + \alpha\mathbf{I}, \tag{2.73}$$

where we denote with $\alpha = 1/\sigma^2$ the prior precision and with H the contribution from the likelihood. Computing H involves computing the Hessian of a large neural network, which is generally not positive definite (Sagun et al., 2016). In addition to that, many architectures we use in deep learning are not twice differentiable, as is the case if we use the ReLU activation. Thus, the Hessian is commonly approximated by the Generalized Gauss-Newton GGN approximation (Schraudolph, 2002), which is given by:

$$\begin{aligned} -\nabla^2_\theta \sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x}_i, \theta) &\approx \sum_{i=1}^{N} (\nabla_\theta f(\mathbf{x}_i, \theta)|_{\theta^*_{\text{MAP}}})^T (-\nabla^2_f \log p(\mathbf{y}_i|f)|_{f=f(\mathbf{x}_i, \theta^*_{\text{MAP}})}) (\nabla_\theta f(\mathbf{x}_i, \theta)|_{\theta^*_{\text{MAP}}}) \\ &= \sum_{i=1}^{N} J^T_{\theta^*_{\text{MAP}}} \Lambda(\mathbf{x}_i, \mathbf{y}_i) J_{\theta^*_{\text{MAP}}}, \end{aligned} \tag{2.74}$$

where $J_{\theta^*_{\text{MAP}}}$ is the Jacobian of the output of the network with respect to the parameter of the network and $\Lambda(\mathbf{x}_i, \mathbf{y}_i)$ is the output Hessian, that is, the Hessian of the log-likelihood with respect to the network output, i.e. the logits in case of classification. The GGN approximation is guaranteed to be positive definite, resulting in Σ being invertible.

An additional problem is that for big networks the Hessian H cannot be stored as it involves storing a matrix of size $k \times k$, where k is the number of parameters. In Chapter 4, we limit ourselves to considering only small networks for which the full Hessian computation is possible. To apply Laplace to big models one has to rely on further factorization assumptions for a memory-efficient approximation of the Hessian. The two most common factorizations are the diagonal approximation (LeCun et al., 1989; Denker & LeCun, 1990) and block-diagonal factorizations such as the Kronecker-factored approximate curvature (KFAC) (Heskes, 2000; Martens & Grosse, 2015; Botev et al., 2017; Eschenhagen et al., 2024).

Post-hoc prior precision tuning and linearization. Having an approximation for the posterior distribution allows us to compute the predictive distribution defined in (2.63). While there are different ways to approximate it depending on the task

one is solving, in this thesis, we focus on approximating the integral by using Monte Carlo samples:

$$\begin{aligned} p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) &= \int p(\mathbf{y}^*|\mathbf{x}^*, \theta) p(\theta|\mathcal{D}) \mathrm{d}\theta \\ &= \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})}[p(\mathbf{y}^*|\mathbf{x}^*, \theta)] \\ &\approx \mathbb{E}_{\theta \sim \mathrm{N}(\theta|\theta^*_{\mathrm{MAP}}, \Sigma^{-1})}[p(\mathbf{y}^*|\mathbf{x}^*, \theta)] \\ &\approx \frac{1}{S} \sum_{i=1}^{S} p(\mathbf{y}^*|\mathbf{x}^*, \theta^{(s)}) \qquad \theta^{(s)} \sim \mathrm{N}(\theta|\theta^*_{\mathrm{MAP}}, \Sigma^{-1}), \end{aligned} \tag{2.75}$$

where in the third line we approximated the posterior $p(\theta|\mathcal{D})$ with the recently introduced Laplace approximation $\mathrm{N}(\theta|\theta^*_{\mathrm{MAP}}, \Sigma^{-1})$. However, estimating the predictive distribution using Monte Carlo samples from the Laplace approximated posterior (2.75) has been shown to perform poorly (Lawrence, 2001, Chapter 5)(Ritter et al., 2018) even for small networks. This has been mostly associated with the problem of LA placing probability mass in regions of low posterior probability density. As a reminder, we assume that $p(\mathbf{y}^*|\mathbf{x}^*, \theta) = \Phi(\mathbf{y}^*|f(\mathbf{x}^*, \theta))$, i.e. the likelihood function is parameterized by a neural network $f(\cdot, \theta)$. This can also be denoted as $p(\mathbf{y}^*|\mathbf{x}^*, \theta) = p(\mathbf{y}^*|f(\mathbf{x}^*, \theta))$. A solution to the underfitting problem of the classic Laplace approximation, already proposed by (Mackay, 1992, Chapter 4), is to consider a first-order Taylor expansion of $f(\mathbf{x}^*_i, \theta)$ around θ^*_{MAP} and consider the linearized function when computing (2.75):

$$f_{\mathrm{lin}}(\mathbf{x}^*_i, \theta) = f(\mathbf{x}^*_i, \theta^*_{\mathrm{MAP}}) + (\nabla_\theta f(\mathbf{x}^*_i, \theta)|_{\theta^*_{\mathrm{MAP}}})^T (\theta - \theta_{\mathrm{MAP}}), \tag{2.76}$$

where $\nabla_\theta f(\mathbf{x}^*_i, \theta)|_{\theta^*_{\mathrm{MAP}}}$ is the Jacobian. The predictive distribution can be approximated as $p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) \approx \frac{1}{S} \sum_{i=1}^{S} p(\mathbf{y}^*|f_{\mathrm{lin}}(\mathbf{x}^*, \theta^{(s)}))$ with $\theta^{(s)} \sim \mathrm{N}(\theta|\theta^*_{\mathrm{MAP}}, \Sigma^{-1})$.This approach was also justified by Khan et al. (2019); Immer et al. (2021b), who proved that the GGN approximation we use to compute the Hessian in (2.74) is the exact Hessian of this new linearized model. Even if this is a linear function with respect to the parameters θ, empirically it achieves better performance than the classic Laplace approximation.

An additional ingredient that plays a crucial role in making the Laplace approximation work is the post-hoc tuning of the prior precision α in (2.73) (Ritter et al., 2018; Kristiadi et al., 2020; Immer et al., 2021a; Daxberger et al., 2021a). While this is not theoretically justified, it has the effect of regularizing the Hessian, usually leading to better performance in terms of predictive performance. The optimization is typically done using cross-validation or by maximizing the log-marginal likelihood. Indeed, by

looking at (2.70), we can get an approximation to $\log p(D)$:

$$
\begin{aligned}
p(D) &\approx \exp\{\mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D})\} \frac{(2\pi)^{\frac{p}{2}}}{(\det \Sigma)^{\frac{1}{2}}} \\
&\text{(by taking the logarithm on both sides)} \\
\log p(D) &\approx \mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D}) + \log \left\{ \frac{(2\pi)^{\frac{p}{2}}}{(\det \Sigma)^{\frac{1}{2}}} \right\} \\
&\approx \mathcal{L}(\theta^*_{\text{MAP}}, \mathcal{D}) + \frac{p}{2} \log 2\pi - \frac{1}{2} \log \det \Sigma, \qquad (2.77)
\end{aligned}
$$

which can be then used to optimize α.

On a final note, despite in Chapter 4 we will put the attention on approximating the posterior distribution of all the weights of the neural network, a lot of works showed that just considering a subset of the network parameters (Daxberger et al., 2021b; Sharma et al., 2023), especially the last-layer, is enough for obtaining good uncertainty estimation, matching and sometimes out-performing fully stochastic networks.

2.3 A primer on differential geometry

In the previous sections we presented four different types of deep generative models and we showed how one can use Laplace approximation to practically turn a standard neural network into a Bayesian neural network. In this section, instead, we present some basic concepts from differential and Riemannian geometry that are needed to understand in detail the core idea we propose in Paper 4. We are going to introduce those from the machine learning perspective trying to make them as accessible as possible. This dissertation follows the structure of Hauberg (2024) and for a more formal introduction to the topic, we suggest the reader to look at Lee (2019); do Carmo (1992).

Geometry is the branch of mathematics that deals with surfaces and their properties, such as curves, distances, angles, and volumes. Geometry plays an important role also in machine learning, because the statistical tools we are using are usually built based on certain geometrical assumptions on the space we are working on. Therefore, if the assumptions are wrong, then the results obtained by using those methods are not reliable. Most tools we use in machine learning are built on the assumption that we are operating in a Euclidian space, i.e., roughly speaking a space that is flat where distances between two points can be represented as a straight line. An example where this assumption does not hold is the low-dimensional latent space learned by a VAE (Hauberg, 2024). In Paper 4, instead, we show that there are some benefits by also interpreting the parameter space of a trained neural network as a non-Euclidean space. To deal with non-Euclidian spaces, then we need to use tools from differential

geometry. This section will introduce these specific tools. We will define what a *manifold* is and, more specifically, a Riemannian manifold, and we will present how to compute operations that we usually compute in the Euclidean case, such as curves and distances, on it.

Manifolds. The usual definition or introduction to manifolds is done starting from topology. This definition is rather abstract and formal. We will try, instead, to introduce the manifold in a more intuitive way. The easiest way to conceptualize the notion of a manifold is as a hypersurface embedded in a higher-dimensional space. Another way to visualize a manifold is to think of a collection of points in $\mathbb{R}^D$ that are somewhat connected to create a surface. Suppose that these surfaces locally resemble a d-dimensional Euclidian space, i.e. every point in the surface has a small local neighborhood that looks like a regular n-dimensional space. In that case, these are referred to as *topological manifolds*. In the case where we are interested in defining functions on these manifolds and performing calculus operations including differentiation on them, then we have to consider surfaces that do not contain any critical points like sharp corners. If a topological manifold satisfies these constraints, then it is a *smooth manifold*. For example, consider a cone or a sphere embedded in a three-dimensional space; then these surfaces are manifolds. We can see that the tip of the cone is a critical point that is not smooth, making the cone a topological manifold but not a smooth one. The sphere, on the other hand, is a smooth manifold. In the following, we will consider only smooth manifolds.

One way to introduce the concept of a manifold and the operations that can be done on it is by starting with its parameterization or mapping. Let us assume, as in the example mentioned above, that we have some points living in a certain dataspace $\mathcal{X}$ and we want to fit a lower-dimensional manifold to those, i.e. we are interested in finding the smooth surface where the points lie. For latent variable models, this is also motivated by the manifold hypothesis that we introduced in Section 2.1.1. We can then start by considering a smooth function:

$$f : \Omega \to \mathcal{X}, \tag{2.78}$$

where usually Ω have a lower dimension than $\mathcal{X}$. As a running example, we can consider $\mathcal{X} \subset \mathbb{R}^D$ and $\Omega \subset \mathbb{R}^d$, with $d < D$. Therefore, we can define a *manifold* $\mathcal{M}$ as $\mathcal{M} = f(\Omega)$, that is, the manifold $\mathcal{M}$ is the image of Ω under the mapping function f. The space $\mathcal{X}$ is commonly referred to as *ambient space*. The space Ω, instead, is usually referred to as *coordinate chart*. A technical constraint, as we will see further in the section, is that the inverse function f^{-1} exists. For this reason, sometimes we need to use multiple functions f_i, each with its own coordinate chart Ω_i, to parameterize a manifold. This is the case, for example, if our manifold is a sphere. The collection of all the charts Ω_i is called *smooth atlas*. In this dissertation, as we did in Paper 4, instead, we assume that there exists a global parameterization

for the manifold, i.e. by using the function f we can uniquely describe all the points in the manifold. In this specific case, the space Ω is also referred to as the *intrinsic coordinates*.

Until now, we only required the function f to be smooth. If the function f is also invertible in $\mathcal{M}$, then we say that we have *embedded manifold*. If this is the case, then the manifold does not self-intersect. This can be achieved by having the function f as an injective, i.e. $f(a) = f(b) \implies a = b$. Thus, the inverse mapping f^{-1} exists. Requiring that the mapping function f be invertible everywhere can be too restrictive. If we consider f to be only locally invertible, then we have am *immersed manifold*. By locally invertible, we required Jacobian $\frac{\partial f}{\partial x}$ to be full rank for all $x \in \Omega$, meaning that the manifold is not collapsing. If, for example, we consider the case where the function f is parametrized by a neural network, then f is not guaranteed to be injective, and therefore the manifold we get will be immersed. If, instead, we consider, as we did in Paper 4 the following function $f(\theta) = [\mathcal{L}(\theta), \theta]^T \triangleq \bar{\mathbf{x}} \in \mathcal{M}$, where θ are the parameters of a neural network and $\mathcal{L}(\theta)$ the associated loss function, this results in an embedded manifold. Indeed, the inverse function is defined everywhere and it has the following form $f^{-1}(\bar{\mathbf{x}}) = \theta$, where we are just returning the $\bar{\mathbf{x}}$ without the first element.

Tangent spaces and Riemannian metric. We have presented a simple way we can use to parameterize a manifold. To be able to perform computations involving the manifold, we have to introduce the concept of tangent space and the Riemannian metric. Since we assume the function f to be smooth, we can approximate the function f around a point $\mathbf{x} \in \Omega$ using a first-order Taylor approximation

$$f(\mathbf{x} + \epsilon) = f(\mathbf{x}) + \mathbf{J}_{\mathbf{x}}\epsilon + o(\|\epsilon\|), \tag{2.79}$$

where $\mathbf{J}_{\mathbf{x}} \in \mathbb{R}^{D \times d}$ is the Jacobian matrix evaluated at $\mathbf{x}$. The first-order Taylor approximation is also referred to as a *linearization* of the function f at $\mathbf{x}$, for example, if our function lives in $\mathbb{R}^3$, then the first-order Taylor expansion is a plane, i.e. $\mathbb{R}^2$ living in $\mathbb{R}^3$. Thus we can think of $\mathbf{J}_{\mathbf{x}}$ as the basis of the d-dimensional linear subspace inside $\mathbb{R}^D$ that is tangential to the manifold $\mathcal{M}$ at the point $f(\mathbf{x})$. We call this plane *tangent space* and denote it as $\mathcal{T}_{f(\mathbf{x})}\mathcal{M}$. This contains all vectors tangential to the point $f(\mathbf{x}) \in \mathcal{M}$. The tangent space is important because it gives us access to a local linearized view of the manifolds that is only valid in the neighborhood around $\mathbf{x}$. In addition to that, the Jacobian matrix can also be interpreted as a linear map that uniquely maps a vector $\mathbf{v} \in \Omega$ from the tangent coordinate to a tangent vector in the tangent space $\bar{\mathbf{v}} \in \mathcal{T}_{f(\mathbf{x})}\mathcal{M}$, as $\bar{\mathbf{v}} = \mathbf{J}_{\mathbf{x}}\mathbf{v} \in \mathbb{R}^D$.

Let us consider two vectors $\mathbf{v}_1, \mathbf{v}_2 \in \Omega$ starting from $\mathbf{x} \in \Omega$. Their representation in the tangent space $\mathcal{T}_{f(\mathbf{x})}\mathcal{M}$ is given by $\bar{\mathbf{v}}_1 = \mathbf{J}_{\mathbf{x}}\mathbf{v}_1$ and $\bar{\mathbf{v}}_2 = \mathbf{J}_{\mathbf{x}}\mathbf{v}_2$ and since the ambient space is Euclidean by assumption, we can compute the inner product between the

two vectors as:

$$\langle \bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2 \rangle_{\mathbf{x}} = \langle \mathbf{J}_{\mathbf{x}}\mathbf{v}_1, \mathbf{J}_{\mathbf{x}}\mathbf{v}_2 \rangle_{\mathbf{x}} = (\mathbf{J}_{\mathbf{x}}\mathbf{v}_1)^T(\mathbf{J}_{\mathbf{x}}\mathbf{v}_2) = \mathbf{v}_1^T\mathbf{J}_{\mathbf{x}}^T\mathbf{J}_{\mathbf{x}}\mathbf{v}_2 \tag{2.80}$$

where $\langle \cdot, \cdot \rangle_{\mathbf{x}}$ indicates that the inner product should be computed in the tangent space at the point $\mathbf{x}$. From (2.80) we can see that if we are interested in computing the inner product in the tangent space of the manifold, we can easily compute it on the intrinsic coordinates Ω if we know the matrix $\mathbf{J}_{\mathbf{x}}^T\mathbf{J}_{\mathbf{x}}$. The matrix $\mathbf{J}_{\mathbf{x}}^T\mathbf{J}_{\mathbf{x}}$ is referred to as the *Riemannian metric*, usually denoted as $\mathbf{M}(\mathbf{x}) = \mathbf{J}_{\mathbf{x}}^T\mathbf{J}_{\mathbf{x}}$, and it defines a local inner product[17] $\langle \mathbf{a}, \mathbf{b} \rangle_{\mathbf{x}} = \mathbf{a}^T\mathbf{M}(\mathbf{x})\mathbf{b}$ associated with the tangent space $\mathcal{T}_{f(\mathbf{x})}\mathcal{M}$, with $\mathbf{a}, \mathbf{b} \in \mathcal{T}_{f(\mathbf{x})}\mathcal{M}$. By equipping the intrinsic coordinate space Ω with the Riemannian metric $\mathbf{M}(\mathbf{x})$ we take into account how the infinitesimal volume $\mathrm{d}\mathbf{x} \in \Omega$ is distorted when mapped on the manifold $\mathcal{M}$ through the immersion function f.

By putting everything together, we can define a *Riemannian manifold* as a smooth manifold $\mathcal{M}$ where each tangent space $\mathcal{T}_{f(\mathbf{x})}\mathcal{M}$ is equipped with a inner product called *Riemannian metric* $\langle \mathbf{a}, \mathbf{b} \rangle_{\mathbf{x}} = \mathbf{a}^T\mathbf{M}(\mathbf{x})\mathbf{b}$ that changes smoothly on the manifold.

Remark: A graph of a function is a manifold. A simple and intuitive example of a manifold can be borrowed from calculus. Let us assume we have a function $\mathbf{y} = f(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ and $\mathbf{y} = (y_1, y_2, y_3) \in \mathbb{R}^3$. If we consider the following function $f : \mathbb{R}^2 \to \mathbb{R}^3$:

$$\mathbf{y} = f(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} f_1(x_1, x_2) \\ x_1 \\ x_2 \end{pmatrix},$$

this usually known as the *graph of a function*, and this is a way to parametrize surfaces in $\mathbb{R}^3$. The image of this function is a manifold. The Jacobian of this function is then given by:

$$\mathbf{J}_f = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial \mathbf{x}} \\ \frac{\partial f_2(\mathbf{x})}{\partial \mathbf{x}} \\ \frac{\partial f_3(\mathbf{x})}{\partial \mathbf{x}} \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} \\ \frac{\partial f_3(\mathbf{x})}{\partial x_1} & \frac{\partial f_3(\mathbf{x})}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The Jacobian can also be written as follows $\mathbf{J}_f = [\nabla_{\mathbf{x}} f_1(\mathbf{x}), \mathbb{I}_2]^T$. This is one of the ways used in calculus to parameterize a surface in $\mathbb{R}^3$.

We have seen that the Jacobian plays an important role, especially the term

[17] Sometimes it can be found simply written as $\langle \mathbf{a}, \mathbf{M}(\mathbf{x})\mathbf{b} \rangle$.

$\mathbf{J}_f^T\mathbf{J}_f$, which defines the *metric*. This is given by:

$$\begin{aligned}
\mathbf{M}(\mathbf{x}) = \mathbf{J}_f^T\mathbf{J}_f &= \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & 1 & 0 \\ \frac{\partial f_1(\mathbf{x})}{\partial x_2} & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \left(\frac{\partial f_1(\mathbf{x})}{\partial x_1}\right)^2 + 1 & \frac{\partial f_1(\mathbf{x})}{\partial x_1}\frac{\partial f_1(\mathbf{x})}{\partial x_2} \\ \frac{\partial f_1(\mathbf{x})}{\partial x_1}\frac{\partial f_1(\mathbf{x})}{\partial x_2} & \left(\frac{\partial f_1(\mathbf{x})}{\partial x_2}\right)^2 + 1 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} \left(\frac{\partial f_1(\mathbf{x})}{\partial x_1}\right)^2 & \frac{\partial f_1(\mathbf{x})}{\partial x_1}\frac{\partial f_1(\mathbf{x})}{\partial x_2} \\ \frac{\partial f_1(\mathbf{x})}{\partial x_1}\frac{\partial f_1(\mathbf{x})}{\partial x_2} & \left(\frac{\partial f_1(\mathbf{x})}{\partial x_2}\right)^2 \end{pmatrix} \\
&= \mathbb{I}_2 + \nabla_{\mathbf{x}} f_1(\mathbf{x}) \nabla_{\mathbf{x}} f_1(\mathbf{x})^T,
\end{aligned}$$

where we have define $\nabla_{\mathbf{x}} f_1(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} \\ \frac{\partial f_1(\mathbf{x})}{\partial x_2} \end{pmatrix}$.

Curves and Geodesics. Now that we know how to compute the inner product in the tangent space, we can explain how to compute the length of a smooth curve. We start by considering a smooth curve $c : [0, 1] \to \Omega$ in the intrinsic coordinates Ω. We know that if the function f is smooth, then the mapping of such a curve c given by $f(c)$ also results in a smooth curve on the manifold. We can write the length of this curve c by mapping it to the manifold and using the Euclidean definition of the

length of the curve in ambient space. This corresponds to:

$$\begin{aligned}
\text{Length}_{\mathcal{M}}[c] \coloneqq \text{Length}[f(c)] &= \int_0^1 \left\| \frac{\partial}{\partial t} f(c(t)) \right\| \mathrm{d}t \\
&= \int_0^1 \left\| \frac{\partial f(c(t))}{\partial c(t)} \frac{\partial c(t)}{\partial t} \right\| \mathrm{d}t \\
&= \int_0^1 \|\mathbf{J}_{c(t)} \dot{c}(t)\| \mathrm{d}t \\
&= \int_0^1 \sqrt{(\mathbf{J}_{c(t)} \dot{c}(t))^T (\mathbf{J}_{c(t)} \dot{c}(t))} \mathrm{d}t \\
&= \int_0^1 \sqrt{\dot{c}(t)^T \mathbf{J}_{c(t)}^T \mathbf{J}_{c(t)} \dot{c}(t)} \mathrm{d}t \\
&= \int_0^1 \sqrt{\dot{c}(t)^T \mathbf{M}(c(t)) \dot{c}(t)} \mathrm{d}t \\
&= \int_0^1 \|\dot{c}(t)\|_{c(t)} \mathrm{d}t, \qquad (2.81)
\end{aligned}$$

where we used the chain rule in the second row, the definition of Euclidean norm in the fourth row and in the end we used the fact that $\sqrt{\langle \mathbf{u}, \mathbf{u} \rangle_{\mathbf{x}}} = \sqrt{\mathbf{u}^T \mathbf{M}(\mathbf{x}) \mathbf{u}} = \|\mathbf{u}\|_{\mathbf{x}}$ is the definition of the local norm. Additionally, $\dot{c}(t) = \frac{\partial c(t)}{\partial t}$ is the *velocity* we are using to traverse the curve. Equation (2.81) also shows that if we are given the metric $\mathbf{M}(c(t))$, we can calculate the length of the curve directly in the intrinsic coordinates Ω, without having to consider the mapping $f(c(t)) \in \mathcal{M}$. If then we are interested in computing *shortest path* between two points $\mathbf{x}_1, \mathbf{x}_2 \in \Omega$, we can just solve the following optimization problem:

$$\begin{aligned}
c^*(t) = &\arg\min_{c(t)} \int_0^1 \|\dot{c}(t)\|_{c(t)} \mathrm{d}t \\
&\text{subject to } c(0) = \mathbf{x}_1 \text{ and } c(1) = \mathbf{x}_2, \qquad (2.82)
\end{aligned}$$

and the resulting curve $c^*(t)$ gives the shortest path between two points. It is important to note that the length of a curve is invariant to reparameterization, which means that we can reparameterize t and get the same length. In fact, the distance of a curve is the same regardless of the velocity used to traverse it. Therefore, the optimization problem in (2.82) does not have a unique solution. The reparameterization issue can be addressed by considering a specific parameterization of the curve. The usual convention is to consider a constant speed curve, i.e. $\|\dot{c}(t)\|_{c(t)}$ is independent of t. A curve that minimizes (2.82) and has constant speed, or equivalently zero acceleration, is called a *geodesic* curve. Roughly speaking, we can think of geodesic on surfaces as what straight lines are on an Euclidian space.

However, in practice, the optimization problem defined in (2.82) using the length of a curve is solved by considering the energy of a curve, which, instead, is invariant to reparameterization. The *energy* of a curve is defined as follows:

$$\mathrm{E}_{\mathcal{M}}[c(t)] = \int_0^1 \|\dot{c}(t)\|^2_{c(t)} \mathrm{d}t, \tag{2.83}$$

and the curve minimizing this has the minimum length and cannot be reparametrized otherwise it will change its energy. It can be shown (see the gray box below) by using the Cauchy-Schwarz inequality that $\mathrm{Length}^2_{\mathcal{M}}[c] \leq \mathrm{E}_{\mathcal{M}}[c]$. This implies that the curve's energy acts as an upper bound for its squared length. Or equivalently, the minimum energy that a curve can have is given by its squared shortest path. This minimum energy will be exactly equal to the squared shortest path if and only if the curve speed is constant, i.e. the only condition for which the Cauchy-Schwarz inequality becomes an equality. Therefore if we minimize the energy of a curve being parametrized with constant speed we also find the curve with minimum length, making it a geodesic. We can thus compute the shortest path between two points by optimizing the energy functional instead, given by:

$$\begin{aligned} c^*(t) &= \arg\min_{c(t)} \mathrm{E}_{\mathcal{M}}[c(t)] = \arg\min_{c(t)} \int_0^1 \|\dot{c}(t)\|^2_{c(t)} \mathrm{d}t. \\ &= \arg\min_{c(t)} \frac{1}{2} \int_0^1 \underbrace{\dot{c}(t)^T \mathbf{M}(c(t)) \dot{c}(t)}_{L(c(t),\dot{c}(t))} \mathrm{d}t \end{aligned}$$

where in the second row we substitute the definition of $\|\dot{c}(t)\|_{c(t)}$ and we will refer to $\sqrt{\dot{c}(t)^T \mathbf{M}(c(t)) \dot{c}(t)}$ as the constant speed term. Arvanitidis et al. (2018) showed that this optimization problem can be solved by applying the Euler-Lagrange equation

$$\frac{\partial L(c(t), \dot{c}(t))}{\partial c(t)} = \frac{\partial}{\partial t} \frac{\partial L(c(t), \dot{c}(t))}{\partial \dot{c}(t)} \tag{2.84}$$

resulting in a system of 2-nd order nonlinear differential equations (ODEs):

$$\ddot{c}(t) = -\frac{\mathbf{M}^{-1}(c(t))}{2} \left[2 \left[\frac{\partial \mathbf{M}(c(t))}{\partial c_1(t)}, \ldots, \frac{\partial \mathbf{M}(c(t))}{\partial c_K(t)} \right] - \frac{\partial \mathrm{vec}[\mathbf{M}(c(t))]}{\partial c(t)}^T \right] (\dot{c}(t) \otimes \dot{c}(t)), \tag{2.85}$$

where the vec[·] operation stacks the columns of a matrix and $\otimes$ the Kronecker product. Solving (2.85) as an initial value problem (IVP) with boundary conditions $c(0) = \mathbf{x}_1$ and $c(1) = \mathbf{x}_2$ returns the geodesic connection $\mathbf{x}_1$ and $\mathbf{x}_2$. As before, we want to highlight that (2.85) can be solved directly in intrinsic coordinates Ω instead of in the extrinsic view of the manifold if we know the Riemannian metric $\mathbf{M}(\mathbf{x})$.

Exponential Maps and Logarithmic Maps. We have the tools to compute inner products, length of curves, and also find the shortest path between two points. We have seen that we can compute these quantities either extrinsically, i.e. in the ambient space where the manifold is embedded, or directly in the coordinate space of the manifold, that is, intrinsically. However, we are still missing two common operations in the Euclidean space: sum and subtraction. Indeed, given a point $\mathbf{x}_1 \in \mathbb{R}^d$ and a vector $\mathbf{v}$, we can easily compute $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{v}$, which results in a new point $\mathbf{x}_2 \in \mathbb{R}^d$, and likewise we can compute $\mathbf{x}_1 - \mathbf{x}_2 = \mathbf{v}$ that results in a vector. We will now present two operations that allow us to compute similar operations involving points in the manifold $f(\mathbf{x}_1), f(\mathbf{x}_2) \in \mathcal{M}$ and a vector $\bar{\mathbf{v}} = \mathbf{J}_{\mathbf{x}_1}\mathbf{v} \in \mathcal{T}_{f(\mathbf{x}_1)}\mathcal{M}$ in the tangent space of $f(\mathbf{x}_1)$. The first operator is the *logarithmic map* defined as $\text{Log}_{f(\mathbf{x}_1)}(f(\mathbf{x}_2)) : \mathcal{M} \to \mathcal{T}_{f(\mathbf{x}_1)}\mathcal{M}$ which takes two points $f(\mathbf{x}_1), f(\mathbf{x}_2) \in \mathcal{M}$ on the manifold and returns a vector $\bar{\mathbf{v}} = \mathbf{J}_{\mathbf{x}_1}\mathbf{v} \in \mathcal{T}_{f(\mathbf{x}_1)}\mathcal{M}$ in the tangent space associated with $\mathbf{x}_1$, i.e. $\mathbf{v} = \text{Log}_{f(\mathbf{x}_1)}(f(\mathbf{x}_2))$. The inverse operator, instead, is *exponential map* denoted as $\text{Exp}_{f(\mathbf{x})}(\bar{\mathbf{v}}) : \mathcal{M} \times \mathcal{T}_{f(\mathbf{x})}\mathcal{M} \to \mathcal{M}$, which given a point $f(\mathbf{x}_1) \in \mathcal{M}$ and a vector $\bar{\mathbf{v}}$ in its tangent space, returns a point $f(\mathbf{x}_2) \in \mathcal{M}$. The exponential map can be interpreted as returning the endpoint of the geodesic obtained by starting at $f(\mathbf{x}_1)$ with velocity $\mathbf{v}$ for unit time. Therefore, we can see that the logarithmic map involves solving (2.85) as a boundary value problem with boundary conditions given by $c(0) = \mathbf{x}_1$ and $c(1) = \mathbf{x}_2$, while the exponential map involves an initial value problem where boundary conditions are given in terms of initial point $c(0) = \mathbf{x}_1$ and initial velocity $\dot{c}(0) = \mathbf{v}$. Although we have defined these operations in an extrinsic way, their computation can be done equivalently on the intrinsic coordinated Ω as we have already mentioned for the previous quantities we have presented.

Remark: Relation between energy and length of a curve $c(t)$. We are going to present how to use the Cauchy-Schwarz inequality to get the relationship between energy and length of a curve. We first introduce the notion of inner-product between two curves $u, v : [0, 1] \to \mathbb{R}$ and the norm of a curve:

$$\langle u, v \rangle = \int_0^1 u_t v_t \mathrm{d}t \qquad \text{(inner product)}$$

$$\|u\| = \sqrt{\langle u, u \rangle} = \sqrt{\int_0^1 u_t^2 \mathrm{d}t} \qquad \text{(norm)}$$

If we start by defining the following functions $x(t) = \|\dot{c}(t)\|_{c(t)}$ and $y(t) = 1$, then we can rewrite the length of a curve as

$$\text{Length}_{\mathcal{M}}[c] = \int_0^1 \|\dot{c}(t)\|_{c(t)} \mathrm{d}t = \underbrace{\int_0^1 x(t)y(t)\mathrm{d}t}_{\langle x,y \rangle}, \tag{2.86}$$

where if we consider two curves defined as $x = \int_0^1 x(t)\mathrm{d}t$ and $y = \int_0^1 y(t)\mathrm{d}t$, the last term can be seen as an inner-product between two curves. Therefore, we can apply the Cauchy-Schwarz inequality, stating that $< u, v > \leq \|u\|\|v\|$:

$$\mathrm{Length}_{\mathcal{M}}[c] \leq \sqrt{\int_0^1 x^2(t)\mathrm{d}t} \cdot \sqrt{\int_0^1 y^2(t)\mathrm{d}t} = \sqrt{\int_0^1 \|\dot{c}(t)\|^2_{c(t)}\mathrm{d}t} = \sqrt{\mathrm{E}_{\mathcal{M}}[c(t)]}. \tag{2.87}$$

Since $\mathrm{Length}_{\mathcal{M}}[c] > 0$, we can the square on both sides and get the inequality we have presented in the main text:

$$\mathrm{Length}^2_{\mathcal{M}}[c] \leq \mathrm{E}_{\mathcal{M}}[c(t)]. \tag{2.88}$$

From the definition of the Cauchy-Schwarz inequality, we know that the inequality becomes an equality if and only if the two functions $x(t)$ and $y(t)$, as we have defined above, are proportional to each other, that is, $x(t) = \alpha y(t)$. By substituting the definition of $x(t)$ and $y(t)$, we get $\|\dot{c}(t)\|_{c(t)} = \alpha$, which means that we can get the equality if and only if the curve velocity is constant.

CHAPTER 3

Model-agnostic out-of-distribution detection using combined statistical tests

This chapter presents our paper "Model-agnostic out-of-distribution detection using combined statistical tests". This is a self-contained research study, and the information provided in Chapter 2 should help make this study more accessible. The paper addresses the problem of deep generative models sometimes assigning a higher likelihood to out-of-distribution examples. We consider Rao's score test and the typicality test, and we show that they are complementary and propose to combine them using Fisher's method. Before getting into the paper, we would like to revisit some of the concepts introduced in it and elaborate them a bit more. We also plan to explain in detail the Benjamini-Hochberg procedure that we use in the paper, but we do not really explain it in detail.

Combining different statistical tests. The paper tackles the problem of deep generative models sometimes assigning higher likelihoods to examples that come from a different distribution than the one they were trained on (Nalisnick et al., 2018; Hendrycks et al., 2019). Therefore, we cannot rely on a threshold for the likelihood of performing out-of-distribution (OOD) or anomaly detection. Since (Nalisnick et al., 2018; Hendrycks et al., 2019)., different test statistics or methodologies for OOD detection using deep generative models were proposed, however, (Zhang et al., 2021) proved that in the case of single-sample OOD detection, there is no test statistic that is constantly better than all the possible alternatives. We propose to combine different statistical tests together using Fisher's method, as we are going to explain in the paper. We want to emphasize that both tests are testing for a similar null hypothesis $\mathcal{H}_0$, i.e. the datapoint is in-distribution. The combined test then can behave similarly to either a logical AND or a logical OR. The procedure proposed in this paper behaves similarly to a logical AND, meaning that the combined test likely accepts the null hypothesis if both tests accept it, while we reject it if one of the tests has rejected it.

Failure cases of typicality and the score. In the paper we present two cases where the typicality statistic and the score statistic fail or succeed, respectively. We assume that the in-distribution data come from the following multivariate Gaussian distribution $\mathrm{N}(\mathbf{0}, \mathbf{I}_D)$ and we fit a model $(\mathrm{N}(\theta, \mathbf{I}_D))_{\theta \in \mathbb{R}^D}$ vie MLE or MAP. In this setting, we can compute the typicality statistic and the score statistic in closed form. By recalling that the log-likelihood of $\mathrm{N}(x|\theta, \mathbf{I}_D)$ is given by $-\frac{D}{2}\log 2\pi - \frac{1}{2}\|x-\theta\|_2^2$, we can then write the score and the typicality kernel as:

$$
\begin{aligned}
\Phi_{\text{Fisher}}(x) &= I(\theta)^{-\frac{1}{2}} \nabla_\theta \log p_\theta(x) \\
&= I(\theta)^{-\frac{1}{2}} \nabla_\theta \big\{ -\frac{D}{2}\log 2\pi - \frac{1}{2}\|x-\theta\|_2^2 \big\} \\
&= I(\theta)^{-\frac{1}{2}} (x-\theta), \\
\Phi_{\text{Typicality}}(x) &= \log p_\theta(x) \\
&= -\frac{D}{2}\log 2\pi - \frac{1}{2}\|x-\theta\|_2^2
\end{aligned}
$$

To simplify the results we assume to have access to an infinite training set, i.e. $x_1, \ldots, x_m$ with $m \to \infty$. Therefore, the estimate of θ found during training using the MLE or MAP estimate will be exactly zero, i.e. $\theta \approx 0$. We can also write the Fisher Information matrix $I(\theta)$ in closed form:

$$
\begin{aligned}
I(\theta) &= \mathbb{E}_{\mathrm{N}(\mathbf{0}, \mathbf{I}_D)}[\nabla_\theta \log p_\theta(x) \nabla_\theta \log p_\theta(x)^T] \\
&= \mathbb{E}_{\mathrm{N}(\mathbf{0}, \mathbf{I}_D)}[x x^T] \\
&= \mathbf{I}_D
\end{aligned}
$$

By putting everything together, we can rewrite the kernels as:

$$
\begin{aligned}
\Phi_{\text{Fisher}}(x) &= \mathbf{I}_D x, \\
\Phi_{\text{Typicality}}(x) &= -\frac{D}{2}\log 2\pi - \frac{1}{2}\|x\|_2^2.
\end{aligned}
$$

We are then interested in testing if the n test samples $\tilde{x}_1, \ldots, \tilde{x}_n$ come from the same distribution as the data used to train θ or not. Performing maximum-mean-

discrepancy using the Fisher kernel is equivalent to:

$$\begin{aligned}
\mathrm{MMD}_{\Phi_{\text{Fisher}}}\left(\frac{1}{m}\sum_{i=1}^{m} x_i, \frac{1}{n}\sum_{i=1}^{n}\tilde{x}_i\right) &= \left\| \frac{I(\theta)^{-\frac{1}{2}}}{m}\sum_{i=1}^{m}\nabla_\theta \log p_\theta(x_i) - \frac{I(\theta)^{-\frac{1}{2}}}{n}\sum_{i=1}^{n}\nabla_\theta \log p_\theta(\tilde{x}_i)\right\|_2 \\
&= \left\| \frac{\mathbf{I}_D}{m}\sum_{i=1}^{m} x_i - \frac{\mathbf{I}_D}{n}\sum_{i=1}^{n}\tilde{x}_i \right\|_2 \\
&= \left\| \mathbb{E}_{x\sim \mathrm{N}(\mathbf{0},\mathbf{I}_D)}[x] - \frac{1}{n}\sum_{i=1}^{n}\tilde{x}_i \right\|_2 \\
&= \left\| -\frac{1}{n}\sum_{i=1}^{n}\tilde{x}_i \right\|_2 = \left\| -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{D}\tilde{x}_{ij} \right\|_2
\end{aligned}$$

where we expressed the first term as an expected value, and since $m \to \infty$ we see that this is exactly 0 and in the last two lines we factorized the sum also along the dimensionality axis. We can follow the same reasoning also to analyze the MMD with the typicality kernel. This becomes:

$$\begin{aligned}
\mathrm{MMD}_{\Phi_{\text{Typicality}}}\left(\frac{1}{m}\sum_{i=1}^{m} x_i, \frac{1}{n}\sum_{i=1}^{n}\tilde{x}_i\right) &= \left\| \frac{1}{m}\sum_{i=1}^{m}\log p_\theta(x_i) - \frac{1}{n}\sum_{i=1}^{n}\log p_\theta(\tilde{x}_i)\right\|_2 \\
&= \left\| -\mathbb{E}_{x\sim \mathrm{N}(\mathbf{0},\mathbf{I}_D)}[-\log p_\theta(x_i)] - \frac{1}{n}\sum_{i=1}^{n}\log p_\theta(\tilde{x}_i)\right\|_2 \\
&= \left\| -\frac{D}{2}\log(2\pi) - \frac{D}{2} - \frac{1}{n}\sum_{i=1}^{n}\left[-\frac{D}{2}\log(2\pi) - \frac{1}{2}\|\tilde{x}_i\|_2^2\right]\right\|_2 \\
&= \left\| -\frac{D}{2} + \frac{1}{2n}\sum_{i=1}^{n}\|\tilde{x}_i\|_2^2 \right\|_2 \\
&= \left\| -\frac{D}{2} + \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{D}\tilde{x}_{ij}^2 \right\|_2
\end{aligned}$$

where the first term can be seen as the negative entropy, i.e. $\mathbb{E}_{x\sim \mathrm{N}(\mathbf{0},\mathbf{I}_D)}[-\log p_\theta(x_i)] = \mathbb{H}[\mathrm{N}(\mathbf{0},\mathbf{I}_D)] = \frac{D}{2}\log(2\pi) + \frac{D}{2}$, while in the second term we substitute the closed form solution for the kernel we derived above. As before, we also factorize the sum along the different dimensions.

Now that we have a closed form for both kernels, we can analyze their behaviors in the two examples we mentioned in the paper in a more clear way. We are interested in analyzing how $\frac{1}{D}(T^{\text{ood}} - T^{\text{id}})$ behaves when the dimensionality increases to infinity

$D \to \infty$. In fact, in this regime, the statistics become deterministic. Since the in-distribution $\mathrm{N}(\mathbf{0}, \mathbf{I})$ is the same for both examples, we can start by computing $T^{\text{id}}_{\text{score}}/D$ and $T^{\text{id}}_{\text{typicality}}/D$.

$$\begin{aligned}\frac{T^{\text{id}}_{\text{score}}}{D} &= \frac{1}{D}\left\| -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{D}\tilde{x}_{ij}\right\|_2 \\ &= \left\| -\frac{1}{n}\frac{1}{D}\sum_{i=1}^{n}\sum_{j=1}^{D}\tilde{x}_{ij}\right\|_2 \\ &\xrightarrow[D\to\infty]{} \left\|\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[\tilde{x}_i]\right\|_2 = \left\|\frac{1}{n}n\mathbb{E}[\tilde{x}]\right\|_2 = 0,\end{aligned}$$

where we used the fact that $\frac{1}{D} > 0$ and brought inside the norm and analyzed the statistics in the very high-dimensional regime, that is, $D \to \infty$, by applying the law of large numbers. In the last step, we used the fact that the mean of $\mathrm{N}(\mathbf{0}, \mathbf{I}_D)$ is 0. The typicality, instead, is given by

$$\begin{aligned}\frac{T^{\text{id}}_{\text{typicality}}}{D} &= \frac{1}{D}\left\| -\frac{D}{2} + \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{D}\tilde{x}_{ij}^2\right\|_2 \\ &= \left\| -\frac{1}{2} + \frac{1}{2n}\frac{1}{D}\sum_{i=1}^{n}\sum_{j=1}^{D}\tilde{x}_{ij}^2\right\|_2 \\ &\xrightarrow[D\to\infty]{} \left\| -\frac{1}{2} + \frac{1}{2n}\sum_{i=1}^{n}\mathbb{E}[\tilde{x}_i^2]\right\| = \left\| -\frac{1}{2} + \frac{1}{2n}n\mathbb{E}[\tilde{x}^2]\right\|_2 = 0,\end{aligned}$$

since for $\mathrm{N}(\mathbf{0}, \mathbf{I}_D)$, we have that $\mathbb{E}[\tilde{x}^2] = \mathrm{Var}(x) + \mathbb{E}[x] = 1$.

We can now compute the same quantities for out-of-distribution detection. In the first example, we consider to have two OOD samples $\tilde{x}_1, \tilde{x}_2$ coming from a truncated normal distribution[1] defined as $\mathrm{N}(\tilde{x}|\mathbf{0}, \mathbf{I}_D) \cdot \mathbf{1}\{\tilde{x}^{(1)} > 0, \ldots, \tilde{x}^{(D)} > 0\}$. If we define the mean of this distribution as μ_{TD}, then $T^{\text{ood}}_{\text{score}}/D$ is given by

$$T^{\text{ood}}_{\text{score}}/D = \left\|\frac{1}{n}n\mathbb{E}[\tilde{x}]\right\|_2 = \sqrt{\mu^2_{\text{TD}}}.$$

For the typicality test, instead, we have that $\mathbb{E}[\tilde{x}^2] = 1$, since the truncation only affects the sign of the sample and therefore it has no effect when computing the

[1] Our definition of truncated normal distribution is also referred to as half-normal distribution.

square[2]. Therefore, we can see that for the first example, we have

$$\frac{1}{D}(T^{\text{ood}}_{\text{score}} - T^{\text{id}}_{\text{score}}) = \sqrt{\mu^2_{\text{TD}}} - 0 = \sqrt{\mu^2_{\text{TD}}}$$
$$\frac{1}{D}(T^{\text{ood}}_{\text{typicality}} - T^{\text{id}}_{\text{typicality}}) = 0 - 0 = 0,$$

that the typicality is failing in detecting the samples as OOD, while the score, on the other hand, succeeds.

In the second example, instead, we have considered the out-of-distribution to be the Dirac distribution with mean 0. In this case, we have $\mathbb{E}[\tilde{x}] = 0$ and $\mathbb{E}[\tilde{x}^2] = 0$. Therefore, we have the quantities we are interested in and behave as follows:

$$\frac{1}{D}(T^{\text{ood}}_{\text{score}} - T^{\text{id}}_{\text{score}}) = 0 - 0 = 0$$
$$\frac{1}{D}(T^{\text{ood}}_{\text{typicality}} - T^{\text{id}}_{\text{typicality}}) = \frac{1}{2} - 0 = \frac{1}{2},$$

meaning that in this case, the typicality is succeeding in detecting the samples as OOD, while the score is failing.

While above we have derived everything step by step, we can also try to explain the different results in the two examples using intuitions. Generally speaking, we can interpret $\text{MMD}_{\Phi_{\text{Fisher}}}$ as comparing the empirical mean of the training set and the set of examples that we want to classify as inliers or outliers. Since the in-distribution data follows a standard normal, the first term of the MMD is exactly zero. The $\text{MMD}_{\Phi_{\text{Typicality}}}$, on the other hand, can be seen as comparing the average squared norm of the two. In both cases, if the samples come from the same distribution, we will expect the MMD to be zero. In the first example, the OOD samples come from a truncated normal distribution defined as $\text{N}(\tilde{x}|\mathbf{0}, \mathbf{I}_D) \cdot \mathbf{1}\{\tilde{x}^{(1)} > 0, \dots, \tilde{x}^{(D)} > 0\}$. While for the in-distribution data points the mean is clearly 0, for these two OOD samples the mean is actually one of the truncated normal, and therefore the $\text{MMD}_{\Phi_{\text{Fisher}}} > 0$ indicating that the two samples are OOD. The average norm of the samples, instead, is the same as that of one of the in-distribution samples, because although their mean is different, they are still filtered samples from $\text{N}(\tilde{x}|\mathbf{0}, \mathbf{I}_D)$. As we have explained before, both samples will be in a shell of radius $\sqrt{D}$ from the center, and therefore the average norm is the same. Therefore, we get $\text{MMD}_{\Phi_{\text{Typicality}}} > 0$ indicating that these examples are in-distribution. This explains why the score statistic succeeds while the typicality fails.

[2]If we consider $X \sim \text{N}(0, 1)$, the samples of the half-normal distribution can be obtained by defining the following random variable $Y = |X|$. The mean of this new distribution is given by $\mu_{\text{TD}} = \frac{\sigma\sqrt{2}}{\sqrt{\pi}} = \frac{\sqrt{2}}{\sqrt{\pi}}$ while the variance is equal to $\text{Var}(Y) = \sigma^2\left(1 - \frac{2}{\pi}\right) = \left(1 - \frac{2}{\pi}\right)$. Therefore, we can see that $\mathbb{E}[Y^2] = \text{Var}(Y) + \mu^2_{\text{TD}} = 1 - \frac{2}{\pi} + \frac{2}{\pi} = 1$, as in the case of the standard normal distribution.

In the second example, instead, the out-of-distribution is a Dirac distribution with mean 0 and we assume that we have only one sample to evaluate. In this case, the mean of this sample is obviously zero as the one of the in-distribution data. The $\text{MMD}_{\Phi_{\text{Fisher}}}$ this will be 0, stating that the sample is in-distribution. The $\text{MMD}_{\Phi_{\text{Typicality}}}$, on the other hand, will be bigger than 0 since the norm of the OOD sample is exactly zero, in contrast to that of the in-distribution data. Therefore, in this case, we see that the typicality succeeds while the score fails.

Using the Stein score as a kernel for MMD. Grathwohl et al. (2019) proposed instead to use the norm of the Stein score as a score for OOD detection:

$$\mathcal{T}_{\text{Stein}}(x) = -\left\| \frac{\partial \log p_\theta(x)}{\partial x} \right\|_2 = -\left\| \nabla_x \log p_\theta(x) \right\|_2 .$$

We will show that this is the same as performing MMD with a kernel that is the Stein score. If we define the following kernel $\Phi_{\text{Stein}} = \nabla_x \log p(x)$, then by considering as before an infinite training set $x_1, \ldots, x_m$ with $m \to \infty$ and a test set with n elements $\tilde{x}_1, \ldots, \tilde{x}_n$. Then we can write the MMD with this new kernel as:

$$\begin{aligned}
\text{MMD}_{\Phi_{\text{Stein}}}\left(\frac{1}{m}\sum_{i=1}^{m} x_i, \frac{1}{n}\sum_{i=1}^{n} \tilde{x}_i \right) &= \left\| \frac{1}{m}\sum_{i=1}^{m} \nabla_x \log p_\theta(x_i) - \frac{1}{n}\sum_{i=1}^{n} \nabla_x \log p_\theta(\tilde{x}_i) \right\|_2 \\
&= \left\| \mathbb{E}_{p(x)}[\nabla_x \log p_\theta(x)] - \frac{1}{n}\sum_{i=1}^{n} \nabla_x \log p_\theta(\tilde{x}_i) \right\|_2 .
\end{aligned}$$

We can rewrite the expected value of the Stein score as:

$$\begin{aligned}
\mathbb{E}_{p(x)}[\nabla_x \log p_\theta(x)] &= \int_{\mathcal{X}} p(x) \log p_\theta(x) \mathrm{d}x = \int_{\mathcal{X}} \frac{\nabla_x p(x)}{p(x)} p(x) \mathrm{d}x \\
&= \int_{\mathcal{X}} \nabla_x p(x) \mathrm{d}x = p(\infty) - p(-\infty) = 0
\end{aligned}$$

where we used the log-derivative trick given by: $\nabla \log p(x) = \frac{\nabla_x p(x)}{p(x)}$. Therefore, the expected Stein score on the training set is 0. Due to this result, we can write $\text{MMD}_{\Phi_{\text{Stein}}}$ as

$$\begin{aligned}
\text{MMD}_{\Phi_{\text{Stein}}}\left(\frac{1}{m}\sum_{i=1}^{m} x_i, \frac{1}{n}\sum_{i=1}^{n} \tilde{x}_i \right) &= \left\| \mathbb{E}_{p(x)}[\nabla_x \log p_\theta(x)] - \frac{1}{n}\sum_{i=1}^{n} \nabla_x \log p_\theta(\tilde{x}_i) \right\|_2 \\
&= \left\| -\frac{1}{n}\sum_{i=1}^{n} \nabla_x \log p_\theta(\tilde{x}_i) \right\|_2 ,
\end{aligned}$$

which apart from the sign, is measuring the same thing as the $\mathcal{T}_{\text{Stein}}(x)$ proposed by Grathwohl et al. (2019).

Benjamini-Hochberg procedure. Framing the out-of-distribution problem as multiple tests has the benefit that we can now use several statistical tools to control the False Discovery Rate, i.e. controlling the number of inliers that are classified as outliers. In the paper, we investigate the use of the Benjamini-Hochberg correction (Benjamini & Hochberg, 1995). Here, we will explain briefly how this procedure works. Assume as before that we have a set of examples $\tilde{x}_1, \ldots, \tilde{x}_n$ that can either be in-distribution or out-of-distribution. We will use our approach to test for the following null hypothesis $\mathcal{H}_0 : \tilde{x}_1, \ldots, \tilde{x}_n \sim p_{\text{data}}$ resulting in n p-values $p_1, \ldots, p_n$, one for each hypothesis we are testing. Then, Benjamini-Hochberg works as follows:

- Order the p-values in ascending order, resulting in $p_{(1)}, \ldots, p_{(n)}$;
- Define $L = \max\{j : p_{(j)} < \frac{\alpha j}{n}\}$, where n is the total number of hypotheses and α is the significance level, i.e. the level or percentage of false discoveries we would like to have and therefore we can choose the value of α. Therefore, L represents the largest index of the ordered p-values that satisfies the equation above.
- We reject all the hypotheses that have $p_i < p_{(L)}$

Model-agnostic out-of-distribution detection using combined statistical tests

Federico Bergamin[*,1], **Pierre-Alexandre Mattei**[*,2], **Jakob D. Havtorn**[1,3], **Hugo Senetaire**[1]
Hugo Schmutz[2,4], **Lars Maaløe**[1,3], **Søren Hauberg**[1], **Jes Frellsen**[1]
[1]Technical University of Denmark [2]Université Côte d'Azur, Inria, LJAD, CNRS [3]Corti AI [4]TIRO, CEA

Abstract

We present simple methods for out-of-distribution detection using a trained generative model. These techniques, based on classical statistical tests, are model-agnostic in the sense that they can be applied to any differentiable generative model. The idea is to combine a classical parametric test (Rao's score test) with the recently introduced typicality test. These two test statistics are both theoretically well-founded and exploit different sources of information based on the likelihood for the typicality test and its gradient for the score test. We show that combining them using Fisher's method overall leads to a more accurate out-of-distribution test. We also discuss the benefits of casting out-of-distribution detection as a statistical testing problem, noting in particular that false positive rate control can be valuable for practical out-of-distribution detection. Despite their simplicity and generality, these methods can be competitive with model-specific out-of-distribution detection algorithms without any assumptions on the out-distribution.

1 Introduction

The ability to recognise when data are anomalous, i.e. if they originate from a distribution different from that of the training data, is a necessary property for machine learning models for safe and reliable applications in the real world. Historically, Bishop (1994) proposed to use a one-sided threshold on the log-likelihoods of a learned model as a decision rule to identify outliers in a dataset. However, recently, Nalisnick et al. (2018); Hendrycks et al. (2019) showed that state-of-the-art deep generative models (DGMs) failed in this task, assigning higher a likelihood to out-of-distribution (OOD) data than in-distribution data. Most of the recent works focused on proposing new test statistics to alleviate the problem of using the plain likelihood, see Section 5 for details.

We believe that OOD detection should be formulated as statistical hypothesis testing (Nalisnick et al., 2019; Ahmadian and Lindsten, 2021; Haroush et al., 2021). Since the power of a single test depends on the out-distribution (Zhang et al., 2021), we propose to approach this problem by using a combination of multiple statistical tests. While the power of the combined test also depends on the out-distribution, we hypothesise that the combined test empirically will perform better, especially in situations where one of the statistics fails. Furthermore, the use of the statistical testing framework has several advantages. Since we obtain a p-value, it is more natural deciding on a threshold as this corresponds to the significance level. In addition to that, it also allow us to correct for the multiple comparisons problem when identifying outliers in a dataset by controlling the number of Type I errors through the false discovery rate (FDR).

In summary, our contributions are the following:

- We illustrate the benefits of combining multiple statistical tests to perform OOD detection with DGMs using well-established methods. This allows for a proper decision procedure to control the FDR in a real outlier detection setting.
- We revisit some proposed detection scores and highlight their alternative formulation as classical significance tests.
- Empirically we show the complementarity of the typicality and the score statistics and that their combination leads to a robust score for anomaly detection.

Proceedings of the 25[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.

2 Using statistical tests for out-of-distribution detection

We consider some data of interest that live in a space $\mathcal{X}$. Assume that we have a curated dataset $x_1, \dots, x_m$, i.e. there are no outliers, and we are interested in understanding if some new data $\tilde{x}_1, \dots, \tilde{x}_n$ are collectively anomalies. In other words, we wonder whether or not $\tilde{x}_1, \dots, \tilde{x}_n$ are likely to come from the same distribution that generated our curated dataset. We present in this section two different approaches for doing out-of-distribution detection using statistical tests: one based on classical parametric tests and one based on maximum mean discrepancy. A convenient property of the tests we consider is that they are all one-sided, which means we can expect them to be larger when the data are more likely to be OOD. This allows us to compute p-values by simply using the empirical CDF, which is hyperparameter-free.

Note that in this problem formulation, the case $n = 1$ corresponds to the situation where we need to decide if a *single* data point is out-of-distribution. This hardest setting will be of particular interest, and this is also the main focus of recent work, see Section 5.

2.1 Parametric tests for out-of-distribution detection

The typical approach is to consider a parametric family $(p_\theta)_{\theta \in \Theta}$ of probability densities over $\mathcal{X}$ and learn a suitable $\theta_0 \in \Theta$ using any inference technique, for example maximum likelihood, and the clean data $x_1, \dots, x_m$. Depending on the input domain, $(p_\theta)_{\theta \in \Theta}$ could be composed of DGMs (in that case, θ would be neural network weights) or Gaussian mixture models (in that case, θ would be composed of means, covariances, and proportions). The question we wish to answer may then be phrased: *is p_{θ_0} an appropriate model for $\tilde{x}_1, \dots, \tilde{x}_n$?*

We choose to formalize this problem as a *parametric test* whose alternative hypothesis is that $\tilde{x}$ is *out-of-distribution*. More specifically, if we assume that $\tilde{x}_1, \dots, \tilde{x}_n \sim_{\text{i.i.d.}} p_{\tilde{\theta}}$ for some unknown $\tilde{\theta} \in \Theta$, we wish to test $\mathcal{H}_0 : \tilde{\theta} = \theta_0$ against $\mathcal{H} : \tilde{\theta} \neq \theta_0$, where the alternative hypothesis $\mathcal{H}$ is that the test points are OOD.

Many tests have been proposed for this purpose. The three most famous are the *likelihood ratio test* of Neyman and Pearson (1928), Rao's (1948) *score test*, and *the Wald test* (Wald, 1943). These three classics are nicely reviewed by Buse (1982) or by Rao (2005), who called them the "Holy Trinity". A recent and interesting one is the *gradient test* of Terrell (2002), which is reviewed in great detail in Lemonte's (2016) monograph.

Let us review the statistics of these four tests:

- likelihood ratio statistic is $S_{LR} = 2(\ell(\hat{\theta}) - \ell(\theta_0))$,
- Wald statistic is $S_W = (\hat{\theta} - \theta_0)^T I(\hat{\theta})(\hat{\theta} - \theta_0)$,
- score statistic is $S_S = \nabla\ell(\theta_0)^T I(\theta_0)^{-1} \nabla\ell(\theta_0)$,
- gradient statistic is $S_G = \nabla\ell(\theta_0)^T(\hat{\theta} - \theta_0)$,

where $\ell(\theta) = \log p_\theta(\tilde{x}_1, \dots, \tilde{x}_n)$ is the likelihood function, $I(\theta) = \mathbb{E}_{p_\theta}[\nabla\ell(\theta)\nabla\ell(\theta)^T]$ is the Fisher information matrix (FIM), and $\hat{\theta} \in \arg\max_{\theta \in \Theta} \ell(\theta)$.

The likelihood ratio statistic, the Wald statistic and the gradient statistic all require to fit a model on the additional datapoints $\tilde{x}_1, \dots, \tilde{x}_n$ in order to compute either $\ell(\hat{\theta})$ or $\hat{\theta}$. In our setting, if we want to use one of those statistics as an OOD score for a single example, we should fit a DGM on that single datapoint. Xiao et al. (2020) did this for a variational autoencoder (VAE, Kingma and Welling, 2013; Rezende et al., 2014) by only re-fitting inference network (or encoder) to the additional example, which is a typical approach to dealing with out-of-sample data in VAEs, as argued by Cremer et al. (2018) and Mattei and Frellsen (2018). However, much of the recent works in the literature (Ren et al., 2019; Schirrmeister et al., 2020; Serrà et al., 2020) mainly focus on deriving different versions of what they call a likelihood ratio statistic.

We tried to derive a general way to compute both the Wald statistic and the gradient statistic, by computing $\hat{\theta}$ with a few steps of a gradient-based optimization algorithm initialized at θ_0, but this resulted in a very unstable update leading to computational issues (results not shown). Therefore, in this work we focus on studying the relevance of the score statistic for performing out-of-distribution detection since it is the only statistic that does not require fitting an additional model to the OOD data.

2.2 Maximum mean discrepancy for out-of-distribution detection

Another way of approaching out-of-distribution detection from a testing perspective is through a *two-sample test*. Denoting p_{data} the true training data distribution, the goal is to test $\mathcal{H}_0 : \tilde{x}_1, \dots, \tilde{x}_n \sim p_{\text{data}}$ against $\mathcal{H} : \tilde{x}_1, \dots, \tilde{x}_n \not\sim p_{\text{data}}$, where the alternative hypothesis $\mathcal{H}$ again is that the test points are OOD.

A popular way of building statistics for two-sample tests is to use a measure of distance between p_{data} and the distribution of $\tilde{x}_1, \dots, \tilde{x}_n$. The key idea here will be to use the trained generative model to build this measure of distance. To this end, we will use the

maximum mean discrepancy (MMD) of Gretton et al. (2012), which is a kernel-based measure of distance. Then, p_θ will be used to specify an appropriate kernel.

More specifically, given a kernel whose feature map is $\Phi : \mathcal{X} \to \mathcal{H}$, the MMD between two distributions P and Q over $\mathcal{X}$ is defined as

$$\mathrm{MMD}_\Phi(P, Q) = \|E_{X\sim P}[\Phi(X)] - E_{Y\sim Q}[\Phi(Y)]\|_\mathcal{H}. \quad (1)$$

In our context, the test statistics will be of the form

$$\mathrm{MMD}_\Phi\left(\frac{1}{m}\sum_{i=1}^{m}\delta(x_i), \frac{1}{n}\sum_{i=1}^{n}\delta(\tilde{x}_i)\right) = \left\|\frac{1}{m}\sum_{i=1}^{m}\Phi(x_i) - \frac{1}{n}\sum_{i=1}^{n}\Phi(\tilde{x}_i)\right\|_\mathcal{H}, \quad (2)$$

where Φ is a kernel feature map built using the generative model and $x_1, \ldots, x_m$ is the training data, i.e. samples from p_{data}. When $\mathcal{H}$ is a simple finite-dimensional Hilbert space and Φ can be computed easily, then (2) can be computed by going through the data and computing the means in an online fashion.

As always with kernel methods, a key question is how to choose the kernel, or its feature map Φ. Here, we want to use the trained generative model p_θ to build our kernel feature map Φ.

The Fisher kernel An important example of kernel based on a generative model is the *Fisher kernel* of Jaakkola and Haussler (1999). The embedding of this kernel is the Fisher score

$$\Phi_{\text{Fisher}}(x) = I(\theta)^{-\frac{1}{2}}\nabla \log p_\theta(x), \quad (3)$$

and the corresponding reproducing kernel Hilbert space norm is just the ℓ_2 norm: $\|\cdot\|_\mathcal{H} = \|\cdot\|_2$. In the case of the Fisher kernel, this means that Equation (2) becomes:

$$\mathrm{MMD}_{\Phi_{\text{Fisher}}}\left(\frac{1}{m}\sum_{i=1}^{m}\delta(x_i), \frac{1}{n}\sum_{i=1}^{n}\delta(\tilde{x}_i)\right) = \left\|\frac{I(\theta)^{-\frac{1}{2}}}{m}\sum_{i=1}^{m}\nabla \log p_\theta(x_i) - \frac{I(\theta)^{-\frac{1}{2}}}{n}\sum_{i=1}^{n}\nabla \log p_\theta(\tilde{x}_i)\right\|_2. \quad (4)$$

We will see later that MMD with a Fisher kernel is closely related to the score statistic. In Appendix B, we additionally show that another popular OOD metric known as the *Mahalanobis score* (Lee et al., 2018) can be interpreted as a MMD statistic with a certain Fisher kernel.

The typicality kernel A very simple approach of embedding the data using p_θ is to choose $\Phi_{\text{Typical}}(x) = \log p_\theta(x)$. Then, MMD is exactly equivalent to the *typicality test statistic* of Nalisnick et al. (2019), although this connection was not explicitly stated by Nalisnick et al. (2019). Because of this, we call the kernel $k(x, y) = \log p_\theta(x) \cdot \log p_\theta(y)$ the *typicality kernel*. While Φ_{Typical} is not as well motivated as a kernel as Φ_{Fisher}, the concepts of typicality and typical set can be used to explain unintuitive behaviours of probability distributions in high-dimensional space as highlighted by Nalisnick et al. (2018). We also found that using this kernel generally gives good results for OOD tasks. An interesting analysis that we d not consider in this paper would be to study the properties of this kernel.

In general, neither of these two kernels are characteristic, meaning that our MMD can be zero even if the distributions are not identical. This could be solved by combining them with a characteristic kernel, as in Liu et al. (2020), at the price of including a new hyperparameter.

3 Combining different test statistics

For single-sample OOD detection, Zhang et al. (2021) proved that there is not a single statistic that is constantly better compared to all the possible alternatives of interest. For this reason, we believe that using a combination of different test statistics should lead to an overall better OOD detection in settings where a single statistic might fail. Assume we compute k different test statistics $T_1, \ldots, T_k$, each testing $\mathcal{H}_0$ against $\mathcal{H}$ as defined in Sections 2.1 and 2.2. The goal is to combine these different tests into a single statistical test that ideally will perform better than the initial single tests. However, different tests can have different magnitudes and they can differ also in the direction of out-of-distribution detection, i.e. for some statistics having a higher values is associated with being OOD, while for other smaller values are OOD. This makes a combination non-trivial.

Morningstar et al. (2021) proposed the density of states estimator (DoSE) to overcome this problem. They only focused on the single-sample detection task, i.e. $n = 1$ following our problem formulation. Their idea is to fit different nonparametric density estimators, such as a kernel-density estimator (KDE) or a one-class support vector machines (SVM), for each different statistic $T_1, \ldots, T_k$ by using the values computed on the training set examples. For a single test example, $\tilde{x}_1$, they first compute $T_1, \ldots, T_k$ and then combine those statistics by summing the different KDEs log-density. While this approach can be used for

any type of statistic, and thus is more general, it uses less prior information. Indeed, if we use only statistics that are truly one-sided, then we assume that a method that leverages the true nature of the statistics should work better. In addition to that, fitting a KDE introduces an additional hyperparameter.

In our work, instead, we propose a different approach and leverage the fact that we use only one-sided test statistics. This setting is a well-studied problem in the literature both for independent (Fisher, 1925; Folks and Little, 1971) and dependent one-sided test statistics (Brown, 1975; Wilson, 2019). All these approaches rely on the computation of p-values of each statistic for the test set $\tilde{x}_1, \ldots, \tilde{x}_n$. This corresponds to computing $p_j = \Pr(T_j > t_j \mid \mathcal{H}_0)$, i.e. the probability that the j'th test is bigger than the observed value under the null hypothesis $\mathcal{H}_0$, where we assume that each T_j has a continuous distribution. Using p-values also solves the problem of the statistics having different scales. Indeed, p-values transform the different test statistics into the unit interval.

Computation of p-values We want to approximate the distribution of the p-values $p_1, \ldots, p_k$ of $\tilde{x}_1, \ldots, \tilde{x}_n$ under the null hypothesis $\mathcal{H}_0$. When $\mathcal{H}_0$ is true, then p_j is uniformly distributed on the interval $[0, 1]$. To succeed in this, we should be able to compute $p_j = \Pr(T_j > t_j \mid \mathcal{H}_0)$, therefore we need to estimate the distribution of each statistic T_j under $\mathcal{H}_0$. As done by Nalisnick et al. (2019), we assume the existence of a validation set $\mathbf{X}'$ that was not used to train our generative model. From $\mathbf{X}'$ we bootstrap S new datasets $\{\mathbf{X}'_s\}_{s=1}^{S}$ of size M' by using bootstrap resampling. When n is small, for example $n = 1$ or $n = 2$, where $n = 1$ corresponds to single-sample OOD detection, and the validation set is big, a convenient alternative to bootstrapping is to directly evaluate each test statistic T_j on every single validation example. Asymptotically, this is equivalent to creating S new datasets of size $M' = 1$ when $S \to \infty$. In case of $n = 2$, i.e. two-samples OOD detection, and a big validation set we can simply bootstrap without resampling. We then use these values to estimate the empirical distribution function (eCDF) of the considered statistic T_j under $\mathcal{H}_0$. To obtain the p-values of test examples $\tilde{x}_1, \ldots, \tilde{x}_n$ for the test statistic $T_j = t_j$, we simply compute $p_j = 1 - \Pr(T_j < t_j \mid \mathcal{H}_0)$ using the eCDF.

Combining test statistics by combining p-values Fisher's (1925) method is a procedure to combine different p-values $p_1, \ldots, p_k$. This method assumes that all the considered test statistics are independent, and Folks and Little (1971) proved that it is asymptotically optimal among all methods of combining independent tests. Given $T_1, \ldots, T_k$ and corresponding p-values $p_1, \ldots, p_k$, Fisher's method combines the p-values into a test statistic X^2 defined as

$$X^2 \sim -2 \sum_{j=1}^{k} \ln(p_j). \tag{5}$$

In case all null-hypotheses are accepted, the resulting test statistic X^2 follows a chi-squared distribution with $2k$ degrees of freedom. In the Appendix D.2 , we also consider the Harmonic mean p-value (Wilson, 2019) as a way to combine p-values from different statistics. This method usually works best when the statistics are not independent.

4 From test statistics to practical out-of-distribution scores

Several of the test statistics that we consider make use of the inverse of the Fisher information matrix $I(\theta)$. The true Fisher information matrix requires an identifiable model to be invertible (Watanabe, 2009) and computing its inverse is $\mathcal{O}(m^3)$, where m is the number of model parameters. For DGMs, the Fisher information matrix might not be invertible due to the fact that DGMs typically do not satisfy the identifiability condition. Also, the inversion may be computationally impractical, since state-of-the-art DGMs involve very high-dimensional parameter spaces Θ. For the same reason, storing $I(\theta)$ can also be challenging.

We replace it by using a proxy matrix that has to be easy to compute and invert. A first idea is to simply replace $I(\theta)$ by the identity matrix. A more refined way is to look for a diagonal approximation. In Appendix A, we describe cheap ways of computing such approximations. In particular, we will study two cases: the case where $I(\theta)$ is replaced by the identity matrix and the case where $I(\theta)$ is replaced by a diagonal matrix estimated using the training data.

A possible third option would be to estimate the diagonal of $I(\theta)$ using samples from the model. However, for autoregressive models as the PixelCNN, sampling is a sequential procedure and therefore it is computationally expensive to generate many samples when the input-space is high-dimensional. For this reason, we do not consider it in this work. More complex and precise approximations of the FIM exists, such as the Kronecker-factored Approximate Curvature (K-FAC, Martens and Grosse, 2015), but these are not defined for all types of layers used by state-of-the-art models.

On the difficulty of computing per-example gradients Both the diagonal approximation of the FIM and the computation of the MMD with Fisher kernel of

Equation (4) require the gradient computation for all training and test examples. This is known as a costly procedure. For example, if we have to compute the gradient for N examples using a simple fully connected network with l layers of size p, the naive procedure of using a batch-size of dimension 1 is $\mathcal{O}(Nlp^2)$ (Goodfellow, 2015). While more efficient per-example gradient computations were proposed (Goodfellow, 2015; Rochette et al., 2019), these techniques can only be applied on simple fully connected or convolutional networks. While for this paper we relied on the naive solution of looping through every example one at the time, a more efficient solution is provided by the BackPACK library (Dangel et al., 2020) which allows to compute the gradient with respect each sample in a minibatch.

4.1 Relationship between MMD with Fisher kernel and the score statistic and gradient norm

Depending on the choice of the Fisher information approximation, we can notice that there is a strong connection between the MMD using a Fisher kernel, the score statistic and the gradient norm in terms of expected OOD performance. Let us start by looking at the case where we approximate $I(\theta)$ with a diagonal matrix estimated using the training data. At the maximum likelihood estimate, we have that $\mathbb{E}[\nabla \log p_\theta(x)] = 0$, i.e. the first term inside the norm is 0. Therefore, we expect that the differences between the OOD scores computed by using Equation (4) will be preserved if we only consider $\left\| I(\theta)^{-1/2} \nabla \log p_\theta(\tilde{x}_1, \dots, \tilde{x}_n) \right\|_2$, which corresponds to the square root of the score statistic. Since taking the square root still preserves the difference between values, we can expect that the MMD using a Fisher kernel will perform closely to the score statistic. The same reasoning also holds in case we replace the FIM with an identity matrix. In this specific case, instead, we will get that $\left\| I(\theta)^{-1/2} \nabla \log p_\theta(\tilde{x}_1, \dots, \tilde{x}_n) \right\|_2 = \| \nabla \log p_\theta(\tilde{x}_1, \dots, \tilde{x}_n) \|_2$, which corresponds to considering the gradient norm.

Computationally speaking, considering the score statistic instead of the MMD Fisher lets us avoid going through the entire training set to compute the average gradient (first term in Equation (4)) while carrying the same information. Therefore, in this paper, we will mainly focus on the combination of the typicality test and the score statistic.

4.2 Why does it make sense to combine the score statistic and the typicality test?

Let us discuss our choice of combining the score statistic and the typicality test. We will try to look in which situations one of the test fails and the other works and vice versa. Both examples assume that the in-distribution data follows a $\mathcal{N}(0, I_D)$ distribution, and that the correct model has been learned by fitting $(\mathcal{N}(\theta, I_D))_{\theta \in \mathbb{R}^D}$ via maximum-likelihood. Even in this simple setting with no model misspecification, we will see that the two statistics that we consider may have very different strengths.

In this simple Gaussian case, the score statistic can be computed exactly and will be $||\tilde{x}_1 + \ldots + \tilde{x}_n||_2^2$. On the other hand, the typicality statistic will be $|(||\tilde{x}_1||_2^2 + \ldots + ||\tilde{x}_n||_2^2)/(2 \cdot n) - D/2|$. One interesting regime is the very high-dimensional one ($D \to \infty$). Indeed, by the law of large numbers, these random statistics become deterministic quantities.

Typicality fails, the score succeeds Assume that we have two independent OOD data samples that follow a product of truncated normal distributions, with density proportional to

$$\mathcal{N}(x|0, I_D) \cdot \mathbf{1}\{x_1 > 0, \dots, x_D > 0\}.$$

We denote by $T^{\text{ood}}_{\text{score}}, T^{\text{id}}_{\text{score}}$ and $T^{\text{ood}}_{\text{typicality}}, T^{\text{id}}_{\text{typicality}}$ the statistics obtained when confronted with either OOD data from the truncated normal, or the in-distribution data. While these statistics are random in general, they will become deterministic when $D \to \infty$, by virtue of the law of large numbers.

For the typicality statistic, these two OOD samples will be indistinguishable from Gaussian ones. Indeed, when $D \to \infty$, both $T^{\text{ood}}_{\text{typicality}}$ and $T^{\text{id}}_{\text{typicality}}$ will be $\mathcal{O}(D)$. On the other hand, for the score, one can show that

$$T^{\text{ood}}_{\text{score}} - T^{\text{id}}_{\text{score}} \sim D\mu_{\text{TN}}, \tag{6}$$

where $\mu_{\text{TN}} > 0$ is the mean of the truncated normal distribution.

Typicality succeeds, the score fails Let us now consider as the OOD distribution a Dirac distribution with mean 0. Suppose that we see a single sample from this distribution. In this case, the score statistic will be 0, and will therefore not detect that the point is actually OOD. However, when D is large, the typicality test will be able to declare that this point is anomalous, as shown by Nalisnick et al. (2019).

Therefore, we have that the typicality test and the score statistic are complementary and measure a different type of information. In Appendix D.1, we empirically show that they are not correlated, by plotting the two measures against each other and by computing the correlation matrix.

5 Related works

Since Nalisnick et al. (2018) and Hendrycks et al. (2019), different test statistics or methodologies for OOD detection using DGMs were proposed. Most of the recent solutions were highly influenced by three major lines of work: *typicality set*, *likelihood ratio* test statistics, and *model misestimation*.

The typicality set hypothesis was introduced by Nalisnick et al. (2019) as a possible explanation for the DGMs assigning higher likelihood to OOD data. The typicality set is the subset of the model full support where the model samples from and this does not intersect with the region of higher likelihood. While the typicality test was introduced for batch-OOD detection, Morningstar et al. (2021) shows that it also works well in the single-sample case. This is also confirmed by our own experiments.

The likelihood ratio test statistic method by Ren et al. (2019) assumes that every input is composed by a background component and a semantic component. For OOD detection, only the semantic component matters. In addition to a model trained on the in-distribution data, they proposed to train a background model on perturbed inputs data and then for each test example consider as OOD score the likelihood ratio between the two models. Schirrmeister et al. (2020), instead, trained the background model on a more general distribution of images by considering 80 million general tiny images. Similarly to these approaches, Serrà et al. (2020) argued that the failure of DGMs is due to the high-influence that the input complexity has on the likelihood. Therefore, they proposed to use a general lossless image compression algorithm as a background model. All these methods, however, require additional knowledge of the OOD data for either choosing an image augmentation procedure to perturb the input data or for choosing a specific compressor.

Another line of works blame the models themselves and not the test statistics. Zhang et al. (2021) argued that model misestimation is the main cause of higher likelihood assigned to OOD data. This can be due to both the model architecture and the maximum likelihood objective. Kirichenko et al. (2020) and Schirrmeister et al. (2020) showed that normalizing flows can achieve better OOD performance despite achieving a worse likelihood if one changes some model design choices. Other works in the literature focused on deriving specific test statistics that works only for a specific model, for example for VAEs (Xiao et al., 2020; Maaløe et al., 2019; Havtorn et al., 2021), or for normalizing flows (Kirichenko et al., 2020; Ahmadian and Lindsten, 2021).

As mentioned in the introduction, we frame the OOD detection problem in terms of statistical tests problem. Recently, Haroush et al. (2021) showed that adopting hypothesis testing at the layer and channel level of a neural network can be used for OOD detection in the discriminative setting. They used both Fisher's method and Simes' method to combine class-conditional p-values computed for each convolutional and dense layer of a deep neural network. We focus on the unsupervised setting using DGMs and use hypothesis testing on statistics that can be computed on all differentiable DGM. As already explained in section Section 3, Morningstar et al. (2021) considered the combination of different statistics for OOD detection. The main difference with their approach is that we propose statistics that can be applied to any differentiable generative model and combine them by using Fisher's method, which takes advantage of using only one-sided independent statistics. Concurrently, Choi et al. (2021) derived the score statistic by starting from the likelihood ratio statistic and applying a Laplace approximation. They computed the score statistic only for certain layers of the model and for a specific example, the OOD score is given by the infinity norm of these different layer scores after a ReLU operation. Our procedure differs both in the derivation of the score statistic and its usage since we compute the score statistic for the entire model.

6 Experimental Setup

To evaluate the performance of the combination of the typicality test and the score statistic in detecting OOD data, we follow the experiments of Nalisnick et al. (2018); Hendrycks et al. (2019) and considered the OOD detection task on three image dataset pairs that have been proven challenging for DGMs, i.e. FashionMNIST (Xiao et al., 2017) vs MNIST (LeCun, 1998), CIFAR10 (Krizhevsky, 2009) vs SVHN (Netzer et al., 2011), and CIFAR10 vs CIFAR100. Winkens et al. (2020) divide these tasks into *far*-OOD tasks, where the in-distribution and out-distribution are different such as in the case of CIFAR10 against SVHN, and *near*-OOD where the two distributions are pretty similar, such as CIFAR10 and CIFAR100. *Near*-OOD tasks are usually most challenging.

For each task, we trained three different state-of-the-art DGMs, a PixelCNN++ (Salimans et al., 2017), a Glow model (Kingma and Dhariwal, 2018), and a hierarchical variational autoencoder (Kingma and Welling, 2013; Rezende et al., 2014) with bottom-up inference (HVAE, Burda et al., 2016). These are DGMs parametrized by neural networks that make different assumptions in the modelling choice of the target distribution. In addition to that, for PixelCNN++ and

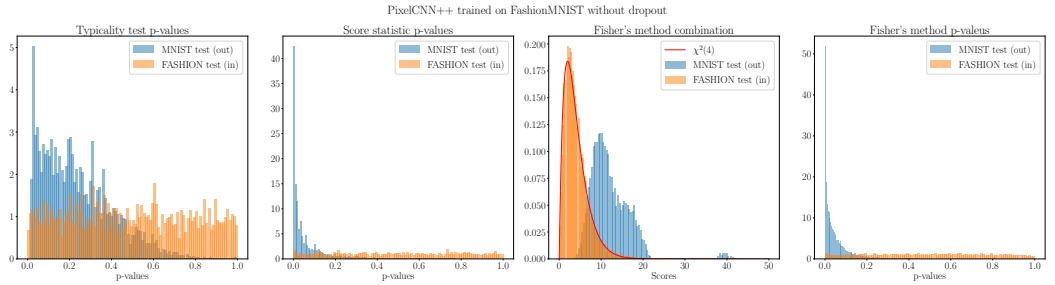


Figure 1: *First plot*: p-values of the typicality test on the two test sets. We can see that under $\mathcal{H}_0$, they should be uniformly distributed. *Second plot*: p-values of the score statistic. *Third plot*: values obtained by the Fisher's method. In red, we plot the density function of a χ^2-distribution with 4 degrees. This shows that the statistics are independent. *Fourth plot*: p-values obtained of the combination. These plots refer to a PixelCNN++ trained on FashionMNIST without dropout.

Glow we have a tractable likelihood while for HVAE we can only estimate a lower bound. A more in-depth description of these methods and additional results testing MNIST against FashionMNIST and SVHN against CIFAR10 can be found in Appendix D.6. We also extensively analyzed, focusing mostly in the influence of the preprocessing, the results on CIFAR10 vs CelebA (Liu et al., 2015) in Appendix E. In Appendix D.7, we also considered a Gaussian Mixture Model and a Probabilistic PCA as simple generative models.

Models To analyze the effect of model architecture choices and optimization choice, we also consider different versions of the same model that reaches a similar log-likelihood. We consider 5 different models for each dataset pair. On FashionMNIST, we consider two Glow models, one trained using Adam and one using RMSProp and two PixelCNN++, trained with and without dropout. For CIFAR10, we consider two different PixelCNN++, one trained by us (model1) and one using a checkpoint given by the repository we used[1] (model2), and two Glow models (Adam and RMSProp). For both datasets, instead, we consider only one HVAE.

Baselines We are mostly interested in testing our methods with other model-agnostic test statistics in the literature. Apart from using the plain likelihood as an OOD score, the only test statistic we are aware of that can be applied to any generative model without requiring any background model or OOD assumptions is the typicality test statistic of Nalisnick et al. (2019). We also considered the gradient norm, which in general seem to work well but fails in the case of SVHN vs CIFAR10 (see Appendix D.6). In addition to that, we compare our methods to a model-agnostic version of DoSE by Morningstar et al. (2021), where we used KDEs to combine the score statistic and the typicality test statistic.

Evaluation We compare our methods with the baselines by computing the area under the receiver operating characteristic curve (AUROC) as done in previous works (Hendrycks et al., 2019; Ren et al., 2019; Morningstar et al., 2021). We also evaluate our methods in terms of False Discovery Rate (FDR) control Benjamini and Hochberg (1995), i.e. the proportion of false positive among the rejected hypothesis. Note that both quantities need to know the true label (OOD or in-distribution) to be computed.

7 Results

One-sample OOD We first evaluate our proposed method in the single-sample OOD detection task. Results are summarized in Table 1. We start by considering the OOD task on FashionMNIST against MNIST. Looking at the single statistics, we notice that the score statistic is the one that works the best and the combination of the typicality test and the score statistic usually improve the AUROC than the two standalone statistics. In addition to that, it is better than the combination of the two statistics by using a KDE. DoSE seems to perform better on Glow trained with RMSProp, where the typicality is failing.

On natural images, instead, we have a different trend. The typicality test is better than the score statistic overall. The gradient norm surprisingly performs well in the two dataset pairs, but it fails badly when the model is trained on SVHN (see Appendix D.6). Regarding the combination of the two statistics, the

[1] `https://github.com/pclucas14/pixel-cnn-pp`

Table 1: AUROC↑ for single-sample OOD detection. For Fisher's method we mean the combination of the typicality test and the test statistic. These are also combined using DoSE.

	FashionMNIST (in) / MNIST (out)					
	single statistics				combination	
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
PixelCNN++ (dropout)	0.0762	0.8709	0.8314	**0.8822**	**0.9369**	0.8822
PixelCNN++ (no dropout)	0.1048	**0.9532**	0.7575	0.9381	**0.9536**	0.9382
Glow (RMSProp)	0.1970	0.8904	0.4807	**0.9114**	0.8598	**0.8901**
Glow (Adam)	0.1223	0.7705	0.6987	**0.8745**	**0.8839**	0.8752
HVAE	0.2620	0.8714	0.4884	**0.9578**	0.9383	**0.9498**

	CIFAR10 (in) / SVHN (out)					
	single statistics				combination	
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
PixelCNN++ (model1)	0.1553	**0.8006**	0.6457	0.6407	**0.6826**	0.6571
PixelCNN++ (model2)	0.1567	**0.7923**	0.6498	0.7067	**0.7300**	0.7243
Glow (RMSProp)	0.0630	0.8585	**0.8651**	0.7940	**0.8683**	0.8510
Glow (Adam)	0.0627	0.7844	**0.8624**	0.7655	**0.8613**	0.8588
HVAE	0.0636	0.8067	**0.8679**	0.7335	**0.8603**	0.8179

	CIFAR10 (in) / CIFAR100 (out)					
	single statistics				combination	
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
PixelCNN++ (model1)	0.5153	0.5306	**0.5458**	0.5362	**0.5563**	0.5477
PixelCNN++ (model2)	0.5150	0.5230	**0.5455**	0.5325	**0.5543**	0.5453
Glow (RMSProp)	0.5206	0.5547	0.5507	**0.5801**	**0.5844**	**0.5842**
Glow (Adam)	0.5206	0.5593	0.5508	**0.5692**	**0.5775**	**0.5767**
HVAE	0.5340	0.5280	0.5493	**0.5798**	0.5879	**0.5941**

Fisher's method is always better than DoSE, but in this setting, it improves over the best of the single statistics three out of five times. In the *near*-OOD task, we have that both our method and DoSE using our suggested statistics perform closely. We want to highlight that for this challenging task we get results that are comparable with those reported in Morningstar et al. (2021), but by using two model-agnostic statistics instead of three model-specific ones. It can be noticed that the way we train our models has a strong influence on both the typicality test and the score statistic, although the models get the same test log-likelihood. In Appendix D.4, we also show that this can happen between different checkpoints of the same model.

In Figure 1, we show that the p-values distributions for both the typicality and the score statistic are uniformly distributed under the null-hypothesis and that the combination under the null follows a χ^2 distribution with 4 degrees of freedom. This also supports the fact that the typicality test and the score statistic are independent.

Two-sample OOD As Nalisnick et al. (2019), we consider how these test statistics change when performing two-sample OOD detection. Results are summarized in Table 2. As shown by Nalisnick et al. (2019), the typicality improves but also the score statistic gets better if we consider more samples. Combining those leads to an improvement of performance in terms of AUROC with almost all the models. When training on FashionMNIST, the model can almost perfectly distinguish between the in-distribution test set and the OOD test set. While the performance improves for the two *far*-OOD task, we have that the improvement is slightly less evident in the *near*-OOD task of CIFAR10 vs CIFAR100.

7.1 Practical OOD detection with FDR control

One of the advantages of framing the problem as multiple testing is that we have a well-defined procedure to decide on which hypotheses to reject while controlling the False Discovery Rate (FDR, Benjamini and Hochberg, 1995). Imagine we are interested in finding the outliers from the dataset given by the combination of the two test-sets but we do not want to discard too many inliers, then we can use the Benjamini-Hochberg (BH) procedure (Benjamini and Hochberg, 1995) to decide a threshold and reject all hypothesis below that threshold. For a specific significance level

Table 2: AUROC↑ for two-sample OOD detection using the usual considered model.

FashionMNIST (in) / MNIST (out)				
models	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
PCNN++ (drop.)	0.9514	0.9828	**0.9934**	**0.9912**
PCNN++ (no drop)	0.9081	0.9853	**0.9916**	**0.9921**
Glow (RMSProp)	0.6190	**0.9588**	0.9187	0.7201
Glow (Adam)	0.8525	**0.9716**	**0.9708**	**0.9736**
HVAE	0.6634	**0.9881**	**0.9837**	**0.9889**

CIFAR10 (in) / SVHN (out)				
models	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
PCNN++ (m1)	0.7675	0.6555	**0.7800**	0.7046
PCNN++ (m2)	0.7720	0.7235	**0.8227**	0.7850
Glow (RMSProp)	0.9497	0.8624	**0.9536**	0.9379
Glow (Adam)	0.9480	0.8370	**0.9519**	0.9329
HVAE	**0.9623**	0.7754	0.9560	0.9133

CIFAR10 (in) / CIFAR100 (out)				
models	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
PCNN++ (m1)	0.5433	0.5450	**0.5540**	**0.5508**
PCNN++ (m2)	0.5435	0.5370	**0.5533**	0.5470
Glow (RMSProp)	0.5550	**0.6211**	0.6165	**0.6233**
Glow (Adam)	0.5558	0.6073	0.6083	**0.6117**
HVAE	0.5594	0.6188	**0.6218**	**0.6273**

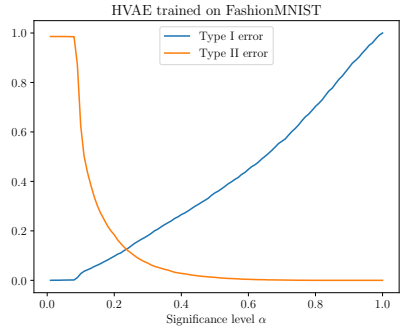


Figure 2: Type I (probability of an inlier to be classified as outlier) and Type II (probability of an outlier to be considered as inlier) errors versus the significance level α on the combination values. By using Benjamini-Hochberg correction, we get that the Type I error stays below identity line.

α, the procedure guarantees that the FDR stays below that level. Therefore, we can guarantee that the rate of inliers that are classified as outliers is less than the chosen α.

We leverage the fact that when the null hypothesis is true and the p-values are independent, then the scores obtained by combining k different statistics are χ^2_{2k} distributed to compute the p-values. Alternatively, the procedure can be also applied to the p-values of a single test-statistic. Usually, it is better to use a FDR control when it is actually possible to make few false discoveries, i.e. when we have a strong statistic. Therefore, we expect the procedure to work well when the AUROC is good, for examples on models trained on FashionMNIST.

As can be seen in Figure 2, we have that the Type I ratio line stays below the identity line, meaning that the BH correction is working. When deciding for a specific threshold α, we usually have to trade-off between Type I and Type II error and in most cases the threshold to choose depends on the application domain. Ideally, we would like to have a low Type I and a low Type II error rate, meaning that we are not considering a lot of in-distribution examples as OOD and at the same time considering a lot of outliers as in-distribution. Figure 2 shows that we can achieve this for low values of α. When training on CIFAR, instead, we are able to control the FDR only from a certain significance level (see Appendix D.5). This is expected given that the AUROC is not as good as when testing on MNIST.

8 Discussion and Conclusions

In this paper we studied the task of out-of-distribution detection using deep generative models and a combination of multiple statistical tests. We tested our method using different state-of-the art DGMs on classic image benchmark for OOD detection. We found that combining the two statistic leads to a more robust score that in some cases is close to state-of-the-art model-specific scores that require more assumptions. We also noticed that both the model design choice and the optimization choices have an influence on the score we are computing.

When considering only one-sided independent statistics, we showed that the Fisher's method tends to works better than combine them by summing the log-density of a KDE. We also noticed that the score statistic tends to perform a bit worse when the number of parameters of the models increases, i.e. in the context of natural images. One possible reason can be that in this setting the diagonal approximation is not good, and therefore one could consider different approximations, such as K-FAC.

DGMs have recently been used for handling missing data (see e.g. Mattei and Frellsen, 2019; Ma et al., 2019; Nazábal et al., 2020; Ipsen et al., 2021). An interesting future direction would be to extend these OOD detection methods to handle missing values.

The methods presented in this paper can also easily be applied when using model-specific one-sided statistics. In addition to obtain a more accurate score if one want to combine the test statistics, this also allows one to use well-defined procedure to control the FDR when choosing a which example to mark as outliers. Having this control, is necessary when we want to apply these methods in real settings.

Acknowledgements

Federico Bergamin and Pierre-Alexandre Mattei contributed eqaully to this paper, which is indicated by the asterisk (*) in the author list. The work was supported by the Innovation Fund Denmark (0175-00014B and 0153-00167B), the Independent Research Fund Denmark (9131-00082B) and the Novo Nordisk Foundation (NNF20OC0062606 and NNF20OC0065611). Furthermore, it was supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

References

A. Ahmadian and F. Lindsten. Likelihood-free Out-of-Distribution detection with invertible generative models. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2021.

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

C. M. Bishop. Novelty detection and neural network validation. *IEE Proceedings-Vision, Image and Signal processing*, 141(4):217–222, 1994.

M. B. Brown. 400: A method for combining non-independent, one-sided tests of significance. *Biometrics*, 1975.

Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. In *4th International Conference on Learning Representations, (ICLR), 2016*, 2016.

A. S. Buse. The likelihood ratio, Wald, and Lagrange multiplier tests: An expository note. *The American Statistician*, 1982.

J. Choi, C. Yoon, J. Bae, and M. Kang. Robust out-of-distribution detection on deep probabilistic generative models. *arXiv preprint arXiv:2106.07903*, 2021.

C. Cremer, X. Li, and D. Duvenaud. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning (ICML)*. PMLR, 2018.

F. Dangel, F. Kunstner, and P. Hennig. BackPACK: Packing more into backprop. In *International Conference on Learning Representations (ICLR)*, 2020.

R. Fisher. *Statistical methods for research workers*. Edinburgh Oliver & Boyd, 1925.

J. Folks and R. Little. Asymptotic optimality of fisher's method of combining independent tests. *Journal of the American Statistical Association*, 1971.

I. Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 2012.

M. Haroush, T. Frostig, R. Heller, and D. Soudry. Statistical testing for efficient out of distribution detection in deep neural networks. *arXiv preprint arXiv:2102.12967*, 2021.

J. D. Havtorn, J. Frellsen, S. Hauberg, and L. Maaløe. Hierarchical VAEs know what they don't know. In *International Conference on Machine Learning (ICML)*. PMLR, 2021.

D. Hendrycks, M. Mazeika, and T. Dietterich. Deep anomaly detection with outlier exposure. In *7th International Conference on Learning Representations, (ICLR)*, 2019.

N. B. Ipsen, P. Mattei, and J. Frellsen. not-MIWAE: Deep generative modelling with missing not at random data. In *9th International Conference on Learning Representations, (ICLR)*, 2021.

T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. *Advances in Neural Information Processing Systems (NIPS)*, 1999.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, (ICLR)*, 2015.

D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

P. Kirichenko, P. Izmailov, and A. G. Wilson. Why normalizing flows fail to detect out-of-distribution data. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 2017.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Y. LeCun. *Modèles connexionnistes de l'apprentissage*. PhD thesis, Université Paris 6, 1987.

Y. LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

K. Lee, K. Lee, H. Lee, and J. Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

A. Lemonte. *The gradient test: another likelihood-based test*. Academic Press, 2016.

F. Liu, W. Xu, J. Lu, G. Zhang, A. Gretton, and D. J. Sutherland. Learning deep kernels for non-parametric two-sample tests. In *International Conference on Machine Learning (ICML)*. PMLR, 2020.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

C. Ma, S. Tschiatschek, K. Palla, J. M. Hernández-Lobato, S. Nowozin, and C. Zhang. EDDI: efficient dynamic discovery of high-value information with partial VAE. In *Proceedings of the 36th International Conference on Machine Learning, (ICML)*, Proceedings of Machine Learning Research. PMLR, 2019.

L. Maaløe, M. Fraccaro, V. Liévin, and O. Winther. Biva: A very deep hierarchy of latent variables for generative modeling. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

J. Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 2020.

J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*. PMLR, 2015.

P. Mattei and J. Frellsen. MIWAE: deep generative modelling and imputation of incomplete data sets. In *Proceedings of the 36th International Conference on Machine Learning, (ICML)*, Proceedings of Machine Learning Research. PMLR, 2019.

P.-A. Mattei and J. Frellsen. Refit your encoder when new data comes by. In *3rd NeurIPS Workshop on Bayesian Deep Learning*, 2018.

W. Morningstar, C. Ham, A. Gallagher, B. Lakshminarayanan, A. Alemi, and J. Dillon. Density of states estimation for out of distribution detection. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 2021.

E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations (ICLR)*, 2018.

E. Nalisnick, A. Matsukawa, Y. W. Teh, and B. Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using a test for typicality. *arXiv preprint arXiv:1906.02994*, 2019.

A. Nazábal, P. M. Olmos, Z. Ghahramani, and I. Valera. Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, 107:107501, 2020.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

J. Neyman and E. S. Pearson. On the use and interpretation of certain test criteria for purposes of statistical inference. *Biometrika*, 1928.

F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *European Conference on Computer Vision (ECCV)*. Springer, 2010.

C. R. Rao. Large sample tests of statistical hypotheses concerning several parameters with applications to problems of estimation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 44(1):50–57, 1948.

C. R. Rao. Score test: historical review and recent developments. In *Advances in Ranking and Selection, Multiple Comparisons, and Reliability*. Springer, 2005.

J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. Depristo, J. Dillon, and B. Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning (ICML)*. PMLR, 2014.

G. Rochette, A. Manoel, and E. W. Tramel. Efficient per-example gradient computations in convolutional neural networks. *arXiv preprint arXiv:1912.06015*, 2019.

T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modi-

fications. In *5th International Conference on Learning Representations (ICLR)*, 2017.

J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the Fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013.

R. Schirrmeister, Y. Zhou, T. Ball, and D. Zhang. Understanding anomaly detection with deep invertible networks through hierarchies of distributions and features. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

J. Serrà, D. Álvarez, V. Gómez, O. Slizovskaia, J. F. Núñez, and J. Luque. Input complexity and out-of-distribution detection with likelihood-based generative models. In *International Conference on Learning Representations (ICLR)*, 2020.

M. Tanaka, A. Torii, and M. Okutomi. Fisher vector based on full-covariance Gaussian mixture model. *Information and Media Technologies*, 2013.

G. R. Terrell. The gradient statistic. *Computing Science and Statistics*, 2002.

T. Tieleman, G. Hinton, et al. Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012.

A. Wald. Tests of statistical hypotheses concerning several parameters when the number of observations is large. *Transactions of the American Mathematical society*, 1943.

S. Watanabe. *Algebraic Geometry and Statistical Learning Theory*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2009.

D. J. Wilson. The harmonic mean p-value for combining dependent tests. *Proceedings of the National Academy of Sciences (PNAS)*, pages 1195–1200, 2019.

J. Winkens, R. Bunel, A. G. Roy, R. Stanforth, V. Natarajan, J. R. Ledsam, P. MacWilliams, P. Kohli, A. Karthikesalingam, S. Kohl, et al. Contrastive training for improved out-of-distribution detection. *arXiv preprint arXiv:2007.05566*, 2020.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Z. Xiao, Q. Yan, and Y. Amit. Likelihood regret: An out-of-distribution detection score for variational auto-encoder. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

L. Zhang, M. Goldstein, and R. Ranganath. Understanding failures in out-of-distribution detection with deep generative models. In *International Conference on Machine Learning (ICML)*. PMLR, 2021.

Supplementary Material: Model-agnostic out-of-distribution detection using combined statistical tests

A Crude approximation of the Fisher information

The Fisher information is defined as:

$$I(\theta) = \mathbb{E}_{x\sim p_\theta}[\nabla \log p_\theta(x) \nabla \log p_\theta(x)^T]. \tag{7}$$

A crude diagonal approximation can be computed by simply estimating the diagonal of $I(\theta)$ and setting all off-diagonal elements to zero. Such diagonal approximations have been used in machine learning for decades: for instance, LeCun (1987, Section 3.12.2) used a similar approximation of the Hessian matrix, and called it "outrageously simplifying". Much more complex approximations have been derived, although diagonal approximations have been consistently used (e.g. by Kirkpatrick et al., 2017, who used essentially the same approximation in a supervised context), and are linked to several adaptive optimisation techniques like Adam (Kingma and Ba, 2015) or RMSProp (Tieleman et al., 2012). A good discussion on these issues is provided in Martens's (2020) recent review.

The approximation we used in the paper works as follows:

- By using the training examples $x_1, ..., x_T$, we form the estimate

$$D_T(\theta) = \frac{1}{T}\sum_{t=1}^{T} \text{Diag}(\nabla \log p_\theta(x_t)^2),$$

  where the square in $\nabla \log p_\theta(x_t)^2$ is computed elementwise.

- While we could directly use $D_T(\theta)$ as an estimate. A slightly more refined approach is to slightly regularise $D_T(\theta)$. Following Martens (2020), our final estimate of the Fisher information matrix is

$$\hat{I}_T(\theta) = (D_T(\theta) + \varepsilon)^\xi, \tag{8}$$

  with all operations performed elementwise. The diagonal matrix $\hat{I}_T(\theta)$ is then easy to invert and can be used to compute our statistics.

How to choose ε and ξ? The Adam optimizer uses a similar estimate, with default hyperparameters $\varepsilon = 10^{-8}$ and $\xi = 1$. As argued by Martens (2020), it can be interesting to use $\xi < 1$ in order to diminish the influence of extreme values of $D_T(\theta)$. In particular, Martens (2020) suggests taking $\xi = 0.75$. When $\xi \longrightarrow 0$, then $\hat{I}_T(\theta)$ will approach the identity matrix. We tested the two settings by using a PixelCNN++ trained on CIFAR. Results are shown in table 3. In terms of OOD detection, it seems that using $\varepsilon = 10^{-8}$ and $\xi = 1$ is slightly better. All results presented in the paper and in the supplementary material are computed by using $\varepsilon = 10^{-8}$ and $\xi = 1$.

A few notes on the computation of $D_T(\theta)$ While it seems more sensible to use samples $x_1, ..., x_m \sim p_\theta$ from the model, we decided to simply reuse the training data $x_1, ..., x_T$ instead. There are two computational advantages to this. The first one is that sampling many data points can be expensive (in particular for deep autoregressive models à la PixelCNN). The second advantage is that, if we wish to compute a MMD statistic, such as the MMD with the Fisher kernel or the MMD typicality (that require the average of gradient or the average log-likelihood over the training), computing the average of the square of the gradient costs very little. One can just do a single loop over the data, and use the usual formulas for online estimation of a mean, see Algorithm 1.

Table 3: AUROC↑ for single-sample OOD detection. Comparison between two different estimates of the Fisher information matrix. For (‡) we used the Adam parameter choice, i.e. $\varepsilon = 10^{-8}$ and $\xi = 1$. For (§), instead, we used $\varepsilon = 10^{-8}$ and $\xi = 0.75$, as suggested by Martens (2020). As a results we have that using Adam parameters choice is slightly better for our task.

	CIFAR10 (IN) / SVHN (OUT)			
MODELS	MMD DIAGONAL	TYPICALITY	SCORE STAT	FISHER'S METHOD
PIXELCNN++ (model2) (‡)	0.7070	0.6498	0.7067	0.7300
PIXELCNN++ (model2) (§)	0.6881	0.6498	0.6878	0.7176

(‡) With $\varepsilon = 10^{-8}$ and $\xi = 1$
(§) With $\varepsilon = 10^{-8}$ and $\xi = 0.75$

Do we really need to approximate the diagonal of $I(\theta)$? Another possibility is to just use the identity matrix as FIM instead of of approximating the diagonal through the procedure explained above. In our experiments (see table 5 and table 9), we can see that sometimes using the identity matrix seems to work equally well or a bit better for some models trained on FashionMNIST and CIFAR10. However, when we train on SVHN or MNIST, there are a cases where the statistic that is using the identity matrix as approximation fails, sometimes being worse than random chance. In those setting, using the diagonal approximation leads to way better results. Therefore, considering a test statistic that uses the diagonal approximation of the FIM is more robust for OOD detection.

B The Mahalanobis score as MMD

Lee et al. (2018) introduced a simple metric to perform OOD detection with a trained deep classifier. The key idea is to train a simple generative model (linear discriminant analysis) in the feature space of the classifier. Let y denote the labels, and $z = f(x)$ the data in feature space. In the simplest case, f is just the trained deep net devoid of the last softmax layer. The linear discriminant analysis model is

$$y \sim \text{Cat}(\pi), \quad z|y \sim \mathcal{N}(\mu_y, \Sigma), \tag{9}$$

where $\mu_1, ..., \mu_K$ are class-dependent means, Σ a common covariance matrix, and $\pi_1, ..., \pi_K$ are the class proportions, estimated by maximum-likelihood. The *Mahalanobis score* is then

$$M(x) = \max_{k \in \{1,...,K\}} -(z - \mu_k)^T \Sigma^{-1} (z - \mu_k), \tag{10}$$

which may be rewritten

$$M(x) = \max_{k \in \{1,...,K\}} p(z|k), \tag{11}$$

under the assumption of equal class proportions (i.e. $\pi_1 = ... = \pi_K = 1/K$).

We show here that it is possible to re-interpret this score as a MMD score with a certain Fisher kernel. The generative model induced on z by linear disciminant analysis is a Gaussian mixture:

$$p_{\pi,\mu,\Sigma}(z) = \sum_{k=1}^{K} \pi_k \mathcal{N}(z|\mu_k, \Sigma). \tag{12}$$

If we want a powerful deep kernel, it seems somewhat natural to consider the Fisher kernel associated with this generative model. The most important part of this mixture model are arguably the class-specific means (indeed, the model has been trained to discriminate the classes as well as possible). Therefore, we will only include these means in the Fisher kernel, and look at

$$\Phi_{\text{Fisher}}(x) = I(\mu)^{-1/2} \nabla_\mu \log p_{\pi,\mu,\Sigma}(z), \tag{13}$$

assuming that π and Σ are fixed at their maximum likelihood estimates. Similar mixture-based Fisher kernels have been very popular in the past, and were actually a key element of state-of-the art classification models on

Imagenet before deep nets won the competition (Perronnin et al., 2010). Our idea is to re-use ideas introduced by this computer vision litterature. Under the assumption that the Gaussian clusters are well-separated, Tanaka et al. (2013), extending an earlier analysis of Sánchez et al. (2013, Appendix A), showed that

$$[\Phi_{\text{Fisher}}(x)]_{\mu_k} \approx \sqrt{\frac{p(z|k)}{\pi_k}}\Sigma^{-1/2}(z-\mu_k). \tag{14}$$

Now, using the fact that the expected value of the score is approximatively zero, we can write that

$$\text{MMD}^2_{\Phi_{\text{Fisher}}} \approx \sum_{k=1}^{K} ||[\Phi_{\text{Fisher}}(x)]_{\mu_k}||_2^2 \approx \sum_{k=1}^{K} \frac{p(z|k)}{\pi_k}(z-\mu_k)^T\Sigma^{-1}(z-\mu_k). \tag{15}$$

Using again the fact that the clusters are well-separated, we may say that $z|k$ is approximatively a point mass at the most probable label, i.e. that $p(z|k) \approx \delta_k^{\text{argmax}_c p(z|c)}$. This leads to the approximation

$$\text{MMD}^2_{\Phi_{\text{Fisher}}} \approx \max_{k\in\{1,\dots,K\}} \frac{1}{\pi_k}(z-\mu_k)^T\Sigma^{-1}(z-\mu_k). \tag{16}$$

Finally, assuming that the class proportions are equal leads to the equivalence of $\text{MMD}_{\Phi_{\text{Fisher}}}$ and the Mahalanobis score.

C More Information on the experimental setup

C.1 A bit more background

The three considered DGMs are both parametrized by neural networks but they differ in the way they model the data distribution of interest. Assume we are interested in approximating a target distribution $p^*(\mathbf{x})$, for example a distribution of natural images, as it is done when using CIFAR10. PixelCNN++ is an autoregressive model and it models $p^*(\mathbf{x})$ as a product of conditional distribution over the variables, i.e. $p(\mathbf{x}) = p(x_1)\prod_{d=2}^{D} p(x_d \mid \mathbf{x}_{<d})$, where $\mathbf{x}_{<d} = [x_1, \dots, x_{d-1}]^T$. Glow is a normalizing flow model and it approximate $p^*(\mathbf{x})$ by using a sequence of bijiective transformations starting from a simple distribution, also called base distribution. If we use only a single invertible transformation f, the normalizing flow is defined as $\mathbf{x} = f(\mathbf{z})$, where $\mathbf{z} \sim p_Z(\mathbf{z})$, and $p_X(\mathbf{x}) = p_Z(\mathbf{z})|\det J_f(\mathbf{z})|^{-1}$, where we used the change of variable formula. For these two types of model we have a tractable likelihood that can be used to optimize the model parameters. The Variational Autoencoder (VAE), instead, is a framework to model the data with a latent variable model, i.e. $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})$, where $\mathbf{x}$ is the observed input data and $\mathbf{z}$ is a stochastic latent variable and the prior distribution $p(\mathbf{z})$ is usually a standard Normal. Since the posterior $p(\mathbf{z} \mid \mathbf{x})$ is not tractable, a variational distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$ is used as an approximation. Due to the intractability of the posterior, we cannot directly optimize the likelihood of the model, but instead the model parameters are optimized by maximizing the evidence lower bound (ELBO): $\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \equiv \mathcal{L}$. In this work we are considering an Hierarchical VAE (HVAE) with bottom-up inference as done in Havtorn et al. (2021). This is an extension of the VAE framework that consider an hierarchy of L latent variables $\mathbf{z} = \mathbf{z}_1, \dots, \mathbf{z}_L$. The bottom-up inference is defined as $q_\phi(\mathbf{z} \mid \mathbf{x}) = q_\phi(\mathbf{z}_1 \mid \mathbf{x})\prod_{i=2}^{L} q_\theta(\mathbf{z}_i \mid \mathbf{z}_{i-1})$, while the generative path is top-down, meaning $p_\theta(\mathbf{x} \mid \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z}_1)p_\theta(\mathbf{z}_1 \mid \mathbf{z}_2)\cdots p_\theta(\mathbf{z}_{L-1} \mid \mathbf{z}_L)$. This is still trained by maximizing the ELBO. For a more in-depth explanation of these models we refer to their papers.

C.2 Generative model details

We will briefly describe the different model architectures and training procedures used in this paper. Since most of the models are taken from public code repositories and related papers, we will mostly invite the reader to have a look at the cited paper for a more in-depth description of the training details. For MNIST, CIFAR10, and FashionMNIST we used 3000 examples from the test set as validation set. For SVHN, instead, we used 6032 datapoints from the test set as validation, leaving the remaining 20000 examples as test set. In Table 4, we reported test log-likelihood of the models used in this paper.

Table 4: Test log-likelihood (bits/dim) achieved by the models used in the paper.

Models trained on FashionMNIST)	
models	Log-Likelihood (bits/dim)
PixelCNN++ (dropout)	2.75
PixelCNN++ (no dropout)	2.72
Glow (RMSProp)	3.04
Glow (Adam)	3.02
HVAE (**)	0.43

Models trained on MNIST	
models	Log-Likelihood (bits/dim)
PixelCNN++ (dropout)	0.90
Glow (RMSProp)	1.32
Glow (Adam)	1.30
HVAE (**)	0.16

Models trained on CIFAR10	
models	Log-Likelihood (bits/dim)
PixelCNN++ (model1)	2.94
PixelCNN++ (model2)	2.94
Glow (RMSProp)	3.62
Glow (Adam)	3.62
HVAE	3.87

(**) Binarized FashionMNIST

Models trained on SVHN	
models	Log-Likelihood (bits/dim)
PixelCNN++ (dropout)	1.58
Glow (RMSProp)	2.23
Glow (Adam)	2.21
HVAE	2.38

(**) Binarized MNIST

PixelCNN++ For PixelCNN++ we used the code available in this repository[2]. For the greyscale images, we used one residual block per stage with 32 filters and 5 logistic components in the discretized mixture of logistics. For natural images, instead, we used 5 residual blocks per stage with 160 filters and 10 components in the mixture. We trained all the models using Adam optimizer.

Glow For training Glow models we follow Kirichenko et al. (2020) and their repository[3]. They closely follow Nalisnick et al. (2018) and Kingma and Dhariwal (2018) implementation for multi-scale Glow, where a scale is defined as the sequence of actorm, invertible 1×1 convolution and coupling layers. While Kirichenko et al. (2020) only considers the RMSProp optimizer, we trained two different models, one using RMSProp and one using Adam with batch-size 32. For the greyscale dataset our Glow is made up of 2 scales with 16 coupling layers, and a 3-layers highway network with 200 hidden units is used to predict the scale and shift parameters. For CIFAR10 and SVHN, instead, we used 3 scales with 8 coupling layers, and 400 hidden units for the 3-layers highway network. For a more in-depth description, we refer to the codebase and the Appendix C of Kirichenko et al. (2020).

Hierarchical VAE We follow Havtorn et al. (2021) for both model architecture design and training choices for our hierarchical VAEs. We used their open-sourced repository[4]. As mentioned in the paper, the HVAE model we used has a bottom-up inference path and a top-down generative path. We trained each model for 1000 epochs using Adam optimizer with learning rate $3e-4$ and a batch-size of 128. All models were initialized using the data-dependent initialization and they used weight-normalization (Salimans and Kingma, 2016). In addition to that, we always consider a hierarchy of three latent variables. For greyscale images (MNIST and FashionMNIST) we used a latent dimension of $8-16-8$ for $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3$ respectively, while for natural images (CIFAR10 and SVHN) we used $8-16-32$. For a more in depth description of the model, we refer to Appendix B of Havtorn et al. (2021).

D Additional results

D.1 Typicality test and score statistic are uncorrelated

To test if the typicality test and the score statistic are uncorrelated, we plot the two scores computed on the validation set. As can be seen from figure 3, we have that the two measures are not correlated as it is also highlight by the correlation coefficient.

[2] https://github.com/pclucas14/pixel-cnn-pp
[3] https://github.com/PolinaKirichenko/flows_ood
[4] https://github.com/JakobHavtorn/hvae-oodd

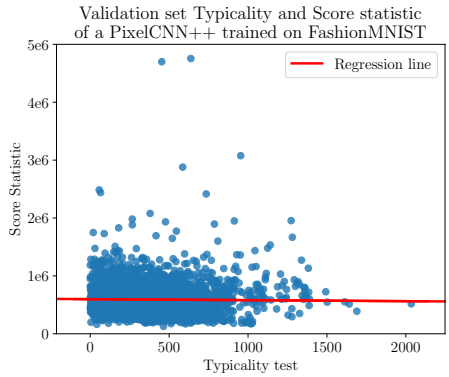


Figure 3: Correlation of Typicality Test and Score Statistic computed on the validation set using a PixelCNN++ trained on FashionMNIST. The correlation coefficient is -0.014. This can also be seen by looking at the regression line, which is almost straight.

D.2 Harmonic Mean

In the paper we mentioned that another way to combine p-values from different test statistics is the Harmonic mean (Wilson, 2019). This is defined as:

$$\mathring{p} = \frac{\sum_{i=1}^{k} w_i}{\sum_{i=1}^{k} w_i / p_i}, \tag{17}$$

where $w_1, \ldots, w_k$ are weights that sum up to 1. In our setting, we considered equal weights, i.e. $w_i = 1/k$. Therefore, if we simply consider two test statistics T_1 and T_2 and corresponding p-values p_1 and p_2, the harmonic mean p-values becomes:

$$\mathring{p} = \frac{2p_1p_2}{p_1 + p_2}. \tag{18}$$

As expected, this combination should work better when the statistics that we are combining are somewhat correlated. Indeed, since in our setting we have that the typicality and the score statistic are independent, we would expect this to work worse than the Fisher's combination. This is confirmed by table 6, where we are reporting the results when combining the two statistics using the three different ways we analyzed.

D.3 Results considering maximum-mean-discrepancy

In Section 4, we discussed the relationship between the maximum-mean-discrepancy with a Fisher kernel and the score statistic and the gradient norm, which depends on the choice of approximation of the Fisher information matrix we use. In table 5 we reported also the AUROC scores for the MMD with Fisher kernel considering both the diagonal approximation of the FIM (called *MMD Diagonal* in the table) and the FIM being the identity matrix (called *MMD Identity*). As expected, we have that the AUROC of the MMD with the diagonal approximated FIM is pretty close to the AUROC we obtained by using the score statistic. Likewise, we have that the AUROC of MMD with the identity matrix as FIM is close to the gradient norm when we trained on FashionMNIST and CIFAR10.

So, why did we decide to use the score statistic instead of the MMD with Fisher kernel and diagonal approximation of the FIM? The main reason is Occam's razor. If we have two things that work equally well, we should keep the simplest one. In our case, we have that for computing the MMD with the Fisher kernel, we need to compute both the average gradient and the FIM using the training set. For the score statistic, instead, we just need the FIM. In addition to that, from all our experiments (see table 5 and table 9) we do not have any evidence for one statistic working better than the other, because they are always pretty close to each other.

Table 5: AUROC↑ for single-sample OOD detection. In this table we consider all the different single statistics we mentioned in the paper. One can notice that MMD Diagonal is pretty close to the score statistic and the MMD Identity is close to the gradient norm, as expected (see Section 4.1 in the paper).

	FashionMNIST (in) / MNIST (out)					
	single statistics					
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	MMD Diagonal	MMD Identity	Typicality	Score Stat
PixelCNN++ (dropout)	0.0762	0.8709	0.8903	0.8690	0.8314	0.8822
PixelCNN++ (no dropout)	0.1048	0.9532	0.9393	0.9539	0.7575	0.9381
Glow (RMSProp)	0.1970	0.8904	0.9115	0.8986	0.4807	0.9114
Glow (Adam)	0.1223	0.7705	0.8540	0.7217	0.6987	0.8745
HVAE	0.0653	0.8714	0.9574	0.8726	0.8336	0.9578

	CIFAR10 (in) / SVHN (out)					
	single statistics					
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	MMD Diagonal	MMD Identity	Typicality	Score Stat
PixelCNN++ (model1)	0.1553	0.8006	0.6406	0.8126	0.6457	0.6407
PixelCNN++ (model2)	0.1567	0.7923	0.7070	0.7955	0.6498	0.7067
Glow (RMSProp)	0.0630	0.8585	0.7929	0.8621	0.8651	0.7940
Glow (Adam)	0.0627	0.7844	0.7620	0.7838	0.8624	0.7655
HVAE	0.0455	0.8041	0.7268	0.7634	0.8845	0.7334

Table 6: AUROC↑ for single-sample OOD detection. Comparison between the three method we mentioned to combine different statistics. Since the typicality and the score statistic are not correlated, we have that the Fisher's method is mostly working better than the other two methods.

	FashionMNIST (in) / MNIST (out)		
	combinations		
models	Fisher's method	Harmonic mean	DoSE_{KDE}
PixelCNN++ (dropout)	0.9369	0.9148	0.8822
PixelCNN++ (no dropout)	0.9536	0.9392	0.9382
Glow (RMSProp)	0.8598	0.8853	0.8901
Glow (Adam)	0.8839	0.8632	0.8752
HVAE	0.9708	0.9569	0.9630

	CIFAR10 (in) / SVHN (out)		
	combinations		
models	Fisher's method	Harmonic mean	DoSE_{KDE}
PixelCNN++ (model1)	0.6826	0.6667	0.6571
PixelCNN++ (model2)	0.7300	0.7105	0.7243
Glow (RMSProp)	0.8683	0.8551	0.8510
Glow (Adam)	0.8613	0.8493	0.8588
HVAE	0.8699	0.8525	0.8245

D.4 Variability within the same model in different checkpoints

As mentioned in the paper, we noticed that all statistics depend on choices we made about our model and the training procedure, such as deciding between Adam or RMSProp, or between using dropout or not. In addition to that, we find out that they can differ also within the same model at different checkpoints that obtain almost the same log-likelihood. Here we consider two Glow models, one trained with Adam and one using RMSProp on CIFAR10. For both, we consider two checkpoints that achieve the same test log-likelihood. Those trained with Adam get a log-likelihood of 3.63 bits/dim, while the ones trained with RMSProp get 3.62 bits/dim. Results are shown in Table 7. It can be noticed, that although the models are similar in terms of test bits/dim the statistics vary a lot, mostly when training with RMSProp.

D.5 Benjamini-Hochberg procedure when training on CIFAR10

In the main paper we focused on the Benjamini-Hochberg procedure applied to a model trained on FashionMNIST. Although one should use a False Discovery Rate control procedure when the statistics we are using are strong, for completeness, we will present what happens when we apply the BH procedure on a model trained

Table 7: AUROC↑ for single-sample OOD detection. In this table we are comparing two different Glow models trained on CIFAR10 by considering two different checkpoints with almost the same test log-likelihood. We can see that both statistics vary a bit.

	CIFAR10 (IN) / SVHN (OUT)			
	SINGLE STATISTICS		COMBINATION	
MODELS	TYPICALITY	SCORE STAT	FISHER'S METHOD	DoSE_{KDE}
GLOW (RMSProp) {*check1*}	0.8651	0.7940	0.8683	0.8510
GLOW (RMSProp) {*check2*}	0.8532	0.6894	0.8275	0.7815
GLOW (Adam) {*check1*}	0.8624	0.7655	0.8613	0.8588
GLOW (Adam) {*check2*}	0.8558	0.7327	0.8402	0.8303

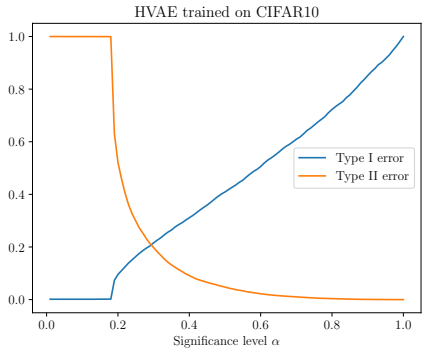


Figure 4: Type I and Type II errors versus the significance level α on the combination values. We can control the FDR only for $\alpha > 0.2$ in this case. For $\alpha > 0.2$, since we are using Benjamini-Hochberg procedure, we get that the Type I error stays below identity line.

on CIFAR10. In Fig. 4, we report the Type I error ratio and the Type II error ratio for different significance levels α. We can see that we can actually control the FDR for $\alpha > 0.2$, and for these significance levels we are actually controlling the FDR. What is happening for $\alpha < 0.2$? We have that the procedure is only rejecting 5 hypotheses and all these hypotheses corresponds to in-distribution examples. Therefore, we have that the ratio of Type I error is still low, but we are making a lot of Type II errors because we are accepting all the examples whose hypotheses should be rejected.

D.6 Results when training on MNIST and SVHN

We also evaluated our methods in the two dataset pairs, MNIST against FashionMNIST and SVHN against CIFAR10, that are usually considered easier than the tasks presented in the main paper. For both tasks, we trained two Glow models, one trained with Adam and one trained with RMSProp, one PixelCNN++ trained with dropout and a hierarchical-VAE. Results are reported in table 8. We can see that almost all the statistics we considered are able to almost perfectly distinguish between the in-distribution test-set and the OOD test-set. However, we can notice that the gradient norm is failing sometimes both when we trained on CIFAR10 and when we trained on FashionMNIST. From table 9, instead, it is clear that we need to approximate the diagonal of the Fisher Information Matrix because if we simply consider the identity matrix, this will also fail as the gradient norm is doing.

D.7 Application of our method to Gaussian Mixture Model and Probabilistic PCA

Since the method we propose is model-agnostic, we show that it can be used for out-of-distribution detection also using two simple generative models, Gaussian Mixture Model (GMM) and Probabilistic PCA (PPCA). We consider the two pairs of datasets as before, i.e. FashionMNIST vs MNIST and CIFAR10 vs SVHN. Results can be seen in Table 10 and Table 11. For both GMM and PPCA trained on FashionMNIST the likelihood can be used to perform OOD detection. Indeed, in this setting, they are not assigning higher likelihood to OOD data as

Table 8: AUROC↑ for single-sample OOD detection when training on MNIST and testing again FashionMNIST and when training on SVHN and testing against CIFAR10. As before, Fisher's method is the combination of the typicality test and the test statistic. These are also combined using DoSE.

	MNIST (in) / FashionMNIST (out)					
	single statistics				combination	
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
PixelCNN++ (dropout) (†)	0.9999	0.8534	0.9996	0.9993	0.9999	0.9999
Glow (RMSProp)	0.9997	0.9936	0.9991	0.9936	0.9992	0.9994
Glow (Adam)	0.9999	0.6506	0.9995	0.9992	0.9998	0.9999
HVAE	0.9999	0.9998	0.9997	0.9999	0.9999	0.9999

	SVHN (in) / CIFAR10 (out)					
	single statistics				combination	
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
PixelCNN++ (dropout)	0.9820	0.2670	0.9590	0.9543	0.9914	0.9824
Glow (RMSProp)	0.9917	0.9180	0.9830	0.9823	0.9913	0.9913
Glow (Adam)	0.9913	0.5658	0.9779	0.9641	0.9883	0.9863
HVAE	0.9943	0.1011	0.9857	0.9862	0.9934	0.9862

(†) Trained using 50000 datapoints

Table 9: AUROC↑ for single-sample OOD detection. In this table we consider all the different single statistics we mentioned in the paper but for the models trained on MNIST and SVHN this time. In this case, it is important to notice that the gradient norm and the MMD identity sometimes fail to a different extent.

	MNIST (in) / FashionMNIST (out)					
	single statistics					
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	MMD Diagonal	MMD Identity	Typicality	Score Stat
PixelCNN++ (dropout) (†)	0.9999	0.8534	0.9993	0.8608	0.9996	0.9993
Glow (RMSProp)	0.9997	0.9936	0.9942	0.6609	0.9991	0.9936
Glow (Adam)	0.9999	0.6506	0.9993	0.9124	0.9997	0.9992
HVAE	0.9999	0.9998	0.9999	0.9999	0.9999	0.9999

	SVHN (in) / CIFAR10 (out)					
	single statistics					
models	$\log p(x)$	$\|\nabla \log p(x)\|_2$	MMD Diagonal	MMD Identity	Typicality	Score Stat
PixelCNN++ (dropout)	0.9820	0.2670	0.9543	0.3185	0.9590	0.9543
Glow (RMSProp)	0.9917	0.9180	0.9824	0.9317	0.9830	0.9823
Glow (Adam)	0.9913	0.5658	0.9653	0.7096	0.9779	0.9641
HVAE	0.9943	0.1011	0.9865	0.4508	0.9857	0.9862

(†) Trained using 50000 datapoints

it is the case for DGMs. This happens instead when we fit these models on CIFAR10. However, this behaviour can be due to the fact that they are really poor generative models for this dataset. It is also surprising that when training on CIFAR10 the score statistic is failing in both models. We think that this is also due to the fact that both the GMM and the PPCA are far from being good generative models for this dataset.

D.8 More in depth analysis of the variability of the results for different HVAE

As we have pointed out before, test statistics and consequentially out-of-distribution performances can vary between the same model trained several times on the same dataset. To test the variability of the results shown in the main paper, we trained five different hierarchical VAEs and compute mean and standard deviations of the final AUROC scores. All models have the same architecture and were trained with the same procedure. Results can be found in Table 12. For the models trained on CIFAR10, most of the variability in terms of performance is due to the score statistic, which has the highest standard deviation. When training on FashionMNIST, instead, it seems that the typicality performance is the one varying the most between the five models.

Table 10: AUROC↑ for single-sample OOD detection using a Gaussian mixture model (GMM). For Fisher's method we mean the combination of the typicality test and the test statistic. These are also combined using DoSE.

	FashionMNIST (in) / MNIST (out)					
	single statistics				combination	
components	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
50	0.6627	0.5514	0.5196	0.8777	0.7689	0.8152
100	0.6872	0.5509	0.5575	0.8742	0.7965	0.7989

	CIFAR10 (in) / SVHN (out)					
	single statistics				combination	
components	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
50	0.2335	0.6087	0.6759	0.3512	0.6098	0.6569
100	0.2372	0.6136	0.6714	0.3294	0.5898	0.6573

Table 11: AUROC↑ for single-sample OOD detection using a Probabilistic PCA. For Fisher's method we mean the combination of the typicality test and the test statistic. These are also combined using DoSE.

	FashionMNIST (in) / MNIST (out)					
	single statistics				combination	
components	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
50	0.9727	0.9637	0.9587	0.9505	0.9635	0.9610
100	0.9557	0.9715	0.9309	0.9626	0.9566	0.9585

	CIFAR10 (in) / SVHN (out)					
	single statistics				combination	
components	$\log p(x)$	$\|\nabla \log p(x)\|_2$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
50	0.0770	0.1494	0.8468	0.1308	0.7568	0.8210
100	0.0357	0.0778	0.8944	0.0755	0.7966	0.8830

E Yes, we should talk about CelebA

Out-of-distribution detection performance is not only influenced by the model architecture or the training process. Indeed, transformations applied to the input data play an important role by transforming a difficult task into an easier problem where the likelihood can detect OOD data. By looking at the different results for Glow trained on CIFAR10 and tested on CelebA shown in Hendrycks et al. (2019), Kirichenko et al. (2020), Morningstar et al. (2021), and Ahmadian and Lindsten (2021) we can see that the AUROC scores obtain by the plain log-likelihood are pretty different. In Hendrycks et al. (2019) and Kirichenko et al. (2020) the log-likelihood gets a poor performance, confirming that CIFAR10-CelebA is a challenging pair for DGMs, while in Morningstar et al. (2021) the likelihood is able to distinguish OOD data. While the main reason for these different results can be due to model implementation and training procedure, we decided to investigate how different transformations can influence OOD detection. Indeed, CelebA examples originally have a shape of $(218, 178, 3)$ and to transform them into $(32, 32, 3)$-shaped images, as CIFAR10, we have to resize them and then crop their center. The resize function is performing an interpolation, therefore we analyze how different interpolation strategies influence the OOD task.

We considered three different interpolations: bilinear (default in PyTorch), Lanczos, and nearest. As can be seen from Fig. 5, these transformations mostly affect the sharpness of the images. In Table 13 we show how the OOD performance changes for our considered models when testing on CelebA where we applied different interpolations. We can notice that when using the bilinear interpolation we get results that are pretty similar

Table 12: Mean and standard deviation of the performance in terms of AUROC of our method. Quantities are computed by taking the performance of 5 different trained HVAEs both trained on CIFAR10 and FashionMNIST.

$\mathcal{D}_{\text{OUT}}$	$\log p(x)$	Typicality	Score Stat	Fisher's method	DoSE_{KDE}
	HVAE trained on CIFAR10				
SVHN	0.0631 (0.0008)	0.8711 (0.0028)	0.7808 (0.0255)	0.8844 (0.0140)	0.8519 (0.0194)
CIFAR100	0.5349 (0.0007)	0.5496 (0.0003)	0.5857 (0.0042)	0.5924 (0.0029)	0.5985 (0.0028)
CelebA	0.9004 (0.0035)	0.8203 (0.0046)	0.7565 (0.0369)	0.8505 (0.0138)	0.8247 (0.0228)
	HVAE trained on FashionMNIST				
MNIST	0.2487 (0.0152)	0.5064 (0.0245)	0.9532 (0.0084)	0.9220 (0.01491)	0.9377 (0.0126)

Figure 5: Comparison of different interpolation methods for CelebA dataset.

to Hendrycks et al. (2019), Kirichenko et al. (2020), and Ahmadian and Lindsten (2021) in terms of likelihood OOD performance. When using the nearest interpolation, instead, we get results that are closer to Morningstar et al. (2021).

In conclusion, with these experiments, we wanted to highlight the importance of reporting the preprocessing steps used in loading CelebA in order to be able to make a fair comparison with the other proposed methods in the literature.

F Comparison with the original DoSE statistics

As the last experiment, we study how our proposed method with our model agnostic statistic performs against DoSE using the original statistics proposed in Morningstar et al. (2021). For the VAEs model, they suggested to use the following 5 statistics: the posterior/prior cross-entropy $\mathrm{H}[q_\phi(\mathbf{z} \mid \mathbf{x}), p(\mathbf{z})]$, the posterior entropy $\mathrm{H}[q_\phi(\mathbf{z} \mid \mathbf{x})]$, the posterior/prior KL divergence $\mathrm{D}_{\mathrm{KL}}[q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p(\mathbf{z})]$, the posterior expected log-likelihood $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log q_\phi(\mathbf{z} \mid \mathbf{x})]$, and the log-likelihood $\log \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$. For DoSE on Glow, instead, they considered three metrics: the log-likelihood $p_X(\mathbf{x} \mid \theta_n)$ and its two components, i.e. the log-probability of the latent variable $p_Z(\mathbf{z} \mid \mathbf{x}, \theta_n)$ and the log-determinant of the Jacobian $\log |\mathrm{J}_f(\mathbf{x})|$.

In this setting, since DoSE is using statistics that are HVAE and Glow specific, it is not model agnostic anymore. Indeed, we cannot use those statistics also for a PixelCNN++ for example or any other DGM. We want also to highlight that the models used in Morningstar et al. (2021) are a bit different from the ones used in this work. For example, they are considering a beta-VAE with only one stochastic layer, while in our case we used a HVAE with 3-stochastic layers.

G Algorithmic implementation

A pseudocode describing step-by-step how to implement our method is given in Algorithm 1.

Table 13: AUROC↑ for single-sample OOD detection for CIFAR10 vs CelebA considering all the three interpolations when using CelebA.

	CIFAR10 (IN) / CelebA (OUT) (†)					
	SINGLE STATISTICS				COMBINATION	
MODELS	$\log p(x)$	$\|\nabla \log p(x)\|_2$	TYPICALITY	SCORE STAT	FISHER'S METHOD	DoSE_{KDE}
PixelCNN++ (model1)	0.7027	0.5856	0.5581	0.7001	0.6450	0.6931
PixelCNN++ (model2)	0.7034	0.4298	0.5554	0.7505	0.6879	0.7430
Glow (RMSProp)	0.5337	0.5616	0.3926	0.6561	0.5400	0.5866
Glow (Adam)	0.5308	0.5820	0.3914	0.5850	0.4818	0.5212
HVAE	0.5643	0.5214	0.4011	0.6712	0.5483	0.5987

	CIFAR10 (IN) / CelebA (OUT) (⸸)					
	SINGLE STATISTICS				COMBINATION	
MODELS	$\log p(x)$	$\|\nabla \log p(x)\|_2$	TYPICALITY	SCORE STAT	FISHER'S METHOD	DoSE_{KDE}
PixelCNN++ (model1)	0.8284	0.5035	0.7399	0.6714	0.7477	0.7123
PixelCNN++ (model2)	0.8284	0.3530	0.7370	0.70088	0.7631	0.7446
Glow (RMSProp)	0.7556	0.4427	0.6222	0.7865	0.7423	0.7632
Glow (Adam)	0.7499	0.4800	0.6177	0.6442	0.6460	0.6467
HVAE	0.7561	0.4097	0.6051	0.6779	0.6775	0.6772

	CIFAR10 (IN) / CelebA (OUT) (‡)					
	SINGLE STATISTICS				COMBINATION	
MODELS	$\log p(x)$	$\|\nabla \log p(x)\|_2$	TYPICALITY	SCORE STAT	FISHER'S METHOD	DoSE_{KDE}
PixelCNN++ (model1)	0.9270	0.4196	0.8902	0.8320	0.9287	0.8908
PixelCNN++ (model2)	0.9270	0.3065	0.8886	0.8448	0.9339	0.9236
Glow (RMSProp)	0.9364	0.5345	0.8880	0.9286	0.9390	0.9423
Glow (Adam)	0.9322	0.5957	0.8829	0.8350	0.9017	0.8933
HVAE	0.8964	0.3515	0.8158	0.7952	0.8620	0.8455

(†) Bilinear interpolation
(⸸) Lanczos interpolation
(‡) Nearest interpolation

Table 14: Comparison between our method and DoSE using the original statistics. In these experiments we considered only Glow trained with Adam.

D_{OUT}	OUR METHOD	$\text{DoSE}_{\text{orig}}$
	GLOW TRAINED ON CIFAR10	
SVHN	**0.8613**	0.7819
CIFAR100	**0.5775**	0.5700
CelebA	0.9017	**0.9663**
	GLOW TRAINED ON FashionMNIST	
MNIST	0.8839	**0.9568**
	HVAE TRAINED ON FashionMNIST	
MNIST	0.9383	**0.9762**
	HVAE TRAINED ON CIFAR10	
SVHN	0.8605	**0.8823**
CIFAR100	**0.5888**	0.5608
CelebA	**0.8620**	0.8203

Algorithm 1 Computing p-values for OOD detection using a trained generative model.

Input: Training data $\mathbf{X} = (x_1, \ldots, x_m)^T$, validation data $\mathbf{X}'$, trained model $p_\theta(x)$.

Approximation of the diagonal of the Fisher Information Matrix $I(\theta)$ and average log-likelihood $(1/m) \log p_\theta(x_1, \ldots, x_m)$, indicated by $L(\theta)$. We do it in an online fashion.
Initialize $I(\theta) = 0$ and $L(\theta) = 0$
For all $i \in \{1, \ldots, m\}$:
  Compute $\log p_\theta(x_i)$
  Compute $\nabla_\theta \log p(x_i \mid \theta)$
  Set $I(\theta) = \frac{1}{i+1} \cdot (i \cdot I(\theta) + (\nabla_\theta \log p_\theta(x_i))^2)$
  Set $L(\theta) = \frac{1}{i+1} \cdot (i \cdot L(\theta) + \log p_\theta(x_i))$

Estimation of distributions over the test statistics
Sample S M'-sized datasets from $\mathbf{X}'$ using bootstrap resampling.
(For single-sample OOD we just cycle through each example, see Sec. 3)
Initialize $T^{\text{typicality}} = [\,]$ and $T^{\text{score}} = [\,]$
For every bootstrapped dataset $\mathbf{X}'_s = (x_1, \ldots, x_{M'})^T$:
  Compute $\frac{1}{m'} \sum_{m'=1}^{M'} \log p_\theta(x_{m'})$
  Compute $\frac{1}{m'} \sum_{m'=1}^{M'} \nabla_\theta \log p_\theta(x_{m'})$
  Compute MMD Typicality for $x_{m'}$ by $\left\| \frac{1}{m'} \sum_{m'=1}^{M'} \log p_\theta(x_{m'}) - L(\theta) \right\|_2$ and add it to $T^{\text{typicality}}$
  Compute Score Statistic for $x_{m'}$ by $\left\| I(\theta)^{-1/2} \frac{1}{m'} \sum_{m'=1}^{M'} \nabla \log p_\theta(x_{m'}) \right\|_2$ and add it to T^{score}
Return Two vectors of size S containing the two statistics for $T^{\text{typicality}}$ and T^{score}

Compute $\hat{F}^{\text{typicality}}$ and $\hat{F}^{\text{score}}$, the two empirical CDFs, from $T^{\text{typicality}}$ and T^{score}. For example, we used `statsmodels` library (Seabold and Perktold, 2010).

Given a test set $\tilde{x}_1, \ldots, \tilde{x}_n$:
($n = 1$ corresponds to perform single-sample OOD detection)
  Compute $\frac{1}{n} \sum_{i=1}^{n} \log p_\theta(\tilde{x}_i)$ and $\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \log p_\theta(\tilde{x}_i)$
  Compute MMD Typicality $\tilde{t}$ and Score Statistic $\tilde{s}$
  Compute p-values $p_T = 1 - \hat{F}^{\text{typicality}}(\tilde{t})$ and $p_S = 1 - \hat{F}^{\text{score}}(\tilde{s})$
  Combine the two p-values using Fisher's method Eq. 5

CHAPTER 4

Riemannian Laplace approximations for Bayesian neural networks

This chapter presents our work "Riemannian Laplace approximations for Bayesian neural networks". In this work, we proposed a way to solve the problem of classic Laplace approximation getting samples from regions where the true posterior has low probability mass. We did this by borrowing tools from Riemannain geometry and equipping the parameter or weight space of a neural network with a simple Riemannian metric. If we consider the samples from classic Laplace as initial velocities starting from the MAP estimate, we can compute exponential maps as described in Section 2.3. The endpoints of these geodesic curves are our new samples. We show that this corresponds to performing Laplace approximation tangentially to the associated manifold. The final result is a Gaussian approximation that can locally adapt to the specific mode of the posterior distribution.

Riemannian Laplace approximations for Bayesian neural networks

Federico Bergamin, Pablo Moreno-Muñoz, Søren Hauberg, Georgios Arvanitidis
Section for Cognitive Systems, DTU Compute, Technical University of Denmark
{fedbe, pabmo, sohau, gear}@dtu.dk

Abstract

Bayesian neural networks often approximate the weight-posterior with a Gaussian distribution. However, practical posteriors are often, even locally, highly non-Gaussian, and empirical performance deteriorates. We propose a simple parametric approximate posterior that adapts to the shape of the true posterior through a Riemannian metric that is determined by the log-posterior gradient. We develop a Riemannian Laplace approximation where samples naturally fall into weight-regions with low negative log-posterior. We show that these samples can be drawn by solving a system of ordinary differential equations, which can be done efficiently by leveraging the structure of the Riemannian metric and automatic differentiation. Empirically, we demonstrate that our approach consistently improves over the conventional Laplace approximation across tasks. We further show that, unlike the conventional Laplace approximation, our method is not overly sensitive to the choice of prior, which alleviates a practical pitfall of current approaches.

1 Introduction

Bayesian deep learning estimates the weight-posterior of a neural network given data, i.e. $p(\theta|\mathcal{D})$. Due to the generally high dimensions of the weight-space, the normalization of this posterior is intractable and approximate inference becomes a necessity. The most common parametric choice approximates the posterior with a Gaussian distribution, $p(\theta|\mathcal{D}) \approx q(\theta|\mathcal{D}) = \mathcal{N}(\theta|\mu, \Sigma)$, which is estimated *variationally* [Blundell et al., 2015], using *Laplace approximations* [MacKay, 1992] or with other techniques [Maddox et al., 2019]. Empirical evidence, however, suggests that the log-posterior is not locally concave [Sagun et al., 2016], indicating that the Gaussian approximation is overly crude. Indeed, this approximation is known to be brittle as the associated covariance is typically ill-conditioned implying a suboptimal behavior [Daxberger et al., 2021a, Farquhar et al., 2020], and for this reason, alternative approaches have been proposed to fix this issue [Mackay, 1992]. Nonetheless, the Gaussian approximation is widely used due to the many benefits of parametric distributions, over e.g. *Monte Carlo sampling* [Neal, 1995] or *deep ensembles* [Lakshminarayanan et al., 2017].

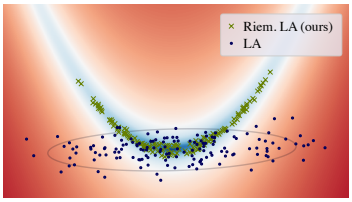


Figure 1: Our Riemannian Laplace approximation is a simple parametric distribution, which is shaped according to the local loss landscape through a Riemannian metric.

In this paper we argue that the underlying issue is not with the Gaussian approximation, but rather with the weight-space over which the approximation is applied. We show that a Gaussian approximation can locally adapt to the loss by equipping the weight-space with a simple Riemannian metric and performing the approximation tangentially to the associated manifold. Practically, this ensures that samples from the Riemannian approximate posterior land in regions of weight-space yielding low training loss, which significantly improves over the usual Gaussian approximation. We obtain our

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Riemannian approximate posterior using a generalization of the Laplace approximation [MacKay, 1992] to general Riemannian manifolds. Sampling from this distribution requires solving a system of ordinary differential equations, which we show can be performed efficiently by leveraging the structure of the used Riemannian metric and automatic differentiation. Empirically, we demonstrate that this significantly improves upon conventional Laplace approximations across tasks.

2 Background

Notation & assumptions. We consider independent and identically distributed (i.i.d.) data $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, consisting of inputs $\mathbf{x} \in \mathbb{R}^D$ and outputs $\mathbf{y} \in \mathbb{R}^C$. To enable *probabilistic modeling*, we use a likelihood $p(\mathbf{y}|f_\theta(\mathbf{x}))$ which is either Gaussian (regression) or categorical (classification). This likelihood is parametrized by a deep neural network $f_\theta : \mathbb{R}^D \to \mathbb{R}^C$, where $\theta \in \mathbb{R}^K$ represent the weights for which we specify a Gaussian prior $p(\theta)$. The predictive distribution of a new test point $\mathbf{x}'$ equals $p(\mathbf{y}|\mathbf{x}') = \int p(\mathbf{y}|\mathbf{x}', \theta)p(\theta|\mathcal{D})\mathrm{d}\theta$ where $p(\theta|\mathcal{D})$ is the true weight-posterior given the data $\mathcal{D}$. To ensure tractability, this posterior is approximated. This paper focuses on the Laplace approximation, though the bulk of the methodology applies to other approximation techniques as well.

2.1 The Laplace approximation

The Laplace approximation (LA) is widely considered in *probabilistic* models for approximating intractable densities [Bishop, 2007]. The idea is to perform a second-order Taylor expansion of an unnormalized log-probability density, thereby yielding a Gaussian approximation. When considering inference of the true posterior $p(\theta|\mathcal{D})$, LA constructs an approximate posterior distribution $q_{\text{LA}}(\theta|\mathcal{D}) = \mathcal{N}(\theta|\theta_*, \Sigma)$ that is centered at the *maximum a-posteriori* (MAP) estimate

$$\theta_* = \arg\max_\theta \{\log p(\theta|\mathcal{D})\} = \arg\min_\theta \underbrace{\left\{ -\sum_{n=1}^N \log p(\mathbf{y}_n \mid \mathbf{x}_n, \theta) - \log p(\theta) \right\}}_{\mathcal{L}(\theta)}. \tag{1}$$

A Taylor expansion around θ_* of the regularized loss $\mathcal{L}(\theta)$ then yields

$$\hat{\mathcal{L}}(\theta) \approx \mathcal{L}(\theta_*) + \langle \nabla_\theta \mathcal{L}(\theta)\big|_{\theta=\theta_*}, (\theta - \theta_*)\rangle + \frac{1}{2}\langle (\theta - \theta_*), \mathrm{H}_\theta[\mathcal{L}](\theta)\big|_{\theta=\theta_*}(\theta - \theta_*)\rangle, \tag{2}$$

where we know that $\nabla_\theta \mathcal{L}(\theta)\big|_{\theta=\theta_*} \approx 0$, and $\mathrm{H}_\theta[\mathcal{L}](\theta) \in \mathbb{R}^{K\times K}$ denotes the Hessian of the loss. This expansion suggests that the approximate posterior covariance should be the inverse Hessian $\Sigma = \mathrm{H}_\theta[\mathcal{L}](\theta)\big|^{-1}_{\theta=\theta_*}$. The marginal likelihood of the data is then approximated as $p(\mathcal{D}) \approx \exp(-\mathcal{L}(\theta_*))(2\pi)^{D/2}\det(\Sigma)^{1/2}$. This is commonly used for training hyper-parameters of both the likelihood and the prior [Immer et al., 2021a, Antorán et al., 2022]. We refer to appendix A for further details.

Tricks of the trade. Despite the simplicity of the Laplace approximation, its application to modern neural networks is not trivial. The first issue is that the Hessian matrix is too large to be stored in memory, which is commonly handled by approximately reducing the Hessian to being diagonal, low-rank, Kronecker factored, or only considered for a subset of parameters (see Daxberger et al. [2021a] for a review). Secondly, the Hessian is generally not positive definite [Sagun et al., 2016], which is commonly handled by approximating the Hessian with the generalized Gauss-Newton approximation [Foresee and Hagan, 1997, Schraudolph, 2002]. Furthermore, estimating the predictive distribution using Monte Carlo samples from the Laplace approximated posterior usually performs poorly [Lawrence, 2001, Chapter 5][Ritter et al., 2018] even for small models. Indeed, the Laplace approximation can place probability mass in low regions of the posterior. A solution, already proposed by [Mackay, 1992, Chapter 4], is to consider a first-order Taylor expansion around θ_*, and use the sample to use the "linearized" function $f^{\text{lin}}_\theta(\mathbf{x}) = f_{\theta_*}(\mathbf{x}) + \langle \nabla_\theta f_\theta(\mathbf{x})\big|_{\theta=\theta_*}, \theta - \theta_*\rangle$ as predictive, where $\nabla_\theta f_\theta(\mathbf{x})\big|_{\theta=\theta_*} \in \mathbb{R}^{C\times K}$ is the Jacobian. Recently, this approach has been justified by Khan et al. [2019], Immer et al. [2021b], who proved that the generalized Gauss-Newton approximation is the exact Hessian of this new linearized model. Even if this is a linear function with respect to the parameters θ, empirically it achieves better performance than the classic Laplace approximation.

Although not theoretically justified, optimizing the prior precision post-hoc has been shown to play a crucial role in the Laplace approximation [Ritter et al., 2018, Kristiadi et al., 2020, Immer et al., 2021a, Daxberger et al., 2021a]. This is usually done either using cross-validation or by maximizing the log-marginal likelihood. In principle, this regularizes the Hessian, and the associated approximate posterior concentrates around the MAP estimate.

Strengths & weaknesses. The main strength of the Laplace approximation is its simplicity in implementation due to the popularization of automatic differentiation. The Gaussian approximate posterior is, however, quite crude and often does not capture the shape locally of the true posterior [Sagun et al., 2016]. Furthermore, the common reduction of the Hessian to not correlate all model parameters limit the expressive power of the approximate posterior.

3 Riemannian Laplace approximations

We aim to construct a parametric approximate posterior that better reflects the local shape of the true posterior and captures nonlinear correlations between parameters. The basic idea is to retain the Laplace approximation but change the parameter space Θ to locally encode the *training loss*. To realize this idea, we will first endow the parameter space with a suitable Riemannian metric (Sec. 3.1) and then construct a Laplace approximation according to this metric (Sec. 3.2).

3.1 A loss-aware Riemannian geometry

For a given parameter value $\theta \in \Theta$, we can measure the training loss $\mathcal{L}(\theta)$ of the associated neural network. Assuming that the loss changes smoothly with θ, we can interpret the loss surface $\mathcal{M} = g(\theta) = [\theta, \mathcal{L}(\theta)] \in \mathbb{R}^{K+1}$ as a K-dimensional manifold in $\mathbb{R}^{K+1}$. The goal of Riemannian geometry [Lee, 2019, do Carmo, 1992] is to do calculations that are restricted to such manifolds.

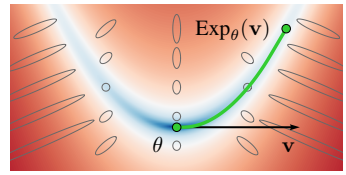


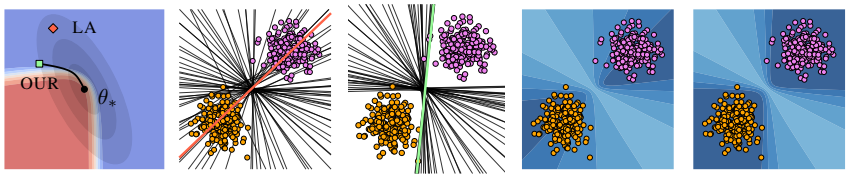
Figure 2: The parameter space Θ of the BNN together with examples of the Riemannian metric and the exponential map. Note that the Riemannian metric adapts to the shape of the loss which causes the geodesic to follow its shape.

The metric. We can think of the parameter space Θ as being the *intrinsic coordinates* of the manifold $\mathcal{M}$, and it is beneficial to do all calculations directly in these coordinates. Note that a vector tangential to the manifold can be written as $\mathbf{J}_g(\theta)\mathbf{v} \in \mathbb{R}^{K+1}$, where $\mathbf{J}_g : \Theta \to \mathbb{R}^{K+1 \times K}$ is the Jacobian of g that spans the tangent space $\mathcal{T}_{g(\theta)}\mathcal{M}$ at the point $g(\theta) \in \mathcal{M}$ and $\mathbf{v} \in \mathbb{R}^K$ is the vector of *tangential coordinates* for this basis of the tangent space. We can take inner products between two tangent vectors in the same tangent space as $\langle \mathbf{J}_g(\theta)\mathbf{v}_1, \mathbf{J}_g(\theta)\mathbf{v}_2 \rangle = \mathbf{v}_1^\intercal \mathbf{J}_g(\theta)^\intercal \mathbf{J}_g(\theta)\mathbf{v}_2$, which, we note, is now expressed directly in the intrinsic coordinates. From this observation, we define the *Riemannian metric* $\mathbf{M}(\theta) = \mathbf{J}_g(\theta)^\intercal \mathbf{J}_g(\theta)$, which gives us a notion of a local inner product in the intrinsic coordinates of the manifold (see ellipsoids in Fig. 2). The Jacobian of g is particularly simple $\mathbf{J}_g(\theta) = [\mathbb{I}_K, \nabla_\theta \mathcal{L}]^\intercal$, such that the metric takes the form

$$\mathbf{M}(\theta) = \mathbb{I}_K + \nabla_\theta \mathcal{L}(\theta) \nabla_\theta \mathcal{L}(\theta)^\intercal. \tag{3}$$

The exponential map. A local inner product allows us to define the length of a curve $c : [0, 1] \to \Theta$ as $\texttt{length}[c] = \int_0^1 \sqrt{\langle \dot{c}(t), \mathbf{M}(c(t))\dot{c}(t) \rangle} \mathrm{d}t$, where $\dot{c}(t) = \partial_t c(t)$ is the velocity. From this, the *distance* between two points can be defined as the length of the shortest connecting curve, where the latter is known as the *geodesic curve*. Such geodesics can be expressed as solutions to a system of second-order non-linear ordinary differential equations (ODEs), which is given in appendix B alongside further details on geometry. Of particular interest to us is the *exponential map*, which solves these ODEs subject to an initial position and velocity. This traces out a geodesic curve with a given starting point and direction (see Fig. 2). Geometrically, we can also think of this as mapping a tangent vector *back to the manifold*, and we write the map as $\mathrm{Exp} : \mathcal{M} \times \mathcal{T}_\theta \mathcal{M} \to \mathcal{M}$.

The tangential coordinates $\mathbf{v}$ can be seen as a coordinate system for the neighborhood around θ, and since the exponential map is locally a bijection we can represent any point locally with a unique tangent vector. However, these coordinates correspond to the tangent space that is spanned by $\mathbf{J}_g(\theta)$, which implies that by changing this basis the associated coordinates change as well. By



(a) True posterior (b) LA sample (c) OUR sample (d) LA model (e) OUR model

Figure 3: The LA assigns probability mass to regions where the true posterior is nearly zero, and a sample from this region corresponds to a poor classifier. Considering this sample as the initial velocity for the exponential map, the generated sample falls within the true posterior and the associated classifier performs well. As a result, our model quantifies better the uncertainty.

orthonormalizing this basis we get the *normal coordinates* where the metric vanishes. Let $\mathbf{v}$ the tangential coordinates and $\bar{\mathbf{v}}$ the corresponding normal coordinates, then it holds that

$$\langle \mathbf{v}, \mathbf{M}(\theta)\mathbf{v} \rangle = \langle \bar{\mathbf{v}}, \bar{\mathbf{v}} \rangle \Rightarrow \mathbf{v} = \mathbf{A}(\theta)\bar{\mathbf{v}} \quad \text{with} \quad \mathbf{A}(\theta) = \mathbf{M}(\theta)^{-1/2}. \tag{4}$$

We will use the normal coordinates when doing Taylor expansions of the log-posterior, akin to standard Laplace approximations.

3.2 The proposed approximate posterior

In order to Taylor-expand the loss according to the metric, we first express the loss in normal coordinates of the tangent space at θ_*, $h(\bar{\mathbf{v}}) = \mathcal{L}(\mathrm{Exp}_{\theta_*}(\mathbf{M}(\theta_*)^{-1/2}\bar{\mathbf{v}}))$. Following the standard Laplace approximation, we perform a second-order Taylor expansion of h as

$$\hat{h}(\bar{\mathbf{v}}) \approx h(0) + \langle \partial_{\bar{\mathbf{v}}} h(\bar{\mathbf{v}})\big|_{\bar{\mathbf{v}}=0}, \bar{\mathbf{v}} \rangle + \frac{1}{2}\langle \bar{\mathbf{v}}, \mathrm{H}_{\bar{\mathbf{v}}}[h](\bar{\mathbf{v}})\big|_{\bar{\mathbf{v}}=0}\bar{\mathbf{v}} \rangle, \tag{5}$$

where $\partial_{\bar{\mathbf{v}}} h(\bar{\mathbf{v}})\big|_{\bar{\mathbf{v}}=0} = \mathbf{A}(\theta_*)^\intercal \nabla_\theta \mathcal{L}(\theta)\big|_{\theta=\theta_*} \approx 0$ as θ_* minimize the loss and $\mathrm{H}_{\bar{\mathbf{v}}}[h](\bar{\mathbf{v}})\big|_{\bar{\mathbf{v}}=0} = \mathbf{A}(\theta_*)^\intercal \mathrm{H}_\theta[\mathcal{L}](\theta)\mathbf{A}(\theta_*)\big|_{\theta=\theta_*}$ with $\mathrm{H}_\theta[\mathcal{L}](\theta)$ the standard Euclidean Hessian matrix of the loss. Further details about this step can be found in appendix B.

Tangential Laplace. Similar to the standard Laplace approximation, we get a Gaussian approximate posterior $\bar{q}(\bar{\mathbf{v}}) = \mathcal{N}(\bar{\mathbf{v}} \mid 0, \overline{\Sigma})$ on the tangent space in the normal coordinates with covariance $\overline{\Sigma} = \mathrm{H}_{\bar{\mathbf{v}}}[h](\bar{\mathbf{v}})\big|^{-1}_{\bar{\mathbf{v}}=0}$. Note that changing the normal coordinates $\bar{\mathbf{v}}$ to tangential coordinates $\mathbf{v}$ is a linear transformation and hence $\mathbf{v} \sim \mathcal{N}(0, \mathbf{A}(\theta_*)\overline{\Sigma}\mathbf{A}(\theta_*)^\intercal)$, which means that this covariance is equal to $\mathrm{H}_\theta[\mathcal{L}](\theta)\big|^{-1}_{\theta=\theta_*}$ since $\mathbf{A}(\theta_*)$ is a symmetric matrix, and hence, it cancels out. The approximate posterior $q_\mathcal{T}(\mathbf{v}) = \mathcal{N}(\mathbf{v} \mid 0, \Sigma)$ in tangential coordinates, thus, matches the covariance of the standard Laplace approximation.

The predictive posterior. We can approximate the predictive posterior distribution using Monte Carlo integration as $p(y|\mathbf{x}', \mathcal{D}) = \int p(y|\mathbf{x}', \mathcal{D}, \theta) q(\theta) \mathrm{d}\theta = \int p(y|\mathbf{x}', \mathcal{D}, \mathrm{Exp}_{\theta_*}(\mathbf{v})) q_\mathcal{T}(\mathbf{v}) \mathrm{d}\mathbf{v} \approx \frac{1}{S}\sum_{s=1}^{S} p(y|\mathbf{x}', \mathcal{D}, \mathrm{Exp}_{\theta_*}(\mathbf{v}_s)),\ \mathbf{v}_s \sim q_\mathcal{T}(\mathbf{v})$. Intuitively, this generates tangent vectors according to the standard Laplace approximation and maps them back to the manifold by solving the geodesic ODE. This lets the Riemannian approximate posterior take shape from the loss landscape, which is largely ignored by the standard Laplace approximation. We emphasize that this is a general construction that applies to the same Bayesian inference problems as the standard Laplace approximation and is not exclusive to Bayesian neural networks.

The above analysis also applies to the linearized Laplace approximation. In particular, when the $f^{\mathrm{lin}}_\theta(\mathbf{x})$ is considered instead of the $f_\theta(\mathbf{x})$ the loss function in (1) changes to $\mathcal{L}^{\mathrm{lin}}(\theta)$. Consequently, our Riemannian metric is computed under this new loss, and $\nabla_\theta \mathcal{L}^{\mathrm{lin}}(\theta)$ appears in the metric (3).

Example. To build intuition, we consider a logistic regressor on a linearly separable dataset (Fig. 3). The likelihood of a point $\mathbf{x} \in \mathbb{R}^2$ to be in one class is $p(C = 1|\mathbf{x}) = \sigma(\mathbf{x}^\intercal\theta + b)$, where $\sigma(\cdot)$ is the `sigmoid` function, $\theta \in \mathbb{R}^2$ and $b \in \mathbb{R}$. After learning the parameters, we fix b_* and show the posterior with respect to θ together with the corresponding standard Laplace approximation (Fig. 3a).

We see that the approximation assigns significant probability mass to regions where the true posterior is near-zero, and the result of a corresponding sample is a poor classifier (Fig. 3b). Instead, when we consider this sample as the initial velocity and compute the associated geodesic with the exponential map, we generate a sample at the tails of the true posterior which corresponds to a well-behaved model (Fig. 3c). We also show the predictive distribution for both approaches and even if both solve easily the classification problem, our model better quantifies uncertainty (Fig. 3e).

3.3 Efficient implementation

Our approach is a natural extension of the standard Laplace approximation, which locally adapts the approximate posterior to the true posterior. The caveat is that computational cost increases since we need to integrate an ODE for every sample. We now discuss partial alleviations.

Integrating the ODE. In general, the system of second-order nonlinear ODEs (see appendix B for the general form) is non-trivial as it depends on the geometry of the loss surface, which is complicated in the over-parametrized regime [Li et al., 2018]. In addition, the dimensionality of the parameter space is high, which makes the solution of the system even harder. Nevertheless, due to the structure of our Riemannian metric (3), the ODE simplifies to

$$\ddot{c}(t) = -\nabla_\theta \mathcal{L}(c(t)) \left(1 + \nabla_\theta \mathcal{L}(c(t))^\intercal \nabla_\theta \mathcal{L}(c(t))\right)^{-1} \langle \dot{c}(t), \mathrm{H}_\theta[\mathcal{L}](c(t))\dot{c}(t) \rangle, \tag{6}$$

which can be integrated reasonably efficiently with standard solvers. In certain cases, this ODE can be further simplified, for example when we consider the linearized loss $\mathcal{L}^{\text{lin}}(\theta)$ and Gaussian likelihood.

Automatic-differentiation. The ODE (6) requires computing both gradient and Hessian, which are high-dimensional objects for modern neural networks. While we need to compute the gradient explicitly, we do not need to compute and store the Hessian matrix, which is infeasible for large networks. Instead, we rely on modern automatic-differentiation frameworks to compute the Hessian-vector product between $\mathrm{H}_\theta[\mathcal{L}](c(t))$ and $\dot{c}(t)$ directly. This both reduces memory use, increases speed, and simplifies the implementation.

Mini-batching. The cost of computing the metric, and hence the ODE scales linearly with the number of training data, which can be expensive for large datasets. A reasonable approximation is to mini-batch the estimation of the metric when generating samples, i.e. construct a batch $\mathcal{B}$ of B random data points and use the associated loss in the ODE (6). As usual, we assume that $\mathcal{L}(\theta) \approx (N/B)\mathcal{L}_\mathcal{B}(\theta)$. Note that we only mini-batch the metric and not the covariance of our approximate posterior $q_\mathcal{T}(\mathbf{v})$.

We analyze the influence of mini-batching in our methods and provide empirical evidence in Fig. 4. In principle, the geometry of the loss surface $\mathcal{L}(\theta)$ controls the geodesics via the associated Riemannian metric, so when we consider the full dataset we expect the samples to behave similarly to $f_{\theta_*}(\mathbf{x})$. In other words, our approximate posterior generates weights near θ_* resulting in models with similar or even better loss. When we consider a batch the geometry of the associated loss surface $\mathcal{L}_\mathcal{B}(\theta)$ controls the generated geodesic. So if the batch represents well the structure of the full dataset, then the resulting model will be meaningful with respect to the original problem, and in addition, it may exhibit some variation that is beneficial from the Bayesian perspective for the quantification of the uncertainty. The same concept applies in the linearized version, with the difference that when the full dataset is considered the geometry of $\mathcal{L}^{\text{lin}}(\theta)$ may over-regularize the geodesics. Due to the linear nature of $f_\theta^{\text{lin}}(\theta)$ the associated Riemannian metric is small only close to θ_* so the generated samples are similar to $f_{\theta_*}(\mathbf{x})$. We relax this behavior and potentially introduce variations in the resulting models when we consider a different batch whenever we generate a sample. Find more details in appendix D.

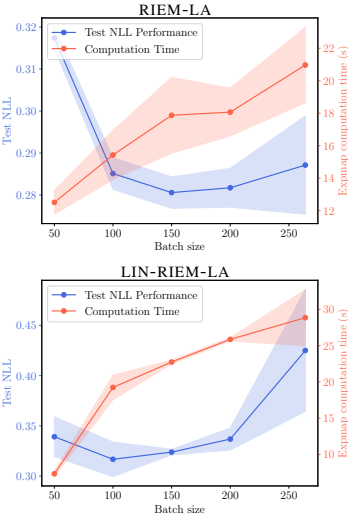


Figure 4: Analysis of mini-batching

4 Related work

Bayesian neural networks. Exact inference for BNNs is generally infeasible when the number of parameters is large. Several methods rely on approximate inference, which differs in their trade-off between computational cost and the goodness of the approximation. These techniques are usually based on the Laplace approximation [MacKay, 1992], variational inference [Graves, 2011, Blundell et al., 2015, Khan et al., 2018], dropout [Gal and Ghahramani, 2016], stochastic weight averaging [Izmailov et al., 2018, Maddox et al., 2019] or Monte Carlo based methods [Neal, 1995], where the latter is often more expensive.

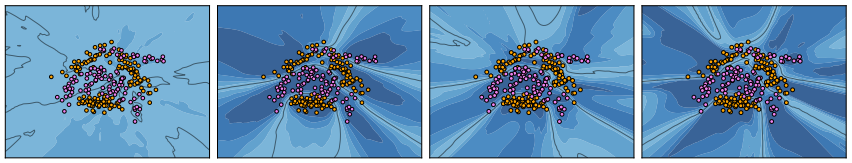
Laplace approximations. In this work, we are primarily focused on Laplace approximations, although the general geometric idea can be used in combination with any other inference approach listed above. Particularly, Laplace's method for BNNs was first proposed by Mackay [1992] in his *evidence* framework, where a closed-form approximation of predictive probabilities was also derived. This one uses a first-order Taylor expansion, also known as *linearization* around the MAP estimate. For long, Laplace's method was infeasible for modern architectures with large networks due to the exact computation of the Hessian. The seminal works of Martens and Grosse [2015] and Botev et al. [2017] made it possible to approximate the Hessian of large networks, which made Laplace approximations feasible once more [Ritter et al., 2018]. More recently, the Laplace approximation has become a go-to tool for turning trained neural networks into BNNs in a *post-hoc* manner, thanks to easy-to-use software [Daxberger et al., 2021a] and new approaches to scale up computation [Antorán et al., 2022]. In this direction, other works have only considered a subset of the network parameters [Daxberger et al., 2021b, Sharma et al., 2023], especially the last-layer. This is *de facto* the only current method competitive with *ensembles* [Lakshminarayanan et al., 2017].

Posterior refinement. Much work has gone into building more expressive approximate posteriors. Recently, Kristiadi et al. [2022] proposed to use normalizing flows to get a non-Gaussian approximate distribution using the Laplace approximation as a base distribution. Although this requires training an additional model, they showed that few bijective transformations are enough to improve the last-layer posterior approximation. Immer et al. [2021b], instead, propose to refine the Laplace approximation by using Gaussian variational Bayes or a Gaussian process. This still results in a Gaussian distribution, but it has proven beneficial for linearized Laplace approximations. Other approaches rely on a mixture of distributions to improve the goodness of the approximation. Miller et al. [2017] expand a variational approximation iteratively adding components to a mixture, while Eschenhagen et al. [2021] use a weighted sum of posthoc Laplace approximations generated from different pre-trained networks. Havasi et al. [2021], instead, introduces auxiliary variables to make a local refinement of a mean-field variational approximation.

Differential geometry. Differential geometry is increasingly playing a role in inference. Arvanitidis et al. [2016] make a Riemannian normal distribution locally adapt to data by learning a suitable Riemannian metric from data. In contrast, our metric is derived from the model. This is similar in spirit to work that investigates pull-back metrics in latent variable models [Tosi et al., 2014, Arvanitidis et al., 2018, Hauberg, 2018b]. A similar Riemannian metric has been used as a surrogate for the Fisher metric for Riemannian Hamiltomian Monte-Carlo sampling [Hartmann et al., 2022]. In addition to that, the geometry of the latent parameter space of neural networks was recently analyzed by Kristiadi et al. [2023] focusing on the invariance of flatness measures with respect to re-parametrizations. Finally, we note that Hauberg [2018a] considers Laplace approximations on the sphere as part of constructing a recursive Kalman-like filter.

5 Experiments

We evaluate our Riemannian LA (RIEM-LA) using illustrative examples, image datasets where we use a convolutional architecture, and real-world classification problems. We compare our method and its linearized version to standard and linearized LA. All predictive distributions are approximated using Monte Carlo (MC) samples. Although last-layer LA is widely used lately, we focus on approximating the posterior of all the weights of the network. In all experiments, we maximize the marginal log-likelihood to tune the hyperparameters of the prior and the likelihood as proposed in [Daxberger et al., 2021a]. To evaluate the performance in terms of uncertainty estimation we considered the standard metrics in the literature: negative log-likelihood (NLL), the Brier score (BRIER), the expected calibration error (ECE), and the maximum calibration error (MCE). More experiments are



(a) VANILLA LA (b) RIEM-LA (c) LIN-LA (d) LIN-RIEM-LA

Figure 6: Binary classification confidence estimate using 100 Monte-Carlo samples on the banana dataset. Vanilla LA underfit, while the other three methods are able to be certain within the data and uncertain far away. Note, for linearized RIEM-LA we solve the expmap using a different subset of the data. Confidence plots of all different methods can be found in the supplementary material. Black lines are the decision boundaries.

available in appendix D together with the complete training and modeling details. In appendix C.2 we analyze the runtime to compute the exponential map over different dataset and model sizes. Code to reproduce the results is publicly available at `https://github.com/federicobergamin/riemannian-laplace-approximation`

5.1 Regression problem

We consider the toy-regression problem proposed by Snelson and Ghahramani [2005]. The dataset contains 200 data points, and we randomly pick 150 examples as our training set and the remaining 50 as a test set. As shown by Lawrence [2001], Ritter et al. [2018], using samples from the LA posterior performs poorly in regression even if the Hessian is not particularly ill-conditioned, i.e. when the prior precision is optimized. For this reason, the linearization approach is necessary for regression with standard LA. Instead, we show that even our basic approach fixes this problem when the prior is optimized. We tested our approach by considering two fully connected networks, one with one hidden layer with 15 units and one with two layers with 10 units each, both with `tanh` activations. Our approach approximates well the true posterior locally, so the resulting function samples follow the data. Of course, if the Hessian is extremely degenerate our approach also suffers, as the initial velocities are huge. When we consider the linearized version of our approach the result is the same as the standard LA-linearization, which we include in the appendix D, where we also report results for in-between uncertainty as proposed by Foong et al. [2019] and a comparison with Hamiltonian Monte Carlo.

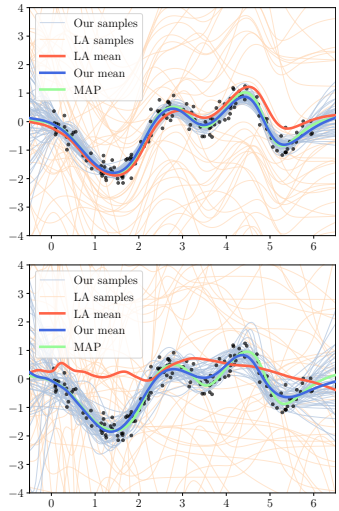


Figure 5: Posterior samples under a simple *(top)* and an overparametrized model *(bottom)*. Vanilla LA is known to generate bad models, while our samples from RIEM-LA quantify well the uncertainty.

5.2 Classification problems

Illustrative example. We consider a 2-dimensional binary classification problem using the banana dataset which is shown in Fig. 6. We train a 2-layer fully connected neural net with 16 hidden units per layer and `tanh` activation. For all methods, we use 100 MC samples for the predictive distribution.

As in regression, direct samples from the vanilla LA lead to a really poor model (Fig. 6a) with high uncertainty both within and away from the data support. Instead, the other three methods (Fig. 6b-6d) show a better-behaved confidence that decreases outside of the data support. This is also supported by the metrics in Table 1, where remarkably RIEM-LA performs better in terms of NLL and Brier score on a separate test set.

Table 1: In-distribution results in the banana dataset. We use 100 MC samples both for the LA and our variants. ECE and MCE are computed using $M = 10$ bins. We report mean and standard error over 5 different seeds.

	BANANA DATASET					
	PRIOR OPTIMIZED			PRIOR NOT OPTIMIZED		
METHOD	Accuracy ↑	NLL↓	Brier↓	Accuracy ↑	NLL↓	Brier↓
MAP	86.69 ± 0.34	0.333 ± 0.005	0.0930 ± 0.0015	86.69 ± 0.34	0.333 ± 0.005	0.0930 ± 0.0015
VANILLA LA	59.50 ± 5.07	0.678 ± 0.009	0.2426 ± 0.0046	48.85 ± 2.32	0.700 ± 0.003	0.2534 ± 0.0017
LIN-LA	86.99 ± 0.37	0.325 ± 0.008	0.0956 ± 0.0023	$\mathbf{86.92 \pm 0.40}$	0.403 ± 0.012	0.1196 ± 0.0044
RIEM-LA	$\mathbf{87.57 \pm 0.07}$	$\mathbf{0.287 \pm 0.002}$	$\mathbf{0.0886 \pm 0.0006}$	$\mathbf{87.14 \pm 0.20}$	$\mathbf{0.285 \pm 0.001}$	$\mathbf{0.0878 \pm 0.0006}$
RIEM-LA (BATCHES)	87.30 ± 0.08	$\mathbf{0.286 \pm 0.001}$	$\mathbf{0.0890 \pm 0.0000}$	$\mathbf{87.32 \pm 0.17}$	$\mathbf{0.294 \pm 0.002}$	$\mathbf{0.0895 \pm 0.0004}$
LIN-RIEM-LA	87.02 ± 0.38	0.415 ± 0.029	0.0967 ± 0.0024	85.33 ± 0.31	0.884 ± 0.037	0.1252 ± 0.0022
LIN-RIEM-LA (BATCHES)	$\mathbf{87.77 \pm 0.24}$	0.298 ± 0.006	$\mathbf{0.0887 \pm 0.0011}$	86.16 ± 0.21	0.352 ± 0.002	0.0994 ± 0.0011

As we discussed in Sec. 3.3, using a subset of the dataset for computing the exponential map can be beneficial for our linearized manifold in addition to speeding up computation. In Fig. 6d we plot the confidence for our linearized approach using batches while in appendix D we show the confidence of the same approach using the full data for solving the ODEs. We can see that our linearized RIEM-LA tends to be overconfident outside the data region and also close to the decision boundary. This behaviour can be found in the high NLL that linearized RIEM-LA gets compared to our vanilla approach and linearized LA.

UCI datasets. We compare our approach against the standard LA on a set of six UCI classification datasets using a fully connected network with a single layer, 50 hidden units and `tanh` activation. The predictive distribution is estimated using MC with 30 samples from the approximate posterior of each approach. In Table 2 we compare the methods in terms of their negative log-likelihood (NLL) in the test set. All other metrics are reported in appendix D. We are considering the setting where we optimize the prior-precision post-hoc, which is the optimal setting for LA and linearized LA. We consider our standard approaches without using batches, which we have seen that specifically for our linearized approach may lead to sub-optimal performance.

From the results in Table 2 we see that our RIEM-LA consistently performs better in terms of negative log-likelihood than vanilla and linearized LA. We also observe that in two datasets the performance of our linearized RIEM-LA is not optimal. This implies that the loss surface of the linearized loss potentially over-regularizes the geodesics as we analyzed in Sec. 3.3, and in this case, considering mini-batching could have been beneficial.

Table 2: Negative log-likelihood (lower is better) on UCI datasets for classification. Predictive distribution is estimated using 30 MC samples. Mean and standard error over 5 different seeds.

	PRIOR PRECISION OPTIMIZED				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	0.975 ± 0.081	1.209 ± 0.020	$\mathbf{0.454 \pm 0.024}$	0.875 ± 0.020	$\mathbf{0.494 \pm 0.044}$
GLASS	2.084 ± 0.323	1.737 ± 0.037	$\mathbf{1.047 \pm 0.224}$	1.365 ± 0.058	1.359 ± 0.299
IONOSPHERE	1.032 ± 0.175	0.673 ± 0.013	$\mathbf{0.344 \pm 0.068}$	0.497 ± 0.015	0.625 ± 0.110
WAVEFORM	1.076 ± 0.110	0.888 ± 0.030	$\mathbf{0.459 \pm 0.057}$	0.640 ± 0.002	0.575 ± 0.065
AUSTRALIAN	1.306 ± 0.146	0.684 ± 0.011	$\mathbf{0.541 \pm 0.053}$	$\mathbf{0.570 \pm 0.016}$	0.833 ± 0.108
BREAST CANCER	$\mathbf{0.225 \pm 0.076}$	0.594 ± 0.030	$\mathbf{0.176 \pm 0.092}$	0.327 ± 0.022	$\mathbf{0.202 \pm 0.073}$

Image classification. We consider a small convolutional neural network on MNIST and FashionMNIST. Our network consists of two convolutional layers followed by average pooling layers and three fully connected layers. We consider a model of this size as the high dimensionality of the parameter space is one of the main limitations of the ODE solver. For the training of the model, we subsample each dataset and we consider 5000 observations by keeping the proportionality of labels, and we test in the full test set containing 8000 examples. In Table 3 we compare the different methods with the prior precision optimized as this is the ideal setting for the linearized LA. We refer to appendix D for the setting with the prior precision not optimized.

From the results we observe that our standard RIEM-LA performs better than all the other methods in terms of NLL and Brier score, meaning that the models are better calibrated, but it also leads to a more accurate classifier than the MAP. In terms of ECE, it seems that considering the linearized

Table 3: Image classification results using a CNN on MNIST and FashionMNIST. The network is trained on 5000 examples and we test the in-distribution performance on the test set, which contains 8000 examples. We use 25 Monte Carlo samples to approximate the predictive distribution and 1000 datapoints per batch in our batched manifolds. Calibration metrics are computed using $M = 15$ bins. We report mean and standard error for each metric over 3 different seeds.

	CNN ON MNIST - PRIOR PRECISION OPTIMIZED				
METHOD	Accuracy ↑	NLL↓	Brier↓	ECE↓	MCE↓
MAP	95.02 ± 0.17	0.167 ± 0.005	0.0075 ± 0.0002	1.05 ± 0.14	39.94 ± 14.27
VANILLA LA	88.69 ± 1.84	0.871 ± 0.026	0.0393 ± 0.0013	42.11 ± 1.22	50.52 ± 1.45
LIN-LA	94.91 ± 0.26	0.204 ± 0.006	0.0087 ± 0.0003	6.30 ± 0.08	39.30 ± 16.77
RIEM-LA	$\mathbf{96.74 \pm 0.12}$	$\mathbf{0.115 \pm 0.003}$	$\mathbf{0.0052 \pm 0.0002}$	2.48 ± 0.06	38.03 ± 15.02
RIEM-LA (BATCHES)	95.67 ± 0.19	0.170 ± 0.005	0.0072 ± 0.0002	5.40 ± 0.06	22.40 ± 0.51
LIN-RIEM-LA	95.44 ± 0.18	0.149 ± 0.004	0.0068 ± 0.0003	$\mathbf{0.66 \pm 0.03}$	39.40 ± 14.75
LIN-RIEM-LA (BATCHES)	95.14 ± 0.20	0.167 ± 0.004	0.0076 ± 0.0002	3.23 ± 0.04	$\mathbf{18.10 \pm 2.50}$

	CNN ON FASHIONMNIST - PRIOR PRECISION OPTIMIZED				
METHOD	Accuracy ↑	NLL↓	Brier↓	ECE↓	MCE↓
MAP	79.88 ± 0.09	0.541 ± 0.002	0.0276 ± 0.0000	$\mathbf{1.66 \pm 0.07}$	24.07 ± 1.50
VANILLA LA	74.88 ± 0.83	1.026 ± 0.046	0.0482 ± 0.0019	31.63 ± 1.28	43.61 ± 2.95
LIN-LA	79.85 ± 0.13	0.549 ± 0.001	0.0278 ± 0.0000	3.23 ± 0.44	37.88 ± 17.98
RIEM-LA	$\mathbf{83.33 \pm 0.17}$	$\mathbf{0.472 \pm 0.001}$	$\mathbf{0.0237 \pm 0.0001}$	3.13 ± 0.48	10.94 ± 2.11
RIEM-LA (BATCHES)	81.65 ± 0.18	0.525 ± 0.004	0.0263 ± 0.0002	5.80 ± 0.73	35.30 ± 18.40
LIN-RIEM-LA	81.33 ± 0.10	0.521 ± 0.004	0.0261 ± 0.0002	$\mathbf{1.59 \pm 0.40}$	25.53 ± 0.10
LIN-RIEM-LA (BATCHES)	80.49 ± 0.13	0.529 ± 0.003	0.0269 ± 0.0002	2.10 ± 0.42	$\mathbf{6.14 \pm 1.42}$

approach is beneficial in producing better-calibrated models in both datasets. This holds both for our approach linearized RIEM-LA and the standard LA. Optimizing the prior precision post-hoc is crucial for the vanilla LA and associated results can be seen in appendix D. Instead, both our methods appear to be robust and consistent, as they achieve similar performance no matter if the prior precision is optimized or not.

Note that for the mini-batches for our approaches, we consider 20% of the data by randomly selecting 1000 observations while we respect the label frequency based on the full dataset. Clearly, the batch-size is a hyperparameter for our methods and can be estimated systematically using cross-validation. Even if we do not optimize this hyperparameter, we see that our batched version of RIEM-LA and LIN-RIEM-LA perform better than the standard LA and on-par with our LIN-RIEM-LA without batches, implying that a well-tuned batch-size can potentially further improve the performance. Nevertheless, this also shows that our method is robust with respect to the batch-size.

6 Conclusion and future directions

We propose an extension to the standard Laplace approximation, which leverages the natural geometry of the parameter space. Our method is parametric in the sense that a Gaussian distribution is estimated using the standard Laplace approximation, but it adapts to the true posterior through a nonparametric Riemannian metric. This is a general mechanism that, in principle, can also apply to, e.g., variational approximations. In a similar vein, while the focus of our work is on Bayesian neural networks, nothing prevents us from applying our method to other model classes.

Empirically, we find that our Riemannian Laplace approximation is better or on par with alternative Laplace approximations. The standard Laplace approximation crucially relies on both linearization and on a fine-tuned prior to give useful posterior predictions. Interestingly, we find that the Riemannian Laplace approximation requires neither. This could suggest that the standard Laplace approximation has a rather poor posterior fit, which our adaptive approach alleviates.

Limitations. The main downside of our approach is the computational cost involved in integrating the ODE, which is a common problem in computational geometry [Arvanitidis et al., 2019]. The cost of evaluating the ODE scales linearly with the number of observations, and in particular it is $O(SWN)$, where S is the number of steps of the solver, W is the number of model parameters and N is the dataset size. Indeed, the solver needs all the training data points to compute both gradient and hvp at each step. We refer to appendix C.1 for a detailed explanation. For big datasets, we have considered

the "obvious" trick of using a random (small) batch of the data when solving the ODE to reduce the complexity. Empirically, we find that this introduces some stochasticity in the sampling, which sometimes is beneficial to explore the posterior distribution better and eventually boost performance, motivating further research. The computational cost also grows with the dimensionality of the parameter space, as the number of necessary solver steps increases, as well as the cost of the Hessian vector products. Our implementation relies on an off-the-shelf ODE solver which runs on the CPU while our automatic-differentiation based approach runs on the GPU, which is inefficient. We expect significant improvements by using an ODE solver that runs exclusively on GPU, while tailor-made numerical integration methods are also of particular interest.

Future directions. We showed that equipping the weight-space with a simple Riemannian metric is a promising approach that solves some of the classic LA issues. We believe that this opens up different interesting research directions that either aim to use different metrics instead of the one used in this work or to make this method scale to bigger dataset. Regarding the metric, a possible extension of this work would be to consider the Fisher information matrix in the parameter space. Potential ideas to improve efficiency is to consider approximations of the Riemannian metric leading to simpler ODE systems, for example by using the KFAC instead of the full-Hessian in (6). Another directions, instead, would be to focus on developing a better solver. Indeed, we know the structure and behavior of our ODE system, i.e. geodesics start from low loss which increases along the curve, therefore, a potential direction would be to develop solvers that exploit this information. Usually, general purpose solvers aim for accuracy, while in our case even inexact solutions could be potentially useful if computed fast.

Acknowledgments and Disclosure of Funding

This work was funded by the Innovation Fund Denmark (0175-00014B) and the Novo Nordisk Foundation through the Center for Basic Machine Learning Research in Life Science (NNF20OC0062606). It also received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research, innovation programme (757360). SH was supported in part by a research grant (42062) from VILLUM FONDEN.

References

P.-A. Absil, R. Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.

Javier Antorán, David Janz, James U Allingham, Erik Daxberger, Riccardo Rb Barbano, Eric Nalisnick, and José Miguel Hernández-Lobato. Adapting the linearised laplace model evidence for modern deep learning. In *International Conference on Machine Learning (ICML)*, 2022.

Georgios Arvanitidis, Lars K Hansen, and Søren Hauberg. A locally adaptive normal distribution. In *Neural Information Processing Systems (NeurIPS)*, 2016.

Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. Latent space oddity: on the curvature of deep generative models. In *International Conference on Learning Representations (ICLR)*, 2018.

Georgios Arvanitidis, Søren Hauberg, Philipp Hennig, and Michael Schober. Fast and robust shortest paths on manifolds learned from data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning (ICML)*, 2015.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *International Conference on Machine Learning (ICML)*, 2017.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux–effortless Bayesian deep learning. In *Neural Information Processing Systems (NeurIPS)*, 2021a.

Erik Daxberger, Eric Nalisnick, James U Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Bayesian deep learning via subnetwork inference. In *International Conference on Machine Learning (ICML)*, 2021b.

M.P. do Carmo. *Riemannian Geometry*. Mathematics (Boston, Mass.). Birkhäuser, 1992.

John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Runa Eschenhagen, Erik Daxberger, Philipp Hennig, and Agustinus Kristiadi. Mixtures of Laplace approximations for improved post-hoc uncertainty in deep learning. In *NeurIPS Workshop of Bayesian Deep Learning*, 2021.

Sebastian Farquhar, Michael A Osborne, and Yarin Gal. Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

Andrew YK Foong, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. 'In-Between' Uncertainty in Bayesian Neural Networks. *arXiv preprint arXiv:1906.11537*, 2019.

F Dan Foresee and Martin T Hagan. Gauss-Newton approximation to Bayesian learning. In *Proceedings of International Conference on Neural Networks (ICNN)*. IEEE, 1997.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, 2016.

Alex Graves. Practical variational inference for neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2011.

Marcelo Hartmann, Mark Girolami, and Arto Klami. Lagrangian manifold monte carlo on monge patches. In *International Conference on Artificial Intelligence and Statistics*, 2022.

Søren Hauberg. Directional statistics with the spherical normal distribution. In *FUSION 2018*, 2018a.

Søren Hauberg. Only bayes should learn a manifold. *arXiv preprint*, 2018b.

Marton Havasi, Jasper Snoek, Dustin Tran, Jonathan Gordon, and José Miguel Hernández-Lobato. Refining the variational posterior through iterative optimization. *Entropy*, 2021.

E Heirer, SP Nørsett, and G Wanner. Solving ordinary differential equations i: Nonstiff problems, 1987.

Philipp Hennig and Søren Hauberg. Probabilistic solutions to differential equations and their application to riemannian statistics. In *Proceedings of the 17th international Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

Richard Zou Horace He. functorch: Jax-like composable function transforms for pytorch. https://github.com/pytorch/functorch, 2021.

Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Khan Mohammad Emtiyaz. Scalable marginal likelihood estimation for model selection in deep learning. In *International Conference on Machine Learning (ICML)*, 2021a.

Alexander Immer, Maciej Korzepa, and Matthias Bauer. Improving predictions of Bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021b.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Uncertainty in Artificial Intelligence (UAI)*, 2018.

Mohammad Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In *International Conference on Machine Learning (ICML)*, 2018.

Mohammad Emtiyaz E Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into Gaussian Processes. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR*, 2015.

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in ReLU networks. In *International Conference on Machine Learning (ICML)*, 2020.

Agustinus Kristiadi, Runa Eschenhagen, and Philipp Hennig. Posterior refinement improves sample efficiency in bayesian neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Agustinus Kristiadi, Felix Dangel, and Philipp Hennig. The geometry of neural nets' parameter spaces under reparametrization. *arXiv preprint*, 2023.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Neil David Lawrence. *Variational inference in probabilistic models*. PhD thesis, Citeseer, 2001.

J.M. Lee. *Introduction to Riemannian Manifolds*. Graduate Texts in Mathematics. Springer International Publishing, 2019.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR*, 2017.

David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 1992.

David John Cameron Mackay. *Bayesian methods for adaptive models*. California Institute of Technology, 1992.

Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, 2015.

Andrew C Miller, Nicholas J Foti, and Ryan P Adams. Variational boosting: Iteratively refining posterior approximations. In *International Conference on Machine Learning (ICML)*, 2017.

Radford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

Xavier Pennec. Intrinsic Statistics on Riemannian Manifolds: Basic Tools for Geometric Measurements. *Journal of Mathematical Imaging and Vision*, 2006.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 2002.

Mrinank Sharma, Sebastian Farquhar, Eric Nalisnick, and Tom Rainforth. Do Bayesian Neural Networks Need To Be Fully Stochastic? In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Neural Information Processing Systems (NeurIPS)*, 2005.

Alessandra Tosi, Søren Hauberg, Alfredo Vellido, and Neil D. Lawrence. Metrics for probabilistic geometries. In *Uncertainty in Artificial Intelligence (UAI)*, 2014.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Riemannian Laplace approximations for Bayesian neural networks (Appendix)

Contents of the Appendix

A Laplace background . . . 1
B Riemannian geometry . . . 2
C Implementation details . . . 5
C.1 Cost of solving the ODE system . . . 5
C.2 Analysis of exponential map . . . 6
D Experiments details and additional results . . . 7
D.1 Regression example . . . 7
D.2 Illustrative 2D classification example . . . 7
D.3 An additional illustrative example . . . 7
D.4 Additional results in the UCI classification tasks . . . 8
D.5 Complete results on MNIST and FMNIST . . . 9
D.6 Out-of-distribution results . . . 9

A Laplace background

We provide a more detailed introduction to Laplace approximation and discuss the optimiziation of the prior and likelihood hyperparameters by maximizing the marginal likelihood. In BNNs, Laplace approximation is use to approximate the posterior distribution, i.e.:

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})} = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{\int_\Theta p(\mathcal{D} \mid \theta)p(\theta)} = \frac{1}{Z} p(\mathcal{D} \mid \theta)p(\theta). \tag{A.1}$$

This is done by fitting a Gaussian approximation to the unnormalized distribution $p(\mathcal{D} \mid \theta)p(\theta)$ at its peak, where $p(\mathcal{D} \mid \theta)$ is the likelihood distribution and $p(\theta)$ is the prior over the weights. In standard training of neural networks the mean-squared error loss is usually used for regression. This corresponds to optimizing a Gaussian log-likelihood up to a scaling factor while using the cross-entropy loss in classification correspond to minimizing the negative log-likelihood. The usual weight decay, or L2 regularization, corresponds instead to a Gaussian prior $p(\theta)$ distribution. More specifically, training with $\lambda||\theta||_2^2$ corresponds to a Gaussian prior $\mathcal{N}(\theta; 0, \frac{\lambda^{-1}}{2} I)$

Therefore a natural peak to choose is given by the θ_*, which is the set of weights obtained at the end of the training. Indeed, θ_* is usually computed by:

$$\theta_* = \arg\min_\theta \underbrace{\left\{ -\sum_{n=1}^{N} \log p(\mathbf{y}_n \mid \mathbf{x}_n, \theta) - \log p(\theta) \right\}}_{\mathcal{L}(\theta)}. \tag{A.2}$$

Once we have θ_*, the Laplace approximation uses a a second-order Taylor expansion of $\mathcal{L}(\theta)$ around θ_*, which yields:

$$\hat{\mathcal{L}}(\theta) \approx \mathcal{L}(\theta_*) + \langle \nabla_\theta \mathcal{L}(\theta)\big|_{\theta=\theta_*}, (\theta - \theta_*)\rangle + \frac{1}{2}\langle (\theta - \theta_*), \mathrm{H}_\theta[\mathcal{L}](\theta)\big|_{\theta=\theta_*}(\theta - \theta_*)\rangle, \tag{A.3}$$

where the first-order term $\nabla_\theta \mathcal{L}(\theta)\big|_{\theta=\theta_*} \approx 0$ because the gradient at θ_* is 0.

By looking at $\mathcal{L}(\theta)$, we can notice that the Hessian is composed by two terms: a data fitting term and a prior term. Assuming $p(\theta) = \mathcal{N}(\theta; 0, \gamma^2 I)$, then the Hessian can be expressed as

$$\mathrm{H}_\theta[\mathcal{L}](\theta)\big|_{\theta=\theta_*} = \gamma^{-2}\mathbf{I} + \sum_{i=1}^{n} \nabla_\theta^2 \log p(y_i \mid x_i)\ |_{\theta_*} = (H + \alpha \mathbf{I}), \tag{A.4}$$

where we defined $\alpha = \frac{1}{\gamma^2}$, i.e. the prior precision.

Using the fact that $\hat{\mathcal{L}}(\theta)$ is the negative log-numerator of (A.1), we can get $p(\mathcal{D} \mid \theta)p(\theta)$ by taking the exponential of $-\hat{\mathcal{L}}(\theta)$. By doing so we have:

$$p(\mathcal{D} \mid \theta)p(\theta) \approx \exp(-\hat{\mathcal{L}}(\theta)) = \exp(-\mathcal{L}(\theta_*) - \frac{1}{2}(\theta - \theta_*)^T(\mathbf{H} + \alpha\mathbf{I})(\theta - \theta_*)). \tag{A.5}$$

For simplicity, we can define $\Sigma = (\mathbf{H} + \alpha\mathbf{I})^{-1}$ and rewrite the equation above to obtain:

$$p(\mathcal{D} \mid \theta)p(\theta) \approx \exp(-\mathcal{L}(\theta_*)) \exp(-\frac{1}{2}(\theta - \theta_*)^T\Sigma^{-1}(\theta - \theta_*))) \tag{A.6}$$

We can then use this approximation to estimate the normalizing constant of our approximate posterior, which corresponds to the marginal log-likelihood $p(\mathcal{D})$:

$$p(\mathcal{D}) = Z \approx \int_\Theta \exp(-\mathcal{L}(\theta_*)) \exp(-\frac{1}{2}(\theta - \theta_*)^T\Sigma^{-1}(\theta - \theta_*)))d\theta. \tag{A.7}$$

This can be rewritten as

$$p(\mathcal{D}) \approx \exp(-\mathcal{L}(\theta_*)) \int_\Theta \exp(-\frac{1}{2}(\theta - \theta_*)^T\Sigma^{-1}(\theta - \theta_*))), \tag{A.8}$$

and by using the Gaussian integral properties we can write it as:

$$p(\mathcal{D}) \approx \exp(-\mathcal{L}(\theta_*))(2\pi)^{d/2}(\det \Sigma)^{1/2}, \tag{A.9}$$

and by taking the logarithm we get

$$\log p(\mathcal{D}) \approx -\mathcal{L}(\theta_*) + \log(2\pi)^{d/2} + \log(\det \Sigma)^{1/2}, \tag{A.10}$$

which is the approximation of the log marginal likelihood that we want to maximize to optimize the parameters that appears in $\mathcal{L}(\theta_*)$. In a regression problem, we are interested in optimizing both the variance of the Gaussian likelihood and the prior precision. In classification, instead, we just have the prior precision as a hyperparameter to tune.

B Riemannian geometry

We rely on Riemannian geometry [Lee, 2019, do Carmo, 1992] in order to construct our approximate posterior. In a nutshell, a d-dimensional Riemannian manifold can be seen intuitively as a smooth d-dimensional surface that lies within a Euclidean space of dimension $D > d$, which allows one to compute distances between points that respect the geometry of the surface.

Definition B.1. A Riemannian manifold $\mathcal{M}$ is a smooth manifold together with a Riemannian metric $\mathbf{M}(\mathbf{x})$ that acts on the associated tangent space $\mathcal{T}_\mathbf{x}\mathcal{M}$ at any point $\mathbf{x} \in \mathcal{M}$.

Definition B.2. A Riemannian metric $\mathbf{M} : \mathcal{M} \to \mathbb{R}^{\dim(\mathcal{M}) \times \dim(\mathcal{M})}$ is a smoothly changing positive definite metric tensor that defines an inner product on the tangent space $\mathcal{T}_\mathbf{x}\mathcal{M}$ at any point $\mathbf{x} \in \mathcal{M}$.

We focus on the Bayesian neural network framework where $\Theta = \mathbb{R}^K$ is the parameter space of the associated deep network, and we consider the manifold $\mathcal{M} = g(\theta) = [\theta, \mathcal{L}(\theta)]$. This is essentially the loss surface which is K-dimensional and lies within a $(K+1)$-dimensional Euclidean space. In order to satisfy the smoothness condition for $\mathcal{M}$ we restrict to activation functions as the `tanh`, while common loss function as the mean squared error and softmax are also smooth.

The parameter space Θ is a parametrization of this surface and technically represents the *intrinsic coordinates* of the manifold, which is known as the *global chart* in the literature. The Jacobian $\mathbf{J}_g(\theta) \in \mathbb{R}^{K+1 \times K}$ of the map g spans the tangent space on the manifold, and a tangent vector can be written as $\mathbf{J}_g(\theta)\mathbf{v}$, where $\mathbf{v} \in \mathbb{R}^K$ are the *tangential coordinates*. We can thus compute the inner product between two tangent vectors in the ambient space using the Euclidean metric therein as

$$\langle \mathbf{J}_g(\theta)\mathbf{v}, \mathbf{J}_g(\theta)\mathbf{u} \rangle = \langle \mathbf{v}, \mathbf{M}(\theta)\mathbf{u} \rangle. \tag{B.1}$$

Here the matrix $\mathbf{M}(\theta) = \mathbf{J}_g(\theta)^\intercal \mathbf{J}_g(\theta) = \mathbb{I}_K + \nabla_\theta \mathcal{L}(\theta) \nabla_\theta \mathcal{L}(\theta)^\intercal$ is a Riemannian metric that is known in the literature as the *pull-back metric*. Note that the flat space Θ is technically a smooth manifold and together with $\mathbf{M}(\theta)$ is transformed into a Riemannian manifold. This can be also seen as the *abstract manifold* definition where we only need the intrinsic coordinates and the Riemannian metric to compute geometric quantities, and not the actual embedded manifold in the ambient space.

One example of a geometric quantity is the shortest path between two points $\theta_1, \theta_2 \in \Theta$. Let a curve $c : [0,1] \to \Theta$ with $c(0) = \theta_1$ and $c(1) = \theta_2$, the its length defined under the Riemannian metric as $\texttt{length}[c] = \int_0^1 \sqrt{\langle \dot{c}(t), \mathbf{M}(c(t))\dot{c}(t) \rangle} dt$. This quantity is computed intrinsically but it also corresponds to the length of the associated curve on $\mathcal{M}$. The shortest path then is defined as $c^*(t) = \arg\min_c \texttt{length}[c]$, but as the length is invariant under re-parametrizations of time, we consider the energy functional instead, which we optimize using the Euler-Lagrange equations. This gives the following system of second-order nonlinear ordinary differential equations (ODEs)

$$\ddot{c}(t) = -\frac{\mathbf{M}^{-1}(c(t))}{2} \left[2 \left[\frac{\partial \mathbf{M}(c(t))}{\partial c_1(t)}, \dots, \frac{\partial \mathbf{M}(c(t))}{\partial c_K(t)} \right] - \frac{\partial \mathrm{vec}[\mathbf{M}(c(t))]^\intercal}{\partial c(t)} \right] (\dot{c}(t) \otimes \dot{c}(t)), \tag{B.2}$$

where $\mathrm{vec}[\cdot]$ stacks the columns of a matrix and $\otimes$ the Kronecker product [Arvanitidis et al., 2018]. A curve that satisfies this system is known as *geodesic* and is potentially the shortest path. When the system is solved as a Boundary Value Problem (BVP) with initial condition $c(0) = \theta_1$ and $c(1) = \theta_2$, we get the geodesic that connects these two points. Let $\mathbf{v} = \dot{c}(0)$ be the velocity of this curve at $t = 0$. When the system is solved as an Initial Value Problem (IVP) with conditions $c(0) = \theta_1$ and $\dot{c}(0) = \mathbf{v}$, we get the geodesic $c_\mathbf{v}(t)$ between $c_\mathbf{v}(0) = \theta_1$ and $c_\mathbf{v}(1) = \theta_2$. This operation is known as the *exponential map* and we use it for our approximate posterior.

In general, an analytic solution for this system of ODEs does not exist [Hennig and Hauberg, 2014, Arvanitidis et al., 2019], and hence, we rely on an approximate numerical off-the-shelf ODE solver. Note that this is a highly complicated system, especially when the Riemannian metric depends on a finite data set, while it is computationally expensive to evaluate it. As we show below, the structure of the Riemannian metric that we consider, allows us to simplify significantly this system.

Lemma B.3. *For the Riemannian metric in* (3) *the general* ODEs *system in* (B.2) *becomes*

$$\ddot{c}(t) = -\frac{\nabla_\theta \mathcal{L}(c(t))}{1 + \langle \nabla_\theta \mathcal{L}(c(t)), \nabla_\theta \mathcal{L}(c(t)) \rangle} \langle \dot{c}(t), \mathrm{H}_\theta[\mathcal{L}](c(t))\dot{c}(t) \rangle \tag{B.3}$$

Proof. We consider the general system, and we compute each term individually. To simplify notation we will use $\mathbf{M} := \mathbf{M}(c(t))$, $\nabla := \nabla_\theta \mathcal{L}(c(t))$, $\mathrm{H} := \mathrm{H}_\theta[\mathcal{L}](c(t))$, H_i the i-th column of the Hessian and ∇_i the i-th element of the gradient ∇.

Using the Sherman–Morrison formula we have that

$$\mathbf{M}(c(t))^{-1} = \mathbb{I}_K - \frac{\nabla_\theta \mathcal{L}(c(t)) \nabla_\theta \mathcal{L}(c(t))^\intercal}{1 + \langle \nabla_\theta \mathcal{L}(c(t)), \nabla_\theta \mathcal{L}(c(t)) \rangle} \tag{B.4}$$

The first term in the brackets is

$$2 \left[\frac{\partial \mathbf{M}(c(t))}{\partial c_1(t)}, \dots, \frac{\partial \mathbf{M}(c(t))}{\partial c_K(t)} \right] = 2 \left[\mathrm{H}_1 \nabla^\intercal + \nabla \mathrm{H}_1^\intercal, \dots, \mathrm{H}_K \nabla^\intercal + \nabla \mathrm{H}_K^\intercal \right]_{D \times D^2} \tag{B.5}$$

and the second term in the brackets is

$$\frac{\partial \text{vec}[\mathbf{M}(c(t))]^\intercal}{\partial c(t)} = [\nabla_1 \mathrm{H} + \mathrm{H}_1 \nabla^\intercal, \ldots, \nabla_K \mathrm{H} + \mathrm{H}_K \nabla^\intercal] \tag{B.6}$$

and their difference is equal to

$$[2\nabla \mathrm{H}_1^\intercal + \mathrm{H}_1 \nabla^\intercal - \nabla_1 \mathrm{H}, \ldots, 2\nabla \mathrm{H}_K^\intercal + \mathrm{H}_K \nabla^\intercal - \nabla_K \mathrm{H}]\,. \tag{B.7}$$

We compute the matrix-vector product between the matrix (B.7) and the Kronocker product $\dot{c}(t) \otimes \dot{c}(t) = [\dot{c}_1 \dot{c}, \ldots, \dot{c}_K \dot{c}]^\intercal \in \mathbb{R}^{D^2 \times 1}$ which gives

$$\sum_{i=1}^{K} 2\nabla \mathrm{H}_i^\intercal \dot{c}\dot{c}_i + \mathrm{H}_i \nabla^\intercal \dot{c}\dot{c}_i - \nabla_i \mathrm{H}\dot{c}\dot{c}_i = 2\nabla \sum_{i=1}^{K} \dot{c}_i \mathrm{H}_i^\intercal \dot{c} + \nabla^\intercal \dot{c} \sum_{i=1}^{K} \mathrm{H}_i \dot{c}_i - \mathrm{H}\dot{c} \sum_{i=1}^{K} \nabla_i \dot{c}_i = 2\nabla \langle \dot{c}, \mathrm{H}\dot{c} \rangle \tag{B.8}$$

where we used that $\sum_{i=1}^{K} \mathrm{H}_i \dot{c}_i = \mathrm{H}\dot{c}$ and $\sum_{i=1}^{K} \nabla_i \dot{c}_i = \nabla^\intercal \dot{c}$. As a final step, we plug-in this result and the inverse of the metric in the general system which gives

$$\ddot{c} = -\frac{1}{2}\left(\mathbb{I}_K - \frac{\nabla\nabla^\intercal}{1 + \langle \nabla, \nabla \rangle}\right) 2\nabla \langle \dot{c}, \mathrm{H}\dot{c} \rangle = -\frac{\nabla}{1 + \langle \nabla, \nabla \rangle} \langle \dot{c}, \mathrm{H}\dot{c} \rangle. \tag{B.9}$$

□

Taylor expansion. As regards the *Taylor expansion* we consider the space Θ and the Riemannian metric therein $\mathbf{M}(\theta)$ and an arbitrary smooth function $f : \Theta \to \mathbb{R}$. If we ignore the Riemannian metric the second-order approximation of the function f around a point $\mathbf{x} \in \Theta$ is known to be

$$\hat{f}_{\text{Eucl}}(\mathbf{x} + \mathbf{v}) \approx f(\mathbf{x}) + \langle \nabla_\theta f(\theta)\big|_{\theta=\mathbf{x}}, \mathbf{v} \rangle + \frac{1}{2} \langle \mathbf{v}, \mathrm{H}_\theta[f](\theta)\big|_{\theta=\mathbf{x}} \mathbf{v} \rangle, \tag{B.10}$$

where $\nabla_\theta f(\theta)\big|_{\theta=\mathbf{x}}$ is the vector with the partial derivatives evaluated at $\mathbf{x}$ and $\mathrm{H}_\theta[f](\theta)\big|_{\theta=\mathbf{x}}$ the corresponding Hessian matrix with the partial derivatives $\frac{\partial^2 f(\theta)}{\partial \theta_i \partial \theta_j}\Big|_{\theta=\mathbf{x}}$. When we take into account the Riemannian metric, then the approximation becomes

$$\hat{f}_{\text{Riem}}(\mathbf{x} + \mathbf{v}) \approx f(\mathbf{x}) + \langle \nabla_\theta f(\theta)\big|_{\theta=\mathbf{x}}, \mathbf{v} \rangle + \frac{1}{2} \langle \mathbf{v}, \left[\mathrm{H}_\theta[f](\theta) - \Gamma_{ij}^k \nabla_\theta f(\theta)_k\right]\big|_{\theta=\mathbf{x}} \mathbf{v} \rangle, \tag{B.11}$$

where Γ_{ij}^k are the Christoffel symbols and the Einstein summation is used. Note that even if the Hessian is different, the approximation again is a quadratic function.

Now we consider the Taylor expansion on the associated tangent space at the point $\mathbf{x}$ instead of directly on the parameter space Θ. We define the function $h(\mathbf{v}) = f(\text{Exp}_\mathbf{x}(\mathbf{v}))$ on the tangent space centered at $\mathbf{x}$ and we get that

$$\hat{h}(\mathbf{u}) \approx h(0) + \langle \partial_\mathbf{v} f(\text{Exp}_\mathbf{x}(\mathbf{v}))\big|_{\mathbf{v}=0}, \mathbf{u} \rangle \tag{B.12}$$

$$+ \frac{1}{2} \langle \mathbf{u}, \left[\mathrm{H}_\mathbf{v}[f](\text{Exp}_\mathbf{x}(\mathbf{v})) - \Gamma_{ij}^k \partial_\mathbf{v} f(\text{Exp}_\mathbf{x}(\mathbf{v}))_k\right]\big|_{\mathbf{v}=0} \mathbf{u} \rangle, \tag{B.13}$$

where we apply the chain-rule and we use the fact that $\partial_\mathbf{v} \text{Exp}_\mathbf{x}(\mathbf{v})\big|_{\mathbf{v}=0} = \mathbb{I}$ and $\partial_\mathbf{v}^2 \text{Exp}_\mathbf{x}(\mathbf{v})\big|_{\mathbf{v}=0} = 0$. So, we get that $\partial_\mathbf{v} f(\text{Exp}_\mathbf{x}(\mathbf{v}))\big|_{\mathbf{v}=0} = \nabla_\theta f(\theta)\big|_{\theta=\mathbf{x}}$ and $\mathrm{H}_\mathbf{v}[f](\text{Exp}_\mathbf{x}(\mathbf{v}))\big|_{\mathbf{v}=0} = \frac{\partial^2 f(\theta)}{\partial \theta_i \partial \theta_j}\Big|_{\theta=\mathbf{x}}$. The difference here is that the quadratic function is defined on the tangent space, and the exponential map is a non-linear mapping. Therefore, the actual approximation on the parameter space $\hat{f}_{\text{Tangent}}(\text{Exp}_\mathbf{x}(\mathbf{v})) = \hat{h}(\mathbf{v})$ is not a quadratic function as before, but it adapts to the structure of the Riemannian metric. Intuitively, the closer a point is to the base point $\mathbf{x}$ with respect to the Riemannian distance, the more similar is the associated value $\hat{f}_{\text{Tangent}}(\text{Exp}_\mathbf{x}(\mathbf{v}))$ to $f(\mathbf{x})$. In our problem of interest, this behavior is desirable implying that if a parameter θ' is connected through low-loss regions with a continuous curve to θ_*, then we will assign to θ high approximate posterior density.

We can easily consider the approximation on the normal coordinates, where we know that the Christoffel symbols vanish, by using the relationship $\mathbf{u} = \mathbf{A}\bar{\mathbf{u}}$ with $\mathbf{A} = \mathbf{M}(\mathbf{x})^{-1/2}$, and thus the Taylor approximation of the function $\bar{h}(\bar{\mathbf{u}}) = f(\text{Exp}_{\mathbf{x}}(\mathbf{A}\bar{\mathbf{u}}))$ becomes

$$\hat{\bar{h}}(\bar{\mathbf{u}}) \approx \hat{h}(0) + \langle \mathbf{A}^\intercal \nabla_\theta f(\theta)\big|_{\theta=\mathbf{x}}, \mathbf{A}\bar{\mathbf{u}}\rangle + \frac{1}{2}\langle \bar{\mathbf{u}}, \mathbf{A}^\intercal \mathrm{H}_\theta[f](\theta)\big|_{\theta=\mathbf{x}} \mathbf{A}\bar{\mathbf{u}}\rangle. \tag{B.14}$$

Further details can be found in related textbooks [Absil et al., 2008] and articles [Pennec, 2006].

Linearized manifold. Let us consider a regression problem with likelihood $p(\mathbf{y}|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{y}|f_\theta(\mathbf{x}), \sigma^2)$, where f_θ is a deep neural network, and prior $p(\theta) = \mathcal{N}(\theta|0, \lambda\mathbb{I}_K)$. The loss of the $f^{\text{lin}}_\theta(\mathbf{x})$ is then defined as

$$\mathcal{L}^{\text{lin}}(\theta) = \frac{1}{2\sigma^2}\sum_{n=1}^{N}(\mathbf{y} - f_{\theta_*}(\mathbf{x}_n) - \langle \nabla_\theta f(\mathbf{x}_n)\big|_{\theta=\theta_*}, \theta - \theta_*\rangle)^2 + \lambda||\theta||^2. \tag{B.15}$$

The gradient and the Hessian of this loss function can be easily computed as

$$\nabla_\theta \mathcal{L}^{\text{lin}}(\theta) = \frac{1}{\sigma^2}\sum_{n=1}^{N} -(\mathbf{y} - f_{\theta_*}(\mathbf{x}_n) - \langle \nabla_\theta f(\mathbf{x}_n)\big|_{\theta=\theta_*}, \theta - \theta_*\rangle)\nabla_\theta f(\mathbf{x}_n)\big|_{\theta=\theta_*} + 2\lambda\theta \tag{B.16}$$

$$\mathrm{H}_\theta[\mathcal{L}^{\text{lin}}](\theta) = \frac{1}{\sigma^2}\sum_{n=1}^{N} \nabla_\theta f(\mathbf{x}_n)\big|_{\theta=\theta_*} \nabla_\theta f(\mathbf{x}_n)\big|^\intercal_{\theta=\theta_*} + 2\lambda\mathbb{I}_K, \tag{B.17}$$

which can be used to evaluate the ODEs system in (6). A similar result can be derived for the binary cross entropy loss and the Bernoulli likelihood.

C Implementation details

In this section we present the implementation details of our work. The code will be released upon acceptance.

Gradient, Hessian, and Jacobian computations. The initial velocities used for our methods are samples from the Laplace approximation. We rely in the `Laplace` library [Daxberger et al., 2021a] for fitting the Laplace approximation and for optimizing the hyperparameters by using the marginal log-likelihood. We also used the same library to implement all the baselines consider in this work. As we have seen from Sec. 3.3, to integrate the ODE we have to compute (6), which we report also here for clarity:

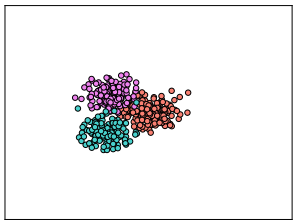
$$\ddot{c}(t) = -\nabla_\theta \mathcal{L}(c(t))\left(1 + \nabla_\theta \mathcal{L}(c(t))^\intercal \nabla_\theta \mathcal{L}(c(t))\right)^{-1} \langle \dot{c}(t), \mathrm{H}_\theta[\mathcal{L}](c(t))\dot{c}(t)\rangle. \tag{C.1}$$

We use `functorch` [Horace He, 2021] to compute both the gradient and the Hessian-vector-product. We then rely on `scipy` [Virtanen et al., 2020] implementation of the explicit Runge-Kutta method of order $5(4)$ [Dormand and Prince, 1980] to solve the initial-value problem. We use default tolerances in all our experiments.

Linearized manifold. We define our linearized manifold by considering the "linearized" function $f^{\text{lin}}_\theta(\mathbf{x}) = f_{\theta_*}(\mathbf{x}) + \langle \nabla_\theta f_\theta(\mathbf{x})\big|_{\theta=\theta_*}, \theta - \theta_*\rangle$ to compute the loss, where $\nabla_\theta f_\theta(\mathbf{x})\big|_{\theta=\theta_*} \in \mathbb{R}^{C\times K}$ is the Jacobian. To compute $\langle \nabla_\theta f_\theta(\mathbf{x})\big|_{\theta=\theta_*}, \theta - \theta_*\rangle$ we use `functorch` to compute a jacobian-vector product. This way, we avoid having to compute and store the Jacobian, which for big networks and large dataset is infeasible to store.

C.1 Cost of solving the ODE system

Our proposed approach builds on top of Laplace approximation, and therefore it comes with a computational overhead. To be able to estimate this overhead, we have to assume that our solver does S steps to solve an ODE, which is not entirely correct, since the explicit Runge-Kutta method of order 5(4) (RK-45) [Dormand and Prince, 1980] uses an adaptive step-size and therefore the value of S differs for every ODE. With this assumption, the cost of evaluating a single ODE results in $O(SWN)$, where S is the number of steps of the solver, W is the number of model parameters and N is the



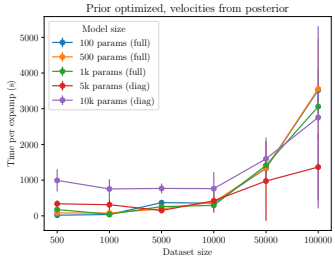


Figure C.1: *Left*: Average runtime per posterior sample in terms of model size and dataset size. *Right*: Average runtime and standard error per posterior sample in terms of model size and dataset size computed over 5 different samples. For the models with $5k$ and $10k$ parameters, we used the diagonal Hessian over all the weights to sample the initial velocities.

dataset size. Indeed, at each step, we need to compute the gradient and the hessian-vector product, which is $O(NW)$, and perform a Runge-Kutta step, which scales linearly with the dimension of the problem, i.e. $O(W)$ [Heirer et al., 1987]. Since the solver would take S steps to solve the system we get $O(SNW)$. If we want to use K posterior samples to estimate our predictive distribution, then we have to solve K different ODEs, therefore the overall overhead is $O(KSNW)$. However, we want to highlight that all these computations happen before deployment of the model. At test time, the complexity is the same as LA when using MC sample to approximate the predictive distribution.

In the derivation of the complexity of our method, we assumed that the ODE solver did exactly S steps. As mentioned before, RK-45 uses an adaptive step-size, therefore the number of steps the solver needs to converge mainly depends on the complexity of the associated ODE problem, which is defined by the geometry of the loss landscape and the initial velocity. To show this, in the next section, we include a benchmark using a synthetic example to analyze how the runtime changes with respect to the size of the model and dataset. While increasing the model parameters and dataset size affect W and N, it may be the case that ODE systems do not necessarily get harder, therefore the value of S would be small. In other words, evaluating the ODE becomes more expensive as dimensions increase, but perhaps the actual system gets easier to solve.

Limitations of our implementation The cost of obtaining a posterior sample using our RIEM-LA scales linearly in terms of the model and dataset size. In addition to that, our implementation is not highly engineered. Although we can generate these samples in parallel, we use a general purpose Python solver (`scipy.integrate.solve_ivp` [Virtanen et al., 2020]) to solve the ODE system, which runs on the CPU. When the solver needs to evaluate the ODE, our automatic-differentiation based approach runs on the GPU, and the result is moved to the CPU (`.detach().cpu().numpy()`) causing a significant overhead, which is inefficient. Especially when dimensions increase, both the transfer of the data and the computations on the CPU are sub-optimal. Implementing an ODE solver on a suitable automatic-differentiation framework (e.g. JAX) solely running on GPU will dramatically improve performance.

C.2 Analysis of exponential map

We mentioned scalability as one of the main limitation of our proposed method. We use a simple toy-example (`sklearn.datasets.make_blobs` with three classes, see Fig. C.1 (*left*)) to analyze the running time of computing a single sample from the posterior with our methods. A two layers neural-network is considered, and we vary the number of parameters and datapoints in the dataset. For the biggest models, i.e. the one with 5k and 10k parameters, we use a diagonal approximation of the Hessian to sample the initial velocities, while for all the others we use the full Hessian. In all cases, the ODE system always relies on the hvp which uses the full Hessian. In Fig. C.1 (*right*), we present runtime for five different models and six dataset sizes. We report the mean and standard deviation computed over five different exponential maps. The results show that the runtime increases

for all models as the dataset size increases. We also see that using a diagonal approximation of the Hessian leads to faster exponential maps.

D Experiments details and additional results

D.1 Regression example

For the regression example we consider two fully connected networks, one with one hidden layer with 15 units and one with two layers with 10 units each, both with `tanh` activations. We train both model using full-dataset GD, using a weight decay of $1e-2$ for the larger model and $1e-3$ for the smaller model for 35000 and 700000 epochs respectively. In both cases, we use a learning rate of $1e-3$. In Sec. 5, we show some samples from the posterior distribution obtained by using vanilla LA and our RIEM-LA approach. In Fig. D.1 we report the predictive distribution for our classic approach while in Fig. D.2 we show both the posterior and the predictive for our linearized manifold and linearized LA. We can see that our linearized approach perform similarly to linearized LA in terms of posterior samples and predictive distribution.

A more interesting experiment is to consider a gap in our dataset to measure the in-between uncertainty. A good behaviour would be to give calibrated uncertainty estimates in between separated regions of observations [Foong et al., 2019]. In our regression example, we consider the points between 1.5 and 3 as test set, and report posterior samples and predictive distribution for both our methods, vanilla and linearized, and LA. From Fig. D.3, we can notice that our RIEM-LA is able to generate uncertainty estimates that are reliable both in the small and the overparametrized model. It gets more interesting when we consider our linearized manifold approach as it can be seen in Fig. D.4. While for a small model the predictive distribution we get is comparable to linearized LA, when we consider the overparametrized model, it tends to overfit to a solution that it is better from the perspective of the linearized loss but very different from the MAP. Therefore, this results in uncertainty estimates that are not able to capture the true data set in this case. This behaviour is related to our discussion in Sec. 3.3. By using a subset of the training set to solve the ODEs system we alleviate this behavior by giving more reliable uncertainty estimates.

For each all the regression setting described above, we also report the mean and the uncertainty estimates by sampling from a closer approximation of the true posterior obtained using Hamiltonian Monte Carlo (HMC) and a No-U-Turn sampler. We use one chain, samples 1000 weights, discard the first 500 as warm-up and use the last 500 to compute the predictive distribution. Results can be seen in Fig. D.5. We can see that in this simple example the uncertainty produced by our method is really close to the one produced by HMC.

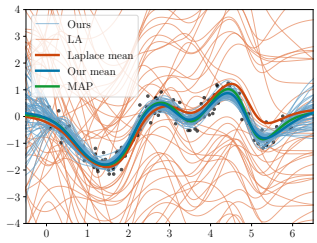
D.2 Illustrative 2D classification example

We consider the `banana` dataset as an illustrative example to study the confidence of our proposed method against Laplace and linearized Laplace. We train a 2-layer fully connected neural net with 16 hidden units per layer and `tanh` activation using SGD for 2500 epochs. We use a learning rate of $1e-3$ and weight-decaay of $1e-2$. Although it is a binary classification problem, we use a network with two outputs and use the cross-entropy loss to train the model because the `Laplace` library does not support binary cross-entropy at the moment. For all methods, we use 100 MC samples for the predictive distribution.

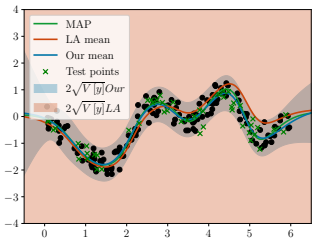
As we have mentioned in Sec. 5, our linearized manifold when we solve the ODEs system by using the entire training set tends to be overconfident compared to the our classic non-linearized approach. This can be easily seen form Fig. D.6 where we can compare the last two rows and see that solving the ODEs system using batches is beneficial in terms of uncertainty quantification. We also plot the confidence of all different methods when we do not optimize the prior precision. We can see that our proposed approaches are robust to it, while linearized LA highly depends on it.

D.3 An additional illustrative example

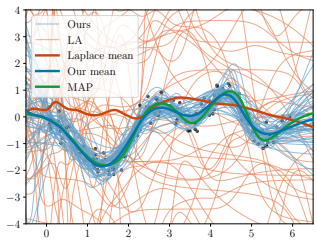
We consider the `pinwheel` dataset with five different classes as an additional illustrative classification example. Given the way these classes clusters, it is interesting to see if some of the approaches are able to be confident only in regions where there is data. We generate a dataset considering 200



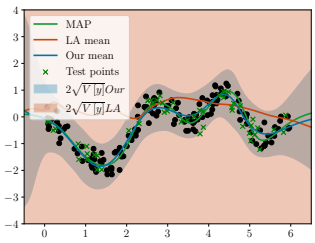
POSTERIOR - SINGLE LAYER MODEL



PREDICTIVE - SINGLE LAYER MODEL



POSTERIOR - TWO LAYERS MODEL



PREDICTIVE - TWO LAYERS MODEL

Figure D.1: Regression results in terms of posterior and predictive distribution using vanilla LA and our RIEM-LA approach. Our proposed approach is able to get better samples and to give a reliable uncertainty estimate compared to vanilla LA. If we consider an overparametrized model.

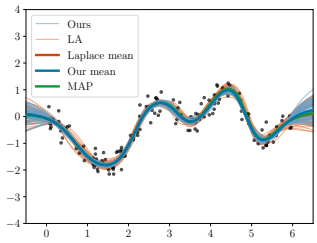
examples per classes and we use 350 example as training set and the remaining as test set. We consider a two layer fully-connected network with 20 hidden units per layer and `tanh` activation. As before, we train it using SGD with a learning rate of $1e-3$ and a weight decay of $1e-2$ for 5000 epochs.

From Fig. D.7, we can see not only vanilla LA but also linearized LA is failing in being confident also in-data region when we use 50 posterior samples. Our proposed approaches instead give meaningful uncertainty estimates, but we can see that by optimizing the prior precision, our methods get slightly more confident far from the data.

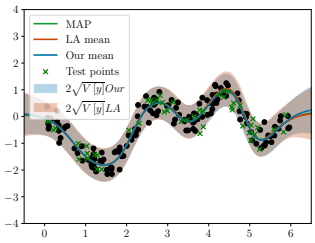
D.4 Additional results in the UCI classification tasks

In the main paper we present results on five different UCI classification tasks in terms of test negative log-likelihood. Here, we report also results in terms of test accuracy, Brier score, and expected calibration error. Results with prior precision optimized are shown in Table 5, while for prior precision not optimized we refer to Table 6. In both cases, we consider a neural network with a single fully connected layer consisting of 50 hidden units and `tanh` activation. We train it using Adam optimizer [Kingma and Ba, 2015] for 10000 epochs using a learning rate of $1e-3$ and a weight decay of $1e-2$.

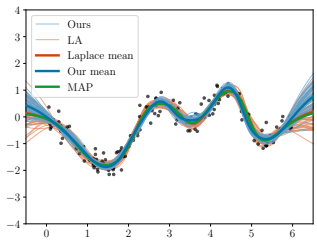
From the two tables, we can see that our RIEM-LA is better than all the other methods both in terms of NLL, brier, and ECE. It is also surprising that our method, both the classic and the linearized approach, are able to improve the accuracy of the MAP estimate in most datasets.



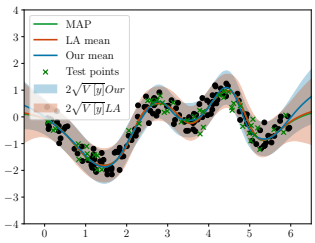
POSTERIOR - SINGLE LAYER MODEL



PREDICTIVE - SINGLE LAYER MODEL



POSTERIOR - TWO LAYERS MODEL



PREDICTIVE - TWO LAYERS MODEL

Figure D.2: Regression results in terms of posterior and predictive distribution using linearized LA and our linearized manifold approach. Our linearized manifold perform similarly to linearized LA no matter what model architecture we are considering.

D.5 Complete results on MNIST and FMNIST

For these two image classification tasks we consider a small convolutional neural network. Our network consists of the following layers: an initial convolutional layer with 4 channels and 5×5 filter followed by `tanh` activation and an average pooling layer. The we have another convolutional layer still with 4 channels and 5×5 kernel also followed by `tanh` activation and an average pooling layer. Then we have three fully connected layer with 16, 10, and 10 hidden units respectively and `tanh` activation.

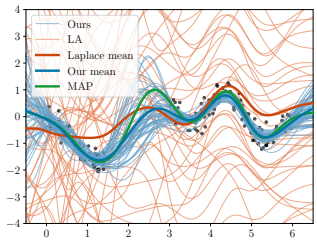
We train both models using SGD with a learning rate of $1e-3$ and weight decay of $5e-4$ for 100 epoch. The learning rate is annealed using the cosine decay method [Loshchilov and Hutter, 2017].

In Table 7 and Table 8, we can see that also when we do not optimize the prior precision our RIEM-LA is mostly performing better than all the alternatives. In particular, for the CNN trained on MNIST and no prior precision optimization, we have that the MAP is also performing well in terms of NLL and Brier score. On FashionMNIST, instead, if we do not optimize the prior we have that both our approaches are better than all the other methods.

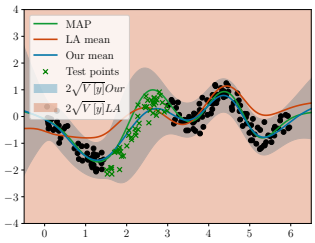
D.6 Out-of-distribution results

The benefit of having meaningful and robust uncertainty estimation is that our model would then be confident in-data region while being uncertain in region without data. Therefore, if this is happening, then we would expect the model to be able to detect out-of-distribution (OOD) examples more successfully.

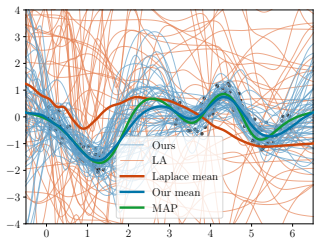
We consider classic OOD images detection tasks, where we train a model on MNIST and tested on FashionMNIST, EMNIST, and KMNIST and one trained on FMNIST and tested on the remaining datasets. It's well known that linearized LA is one of the strongest method for OOD detection



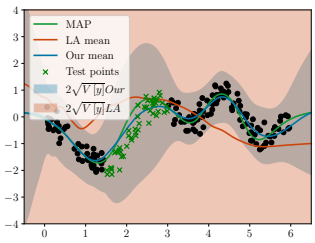
POSTERIOR - SINGLE LAYER MODEL



PREDICTIVE - SINGLE LAYER MODEL



POSTERIOR - TWO LAYERS MODEL

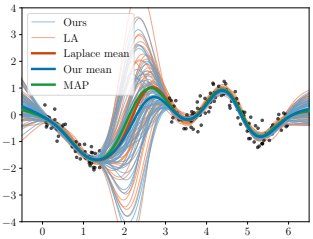


PREDICTIVE - TWO LAYERS MODEL

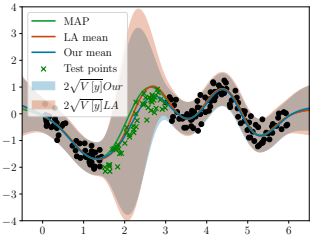
Figure D.3: In-between uncertainty example using vanilla LA and our `riem-la` approach. Also in this setting, our vanilla approach is able to overcome the difficulties of vanilla LA in this problem setting and giving both meaningful samples from the posterior and a reliable predictive distribution.

[Daxberger et al., 2021a]. We consider the same MAP estimates we used in the previous section and present OOD performance in Table 9 and Table 10.

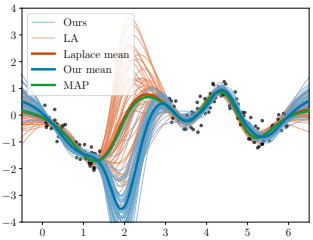
We can see that our proposed method is consistently working better than linearized and classic LA in all the considered setting apart for models trained on FMNIST where we do not optimize the prior precision to compute our initial velocities. In that setting, however, our linearized approach using batches is getting similar performance than linearized LA.



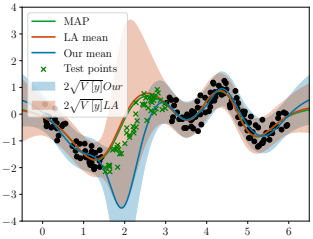
POSTERIOR - SINGLE LAYER MODEL



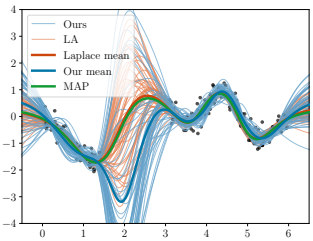
PREDICTIVE - SINGLE LAYER MODEL



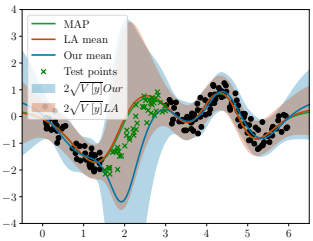
POSTERIOR - TWO LAYERS MODEL



PREDICTIVE - TWO LAYERS MODEL

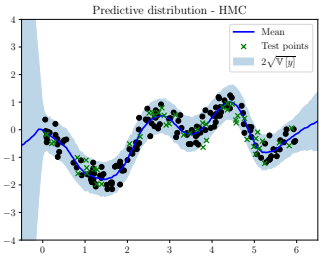


POSTERIOR - TWO LAYERS MODEL (BATCH)

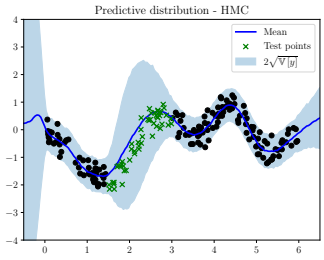


PREDICTIVE - TWO LAYERS MODEL (BATCH)

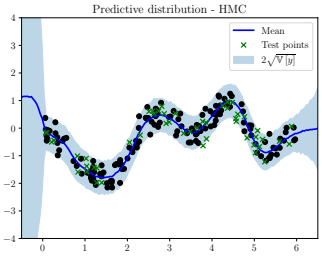
Figure D.4: In-between uncertainty example using linearized LA and our linearized manifold. For our linearized approach we both consider solving the ODEs system using the entire training set and using subsets of it. We already highlight how our linearized approach tend to overfit on the linearized loss, and we can clearly see it here from the plots in the center. Using batches to solve the ODEs system gives more reliable uncertainty estimates.



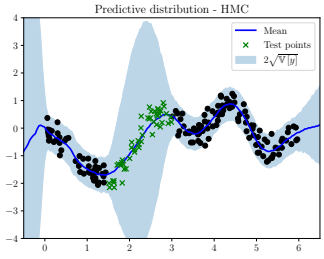
PREDICTIVE - SINGLE LAYER MODEL



PREDICTIVE - SINGLE LAYER MODEL



PREDICTIVE - TWO LAYERS MODEL



PREDICTIVE - TWO LAYERS MODEL

Figure D.5: Predictive distribution obtained using HMC on all the different regression examples considered using a one-layer and a two-layers model.

Table 4: Full in-distribution results in the banana dataset. We used 100 samples both for the Laplace approximation and our method. ECE and MCE are computed using $M = 10$ bins.

	Banana dataset				
	Prior Precision NOT optimized				
Method	Accuracy ↑	NLL↓	Brier↓	ECE↓	MCE↓
MAP	86.69 ± 0.34	0.333 ± 0.005	0.0930 ± 0.0015	5.91 ± 0.34	22.75 ± 2.42
VANILLA LA	48.85 ± 2.32	0.700 ± 0.003	0.2534 ± 0.0017	7.10 ± 1.96	23.10 ± 6.69
LIN-LA	86.92 ± 0.40	0.403 ± 0.012	0.1196 ± 0.0044	13.04 ± 0.60	17.11 ± 0.54
RIEM-LA	$\mathbf{87.14 \pm 0.20}$	$\mathbf{0.285 \pm 0.001}$	$\mathbf{0.0878 \pm 0.0006}$	$\mathbf{2.75 \pm 0.22}$	$\mathbf{6.30 \pm 0.89}$
RIEM-LA (BATCHES)	$\mathbf{87.32 \pm 0.17}$	0.294 ± 0.002	0.0895 ± 0.0004	4.79 ± 0.31	8.28 ± 0.84
LIN-RIEM-LA	85.33 ± 0.31	0.884 ± 0.037	0.1252 ± 0.0022	11.50 ± 0.31	36.27 ± 2.84
LIN-RIEM-LA (BATCHES)	86.16 ± 0.21	0.352 ± 0.002	0.0994 ± 0.0011	4.06 ± 0.11	13.67 ± 0.69

	Prior Precision optimized				
Method	Accuracy ↑	NLL↓	Brier↓	ECE↓	MCE↓
MAP	86.69 ± 0.34	0.333 ± 0.005	0.0930 ± 0.0015	5.91 ± 0.34	22.75 ± 2.42
VANILLA LA	59.50 ± 5.07	0.678 ± 0.009	0.2426 ± 0.0046	12.08 ± 2.31	27.49 ± 5.05
LIN-LA	86.99 ± 0.37	0.325 ± 0.008	0.0956 ± 0.0023	6.59 ± 0.41	11.44 ± 1.85
RIEM-LA	$\mathbf{87.57 \pm 0.07}$	$\mathbf{0.287 \pm 0.002}$	$\mathbf{0.0886 \pm 0.0006}$	$\mathbf{2.55 \pm 0.37}$	10.95 ± 1.16
RIEM-LA (BATCHES)	87.30 ± 0.08	$\mathbf{0.286 \pm 0.001}$	$\mathbf{0.0890 \pm 0.0000}$	$\mathbf{2.57 \pm 0.04}$	$\mathbf{6.08 \pm 0.67}$
LIN-RIEM-LA	87.02 ± 0.38	0.415 ± 0.029	0.0967 ± 0.0024	6.97 ± 0.43	21.24 ± 2.28
LIN-RIEM-LA (BATCHES)	$\mathbf{87.77 \pm 0.24}$	0.298 ± 0.006	$\mathbf{0.0887 \pm 0.0011}$	$\mathbf{2.38 \pm 0.28}$	8.04 ± 1.61

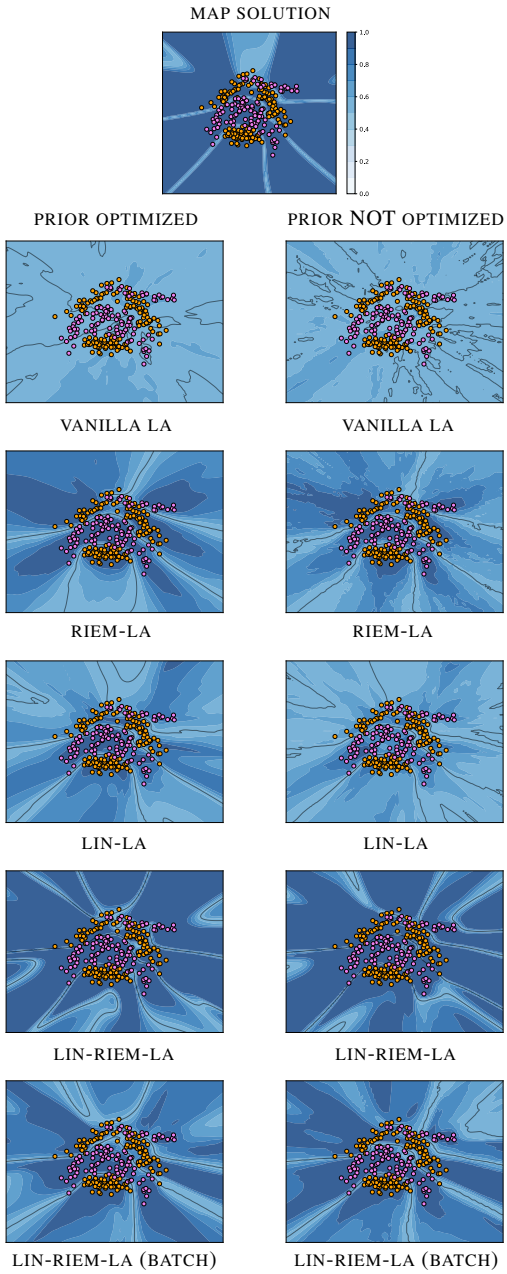


Figure D.6: Confidence for all the considered approach on the `banana` dataset. All plots were computed using 50 MC samples from the posterior. We can notice that our RIEM-LA and LIN-RIEM-LA with batches are giving the best confidence of all methods followed by linearized LA. We can also see that our approaches are robust to prior optimization, while linearized LA gets really better if we optimize the prior precision.

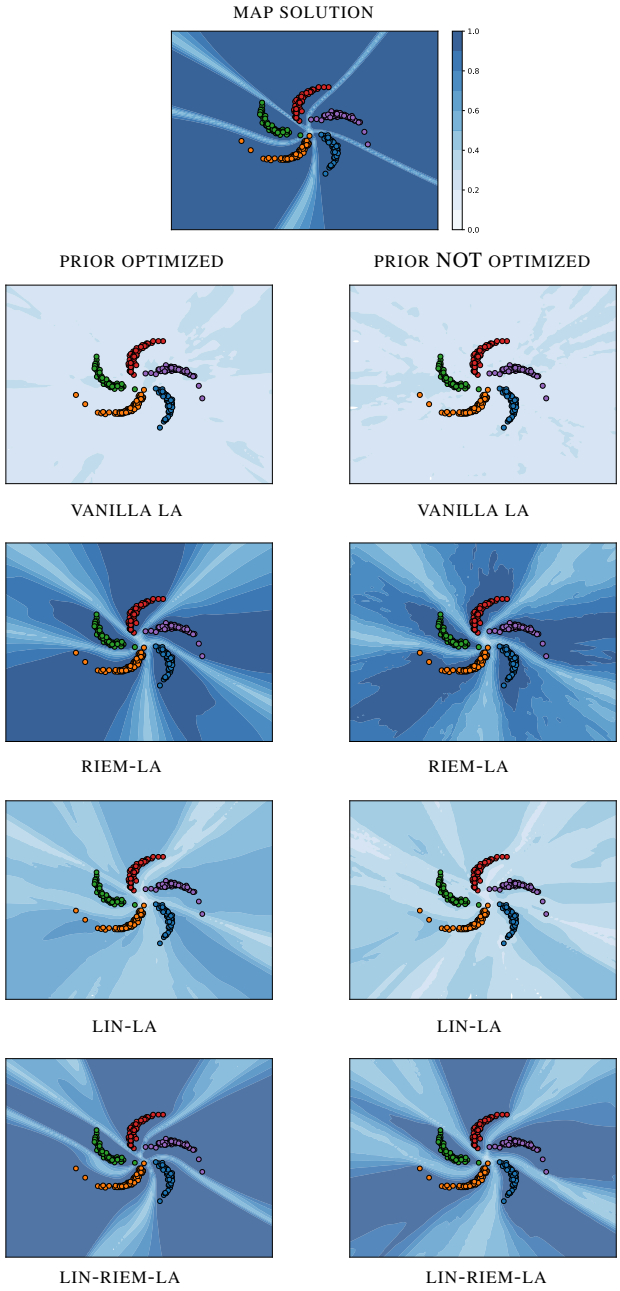


Figure D.7: Confidence for all the considered approach on the `pinwheel` dataset. All plots were computed using 50 MC samples from the posterior. We can notice that in this setting RIEM-LA gives the better confidence followed by LIN-RIEM-LA. However, priorprecision optimization is making our approaches more confident outside the data region. Linearized LA struggle in producing meaningful uncertainty estimates.

Table 5: Results for all the different techniques on UCI datasets for classification. Predictive distribution is estimated using 30 MC samples when prior precision is optimized. Mean and standard error over 5 different seeds.

	PRIOR PRECISION OPTIMIZED				
	ACCURACY ↑				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	$\mathbf{79.52 \pm 0.99}$	49.61 ± 3.05	$\mathbf{80.47 \pm 1.12}$	73.86 ± 1.47	$\mathbf{78.58 \pm 1.31}$
GLASS	63.75 ± 3.60	26.25 ± 3.91	$\mathbf{67.5 \pm 3.40}$	55.62 ± 2.40	$\mathbf{66.25 \pm 3.47}$
IONOSPHERE	86.04 ± 2.30	58.11 ± 4.04	$\mathbf{90.19 \pm 1.95}$	80.38 ± 1.57	$\mathbf{88.30 \pm 1.12}$
WAVEFORM	$\mathbf{82.27 \pm 1.66}$	64.27 ± 3.19	$\mathbf{84.93 \pm 1.42}$	77.07 ± 1.52	$\mathbf{82.93 \pm 1.08}$
AUSTRALIAN	82.30 ± 1.00	56.92 ± 2.33	$\mathbf{84.81 \pm 1.03}$	75.00 ± 1.90	82.69 ± 0.82
BREAST CANCER	$\mathbf{96.08 \pm 0.73}$	71.96 ± 8.58	$\mathbf{96.86 \pm 0.85}$	94.90 ± 0.89	$\mathbf{96.08 \pm 1.30}$

	NLL ↓				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	0.975 ± 0.081	1.209 ± 0.020	$\mathbf{0.454 \pm 0.024}$	0.875 ± 0.020	$\mathbf{0.494 \pm 0.044}$
GLASS	2.084 ± 0.323	1.737 ± 0.037	$\mathbf{1.047 \pm 0.224}$	1.365 ± 0.058	1.359 ± 0.299
IONOSPHERE	1.032 ± 0.175	0.673 ± 0.013	$\mathbf{0.344 \pm 0.068}$	0.497 ± 0.015	0.625 ± 0.110
WAVEFORM	1.076 ± 0.110	0.888 ± 0.030	$\mathbf{0.459 \pm 0.057}$	0.640 ± 0.002	0.575 ± 0.065
AUSTRALIAN	1.306 ± 0.146	0.684 ± 0.011	$\mathbf{0.541 \pm 0.053}$	$\mathbf{0.570 \pm 0.016}$	0.833 ± 0.108
BREAST CANCER	$\mathbf{0.225 \pm 0.076}$	0.594 ± 0.030	$\mathbf{0.176 \pm 0.092}$	0.327 ± 0.022	$\mathbf{0.202 \pm 0.073}$

	BRIER ↓				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	0.0877 ± 0.0052	0.1654 ± 0.0028	$\mathbf{0.0671 \pm 0.0035}$	0.1210 ± 0.0028	0.0720 ± 0.0048
GLASS	0.1058 ± 0.0095	0.1353 ± 0.0025	$\mathbf{0.0740 \pm 0.0075}$	0.1102 ± 0.0031	$\mathbf{0.0787 \pm 0.0083}$
IONOSPHERE	0.1308 ± 0.0182	0.2398 ± 0.0064	$\mathbf{0.0840 \pm 0.0127}$	0.1596 ± 0.0069	$\mathbf{0.0868 \pm 0.0095}$
WAVEFORM	0.1043 ± 0.0091	0.1766 ± 0.0065	$\mathbf{0.0825 \pm 0.0076}$	0.1260 ± 0.0011	$\mathbf{0.0871 \pm 0.0053}$
AUSTRALIAN	0.1642 ± 0.0105	0.2452 ± 0.0052	$\mathbf{0.1209 \pm 0.0113}$	0.1901 ± 0.0069	$\mathbf{0.1340 \pm 0.0082}$
BREAST CANCER	0.0351 ± 0.0077	0.2020 ± 0.0148	$\mathbf{0.0269 \pm 0.0075}$	0.0866 ± 0.0082	$\mathbf{0.0310 \pm 0.0094}$

	ECE ↓				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	16.75 ± 1.06	15.01 ± 2.41	$\mathbf{8.43 \pm 1.06}$	26.61 ± 1.40	10.49 ± 0.95
GLASS	31.77 ± 2.67	$\mathbf{11.77 \pm 1.08}$	17.06 ± 2.96	23.17 ± 1.47	20.84 ± 2.57
IONOSPHERE	13.92 ± 2.02	11.22 ± 1.59	9.05 ± 0.99	15.22 ± 1.38	$\mathbf{7.70 \pm 1.17}$
WAVEFORM	15.73 ± 1.29	18.73 ± 2.41	10.72 ± 1.31	19.12 ± 1.60	$\mathbf{9.10 \pm 0.94}$
AUSTRALIAN	16.25 ± 0.96	$\mathbf{5.55 \pm 1.41}$	8.49 ± 1.62	12.42 ± 1.43	10.20 ± 0.90
BREAST CANCER	$\mathbf{3.85 \pm 0.89}$	20.39 ± 3.28	$\mathbf{3.20 \pm 0.72}$	20.24 ± 0.99	$\mathbf{3.16 \pm 0.92}$

Table 6: Results for all the different techniques on UCI datasets for classification. Predictive distribution is estimated using 30 MC samples when prior precision is not optimized. Mean and standard error over 5 different seeds.

	PRIOR PRECISION NOT OPTIMIZED				
	ACCURACY ↑				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	79.52 ± 0.99	23.94 ± 1.62	$\mathbf{82.05 \pm 1.10}$	44.88 ± 1.74	$\mathbf{80.16 \pm 1.45}$
GLASS	63.75 ± 3.60	15.00 ± 3.24	$\mathbf{68.13 \pm 3.11}$	30.63 ± 3.89	63.75 ± 2.88
IONOSPHERE	86.04 ± 2.30	47.55 ± 1.72	$\mathbf{91.32 \pm 1.26}$	69.81 ± 4.10	$\mathbf{89.81 \pm 0.68}$
WAVEFORM	$\mathbf{82.27 \pm 1.66}$	24.13 ± 4.30	$\mathbf{83.60 \pm 1.34}$	54.40 ± 2.11	$\mathbf{83.47 \pm 1.28}$
AUSTRALIAN	82.30 ± 1.00	40.19 ± 1.77	$\mathbf{86.73 \pm 1.10}$	61.92 ± 2.57	84.04 ± 0.80
BREAST CANCER	$\mathbf{96.08 \pm 0.73}$	51.37 ± 11.68	$\mathbf{97.06 \pm 0.73}$	84.51 ± 3.72	$\mathbf{95.29 \pm 1.22}$

	NLL ↓				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	0.975 ± 0.081	1.424 ± 0.018	$\mathbf{0.398 \pm 0.016}$	1.197 ± 0.018	$\mathbf{0.415 \pm 0.016}$
GLASS	2.084 ± 0.323	2.057 ± 0.084	$\mathbf{0.981 \pm 0.202}$	1.697 ± 0.044	$\mathbf{1.116 \pm 0.270}$
IONOSPHERE	1.032 ± 0.175	0.726 ± 0.011	$\mathbf{0.284 \pm 0.080}$	0.611 ± 0.014	$\mathbf{0.289 \pm 0.023}$
WAVEFORM	1.076 ± 0.110	1.185 ± 0.031	$\mathbf{0.465 \pm 0.051}$	0.962 ± 0.015	$\mathbf{0.456 \pm 0.059}$
AUSTRALIAN	1.306 ± 0.146	0.750 ± 0.009	$\mathbf{0.518 \pm 0.070}$	0.658 ± 0.012	$\mathbf{0.593 \pm 0.030}$
BREAST CANCER	0.225 ± 0.076	0.690 ± 0.051	$\mathbf{0.197 \pm 0.093}$	0.503 ± 0.0222	$\mathbf{0.171 \pm 0.064}$

	BRIER ↓				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	0.0877 ± 0.0052	0.1917 ± 0.0020	$\mathbf{0.0604 \pm 0.0024}$	0.1645 ± 0.0023	0.0657 ± 0.0027
GLASS	0.1058 ± 0.0095	0.1458 ± 0.0024	$\mathbf{0.0711 \pm 0.0063}$	0.1328 ± 0.0028	$\mathbf{0.0768 \pm 0.0067}$
IONOSPHERE	0.1308 ± 0.0182	0.2654 ± 0.0053	$\mathbf{0.0689 \pm 0.0076}$	0.1596 ± 0.0069	0.0868 ± 0.0095
WAVEFORM	0.1043 ± 0.0091	0.2403 ± 0.0068	$\mathbf{0.0789 \pm 0.0056}$	0.1929 ± 0.0033	$\mathbf{0.0821 \pm 0.0060}$
AUSTRALIAN	0.1642 ± 0.0105	0.2774 ± 0.0042	$\mathbf{0.1081 \pm 0.0096}$	0.2328 ± 0.0059	0.1205 ± 0.0072
BREAST CANCER	$\mathbf{0.0351 \pm 0.0077}$	0.2486 ± 0.0248	$\mathbf{0.0263 \pm 0.0068}$	0.1597 ± 0.0106	$\mathbf{0.0316 \pm 0.0080}$

	ECE ↓				
DATASET	MAP	VANILLA LA	RIEM-LA	LINEARIZED LA	LINEARIZED RIEM-LA
VEHICLE	16.75 ± 1.06	10.78 ± 1.31	$\mathbf{6.65 \pm 0.81}$	9.00 ± 1.32	$\mathbf{5.43 \pm 0.74}$
GLASS	31.77 ± 2.67	15.27 ± 1.11	14.84 ± 2.23	$\mathbf{8.18 \pm 2.30}$	19.63 ± 1.74
IONOSPHERE	13.92 ± 2.02	11.77 ± 2.07	$\mathbf{6.32 \pm 0.85}$	13.25 ± 1.99	8.47 ± 1.27
WAVEFORM	15.73 ± 1.29	18.50 ± 4.41	$\mathbf{6.34 \pm 1.01}$	10.17 ± 1.74	$\mathbf{5.32 \pm 0.94}$
AUSTRALIAN	16.25 ± 0.96	16.98 ± 1.70	$\mathbf{5.92 \pm 1.50}$	$\mathbf{7.17 \pm 0.79}$	$\mathbf{6.36 \pm 0.97}$
BREAST CANCER	$\mathbf{3.85 \pm 0.89}$	23.00 ± 5.45	$\mathbf{3.27 \pm 0.68}$	21.63 ± 3.01	$\mathbf{3.09 \pm 0.55}$

Table 7: Results on a simple CNN architecture on MNIST dataset. We train the network on 5000 examples and test the in-distribution performance on the test set, which contains 8000 examples. We used 25 Monte Carlo samples to approximate the predictive distribution and in cases we used a subset of the data to solve the ODEs system we rely on 1000 samples. Calibration metrics are computed using 15 bins.

	MNIST DATASET				
	PRIOR PRECISION NOT OPTIMIZED				
METHOD	Accuracy ↑	NLL↓	Brier↓	ECE↓	MCE↓
MAP	95.02 ± 0.17	$\mathbf{0.167 \pm 0.005}$	$\mathbf{0.0075 \pm 0.0002}$	$\mathbf{1.05 \pm 0.14}$	39.94 ± 14.27
VANILLA LA	8.10 ± 0.74	2.521 ± 0.006	0.0937 ± 0.0000	11.79 ± 0.83	35.69 ± 6.47
LIN-LA	94.00 ± 0.29	0.350 ± 0.008	0.0143 ± 0.0004	16.94 ± 0.20	35.60 ± 0.09
RIEM-LA	$\mathbf{96.18 \pm 0.23}$	$\mathbf{0.177 \pm 0.007}$	$\mathbf{0.0073 \pm 0.0003}$	7.10 ± 0.23	$\mathbf{25.41 \pm 1.96}$
RIEM-LA (BATCHES)	94.98 ± 0.20	0.284 ± 0.008	0.0111 ± 0.0004	13.64 ± 0.20	29.23 ± 0.50
LIN-RIEM-LA	95.12 ± 0.27	0.180 ± 0.006	0.0080 ± 0.0003	4.19 ± 0.07	$\mathbf{22.02 \pm 3.97}$
LIN-RIEM-LA (BATCHES)	94.63 ± 0.19	0.229 ± 0.007	0.0097 ± 0.0004	7.84 ± 0.18	28.74 ± 3.67

	PRIOR PRECISION OPTIMIZED				
METHOD	Accuracy ↑	NLL↓	Brier↓	ECE↓	MCE↓
MAP	95.02 ± 0.17	0.167 ± 0.005	0.0075 ± 0.0002	1.05 ± 0.14	39.94 ± 14.27
VANILLA LA	88.69 ± 1.84	0.871 ± 0.026	0.0393 ± 0.0013	42.11 ± 1.22	50.52 ± 1.45
LIN-LA	94.91 ± 0.26	0.204 ± 0.006	0.0087 ± 0.0003	6.30 ± 0.08	39.30 ± 16.77
RIEM-LA	$\mathbf{96.74 \pm 0.12}$	$\mathbf{0.115 \pm 0.003}$	$\mathbf{0.0052 \pm 0.0002}$	2.48 ± 0.06	38.03 ± 15.02
RIEM-LA (BATCHES)	95.67 ± 0.19	0.170 ± 0.005	0.0072 ± 0.0002	5.40 ± 0.06	22.40 ± 0.51
LIN-RIEM-LA	95.44 ± 0.18	0.149 ± 0.004	0.0068 ± 0.0003	$\mathbf{0.66 \pm 0.03}$	39.40 ± 14.75
LIN-RIEM-LA (BATCHES)	95.14 ± 0.20	0.167 ± 0.004	0.0076 ± 0.0002	3.23 ± 0.04	$\mathbf{18.10 \pm 2.50}$

Table 8: Results on a simple CNN architecture on FMNIST dataset. We train the network on 5000 examples and test the in-distribution performance on the test set, which contains 8000 examples. We used 25 Monte Carlo samples to approximate the predictive distribution and in cases we used a subset of the data to solve the ODEs system we rely on 1000 samples. Calibration metrics are computed using 15 bins.

	FMNIST DATASET				
	PRIOR PRECISION NOT OPTIMIZED				
METHOD	Accuracy ↑	NLL↓	Brier↓	ECE↓	MCE↓
MAP	79.88 ± 0.09	0.541 ± 0.002	0.0276 ± 0.0000	$\mathbf{1.66 \pm 0.07}$	24.07 ± 1.50
VANILLA LA	10.16 ± 0.59	2.548 ± 0.050	0.0936 ± 0.0007	10.79 ± 1.05	27.19 ± 6.64
LIN-LA	79.18 ± 0.14	0.640 ± 0.004	0.0308 ± 0.0001	10.99 ± 0.54	$\mathbf{18.69 \pm 0.57}$
RIEM-LA	$\mathbf{82.70 \pm 0.23}$	$\mathbf{0.528 \pm 0.004}$	$\mathbf{0.0262 \pm 0.0002}$	9.95 ± 0.38	19.11 ± 0.43
RIEM-LA (BATCHES)	80.77 ± 0.10	0.582 ± 0.004	0.0285 ± 0.0001	10.34 ± 0.60	$\mathbf{18.52 \pm 0.64}$
LIN-RIEM-LA	80.94 ± 0.20	$\mathbf{0.528 \pm 0.004}$	$\mathbf{0.0265 \pm 0.0002}$	$\mathbf{1.59 \pm 0.18}$	$\mathbf{17.58 \pm 3.60}$
LIN-RIEM-LA (BATCHES)	79.95 ± 0.22	0.567 ± 0.005	0.0281 ± 0.0002	4.79 ± 0.58	$\mathbf{16.35 \pm 1.88}$

	PRIOR PRECISION OPTIMIZED				
METHOD	Accuracy ↑	NLL↓	Brier↓	ECE↓	MCE↓
MAP	79.88 ± 0.09	0.541 ± 0.002	0.0276 ± 0.0000	$\mathbf{1.66 \pm 0.07}$	24.07 ± 1.50
VANILLA LA	74.88 ± 0.83	1.026 ± 0.046	0.0482 ± 0.0019	31.63 ± 1.28	43.61 ± 2.95
LIN-LA	79.85 ± 0.13	0.549 ± 0.001	0.0278 ± 0.0000	3.23 ± 0.44	37.88 ± 17.98
RIEM-LA	$\mathbf{83.33 \pm 0.17}$	$\mathbf{0.472 \pm 0.001}$	$\mathbf{0.0237 \pm 0.0001}$	3.13 ± 0.48	10.94 ± 2.11
RIEM-LA (BATCHES)	81.65 ± 0.18	0.525 ± 0.004	0.0263 ± 0.0002	5.80 ± 0.73	35.30 ± 18.40
LIN-RIEM-LA	81.33 ± 0.10	0.521 ± 0.004	0.0261 ± 0.0002	$\mathbf{1.59 \pm 0.40}$	25.53 ± 0.10
LIN-RIEM-LA (BATCHES)	80.49 ± 0.13	0.529 ± 0.003	0.0269 ± 0.0002	2.10 ± 0.42	$\mathbf{6.14 \pm 1.42}$

Table 9: AUROC ↑ for OOD tasks for models trained on MNIST and tested against FashionMNIST, EMNIST, KMNIST. *(B)* indicates exponential maps solved using a subset of the training set. We can notice that our proposed RIEM-LA is consistently performing better than all the other approaches for OOD detection.

	PRIOR PRECISION NOT OPTIMIZED						
OOD DATA	MAP	VANILLA LA	RIEM-LA	LIN. LA	LIN. RIEM-LA	RIEM-LA (B)	LIN. RIEM-LA (B)
FMNIST	0.777 ± 0.057	0.523 ± 0.020	$\mathbf{0.911 \pm 0.011}$	0.876 ± 0.018	0.873 ± 0.023	0.891 ± 0.013	0.862 ± 0.026
EMNIST	0.851 ± 0.008	0.490 ± 0.015	$\mathbf{0.900 \pm 0.004}$	0.897 ± 0.005	0.905 ± 0.003	0.872 ± 0.005	$\mathbf{0.901 \pm 0.004}$
KMNIST	0.877 ± 0.005	0.511 ± 0.013	$\mathbf{0.949 \pm 0.002}$	0.930 ± 0.002	0.941 ± 0.002	0.938 ± 0.003	0.941 ± 0.002

	PRIOR PRECISION OPTIMIZED						
OOD DATA	MAP	VANILLA LA	RIEM-LA	LIN. LA	LIN. RIEM-LA	RIEM-LA (B)	LIN. RIEM-LA (B)
FMNIST	0.777 ± 0.057	0.681 ± 0.029	$\mathbf{0.917 \pm 0.017}$	0.854 ± 0.024	0.809 ± 0.027	0.897 ± 0.012	0.841 ± 0.025
EMNIST	0.851 ± 0.008	0.768 ± 0.018	$\mathbf{0.917 \pm 0.001}$	0.888 ± 0.006	0.871 ± 0.003	0.899 ± 0.003	0.888 ± 0.003
KMNIST	0.877 ± 0.005	0.813 ± 0.012	$\mathbf{0.953 \pm 0.001}$	0.921 ± 0.003	0.906 ± 0.002	0.948 ± 0.002	0.926 ± 0.002

Table 10: AUROC ↑ for OOD tasks for models trained on FashionMNIST and tested against MNIST, EMNIST, KMNIST. *(B)* indicates exponential maps solved using a subset of the training set. It is interesting to notice that if we do not optimize the prior precision, then LIN-LA is performing the best together with our linearized approaches. If we optimize the prior, RIEM-LA performs the best of all approaches.

	PRIOR PRECISION NOT OPTIMIZED						
OOD DATA	MAP	VANILLA LA	RIEM-LA	LIN. LA	LIN. RIEM-LA	RIEM-LA (B)	LIN. RIEM-LA (B)
MNIST	0.715 ± 0.022	0.494 ± 0.014	0.895 ± 0.010	$\mathbf{0.937 \pm 0.012}$	$\mathbf{0.929 \pm 0.015}$	0.917 ± 0.006	$\mathbf{0.940 \pm 0.012}$
EMNIST	0.649 ± 0.009	0.495 ± 0.013	0.794 ± 0.011	$\mathbf{0.907 \pm 0.007}$	0.881 ± 0.009	0.814 ± 0.016	0.891 ± 0.008
KMNIST	0.718 ± 0.013	0.495 ± 0.013	0.886 ± 0.006	$\mathbf{0.924 \pm 0.005}$	$\mathbf{0.921 \pm 0.007}$	0.898 ± 0.007	$\mathbf{0.925 \pm 0.006}$

	PRIOR PRECISION OPTIMIZED						
OOD DATA	MAP	VANILLA LA	RIEM-LA	LIN. LA	LIN. RIEM-LA	RIEM-LA (B)	LIN. RIEM-LA (B)
MNIST	0.715 ± 0.022	0.861 ± 0.037	$\mathbf{0.921 \pm 0.011}$	0.864 ± 0.020	0.807 ± 0.016	$\mathbf{0.934 \pm 0.005}$	0.869 ± 0.016
EMNIST	0.649 ± 0.009	0.765 ± 0.050	$\mathbf{0.857 \pm 0.009}$	0.806 ± 0.004	0.750 ± 0.012	$\mathbf{0.851 \pm 0.010}$	0.795 ± 0.001
KMNIST	0.718 ± 0.013	0.827 ± 0.030	$\mathbf{0.910 \pm 0.006}$	0.846 ± 0.010	0.800 ± 0.009	$\mathbf{0.907 \pm 0.006}$	0.849 ± 0.009

CHAPTER 5

Clustering and co-clustering incomplete data with deep latent variable models

In this chapter, we present our paper "Clustering and co-clustering incomplete data with deep latent variable models". The paper presents preliminary results that are currently under review. The paper aims to first formalize clustering using a VAE and then analyze the implications of computing the cluster assignment by estimating the correct quantity and clustering using the mean of the encoding as mostly done in the literature. We then move our focus on extending the VAE with a mixture of Gaussians prior framework for tackling the co-clustering task.

Clustering and co-clustering incomplete data with deep latent variable models

Federico Bergamin [1] **Pierre-Alexandre Mattei** [*2] **Jes Frellsen** [*1]

Abstract

We focus on the problems of clustering and co-clustering with deep latent variable models. While deep learning methods have significantly advanced the task of clustering, this has not happened for co-clustering, where most of the advancements are powerful metric-based methods rather than deep probabilistic methods. In this paper, we start by formalizing clustering using deep latent variable models and then demonstrate how these models can be extended to perform co-clustering tasks. Our results show that deep latent variable models can achieve better or on-par results compared to metric-based methods in co-clustering tasks when data is fully observed. Leveraging their probabilistic nature, we show how they can naturally co-cluster data with missing values, maintaining performances close to the fully observed case. Additionally, we present methods for utilizing the model to impute missing values.

1. Introduction and Background

Cluster analysis is one of the fundamental tasks in machine learning. The goal is to find meaningful groups in unlabelled data, called clusters, such that examples of the same cluster are similar while distinct from instances in other clusters. The quality of the data representation plays a crucial role in the performance and effectiveness of clustering techniques. For this reason, deep learning based clustering approaches rapidly gained success in the last decade. Some techniques directly train standard deep classifier using information-theoretic unsupervised losses (Hu et al., 2017; Ohl et al., 2022). However, most deep clustering methods use deep nets to map the original high-dimensional data into a feature space more fit for clustering. Indeed, much research focuses on devising training paradigms that lead to data representations that are easy to cluster (Song et al., 2013; Xie et al., 2016; Yang et al., 2017; Tao et al., 2021). A classically probabilistic approach assumes data is generated conditioned on a latent cluster distribution. An example of such an approach is Variational Autoencoders (VAEs, Kingma & Welling, 2014; Rezende et al., 2014): a VAE can be fitted to the data, and subsequently, the latent representations can be clustered. To enforce a clustered latent space, Jiang et al. (2017) proposed to use a mixture of Gaussians (MoG) as a prior for the latent representation. This results in better performance than using a standard Gaussian prior and then using K-means on the learned representation. This method was then combined with similarity-based method (Yang et al., 2019) and extended to perform a wide variety of clustering tasks, such as constrained clustering (Manduchi et al., 2021), clustering survival data (Manduchi et al., 2022), and to discover multi-facet clusters of the same data (Li et al., 2019). For a more comprehensive description of all the possible aspects of deep clustering approaches, we refer to Aljalbout et al. (2018) and Ren et al. (2022).

Despite the success of deep learning methods in clustering, the use of deep learning in co-clustering is still limited. The idea of co-clustering is to cluster observations and features simultaneously (Hartigan, 1972). Applications of co-clustering can be found in recommender systems (George & Merugu, 2005; Deodhar & Ghosh, 2010; Xu et al., 2012), bioinformatics (Hanisch et al., 2002; Cheng et al., 2008), medicine (Marchello et al., 2022), and text data (Dhillon et al., 2003; Wang et al., 2009; Bergé et al., 2019).

Following Govaert & Nadif (2013, Chapter 3), co-clustering methods can be divided into probabilistic methods and metric-based methods. As in clustering, probabilistic approaches make an explicit assumption on the data generation process. Metric-based approaches, on the other hand, optimize a criterion defined by a proper metric function that tries to describe in the most precise way the data geometry by taking into account intra- and inter-block variances. A classic example is spectral co-clustering (Dhillon, 2001). Since the initial work of Govaert & Nadif (2003) for binary

[*]Equal supervision [1]Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark [2]Université Côte d'Azur, Inria, Maasai, LJAD, CNRS, Nice, France. Correspondence to: Federico Bergamin <fedbe@dtu.dk>.

This is paper is presenting preliminary work that is currently under review.

matrices, most works on probabilistic co-clustering focused on extending the framework to different input distributions, for example count data (Govaert & Nadif, 2010), categorical data (Keribin et al., 2015), and ordinal data (Jacques & Biernacki, 2018; Corneli et al., 2020). Metric-based method research, instead, focused on deriving different metrics, with most successful methods relying on optimal-transport based metrics (Laclau et al., 2017; Fettal et al., 2022). One attempt to combine deep learning and co-clustering was done by Xu et al. (2019) by using two deep autoencoders to learn an embedding for each observation and feature and then feed this representation to two different inference networks whose output is used by a variant of a Gaussian Mixture model to produce the co-cluster assignment. Although improving performance over non-deep learning methods, their approach results in more parameters than fitting two separate models on observation and features of a data matrix, giving up one of the main advantage of co-clustering (Nadif & Govaert, 2010). Although seeming a natural approach to explore, nobody extended the VAE framework, successful in the clustering setting, to the co-clustering task. The goal of this paper is to fill this gap.

Our contributions are as follows: 1) we formalize clustering using a VAE with MoG prior and study what are the main difference between computing the cluster assignment by estimating the correct quantity and clustering using the mean of the encoding as mostly done in the literature; 2) we extend the VAE with MoG prior framework to perform co-clustering both when data is fully observed and when data contain missing values both missing-at-random and missing completely at random; 3) we present two ways to perform missing values imputation using this model.

2. Clustering with a VAE: challenges and fixes

The starting point of our work is formalising clustering with deep latent variable models (DLVMs). In this context, we are interested in clustering the input data by using their latent representation. A typical approach is to use a VAE with a mixture of Gaussians prior, which forces the latent representation to be grouped in different clusters. As in the VAE setting, we refer to $p_\theta(\mathbf{x}|\mathbf{z})$ as our observation model and to $q_\phi(\mathbf{z}|\mathbf{x})$ as the variational distribution, where $\mathbf{x} \in \mathbb{R}^N$ is the input data and $\mathbf{z} \in \mathbb{R}^d$ refers to the unobserved latent variable. Instead of having a standard Gaussian prior, in this case, the prior distribution over the latent codes is defined as

$$p_\theta(\mathbf{z}) = \sum_{c=1}^{C} \pi_c \mathcal{N}(\mathbf{z}; \mu_c, \mathbf{\Sigma}_c), \tag{1}$$

where C represents the number of Gaussian components, which will determine our clusters.

The quantity of interest to perform clustering is the probability of example $\mathbf{x}_i$ belonging to a particular class, i.e., $p_\theta(c \mid \mathbf{x}_i)$. This quantity is given by

$$\begin{aligned} p_\theta(c \mid \mathbf{x}_i) &= \int p_\theta(c, \mathbf{z} \mid \mathbf{x}_i)\, \mathrm{d}\mathbf{z} \\ &= \int \frac{p_\theta(c \mid \mathbf{z}, \mathbf{x}_i) p_\theta(\mathbf{x}_i \mid \mathbf{z}) p_\theta(\mathbf{z})}{p(\mathbf{x}_i)}\, \mathrm{d}\mathbf{z}, \end{aligned} \tag{2}$$

which involves an integral over the latent space that is analytically intractable.

A workaround was suggested by Jiang et al. (2017), who assumed that $p_\theta(c|\mathbf{z}_i)$ is a good approximation for $p_\theta(c|\mathbf{x}_i)$, where $\mathbf{z}_i$ is the latent representation associated with $\mathbf{x}_i$. Therefore, given a sample $\mathbf{z}_i \sim q_\phi(\mathbf{z}|\mathbf{x}_i)$, clustering boils down to compute the probability of the latent variable belonging to a particular component of the prior, also called responsibility, which is defined as

$$p_\theta(c \mid \mathbf{z}) = \frac{\pi_c \mathcal{N}(\mathbf{z}; \mu_c, \mathbf{\Sigma}_c)}{\sum_{k=1}^{C} \pi_k \mathcal{N}(\mathbf{z}; \mu_k, \mathbf{\Sigma}_k)}. \tag{3}$$

When it comes to the implementation, some works use a random sample from the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x}_i)$ (Manduchi et al., 2021), while others use the mean of $q_\phi(\mathbf{z}|\mathbf{x}_i)$ (Jiang et al., 2017; Yang et al., 2019; Manduchi et al., 2022) to compute the responsibilities.

In this paper, we propose an efficient and tractable estimate of Equation (2) based on self-normalized importance sampling (SNIS) and analyse the benefits, if any, of using this approach compared to the one that is normally used. If we rewrite $p_\theta(c|\mathbf{x}_i)$ as

$$\begin{aligned} &p_\theta(c \mid \mathbf{x}_i) = \\ &\frac{1}{p(\mathbf{x}_i)} \int \frac{p_\theta(c|\mathbf{z}, \mathbf{x}_i) p_\theta(\mathbf{x}_i|\mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x}_i)} q_\phi(\mathbf{z}|\mathbf{x}_i)\, \mathrm{d}\mathbf{z}, \end{aligned} \tag{4}$$

then using SNIS, we obtain the estimate

$$p_\theta(c \mid \mathbf{x}_i) \approx \sum_{l=1}^{L} w_l p_\theta(c \mid \mathbf{z}_l), \tag{5}$$

where $\mathbf{z}_1, \ldots, \mathbf{z}_L \sim q(\mathbf{z} \mid \mathbf{x}_i)$ are L samples from the variational distribution and $w_l = \frac{r_l}{r_1 + \ldots + r_L}$ are the importance weights, with r_l is defined as:

$$r_l = \frac{p_\theta(\mathbf{x}_i \mid \mathbf{z}_l) p_\theta(\mathbf{z}_l)}{q_\phi(\mathbf{z}_l \mid \mathbf{x}_i)}. \tag{6}$$

In the limit $L \to \infty$, the estimate in Equation (5) will converge to the true value (Andrieu et al., 2003).

SNIS is less overconfident We consider two different experimental setups to study the benefits of using SNIS over the mean of the variational distribution $q_\phi(\mathbf{z}|\mathbf{x}_i)$ for clustering. In both setups, we consider a dataset consisting

Table 1: Clustering performance obtained using the tractable estimate using SNIS and using the mean of $q_\phi(\mathbf{z} \mid \mathbf{x})$ compared to using the true model used to generate the data. We evalaute the two techniques in terms of ARI, accuracy, cross-entropy, and average entropy. We report also the Wasserstein distance $\mathcal{W}(H(p), H(\hat{p}))$ between the per-sample entropy obtained by the two approaches and the one given by the oracle. We report mean and standard error from the mean computed using 5 different seeds.

	SETTING 1: N=1000, POOR CLUSTERS SEPARATION, HEAVY TAILS					
METHOD	ARI ↑	Accuracy↑	Cross-Entropy↓	Brier↓	avg entropy	$\mathcal{W}(H(p), H(\hat{p}))\downarrow$
MEAN	0.841 ± 0.001	0.938 ± 0.000	0.242 ± 0.015	0.0257 ± 0.0005	0.059 ± 0.004	0.1365 ± 0.0069
SNIS	0.839 ± 0.002	0.937 ± 0.001	0.214 ± 0.006	0.0252 ± 0.0004	0.203 ± 0.005	0.1025 ± 0.0062
ORACLE	0.853 ± 0.002	0.943 ± 0.001	0.174 ± 0.005	0.0225 ± 0.0005	0.134 ± 0.001	0.

	SETTING 2: N=5000, GOOD CLUSTERS SEPARATION, VERY HEAVY TAILS					
METHOD	ARI ↑	Accuracy↑	Cross-Entropy↓	Brier↓	avg entropy	$\mathcal{W}(H(p), H(\hat{p}))\downarrow$
MEAN	0.976 ± 0.004	0.991 ± 0.002	0.132 ± 0.037	0.0044 ± 0.0008	0.0008 ± 0.0002	0.0759 ± 0.0015
SNIS	0.978 ± 0.004	0.992 ± 0.002	0.063 ± 0.022	0.0037 ± 0.0008	0.0091 ± 0.0013	0.0434 ± 0.0031
ORACLE	0.986 ± 0.001	0.995 ± 0.000	0.024 ± 0.001	0.0023 ± 0.0001	0.0275 ± 0.0003	0.

of $K = 4$ clusters, three being drawn from multivariate Gaussian distributions and one from a multivariate Student-t distribution. The two setups differ in terms of distance between the clusters, number of datapoints, and degrees-of-freedom of the Student-t distribution. The first setting is characterized by poor cluster separation and heavy tails, while the second setup has good cluster separation but very heavy tails. The second setting is exactly the same one as considered in (Ohl et al., 2022). A full description of the experiment can be found in Appendix A. We compare the two approaches using classical clustering metrics like Adjusted Rand-Index (ARI) and clustering accuracy, but also in terms of cross-entropy. Since there is no correspondence between the true cluster label and the label given by the algorithm, we first have to compute the optimal assignment between the labels to compute the clustering accuracy and the cross-entropy. By knowing the true model used to generate the data, we can also compare the two approaches against the oracle model to assess for overconfidence.

From Table 1, we can see that while there is no difference in terms of ARI and accuracy between using the mean or SNIS to compute the clustering, the "classifier" obtained by using the mean is usually over-confident. This is reflected in the cross-entropy, brier-score, and average $p(c|\mathbf{x})$ entropy. SNIS, on the other hand, is less over-confident as it gets entropy values that are closer to the oracle model. Moreover, the uncertainty we get by using SNIS is a better estimation of the true model uncertainty, i.e., our model specification, than the one we obtain by using the mean as reflected by the Wasserstein distance between the per-sample entropy obtained by the two approaches and the one given by the oracle. The analysis above was carried out by considering the correct number of components in the MoG prior. In Appendix A, we also consider the case of using the wrong number of components. Although making the computation of uncertainty metrics challenging, we find that using the mean of $q_\phi(\mathbf{z}|\mathbf{x})$ for clustering works fine in terms of clustering accuracy, but the SNIS approach is able to capture the uncertainty better.

SNIS is independent of the encoder Another advantage of using SNIS is that it makes clustering robust concerning the learned encoder quality. In Appendix A, we present a simple illustrative example, where we consider the same dataset and trained models as before, but we substitute the encoder with a Gaussian $\mathcal{N}(0, 5)$. While the mean of $q_\phi(\mathbf{z}|\mathbf{x})$ assigned all datapoints to a single cluster, SNIS, on the other hand, is able to get results that are similar to using the trained encoder.

Takeaway: how should we cluster when using a VAE with MoG prior? If the goal is just a good cluster accuracy or other related clustering performance metrics and the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ is precise, then using the mean of the encoder is the way to go. If you are interested in reasoning under uncertainty, using SNIS provides a more meaningful uncertainty estimation.

3. Deep generative co-clustering

We now focus on extending the VAE with MoG prior to co-clustering. Assume $\mathbf{X} \in \mathbb{R}^{N \times M}$ to be a data matrix, where X_{ij} represents the value that feature j takes for observation i. We are going to assume the model in Figure 1 (left) with joint density given by

$$p(\mathbf{X}, \mathbf{U}, \mathbf{V}|\theta) = \left(\prod_{i=1}^{N} \prod_{j=1}^{M} p_\theta(X_{ij}|\mathbf{u}_i, \mathbf{v}_j) \right) \left(\prod_{i=1}^{N} p(\mathbf{u}_i) \right) \left(\prod_{j=1}^{M} p(\mathbf{v}_j) \right), \quad (7)$$

where $\mathbf{u}_i \in \mathbb{R}^d$ is the latent variable (or code) associated with observation i, and $\mathbf{v}_j \in \mathbb{R}^h$ is the latent variable (or code) associated with feature j. Also, $\mathbf{U} = (\mathbf{u}_i)_{i=1}^N \in \mathbb{R}^{N\times d}$ and $\mathbf{V} = (\mathbf{v}_j)_{j=1}^M \in \mathbb{R}^{M\times h}$. Depending on the modelling and parametrization choices, we can retrieve the same model under different names in the literature. For example, assuming a Gaussian prior with zero-mean and learnable variance over $\mathbf{U}$ and $\mathbf{V}$ and a conditional distribution $p(X_{ij}|\mathbf{u}_i, \mathbf{v}_j) = \mathcal{N}(X_{ij}|\mathbf{u}_i^T\mathbf{v}_j, \sigma^2)$, we recover probabilistic matrix factorization (Mnih & Salakhutdinov, 2007). By placing a Gaussian-Wishart prior over the mean and the precision of the priors, we get Bayesian Matrix Factorization (Salakhutdinov & Mnih, 2008). If we, instead, specify a multinomial distribution for both $\mathbf{U}$ and $\mathbf{V}$, with g and k classes, respectively, and we define an observation model as a parametric family with parameters depending on each one of the $g \times k$ blocks we get the Latent Block Model (Keribin et al., 2017).

In our work, we assume MoG priors for both $\mathbf{U}$ and $\mathbf{V}$ (Figure 1, right), while for the likelihood, we assume that

$$p_\theta(X_{ij}|\mathbf{u}_i, \mathbf{v}_i) = \Phi(X_{ij}|f_\theta(\mathbf{u}_i, \mathbf{v}_i)), \tag{8}$$

where Φ belongs to an appropriate family of parametric distributions, and f_θ is the *decoder* with parameters θ, parametrized with a neural network. For $\mathbf{X} \in \mathbb{R}^{N\times M}$, we would typically take Φ to be Gaussian and for $X \in \{0, 1\}^{N\times M}$ a product of Bernoulli distributions. The model we proposed can be seen as either *non-linear probabilistic matrix factorization* or *probabilistic neural matrix factorization*. The MoG prior introduces additional structure over the latent codes to encourage clustering in the same way that we need a MoG prior on a VAE to cluster data (see Appendix A for a discussion on clustering using different prior distributions). This makes our model different from other deep-learning or graph-based collaborative filtering approaches (He et al., 2017; Berg et al., 2017): while they still learn an embedding for both rows and columns, they are not encouraging those embedding to cluster.

Inference Since estimating θ directly by maximum likelihood is intractable, we rely on variational inference and propose the variational distribution

$$q_{\gamma\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X}) = \left(\prod_{i=1}^N q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot})\right)\left(\prod_{j=1}^M q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j})\right), \tag{9}$$

where $\mathbf{X}_{i\cdot}$ is the i row of $\mathbf{X}$, and $\mathbf{X}_{\cdot j}$ is the j'th column. We consider $q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot}) = \Psi(\mathbf{u}_i|g_\gamma(\mathbf{X}_{i\cdot}))$ and $q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j}) = \Psi(\mathbf{v}_j|h_\zeta(\mathbf{X}_{\cdot j}))$, where Ψ belongs to the variational family of distribution (usually Gaussian), g is the *encoder* for the row code and h is the *encoder* for the column code (both are neural networks with parameters ψ and ζ, respectively). To

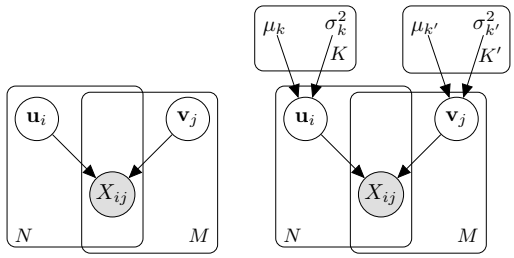


Figure 1: *Left:* Graphical models that is shared between several models in the literature, depending on the modelling choices (see Section 3). *Right:* Our proposed model with a Mixture of Gaussians prior for both $\mathbf{u}$ and $\mathbf{v}$ that we denote as *deep generative co-clustering* (DGCC).

estimate θ, γ, ζ, and the MoG prior parameters, we optimize the variational lower bound

$$\begin{aligned}\log p_\theta(\mathbf{X}) &= \log \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\left[\frac{p_\theta(\mathbf{X}, \mathbf{U}, \mathbf{V})}{q_{\gamma,\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X})}\right] \\ &\geq \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\left[\log \frac{p_\theta(\mathbf{X}, \mathbf{U}, \mathbf{V})}{q_{\gamma,\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X})}\right] = \mathcal{L}(\gamma, \zeta).\end{aligned} \tag{10}$$

We refer to the model above as *deep generative co-clustering* (DGCC). Finding an unbiased estimate of the gradient of Equation (10) suited for stochastic optimization, i.e., the use of mini-batches, is not straightforward. We provide a description of how to derive such estimate as well the full description of the training process in Appendix C. A similar parametrization to the one that we have presented was also proposed by Truong et al. (2021), however they do not consider a MoG prior and they restrict their observation model to a Bernoulli likelihood parametrized by the inner-product between the row and column latent codes. Moreover, they are not handling missing values in the dataset.

Why does this variational factorization make sense? The variational bound of Equation (10) will be tight when $q_{\gamma\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X}) = p_\theta(\mathbf{U}, \mathbf{V}|\mathbf{X})$. The intractable posterior $p_\theta(\mathbf{U}, \mathbf{V}|\mathbf{X})$ is a probability distribution over the relatively large space $\mathbb{R}^{d\times h}$ that we approximate by a variational distribution $q_{\gamma\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X})$ defined as in Equation (9), which may seem like a very strong assumption. Indeed, there is no reason for the true posterior $p_\theta(\mathbf{U}, \mathbf{V}|\mathbf{X})$ to factorise in a similar fashion. Nonetheless, we argue that this factorization makes sense because it is deeply tied to mean-field variational inference. Indeed, we will show that $q_{\gamma\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X})$ should be a good approximation of another (intractable) variational distribution, the mean-field solution $q^*(\mathbf{U}, \mathbf{V}) = \operatorname{argmin}_{\mathcal{Q}} D_{KL}(q(\mathbf{U}, \mathbf{V})||p_\theta(\mathbf{U}, \mathbf{V}|\mathbf{X}))$, where $\mathcal{Q} = \{q \text{ such that } q(\mathbf{U}, \mathbf{V}) = q(\mathbf{U})q(\mathbf{V})\}$. A stan-

dard result in mean-field variational inference (Blei et al., 2017, Section 2.3) implies that

$$q^*(\mathbf{U}) \propto \exp\left(\mathbb{E}_{q^*(\mathbf{V})}[\log p(\mathbf{U}, \mathbf{V}, \mathbf{X})]\right). \quad (11)$$

Using the definition of our generative model, we get

$$q^*(\mathbf{U}) \propto \prod_{i=1}^{N} \exp\left(\mathbb{E}_{q^*(\mathbf{V})} \log(p(\mathbf{X}_{i\cdot}|\mathbf{u}_i, \mathbf{V})p(\mathbf{u}_i)p(\mathbf{V}))\right), \quad (12)$$

which means that $q^*(\mathbf{U})$ is proportional to a product of n factors, one for each $\mathbf{u}_i$, depending on $\mathbf{X}$ directly through $\mathbf{X}_{i\cdot}$, and indirectly through $q^*(\mathbf{V})$. This motivates the factorisation

$$q_\gamma(\mathbf{U}|\mathbf{X}) = \prod_{i=1}^{N} q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot}), \quad (13)$$

since $q_\gamma(\mathbf{U}|\mathbf{X})$ is also written as a product of n factors, one for each $\mathbf{u}_i$, depending on $\mathbf{X}$ directly through $\mathbf{X}_{i\cdot}$. Since $q^*(\mathbf{U})$ and $q_\gamma(\mathbf{U}|\mathbf{X})$ have the same functional dependence on the $\mathbf{X}_{1\cdot}, \dots, \mathbf{X}_{n\cdot}$, this means that, if $q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot})$ is expressive enough (i.e., if the encoder has very large capacity and the variational family can approximate any distribution), it is possible to have $q^*(\mathbf{U}) = q_\gamma(\mathbf{U}|\mathbf{X})$. We can follow the same derivations to motivate the factorisation of $q_\zeta(\mathbf{V}|\mathbf{X})$. An analysis of the same vein was recently conducted for simple hierachical models by Margossian & Blei (2023). In their language, we showed that, for our model and variational approximation, there is a solution to the amortisation interpolation problem. We can summarise these insights by the following chain of approximations

- *mean-field:* $p_\theta(\mathbf{U}, \mathbf{V}|\mathbf{X}) \approx q^*(\mathbf{U}, \mathbf{V})$ will be tight when $p_\theta(\mathbf{U}, \mathbf{V}|\mathbf{X}) = p_\theta(\mathbf{U}|\mathbf{X})p_\theta(\mathbf{V}|\mathbf{X})$,
- *amortised approximation of the mean-field:* $q^*(\mathbf{U}, \mathbf{V}) \approx q_{\gamma\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X})$ will be tight when the encoders have very large capacity and the variational family is large enough.

Beyond motivating the factorisation, this allows us to know under which conditions our bounds are tight. In particular, unlike standard VAEs, our bound will not be tight when the encoder has large capacity and the the variational family is very flexible, because this will not guarantee the tightness of the mean-field bound. In that case, our bound will be as good as mean-field variational inference.

Refitting the prior As already noticed by Ghosh et al. (2020), refitting the prior, or employing an *ex-post density estimation* over the latent space as they called it, can be beneficial not only for their regularized autoencoder but also for regular VAEs. Their work found that replacing the standard prior with a 10-component MoG post-hoc improves the VAE sample quality. In our work, instead, we found that refitting the prior over $\mathbf{U}$ and $\mathbf{Z}$ after the training is usually beneficial especially in cases where the latent space is clustered but the learning process led to merging components. This consequentially improves clustering performance.

3.1. Computing cluster assignments

Once the model is trained, we are interested in clustering both the rows and the columns of our dataset, i.e., computing $p(c|\mathbf{X}_{i\cdot})$ and $p(c|\mathbf{X}_{\cdot j})$ for all $i \in \{1, \dots, N\}$ and $j \in \{1, \dots, M\}$. We present how to compute the clustering assignment for the a row $\mathbf{X}_{i\cdot}$. This can be easily generalize to the column cluster assignment. We want to compute the quantity

$$p_\theta(c|\mathbf{X}_{i\cdot}) = \\ = \iint \frac{p_\theta(c|\mathbf{u}_i)\prod_{j=1}^{M} p_\theta(X_{1j}|\mathbf{u}_i, \mathbf{v}_j)p_\theta(\mathbf{u}_i)\prod_{j=1}^{M} p_\theta(\mathbf{v}_j)}{p_\theta(\mathbf{X}_{i\cdot})} \\ \frac{q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot})\prod_{j=1}^{M} q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j})}{q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot})\prod_{j=1}^{M} q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j})} \mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V} \quad (14)$$

which can be estimate also in this case using self-normalized importance sampling by first sampling $\mathbf{u}_i, \mathbf{v}_1, \dots \mathbf{v}_M \sim q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{1\cdot})\prod_{j=1}^{M} q_\zeta(\mathbf{v}_j \mid \mathbf{X}_{\cdot j})$ and then computing importance weights $w_l = \frac{r_l}{r_1+\dots+r_L}$, where r_l is defined as

$$r_l = \frac{\prod_{j=1}^{M} p_\theta(X_{ij} \mid \mathbf{u}_i^{(l)}, \mathbf{v}_j^{(l)})\prod_{j=1}^{M} p_\theta(\mathbf{v}_j^{(l)})p_\theta(\mathbf{u}_i^{(l)})}{q_\gamma(\mathbf{u}_i^{(l)} \mid \mathbf{X}_{i\cdot})\prod_{j=1}^{M} q_\zeta(\mathbf{v}_j^{(l)} \mid \mathbf{X}_{\cdot j})}. \quad (15)$$

A step-by-step derivation is presented in Appendix E.

The likelihood hints challenges in SNIS The ELBO of the DGCC model in Equation (10) can help us identify the possible challenges in computing the importance weights in Equation (15). Each $\log r_l$ is proportional to the variational lower-bound and, in a large dataset, it corresponds to to sample a large number of latent codes to estimate it. A single bad code has an impact in the $p(X_{ij}|\mathbf{u}_i, \mathbf{v}_j)$ of a full row or column, which will causes r_l to have high variance. This may result in an effective sample size of 1 in big dataset, with one single sample getting a weight equal to one and, therefore, loosing the advantages of performing importance sampling. We found that an effective and yet simple way to mitigate this behaviour is to use truncated importance sampling (Ionides, 2008) when estimating Equation (14).

3.2. Training in presence of missing values

Given the probabilistic nature of our DGCC model, we can easily learn it on datasets with missing values. In this case, we can represent the dataset $\mathbf{X}$ as the union of the observed values $\mathbf{X}^o$ and the missing values $\mathbf{X}^m$. Following previous work on training deep latent variable models in

Table 2: Co-clustering error in the synthetic datasets. Mean and standard error over 25 different experiments are reported. We consider 5 different seed for creating a synthetic dataset and for each of them we train 5 different models. $(*)$ indicates that the results were taken from (Laclau et al., 2017).

EXPERIMENT	GLBM$^{(*)}$	CCOT-GW	BCOT	BCOT (gt prop)	DGCC	DGCC (prior refitted)
D1	0.021 ± 0.001	0.0449 ± 0.0018	0.0585 ± 0.0008	0.0605 ± 0.0016	0.0129 ± 0.0025	$\mathbf{0.0 \pm 0.0}$
D2	0.032 ± 0.004	0.0214 ± 0.0021	0.3068 ± 0.0011	0.1911 ± 0.0024	0.0165 ± 0.0022	$\mathbf{0.0 \pm 0.0}$
D3	0.262 ± 0.002	0.3909 ± 0.0406	–	–	0.1020 ± 0.0020	$\mathbf{0.0 \pm 0.0}$
D4	0.115 ± 0.005	0.0764 ± 0.0090	–	–	0.0432 ± 0.0031	$\mathbf{0.0 \pm 0.0}$

presence of missing data (Nazabal et al., 2020; Mattei & Frellsen, 2019), under the missing-completely at random (MCAR) and missing-at-random (MAR) assumption, we can maximize the likelihood of the observed data $\mathbf{X}^o$. For the DGCC model, this corresponds to

$$\mathcal{L}(\gamma, \zeta) = \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X}^o)} \left[\log \frac{p_\theta(\mathbf{X}^o, \mathbf{U}, \mathbf{V})}{q_{\gamma,\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X}^o)} \right]. \quad (16)$$

The variational distribution factorizes as before, but in this setting, the two components will be defined as $q_\gamma(\mathbf{u}_i|\mathbf{X}^o_{i\cdot}) = \Psi(\mathbf{u}_i|g_\gamma(\iota(\mathbf{X}^o_{i\cdot})))$ and $q_\zeta(\mathbf{v}_j|\mathbf{X}^o_{\cdot j}) = \Psi(\mathbf{v}_j|h_\zeta(\iota(\mathbf{X}^o_{\cdot j})))$, where ι is an imputation function. In all our experiments, we let ι to be the *zero-imputation* function, which replace the missing values with 0s. Mattei & Frellsen (2019) suggested to estimate Equation (16) using a tighter bound based on importance sampling. However, due to the assumed model factorization in Equation (7) and in particular because rows are not i.i.d., optimizing such a bound using mini-batching is not straightforward.

3.3. Missing values imputation

Despite missing data imputation not being the main goal of this paper, we can leverage the probabilistic nature of DGCC to briefly present two different ways to impute missing values. A more detailed explanations of these procedure can be found in Appendix F. The first approach follows the recipe presented by Mattei & Frellsen (2019). For single imputations, we are interested in the expectation

$$\mathbb{E}[\mathbf{X}^m \mid \mathbf{X}^o] = \int \mathbf{X}^m p_\theta(\mathbf{X}^m \mid \mathbf{X}^o) \mathrm{d}\mathbf{X}^m. \quad (17)$$

As in case of the cluster assignment, this expectation can be estimated by self-normalized importance sampling with $p_\theta(\mathbf{X}^m|\mathbf{X}^o, \mathbf{U}, \mathbf{V}) q_{\gamma\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X}^o)$ as proposal distribution. This leads to the estimate

$$\mathbb{E}[\mathbf{X}^m \mid \mathbf{X}^o] \approx \sum_{l=1}^{L} w_l \mathbf{X}^m_l, \quad (18)$$

where $(\mathbf{X}^m_{(1)}, \mathbf{U}_{(1)}, \mathbf{V}_{(1)}), \ldots, (\mathbf{X}^m_{(L)}, \mathbf{U}_{(L)}, \mathbf{V}_{(L)})$ are L samples from the proposal distribution via ancestral sampling and $\mathbf{U}_{(l)} = \{\mathbf{u}^{(l)}_1, \ldots, \mathbf{u}^{(l)}_N\}$ and $\mathbf{V}_{(l)} = \{\mathbf{v}^{(l)}_1, \ldots, \mathbf{v}^{(l)}_M\}$. For all $l \in \{1, \ldots, L\}$, the importance weights are given by $w_l = \frac{r_l}{r_1 + \cdots + r_L}$, where

$$r_l = \frac{\prod_{i=1}^N \prod_{j=1}^M p_\theta(\mathbf{X}^o_{ij}|\mathbf{u}^{(l)}_i, \mathbf{v}^{(l)}_j) \prod_{i=1}^N p_\theta(\mathbf{u}^{(l)}_i) \prod_{j=1}^M p_\theta(\mathbf{v}^{(l)}_j)}{\prod_{i=1}^N q_\gamma(\mathbf{u}^{(l)}_i \mid \mathbf{X}^o_{i\cdot}) \prod_{j=1}^M q_\zeta(\mathbf{v}^{(l)}_j \mid \mathbf{X}^o_{\cdot j})}. \quad (19)$$

In this case, we also relied on truncated IS to overcome the issue discussed in Section 3.1. A step-by-step derivation of this is presented in Appendix F, where we also show, how we can impute the missing values of a test set. The approach for single imputations, can easily be extended to estimate $\mathbb{E}[h(\mathbf{X}^m) \mid \mathbf{X}^o]$, where $h(\mathbf{X}^m)$ is any integrable function of $\mathbf{X}^m$, which can be useful, e.g., if one wants to compute the optimal imputation under another loss than the square loss (Ipsen et al., 2021, Appendix B).

The second imputation approach we considered is based on Markov-Chain Monte Carlo. We derived a component-wise Metropolis-Hastings algorithm with independent samplers specific for our model (see Appendix F.3).

Table 3: Accuracy of clustering and imputing values on MNIST. Mean and standard error over 5 different seed is reported.

Method	MCAR	MAR
	CLUSTERING ACCURACY	
DGCC	0.5713 ± 0.0245	0.6388 ± 0.0202
K-means	0.5065 ± 0.0009	0.4023 ± 0.0103
	IMPUTATION ACCURACY	
MissForest	0.9519 ± 0.0001	0.8760 ± 0.0003
MIWAE	0.9567 ± 0.0001	0.9240 ± 0.0002
DGCC (SNIS)	0.9474 ± 0.0005	0.9189 ± 0.0002
DGCC (MCMC)	0.9473 ± 0.0004	0.9145 ± 0.0003

4. Experiments

To evaluate the performance of our approach, we first consider MNIST, which, although not being a co-clustering dataset, serves as a sanity check to give us insight of what our model is learning but also as a scalability test for our method. Then, we focus on evaluating the DGCC co-

clustering both when data is fully observed and incomplete. More experiments, such as clustering in term-document matrices and other imputation experiments are presented in Appendix H, where we also report a detailed description of all model architectures used for these experiments.

4.1. Insights from training DGCC on MNIST

We train our DGCC model on binarized flattened MNIST, i.e., considering each digit is now a row, and in presence of missing values. While MNIST is not a particularly suitable dataset for co-clustering, and therefore the assumptions of our model can be wrong, it has the advantage of being visual, of moderate scale, and it is also a standard toy dataset for deep generative models. We consider both a *missing-completely-at-random* (MCAR) and a *missing-at-random* (MAR) mechanism for MNIST, following the experimental setup of Mattei & Frellsen (2019), see Appendix G.2.2 for details. First, we want to understand to what extent our DGCC is able to cluster the rows of this matrix. From Table 3, we can see that in both missing values settings, our model is able to produce more accurate clusters than K-means, which is a good sanity check.

Another challenge of MNIST not being a co-clustering dataset is that we do not have a natural cluster label for the columns, which in our case correspond to specific pixels positions. We visualize the pattern our DGCC model is learning in the columns by comparing the encoding distribution $q_\zeta(\mathbf{v}_j|X_{\cdot j})$ of each column $X_{\cdot j}$ in the dataset against all the other $q_\zeta(\mathbf{v}_{j'}|X_{\cdot j'})$, where $j' \neq j$, by using the reverse KL divergence. This results in a 28×28 plot for each feature j, and in Figure 2 we show a 28×28 matrix of these 784 plots. We can notice that despite being trained on flattened MNIST, the encoding distributions of peripheral pixels are very close to each other. For pixels at the center of the image, instead, the model learns a spatial correlation with neighbouring pixels, which is usually a signal that is leveraged by a convolution neural network. This emergence of spatial information in DGCC latent variables is, again, a good sanity check.

We also test the quality of the learned generative model by evaluating its performance in imputing the missing values in both the MCAR and MAR setting. We compare our DGCC with MIWAE (Mattei & Frellsen, 2019) implemented using convolutional layers and with missForest, a state-of-the-art single imputation algorithms based on random forests (Stekhoven & Bühlmann, 2012). We use 10000 samples to compute the imputation with MIWAE and 250 and 500 samples for DGCC with SNIS and DGCC with MCMC respectively. From Table 3, we can see that in both settings our approach is competitive with the two state-of-the-art methods despite having a strong assumption on the data generation process that might not be the appropriate one for the MNIST dataset and without exploiting any inductive biases as opposed to MIWAE.

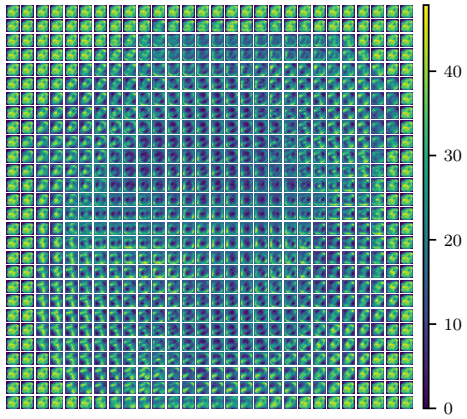


Figure 2: Each small plot in this 28×28 matrix represents the reverse KL divergence between the encoding distribution $q_\zeta(\mathbf{v}_j|X_{\cdot j})$ of that specific column $X_{\cdot j}$ and all other columns' encoding distribution. The matrix illustrates the clusters our DGCC model learned for every pixel in MNIST.

4.2. Co-clustering

The main goal of our proposed model is to perform co-clustering. To evaluate our DGCC model on this task, we consider the four synthetic datasets proposed by Laclau et al. (2017). The datasets are generated following a Gaussian latent block model (GBLM, Govaert & Nadif, 2013) and differ in the number of co-clusters, dataset sizes, and proportions of the clusters. In D1 and D2, co-clusters are well separated but with equal proportions in D1 and unequal proportions in D2. D3 and D4, instead, present ill-separated co-clusters with equal and unequal proportions respectively. A full description of the four datasets is provided in Appendix G.1. For each setting, we generate 5 different synthetic datasets and train all the considered models using 5 different seeds. We compare our method against the best metric-based co-clustering approaches: co-clustering through optimal trans-

Table 4: Co-clustering error in the D1 and D2 synthetic dataset considered. We consider 5 different MCAR missing values percentages. Mean and standard error over 5 different datasets and 5 different seeds.

Results D1, well separated co-clusters and equal proportions					
missing %	0.25	0.5	0.75	0.95	0.99
BCOT	0.058 ± 0.001	0.058 ± 0.001	0.060 ± 0.001	0.723 ± 0.006	0.861 ± 0.001
BCOT (gt)	0.062 ± 0.01	0.065 ± 0.002	0.063 ± 0.002	0.691 ± 0.007	0.861 ± 0.000
DGCC	0.035 ± 0.007	0.035 ± 0.007	0.176 ± 0.013	0.538 ± 0.014	$\mathbf{0.837 \pm 0.001}$
DGCC (model select.)	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.138 \pm 0.034}$	0.846 ± 0.003

Results D2, well separated co-clusters and unequal proportions					
missing %	0.25	0.5	0.75	0.95	0.99
BCOT	0.311 ± 0.001	0.319 ± 0.002	0.403 ± 0.005	0.845 ± 0.001	0.866 ± 0.001
BCOT (gt)	0.226 ± 0.003	0.210 ± 0.005	0.203 ± 0.004	0.715 ± 0.001	0.840 ± 0.001
DGCC	0.001 ± 0.001	0.006 ± 0.001	0.005 ± 0.001	0.043 ± 0.002	$\mathbf{0.426 \pm 0.003}$
DGCC (model select.)	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.007 \pm 0.003}$	0.471 ± 0.003

port with Gromov-Wasserstein barycenters (CCOT-GW, Laclau et al., 2017) and biclustering through optimal transport (BCOT, Fettal et al., 2022). CCOT-GW has the advantage of inferring the number of cluster for both the rows and the column during training, however, we did not manage to reproduce the results on the toy datasets using the provided MATLAB code, making it difficult to use the algorithm in the missing data setting. BCOT, on the other hand, always assume that rows and columns have the same number of clusters, which limited its application to only half of the considered synthetic datasets. We consider also the BCOT version that used the ground-truth cluster proportions, we denote it "BCOT (gt)". Furthermore, we report performance obtained using a Gaussian latent block model (GLBM). For our DGCC model, we used 1000 samples to estimate the cluster assignments in Equation (14) via self-normalized importance sampling. We compare all the methods in terms of the co-clustering error as defined in (Patrikainen & Meila, 2006), see Appendix G.1 for a detailed definition.

Fully observed-case Table 2 shows that our proposed approach consistently results in more precise clusters, while all baselines, on average, assign some observations to the wrong cluster. It also shows that refitting the prior is helpful, improving consistently the cluster assignments.

In presence of missing values We evaluate how robust these techniques are in performing co-clustering when the dataset contains missing values. Also, in this case we consider both MCAR and MAR settings. For MCAR, since the missing mechanism is independent of the data, we simply consider different percentages of missing uniformly at random values. Due to the fact that we did not manage to reproduce results using CCOT-GW, we consider only BCOT as a baseline for these experiments. Results are shown in Table 4 and Figure 3. We can see that DGCC is generally better and more robust than BCOT, resulting in general in more accurate clusters. Results on D3 and D4 are in the Appendix H. Although BCOT does not have a proper way to deal with missing values, we found that by using zero-imputation, it still produced meaningful clusters. An advantage of our DGCC method, with respect the considered metric-based approaches, is the possibility to perform model selection without looking at the co-clustering results. Indeed, since we consider 5 different datasets and 5 different seeds, we can select the best model for each dataset based on the log-likelihood. By doing so, we are able to co-cluster the datasets perfectly even with 75% missing values.

In the MAR setting, missing values depends on the observed data. To create such scenario, we consider the first half of each row in the data matrix to be fully observed, while the remaining part of each row is observed depending on the values of the first half. We describe the detailed simulation scheme in the Appendix H. From Table 5, we see that having a proper way to deal with missing values is beneficial and DGCC consistently finds the correct clusters, while metric-based approach suffers in the MAR setting. Indeed, it is common, for non-likelihood based method, to fail in the MAR setting (Mattei & Frellsen, 2019).

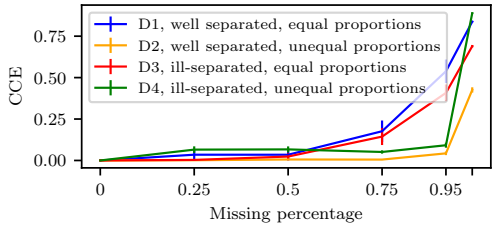


Figure 3: Co-clustering error for our DGCC model in the four different synthetic datasets considered for different percentage of missing values.

Table 5: Co-clustering error on different synthetic datasets with missing values in the MAR setting. We report mean and standard error from the mean over 5 different synthetic datasets and 5 different seeds.

Dataset	BCOT	BCOT (gt)	DGCC
D1	0.4049 ± 0.0108	0.5211 ± 0.0087	$\mathbf{0.0 \pm 0.0}$
D2	0.6521 ± 0.0019	0.3666 ± 0.0104	$\mathbf{0.0 \pm 0.0}$

5. Conclusions

We focus on clustering and co-clustering form the point of view of DLVMs. We propose a tractable estimate for $p(c|\mathbf{x})$, which is the genuine quantity we are interested in for clustering in this type of models and analyze the benefits of using this approach over computing the responsibilities by using the mean of $q_\phi(\mathbf{z}|\mathbf{x})$. We then proposed a DLVM to tackle co-clustering tasks presenting a technique to train such model when the dataset has missing values as well as two procedures to impute these values. We found empirically that our model performs better than metric-based co-clustering methods both on fully-observed data and in presence of missing values with different missing mechanisms.

Limitations The tractable estimate and our proposed model for co-clustering we come with an increase in the computational cost compared to the techniques we compare to. However, those limitations are balanced by the benefits of having more meaningful uncertainties and a model that is robust to missing values. A possible limitation is that quantitative evaluation of clustering methods is complicated (Hennig, 2018), and we need to rely on synthetic datasets where the true clusters are known to evaluate the models.

Impact statement

This paper tackles the problem of co-clustering using a deep latent variable model. As mentioned in the introduction, possible applications of co-clustering can be found in recommender systems, bioinformatics, medicine, and text data. However, this paper is not tackling any of these applications specifically but mostly focusing on building and developing the essential tools to make deep latent variable models applicable to those co-clustering tasks.

References

Aljalbout, E., Golkov, V., Siddiqui, Y., Strobel, M., and Cremers, D. Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*, 2018.

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. An introduction to mcmc for machine learning. *Machine learning*, 50:5–43, 2003.

Berg, R. v. d., Kipf, T. N., and Welling, M. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

Bergé, L. R., Bouveyron, C., Corneli, M., and Latouche, P. The latent topic block model for the co-clustering of textual interaction data. *Computational Statistics & Data Analysis*, 137:247–270, 2019.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Cheng, K.-O., Law, N.-F., Siu, W.-C., and Liew, A. W.-C. Identification of coherent patterns in gene expression data using an efficient biclustering algorithm and parallel coordinate visualization. *BMC bioinformatics*, 9(1):1–28, 2008.

Corneli, M., Bouveyron, C., and Latouche, P. Co-clustering of ordinal data via latent continuous random variables and not missing at random entries. *Journal of Computational and Graphical Statistics*, 29(4):771–785, 2020.

Deodhar, M. and Ghosh, J. Scoal: A framework for simultaneous co-clustering and learning from complex data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(3):1–31, 2010.

Dhillon, I. S. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 269–274, 2001.

Dhillon, I. S., Mallela, S., and Kumar, R. A divisive information theoretic feature clustering algorithm for text classification. *The Journal of machine learning research*, 3:1265–1287, 2003.

Dziugaite, G. K. and Roy, D. M. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

Fettal, C., Nadif, M., et al. Efficient and effective optimal transport-based biclustering. *Advances in Neural Information Processing Systems*, 35:32989–33000, 2022.

George, T. and Merugu, S. A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pp. 4–pp. IEEE, 2005.

Ghosh, P., Sajjadi, M. S. M., Vergari, A., Black, M. J., and Schölkopf, B. From variational to deterministic autoencoders. In *8th International Conference on Learning Representations, ICLR*, 2020.

Govaert, G. and Nadif, M. Clustering with block mixture models. *Pattern Recognition*, 36(2):463–473, 2003.

Govaert, G. and Nadif, M. Latent block model for contingency table. *Communications in Statistics—Theory and Methods*, 39(3):416–425, 2010.

Govaert, G. and Nadif, M. *Co-clustering: models, algorithms and applications*. John Wiley & Sons, 2013.

Hanisch, D., Zien, A., Zimmer, R., and Lengauer, T. Co-clustering of biological networks and gene expression data. *Bioinformatics*, 18:S145–S154, 2002.

Hartigan, J. A. Direct clustering of a data matrix. *Journal of the american statistical association*, 67(337):123–129, 1972.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pp. 173–182, 2017.

Hennig, C. Some thoughts on simulation studies to compare clustering methods. *Archives Data Sci. A*, 5(1), 2018.

Hu, W., Miyato, T., Tokui, S., Matsumoto, E., and Sugiyama, M. Learning discrete representations via information maximizing self-augmented training. In *International conference on machine learning*, pp. 1558–1567. PMLR, 2017.

Ionides, E. L. Truncated importance sampling. *Journal of Computational and Graphical Statistics*, 17(2):295–311, 2008.

Ipsen, N. B., Mattei, P., and Frellsen, J. not-miwae: Deep generative modelling with missing not at random data. In *9th International Conference on Learning Representations, ICLR*, 2021.

Jacques, J. and Biernacki, C. Model-based co-clustering for ordinal data. *Computational Statistics & Data Analysis*, 123:101–115, 2018.

Jiang, Z., Zheng, Y., Tan, H., Tang, B., and Zhou, H. Variational deep embedding: An unsupervised and generative approach to clustering. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 1965–1972, 2017.

Keribin, C., Brault, V., Celeux, G., and Govaert, G. Estimation and selection for the latent block model on categorical data. *Statistics and Computing*, 25(6):1201–1216, 2015.

Keribin, C., Celeux, G., and Robert, V. The latent block model: a useful model for high dimensional data. In *ISI 2017-61st world statistics congress*, pp. 1–6, 2017.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR,*, 2014.

Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, pp. 30–37, 2009.

Laclau, C., Redko, I., Matei, B., Bennani, Y., and Brault, V. Co-clustering through optimal transport. In *International conference on machine learning (ICML)*, pp. 1955–1964. PMLR, 2017.

Li, X., Chen, Z., Poon, L. K. M., and Zhang, N. L. Learning latent superstructures in variational autoencoders for deep multidimensional clustering. In *7th International Conference on Learning Representations, ICLR*, 2019.

Manduchi, L., Chin-Cheong, K., Michel, H., Wellmann, S., and Vogt, J. Deep conditional gaussian mixture model for constrained clustering. *Advances in Neural Information Processing Systems*, 34:11303–11314, 2021.

Manduchi, L., Marcinkevics, R., Massi, M. C., Weikert, T. J., Sauter, A., Gotta, V., Müller, T., Vasella, F., Neidert, M. C., Pfister, M., Stieltjes, B., and Vogt, J. E. A deep variational approach to clustering survival data. In *The Tenth International Conference on Learning Representations, ICLR*, 2022.

Marchello, G., Fresse, A., Corneli, M., and Bouveyron, C. Co-clustering of evolving count matrices with the dynamic latent block model: application to pharmacovigilance. *Statistics and Computing*, 32(3):41, 2022.

Margossian, C. C. and Blei, D. M. Amortized variational inference: When and why? *arXiv preprint arXiv:2307.11018*, 2023.

Mattei, P.-A. and Frellsen, J. Miwae: Deep generative modelling and imputation of incomplete data sets. In *International conference on machine learning*, pp. 4413–4423. PMLR, 2019.

Mnih, A. and Salakhutdinov, R. R. Probabilistic matrix factorization. *Advances in neural information processing systems*, 20, 2007.

Nadif, M. and Govaert, G. Model-based co-clustering for continuous data. In *2010 Ninth international conference on machine learning and applications*, pp. 175–180. IEEE, 2010.

Nazabal, A., Olmos, P. M., Ghahramani, Z., and Valera, I. Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, 107:107501, 2020.

Ohl, L., Mattei, P.-A., Bouveyron, C., Harchaoui, W., Leclercq, M., Droit, A., and Precioso, F. Generalised mutual information for discriminative clustering. In *Advances in Neural Information Processing Systems*, 2022.

Owen, A. B. *Monte Carlo theory, methods and examples*. https://artowen.su.domains/mc/, 2013.

Patrikainen, A. and Meila, M. Comparing subspace clusterings. *IEEE Transactions on Knowledge and Data Engineering*, 18(7):902–916, 2006.

Ren, Y., Pu, J., Yang, Z., Xu, J., Li, G., Pu, X., Yu, P. S., and He, L. Deep clustering: A comprehensive survey. *arXiv preprint arXiv:2210.04142*, 2022.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pp. 1278–1286. PMLR, 2014.

Rubinsteyn, A. and Feldman, S. fancyimpute: An imputation library for python, 2016. URL https://github.com/iskandr/fancyimpute.

Salakhutdinov, R. and Mnih, A. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pp. 880–887, 2008.

Song, C., Liu, F., Huang, Y., Wang, L., and Tan, T. Auto-encoder based data clustering. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 117–124, 2013.

Stekhoven, D. J. and Bühlmann, P. Missforest—nonparametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012.

Tao, Y., Takagi, K., and Nakata, K. Clustering-friendly representation learning via instance discrimination and feature decorrelation. In *9th International Conference on Learning Representations, ICLR*, 2021.

Truong, Q.-T., Salah, A., and Lauw, H. W. Bilateral variational autoencoder for collaborative filtering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pp. 292–300, 2021.

Wang, P., Domeniconi, C., and Laskey, K. B. Latent dirichlet bayesian co-clustering. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD*, pp. 522–537, 2009.

Xie, J., Girshick, R., and Farhadi, A. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pp. 478–487. PMLR, 2016.

Xu, B., Bu, J., Chen, C., and Cai, D. An exploration of improving collaborative recommender systems via user-item subgroups. In *Proceedings of the 21st international conference on World Wide Web*, pp. 21–30, 2012.

Xu, D., Cheng, W., Zong, B., Ni, J., Song, D., Yu, W., Chen, Y., Chen, H., and Zhang, X. Deep co-clustering. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pp. 414–422. SIAM, 2019.

Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*, pp. 3861–3870. PMLR, 2017.

Yang, L., Cheung, N.-M., Li, J., and Fang, J. Deep clustering by gaussian mixture variational autoencoders with graph embedding. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6440–6449, 2019.

Supplementary to: Clustering and co-clustering incomplete data with deep latent variable models

A. Clustering with a VAE

A.1. Do we really need a MoG prior for clustering?

Jiang et al. (2017) already showed that using a MoG prior on a VAE results in higher clustering quantitative metrics than using a standard prior and the use GMM or KMeans on the learned latent codes. This is mainly due to the fact that without a MoG prior, latents are not really force to cluster. In addition to that, the MoG prior can be seen as a more flexible prior than the standard Gaussian one for a VAE. More flexible priors, e.g., the Vamprior, have been proposed to alleviate the prior vs. aggregate posterior mismatch phenomena in VAEs, but these does not explicitly favor clustering in the latent space. To highlight this, we considered a VAE and the three mentioned prior and quantitatively compare performance on clustering MNIST. From Table 6 it is clear that if the goal is to perform clustering a MoG prior is needed as it results in better performance than all the other considered priors.

We can extend the same reasoning to co-clustering using VAE, and specifically our model. This is the reason why deep learning or graph-convolutional based collaborative filtering methods like (He et al., 2017; Berg et al., 2017) cannot be used out of the box to perform co-clustering. Indeed, while they are learning embeddings they are not forcing the embeddings to form clusters.

A.2. Example considered

We describe here the two datasets used to analyze the differences between computing clustering by using our tractable estimate of $p_\theta(c \mid \mathbf{x})$ using SNIS and directly computing the responsibility of the prior by using the mean of $q_\phi(\mathbf{z} \mid \mathbf{x})$. We follow the experiential setup proposed by (Ohl et al., 2022). The dataset is a mixture of three Gaussian distributions and one Student distribution. The four clusters are centered around the means

$$\mu_1 = [\alpha, \alpha], \quad \mu_2 = [\alpha, -\alpha], \quad \mu_3 = [-\alpha, \alpha], \quad \mu_4 = [-\alpha, -\alpha], \tag{20}$$

where α controls the distance between them. The covariance of all distributions is the identity scaled by $\sigma \in \mathbb{R}^+$.

For the first setting, we considered $N = 1000$ observation, $\sigma = 1$, $\alpha = 2$, and we consider 2 degrees of freedom for the Student-T distribution. The second setting is exactly the one considered by Ohl et al. (2022), where $N = 5000$, $\sigma = 1$, $\alpha = 5$, and 1 degree of freedom for the Student-t distribution. The datasets are illustrated in Figure 6.

For the experiments, we used a fully connected VAE with a MoG prior with 4 components and 2-dimensional latent space. In the first setting, both the encoder and the decoder have one hidden layer with 20 hidden units and `tanh` activation, while in the second setting, they still have only one layer but with 30 and 40 hidden units, respectively. Both models were trained for 7000 epochs with a batch size of 128. We used a learning rate of $1e-3$ for the VAE parameters and $1e-2$ for the mean and log-variance of the prior components. We kept the mixing probabilities fixed.

Table 6: Effect of VAE prior on clustering

Prior	ARI↑	ACC↑
$\mathcal{N}(0,1)$ + Kmeans	0.6204 ± 0.0118	0.7831 ± 0.0117
Vamprior + Kmeans	0.4684 ± 0.0040	0.6114 ± 0.0058
MoG prior	0.7850 ± 0.0209	0.8751 ± 0.0196

A.3. Clustering with a naive encoder

As mentioned in section 2, we considered the same datasets that we described above and trained models used to compute the main results, but we substitute the encoder with a Gaussian $\mathcal{N}(0, 5)$. From Table 7, we can see that in this setting, by using the mean, all the datapoints are assigned to a single cluster, resulting in $\sim 25\%$ cluster accuracy. By performing SNIS, on the other hand, we are able to get results that are similar to or better than using the trained encoder. In this case, results are better than using the trained encoder itself, but this is due to the small dimensionality of the latent space, where having a Gaussian with large variance does not suffer from the soaping bubble effect, i.e., samples of a d-dimensional unit Gaussian distribution being concentrated in a shell of radius $\sqrt{d}$.

Table 7: Naive encoder example. We consider the model consider in section 2 and substitute the trained encoder with a simpler but worse one. We can see that by using SNIS we have something that is independent and therefore robust compared on using the mean to cluster the data.

	Setting 1: N=1000, poor clusters separation, heavy tails				
Method	ARI ↑	Accuracy↑	Cross-Entropy↓	Brier↓	avg entropy
mean	0.0 ± 0.0	0.273 ± 0.006	4.179 ± 0.770	0.3143 ± 0.0163	0.288 ± 0.076
snis	0.840 ± 0.002	0.937 ± 0.001	0.208 ± 0.005	0.0250 ± 0.0004	0.203 ± 0.004

	Setting 2: N=5000, good clusters separation, very heavy tails				
Method	ARI ↑	Accuracy↑	Cross-Entropy↓	Brier↓	avg entropy
mean	0.0 ± 0.0	0.257 ± 0.001	21.196 ± 1.495	0.372 ± 0.0007	0.0008 ± 0.0002
snis	0.982 ± 0.003	0.993 ± 0.001	0.028 ± 0.004	0.0028 ± 0.0004	0.0187 ± 0.0011

A.4. What if we are using the wrong number of components in our prior?

In clustering, usually we do not know the correct number of cluster in advanced. In the analysis carried in Section 2, we highlight that using the mean of the approximate posterior is fine if metrics like ARI and accuracy are of interest, while SNIS provides a more accurate representation of the uncertainty. In that setting we considered the correct number of components for our MoG prior. We now present the same analysis when the number of clusters is misspecified. We consider a VAE with a MoG prior with six components, instead of four. Assessing uncertainty-related metrics is challenging when the number

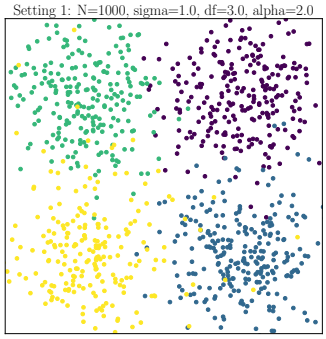


Figure 4: Setting 1: N=1000, poor clusters separation, heavy tails

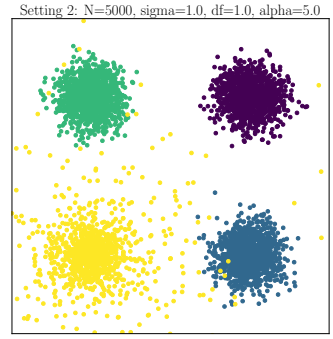


Figure 5: Setting 2: N=5000, good clusters separation, very heavy tails

Figure 6: Two datasets used to evaluate clustering using our SNIS estimate and using the mean of $q_\phi(\mathbf{z} \mid \mathbf{x})$.

of clusters is misspecified. For computing the cross-entropy, we considered all possible mergings of 6 clusters into 4 and reported the averages over the best merging for all seeds. Results in Table 8 are consistent with the conclusions outlined in Section 2.

B. Deriving the SNIS estimate for a VAE with MoG prior

In this section, we are going to derive step-by-step the tractable estimate of $p_\theta(c \mid \mathbf{x})$ by using self-normalized importance sampling. We have that

$$
\begin{aligned}
p_\theta(c \mid \mathbf{x}_i) &= \int p_\theta(c, \mathbf{z} \mid \mathbf{x}_i)\, \mathrm{d}\mathbf{z} && (21)\\
&= \int \frac{p_\theta(c, \mathbf{z}, \mathbf{x}_i)}{p(\mathbf{x}_i)}\, \mathrm{d}\mathbf{z} && (22)\\
&= \int \frac{p_\theta(c \mid \mathbf{z}, \mathbf{x}_i) p_\theta(\mathbf{z}, \mathbf{x}_i)}{p(\mathbf{x}_i)}\, \mathrm{d}\mathbf{z} && (23)\\
&= \int \frac{p_\theta(c \mid \mathbf{z}, \mathbf{x}_i) p(_\theta \mathbf{x}_i \mid \mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{x}_i)}\, \mathrm{d}\mathbf{z} && (24)\\
&= \frac{1}{p_\theta(\mathbf{x}_i)} \int p_\theta(c \mid \mathbf{z}, \mathbf{x}_i) p_\theta(\mathbf{x}_i \mid \mathbf{z}) p_\theta(\mathbf{z})\, \mathrm{d}\mathbf{z} && (25)\\
&= \frac{1}{p_\theta(\mathbf{x}_i)} \int p_\theta(c \mid \mathbf{z}, \mathbf{x}_i) p_\theta(\mathbf{x}_i \mid \mathbf{z}) p_\theta(\mathbf{z}) \frac{q_\phi(\mathbf{z} \mid \mathbf{x}_i)}{q_\phi(\mathbf{z} \mid \mathbf{x}_i)}\, \mathrm{d}\mathbf{z} && (26)\\
&= \frac{1}{p_\theta(\mathbf{x}_i)} \int \frac{p_\theta(c \mid \mathbf{z}, \mathbf{x}_i) p_\theta(\mathbf{x}_i \mid \mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x}_i)} q_\phi(\mathbf{z} \mid \mathbf{x}_i)\, \mathrm{d}\mathbf{z}. && (27)
\end{aligned}
$$

Although we have rewritten $p_\theta(c \mid \mathbf{x})$, it still involves an intractable integral and in addition to that, computing $p_\theta(\mathbf{x}_i)$ also involves solving another high-dimensional integral, which is still intractable. However, we can approximate this by using self-normalizing importance sampling, which give us the estimate

$$
\mathbb{E}[p_\theta(c \mid \mathbf{x}_i)] \approx \sum_{l=1}^{L} w_l p_\theta(c \mid \mathbf{z}_l), \tag{28}
$$

where $\mathbf{z}_1, \ldots, \mathbf{z}_L \sim q(\mathbf{z} \mid \mathbf{x}_i)$ are L samples from the variational distribution and $w_l = \frac{r_l}{r_1 + \ldots + r_L}$ are the importance weights with r_l is defined as

$$
r_l = \frac{p_\theta(\mathbf{x}_i \mid \mathbf{z}_l) p_\theta(\mathbf{z}_l)}{q_\phi(\mathbf{z}_l \mid \mathbf{x}_i)}. \tag{29}
$$

In the limit $L \to \infty$, the estimate $\mathbb{E}[p_\theta(c \mid \mathbf{x})]$ will converge to the true value (Andrieu et al., 2003).

The integral above can be further simplified by substituting the definition of $p_\theta(c \mid \mathbf{z})$ and $p_\theta(\mathbf{z})$ used in our model parametrization. $p_\theta(\mathbf{z})$ is the mixture of Gaussian prior, while $p_\theta(c \mid \mathbf{z})$ is the responsibility. By using their definition, we can write:

Table 8: Quantitative evaluation of clustering using a VAE and a MoG prior with misspecified number of components

METHOD	ARI↑	cross-entropy↓	avg entropy
	SETTING 1		
MEAN	0.7605 ± 0.0051	1.7814 ± 0.2371	0.3938 ± 0.0031
SNIS	0.7366 ± 0.0068	1.3185 ± 0.1473	0.5582 ± 0.0055
	SETTING 2		
MEAN	0.8265 ± 0.0420	2.2768 ± 0.3834	0.2765 ± 0.0542
SNIS	0.7768 ± 0.0404	1.6435 ± 0.2809	0.3553 ± 0.0503

$$p_\theta(c \mid \mathbf{x}_i) = \frac{1}{p_\theta(\mathbf{x}_i)} \int \frac{p_\theta(c \mid \mathbf{z}, \mathbf{x}_i) p_\theta(\mathbf{x}_i \mid \mathbf{z}) p_\theta(\mathbf{z})}{q(\mathbf{z} \mid \mathbf{x}_i)} q(\mathbf{z} \mid \mathbf{x}_i)\, \mathrm{d}\mathbf{z} \tag{30}$$

$$= \frac{1}{p_\theta(\mathbf{x}_i)} \int \frac{\pi_c \mathcal{N}(\mathbf{z}; \mu_c, \mathbf{\Sigma}_c) p_\theta(\mathbf{x}_i \mid \mathbf{z}) \sum_{k=1}^{C} \pi_k \mathcal{N}(\mathbf{z}; \mu_k, \mathbf{\Sigma}_k)}{\sum_{k=1}^{C} \pi_k \mathcal{N}(\mathbf{z}; \mu_k, \mathbf{\Sigma}_k) q(\mathbf{z} \mid \mathbf{x}_i)} q(\mathbf{z} \mid \mathbf{x}_i)\, \mathrm{d}\mathbf{z} \tag{31}$$

$$= \frac{1}{p_\theta(\mathbf{x}_i)} \int \frac{\pi_c \mathcal{N}(\mathbf{z}; \mu_c, \mathbf{\Sigma}_c) p_\theta(\mathbf{x}_i \mid \mathbf{z})}{q(\mathbf{z} \mid \mathbf{x}_i)} q(\mathbf{z} \mid \mathbf{x}_i)\, \mathrm{d}\mathbf{z}. \tag{32}$$

C. Training our model

To train our model, we optimize the following variational lower-bound introduced in section 3:

$$\log p_\theta(\mathbf{X}) = \log \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})} \left[\frac{p_\theta(\mathbf{X}, \mathbf{U}, \mathbf{V})}{q_{\gamma,\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X})} \right] \tag{33}$$

$$\geq \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})} \left[\log \frac{p_\theta(\mathbf{X}, \mathbf{U}, \mathbf{V})}{q_{\gamma,\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X})} \right] = \mathcal{L}(\gamma, \zeta). \tag{34}$$

By using the factorization of our model, we can rewrite the bound as

$$\mathcal{L}(\gamma, \zeta) = \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})} \left[\log \frac{p_\theta(\mathbf{X}, \mathbf{U}, \mathbf{V})}{q_{\gamma,\zeta}(\mathbf{U}, \mathbf{V}|\mathbf{X})} \right] \tag{35}$$

$$= \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})} \left[\log \frac{\left(\prod_{i=1}^{N} \prod_{j=1}^{M} p_\theta(X_{ij}|\mathbf{u}_i, \mathbf{v}_j) \right) \left(\prod_{i=1}^{N} p(\mathbf{u}_i) \right) \left(\prod_{j=1}^{M} p(\mathbf{v}_j) \right)}{\left(\prod_{i=1}^{N} q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot}) \right) \left(\prod_{j=1}^{M} q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j}) \right)} \right] \tag{36}$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{M} \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})} \log p_\theta(X_{ij}|\mathbf{u}_i, \mathbf{v}_j) + \sum_{i=1}^{N} \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})} \left[\log \frac{p(\mathbf{u}_i)}{q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot})} \right] \tag{37}$$

$$+ \sum_{j=1}^{M} \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})} \left[\log \frac{p(\mathbf{v}_j)}{q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j})} \right] \tag{38}$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{M} \mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})} \log p_\theta(X_{ij}|\mathbf{u}_i, \mathbf{v}_j) - \sum_{i=1}^{N} D_{KL}(q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot}) \,||\, p(\mathbf{u}_i))) \tag{39}$$

$$- \sum_{j=1}^{M} D_{KL}(q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j}) \,||\, p(\mathbf{v}_j)). \tag{40}$$

How we create batches We then split our data matrix into non-overlapping blocks of size $k \times k$, where k defined the number of rows and columns considered. This represents our batch of data to train our model at each optimization step. The tricky thing in implementing this procedure is to ensure that the two KL divergences appearing in the lower bound are correctly scaled, such that the sum of the whole batches results in the true lower bound. At the end of the epoch, i.e., after we process all the block of the data matrix, we permute both rows and columns and create new blocks.

Stochastic optimization – minibatching One of the challenges in training our DGCC model is to make the ELBO suitable for stochastic optimization, i.e., using minibatches. Specifically, we want to show that the way we trained our model using the batching mechanism introduced in the previous paragraph is unbiased.

We start by considering $\mathbf{X} \in \mathbb{R}^{N \times M}$ as before. If we look at the ELBO computed using the entire dataset we should have $N \times M$ log-likelihood terms $p(X_{ij}|\mathbf{u}_i, \mathbf{v}_j)$, and then we have N row-KL terms and M column-KL terms. Using gradient

descent on the full dataset with the ELBO loss normalised over rows and columns, would correspond to the following:

$$\mathcal{L}^{\text{norm}}(\gamma,\zeta) = \frac{1}{N\cdot M}\mathcal{L}(\gamma,\zeta) \tag{41}$$

$$= \frac{1}{N\cdot M}\sum_{i=1}^{N}\sum_{j=1}^{M}\mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\log p_\theta(X_{ij}|\mathbf{u}_i,\mathbf{v}_j) + \frac{1}{N\cdot M}\sum_{i=1}^{N}\mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\left[\log\frac{p(\mathbf{u}_i)}{q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot})}\right] \tag{42}$$

$$+ \frac{1}{N\cdot M}\sum_{j=1}^{M}\mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\left[\log\frac{p(\mathbf{v}_j)}{q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j})}\right], \tag{43}$$

and if we have a look at the gradient we get:

$$\nabla\mathcal{L}^{\text{norm}}(\gamma,\zeta) = \frac{1}{N\cdot M}\sum_{i=1}^{N}\sum_{j=1}^{M}\nabla\mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\log p_\theta(X_{ij}|\mathbf{u}_i,\mathbf{v}_j) + \frac{1}{N\cdot M}\sum_{i=1}^{N}\nabla\mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\left[\log\frac{p(\mathbf{u}_i)}{q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot})}\right]$$

$$+ \frac{1}{N\cdot M}\sum_{j=1}^{M}\nabla\mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\left[\log\frac{p(\mathbf{v}_j)}{q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j})}\right].$$

It is possible to compute unbiased estimates of this gradient using random reshuffling, minimatching and the reparametrisation trick as follows. Let $\sigma \in S(N)$ and $\rho \in S(M)$ be uniform random permutations of the rows and the columns, respectively. Then, each batch of size $n \times m$ gets examples from rows in the set $A_k = \{\sigma(k),\dots,\sigma(k+n-1)\}$, and from columns in the set $B_l = \{\rho(l),\dots,\rho(l+m-1)\}$. We define the ELBO over the batch as

$$\mathcal{L}_{\text{batch}}(\gamma,\zeta) = \sum_{i\in A}\sum_{j\in B}\mathbb{E}_{q_{\gamma,\zeta}(\mathbf{U},\mathbf{V}|\mathbf{X})}\log p_\theta(X_{ij}|\mathbf{u}_i,\mathbf{v}_j) - \frac{m}{M}\sum_{i\in A}D_{KL}(q_\gamma(\mathbf{u}_i|\mathbf{X}_{i\cdot})\,||\,p(\mathbf{u}_i))) \tag{44}$$

$$- \frac{n}{N}\sum_{j\in B}D_{KL}(q_\zeta(\mathbf{v}_j|\mathbf{X}_{\cdot j})\,||\,p(\mathbf{v}_j)). \tag{45}$$

Because subsampling is unbiased, the gradient of $\mathcal{L}_{\text{batch}}$, properly normalised, is unbiased, i.e., it is an estimate of the true gradient:

$$\mathbb{E}\left[\nabla\frac{1}{n\cdot m}\mathcal{L}_{\text{batch}}(\gamma,\zeta)\right] = \nabla\mathcal{L}^{\text{norm}}(\gamma,\zeta). \tag{46}$$

Moreover, it is possible to compute unbiased estimates of $\nabla\mathcal{L}_{\text{batch}}(\gamma,\zeta)$ using the reparametrisation trick. This finally allows us to use stochastic optimisation for the ELBO.

Coordinate descent Our DGCC model has the following parameters $\{\gamma,\zeta,\theta,\theta_r,\theta_c\}$, where γ and ζ are the parameters of the row and columns encoders, θ are the parameters of the decoder and θ_r and θ_c represents the parameters of the row and column MoG prior respectively. θ_r and θ_c contain both the mean and the variance of the components but also the mixing probabilities. During training, we optimize γ,ζ,θ at each iteration, while we alternate in optimizing θ_r and θ_c. In addition to that, when learning the mixing proportions, we leave them fixed for the first part of the training, and we start optimize them afterwards. We usually start learning them after $\frac{1}{4}$ of the epochs, but we did not run a full hyperparameter search to understand when it is best to start learning them.

D. Computing out-of-matrix log-likelihood

Assume we have a training set $\mathbf{X}$ of size $n \times m$ and a test-set $\hat{\mathbf{X}}$ of size $n' \times m$. If we are interested in computing the likelihood of $\hat{\mathbf{X}}$, due to our model factorization, we have to compute $p_\theta(\hat{\mathbf{X}} \mid \mathbf{X})$. We are going to derive step-by-step a tractable estimate for that. We have that

$$p_\theta(\hat{\mathbf{X}} \mid \mathbf{X}) = \iint p_\theta(\hat{\mathbf{X}},\hat{\mathbf{U}},\mathbf{V} \mid \mathbf{X})\,\mathrm{d}\hat{\mathbf{U}}\,\mathrm{d}\mathbf{V} \tag{47}$$

$$= \log\iint p_\theta(\hat{\mathbf{X}} \mid \hat{\mathbf{U}},\mathbf{V})p_\theta(\hat{\mathbf{U}},\mathbf{V} \mid \mathbf{X})\,\mathrm{d}\hat{\mathbf{U}}\,\mathrm{d}\mathbf{V} \tag{48}$$

We start by focusing on $p_\theta(\hat{\mathbf{U}}, \mathbf{V} \mid \mathbf{X})$, which as an initial step can be rewritten as:

$$p_\theta(\hat{\mathbf{U}}, \mathbf{V} \mid \mathbf{X}) = \frac{p_\theta(\hat{\mathbf{U}}, \mathbf{V}, \mathbf{X})}{p_\theta(\mathbf{X})}. \tag{49}$$

We can now look at the numerator:

$$\begin{aligned} p_\theta(\hat{\mathbf{U}}, \mathbf{V}, \mathbf{X}) &= \int p_\theta(\hat{\mathbf{U}}, \mathbf{V}, \mathbf{X}, \mathbf{U}) \, \mathrm{d}\mathbf{U} && (50)\\ &= \int p_\theta(\mathbf{X} \mid \mathbf{U}, \hat{\mathbf{U}}, \mathbf{V}) p_\theta(\mathbf{U}) p_\theta(\hat{\mathbf{U}}) p_\theta(\mathbf{V}) \, \mathrm{d}\mathbf{U} && (51)\\ &= \int p_\theta(\mathbf{X} \mid \mathbf{U}, \mathbf{V}) p_\theta(\mathbf{U}) p_\theta(\hat{\mathbf{U}}) p_\theta(\mathbf{V}) \, \mathrm{d}\mathbf{U} && (52) \end{aligned}$$

Plugging everything into our initial integral, we get:

$$p_\theta(\hat{\mathbf{X}} \mid \mathbf{X}) = \iiint p_\theta(\hat{\mathbf{X}} \mid \hat{\mathbf{U}}, \mathbf{V}) \frac{p_\theta(\hat{\mathbf{U}}) p_\theta(\mathbf{X} \mid \mathbf{V}, \mathbf{U}) p_\theta(\mathbf{V}) p_\theta(\mathbf{U})}{p_\theta(\mathbf{X})} \, \mathrm{d}\hat{\mathbf{U}} \, \mathrm{d}\mathbf{V} \, \mathrm{d}\mathbf{U} \tag{53}$$

We can then multiply and divide by the approximate posterior distributions:

$$p_\theta(\hat{\mathbf{X}} \mid \mathbf{X}) = \int \int \int p_\theta(\hat{\mathbf{X}} \mid \hat{\mathbf{U}}, \mathbf{V}) \frac{p_\theta(\hat{\mathbf{U}}) p_\theta(\mathbf{X} \mid \mathbf{V}, \mathbf{U}) p_\theta(\mathbf{V}) p_\theta(\mathbf{U})}{p_\theta(\mathbf{X})} \frac{q_\zeta(\mathbf{V} \mid \mathbf{X})}{q_\zeta(\mathbf{V} \mid \mathbf{X})} \frac{q_\gamma(\mathbf{U} \mid \mathbf{X})}{q_\gamma(\mathbf{U} \mid \mathbf{X})} \frac{q_\gamma(\hat{\mathbf{U}} \mid \hat{\mathbf{X}})}{q_\gamma(\hat{\mathbf{U}} \mid \hat{\mathbf{X}})} \, \mathrm{d}\hat{\mathbf{U}} \, \mathrm{d}\mathbf{V} \, \mathrm{d}\mathbf{U} \tag{54}$$

and this integral can be estimated by SNIS as

$$\mathbb{E}[p_\theta(\hat{\mathbf{X}} \mid \mathbf{X})] \approx \sum_{l=1}^{L} w_l p_\theta(\hat{\mathbf{X}} \mid \hat{\mathbf{U}}^{(l)}, \mathbf{V}^{(l)}), \tag{55}$$

by sampling $\mathbf{u}_1^{(l)}, \ldots, \mathbf{u}_N^{(l)} \mathbf{v}_1^{(l)}, \ldots \mathbf{v}_M^{(l)}, \hat{\mathbf{u}}_1^{(l)}, \ldots, \hat{\mathbf{u}}_{N'}^{(l)} \sim \prod_{i=1}^{N} q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}) \prod_{j=1}^{M} q_\zeta(\mathbf{v}_j \mid \mathbf{X}_{\cdot j}) \prod_{i=1}^{N'} q_\gamma(\hat{\mathbf{u}}_i \mid \hat{\mathbf{X}}_{i\cdot})$ and by computing weights $w_l = \frac{r_l}{r_1 + \ldots + r_L}$ where r_l is defined as follows:

$$r_l = \frac{\prod_{i=1}^{N} \prod_{j=1}^{M} p_\theta(X_{ij} \mid \mathbf{u}_i^{(l)}, \mathbf{v}_j^{(l)}) \prod_{j=1}^{M} p_\theta(\mathbf{v}_j^{(l)}) \prod_{i=1}^{N} p_\theta(\mathbf{u}_i^{(l)}) \prod_{i=1}^{N'} p_\theta(\hat{\mathbf{u}}_i^{(l)})}{\prod_{i=1}^{N} q_\gamma(\mathbf{u}_i^{(l)} \mid \mathbf{X}_{i\cdot}) \prod_{j=1}^{M} q_\zeta(\mathbf{v}_j^{(l)} \mid \mathbf{X}_{\cdot j}) \prod_{i=1}^{N'} q_\gamma(\hat{\mathbf{u}}_i^{(l)} \mid \hat{\mathbf{X}}_{i\cdot})} \tag{56}$$

E. Cluster assignment

We will now derive step-by-step the estimate based on SNIS that we used for clustering using our proposed model. We have that

$$\begin{aligned} p_\theta(c \mid \mathbf{X}_{i\cdot}) &= \int p_\theta(c, \mathbf{u}_i \mid \mathbf{X}_{i\cdot}) \, \mathrm{d}\mathbf{U} && (57)\\ &= \int \frac{p_\theta(c, \mathbf{u}_i, \mathbf{X}_{i\cdot})}{p_\theta(\mathbf{X}_{i\cdot})} \, \mathrm{d}\mathbf{U} && (58)\\ &= \int \frac{p_\theta(c \mid \mathbf{u}_i) p_\theta(\mathbf{u}_i, \mathbf{X}_{i\cdot})}{p_\theta(\mathbf{X}_{i\cdot})} \, \mathrm{d}\mathbf{U} && (59)\\ &= \int \frac{p_\theta(c \mid \mathbf{u}_i) \int p_\theta(\mathbf{X}_{i\cdot} \mid \mathbf{u}_i, \mathbf{V}) p_\theta(\mathbf{u}_i) p_\theta(\mathbf{V})}{p_\theta(\mathbf{X}_{i\cdot})} \, \mathrm{d}\mathbf{U} \, \mathrm{d}\mathbf{V} && (60)\\ &= \iint \frac{p_\theta(c \mid \mathbf{u}_i) p_\theta(\mathbf{X}_{i\cdot} \mid \mathbf{u}_i, \mathbf{V}) p_\theta(\mathbf{u}_i) p_\theta(\mathbf{V})}{p_\theta(\mathbf{X}_{i\cdot})} \, \mathrm{d}\mathbf{U} \, \mathrm{d}\mathbf{V}, && (61)\\ & && (62) \end{aligned}$$

where we rewrote $p_\theta(\mathbf{u}_i, \mathbf{X}_{i\cdot})$ by marginalizing out $\mathbf{V}$, and then used the our model factorization, i.e.,

$$p_\theta(\mathbf{u}_i, \mathbf{X}_{i\cdot}) = \int p_\theta(\mathbf{u}_i, \mathbf{X}_{i\cdot}, \mathbf{V}) \, \mathrm{d}\mathbf{V} \tag{63}$$

$$= \int p_\theta(\mathbf{X}_{i\cdot} \mid \mathbf{u}_i, \mathbf{V}) p_\theta(\mathbf{u}_i) p_\theta(\mathbf{V}) \, \mathrm{d}\mathbf{V}. \tag{64}$$

We then go on with the derivation of our estimate:

$$p_\theta(c \mid \mathbf{X}_{i\cdot}) = \iint \frac{p_\theta(c \mid \mathbf{u}_i) p_\theta(\mathbf{X}_{i\cdot} \mid \mathbf{u}_i, \mathbf{V}) p_\theta(\mathbf{u}_i) p_\theta(\mathbf{V})}{p_\theta(\mathbf{X}_{i\cdot})} \frac{q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}) q_\zeta(\mathbf{V} \mid \mathbf{X})}{q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}) q_\zeta(\mathbf{V} \mid \mathbf{X})} \, \mathrm{d}\mathbf{U} \, \mathrm{d}\mathbf{V} \tag{65}$$

$$= \iint \frac{p_\theta(c \mid \mathbf{u}_i) \prod_{j=1}^M p_\theta(X_{ij} \mid \mathbf{u}_i, \mathbf{v}_j) p_\theta(\mathbf{u}_i) \prod_{j=1}^M p_\theta(\mathbf{v}_j)}{p_\theta(\mathbf{X}_{i\cdot})} \frac{q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}) \prod_{j=1}^M q_\zeta(\mathbf{v}_j \mid \mathbf{X}_{\cdot j})}{q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}) \prod_{j=1}^M q_\zeta(\mathbf{v}_j \mid \mathbf{X}_{\cdot j})} \tag{66}$$

$$= \frac{1}{p_\theta(\mathbf{X}_{i\cdot})} \iint \frac{p_\theta(c \mid \mathbf{u}_i) \prod_j^M p_\theta(X_{ij} \mid \mathbf{u}_i, \mathbf{v}_j) p_\theta(\mathbf{u}_i) \prod_{j=1}^M p_\theta(\mathbf{v}_j)}{q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}) \prod_{j=1}^M q_\zeta(\mathbf{v}_j \mid \mathbf{X}_{\cdot j})} q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}) \prod_{j=1}^M q_\zeta(\mathbf{v}_j \mid \mathbf{X}_{\cdot j}) \tag{67}$$

The integral can then be estimate using SNIS as

$$\mathbb{E}[p_\theta(c \mid \mathbf{X}_{i\cdot})] \approx \sum_{l=1}^L w_l p_\theta(c \mid \mathbf{u}_i^{(l)}), \tag{68}$$

by sampling $\mathbf{u}_i^{(l)}, \mathbf{v}_1^{(l)}, \dots \mathbf{v}_M^{(l)} \sim q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}) \prod_{j=1}^M q_\zeta(\mathbf{v}_j \mid \mathbf{X}_{\cdot j})$ and by computing weights $w_l = \frac{r_l}{r_1 + \dots + r_L}$ where r_l is defined as follows:

$$r_l = \frac{\prod_{j=1}^M p_\theta(X_{ij} \mid \mathbf{u}_i^{(l)}, \mathbf{v}_j^{(l)}) \prod_{j=1}^M p_\theta(\mathbf{v}_j^{(l)}) p_\theta(\mathbf{u}_i^{(l)})}{q_\gamma(\mathbf{u}_i^{(l)} \mid \mathbf{X}_{i\cdot}) \prod_{j=1}^M q_\zeta(\mathbf{v}_j^{(l)} \mid \mathbf{X}_{\cdot j})} \tag{69}$$

Also in this setting, in the limit $L \to \infty$, the estimate $\mathbb{E}[p_\theta(c \mid \mathbf{X}_{i\cdot})]$ will converge to the true value

F. Missing values imputation

In this section, we present in a more detailed manner the two techniques we used to impute the missing values in our dataset.

F.1. Derivation of the SNIS estimate

The first approach we proposed follows the recipe presented by (Mattei & Frellsen, 2019). If single imputation is the goal we are interested in the following quantity:

$$\mathbb{E}[\mathbf{X}^m \mid \mathbf{X}^o] = \int \mathbf{X}^m p_\theta(\mathbf{X}^m \mid \mathbf{X}^o) \, \mathrm{d}\mathbf{X}^m. \tag{70}$$

$$= \iiint \mathbf{X}^m p_\theta(\mathbf{X}^m, \mathbf{U}, \mathbf{V} \mid \mathbf{X}^o) \, \mathrm{d}\mathbf{X}^m \, \mathrm{d}\mathbf{U} \, \mathrm{d}\mathbf{V} \tag{71}$$

$$= \iiint \mathbf{X}^m p_\theta(\mathbf{X}^m \mid \mathbf{X}^o, \mathbf{U}, \mathbf{V}) p_\theta(\mathbf{U}, \mathbf{V} \mid \mathbf{X}^o) \, \mathrm{d}\mathbf{X}^m \, \mathrm{d}\mathbf{U} \, \mathrm{d}\mathbf{V} \tag{72}$$

$$\tag{73}$$

By using the fact that

$$p_\theta(\mathbf{U}, \mathbf{V} \mid \mathbf{X}^o) = \frac{p_\theta(\mathbf{U}, \mathbf{V}, \mathbf{X}^o)}{p_\theta(\mathbf{X}^o)} \tag{74}$$

$$= \frac{p_\theta(\mathbf{X}^o | \mathbf{U}, \mathbf{V}) p_\theta(\mathbf{U}) p_\theta(\mathbf{V})}{p_\theta(\mathbf{X}^o)}, \tag{75}$$

we can then write

$$\mathbb{E}[\mathbf{X}^m \mid \mathbf{X}^o] = \iiint \mathbf{X}^m p_\theta(\mathbf{X}^m \mid \mathbf{X}^o, \mathbf{U}, \mathbf{V}) \frac{p_\theta(\mathbf{X}^o|\mathbf{U}, \mathbf{V}) p_\theta(\mathbf{U}) p_\theta(\mathbf{V})}{p_\theta(\mathbf{X}^o)} \,\mathrm{d}\mathbf{X}^m \,\mathrm{d}\mathbf{U} \,\mathrm{d}\mathbf{V} \tag{76}$$

$$\iiint \mathbf{X}^m p_\theta(\mathbf{X}^m \mid \mathbf{X}^o, \mathbf{U}, \mathbf{V}) \frac{p_\theta(\mathbf{X}^o|\mathbf{U}, \mathbf{V}) p_\theta(\mathbf{U}) p_\theta(\mathbf{V})}{p_\theta(\mathbf{X}^o)} \frac{q_\gamma(\mathbf{U}|\mathbf{X}^o) q_\zeta(\mathbf{V}|\mathbf{X}^o)}{q_\gamma(\mathbf{U}|\mathbf{X}^o) q_\zeta(\mathbf{V}|\mathbf{X}^o)} \,\mathrm{d}\mathbf{X}^m \,\mathrm{d}\mathbf{U} \,\mathrm{d}\mathbf{V} \tag{77}$$

which can be approximated using self-normalized importance sampling. The estimate can be computed as

$$\mathbb{E}[\mathbf{X}^m \mid \mathbf{X}^o] \approx \sum_{l=1}^{L} w_l \mathbf{X}_l^m, \tag{78}$$

where $(\mathbf{X}_{(1)}^m, \mathbf{U}_{(1)}, \mathbf{V}_{(1)}), \ldots, (\mathbf{X}_{(L)}^m, \mathbf{U}_{(L)}, \mathbf{V}_{(L)})$ are samples from the proposal distribution via ancestral sampling and $\mathbf{U}_{(l)} = \{\mathbf{u}_1^{(l)}, \ldots, \mathbf{u}_N^{(l)}\}$ and $\mathbf{V}_{(l)} = \{\mathbf{v}_1^{(l)}, \ldots, \mathbf{v}_M^{(l)}\}$. For all $l \in 1, \ldots, L$, we have that the importance weights are given by $w_l = \frac{r_l}{r_1 + \cdots + r_L}$, where

$$r_l = \frac{\prod_{i=1}^N \prod_{j=1}^M p_\theta(\mathbf{X}_{ij}^o | \mathbf{u}_i^{(l)}, \mathbf{v}_j^{(l)}) \prod_{i=1}^N p_\theta(\mathbf{u}_i^{(l)}) \prod_{j=1}^M p_\theta(\mathbf{v}_j^{(l)})}{\prod_{i=1}^N q_\gamma(\mathbf{u}_i^{(l)} \mid \mathbf{X}_{i\cdot}^o) \prod_{j=1}^M q_\zeta(\mathbf{v}_j^{(l)} \mid \mathbf{X}_{\cdot j}^o)}. \tag{79}$$

F.2. What if we want to impute the test set instead of the training set?

If we are interested, instead, in imputing the missing values in a different dataset $\hat{\mathbf{X}} \in \mathbb{R}^{n' \times m}$, we can do it by a simple modification of the estimate we presented in the previous section. In this setting, we are interested in the the following:

$$\mathbb{E}[\hat{\mathbf{X}}^m \mid \hat{\mathbf{X}}^o, \mathbf{X}^o] = \int \hat{\mathbf{X}}^m p_\theta(\hat{\mathbf{X}}^m \mid \hat{\mathbf{X}}^o, \mathbf{X}^o) \,\mathrm{d}\mathbf{X}^m \tag{80}$$

$$= \iiint \hat{\mathbf{X}}^m p_\theta(\hat{\mathbf{X}}^m \mid \hat{\mathbf{X}}^o, \mathbf{X}^o, \hat{\mathbf{U}}, \mathbf{V}) p_\theta(\hat{\mathbf{U}}, \mathbf{V} \mid \hat{\mathbf{X}}^o, \mathbf{X}^o) \,\mathrm{d}\mathbf{X}^m \,\mathrm{d}\hat{\mathbf{U}} \,\mathrm{d}\mathbf{V} \tag{81}$$

$$= \iiint \hat{\mathbf{X}}^m p_\theta(\hat{\mathbf{X}}^m \mid \hat{\mathbf{U}}, \mathbf{V}) p_\theta(\hat{\mathbf{U}}, \mathbf{V} \mid \hat{\mathbf{X}}^o, \mathbf{X}^o) \,\mathrm{d}\mathbf{X}^m \,\mathrm{d}\hat{\mathbf{U}} \,\mathrm{d}\mathbf{V} \tag{82}$$

We can focus on $p(\hat{\mathbf{U}}, \mathbf{V} \mid \hat{\mathbf{X}}^o, \mathbf{X}^o)$, we can write it as:

$$p_\theta(\hat{\mathbf{U}}, \mathbf{V} \mid \hat{\mathbf{X}}^o, \mathbf{X}^o) = \frac{p_\theta(\hat{\mathbf{U}}, \mathbf{V}, \hat{\mathbf{X}}^o, \mathbf{X}^o)}{p_\theta(\hat{\mathbf{X}}^o, \mathbf{X}^o)} \tag{83}$$

$$= \frac{p_\theta(\hat{\mathbf{X}}^o \mid \hat{\mathbf{U}}, \mathbf{V}, \mathbf{X}^o) p_\theta(\hat{\mathbf{U}}, \mathbf{V}, \mathbf{X}^o)}{p_\theta(\hat{\mathbf{X}}^o, \mathbf{X}^o)} \tag{84}$$

$$= \frac{p_\theta(\hat{\mathbf{X}}^o \mid \hat{\mathbf{U}}, \mathbf{V}) p(\hat{\mathbf{U}}) p_\theta(\mathbf{V}) \int p_\theta(\mathbf{X}^o \mid \mathbf{U}, \mathbf{V}) p_\theta(\mathbf{U}) \,\mathrm{d}\mathbf{U}}{p(\hat{\mathbf{X}}^o, \mathbf{X}^o)} \tag{85}$$

where we used the fact that under our model factorization

$$p_\theta(\hat{\mathbf{U}}, \mathbf{V}, \mathbf{X}^o) = p_\theta(\hat{\mathbf{U}}) p_\theta(\mathbf{V}, \mathbf{X}^o) \tag{86}$$

$$= p_\theta(\hat{\mathbf{U}}) \int p_\theta(\mathbf{V}, \mathbf{X}^o, \mathbf{U}) \,\mathrm{d}\mathbf{U} \tag{87}$$

$$= p_\theta(\hat{\mathbf{U}}) \int p_\theta(\mathbf{X}^o | \mathbf{V}, \mathbf{U}) p_\theta(\mathbf{U}) p_\theta(\mathbf{V}) \,\mathrm{d}\mathbf{U} \tag{88}$$

When putting everything together, we have

$$\mathbb{E}[\hat{\mathbf{X}}^m \mid \hat{\mathbf{X}}^o] = \tag{89}$$

$$= \iiiint h(\mathbf{X}^m) p(\mathbf{X}^m \mid \mathbf{X}^o, \mathbf{U}, \mathbf{V}) \frac{p(\hat{\mathbf{X}}^o \mid \hat{\mathbf{U}}, \mathbf{V}) p(\hat{\mathbf{U}}) p(\mathbf{V}) p(\mathbf{X}^o \mid \mathbf{U}, \mathbf{V}) p(\mathbf{U})}{p(\hat{\mathbf{X}}^o, \mathbf{X}^o)} \,\mathrm{d}\hat{\mathbf{U}} \,\mathrm{d}\mathbf{X}^m \,\mathrm{d}\mathbf{U} \,\mathrm{d}\mathbf{V} \tag{90}$$

By multiplying and divide with the variational distributions, we can derive a SNIS estimate for this, as we did in the previous section.

F.3. MCMC procedure

The goal of our MCMC sampler is to get samples from the following distribution $\pi(\mathbf{U}, \mathbf{V}) = p_\theta(\mathbf{U}, \mathbf{V}|\mathbf{X}^o)$. We cannot sample from this distribution, but we are able to evaluate it up to a normalizing constant. Indeed, we can write it as

$$\pi(\mathbf{U}, \mathbf{V}) = p_\theta(\mathbf{U}, \mathbf{V} \mid \mathbf{X}^o) \tag{91}$$

$$= \frac{p_\theta(\mathbf{U}, \mathbf{V}, \mathbf{X}^o)}{p_\theta(\mathbf{X}^o)} = \frac{p_\theta(\mathbf{X}^o \mid \mathbf{U}, \mathbf{V}) p_\theta(\mathbf{U}) p_\theta(\mathbf{V})}{P(\mathbf{X}^o)} \tag{92}$$

$$= \frac{\prod_{i=1}^N \prod_{j=1}^M p_\theta(X_{ij}^o \mid \mathbf{u}_i, \mathbf{v}_j) \prod_{i=1}^N p_\theta(\mathbf{u}_i) \prod_{j=1}^M p_\theta(\mathbf{v}_j)}{p_\theta(\mathbf{X}^o)} \tag{93}$$

For simplicity, we can also rephrase this problem in terms of a random variable $\mathbf{S} = [\mathbf{U}, \mathbf{V}]$, which is the concatenation of our row and columns latent codes, i.e., $\mathbf{S} = [\mathbf{u}_1, \dots, \mathbf{u}_N, \mathbf{v}_1, \dots, \mathbf{v}_M] \in \mathbb{R}^{(N+M)\times d}$, and we want to get samples from $\pi(\mathbf{S}) = p_\theta(\mathbf{S} \mid \mathbf{X}^o)$.

We proposed to use a component-wise Metropolis-Hastings algorithms with independent samplers, which sometimes is also called *Metropolized independence sampler* (Owen, 2013, Section 11.6). Specifically, we define a Markov chain update of $\mathbf{S}^{(t)}$, where the update consists of several sub-steps in which we update each one component of $\mathbf{S}$ at the time[1]. At each step, instead of using a proposal distribution $q(\mathbf{S}_i^{(t)} \mid \mathbf{S}_i^{(t-1)})$, that depends on the previous sample, we use the two learned encoders $q_\gamma(\mathbf{u}_i \mid \mathbf{X}_{i\cdot}^o)$ and $q_\zeta(\mathbf{v}_j \mid \mathbf{X}_{\cdot j}^o)$. These proposal distribution do not depend on the previous state and they are usually refer as independent sampler in the MC literature. At each sub-step i, we sample $\mathbf{S}_i^*$ for its proposal distribution and the acceptance function is given by:

$$\alpha_i = \min\left(1, \frac{\pi_i(\mathbf{S}_i^*, \mathbf{S}_{\neg i})}{\pi_i(\mathbf{S}_i, \mathbf{S}_{\neg i})} \frac{q(\mathbf{S}_i)}{q(\mathbf{S}_i^*)}\right) \tag{94}$$

Suppose that at sub-step k, we are sampling the the latent of the k-th row of the dataset, i.e., $\mathbf{u}_k^*$. To derive how the acceptance function looks like for our model, we start by rewriting $\frac{\pi_k(\mathbf{S}_k^*, \mathbf{S}_{\neg k})}{\pi_i(\mathbf{S}_k, \mathbf{S}_{\neg k})}$:

$$\frac{\pi_k(\mathbf{S}_k^*, \mathbf{S}_{\neg k})}{\pi_k(\mathbf{S}_k, \mathbf{S}_{\neg k})} = \frac{\prod_{j=1}^M p_\theta(X_{kj}^o \mid \mathbf{u}_k^*, \mathbf{v}_j) p_\theta(\mathbf{u}_k^*)}{\prod_{j=1}^M p_\theta(X_{kj}^o \mid \mathbf{u}_k, \mathbf{v}_j) p_\theta(\mathbf{u}_k)} \tag{95}$$

where we used the fact that most terms cancelled out because they are the same both in the numerator and in the denominator. The acceptance function in this case becomes

$$\alpha_k = \min\left(1, \frac{\prod_{j=1}^M p_\theta(X_{kj}^o \mid \mathbf{u}_k^*, \mathbf{v}_j) p_\theta(\mathbf{u}_k^*)}{\prod_{j=1}^M p_\theta(X_{kj}^o \mid \mathbf{u}_k, \mathbf{v}_j) p_\theta(\mathbf{u}_k)} \frac{q_\gamma(\mathbf{u}_k|\mathbf{X}_{k\cdot})}{q_\gamma(\mathbf{u}_k^*|\mathbf{X}_{k\cdot})}\right). \tag{96}$$

We can follow the same procedure to derive the acceptance function in case we are sampling the latent of the k-th column, $\mathbf{v}_k^*$. In this case, the acceptance function is given by

$$\alpha_k = \min\left(1, \frac{\prod_{i=1}^N p_\theta(X_{ik}^o \mid \mathbf{u}_i, \mathbf{v}_k^*) p_\theta(\mathbf{v}_k^*)}{\prod_{i=1}^N p_\theta(X_{ik}^o \mid \mathbf{u}_i, \mathbf{v}_k) p_\theta(\mathbf{v}_k)} \frac{q_\zeta(\mathbf{v}_k|\mathbf{X}_{\cdot k})}{q_\zeta(\mathbf{v}_k^*|\mathbf{X}_{\cdot k})}\right). \tag{97}$$

G. Experimental setup

In this section we are going to present in detail the experimental setup used in the experiment section of the paper. We will present more in depth the dataset we used and the architectures used for our model as well for the baselines.

[1] See section 3.5 of https://www.math.wustl.edu/~sawyer/hmhandouts/MetropHastingsEtc.pdf

G.1. Co-clustering

The datasets considered in the co-clustering experiments are taken from (Laclau et al., 2017). The datasets are generated following a Gaussian Latent block model (GBLM) (Govaert & Nadif, 2013) and differs in the number of co-clusters, dataset sizes, and proportions of the clusters. A detailed description of the differences are presented in Table 9 while an visual representation of the dataset clustered using the ground truth cluster assignments are in Figure 7.

Model architecture We train the same model architecture in all four datasets. The row and column encoder has a single layer with 5 hidden units and project the input to a 5-dimensional latent space. The decoder has also the same structure. We used Adam optimizer to learn all the parameters in our model. For the two encoders and the decoder parameters we used a learning rate of $1e-4$, for the mean and variance of the prior components we used a learning rate $5e-4$ while for the mixing probabilities $1e-4$.

Missing values In the missing-completely-at-random (MCAR) setting, we simply consider different percentages of missing uniformly at random values. For the missing-at-random (MAR) setting, instead, we first consider the first half of the matrix, i.e. $M/2$ columns, to be fully observed. For each row, then, we remove the second-half part with probability

$$\pi(\mathbf{X}_{i\cdot}) = \text{sigmoid}\left(\frac{1}{M/2}\sum_{k=1}^{M/2}\hat{X}_{ik}\right), \tag{98}$$

where we normalized the values of the first half of the matrix. Since $\pi(\mathbf{X}_{i\cdot})$ depends only on the observed feature, this schema is MAR but not MCAR.

Metric As mention in section 4, we compare all the methods in terms of the co-clustering error as defined in (Patrikainen & Meila, 2006):

$$\text{CCE}((\mathbf{z},\mathbf{w}),(\hat{\mathbf{z}},\hat{\mathbf{w}})) = e(\mathbf{z},\hat{\mathbf{z}}) + e(\mathbf{w},\hat{\mathbf{w}}) - e(\mathbf{z},\hat{\mathbf{z}}) \times e(\mathbf{w},\hat{\mathbf{w}}), \tag{99}$$

where $\hat{\mathbf{z}}$ and $\hat{\mathbf{w}}$ are the predicted cluster assignments and $\mathbf{z}$ and $\mathbf{w}$ contains the true clusters. The function e measure the proportion of misclassified cluster assignments.

Baselines As baselines we consider CCOT-GW (Laclau et al., 2017) and BCOT (Fettal et al., 2022). The code we used to run the baselines was taken from the open-sourced implementation released with the two papers. For CCOT-GW we used the following repository in MATLAB, while for BCOT, we used the following repository in Python. For both there is no license.

G.2. Missing data imputation

We evaluate our model in terms of missing data imputation on Movielens-100k and Movielens-1M, and on MNIST with both MCAR and MAR missing scheme. In this section, we are going to describe the experimental setup we used for both our model and the baselines.

G.2.1. MOVIELENS-100K AND MOVIELENS-1M EXPERIMENTS

We consider the same experimental setup as the one proposed by Dziugaite & Roy, 2015. We first split the dataset into a test test that contains 10% of the observations selected at random, while for the validation set we used 2% and 0.5% of the remaining training data on the MovieLens 100K and 1M datasets, respectively. We also centered the data around zero.

Model architecture We consider the same architecture for both datasets. Both row and column encoder consists of a single fully connected hidden layer of 200 hidden units. The decoder has the same structure but only 100 hidden units. For both the MoG prior we used 15 components and the encoder project the inputs into a 15-dimensional and 10-dimensional latent space for Movielens-100k and Movielens-1M, respectively. We used Adam optimizer with a learning rate of $1e-3$ for the encoders and decoder parameters, and the mean and variance of the MoG priors. For the mixing probabilities we still used Adam optimizer but with a learning rate of $1e-4$. We trained the models for 400 epochs. Since the metric is the RMSE we choose the best model in terms of RMSE on the validation set. Choosing the best model in terms of validation log-likelihood results in worse RMSE.

Baselines We consider biased matrix factorization (Koren et al., 2009) and fully connected MIWAE (Mattei & Frellsen, 2019) as baseline. For BiasedMF, we used the implementation provided by the library `fancyimpute` (Rubinsteyn & Feldman, 2016, Apache License 2.0). We train the model for 100 iterations using a learning rate of 0.001 and rank of 60. We implemented a fully connected MIWAE. For both Movielens-100k and Movielens-1M we train it on the users as input data, i.e. a single datapoint is given by the scores given by one user, and on the items as input data, i.e. a single datapoint is given by the scores obtained by a single movie. In Movielens-100k, it consists of a encoder with a single hidden layer with 100 hidden units and a decoder with a single layer with 25 units. In Movielens-1M, instead, the encoder has still one layer but with 400 units and the decoder has 100 units. We trained the models for 200 epochs with a learning rate of $1e-3$ using Adam optimizer.

G.2.2. MNIST EXPERIMENTS

We follow the experimental setup of Mattei & Frellsen (2019) on MNIST both for the MCAR and MAR setting experiments. In the MCAR setting, we consider 50% of the pixels missing uniformly at random. For the MAR setting instead, they propose this simple scheme: for each datapoints, either the top half, top quarter, or second quarter, is missing depending on the number of white pixels in the bottom half. Specifically, for each digit, they sample a binomial random variable h with 2 trials and success probability:

$$\pi(\mathbf{x}) = \frac{1}{\frac{784}{2}} \left(\sum_{j \in \text{bottom half of } \mathbf{x}} x_j \right) + 0.3$$

Then, depending on the value of the sample they:

- removed the second quarter if $h = 0$,
- remove top half if $h = 1$,
- remove first quarter if $h = 2$.

Example of data with missing values and with the imputation given by DGCC are shown in Figure 8 and Figure 9.

Model architecture The two encoders of our DGCC model has two hidden layers with 300 units each and `tanh` activation. The decoder has also two layers but with 200 units each. The MoG prior over the rows has 10 components while the one over the column has 5. We do not learn the mixing probabilities of the row prior. And both latent spaces are 10-dimensional. As before, we used Adam optimizer to train the parameters. We used a learning rate of $1e-3$ for all the parameters, except for the column mixing probabilities, where we used $1e-4$. We choose the model that get the best validation log-likelihood during training. Each model was trained for 400 epochs.

Baselines We compare our DGCC model with a MIWAE with convolutional layers and with missForest, a state-of-the-art single imputation algorithms based on random forests (Stekhoven & Bühlmann, 2012). We implemented missForest using `scikit-learn` `IterativeImputer` class in combination with a `RandomForestClassifier`. We used maximum 20 iterations and 100 tree with maximum depth 15 in our implementation. We implemented MIWAE using PyTorch. The encoder has 3 `conv2d` layers with 16, 32, 32 channels respectively and for all of them we used kernel of size 5, and stride and padding of 2. Each layer is follow by `ReLU` activation. These convolutional layers are then followed by two fully connected layers with 512 and 256 hidden units respectively. The decoder is the mirrored version of the encoder. Indeed, it starts with two fully connected layers with 256 and 512 hidden units followed by three `ConvTranspose2d` layers with 32, 16, and 1 channel respectively. As before we used kernel of size 5, stride and padding equal 2. For the last two `ConvTranspose2d` layers we also used `output_padding = 1`. We train MIWAE for 1000 epochs using Adam optimizer with learning rate equals to $1e-3$ and 50 IWAE samples during training. Batch size is 64 and we used a 50-dimensional latent space.

G.3. Software and license

For implementing our model, we relied on the following software/packages:

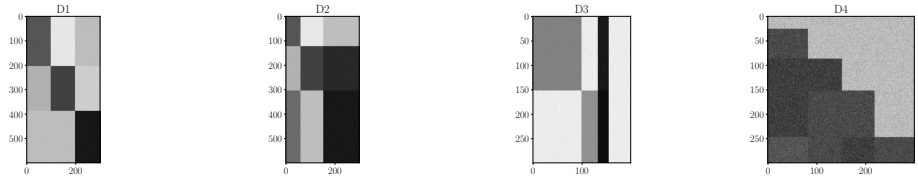


Figure 7: The four datasets considered for the co-clustering experiments.

- `numpy` (BSD-3-Clause license)
- `PyTorch` (BSD-style license)
- `scikit-learn` (BSD-3-Clause license)

For the baselines, instead, we rely on open-sourced implementations from the authors, although as mentioned before, these are without any license, and on personal implementations. For biased matrix-factorization, we used the `fancyimpute` library, which has an Apache License 2.0.

H. Additional results

In this section, we are going to present results and tables that did not make it to the main paper due to the space limit.

Complete results for the MAR setting in co-clustering In section 4.1 of the paper, we presented results for our DGCC when performing co-clustering when the dataset has missing values. In the MAR setting, we just presented the results for D1 and D2 because these are the only dataset were BCOT can be run on. In Table 10, we present results also for D3 and D4. We can see that in all the different dataset we considered our model is really robust to the MAR missing scheme and it is able to recover the correct co-clusters in most of the cases.

Term-document matrices results We consider two document-term datasets, the ACM and the DBLP datasets, that were considered in Fettal et al. (2022). Both are binary datasets, where the 1s indicated that a specific term is in the document. ACM has 3025 documents and 1870 words, while DBLP contains 4057 rows and 334 columns. There is a cluster label only for the rows. We use two different models for the two different datasets. For ACM, our DGCC both the encoders and the decoder have the same structure, i.e. a single hidden layer with 300 units. For DBLP, instead, both encoders have 2 hidden layers with 100 hidden units each, while the decoder is the same as before. In both experiments we used 15 dimensional latent codes for both rows and columns. We compare our DGCC model with BCOT with and without using the ground-truth cluster proportions. From Table 11, we can see that while our method performs better in the ACM dataset, in the DBLP dataset, instead, performance are more similar, with BCOT performing slightly better in terms of cluster accuracy and adjusted rand index.

Single imputation on Movielens Movielens-100k and Movielens-1M are two popular benchmarks containing user–movie ratings ranging from 1 to 5. We follow the experiment setup considered by Dziugaite & Roy (2015). As baselines, we consider a simple yet effective matrix factorization method in the form of biased matrix factorization (BiasedMF, Koren et al., 2009) and a fully-connected version of an importance-weighted autoencoder designed for handling missing values (MIWAE, Mattei & Frellsen, 2019). For MIWAE, we train the model both considering the users observations as input data as well as the items. For the imputation of missing values, we used 10,000 samples, while for our DGCC method, we used 1,000 samples both for SNIS and MCMC. Although having a strong assumption, we can see from Table 12 that our DGCC method is competitive with the other considered techniques when it comes to fill-in missing data in both Movielens-100k and Movielens-1M. In addition to that, our DGCC method over has the advantage to directly fill the missing data for a new user or a new movie thanks to its factorization without having to retrain it and change this architecture.

Imputation examples for MNIST In Figure 8 and Figure 9 we present imputation obtained when using our DGCC model on MNIST in the MCAR and MAR missing settings. We also plot Figure 2 again but in a larger format.

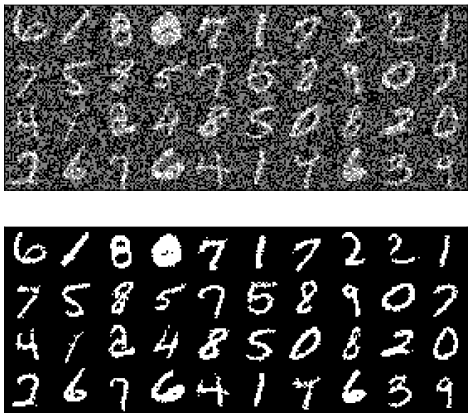


Figure 8: *Above*: examples of MNIST digits with missing completely at random (MCAR) missing patterns, *Below* imputation obtained by using our DGCC model.

Table 9: Complete description of the generative process behind all the datasets considered for co-clustering. $g \times m$ indicates the number of co-clusters. π and ρ are the cluster proportions for the rows and the columns respectively. Each co-cluster is defined by a specific Gaussian distribution with mean μ_{ij} and σ.

Dataset	Size	$g \times m$	Proportions	σ	μ
D1	600×300	3×3	$\pi = \rho = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$	0.1	$\begin{pmatrix} 4.0 & 0.5 & 1.5 \\ 1.8 & 4.5 & 1.1 \\ 1.5 & 1.5 & 5.5 \end{pmatrix}$
D2	600×300	3×3	$\pi = \rho = (0.2, 0.3, 0.5)$	0.15	$\begin{pmatrix} 4.0 & 0.5 & 1.5 \\ 1.8 & 4.5 & 5.1 \\ 3.5 & 1.5 & 5.5 \end{pmatrix}$
D3	300×200	2×4	$\pi = (0.5, 0.5)$ $\rho = (0.5, 0.2, 0.1, 0.2)$	0.2	$\begin{pmatrix} 4.0 & 0.5 & 7.5 & 0.5 \\ 0.5 & 3.5 & 7.8 & 0.5 \end{pmatrix}$
D4	300×300	5×4	$\pi = (0.1, 0.2, 0.2, 0.3, 0.2)$ $\rho = (0.25, 0.25, 0.25, 0.25)$	0.2	$\begin{pmatrix} 1.5 & 1.5 & 1.5 & 1.5 \\ 2.5 & 1.5 & 1.5 & 1.5 \\ 2.6 & 2.6 & 1.5 & 1.5 \\ 2.6 & 2.5 & 2.5 & 1.5 \\ 2.4 & 2.5 & 2.6 & 2.5 \end{pmatrix}$

Table 10: Co-clustering error on different synthetic datasets with missing values in the MAR setting. We report mean and standard error from the mean over 5 different synthetic datasets and 5 different seeds.

Dataset	BCOT	BCOT (gt)	DGCC
D1	0.4049 ± 0.0108	0.5211 ± 0.0087	$\mathbf{0.0 \pm 0.0}$
D2	0.6521 ± 0.0019	0.3666 ± 0.0104	$\mathbf{0.0 \pm 0.0}$
D3	–	–	$\mathbf{0.0 \pm 0.0}$
D4	–	–	$\mathbf{0.0108 \pm 0.0064}$

Table 11: Clustering results in documents-terms dataset. We measure clustering performance in terms of clustering accuracy (ACC), adjusted Rand-Index (ARI), and normalized mutual information (NMI). For all of them the higher the value the better. We report mean and standard error computed using 5 different seeds.

	ACM			DBLP		
METHOD	ACC	ARI	NMI	ACC	ARI	NMI
BCOT	0.748 ± 0.004	0.403 ± 0.007	0.360 ± 0.005	$\mathbf{0.592 \pm 0.023}$	$\mathbf{0.259 \pm 0.018}$	$\mathbf{0.249 \pm 0.016}$
BCOT (GT)	0.754 ± 0.002	0.412 ± 0.003	0.362 ± 0.003	0.428 ± 0.013	0.132 ± 0.012	0.136 ± 0.011
DGCC	$\mathbf{0.876 \pm 0.009}$	$\mathbf{0.681 \pm 0.017}$	$\mathbf{0.625 \pm 0.013}$	0.537 ± 0.008	0.196 ± 0.008	$\mathbf{0.251 \pm 0.007}$

Table 12: RMSE of imputed values on variants of Movielens. Mean and standard error from the mean over 5 different datasets splits.

Method	Movielens-100k	Movielens-1M
BiasedMF	0.946 ± 0.006	0.859 ± 0.003
User-MIWAE	0.928 ± 0.004	0.859 ± 0.001
Item-MIWAE	0.924 ± 0.002	0.856 ± 0.001
DGCC (SNIS)	$\mathbf{0.905 \pm 0.002}$	$\mathbf{0.851 \pm 0.001}$
DGCC (MCMC)	$\mathbf{0.904 \pm 0.002}$	$\mathbf{0.851 \pm 0.001}$

Figure 9: *Above*: examples of MNIST digits with missing at random (MAR) missing patterns, *Below* imputation obtained by using our DGCC model.

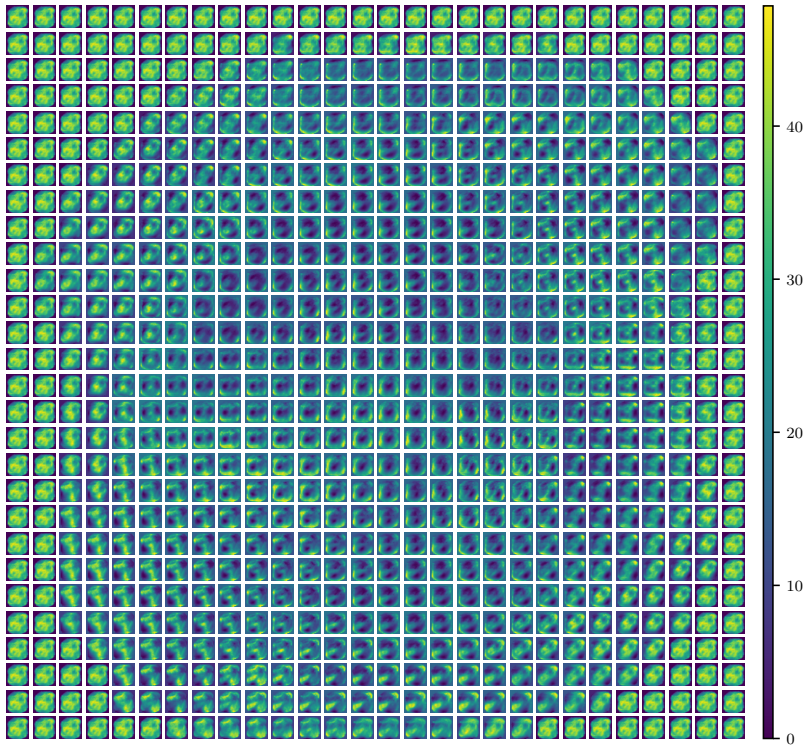


Figure 10: Each small plot in this 28×28 matrix represents the reverse KL divergence between the encoding distribution $q_\varsigma(\mathbf{v}_j|X_{\cdot j})$ of that specific column $X_{\cdot j}$ and all other columns' encoding distribution. The matrix illustrates the clusters our DGCC model learned for every pixel in MNIST.

CHAPTER 6

Guided autoregressive diffusion models with applications to PDE simulation

In this chapter, we present our work "Guided autoregressive diffusion models with applications to PDE simulation". The paper presents preliminary results that are currently under review. This work aims at analyzing and comparing different ways to use diffusion models for PDE simulation. We investigated the differences between learning a general and flexible joint score trained on short segments of trajectories as proposed by Rozet & Louppe (2023a) and conditioned it a posterior to different observations using reconstruction guidance and directly learning an amortized conditional score.

Guided Autoregressive Diffusion Models with Applications to PDE Simulation

Federico Bergamin [*1] **Cristiana Diaconu** [*2] **Aliaksandra Shysheya** [*2] **Paris Perdikaris** [3 4]
José Miguel Hernández-Lobato [2] **Richard E Turner** [2 4] **Emile Mathieu** [2]

Abstract

Solving partial differential equations (PDEs) is of crucial importance in science and engineering. Yet numerical solvers necessitate high space-time resolution which in turn leads to heavy computational cost. Often applications require solving the same PDE many times, only changing initial conditions or parameters. In this setting, data-driven machine learning methods have shown great promise, a principle advantage being the ability to simultaneously train at coarse resolutions and produce fast PDE solutions. In this work we introduce the Guided AutoRegressive Diffusion model (GUARD), which is trained over short segments from PDE trajectories and a posteriori sampled by conditioning over (1) some initial state to tackle *forecasting* and/or over (2) some sparse space-time observations for *data assimilation* purposes. We empirically demonstrate the ability of such a sampling procedure to generate accurate predictions of long PDE trajectories.

1. Introduction

Partial differential equations (PDEs) are ubiquitous as they are a powerful mathematical framework for modelling physical phenomena, ranging from the motion of fluids, to acoustics and thermodynamics. Applications are as varied as weather forecasting (Bauer et al., 2015), train aerodynamics profiling (Szudarek et al., 2022) and concert hall design (Vorlaender et al., 2015). Solving PDEs relies on numerical methods like finite element methods (Liu et al., 2022) which require very small space and time discretisation to obtain accurate and reliable solutions. As such, these conventional methods require significant computational resources. Subsequently, there has been a rising interest in leveraging deep learning methods to learn neural approximations of PDE solutions (Raissi et al., 2017). These approaches have recently been showing great promise for fluid dynamics in generating accurate solutions with significant improvement in computational efficiency compared to conventional numerical solvers (Pathak et al., 2022).

However, despite some significant progress, the problem of predicting long trajectories whilst preserving accuracy and stability remains a major challenge (Kochkov et al., 2021). Dynamics arising from PDEs can be particularly complex, due in many instances to non-linear terms, or high order derivatives, making it difficult to model such physical phenomena over long periods of time. Perhaps most critically, many PDEs can even exhibit chaotic behaviour (Hyman & Nicolaenko, 1986; Kevrekidis et al., 1990), causing small ambiguities in the dynamics to accumulate and leading to dramatically different solutions over time.

Both conventional and machine learning methods for (approximately) solving PDEs tend to process trajectories as deterministic sequences. As such, they are unable to cope with ambiguities arising from low-resolution dynamics from the discretisation, and tend to lack robustness, leading to error accumulation. Building on prior work, we propose to address these issues taking a *probabilistic* treatment of PDE dynamics (Yang & Sommer, 2023; Cachay et al., 2023), leveraging diffusion models as they have shown remarkable performance (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021). In particular, autoregressive amortised diffusion style of models, which work by additionally feeding previous states to the score network, are currently state of the art for long rollouts (Lippe et al., 2023; Kohl et al., 2023). However, (a) prior work has found that higher Markov order harms performance, and (b) amortisation does not straightforwardly handle data assimilation.

In this work we show how a diffusion model trained on short segments of joint states can solve both of these limitations. Our contributions are the following:

- We introduce a joint diffusion model which can be used autoregressively for sampling variable length trajectories, by iteratively generating states conditioned on past ones via reconstruction guidance (Song et al., 2022).

[*]Equal contribution [1]Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark [2]Department of Engineering, University of Cambridge, Cambridge, United Kingdom [3]Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, United States of America [4]Microsoft Research AI4Science.

This is paper is presenting preliminary work that is currently under review.

- We empirically demonstrate that this approach can perform on par with amortised models for generating long trajectories, and in particular that unlike amortised models, using higher Markov order improves its performance.
- We show that this model can jointly tackle data assimilation, as well as forecasting, and in most settings outperforms the strategy introduced in Rozet & Louppe (2023a) which consists of sampling the full sequence at once.

The proposed approach—coined GUARD—is the best of both worlds as it can generate trajectories under any set of observations, whilst it performs comparably to amortised models.

2. Background

Continuous-time diffusion models. In this section we briefly recall the main concepts underlying continuous diffusion models, and refer readers to Song et al. (2021) for a thorough presentation. We consider a forward noising process $(x(t))_{t\geq 0}$ associated with the following stochastic differential equation (SDE)

$$\mathrm{d}x(t) = -\tfrac{1}{2}x(t)\mathrm{d}t + g_t\mathrm{d}w(t),\ x(0) \sim p_0, \tag{1}$$

with $w(t)$ an isotropic Wiener process, p_0 the data distribution, and g_t a diffusion coefficient. The process defined by (1) is the well-known Ornstein–Uhlenbeck process, which geometrically converges to $\mathrm{N}(0, \mathrm{I})$. Additionally, for any $t \geq 0$, the noising kernel of (1) admits the following form

$$p_{t|0}(x(t)|x(0)) = \mathrm{N}(x(t)|\mu_t x(0), \sigma_t^2 \mathrm{I}), \tag{2}$$

with $\mu_t = x(0)e^{-\frac{1}{2}t}$ and $\sigma_t^2 = 1 - e^{-\int_0^t g_s^2 \mathrm{d}s}$. Importantly, under mild assumption on p_0, the time-reversal process $(\overleftarrow{x}(t))_{t\geq 0}$ also satisfies an SDE (Cattiaux et al., 2022; Haussmann & Pardoux, 1986) which is given by

$$\mathrm{d}\overleftarrow{x}(t) = \left\{-\tfrac{1}{2}\overleftarrow{x}(t) - g_t^2 \nabla \log p_t(\overleftarrow{x}(t))\right\}\mathrm{d}t + g_t \mathrm{d}w(t), \tag{3}$$

where p_t denotes the density of $x(t)$. Further, there exists an ordinary differential equation (ODE) which admits the same marginals as $\overleftarrow{x}(t)$, referred to as the *probability flow*

$$\mathrm{d}\bar{x}(t) = \left\{-\tfrac{1}{2}\bar{x}(t) - \tfrac{1}{2}g_t^2 \nabla \log p_t(\bar{x}(t))\right\}\mathrm{d}t. \tag{4}$$

In practice, the score $\nabla \log p_t$ is unavailable, and is approximated by a neural network $\mathbf{s}_\theta(t, \cdot) \approx \nabla \log p_t$, referred to as the *score network*. The parameters θ are learnt by minimising the denoising score matching (DSM) loss (Hyvärinen & Dayan, 2005; Vincent, 2011; Song et al., 2021)

$$\mathcal{L}(\theta) = \mathbb{E}[\|\mathbf{s}_\theta(t, x(t)) - \nabla \log p_t(x(t)|x(0))\|^2],$$

where the expectation is taken over the joint distribution of $t \sim \mathcal{U}([0, 1])$, $x(0) \sim p_0$ and $x(t) \sim p_{t|0}(\cdot|x(0))$ from (2).

Conditioning with diffusion models. The methodology introduced above allows for (approximately) sampling from $x(0) \sim p_0$. However, in many settings, one is actually interested in sampling from the conditional $p(x(0)|y)$ given some observations $y \in \mathcal{Y}$ with likelihood $p(y|\mathcal{A}(x))$ and $\mathcal{A} : \mathcal{X} \to \mathcal{Y}$ being a *measurement* operator. This conditional is given by Bayes' rule $p(x(0)|y) = p(y|x(0))p(x(0))/p(y)$, yet it is not available in closed form. Instead, an alternative is to simulate directly the *conditional* denoising process (Song et al., 2021; Chung et al., 2023; Didi et al., 2023)

$$\mathrm{d}\overleftarrow{x}(t) = \left\{-\tfrac{1}{2}\overleftarrow{x}(t) - g_t^2 \nabla \log p_t(\overleftarrow{x}(t)|y)\right\}\mathrm{d}t + g_t \mathrm{d}w(t).$$

There are two main ways to sample from this process. One can directly learn an approximation to the conditional score $\nabla \log p_t(x(t)|y)$ with a neural network that is *amortised* over observations y. Alternatively, one can combine a learnt diffusion prior model with the likelihood to obtain a posterior diffusion model. We recall these two approaches.

Amortising over observations. The simplest approach is to directly learn the conditional score $\log p_t(\overleftarrow{x}(t)|y)$ (Song et al., 2021; Ho et al., 2020). This can be achieved by additionally feeding the observation y to the score network $\mathbf{s}_\theta$, whose parameters are then learnt by minimising the following DSM loss $\mathcal{L}(\theta)$, with $(x(0), y) \sim p_{0\times y}$

$$\mathbb{E}[\|\mathbf{s}_\theta(t, x(t), y) - \nabla \log p_t(x(t)|x(0))\|^2]. \tag{5}$$

Reconstruction guidance. Alternatively, one can leverage Bayes rule, which allows to express the conditional score w.r.t. $x(t)$ as the sum of the following two gradients

$$\nabla \log p(x(t)|y) = \underbrace{\nabla \log p(x(t))}_{\text{prior}} + \underbrace{\nabla \log p(y|x(t))}_{\text{guidance}}, \tag{6}$$

where $\nabla \log p(x(t))$ can be approximated by a score network $s_\theta(t, x(t)) \approx \nabla \log p(x(t))$ trained on the prior data—with no information about the observation y. The conditioning comes into play via the $\nabla \log p(y|x(t))$ term which is referred to as *reconstruction guidance* (Ho et al., 2022b; Chung et al., 2022a; 2023; Meng & Kabashima, 2023; Wu et al., 2022; Song et al., 2022; 2023). Assuming a Gaussian likelihood $p(y|x(0)) = \mathrm{N}(y|\mathrm{A}x(0), \sigma_y^2\mathrm{I})$ with linear measurement $\mathcal{A}(x) = \mathrm{A}x$, we have that

$$\begin{aligned} p(y|x(t)) &= \int p(y|x(0))p(x(0)|x(t))\,\mathrm{d}x(0) \\ &\approx \int \mathrm{N}(y|\mathrm{A}x(0), \sigma^2)q(x(0)|x(t))\,\mathrm{d}x(0) \\ &= \mathrm{N}\left(y|\mathrm{A}\hat{x}(x(t)), (\sigma_y^2 + r_t^2)\mathrm{I}\right) \end{aligned} \tag{7}$$

where $p(x(0)|x(t))$ is not available in closed form and is therefore approximated by a Gaussian $q(x(0)|x(t)) \triangleq$

$\mathrm{N}(x(0)|\hat{x}(x(t)), r_t^2\mathrm{I}) \approx p(x(0)|x(t))$ with mean given by Tweedie's formula (Efron, 2011)

$$\hat{x}(x(t)) \triangleq \mathbb{E}[x(0)|x(t)] = \frac{x(t)+\sigma_t^2 \nabla \log p_t(x(t))}{\mu_t}, \quad (8)$$

and variance $\mathrm{Var}[x(0)|x(t)] \approx r_t^2$ being chosen as a monotonically increasing function (Rozet & Louppe, 2023a; Finzi et al., 2023; Song et al., 2022; Pokle et al., 2023).

3. Score-based forecasting and data assimilation

In this section, we present how diffusion models can be practically used for tackling forecasting and data-assimilation tasks in the context of PDE dynamics.

3.1. Problem setting

We denote by $\mathcal{P} : \mathcal{U} \to \mathcal{U}$ a differential operator taking functions $u : \mathbb{R}_+ \times \mathcal{Z} \to \mathcal{X} \in \mathcal{U}$ as input. Along with some initial $u(0, \cdot) = u_0$ and boundary $u(\tau, \partial\mathcal{Z}) = f$ conditions, these define a partial differential equation (PDE)

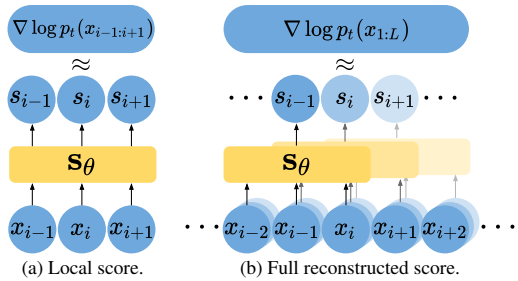
$$\frac{\partial u}{\partial \tau} = \mathcal{P}(u; \alpha) = \mathcal{P}\left(\frac{\partial u}{\partial z}, \frac{\partial^2 u}{\partial z^2}, \ldots; \alpha\right), \quad (9)$$

with coefficients α. For instance, the heat (i.e. diffusion) equation is given by $\frac{\partial u}{\partial \tau} = \mathcal{P}(u; \alpha) = \alpha \Delta u$, with α the thermal diffusivity. We assume to have access to many accurate numerical solutions from conventional solvers, which are discretised both in space and time. Let's denote such trajectories by $x_{1:L} = (x_1, \ldots, x_L) \in \mathbb{R}^{D \times L} \sim p_0$ with D and L being the size of the discretised space and time domains, respectively. The goal is then to solve similar PDEs (9) but with a possibly different initial condition u_0, boundary condition f or parameter α, as fast as possible while staying close to the true solution, as in weather forecasting for instance. Training a machine learning model on these solved trajectories can be worth the compute cost, as it may be able to provide fast approximation for similar PDEs.

We are generally interested in the problem of generating realistic PDE trajectories given observations, i.e. sampling from conditionals $p(x_{1:L}|y)$. Specifically, in this work we focus on two types of problems: (a) *forecasting*, which involves predicting future states $x_{i:i+H}$ given a past trajectory $x_{1:i-1}$, i.e. sampling from $p(x_{i:i+H}|x_{1:i-1})$, with H the forecast *horizon*; (b) *data assimilation*, which is concerned with inferring the ground state of a dynamical system given some sparse observed states x_o, i.e. sampling from $p(x_{1:L}|x_o)$.

3.2. Markovian dynamics

Markov score. Learning a diffusion model over the full joint distribution $p(x_{1:L}(t))$ can become prohibitively expensive for long trajectories, as it requires a score network taking the full sequence as input—$\mathbf{s}_\theta(t, x_{1:L}(t)) \approx \nabla_{x_{1:L}(t)} \log p(x_{1:L}(t))$, with memory footprint scaling linearly with the length L of the sequence.



(a) Local score. (b) Full reconstructed score.

Figure 1. Illustration of the local (a) and full (b) score. At training time, the local score is trained as shown in (a). When sampling simultaneously all states, the full score is reconstructed via multiple calls of the local score as shown in (b). We are not showing the diffusion time dependency t of random variables for clarity.

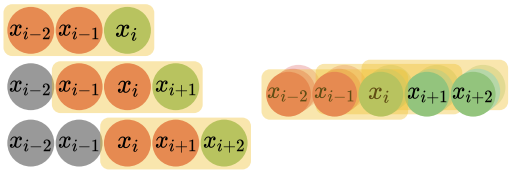
One approach to alleviate this, as suggested by Rozet & Louppe (2023a), is to assume a Markovian structure of order k such that the joint over data trajectories can be factorised into a series of conditionals $p(x_{1:L}) = p(x_1)p(x_2|x_1)\ldots p(x_{k+1}|x_{1:k}) \prod_{i=k+2}^{L} p(x_i|x_{i-k:i-1})$. Here we are omitting the time dependency $x_{1:L} = x_{1:L}(0)$ for clarity's sake. The score w.r.t. x_i can be written as

$$\begin{aligned} \nabla_{x_i} \log p(x_{1:L}) &= \nabla_{x_i} \log p(x_i|x_{i-k:i-1}) \\ &\quad + \textstyle\sum_{j=i+1}^{i+k} \nabla_{x_i} \log p(x_j|x_{j-k:j-1}) \\ &= \nabla_{x_i} \log p(x_i, x_{i+1:i+k}|x_{i-k:i-1}) \\ &= \nabla_{x_i} \log p(x_{i-k:i+k}) \end{aligned} \quad (10)$$

with $k+1 \leq i \leq L-k-1$, whilst formulas for $i \leq k$ and $i \geq L-k$ can be found in App. B.1. When noising the sequence with the SDE (1), there is no guarantee that $x_{1:L}(t)$ will still be a Markov. However, we still assume that $x_{1:L}(t)$ is Markov but with a larger order $k' > k$, as motivated in Rozet & Louppe (2023a, Sec 3.1). To simplify notations we use $k' = k$ in the rest of the paper. Consequently, instead of learning the entire joint score at once $\nabla_{x_{1:L}(t)} \log p(x_{1:L}(t))$, we only need to learn the *local* scores $\mathbf{s}_\theta(t, x_{i-k:i+k}(t)) \approx \nabla_{x_{i-k:i+k}(t)} \log p_t(x_{i-k:i+k}(t))$ with an input window size of $2k+1$ as illustrated in Fig. 1a. The main benefit is that the neural network only takes as input sequences of size $2k+1$ instead of L, with $k << L$. The local score network $\mathbf{s}_\theta$ is trained by minimising the following DSM loss $\mathcal{L}(\theta)$:

$$\mathbb{E}\left\|\mathbf{s}_\theta(t, x_{i-k:i+k}(t)) - \nabla \log p_{t|0}(x_{i-k:i+k}(t)|x_{i-k:i+k})\right\|^2$$

where the expectation is taken over the joint $i \sim \mathcal{U}([k+1, L-k])$, $t \sim \mathcal{U}([0, 1])$, $x_{i-k:i+k} \sim p_0$ and $x_{i-k:i+k}(t) \sim$



(a) Autoregressive (AR). (b) All at once (AAO).

Figure 2. Illustration of forecasting for two different sampling strategies using the same trained *local* score network. Each row corresponds to the sampling of a (conditional) denoising process. Red nodes are conditioned over, green nodes are being sampled, and grey nodes have already previously been sampled.

$p_{t|0}(\cdot|x_{i-k:i+k})$ given by the noising process (2).

Reconstructing full score. When necessary, the full score $\mathbf{s}_\theta(t, x_{1:L}(t)) \approx \nabla \log p_t(x_{1:L}(t))$ can then be reconstructed from these local scores, as shown in App. B.1 and summarised in Rozet & Louppe (2023a, Alg 2.). The ith component is given by $\mathbf{s}_i = \mathbf{s}_\theta(t, x_{i-k:i+k}(t))[k+1]$ which is the centre output of the score network as illustrated in Fig. 1b, except for the beginning and the end of the sequence for which the score is respectively given by $\mathbf{s}_{1:k} = \mathbf{s}_\theta(t, x_{1:2k+1}(t))[:k+1]$ and $\mathbf{s}_{L-k:L} = \mathbf{s}_\theta(t, x_{L-2k:L}(t))[k+1:]$. Sampling from the time-reversal (3) by plugging in $\mathbf{s}_\theta(t, x_{1:L}(t))$, we obtain a model for the full joint $p(x_{1:L})$. We stress that due to the structure of the score network, this can be achieved for any $L \geq 2k+1$.

3.3. Sampling

We can generate samples $x_{1:L} \sim p(x_{1:L})$ by solving the time-reversal SDE (3) over the sequence. We replace the true score with the trained score network and discretise the probability flow (4) with the DPM solver (Lu et al., 2023) which builds over the exponential integrator (Zhang & Chen, 2023)—also making use of the semi-linearity of (3). Optionally, to avoid errors accumulating along the denoising discretisation, each *predictor* denoising step can be followed by a small number of Langevin Monte Carlo (*corrector*) steps. Following Lu et al. (2023); Yim et al. (2023), at the end of the sampling algorithm, we use Tweedie's formula (8) to replace the resulting noisy sample with the posterior mean over the noise free data.

Conditional sampling with guidance. We now describe how to tackle the forecasting or data assimilation tasks by sampling from conditionals $p(x_{1:L}|x_o)$—instead of the prior $p(x_{1:L})$—with the trained local score $\mathbf{s}_\theta$ described in Sec. 3.2. For both of these tasks, the conditioning observation $y = x_o \in \mathbb{R}^O$ is a (noisy) measurement of a subset of variables in the space-time domain, i.e. $x_o = \mathcal{A}(x) = \mathrm{A}\ \mathrm{vec}(x) + \eta$ with $\mathrm{vec}(x) \in \mathbb{R}^N$ the space-time vectorised trajectory, $\eta \sim \mathrm{N}(0, \sigma_y^2 \mathrm{I})$ and a masking matrix $\mathrm{A} = \{0,1\}^{O \times N}$ where rows indicate with 1 which variable is observed. Plugging this in (7) and computing the score we get the following reconstruction guidance term

$$\nabla \log p(x_o|x(t)) \approx \frac{1}{r_t^2 + \sigma_y^2}(y - \mathrm{A}\hat{x}(x(t)))^\top \mathrm{A} \frac{\partial \hat{x}(x(t))}{\partial x(t)}. \tag{11}$$

Summing up this guidance with the trained score over the prior as in (6), we get an approximation of the conditional score $\nabla \log p(x_{1:L}(t)|x_o)$ and can thus generate conditional trajectories by simulating the conditional denoising process.

All-at-once (AAO). We can generate entire trajectories $x_{1:L}$ in one go as illustrated in Fig. 2b. Assuming we are observing some values $x_o \sim \mathrm{N}(\cdot|\mathrm{A}x_{1:L}, \sigma_y^2 \mathrm{I})$, the conditional score $\nabla \log p(x_{1:L}(t)|x_o)$ is obtained by summing the full sequence score $\mathbf{s}_\theta(t, x_{1:L}(t))$ built by combining local scores as developed in Sec. 3.2, with the guidance term $\nabla \log p(x_o|x_{1:L}(t))$ estimated as per (11). This is the approach taken by Rozet & Louppe (2023a).

Autoregressive (AR). Let's first focus on forecasting, i.e. sampling $x_{C+1:L}$ given $x_{1:C}$ initial states. An alternative approach to AAO is to factor the forecasting prediction problem as $p(x_{1:L}|x_{1:C}) = \prod_i p(x_i|x_{i-C:i-1})$ with $C \leq k$. As suggested in Ho et al. (2022b), it then follows that we can sample the full trajectory by iteratively sampling each conditional diffusion process, see illustration in Fig. 2a. More generally, instead of sampling one state at a time, we can autoregressively generate H states, conditioning on the previous C ones, such that $H + C = 2k + 1$ with k being the Markov order. We refer to this sampling approach as GUARD for Guided AutoRegressive diffusion model. Data assimilation can similarly be tackled by further conditioning on additional observations appearing within the predictive horizon H at each AR step of the rollout.

Scalability. Both sampling approaches involve a different number of neural function evaluations (NFEs). Assuming p predictor steps and c corrector steps, AAO sampling requires $(1+c) \times p$ NFEs. In contrast, the AR scheme is more computationally intensive as each autoregressive step also costs $(1+c) \times p$ NFEs, but there are $(1 + \frac{L-(2k+1)}{H})$ AR steps. We stress though that this is assuming the full score network evaluation $\mathbf{s}_{1:L} = \mathbf{s}_\theta(t, x_{1:L}(t))$ to be parallelised. Yet in practice, due to memory constraints, the full score in AAO would have to be sequentially computed. Assuming only one local score evaluation fits in memory, it would require as many NFEs as AR steps. What's more, as discussed in the next paragraph, AAO typically requires more corrector steps. See App. E for a longer discussion on scalability.

Modelling capacity. There are two main reasons why the AR sampling strategy outperforms the AAO approach. First, for the same score network with an input window size of $2k+1$, the AAO model has a Markov order k whilst the AR model has an effective order of $2k$. Indeed, as shown in (10), a process of Markov order k yields a score for which each component depends on $2k+1$ inputs. Yet parameterising a score network taking $2k+1$ inputs, means that the AR model would condition on the previous $2k$ states at each autoregressive step. This twice greater Markov order implies that the AR approach is able to model a strictly larger class of data processes than the AAO sampling. See App. B.1 for further discussion on this.

Second, with AAO sampling the denoising process must ensure the start of the generated sequence agrees with the end of the sequence. This requires information to be propagated between the two ends of the sequence, but at each step of denoising, the score network has a limited receptive field of $2k+1$ limiting information propagation. Langevin corrector steps allow extra information flow, but come with additional cost. In contrast, in AR sampling, there is no limit in information flowing forward due to the nature of the autoregressive scheme.

3.4. Amortised model

Alternatively, instead of learning a score for the joint distribution and conditioning *a posteriori*, one can directly learn an approximation to the *conditional* score. Such a conditional score network $\mathbf{s}_\theta(t, x_{i:i+H}(t)|x_{i-C:i})$ is trained to fit the conditional score given these states $x_{i-C:i}$ by minimising a DSM loss akin to (5).

Practically, conditioning on the previous states $x_{i-C:i}$ is achieved by feeding them to the score network along with the input as separate channels (Voleti et al., 2022). We also add binary masks to inputs indicating whether a particular channel is present. This is needed for the cases where some channels are empty, e.g. when we are generating unconditionally from the beginning. During training, we randomly drop $C' \leq C$ frames from the start of the conditioning sequence, where C' is uniformly sampled for every data point in a batch. Sampling is achieved similarly as in Sec. 3.3 by autoregressively generating H states at a time.

4. Related work

Conditional diffusion models. The development of conditional generation methods for diffusion models is an active area of research, whereby a principled approach is to frame this as the problem of simulating a conditional denoising process which converges to the conditional of interest when fully denoised (Song et al., 2021; 2022; Chung et al., 2022b). One approach is to leverage Bayes's rule, and combine a pretrained score network, with an estimate of the conditional over observation given noised data—referred to as a *guidance* term (Chung et al., 2023; Meng & Kabashima, 2023; Song et al., 2023; 2022; Ho et al., 2022b). Several refinements over the estimation of this guidance have been suggested (Rozet & Louppe, 2023a; Finzi et al., 2023). Another line of work has focused on 'inpainting' tasks—problems where the observations lie in a subspace of the diffused space—and suggested 'freezing' the conditioning data when sampling (Song et al., 2021; Trippe et al., 2023; Lugmayr et al., 2022; Mathieu et al., 2023). Lastly, another popular approach, sometimes referred to as *classifier-free guidance*, is to directly learn the conditional score *amortised* over the observation of interest (Ho et al., 2020; Watson et al., 2023; Torge et al., 2023). This approach requires having access at training time to pairs of data and observation.

Machine learning models for PDEs have been proposed as methods that can learn to approximately solve PDEs given some initial state. One such class of methods assumes a fixed finite-dimensional spatial discretisation (i.e. mesh), typically a regular grid, and, leveraging convolutional neural networks, learn to map initial values of the grid to future states (Gu et al., 2022; Zhu & Zabaras, 2018; Bhatnagar et al., 2019). These are data-driven methods and are trained on trajectories generated by conventional solvers, typically at fine resolution but then downscaled.

Alternatively mesh-free approaches learn infinite-dimensional operators via neural networks, coined *neural operators* (Anandkumar et al., 2019; Kovachki et al., 2023; Lu et al., 2021; Patel et al., 2021; Nelsen & Stuart, 2021; Bhattacharya et al., 2021). These methods are able to map parameters along with an initial state to a set of neural network parameters that can then be used for simulation at a different discretisation. Neural operators typically achieve this by leveraging the Fourier transform and working in the spectral domain (Li et al., 2021a; Guibas et al., 2021; Pathak et al., 2022). Building on this line of work, we could leverage Fourier neural operators to extend our method to work at any resolution. Yet we note that U-net (Ronneberger et al., 2015) based models have been shown to outperform neural operators (Gupta & Brandstetter, 2023).

Diffusion model for PDEs. Recently, several works have leveraged diffusion models for solving PDEs, with a particular focus on fluid dynamics. Amortising the score network on the initial state, Kohl et al. (2023) introduced an autoregressive scheme, whilst Yang & Sommer (2023) suggested to directly predict future states. Recently, Cachay et al. (2023) proposed to unify the denoising time and the physical time to improve scalability. They do so by first learning a time interpolator network, which is then used to train a denoising network by constructing a 'noising' process in-

terpolating between neighbour states. Lippe et al. (2023) built upon neural operators with an iterative denoising refinement, particularly effective to better capture higher frequencies and enabling long rollouts. In contrast to the above-mentioned work, others have tackled data-assimilation and super-resolution tasks. Shu et al. (2023) and Jacobsen et al. (2023) suggested using the underlying PDE to enforce adherence to physical laws. Rozet & Louppe (2023a) decomposed the score of long trajectory into a series of local scores over short segments to be able to work with flexible lengths and to dramatically improve scalability w.r.t. memory use. In this work, we show that such a trained score network can alternatively be sampled autoregressively, which is guaranteed to lead to a higher Markov order, and empirically produce accurate long-range forecasts.

5. Experimental results

Our implementation is built on PyTorch (Paszke et al., 2019) and the codebase will soon be made publicly available. We parameterise the score network s_θ with a modern U-net architecture (Ronneberger et al., 2015; Gupta & Brandstetter, 2023) with residual connections and layer normalization. If not stated differently, we generate conditional samples by simulating the reverse diffusion ODE using the DPM solver (Lu et al., 2023) in 128 evenly spaced discretisation steps. As a final step, we return the posterior mean over the noise free data via Tweedie's formula. Unless specified otherwise, we set the guidance schedule to $r_t^2 = \gamma\sigma_t^2/\mu_t^2$ (Finzi et al., 2023; Rozet & Louppe, 2023a), tune γ via grid-search and set $\sigma_y = 0.01$. More results are presented in the appendix.

5.1. Data

In this experimental investigation, we look into the Burgers' and the Kuramoto-Sivashinsky (KS) equations described below. We focus on these since their dynamics are interesting yet challenging, and have been investigated in prior work (Zhang et al., 2019; Du & Zaki, 2021, e.g.). We refer to App. C for more details on the data generating processes used for training and evaluating models.

Burgers' equation is a second-order nonlinear PDE describing the motion of viscous fluid or gas with speed u:

$$\frac{\partial u}{\partial \tau} + u\frac{\partial u}{\partial z} = \nu\frac{\partial^2 u}{\partial z^2},$$

where $\nu > 0$ is the viscosity. For the data generation we follow the setting of Wang et al. (2021), using $\nu = 0.01$. Varying initial conditions, we generate 1200 ground-truth trajectories by integrating Burgers' equation for 1 second with timestep $\Delta\tau = 0.01$, resulting in 101 timesteps. Then, 800 are used as training set, while the remaining are split evenly into a validation and a test set of 200 trajectories.

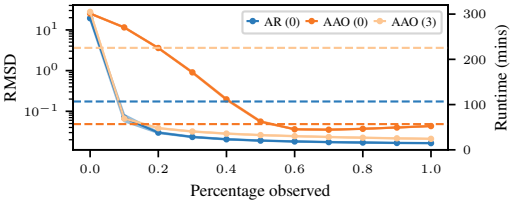


Figure 3. RMSD (↓) computed on KS's dataset when varying the percentage of values observed. The value in bracket indicates the number of correction steps. Means and confidence intervals are estimated over 50 samples. Dashed lines represent the runtime for each method computed on a Titan V with 12GB of memory.

1D Kuramoto-Sivashinsky is a fourth-order nonlinear one dimensional PDE describing flame fronts and solidification dynamics. The position of the flame u evolves as

$$\frac{\partial u}{\partial \tau} + u\frac{\partial u}{\partial z} + \frac{\partial^2 u}{\partial z^2} + \nu\frac{\partial^4 u}{\partial z^4} = 0, \tag{12}$$

where $\nu > 0$ is the viscosity. and is characterised by a chaotic behaviour. We follow Brandstetter et al. (2022, App A.) and Lippe et al. (2023) for the data generation, which solve (12) on a periodic domain with 256 points for the space discretisation and timestep $\Delta\tau = 0.2$. We generate 2048 training trajectory of length $140\Delta\tau$, which corresponds to simulating the system for 28 seconds. For validation and testing we consider 128 trajectories of rollout length $640\Delta\tau$ corresponding to 128 seconds.

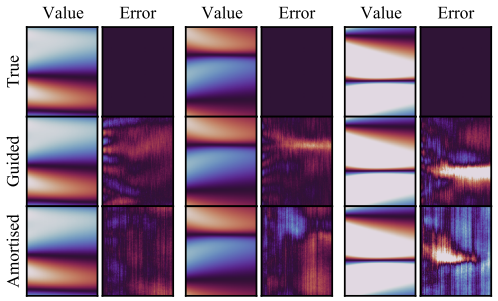
5.2. Guided data assimilation

First, we wish to showcase that guided diffusion models are polyvalent, as they can generate trajectories in a single all-at-once fashion, as well as autoregressively. In particular, we aim to empirically assess the two different sampling regimes AAO vs AR—introduced in Sec. 3.3—when dealing with a wide range of conditioning observation regimes.

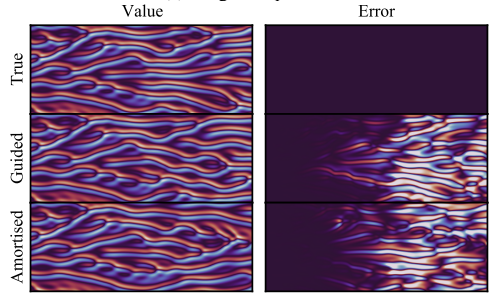
The experimental setting goes as follows. We first choose a number of variables which are observed at each time step—these are sparse in space—then the associated indices are uniformly sampled. We assume to always observe full initial states as for forecasting. We repeat this for different numbers of values observed, varying the *sparsity* of observations, from almost fully observed, to exclusively conditioning on the initial states, i.e. plain forecasting. For this task, we consider a joint diffusion model trained with a window of size 9. The AR model generates 3 states at a time conditioning on the 6 previous ones. We compute the RMSD of the generated trajectories for each sparsity value ranging in $[0, 1]$.

We observe in Fig. 3 that apart from very dense observation regimes, the AR sampling strategy outperforms AAO. We

believe this is due to the effective Markov order being k in AAO vs $2k$ for AR when using a local score network with a window of size $2k+1$, as discussed in Sec. 3.3. We highlight that correction steps are crucial for the AAO sampling strategy to work well, making it computationally very intensive—see App. D.

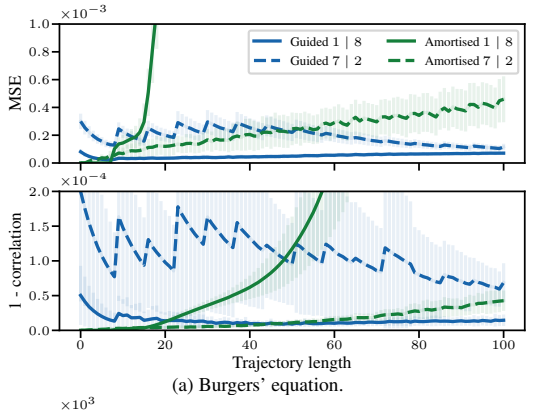


(a) Burger's equation.

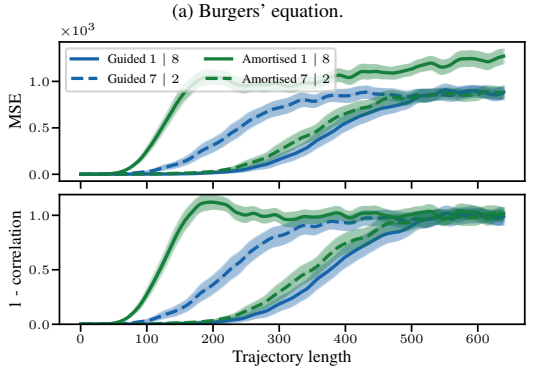


(b) Kuramoto-Sivashinsky equation.

Figure 4. Forecasting rollouts for guided and amortised autoregressive sampling strategies. The first 6 initial states are known.



(a) Burgers' equation.



(b) Kuramoto-Sivashinsky equation.

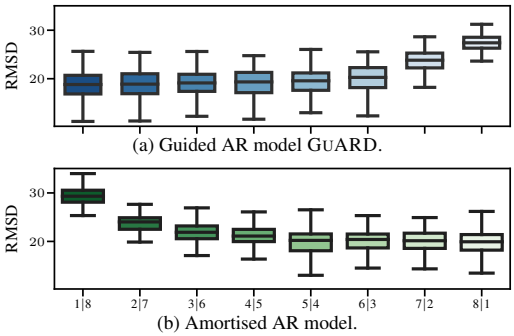
Figure 5. Evolution of MSE and Pearson correlation along trajectories length for guided and amortised models. $H|C$ indicates that H states are generated at each AR step conditioning on C states.

5.3. Autoregressive forecasting

Now we focus on forecasting, where one has access to initial states and aims to predict future states. We investigate the ability of trained diffusion models to sample long rollouts. Since the models introduced in Sec. 3 are built upon a score network trained on short segments from full sequences, it is uncertain whether they would be capable of generating accurate long range trajectories. Here we focus on autoregressive models, with an *amortised* model trained on conditional states, and GUARD, a *guided* model trained on joint states and conditioned a posteriori.

We train a local score network on contiguous segments randomly sampled from training trajectories. We focus on a window size of 9 since larger windows did not improve performance, see App. G.2.3. The models are then evaluated on test trajectories of lengths 101 for Burgers and 640 for the KS dataset, by unrolling models predictions in an *autoregressive fashion*. We use two sets of metrics to measure the accuracy of samples: (a) per time step—with the mean squared error $\texttt{MSE}_{1:L} = \mathbb{E}[(x_{1:L} - \hat{x}_{1:L})^2]$ and Pearson correlation $\rho_{1:L} = \frac{\text{Cov}(x_{1:L}, \hat{x}_{1:L})}{\text{Var}(x_{1:L})\text{Var}(\hat{x}_{1:L})}$ between model samples $\hat{x}_{1:L} \sim p_\theta$ and ground truth trajectories $x_{1:L} \sim p_0$, and (b) for the whole trajectory—via $\texttt{RSMD} = \sqrt{\frac{1}{L}\sum_{l=1}^{L} \texttt{MSE}_l}$ and *high correlation time* $t_{\text{max}} = l_{\text{max}} \times \Delta t$ where $l_{\text{max}} = \arg\max_{l \in 1:L} \{\rho_l > 0.8\}$.

Fig. 4 shows sampled trajectories conditioned on initial states. We see that qualitatively both models are able to generate accurate forecast for Burgers' equation, and similarly up to ∼60s for the KS equation. In Fig. 5 we observe that the best guided model (1 | 8) performs on par with the best amortised model (7 | 2). Moreover, the guided model is not as sensitive to the conditioning length C as the amortised one, as also outlined in App. G.2.3. In the amortised model, any prediction mistake tends to propagate as this would likely lie outside of the distribution that the amortised score



(a) Guided AR model GUARD.

(b) Amortised AR model.

Figure 6. RMSD (↓) on the KS test dataset for different values $H|C$ of horizon H and number of conditioning variables C.

network has been trained over. In contrast, the guided model seems more robust to this phenomenon, as the conditional score contains a contribution from the prior score network trained over segments, which may be able to encourage the states to get closer to a physically realistic regime (i.e. the prior distribution).

Predictive horizon and past states. What is more, the guided model is more flexible since one can choose the predictive horizon and the number of conditioned frames at *sampling* time—with the constraint that these sum to the window size—without the need to train a separate model in contrast to the amortised model. As shown in Fig. 6, this offers a trade-off between sample quality and computational time (as smaller horizon H increases computational time). We also observe from Fig. 6, that the guided model is able leverage longer past trajectory to increase accuracy whilst the amortised model's performance quickly degrades as C increases. See App. G for further ablation studies.

Observation noise and guidance ablation. We further explore different variants of the amortised and guided AR models. Specifically, the reconstruction guidance term

$$\nabla_{x(t)} \log p(y|x(t)) \approx \frac{1}{\gamma r_t^2 + \sigma_y^2} \nabla_{x(t)} \|y - \mathrm{A}\hat{x}(x(t))\|^2,$$

involves a choice of *guidance schedule* $r_t^2 \approx \mathrm{Var}[x(0)|x(t)]$, and a *guidance scaling* γ to be tuned. We explore several heuristics proposed in the literature:

- SDA: $r_t^2 = \frac{\sigma_t^2}{\mu_t^2}$, $\sigma_y = 0.01$ (Rozet & Louppe, 2023a; Finzi et al., 2023)
- ΠGDM: $r_t^2 = \frac{\sigma_t^2}{\mu_t^2 + \sigma_t^2}$, $\sigma_y = 0.01$ (Pokle et al., 2023; Song et al., 2022)
- DPS: $r_t^2 = \|y - A\hat{x}(x(t))\|$, $\sigma_y = 0$ (Chung et al., 2023)
- Video diffusion: $r_t^2 = 1/\mu_t$, $\sigma_y = 0$ (Ho et al., 2022b).

Although we do not know the exact variance, we know that $r_t^2 = \mathrm{Var}[x(0)|x(t)] \xrightarrow[t\to 0]{} 0$, and $\mathrm{Var}[x(0)|x(t)] \xrightarrow[t\to\infty]{} \mathrm{Var}[x(0)] = \sigma_0^2$ and that the variance should be monotonically increasing with t. For SDA and ΠGDM, we have that $r_0 = 0$, whilst $r_0 = 1$ for Video diffusion and $r_t \xrightarrow[t\to 0]{} \infty$ for DPS (assuming the error goes to 0). Also, for SDA and Video diffusion, $r_t \xrightarrow[t\to\infty]{} \infty$ whilst $r_t \xrightarrow[t\to\infty]{} 1$ for ΠGDM. These properties make the ΠGDM and SDA guidance schedules reasonable heuristics to choose, and indeed from Fig. 7 we observe that these perform best.

For the amortised model, we also consider adding Gaussian noise $\eta \sim \mathrm{N}(0, s^2)$ to the conditional frames during training, with a fixed noise $s = 0.01$. On Fig. 7 we see that noising the conditional information hurts the amortised model's performance, although one might think that it could improve robustness w.r.t. error accumulation. Also, App. I shows that the amortised model trained without noise is much more sensitive to noisy initial condition when sampling compared to GUARD and the amortized model trained with $s = 0.01$.

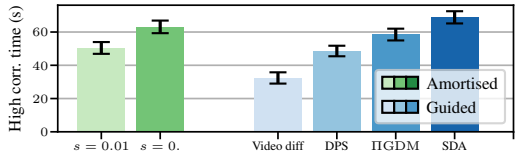


Figure 7. High correlation time on the Kuramoto-Sivashinsky equation. Means and standard errors are estimated over 128 sampled trajectories. The guided models using ΠGDM, Video diff, and DPS were trained on rescaled data, such that dynamic thresholding over the x_0 prediction could be employed to ensure stability; otherwise, the performance was very poor (see App. H).

6. Discussion

In this work, we have explored different ways to train and sample diffusion models for forecasting and data-assimilation tasks. In particular, we have empirically demonstrated the long-range rollout ability of diffusion models trained on short trajectory segments, and autoregressively sampled. We have shown that a diffusion model trained on joint segments and conditioned via reconstruction guidance performs similarly to diffusion models trained conditionally and that therefore were trained for that specific task.

Limitations and future work. Although we have explored different solvers and a number of discretisation steps, the autoregressive sampling strategy requires simulating a denoising process at each step, which is computationally costly. We believe that this can be further improved, perhaps by reusing previous sampled states. Additionally, the guided

sampling strategy requires tuning the guidance strength, yet we suspect there exist some simple heuristics that are able to decrease the sensitivity of this hyperparameter.

Impact Statement

This work tackles the task of generating realistic trajectories of dynamical systems with underlying physics described by partial differential equations (PDEs). PDEs are ubiquitous in Science and Engineering fields, as they are the foundation for describing and solving many practical problems. Fluid dynamics, which studies the flow of liquids and gases, is perhaps the field that relies the most on solving PDEs. Applications can range from profiling turbines with optimal performance when designing wind turbines or hydroelectric dams, to optimising the aerodynamics of airborne objects such as planes or missiles. Our work does not engage with any of these tasks specifically.

References

Anandkumar, A., Azizzadenesheli, K., Bhattacharya, K., Kovachki, N., Li, Z., Liu, B., and Stuart, A. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2019.

Bauer, P., Thorpe, A., and Brunet, G. The quiet revolution of numerical weather prediction. *Nature*, 525(7567), 2015.

Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K., and Kaushik, S. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2), 2019.

Bhattacharya, K., Hosseini, B., Kovachki, N. B., and Stuart, A. M. Model Reduction And Neural Networks For Parametric PDEs. *The SMAI Journal of computational mathematics*, 7:121–157, 2021.

Brandstetter, J., Welling, M., and Worrall, D. E. Lie Point Symmetry Data Augmentation for Neural PDE Solvers. In *International Conference on Machine Learning*. PMLR, 2022.

Burgers, J. M. A mathematical model illustrating the theory of turbulence. *Advances in applied mechanics*, 1:171–199, 1948.

Cachay, S. R., Zhao, B., Joren, H., and Yu, R. DYffusion: a dynamics-informed diffusion model for spatiotemporal forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Cattiaux, P., Conforti, G., Gentil, I., and Léonard, C. Time reversal of diffusion processes under a finite entropy condition, September 2022.

Chung, H., Sim, B., Ryu, D., and Ye, J. C. Improving Diffusion Models for Inverse Problems using Manifold Constraints. *Advances in Neural Information Processing Systems*, 35, 2022a.

Chung, H., Sim, B., and Ye, J. C. Come-Closer-Diffuse-Faster: Accelerating Conditional Diffusion Models for Inverse Problems through Stochastic Contraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022b.

Chung, H., Kim, J., Mccann, M. T., Klasky, M. L., and Ye, J. C. Diffusion Posterior Sampling for General Noisy Inverse Problems. In *The Eleventh International Conference on Learning Representations, ICLR*, 2023.

Cox, S. M. and Matthews, P. C. Exponential time differencing for stiff systems. *Journal of Computational Physics*, 176(2):430–455, 2002.

Didi, K., Vargas, F., Mathis, S. V., Dutordoir, V., Mathieu, E., Komorowska, U. J., and Lio, P. A framework for conditional diffusion modelling with applications in motif scaffolding for protein design, December 2023.

Driscoll, T. A., Hale, N., and Trefethen, L. N. Chebfun guide, 2014.

Du, Y. and Zaki, T. A. Evolutional Deep Neural Network. *Physical Review E*, 104(4), 2021.

Efron, B. Tweedie's formula and selection bias. *Journal of the American Statistical Association*, 106, 2011.

Finzi, M. A., Boral, A., Wilson, A. G., Sha, F., and Zepeda-Nunez, L. User-defined Event Sampling and Uncertainty Quantification in Diffusion Models for Physical Dynamical Systems. In *International Conference on Machine Learning*. PMLR, 2023.

Gu, S., Chen, D., Bao, J., Wen, F., Zhang, B., Chen, D., Yuan, L., and Guo, B. Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Guibas, J., Mardani, M., Li, Z., Tao, A., Anandkumar, A., and Catanzaro, B. Adaptive Fourier Neural Operators: Efficient Token Mixers for Transformers. In *International Conference on Learning Representations*, 2021.

Gupta, J. K. and Brandstetter, J. Towards multi-spatiotemporal-scale generalized PDE modeling, 2023.

Haussmann, U. G. and Pardoux, E. Time reversal of diffusions. *The Annals of Probability*, 1986.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

Ho, J., Chan, W., Saharia, C., Whang, J., Gao, R., Gritsenko, A., Kingma, D. P., Poole, B., Norouzi, M., Fleet, D. J., et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022a.

Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., and Fleet, D. J. Video diffusion models. In *Deep Generative Models for Highly Structured Data Workshop, ICLR*, volume 10, 2022b.

Hyman, J. M. and Nicolaenko, B. The kuramoto-sivashinsky equation: a bridge between pde's and dynamical systems. *Physica D: Nonlinear Phenomena*, 18, 1986.

Hyvärinen, A. and Dayan, P. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

Jacobsen, C., Zhuang, Y., and Duraisamy, K. CoCoGen: Physically-Consistent and Conditioned Score-based Generative Models for Forward and Inverse Problems, December 2023.

Kevrekidis, I. G., Nicolaenko, B., and Scovel, J. C. Back in the saddle again: A computer assisted study of the kuramoto–sivashinsky equation. *SIAM Journal on Applied Mathematics*, 50(3), 1990.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Kohl, G., Chen, L.-W., and Thuerey, N. Turbulent flow simulation using autoregressive conditional diffusion models. *arXiv preprint arXiv:2309.01745*, 2023.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural Operator: Learning Maps Between Function Spaces. *Journal of Machine Learning Research*, 24(89), 2023.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier Neural Operator for Parametric Partial Differential Equations. In *9th International Conference on Learning Representations, ICLR*, 2021a.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021b.

Lippe, P., Veeling, B. S., Perdikaris, P., Turner, R. E., and Brandstetter, J. PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Liu, W. K., Li, S., and Park, H. S. Eighty Years of the Finite Element Method: Birth, Evolution, and Future. *Archives of Computational Methods in Engineering*, 29(6), 2022.

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models, 2023.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3), 2021.

Lugmayr, A., Danelljan, M., Romero, A., Yu, F., Timofte, R., and Van Gool, L. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Mathieu, E., Dutordoir, V., Hutchinson, M. J., De Bortoli, V., Teh, Y. W., and Turner, R. E. Geometric Neural Diffusion Processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Meng, X. and Kabashima, Y. Diffusion Model Based Posterior Sampling for Noisy Linear Inverse Problems, May 2023.

Nelsen, N. H. and Stuart, A. M. The Random Feature Model for Input-Output Maps between Banach Spaces. *SIAM Journal on Scientific Computing*, 43(5), 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. 2019.

Patel, R. G., Trask, N. A., Wood, M. A., and Cyr, E. C. A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373, January 2021.

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., Hassanzadeh, P., Kashinath, K., and Anandkumar, A. FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, February 2022.

Pokle, A., Muckley, M. J., Chen, R. T. Q., and Karrer, B. Training-free Linear Image Inversion via Flows, September 2023.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. 2017.

Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention, 18th International Conference,*. Springer, 2015.

Rozet, F. and Louppe, G. Score-based Data Assimilation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Rozet, F. and Louppe, G. Score-based data assimilation for a two-layer quasi-geostrophic model. *arXiv preprint arXiv:2310.01853*, 2023b.

Shu, D., Li, Z., and Farimani, A. B. A Physics-informed Diffusion Model for High-fidelity Flow Field Reconstruction. *Journal of Computational Physics*, 478, 2023.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2015.

Song, J., Vahdat, A., Mardani, M., and Kautz, J. Pseudoinverse-Guided Diffusion Models for Inverse Problems. In *International Conference on Learning Representations*, 2022.

Song, J., Zhang, Q., Yin, H., Mardani, M., Liu, M.-Y., Kautz, J., Chen, Y., and Vahdat, A. Loss-Guided Diffusion Models for Plug-and-Play Controllable Generation. 2023.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Szudarek, M., Piechna, A., Prusiński, P., and Rudniak, L. Cfd study of high-speed train in crosswinds for large yaw angles with rans-based turbulence models including geko tuning approach. *Energies*, 15(18), 2022.

Torge, J., Harris, C., Mathis, S. V., and Lio, P. Diffhopp: A graph diffusion model for novel drug design via scaffold hopping. *arXiv preprint arXiv:2308.07416*, 2023.

Trippe, B. L., Yim, J., Tischer, D., Broderick, T., Baker, D., Barzilay, R., and Jaakkola, T. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *International Conference on Learning Representations (ICLR)*, 2023.

Vincent, P. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

Voleti, V., Jolicoeur-Martineau, A., and Pal, C. Mcvd: Masked conditional video diffusion for prediction, generation, and interpolation. In *(NeurIPS) Advances in Neural Information Processing Systems*, 2022.

Vorlaender, M., Schröder, D., Pelzer, S., and Wefers, F. Virtual reality for architectural acoustics. *Journal of Building Performance Simulation*, 8, 01 2015.

Wang, S., Wang, H., and Perdikaris, P. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40), 2021.

Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Hanikel, N., Pellock, S. J., Courbet, A., Sheffler, W., Wang, J., Venkatesh, P., Sappington, I., Torres, S. V., Lauko, A., De Bortoli, V., Mathieu, E., Ovchinnikov, S., Barzilay, R., Jaakkola, T. S., DiMaio, F., Baek, M., and Baker, D. De novo design of protein structure and function with RFdiffusion. *Nature*, 2023.

Wu, K. E., Yang, K. K., Berg, R. v. d., Zou, J. Y., Lu, A. X., and Amini, A. P. Protein structure generation via folding diffusion. *arXiv preprint arXiv:2209.15611*, 2022.

Yang, G. and Sommer, S. A denoising diffusion model for fluid field prediction. *arXiv e-prints*, pp. arXiv–2301, 2023.

Yim, J., Campbell, A., Foong, A. Y. K., Gastegger, M., Jiménez-Luna, J., Lewis, S., Satorras, V. G., Veeling, B. S., Barzilay, R., Jaakkola, T., and Noé, F. Fast protein backbone generation with SE(3) flow matching, October 2023.

Zhang, D., Guo, L., and Karniadakis, G. E. Learning in Modal Space: Solving Time-Dependent Stochastic PDEs Using Physics-Informed Neural Networks. *arXiv:1905.01205*, September 2019.

Zhang, Q. and Chen, Y. Fast Sampling of Diffusion Models with Exponential Integrator. In *The Eleventh International Conference on Learning Representations, ICLR*. OpenReview.net, 2023.

Zhu, Y. and Zabaras, N. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366, August 2018.

Supplementary to: Guided Autoregressive Diffusion Models with Applications to PDE Simulation

A. Organisation of the supplementary

In this supplementary, we first motivate in App. B the choice made regarding the (local) score network given the Markov assumption on the sequences. We then describe the data generating process in App. C for the datasets used in the experiments in Sec. 5. After, in App. D, we assess the ability of the all-at-once sampling strategy—which relies on reconstructing the full score from the local ones—to perform accurate forecast, and in particular show the crucial role of correction steps in the discretisation. Then in App. E, we show that the same learnt local score can give rise to two different sampling strategies—all-at-once and autoregressive—and empirically assess their performance for forecasting and data assimilation. In App. F, we look at how the guided autoregressive sampling approach behaves when trading off the prediction horizon with the number of states being conditioned over. Then in App. G, we compare the amortised model with the guided approach in the context of forecasting, and investigate how the window size and the combination between prediction horizon H and number of conditioned frames C affect the performance of each model. In App. H, we empirically explore different guidance schedules and their impact on forecast accuracy. Finally, in App. I we examine how sensitive the amortised and guided models are to noise in the initial observed states.

B. Markov score

In this section we look at the structure of the score induced by a Markov time series. In particular we lay out different ways to parameterise and train this score with a neural network. In what follows we assume an AR-1 Markov model for the sake of clarity, but the reasoning developed trivially generalises to AR-k Markov models. Assuming an AR-1 Markov model, we have that the joint density factorise as

$$p(x_{1:L}) = p(x_1) \prod_{t=2}^{T} p(x_i|x_{i-1}). \tag{13}$$

B.1. Window $2k+1$:

Looking at the score of (13) for intermediary states we have

$$\begin{aligned} \nabla_{x_i} \log p(x_{1:L}) &= \nabla_{x_i} \log p(x_{i+1}|x_i) + \nabla_{x_i} \log p(x_i|x_{i-1}) \qquad (14) \\ &= \nabla_{x_i} \log p(x_{i+1}, x_i|x_{i-1}) \\ &= \nabla_{x_i} \log p(x_{i+1}, x_i|x_{i-1}) + \underbrace{\nabla_{x_i} \log p(x_{i-1})}_{0} \\ &= \nabla_{x_i} \log p(x_{i+1}, x_i, x_{i-1}) \\ &= \nabla_{x_i} \log p(x_{i-1}, x_i, x_{i+1}). \qquad (15) \end{aligned}$$

Similarly, for the first state we get

$$\begin{aligned}
\nabla_{x_1} \log p(x_{1:L}) &= \nabla_{x_1} \log p(x_1) + \nabla_{x_1} \log p(x_2|x_1) \\
&= \nabla_{x_1} \log p(x_1, x_2) \\
&= \nabla_{x_1} \log p(x_1, x_2) + \underbrace{\nabla_{x_1} \log p(x_3|x_2)}_{0} \\
&= \nabla_{x_1} \log p(x_1, x_2) + \nabla_{x_1} \log \underbrace{p(x_3|x_2, x_1)}_{\text{since } = p(x_3|x_2)} \\
&= \nabla_{x_1} \log p(x_1, x_2, x_3), \qquad (16)
\end{aligned}$$

and for the last state

$$\begin{aligned}
\nabla_{x_L} \log p(x_{1:L}) &= \nabla_{x_L} \log p(x_L|x_{L-1}) \qquad (17) \\
&= \nabla_{x_L} \log \underbrace{p(x_L|x_{L-1}, x_{L-2})}_{\text{since } = p(x_L|x_{L-1})} \\
&= \nabla_{x_L} \log p(x_L|x_{L-1}, x_{L-2}) + \underbrace{\nabla_{x_L} \log p(x_{L-1}, x_{L-2})}_{0} \\
&= \nabla_{x_L} \log p(x_L, x_{L-1}, x_{L-2}) \\
&= \nabla_{x_L} \log p(x_{L-2}, x_{L-1}, x_L). \qquad (18)
\end{aligned}$$

Noised sequence Denoising diffusion requires learning the score of *noised* states. The sequence $x_{1:L}$ is noised with the process defined in (1) for an amount of time t: $x_{1:L}(t)|x_{1:L} \sim p_{t|0}$ (2). With $x_{1:L}$ an AR-k Markov model there is no guarantee that $x_{1:L}(t)$ is still k-Markov. Yet, we can assume that $x_{1:L}(t)$ is an AR-k' Markov model where $k' > k$, as in Rozet & Louppe (2023a). Although it may seem in contradiction with the previous statement, in what follows we focus on the AR-1 assumption since derivations and the general reasoning is much easier to follow, yet reiterate that this generalises to the order k.

Local score Eqs. (15), (16) and (18) suggest training a score network to fit $\mathbf{s}_\theta(t, x_{i-1}(t), x_i(t), x_{i+1}(t)) \approx \nabla \log p_t(x_{i-1}(t), x_i(t), x_{i+1}(t))$ by sampling tuples $x_{i-1}, x_i, x_{i+1} \sim p_0$ and noising them.

Full score The first, intermediary and last elements of the score are then respectively given by

- $\nabla_{x_1(t)} \log p_t(x_{1:L}(t)) \approx \mathbf{s}_\theta(t, x_1(t), x_2(t), x_3(t))[1]$
- $\nabla_{x_i(t)} \log p_t(x_{1:L}(t)) \approx \mathbf{s}_\theta(t, x_{i-1}(t), x_i(t), x_{i+1}(t))[2]$
- $\nabla_{x_L(t)} \log p_t(x_{1:L}(t)) \approx \mathbf{s}_\theta(t, x_{L-2}(t), x_{L-1}(t), x_L(t))[3]$

Joint sampling The full score over the sequence $x_{1:L}$ can be reconstructed via the trained local score network taking segments of length 3 as input. This allows for sampling from the joint. All of the above generalise to AR-k models, where the local score network takes a segment of length $2k + 1$ as input.

Guided AR Alternatively for forecasting, having learnt a local score network $\mathbf{s}_\theta(t, x_{i-1}(t), x_i(t), x_{i+1}(t))$, i.e. modelling joints $p(x_{i-1}, x_i, x_{i+1})$, we can condition on x_{i-1}, x_i to sample x_{i+1} leveraging reconstruction guidance—as described in Sec. 3.3. Effectively, this induces an AR-2—instead of AR-1— Markov model, or in general, an AR-$2k$—instead of k—Markov model. As such, this score parameterisation takes more input than strictly necessary to satisfy the k Markov assumption.

Amortised AR Obviously we note that, looking at (14) and (17), one could learn a conditional score network amortised on the previous value such that $\mathbf{s}_\theta(t, x_i(t)|x_{i-1}(0)) \approx \nabla_{x_i(t)} \log p(x_i(t)|x_{i-1}(0))$. This unsurprisingly allows forecasting with autoregressive rollout as in Sec. 3.4. Additionally learning a model over $p(x_1)$ and summing pairwise the scores as in (14) would also allow for joint sampling.

B.2. Window $k+1$:

In App. B.1, we started with an AR-k model, but ended up with learning a local score taking $2k+1$ inputs. Below we derive an alternative score parameterisation. Still assuming an AR-1 model (13), we have that the score for intermediary states can be expressed as

$$\begin{aligned}
\nabla_{x_i} \log p(x_{1:L}) &= \nabla_{x_i} \log p(x_{i+1}|x_i) + \nabla_{x_i} \log p(x_i|x_{i-1}) \\
&= \underbrace{\nabla_{x_i} \log p(x_i|x_{i+1}) - \nabla_{x_i} \log p(x_i)}_{\text{via Bayes' rule}} + \nabla_{x_i} \log p(x_i|x_{i-1}) + \underbrace{\nabla_{x_i} \log p(x_{i-1})}_{0} \\
&= \nabla_{x_i} \log p(x_i|x_{i+1}) + \underbrace{\nabla_{x_i} \log p(x_{i+1})}_{0} - \nabla_{x_i} \log p(x_i) + \nabla_{x_i} \log p(x_i, x_{i-1}) \\
&= \nabla_{x_i} \log p(x_i, x_{i+1}) - \nabla_{x_i} \log p(x_i) + \nabla_{x_i} \log p(x_i, x_{i-1}). \qquad (19)
\end{aligned}$$

Local score From (19), we notice that we could learn a pairwise score network such that $\tilde{\mathbf{s}}_\theta(t, x_{i-1}(t), x_i(t)) \approx \nabla_{x_i(t)} \log p(x_{i-1}(t), x_i(t))$, and $\tilde{\mathbf{s}}_\theta(t, x_i(t)) \approx \nabla_{x_i(t)} \log p(x_i(t))$, trained by sampling tuples $x_{i-1}, x_i, x_{i+1} \sim p_0$ and noising them. Note that this requires 3 calls to the local score network instead of 1 as in App. B.1.

Full score The first, intermediary and last elements of the score are then respectively given by

$$\begin{aligned}
\nabla_{x_1(t)} \log p(x_{1:L}(t)) &= \nabla_{x_1(t)} \log p(x_1(t)) + \nabla_{x_1(t)} \log p(x_2(t)|x_1(t)) \\
&= \nabla_{x_1(t)} \log p(x_1(t), x_2(t)) \\
&\triangleq \tilde{\mathbf{s}}_\theta(t, x_1(t), x_2(t))[1].
\end{aligned}$$

$$\begin{aligned}
\nabla_{x_i(t)} \log p(x_{1:L}(t)) &= \mathbf{s}_\theta(t, x_{i-1}(t), x_i(t), x_{i+1}(t)) \\
&\triangleq \tilde{\mathbf{s}}_\theta(t, x_i(t), x_{i+1}(t))[1] + \tilde{\mathbf{s}}_\theta(t, x_{i-1}(t), x_i(t))[2] - \tilde{\mathbf{s}}_\theta(t, x_i(t)).
\end{aligned}$$

$$\begin{aligned}
\nabla_{x_L(t)} \log p(x_{1:L}(t)) &= \nabla_{x_L(t)} \log p(x_L(t)|x_{L-1}(t)) \\
&= \nabla_{x_L(t)} \log p(x_L(t), x_{L-1}(t)) \\
&= \tilde{\mathbf{s}}_\theta(t, x_{L-1}(t), x_L(t))[2].
\end{aligned}$$

Joint sampling The full score over $x_{1:L}(t)$ could therefore be reconstructed from these, enabling joint sampling.

AR Additionally, autoregressive forecasting could be achieved with guidance by iterating over pairs (x_{i-1}, x_i), thus using a local score of window size $k+1$, in contrast to App. B.1 which requires an inflated window of size $2k+1$.

C. Data generation

C.1. Burgers' equation

Groundtruth trajectories for the Burgers' equation (Burgers, 1948)

$$\frac{\partial u}{\partial \tau} + u \frac{\partial u}{\partial z} = \nu \frac{\partial^2 u}{\partial z^2},$$

are obtained from the open-source dataset made available by Li et al. (2021b), where they consider $(x, \tau) \in (0, 1) \times (0, 1]$. The dataset consists of 1200 trajectories of 101 timesteps, i.e. $\Delta\tau = 0.01$, and where the spatial discretisation used is 128. 800 trajectories were used as training set, 200 as validation set and the remaining 200 as test set. These trajectories were generated following the setup described in (Wang et al., 2021) using the code available in their public repository [1]. Initial

[1] https://github.com/PredictiveIntelligenceLab/Physics-informed-DeepONets/tree/main/Burger/Data

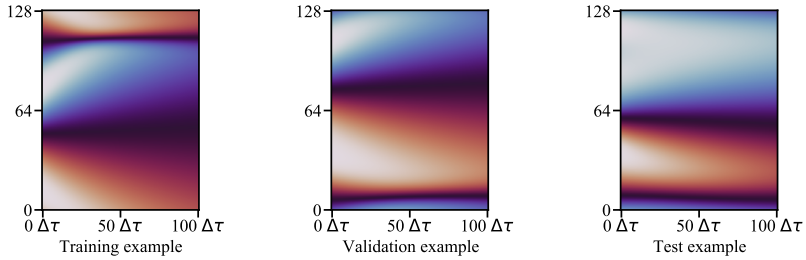


Figure 8. True test trajectories from solving the Burgers' equation following the setup of (Wang et al., 2021). In this case both training, validation, and test trajectories have the same length. We used $\Delta\tau = 0.01s$, so the trajectory contain states from time $\tau = 0s$ to $\tau = 1s$. The spatial dimension is discretized in 128 evenly distributed states

conditions are sampled from a Gaussian random field defined as $\mathrm{N}(0, 625^2(-\Delta + 5^2 I)^{-4})$ (although be aware that in the paper they state to be using $\mathrm{N}(0, 25^2(-\Delta + 5^2 I)^{-4})$, which is different from what they are doing in the code) and then they integrate the Burgers' equation using spectral methods up $\tau = 1$ using a viscosity of 0.01. Specifically, they solve the equations using a spectral Fourier discretisation and a fourth-order stiff time-stepping scheme (ETDRK4) (Cox & Matthews, 2002) with step-size 10^{-4} using the MATLAB Chebfun package (Driscoll et al., 2014). Examples of training, validation, and test trajectories are shown in Fig. 8.

C.2. 1D Kuramoto-Sivashinsky

We generate the groundtruth trajectories for the 1D Kuramoto-Sivashinsky equation

$$\frac{\partial u}{\partial \tau} + u\frac{\partial u}{\partial z} + \frac{\partial^2 u}{\partial z^2} + \nu\frac{\partial^4 u}{\partial z^4} = 0,$$

following the setup of Lippe et al. (2023), which is based on the setup defined by Brandstetter et al. (2022). In contrast to Lippe et al. (2023), we generate the data by keeping Δx and $\Delta\tau$ fixed, i.e. using a constant time and spatial discretisation[2]. The public repository of Brandstetter et al. (2022) can be used to generate our dataset by using the same arguments as the one defined in Lippe et al. (2023, App D1.).

Brandstetter et al. (2022) solves the KS-equation using the method of lines, with spatial derivatives computed using the pseudo-spectral method. Initial conditions are sampled from a distribution defined over truncated Fourier series, i.e. $u_0(x) = \sum_{k=1}^{K} A_k \sin(2\pi l_k x/L + \phi_k)$, where A_k, l_k, ϕ_{kk} are random coefficients representing the amplitude of the different sin waves, the phase shift, and the space dependent frequency. The viscosity parameter is set to $\nu = 1$. Data is initially generated with `float64` precision and then converted to `float32` for training the different models. We generated 1024 trajectories of $140\Delta\tau$, where $\Delta\tau = 0.2s$, as training set, while the validation and test set both contain 128 trajectories of length $640\Delta\tau$. The spatial domain is discretised into 256 evenly spaced points. Trajectories used to train and evaluate the models can be seen in Fig. 9.

D. All-at-once joint sampling for forecasting

One approach for generating long rollouts conditioned on some observations using score-based generative modelling is to directly train a joint posterior score $s_\theta(t, x_{1:L}(t)|y)$ to approximate $\nabla_{x_{1:L}(t)} \log p(x_{1:L}(t)|y)$. The posterior can be composed from a prior score and a likelihood term, just as in the main paper. However, this approach cannot be used for generating long rollouts (large L), as the score network becomes impractically large. Moreover, depending on the dimensionality of each state, with longer rollouts the probability of running into memory issues increases significantly. To deal with this issue, in this section we develop the approach proposed by Rozet & Louppe (2023a), whereby the prior score $(s_\theta(t, x_{1:L}(t)))$ is approximated with a series of local scores $(s_\theta(t, x_{i-k:i+k}(t)))$, which are easier to learn. The size of these local scores is in general significantly smaller than the length of the generated trajectory and will be denoted as the window

[2]This just requires changing lines 166-171 of the `generate_data.py` file in the public repository of Brandstetter et al. (2022).

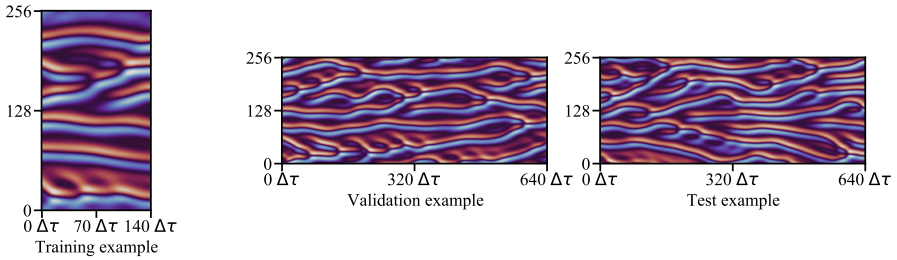


Figure 9. Examples from the Kuramoto-Sivashinsky dataset. Training trajectories are shorter than those used to evaluate the models. Training trajectories has 140 time steps generated every $\Delta\tau = 0.2s$, i.e. training trajectories contains examples from $0s$ to $28s$. Validation and test trajectories, instead, show the evolution of the equation for 640 steps, i.e. from $0s$ to $128s$. The spatial dimension is discretized in 256 evenly distributed states.

of the model W.

At sampling time, the entire trajectory is generated all-at-once, with the ability to condition on initial and/or intermediary states. We study two sampling strategies. The first one is based on the exponential integrator (EI) discretization scheme (Zhang & Chen, 2023), while the second one uses the DPM++ solver (Lu et al., 2023). For each strategy, the predictor steps can be followed by a few corrector steps (Langevin Monte Carlo) (Song et al., 2021; Mathieu et al., 2023). For the results below, we consider the case of conditional generation, where we generate trajectories of 101 states, conditioned on a noisy version of the initial 6 states (with standard deviation $\sigma_y = 0.01$). The results are computed for the Burgers' dataset, based on 30 test trajectories, each with different initial conditions. The sampling procedure uses 128 diffusion steps and a varying number of corrector steps.

Influence of solver and time scheduling We show the results using two solvers: EI (Zhang & Chen, 2023) and DPM++ (Lu et al., 2023). For each, we study two time scheduling settings, which determine the spacing between the time steps during the sampling process

- Linear/uniform spacing: $t_i = t_N - (t_N - t_0)\frac{i}{N}$
- Quadratic: $t_i = (\frac{N-i}{N} t_0^{1/\kappa} + \frac{i}{N} t_N^{1/\kappa})^\kappa$

where N indicates the number of diffusion steps, t_N and t_0 are chosen to be 1 and 10^{-3} and κ is chosen to be 2.

Fig. 10 shows the RMSD between the generated samples and the true trajectories, averaged across the trajectory length. It indicates that for AAO sampling with 0 corrections, there is little difference between the two time schedules, and that DPM++ slightly outperforms EI in terms of RMSD. However, the differences are not significant when taking into account the error bars. As the number of corrections is increased, the difference in performance between the two solvers gets negligible. However, it does seem that with a large number of corrections (25), the quadratic time schedule leads to lower RMSD. Nevertheless, given that, in general, only 3-5 corrector steps are used, we argue that the choice of solver and time schedule has a small impact on overall performance.

Influence of corrector steps and window size Figs. 11 and 12 illustrate the effect of the window size and number of corrector steps on the mean squared error (MSE) and the correlation coefficient (as compared to the ground truth samples). Based on the findings from above, here we used the DPM++ solver with quadratic time discretisation.

The corrector steps seem to be crucial to prevent the error accumulation, with performance generally improving with increasing corrections for both models (window 5 and 9). However, this comes at a significantly higher computational cost, as each correction requires one extra function evaluation. Unsurprisingly, the bigger model (window 9) is better at capturing time dependencies between states, but it comes with increased memory requirements.

Finally, the findings from above are also confirmed in Fig. 13, which illustrates that the predictive performance increases with the window size and the number of corrector steps. Both of these come at an increased computational cost; increasing

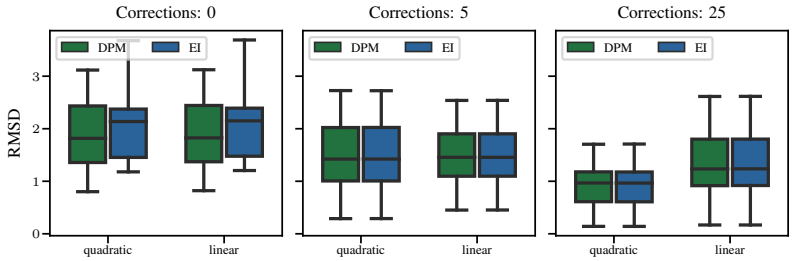


Figure 10. RMSD for Burgers' for the two solvers (DPM++ and EI), and the two time schedules (quadratic and linear) used. The results are shown for 0 (left), 5 (middle), and 25 (right) corrector steps. When using no/only a few corrector steps, the RMSD is not highly impacted by the choice of solver and time schedule. However, at 25 corrector steps, the quadratic time schedule seems to lead to lower RMSD.

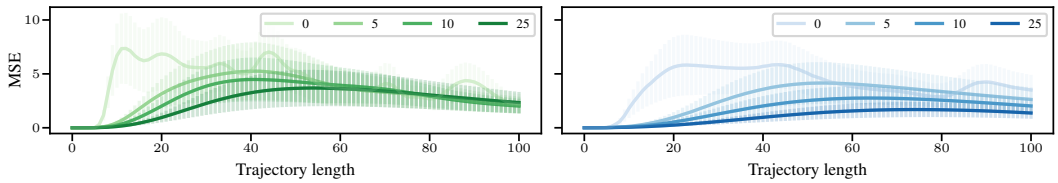


Figure 11. Evolution of the MSE for Burgers' along the trajectory length for a window of 5 (left) and 9 (right). The error bars indicate $\pm$ 3 standard errors. Each line corresponds to a different number of corrector steps. Unsurprisingly, the window 9 model performs better than the window 5 model (when using the same number of corrector steps). Performing corrector steps, alongside predictor steps, is crucial for preventing errors from accumulating. However, even with 25 corrector steps the performance is not very good.

the window size too much might lead to memory issues, while increasing the number of corrections has a negative impact on the sampling time.

E. Comparison between all-at-once and autoregressive sampling

Once the joint model $s_\phi(t, x_{1:L}(t)|y) \approx \nabla_{x_{1:L}(t)} \log p(x_{1:L}(t)|y)$ is trained, there are multiple ways in which it can be queried

- All-at-once (AAO): sampling the entire trajectory in one go (potentially conditioned on some initial and/or intermediary states);
- Autoregressively (AR): sampling a few states at a time, conditioned on a few previously generated states (as well as potentially on some initial and/or intermediary states).

Let us denote the length of the generated trajectory with L, the number of predictor steps used during the sampling process with p, and the number of corrector steps with c. Moreover, in the case of the AR procedure, we have two extra hyperparameters: the predictive horizon H (the number of states generated at each iteration), and the number of conditioned states C. Finally, let the window of the model be W. Each sampling procedure has its advantages and disadvantages.

1. Sampling time: the AAO procedure is faster than the AR one due to its non-recursive nature. We express the computational cost associated with each procedure in terms of the number of function evaluations (NFEs) (i.e. number of calls of the neural network) needed to generate the samples.
   - **AAO**: NFE = (1 + c) $\times$ p
   - **AR**: NFE = (1 + c) $\times$ p $\times (1 + \frac{L-W}{H})$

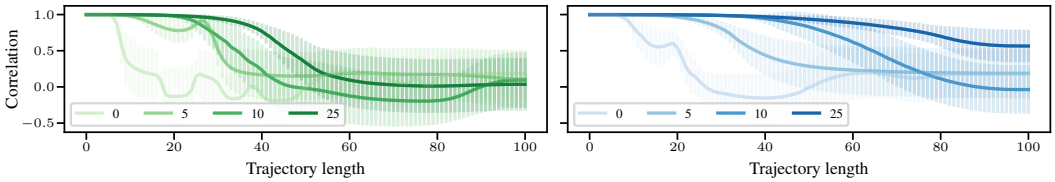


Figure 12. Evolution of the Pearson correlation coefficient for Burgers' along the trajectory length for a window of 5 (left) and 9 (right). The error bars indicate $\pm$ 3 standard errors. The findings are entirely consistent with the ones suggested by the MSE results.

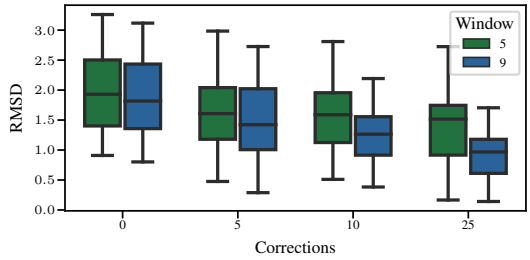


Figure 13. RMSD for Burgers' as a function of corrector steps for the two studied models (window 5 and window 9). Increasing both the window size and the number of corrections positively impacts the predictive performance.

2. Memory cost: the AAO procedure is more memory-intensive than the AR one, as the entire generated trajectory length needs to fit in the memory when batching is not performed. If batching is employed, then the NFEs (and hence, sampling time) increases. In the limit that only one local score evaluation fits in memory, the AAO and AR procedures require the same NFEs.

3. Redundancy of generated states: in the AR procedure, at each iteration the model re-generates the states it conditioned upon, leading to computational redundancy.

4. Quality of generated samples: the AR procedure manages to capture long-term temporal dependencies between states better than the AAO procedure. One of the main reasons likely has to do with the fact that AR sampling induces an AR-$2k$ Markov model, as opposed to a AR-k model in the AAO case (see App. B).

E.1. Guided forecasting

This section aims to illustrate a fair comparison between the AAO and AR sampling methodologies using the Burgers' dataset in the context of forecasting. As mentioned above, the AR method comes at an increased computational cost at sampling time, but generates samples of higher quality and manages to maintain temporal coherency even during long rollouts. However, as illustrated in App. D, the quality of the AAO procedure can be improved by using multiple corrector steps. In particular, to obtain the same NFE for AAO and AR sampling (assuming the same number of predictor steps p and no batching for AAO), we can use $c = \frac{L-W}{H}$ corrector steps.

Window = 5 Fig. 14 shows the results for a trajectory with $L = 101$, using a model with $W = 5$. The results are expressed in terms of the RMSD, calculated based on 30 test trajectories. For AAO sampling $c = 48$, while for AR sampling $c = 0$. In the AR sampling, we generated 2 states at a time conditioned on the 3 previously generated states. In both cases we conditioned on a noisy version of the initial 3 true states (with standard deviation $\sigma_y = 0.01$).

For all experiments, we used the EI discretization scheme, with a quadratic time schedule. The AR procedure is clearly superior to the AAO, even with 48 corrector steps. This can be easily observed by comparing the generated samples to the ground truth as shown in Fig. 15.

Window = 9 For a window of 9, we used the following number of corrector steps for each conditioning scenario:

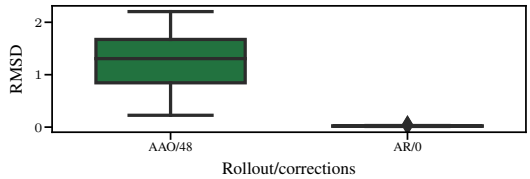


Figure 14. RMSD for Burgers' for AAO and AR generation with a window of 5. The number of corrections for AAO generation was chosen to match the computational budget of the AR sampling technique, yet the AR technique is clearly superior.

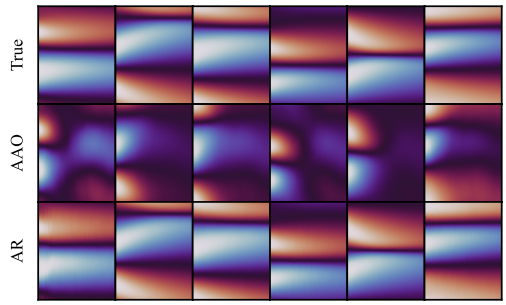


Figure 15. Comparison between the true samples (top), AAO samples (middle), and AR samples (bottom) for Burgers'. The AAO are only able to capture the initial states correctly, while the AR-generated samples maintain very good correlation with the ground truth even at long rollouts (101 time steps).

- $H = 2, C = 7 : c = \frac{101-9}{2} = 46$
- $H = 3, C = 6 : c = \frac{101-9}{3} = 31$
- $H = 4, C = 5 : c = \frac{101-9}{4} = 23$
- $H = 5, C = 4 : c = \frac{101-9}{5} = 19$

Moreover, the number of states we initially condition upon is equal to C (i.e., when comparing AAO with 46 corrections to AR with $H = 2, C = 7$ ($2 \mid 7$), we condition on the first 7 initial noised up states).

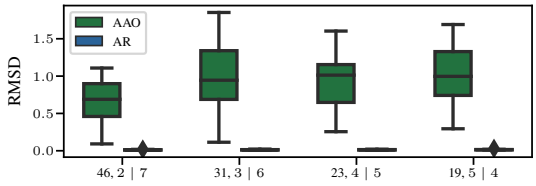


Figure 16. RMSD for Burgers' for AAO and AR generation with a window of 9 and with a varying computational budget. In the AAO case the computational budget is dictated by the number of corrector steps (indicated on the x-axis through the number before the comma). In the AR case, the budget is determined by the number of states generated at each iteration (H). This is also indicated on the x-axis in the format $H \mid C$ (e.g., $2 \mid 7$ implies we generate two states at a time and condition on 7). AR is clearly superior to AAO for forecasting.

Once again, the AR-generated trajectories achieve significantly better metrics, which also reflects in the quality of the generated samples. Fig. 17 shows examples for AAO: 46/AR: $2 \mid 7$.

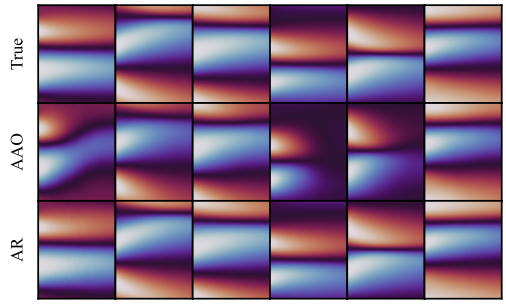


Figure 17. Comparison for Burgers' between the true samples (top), AAO samples with 46 corrections (middle), and AR 2 | 7 samples (bottom). With 46 corrector steps, some AAO trajectories manage to maintain good correlation with some of the easier test trajectories. However, the quality of the AR samples is clearly superior to the AAO samples, managing to model all test trajectories well.

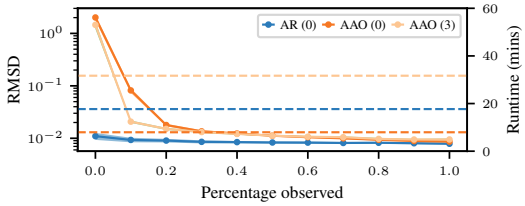


Figure 18. RMSD computed on the Burgers' dataset when varying the percentage of values observed at each physical timestep. The value between brackets indicates the number of correction steps. Means and confidence intervals are estimated over 50 samples. Dashed lines represent the runtime for each method computed on a Titan V with 12GB of memory.

E.2. Guided data assimilation

Performing data assimilation using the AAO sampling procedure simply corresponds to conditioning at generation time on all observed variables regardless of the time-step τ they appear at using reconstruction guidance. This is possible because in AAO all the states of the trajectory are generated simultaneously. In the AR case, instead, we sample a full trajectory in multiple rollout steps. At each step we generate just $H + C$ states, where C refers to the previous states we are conditioning on and H to the number of new states we generate in one iteration. In the forecasting setting, we keep the H newly generated states and discard the C conditioning states. When performing data assimilation, we need to condition on both the previous C states, as well as the observations appearing within the predictive horizon H at each rollout step. If there are no observations in H, then we just perform a forecast step. If instead some observed variables are present in the predictive horizon, we have a likelihood for the conditioning states and one for the true observations within H. In this case, we also keep the resampled C states to ensure more coherency between nearby states. In this section, since we follow the data assimilation set-up from (Rozet & Louppe, 2023b), we have observations at each time step and therefore we never perform any plain forecast rollout in the data assimilation experiments.

Results on Burgers' equation dataset In Sec. 5 we presented the results for the data assimilation task on the KS dataset. A similar behaviour can be also found when performing data assimilation on Burgers'. Burgers' equation, while still being non-linear, is less chaotic than the Kuramoto-Sivashinsky, resulting in an easier task for data assimilation when the setup is the same as the one presented in Sec. 5. This can be noticed in Fig. 18, where the main performance differences between the considered approaches are mostly in the very sparse observation setting (percentage observed of up to 20%). In addition to that, the use of correction steps is not as important as in the KS dataset.

Impact of the guidance scaling in the data assimilation task The performance in the DA task seems to be slightly influenced by the value of the guidance scaling parameter, denoted with γ. To analyse this behaviour, we run the DA

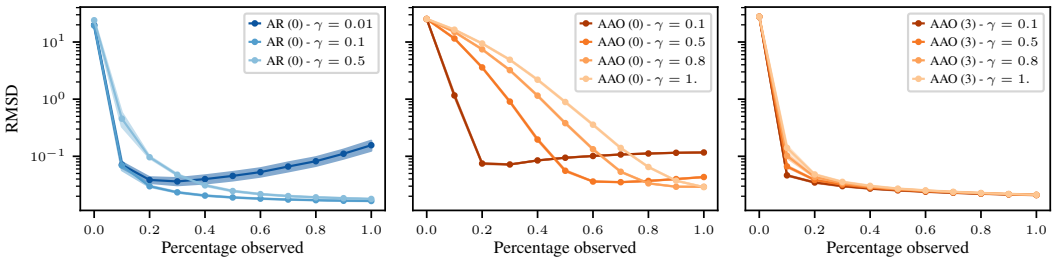


Figure 19. Ablation study on the effect of the guidance scaling parameter γ in the data assimilation task. We report RMSD computed on the KS dataset when varying the percentage of values observed at each physical timestep for several different values of γ. We consider both the AR and AAO (with and without corrections) sampling procedures. Means and confidence intervals are estimated over 50 samples.

experiment on the KS dataset by using different scaling parameters. From Fig. 19 we can see that the value of γ significantly influences the results, with the most significant differences seen when performing AAO sampling with 0 corrections. In this setting, we found that using corrections was crucial for making the AAO approach more robust (i.e. less sensitive to the value of γ).

The AR sampling scheme also seems to be impacted by the choice of γ. In particular, when the percentage of observed values is low (up to 20%), a smaller γ (higher guidance strength) seems to be preferred. In contrast, when a lot of observations are available, if the guidance strength is too high (i.e. γ is too small), the results get significantly worse. Therefore, it is clear that the best approach would be to fine-tune the guidance scaling parameter for each percentage of observed values. This opens up the possibility to investigate heuristics to automatically tune the scaling parameter depending on the number of observed values, but we leave this as future work.

F. Autoregressive guided sampling with varying predictive horizons

As already eluded to in App. E, the autoregressive sampling method offers some flexibility in terms of the number of states generated at each iteration (H). From a computational budget perspective, the higher H, the fewer iterations need to be performed. However, this might come at the cost of a decrease in the quality of the generated samples. In this section, we investigate this trade-off for the Burgers' and KS datasets.

All results are shown for a window of 9, with varying predictive horizons H: 1, 2, 3, 4, 5, 6, 7, and 8. For each setting, the number of previously generated states we condition upon is $C = W - H = 9 - H$: 8, 7, 6, 5, 4, 3, 2, and 1. The trajectories are conditioned upon the first $9 - H$ true states.

The results are presented in terms of the MSE and correlation (Fig. 20), which are computed based on 200 test trajectories for the Burgers dataset, and on 128 test trajectories for the KS dataset. The number of diffusion steps used is 128. For all experiments, we used the DPM++ solver, with a linear time discretisation scheme. At the end, we also performed an additional denoising step, and adopted the dynamic thresholding method for the Burgers' dataset.

Fig. 21 also shows the RMSD for all the different studied settings for Burgers'. The performance is not very sensitive to H for low to medium values of H, although the model seems to perform better with lower H (and consequently, higher number of conditioning frames C). However, the performance quickly deteriorates for very high values of H (i.e., $H = 8$).

Repeating the same experiment for the KS dataset, we see in Fig. 23 two summary metrics: RMSD and the high correlation time, defined as the time until the correlation drops to below 0.8. In general, lower RMSD and larger high correlation times are associated with better performance. The conclusions from the Burgers' dataset are confirmed for KS as well, with the best performing setting being $1 \mid 8$. However, the median RMSD does not decrease drastically even for higher H (up to about $H = 6$).

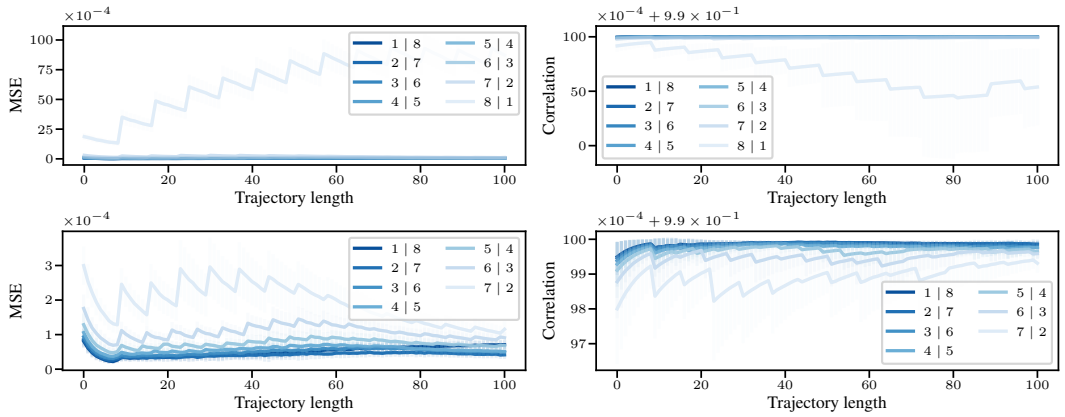


Figure 20. Experimental results on the Burgers' equation. Evolution of the MSE (left) and correlation (right) along the trajectory length. Bottom plots are a zoomed in version of the top ones, where we excluded the $8 \mid 1$ setting. The error bars indicate $\pm$ 3 standard errors. Each line corresponds to a different predictive horizon H. Lower values of H tend to lead to better performance metrics.

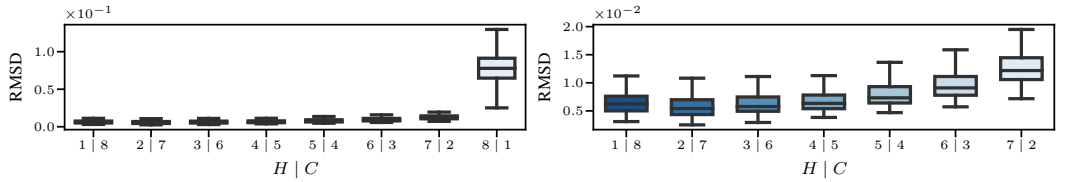


Figure 21. Burgers' dataset. RMSD for different values of H. The right plot is a zoomed in version of the left one, where we excluded the $8 \mid 1$ setting. Outliers have been removed.

G. Comparison between guided and amortised model

In this section we analyse the following two conditioning approaches in the context of forecasting

- by directly learning a conditional model with a score network that is *amortised* over observations y - *amortised model*.
- by learning a diffusion model over the prior, and sampling from the conditional *a posteriori*. Conditioning is introduced through reconstruction guidance, hence we will call this model the *guided model*.

We analyse two datasets: Burgers' and KS. For each dataset, the sampling in the guided model is performed autoregressively. We set the predictive horizon and the number of conditioning frames in such a way as to allow for a fair comparison between the amortised and the guided models.

G.1. Burgers' dataset

In this case we study the following setting:

- Guided model with a window of 5, a prediction horizon H of 2 and 3 conditioned frames. Amortised model, where we generate 2 samples at a time conditioned on the previous 3.

For the guided model, the conditioning from the first iteration uses a noisy version of the first three states (with standard deviation $\sigma_y = 0.01$), while for the amortised model, we use the first three true states. In the sampling process we use 128 diffusion steps. For the discretisation scheme, we study both the EI scheme (Zhang & Chen, 2023), as well as DPM++ solver with order 1 (Lu et al., 2023). The main difference between the two is that the first uses the ϵ prediction model,

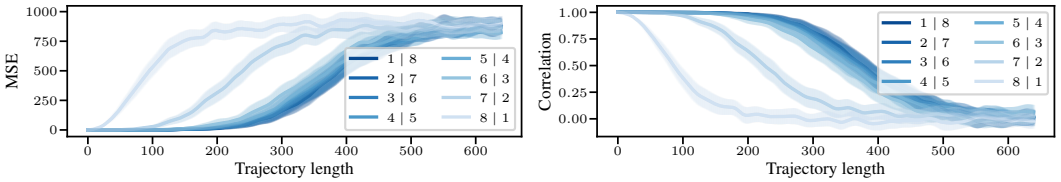


Figure 22. Experimental results on the Kuramoto-Sivashinsky equation. Evolution of the MSE (Left) and correlation (Right) along the generated trajectory length. The error bars indicate $\pm$ 3 standard errors. Each line corresponds to a different predictive horizon H. The same conclusions as in the Burgers' experiment can be drawn.

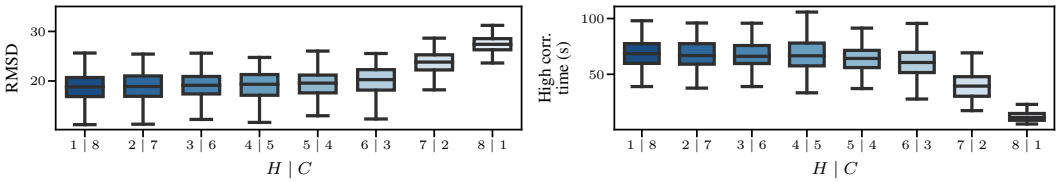


Figure 23. Experimental results on the Kuramoto-Sivashinsky equation. RMSD (left) and high correlation time (right) for different values of H. For RMSD, the smaller, the better, while the reverse is true for the high correlation time. Outliers have been removed.

whereas the DPM++ solver uses the data prediction model. This allows us to also perform dynamic thresholding during the sampling process, which was shown to stabilise sampling with large guidance scales in the case of images (Ho et al., 2022a). Moreover, we can optionally take a final denoising step, specified through the parameter `denoise_to_zero`. Finally, we study the two time discretisation schemes introduced in App. D: quadratic and linear. The results are determined based on 200 test trajectories.

- **Final denoising step**: This was only studied for the DPM++ solver. The performance is clearly improved by using an extra denoising step at the end, as shown in Fig. 24. This is true for all models. In the amortised case, using the final denoising step helps avoid the jump in MSE around the initial states.
- **Solver and time discretisation**: The choice of solver and time discretisation schedule does not affect the results drastically, but it does lead to some differences. In the Burgers' dataset, the best setting seems to be to use the DPM++ solver with the linear time discretisation schedule when using 128 diffusion steps. However, this might depend on the dataset and on the number of diffusion steps used.
- **Guided vs. amortised**: Fig. 24 clearly shows the that guided model outperforms the amortised one both in terms of MSE (lower MSE), as well as correlation (higher correlation). The amortised model is better at modelling the initial states, as it does not contain any observation noise (unlike the guided one). However, errors in modelling initial states propagate and make the amortised model diverge faster than the guided one. Thus, the amortised model suffers from the main drawbacks of autoregressive models - because the conditioning is done through previously generated states, errors propagate rapidly and lead to divergent states. This is true in the guided model as well, but to a lesser extent. Moreover, the guided model has two components: the prior joint score and the conditional information from reconstruction guidance. It is possible that the influence from the prior model also helps to avoid divergence.

A summary of these findings can be found in Fig. 25, where we can easily see the positive impact of the final denoising step. Moreover, Fig. 25 shows that the guided model outperforms or performs on par with the amortised model in terms of RMSD. However, note that we only looked at the setting $2 \mid 3$ in this analysis and the findings might not hold for other settings.

G.2. KS dataset

In this case we study the following settings:

- Guided model with a window of 5, a prediction horizon H of 2 and 3 conditioned frames. Amortised model, where we generate 2 samples at a time conditioned on the previous 3.

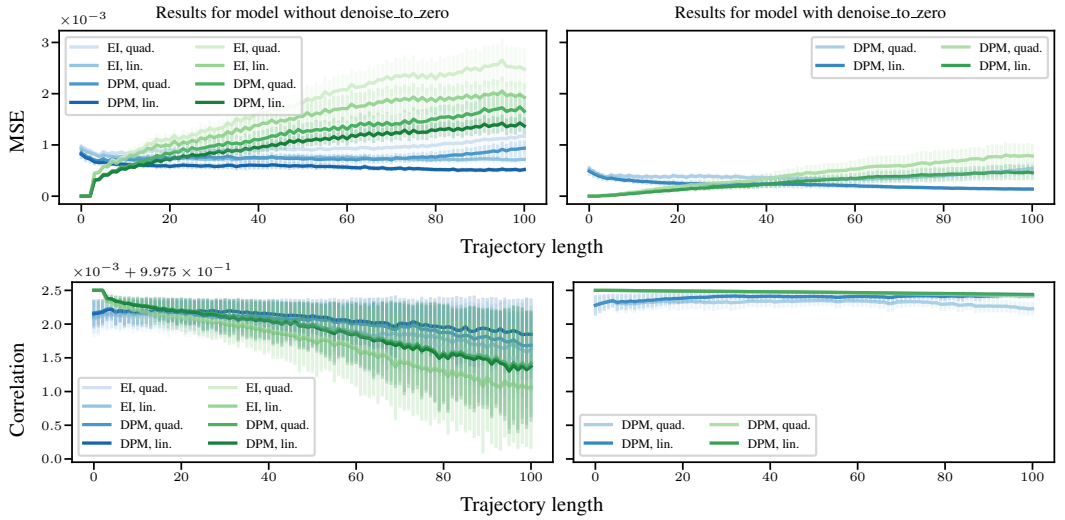


Figure 24. Evolution of the MSE (top) and Pearson correlation (bottom) along the trajectory length for the guided (blue) and amortised (green) models. In each model, we generate 2 states at a time conditioned on the previous 3. For each model, we study the EI and DPM++ solvers, alongside the quadratic and linear time discretisation. The left plot shows the results when we do not employ a final denoising step, while the right plot also performs one final denoising step. The final denoising step clearly improves performance in this setting, allowing the amortised model to avoid the jump in MSE around the first states. In the guided model, the initial error is dominated by the observation noise.

- Guided model with a window of 9, a prediction horizon H of 3 and 6 conditioned frames. Amortised model, where we generate 3 samples at a time conditioned on the previous 6.

- Varying window sizes (H and C) for guided and amortized models.

For the KS dataset we did not use dynamic thresholding, but still used the final denoising step. We also just investigated the DPM++ solver with either the quadratic or linear time schedule. The results are derived on 128 test trajectories, for a trajectory length of 640 time steps, corresponding to 128s. As a summary metric, besides the RMSD, we also compute the high correlation time, defined as the time until the correlation drops to below 0.8 (the higher, the better).

G.2.1. Generate 2 conditioned on 3

Fig. 26 shows that the guided model performs better than the amortised one on average, although the metrics of both models are within error bars. However, we only analysed the setting $2 \mid 3$ and the findings might not hold for other settings. In particular, we observed that the guided model tends to perform better with more conditioning information, whereas increasing C harms the performance of the amortised model. The summary metrics over the entire trajectory are shown in Fig. 27. The plots show that the guided model has better median metrics, but the two models are comparable when taking into account the error bars.

G.2.2. Generate 3 conditioned on 6

Similar conclusions as in the previous section can be drawn if we generate 3 states at a time conditioned on the previous 6. Fig. 29 suggests that the difference between the two models is more pronounced in this setting, with the guided model outperforming the amortised one.

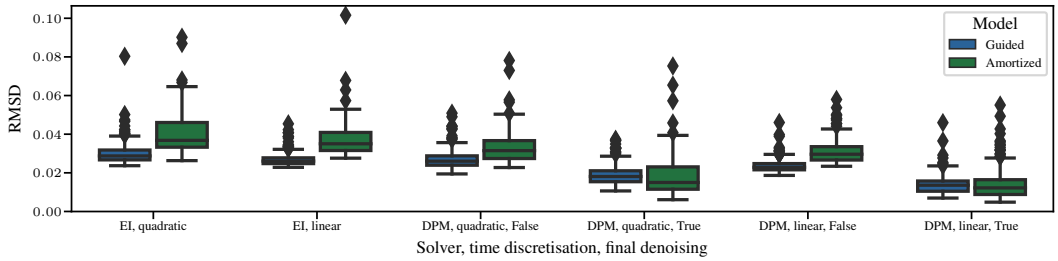


Figure 25. Comparison for Burgers' of RMSD for guided vs amortised model with different solvers and time schedules. For the DPM solver, we also show the results with and without taking a final denoising step. In this case we generate 2 states at a time, conditioned on the previous 3.

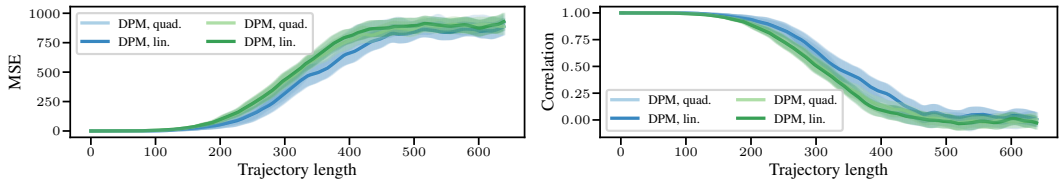


Figure 26. Evolution of the MSE (left) and Pearson correlation (right) along the trajectory length for the guided (blue) and amortised (green) model for the KS dataset. In this case, we generate 2 states conditioned on 3 (with a window of 5 for the guided model). Two different time schedules are investigated: quadratic and linear.

G.2.3. Varying window size

Fig. 30 shows the performance of the amortised model for different $H \in \{1, 2, 4, 6, 8, 12\}$ and $C \in \{1, 2, 4, 8\}$. Regardless of C, the model underperforms for small $H = 1$, with high correlation time of less than 50s. We hypothesise that the reason for this might be the combination of the following: i) generating larger number of frames at the same time requires less steps for the same trajectory length, so there is less error accumulation; and ii) for larger H the model is trained with a better learning task compared to $H = 1$, as larger H provides more information to the model. For $H \geq 4$, the model performs similarly for any H, being slightly more sensitive to larger C when H is smaller. Regardless of H, the performance of the model deteriorates significantly with larger $C = 8$, similar to the behaviour shown in Lippe et al. (2023).

In contrast, the performance of the guided model is not that sensitive to the window size, as shown in Fig. 31. The best performance is observed for medium window sizes ($W = 9$); if the window is too small, the effective Markov order assumed might be too low. As the window increases, so does the effective order, but there is probably a limit to how much increasing the effective Markov order can help performance. While the noised up states within the blanket might have infinite Markov order, the correlation between them likely shows a sharp decrease, so there is little gain between using a window of 9 or a window of 11. Moreover, increasing the window size might make the optimisation harder, explaining why we don't see improvement in performance beyond $W = 9$.

Conclusion on guided vs. amortised model The investigation showed that the guided and amortised model are governed by different dynamics - the guided model generally seems to benefit from more conditioning information (high C) and from generating only a few states at a time (low H). In contrast, the amortised model performs best when generating a larger number of frames at a time (high H) and conditioning on fewer frames (low C). This means that the best $H \mid C$ setting for each model will likely be different. However, the guided model has the benefit that it can be queried in any setting, without any further training. In contrast, to discover the best setting in the amortised case, a model needs to be trained for each $H \mid C$ combination.

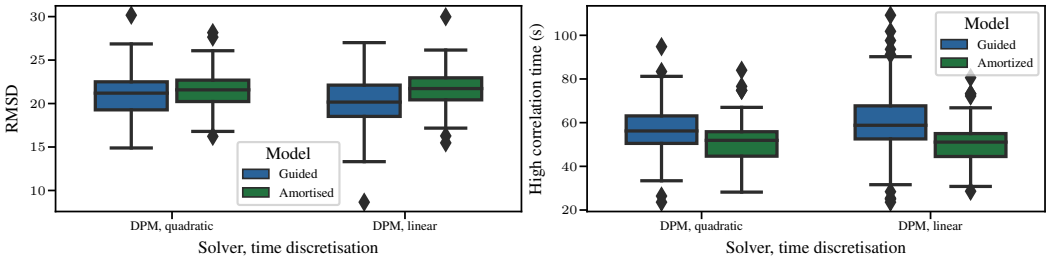


Figure 27. RMSD (left) and high correlation time (right) for the guided and amortised model for the KS dataset. In this case, we generate 2 states conditioned on 3 (with a window of 5 for the guided model). For RMSD, the lower the value, the better the model, while for the high correlation time a larger value is preferred.

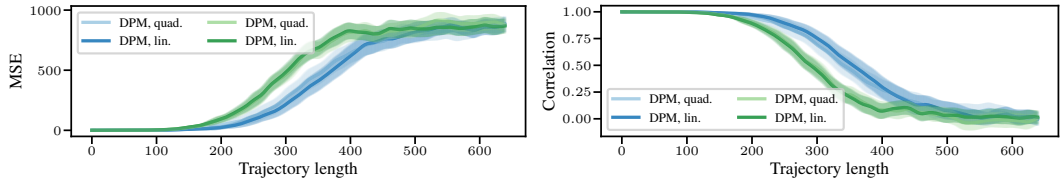


Figure 28. Evolution of the MSE (left) and Pearson correlation (right) along the trajectory length for the guided (blue) and amortised (green) model. In this case, we generate 3 states conditioned on 6 (with a window of 9 for the guided model).

H. Guidance schedule

Sec. 5.3 introduces the four types of guidance studied in this paper. This section provides a comparison between these guidance formulations, applied on the Burgers' and KS datasets in the context of forecasting. The results are derived using AR sampling, using models with a window of 9, with $H = 1$ and $C = 8$ $(1 \mid 8)$. Additionaly, we also condition on the first initial 8 states. In all experiments from this section 128 diffusion steps were employed, and the solver used was DPM++ with a linear time discretisation schedule. At the end, one final denoising step was taken. For both datasets, we also studied the influence of dynamic thresholding, focusing on the stability of the methods at high guidance strenghts.

H.1. Burgers'

We analysed several settings of the γ hyperparameter: [0.01, 0.03, 0.05, 0.1, 0.5, 1] for SDA, [0.001, 0.01, 0.03, 0.1, 1] for DPS, [0.0002, 0.001, 0.01, 0.2, 1.0, 10.0] for VideoDiff, and [0.02, 0.1, 0.2, 2.0, 10.0, 100.0] for ΠGDM. Note that we have studied different γ ranges - for each guidance mechanism the object γ is multiplied by has different scales, so the most appropriate γ values will vary from one mechanism to another.

The results in Fig. 32 correspond to the best setting for each guidance type, indicating that in terms of RMSD, SDA has the best performance, followed by ΠGDM, DPS, and then Video diffusion.

Stability of each guidance type Fig. 33 shows the RMSD for each guidance type when applied to the Burgers' data, both with and without using dynamic thresholding.

By comparing the four plots we can see that

- The lowest RMSD is achieved when using dynamic thresholding for all four types of guidance. However, SDA is the only guidance mechanism that is able to achieve similar performance (within error bounds) without the use of dynamic thresholding. For all the other three guidance types, not using dynamic thresholding resulted in a significant increase in RMSD for the best γ setting.
- The extent of the increase in RMSD resulting from not employing dynamic thresholding (i.e., lowest RMSD achieved with vs without dynamic thresholding) varies across guidance mechanisms

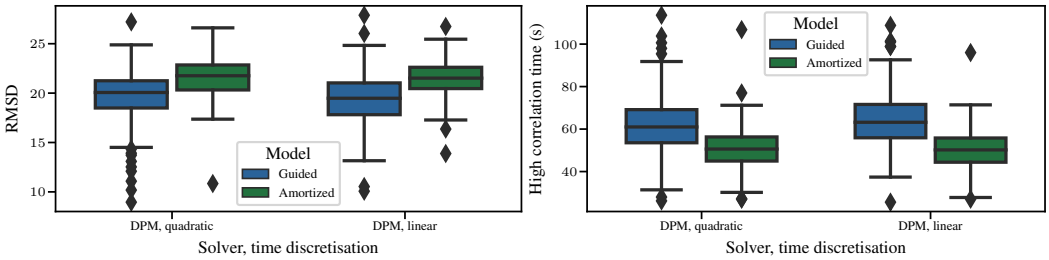


Figure 29. RMSD (left) and high correlation time (right) for the guided and amortised model when generating 3 states and conditioning on 6 for KS. The difference between the guided and the amortised model is more pronounced than in the case when we generate 2 states conditioned on 3 (see Fig. 27).

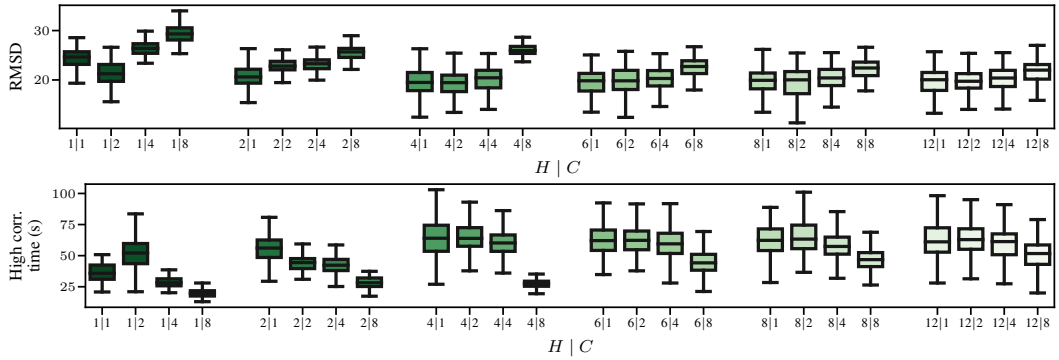


Figure 30. RMSD (top) and high correlation time (bottom) for the amortised model with varying H and C. The model performs the best when generating relatively large blocks ($H \geq 4$) and conditioning on the smaller number of frames ($C \leq 4$).

  - SDA: negligible increase;
  - DPS: significant, but relatively small increase; Averaged across all test samples, the mean RMSD increased more than five times (from 0.8×10^{-2} to 4.5×10^{-2});
  - Video diffusion: big increase in RMSD, with the average across the 200 test samples increasing by almost two orders of magnitude;
  - ΠGDM: big increase in RMSD, with the average across the 200 test samples increasing by more than two orders of magnitude.

- For the investigated guidance strengths, DPS is the only guidance mechanism that did not lead to numerical overflow. For high guidance strengths the results were highly inaccurate, but not unstable. We hypothesise this is due to the $\|y - \mathrm{A}\hat{x}(x(t))\|$ term that probably promotes stability.

These findings suggest that while the SDA guidance leads to similar results regardless of whether other tricks are employed (such as dynamic thresholding), the performance of other guidance types is more sensitive to them. In particular, dynamic thresholding was introduced to stabilise sampling at high guidance strengths (low γ), and it seems to be crucial for all guidance types but SDA.

Fig. 34 also shows some example test trajectories and the corresponding sampled trajectories for the best setting of each guidance type. For each guidance mechanism, we also show the absolute error between the test samples and the generated trajectories.

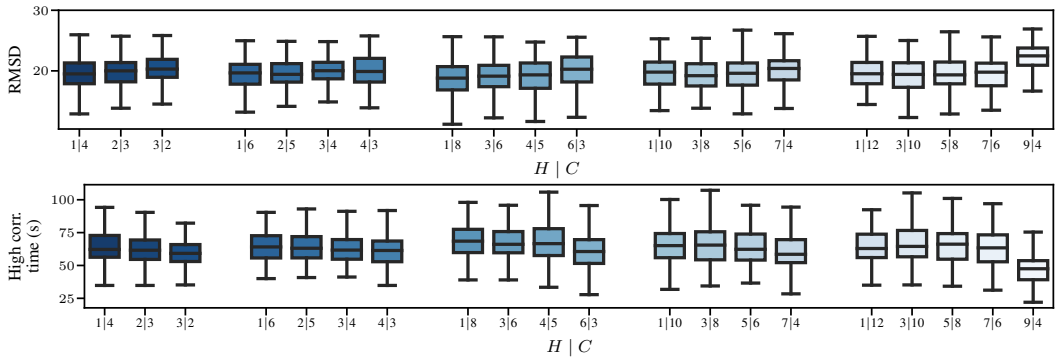


Figure 31. RMSD (top) and high correlation time (bottom) for the guided model with varying window sizes (5, 7, 9, 11, 13) and different $H \mid C$ combinations. The model is not highly sensitive to window size, with best median performance seen for medium window size ($W = 9$) in the setting $1 \mid 8$.

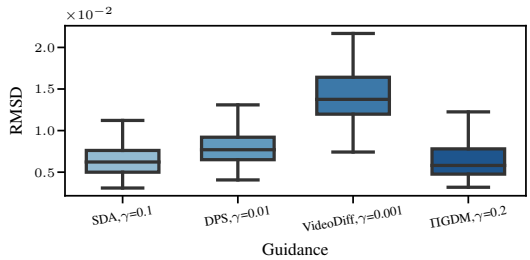


Figure 32. RMSD for the Burgers' dataset using different types of guidance (and the best corresponding γ value). Outliers have been removed. SDA and ΠGDM show the best results, with a similar performance in terms of RMSD, followed by DPS and then Video diffusion.

H.2. KS

H.2.1. Model using original data

The models used in Sec. 5 were trained on the original KS data (i.e., no pre-processing was applied). Given that the training data values lie in the range [-3.61, 3.56], dynamic thresholding could not be applied for this model. One requirement for dynamic threhsolding to work is for the data values to lie within [-1, 1]. Thus, we investigate the performance of each guidance type without using dynamic thresholding.

Fig. 35 shows that SDA has a significantly better performance as compared to all other guidance types. This is in line with the findings from the Burgers' dataset, where we found dynamic thresholding to be crucial for DPS, Video diffusion, and ΠGDM to achieve a similar performance to SDA.

H.2.2. Model using transformed data

To confirm the findings from the Burgers' dataset, we also investigated the four guidance types with and without dynamic thresholding on a model trained on transformed KS data; we modified the data range to [-1, 1], such that dynamic thresholding could be employed. The following γ ranges were studied: [0.03, 0.05, 0.1, 0.5, 1.0] for SDA, [0.001, 0.005, 0.01, 0.05, 0.1, 1.0] for DPS, [0.0003, 0.001, 0.002, 0.01, 0.1, 0.5] for VideoDiff, and [0.02, 0.1, 0.2, 2.0, 10.0, 100.0] for ΠGDM.

Fig. 36 confirms that dynamic thresholding is crucial for stable sampling at high guidance strengths (low γ) for DPS, Video diffusion, and ΠGDM, allowing them to reach a performance similar to SDA. In contrast, SDA is not that sensitive to dynamic thresholding - the RMSDs with and without dynamic thresholding are not significantly different to one another in

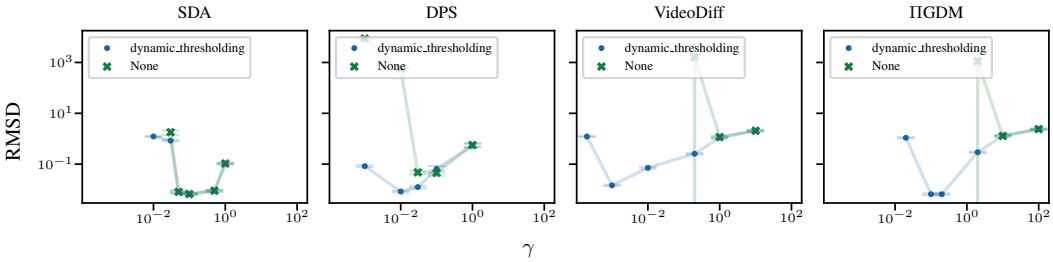


Figure 33. RMSD for the four types of guidance using different values of the γ hyperparameter for Burgers'. Note that a lower γ implies a higher guidance strength. Also note that the scale of the x- and y-axes is logarithmic. For each γ, if the plot does not show a result for the case in which dynamic thresholding was not employed (green X), this is because the results were unstable.

the best setting ($\gamma = 0.1$).

Moreover, we found that the best setting of γ was consistent between datasets for SDA and DPS. ΠGDM showed a similar behaviour - for both datasets, the results for $\gamma = 0.1$ and $\gamma = 0.2$ were among the best two, and not significantly different (i.e., within error bounds). Video diffusion was the only guidance type where the best γ setting was dataset-dependent.

The same conclusion can be drawn from Fig. 37, where we show that employing dynamic thresholding does not improve the RMSD significantly for SDA, but it has a crucial effect for the other three guidance mechanisms.

Finally, one test trajectory for the best setting of each guidance mechanism is shown in Fig. 38.

I. Sensitivity to noise in observations

In this section we study the robustness of the guided AR and amortised AR models to noise in the initial observations in the context of forecasting. More specifically, for each model we analyse its forecasting performance (in terms of high correlation time) in the following two settings

1. Condition on the true initial observations.
2. Condition on noised initial observations, with noise with standard deviation $\sigma_y = 0.01$.

The models we analyse are the same as the ones in Sec. 5.3. For the amortised case, we study both a model that was trained on noiseless conditional frames ($s = 0$), as well as one that was trained on noisy frames with fixed noise $s = 0.01$, with the hypothesis being that the latter is more robust to noise in the initial observations.

1. Guided AR model, generating 1 state at a time and conditioning on 8 (1|8). We also condition on the initial 8 true or noised up states.
2. Amortised AR models, generating 7 states at a time and conditioning on 2 (7|2). We also condition on the initial 2 true or noised up states.
   - Trained on noiseless conditional frames $s = 0$
   - Trained on noisy conditional frames $s = 0.01$

We used the DPM++ solver with a linear time discretisation schedule for all models from this section. We performed 128 diffusion steps with no Langevin corrector steps, and performed a final denoising step, as discussed in Sec. 5.

The results on the KS dataset are shown in Fig. 39. We can see that the guided model (using the SDA guiding mechanism) is very robust to noise in the initial observations, showing only a slight decrease in high correlation time when we condition on noisy initial states, as opposed to the true ones (less than 2s). In comparison, the performance of the amortised model that was trained on noiseless conditional frames ($s = 0$) suffers more, with a decrease in high correlation time of more than 7s

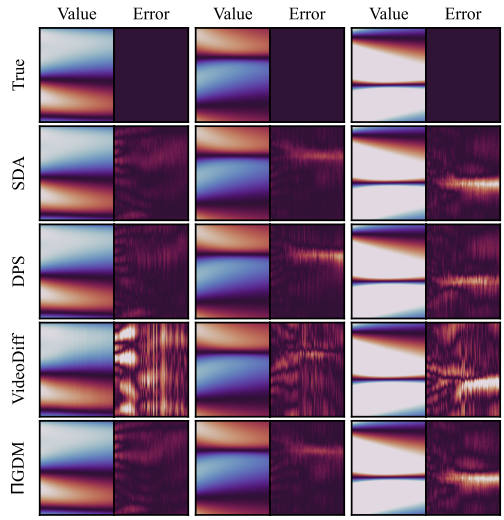


Figure 34. True test trajectories (top), followed by trajectories generated using SDA, DPS, Video diffusion, and ΠGDM for the Burgers' dataset. The results are shown for three example test trajectories. For each guidance mechanism, near the generated samples, we also show the absolute error between them and the ground truth. The scale is the same between the error images.

(see right-most green bars in Fig. 39). We can also equip the amortised model with a similar robustness to noise in the initial states by training it on noisy conditioned frames (see left-most green bars in Fig. 39). This strategy clearly leads to less sensitivity to the amount of noise present in the initial observations, with a decrease in high correlation time of less than 2s. However, although more robust to noise, it is significantly inferior to the original amortised model (trained on noiseless conditioned frames), managing to achieve a high correlation time of only about 50s.

Thus, it seems that the guided model is inherently less sensitive to the amount of noise present in the initial states. While there are strategies to make the amortised model similarly robust to noise, they significantly harm performance.

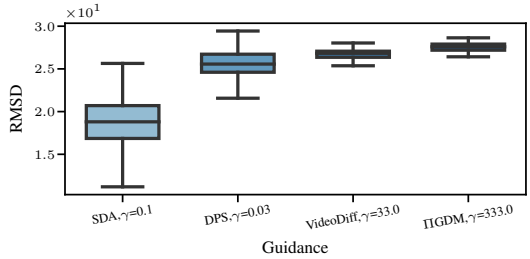


Figure 35. RMSD for the KS dataset using different types of guidance (and the best corresponding γ value). Outliers have been removed. SDA has the best performance by far, with the other guidance types achieving much higher RMSD when dynamic thresholding is not employed.

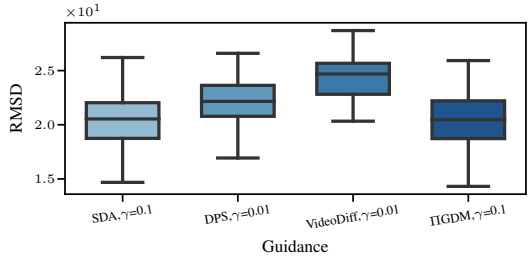


Figure 36. RMSD for the transformed KS dataset using different types of guidance (and the best corresponding γ value). Outliers have been removed. SDA and ΠGDM have a similar performance, followed by DPS and Video diffusion.

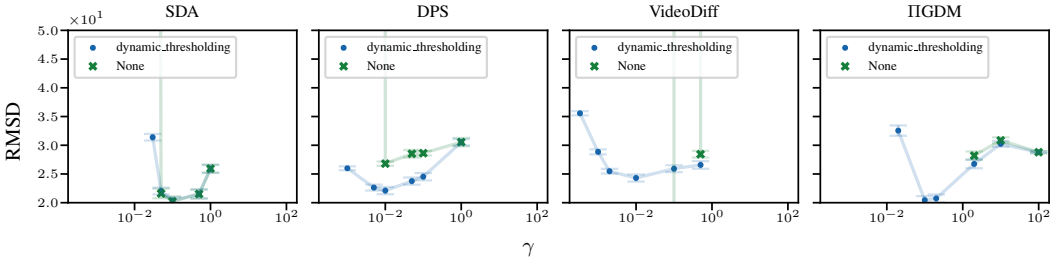


Figure 37. RMSD for the four types of guidance using different values of the γ hyperparameter for the transformed KS dataset. Note that a lower γ implies a higher guidance strength. Only values between 20-50 are shown on the y-axis for clarity. For each γ, if the plot does not show a result for the case in which dynamic thresholding was not employed (green X), this is because the results were unstable.

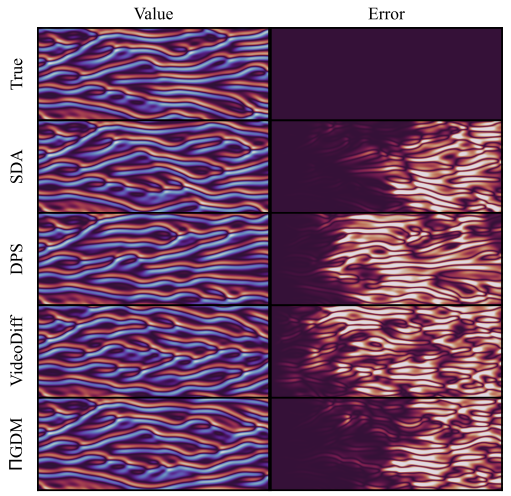


Figure 38. True test trajectory (top), followed by trajectories generated using SDA, DPS, Video diffusion, and ΠGDM for the KS dataset. For each guidance mechanism, near the generated samples, we also show the absolute error between them and the ground truth. The scale is the same between the error images.

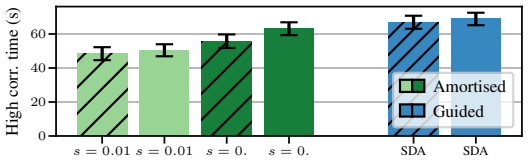


Figure 39. High correlation time for the KS dataset for the amortised (green) and guided (blue) models. The hatched bars indicate that the initial observations we conditioned upon were noisy (with $\sigma_y = 0.01$), whereas the plain ones indicate that we conditioned on the true initial states. For the amortised models, the left-most two bars correspond to the model trained with noise ($s = 0.01$), whereas the other two bars correspond to the model trained without noise ($s = 0$).

CHAPTER 7

Final Remarks

This dissertation concludes a long period of research, covering four distinct research questions and presenting our contributions to address them. In the introduction, we framed each considered problem with a different research question. In the following, we will recap them and highlight the way we addressed them, the main takeaways, and the potential future directions that stem from our works.

Research Question 1: *How can we make out-of-distribution detection using deep-generative models more robust?*

For Research Question 1, in Chapter 3, we borrowed ideas from classical statistical tests and proposed to combine two different statistical tests using Fisher's method. In the paper, we revisit some proposed detection scores and highlight their alternative formulation as a classical significance test. We ended up combining Rao's score test, which is a classical parametric test, and the typicality test, a test that was introduced recently for out-of-distribution detection. We have shown how these two tests are complementary and that they led to a robust score for anomaly detection on different datasets and on a range of different deep generative models. Moreover, we highlighted the benefit of framing the out-distribution problem as multiple-hypothesis testing as it gives us access to well-defined procedures for false discovery rate control, such as the Benjamini- Hochberg (BH) procedure. In the paper, we have shown that it is effective when applied on top of our proposed procedure.

Although we showed that the method was effective, we tested it only on classic benchmarks generally considered in the literature. It would be interesting to study how effective all these anomaly detection procedures are once applied to real-world datasets. Moreover, most of the research focuses on considering new statistics that work for anomaly detection, but there is still no answer to why deep generative models might assign a higher likelihood to OOD data in the first place.

Research Question 2: *Can we define a flexible approximate posterior that adapts to the nonlinear structure of the true posterior while sampling remains efficient?*

For Research Question 2, we proposed to borrow tools from Riemannian geometry to address the problem of the classic Laplace approximation spreading mass in the region of the true posterior that has a low probability. Our proposed solution is to equip the parameter space with a simple Riemannian metric that contains infor-

mation about the loss landscape and to interpret all the samples from the classic Laplace approximation as initial velocities starting from the MAP estimate. We can then compute exponential maps, i.e. geodesic curves subject to an initial point, that is, the θ_{MAP}, and an initial velocity. The end-points of these curves become our new posterior samples. We have shown that this leads to samples that stay within the high posterior region, or equivalently that geodesics tend to travel within low-loss regions, which results in better predictive confidence in examples where classic Laplace underfits.

Despite the success of our proposed method, there is still a lot of promising research to be done in these areas. The main bottleneck of our proposed method is that the computation of the exponential map involves solving a system of second-order non-linear ODEs. This scales linearly with the number of observations and the number of parameters. Therefore, this opens up the possibility of researching new solvers. In addition to that, a different research direction is to use different metrics instead of the one used in this work, and indeed Yu et al. (2023) has already proposed a new metric for this task.

Research Question 3: *Which is the correct way to perform clustering using variational autoencoders? Can we easily extend this framework to perform co-clustering?*

For Research Question 3, we proposed an efficient and tractable estimate of the quantity we are interested in when clustering using deep latent variable models, that is, $p_\theta(c|\mathbf{x})$, using self-normalized importance sampling. We compare the benefits of using it instead of directly computing the responsibilities using the mean of the variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and found that using our estimate provides a more meaningful uncertainty estimation, but there was no significant improvement in terms of clustering metrics. Given the success of clustering using VAE, we also extended the framework to tackle co-clustering tasks.

The tractable estimate for $p_\theta(c|\mathbf{x})$ we proposed in Chapter 5 comes with an increase in complexity. Despite providing a more meaningful uncertainty estimate, it does not lead to a better clustering accuracy in most cases. Regarding the co-clustering model, there are not so many available co-clustering datasets with known ground-truth clusters for both rows and columns. Therefore, most of the evaluations for this task are done on synthetic datasets. A possible extension of this model would be to have a prior over the number of clusters and infer them during training.

Research Question 4: *How can we create a generative model to model and simulate accurately long partial-differential equations (PDEs) trajectories? Can diffusion models be used for this task?*

For Research Question 4, we considered and analyzed different techniques to obtain a conditional diffusion model for the specific goal of modeling long-range rollouts

of PDEs. In the paper, we compare the use of an amortized conditional score and a joint score that is made conditional a posteriori via reconstruction guidance. The joint score is trained on short segments as proposed by Rozet & Louppe (2023a) which corresponds to learning a Markov score network. We highlight how this approach is effective in the data assimilation task but does not work if it is directly applied to the forecasting task. We propose to use the same score and autoregressive rollouts to solve this problem. In the paper, we performed an extensive comparison of amortization versus reconstruction guidance in the context of long-range rollouts. The general conclusion is that training a joint score and conditioning it a posteriori depending on the task one is interested in solving results in a very flexible and general method that, in terms of forecasting, is competitive with the amortized score, which is trained on the specific forecasting task.

The approach is a promising way to have a single model that can be used for multiple tasks of interest in the context of PDE modeling. However, sampling is slow for diffusion models; therefore, having to rely on an autoregressive rollout to generate trajectories makes these models slower compared to other approaches directly trained to predict the next state in just one forward pass.

Bibliography of the Background

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Anderson, B. D. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. An introduction to mcmc for machine learning. *Machine learning*, 50:5–43, 2003.

Antorán, J., Janz, D., Allingham, J. U., Daxberger, E., Barbano, R. R., Nalisnick, E., and Hernández-Lobato, J. M. Adapting the linearised laplace model evidence for modern deep learning. In *International Conference on Machine Learning (ICML)*, 2022.

Arvanitidis, G., Hansen, L. K., and Hauberg, S. Latent space oddity: on the curvature of deep generative models. In *International Conference on Learning Representations (ICLR)*, 2018.

Babaeizadeh, M., Saffar, M. T., Nair, S., Levine, S., Finn, C., and Erhan, D. Fitvid: Overfitting in pixel-level video prediction. *arXiv preprint arXiv:2106.13195*, 2021.

Bauer, M. and Mnih, A. Resampled priors for variational autoencoders. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 66–75. PMLR, 2019.

Bengio, Y. and Bengio, S. Modeling high-dimensional discrete data with multi-layer neural networks. *Advances in Neural Information Processing Systems*, 12, 1999.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Benjamini, Y. and Hochberg, Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

Bishop, C. M. Novelty detection and neural network validation. *IEE Proceedings-Vision, Image and Signal processing*, 141, 1994.

Botev, A., Ritter, H., and Barber, D. Practical Gauss-Newton optimisation for deep learning. In *International Conference on Machine Learning (ICML)*, 2017.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Burda, Y., Grosse, R., and Salakhutdinov, R. Importance weighted autoencoders. In *4th International Conference on Learning Representations, (ICLR), 2016*, 2016.

Butler, K. T., Davies, D. W., Cartwright, H., Isayev, O., and Walsh, A. Machine learning for molecular and materials science. *Nature*, 559(7715):547–555, 2018.

Cattiaux, P., Conforti, G., Gentil, I., and Léonard, C. Time reversal of diffusion processes under a finite entropy condition, September 2022.

Child, R. Very deep vaes generalize autoregressive models and can outperform them on images. In *9th International Conference on Learning Representations, ICLR*, 2021.

Chung, H., Sim, B., Ryu, D., and Ye, J. C. Improving Diffusion Models for Inverse Problems using Manifold Constraints. *Advances in Neural Information Processing Systems*, 35, 2022.

Chung, H., Kim, J., Mccann, M. T., Klasky, M. L., and Ye, J. C. Diffusion Posterior Sampling for General Noisy Inverse Problems. In *The Eleventh International Conference on Learning Representations, ICLR*, 2023.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. A recurrent latent variable model for sequential data. *Advances in neural information processing systems*, 28, 2015.

Clark, A., Donahue, J., and Simonyan, K. Adversarial video generation on complex datasets. *arXiv preprint arXiv:1907.06571*, 2019.

Corso, G., Stärk, H., Jing, B., Barzilay, R., and Jaakkola, T. Diffdock: Diffusion steps, twists, and turns for molecular docking. *International Conference on Learning Representations (ICLR)*, 2023.

Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M., and Hennig, P. Laplace redux–effortless Bayesian deep learning. In *Neural Information Processing Systems (NeurIPS)*, 2021a.

Daxberger, E., Nalisnick, E., Allingham, J. U., Antorán, J., and Hernández-Lobato, J. M. Bayesian deep learning via subnetwork inference. In *International Conference on Machine Learning (ICML)*, 2021b.

Denker, J. and LeCun, Y. Transforming neural-net output levels to probability distributions. *Advances in neural information processing systems*, 3, 1990.

Didi, K., Vargas, F., Mathis, S. V., Dutordoir, V., Mathieu, E., Komorowska, U. J., and Lio, P. A framework for conditional diffusion modelling with applications in motif scaffolding for protein design, December 2023.

do Carmo, M. *Riemannian Geometry*. Mathematics (Boston, Mass.). Birkhäuser, 1992.

Du, Y. and Mordatch, I. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32, 2019.

Duval, A., Mathis, S. V., Joshi, C. K., Schmidt, V., Miret, S., Malliaros, F. D., Cohen, T., Liò, P., Bengio, Y., and Bronstein, M. A hitchhiker's guide to geometric gnns for 3d atomic systems. *arXiv preprint arXiv:2312.07511*, 2023.

Efron, B. Tweedie's formula and selection bias. *Journal of the American Statistical Association*, 106, 2011.

Eschenhagen, R., Immer, A., Turner, R., Schneider, F., and Hennig, P. Kronecker-factored approximate curvature for modern neural network architectures. *Advances in Neural Information Processing Systems*, 36, 2024.

Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 512–519. IEEE, 2016.

Finzi, M. A., Boral, A., Wilson, A. G., Sha, F., and Zepeda-Nunez, L. User-defined Event Sampling and Uncertainty Quantification in Diffusion Models for Physical Dynamical Systems. In *International Conference on Machine Learning*. PMLR, 2023.

Fisher, R. *Statistical methods for research workers*. Edinburgh Oliver & Boyd, 1925.

Fong, E. and Holmes, C. C. On the marginal likelihood and cross-validation. *Biometrika*, 107(2):489–496, 2020.

Foresee, F. D. and Hagan, M. T. Gauss-Newton approximation to Bayesian learning. In *Proceedings of International Conference on Neural Networks (ICNN)*. IEEE, 1997.

Gershman, S. and Goodman, N. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

Ghahramani, Z. and Jordan, M. I. Learning from incomplete data. 1995.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., Norouzi, M., and Swersky, K. Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*, 2019.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *The Journal of Machine Learning Research*, 2012.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. In *International conference on machine learning*, pp. 1321–1330. PMLR, 2017.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

Hauberg, S. *Differential geometry for generative modeling*. 2024.

Haussmann, U. G. and Pardoux, E. Time reversal of diffusions. *The Annals of Probability*, 1986.

Hendrycks, D., Mazeika, M., and Dietterich, T. Deep anomaly detection with outlier exposure. In *7th International Conference on Learning Representations, (ICLR)*, 2019.

Heskes, T. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33, 2020.

Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., and Fleet, D. J. Video diffusion models. In *Deep Generative Models for Highly Structured Data Workshop, ICLR*, volume 10, 2022.

Hoffman, M. D. Learning deep latent gaussian models with markov chain monte carlo. In *International conference on machine learning*, pp. 1510–1519. PMLR, 2017.

Hoffman, M. D. and Johnson, M. J. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, volume 1, 2016.

Höppe, T., Mehrjou, A., Bauer, S., Nielsen, D., and Dittadi, A. Diffusion models for video prediction and infilling. *arXiv preprint arXiv:2206.07696*, 2022.

Hyvärinen, A. and Dayan, P. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

Immer, A., Bauer, M., Fortuin, V., Rätsch, G., and Emtiyaz, K. M. Scalable marginal likelihood estimation for model selection in deep learning. In *International Conference on Machine Learning (ICML)*, 2021a.

Immer, A., Korzepa, M., and Bauer, M. Improving predictions of Bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021b.

Ingraham, J., Garg, V., Barzilay, R., and Jaakkola, T. Generative models for graph-based protein design. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Ivanov, O., Figurnov, M., and Vetrov, D. Variational autoencoder with arbitrary conditioning. In *International Conference on Learning Representations*, 2018.

Jefferys, W. H. and Berger, J. O. Ockham's razor and bayesian analysis. *American scientist*, 80(1):64–72, 1992.

Jiang, Z., Zheng, Y., Tan, H., Tang, B., and Zhou, H. Variational deep embedding: An unsupervised and generative approach to clustering. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 1965–1972, 2017.

Kalchbrenner, N., Oord, A., Simonyan, K., Danihelka, I., Vinyals, O., Graves, A., and Kavukcuoglu, K. Video pixel networks. In *International Conference on Machine Learning*, pp. 1771–1779. PMLR, 2017.

Khan, M. E. E., Immer, A., Abedi, E., and Korzepa, M. Approximate inference turns deep networks into Gaussian Processes. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR,*, 2014.

Kingma, D. P., Mohamed, S., Jimenez Rezende, D., and Welling, M. Semi-supervised learning with deep generative models. *Advances in neural information processing systems*, 27, 2014.

Kohl, G., Chen, L.-W., and Thuerey, N. Turbulent flow simulation using autoregressive conditional diffusion models. *arXiv preprint arXiv:2309.01745*, 2023.

Kristiadi, A., Hein, M., and Hennig, P. Being bayesian, even just a bit, fixes overconfidence in ReLU networks. In *International Conference on Machine Learning (ICML)*, 2020.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Larochelle, H. and Murray, I. The neural autoregressive distribution estimator. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 29–37. JMLR Workshop and Conference Proceedings, 2011.

Lawrence, N. D. *Variational inference in probabilistic models*. PhD thesis, Citeseer, 2001.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Lee, J. *Introduction to Riemannian Manifolds*. Graduate Texts in Mathematics. Springer International Publishing, 2019.

Lotfi, S., Izmailov, P., Benton, G., Goldblum, M., and Wilson, A. G. Bayesian model selection, the marginal likelihood, and generalization. In *International Conference on Machine Learning*, pp. 14223–14247. PMLR, 2022.

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models, 2023.

Luc, P., Clark, A., Dieleman, S., Casas, D. d. L., Doron, Y., Cassirer, A., and Simonyan, K. Transformation-based adversarial video prediction on large-scale data. *arXiv preprint arXiv:2003.04035*, 2020.

Ma, C., Gong, W., Hernández-Lobato, J. M., Koenigstein, N., Nowozin, S., and Zhang, C. Partial vae for hybrid recommender system. In *NIPS Workshop on Bayesian Deep Learning*, volume 2018, 2018.

Ma, C., Tschiatschek, S., Palla, K., Hernandez-Lobato, J. M., Nowozin, S., and Zhang, C. Eddi: Efficient dynamic discovery of high-value information with partial vae. In *International Conference on Machine Learning*, pp. 4234–4243. PMLR, 2019.

MacKay, D. J. The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736, 1992.

MacKay, D. J. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Mackay, D. J. C. *Bayesian methods for adaptive models*. California Institute of Technology, 1992.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Mattei, P.-A. and Frellsen, J. Miwae: Deep generative modelling and imputation of incomplete data sets. In *International conference on machine learning*, pp. 4413–4423. PMLR, 2019.

Meng, X. and Kabashima, Y. Diffusion Model Based Posterior Sampling for Noisy Linear Inverse Problems, May 2023.

Mitchell, T. R., Thompson, L., Peterson, E., and Cronk, R. Temporal adjustments in the evaluation of events: The "rosy view". *Journal of experimental social psychology*, 33(4):421–448, 1997.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020.

Nalisnick, E., Matsukawa, A., Teh, Y. W., Gorur, D., and Lakshminarayanan, B. Do deep generative models know what they don't know? In *International Conference on Learning Representations (ICLR)*, 2018.

Nalisnick, E., Matsukawa, A., Teh, Y. W., and Lakshminarayanan, B. Detecting out-of-distribution inputs to deep generative models using a test for typicality. *arXiv preprint arXiv:1906.02994*, 2019.

Nazabal, A., Olmos, P. M., Ghahramani, Z., and Valera, I. Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, 107:107501, 2020.

Neal, R. M. Connectionist learning of belief networks. *Artificial intelligence*, 56(1): 71–113, 1992.

Neal, R. M. Probabilistic inference using markov chain monte carlo methods. 1993.

Neal, R. M. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2011.

Nguyen, A., Yosinski, J., and Clune, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 427–436, 2015.

Noé, F., De Fabritiis, G., and Clementi, C. Machine learning for protein folding and dynamics. *Current opinion in structural biology*, 60:77–84, 2020.

Oksendal, B. *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

Oprea, S., Martinez-Gonzalez, P., Garcia-Garcia, A., Castro-Vargas, J. A., Orts-Escolano, S., Garcia-Rodriguez, J., and Argyros, A. A review on deep learning techniques for video prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(6):2806–2826, 2020.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

Parisi, G. Correlation functions and computer simulations. *Nuclear Physics B*, 1981.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., and Ghahramani, Z. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pp. 7563–7574. PMLR, 2020a.

Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., and Ghahramani, Z. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pp. 334–344. PMLR, 2020b.

Pokle, A., Muckley, M. J., Chen, R. T. Q., and Karrer, B. Training-free Linear Image Inversion via Flows, September 2023.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. 2017.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pp. 8821–8831. PMLR, 2021.

Rao, C. R. Large sample tests of statistical hypotheses concerning several parameters with applications to problems of estimation. In *Mathematical Proceedings of the Cambridge Philosophical Society*, 1948.

Rasmussen, C. and Ghahramani, Z. Occam's razor. *Advances in neural information processing systems*, 13, 2000.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pp. 1278–1286. PMLR, 2014.

Ritter, H., Botev, A., and Barber, D. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Robbins, H. E. An empirical bayes approach to statistics. In *Breakthroughs in Statistics: Foundations and basic theory*, pp. 388–394. Springer, 1992.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Rosca, M., Lakshminarayanan, B., and Mohamed, S. Distribution matching in variational inference. *arXiv preprint arXiv:1802.06847*, 2018.

Rozet, F. and Louppe, G. Score-based Data Assimilation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Rozet, F. and Louppe, G. Score-based data assimilation for a two-layer quasi-geostrophic model. *arXiv preprint arXiv:2310.01853*, 2023b.

Rubin, D. B. Inference and missing data. *Biometrika*, 63, 1976.

Sagun, L., Bottou, L., and LeCun, Y. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Saxena, V., Ba, J., and Hafner, D. Clockwork variational autoencoders. *Advances in Neural Information Processing Systems*, 34:29246–29257, 2021.

Schneuing, A., Du, Y., Harris, C., Jamasb, A., Igashov, I., Du, W., Blundell, T., Lió, P., Gomes, C., Welling, M., et al. Structure-based drug design with equivariant diffusion models. *arXiv preprint arXiv:2210.13695*, 2022.

Schraudolph, N. N. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

Sharma, M., Farquhar, S., Nalisnick, E., and Rainforth, T. Do Bayesian Neural Networks Need To Be Fully Stochastic? In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2015.

Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. Ladder variational autoencoders. *Advances in neural information processing systems*, 29, 2016.

Song, J., Meng, C., and Ermon, S. Denoising Diffusion Implicit Models. October 2020a. URL https://arxiv.org/abs/2010.02502v1.

Song, J., Vahdat, A., Mardani, M., and Kautz, J. Pseudoinverse-Guided Diffusion Models for Inverse Problems. In *International Conference on Learning Representations*, 2022.

Song, J., Zhang, Q., Yin, H., Mardani, M., Liu, M.-Y., Kautz, J., Chen, Y., and Vahdat, A. Loss-Guided Diffusion Models for Plug-and-Play Controllable Generation. 2023.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

Song, Y., Garg, S., Shi, J., and Ermon, S. Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in Artificial Intelligence*, pp. 574–584. PMLR, 2020b.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Teh, Y. W., Welling, M., Osindero, S., and Hinton, G. E. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4 (Dec):1235–1260, 2003.

Tomczak, J. and Welling, M. Vae with a vampprior. In *International conference on artificial intelligence and statistics*, pp. 1214–1223. PMLR, 2018.

Vahdat, A. and Kautz, J. Nvae: A deep hierarchical variational autoencoder. *Advances in neural information processing systems*, 33:19667–19679, 2020.

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 12, 2016.

Van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

Van Den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. In *International conference on machine learning*, pp. 1747–1756. PMLR, 2016.

Vapnik, V. Principles of risk minimization for learning theory. *Advances in neural information processing systems*, 4, 1991.

Vincent, P. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Hanikel, N., Pellock, S. J., Courbet, A., Sheffler, W., Wang, J., Venkatesh, P., Sappington, I., Torres, S. V., Lauko, A., De Bortoli, V., Mathieu, E., Ovchinnikov, S., Barzilay, R., Jaakkola, T. S., DiMaio, F., Baek, M., and Baker, D. De novo design of protein structure and function with RFdiffusion. *Nature*, 2023.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Wu, K. E., Yang, K. K., Berg, R. v. d., Zou, J. Y., Lu, A. X., and Amini, A. P. Protein structure generation via folding diffusion. *arXiv preprint arXiv:2209.15611*, 2022.

Yang, G. and Sommer, S. A denoising diffusion model for fluid field prediction. *arXiv e-prints*, pp. arXiv–2301, 2023.

Yang, R., Srivastava, P., and Mandt, S. Diffusion probabilistic modeling for video generation. *Entropy*, 2023.

Yu, H., Hartmann, M., Williams, B., Girolami, M., and Klami, A. Riemannian laplace approximation with the fisher metric. *arXiv preprint arXiv:2311.02766*, 2023.

Zhang, L., Goldstein, M., and Ranganath, R. Understanding failures in out-of-distribution detection with deep generative models. In *International Conference on Machine Learning (ICML)*. PMLR, 2021.

Zhang, Q. and Chen, Y. Fast Sampling of Diffusion Models with Exponential Integrator. In *The Eleventh International Conference on Learning Representations, ICLR*. OpenReview.net, 2023.

The artwork on the cover of this Ph.D. thesis was made by my friend Alessandro MARIA Doro.