Integrating Bayesian Network Structure into Normalizing Flows and Variational Autoencoders

by

Jacobie Mouton

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science in the Faculty of Science at Stellenbosch University

Supervisor: Prof. S. Kroon

March 2023

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: ....March 2023..........

Abstract

Integrating Bayesian Network Structure into Normalizing Flows and Variational Autoencoders

J. Mouton

Computer Science Division,
Stellenbosch University,
Private Bag X1, Matieland 7602, South Africa.

Thesis : MSc (Computer Science)

March 2023

Deep generative models have become more popular in recent years due to their good scalability and representation capacity. However, these models do not typically incorporate domain knowledge. In contrast, probabilistic graphical models specifically constrain the dependencies between the variables of interest as informed by the domain. In this work, we therefore consider integrating probabilistic graphical models and deep generative models in order to construct models that are able to learn complex distributions, while remaining interpretable by leveraging prior knowledge about variable interactions. We specifically consider the type of domain knowledge that can be represented by Bayesian networks, and restrict our study to the deep generative frameworks of normalizing flows and variational autoencoders.

Normalizing flows (NFs) are an important family of deep neural networks for modelling complex distributions as transformations of simple base distributions. Graphical flows add further structure to NFs, allowing one to encode non-trivial variable dependencies in these distributions. Previous graphical flows have focused primarily on a single flow direction: either the normalizing direction for density estimation, or the generative direction for inference and sampling. However, to use a single flow to perform tasks in both directions, the model must exhibit stable and efficient flow inversion. This thesis introduces graphical residual flows (GRFs)—graphical flows based on invertible residual networks—which ensure *stable* invertibility by spectral normalization of its weight matrices. Experiments confirm that GRFs provide performance competitive with other graphical flows for both density estimation and inference tasks. Furthermore, our model provides stable and accurate inversion that is also more time-efficient than alternative flows with similar task performance. We therefore recommend the use of GRFs over other graphical flows when the model may be required to perform reliably in both directions.

Since flows employ a bijective transformation, the dimension of the base or *latent* distribution must have the same dimensionality as the observed data. Variational autoencoders (VAEs) address this shortcoming by allowing practitioners to specify any number of latent variables. Initial work on VAEs assumed independent latent variables with simple prior and variational distributions. Subsequent work has explored incorporating more complex distributions and dependency structures: including NFs in the encoder network allows latent variables to entangle non-linearly, creating a richer class of distributions for the approximate posterior, and stacking layers of latent variables allows more complex priors to be specified. In this vein, this thesis also explores incorporating *arbitrary* dependency structures—as specified by Bayesian networks—into VAEs. This is achieved by extending both the prior and inference network with the above GRF, resulting in the structured invertible residual network (SIReN) VAE. We specifically consider GRFs, since the application of the flow in the VAE prior necessitates stable inversion. We compare our model's performance on several datasets to models that encode no special dependency structures, and show its potential to provide a more interpretable model as well as better generalization performance in data-sparse settings. We also identify posterior collapse—where some latent dimensions become inactive and are effectively ignored by the model—as an issue with SIReN-VAE, as it is linked with the encoded structure. As such, we employ various combinations of existing approaches to alleviate this phenomenon.

Uittreksel

Die Integrasie van Bayesiaanse Netwerkstrukture in Normaliserende Strome en Variasionele Outo-enkodeerders

J. Mouton

Rekenaarwetenskapafdeling,
Stellenbosch Universiteit,
Privaat Sak X1, Matieland 7602, Suid-Afrika.

Tesis : MSc (Rekenaarwetenskap)

Maart 2023

Diep generatiewe modelle het die afgelope paar jaar gewild geword as gevolg van hul goeie skaalbaarheid en verteenwoordigingskapasiteit. Hierdie modelle inkorporeer egter nie tipies domeinkennis nie. In teenstelling hiermee beperk grafiese waarskynlikheidsmodelle spesifiek die voorwaardelike onafhanklikhede tussen die veranderlikes van belang soos deur die domein ingelig. In hierdie werk oorweeg ons dus die integrasie van grafiese waarskynlikheidsmodelle en diep generatiewe modelle om sodoende modelle te konstrueer wat komplekse verdelings kan leer, terwyl hul interpreteerbaar bly deur kennis oor veranderlike interaksies te benut. Ons oorweeg spesifiek die tipe domeinkennis wat deur Bayesiaanse netwerke verteenwoordig kan word, en beperk ons studie tot die diep generatiewe raamwerke van normaliserende strome en variasionele outo-enkodeerders.

Normaliserende strome (NF'e) is 'n belangrike familie van diep neurale netwerke vir die modellering van komplekse verdelings as transformasies van eenvoudige basisverdelings. Grafiese strome voeg verdere struktuur aan NF'e, wat ‘n mens in staat stel om nie-triviale veranderlike afhanklikhede in hierdie verdelings te enkodeer. Vorige grafiese strome het hoofsaaklik op 'n enkele vloeirigting gefokus: die normaliserende rigting vir digtheidskatting, of die generatiewe rigting vir statistiese afleiding en steekproefneming. Om egter 'n enkele stroom te gebruik om take in beide rigtings uit te voer, moet die model stabiele en doeltreffende inversie toon. Hierdie tesis stel grafiese residuele strome (GRF'e) bekend wat gebaseer is op inverteerbare residuele netwerke. GRF'e verseker *stabiele* inverteerbaarheid deur spektrale normalisering van hul gewigsmatrikse. Eksperimente bevestig dat GRF'e kompeterende modeleringsvermoë bied in vergelyking met ander grafiese strome vir beide digtheidsskatting en afleidingstake. Verder bied ons model stabiele en akkurate inversie wat ook meer tydsdoeltreffend is as alternatiewe strome met soortgelyke taakverrigting. Ons beveel dus die gebruik van GRF'e aan wanneer die model vereis is om betroubaar in beide vloeirigtings te werk.

Aangesien strome 'n byektiewe transformasie gebruik, moet die dimensie van die basis of *latente* verspreiding dieselfde dimensionaliteit hê as die waargenome data. Variasionele outo-enkodeerders (VAE's) spreek hierdie tekortkoming aan deur praktisyns toe te laat om enige aantal latente veranderlikes te spesifiseer. Aanvanklike werk op VAE's het onafhanklike latente veranderlikes met eenvoudige verdelings aanvaar. Daaropvolgende werk het ondersoek ingestel na die insluiting van meer komplekse verdelings en afhanklikheidstrukture: die insluiting van NF'e in die enkoderingsnetwerk laat latente veranderlikes toe om nie-lineêr te verstrengel wat 'n ryker klas verdelings vir die benaderde posteriori-verdeling skep, en die stapeling van lae latente veranderlikes laat toe dat meer komplekse priori-verdelings gespesifiseer word. In 'n soortgelyke trant ondersoek hierdie tesis die inkorporering van *arbitrêre* afhanklikheidstrukture, soos gespesifiseer deur Bayesiaanse netwerke, in VAE's. Dit word bewerkstellig deur beide die priori-verdeling en die afleidingsnetwerk uit te brei met die bogenoemde GRF, wat lei tot die gestruktureerde inverteerbare residuele netwerk (SIReN) VAE. Ons oorweeg spesifiek GRF'e, aangesien die toepassing van die stroom in die VAE priori-verdeling stabiele inversie benodig. Ons vergelyk ons model se modeleringsvermoë op verskeie datastelle teenoor modelle wat geen spesifieke afhanklikheidstrukture inkorporeer nie, en toon die potensiaal daarvan om 'n meer interpreteerbare model te verskaf sowel as beter veralgemeningsvermoë wanneer beperkte data beskikbaar is. Ons identifiseer ook posteriori-ineenstorting—waar sommige latente dimensies deur die model geïgnoreer word—as 'n probleem met SIReN-VAE, aangesien dit gekoppel is aan die geënkodeerde struktuur. As sodanig evalueer ons verskeie kombinasies van bestaande tegnieke om hierdie verskynsel te verhoed.

Acknowledgements

First and foremost my praise is to God for the blessing and grace to have been able to complete this work.

I am immensely grateful to my supervisor, Prof Steve Kroon, for his continual guidance and encouragement that frequently went that extra mile. I would also like to thank my parents and friends for their love and support through all the ups and downs of bringing this thesis to completion.

Lastly, I would like to thank the DeepMind scholarship programme for the financial support needed to complete this work.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

BN	Bayesian network	2, 9
CAVI	Coordinate ascent variational inference	13
DAG	Directed acyclic graph	9
DLVM	Deep latent variable model	28
DReG	Doubly-reparameterized gradient	33, 91
ELBO	Evidence Lower BOund	12
FID	Fréchet Inception Distance	100
GNF-M	Graphical normalizing flow with monotonic transformations	63
GNF-A	Graphical normalizing flow with affine transformations	62
GRF	Graphical residual flow	5, 47
i.i.d.	Independently and identically distributed	10
IWAE	Importance weighted autoencoder	89
MI	Mutual information	34, 35
MINE	Mutual information neural estimation	35, 95
MLP	Multi-layer perceptron	18
NF	Normalizing flow	2, 13
ODE	Ordinary differential equation	13
PGM	Probabilistic graphical model	1, 8
SCCNF	Structured conditional continuous normalizing flow	41
SIReN-VAE	Structured invertible residual network VAE	5, 85
VAE	Variational autoencoder	2, 28
VI	Variational inference	11
WU	Warm-up	88

Notation

$\mathbf{x}$	Observed variables	2
$\mathbf{z}$	Latent variables	2
$p(\cdot)$	Probability distribution function	2
$q(\cdot)$	Variational distribution function	12
θ	Model parameters	85
ϕ	Variational or inference model parameters	86
$\mathrm{Lip}(f)$	Lipschitz constant of a Lipschitz continuous function f	20
$\odot$	Element-wise multiplication	32
$\mathrm{KL}(\cdot\|\|\cdot)$	Kullback-Leibler divergence	10
$\mathcal{G}$	Bayesian network graph	9
$\mathrm{Pa}(\cdot)$	Parent vertices in graph	9
$[f(\cdot)]_i$	Component i of the output of a vector-valued function f	16

Chapter 1

Introduction

Machine learning addresses the task of constructing models of observed events from data. The difficulty is that the available data rarely paints a complete picture of the underlying system, and there are typically many unknown factors at play. Fortunately, we can employ probability theory to express different forms of uncertainty and noise associated with these models, and this in turn allows us to use Bayes' rule to make inferences about unknown quantities, to update our models based on new observations, and to make predictions. This approach is known as probabilistic modelling. One of the main classes of probabilistic modelling is generative modelling. Probabilistic generative models aim to learn how to generate new data from the same distribution as the observations. Although there are various approaches to generative modelling, the rise of deep learning in the past decade has lead to the widespread use of neural methods—known as deep generative models. These models are capable of learning complex distributions and leverage (stochastic) gradient descent optimization with automatic differentiation software to efficiently optimize the model parameters.

One of the attractive properties of generative modelling, is that it allows practitioners to constrain the generative process based on knowledge from the application domain, while factors they either do not know or do not care about can simply be treated as noise. One class of probabilistic generative models that specifically constrains dependencies between the variables of interest by specifying conditional independencies, is probabilistic graphical models (PGMs). Graphical models allow experts to specify domain knowledge about the relationships between variables, yielding more compact and interpretable models. Although there are typically other ways in which generative models can be informed by their application domain as well as other types of deep generative models that are not necessarily probabilistic in nature, this work will focus on the two classes of generative models discussed above: graphical models and deep probabilistic generative models.

1.1 Problem Statement & Research Questions

1.1.1 General Problem Statement

Deep generative models have become more popular in recent years due to their good scalability and representation capacity. Unlike graphical models, they typically, however, do not incorporate any specific domain knowledge which could aid in their overall performance and interpretability. We are therefore interested in combining the strengths of these two approaches. As such, the key focus of this work is to investigate the integration of deep generative models and probabilistic graphical models to construct *interpretable* models which can learn *complex* distributions while *leveraging prior knowledge* about variable interactions. Below, we provide the necessary context and scope within which this idea is explored, and which leads up to our specific research questions.

1.1.2 Context

Two prominent classes of deep probabilistic generative models include normalizing flows (NFs) (Rezende and Mohamed, 2015; Tabak and Turner, 2013) and variational autoencoders (VAEs) (Kingma and Welling, 2014; Rezende *et al.*, 2014). NFs combine a simple base distribution with a differentiable and bijective mapping between this base distribution and a more complex distribution. Recently, Wehenkel and Louppe (2021) provided a new insight into the relationship between NFs and Bayesian networks (BNs)—a type of PGM with a directed acyclic graph dependency structure. Specifically, the modelling assumptions underlying certain types of transformations used in NFs correspond to specific classes of BNs with predefined graphical structures. This naturally gave rise to the question of whether one could use this correspondence to encode an arbitrary BN dependency structure. This line of inquiry led to graphical NFs, which use weight masking to encode a dependency structure in the flow architecture (Wehenkel and Louppe, 2021; Weilbach *et al.*, 2020). These studies have, however, only extended certain classes of flows to incorporate graphical dependency information.

VAEs jointly train a generative model and inference network. Since flows employ a bijective transformation, the dimension of the base or latent distribution must have the same dimensionality as the observed data. VAEs address this shortcoming by allowing practitioners to specify any number of latent variables. The generative model and inference network correspond, respectively, to the joint $p(\mathbf{x}, \mathbf{z})$ and an approximation to the posterior $p(\mathbf{z}|\mathbf{x})$ in Bayes' rule: $p(\mathbf{z}|\mathbf{x}) = p(\mathbf{x}, \mathbf{z})/p(\mathbf{x})$. Here, $\mathbf{x} = \{x_i\}_{i=1}^{D}$ and $\mathbf{z} = \{z_i\}_{i=1}^{K}$ are sets of observed and latent variables, respectively. Since a BN's dependency structure directly corresponds to the factorization of a joint distribution like $p(\mathbf{x}, \mathbf{z}) = p(x_1, \ldots, x_D, z_1, \ldots, z_K)$, it seems plausible to use this relationship to inject domain knowledge into the model. We expect that incorporating variable dependency information from BNs into VAEs can improve some, if not all aspects of their performance. Furthermore, since these models are typically used in an unsupervised manner, learning meaningful representations informed by the application domain, could aid in downstream tasks.

In order to limit the scope of our study, we considered only these two prominent classes of deep generative latent variable models, namely NFs and VAEs. Furthermore, we considered only the type of domain knowledge that can be represented by a BN.

1.1.3 Research Questions

Various dependency structures in the VAE generative network, which are intrinsically linked with the factorization of $p(\mathbf{x}, \mathbf{z})$, have been explored to improve performance (Burda *et al.*, 2016; Sønderby *et al.*, 2016). None of these, to the best of our knowledge, consider incorporating arbitrary graphical structures over the latent variables and between the latent and observed variables, based on the dependencies assumed to be present in the data. Besides using structure to improve the performance of VAEs, some work has considered improving modelling capability by increasing the complexity of the latent variable distribution using NFs (Kingma *et al.*, 2016; Rezende and Mohamed, 2015). Based on this idea, we considered whether graphical flows can be used as a vehicle to inject domain knowledge into a VAE. However, if a flow is to be used as part of $p(\mathbf{x}, \mathbf{z})$ in the VAE decoder, it must be stably invertible in practice, to allow both density calculation and sample generation (see Section 6.1). This requirement is not generally met for most types of flows (Behrmann *et al.*, 2021), and specifically not for those types of flows that have previously been extended to incorporate graphical dependency structures. We therefore additionally required a graphical flow that guarantees stable and efficient inversion. One of the main paradigms for ensuring stable inversion of invertible neural networks is bounding the Lipschitz constant of the transformation to a small value (Behrmann *et al.*, 2021). Since residual flows (Chen *et al.*, 2019) place such an upper bound on the Lipschitz constant of each step of the flow transformation by construction, we identified this type of flow as a potential candidate. In light of the above, we posed two main research questions:

Research Question 1: Can a new type of graphical flow be developed by adapting standard residual flows to encode arbitrary graphical structures, while still providing competitive density estimation and inference performance *as well as* stable inversion?

Research Question 2: Is it feasible to use these graphical residual flows to integrate information from a BN into a VAE, and is being able to do this useful?

1.2 Objectives

We specified the following objectives as stepping stones for investigating the above research questions:

Research Question 1

- Perform a literature study pertaining to the first research question.
- Extend residual flows to allow practitioners to specify and encode an arbitrary BN dependency structure in the architecture of the flow, while maintaining theoretical invertibility.
- Implement a proof-of-concept system of the developed graphical residual flow.
- Compare this new graphical flow to existing graphical flows in terms of density estimation and inference performance to determine whether it provides competitive performance.
- Measure the efficiency and stability of flow inversion of this new approach compared to that of existing graphical flows, in practice.

Research Question 2

- Perform a literature study pertaining to the second research question.
- Develop a way of using the graphical residual flow for encoding arbitrary BN dependency structures in a VAE.
- Implement a proof-of-concept system for encoding a BN dependency structure into VAEs using graphical residual flows.
- Evaluate this new graphical VAE using datasets that have an associated BN structure, by comparison with models that encode no specific dependency information.
- Identify and ameliorate any potential issues associated with this approach.
- Explore the extent to which the potential hypothesized benefits of this graphical VAE are realized, which include:
  - its application in data-sparse settings where utilizing domain knowledge could aid the training process, and
  - its potential to produce a more interpretable latent space.

1.3 Contributions

Work from this thesis was presented at the Workshop on Deep Generative Models for Highly Structured Data at the 2022 International Conference on Learning Representations (Mouton and Kroon, 2022*a*,*b*). The work presented in these papers and in this thesis makes the following contributions:

- We propose graphical residual flows (GRFs)—an extension of residual flows (Chen *et al.*, 2019) that encodes an arbitrary Bayesian network structure between the variables of interest (Mouton and Kroon, 2022*a*, see Chapter 4).
- This dependency structure is encoded using a new masking scheme, extending the work of Germain *et al.* (2015), that overcomes the shortcomings of the schemes employed to encode structure in existing graphical flows.
- We show that GRFs exhibit more stable and efficient flow inversion than alternative graphical flows, while providing competitive density estimation and inference performance (Chapter 5).
- We propose an alternative to Chen *et al.*'s (2019) LipSwish activation function: a Lipschitz-constrained Mish (Misra, 2020) activation function we call LipMish, which typically provides better modelling performance on our chosen datasets (Section 4.8).
- Pursuant to the successful development of GRFs, we incorporate graphical flows into VAEs by extending both the decoder prior and inference network with GRFs. The resulting model is called the structured invertible residual network variational autoencoder (SIReN-VAE) (Mouton and Kroon, 2022*b*, see Chapter 6).
- We identify posterior collapse (Razavi *et al.*, 2019)–where some latent dimensions become inactive and are effectively ignored by the model—as an issue with SIReN-VAE, as the encoded structure plays a role in which variables are more likely to collapse. We employ various combinations of existing approaches to alleviate this phenomenon and show that this leads to improved performance (Sections 6.2 and 7.4).
- We empirically show this model's potential for more effective training in data-sparse settings as well as its ability to provide more interpretable latent spaces (Sections 7.5 and 7.6), when practitioners know or can hypothesize about certain latent factors in their domain, as well as their relationships.

1.4 Outline

The rest of this thesis is structured as follows: Chapter 2 introduces the necessary background information on probabilistic modelling, with a specific focus on generative modelling with Bayesian networks, normalizing flows and variational autoencoders. Chapter 3 next provides an overview of relevant recent literature, where we specifically consider existing graphical flows as well as approaches that add structure to VAEs. Chapter 4 presents the proposed graphical residual flow, with Chapter 5 evaluating and comparing its performance to alternative graphical flow approaches. Based on the successful development of GRFs, Chapter 6 continues by presenting the proposed SIReN-VAE model, which integrates GRFs into a VAE. Chapter 7 evaluates the SIReN-VAE, and analyses the results. This includes investigating the phenomenon of posterior collapse, testing various remedies, as well as exploring the potential additional benefits of the SIReN-VAE approach which include its higher degree of interpretability as well as its ability to provide better generalization performance in data-sparse settings. Finally, Chapter 8 provides an overview of the thesis and some concluding remarks.

Chapter 2

Background

A key task in machine learning is modelling observed phenomena from data. Since real-world events are usually influenced by unknown factors, data rarely paints the complete picture. It is therefore beneficial to incorporate uncertainty into these models. This uncertainty is specified in terms of probability distributions, resulting in *probabilistic models*. Probabilistic modelling is a general framework for constructing models that encode domain knowledge and uncertainty about the complex systems we wish to reason about. These complex systems are typically represented by a collection of interacting properties or variables.

A specific class of probabilistic models, known as probabilistic *generative models*, aims to learn how to generate new data from the same distribution as that which is specified by the underlying, and unknown, complex system (or "generative process") that generated the observed data. Suppose this system is characterized in terms of a set of D random variables, $\mathcal{X} = \{X_1, ..., X_D\}$, the values of which define the state of the system. In order to reason probabilistically about the values of one or more of these random variables (possibly given observations of one or more other variables), we can construct a *joint distribution* $P_\theta(X_1, ..., X_D)$, specified by some parameters θ, over the space of possible assignments to this set of random variables. Generative modelling aims to find good values for θ such that $P_\theta(X_1, ..., X_D)$ is a good approximation to the true underlying joint distribution. This approximate distribution should be flexible enough to provide a sufficiently accurate model, while still allowing tractable optimization of the model parameters. Furthermore, it is desirable that these models allow us to incorporate prior knowledge about the true distribution.

There are various approaches to generative modelling. Traditional statistical modelling assume the observed data arises from a fixed family of distributions, and use direct analytical methods to obtain optimal values for the parameters of these distributions. Another subclass, known as probabilistic *graphical* models (PGMs), explicitly models the factorization of the joint distribution. This factorization implies a set of conditional independencies between the individual random variables of the joint. Some generative models introduce hidden or latent variables to help explain the data. The presence of these latent variables often renders direct analytical methods intractable and approximate model-fitting approaches such as variational methods (Jordan *et al.*, 1999) or expectation maximization (Dempster *et al.*, 1977) are therefore required. With the advent of neural networks and deep learning, neural approaches to generative modelling have also become widespread. Two prominent families that we focused on in this work, are variational autoencoders (VAEs) and normalizing flows (NFs). These models are capable of learning complex distributions, and leverage (stochastic) gradient descent optimization using automatic differentiation software to efficiently optimize the models' parameters θ.

The rest of this chapter is structured as follows. Section 2.1 briefly introduces PGMs and provides an overview of the main concepts underlying the class of Bayesian networks. Next, in Section 2.2, we discuss probabilistic generative models that incorporate additional latent variables and describe different techniques to perform inference—estimating the posterior distribution of these latent variables given the observations. Section 2.3 discusses the deep generative modelling framework of normalizing flows, including finite flows (Section 2.3.1), continuous flows (Section 2.3.2), the application of flows to the key tasks of probabilistic modelling and inference (Section 2.3.3), and the invertibility of flows in practice (Section 2.3.4). The final sections of this chapter introduce VAEs as deep latent variable models (Section 2.4.1), and then discuss the evidence lower bound used as training objective for VAEs (Section 2.4.2), how to estimate the marginal distribution of the observed variables (Section 2.4.3), the issue of posterior collapse (Section 2.4.4), and the concept of interpretability in deep latent variable models (Section 2.4.5).

2.1 Probabilistic Graphical Models

Probabilistic modelling, and specifically generative modelling, typically involves constructing a joint distribution, $P(X)$, over a set of D random variables, $\mathcal{X} = \{X_1, ..., X_D\}$. However, naïvely specifying or storing such a joint distribution is generally intractable. For example, in the simple case where each X_i is binary-valued, one already has to specify $2^D - 1$ probabilities, which is infeasible for all but the smallest values of D.

Fortunately, it is often the case that a given variable interacts *directly* with only a limited number of other variables. As a result, there is limited dependence between the various variables of the model. Knowledge of this structure allows one to encode the joint distribution in a manner that is more compact and tractable and enables modelling of distributions involving more variables. PGMs make use of graph-based representations to represent this structure and compactly encode the form of the joint distribution of $\mathcal{X}$. In this graphical representation, the random variables are denoted by vertices and the presence of direct variable interactions, by edges. The two main types of PGMs are Bayesian networks, which make use of a directed acyclic graph (DAG) representation, and Markov networks, which encode the joint using an undirected graph. Since this work focused primarily on Bayesian networks, we consider this class of PGMs in more detail.

2.1.1 Bayesian Networks

A Bayesian network (BN) is a type of PGM that uses DAGs for its graphical representation. Before fully defining a Bayesian network, we first formalize the idea of independence and conditional independence between random variables.

Definition 1 (Independence (Koller and Friedman, 2009)). *Let X and Y be sets of random variables. We say that X is* independent *of Y in distribution P, denoted by $(X \perp Y)$, if $P(X,Y) = P(X)P(Y)$, where $P(X)$ and $P(Y)$ are the marginal distributions.*

Definition 2 (Conditional independence (Koller and Friedman, 2009)). *Let X, Y and Z be sets of random variables. We say that X and Y are* conditionally independent *given Z in P, denoted by $(X \perp Y|Z)$, if $P(X,Y|Z) = P(X|Z)P(Y|Z)$.*

BNs exploit the *conditional independence* properties of a given joint distribution P in order to obtain a compact graphical representation.

Definition 3 (Bayesian network structure (Koller and Friedman, 2009)). *A BN structure $\mathcal{G}$ is a DAG whose vertices represent random variables $\mathcal{X} = \{X_1, \ldots, X_D\}$. Let $\mathrm{Pa}^{\mathcal{G}}_{X_i}$ denote the parent vertices of X_i in $\mathcal{G}$ and let $\mathrm{ND}^{\mathcal{G}}_{X_i}$ denote the variables that are not descendants of X_i in $\mathcal{G}$. Then $\mathcal{G}$ encodes the following set of conditional independencies:*

$$\left\{\left(X_i \perp \mathrm{ND}^{\mathcal{G}}_{X_i} \middle| \mathrm{Pa}^{\mathcal{G}}_{X_i}\right) : i = 1, \ldots, D\right\}.$$

Definition 4 (Factorization (Koller and Friedman, 2009)). *Let $\mathcal{G}$ be a DAG with vertices corresponding to random variables $\mathcal{X} = \{X_1, \ldots, X_D\}$. The joint distribution P over $\mathcal{X}$* factorizes *according to $\mathcal{G}$ if P has the necessary conditional independencies such that it can be expressed as the following product of* conditional probability distributions*:*

$$P(\mathcal{X}) = \prod_{X_i \in \mathcal{X}} P_i\left(X_i \middle| \mathrm{Pa}^{\mathcal{G}}_{X_i}\right). \tag{2.1}$$

The above equation is called the *chain rule for Bayesian networks.* This pair, consisting of a DAG $\mathcal{G}$ over $\mathcal{X}$ and a joint distribution P that factorizes according to $\mathcal{G}$, is known as a Bayesian network.

Definition 5 (Bayesian network)**.** *A* Bayesian network (Koller and Friedman, 2009) *is a pair $\mathcal{B} = (\mathcal{G}, P)$ where P factorizes over the DAG $\mathcal{G}$, and where P is specified by a set of conditional probability distributions associated with $\mathcal{G}$'s vertices, corresponding to the factors in (2.1).*

The above definitions provide two perspectives by which the graphical structure of a BN can be understood. First, as a data structure that provides the skeleton by which a distribution can be factorized, and second as a compact encoding of conditional *independence* statements about the distribution. The latter interpretation provides a clear indication of how one might utilize information from $\mathcal{G}$ when constructing deep generative models—by encoding these conditional independencies between the variables $\mathcal{X}$ in the neural network functions we construct.

Having specified a factorization of the joint distribution, one still needs to fit the parameters of the model to the data. Given a finite number of independently and identically distributed (i.i.d.) observations $\{\mathbf{x}_n\}_{n=1}^N$, a popular method for achieving this is maximum likelihood estimation. As stated previously, probabilistic generative modelling seeks to find a distribution $p(\mathbf{x})$ that provides a good approximation to the true joint distribution, $p^*(\mathbf{x})$, underlying the observed data. Let $p_\theta(\mathbf{x}) = \prod_{i=1}^D p_\theta(x_i | \operatorname{Pa}_{x_i}^{\mathcal{G}})$ denote a parameterized family of distributions modelled by a BN with graph $\mathcal{G}$ and parameters θ. Maximum likelihood estimation corresponds to minimizing (a Monte Carlo estimate of) the forward Kullback-Leibler (KL) divergence[1] (Kullback and Leibler, 1951) of $p^*(\mathbf{x})$ from $p_\theta(\mathbf{x})$:

$$\begin{aligned} \mathrm{KL}\left(p^*(\mathbf{x}) || \, p_\theta(\mathbf{x})\right) &= \mathbb{E}_{\mathbf{x}\sim p^*}\left[\log p^*(\mathbf{x}) - \log p_\theta(\mathbf{x})\right] \\ &= -\mathbb{E}_{\mathbf{x}\sim p^*}\left[\log p_\theta(\mathbf{x})\right] + \text{const} \\ &\approx -\frac{1}{N}\sum_{n=1}^{N} \log p_\theta(\mathbf{x}_n) + \text{const} \ . \end{aligned} \tag{2.2}$$

Thus, one is able to minimize a Monte Carlo approximation of the forward KL-divergence between the true and approximate distributions by maximizing the log-likelihood,

$$\sum_{n=1}^{N} \log p_\theta(\mathbf{x}_n) = \sum_{n=1}^{N}\sum_{i=1}^{D} \log p_\theta(x_{n,i} | \operatorname{Pa}_{x_{n,i}}^{\mathcal{G}}) \ . \tag{2.3}$$

Maximum likelihood estimation is naturally applicable for fitting a wide variety of probabilistic models, including NFs as we discuss in Section 2.3.3.1.

[1]The KL-divergence (a.k.a. relative entropy) is a quantity from information theory that measures the disparity between two probability distributions, p and q. It is non-symmetric (i.e. generally $\mathrm{KL}(q(\mathbf{x}) || \, p(\mathbf{x})) \neq \mathrm{KL}(p(\mathbf{x}) || \, q(\mathbf{x}))$), non-negative, and equal to zero only when $q(\mathbf{x}) = p(\mathbf{x})$.

2.2 Probabilistic Latent Variable Models

The approach followed in the previous section assumed that all the modelled variables are observed. A potentially much more powerful generative model can be created, however, by introducing unobserved variables, also known as hidden or latent variables. Let $p_\theta(\mathbf{x}, \mathbf{z})$ be the joint distribution over the observed variables $\mathbf{x}$ and additional latent variables $\mathbf{z}$ that do not form part of the dataset. The marginal likelihood of $\mathbf{x}$, also known as the (model) evidence, is recovered by marginalizing over $\mathbf{z}$, i.e. eliminating $\mathbf{z}$ from the joint distribution by integrating it out:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z})\, d\mathbf{z} \ . \tag{2.4}$$

The integral is replaced by a summation in the case where $\mathbf{z}$ is discrete. Typically, however, this integral is intractable to compute, making maximum marginal likelihood optimization of θ using Equation (2.2) infeasible. In fact, calculating the evidence in the general case is known to be an NP-hard problem (Koller and Friedman, 2009).

The evidence is linked to the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ through Bayes' theorem,

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} \ , \tag{2.5}$$

where the terms $p_\theta(\mathbf{x}|\mathbf{z})$ and $p_\theta(\mathbf{z})$ are known as the likelihood and prior, respectively. Due to this relationship between the evidence and posterior, maximizing the evidence $p_\theta(\mathbf{x})$ can generally be tackled by approximating the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$. For Bayesian models, the task of conditioning on the observations to obtain the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$, is known as *inference*.

2.2.1 Inference

Since the posterior is linked to the evidence through Bayes' rule, exact inference can quickly become intractable for more complex models, thus necessitating approximate inference techniques. A leading approximate inference method is Markov chain Monte Carlo (MCMC) (Hastings, 1970). MCMC constructs an ergodic Markov chain on $\mathbf{z}$ that has the posterior $p(\mathbf{z}|\mathbf{x})$ as its stationary distribution, although it can struggle to converge in high dimensions. One can then approximate expectations with respect to this posterior with empirical estimates calculated using samples from the chain. While MCMC algorithms are asymptotically exact, they are often unacceptably slow. In these settings, *variational inference* (VI) (Jordan *et al.*, 1999) provides a good alternative. Note that there are also other approximate inference techniques not discussed here, such as Laplace approximation, expectation propagation (Minka, 2001) and loopy belief propagation (Murphy *et al.*, 1999). We highlight VI, because it is one of the key application domains of NFs, and because it lies at the heart of VAEs.

Whereas MCMC takes a sampling approach, VI views inference as an optimization problem. Let $\mathcal{Q}$ represent a family of distributions over the latent variables. VI aims to find the member of this family, $q^* \in \mathcal{Q}$, that is closest to the true posterior in terms of the (reverse) KL-divergence:

$$q^*(\mathbf{z}) = \underset{q \in \mathcal{Q}}{\arg\min}\, \mathrm{KL}(q(\mathbf{z}) || \, p(\mathbf{z}|\mathbf{x})) \ . \tag{2.6}$$

The distributions in $\mathcal{Q}$ are parameterized by a set of free *variational parameters*. The optimization problem presented in (2.6) therefore corresponds to finding a setting for these parameters such that the approximation is as close as possible to the true posterior. In all cases of interest, we assume that we can not directly compute the optimization objective given by Equation (2.6), since the expression contains the intractable posterior distribution we are trying to approximate in the first place. Fortunately, the definition of the KL-divergence in (2.7) allows us to rewrite the expression in a form that provides us with an alternative, tractable, objective:

$$\begin{aligned} \mathrm{KL}(q(\mathbf{z}) || \, p(\mathbf{z}|\mathbf{x})) &= \mathbb{E}_{\mathbf{z} \sim q}[\log q(\mathbf{z}) - \log p(\mathbf{z}|\mathbf{x})] & (2.7) \\ &= \mathbb{E}_{\mathbf{z} \sim q}[\log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z})] + \log p(\mathbf{x}) \ . & (2.8) \end{aligned}$$

This change relies on the link between the posterior and the evidence through Bayes' theorem, with the evidence term, $\log p(\mathbf{x})$, being constant with respect to the choice of $q(\mathbf{z})$. By using only the (negated) expectation term in (2.8) as the objective, we obtain a quantity that is tractable to compute and which is only a constant shift from the true KL-divergence. This objective is known as the *evidence lower bound* (ELBO):

$$\mathrm{ELBO}(q) = \mathbb{E}_{\mathbf{z} \sim q}[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z})] \ . \tag{2.9}$$

By *maximizing* the ELBO, we thus implicitly *minimize* the KL-divergence between $q(\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$. As its name implies, the ELBO provides a lower bound on the log marginal likelihood of the data:

$$\log p(\mathbf{x}) = \mathrm{ELBO}(q) + \mathrm{KL}(q(\mathbf{z}) || p(\mathbf{z}|\mathbf{x})) \geq \mathrm{ELBO}(q) \ . \tag{2.10}$$

The last step above follows from the non-negativity of the KL-divergence. The more complex the variational family $\mathcal{Q}$, the more complex the above optimization problem. A common strategy for specifying $\mathcal{Q}$ is using a *mean-field* variational family. Such a family partitions the latent variables into mutually independent subsets, such that each is governed by a disctinct factor in the variational distribution. This allows one to use a coordinate ascent approach to optimize the ELBO. The resulting technique is called coordinate ascent variational inference (CAVI), which climbs the ELBO to a local optimum. A drawback of traditional CAVI is that it requires analytically deriving the updates that must be applied to the variational parameters at each iteration. Also, since one can have both latent variables that are local to specific datapoints and latent variables that are globally applicable to the entire dataset, one has to optimize the local variational parameters before one can update the global variational parameters. Traversing the entire dataset at each iteration in this manner can become prohibitively slow for large datasets. An alternative approach is stochastic gradient-based optimization which allows VI to scale to much larger datasets. In this setting, one can also employ automatic differentiation software to more easily compute the gradients needed for each update. This allows fitting more complex models, and is the main approach to inference employed by the models investigated in this work.

2.3 Normalizing Flows

Normalizing flows (NFs) (Rezende and Mohamed, 2015; Tabak and Turner, 2013) are a family of generative models that have tractable distributions in that they can perform density estimation and/or sampling exactly and efficiently. At its core, a flow consists of a simple base distribution and a differentiable bijective transformation that provides a mapping between this base distribution and the more complex model distribution. The change-of-variables formula (Murphy, 2012, p. 50) allows one to keep track of the change in density incurred by this transformation. NFs can be divided into two main classes: finite and continuous (or infinitesimal) flows. Finite flows (Tabak and Turner, 2013) create a complex bijective mapping by composing a *finite number* of simpler transformations. Continuous flows (Chen *et al.*, 2018), on the other hand, define the flow transformation in terms of an ordinary differential equation (ODE). Various techniques are employed to ensure that the flow computations remain tractable in both cases, and there is active research in expanding the classes of transformations that can be used effectively in flow models.

The rest of this section provides further details on finite and continuous flows. For finite flows, we discuss two of the main approaches to constructing tractable transformations, namely autoregressive and residual flows, and provide specific examples that employ these approaches. Next, we discuss the application of flows to the tasks of density estimation and sampling, as well as variational inference. We conclude the section by examining the invertibility of flows in practice. In these subsections, we address the topics and types of flows that are most relevant as background for subsequent chapters. Interested readers are referred to the surveys of Kobyzev *et al.* (2021) and Papamakarios *et al.* (2021) for a broader and more thorough discussion of normalizing flows.

2.3.1 Finite Flows

Let $\mathbf{x} \in \mathbb{R}^D$ be a random vector over which we wish to define a joint distribution. The main goal of flow-based modelling is to construct a function F that provides a mapping between $\mathbf{x}$ and a random vector $\boldsymbol{\epsilon}$ with a simpler known distribution, p_0. In order for F to be regarded as a valid flow transformation, it must be a diffeomorphism, i.e. F must be *invertible* and both F and F^{-1} must be *differentiable*. This requires that $\boldsymbol{\epsilon}$ be a D-dimensional vector as well. Let $F : \mathbb{R}^D \to \mathbb{R}^D$ be such a diffeomorphism, where

$$\mathbf{x} = F^{-1}(\boldsymbol{\epsilon}), \quad \text{and} \quad \boldsymbol{\epsilon} \sim p_0. \tag{2.11}$$

The log density of $\mathbf{x}$ is then given by the change-of-variables formula:

$$\log p(\mathbf{x}) = \log p_0(F(\mathbf{x})) + \log \left|\det\left(J_F(\mathbf{x})\right)\right| \ , \tag{2.12}$$

where $J_F(\mathbf{x})$ denotes the Jacobian of the transformation F at $\mathbf{x}$. In practice, F is typically implemented as a neural network and a common choice for the base distribution is a standard normal distribution, $p_0 = \mathcal{N}(\mathbf{0}, I_D)$. Section 2.3.3 discusses how Equation (2.12) is used during training and density estimation with a flow.

An immediate concern is that Equation (2.12) requires computing the *determinant* of the transformation's Jacobian—a computation that generally requires $\mathcal{O}(D^3)$ time for the $D \times D$ dimensional Jacobian,

$$J_F(\mathbf{x}) = \begin{bmatrix} \frac{\partial \epsilon_1}{\partial x_1} & \cdots & \frac{\partial \epsilon_1}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial \epsilon_D}{\partial x_1} & \cdots & \frac{\partial \epsilon_D}{\partial x_D} \end{bmatrix} . \tag{2.13}$$

This is intractable for all but the smallest values of D. Since training a flow requires evaluating Equation (2.12), additional steps need to be taken to ensure that the flow remains efficient, and more specifically, that $\left|\det\left(J_F(\mathbf{x})\right)\right|$ can be computed in $\mathcal{O}(D)$ (or similar) time.

One of the overarching approaches for achieving this is constructing a single NF as the composition of a series of simpler transformation steps. Such an NF is known as a *finite* NF. These individual transformations are chosen such that they comply with the diffeomorphism requirement and such that their Jacobian determinants can be computed efficiently. By composing multiple such transformations, a single complex flow can be constructed that is more expressive than any of its constituent parts. This construction is valid because the composition of invertible and differentiable functions, is itself an invertible and differentiable function. Let F_1 and F_2 be diffeomorphisms. The inverse and Jacobian determinant of their composition $F_2 \circ F_1$, is then given by

$$(F_2 \circ F_1)^{-1} = F_1^{-1} \circ F_2^{-1} \tag{2.14}$$

$$\det\left(J_{F_2 \circ F_1}(\mathbf{a})\right) = \det(J_{F_2}(F_1(\mathbf{a}))) \cdot \det(J_{F_1}(\mathbf{a})) \ . \tag{2.15}$$

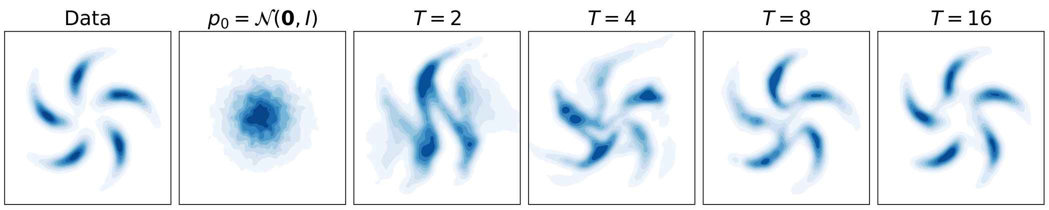


Figure 2.1: Increasing the number of flow transformation steps allows an NF to more closely match the true data distribution. The last four figures on the right correspond to samples $\mathbf{x} = (f_1^{-1} \circ \ldots f_T^{-1})(\boldsymbol{\epsilon})$ for $\boldsymbol{\epsilon}$ sampled from the standard normal base distribution p_0, and where the flows were constructed using 2, 4, 8 and 16 transformation steps, respectively. Note that each figure is a kernel density estimate plot of samples from the corresponding distribution.

As a result, the computation of the flow inverse and Jacobian determinant can be localized to the individual flow steps.

Revisiting Equation (2.12), we now let F be the composition of a series of simpler transformations $f_t : \mathbb{R}^D \to \mathbb{R}^D$, $t = 1, \ldots, T$. That is, we let $F(\mathbf{x}) = (f_T \circ \ldots \circ f_1)(\mathbf{x})$. The joint log density of $\mathbf{x}$ is then given by

$$\begin{aligned} \log p(\mathbf{x}) &= \log p_0(F(\mathbf{x})) + \log \left| \prod_{t=1}^{T} \det \left(J_{f_t}(\mathbf{x}^{(t-1)}) \right) \right| \\ &= \log p_0(F(\mathbf{x})) + \sum_{t=1}^{T} \log \left| \det \left(J_{f_t}(\mathbf{x}^{(t-1)}) \right) \right| \quad , \end{aligned} \tag{2.16}$$

where we introduce $\mathbf{x}^{(0)} = \mathbf{x}$ and $\mathbf{x}^{(t)} = f_t(\mathbf{x}^{(t-1)})$. Figure 2.1 provides an illustration of how composing more transformation steps can increase the expressiveness of an NF. The name *normalizing flow* is in reference to this procedure of letting a variable 'flow' through a series of transformations that 'normalizes' the complex data distribution to a simpler known base distribution. A flow that instead transforms samples from the base distribution to samples from the data distribution is known as a *generative flow*—for further details, see Section 2.3.3.2.[2]

Next, we discuss several of the main approaches to constructing suitable simpler transformations where evaluating Equation (2.16) remains tractable.

2.3.1.1 Autoregressive Flows

One of the first and most popular approaches developed for designing flow components is to constrain their Jacobians to be *triangular*. This can be achieved by enforcing an autoregressive transformation, i.e. ensuring that the

[2]'Normalizing flow' is still widely used in the literature, regardless of the direction of the transformation, but we keep this distinction for clarity in this work.

transformation applied to component i of the input is only a function of that component and all components with strictly lower indices. Recall that f_t is applied to $\mathbf{x}^{(t-1)}$ to obtain $\mathbf{x}^{(t)}$ at flow step t. For autoregressive flows, the i^{th} component of $\mathbf{x}^{(t)}$ satisfies

$$x_i^{(t)} = \left[f_t(\mathbf{x}^{(t-1)})\right]_i = g_{t,i}(x_i^{(t-1)}; c_{t,i}(\mathbf{x}_{<i}^{(t-1)})), \quad i = 1, \dots, D \tag{2.17}$$

where $\mathbf{x}_{<i} = \mathbf{x}_{1:i-1}$ denotes all components of $\mathbf{x}$ with an index less than i. The invertible and differentiable function $g_{t,i}$ is known as the *transformer*[3] or *normalizer*, and is partially parameterized by the differentiable *conditioner* function, $c_{t,i}$. Note that the transformation of the i^{th} component depends only on components with strictly lower indices. Note that it is possible to compute all D components of $\mathbf{x}^{(t)}$ in parallel.

The inverse of the transformation step is given by

$$x_i^{(t-1)} = \left[f_t^{-1}(\mathbf{x}^{(t)})\right]_i = g_{t,i}^{-1}(x_i^{(t)}; c_{t,i}(\mathbf{x}_{<i}^{(t-1)})), \quad i = 1, \dots, D\ . \tag{2.18}$$

Computing $x_i^{(t-1)}$ here requires evaluating the conditioner $c_{t,i}(\mathbf{x}_{<i}^{(t-1)})$, which in turn requires the inverted values of all components with indices smaller than i. This necessitates inverting the components sequentially, which makes inversion less efficient than the forward mapping which can be parallelized. Note that although this setup requires the transformers $g_{t,i}$ to be invertible in order to invert f_t, it fortunately does not place the same restriction on the conditioner functions $c_{t,i}$. They can thus be made arbitrarily complex, which aids the overall expressiveness of the flow.

For these autoregressive transformations, the Jacobian is triangular. Since $x_i^{(t)}$ does not depend on $x_j^{(t-1)}$ for any $j > i$, the partial derivative of $x_i^{(t)}$ with respect to $x_j^{(t-1)}$ is zero. This means that the required Jacobian determinants in Equation (2.16) can be computed in linear time as the product of the matrices' diagonal terms:

$$\begin{aligned}
\log\left|\det\left(J_{f_t}(\mathbf{x}^{(t-1)})\right)\right| &= \log\left|\det\left(\begin{bmatrix} \frac{\partial\left[f_t(\mathbf{x}^{(t-1)})\right]_1}{\partial x_1^{(t-1)}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\partial\left[f_t(\mathbf{x}^{(t-1)})\right]_D}{\partial x_1^{(t-1)}} & \cdots & \frac{\partial\left[f_t(\mathbf{x}^{(t-1)})\right]_D}{\partial x_D^{(t-1)}} \end{bmatrix}\right)\right| \\
&= \log\left|\prod \frac{\partial\left[f_t(\mathbf{x}^{(t-1)})\right]_i}{\partial x_i^{(t-1)}}\right| \\
&= \sum_{i=1}^{D} \log\left|\frac{\partial\left[f_t(\mathbf{x}^{(t-1)})\right]_i}{\partial x_i^{(t-1)}}\right|\ .
\end{aligned} \tag{2.19}$$

[3]The term 'transformer' in this context should not be confused with the well-known transformer architecture used in natural language processing.

The partial derivatives in this expression can be computed either analytically or by making use of automatic differentiation, depending on the type of transformer and conditioner used. Note that the off-diagonal Jacobian entries need not be computed.

Next, we discuss specific examples of transformer functions as well as a general approach for constructing conditoner functions such that the correct autoregressive structure is obtained.

Affine Transformer

One of the simplest examples of an autoregressive flow defined by a transformer-conditioner pair, is the *affine autoregressive flow*:

$$[f_t(\mathbf{x}^{(t-1)})]_i = \exp(s_{t,i}(\mathbf{x}_{<i}^{(t-1)})) \cdot x_i^{(t-1)} + m_{t,i}(\mathbf{x}_{<i}^{(t-1)}) \ , \tag{2.20}$$

where $c_{t,i}(\mathbf{x}_{<i}^{(t-1)}) = \left[s_{t,i}(\mathbf{x}_{<i}^{(t-1)}), m_{t,i}(\mathbf{x}_{<i}^{(t-1)})\right]$ denotes the conditioner functions of the transformation. Note that Equation (2.20) is invertible and that the inverse does not require computing the inverse of either $s_{t,i}$ or $m_{t,i}$:

$$[f_t^{-1}(\mathbf{x}^{(t)})]_i = \frac{x_i^{(t)} - m_{t,i}(\mathbf{x}_{<i}^{(t-1)})}{\exp(s_{t,i}(\mathbf{x}_{<i}^{(t-1)}))} \ . \tag{2.21}$$

Furthermore, the partial derivative of $\left[f_t(\mathbf{x}^{(t-1)})\right]_i$ with respect to $x_i^{(t-1)}$ is simply $\exp(s_{t,i}(\mathbf{x}_{<i}^{(t-1)}))$, so that

$$\log\left|\det\left(J_{f_t}(\mathbf{x}^{(t-1)})\right)\right| = \sum_{i=1}^{D} \log\left|\exp(s_{t,i}(\mathbf{x}_{<i}^{(t-1)}))\right| = \sum_{i=1}^{D} s_{t,i}(\mathbf{x}_{<i}^{(t-1)}) \ . \tag{2.22}$$

Examples of popular models that make use of affine transformations similar to Equation (2.20) include NICE (Dinh *et al.*, 2015), Real NVP (Dinh *et al.*, 2017), masked autoregressive flows (MAF) (Papamakarios *et al.*, 2017) and inverse autoregressive flows (IAF) (Kingma *et al.*, 2016). Note that NICE and Real NVP make use of an alternative *coupling* structure, where only one half of the variables are updated at each flow step conditioned on the other half, which remains constant. This allows for more efficient flow inversion.

Integration-based Transformer

A definite integral of a strictly positive function, is a monotonically increasing function. Wehenkel and Louppe (2019) proposed a transformer based on this observation, with component i of its output given by

$$[f(\mathbf{x})]_i = \int_0^{x_i} h_i(t; c_i(\mathbf{x}_{<i}))\, dt + \beta_i(c_i(\mathbf{x}_{<i})) \ , \tag{2.23}$$

where c_i is a conditioner function (similar to s and m used in the affine transformer), and h_i and β_i are two neural networks with a strictly positive and a real scalar output, respectively. A drawback of this approach is that without constraining $h_i(\cdot; \mathrm{c}_i(\mathbf{x}))$, the integral typically does not have an analytical solution, and thus requires additional numerical methods to solve the given problem. One example of how $h_i(\cdot; c_i(\mathbf{x}))$ could be constrained, is to define it as a nonnegative polynomial, in which case the integral can be computed analytically (Jaini *et al.*, 2019). As an upside on the other hand, the required diagonal entries of the Jacobian are simply given by $h_i(x_i; c_i(\mathbf{x}_{<i}))$ for $i = 1, \ldots, D$.

We discussed affine and integration-based transformers in detail here, because they will be revisited in following chapters. Nevertheless, there are many other types of autoregressive flows. For example, the neural autoregressive flow of Huang *et al.* (2018*a*) applies non-linear transformations by constructing the transformer as a multi-layer perceptron (MLP), with weights and biases determined by the conditioner function. Additional restrictions are placed on the weights and activation functions of the MLP to ensure invertibility. Alternatively, the transformer can be implemented as a piecewise function of simpler segments that are each easy to invert, known as a spline flow (Durkan *et al.*, 2019; Conor Durkan and Papamakarios, 2019).

Constructing the Conditioners

Since the conditioners $c_{t,i}(\cdot)$ do not need to be invertible, they could in principle be implemented as separate and arbitrarily complex neural networks. Training $T \times D$ separate networks can become prohibitively expensive, however. A general approach to overcome this is to share parameters between these conditioners.

One of the most popular approaches, based on *masking*, uses a single network for each timestep t that takes in $\mathbf{x}$ and outputs $(c_{t,1}, \ldots, c_{t,D})$ in a single forward pass. Note that each $c_{t,i}$ could also be a vector, consisting of, for example, both the scaling and translation factors used in the affine transformation of Equation (2.20). The correct autoregressive structure is maintained by removing edges in the network so that there is no path between x_j and $c_{t,i}$ for $j \geq i$. Practically, these edges are removed by multiplying the network's weight matrices by binary masking matrices. These binary masks effectively switch connections between units of the network on and off as per the desired autoregressive structure.

Germain *et al.* (2015) provide a general procedure, referred to as the MADE masking scheme, for constructing masks for neural networks with an arbitrary number of hidden layers and hidden layer widths. The basic idea is to assign to each unit n in the network a label $d_n \in \{1, \ldots, D\}$. Specifically, each input x_i and output $c_{t,i}$ are assigned $d_n = i$ and hidden units are uniformly randomly

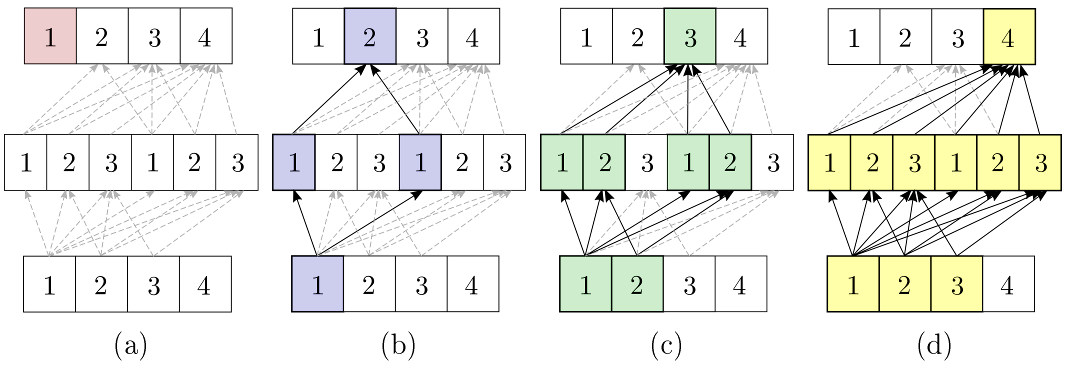


Figure 2.2: The MADE masking scheme. Each of (a)–(d) highlights the nodes and edges in the subnetwork predicting the conditioners c_1 to c_4, respectively. The number shown in each block corresponds to the unit's label. These assignments are used to construct the necessary masks such that the remaining edges (as shown) force output i to be a function of only the input $\mathbf{x}_{<i}$. The edges removed by the masks, as well as the bias terms, are not shown to avoid cluttering.

assigned labels in $\{1, \ldots, D-1\}$. An edge is then retained between a unit n in a hidden layer and a unit n' in the previous layer (either the input layer or another hidden layer) only if $d_n \geq d_{n'}$. An edge between a unit n' in the last hidden layer and a unit n in the output layer is retained only if $d_n > d_{n'}$. In this way, the output labelled $d_n = i$ will be a function only of inputs $\mathbf{x}_{<i}$. Figure 2.2 provides an illustration of this masking scheme when $\mathbf{x} \in \mathbb{R}^4$. We next discuss an alternative type of finite flow known as a residual flow.

2.3.1.2 Residual Flows

Residual flows apply the following general transformation at flow step t:

$$f_t(\mathbf{x}^{(t-1)}) = \mathbf{x}^{(t-1)} + g_t(\mathbf{x}^{(t-1)}). \tag{2.24}$$

Such a transformation is of course not necessarily invertible, and computing the Jacobian determinant is likewise not necessarily efficient. Residual flows thus require constraints on g_t to ensure invertibility of the transformation and tractability of the Jacobian determinant computation. Planar and radial flows (Rezende and Mohamed, 2015) make use of the *matrix determinant lemma* (Murphy, 2012, Corollary 4.3.1) to ensure tractable computation of the Jacobian determinant and are invertible under certain conditions. Sylvester flows (van den Berg *et al.*, 2018) generalize planar flows by instead making use of Sylvester's determinant identity. Below we provide a more in-depth discussion of an alternative type of residual flow considered in more detail in this thesis, that ensures invertibility by *bounding the Lipschitz constant* of the flow transformation.

Contractive Residual Flows

Building on the observation that the traditional residual network (ResNet) (He *et al.*, 2016) can be viewed as an Euler discretization of an ordinary differential equation (ODE), Behrmann *et al.* (2019) show that a finite NF can be constructed by changing the normalization scheme of the ResNet's weights. Consider a ResNet $F(\mathbf{x}) = (f_T \circ \ldots \circ f_1)(\mathbf{x})$, composed of blocks

$$\mathbf{x}^{(t)} := f_t(\mathbf{x}^{(t-1)}) = \mathbf{x}^{(t-1)} + g_t(\mathbf{x}^{(t-1)}) \ , \tag{2.25}$$

with $\mathbf{x}^{(0)} = \mathbf{x}$ and residual blocks $g_t(\mathbf{x}^{(t-1)})$ implemented as neural networks. This formulation is identical to Equation (2.24)—we only require each block f_t to be invertible and differentiable for $F(\cdot)$ to be regarded as a valid finite NF. Contractive residual flows satisfy this requirement by noting that a sufficient condition for invertibility is that each residual block g_t is *contractive* (Behrmann *et al.*, 2019), which is a special case of being *Lipschitz continuous*.

Definition 6 (Lipschitz Continuity (O'Searcoid, 2006))**.** *Consider a metric space (X, d) and the function $f : X \to X$. The function f is* Lipschitz continuous *if there exists a constant $k \geq 0$ such that for all $x, y \in X$,*

$$d(f(x), f(y)) \leq k \cdot d(x, y) \ . \tag{2.26}$$

The value of k is known as the Lipschitz constant *of f.*

Note that we have assumed in the above definition that the function f has the same domain and codomain. Lipschitz continuity can, however, be defined more generally for mappings between *different* metric spaces. Since we are only considering flows, $F : \mathbb{R}^D \to \mathbb{R}^D$, the above definition is sufficient. We will denote the smallest value of k for which (2.26) holds, by $\text{Lip}(f)$.

Definition 7 (Contraction (O'Searcoid, 2006))**.** *A function $f : X \to X$ is a contraction if f is Lipschitz continuous with $Lip(f) < 1$.*

A residual flow F is thus invertible, if $\text{Lip}(g_t) < 1$ for $t = 1, \ldots, T$. Let a residual block g be the composition of L functions: $g = \phi_L \circ \ldots \circ \phi_1$. If g is a neural network, this composition will typically be a combination of affine transformations and non-linear activation functions. To reach our goal of ensuring $\text{Lip}(g) < 1$, we can make use of the following property of the Lipschitz constant of a composition of functions (for a derivation, see Appendix B.2.1):

$$\text{Lip}(f_2 \circ f_1) \leq \text{Lip}(f_2)\,\text{Lip}(f_1) \ . \tag{2.27}$$

Thus, we have that

$$\text{Lip}(g) \leq \prod_{i=1}^{L} \text{Lip}(\phi_i) \ , \tag{2.28}$$

and we can ensure that $\mathrm{Lip}(g) < 1$ by ensuring that all $\mathrm{Lip}(\phi_i) \leq 1$ and for at least one ϕ_i, $\mathrm{Lip}(\phi_i) < 1$. The Lipschitz constant of a fully-connected affine layer $\phi_i(\mathbf{x}) = W_i\mathbf{x} + \mathbf{b}_i$, under the Euclidean norm, is the spectral norm[4] of the weight matrix W_i (Miyato *et al.*, 2018), denoted by $||W_i||_s$, which can be computed using the power method (Von Mises and Pollaczeck-Geiringer, 1929; Gouk *et al.*, 2021). As a result, we can ensure invertibility of the flow by implementing g as a composition of (1) activations h with $\mathrm{Lip}(h) \leq 1$, such as tanh or LipSwish (Chen *et al.*, 2019), and (2) affine layers with weight matrices W_i satisfying $||W_i||_s < 1$. The spectral norm of each residual block's weight matrices can be constrained to not exceed a desired scaling coefficient c by normalizing each W_i as follows:

$$\widetilde{W_i} = \min(c, ||W_i||_s) \cdot \frac{W_i}{||W_i||_s} \ . \tag{2.29}$$

Apart from ensuring invertibility of F, one must also be able to compute its Jacobian determinant in a tractable way. Behrmann *et al.* (2019) showed that the log Jacobian determinant of a block f can be expressed in terms of the trace of the matrix logarithm, and use this to estimate $\log|\det(J_f(\mathbf{x}))|$ with a truncated power series:

$$\log|\det(J_f(\mathbf{x}))| = \mathrm{tr}(\log(I + J_g(\mathbf{x}))) \tag{2.30}$$

$$\approx \sum_{k=1}^{n} (-1)^{k+1} \frac{\mathrm{tr}(J_g^k)}{k} \ . \tag{2.31}$$

The first line follows from the observation that Lipschitz-constrained perturbations of the identity, $\mathbf{x}+g(\mathbf{x})$, will yield positive determinants (Behrmann *et al.*, 2019, Lemma 6) and then from applying the matrix identity, $\log\det(A) = \mathrm{tr}(\log(A))$, for non-singular $A \in \mathbb{R}^{D\times D}$ (Withers and Nadarajah, 2010). This trace can be expressed as an infinite power series, which is truncated to provide the (biased) estimate given in Equation (2.31). Each trace $\mathrm{tr}(J_g^k)$ is further estimated using the Hutchinsons trace estimator (Hutchinson, 1990), which constructs a Monte Carlo approximation of the trace of a matrix $A \in \mathbb{R}^{D\times D}$ via

$$\mathrm{tr}(A) = \mathbb{E}_{p(\mathbf{v})}\left[\mathbf{v}^T A \mathbf{v}\right] \approx \frac{1}{N} \sum_{\mathbf{v}\sim p(\mathbf{v})} \mathbf{v}^T A \mathbf{v} \ , \tag{2.32}$$

where the distribution $p(\mathbf{v})$ must satisfy $\mathbb{E}[\mathbf{v}] = \mathbf{0}$ and $\mathrm{Cov}(\mathbf{v}) = I_D$, such as the standard normal distribution, $\mathcal{N}(\mathbf{0}, I_D)$.

The bias of the estimator in Equation (2.31) grows, however, with both the dimensionality of $\mathbf{x}$ and the Lipschitz constant of g (Chen *et al.*, 2019). To

[4]The spectral norm of a matrix A is given by its largest singular value. It also corresponds to the induced matrix norm of A in the special case that the Euclidean norm for vectors is used.

overcome this, Chen *et al.* (2019) propose to directly sample the number of terms evaluated in Equation (2.31), that is, they sample $n \sim p(N)$. The only constraint on the choice of $p(N)$ is that it must have support over all of the indices. By appropriately reweighting the terms of the series, one obtains an unbiased estimator known as the "Russian roulette" estimator. While this approach resolves the bias of the estimator, it has the drawback of requiring unpredictable memory usage and run time as the number of terms used varies.

Since contractive residual flows are not analytically invertible, Behrmann *et al.* (2019) propose using a fixed-point iteration method to invert each block numerically: to compute $\mathbf{x} = f_t^{-1}(\mathbf{y})$, initialize $\mathbf{x}^{(0)} = \mathbf{y}$ and then apply the following update until convergence:

$$\mathbf{x}^{(n+1)} = \mathbf{y} - g_t(\mathbf{x}^{(n)}) \ . \tag{2.33}$$

According to the Banach fixed-point theorem, this method has a unique fixed-point (Kreyszig, 1989, Theorem 5.1.2), and thus any starting value could in fact be used. Using $\mathbf{x}^{(0)} = \mathbf{y}$ is typically a good choice, however, since $\mathbf{y}$ is obtained from $\mathbf{x}$ only via a bounded perturbation of the identity.

Next, we consider continuous flows which define the flow transformation implicitly in terms of an ODE, instead of using T explicit transformation steps.

2.3.2 Continuous Flows

Finite flows construct complex mappings by composing multiple simpler transformation steps. Continuous flows emerged from considering what would happen if more and more transformations were composed and the step-size of each transformation was reduced. When this process is taken to the limit, the discrete dynamics of the finite transformation steps can be replaced by an ordinary differential equation (ODE) describing the continuous-time dynamics of the flow transformation (Chen *et al.*, 2018):

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t) \ , \tag{2.34}$$

where the ODE is specified by the neural network f. Given data $\mathbf{x}$, the normalizing operation is achieved by solving the initial value problem, $\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t)$ with $\mathbf{x}(t_1) = \mathbf{x}$, to obtain the state of the variables at time t_0:

$$\mathbf{x}(t_0) = \mathbf{x}(t_1) - \int_{t_0}^{t_1} f(\mathbf{x}(t), t)\, dt \ . \tag{2.35}$$

Here, $\mathbf{x}(t_0)$ corresponds to a sample from the base distribution p_0. This initial value problem can be solved by making use of a black-box differential equation solver. The resulting model is known as a *continuous normalizing flow.* For continuous flows, however, the log-density of $\mathbf{x}$ can no longer be computed using the standard change-of-variables formula given by Equation (2.12). Instead, the change in log-density arising from the flow follows a second differential equation,

$$\frac{d \log p(\mathbf{x}(t))}{dt} = -\operatorname{tr}\left(J_{f(\cdot,t)}(\mathbf{x}(t))\right) \quad , \tag{2.36}$$

known as the *instantaneous change-of-variables* formula (Chen *et al.*, 2018). By integrating over time to solve a second initial value problem, we can compute the log-density of a data sample $\mathbf{x}$ under the model:

$$\log p(\mathbf{x}(t_1)) = \log p_0(\mathbf{x}(t_0)) - \int_{t_0}^{t_1} \operatorname{tr}\left(J_{f(\cdot,t)}(\mathbf{x}(t))\right) \, dt \quad . \tag{2.37}$$

The above two differential equations (2.34 and 2.36) can be combined to give the following initial value problem, which can be solved by integrating backwards in time from t_1 to t_0:

$$\underbrace{\begin{bmatrix} \mathbf{x}(t_0) \\ \log p(\mathbf{x}(t_1)) - \log p_0(\mathbf{x}(t_0)) \end{bmatrix}}_{\text{solutions}} = \underbrace{\begin{bmatrix} \mathbf{x}(t_1) \\ 0 \end{bmatrix}}_{\substack{\text{initial values} \\ \text{at } t_1}} + \underbrace{\int_{t_1}^{t_0} \begin{bmatrix} f(\mathbf{x}(t), t) \\ \operatorname{tr}\left(J_{f(\cdot,t)}(\mathbf{x}(t))\right) \end{bmatrix} dt}_{\text{dynamics}}, \tag{2.38}$$

The solution consists of the data's corresponding sample from the base distribution as well as the change in log-density incurred by the flow transformation. This solution exists and is unique only if f and its first derivatives are Lipschitz continuous (Khalil, 2002). This requirement can be satisfied by using only smooth Lipschitz activation functions to implement f. In practice, we set $t_0 = 0$ and $t_1 = 1$.

Unlike the naïve $O(D^3)$ computational cost for computing the Jacobian determinant of finite NFs, continuous flows have a $O(D^2)$ time complexity for computing $\operatorname{tr}\left(J_{f(\cdot,t)}(\mathbf{x}(t))\right)$, along with the additional cost introduced by the numerical ODE solver. The cost of computing $\log p(\mathbf{x}(t_1))$ can be further reduced by making use of an unbiased estimator of the Jacobian trace (Grathwohl *et al.*, 2019). Similar to the approach used for residual flows, the trace can be estimated using the Hutchinson's trace estimator given in (2.32):

$$\begin{aligned} \frac{d \log p(\mathbf{x}(t))}{dt} &= -\mathbb{E}_{p(\mathbf{v})}\left[\mathbf{v}^T J_{f(\cdot,t)}(\mathbf{x}(t))\mathbf{v}\right] \\ &\approx -\frac{1}{N}\sum_{i=1}^{N} \mathbf{v}_i^T J_{f(\cdot,t)}(\mathbf{x}(t))\mathbf{v}_i \quad \text{where} \quad \mathbf{v}_i \sim p(\mathbf{v}). \end{aligned} \tag{2.39}$$

The vector-Jacobian product, $\mathbf{v}_i^T J_{f(\cdot,t)}(\mathbf{x}(t))$, can be computed efficiently using automatic differentiation software. By using a single Monte Carlo sample, i.e. setting $N = 1$, one can obtain an unbiased stochastic estimator of the trace with $O(D)$ cost.

Note that unlike finite flows, no restrictions have been placed on the structure or Lipschitzivity of the neural network f. These models therefore have a 'free-form' Jacobian and tractability is maintained by making use of the estimator in (2.39). Furthermore, the flows can be inverted by simply performing the integration in the opposite direction. This has essentially the same computational cost as a forward pass through the flow.

2.3.3 Primary Applications of NFs

Given a trained flow and the associated change-of-variables formula, one can in principle perform two basic operations: calculate the density of a sample and generate new samples. As such, flows can be applied in a wide variety of settings where a model is required to perform either one or both of these operations. A typical form of application of an NF is for probabilistic modelling of data, allowing the user to perform density estimation for observations (Papamakarios *et al.*, 2017; Grathwohl *et al.*, 2019) as well as to generate new data points similar to the original data the flow was trained on. Flows have been used to generate data such as images (Dinh *et al.*, 2015; Grathwohl *et al.*, 2019; Kingma and Dhariwal, 2018), video (Kumar *et al.*, 2019), audio (van den Oord *et al.*, 2016) and text (Tran *et al.*, 2019). Whereas the above application focuses on modelling observed data and capturing its underlying distribution, flows can also be applied in the context of inference (Rezende and Mohamed, 2015) where one wishes to reason about unknown or hidden variables, given the observed data. Below we discuss these two applications of flows in more detail.

2.3.3.1 Density Estimation & Sampling

Let $p_\theta(\mathbf{x})$ denote the distribution modelled by an NF with the bijective transformation $F_{\mathbf{x}\to\boldsymbol{\epsilon}}$. Here, θ denotes the parameters of the flow and the subscript $\mathbf{x} \to \boldsymbol{\epsilon}$ indicates that a forward pass through the flow is in the normalizing direction. Given a finite number of i.i.d. samples $\{\mathbf{x}_n\}_{n=1}^N$, we showed in Section 2.1.1 that the parameters of this model can be optimized by maximizing the likelihood:

$$\begin{aligned}\sum_{n=1}^{N} \log p_\theta(\mathbf{x}_n) = \sum_{n=1}^{N} &\left[\log p_0(F_{\mathbf{x}\to\boldsymbol{\epsilon}}(\mathbf{x}_n;\theta))\right. \\ &\left. + \log\left|\det\left(J_{F_{\mathbf{x}\to\boldsymbol{\epsilon}}}(\mathbf{x}_n;\theta)\right)\right|\right] \ .\end{aligned} \tag{2.40}$$

For continuous flows one would similarly use the instantaneous change-of-variables formula given by Equation (2.37). Since the flow is typically implemented using neural networks, this objective can be optimized by making use of stochastic gradient descent and automatic differentiation libraries.

The trained flow can be used to calculate the density, under p_θ, of new points not seen during training, by using the change-of-variables formula (2.12). Furthermore, by inverting $F_{\mathbf{x}\to\boldsymbol{\epsilon}}$ one is able to generate new samples with similar properties to the data the flow was trained on:

$$\mathbf{x}' = F^{-1}_{\mathbf{x}\to\boldsymbol{\epsilon}}(\boldsymbol{\epsilon}), \quad \text{where} \quad \boldsymbol{\epsilon} \sim p_0 \ . \tag{2.41}$$

2.3.3.2 Variational Inference

NFs can be employed as the variational family in variational inference (Rezende and Mohamed, 2015; van den Berg *et al.*, 2018; Kingma *et al.*, 2016) as discussed in Section 2.2.1. Given a model $p(\mathbf{x}, \mathbf{z})$, where $\mathbf{x}$ is observed and $\mathbf{z}$ is latent, we wish to learn a distribution $q(\mathbf{z}|\mathbf{x})$ that provides a good approximation to the true posterior $p(\mathbf{z}|\mathbf{x})$. Assume we have access to i.i.d. samples $\{\mathbf{x}_n\}_{n=1}^N$ and are able to evaluate $p(\mathbf{x}, \mathbf{z})$ for a given pair of observed and latent variables. Also, let $\mathcal{Q}$ represent all the conditional distributions $q_\phi(\mathbf{z}|\mathbf{x})$ that can be modelled by a flow $F_{\boldsymbol{\epsilon}\to\mathbf{z}}$, depending on the setting of its parameters, ϕ. Here, the subscript $\boldsymbol{\epsilon} \to \mathbf{z}$ indicates that a forward pass through the flow is in the generative direction. Variational inference amounts to finding the setting of these parameters ϕ such that $q_\phi(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}$ is as close as possible to the true posterior, $p(\mathbf{z}|\mathbf{x})$.

The parameters ϕ can be optimized by minimizing the KL-divergence of $q_\phi(\mathbf{z}|\mathbf{x})$ from the true posterior:

$$\begin{aligned}\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||\, p(\mathbf{z}|\mathbf{x})) &= \mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z}|\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}, \mathbf{z})\right] + \log p(\mathbf{x}) \ .\end{aligned} \tag{2.42}$$

Since $\log p(\mathbf{x})$ is constant with respect to ϕ, the divergence in (2.42) can be minimized by minimizing a Monte Carlo estimate of the expectation:

$$\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||\, p(\mathbf{z}|\mathbf{x})) \approx \frac{1}{N}\sum_{n=1}^{N}[\log q_\phi(\mathbf{z}_n|\mathbf{x}_n) - \log p(\mathbf{x}_n, \mathbf{z}_n)] + \text{const} \ . \tag{2.43}$$

Minimizing the expectation in (2.42) is equivalent to maximizing the ELBO, $\mathbb{E}_{\mathbf{z}\sim q}\left[\log p(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]$, as discussed in Section 2.2.1. Each $\mathbf{z}_n$ in (2.43) is sampled from q_ϕ by first sampling $\boldsymbol{\epsilon}_n \sim p_0$ and then passing $\boldsymbol{\epsilon}_n$ through $F_{\boldsymbol{\epsilon}\to\mathbf{z}}$ while conditioning on $\mathbf{x}_n$. There are several ways in which a flow can be conditioned on an observation. Some directly map $\mathbf{x}_n$ to the parameters of the flow transformation, e.g. the weight and biases of a neural network (Rezende and Mohamed, 2015). Others pass $\mathbf{x}_n$, along with $\mathbf{z}_n$, as the input to the flow (Wehenkel and Louppe, 2021). For autoregressive flows, each $z_{n,i}$ would then not only be dependent on $\mathbf{z}_{n,<i}$ but also on $\mathbf{x}_n$. The density of $\mathbf{z}_n$ is then given by

$$\log q_\phi(\mathbf{z}_n|\mathbf{x}_n) = \log p_0(\boldsymbol{\epsilon}_n) - \log\left|\det\left(J_{F_{\boldsymbol{\epsilon}\to\mathbf{z}}}(\boldsymbol{\epsilon}_n; \mathbf{x}_n, \phi)\right)\right| \ , \tag{2.44}$$

where we made use of the identity

$$\left|\det(J_{F^{-1}_{\boldsymbol{\epsilon}\to\mathbf{z}}}(\mathbf{z}))\right| = \left|\det(J_{F_{\boldsymbol{\epsilon}\to\mathbf{z}}}(\boldsymbol{\epsilon}))\right|^{-1} \quad . \tag{2.45}$$

This identity follows from the inverse function theorem and a property of the determinant of an inverse of an invertible matrix: $\det(A^{-1}) = \det(A)^{-1}$.

Unlike CAVI, which optimizes ϕ on a per-datapoint basis, only a single flow function is learned for the given training data and used for inference on new data points. This process is therefore known as *amortized* variational inference (Gershman and Goodman, 2014), since the flow can be used to infer the latent state for new $\mathbf{x}$ not seen during training. The resulting flow is also known as an *amortization artifact*.

2.3.4 Invertibility of Flows in Practice

All normalizing flows are invertible by definition. When an analytical inverse is not available, one can invert a flow with numerical methods, such as bisection search or fixed-point iteration methods. Recent work has shown that theoretical invertibility does not always imply stable inversion in practice, however (Behrmann *et al.*, 2021). Indeed, depending on the model choices, inversion can lead to exploding inverses or numerical errors. If this occurs, the flow has become numerically non-invertible and violates one of the core assumptions needed to use it practically for exact density calculation. Based on the discussion in the previous section, stable inversion is especially important when using either $F_{\mathbf{x}\to\boldsymbol{\epsilon}}$ to generate $\mathbf{x}$ for a given $\boldsymbol{\epsilon} \sim p_0$, or when using $F_{\boldsymbol{\epsilon}\to\mathbf{z}}$ to compute the density for a $\mathbf{z}$ not generated by the flow.

One of the key aspects that should be considered when analysing a flow's inversion stability, is its bi-Lipschitz properties. This pertains to the *Lipschitz continuity* (Definition 6) of both the forward and inverse flow transformation.

Definition 8 (Bi-Lipschitz Continuous)**.** *Consider a metric space (X, d) and the Lipschitz continuous function $f : X \to X$. If an inverse $f^{-1} : X \to X$ and a constant $k' \geq 0$ exists such that for all $x, y \in X$,*

$$d(f^{-1}(x), f^{-1}(y)) \leq k' \cdot d(x, y) \quad , \tag{2.46}$$

then f is called bi-Lipschitz continuous.

Lipschitz and bi-Lipschitz continuity refers, however, only to the *global* Lipschitz properties of a flows—for certain transformations it is only possible to provide local Lipschitz bounds.

Definition 9 (Local Lipschitz Continuity)**.** *Consider a metric space (X, d), a function $f : X \to X$ and a closed ball $Y = \{x \mid d(x, x_0) \leq r\}$. The function f is* *called* locally Lipschitz continuous *about* x_0 *if there exists some constant* $k \geq 0$ *such that for every* $x, y \in Y$*:*

$$d(f(x), f(y)) \leq k \cdot d(x, y) \ . \tag{2.47}$$

To see why the Lipschitz constant of a flow F affects the numerical stability of inversion, we consider in more detail the role of imprecise floating-point computations in modern hardware (Behrmann *et al.*, 2021). Let $F(x) = y$ be the exact flow transformation and let $F_{\delta_1}(x) = y + \delta_1 := y_{\delta_1}$ be the imprecise floating-point calculation of the function value in practice with error δ_1. Assume for now that the inverse, $F^{-1}(y_{\delta_1}) = x_{\delta_1}$, can be computed exactly. The existence of a Lipschitz constant for the inverse mapping places an upper bound on the error of the reconstruction:

$$\begin{aligned} ||x - x_{\delta_1}||_2 &= d(F^{-1}(y), F^{-1}(y_{\delta_1})) \\ &\leq \text{Lip}(F^{-1}) \cdot d(y, y_{\delta_1}) \\ &= \text{Lip}(F^{-1}) \cdot ||y - (y + \delta_1)||_2 \\ &= \text{Lip}(F^{-1}) \cdot ||\delta_1||_2 \end{aligned} \tag{2.48}$$

where the distance metric used is the Euclidean norm. Naturally, the inverse transformation will also introduce numerical errors due to inexact floating-point calculations. So, let $F_{\delta_2}^{-1}(y_{\delta_1}) = x_{\delta_1} + \delta_2$. The reconstruction error is then bounded as follows:

$$\begin{aligned} ||x - (x_{\delta_1} + \delta_2)||_2 &\leq ||x - x_{\delta_1}||_2 + ||\delta_2||_2 \\ &\leq \text{Lip}(F^{-1}) \cdot ||\delta_1||_2 + ||\delta_2||_2 \ . \end{aligned} \tag{2.49}$$

These bounds provide an indication of the role that a flow's Lipschitz constant can have on inversion stability. Although a flow might be theoretically invertible, numerical errors are always present. The Lipschitz constant places a bound on how much these numerical errors can be amplified when passing values through the flow or its inverse. Bounds on these constants therefore play an important role in understanding and mitigating possible exploding inverses.

Having introduced NFs, we next consider an alternative deep generative model known as a variational autoencoder, which unlike NFs, does not require the latent space to have the same dimension as the observations.

2.4 Variational Autoencoders

Variational autoencoders (VAEs) (Kingma and Welling, 2014; Rezende *et al.*, 2014) are one of the most popular families of deep generative models to have emerged in recent years. The VAE framework provides a principled approach to using stochastic gradient descent to simultaneously learn both a deep latent variable model and a corresponding inference model. While the deep latent variable model aims to capture the generating process underlying the observed data, the inference network is encouraged to provide a good approximation to the generative model's posterior distribution over the latent variables. The following discussion of VAEs is inspired by the comprehensive introduction to the topic by Kingma and Welling (2019). Interested readers are referred to this work for further details and extensions of VAEs.

2.4.1 Simultaneously Learning a Model and Approximate Posterior

Let $p_\theta(\mathbf{x}, \mathbf{z})$ be a latent variable model over observed variables $\mathbf{x}$ and latent variables $\mathbf{z}$, with free parameters θ. If this distribution is parameterized by neural networks, i.e. θ consists of a network's weights and biases, then the resulting model is known as a *deep* latent variable model (DLVM). As discussed in Section 2.2, optimizing θ using maximum likelihood estimation is intractable in this setting because one typically cannot easily compute the marginal distribution, $p_\theta(\mathbf{x})$. Therefore, based on the relationship between the evidence and posterior distributions as captured by Bayes' theorem (2.5), maximizing the evidence can generally be tackled by approximating the posterior distribution.

As discussed in Section 2.2.1, variational inference attempts to approximate the true (intractable) posterior of a known model by finding the member of a family of approximate distributions over the latent variables, $q^* \in \mathcal{Q}$, that minimizes the reverse KL divergence to the exact posterior:

$$q^*(\mathbf{z}|\mathbf{x}) = \underset{q(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}}{\operatorname{argmin}} \operatorname{KL}\left(q(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})\right) \quad . \tag{2.50}$$

If $\mathcal{Q}$ is parameterized by free variational parameters ϕ, this amounts to optimizing ϕ such that $q_\phi(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z}|\mathbf{x})$, using the ELBO as the objective function.

Similar to the DLVM described above, let $q_\phi(\mathbf{z}|\mathbf{x})$ be parameterized by a deep neural network. A VAE is the result of combining such an inference model with a DLVM, $p_\theta(\mathbf{x}, \mathbf{z})$. By maximizing the ELBO, the variational parameters ϕ and the model parameters θ can be optimized simultaneously and efficiently using stochastic gradient descent. In this setting, $q_\phi(\mathbf{z}|\mathbf{x})$ is also known as the encoder, inference network, recognition network or guide, and $p_\theta(\mathbf{x}, \mathbf{z})$ as the decoder or generative model. Unlike standard (mean-field) VI, which optimizes the variational parameters on a per-datapoint basis, the inference model of a VAE usually only consists of a single encoder neural network with weights and biases given by ϕ. The inference model therefore acts as an amortization artifact, since it makes use of a single set of parameters to model the relationship between all observed data points and the latent space, whether these data points were seen during training or not.

Typically, the VAE generative model $p_\theta(\mathbf{x}, \mathbf{z})$ factorizes as $p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})$. The vanilla approach to constructing a VAE furthermore assumes that all the latents are independent and makes use of simple distributions for both the prior and likelihood. A generic configuration for the generative model is (Kingma and Welling, 2019):

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, I) \tag{2.51}$$
$$\boldsymbol{\mu}, \log \boldsymbol{\sigma} = \text{DecoderNeuralNet}(\mathbf{z}; \theta) \tag{2.52}$$
$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) \tag{2.53}$$

where DecoderNeuralNet is a neural network parameterized by θ and $\mathbf{x}$ is assumed to be continuous. For binary data, $p_\theta(\mathbf{x}|\mathbf{z})$ could instead be a component-wise Bernoulli distribution. The vanilla inference network is similarly given by

$$\boldsymbol{\mu}, \log \boldsymbol{\sigma} = \text{EncoderNeuralNet}(\mathbf{x}; \phi) \tag{2.54}$$
$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) \ . \tag{2.55}$$

Note that this model assumes the latent variables to be conditionally independent given $\mathbf{x}$. Since this is unlikely to hold in the true posterior, an alternative approach would be to let the encoder neural network output not only the (log) standard deviation for each latent variable, but also the covariances between them.

2.4.2 The Evidence Lower Bound

As already discussed in previous sections, a VAE can be trained by maximizing the evidence lower bound (ELBO). For the VAE, this objective can be rewritten in several forms that each shed more light on its characteristics:

$$\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \tag{2.56}$$
$$= \mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \text{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \tag{2.57}$$
$$= p_\theta(\mathbf{x}) - \text{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) \ . \tag{2.58}$$

Equation (2.57) is known as the *prior-contrastive form* of the ELBO. The first term encourages the decoder to learn how to provide good reconstructions of the data. The second acts as a regularizer, encouraging the posterior, $q_\phi(\mathbf{z}|\mathbf{x})$, to be similar to the model prior, $p(\mathbf{z})$ (Razavi *et al.*, 2019). Equation (2.58), known as the *posterior-contrastive form*, again provides the established result that the ELBO is a lower bound on the model evidence:

$$p_\theta(\mathbf{x}) = \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x}) + \underbrace{\text{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))}_{\geq 0} \geq \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x}) \ , \tag{2.59}$$

where the KL-divergence is non-negative by definition, and equal to zero if and only if $q_\phi(\mathbf{z}|\mathbf{x})$ exactly matches the true posterior. Thus, the closer the approximate posterior to the true model posterior in terms of the KL-divergence, the tighter the bound on the evidence. This view has inspired much subsequent work into creating more expressive inference networks than the original approach in Equation (2.55). For example, Rezende and Mohamed (2015) propose extending the approximate posterior with an NF (see Section 3.2.1 for a further discussion).

An important property of the ELBO is that it allows joint optimization of θ and ϕ using stochastic gradient descent. The ELBO objective for a set of i.i.d. data, $X = \{\mathbf{x}_n\}_{n=1}^N$, is given by:

$$\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(X) = \sum_{n=1}^{N} \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x}_n) \ . \tag{2.60}$$

Analytically computing the gradient of this objective ($\nabla_{\theta,\phi}\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(X)$) is intractable, so one generally makes use of estimators. An unbiased estimate of the gradient of the objective with respect to the model parameters θ is easy to obtain by simply swapping the gradient and expectation terms and approximating the resulting expression using Monte Carlo sampling:

$$\begin{aligned}
\nabla_\theta \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x}) &= \nabla_\theta \mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] &(2.61)\\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\right] &(2.62)\\
&\approx \frac{1}{K}\sum_{i=1}^{K} \nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z}_i) \ , &(2.63)
\end{aligned}$$

where each $\mathbf{z}_i$ is a random sample from $q_\phi(\mathbf{z}|\mathbf{x})$. An unbiased estimator for $\nabla_\phi \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x})$ is not as easy to obtain. This is because the ELBO's expectation is taken with respect to the distribution q_ϕ, with parameters ϕ, and one cannot therefore directly swap the gradient and expectation terms as done in (2.62).

One approach to handling this, known as the REINFORCE algorithm (Williams, 1992) or the score function estimator (Kleijnen and Rubinstein, 1996), makes use of a differentiation rule called the log-derivative trick: $\nabla_\phi q_\phi = q_\phi \nabla_\phi \log(q_\phi)$.

The gradient can then be rewritten as follows:

$$\begin{aligned}
\nabla_\phi \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x}) =& \nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] && (2.64)\\
=& \nabla_\phi \int (\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))q_\phi(\mathbf{z}|\mathbf{x})\, d\mathbf{z} && (2.65)\\
=& \int \nabla_\phi \left((\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))q_\phi(\mathbf{z}|\mathbf{x})\right)\, d\mathbf{z} && (2.66)\\
=& \int (\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\nabla_\phi q_\phi(\mathbf{z}|\mathbf{x})\, d\mathbf{z}\\
&+ \int q_\phi(\mathbf{z}|\mathbf{x})\nabla_\phi(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\, d\mathbf{z} && (2.67)\\
=& \int (\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))q_\phi(\mathbf{z}|\mathbf{x})\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x})\, d\mathbf{z}\\
&+ \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\nabla_\phi \log p_\theta(\mathbf{x},\mathbf{z}) - \nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x})\right]}_{=0} && (2.68)\\
=& \mathbb{E}_{\mathbf{z}\sim q_\phi}\left[(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x})\right] && (2.69)
\end{aligned}$$

where we have made use of the definition of an expectation, the Leibniz integral rule (see Appendix B.1.1) and the log-derivative trick in lines (2.65), (2.66) and (2.68), respectively. The second term in line (2.68) evaluates to zero, because $\log p_\theta(\mathbf{x},\mathbf{z})$ does not depend on ϕ and thus $\nabla_\phi \log p_\theta(\mathbf{x},\mathbf{z}) = \mathbf{0}$, and because the expected value of the score function, $\mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x})\right]$, is also zero (Roeder *et al.*, 2017). Since the gradient has successfully been rewritten as an expectation over a tractable distribution for sampling from, Monte Carlo samples can be used to estimate the desired gradient. However, a drawback of this approach is that the estimator has very high variance. Fortunately, there exists an alternative method which overcomes this shortcoming, known as the *reparameterization trick* (Kingma and Welling, 2014; Rezende *et al.*, 2014).

An unbiased and lower-variance estimator of $\nabla_\phi \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(x)$ can be obtained by expressing $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ as an invertible and differential transformation of a sample from another distribution, $\boldsymbol{\epsilon} \sim p_\epsilon$:

$$\mathbf{z} = g(\boldsymbol{\epsilon}; \phi, \mathbf{x}) \ , \tag{2.70}$$

where the distribution p_ϵ is independent of both ϕ and $\mathbf{x}$. Under this change of variable, one can use the law of the unconscious statistician to rewrite the original expectation in terms of $\boldsymbol{\epsilon}$:

$$\nabla_\phi \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(x) = \nabla_\phi \mathbb{E}_{\boldsymbol{\epsilon}\sim p_\epsilon}\left[\log p_\theta(\mathbf{x}, g(\boldsymbol{\epsilon}; \phi, \mathbf{x})) - \log q_\phi(g(\boldsymbol{\epsilon}; \phi, \mathbf{x})|\mathbf{x})\right] \ . \tag{2.71}$$

Similar to the analysis of $\nabla_\theta \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x})$ in (2.61) to (2.63), the order of the gradient and expectation operators can now be exchanged, and the gradient can be estimated using Monte Carlo sampling:

$$\nabla_\phi \mathcal{L}^{\text{ELBO}}_{\theta,\phi}(x) = \mathbb{E}_{\boldsymbol{\epsilon}\sim p_\epsilon}\left[\nabla_\phi(\log p_\theta(\mathbf{x}, g(\boldsymbol{\epsilon};\phi,\mathbf{x})) - \log q_\phi(g(\boldsymbol{\epsilon};\phi,\mathbf{x})|\mathbf{x}))\right] \tag{2.72}$$

$$\approx \frac{1}{K}\sum_{i=1}^{K}\nabla_\phi\left(\log p_\theta(\mathbf{x}, g(\boldsymbol{\epsilon}_i;\phi,\mathbf{x})) - \log q_\phi(g(\boldsymbol{\epsilon}_i;\phi,\mathbf{x})|\mathbf{x})\right), \tag{2.73}$$

where a single Monte Carlo sample, i.e $K = 1$, is widely used.

For the vanilla inference network given by Equations (2.54) and (2.55), the reparameterization trick is implemented as follows:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I) \tag{2.74}$$

$$\boldsymbol{\mu}, \log\boldsymbol{\sigma} = \text{EncoderNeuralNet}(\mathbf{x};\phi) \tag{2.75}$$

$$\mathbf{z} = \boldsymbol{\sigma}\odot\boldsymbol{\epsilon} + \boldsymbol{\mu} \tag{2.76}$$

where $\odot$ denotes element-wise multiplication. In practice, these estimates of the gradients $\nabla_\theta\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x})$ and $\nabla_\phi\mathcal{L}^{\text{ELBO}}_{\theta,\phi}(\mathbf{x})$, are computed using automatic differentiation libraries.

2.4.3 Estimating the Marginal

Given a trained VAE, one can use the ELBO to compute a lower bound on the marginal likelihood of new observed data. It is possible to tighten this bound by using *importance weighted* samples (Rezende *et al.*, 2014). Consider the following approach to computing the log marginal:

$$\log p_\theta(\mathbf{x}) = \log\left[\int p_\theta(\mathbf{x},\mathbf{z})\,d\mathbf{z}\right] \tag{2.77}$$

$$= \log\left[\int \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} q_\phi(\mathbf{z}|\mathbf{x})\,d\mathbf{z}\right] \tag{2.78}$$

$$= \log\mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \tag{2.79}$$

$$\approx \log\left[\frac{1}{K}\sum_{i=1}^{K}\frac{p_\theta(\mathbf{x},\mathbf{z}_i)}{q_\phi(\mathbf{z}_i|\mathbf{x})}\right] \tag{2.80}$$

where each $\mathbf{z}_i \sim q_\phi$ is a random sample from the VAE's inference network. Since the final line above is a Monte Carlo estimator, taking $K \to \infty$ results in the estimate converging to the actual log-marginal likelihood. The terms $w_i = p_\theta(\mathbf{x},\mathbf{z}_i)/q_\phi(\mathbf{z}_i|\mathbf{x})$ are known as the importance weights. Note that when $K = 1$, this estimate is exactly the ELBO estimate used for standard VAEs. This has led some to use this estimator, with larger values of K, as the objective function for training VAEs, instead of the standard ELBO. The resulting model is known as the importance weighted autoencoder (IWAE) (Burda *et al.*, 2016).

2.4.4 Posterior Collapse

Although the VAE framework has gained much popularity as a tool to learn unsupervised representations, optimization of its parameters in general suffers from a phenomenon known as *posterior collapse*—a subset of the latent variables 'collapses' to the uninformative prior during training. This results in the generative model effectively ignoring these latent dimensions which is undesirable when a goal of VAE training is to learn meaningful latent features.

As such, much work has focused on trying to understand this phenomenon and to uncover its cause. Early work blamed the regularizing KL-divergence term present in the prior-contrastive form of the ELBO objective, $\mathrm{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$, which encourages the posterior to be similar to the prior (Razavi *et al.*, 2019; Bowman *et al.*, 2016). The most common approaches to addressing posterior collapse therefore focus on diminishing the effect of this term. One such approach, known as warm-up, is where a weight on the KL-term is annealed between 0 and 1 over some initial number of epochs during training (Burda *et al.*, 2016; Bowman *et al.*, 2016; Huang *et al.*, 2018*b*). Alternatively, Razavi *et al.* (2019) carefully choose the prior and variational family such that $q(\mathbf{z}|\mathbf{x})$ is guaranteed to remain some minimal distance from the prior. Kingma *et al.* (2016) implicitly impose the same constraint by ignoring the gradient of this KL-term per dimension of $\mathbf{z}$ if the divergence is below a given threshold. McCarthy *et al.* (2020) argue that optimizing $\mathrm{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$ decreases the mutual information between the latent variables and the observed data and propose a modification of the ELBO objective that promotes mutual information between the latent and observed variables.

Since the above, and other similar approaches, do not fully resolve the optimization issue posed by posterior collapse, additional factors that may lead to collapse have also been investigated. He *et al.* (2019*b*) suggest that posterior collapse could be caused by the inference network lagging behind the true posterior during the early stages of training and recommend more aggressive optimization of the inference network before each update of the generative model. Since posterior collapse is commonly observed in VAEs with very flexible generative models, some argue that bad local minima in the loss surface of deep autoencoder networks could be the direct cause of collapsed latents (Dai *et al.*, 2020). High-variance gradient estimates have also been suggested as a potential cause for posterior collapse (Melis *et al.*, 2022). Since stochastic gradient descent optimization has a preference for flat minima (Hochreiter and Schmidhuber, 1994), and the variance induced by the sampled latent variables can be reduced by ignoring them, this can lead to posterior collapse. This motivates the use of low-variance gradient estimates such as the doubly-reparameterized gradient (DReG) estimate proposed by Tucker *et al.* (2018). The looseness of the ELBO objective can also allow the parameters of the VAE to remain distant from their theoretical optimal setting, which could lead to posterior collapse (Melis *et al.*, 2022). Using objectives with tighter bounds, such as IWAE (Burda *et al.*, 2016), have been proposed to mitigate this. It has also been found that models that are optimal in terms of the ELBO and marginal likelihood obtained, can differ greatly in their corresponding posteriors (Alemi *et al.*, 2018), with some representations including collapsed latents. The optimization task presented by the VAE framework is therefore often underspecified, which can be addressed by constraining the mutual information between latent and observed variables (Melis *et al.*, 2022).

Preventing posterior collapse is particularly desirable if we want the latent factors to represent meaningful encodings of the observed space. The final section of this chapter therefore concludes by discussing the concept of interpretability in the context of DLVMs.

2.4.5 Interpretability of Deep Latent Variable Models

The inclusion of latent variables in DLVMs such as VAEs not only help to make the marginal distribution $p(\mathbf{x})$ more expressive, but also provide a principled way of learning lower-dimensional representations of complex data. These representations or features can be useful in a wide range of downstream tasks. The concept of *interpretability* comes in to play when we want to learn general purpose features that could potentially help us to better understand the application domain as well as the behaviour of the model. Building interpretable models that produce explainable output have recently begun to receive renewed attention and fall under the umbrella term of explainable artificial intelligence (Samek *et al.*, 2019).

A traditional way of obtaining interpretable features in VAEs is by encouraging the model to learn *disentangled* representations, that is, representations that capture independent factors of variation in the observed data. One of the earliest examples of this is the β-VAE (Higgins *et al.*, 2017). By noting how each latent factor affects the generated data, one can try to assign human interpretable meaning to these factors. For example, a β-VAE trained on images of faces could learn factors that represent the colour of hair, or emotion as expressed by a smile. Another approach that can afford the model a higher degree of interpretability, is incorporating a form of factor analysis—learning which subsets of observed variables are affected by which subsets of latent factors (Ainsworth *et al.*, 2018). These types of approaches help us to better understand the relationships between the variables of the model, specifically between the latent and observed variables. In Chapters 6 and 7 of this work, we will also consider the relationships between individual latent variables as informed by a Bayesian network.

One concept for measuring the relationship or dependence between variables is mutual information (MI) (Shannon, 1948). MI is a more general metric than the correlation coefficient, which only captures linear dependence, and is therefore more applicable in situations where highly non-linear deep neural networks are employed.

2.4.5.1 Mutual Information

Given two real random vectors $\mathbf{x}$ and $\mathbf{y}$, mutual information (MI) (Shannon, 1948) measures the mutual dependence between $\mathbf{x}$ and $\mathbf{y}$ and is defined as

$$\mathrm{I}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{p(\mathbf{x},\mathbf{y})} \left[\log \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})p(\mathbf{y})} \right] \ , \tag{2.81}$$

where $p(\mathbf{x}, \mathbf{y})$, $p(\mathbf{x})$, $p(\mathbf{y})$ are the joint distribution and marginal distributions of $\mathbf{x}$ and $\mathbf{y}$, respectively. $\mathrm{I}(\mathbf{x}, \mathbf{y})$ is equivalently the Kullback-Leibler divergence, $\mathrm{KL}(p(\mathbf{x}, \mathbf{y})||p(\mathbf{x})p(\mathbf{y}))$, of the joint distribution of $\mathbf{x}$ and $\mathbf{y}$ from the product of their marginals. The intuition is that a larger divergence between the joint and the product of the marginals implies a stronger dependence between $\mathbf{x}$ and $\mathbf{y}$.

MI is more general than the correlation coefficient, which only captures linear dependence. Unfortunately, MI is intractable to compute in general—Equation (2.81) can only be calculated exactly and efficiently for discrete variables or a limited family of problems with known probability distributions for which the integral represented by the expectation has a closed form solution. As such, a range of approximation techniques have been developed for MI estimation over the years. Early work employed kernel density estimators to individually estimate the joint and marginal distributions from samples (Fraser and Swinney, 1986). Approximating the MI naïvely then involves division of the estimated densities, which can magnify estimation errors (Suzuki *et al.*, 2008). An alternative approach that avoids this division is to directly model the density ratio, as is done by Suzuki *et al.* (2008). Other approaches to density estimation employ methods such as k-nearest neighbours (Kraskov *et al.*, 2004) or the Edgeworth expansion (van Hulle, 2005). The former requires careful selection of an appropriate value for k, while the latter assumes the target distribution to be approximately Gaussian distributed.

Since neural methods have become more effective in the past decade, more recent approaches employ neural networks to estimate MI. The mutual information neural estimator (MINE) of Belghazi *et al.* (2018) maximizes a tight lower bound on the MI using stochastic gradient descent and backpropagation. Belghazi *et al.* (2018) make use of the equivalence outlined above between MI and the KL-divergence of the joint distribution from the product of the marginals. Specifically, they employ the Donsker-Varadhan dual representation of the KL-divergence (Donsker and Varadhan, 1983):

$$\mathrm{KL}(p||q) = \sup_{F:\Omega \to \mathbb{R}} \mathbb{E}_p[F] - \log \mathbb{E}_q[e^F] \ , \tag{2.82}$$

where the supremum is taken over *all* functions F such that the expectations are finite. If we let $\mathcal{F}$ be any family of functions satisfying the above constraint, then Belghazi *et al.* (2018) show that the following lower-bound holds:

$$\mathrm{KL}(p||q) \geq \sup_{F\in\mathcal{F}} \mathbb{E}_p[F] - \log \mathbb{E}_q[e^F] \ . \tag{2.83}$$

Given real random vectors $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$, we can thus choose $\mathcal{F}$ to be the family of functions $F_\theta : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ parameterized by a neural network with parameters θ. Following from (2.83), we have the following lower-bound on the MI between these random vectors:

$$\mathrm{I}(\mathbf{x}, \mathbf{y}) \geq \mathbb{E}_{p(\mathbf{x},\mathbf{z})}[F_\theta] - \log \mathbb{E}_{p(\mathbf{x})p(\mathbf{y})}[e^{F_\theta}] \ . \tag{2.84}$$

By maximizing the right-hand side of the above inequality, Belghazi *et al.* (2018) are able provide a general-purpose parametric neural estimator of MI. The expectations in (2.84) are estimated using i.i.d. samples $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ drawn from the joint distirbution, $p(\mathbf{x}, \mathbf{y})$. Samples from the marginals are obtained by simply dropping either $\mathbf{x}_n$ or $\mathbf{y}_n$ from $(\mathbf{x}_n, \mathbf{y}_n)$. Since this approach is flexible and can efficiently be optimized for samples of continuous random variables using stochastic gradient descent, we will employ this approach when estimating MI in Chapter 7.

2.5 Conclusion

This chapter provided an overview of the prominent probabilistic generative models investigated in this work, namely BNs, NFs and VAEs. The graphical structure of a BN presents a compact encoding of conditional independence statements about its associated distribution, and thus provides an intuitive way of encoding domain knowledge about variable interactions. NFs and VAEs employ deep learning to learn complex distributions. Although NFs are specified by bijective transformations, and thus are theoretically invertible, their inversion stability *in practice* in not always guaranteed, as was discussed in more detail. The chapter was concluded with a discussion on the interpretability of deep latent variables models, such as VAEs. Having presented the most relevant background, we next provide an overview of recent literature relevant to the investigations of this thesis. We specifically consider existing flows that incorporate information from a BN as well as approaches that add structure to VAEs.

Chapter 3

Literature Review

Unlike data-driven deep generative models, Bayesian networks (BNs) emphasize modelling the joint distribution of a set of random variables as a product of conditional distributions. BNs provide a principled way to specify structure through the dependencies in these conditional distributions, and thus provide an intuitive way of injecting domain knowledge into a model. BNs have been at the heart of expert systems since before the advent of large data-driven models (Papaconstantinou *et al.*, 1998; Diez *et al.*, 1997), but they have received progressively less attention in favour of purely data-driven models that are capable of scaling to high-dimensional settings, such as deep generative models (Wehenkel and Louppe, 2021). Nevertheless, there has been work trying to combine the strengths of these two approaches: the simplicity and interpretability of BNs, and the scalability and representation capacity of deep generative models. Since we are specifically interested in the integration of BNs and the deep generative model frameworks of NFs and VAEs, this chapter primarily reviews relevant recent literature in this area. Section 3.1 considers flows that incorporate additional graphical structure, while Section 3.2 reviews similar ideas in the context of VAEs.

3.1 Normalizing Flows with Graphical Structures

Incorporating the dependency structure of BNs into NFs has only begun to receive attention in the past two years. Wehenkel and Louppe (2020) provided insight into the relationship between NFs and BNs: the modelling assumptions underlying the autoregressive and coupling transformations used in NFs correspond to specific classes of BNs with predefined graphical structures and learnable densities at each vertex in the graph. To see why this relationship holds, recall that a BN specifies the following factorization of the joint distri-

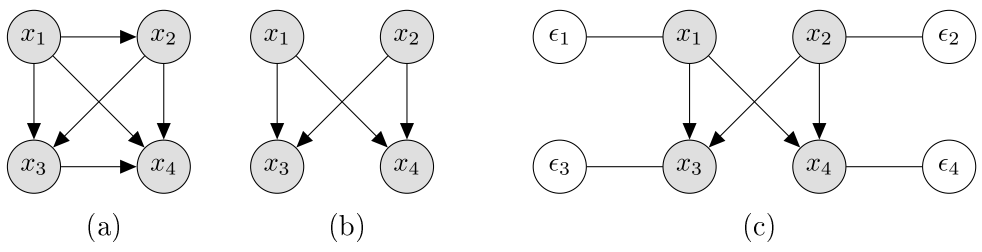


Figure 3.1: BNs associated with a 1-step NF transformation of a vector $\mathbf{x} \in \mathbb{R}^4$ where the flow has (a) an autoregressive structure and (b) a coupling structure. In (c) the latent variables $\boldsymbol{\epsilon}$ are shown explicitly for the coupling transformation in (b). Here, undirected edges correspond to bijective transformations, whereas directed edges are deterministic functions (Wehenkel and Louppe, 2020).

bution over the variables $\mathbf{x} \in \mathbb{R}^D$:

$$p(\mathbf{x}) = \prod_{i=1}^{D} p_i(x_i | \mathbf{x}_{\mathrm{Pa}(i)}) \ , \tag{3.1}$$

where $\mathbf{x}_{\mathrm{Pa}(i)}$ is the subvector of variables corresponding to the parents of x_i in the BN graph. For an autoregressive NF, the joint distribution $p_\theta(\mathbf{x})$ induced by the flow transformation can similarly be factorized as the product of D conditional distributions:

$$p_\theta(\mathbf{x}) = p_\theta(x_1) \prod_{i=2}^{D} p_\theta(x_i | \mathbf{x}_{<i}) \ . \tag{3.2}$$

We can make the connection between BNs and autoregressive flows explicit if we let $\mathbf{x}_{\mathrm{Pa}(i)}$ in (3.1) equal $\mathbf{x}_{<i}$ in (3.2). Flows with autoregressive transformations can therefore be viewed as fully-connected BNs that make no independence assumptions about the data, as depicted in Figure 3.1a. Figure 3.1b shows the BN associated with a coupling transformation applied to $\mathbf{x} \in \mathbb{R}^4$ where one half of the variables (x_3 and x_4) are dependent on the other half (x_1 and x_2). Although not explicitly shown in Equation (3.2), each x_i is a bijective transformation of a latent variable ϵ_i. This relationship is made clear in Figure 3.1c. Note that the above argument, as presented by Wehenkel and Louppe (2020), is only in the context of single-step finite NFs that are applied in the normalizing direction for density estimation. In Section 4.5 we also consider flows applied in the generative direction, as well as the effect of adding more transformation steps on the induced dependencies between the variables of the distribution represented by the flow.

Based on the above relationship between NFs and BNs, Wehenkel and Louppe (2021) explore how to encode a BN with an *arbitrary* graphical structure into a flow. They achieve this with *graphical conditioners*, which enforce the independence assumptions implied by the BN. These graphical conditioners can been seen as sparse versions of the autoregressive conditioners presented in Section 2.3.1.1 (in the same way that the BN is a sparsified version of a complete graph), and as such still yield triangular Jacobians. Wehenkel and Louppe (2021) consider finite flows where each flow step can be divided into a transformer-conditioner pair. Similar work by Weilbach *et al.* (2020) considers continuous flows. They extend the continuous NF of Grathwohl *et al.* (2019) to incorporate a given BN by encoding its graphical structure into the weight matrices of the neural network. This results in sparse weight matrices that enforce the necessary independencies between the variables. The work of Wehenkel and Louppe (2021) and Weilbach *et al.* (2020) only consider autoregressive finite flows (Section 2.3.1.1) and continuous flows (Section 2.3.2), respectively, and only focus on applying their flows in a single transformation direction. We will therefore focus on constructing a graphical flow that can be safely used in both transformation directions via stable inversion guarantees. We do this in Chapter 4 by extending a residual flow (Section 2.3.1.2) to incorporate arbitrary graphical structure.

Since, the work of Wehenkel and Louppe (2021) and Weilbach *et al.* (2020) are the most explicit examples of models that combine BNs with normalizing flows and form the baseline for evaluating our proposed graphical residual flow, we present them in more detail in Sections 3.1.1 and 3.1.2. Section 3.1.3 provides an overview of other works that deal with similar ideas.

3.1.1 Finite Flows with Graphical Structures

Consider a finite flow over $\mathbf{x} \in \mathbb{R}^D$ where each flow step is constructed from a transformer-conditioner pair as presented in Section 2.3.1.1, i.e.:

$$x_i^{(t)} = \left[f_t(\mathbf{x}^{(t-1)})\right]_i = g_{t,i}(x_i^{(t-1)}; c_{t,i}(\mathbf{x}_{<i}^{(t-1)})), \quad i = 1, \dots, D \tag{3.3}$$

where $g_{t,i}$ is the invertible transformer which is partially parameterized by the conditioner function, $c_{t,i}$. To incorporate a BN's dependency structure, Wehenkel and Louppe (2021) replace the autoregressive conditioner, $c_{t,i}(\mathbf{x}_{<i}^{(t-1)})$, with a *graphical conditioner*, $c_{t,i}(\mathbf{x}_{\mathrm{Pa}(i)}^{(t-1)})$, so that the conditioner is only a function of x_i's parents in the BN graph. As long as the variables are ordered according to their topological ordering in the BN, then $\mathbf{x}_{\mathrm{Pa}(i)}$ will be a subvector of $\mathbf{x}_{<i}$. The Jacobian therefore remains triangular, so its determinant can still be computed in linear time. Wehenkel and Louppe (2021) introduce the term graphical NF for these finite flows with graphical conditioners. Note that we use this term in a wider sense as well, to refer to all normalizing flows that incorporate graphical structure through their architecture. In their study, Wehenkel and Louppe (2021) evaluate their graphical conditioner approach with

affine and monotonic transformers (as presented in Section 2.3.1.1):

$$\begin{aligned}
\text{Affine:} \quad & g_i(x_i; s_i(\mathbf{x}_{\mathrm{Pa}(i)}), m_i(\mathbf{x}_{\mathrm{Pa}(i)})) = \exp(s_i(\mathbf{x}_{\mathrm{Pa}(i)})) \cdot x_i + m_i(\mathbf{x}_{\mathrm{Pa}(i)}) \\
\text{Monotonic:} \quad & g_i(x_i; c_i(\mathbf{x}_{\mathrm{Pa}(i)})) = \int_0^{x_i} h_i(t; c_i(\mathbf{x}_{\mathrm{Pa}(i)}))\, dt + \beta_i(c_i(\mathbf{x}_{\mathrm{Pa}(i)})) \ ,
\end{aligned}$$

where $c_i(\mathbf{x}_{\mathrm{Pa}(i)}) = \left[s_i(\mathbf{x}_{\mathrm{Pa}(i)}), m_i(\mathbf{x}_{\mathrm{Pa}(i)})\right]$ denotes the conditioner functions of the affine transformation, and the dependence on the time step has been dropped for notational simplicity.

Although one could implement D separate conditioner functions, c_i for $i = 1, \ldots, D$, that each take a different subset of variables, $\mathbf{x}_{\mathrm{Pa}(i)}$, as input, Wehenkel and Louppe (2021) choose to use only a single neural network to implement their conditioner functions. To ensure that the correct independencies are still maintained, they perform D passes through this neural network, masking out those inputs during forward pass i that are not in $\mathbf{x}_{\mathrm{Pa}(i)}$. An additional one-hot encoded vector of length D encoding the value i is also added to the input. This ensures that all root vertices, i.e. those vertices without any parents in the graph, will not have the same output from the conditioner function. As suggested by (Wehenkel and Louppe, 2021) and (Lachapelle *et al.*, 2020), one could instead use a masking scheme similar to MADE (Germain *et al.*, 2015) (see Section 2.3.1.1), but for arbitrary graphical structures. In Chapter 4 we present such an alternative masking scheme that only requires a *single pass* through the conditioner neural network and does not require the use of additional one-hot encoded inputs. Next, we discuss an approach to incorporating graphical structure in continuous NFs.

3.1.2 Continuous Flows with Graphical Structures

Weilbach *et al.* (2020) explore incorporating a graphical structure into continuous NFs. Consider a neural ODE system (Chen *et al.*, 2018),

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t) \ . \tag{3.4}$$

Weilbach *et al.* (2020) consider neural networks f where each layer takes the form,

$$g(\mathbf{x}, t) = h((W\mathbf{x}) \odot \eta_1(t)) + \mathbf{b} \odot \eta_2(t) \ , \tag{3.5}$$

where η_1 and η_2 are time-dependent linear gating functions, h is any Lipschitz smooth activation function, and W and $\mathbf{b}$ are the layer weights and biases, respectively.

To incorporate a BN graphical dependency structure into such a flow, Weilbach *et al.* (2020) apply a binary adjacency matrix A as a mask to the weight matrix W, where $A_{j,i} = 1$ if $i = j$ or $j \in \mathrm{Pa}(i)$, else $A_{j,i} = 0$. This mask ensures that output i of any specific layer is only dependent on the inputs to that layer corresponding to x_i and $\mathbf{x}_{\mathrm{Pa}(i)}$. Each layer of this new flow, termed the structured conditional continuous NF (SCCNF), is thus given by:

$$g(\mathbf{x}, t) = h((W \odot A)\mathbf{x} \odot \eta_1(t)) + \mathbf{b} \odot \eta_2(t) \ . \tag{3.6}$$

One limitation of this approach is that it restricts the width of each layer of the neural network to be the same as the dimension of $\mathbf{x}$. To ease this restriction, Weilbach *et al.* (2020) allocate a flexible number of auxiliary or nuisance variables to each variable in the flow, creating an augmented neural ODE (Dupont *et al.*, 2019*a*), which has been shown to be more expressive and to generalize better. Alternatively, the masking scheme that we present in Chapter 4, does not have this limitation and allows arbitrary hidden layer widths. Weilbach *et al.* (2020) show that using the SCCNF approach allows one to reduce not only the number of model parameters but also the number of required integration steps without compromising the flow performance, compared to the standard continuous flow of Grathwohl *et al.* (2019).

Apart from presenting a method for incorporating a graphical structure into continuous NFs, Weilbach *et al.* (2020) also propose using the symmetrized KL-divergence, also known as Jeffrey's divergence (Nielsen, 2010), as an objective. The symmetrized KL-divergence is a weighted sum of the forward and reverse KL-divergences:

$$\mathrm{KL}_{\mathrm{sym}}(p(\mathbf{z}|\mathbf{x}) || \, q(\mathbf{z}|\mathbf{x})) = \frac{1}{2} \left[\mathrm{KL}(p(\mathbf{z}|\mathbf{x}) || \, q(\mathbf{z}|\mathbf{x})) + \mathrm{KL}(q(\mathbf{z}|\mathbf{x}) || \, p(\mathbf{z}|\mathbf{x})) \right] . \tag{3.7}$$

They compare the effect of optimizing the symmetrized KL-divergence instead of either the forward or reverse KL-divergence, and based on their experimental results conclude that only optimizing the reverse KL-divergence does not provide a strong enough learning signal. We were, however, unable to reproduce these results and did not find the reverse KL-divergence to perform poorly. See Appendix C for further details. After a disussion with the authors, it was found that the results presented in the original article were indeed erroneous, and that using the reverse KL-divergence as objective leads to comparable performance to when using the symmetrized KL-divergence (C. Weilbach, personal communication, August 21, 2021).

3.1.3 Related Work

Weilbach *et al.* (2020) apply their graphical continuous flows to obtain amortized variational inference (VI) artifacts for input probabilistic programs, which are models specified in a probabilistic programming language (van de Meent *et al.*, 2018). Since these flows only incorporate the graphical structure of the input program and not the forms of its conditional distributions, Ambrogioni *et al.* (2021*b*) propose an alternative approach they call cascading flows, which integrates automatic structured VI (Ambrogioni *et al.*, 2021*a*) and normalizing flows. The cascading flows program interposes flow transformations, based on highway networks (Srivastava *et al.*, 2015), between the conditional distributions of the input program in order to guide it towards the approximate posterior. Unlike the works of Wehenkel and Louppe (2021) and Weilbach *et al.* (2020), which encode the dependency structure via the flow architecture itself, cascading flows use multiple flows placed between the conditional distributions and use auxiliary variables to introduce necessary dependencies. Silvestri *et al.* (2022) provide a more flexible extension of cascading flows which they call embedded-model flows (EMFs). Unlike the graphical normalizing flows of Wehenkel and Louppe (2021), which generalize autoregressive dependency structures to arbitrary DAG dependency structures and thus only constrain the conditional independencies of the flow, EMFs additionally embed the forward-pass of an input probabilistic program. Both cascading flows and EMFs require a user-specified input probabilistic program. As such, they are not directly applicable in our more general setting where we only assume to have access to the graphical structure.

The application of normalizing flows in the field of causality has also begun to receive attention in recent years. Khemakhem *et al.* (2021) exploit the fact that autoregressive flows impose an ordering over the variables of interest, which can be chosen in accordance with a potential or hypothesized causal ordering, to show that these flows are suitable for causal inference tasks such as causal discovery and counterfactual predictions. Since a DAG dependency structure with a suitable causal interpretation is important for causal inference, Balgi *et al.* (2022) combine the above idea with the graphical flow of Wehenkel and Louppe (2021), and propose the causal graphical normalizing flow (c-GNF). Whereas the work of Khemakhem *et al.* (2021) only uses arbitrary autoregressive structures, Balgi *et al.* (2022) can use a c-GNF to impose a desired causal dependency structure with specific independencies.

3.2 Adding Structure to VAEs

The vanilla VAE architecture makes use of simple factorized Gaussian distributions for both the latents' prior and approximate posterior distributions (Kingma and Welling, 2014). These choices can lead to over-regularization of the latent space, resulting in poor latent representations and variational bounds (Hoffman and Johnson, 2016). As such, a great deal of subsequent work has focused on understanding and improving the shortcomings of the standard VAE. Initial work focused mainly on the *inference gap*—the mismatch between the approximate posterior and the true posterior distribution of the model. Cremer *et al.* (2018) further divide this gap into two contributing factors. The first is the *approximation gap*, which arises from the inability of the variational distribution family of the inference network to match the true posterior distribution. The second, known as the *amortization gap*, is caused by the difference between amortizing the variational parameters over the entire training set in stead of optimizing them on a per-datapoint basis, as is done in more traditional variational inference. A popular avenue to reduce the inference gap is to use more powerful representations for the variational posterior distribution (Rezende and Mohamed, 2015; Kingma *et al.*, 2016; Webb *et al.*, 2018; Vahdat *et al.*, 2020; Kim and Pavlovic, 2021).

Another issue that has been identified is the *prior hole* problem (Rosca *et al.*, 2018): if the prior distribution does not match the *aggregate posterior*, defined as $q_{\text{agg}}(\mathbf{z}) := \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})}[q(\mathbf{z}|\mathbf{x})]$ where $p_{\mathcal{D}}(\mathbf{x})$ is the training data distribution, then samples generated from areas that have high density under the prior, but low density under the posterior, can be of suboptimal quality and not match the training data. This has inspired works that use more complex prior distributions (Huang *et al.*, 2017; Xu *et al.*, 2019; Goyal *et al.*, 2017; Takahashi *et al.*, 2019; Tomczak and Welling, 2018).

Many of the above approaches that try to increase the expressive power of the prior and/or the posterior distribution, make use of *structure*. We provide an overview of these methods in Section 3.2.1. We then narrow our discussion in Section 3.2.2 to works that incorporate *graphical* structures, as informed by either the application domain or the training data.

3.2.1 Increasing Latent Space Complexity

As discussed above, many works attempt to tighten the ELBO and improve sample quality by increasing the expressive power of the approximate posterior or prior of the VAE. One of the most commonly used approaches is adding more layers of latent variables in order to incorporate a hierarchical structure in the latent space (Rezende *et al.*, 2014; Burda *et al.*, 2016; Sønderby *et al.*, 2016; Vahdat and Kautz, 2020; Kingma *et al.*, 2016). The joint probability model then factorizes as

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z}_0)p(\mathbf{z}_0|\mathbf{z}_1)\dots p(\mathbf{z}_{L-1}|\mathbf{z}_L)p(\mathbf{z}_L) \ , \tag{3.8}$$

for L additional layers of latent variables. The simplest approach to constructing the accompanying variational posterior is to assume it takes on a complementary factorization (Burda *et al.*, 2016),

$$q(\mathbf{z}|\mathbf{x}) = q(\mathbf{z}_0|\mathbf{x})q(\mathbf{z}_1|\mathbf{z}_0)\dots q(\mathbf{z}_L|\mathbf{z}_{L-1}) \ , \tag{3.9}$$

where the dependencies between the variables have simply been inverted. This approach therefore increases the complexity of the both the prior and approximate posterior distributions. Both (3.8) and (3.9) require sampling and density evaluation to be tractable for each of the conditional distributions. These conditional probability distributions can be as simple as Gaussians (Burda *et al.*, 2016), or be specified by more complex models such as normalizing flows (Kingma *et al.*, 2016). Note that more complex schemes, such as bi-directional inference where each latent receives information from the layers above and below it, are also employed (Kingma *et al.*, 2016; Sønderby *et al.*, 2016).

Consider again the factorization of the hierarchical approximate posterior distribution, $q(\mathbf{z}|\mathbf{x})$, given in Equation (3.9). The dependencies present in this distribution are simply the inverses of those present in Equation (3.8), i.e. since $\mathbf{z}_i$ is only dependent on $\mathbf{z}_{i+1}$ in $p(\mathbf{x}, \mathbf{z})$, $\mathbf{z}_{i+1}$ will only be dependent on $\mathbf{z}_i$ in $q(\mathbf{z}|\mathbf{x})$. Although this factorization simplifies learning the approximate posterior, it does not necessarily correspond to the conditional independencies present in the true posterior.

Looking at the factorization of the VAE generative process, $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$, we see that it corresponds to a BN graph over the observed and latent variables: $\mathbf{z} \rightarrow \mathbf{x}$ (where we have not made any assumptions about the dependency structure between the individual latent variables, which could for example be independent or connected via a hierarchical structure as presented above). Taking this further, Webb *et al.* (2018) showed the importance of having the dependencies induced between the latent variables of the inference network be faithful to the dependencies present in the generative model's BN, in order for it to be able to learn an approximation close to the true posterior. As a result, they propose an algorithm for inverting the BN network underlying the VAE generative model (i.e. inverting the direction of the arrows in the graph such that the flow of information changes from $\mathbf{z} \rightarrow \mathbf{x}$ to $\mathbf{x} \rightarrow \mathbf{z}$), in such a way that the inverted structure over both the observed and latent variables does not encode any conditional independencies not implied by the generative model. For example, based on the BN structure underlying the standard VAE model, its inference network should encode a fully-connected structure over the latent variables. This is needed so that the inference network can take into account the explaining-away effects between the latent variables in the generative model, i.e. given an observation, the latent variables are no longer independent as assumed by the vanilla VAE. Section 4.4 provides illustrated examples and more details on the importance of faithful BN inverses in the context of variational inference.

For the vanilla VAE, the issue of ensuring an appropriate posterior dependency structure can be addressed by for example using a full covariance matrix in the multivariate normal posterior distribution. Another approach is to extend the inference network with a normalizing flow (Rezende and Mohamed, 2015; Kingma *et al.*, 2016). This allows for more expressive distributions, by allowing the latent variables to become entangled, potentially leading to a smaller approximation gap. Other examples of more expressive posterior distributions that have been used in the context of VAEs include undirected graphical models such as Boltzmann machines (Vahdat *et al.*, 2020), and Gaussian processes (Kim and Pavlovic, 2021; Fortuin *et al.*, 2020).

We next consider works focusing on constructing VAEs with more flexible and expressive prior distributions. This is to allow a better match of the prior and aggregate posterior, thus mitigating the prior hole problem. It is not only the inference network that can be extended with normalizing flows: some works have also replaced the prior with a flow. For example, both Huang *et al.* (2017) and Xu *et al.* (2019) use Real NVP (Dinh *et al.*, 2017) as a learnable prior alongside the inference network, and show that this leads to a better lower bound and reconstruction loss. Other approaches to constructing more expressive and learnable priors have been proposed, such as the work of Goyal *et al.* (2017), which combines tree-structured Bayesian nonparametric priors with VAEs to induce a hierarchical structure of latent semantic concepts. Since the optimal prior of the VAE in terms of maximizing the training objective is given by the aggregate posterior, Takahashi *et al.* (2019) suggest using the aggregate posterior as the prior. In this case, the KL-divergence can no longer be calculated in closed form. Takahashi *et al.* (2019) therefore make use of a density ratio trick to estimate the KL-divergence without explicitly computing the aggregate posterior. The work of Tomczak and Welling (2018), presenting the VampPrior—“Variational Mixture of Posteriors” prior—is based on similar ideas. This prior is given by a mixture of variational posteriors conditioned on pseudo-inputs. These inputs are learned through backpropagation and can be thought of as hyperparameters of the prior.

Next, we provide an overview of works that attempt to incorporate some form of graphical structure—as informed by either the data or the application domain—into VAE models.

3.2.2 VAEs with Graphical Structures

The work of He *et al.* (2019*a*), who also try to combine the strengths of Bayesian networks and VAEs, is perhaps most closely aligned with our objectives. They do this by representing the latent prior distribution of the VAE as a BN whose dependency structure is learned during training, and use an inference network whose dependency structure mirrors that of the generative model. The conditional independencies between the latent variables are controlled with a set of global binary gating variables. These gating variables switch the flow of information between individual latent variables ‘on’ or ‘off’, and as such specify the dependencies between these variables. The gating variables are jointly trained with the model and inference network parameters using a single objective. They empirically show that their proposed Graph VAE outperforms the standard VAE (Kingma and Welling, 2014), ladder VAE (Sønderby *et al.*, 2016) and a version of their model that encodes a fully-connected BN, on several image datasets in terms of log-likelihood. They make use of conditional Gaussian distributions for the individual latent dimensions and do not consider potential independencies between the latent and observed variables. We will instead consider using more flexible normalizing flows to represent the latent distributions and will consider the potential benefits of encoding a predefined dependency structure (between the latent variables and between the latent and observed variables) as specified by the domain.

As with normalizing flows, the application of VAEs in the context of causal modelling is also a natural point at which to consider incorporating additional graphical structure. An early example is the causal effect variational autoencoder (CEVAE) (Louizos *et al.*, 2017), which is used to infer causal effects from data. CEVAE specifically focuses on handling confounders (variables that affect both an intervention and its outcome). Louizos *et al.* (2017) extend the basic VAE model, which factorizes as $p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$, by introducing new variables $\mathbf{y}$ and $\mathbf{t}$ and updating the architecture such that the model now factorizes as $p(\mathbf{x}|\mathbf{z})p(\mathbf{y}|\mathbf{t},\mathbf{z})p(\mathbf{t}|\mathbf{x})p(\mathbf{z})$, with a corresponding approximate posterior, $q(\mathbf{z}|\mathbf{t},\mathbf{y},\mathbf{x})q(\mathbf{y}|\mathbf{t},\mathbf{x})q(\mathbf{t}|\mathbf{x})$. In this context, $\mathbf{t}$ is a treatment or intervention, $\mathbf{y}$ is the outcome, $\mathbf{z}$ is an unobserved confounder and $\mathbf{x}$ is a noisy view of this hidden confounder. Work by Kim *et al.* (2021*b*) builds on CEVAE by introducing additional variables within the causal structure, and propose a different factorization of the joint which is encoded via the generative model architecture. These CEVAE-based approaches fit data to a predefined causal structure which is independent of the dataset being considered. Sharma *et al.* (2021) instead use an assumed causal structure between the variables of the dataset, as represented by a BN DAG, by encoding this in a conditional VAE (Sohn *et al.*, 2015) to perform survival analysis. Unlike the Graph VAE of He *et al.* (2019*a*), they only consider conditional independencies between the observed variables, and make use of a simple Gaussian distribution as a prior over the latent variables.

Johnson *et al.* (2016) propose a framework that combines the flexibility of VAEs with the tractability of conjugate graphical model inference. They use a conditional random field (CRF)—a type of undirected PGM—as a variational family to capture the structure in the latent space, and learn inference networks that output the required conjugate graphical model potentials in lieu of the variational distribution's parameters. These potentials are then used in a graphical model inference algorithm such as message passing.

Although the list of works presented in this chapter is far from comprehensive, it provides a clear indication of the general interest in combining the strengths of modern deep learning techniques and traditional graphical models, specifically BNs, NFs and VAEs. In the next chapter, we present our approach to encoding conditional independencies in a residual flow.

Chapter 4

Graphical Residual Flows

The graphical flows presented in Section 3.1 do not emphasize providing stable and accurate inversion, required for using a single flow in both the normalizing and generative direction. All NFs are theoretically invertible, but stable inversion is not always guaranteed in practice (Behrmann *et al.*, 2021): if the Lipschitz constant of the inverse flow transformation is too large, numerical errors may be amplified, leading to exploding inverses. Residual flows (Chen *et al.*, 2019) obtain stable inversion as a byproduct of their Lipschitz constraints, but do not encode domain knowledge about the target distribution's dependency structure.

In this chapter, we therefore propose the graphical residual flow (GRF), which encodes domain knowledge from a BN into a residual flow in a manner similar to the graphical NFs of Wehenkel and Louppe (2021) and Weilbach *et al.* (2020). This model is presented in more detail in Section 4.1. These flows capture a predefined dependency structure through masking of the residual blocks' weight matrices. Previous masking schemes required either D passes through the neural network (Wehenkel and Louppe, 2021) or restricted the width of the network's hidden layers (Weilbach *et al.*, 2020). We therefore propose a novel scheme, based on the MADE approach discussed in Section 2.3.1.1, that overcomes both these shortcomings. This new masking scheme is presented in Section 4.2.

In contrast to traditional residual flows, the incorporation of dependency information also leads to tractable computation of the exact Jacobian determinant, which is discussed in Section 4.3. We continue in Section 4.4 by considering the application of these flows to the task of amortized variational inference, in which case one has to condition the flow on a given observation. In Section 4.5, we investigate whether the distribution represented by a GRF does indeed encode all the conditional independencies specified by the provided BN. GRFs cannot be inverted analytically, and one has to resort to numerical methods as discussed in Section 4.6. Stable and accurate inversion is however achieved by constraining a Lipschitz bound on the transformation which arises from the invertible residual network architecture. In Section 4.7 we examine the Lipschitz bounds of GRFs compared to existing approaches, and show that previous graphical flows do not offer the same inversion stability guarantees. Lastly in Section 4.8, we introduce a new Lipschitz-constrained activation function that is based on the Mish function proposed by Misra (2020). Since Misra (2020) found that Mish either matched or improved the performance of the non-monotonic function Swish (Ramachandran *et al.*, 2017), we hope that this function will give similar performance gains over the Lipschitz constrained LipSwish function used by Chen *et al.* (2019) in residual flows. The discussion in this chapter is based in part on the work presented in Mouton and Kroon (2022a).

4.1 Encoding Structure in Residual Flows

Assume a residual flow $F = f_T \circ \ldots \circ f_1$, where each residual block g_t, $t = 1, \ldots, T$, is a fully-connected neural network with a single hidden layer[1] and activation function $h(\cdot)$ where $\mathrm{Lip}(h) \leq 1$:

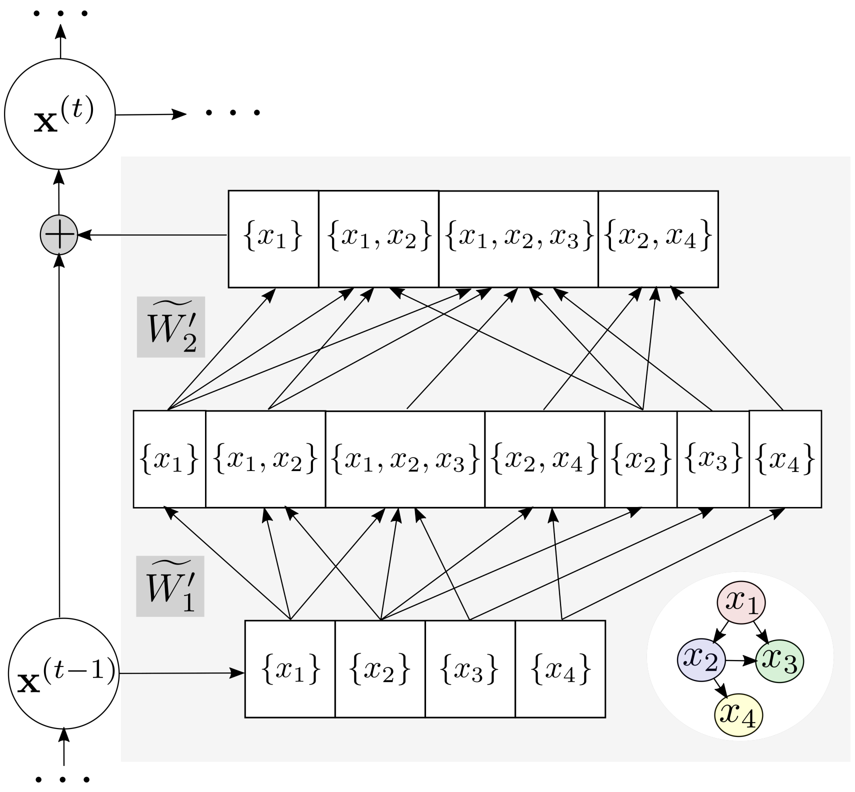
$$\mathbf{x}^{(t)} := f_t(\mathbf{x}^{(t-1)}) = \mathbf{x}^{(t-1)} + \widetilde{W}_2 \cdot h(\widetilde{W}_1 \cdot \mathbf{x}^{(t-1)} + \mathbf{b}_1) + \mathbf{b}_2 \ . \tag{4.1}$$

Here, $\widetilde{W}_i$ indicates a normalized weight matrix as in Equation (2.29), such that $\mathrm{Lip}(g_t) < 1$. Similar to the work of Wehenkel and Louppe (2021) and Weilbach *et al.* (2020), we can encode the graphical structure of a BN, by ensuring that output i of each f_t is only a function of those inputs corresponding to x_i and its parents in the BN graph. This can be achieved by suitably masking the weight matrices of each of the above residual blocks before applying spectral normalization. Given a BN graph, $\mathcal{G}$, over the components of $\mathbf{x} \in \mathbb{R}^D$, let $W_i' = W_i \odot M_i$, $i = 1, 2$, be the new masked weight matrices where $\odot$ denotes element-wise multiplication, and the M_i are binary masking matrices ensuring that component j of the residual block's output is only a function of the inputs corresponding to $\{x_j\} \cup \mathrm{Pa}^{\mathcal{G}}_{x_j}$. The update to $\mathbf{x}^{(t-1)}$ in block f_t is then defined as follows:

$$\mathbf{x}^{(t)} := \mathbf{x}^{(t-1)} + \widetilde{W_2'} \cdot h(\widetilde{W_1'} \cdot \mathbf{x}^{(t-1)} + \mathbf{b}_1) + \mathbf{b}_2 \ . \tag{4.2}$$

The resulting flow is called a graphical residual flow (GRF). The masks M_1 and M_2 are constructed according to a variant of MADE (Germain *et al.*, 2015). MADE by default facilitates only fully-connected BN structures; our generalization of MADE allows for arbitrary graphical dependencies and is discussed in more detail in the next section. Figure 4.1 provides an illustration of the structured update applied to $\mathbf{x}^{(t-1)}$ by f_t, as given in Equation (4.2). Section 4.5 discusses the dependency structure induced between the variables of the flow when composing a number of these blocks.

[1]The rest of this discussion is easily extended to residual blocks with more hidden layers.



$\mathbf{x}^{(t)}$
$\mathbf{x}^{(t-1)}$
$\widetilde{W}_2'$
$\widetilde{W}_1'$
$\{x_1\}$ $\{x_1, x_2\}$ $\{x_1, x_2, x_3\}$ $\{x_2, x_4\}$
$\{x_1\}$ $\{x_1, x_2\}$ $\{x_1, x_2, x_3\}$ $\{x_2, x_4\}$ $\{x_2\}$ $\{x_3\}$ $\{x_4\}$
$\{x_1\}$ $\{x_2\}$ $\{x_3\}$ $\{x_4\}$
x_1 x_2 x_3 x_4

Figure 4.1: The update to $\mathbf{x}^{(t-1)}$ at flow step t of a graphical residual flow. Edges removed by the masks M_1 and M_2 are not shown. The remaining edges encode the graphical structure of the given BN. Matrices $\widetilde{W}_1'$ and $\widetilde{W}_2'$ are the masked and spectrally normalized weight matrices. The bias terms are omitted to reduce clutter. See Section 4.2 for a discussion on how associating each neural network unit with a specific subset of variables can be used to construct the masks.

4.2 Extending MADE for Arbitrary Graphical Structures

The original approach employed by MADE (Germain *et al.*, 2015) was aimed at enforcing an autoregressive structure between the variables of interest by suitably masking the weight matrices of a neural network (See Section 2.3.1.1). This autoregressive structure is equivalent to encoding a fully-connected BN graph. We are interested in using a similar approach to encode a BN with an arbitrary dependency structure.

Given a BN graph $\mathcal{G}$, the corresponding joint distribution over the variables $\mathbf{x} \in \mathbb{R}^D$ will factorize as follows:

$$p(\mathbf{x}) = \prod_{i=1}^{D} p_i(x_i|\mathrm{Pa}_{x_i}^{\mathcal{G}}) \ . \tag{4.3}$$

For a given residual block's neural network that takes $\mathbf{x}$ as input, the goal is to have the output units associated with x_i be computed from only those input units associated with x_i and its parents. This means that there should be no computational paths between an input and an output unit if there is no direct dependency between the associated variables in $\mathcal{G}$. This can be achieved by applying a masking matrix to the weights of each neural network layer (which can be of arbitrary width) such that at least one weight on any such computational path is set to zero.

We begin by assigning a specific subset of variables to each unit in the neural network. Specifically, each input unit is assigned a unit set containing its corresponding input variable: $\{x_i\}$. Each output unit is assigned a set consisting of its associated variable and that variable's parents in the BN: $\{x_i\} \cup \mathrm{Pa}_{x_i}^{\mathcal{G}}$. Lastly, each hidden unit is uniform randomly assigned one of the following sets: $\{x_i\}$ or $\{x_i\} \cup \mathrm{Pa}_{x_i}^{\mathcal{G}}$ where i can be any of $1, \ldots, D$.[2]

A correct mask can then be constructed by ensuring that it zeroes out any weight between two neural network units if the set assigned to the unit in the next layer is not a superset of the set assigned to the unit in the previous layer. This has the implication that any path from input to output for any variable with one or more parents has a single associated set switch from $\{x_i\}$ to $\{x_i\} \cup \mathrm{Pa}_{x_i}^{\mathcal{G}}$. Algorithm 1 details the above masking scheme and Figure 4.2 provides an illustration of how this scheme encodes the desired dependency structure in the residual block for the BN given in Figure 4.1.

[2]To prevent situations where there are no valid paths from an input to the corresponding output, we require at least one unit in each hidden layer to be associated with $\{x_i\}$. Also note that MADE typically does not include $i = D$ as an input, since each output j is only dependent on the inputs $i = 1, \ldots, j-1$, and so no output units will be dependent on the highest input dimension. For GRFs however, we require self-dependence, i.e. output j should also be a function of input j. Otherwise we will simply have an affine transformation with a fixed scaling factor of one.

Algorithm 1 Computing mask matrices for arbitrary graphical structures.

```
Require: BN graph $\mathcal{G}$, hidden layer widths $H_1, \dots, H_L$
1: $D \leftarrow |\mathcal{G}|$                                  ▷ Input/output dimension
2: $M_1 \leftarrow \text{zeros}(H_1, D)$          ▷ Initialize first mask as $H_1 \times D$ matrix of zeros
3: for $l \leftarrow 2$ to $L$ do
4:     $M_l \leftarrow \text{zeros}(H_l, H_{l-1})$
5: end for
6: $M_{L+1} \leftarrow \text{zeros}(D, H_L)$
7:
8: $possible\_assignments = [\,]$        ▷ List of possible assignments to hidden units
9: for $i \leftarrow 1$ to $D$ do
10:     $possible\_assignments.\text{append}(\{i\})$
11:     $possible\_assignments.\text{append}(\{i\} \cup \text{Pa}_i^{\mathcal{G}})$
12: end for
13: for $l \leftarrow 1$ to $L$ do                  ▷ Initialize subsets assigned to hidden units
14:     $hidden\_subsets[l] = [\,]$
15:     for $i \leftarrow 1$ to $D$ do                          ▷ Ensure at least one valid path
16:         $hidden\_subsets[l].\text{append}(\{i\})$
17:     end for
18:     for $i \leftarrow D + 1$ to $H_l$ do
19:         $j \sim \text{Uniform}(1, 2D)$
20:         $hidden\_subsets[l]\text{append}(possible\_assignments[j])$
21:     end for
22: end for
23:
24: for $h \leftarrow 1$ to $H_1$ do           ▷ Create mask between input and first hidden layer
25:     for $i \leftarrow 1$ to $D$ do
26:         if $hidden\_subsets[1][h] \supseteq \{i\}$ then
27:             $M_1[h, i] \leftarrow 1$
28:         end if
29:     end for
30: end for
31:
32: for $l \leftarrow 2$ to $L$ do                           ▷ Create masks between hidden layers
33:     for $h1 \leftarrow 1$ to $H_l$ do
34:         for $h2 \leftarrow 1$ to $H_{l-1}$ do
35:             if $hidden\_subsets[l][h1] \supseteq hidden\_subsets[l-1][h2]$ then
36:                 $M_l[h1, h2] \leftarrow 1$
37:             end if
38:         end for
39:     end for
40: end for
41:
42: for $i \leftarrow 1$ to $D$ do           ▷ Create mask between final hidden and output layer
43:     for $h \leftarrow 1$ to $H_L$ do
44:         if $\{i\} \cup \text{Pa}_i^{\mathcal{G}} \supseteq hidden\_subsets[L][h]$ then
45:             $M_{L+1}[i, h] \leftarrow 1$
46:         end if
47:     end for
48: end for
49: return $[M_1, \dots, M_{L+1}]$
```

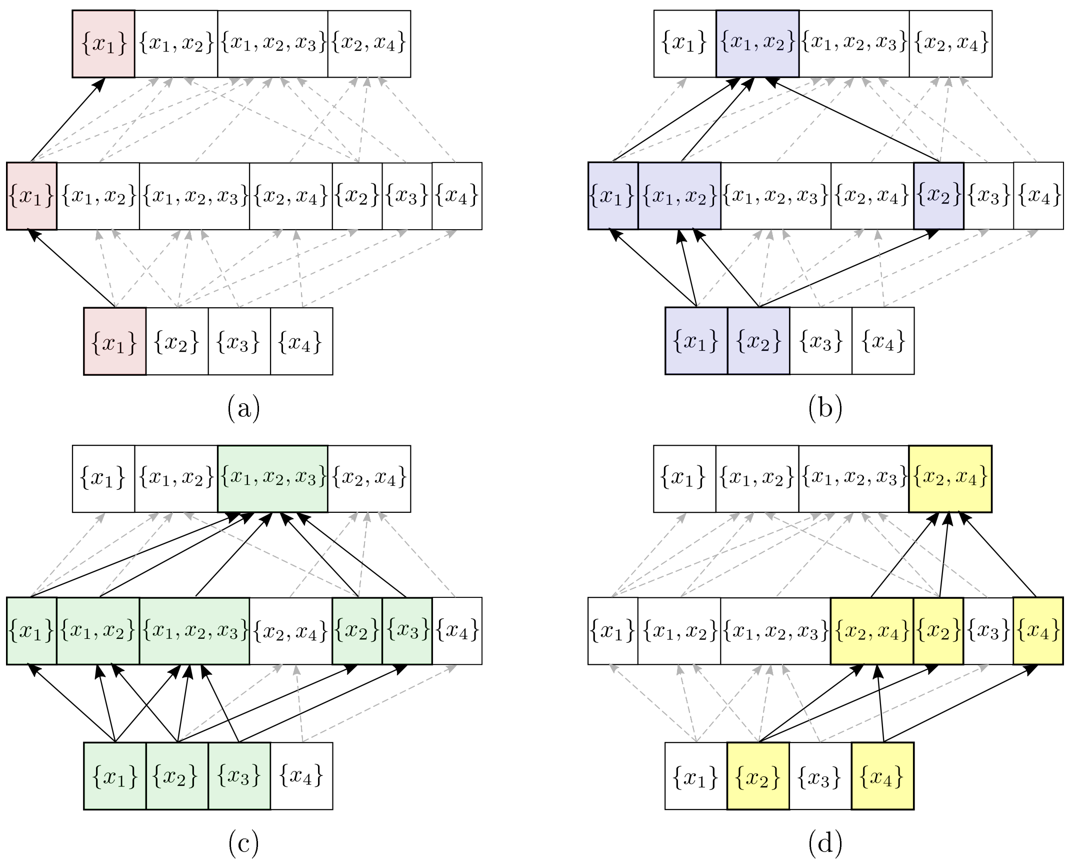

$\{x_1\}$ $\{x_1,x_2\}$ $\{x_1,x_2,x_3\}$ $\{x_2,x_4\}$
$\{x_1\}$ $\{x_1,x_2\}$ $\{x_1,x_2,x_3\}$ $\{x_2,x_4\}$ $\{x_2\}$ $\{x_3\}$ $\{x_4\}$
$\{x_1\}$ $\{x_2\}$ $\{x_3\}$ $\{x_4\}$
(a)
(b)
(c)
(d)

Figure 4.2: Mask construction. Edges are retained only when the set in the next layer is a superset of the set in the previous layer. Edges removed by the masks are not shown. The coloured units in each subfigure correspond to the computational path of the variable with the same colour in the BN in Figure 4.1. Since x_1 has no parent vertices in this BN, its update (a) will not depend on any other variables. The updates to variables (b) x_2, (c) x_3 and (d) x_4 at each flow step are only functions of the current states of these variable and their respective parents in the BN, as desired.

4.3 Computing the Jacobian Determinant

Since we are enforcing the BN's DAG dependency structure between the variables of the flow, we are in effect encoding a 'sparse' autoregressive structure. That is, if the variables were ordered according to their topological ordering in the BN, the update applied to each variable will only be conditioned on those variables with a strictly lower index (though not necessarily on *all* variables with a lower index). This construction would result in a triangular Jacobian $J_{f_t}(\mathbf{x})$ for which the determinant is easy to compute *exactly* as the product of the matrix's diagonal terms. This is in contrast to standard residual flows, which require approximation of the Jacobian determinant.

Note further that it is not necessary for the variables to actually be ordered in their topological ordering. It is sufficient to know that some permutation of the variables exists for which the correspondingly permuted version of the Jacobian is triangular. This is true, because the application of a permutation matrix exchanges both the rows and columns of the matrix, so the diagonal entries remain unchanged (just reshuffled). Furthermore, permutations are volume-preserving operations (i.e., the Jacobian determinants are ± 1) which will make no contribution to the change-of-variables formula, and can thus be ignored in practice. The logarithm of the Jacobian determinant of the complete flow F, follows as

$$\log|\det(J_F(\mathbf{x}))| = \sum_{t=1}^{T}\sum_{i=1}^{D} \log\left|\frac{\partial\left[f_t(\mathbf{x}^{(t-1)})\right]_i}{\partial\, x_i^{(t-1)}}\right| \tag{4.4}$$

$$= \sum_{t=1}^{T}\sum_{i=1}^{D} \log\left|1 + \frac{\partial\left[g_t(\mathbf{x}^{(t-1)})\right]_i}{\partial\, x_i^{(t-1)}}\right| \;. \tag{4.5}$$

If residual block g_t is constructed with a single hidden layer as in Equation 4.2, then the partial derivative of $[g_t(\mathbf{x}^{(t-1)})]_i$ with respect to input i, is given by:

$$\frac{\partial\left[g_t(\mathbf{x}^{(t-1)})\right]_i}{\partial\, x_i^{(t-1)}} = \left[\widetilde{W_2'} \cdot \mathrm{diag}\left(\frac{\partial\, h(\widetilde{W_1'} \cdot \mathbf{x}^{(t-1)} + \mathbf{b}_1)}{\partial\,\mathbf{x}^{(t-1)}}\right) \cdot \widetilde{W_1'}\right]_{ii} \;. \tag{4.6}$$

In practice one can use automatic differentiation to compute the above quantities, without explicitly computing the full matrix multiplications in this expression.

4.3.1 Reducing Memory Requirements

Naïve computation of Equation (4.5) does however have a drawback. Recall that $\log|\det(J_F(\mathbf{x}))|$ is used as part of the objective when training a finite flow. Vanilla (stochastic) gradient descent optimization with backpropagation, computes the gradient of the objective at the end of a forward pass. Since the forward pass involves T flow steps, this requires storing the computational graphs of all T Jacobian determinant computations in order to perform backpropagation for gradient computation. Affine flows, for example, primarily only need to store the computation graph of the output of the conditioner function in Equation (2.20) for optimization, since the scaling components are already the required diagonal terms of the Jacobian as given Equation (2.22). GRFs, on the other hand, require storing the computation graphs of the outputs of the residual blocks *as well as* the first derivatives of these outputs with respect to the inputs. Since these computational graphs are much more involved, training can consume a sizeable amount of memory, that increases linearly with the number of flow steps used.

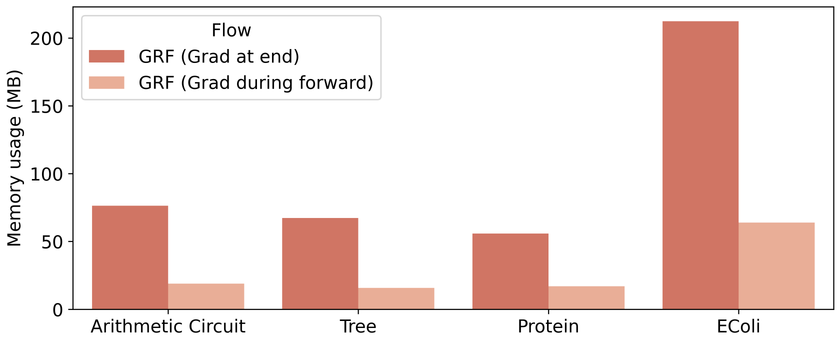


Figure 4.3: Peak memory usage during training of flow models for density estimation on various datasets (Arithmetic Circuit, Tree, Protein and EColi). Note the memory savings obtained by partially computing the gradients during the forward pass.[3]See Section 5.1.1 for further information on the datasets and flow models.

As in Chen *et al.* (2019), we overcome excessive memory usage by partially computing the gradients *during* the forward pass. To achieve this, we leverage the fact that the gradient of the log Jacobian determinant term for the entire flow decomposes across the flow steps and can be written as the sum of the gradients of the individual flow steps' log Jacobian determinants:

$$\nabla_\theta \log|\det(J_F(\mathbf{x}))| = \sum_{t=1}^{T} \nabla_\theta \log|\det(J_{f_t}(\mathbf{x}^{(t-1)}))| \ . \tag{4.7}$$

As a consequence, we can compute the contribution of each block's transformation to the gradient of the flow's log Jacobian determinant directly after completing the forward pass *for that block*. This allows one to immediately release the memory associated with the computational graph of $\log|\det(J_{f_t}(\mathbf{x}^{(t-1)}))|$, which results in a significant reduction in peak memory consumption during training for graphical residual flows. Figure 4.3 provides an indication of the memory-savings that can be achieved when partially computing the gradients at each flow step during a forward pass rather than at the end, as is traditionally done.

4.4 Variational Inference

In the previous sections we presented the GRF in the context of density estimation where all variables are observed and we apply the flow in the normalizing direction to obtain the density of a datapoint $\mathbf{x}$. As introduced in

[3]This memory-saving technique is a general improvement first used by Chen *et al.* (2019). Memory consumption was, however, not an issue for the datasets we considered in this thesis when evaluating GRFs.

Section 2.3.3.2, flows, and specifically GRFs, can also be applied in the context of variational inference, where we have access to a latent variable model $p(\mathbf{x}, \mathbf{z})$ and wish to obtain a good approximation to the posterior distribution of the latent variables, $p(\mathbf{z}|\mathbf{x})$. In this setting, one typically only has access to the forward BN that models the generating process for an observation $\mathbf{x}$. That is, the BN generally encodes the following factorization of the joint: $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$.

In order to construct a flow that is conditioned on the observation $\mathbf{x}$ and which takes the BN's dependency structure into account, we must first *invert* the BN. That is, given a graphical model capturing independence assumptions about the generative model, we wish to find a graphical model capturing the implied independence information about the corresponding posterior. Figure 4.4 provides a comparison of various inversion schemes for a simple BN. A heuristic approach is to simply invert the directionality of the edges in the BN (Kingma and Welling, 2014; Ranganath *et al.*, 2015). A more principled, yet still heuristic, approach is to invert the directionality of the edges and additionally insert edges between variables that share children in the forward BN (Stuhlmuller *et al.*, 2013; Paige and Wood, 2016). However, both of these approaches retain conditional independencies that may not be present in (or 'are not faithful to') the original distribution, and therefore cannot accurately represent the true posterior. An alternative method is therefore to make no independence assumptions at all about the posterior distribution, and use a fully-connected graph for the inverted BN (Le *et al.*, 2017). The drawback of this approach is that it ignores information available in the forward BN, which can lead to reduced performance for finite network capacities and training budgets.

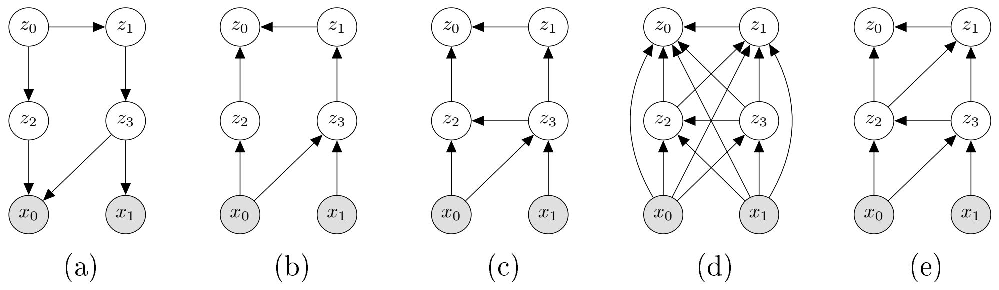


Figure 4.4: BN inversion schemes. Given (a) the forward BN, heuristic approaches either (b) only invert the edges or (c) invert all the edges and connect the vertices that share children in the forward BN. These approaches do not capture the true independence assumptions encoded by the forward BN, which could lead to poor modelling performance. Alternatively, one could make no independence assumptions and (d) use a fully-connected BN, but this ignores information available to us via the forward BN. Instead, Webb *et al.* (2018) propose an algorithm for providing (d) an inverse that encodes independencies faithful to the forward BN and is minimal in terms of its number of edges.

Algorithm 2 Graph Inversion (Webb *et al.*, 2018)

```
Require: BN graph $\mathcal{G}$, latent variables $Z$
 1: $\mathcal{J} \leftarrow$ moralize($\mathcal{G}$)  ▷ Add undirected edges between variables that share a
    child in $\mathcal{G}$ and remove directionality of the rest of the edegs
 2: Set all vertices of $\mathcal{J}$ to be unmarked
 3: $\mathcal{H} \leftarrow \{\text{variables}(\mathcal{G}), \emptyset\}$          ▷ Create a fully disconnected graph
 4: $S \leftarrow$ all $z \in Z$ without parent latent in $\mathcal{G}$
 5: while $S \neq \emptyset$ do
 6:     Select $v \in S$ that adds the fewest edges to $\mathcal{J}$ in the next step
 7:     Add edges in $\mathcal{J}$ between unmarked neighbours of $v$
 8:     Make unmarked neighbours of $v \in \mathcal{J}$, $v$'s parents in $\mathcal{H}$
 9:     Mark $v$ and remove from $S$
10:     for unmarked latents child $u$ of $v$ in $\mathcal{G}$ do
11:         Add $u$ to $S$ if all its parent latents in $\mathcal{G}$ are marked
12:     end for
13: end while
14: return $\mathcal{H}$
```

Fortunately, a more elegant solution has been provided by Webb *et al.* (2018), who propose a tractable algorithm for computing an approximate minimal inverse faithful to the graphical structure of the generative model. This inverse is *faithful* in the sense that it does not encode any independence statements not implied by the generative model, and it is *minimal* in that it is a local minimum in terms of the number of required edges (corresponding to a local maximum in terms of the number of true independence statements that it does encode). We can therefore use this algorithm to generate faithful and minimal inverse BNs to use when constructing flows as variational inference artifacts. Algorithm 2 provides the pseudocode for inverting a BN according to Webb *et al.* (2018). Note that their algorithm produces *natural* inverses, i.e. the topological ordering of the latent variables in the inverted BN is the inverse of the topological ordering of the latent variables in the forward BN (although it is also possible to obtain in inverse where the order of the random choices remains the same as in the forward BN).

Having obtained an inverted BN that encodes the joint factorization: $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}|\mathbf{x})p(\mathbf{x})$, we can now construct a generative GRF over the latent variables that is conditioned on $\mathbf{x}$. Each block of this flow is defined as follows for residual blocks with a single hidden layer:

$$\mathbf{z}^{(t)} = f_t(\mathbf{z}^{(t-1)}; \mathbf{x}) := \mathbf{z}^{(t-1)} + \widetilde{W_2'} \cdot h(\widetilde{W_1'} \cdot \mathbf{y}^{(t-1)} + \mathbf{b}_1) + \mathbf{b}_2 \ , \tag{4.8}$$

where $W_i' = W_i \odot M_i$, for $i = 1, 2$, $\mathbf{y}^{(t-1)} = \mathbf{z}^{(t-1)} \oplus \mathbf{x}$ with $\oplus$ denoting concatenation, and $\mathbf{z}^{(0)} = \boldsymbol{\epsilon} \sim p_0$ is a sample from the known base distribution. The binary masks M_1 and M_2 are again constructed using the method detailed in Section 4.2. When the above flow is used as a variational inference artifact q, the training objective is to maximize the ELBO, $\mathbb{E}_{\mathbf{z}\sim q}[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})]$.

Imposing a graphical structure on the update in Equation (4.8) does, however, place a lower bound on the number of flow steps, T, that should be used. Since the task is to infer the latent distribution given an observation $\mathbf{x}$, it is necessary that the information contained in $\mathbf{x}$ is shared with all latent variables. Given the above flow construction, note that after one flow step, only those latent variables that are children of $\mathbf{x}$ in the inverted BN will have received information about $\mathbf{x}$. After the next flow step, all latent variables within distance of two of an observed variable in the inverted BN will have received some information regarding the observed state through their parent vertices. Following this pattern, one sees that the required number of flow steps is dependent on how long it takes for information to propagate from $\mathbf{x}$ to all of $\mathbf{z}$. Specifically, T should be chosen such that

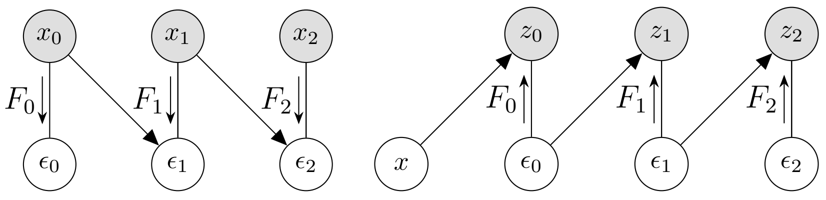
$$T \geq \max_{x_i \in \mathbf{x}, z_j \in \mathbf{z}} d(x_i, z_j) \ , \tag{4.9}$$

where $d(\cdot)$ is the graph distance between any observed and latent vertex, with $d(x_i, z_j) = 0$ when there is no path between x_i and z_j. That is, T should be greater than or equal to the 'longest shortest path' between any observed and latent vertex.

4.5 Formalizing the Encoded Dependency Structure

In the preceding sections, we discussed an approach to incorporating a predefined graphical structure into residual flows. These graphical residual flows can be applied in either the normalizing direction to calculate the density of an observation, or in the generative direction to sample a latent variable from the inferred posterior while conditioning on an observation. We include a given dependency structure in the manner discussed above as an inductive bias in the flows we construct. We now investigate whether the distribution represented by a given GRF does indeed encode all the conditional independencies specified by the provided BN.

We first consider a normalizing GRF with a single flow step, $F(\mathbf{x}) = \boldsymbol{\epsilon}$, which encodes the following BN chain structure: $x_0 \rightarrow x_1 \rightarrow x_2$. Let the bijective transformations applied to each of the dimensions be given by $F_0(x_0) = \epsilon_0$, $F_1(x_1; x_0) = \epsilon_1$ and $F_2(x_2; x_1) = \epsilon_2$, where each transformation is conditioned on a subset of $\mathbf{x}$, as encoded by the masking scheme in line with the provided BN. A graphical illustration of this flow is given in Figure 4.5a. Note that these individual bijective transformations are implemented using a single residual block neural network, as presented in Section 4.1. We treat them



(a) Normalizing GRF (b) Generative GRF

Figure 4.5: A graphical illustration of the transformation applied by (a) a one-step normalizing GRF and (b) a one-step generative GRF when encoding the chain dependency structures: $x_0 \to x_1 \to x_2$ and $x \to z_0 \to z_1 \to z_2$, respectively. The variables of the base distribution are represented by ϵ_0, ϵ_1 and ϵ_2. Undirected edges represent a bijective transformation (F_0, F_1 or F_2) between the associated variables with the small arrow indicating the direction of the forward mapping of the flow. Directed edges indicate the additional variables these bijective transformations are conditioned on as specified by the given BN and enforced by the presented masking scheme.

as separate functions here to simplify the discussion, and also no not consider their dependence on the parameters of this network. We are interested in whether the distribution represented by the flow, $p(\mathbf{x})$, respects the conditional independence assumptions specified by this BN.

As an example, we note that after a single transformation step, the distribution of x_2 will only depend on x_1 as desired, since it can be computed as: $\log p(x_2|x_1) = \log p_0(F_2(x_2|x_1)) + \log|\det(J_{F_2}(x_2;x_1))|$. It is easy to see that for any variable, its density under the flow can be computed knowing only its parents, and that it is thus conditionally independent of all other ancestors in the BN. Therefore, the distribution represented by a normalizing GRF with a single transformation step will adhere to the conditional independencies specified by the BN, which is in line with the argument presented in Section 3.1. This is because the bijective transformation associated with each variable will only be conditioned on the parents of that variable within the BN graph.

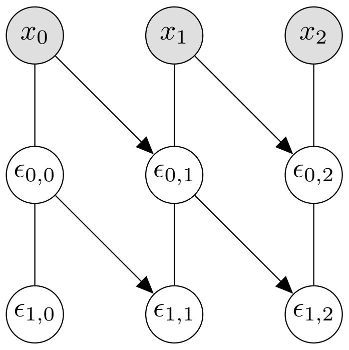


Figure 4.6: A normalizing GRF with two transformation steps.

Additional dependencies are however introduced when the number of flow steps is increased. Considering Figure 4.6, which depicts a 2-step normalizing GRF, one can note that when computing the latent representation of x_2, information will 'leak' from x_0 via the intermediate transformations of the observed variables, in this case $\epsilon_{0,1}$. If enough transformation steps are applied, the distribution of any observed variable will ultimately depend on all its ancestors in the BN graph. As a result, the encoded structure may end up corresponding to a graph with more connections than the original BN. In the worst case, this BN structure may correspond to the transitive closure of the original BN structure.

Next, consider a *generative* GRF with a single flow step, $\mathbf{z} = F(\boldsymbol{\epsilon}; x)$ for $\boldsymbol{\epsilon} \sim p_0$, which encodes the following BN chain structure: $x \to z_0 \to z_1 \to z_2$. Again, let the bijective transformations applied to each of the dimensions be given by $z_0 = F_0(\epsilon_0; x)$, $z_1 = F_1(\epsilon_1; \epsilon_0)$ and $z_2 = F_2(\epsilon_2; \epsilon_1)$, where each transformation is conditioned on a subset of $\boldsymbol{\epsilon}$, and only F_0 is conditioned on the observation, x. A graphical illustration of this flow is given in Figure 4.5b. We are interested in whether the distribution of the generated samples, $q(\mathbf{z}|x)$, respects the conditional independence assumptions specified by this BN.

The BN specifies that z_2 is conditionally independent of z_0 and x, given z_1. That is, $q(z_2|z_0, z_1, x) = q(z_2|z_1)$ should hold in the distribution represented by the flow. Since z_2 is a bijective transformation of ϵ_2 conditioned on ϵ_1, we can compute:

$$\begin{aligned}
&\log q(z_2|z_0, z_1, x)\\
&= \log p_0(F_2^{-1}(z_2; \epsilon_1)) + \log \left|\det(J_{F_2^{-1}}(z_2; \epsilon_1))\right|\\
&= \log p_0(F_2^{-1}(z_2|F_1^{-1}(z_1;\epsilon_0))) + \log \left|\det(J_{F_2^{-1}}(z_2; F_1^{-1}(z_1;\epsilon_0)))\right|\\
&= \log p_0(F_2^{-1}(z_2; F_1^{-1}(z_1;F_0^{-1}(z_0; x)))) + \log \left|\det(J_{F_2^{-1}}(z_2; F_1^{-1}(z_1;F_0^{-1}(z_0; x))))\right| .
\end{aligned}$$

By expanding the expression in this way, we make clear the direct dependence of z_2 on z_0 and x—knowing only z_1 is not sufficient to specify $q(z_2|z_1, z_0, x)$. This dependence arises from the fact that the bijective transformation between z_2 and ϵ_2 is only specified once ϵ_1 is known, and ϵ_1 is a function of both z_1 *and* z_0. Thus, $q(z_2|z_0, z_1, x) \neq q(z_2|z_1)$. In this way, each variable could be dependent on all its ancestors in the BN, and the dependency structure induced by the flow may again ultimately correspond to the transitive closure of the encoded BN.

We therefore have that for both a normalizing and generative GRF, the dependency structure induced by the flow could correspond to the encoded BN's transitive closure. In the *worst-case* scenario, this transitive closure corresponds to a fully-connected graph, in which case one would seemingly not have gained any benefit from encoding the given structure in this way. The dependencies induced by these graphical flows are arguably more subtle, however. For normalizing GRFs, each variable only receives information from its ancestors via intermediate bijective transformations of its parents. Even though there is some 'information leakage', it would not be unreasonable to expect that the distribution of a given variable will be more strongly influenced by its parents, rather than by the potentially 'diluted' information the variable receives about the rest of its ancestors. The degree of information leakage for generative GRFs is even less clear. This is primarily because each variable is in fact never a direct function of its parents, but rather depends on bijective transformations of these parents where each bijection is itself conditioned on those variables' parents. For example, we showed that knowing only the parent of z_2 is not sufficient to calculate its density (if $\boldsymbol{\epsilon}$ is unknown), and one additionally needs knowledge of its ancestors, including the observation x. When performing density estimation with a generative flow, this dependence is thus already introduced after only one flow step. When generating new samples, one however still needs three flow transformations for a generated sample of z_2 to contain *any* information about the observed state (similar to how $\epsilon_{1,2}$ only receives information about x_0 after two steps in the normalizing GRF depicted in Figure 4.6). Therefore, even though GRFs cannot in general guarantee that the distribution represented by the flow respects the independence statements specified by the encoded BN, we still expect these flows to incorporate an inductive bias that encourages the variables to adhere to the desired dependency structure. See Appendix A.1.4.2 for empirical results supporting this argument.

Future work is needed to analyse the extent of information leakage as well as its effect on overall performance, compared to alternative approaches that more strictly adhere to the BN structure. Any significant information leakage between outputs could for instance be estimated using conditional mutual information (Wyner, 1978), and an alternative model could for example use a separate flow at each node in the BN. For generative graphical flows, this construction would however require sequential evaluation of these sub-flows, since each node first requires samples from all its parent variables to be generated. As such, this approach would scale with the length of the longest path in the BN, which could slow down training and application of the model.

Another argument for why it is not necessarily crucial that we capture the exact BN in the flow model, is that BNs in the real-world are rarely guaranteed to be exactly correct and are usually obtained from either a structure learning process or are hypothesized by domain experts. Allowing the model to capture additional dependencies could therefore be beneficial in this setting, whilst still guiding the training process according to prior beliefs about the domain. Since GRFs incorporate BN dependency information in the same way as GNF and SCCNF (only the masking scheme differs—the overall dependency structure between the (final and intermediate) variables of the flow remain the same), the above arguments hold for these graphical flows as well.

We have now considered both transformation directions in which a GRF can typically be applied: in the normalizing direction for density estimation and in the generative direction for inference, as well as the dependency structure induced by flows applied in these different directions. Next, we consider how to invert these flows once they have been trained. This is necessary, for example, when one wants to generate new samples using a flow trained for density estimation, or when one wants to evaluate the density of sample $\mathbf{z}$ not generated by a flow trained in the generative direction.

4.6 Inverting GRFs

Similar to standard residual flows, the inverse of a GRF is not avaiable in closed form. Thus, we resort to numerical methods such as the Banach fixed-point approach (2.33) to compute $\mathbf{x} = f_t^{-1}(\mathbf{y})$ for each block f_t. From the Banach fixed-point theorem, we have that after n iterations of Equation (2.33), the bound on the inversion error is given by (Kreyszig, 1989, p. 302)

$$||\mathbf{x} - \mathbf{x}^{(n)}|| \leq \frac{\text{Lip}(g_t)^n}{1 - \text{Lip}(g_t)} ||\mathbf{x}^{(1)} - \mathbf{x}^{(0)}|| \ . \tag{4.10}$$

This shows that the Banach fixed-point approach converges exponentially in the number of iterations n, and that the rate is also dependent on the magnitude of the residual block's Lipschitz constant. While smaller Lipschitz constants lead to faster convergence and a smaller upper bound on the error, they also limit the expressivity of each flow step. Using the Banach fixed-point approach therefore introduces a trade-off between inversion speed and the flow's modelling capabilities with a certain number of steps.

Unlike general contractive residual flows, we showed in Section 4.3 that we are able to compute the diagonal entries of the Jacobian matrix exactly in the GRF setting. This derivative information allows us to use an alternative fixed-point iteration method that does not force this trade-off. Specifically, we can invert each block numerically using the Newton-like fixed-point method proposed by Song *et al.* (2019). To compute $\mathbf{x} = f_t^{-1}(\mathbf{y})$, the following update is applied until convergence:

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \alpha(\text{diag}(J_{f_t}(\mathbf{x}^{(n)})))^{-1}[f_t(\mathbf{x}^{(n)}) - \mathbf{y}] \ , \tag{4.11}$$

using the initialization $\mathbf{x}^{(0)} = \mathbf{y}$. This is an approximation of the relaxed multivariate Newton's method for inverting this flow, which is given by $\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \lambda(J_{f_t}(\mathbf{x}^{(n)}))^{-1}[f_t(\mathbf{x}^{(n)}) - \mathbf{y}]$, with a relaxation factor $0 < \lambda < 2$. Using only the diagonal terms instead of computing and inverting the full Jacobian, speeds up the inversion process at the potential cost of less effective individual iterations. Song *et al.* (2019) show that the iterative method presented in Equation (4.11) will converge *locally* for $0 < \alpha < 2$. Note that this convergence is not necessarily to the correct $\mathbf{x}$: we implicitly rely on the assumption that it is unlikely inversion will fail using the initialization $\mathbf{x}^{(0)} = \mathbf{y}$. We empirically demonstrate that the convergence rate of this inversion procedure does not exhibit the same dependence on the Lipschitz constant as the Banach fixed-point

Algorithm 3 Fixed-point iteration to compute $\mathbf{x} = f_t^{-1}(\mathbf{y})$ (Song *et al.*, 2019).

```
Require: $0 < \alpha < 2$
1: Initialize $\mathbf{x}_0 \leftarrow \mathbf{y}$
2: while not converged do
3:     Compute $f_t(\mathbf{x}_{n-1})$
4:     Compute $\text{diag}(J_{f_t}(\mathbf{x}_{n-1}))$
5:     $\mathbf{x}_n \leftarrow \mathbf{x}_{n-1} - \alpha(\text{diag}(J_{f_t}(\mathbf{x}_{n-1})))^{-1}[f_t(\mathbf{x}_{n-1}) - \mathbf{y}]$
6: end while
7: return $\mathbf{x}_n$
```

approach (see Section 5.3), and we are not aware of any theoretical bounds on the convergence rate that are dependent on the transformation's Lipschitz constant. We thus expect Equation (4.11) to perform better when larger Lipschitz bounds for the residual blocks (≈ 0.99) are used (which is desirable for more expressive flow steps). As with the Banach fixed-point approach, the number of iterations needed for convergence is not known beforehand. Algorithm 3 describes the pseudocode of this Newton-like fixed-point method.

4.7 Invertibility of Graphical Flows in Practice

The Lipschitz constants of the forward and inverse transformation of an invertible neural network quantify its worst-case stability (Behrmann *et al.*, 2021). Bounds on these constants therefore play an important role in understanding and mitigating possible exploding inverses. Below we compare the worst-case inversion stability of the different graphical normalizing flows implied by their corresponding Lipschitz constants. Since a graphical residual flow places a strict bound on its Lipschitz constant, this type of flow should be expected to provide better inversion stability than alternative approaches.

Affine Graphical Normalizing Flows

Recall that each flow step of a graphical normalizing flow (Wehenkel and Louppe, 2021, Section 3.1.1) with affine transformations (GNF-A) is given by:

$$f(\mathbf{x}) = \exp(s(\mathbf{x})) \odot \mathbf{x} + m(\mathbf{x}) \ , \tag{4.12}$$

where outputs i of $s(\cdot)$ and $m(\cdot)$ are functions of only those inputs corresponding to the parents of x_i in the associated BN graph. No global bounds can be placed on the Lipschitz constant of this type of flow, which complicates the task of ensuring stable inversion in all scenarios. Behrmann *et al.* (2021) provide the following simple illustration of why GNF-A only has local Lipschitz bounds. Assume $\mathbf{x}$ consists of two variables, x_0 and x_1, where x_1 is dependent on x_0 in the corresponding BN. Let $[F(\mathbf{x})]_1 = x_1 \exp(s(x_0))$ be the transformation applied to x_1 by a single-step GNF-A, where $s(x_0) = [s(\mathbf{x})]_1$ is the second output dimension of the conditioner function $s(\cdot)$ and the dependence on only x_0 has been made explicit. The output of the conditioner function $[m(\mathbf{x})]_1$ is taken to be 0 for simplicity. Then

$$\frac{\partial [F(\mathbf{x})]_1}{\partial x_0} = x_1 \frac{\partial \exp(s(x_0))}{\partial x_0} = x_1 \exp(s(x_0)) s'(x_0) \ . \tag{4.13}$$

Thus, if x_1 may grow arbitrarily large, this derivative will be unbounded, which could allow the Jacobian, $J_F(\mathbf{x})$, to have an unbounded Frobenius norm. Due to the equivalence of norms in finite dimensions, this in turn can induce an unbounded spectral norm of the Jacobian. Lastly, we consider the following theorem (Federer, 1996; Kim *et al.*, 2021*a*): if $F : \mathbb{R}^D \to \mathbb{R}^D$ is a Lipschitz continuous and differentiable function under the Euclidean norm, then

$$\text{Lip}(F) = \sup_{\mathbf{x} \in \mathbb{R}^D} ||J_F(\mathbf{x})||_2, \tag{4.14}$$

where $||\cdot||_2$ denotes the spectral norm. Based on Equation (4.14), we conclude that if the spectral norm of the Jacobian is unbounded, then no global Lipschitz bound can be obtained.

Monotonic Graphical Normalizing Flows

We can employ a similar illustration to investigate the Lipschitz bounds of a GNF with monotonic transformations (GNF-M), as presented in Section 3.1.1. Again, assume $\mathbf{x}$ consists of two variables, x_0 and x_1, where x_1 depends on x_0 in the corresponding BN. The transformation applied to x_1 by a single-step GNF-M is then given by $[F(\mathbf{x})]_1 = \int_0^{x_1} h(t, c_1(x_0))\, dt + \beta(c_1(x_0))$. We take the partial derivative of the above transformation and apply Leibniz's integral rule (see Appendix B.1.1) and the chain rule:

$$\begin{aligned} \frac{\partial [F(\mathbf{x})]_1}{\partial x_0} &= \frac{\partial}{\partial x_0} \int_0^{x_1} h(t, c_1(x_0))\, dt + \frac{\partial \beta(c_1(x_0))}{\partial x_0} \\ &= \int_0^{x_1} \frac{\partial h(t, c_1(x_0))}{\partial c_1(x_0)} \frac{\partial c_1(x_0)}{\partial x_0}\, dt + \frac{\partial \beta(c_1(x_0))}{\partial x_0} \ . \end{aligned} \tag{4.15}$$

The integrand above is the product of the derivatives of two neural networks with respect to their inputs. For general networks, this integrand's shape will depend not only on the chosen activation functions, but also the weights obtained during training. If x_1 may grow arbitrarily large, and if the area under the curve given by the integrand is not bounded by some maximum value as x_1 increases, then we can apply similar reasoning as above to show that this flow has no global Lipschitz bounds.

Thus, either the architecture of the flow must be adapted to ensure that this integral remains bounded as a function of $x_1 > 0$, or other techniques must be used to improve local stability, as discussed in Behrmann *et al.* (2021).

Structured Conditional Continuous Normalizing Flows

The structured conditional continuous normalizing flow (SCCNF) of Weilbach *et al.* (2020), is defined by a neural ODE, $\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t)$, where $t \in [0, 1]$, and f is a neural network masked such that it encodes a given BN dependency structure. A global Lipschitz bound for this transformation needs to hold for all $t \in [0, 1]$. This bound is a standard result and is given by Behrmann *et al.* (2021):

$$\mathrm{Lip}(F) \leq e^{\mathrm{Lip}(f)\cdot t} \ . \tag{4.16}$$

The inverse of an SCCNF is simply given by $\frac{d\mathbf{x}(t)}{dt} = -f(\mathbf{x}(t), t)$, and thus its Lipschitz bound is unchanged, i.e. $\mathrm{Lip}(F^{-1}) = \mathrm{Lip}(F)$.

Graphical Residual Flows

Residual flows implement strict Lipschitz bounds to ensure the transformation remains bijective. Since graphical residual flows only extend residual flows in terms of the connectivity between the variables of the flow, the same Lipschitz bounds will hold. Let $F(\mathbf{x}) = \mathbf{x} + g(\mathbf{x})$ be a transformation step in a GRF. By construction, $\mathrm{Lip}(g) < 1$, to ensure invertibility. A (non-trivial) upper bound on the Lipschitz constant of this contractive residual block g implies Lipschitz bounds for both F and F^{-1} (Behrmann *et al.*, 2019):

$$\mathrm{Lip}(F) \leq 1 + \mathrm{Lip}(g) \qquad \text{and} \qquad \mathrm{Lip}(F^{-1}) \leq \frac{1}{1 - \mathrm{Lip}(g)} \ . \tag{4.17}$$

This bi-Lipschitzivity is an attractive property for stable and efficient inversion of the flow step.

4.8 LipMish Activation Function

GRFs impose strict Lipschitz bounds to facilitate both theoretical and numerical invertibility. As such, the activation functions used in the residual blocks need to be chosen accordingly. Here, we propose a new Lipschitz-constrained activation function based on the Mish activation function of Misra (2020).

The training objective of the GRF contains the first derivatives of the residual block activation functions through the Jacobian term. Thus, the gradients used during training will depend on these activations' second derivatives. For many of the common smooth activation functions, the second derivative is

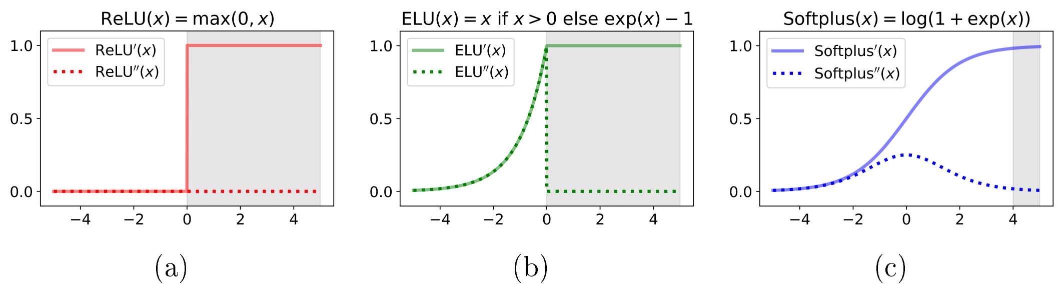


Figure 4.7: Vanishing second derivatives of common smooth activations with the required Lipschitz bound $\mathrm{Lip}(h) \leq 1$, when their first derivatives approach one. The first derivative of the residual block's activation function is present in the training objective of GRFs. The behaviour of the second derivative when the first derivative of these activation is close to one, as shown in the shaded areas, can thus lead to vanishing gradients.

close to zero ('vanishes') in the region where the first derivative reaches its maximum. For example, as shown in Figure 4.7a, the second derivative of the popular ReLU activation function is zero everywhere except at zero, and more specifically, it is zero everywhere that the first derivative is at its maximum of one. Even though the activation functions presented in Figure 4.7a satisfy the required Lipschitz constraint, their second derivative vanishes in the region that the first derivative is close to one. This issue is similar to the problem of saturated activation functions, only in this case it applies to the first and second derivative of the function. Instead, we would prefer an activation function, h, which adheres to the imposed Lipschitz bounds, i.e. $\mathrm{Lip}(h) \leq 1$, and for which the second derivative does not vanish in region where the first derivative is close to one. Some smooth *non-monotonic* activation functions with this behaviour have previously been identified (Chen *et al.*, 2019). For example, Chen *et al.* (2019) use a scaled version of the non-monotonic Swish activation function, $\mathrm{Swish}(x) = x/(1 + \exp(-\mathrm{softplus}(\beta) \cdot x))$ (Ramachandran *et al.*, 2017), called LipSwish, such that $\mathrm{Lip}(\mathrm{LipSwish}) \leq 1$:

$$\mathrm{LipSwish}(x; \beta) = \frac{1}{1.1} \cdot \left(\frac{x}{1 + e^{-\mathrm{softplus}(\beta) \cdot x}} \right) , \tag{4.18}$$

where an additional positive scaling factor is obtained by passing a parameter $\beta \in \mathbb{R}$ through a softplus function.

As an alternative, we instead consider the smooth non-monotonic activation function Mish (Misra, 2020): $\mathrm{Mish}(x) = x \tanh(\mathrm{softplus}(x))$. Similar to Swish, the Lipschitz constant of Mish is not bounded by one. Thus, a scaling factor is needed to ensure adherence to the imposed Lipschitz constraint. We call the resulting scaled activation the "LipMish" function:

$$\mathrm{LipMish}(x) = \frac{x}{1.088} \tanh(\mathrm{softplus}(\mathrm{softplus}(\beta) \cdot x)) .$$

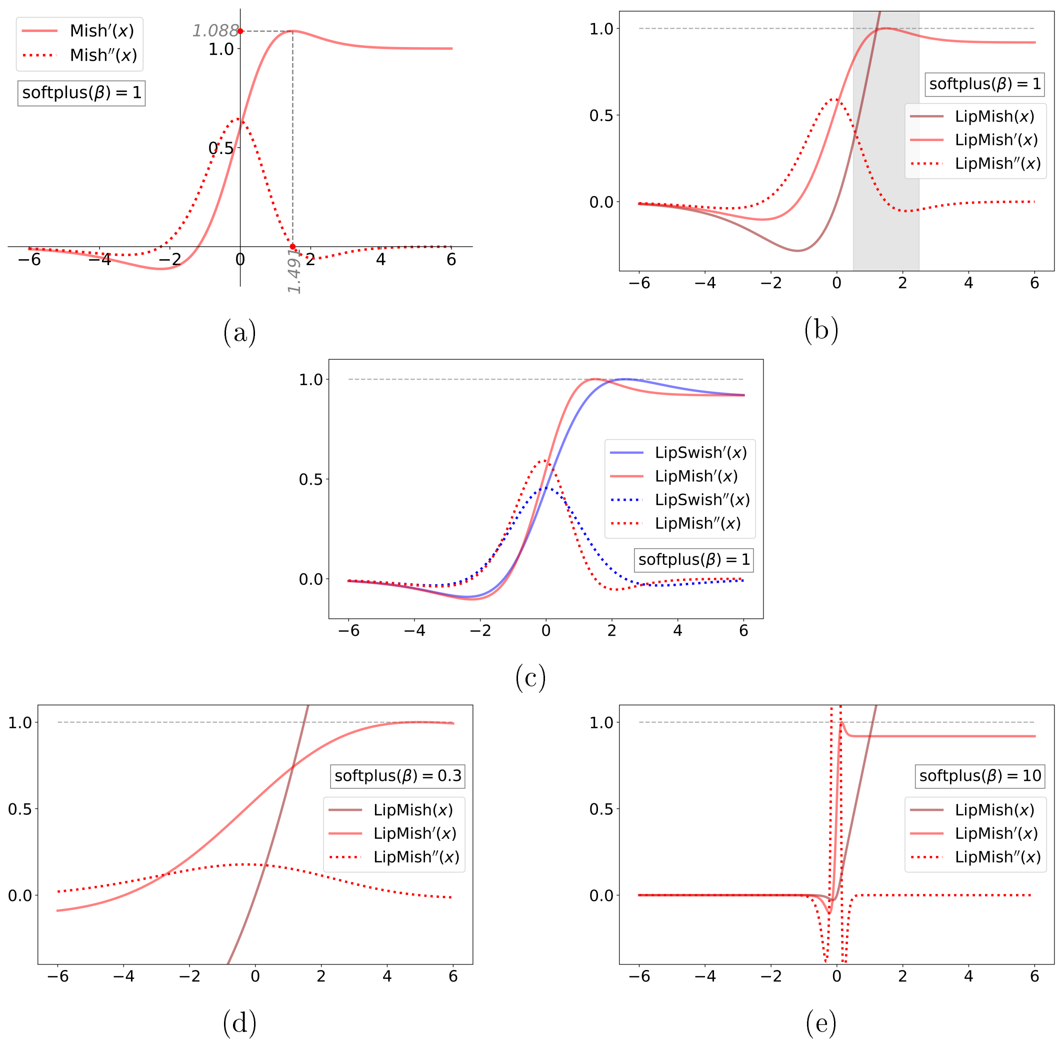


Figure 4.8: The LipMish activation function. Figure (a) indicates how the necessary scaling factor (1/1.088) is obtained: it is the point where the first derivative of Mish is maximal (which corresponds to the root of the second derivative). By scaling Mish with this factor, one obtains (b) a non-monotonic function, LipMish, that adheres to the constraint Lip(LipMish) ≤ 1, and whose second derivative does not vanish in the region where the first derivative is maximal. Figure (c) provides a comparison between the first and second derivatives of LipSwish and the proposed LipMish function. Similar to LipSwish, incorporating an additional parameter $\beta \in \mathbb{R}$, allows LipMish to have different degrees of curvature as shown in(d) and (e).

As with LipSwish, we have incorporated an additional positive scaling factor for x within the softplus function, which allows LipMish to have different degrees of curvature. For example, if we take the limit softplus$(\beta) \to 0$, LipMish becomes a linear function and if softplus$(\beta) \to \infty$, LipMish simplifies to ReLU. The scaling factor of (1/1.088) ensures that Lip(LipMish) ≤ 1 for all β. Figure 4.8 provides more details on how this scaling factor is obtained and illustrates different characteristics of LipMish.

4.9 Conclusion

In this chapter we have proposed a new graphical flow based on residual flows, that encodes the conditional independencies of a BN by applying a novel masking scheme to the weight matrices of the flow's residual blocks. We showed how this flow can be applied for the tasks of density estimation and variational inference, and discussed the dependency structure induced between the variables by the architecture of the flow. We also discussed the inversion guarantees GRFs provide compared to existing graphical flows. In the next chapter we proceed by evaluating GRFs on a range of datasets that each have an associated BN, and confirm that these flows provide accurate inversion performance.

Chapter 5

Empirical Investigation I: Graphical Residual Flows

This chapter presents an empirical investigation comparing the performance of our proposed GRF model to existing graphical flows. More specifically, the goals of this chapter were as follows:

1. To evaluate the key task performance of GRFs compared to existing finite and continuous graphical flows when encoding conditional independence information from a dataset's associated BN graph. The key tasks identified were:
   a) density estimation, and
   b) amortized inference.
2. To empirically analyse the accuracy and efficiency of inverting GRFs compared to existing approaches.

Section 5.1 gives an outline of the methodology employed to achieve these goals. This includes an overview of the datasets used, as well as the training setup and model architectures. Sections 5.2 and 5.3 provide and critically discuss the experimental results in line with goals 1 and 2 above, respectively.

5.1 Methodology

We evaluated our proposed GRFs by comparing them to existing graphical flow models using various performance metrics. These comparisons were performed using both synthetic and real-world datasets, where the domain problem represented by each dataset has an associated true or hypothesized BN structure (although we typically refer to the "dataset's BN" for short). We specifically compared GRFs to the graphical normalizing flow (GNF) proposed by Wehenkel and Louppe (2021) (see Section 3.1.1), and the structured conditional continuous normalizing flow (SCCNF) presented by Weilbach *et al.* (2020) (see Section 3.1.2). For GNFs, we considered both affine and monotonic transformation functions, denoted by GNF-A and GNF-M, respectively.

First, we measured the density estimation performance of the different approaches for a given set of observed variables, $\mathbf{x}$, of which the true joint distribution was assumed to factorize according to a given BN. Similarly, we measured the models' performance as variational inference artifacts for a given set of observed and latent variables. In this setting, the model $p(\mathbf{x}, \mathbf{z})$ was known and the structure of the inverse BN was obtained using the faithful inversion algorithm of Webb *et al.* (2018) discussed in Section 4.4. Lastly, the efficiency and accuracy of inverting the respective flows were analysed empirically. We also verified empirically whether the Newton-like inversion algorithm, given by Equation (4.11), requires fewer iterations per flow step than the Banach fixed-point approach and whether or not the number of required steps is dependent on the Lipschitz constant of the flow.

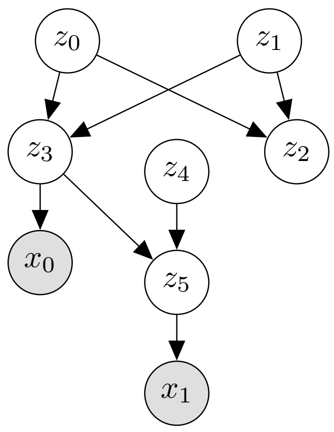
Below we provide more details regarding the datasets used during the investigation (Section 5.1.1) and the chosen model architectures and training procedures (Section 5.1.2).

5.1.1 Datasets & Bayesian Networks

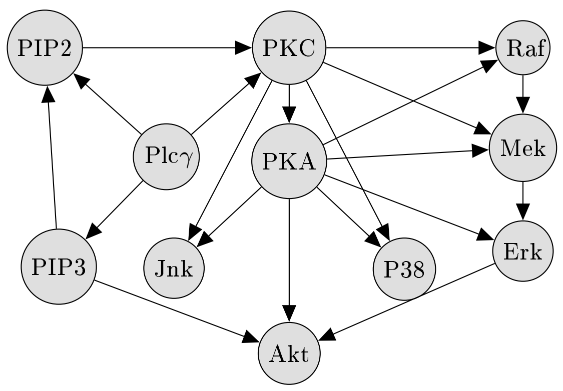
The various performance metrics of the different graphical flow models were evaluated using both synthetic and real-world datasets that each have an associated true or hypothesized BN structure. The synthetic datasets were generated from fully specified BNs. These consisted of the Arithmetic Circuit dataset used by Weilbach *et al.* (2020) and Wehenkel and Louppe (2021), an adaptation of the Tree dataset used by Wehenkel and Louppe (2021) as well as an adaptation of a Gaussian BN, EColi, from the BN repository of Scutari (2022). The Arithmetic Circuit and Tree BNs were designed to represent complex conditional distributions with non-trivial dependency structures, whereas the EColi BN is based on the assumed gene association network of the Escherichia coli bacteria. We also considered two real-word datasets. The first is related to human proteins (Sachs *et al.*, 2005), and the second, MEHRA, to the relationship between air pollution, weather and health (Vitolo *et al.*, 2018). Figure 5.1 illustrates the BN graphs associated with each of these datasets, with a summary of their properties given in Table 5.1. Further information on the real-world datasets and the specifications of the synthetic BNs is given in Appendix A.1.1. Since variational inference requires knowledge of the model $p(\mathbf{x}, \mathbf{z})$, which is unavailable for the real-world datasets, we did not consider these datasets for the variational inference tasks.

Table 5.1: A summary of the BN graphs associated with each dataset. D and K denote the number of observed and latent variables in each graph, respectively, and E denotes the number of directed edges. We also provide the longest shortest path between any observed and latent vertex reachable from that observation in the inverted graph, $\mathcal{G}^{-1}$. This is the minimum number of flow steps required for inference as discussed in Section 4.4.

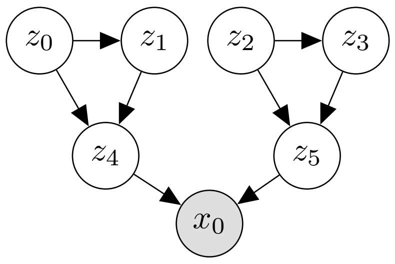
BN	Real/Synthetic	D	K	E	$\max\limits_{x_i, z_j \in \mathcal{G}^{-1}} d(x_i, z_j)$
Arithmetic Circuit	Synthetic	2	6	8	3
Tree	Synthetic	1	6	8	2
Protein	Real	11	0	20	N/A
EColi	Synthetic	29	15	59	4
MEHRA	Real	10	0	10	N/A



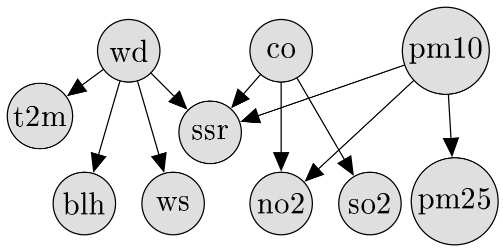
(a) Arithmetic Circuit



(b) Protein

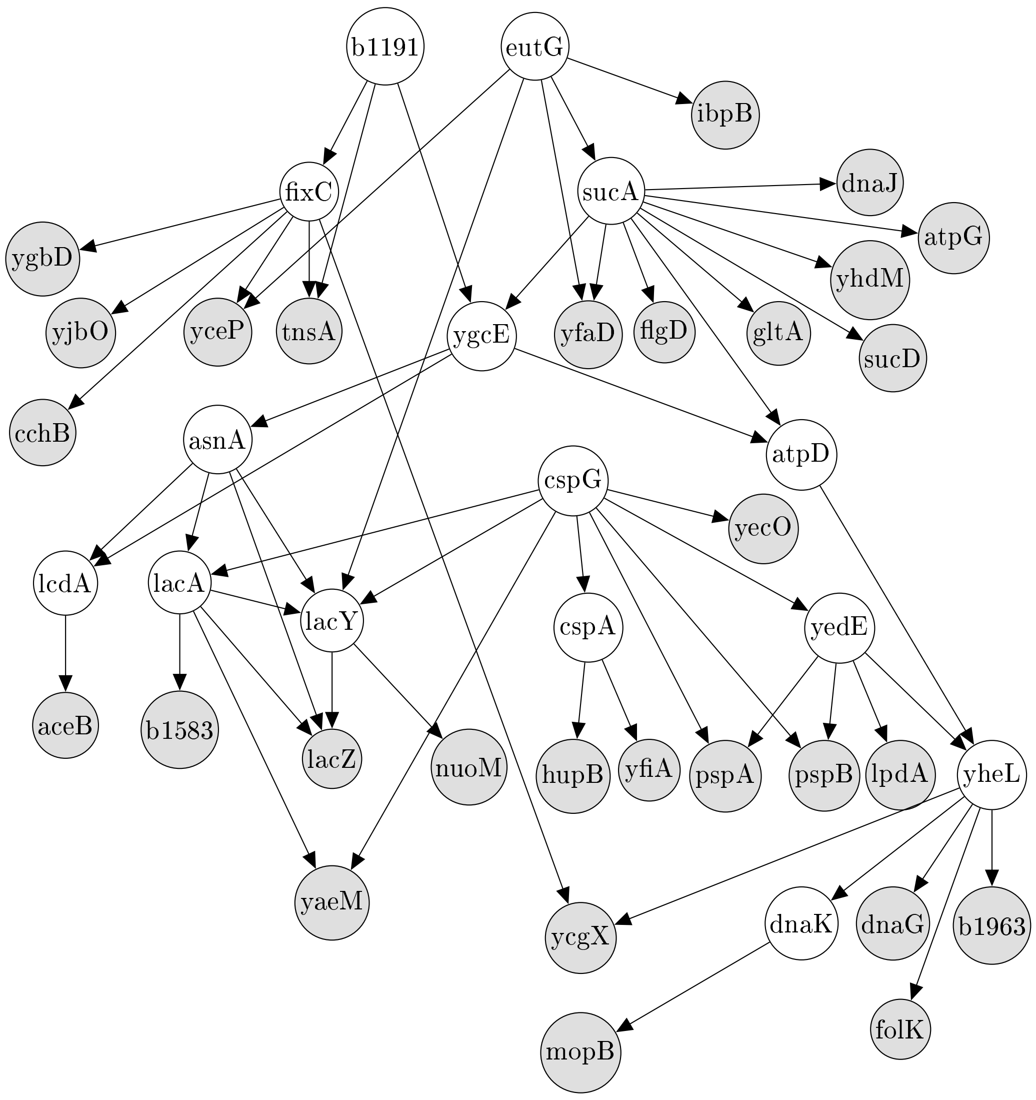


(c) Tree



(d) MEHRA

Figure 5.1 (Continued on following page): BN graphs associated with the graphical datasets. White nodes indicate latent variables, whereas shaded nodes are observed. Since the real-world datasets are only used in density estimation tasks, and not inference, all variables are observed. All datasets include values for all latent variables.



(e) EColi

Figure 5.1 (Continued): BN graphs associated with the graphical datasets.

5.1.2 Model Architectures & Training

Architectures To provide more informative comparisons between the flows, we considered two model sizes for each approach on each task. Smaller models had a maximum capacity of 5000 trainable parameters, and are denoted by a subscript S, e.g., GRF_S. Larger models had a maximum capacity of 15000 parameters, and are denoted by the subscript L. The details of the small and large flow architectures used for each of the datasets are given in Appendix A.1.2. The chosen architectures were obtained by performing a grid search over flow depth and neural network width, while using density estimation performance as a metric and adhering to the model size restriction. These same architectures were used for inference in each model size group, except when the number of flow steps was less than the longest shortest path between any observed and latent vertex, as given in Table 5.1. In these cases, the number of steps were increased to the appropriate value, while the hidden layer widths were decreased in order to comply with the parameter capacity restrictions.

Each residual block and conditioner neural network of the GRF and GNF models had a single hidden layer, since it was found that adding more hidden layers had very little impact on the overall performance compared to adding more flow steps. The integral neural network of all monotonic flows consisted of one hidden layer of size 100, with ELU activation functions on both the hidden and final layer. The activation functions used in the conditioner neural networks of GNF and in the main flow transformation neural network of SCCNF are the same as those used in the original studies of Wehenkel and Louppe (2021) and Weilbach *et al.* (2020). For all the GRF models we used the proposed LipMish activation function, which is shown to outperform LipSwish on the graphical datasets in Appendix A.1.4.1. The hyperparameter c, constraining the spectral norm in the graphical residual flow, was set to 0.99 in all cases. Setting c close to one is desirable, since it makes each flow step less restrictive, and thus allows better modelling performance with fewer transformation steps.

To further provide a fair comparison, we used a single masking scheme to encode conditional independencies in all of the graphical flow models. Since our proposed approach, as presented in Section 4.2, overcomes the shortcomings of the schemes employed by Wehenkel and Louppe (2021) and Weilbach *et al.* (2020), we used this approach to encode the information from a BN into not only the GRF, but also GNF and SCCNF. Appendix A.1.4.3 provides additional results supporting our use of this new masking scheme.

Training All flows were trained using the Adam optimizer (Kingma and Ba, 2015) (with the default setting for the parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$) with an initial learning rate of either 0.01 or 0.001 and a batch size of 100. The learning rate was decreased by a factor of 10 each time no improvement in the loss was observed for a set number of consecutive epochs, until a minimum learning rate of 10^{-6} was reached, at which point training was terminated. This 'reduce on plateau' learning is similar to the one used by Webb *et al.* (2018). The initial learning rate and duration before learning rate reduction was chosen based on the lowest validation loss obtained over the grid $\{0.01, 0.001\} \times \{10, 20, 30\}$. The training, validation and test sets of the synthetic datasets consisted of 10 000, 5000 and 5000 instances, respectively. The MEHRA dataset was randomly split into 4000 training, 1000 validation and 1885 test instances. For the Protein dataset we used the same test set as used by Wehenkel and Louppe (2021), which consists of 1672 samples. We randomly split their training set to obtain a training and validation set of 9000 and 1000 instances, respectively. All datasets were standardized before training.

All experiments were conducted on a machine with a 16-core Intel Core i9-11900 (2.5 GHz), 32 GB of RAM and a single Nvidia GeForce RTX 3070 Graphics Processing Unit.

5.2 Density Estimation & Inference Performance

Table 5.2 provides the average negative log-likelihood ($-\log p(\mathbf{x})$) achieved by each model on the test set for the different datasets over five independent runs.[1] Here, we assumed all variables to be observed, and used the BN structures illustrated in Figure 5.1. We also performed inference on the synthetic datasets. Table 5.3 provides the average ELBOs achieved by each model for these inference tasks. To further illustrate the characteristics of the different flow models, we also determined the density estimation performance as a function of the flow depth for the Protein dataset (where for the finite flows, flow depth refers to the number of flow steps, and for SCCNF, to the number of layers in its neural network). Figure 5.2 plots the average negative log-likelihood versus flow depth achieved over five independent runs for the different approaches. Except for the flow depth, the architectures of these models were the same as given in Table A.1, and did not necessarily adhere to a specific size budget.

We found that GRFs provided competitive density estimation and inference performance compared to GNF-M and SCCNF, with the GRF models achieving the best performance on the majority of the datasets. It is expected that these three models should in general provide similar performance since residual flows, flows with monotonic transformations and continuous flows are all highly flexible models that can model a wide range of distributions. GNF-A, with its reliance on simple affine transformations, was however unable to provide

[1]For all performance metrics reported here and in Chapter 7, we report the average and standard deviation calculated over the average performances of the model on the test set over the different runs.

Table 5.2: Density estimation performance of the different models. Each entry indicates the average negative log-likelihood on the test set (lower is better) over five independent runs, with the standard deviation given in the subscript. A standard deviation of less than 0.005 is indicated with Δ. Bold indicates the best average result in each model size category.

Flow	Arithmetic Circuit	Tree	Protein	EColi	MEHRA
GNF-A$_S$	$1.26_{\pm 0.02}$	$9.32_{\pm \Delta}$	$6.93_{\pm 0.90}$	$40.11_{\pm 0.01}$	$12.90_{\pm 0.02}$
GNF-M$_S$	$1.19_{\pm 0.07}$	$8.65_{\pm 0.01}$	$-3.00_{\pm 0.77}$	$40.13_{\pm 0.01}$	$11.74_{\pm 0.02}$
SCCNF$_S$	$\mathbf{0.86_{\pm 0.01}}$	$\mathbf{8.59_{\pm 0.01}}$	$-4.88_{\pm 0.21}$	$40.12_{\pm 0.02}$	$11.80_{\pm 0.05}$
GRF$_S$	$1.25_{\pm 0.01}$	$8.64_{\pm 0.01}$	$\mathbf{-5.26_{\pm 0.01}}$	$\mathbf{40.06_{\pm \Delta}}$	$\mathbf{11.66_{\pm 0.02}}$
GNF-A$_L$	$1.41_{\pm 0.16}$	$9.32_{\pm \Delta}$	$6.92_{\pm 0.57}$	$40.11_{\pm \Delta}$	$12.93_{\pm 0.03}$
GNF-M$_L$	$1.14_{\pm 0.04}$	$8.65_{\pm 0.01}$	$-5.48_{\pm 0.23}$	$40.13_{\pm \Delta}$	$11.67_{\pm 0.02}$
SCCNF$_L$	$\mathbf{0.85_{\pm \Delta}}$	$\mathbf{8.59_{\pm \Delta}}$	$-5.60_{\pm 0.05}$	$40.08_{\pm 0.01}$	$11.76_{\pm 0.02}$
GRF$_L$	$1.11_{\pm 0.01}$	$8.64_{\pm \Delta}$	$\mathbf{-6.11_{\pm 0.01}}$	$\mathbf{40.06_{\pm \Delta}}$	$\mathbf{11.61_{\pm 0.03}}$

Table 5.3: Inference performance of the different models. Each entry indicates the average negative ELBO on the test set (lower is better) over five independent runs, with the standard deviation given in the subscript. A standard deviation of less than 0.005 is indicated with Δ. Bold indicates the best average result in each model size category.

	Arithmetic Circuit	Tree	EColi
GNF-A$_S$	$4.90_{\pm 0.79}$	$2.36_{\pm 0.04}$	$34.98_{\pm \Delta}$
GNF-M$_S$	$\mathbf{3.96_{\pm 0.19}}$	$\mathbf{1.72_{\pm 0.01}}$	$34.99_{\pm 0.01}$
SCCNF$_S$	$4.01_{\pm 0.07}$	$1.78_{\pm 0.01}$	$35.24_{\pm 0.01}$
GRF$_S$	$4.19_{\pm 0.19}$	$1.74_{\pm \Delta}$	$\mathbf{34.96_{\pm \Delta}}$
GNF-A$_L$	$4.59_{\pm 0.28}$	$2.38_{\pm 0.05}$	$34.98_{\pm \Delta}$
GNF-M$_L$	$3.92_{\pm 0.08}$	$\mathbf{1.70_{\pm \Delta}}$	$34.98_{\pm 0.01}$
SCCNF$_L$	$3.97_{\pm 0.03}$	$1.76_{\pm 0.01}$	$35.24_{\pm 0.01}$
GRF$_L$	$\mathbf{3.71_{\pm 0.14}}$	$1.71_{\pm \Delta}$	$\mathbf{34.96_{\pm \Delta}}$

matching performance for these primary modelling tasks. The only dataset on which GNF-A did relatively well was the EColi dataset, which is likely due to the fact that the conditional distribution of each of the observations is a simple Gaussian. We also found that GRFs typically required more transformation steps than either GNF-M or SCCNF, as shown in Figure 5.2. This is due to the Lipschitz constraint placed on each of the residual blocks, which limits the representation capacity of any single step.

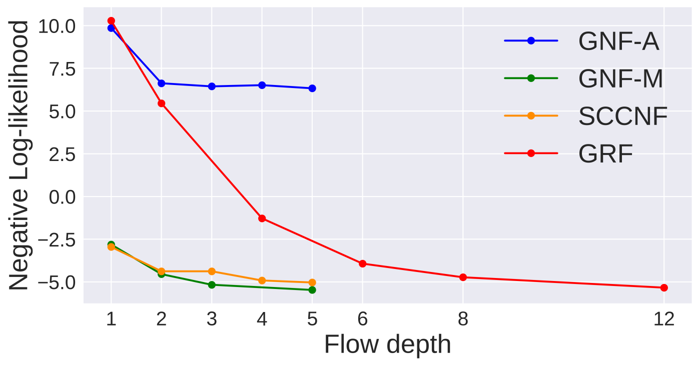


Figure 5.2: Density estimation performances (average negative log-likelihood) as a function of flow depth on the Protein dataset. For the finite flows (GNF-A, GNF-M and GRF), flow depth refers to the number of flow steps, whereas for the continuous flow (SCCNF), it is the number of hidden layers in its neural network. GRF typically requires more flow steps to achieve performance comparable to GNF-M and SCCNF. GNF-A reaches its maximum representation capacity within only a few flow steps and is not able to match the performance of the other flows.

The main reason for introducing GRFs is their potential to provide more accurate inversion. Having established that this new graphical flow provides key task performance comparable to the best existing graphical flows, we next proceeded to investigate the inversion accuracy of the various approaches.

5.3 Inversion

Before evaluating the inversion performance of the graphical flows, we first confirmed the computational advantage of using the Newton-like inversion procedure of Equation (4.11) to invert the flow steps of a GRF, rather than the Banach fixed-point iteration method. Although the latter ensures convergence to the correct inverse no matter the starting point, its convergence rate is heavily dependent on the choice of the hyperparameter c in Equation (2.29), which controls the Lipschitz bound on the residual block. As illustrated in Figure 5.3 for the Arithmetic Circuit dataset, it required many more iterations per block to invert a flow with $c = 0.99$ using the Banach fixed-point iteration method than with the update of Equation (4.11). In fact, fewer iterations were required with $c = 0.99$ for Equation (4.11) than with $c = 0.7$ for the Banach fixed-point method. Setting c close to one is desirable, since it makes each flow step less restrictive, and thus allows better modelling performance with fewer transformation steps. Similar behaviour as depicted in Figure 5.3, was observed for the other datasets as well (see Appendix A.1.4.4).

Having established the preference for inversion of GRFs with the Newton-like inversion procedure of Equation (4.11), we next explored the inversion stability

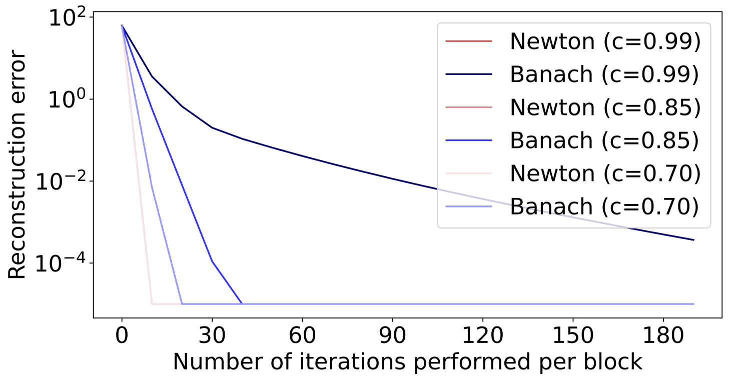


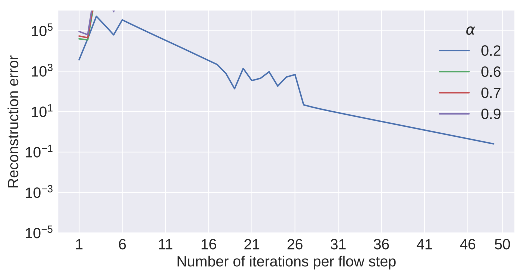
Figure 5.3: Using the Newton-like inversion procedure of Equation (4.11) required far fewer iterations per block to accurately invert a GRF than using the Banach fixed-point approach. The plot shows the average reconstruction error (log-scale) for 100 samples from the Arithmetic Circuit test set. Note that all the plots for the Newton-like inversion procedure, corresponding to different values of c, overlap.

and efficiency of GRFs compared to alternative graphical models with similar task performance, namely GNF-M and SCCNF. Note that while we could expect GNF-As to exhibit fast and accurate inversion, their relatively poor task performance makes them irrelevant for our purposes. We therefore proceeded by inverting the two finite flows, GNF-M and GRF, using the Newton-like inversion procedure given by Algorithm 3. We considered values for the step-size α in the set $\{0.1 \times t \,|\, t = 1, \ldots, 19\}$ and the inversion process was deemed to have converged when a reconstruction error of less than 10^{-4} was achieved. To better illustrate potential inversion instability, we performed the inversion process above on a per-data-point basis[2] for 100 test data points from each dataset. For each data point, we noted the value of α that required the fewest iterations, N, for convergence. Table 5.4 summarizes these results, where we have recorded the number of data points for which the desired reconstruction error of 10^{-4} was achieved, within $N \leq 50$ iterations in the case of GNF-M and GRF. We also measured the time it took to invert the flow for the entire batch using the settings that allowed the most data points to achieve the desired reconstruction error. In Table 5.5 we compare these timings to the inversion time using the Banach fixed-point approach for the GRFs on the same 100 test samples. Figures 5.4 to 5.8 show the average reconstruction error over the 100 test samples (inverted as a batch), for different values of α, when varying the number of iterations used at each step while inverting the finite flows. Since we do not have direct access to the Jacobian for the transformation applied by SCCNF, we did not use the above approach for the continuous flows. Instead, we inverted SCCNF by simply executing the integration in the opposite direction. Figures 5.4 to 5.8 are therefore not applicable for SCCNF.

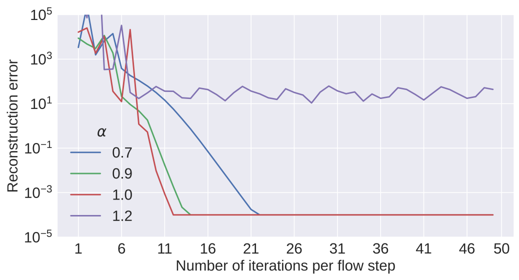
[2]In practice one would typically invert the sample as a single batch.

Table 5.4: Comparison of the inversion performance for the different flow models on 100 test data points from each of the datasets. Bold indicates the best results in each column. N and α are not applicable for SCCNF. Ranges indicate different optimal settings for N and α for different data points. Inversion time is measured for the smallest $N \leq 50$ that allowed the most data points in the batch to have a reconstruction error of less than 10^{-4}, and is the time taken to invert the entire batch.

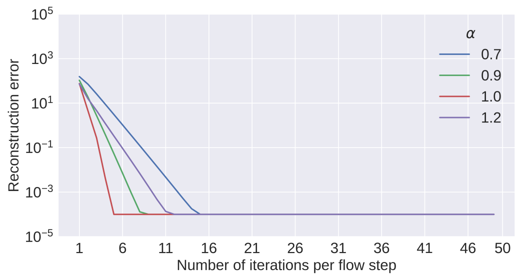
BN	Flow	Converged within 50 steps	N	α	Inversion time (ms)
Arithmetic circuit	GNF-M$_S$	99	4–42	0.3–1.1	226.17
	SCCNF$_S$	82	—	—	294.63
	GRF$_S$	**100**	4–5	1.0	**50.15**
	GNF-M$_L$	**100**	5–12	1.0	141.83
	SCCNF$_L$	97	—	—	540.38
	GRF$_L$	**100**	3–4	1.0	**91.88**
Tree	GNF-M$_S$	**100**	4–9	0.9–1.0	53.67
	SCCNF$_S$	98	—	—	140.51
	GRF$_S$	**100**	3–5	1.0	**49.73**
	GNF-M$_L$	98	4–47	0.4–1.0	488.51
	SCCNF$_L$	94	—	—	392.97
	GRF$_L$	**100**	4–6	0.9–1.0	**122.27**
Protein	GNF-M$_S$	97	9–50	0.5–1.4	145.88
	SCCNF$_S$	93	—	—	186.08
	GRF$_S$	**100**	5–7	0.9–1.0	**71.90**
	GNF-M$_L$	**100**	5–32	0.8–1.2	268.28
	SCCNF$_L$	81	—	—	890.38
	GRF$_L$	**100**	4–8	0.9–1.0	**265.23**
EColi	GNF-M$_S$	**100**	7–45	0.4–1.0	182.40
	SCCNF$_S$	23	—	—	121.66
	GRF$_S$	**100**	5–6	1.0	**57.91**
	GNF-M$_L$	**100**	6–8	1.0	**98.90**
	SCCNF$_L$	6	—	—	516.53
	GRF$_L$	**100**	4–5	1.0	101.90
MEHRA	GNF-M$_S$	**100**	3–11	0.9–1.0	**32.91**
	SCCNF$_S$	**100**	—	—	94.27
	GRF$_S$	**100**	3–4	1.0	48.15
	GNF-M$_L$	**100**	3–7	0.8–1.1	**62.81**
	SCCNF$_L$	99	—	—	190.92
	GRF$_L$	**100**	3–4	1.0	89.69



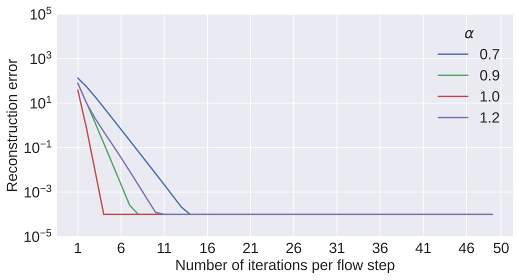
(a) GNF-M$_S$



(b) GNF-M$_L$

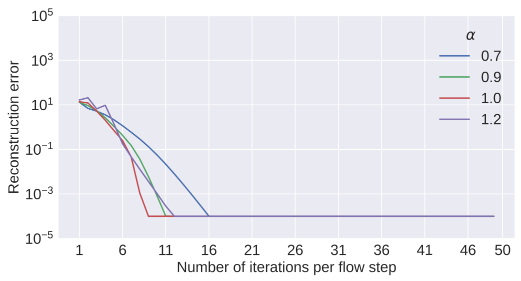


(c) GRF$_S$

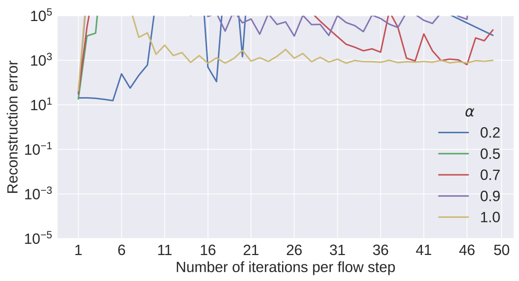


(d) GRF$_L$

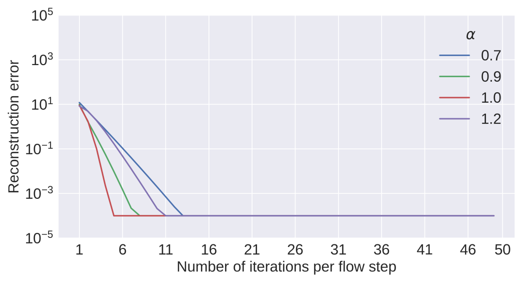
Figure 5.4: Reconstruction error achieved when inverting GNF-M and GRF on 100 test samples from the Arithmetic Circuit dataset. The reconstruction error is plotted as a function of the number of iterations used to invert each flow step, for different values of the step-size, α.



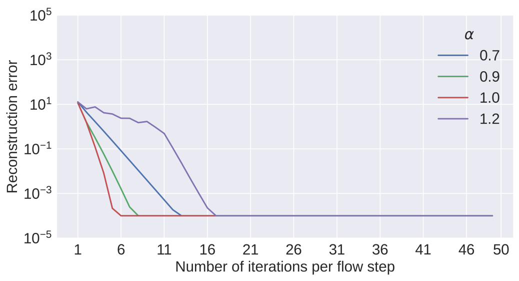
(a) GNF-M$_S$



(b) GNF-M$_L$

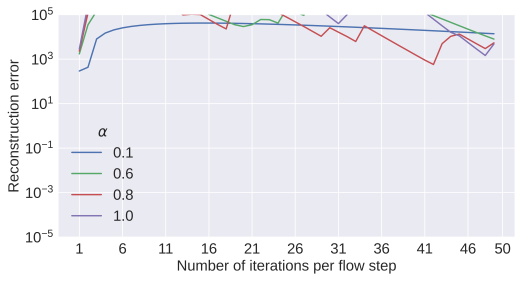


(c) GRF$_S$

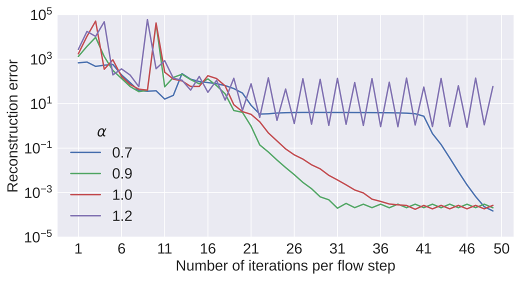


(d) GRF$_L$

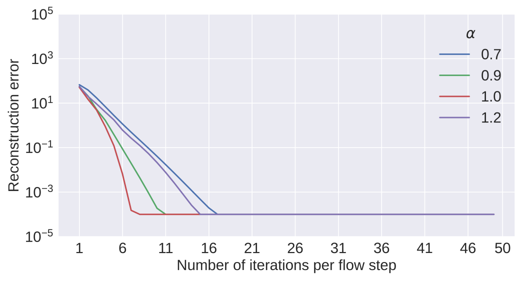
Figure 5.5: Reconstruction error achieved when inverting GNF-M and GRF on 100 test samples from the Tree dataset. The reconstruction error is plotted as a function of the number of iterations used to invert each flow step, for different values of the step-size, α.



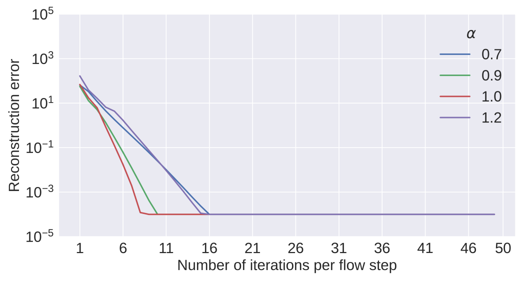
(a) GNF-M$_S$



(b) GNF-M$_L$

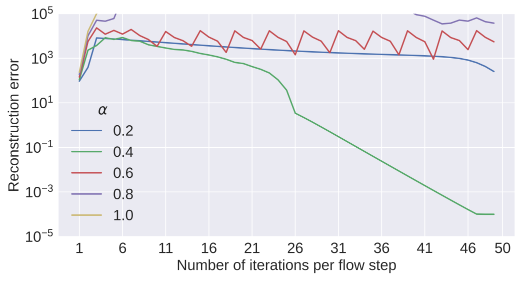


(c) GRF$_S$

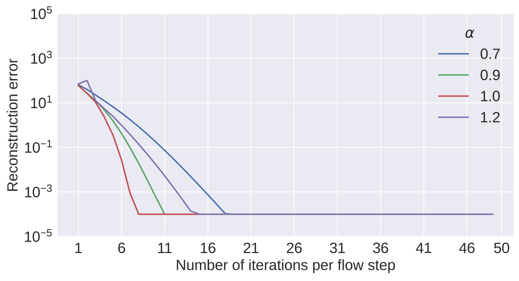


(d) GRF$_L$

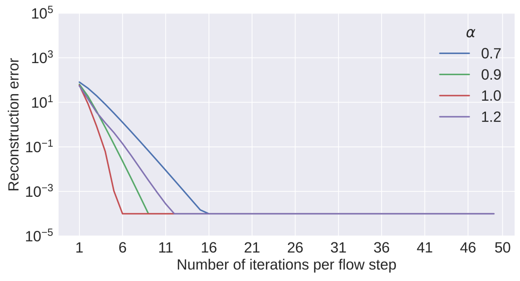
Figure 5.6: Reconstruction error achieved when inverting GNF-M and GRF on 100 test samples from the Protein dataset. The reconstruction error is plotted as a function of the number of iterations used to invert each flow step, for different values of the step-size, α.



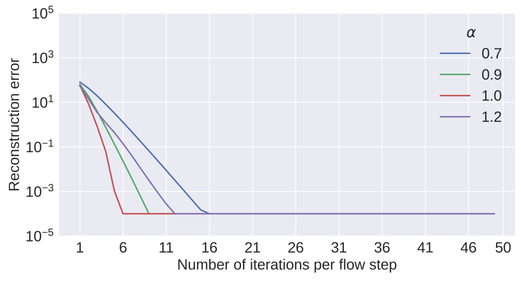
(a) GNF-M$_S$



(b) GNF-M$_L$

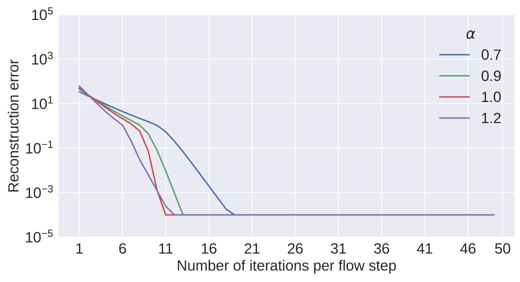


(c) GRF$_S$

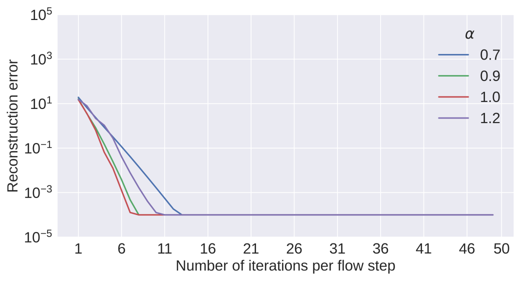


(d) GRF$_L$

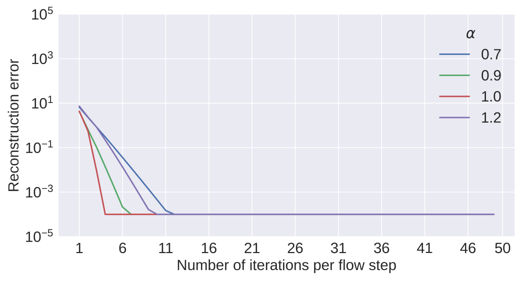
Figure 5.7: Reconstruction error achieved when inverting GNF-M and GRF on 100 test samples from the EColi dataset. The reconstruction error is plotted as a function of the number of iterations used to invert each flow step, for different values of the step-size, α.



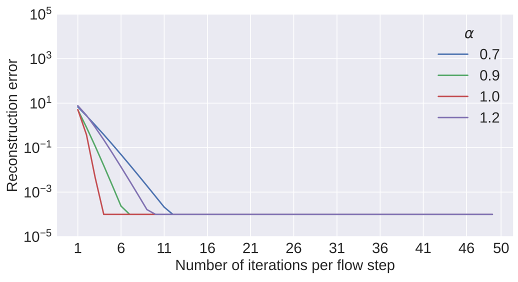
(a) GNF-M$_S$



(b) GNF-M$_L$



(c) GRF$_S$



(d) GRF$_L$

Figure 5.8: Reconstruction error achieved when inverting GNF-M and GRF on 100 test samples from the MEHRA dataset. The reconstruction error is plotted as a function of the number of iterations used to invert each flow step, for different values of the step-size, α.

Table 5.5: Inversion time of GRF for the Banach and the Newton-like fixed-point inversion schemes on a batch of 100 test samples. In each case, the convergence criterion is a reconstruction error of less than 10^{-4}. For the Newton-like inversion scheme, $\alpha = 1$ in all cases.

BN	Flow	Inversion time (ms)	
		Banach	Newton
Arithmetic circuit	GRF$_S$	595.56	**50.15**
	GRF$_L$	958.41	**91.88**
Tree	GRF$_S$	125.21	**49.73**
	GRF$_L$	573.10	**122.27**
Protein	GRF$_S$	598.97	**71.90**
	GRF$_L$	533.79	**265.23**
EColi	GRF$_S$	61.61	**57.91**
	GRF$_L$	120.30	**101.90**
MEHRA	GRF$_S$	148.26	**48.15**
	GRF$_L$	359.15	**89.69**

One of the main paradigms for enforcing global stability for flow inversion is using Lipschitz-constrained flow transformations (Behrmann *et al.*, 2021). In the case of GRFs, this stability is automatically achieved as a byproduct of the flow design, and based on the results in Table 5.4, we see that GRFs showed excellent inversion accuracy when compared to the other models with similar task performance. Even though GRFs typically required more flow steps than GNF-M or SCCNF, each step was relatively fast to invert and the GRF models had the fastest inversion times on the majority of the datasets when using the Newton-like fixed-point approach. As shown in Table 5.5, this Newton-like inversion scheme also provided much faster inversions than the Banach fixed-point approach. Note that the hyperparameter $c = 0.99$ was used for all models, which was shown in Figure 5.3 to slow down the Banach approach. Even though the Newton-like approach, unlike the Banach approach, does not guarantee convergence to the correct inverse, we did not find this to be an issue in our experiments and the Newton-like approach provided accurate inversion of GRFs in all cases. The Newton-like fixed-point approach is therefore a promising alternative for inverting GRFs, instead of the more traditional Banach inversion scheme.

GNFs with monotonic normalizers were not able to match the inversion accuracy of GRFs. These flows can, depending on the architecture and learned weights, have either potentially very large Lipschitz bounds, or have no global Lipschitz bounds at all—see Section 4.7. This helps to explain the poor inversion results observed for GNF-M on some of the datasets. For these models, more care needs to be taken when choosing a value for α. Table 5.4 shows that α typically had to be set to a very small value to try to obtain accurate inverses for all data instances, which slows down inversion. Taking too large a step can result in large inversion errors as clearly seen in Figures 5.4a, 5.5b, 5.6a and 5.7a. Only on the MEHRA dataset did GNF-M provide good inversion performance. In contrast, the GRF achieved good inversion performance on all datasets by simply setting $\alpha = 1$, as can be seen in the final two plots of Figures 5.4 to 5.8. GNF-M can also be inverted using bisection search as proposed in the original work by Wehenkel and Louppe (2021). However, bisection search requires choosing *two* appropriate starting values to guarantee convergence within a reasonable amount of steps, which cannot easily be automated. This approach also tends to be slower than the proposed Newton-like inversion scheme.

While SCCNFs have global Lipschitz bounds, these are not controlled during training and numerical instability can thus occur. Although we did not observe any exploding inverses, running the integration in the opposite direction in order to invert SCCNF, was not able to provide the specified reconstruction accuracy on many of the test instances. This is perhaps due to the black-box ODE solver only providing an approximation to the true inverse transform.

Furthermore, although SCCNF only requires a single step to perform inversion, this step is relatively slow compared to the finite flows.

If a flow, like GNF-M for example, has become numerically non-invertible on certain data points, then the computed densities of these points are no longer reliable, because the change-of-variables formula relies on accurate inversion. Behrmann *et al.* (2021) further showed that flows that do not impose some sort of constraint on the Lipschitz constant of their transformation tend to have large reconstruction errors on out-of-distribution data, i.e. data instances lying outside the distribution of the data points the flow was trained on. For real-world datasets, where it is expected that one would come across outlier data points, this could be an issue. This idea also extends to sampling from a flow trained in the normalizing direction. For complex datasets, it could be that the region in the latent space that the flow has learned to map observations to, does not completely match the true base distribution used during training. That is, there might be regions within the base distribution that a forward pass through the flow does not map any data samples to. This is similar to the prior-hole problem discussed in the context of VAEs in Section 3.2. When generating new samples using this flow, one first samples from the base distribution and it could then be likely that some of these samples lie within these regions not utilized by the flow. When inverting the flow on these samples, it is tantamount to presenting the flow with out-of-distribution data. The above issue emphasizes the need for flows that are expressive enough to learn a complex mapping that fully utilizes the base distribution, while remaining stably invertible in practice.

5.4 Conclusion

This chapter evaluated the proposed GRF model, which incorporates graphical dependency information from a BN into a residual flow, by comparing it to existing graphical flows on synthetic and real-world datasets. Section 5.2 evaluated the performance of various graphical flows on two key tasks, namely density estimation and variational inference. It was found that GRFs provide competitive performance for both density estimation and variational inference, compared to GNF-M and SCCNF. Finite graphical flows with simple affine transformations were unable to match the performance of the above three models for these primary modelling tasks.

The main reason for introducing GRFs is their potential to provide more accurate inversion. Having established that these new graphical flows provide key task performance comparable to the best existing graphical flows, we proceeded in Section 5.3 to investigate the inversion accuracy of the various approaches. It was found that the global Lipschitz bounds of the GRF residual blocks afford it a greater degree of inversion stability and accuracy than alternative flows, and that GRFs typically exhibit faster inversion than alternative flow models with comparable modelling performance.

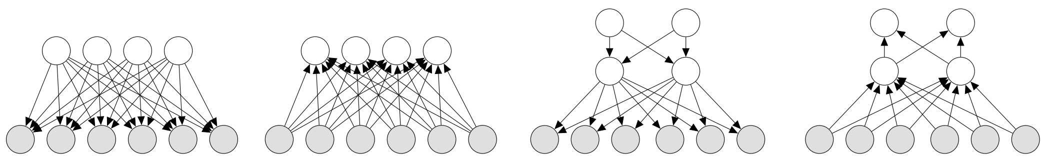
In summary, GRFs are on par with existing graphical flows in terms of modelling ability, and additionally guarantee stable inversion in practice due to the strict Lipschitz bounds placed on each of the flow steps. These flows are therefore a better choice than GNF-M or SCCNF when a single flow will be used in both transformation directions. Next, we incorporate graphical dependency information into an alternative type of deep generative model—the variational autoencoder—and specifically consider how to do this using the proposed GRF.

Chapter 6

SIReN-VAE: Structured Invertible Residual Network VAE

VAEs provide a natural bridge between graphical models and deep neural networks. The generative model of the vanilla VAE introduced in Section 2.4.1 can be viewed as a BN (skeleton) due to the very specific independence assumptions made in the model: the generative model is of the form $p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$ where the prior factorizes as $\prod_i p(\mathrm{z}_i)$, see Figure 6.1a. In the case where the latent variables have a hierarchical structure (introduced in Section 3.2.1), the BN instead corresponds to the factorization $p(\mathbf{x}|\mathbf{z}_0)p(\mathbf{z}_0|\mathbf{z}_1)\dots p(\mathbf{z}_{L-1}|\mathbf{z}_L)p(\mathbf{z}_L)$, see Figure 6.1c. The inference networks of the above VAE models similarly correspond to specific BNs, depending on the dependencies induced by the chosen variational family. Figure 6.1 provides diagrams of the BN graphs associated with a vanilla and hierarchical VAE's generative and inference networks. Since the conditional factors can be specified using deep neural networks, VAEs provide an intuitive way of combining BN structures with the modelling capacity of deep learning.

Most prior work has restricted its attention to the above generic BN structures in the generative model of a VAE. Furthermore, the dependencies induced by



(a) Vanilla VAE generative model (b) Vanilla VAE inference network (c) Hierarchical VAE generative model (d) Hierarchical VAE inference network

Figure 6.1: BN graphs encoded by the generative model and inference network of different VAE approaches. The latent variables in the (a) prior and (b) variational posterior of a vanilla VAE are completely independent. Hierarchical VAEs (c) and (d) have layers of latent variables, in this case two.

the inference network are typically only based on a heuristic inverse of the dependencies in the generative model. This limits the ability of the variational distribution to learn an approximation close to the true posterior (Webb *et al.*, 2018). Here, we consider how to specify an *arbitrary* BN dependency structure over the latent variables and between the latent and observed variables of the VAE. We specifically consider using graphical residual flows (GRFs) in both the prior and posterior of a VAE to achieve this objective, since it has been shown in Chapter 5 that they provide good modelling capability while maintaining stable and accurate inversion performance. These flows allow the user to inject domain knowledge or prior beliefs, as specified by a BN, into a VAE model with minimal effort. GRFs (as with other flows) further increase the representation capacity of the variational posterior of a VAE, which is considered by many to be an advantageous extension of the standard VAE model—see Section 3.2.1. Using GRFs and the faithful inversion scheme of Webb *et al.* (2018), we are also able to encode an informed dependency structure in the inference network based on the dependencies of the generative model. The resulting structured invertible residual network VAE (SIReN-VAE) is presented in Section 6.1. Section 6.2 provides additional details on various approaches we considered for tackling posterior collapse. The discussion in this chapter is based in part on the work presented in Mouton and Kroon (2022*b*).

6.1 Structuring VAEs with Graphical Residual Flows

If we wish to construct a VAE, and have prior beliefs about the data generating process, then it seems beneficial to incorporate this knowledge in the VAE. In what follows, we assume that we have access to a BN specifying a suggested dependency structure over D observed and K latent variables. Our goal is to suitably incorporate this dependency information into the VAE's generative and inference networks. Using θ for the generative network parameters, this means that we ideally want the likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ and prior $p_\theta(\mathbf{z})$ to factorize according to the BN's conditional independencies. However, Webb *et al.* (2018) also showed the value of encoding the generative model's true inverted dependency structure as far as possible in the VAE's inference network. That is, the structure of the inference network should respect knowledge about $p(\mathbf{z}|\mathbf{x})$ which can be deduced from the factorization of $p(\mathbf{x}, \mathbf{z})$. Approximating the posterior distribution $p(\mathbf{z}|\mathbf{x})$ in such a way requires inverting the BN (so that edges go from $\mathbf{x}$ to $\mathbf{z}$) while taking into account the independencies encoded by the model (as discussed in Section 4.4). We use GRFs to incorporate these desired structures into the generative and inference network of a VAE, yielding the structured invertible residual network (SIReN) VAE. The modifications applied to a vanilla VAE to obtain the SIReN-VAE are discussed below.

6.1.1 Modifications to the Generative Phase

SIReN-VAE encodes a given BN's prior latent structure by replacing the standard normal prior of the vanilla VAE with a (normalizing) GRF. Similar to the standard VAE, the likelihood has an appropriate form depending on the data, e.g. for continuous observations, it could be a fully-factorized Gaussian distribution, while for binary data each observed component could be specified by a Bernoulli distribution. The parameters of the likelihood are output by a neural network denoted by DecoderNN that takes a latent vector $\mathbf{z}$ as input. To encode the conditional independencies between the latent and observed variables in the BN, the decoder neural network is also masked according to the scheme discussed for GRFs in Section 4.2. In summary, for continuous observations with a Gaussian distribution:

$$\begin{aligned} p_\theta(\mathbf{z}) &= p_0(\mathrm{GRF}_n(\mathbf{z};\theta))\,|\det(J_{\mathrm{GRF}_n}(\mathbf{z};\theta))| && (6.1)\\ \boldsymbol{\mu}, \log\boldsymbol{\sigma} &= \mathrm{DecoderNN}(\mathbf{z};\theta) && (6.2)\\ p_\theta(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)) \ . && (6.3) \end{aligned}$$

The subscript n in line (6.1) above indicates that the flow is in the normalizing direction, and we set p_0 to $\mathcal{N}(\mathbf{0}, I_K)$. An illustration of this generative model is given in the right panel of Figure 6.2.

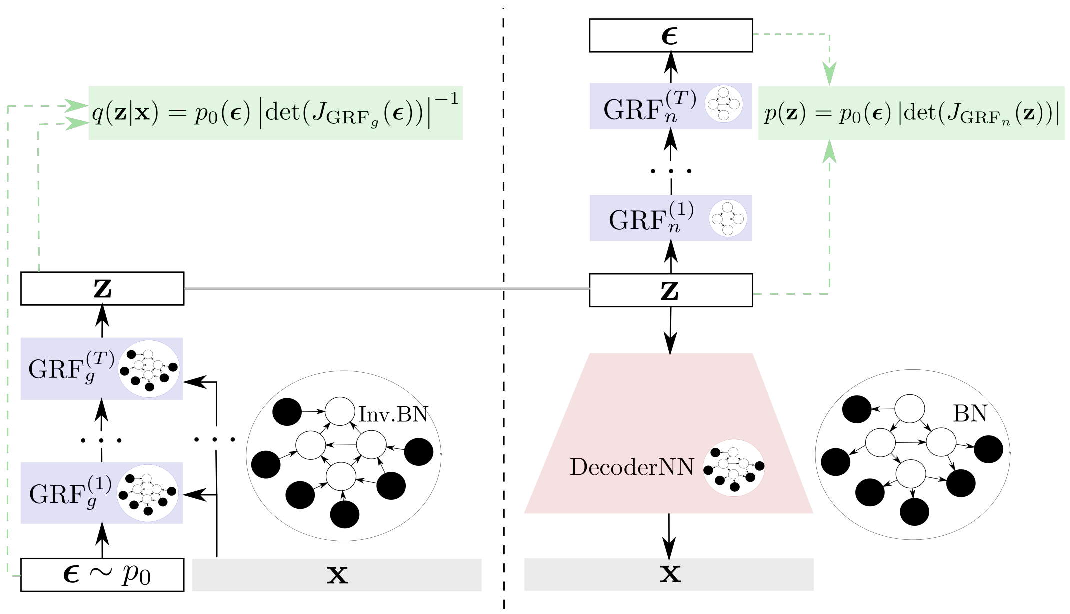
In order to generate new instances using the above model, one has to invert the GRF_n of the prior, as illustrated in Figure 6.3. This makes sampling slower than with regular VAEs. A key benefit of GRFs over other graphical flows, however, is that they are designed to provide stable inversion, as shown in the previous chapter. The inversion time per flow step of a GRF is also relatively low compared to other graphical flows with similar modelling capability. This motivates our use of GRFs instead of other existing graphical flows.

6.1.2 Modifications to the Inference Phase

For constructing the inference network, the BN used for the generative phase is inverted using the minimally faithful inversion algorithm proposed by Webb *et al.* (2018). Similar to the prior, the inference network, with parameters ϕ, is defined as a GRF conditioned on $\mathbf{x}$, which encodes this inverse BN structure:

$$\begin{aligned} \mathbf{z} &= \mathrm{GRF}_g(\boldsymbol{\epsilon};\mathbf{x},\phi) \quad \text{where} \quad \boldsymbol{\epsilon} \sim p_0 && (6.4)\\ q_\phi(\mathbf{z}|\mathbf{x}) &= p_0(\boldsymbol{\epsilon})\,\left|\det(J_{\mathrm{GRF}_g}(\boldsymbol{\epsilon};\phi))\right|^{-1} \ . && (6.5) \end{aligned}$$

Here, the subscript g denotes that this flow is in the generative direction (for a more in depth discussion on conditional GRFs, refer back to Section 4.4). An illustration of this inference network is given in the left panel of Figure 6.2. Using GRFs, SIReN-VAE is thus able to encode a wide range of different dependency structures in a VAE as shown in Figure 6.4.



$q(\mathbf{z}|\mathbf{x}) = p_0(\boldsymbol{\epsilon})\left|\det(J_{\mathrm{GRF}_g}(\boldsymbol{\epsilon}))\right|^{-1}$
$\mathbf{z}$
$\mathrm{GRF}_g^{(T)}$
$\mathrm{GRF}_g^{(1)}$
$\boldsymbol{\epsilon} \sim p_0$
$\mathbf{x}$
Inv.BN
$\boldsymbol{\epsilon}$
$\mathrm{GRF}_n^{(T)}$
$p(\mathbf{z}) = p_0(\boldsymbol{\epsilon})\left|\det(J_{\mathrm{GRF}_n}(\mathbf{z}))\right|$
$\mathrm{GRF}_n^{(1)}$
$\mathbf{z}$
DecoderNN
BN
$\mathbf{x}$

Figure 6.2: SIReN-VAE encodes the BN's graphical structure into the decoder (right) via masking of the normalizing GRF (GRF_n) and decoder neural network (DecoderNN) weights. The inference network (left) similarly encodes the inverted BN structure in its generating GRF (GRF_g).

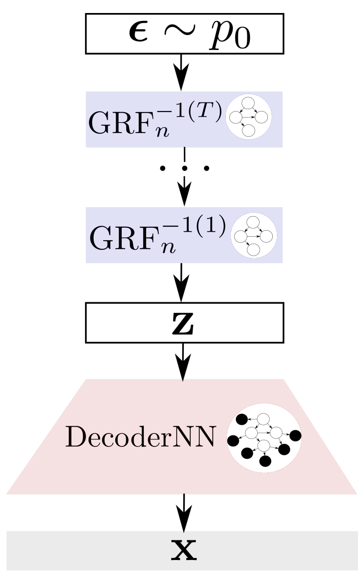


Figure 6.3: Sampling from SIReN-VAE requires inverting GRF_n in the prior for a sample $\boldsymbol{\epsilon}$ from the flow's base distribution.

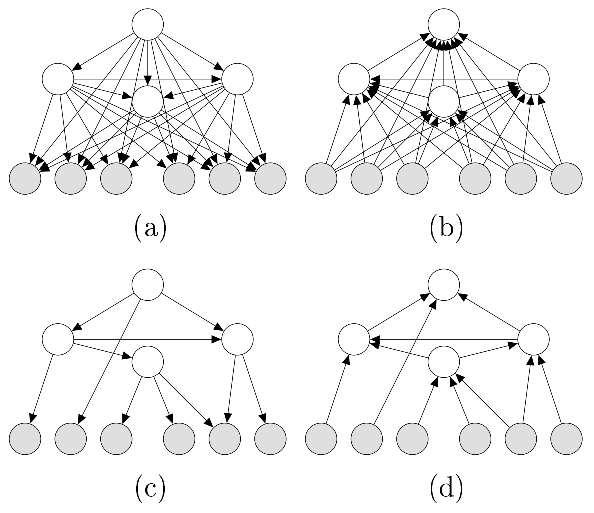


Figure 6.4: The generative model of SIReN-VAE can encode a range of BN graphs, from (a) fully-connected to (c) sparser dependency structures. The inference network encodes a minimal and faithful inverse of this dependency structure, as shown in (b) and (d), respectively.

6.2 Posterior Collapse in a Structured Latent Space

As discussed in Section 2.4.4, posterior collapse is one of the most common challenges encountered when fitting VAEs. Burda *et al.* (2016) found that latent variables in the higher layers of a hierarchical VAE are more prone to collapse than those directly connected to the observed variables. This suggests that the structure of dependencies in the latent distribution influences the likelihood of various latent variables becoming inactive during training. Since SIReN-VAE can encode arbitrary BN structures (including a hierarchical latent distribution as a special case) we expect similar challenges with posterior collapse. Our desire for the learned latent distribution to respect the provided BN structure and utilize all the latent dimensions, especially for interpretability, further motivates trying to prevent posterior collapse. Below we present, in more detail than in Section 2.4.4, several approaches from the literature that we investigate for this purpose.

Warm-up

The regularizing KL-divergence term (also known as the variational regularization term) present in the prior-contrastive form of the ELBO objective,

$$\mathcal{L}^{\text{ELBO}}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \text{KL}(q_\phi(\mathbf{z}|\mathbf{x})||\, p(\mathbf{z})) \ , \tag{6.6}$$

plays a role in posterior collapse, since it encourages the posterior to be similar to the uninformative prior (Razavi *et al.*, 2019; Bowman *et al.*, 2016). As such, one of the first and most simple remedies proposed is to add a weight to the KL-term that is then shifted from 0 to 1 over an initial number of epochs during training (Burda *et al.*, 2016; Bowman *et al.*, 2016; Huang *et al.*, 2018*b*). This strategy, known as warm-up (WU), helps to diminish the initial effect of the KL-term and thus helps prevent posterior collapse. This new objective is given by:

$$\begin{aligned}\mathcal{L}_t^{\text{WU}}(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}\sim q}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta_t \cdot \text{KL}(q_\phi(\mathbf{z}|\mathbf{x})||\, p(\mathbf{z})) &(6.7)\\ &= \mathbb{E}_{\mathbf{z}\sim q}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \beta_t \cdot (\log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\right] \ , &(6.8)\end{aligned}$$

where $\beta_t \in \mathbb{R}$ is increased linearly from 0 to 1 during the first N_{WU} epochs, i.e.

$$\beta_t = \min\left(1, \frac{t}{N_{\text{WU}}}\right) \ . \tag{6.9}$$

The KL-divergence term therefore ultimately has the same effect as in the original objective, only its initial effect is reduced to allow training to find a location with good reconstruction loss.

Importance-weighted Objective

The looseness of the ELBO objective has also been identified as a potential source of posterior collapse as it can allow the parameters of the model to settle away from their theoretically optimal setting (Melis *et al.*, 2022). A remedy for this concern is to use an objective with a tighter bound, such as the importance weighted objective of the importance weighted autoencoder (IWAE) (Burda *et al.*, 2016). Let $w_i = p_\theta(\mathbf{x}, \mathbf{z}_i)/q_\phi(\mathbf{z}_i|\mathbf{x})$ for $\mathbf{z}_i \sim q(\mathbf{z}|\mathbf{x})$ be the *importance weights*, then

$$\mathcal{L}_K^{\mathrm{IW}}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}_{1:K} \sim q_\phi(\cdot|x)} \left[\log \frac{1}{K} \sum_{i=1}^{K} w_i \right] \tag{6.10}$$

$$= -\log K + \mathbb{E}_{\mathbf{z}_{1:K} \sim q_\phi(\cdot|x)} \left[\underbrace{\log \left(\sum_{i=1}^{K} \exp\left(\log w_i\right) \right)}_{\text{log-sum-exp}} \right] , \tag{6.11}$$

where the second term in the expectation can be stably computed using the log-sum-exponent trick. Burda *et al.* (2016) showed that $\mathcal{L}_K^{\mathrm{IW}}$ is at least as tight as $\mathcal{L}^{\mathrm{ELBO}}$ and that it converges to the marginal likelihood $p_\theta(\mathbf{x})$ (under mild conditions) as $K \to \infty$. This objective is equivalent to the standard ELBO for $K = 1$:

$$\mathcal{L}_1^{\mathrm{IW}}(\mathbf{x}) = \mathcal{L}^{\mathrm{ELBO}}(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\cdot|x)} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] . \tag{6.12}$$

Doubly-reparameterized Gradient (DReG) Estimator

Another factor that can contribute to posterior collapse is the adverse effect of high-variance gradient estimators on optimization (Melis *et al.*, 2022). If q_ϕ is reparameterizable, i.e. $\mathbf{z} = g(\boldsymbol{\epsilon}; \phi)$, where g is some deterministic and differentiable function and $\boldsymbol{\epsilon} \sim p_{\boldsymbol{\epsilon}}$ does not depend on θ or ϕ, then the reparameterization trick (Kingma and Welling, 2014; Rezende *et al.*, 2014)—see Section 2.4.2—allows us to rewrite the expectation in line (6.10) above, with respect to $\boldsymbol{\epsilon}_{1:K}$. This allows one to exchange the gradient and expectation operators so that the standard gradient of $\mathcal{L}_K^{\mathrm{IW}}$ with respect to the inference network parameters is given by:

$$\begin{aligned}
\nabla_\phi \mathcal{L}_K^{\mathrm{IW}}(\mathbf{x}) &= \nabla_\phi \mathbb{E}_{\mathbf{z}_{1:K}\sim q_\phi(\cdot|\mathbf{x})}\left[\log \frac{1}{K}\sum_{i=1}^{K}\frac{p_\theta(\mathbf{x},\mathbf{z}_i)}{q_\phi(\mathbf{z}_i|\mathbf{x})}\right] \\
&= \nabla_\phi \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}\sim p_{\boldsymbol{\epsilon}}}\left[\log \frac{1}{K}\sum_{i=1}^{K}\frac{p_\theta(\mathbf{x},g(\boldsymbol{\epsilon}_i;\phi))}{q_\phi(g(\boldsymbol{\epsilon}_i;\phi)|\mathbf{x})}\right] && (6.13)\\
&= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}\sim p_{\boldsymbol{\epsilon}}}\left[\nabla_\phi \log \frac{1}{K}\sum_{i=1}^{K} w_i\right] && (6.14)\\
&= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}\sim p_{\boldsymbol{\epsilon}}}\left[\sum_{i=1}^{K}\frac{1}{\sum_j w_j}\nabla_\phi w_i\right] && (6.15)\\
&= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}\sim p_{\boldsymbol{\epsilon}}}\left[\sum_{i=1}^{K}\frac{w_i}{\sum_j w_j}\nabla_\phi \log w_i\right], && (6.16)
\end{aligned}$$

where $w_i = p_\theta(\mathbf{x},\mathbf{z}_i)/q_\phi(\mathbf{z}_i|\mathbf{x})$ are the importance weights, and we applied the log-derivative (or score function) trick in line (6.16): $\nabla_\phi w_i = w_i \nabla_\phi \log w_i$.

To see why this gradient has high variance when estimated with Monte Carlo samples, Roeder *et al.* (2017) expand the total derivative of $\log w_i$ with respect to the variational parameters, denoted by $\nabla_\phi^{\mathrm{TD}} \log w_i$, into two terms:

$$\begin{aligned}
\nabla_\phi^{\mathrm{TD}} \log w_i &= \nabla_{\mathbf{z}_i} \log w_i \nabla_\phi \mathbf{z}_i + \nabla_\phi \log w_i && (6.17)\\
&= \nabla_{\mathbf{z}_i} \log w_i \nabla_\phi \mathbf{z}_i + \underbrace{\nabla_\phi \log p_\theta(\mathbf{z}_i,\mathbf{x})}_{=0} - \nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x}) && (6.18)\\
&= \nabla_{\mathbf{z}_i} \log w_i \nabla_\phi \mathbf{z}_i - \nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x}) \;, && (6.19)
\end{aligned}$$

where $\nabla_{\mathbf{z}_i} \log w_i \nabla_\phi \mathbf{z}_i$ is known as the path derivative[1], and $\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})$ as the score function component. Note that the path derivative measures the dependence of the total derivative on ϕ only through the sample $\mathbf{z}_i = g(\boldsymbol{\epsilon}_i;\phi)$, while the score function measures the dependence on $\log w_i$ directly and considers $\mathbf{z}_i$ as constant. Although increasing K in $\mathcal{L}_K^{\mathrm{IW}}$ provides tighter evidence bounds, Rainforth *et al.* (2018) showed that the gradient estimator (6.16) for the inference network gradually becomes noisier as K increases. Furthermore, Roeder *et al.* (2017) found that the score function can contribute significant variance to the gradient estimate since it will not be zero even if $q_\phi(\mathbf{z}_i|\mathbf{x})$ exactly matches the true posterior. As such, Roeder *et al.* (2017) suggest dropping $\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})$ when $K > 1$. Tucker *et al.* (2018), however, argue that this results in a biased estimator and show how one can instead estimate $\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})$ using a second application of the reparameterization trick,

[1]Note that we have abused notation here to simplify the exposition. The path derivative is a vector-matrix multiplication and is more correctly given by $(\nabla_{\mathbf{z}_i} \log w_i)^T J_g(\phi)$.

which results in an *unbiased* and lower-variance gradient estimator called the doubly-reparameterized gradient (DReG) estimator. This DReG estimator is given by:

$$\nabla_\phi \mathbb{E}_{\mathbf{z}_{1:K}\sim q_\phi(\cdot|x)}\left[\log \frac{1}{K}\sum_{i=1}^{K} w_i\right] = \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}\sim p_{\boldsymbol{\epsilon}}}\left[\sum_{i=1}^{K}\left(\frac{w_i}{\sum_j w_j}\right)^2 \nabla_{\mathbf{z}_i}\log w_i \nabla_\phi \mathbf{z}_i\right]. \tag{6.20}$$

See Appendix B.2.2 for the full derivation.

In practice, one typically makes use of automatic differentiation libraries to efficiently compute the required gradients. However, naïvely applying automatic differentiation will not result in the gradients being computed according to the right-hand side of Equation 6.20. One therefore has to take special care that the way in which the loss is calculated, forces an automatic differentiation library to compute the gradients in the desired manner. Specifically, the DReG estimator can be implemented implicitly by ensuring that the computational path from ϕ to $\log w_i$ passes strictly through $\mathbf{z}_i$. This forces any automatic differentiation library to apply the chain rule: $\nabla_{\mathbf{z}_i}\log w_i \nabla_\phi \mathbf{z}_i$ as in Equation (6.20).

Recall that $\log w_i = \log p_\theta(\mathbf{x},\mathbf{z}_i) - \log q_\phi(\mathbf{z}_i|\mathbf{x})$. For generative GRFs, $\log q_\phi(\mathbf{z}_i|\mathbf{x})$ is typically calculated as

$$\log q_\phi(\mathbf{z}_i|\mathbf{x}) = \log p_0(\boldsymbol{\epsilon}_i) - \log\left|\det\left(J_{\mathrm{GNF}_g}(\boldsymbol{\epsilon}_i;\mathbf{x},\phi)\right)\right| \ , \tag{6.21}$$

where $\boldsymbol{\epsilon}_i \sim p_0$ is a sample from the base distribution. Note that this expression does not include the sampled $\mathbf{z}_i$ and one can therefore not obtain the desired gradient $\nabla_{\mathbf{z}_i}\log w_i \nabla_\phi \mathbf{z}_i$ using automatic differentiation as is. In order to implement the DReG estimator, one has to invert the flow such that $\log q_\phi(\mathbf{z}_i|\mathbf{x})$ is given by

$$\log q_\phi(\mathbf{z}_i|\mathbf{x}) = \log p_0(\mathrm{GNF}_g^{-1}(\mathbf{z}_i;\mathbf{x},\phi)) + \log\left|\det\left(J_{\mathrm{GNF}_g^{-1}}(\mathbf{z}_i;\mathbf{x},\phi)\right)\right| \ . \tag{6.22}$$

Now, both $\log p_\theta(\mathbf{x},\mathbf{z}_i)$ and $\log q_\phi(\mathbf{z}_i|\mathbf{x})$ are computed using the sample $\mathbf{z}_i$. However, unlike for $\log p_\theta(\mathbf{x},\mathbf{z}_i)$, the calculation of $\log q_\phi(\mathbf{z}_i|\mathbf{x})$ involves the parameters ϕ not only through $\mathbf{z}_i$, but also directly through $\mathrm{GNF}_g^{-1}(\cdot;\mathbf{x},\phi)$. In order to force gradient information to pass only through $\mathbf{z}_i$, we employ a similar strategy as Roeder *et al.* (2017) by treating ϕ in Equation (6.22) as constant. This will then force gradient information about ϕ to pass strictly through $\mathbf{z}$, as required. Automatic differentiation libraries have functions that can be used to stop the flow of gradient information and cause a variable to be treated as constant during backpropagation. For example, in PyTorch (Paszke *et al.*, 2019) one can call the `detach()` function on a given tensor. Algorithm 4 provides the pseudocode for constructing the objective as described above, so

Algorithm 4 DReG Estimation.

```
Require: Variational parameters $\phi$, data $\mathbf{x}$
Return: Unbiased low-variance estimate of $\nabla_\phi \mathcal{L}_K^{\mathrm{IW}}(\mathbf{x})$
1: $\boldsymbol{\epsilon}_1, \dots, \boldsymbol{\epsilon}_K \sim p_0$
2:
3: def $w(\boldsymbol{\epsilon}, \phi)$ :
4:     $\mathbf{z} \leftarrow \mathrm{GRF}_g(\boldsymbol{\epsilon}; \mathbf{x}, \phi)$
5:     $\phi' \leftarrow \mathrm{stop\_grad}(\phi)$                ▷ Treat $\phi$ as constant
6:     $q \leftarrow p_0(\mathrm{GNF}_g^{-1}(\mathbf{z}; \mathbf{x}, \phi')) \left|\det\left(J_{\mathrm{GNF}_g^{-1}}(\mathbf{z}; \mathbf{x}, \phi')\right)\right|$
7:     return $\frac{p(\mathbf{x}, \mathbf{z}; \theta)}{q}$
8:
9: return $\nabla_\phi \sum_{i=1}^K \left(\mathrm{stop\_grad}\left(\frac{w(\boldsymbol{\epsilon}_i, \phi)}{\sum_j w(\boldsymbol{\epsilon}_j, \phi)}\right)\right)^2 \log w(\boldsymbol{\epsilon}_i, \phi)$
```

that the gradients are computed according to Equation (6.20). Note that this also requires the term $(w_i / \sum_j w_j)^2$ to be treated as constant.

When using DReG with SIReN-VAE, it is crucial to use a flow that is invertible in practice .[2] A drawback of this approach is therefore that each inversion step requires a number of iterations to converge when using the Newton-like inversion procedure discussed in Section 4.6. This inversion slows down training and also results in larger memory consumption since the computation graphs of the inversion iterations of each flow step need to be stored to facilitate backpropagation.

6.3 Conclusion

In this chapter we have proposed a new graphical VAE that employs GRFs and masking of the decoder neural network to encode an arbitrary BN structure between the latent variables and between the latent and observed variables of the model. Since we desire for the learned latent distribution of SIReN-VAE to respect the provided BN structure, we are especially interest in preventing posterior collapse. Since the three mitigation techniques discussed in this chapter, namely warm-up, importance weighted objectives and lower-variance gradient estimates, address some of the main previously identified causes of posterior collapse, we hope that they will aid SIReN-VAE in learning meaningful latent representations. We continue in the next chapter by evaluating SIReN-VAE on a range of datasets. We specifically examine the occurrence of inactive latent variables and assess the effectiveness of the above remedies.

[2]This is in line with work done by Vaitl *et al.* (2022), who show how to use the path derivative for any NF that is invertible in practice. This work was published after the work reported on in this thesis was completed.

Chapter 7

Empirical Investigation II: Structured Invertible Residual Network VAE

This chapter presents an empirical investigation into the performance of the proposed SIReN-VAE model, in order to shed light on its strengths and shortcomings. More specifically, the goals of this chapter were as follows:

1. To quantify the effect of using GRFs as the latent prior and variational approximate posterior in a VAE.
2. To evaluate the modelling performance of SIReN-VAE when encoding a dataset's associated BN graph, as compared to the standard VAE and a SIReN-VAE where no independence assumptions are made.
3. To explore how much of an issue posterior collapse is when encoding various dependency structures, and to test the effectiveness of the various mitigation techniques discussed in the previous chapter.
4. To investigate the extent to which the potential hypothesized benefits of SIReN-VAE are realized, specifically:
    a) to test whether encoding a dataset's associated dependency structure with SIReN-VAE results in models with a higher degree of interpretability, and
    b) to assess whether SIReN-VAE provides better generalization performance in data-sparse settings.

Section 7.1 outlines the methodology employed to achieve these goals. This includes an overview of the datasets used, as well as the training setup and model architectures. Sections 7.2 to 7.6 provide and critically discuss the experimental results in line with each of the goals above.

7.1 Methodology

As an initial investigation, we verified whether the more complex latent distributions enabled by our choice of using a GRF prior and posterior in the SIReN-VAE model, results in better modelling performance than the vanilla VAE architecture. For the purpose of validating our basic model architecture, independent of any assumed BN dependency structure, we did not assume any conditional independence in this setting, and simply used our graphical flows to encode a fully-connected BN structure. We measured the effect of the GRFs by first replacing only the inference network by a GRF, and thereafter also adding a GRF to the latent prior of the VAE. More information is provided in Section 7.1.1 about the datasets used in this investigation.

Next, we investigated the benefits of incorporating the conditional independencies from a BN into a SIReN-VAE using GRFs. We began by comparing SIReN-VAE models with various encoded dependency structures on a number of synthetic and real-world datasets, where the domain problem represented by each of these datasets has an associated true or hypothesized BN structure. We considered fully-connected dependency structures (denoted by SIReN-VAE$_{\text{FC}}$), structures equal to the datasets' associated BN graphs (SIReN-VAE$_{\text{True}}$), as well as random dependency structures (SIReN-VAE$_{\text{Rand}}$). Since we expected posterior collapse to be a potential problem, we continued the study by testing for inactive units in the trained models. Based on these results, we further investigated whether or not the position and neighbourhood of a latent variable within the BN plays a role in whether the corresponding SIReN-VAE latent variable collapses. We continued by evaluating the effectiveness of various combinations of warm-up, importance-weighted objectives and DReG estimates in combatting posterior collapse. Section 7.1.1 provides more information about the graphical datasets used during the above investigations, with Section 7.1.2 detailing the chosen model architectures and training procedures.

Finally, we explored the extent to which the additional hypothesized benefits of using the proposed SIReN-VAE approach for datasets with an associated BN dependency structure, are realized. We hypothesized that incorporating the conditional independencies specified by a BN leads to improved performance in data-sparse settings. To test this, we compared the generalization performance of SIReN-VAE$_{\text{True}}$ to SIReN-VAE$_{\text{FC}}$ and standard VAEs, when training these models on only a small subset of the original training set. Another anticipated benefit is the potential of a more interpretable latent space. As discussed in Section 2.4.5, a central aspect of interpretability in deep latent variable models is understanding the relationships between (latent and observed) variables. Considering the synthetic datasets, where we know exactly how the true latent variables affect each other and the observations, the ideal would be if the model is able to learn the true prior conditional distributions that gave rise to the observations. Doing so from only the given conditional independence structure is a challenging problem, as many different configurations of the conditional latent distributions might lead to the same observed data. For real-world datasets, where we typically only have access to a *hypothetical* BN, this is an even more difficult task. The aim is therefore not to learn the true underlying distributions, but rather, given the additional conditional independence structure, to explore whether the fitted individual latent distributions of SIReN-VAE$_{\text{True}}$ play a similar role within the BN as their true counterparts. Since we have access to the true latent variables of the synthetic datasets, we investigated this by computing the mutual information (MI) between samples drawn from the inferred latent distribution of a SIReN-VAE$_{\text{True}}$ model for a given set of observations and the true latent variables that gave rise to those observations. MI is a more general measure of the dependence between variables than linear correlation, but is generally intractable to compute exactly. We therefore used a neural-network-based MI estimator known as MINE (Belghazi *et al.*, 2018), which was described in Section 2.4.5.1.

7.1.1 Datasets & Bayesian Networks

For the initial study, where we investigated the effect of using GRFs on the complexity of the learned latent space of a VAE, we considered two datasets: OneHot (Mescheder *et al.*, 2017; Takahashi *et al.*, 2019) and MNIST (LeCun *et al.*, 2014; Deng, 2012). OneHot's training and test sets consisted of five-dimensional, one-hot encoded vectors: $[1, 0, 0, 0, 0]^T$, $[0, 1, 0, 0, 0]^T$, $[0, 0, 1, 0, 0]^T$, $[0, 0, 0, 1, 0]^T$ and $[0, 0, 0, 0, 1]^T$. MNIST is an image dataset of greyscale handwritten digits in the range 0 to 9 where digits are size-normalized and centred in fixed-sized images of size 28×28. The training and test sets consisted of 50 000 and 10 000 instances, respectively. The dataset was binarized by first standardizing all images and then setting each pixel to 0 or 1 depending on whether the normalized value is negative or positive, respectively.

As in Chapter 5, we also evaluated our SIReN-VAE model by making use of various synthetic and real-world datasets that each have an associated true or hypothesised BN graph. These datasets will be referred to as the *graphical* datasets. The synthetic graphical datasets were generated from fully specified BNs. For each BN, all leaf nodes were considered observed, and the rest were taken to be latent. Since VAEs are typically used to encode information into a lower-dimensional representation, we only considered the datasets from the GRF investigation for which there were fewer latent than observed variables. We therefore again used the EColi and MEHRA datasets presented in Chapter 5. We also used an altered version of the Arithmetic Circuit dataset, where the number of observed variables had been increased from 2 to 10 (see Appendix A.2.1 for more detail). Lastly, we introduced a second, larger Gaussian BN, Arth, which, like EColi, is also from the BN repository of Scutari (2022).

Based on the current methodology of using neural networks and standard gradient descent optimization, we were limited to datasets of continuous variables. Figure 7.1 illustrates the BN graphs associated with these datasets. Table 7.1 provides a summary of the properties of the BNs, with EColi and MEHRA included for completeness. Further information on the specifications of the synthetic BNs is given in Appendix A.2.1.

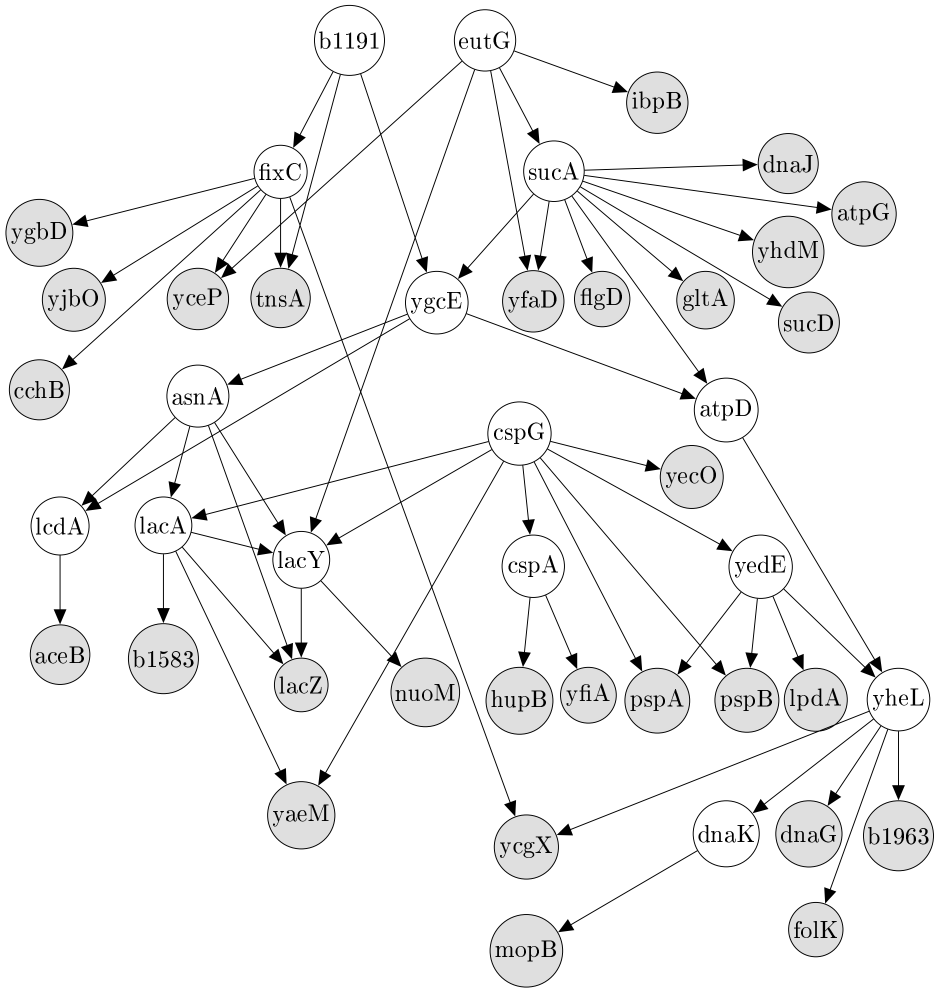
Table 7.1: A summary of key properties of the BN graphs associated with each dataset. D and K denote the number of observed and latent variables in each graph, respectively, and E is the number of directed edges. We also provide the longest shortest path between any observed and latent vertex reachable from that observation in the inverted graph, $\mathcal{G}^{-1}$. This is the minimum number of flow steps needed for inference as discussed in Section 4.4.

BN	Real/Synthetic	D	K	E	$\max_{x_i, z_j \in \mathcal{G}^{-1}} d(x_i, z_j)$
Arithmetic Circuit 2	Synthetic	10	5	15	3
EColi	Synthetic	29	15	59	4
Arth	Synthetic	67	40	150	5
MEHRA	Real	7	3	10	2

7.1.2 Model Architecture & Training

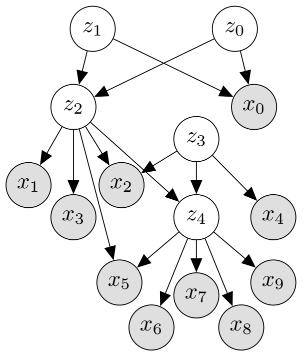
Architectures We began the empirical investigation in Section 7.2 by examining the effect on the learned latent distributions when using GRFs for the prior and posterior of a VAE. We considered a VAE where only the inference network of a vanilla VAE was replaced by a (generative) GRF, and the prior remained a standard Gaussian distribution. We denote this by VAE+GRF$_g$. We also considered a VAE where both the prior and posterior latent distributions were enhanced with GRFs, denoted by VAE+GRF$_g$+GRF$_n$. This is the SIReN-VAE model, although we did not assume any specific conditional independencies at this point. Details on the exact model architectures used for each of the datasets in this setting is given in Appendix A.2.2.1.

For the rest of the investigation, we considered incorporating various dependency structures into a SIReN-VAE. These dependency structures were either fully-connected, random, or adhered to the true dependency structure associated with the given dataset. We denote these models by SIReN-VAE$_{\mathrm{FC}}$, SIReN-VAE$_{\mathrm{Rand}}$ and SIReN-VAE$_{\mathrm{True}}$, respectively. SIReN-VAE$_{\mathrm{FC}}$ made no independency assumptions and used GRF$_n$ and GRF$_g$ to encode a fully-connected structure between the latent variables of the generative and inference model. In this setting, each observed variable was conditioned on all latent variables. For a specific dataset, SIReN-VAE$_{\mathrm{Rand}}$ encoded a BN graph with the same number of edges as the graph encoded by SIReN-VAE$_{\text{True}}$, but where these edges had been assigned to random pairs of vertices at initialization. Care was however taken that all observed variables were connected to at least one latent variable, and that no latent variables were completely disconnected from the graph. This was done by first connecting each observed variable to a random latent variable, and if disconnected latent variables remained, connecting each remaining one to another latent or observed variable. Any remaining edges were assigned between a random latent variable and another latent or observed variable. Both SIReN-VAE$_{\text{FC}}$ and SIReN-VAE$_{\text{Rand}}$ used the same latent dimension as SIReN-VAE$_{\text{True}}$ (i.e. the true BN).

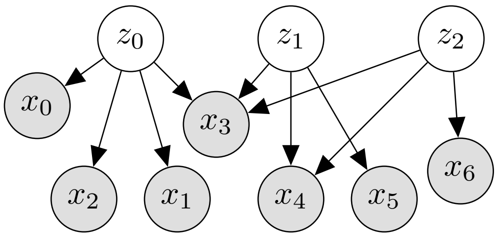


(a) EColi

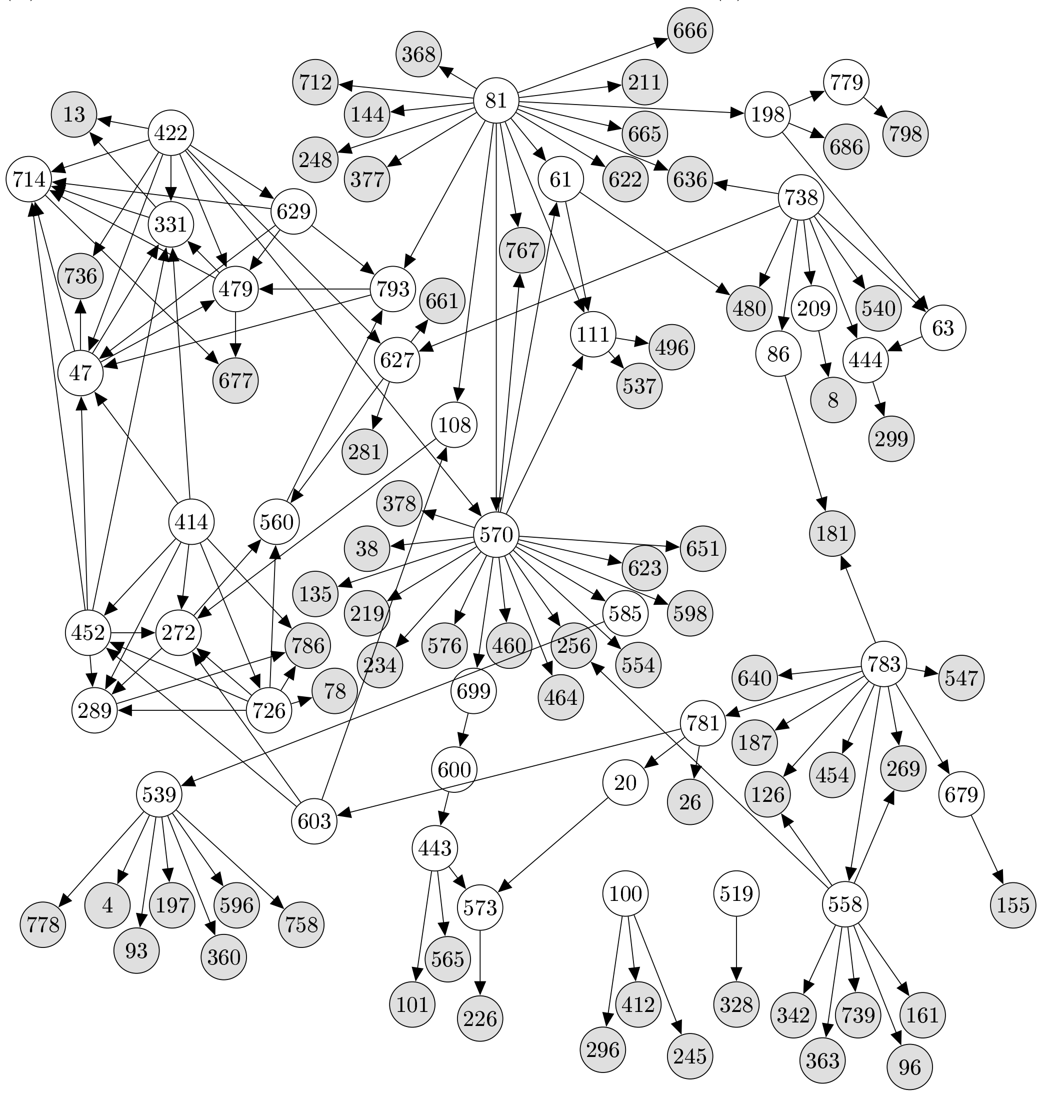
Figure 7.1 (Continued on following page): BN graphs associated with the graphical datasets. White nodes indicate latent variables, whereas shaded nodes are observed.



(b) Arithmetic Circuit 2



(c) MEHRA



(d) Arth

Figure 7.1 (Continued): BN graphs associated with the graphical datasets.

Details on the exact model architectures used for each of the datasets is given in Appendix A.2.2.2. Briefly, each GRF_n and GRF_g had five flow steps, where each residual block had a single hidden layer of width 100 and used the LipMish activation function. Similarly, the decoder neural network had a single hidden layer of width 100. The number of flow steps were chosen so that for all datasets the inference network would have enough steps in line with the longest shortest path presented in Table 7.1. As found during the GRF investigation, adding more hidden layers to the residual blocks had very little impact compared to adding more flows steps and we therefore only considered using one hidden layer in all residual blocks. The hyperparameter c, used for constraining the spectral norm of GRF residual blocks, was set to 0.99 in all cases, to allow each step to have as much expressive capacity as possible.

When SIReN-VAE models were compared to the vanilla VAE architecture, the vanilla VAE encoder and decoder neural networks used an architecture corresponding to the SIReN-VAE decoder: these networks had a single hidden layer with 100 hidden units. The vanilla VAE also used the same latent dimension as the given SIReN-VAE model. Since the Arth BN is much larger than the others we considered, we set the hidden layer width to 200 units in all residual blocks and decoder (and encoder) neural networks for this dataset only.

Training All models were trained using the Adam optimizer (Kingma and Ba, 2015) (with default parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$) with an initial learning rate of either 0.01 or 0.001 and a mini-batch size of 100 instances. The learning rate was decreased by a factor of 10 each time no improvement in the loss was observed for a set number of consecutive epochs, until a minimum learning rate of 10^{-6} was reached, at which point training was terminated. The initial learning rate and duration before learning rate reduction was chosen based on the lowest validation loss obtained over the grid $\{0.01, 0.001\} \times \{10, 20, 30\}$. The training, validation and test sets of the synthetic datasets consisted of 10 000, 5000 and 5000 instances, respectively. The MEHRA dataset was randomly split into 4000 training, 1000 validation and 1885 test instances.

7.2 Effect of GRFs on the Latent Distribution

We investigated the effect of using GRFs to represent the latent variables' distribution in a VAE. We proceeded by first replacing only the inference network of a standard VAE with a (conditional) generative GRF, denoted by VAE+GRF$_g$. Next, we also replaced the prior latent distribution with a normalizing GRF. The resulting model, denoted by VAE+GRF$_g$+GRF$_n$, corresponds to a version of our proposed SIReN-VAE that encodes a fully-connected BN. Figure 7.2 shows the learned latent distributions of these models on the OneHot dataset, and Figure 7.3 shows some samples drawn from these models after being trained on MNIST. We also summarize different performance metrics for these models on the OneHot and MNIST datasets in Table 7.2. Specifically, we estimated the log-evidence ($\log p(\mathbf{x})$) of new test data under the model using 50 importance weighted samples per test point, as detailed in Section 2.4.3, and calculated the average reconstruction error. For MNIST, we also measured the quality of the images generated by the trained models using the Fréchet Inception Distance (FID) score (Heusel *et al.*, 2017), which mimics human perception of similarity between model generated and true images. To compute the FID scores, we used the implementation of Seitzer (2020) which is a port of the official implementation of FID to PyTorch.

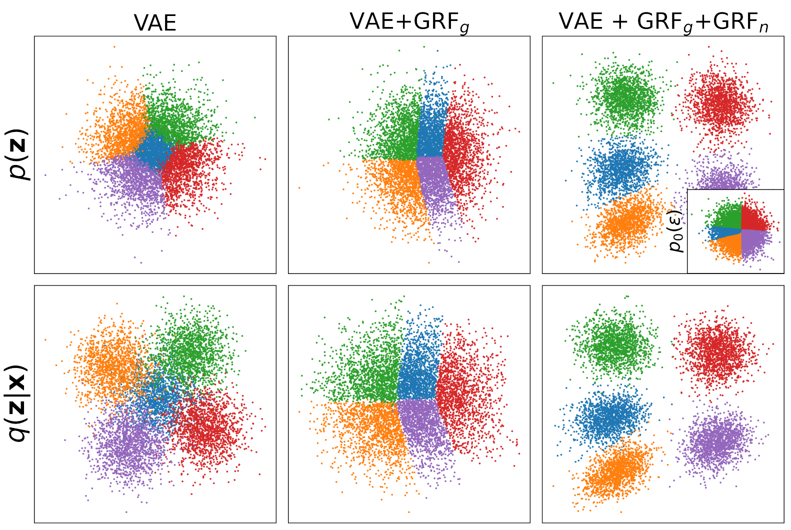


Figure 7.2: Visualization of latent distributions on the OneHot dataset. Each colour corresponds to the latent representation of one dimension of the one-hot encoded observed vectors. The first row plots samples drawn from the learned prior, $p(\mathbf{z})$, of each of the models. In the rightmost panel, $p_0(\boldsymbol{\epsilon}) = \mathcal{N}(\mathbf{0}, I_2)$—see inset—is the base distribution of the flow, GRF$_n$. The second row plots samples drawn from the posterior, $q(\mathbf{z}|\mathbf{x})$. Increasing the complexity of the inference network, as in VAE+GRF$_g$, allows the aggregate posterior to more closely match the prior. Adding a flow prior, allows VAE+GRF$_g$+GRF$_n$ to learn a much more semantically meaningful latent space, with clear separation between the encodings of the different classes.

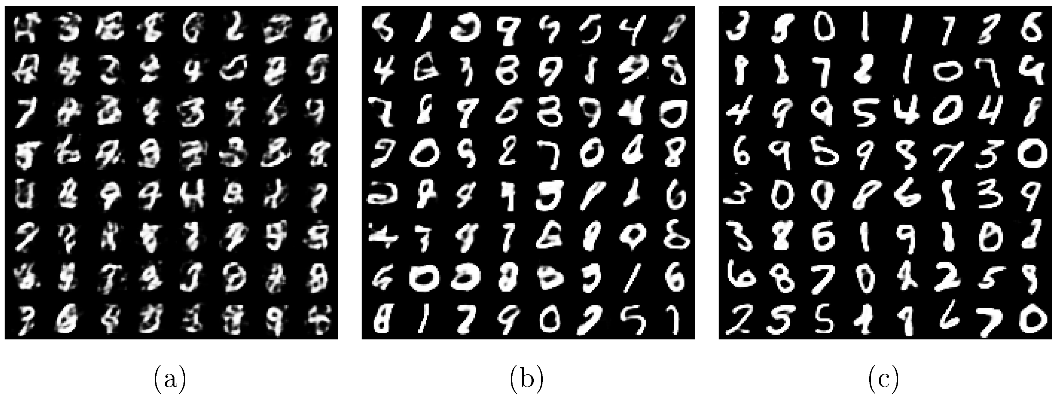


Figure 7.3: Samples drawn from (a) VAE, (b) VAE+GRF$_g$ and (c) VAE+GRF$_g$+GRF$_n$ trained on MNIST. We plot the conditional means given samples drawn from the prior.

Increasing the complexity of the inference network, by for example using a GRF, allowed the aggregate posterior to more closely match the prior. Column 2 of Figure 7.2 shows this clearly for VAE+GRF$_g$ and the OneHot dataset. In contrast, the restrictive Gaussian variational posterior of the standard VAE combined with the over-regularization incurred by the Gaussian prior did not allow this, and caused the posterior distributions of the different classes of OneHot to overlap as can be seen in column 1. This made it more difficult for the generative model to accurately decode the latent variables, which could explain the poorer log-evidence and reconstruction error on this dataset. When extending the prior of the VAE with a GRF as well, the resulting model achieved the highest log-evidence for both datasets, with $-\log p(\mathbf{x})$ on OneHot being very close to the optimal value of $-\log(5) \approx 1.61$. We also noted that in the case of OneHot, VAE+GRF$_g$+GRF$_n$ learned a much more semantically

Table 7.2: Performance of different models on the OneHot and MNIST datasets: $-\log p(\mathbf{x})$ denotes the average negative log-evidence of the test data; RE denotes the average reconstruction error on the test data, which for each test point is measured as the Frobenius norm between the expected value of the reconstruction and the true observation; FID is the Fréchet Inception Distance score for MNIST. Lower is better in all cases. The best result in each column is given in bold.

Model	OneHot		MNIST		
	$-\log p(\mathbf{x})$	RE	$-\log p(\mathbf{x})$	RE	FID
VAE	1.90	0.25	82.30	3.86	119.96
VAE+GRF$_g$	1.66	0.04	65.57	3.25	109.83
VAE+GRF$_g$+GRF$_n$	**1.63**	**0.01**	**59.40**	**3.03**	**99.25**

meaningful latent space, with clear separation between the encodings of the different classes. On MNIST, incorporating GRFs led not only to a better log-evidence and lower reconstruction error, but also to visually higher quality generated samples.

As discussed in Section 3.2.1, increasing the modelling capacity of either the inference network or the model prior can lead to improved performance by reducing the approximation gap and mitigating the prior hole problem. We are therefore not surprised that the bottom row in Table 7.2 shows the best performance measurements for both datasets. Having validated that our use of GRFs allows for more complex latent priors while still allowing the aggregate posterior to match this prior, we next investigated using these flows as a vehicle to inject domain knowledge about the dependency structure between variables.

7.3 Incorporating Graphical Structures

We next applied our SIReN-VAE model to datasets that have an associated true or hypothesised BN dependency structure. Table 7.3 presents the results for these models for the graphical datasets from Table 7.1. We compared encoding a fully-connected structure (SIReN-VAE$_{\text{FC}}$), a random structure (SIReN-VAE$_{\text{Rand}}$), and the true dependency structure (SIReN-VAE$_{\text{True}}$), via the GRFs. We again estimated the average log-evidence using 50 importance weighted samples per test point. We also determined the number of latent variables that had collapsed during training. To measure whether a specific latent variable $\mathbf{z}$ has become inactive and collapsed towards the (uninformative) prior, we used the same statistic as presented by Burda *et al.* (2016): $A_z = \text{Var}_{\mathbf{x}\sim p_{\mathcal{D}}}(\mathbb{E}_{z\sim q(z|\mathbf{x})}[z])$, where $p_{\mathcal{D}}$ is the training data distribution.[1] This is based on the assumption that if a latent dimension encodes useful information about the data, then it should change as the observations change. As in Burda *et al.* (2016), a latent dimension is deemed inactive if $A_z \leq 0.01$.

In these initial experiments, SIReN-VAE$_{\text{True}}$ was not able to match the performance of SIReN-VAE$_{\text{FC}}$ in terms of log-evidence on most of the datasets. We also note that except for Arth, the model that achieved the best log-evidence also had the fewest collapsed latent variables. Models with more collapsed variables, must effectively represent the same observation space with a smaller latent dimension, which could partially explain their poorer performance. We will continue our investigation of this phenomenon and possible remedies in the next section.

[1]Note that this differs slightly from the notation used in Burda *et al.* (2016): $A_z = \text{Cov}_{\mathbf{x}}(\mathbb{E}_{z\sim q(z|\mathbf{x})}[z])$, but since covariance is measured between two variables, denoting the statistic as the *variance* over the expected values of the latent variables given the data, is more suitable.

Table 7.3: Performance of different models on the graphical datasets, where $-\log p(\mathbf{x})$ denotes the average negative log-evidence of the test data. We also provide the number of inactive units after training. All results are averages over five independent runs, with standard deviation given in the subscript. A standard deviation of less than 0.005 is indicated with Δ. Lower is better in all cases. The best average result for each dataset is given in bold.

BN	Model	$-\log p(\mathbf{x})$	Number of Inactive Units
Arithmetic Circuit 2	VAE	$9.78_{\pm 0.01}$	$2.00_{\pm 0.00}$
	SIReN-VAE$_{\text{FC}}$	$\mathbf{9.76_{\pm 0.02}}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{Rand}}$	$11.09_{\pm 0.32}$	$1.20_{\pm 0.75}$
	SIReN-VAE$_{\text{True}}$	$10.03_{\pm 0.01}$	$2.00_{\pm 0.00}$
EColi	VAE	$35.03_{\pm 0.02}$	$5.00_{\pm 1.26}$
	SIReN-VAE$_{\text{FC}}$	$35.03_{\pm 0.03}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{Rand}}$	$41.63_{\pm 0.53}$	$1.80_{\pm 1.47}$
	SIReN-VAE$_{\text{True}}$	$\mathbf{34.99_{\pm 0.01}}$	$\mathbf{0.00_{\pm 0.00}}$
Arth	VAE	$\mathbf{37.43_{\pm 0.09}}$	$25.25_{\pm 7.27}$
	SIReN-VAE$_{\text{FC}}$	$37.55_{\pm 0.06}$	$7.20_{\pm 2.40}$
	SIReN-VAE$_{\text{Rand}}$	$40.81_{\pm 0.18}$	$\mathbf{5.20_{\pm 2.04}}$
	SIReN-VAE$_{\text{True}}$	$37.73_{\pm 0.05}$	$14.80_{\pm 0.33}$
MEHRA	VAE	$7.63_{\pm 0.02}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{FC}}$	$\mathbf{7.57_{\pm 0.01}}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{Rand}}$	$8.91_{\pm 0.24}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{True}}$	$8.37_{\pm 0.06}$	$\mathbf{0.00_{\pm 0.00}}$

Even though SIReN-VAE$_{\text{True}}$ had no collapsed variables when trained on the MEHRA dataset, it was still not able to match the performance of SIReN-VAE$_{\text{FC}}$. BNs constructed for real-world domains tend to focus more on the key influences between variables. As such, smaller interactions may be neglected, which could have limited SIReN-VAE$_{\text{True}}$'s ability to achieve comparable log-evidence scores. In all cases, SIReN-VAE$_{\text{Rand}}$ did significantly worse than the other models on the main tasks. This shows that the encoded structure does play a significant role in modelling performance and that using the true (or hypothesised) BN structure aids in learning appropriate observational and latent distributions. We therefore did not consider SIReN-VAE$_{\text{Rand}}$ in subsequent investigations. We note that SIReN-VAE$_{\text{FC}}$ did not notably outperform the standard VAE on the EColi and Arth datasets. This is possibly due to the fact that the variables of these two synthetic datasets have relatively simple Gaussian conditional distributions. The extra complexity added by the flows could therefore have been redundant while complicating optimization.

7.4 Addressing Posterior Collapse

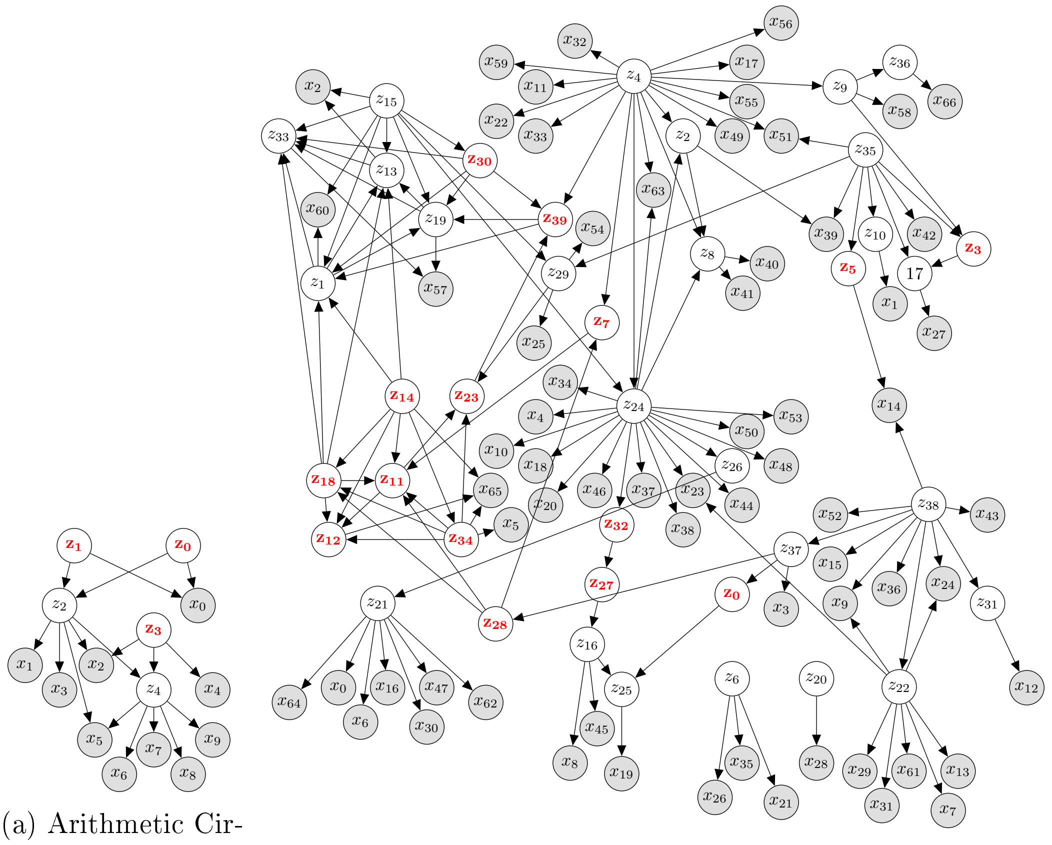
As observed in the previous section, SIReN-VAE$_{\text{True}}$ sometimes suffers from posterior collapse. Although not as severe as with the vanilla VAE—likely because SIReN-VAE$_{\text{True}}$ can obtain a tighter variational bound due to the complexity added by the flows—we are especially motivated to avoid posterior collapse for SIReN-VAE in order to learn meaningful latent distributions in line with the provided BN structure. Before considering different remedies for this issue in Section 7.4.2, we first investigated whether the position of a latent variable within the BN graph gives any indication of its propensity to collapse.

7.4.1 Effect of Encoded Structure on Posterior Collapse

Establishing correspondence between the position of a latent variable within the BN graph and the likelihood that it will collapse, would suggests that encoding the BN structure into the VAE further contributes to posterior collapse, in addition to the general factors discussed in Section 2.4.4. To obtain insights into whether this is the case, we visualize in Figure 7.5 the logarithm of the average value of A_z for each latent variable in each dataset. The averages were computed over the five runs used to the generate the results of the previous section. For comparison, we do the same for the vanilla VAE and SIReN-VAE$_{\text{FC}}$ models of the previous section.

We would expect the indices of collapsed latent variables to be arbitrary if the encoded structure plays no role. The average A_z value over multiple runs should then result in all latent variables having a similar average activity score. This is what we would expect for the vanilla VAE, where the latent variables are independent and have the same prior distribution. Considering Figure 7.5, this was indeed the case as the average activity of the vanilla VAE's latent variables was far more uniform than, for example, the activity of SIReN-VAE$_{\text{True}}$'s latent variables.

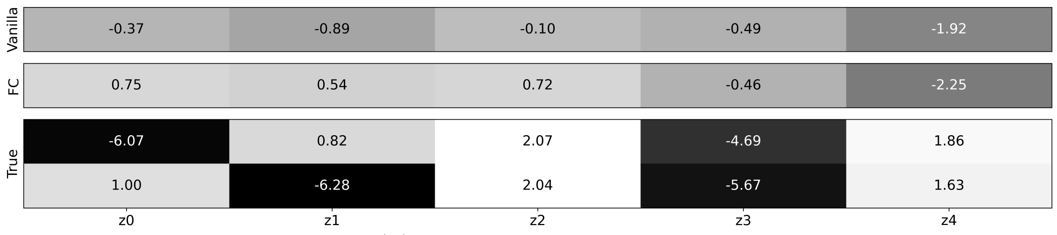
For both SIReN-VAE$_{\text{FC}}$ and SIReN-VAE$_{\text{True}}$ we see some patterns in the activity of the latent variables emerge. Interestingly, for SIReN-VAE$_{\text{FC}}$, the higher the index of the latent, the lower its activity was on average. For the fully-connected BNs, each variable was conditioned on all variables with a lower index. This suggests that the model may tend to de-emphasize latent variables that are conditioned on many other latent variables. Figure 7.5 shows that when posterior collapse occurred in SIReN-VAE$_{\text{True}}$, it tended to happen for certain latent variables far more than for others—unlike with the vanilla VAE. This is perhaps most apparent for the Arithmetic Circuit 2 dataset depicted in Figure 7.5a. where it is clear that z_0, z_1 and z_3 were the most prone to collapse. Figure 7.4 highlights these vertices within the corresponding BN structure. We note that there is structural symmetry between vertices z_0 and z_1 of the Arithmetic Circuit 2 dataset, which could explain why the model chose to ignore one of them during each run. Further inspection of the positions of the collapsed variables in the BNs depicted in Figure 7.4 shows that regularly collapsing variables typically only shared edges with other latent variables, and not with any observed variables. Prominent examples of this are the various intermediate nodes in chains of latent variables in Figure 7.4b. These results corroborate the findings of Burda *et al.* (2016), who observed that the latent variables in the higher layers of a hierarchical VAE (which do not directly influence any observed variables) are more prone to collapse. Note that no posterior collapse occurred for MEHRA, where there is only a single layer of hidden variables that are all connected to observations.



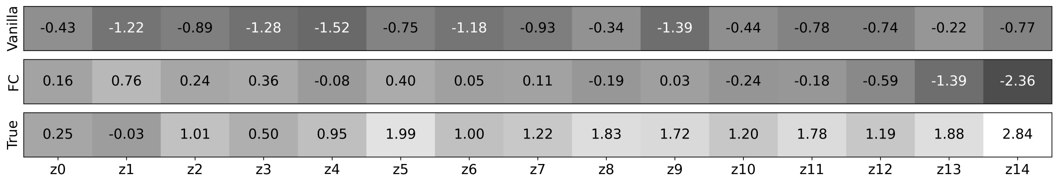
(a) Arithmetic Circuit 2

(b) Arth

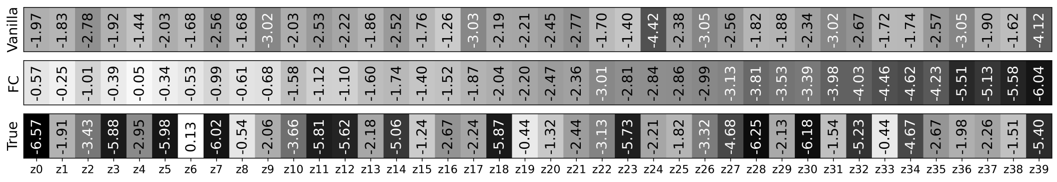
Figure 7.4: Posterior collapse for SIReN-VAE$_{\text{True}}$ in the Arithmetic Circuit 2 and Arth BNs. White nodes indicate latent variables, whereas shaded nodes are observed. Nodes with red labels typically collapsed. These typically have symmetry with other latent variables, as between z_0 and z_1 in (a), or are only connected to other latent variables and not to any observed variables. Note that for MEHRA and EColi, no posterior collapse occurred.



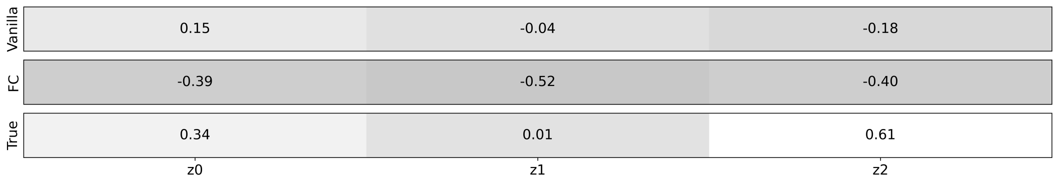
(a) Arithmetic Circuit 2



(b) EColi



(c) Arth



(d) MEHRA

Figure 7.5: The average of the latent variable activity statistic, A_{z_i}, over five independent runs for each of the latent variables z_i, $i = 0, \dots, K-1$ of the graphical datasets. The top, middle and bottom bars for each dataset correspond to the vanilla VAE, SIReN-VAE$_{\text{FC}}$ and SIReN-VAE$_{\text{True}}$ models, respectively. Each entry corresponds to the (logarithm of the) average activity statistic A_{z_i}, of a specific latent variable, z_i. The lower this value, the darker the corresponding block. Note that $\log(A_{z_i}) \leq \log(0.01) \approx -4.6$ is the point at which a latent variable is declared collapsed. We use logarithms here in order to better distinguish the typically inactive and active latent variables. We observed that when SIReN-VAE$_{\text{True}}$ was trained on the Arithmetic Circuit 2 dataset, either z_0 or z_1 collapsed during a given run, but never both. We make this clear by providing two separate plots: the first row in (a) for SIReN-VAE$_{\text{True}}$ gives the average activity statistics over the runs where z_0 collapsed, and the second gives the statistics over the runs where z_1 collapsed.

7.4.2 Mitigating Posterior Collapse

Since we would like the learned latent distribution of SIReN-VAE$_{\text{True}}$ to respect the provided BN structure and thus utilize all the latent dimensions, the fact that aspects of the encoded structure make posterior collapse more likely, makes it doubly important for us to try to address it in some way. We therefore next proceeded to investigate the efficacy of warm-up (WU), importance-weighted (IW) objectives and doubly-reparameterized gradient estimates (DReG) in combatting this posterior collapse.

Table 7.4 provides the results when training SIReN-VAE$_{\text{True}}$ on the graphical datasets using different combinations of these interventions. We present the results also for datasets where no posterior collapse occurred, since we want to find a generally applicable approach. As in Table 7.3, we estimated the average log-evidence using 50 importance weighted samples per test point, as well as the number of latent variables that collapsed during training. Since we found that a certain subset of latent variables was more prone to collapse than others, we compared two different approaches to warm-up. The standard, more naïve, approach applied the warm-up factor, β, to all latent variables, and is denoted by WU$_{\text{all}}$. For the Arithmetic Circuit 2 and Arth datasets, where posterior collapse occurred in the previous section, we also applied the warm-up term to only those latent variables that typically collapsed during the earlier runs as identified in Figure 7.4. We denote this 'retrained' approach by WU$_{\text{select}}$:

$$\mathcal{L}_t^{\text{WU}_{\text{select}}}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \sum_{i=1}^{K}(\beta_t^{c_i}) \cdot (\log p_i(z_i) - \log q_{i,\phi}(z_i|\text{Pa}_{z_i}^{\mathcal{G}}))\right] ,$$

where $\beta_t \in \mathbb{R}$ is increased linearly from 0 to 1 during the first N_{WU} epochs, $\text{Pa}_{z_i}^{\mathcal{G}}$ are the parent vertices of z_i in the dataset's associated BN graph, $\mathcal{G}$, and $c_i = 1$ if z_i typically collapsed earlier, else $c_i = 0$. For comparison, Table 7.5 gives the results when training SIReN-VAE$_{\text{FC}}$ using the same mitigating techniques (where we only applied warm-up if the original model had collapsed variables). In all cases, a default warm-up period of $N_{\text{WU}} = 100$ was chosen. If this did not lead to a reduction in the number of collapsed variables, warm-up periods in the list [50, 70, 125, 150] were also considered in order. Only for the Arth dataset was it necessary to choose an alternative warm-up period, where $N_{\text{WU}} = 75$ was used. For the Arithmetic Circuit 2 and MEHRA datasets, the models using the IW objective employed $k = 32$ importance weighted samples. For the Gaussian BNs, EColi and Arth, only $k = 8$ importance weighted samples were used since further increasing k did not lead to notably better results. Additionally, using DReG estimates comes at both a time and memory cost such that using a larger number of importance weights slowed down training and also led to out-of-memory errors for these larger BNs.[2]

[2]In these situations, using the memory-saving technique for the GRF, as discussed in Section 4.3.1, proved useful. We did not implement it for models applying warm-up, however, due to time constraints and because we did not find it necessary to consider larger values of k for the EColi and Arth datasets.

Table 7.4: Performance of SIReN-VAE$_{\text{True}}$ on the graphical datasets when applying different combinations of posterior collapse prevention techniques, namely: warm-up (WU), importance weighted objectives (IWAE) and doubly-reparameterized gradient estimates (DReG). Warm-up is applied to either all latents (WU$_{\text{all}}$), or to a selected subset that is prone to collapse (WU$_{\text{select}}$). The performance metrics are the same as those for Table 7.3.

BN	Model	$-\log p(\mathbf{x})$	Number of Inactive Units
Arithmetic Circuit 2	SIReN-VAE$_{\text{True}}$	$10.03_{\pm 0.01}$	$2.00_{\pm 0.00}$
	SIReN-IWAE$_{\text{True}}$	$9.86_{\pm 0.04}$	$1.00_{\pm 0.00}$
	SIReN-IWAE$_{\text{True}}$+DReG	**$9.80_{\pm 0.02}$**	$1.00_{\pm 0.00}$
	SIReN-VAE$_{\text{True}}$+WU$_{\text{all}}$	$10.11_{\pm 0.08}$	$1.00_{\pm 0.00}$
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{all}}$	$9.86_{\pm 0.08}$	$1.00_{\pm 0.00}$
	SIReN-IWAE$_{\text{True}}$+DReG+WU$_{\text{all}}$	$9.88_{\pm 0.06}$	$1.00_{\pm 0.00}$
	SIReN-VAE$_{\text{True}}$+WU$_{\text{select}}$	$9.82_{\pm 0.02}$	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{select}}$	**$9.80_{\pm \Delta}$**	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$+DReG+WU$_{\text{select}}$	**$9.80_{\pm 0.02}$**	**$0.00_{\pm 0.00}$**
EColi	SIReN-VAE$_{\text{True}}$	$34.99_{\pm 0.01}$	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$	**$34.98_{\pm \Delta}$**	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$+DReG	**$34.98_{\pm 0.01}$**	**$0.00_{\pm 0.00}$**
	SIReN-VAE$_{\text{True}}$+WU$_{\text{all}}$	$34.99_{\pm 0.01}$	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{all}}$	$34.99_{\pm 0.01}$	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$+DReG+WU$_{\text{all}}$	$34.99_{\pm \Delta}$	**$0.00_{\pm 0.00}$**
Arth	SIReN-VAE$_{\text{True}}$	$37.73_{\pm 0.05}$	$14.80_{\pm 0.33}$
	SIReN-IWAE$_{\text{True}}$	$37.49_{\pm 0.26}$	$11.00_{\pm 1.10}$
	SIReN-IWAE$_{\text{True}}$+DReG	$37.30_{\pm 0.53}$	$6.00_{\pm 1.41}$
	SIReN-VAE$_{\text{True}}$+WU$_{\text{all}}$	$37.65_{\pm 0.08}$	$11.20_{\pm 0.98}$
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{all}}$	$37.48_{\pm 0.47}$	$10.20_{\pm 1.33}$
	SIReN-IWAE$_{\text{True}}$+DReG+WU$_{\text{all}}$	**$37.27_{\pm 0.54}$**	$7.80_{\pm 1.33}$
	SIReN-VAE$_{\text{True}}$+WU$_{\text{select}}$	$37.71_{\pm 0.03}$	$9.80_{\pm 2.64}$
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{select}}$	$37.42_{\pm 0.24}$	$7.00_{\pm 2.28}$
	SIReN-IWAE$_{\text{True}}$+DReG+WU$_{\text{select}}$	$37.43_{\pm 0.22}$	**$1.80_{\pm 0.98}$**
MEHRA	SIReN-VAE$_{\text{True}}$	$8.37_{\pm 0.06}$	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$	$8.19_{\pm 0.06}$	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$+DReG	**$8.08_{\pm 0.02}$**	**$0.00_{\pm 0.00}$**
	SIReN-VAE$_{\text{True}}$+WU$_{\text{all}}$	$8.66_{\pm 0.26}$	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{all}}$	$8.39_{\pm 0.24}$	**$0.00_{\pm 0.00}$**
	SIReN-IWAE$_{\text{True}}$+DReG+WU$_{\text{all}}$	$8.23_{\pm 0.08}$	**$0.00_{\pm 0.00}$**

Table 7.5: Performance of SIReN-VAE$_{\text{FC}}$ on the graphical datasets when applying different combinations of posterior collapse prevention techniques, namely: warm-up (WU), importance weighted objectives (IWAE) and doubly-reparameterized gradient estimates (DReG). Warm-up is applied to all latents. The performance metrics are the same as those for Table 7.3. We include the best performing SIReN-VAE$_{\text{True}}$ (in terms of $-\log p(\mathbf{x})$) from Table 7.4, for ease of comparison.

BN	Model	$-\log p(\mathbf{x})$	Number of Inactive Units
Arithmetic Circuit 2	SIReN-IWAE$_{\text{True}}$+DReG+WU$_{\text{select}}$	$9.80_{\pm 0.02}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{FC}}$	$9.76_{\pm 0.02}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-IWAE$_{\text{FC}}$	$\mathbf{9.73_{\pm 0.01}}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-IWAE$_{\text{FC}}$+DReG	$\mathbf{9.73_{\pm \Delta}}$	$\mathbf{0.00_{\pm 0.00}}$
EColi	SIReN-IWAE$_{\text{True}}$+DReG	$\mathbf{34.98_{\pm 0.01}}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{FC}}$	$35.03_{\pm 0.03}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-IWAE$_{\text{FC}}$	$35.04_{\pm 0.01}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-IWAE$_{\text{FC}}$+DReG	$35.06_{\pm 0.01}$	$\mathbf{0.00_{\pm 0.00}}$
Arth	SIReN-IWAE$_{\text{True}}$+DReG+WU$_{\text{all}}$	$37.27_{\pm 0.54}$	$7.80_{\pm 1.33}$
	SIReN-VAE$_{\text{FC}}$	$37.55_{\pm 0.06}$	$7.20_{\pm 2.40}$
	SIReN-VAE$_{\text{FC}}$+WU$_{\text{all}}$	$37.33_{\pm 0.02}$	$3.40_{\pm 1.02}$
	SIReN-IWAE$_{\text{FC}}$+WU$_{\text{all}}$	$37.21_{\pm 0.14}$	$2.40_{\pm 2.15}$
	SIReN-IWAE$_{\text{FC}}$+DReG+WU$_{\text{all}}$	$\mathbf{37.13_{\pm 0.02}}$	$\mathbf{1.00_{\pm 1.10}}$
MEHRA	SIReN-IWAE$_{\text{True}}$+DReG	$8.08_{\pm 0.02}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{FC}}$	$7.57_{\pm 0.01}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-IWAE$_{\text{FC}}$	$7.58_{\pm 0.03}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-IWAE$_{\text{FC}}$+DReG	$\mathbf{7.50_{\pm 0.03}}$	$\mathbf{0.00_{\pm 0.00}}$

Based on the results in Table 7.4, we cannot single out a single combination of posterior collapse mitigation techniques that provided the best performance for all the datasets, although the addition of these remedies did in general improve the log-evidence and decrease the number of collapsed variables if there were any. We do however take note of the following patterns. For the two datasets where posterior collapse did occur, warm-up aided in reducing the number of ignored latent variables and improving the log-evidence of the model on the test set. When no posterior collapse occurred in the original model, warm-up either did not improve the log-evidence, or led to poorer performance. For the Arithmetic Circuit 2 and Arth datasets, the retrained (WU$_{\text{select}}$) warm-up approach resulted in the fewest collapsed variables. In all cases, the best performing model per dataset incorporated the IW objective and DReG estimates. This indicates that a tighter variational bound and lower-variance gradient estimates are important in mitigating posterior collapse. Using DReG led to significantly higher time and memory usage, however. It is not clear how worthwhile the use of DReG to improve results is, given this additional cost. Future work could potentially consider alternative approaches to mitigating posterior collapse that do not impose these additional costs.

Although warm-up, importance weighted objectives and doubly-reparameterized gradient estimates also helped to improve the performance of SIReN-VAE$_{\text{True}}$, it was still not able to quite match the performance of SIReN-VAE$_{\text{FC}}$ on most of the datasets. This is perhaps most noticeable for the real-world MEHRA dataset, where the issue is that the hypothesized BN might not capture all the subtle dependencies present in the data. We therefore consider alternative benefits of the proposed approach.

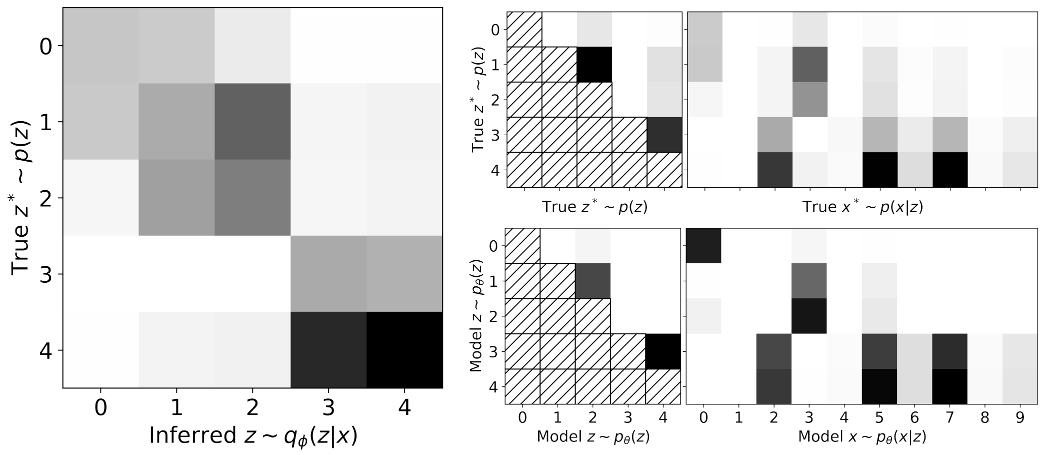
7.5 Interpretability of the Learned Latent Space

One of the anticipated benefits of incorporating hypothesized BN dependency structures into the VAE model is the potential of a more interpretable latent space. Since we had access to the true latent variables of the datasets,[3] we began to investigate whether this benefit is being realized by estimating the mutual information (MI) between the (aggregate) inferred latent distribution of a SIReN-VAE$_{\text{True}}$ model given a set of observations and the true latent distribution that gave rise to those observations. More information is provided below. Mutual information was estimated using the MINE approach of Belghazi *et al.* (2018), as discussed in Section 2.4.5.1. We used the PyTorch implementation of MINE by Tegnér (2020), who reproduced the results of Belghazi *et al.* (2018). We made use of the default settings of the provided implementation, and estimated MI using the same number of samples as in the respective datasets' training sets.

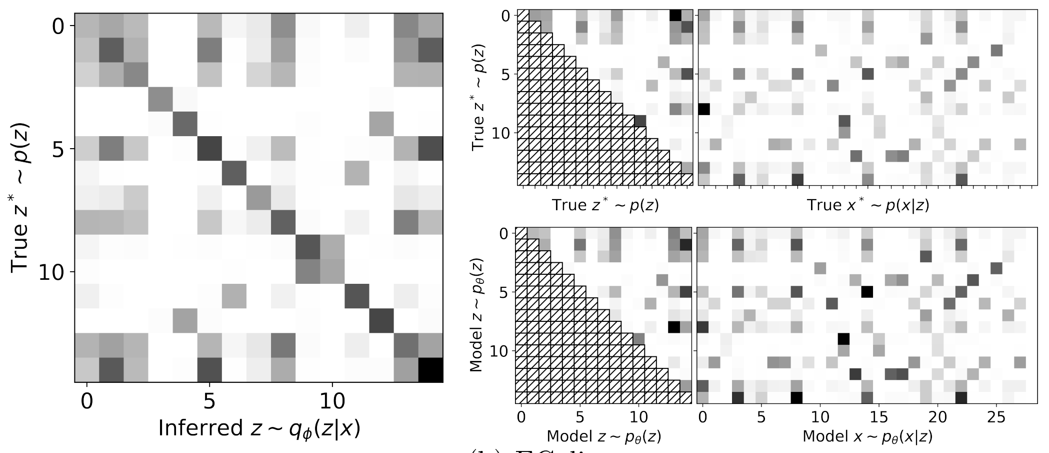
Specifically, we estimated $\mathrm{I}(z_i^*, z_j)$, the MI between each true latent variable z_i^*, and each latent variable of the model, z_j, using the samples of the true latent variables associated with the given dataset's training set of observations, and samples from the inferred latent distributions of SIReN-VAE$_{\text{True}}$ given these observations, respectively. The z_j samples were obtained by sampling a single point from $q_\phi(\mathbf{z}|\mathbf{x})$ for a given observation. Additionally, we also estimated the MI between all pairs of true latent variables, $\mathrm{I}(z_i^*, z_j^*)$, and compared this to the MI between all pairs of latent variables from the SIReN-VAE$_{\text{True}}$ prior.

[3]For the synthetic datasets, the latent variables were sampled from the known model and used to condition the distributions from which the observations were sampled. For the real-world MEHRA dataset, we assumed certain dimensions to be latent for our purposes, and as such have access to the true 'latent' data corresponding to the observations.

Similarly, we also compared the MI between the true latent and observed variables, $\mathrm{I}(z_i^*, x_j^*)$, to the MI between the model's latent and observed variables $\mathrm{I}(z_i, x_j)$, where z_i and x_j are drawn from $p_\theta(\mathbf{x}, \mathbf{z})$. Figure 7.6 gives the results for the graphical datasets. In each case, we used the SIReN-VAE$_{\text{True}}$ model that achieved the best average log-evidence as given in Table 7.4 of the previous section, where this model could employ any combination of the posterior collapse mitigation techniques.

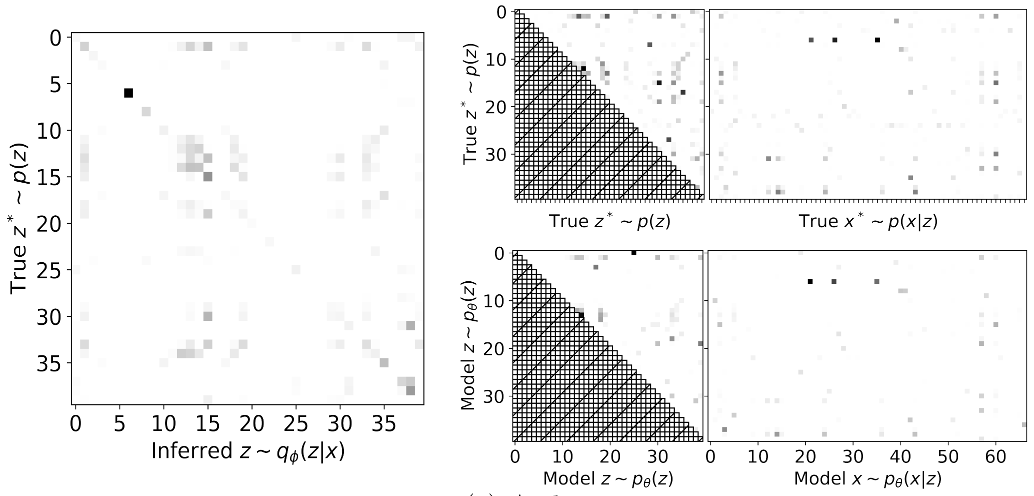


(a) Arithmetic Circuit 2

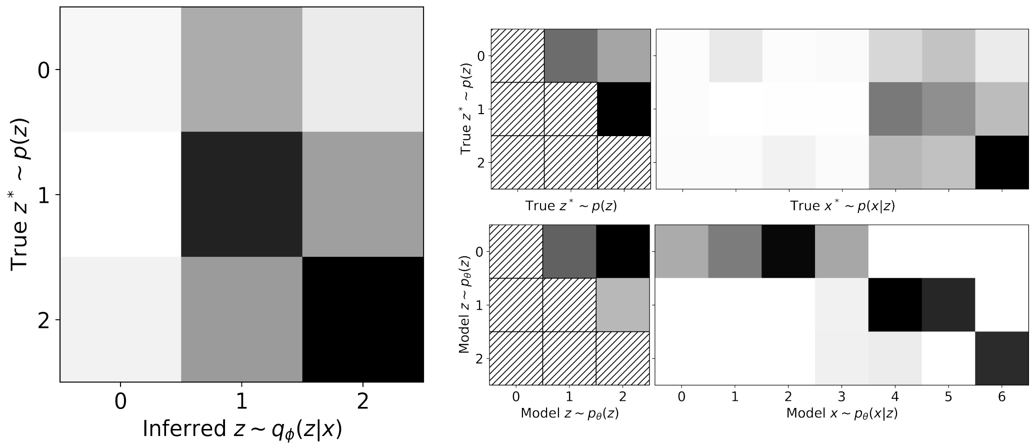


(b) EColi

Figure 7.6 (Continued on following page): Visualization of the MI between latent and observed variables of the true and fitted distributions for different BNs. For each dataset, the entries correspond to (for $i, j = 0, \ldots, K-1$ and $k = 0, \ldots D-1$) the MI between: the true and inferred latent variables, $I(z_i^*, z_j)$ (left); the true latent and observed variables, $I(z_i^*, z_j^*)$ and $I(z_i^*, x_k^*)$ (top right); and the latent and observed variables sampled from the model, $I(z_i, z_j)$ and $I(z_i, x_k)$ (bottom right). Since MI is symmetric, we only plot the upper triangle of the matrices corresponding to $I(z_i^*, z_j^*)$ and $I(z_i, z_j)$. The higher the MI, the darker the associated entry.



(c) Arth



(d) MEHRA

Figure 7.6 (Continued): Visualization of the MI between latent and observed variables of the true and fitted distributions for different BNs.

If SIReN-VAE$_{\text{True}}$ learns meaningful latent representations, with each latent variable affecting its neighbourhood in a way similar to the true hidden variable, we would expect the MI between z_i^* and z_i to be high as well as for $\text{I}(z_i^*, z_j) \approx \text{I}(z_j^*, z_i)$. Visually, this means that we expect the left-most matrix plots of Figure 7.6a to 7.6d to be approximately symmetric and to have a noticeable line on the diagonal. For the synthetic datasets, we see that this holds most noticeably for the EColi dataset (Figure 7.6b). The two plots on the right of Figure 7.6b are also nearly identical, suggesting that SIReN-VAE$_{\text{True}}$ very closely mimics the dependencies of the true model in terms of the MI between variables. Although arguably not as distinct, similar trends are evident for the other datasets as well. For example, for the real-world MEHRA dataset, two out of the three latent variables of the model have very high mutual information with their true counterparts. Although the two right-most plots of

Figure 7.6d are not as similar as those of the synthetic datasets, this is expected since the hypothesized BN does not encode all the subtle dependencies that are present in the data.

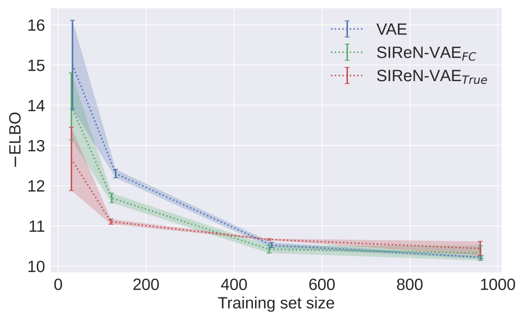
We do however note that the mutual dependencies evident in these plots correspond to the encoded dependency structures of each dataset. One could therefore conclude that these models have been afforded additional interpretability in that one has better insights into which latent variables directly influence each other, than for example in a VAE that encodes no special dependency structure. This could aid in more controlled sample generation, where one could target certain observed variables using the latent variables that they are most dependent on. This could also allow one to assign specific meaning to each of the latent factors of SIReN-VAE$_{\text{True}}$, corresponding to their meaning in the original BN, as for example in the MEHRA BN where the latent variables represent atmospheric conditions.

Next, we investigated another potential strength of the proposed SIReN-VAE$_{\text{True}}$ method: its potential to generalize well when trained on only a small number of instances.

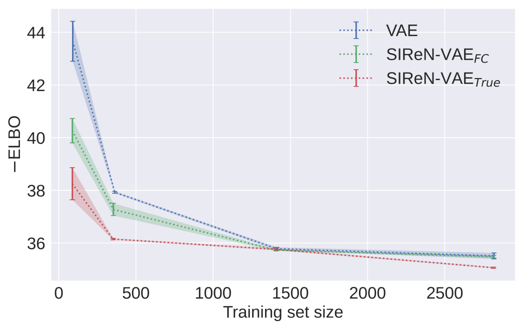
7.6 Benefit of Incorporating Graphical Structures in Data-sparse Settings

As a final investigation, we evaluated the performance of the standard VAE, SIReN-VAE$_{\text{FC}}$ and SIReN-VAE$_{\text{True}}$ (initially without any posterior collapse measures) when trained on only a limited number of training instances. We again measured the models' performance over five independent runs, where for each run we train the model on only $2 \times |\mathcal{G}|$ instances sampled from the original training set, where $|\mathcal{G}| = D + K$ is the number of vertices in the dataset's corresponding BN. We also trained the models while applying selected posterior collapse prevention techniques from Section 7.4. Specifically, we retrained using all combinations of the importance-weighted objective and warm-up. For simplicity, we only considered the naïve warm-up approach where warm-up is applied to all latent variables. The results are given in Table 7.6. We only provide the combination of posterior collapse mitigation techniques that achieved the lowest negative log-evidence. Figure 7.7 additionally shows the performance of the different models (without posterior collapse measures) on each of the datasets, for varying training set sizes.

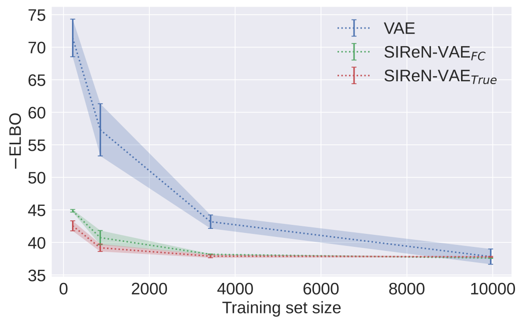
Here, SIReN-VAE$_{\text{True}}$ clearly outperformed the other models, and consistently achieved a better log-evidence on all datasets when trained on a small training set. Promisingly, SIReN-VAE$_{\text{True}}$ performed the best on the real-world dataset MEHRA, even though it did not perform as well as SIReN-VAE$_{\text{FC}}$ when trained on a larger training set. When enough data was available, we argued that the subtle dependencies between variables perhaps not encoded into the MEHRA BN, might have been the reason that SIReN-VAE$_{\text{True}}$ did not match the performance of SIReN-VAE$_{\text{FC}}$ in that setting. When only limited data was available, it was perhaps precisely this coarser-grained structure that prevented SIReN-VAE$_{\text{True}}$ from overfitting and allowed it to achieve better generalization performance. We therefore speculate that the increased sparsity of the neural network weights of SIReN-VAE$_{\text{True}}$, in line with the true (or hypothesized) BN independencies, provides a better inductive bias and thus poses an easier learning task than those posed by the other models, resulting in better generalization to the test set. SIReN-VAE$_{\text{True}}$ is thus valuable in data-sparse settings when one has access to prior knowledge in the form of a BN.



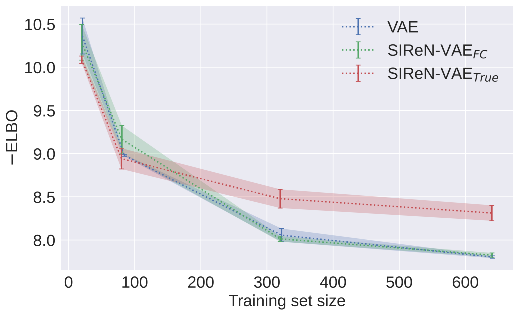
(a) Arithmetic Circuit 2



(b) EColi



(c) Arth



(d) MEHRA

Figure 7.7: Negative ELBO (lower is better) vs training set size achieved by the standard VAE, SIReN-VAE$_{\text{FC}}$ and SIReN-VAE$_{\text{True}}$ on the test set of the various BNs. We considered training sets with sizes of 2, 8, 16 and 32 times the number of vertices in the datasets corresponding BN. Error bars show one standard deviation from the mean over 5 independent runs. SIReN-VAE$_{\text{True}}$ consistently performs the best when limited training data is available.

Table 7.6: Performance of various models on the graphical datasets when trained on only $2 \times |\mathcal{G}|$ samples, where $|\mathcal{G}| = D + K$ is the number of vertices in the corresponding BN. Given, is the negative log evidence ($-\log p(\mathbf{x})$) of test data under each of the models, as well as the number of inactive units after training. All results are the average over five independent runs, with standard deviation given in the subscript. Lower is better in all cases. The best results for each dataset are given in bold.

BN	Model	$2 \times \|\mathcal{G}\|$ training samples	
		$-\log p(\mathbf{x})$	Number of Inactive Units
Arithmetic Circuit 2	VAE	$13.90_{\pm 0.73}$	$\mathbf{0.00_{\pm 0.00}}$
	IWAE	$13.28_{\pm 0.95}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{FC}}$	$13.14_{\pm 0.78}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-IWAE$_{\text{FC}}$	$12.78_{\pm 0.43}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{True}}$	$12.29_{\pm 0.84}$	$2.60_{\pm 0.49}$
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{all}}$	$\mathbf{11.88_{\pm 0.27}}$	$\mathbf{0.00_{\pm 0.00}}$
EColi	VAE	$40.91_{\pm 1.07}$	$\mathbf{0.00_{\pm 0.00}}$
	IWAE	$40.39_{\pm 0.22}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{FC}}$	$38.84_{\pm 0.22}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-IWAE$_{\text{FC}}$	$38.86_{\pm 0.28}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{True}}$	$37.54_{\pm 0.33}$	$4.20_{\pm 0.75}$
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{all}}$	$\mathbf{37.45_{\pm 0.41}}$	$\mathbf{0.00_{\pm 0.00}}$
Arth	VAE	$64.22_{\pm 2.14}$	$\mathbf{0.00_{\pm 0.00}}$
	IWAE	$56.39_{\pm 2.33}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{FC}}$	$43.35_{\pm 0.10}$	$2.40_{\pm 1.36}$
	SIReN-IWAE$_{\text{FC}}$+WU$_{\text{all}}$	$43.02_{\pm 0.80}$	$7.40_{\pm 3.93}$
	SIReN-VAE$_{\text{True}}$	$41.56_{\pm 0.53}$	$19.40_{\pm 2.06}$
	SIReN-IWAE$_{\text{True}}$+WU$_{\text{all}}$	$\mathbf{41.09_{\pm 0.42}}$	$15.00_{\pm 1.26}$
MEHRA	VAE	$10.57_{\pm 0.21}$	$\mathbf{0.00_{\pm 0.00}}$
	IWAE	$10.41_{\pm 0.29}$	$\mathbf{0.00_{\pm 0.00}}$
	SIReN-VAE$_{\text{FC}}$	$10.56_{\pm 0.32}$	$0.50_{\pm 0.50}$
	SIReN-IWAE$_{\text{FC}}$	$10.47_{\pm 0.45}$	$0.33_{\pm 0.75}$
	SIReN-VAE$_{\text{True}}$	$10.60_{\pm 0.05}$	$1.33_{\pm 1.11}$
	SIReN-IWAE$_{\text{True}}$	$\mathbf{10.36_{\pm 0.27}}$	$0.50_{\pm 0.50}$

7.7 Conclusion

This chapter evaluated the proposed SIReN-VAE model on a number of synthetic and real-world datasets. Section 7.2 started off by studying the effect of incorporating GRFs into the prior and inference model of a VAE. The results obtained are not surprising and align with previous work that showed that more complex latent distributions improve overall performance by reducing the approximation gap and mitigating the prior hole problem. This motivates our use of GRFs as a means to inject conditional independence assumptions into a VAE. Sections 7.3 and 7.4 next evaluated SIReN-VAE on a number of datasets with associated BN graphs. We found that the SIReN-VAE models encoding the associated BN's dependency structure (SIReN-VAE$_{\text{True}}$) did not in general match the performance achieved by a SIReN-VAE encoding a fully-connected structure (SIReN-VAE$_{\text{FC}}$). However, we also noted that SIReN-VAE$_{\text{True}}$ suffers more severely from posterior collapse, and that the likelihood of a variable collapsing is linked to its position within the encoded BN. By applying various posterior collapse mitigation techniques, such as warm-up, an importance-weighted objective and low-variance gradient estimates, the performance of SIReN-VAE$_{\text{True}}$ could be significantly improved. This makes SIReN-VAE$_{\text{True}}$ more competitive with SIReN-VAE$_{\text{FC}}$, although it is still not able in general to match the performance of the fully-connected version when enough data is available. Sections 7.5 and 7.6 concluded by exploring additional advantages of using SIReN-VAE$_{\text{True}}$. Section 7.5 showed that SIReN-VAE$_{\text{True}}$ is able to learn individual latent variable representations that in general have a strong correspondence with their true counterparts, which could aid in providing more interpretable models. In Section 7.6 it was shown that SIReN-VAE$_{\text{True}}$ is able to provide better generalization performance than SIReN-VAE$_{\text{FC}}$ and a standard VAE (in terms of log-evidence achieved) in a data-sparse setting.

In summary, encoding a dataset's BN dependency structure does not lead to significantly better generalization performance than simply using a fully-connected structure when enough data is available. The key benefits of SIReN-VAE$_{\text{True}}$ are thus its ability to provide more nuanced interpretability and to allow stable training with good generalization on small training sets. Practitioners who elicit BNs typically omit unknown factors, because they cannot in general use them in current modelling frameworks. SIReN-VAE$_{\text{True}}$ provides a framework in which such latent factors can be considered. Based on the results found in this chapter, we therefore suggest that SIReN-VAE$_{\text{True}}$ might be of use when practitioners know or can hypothesize about the dependency structure of certain latent factors in their domain, especially if they wish to to interpret the model according to the chosen dependency structure. SIReN-VAE$_{\text{True}}$ could also facilitate higher quality data augmentation from limited real-world data.

Chapter 8

Conclusion

Deep generative models have gained widespread popularity in recent years due to their scalability and representation capacity, but unlike probabilistic graphical models, they typically do not incorporate specific domain knowledge. In light of this, this work considered the problem of integrating graphical models and deep generative models as a way to construct interpretable models that are able to learn complex distributions while leveraging prior knowledge about variable interactions. In order to make the scope of our work feasible, we limited our study and refined our research questions to consider only the probabilistic generative modelling frameworks of normalizing flows (NFs) and variational autoencoders (VAEs), as well as the type of domain knowledge that can be encoded using Bayesian networks (BNs). Below we summarize our approach and discuss the key findings which contribute to addressing the problem stated above. We conclude the chapter with an overview of potential avenues for further work.

8.1 Summary & Key Findings

8.1.1 Graphical Residual Flows

Graphical flows, which can be viewed as a confluence of BNs and NFs, have only previously been designed for certain classes of flows. Furthermore, existing graphical flows do not in practice guarantee stable inversion, leading sometimes to exploding inverses. Residual flows on the other hand, provide stable inversion due to the Lipschitz bound placed on each transformation step. We therefore posed the question of whether one could adapt standard residual flows to allow one to encode an arbitrary graphical structure, while still providing competitive density estimation and inference performance *as well as* stable inversion. Our main contribution in line with this first research question is the *graphical residual flow* (GRF)—an extension of residual flows that encodes an arbitrary BN dependency structure between variables of interest through masking of the flow's residual block weights. In this respect, we propose a novel masking scheme extending that in MADE (Germain *et al.*, 2015), that encodes a desired dependency structure by applying binary masks to the neural network weights of the flow's residual blocks, ensuring that each residual block's output corresponding to vertex i in the BN, is only a direct function of the inputs corresponding to i and its parent vertices. Enforcing an (implicit) topological ordering over the variables by encoding a BN dependency structure has the additional benefit of ensuring that the Jacobian matrix of the flow transformation is triangular up to a permutation, allowing efficient and exact computation of the change-of-variables formula.

We evaluated our proposed GRFs by comparing them to existing graphical flow models on a range of synthetic and real-world datasets, that each have an associated true or hypothesized BN structure. We found that GRFs achieved density estimation and inference performance comparable to the best performing existing graphical flows, namely the structured conditional continuous NF (SCCNF) of Weilbach *et al.* (2020) and the graphical NF with a monotonic normalizer (GNF-M) of Wehenkel and Louppe (2021). More specifically, GRFs achieved the best density estimation performance on three out of the five considered datasets, and the best variational inference performance for two out of the three datasets that could be used in this setting. Our proposed masking scheme can be used as a drop-in replacement for GNF and SCCNF and addresses the drawbacks of previous masking approaches: it only requires a single pass through the neural network and does not restrict the hidden layer width. We therefore recommend using our masking scheme as a means to encode domain knowledge in all the graphical flows we considered in this work.

Furthermore, we empirically showed that GRFs additionally provide stable and efficient inversion: all GRF models showed consistent convergence to the correct inverse on all datasets when using a Newton-like fixed-point inversion algorithm. SCCNF and GNF-M, which do not place any specific bounds on the Lipschitz constants of their transformations, could not match the inversion accuracy of GRFs. The small Lipschitz bounds imposed by GRFs do however necessitate flows with a larger number of transformation steps in order to achieve similar performance to SCCNF and GNF-M. Fortunately, the inversion time per step is relatively fast when using a Newton-like fixed-point inversion scheme, and we found that GRFs required less time in general to invert a batch of samples than either SCCNF or GNF-M. This Newton-like fixed-point inversion was also much faster than the traditional Banach fixed-point approach, specifically because we use Lipschitz bounds close to one which slows down the convergence rate of the Banach fixed-point approach. Our proposed GRF aptly fills a gap in the literature on graphical flows, and we conclude that they are a good alternative to existing approaches—especially when one requires a single flow with high representation capacity that is to be used in both transformation directions, for example when a flow trained for density estimation is used to generate new samples as we do in the latent prior of the SIReN-VAE.

8.1.2 Structured Invertible Residual Network VAE

Normalizing flows are bijective transformations and thus require their latent space to have the same dimensionality as the observed space. VAEs, on the other hand, allow the practitioner to specify the number of latent variables. Furthermore, both (graphical) NFs and vanilla VAEs only handle the situation where no assumptions are made about the dependency structure between the latent variables. The second half of our work therefore focused on bridging this gap by specifying a generative model that allows practitioners to encode their knowledge or beliefs about the dependency structure of the latent space. Specifically, we posed the question of whether one could use graphical flows to sensibly integrate information from a BN into a VAE, and in what situations this would be useful. We proposed replacing the Gaussian prior and variational posterior distributions of a standard VAE with graphical NFs as a means to inject domain knowledge about the dependencies between latent variables. Since the flow in the prior is likely to be used in both transformation directions—in the normalizing direction during training and in the generative direction when generating new samples with the VAE—we decided to use our GRFs, as they provide both good modelling capability and stable and accurate inversion. The resulting model was termed the structured invertible residual network (SIReN) VAE.

As with GRFs, we evaluated SIReN-VAE on a range of synthetic and real-world datasets that each have an associated true or hypothesized BN structure. We found that SIReN-VAEs encoding the BN's dependency structure (denoted by SIReN-VAE$_{\text{True}}$) could in general not outperform (in terms of log-evidence), a SIReN-VAE that made no independence assumptions about the latent space and simply encoded a fully-connected structure (denoted by SIReN-VAE$_{\text{FC}}$). On each dataset, SIReN-VAE$_{\text{True}}$, however, markedly outperformed a SIReN-VAE that encoded a random dependency structure with the same number of edges as the true BN. This clearly shows that encoding the wrong independencies is detrimental to model performance. However, when enough data is available, and when one only cares about achieving the best model fit to the observed data, it is safer to make no assumptions about the dependence structure over the variables. We argue rather that the value of SIReN-VAE$_{\text{True}}$ lies in its ability to provide a more interpretable latent space and to provide better modelling performance in data-sparse settings. We found that the latent variables inferred by SIReN-VAE$_{\text{True}}$ had relatively high mutual information with their true counterparts and that the mutual information between different variables of the trained SIReN-VAE$_{\text{True}}$ model (both latent and observed) corresponded to the strength of the mutual information between the variables of the true data. This allows us to better understand the interactions between the latent and observed variables of the model as it corresponds to the encoded BN, which in turn enhances the interpretability of the model. The interpretability afforded by the encoded structure could aid, for example, in more nuanced sample generation, where one could control the conditional distributions of certain observed variables using the latent variables that they are most dependent on. Furthermore, it was found that SIReN-VAE$_{\text{True}}$ noticeably outperformed SIReN-VAE$_{\text{FC}}$ on all datasets when both were trained on only a small number of training instances. We speculate that the increased sparsity of the neural network weights of SIReN-VAE$_{\text{True}}$, in line with the true (or hypothesized) BN independencies, poses an easier learning task than those posed by models that do not include such information, resulting in better generalization to the test set.

Unknown or latent factors are typically omitted by practitioners when eliciting BNs, because they cannot in general use them in current modelling frameworks. SIReN-VAE provides a framework that does allow practitioners to encode their beliefs about the underlying system. Based on the results found in this work, we suggest that SIReN-VAEs might be of use when practitioners know or can hypothesize about the dependency structure of certain latent factors in their domain, especially if they wish to interpret the model according to these variables and their dependency structure, and/or limited data are available.

A key challenge with SIReN-VAE$_{\text{True}}$, is that it is susceptible to posterior collapse. Apart from the known causes of posterior collapse in regular VAEs, we found that the encoded BN structure is an additional contributing factor, with certain variables being much more likely to collapse based on their position within the dependency graph. Since posterior collapse is a common issue with VAE optimization, numerous remedies have been proposed in previous works. We specifically considered various combinations of warm-up, an importance weighted objective and lower-variance gradient estimates. SIReN-VAE$_{\text{True}}$ models implementing one or more of these techniques typically achieved better performance and had fewer collapsed latent variables. Since we would like the learned latent distribution of SIReN-VAE$_{\text{True}}$ to respect the provided BN structure and utilize all the latent dimensions, this is an issue that one should be cognisant of when training these models and, depending on the encoded structure, it is likely that one or more prevention techniques will be needed to obtain good performance.

8.2 Future Work

Based on the work presented in this thesis, we now propose a number of potential avenues for further work, starting with several short-term improvements and leading up to broader concepts and ideas.

To improve and further confirm the findings of this work, the proposed graphical models could be evaluated on a larger collection of datasets, especially real-world datasets, that have an associated BN graph. Since the MEHRA dataset was created by filtering out certain discrete variables, a simple first step could be to create alternative MEHRA datasets by filtering out different values, and investigating whether similar results are obtained. Furthermore, we used a relatively coarse-grained approach to hyperparameter tuning and model selection, often performing grid search over only a small selected number of parameter values and choosing model architectures that can be used for all datasets. Although we believe the chosen approach gives valuable insights, further optimization of hyperparameters and model architectures could be beneficial. Also, since we found that incorporating additional domain knowledge into a SIReN-VAE allowed it to perform better than alternative models in data-sparse settings, it would be interesting to investigate whether the same holds for GRFs.

It is common that BNs are defined over sets of both continuous and discrete variables. Unfortunately, stochastic neural networks cannot backpropagate through discrete samples. This complicates the task of training deep generative models for datasets that include discrete variables (and thus limited the real-world datasets that we considered in this study). It would greatly broaden the number of settings in which GRFs and SIReN-VAEs could be applied if they are able to handle both continuous and discrete variables. One of the most desirable directions for future work is therefore to focus on combining existing gradient estimation approaches for handling discrete variables (Bengio *et al.*, 2013; Tran *et al.*, 2019) with the models proposed in this work.

Graphical residual flows that encode a fully-connected structure, allow one to bypass the Russia Roulette estimator of regular contractive residual flows entirely, by setting the flow up to be effectively autoregressive. This allows exact density calculation and avoids the unpredictable time and memory usage introduced by the Russia Roulette estimator. Future work could investigate how fully-connected GRFs and standard contractive residual flows differ in terms of training time and ultimate performance in light of this.

In order to make the scope of our work feasible, we limited our study to the probabilistic generative modelling frameworks of NFs and VAEs. Another natural extension of our work is therefore to consider incorporating BN dependency information into other types of generative models, for example diffusion models which have recently gained in popularity (Ho *et al.*, 2020; Weilbach *et al.*, 2021). Alternatively, one could consider how to incorporate different forms of domain knowledge. For example, undirected graphical models allow one to encode bidirectional interactions between variables such as is present between the pixels of an image. Such extensions would significantly increase the range of application domains for dependency-aware generative modelling.

Appendices

Appendix A

Empirical Investigations

A.1 Empirical Investigation I: GRF

A.1.1 Datasets & Bayesian Networks

Arithmetic Circuit The synthetic arithmetic circuit BN is the same as that used by Weilbach *et al.* (2020) and Wehenkel and Louppe (2021). See Figure 5.1a in the main text for a diagram of this BN. For density estimation tasks, all variables were observed. For amortized inference tasks, variables z_0 to z_5 were latent, while x_0 and x_1 were observed. This distribution consists of heavy-tailed densities which are linked through non-linear dependencies:

$$\begin{aligned}
z_0 &\sim \text{Laplace}(5, 1)\\
z_1 &\sim \text{Laplace}(-2, 1)\\
z_2 &\sim \mathcal{N}(\tanh(z_0 + z_1 - 2.8), 0.1)\\
z_3 &\sim \mathcal{N}(z_0 \times z_1, 0.1)\\
z_4 &\sim \mathcal{N}(7, 2)\\
z_5 &\sim \mathcal{N}(\tanh(z_3 + z_4), 0.1)\\
x_0 &\sim \mathcal{N}(z_3, 0.1)\\
x_1 &\sim \mathcal{N}(z_5, 0.1).
\end{aligned}$$

Tree This is another synthetic dataset. It is adapted from the model given in Wehenkel and Louppe (2021) to obtain a fully-specified model for which the joint density can be computed in closed form (as needed for the inference tasks). Instead of using the circles 2D dataset from Grathwohl *et al.* (2019) as in Wehenkel and Louppe (2021), the first two variables are sampled from a 2D Gaussian mixture model, GMM$_2$, which consists of two equally weighted components with means at $(1, 1)$ and $(-1, -1)$ and shared covariance matrix $0.2 \times I_2$. As in Wehenkel and Louppe (2021), the second pair of variables is sampled from a GMM with 8 equally weighted components with means at $(0, 1.5)$, $(1, 1)$, $(1.5, 0)$, $(1, -1)$, $(0, -1.5)$, $(-1, -1)$, $(-1.5, 0)$ and $(-1, 1)$ and shared covariance matrix $0.1 \times I_8$. See Figure 5.1c in the main text for a diagram of this BN.

$$\begin{aligned}
z_0, z_1 &\sim \mathrm{GMM}_2 \\
z_2, z_3 &\sim \mathrm{GMM}_8 \\
z_4 &\sim \mathcal{N}(\max(z_0, z_1), 1) \\
z_5 &\sim \mathcal{N}(\min(z_2, z_3), 1) \\
x_0 &\sim \mathcal{N}\left(\frac{1}{2}(\sin(z_4 + z_5) + \cos(z_4 + z_5)), 1\right)
\end{aligned}$$

Protein This real-world dataset consists of 11 observed variables containing information about multiple phosphorylated human proteins (Sachs *et al.*, 2005)—see Figure 5.1b for a diagram of the BN dependency structure. The BN structure encodes the cellular signalling network that is believed to exist between these proteins.

EColi The EColi dataset was generated using a fully-specified BN adapted from the BN repository of Scutari (2022). All vertices are (conditionally) Gaussian, with means given by a linear combination of the parents. Certain leaf vertices were removed from the original BN (namely `nmpc` and `ftsJ`), in order to obtain a BN with fewer latent than observed variables. This was done to make the BN more applicable for the SIReN-VAE setting as well. The following vertices were considered to be latent: `{asnA, atpD, b1191, cspA, cspG, dnaK, eutG, fixC, icdA, lacA, lacY, sucA, yedE, ygcE, yheI}`, while the rest were considered observed. See Figure 5.1e for a diagram of the BN.

MEHRA The Multi-dimensional Environment-Health Risk Analysis dataset (MEHRA) was assembled by Vitolo *et al.* (2018) to help model air pollution, climate and health in English regions using a BN. We only consider the subgraph of the BN obtained during the study corresponding to the continuous variables. As such, only a subset of the original dataset corresponding to a fixed set of discrete variables is used. This reduced dataset consisted of all observations of the continuous variables for the following setting of the observed variables: `{Region=Greater London Authority; Zone=Greater London Urban Area; Type=Traffic Urban; Year=2014; Season=Winter}`.

A.1.2 Model Architectures

Tables A.1 and A.2 provide the model architectures used in the experiments of Sections 5.2 and 5.3. To provide more informative comparisons between the flows, we considered two model sizes for each approach on each task. Smaller models had a maximum capacity of 5000 trainable parameters, and are denoted by a subscript S, e.g., GRF_S (Table A.1). Larger models had a maximum capacity of 15000 parameters, and are denoted by the subscript L (Table A.2). The chosen architectures were obtained by performing a grid search over flow depth and neural network width, using density estimation performance as a metric while adhering to the model size restriction. These same architectures were used for inference in each model size group, except when the number of flow steps was less than the longest shortest path between any observed and latent vertex, as given in Table 5.1 for each BN and as discussed in Section 4.4. In these cases, the number of steps were increased to the appropriate value, while the hidden layer widths were decreased in order to comply with the parameter capacity restrictions.

Table A.1: Small flow model architectures. The hidden layer width and choice of activation function are the settings used for the conditioner neural network for GNF-A and GNF-M, the residual block for GRF, and the main flow transformation neural network for SCCNF, respectively. If the architecture had to be updated for inference tasks, the new number of flow steps and hidden layer width is given in brackets.

BN	Flow	Number of parameters	Number of flow steps*	Hidden layer width	Activation function
Arithmetic circuit	GNF-A_S	4416	4	200	ReLU
	GNF-M_S	4552	2 (3)	50 (30)	ReLU
	SCCNF_S	4621	3	140	Tanh
	GRF_S	4296	8	125	LipMish
Tree	GNF-A_S	4832	4	250	ReLU
	GNF-M_S	4934	2	75	ReLU
	SCCNF_S	4415	3	125	Tanh
	GRF_S	4576	8	125	LipMish
Protein	GNF-A_S	4812	4	175	ReLU
	GNF-M_S	4897	1	150	ReLU
	SCCNF_S	4788	3	150	Tanh
	GRF_S	4788	9	100	LipMish
EColi	GNF-A_S	4852	4	140	ReLU
	GNF-M_S	4664	1 (4)	100 (60)	ReLU
	SCCNF_S	4699	3	250	Tanh
	GRF_S	4347	9	100	LipMish
MEHRA	GNF-A_S	4760	4	150	ReLU
	GNF-M_S	4976	1	125	ReLU
	SCCNF_S	4355	3	150	Tanh
	GRF_S	4797	9	125	LipMish

*For SCCNF, this instead refers to the number of layers in the neural network.

Table A.2: Large model architectures. The hidden layer width and choice of activation function are the settings used for the conditioner neural network for GNF-A and GNF-M, the residual block for GRF, and the main flow transformation neural network for SCCNF, respectively.

BN	Flow	Number of parameters	Number of flow steps*	Hidden layer width	Activation function
Arithmetic circuit	GNF-A_L	14796	9	300	ReLU
	GNF-M_L	14508	4	125	ReLU
	SCCNF$_L$	13312	5	150	Tanh
	GRF$_L$	14518	17	200	LipMish
Tree	GNF-A_L	14490	10	300	ReLU
	GNF-M_L	14208	4	150	ReLU
	SCCNF$_L$	13828	5	140	Tanh
	GRF$_L$	14625	15	215	LipMish
Protein	GNF-A_L	13788	9	225	ReLU
	GNF-M_L	14691	3	150	ReLU
	SCCNF$_L$	14765	5	170	Tanh
	GRF$_L$	1896	28	100	LipMish
EColi	GNF-A_L	14472	9	190	ReLU
	GNF-M_L	13992	3 (4)	100 (90)	ReLU
	SCCNF$_L$	13165	5	300	Tanh
	GRF$_L$	14944	16	200	LipMish
MEHRA	GNF-A_L	14220	9	200	ReLU
	GNF-M_L	14928	3	125	ReLU
	SCCNF$_L$	14214	5	175	Tanh
	GRF$_L$	14382	17	200	LipMish

*For SCCNF, this instead refers to the number of layers in the neural network.

A.1.3 Implementation Details

All models were implemented and trained using the PyTorch (Paszke *et al.*, 2019) machine learning framework. For the GNF (Wehenkel and Louppe, 2021) and SCCNF (Weilbach *et al.*, 2020) models, we used the publicly available implementations provided by the respective authors. These were only adapted in terms of the masking scheme used to encode the desired BN dependency structure. To invert the BNs, we used an implementation of the faithful inversion algorithm of Webb *et al.* (2018), which was obtained directly from the authors.

A.1.4 Additional Results

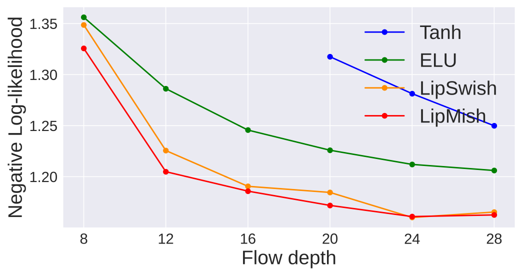
A.1.4.1 Performance of the LipMish Activation Function

Table A.3 gives the performance results of GRF on the different datasets when using either the LipSwish or LipMish activation functions. Apart from the activation function, the model architectures were the same as for GRF$_S$ and GRF$_L$ used in Sections 5.2 and 5.3, and as detailed in Tables A.1 and A.2. We also determined the negative log-likelihoods achieved by GRF as a function of the flow depth, when using different residual block activations. Figure A.1 plots these results for the tanh, ELU, LipSwish and LipMish activation functions on the different datasets. The models used to generate these plots all had residual blocks with one hidden layer of 100 units, and different activation functions and number of flow steps as indicated.

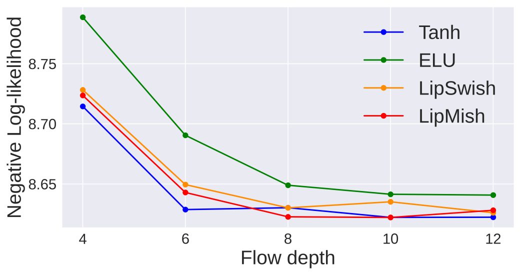
Based on the results in Table A.3, we see that LipMish typically slightly outperformed LipSwish. It achieved the best performance on all datasets except for density estimation on the Arithmetic Circuit dataset and inference on the EColi dataset. Figure A.1 further supports these results, with LipMish performing on par and often better than other activations. These results motivated our use of LipMish instead of alternative activation functions in the experiments of Sections 5.2 and 5.3.

Table A.3: Density estimation and variational inference performance of GRF$_S$ and GRF$_L$ when using either the LipMish or LipSwish activation function. Apart from the activation function, the model architectures are the same as detailed in Tables A.1 and A.2. Lower is better in all cases. Note that the real-world Protein and MEHRA datasets were not used for the inference tasks.

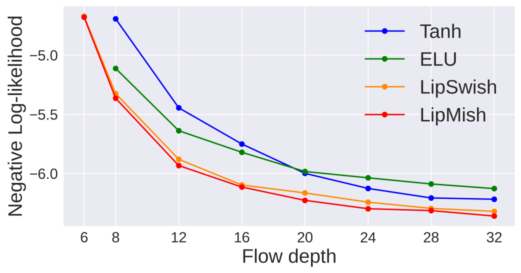
BN	Flow	Density estimation (NLL)		Inference (−ELBO)	
		LipSwish	LipMish	LipSwish	LipMish
Arithmetic Circuit	GRF$_S$	$1.270_{\pm 0.03}$	$\mathbf{1.248}_{\pm \mathbf{0.01}}$	$4.219_{\pm 0.18}$	$\mathbf{4.194}_{\pm \mathbf{0.19}}$
	GRF$_L$	$\mathbf{1.107}_{\pm \mathbf{0.01}}$	$1.110_{\pm 0.01}$	$3.766_{\pm 0.12}$	$\mathbf{3.713}_{\pm \mathbf{0.14}}$
Tree	GRF$_S$	$8.649_{\pm \Delta}$	$\mathbf{8.642}_{\pm \mathbf{0.01}}$	$1.739_{\pm \Delta}$	$\mathbf{1.738}_{\pm \boldsymbol{\Delta}}$
	GRF$_L$	$8.649_{\pm \Delta}$	$\mathbf{8.645}_{\pm \boldsymbol{\Delta}}$	$\mathbf{1.705}_{\pm \boldsymbol{\Delta}}$	$\mathbf{1.705}_{\pm \boldsymbol{\Delta}}$
Protein	GRF$_S$	$-5.230_{\pm 0.02}$	$\mathbf{-5.265}_{\pm \mathbf{0.01}}$	—	—
	GRF$_L$	$-6.035_{\pm 0.07}$	$\mathbf{-6.111}_{\pm \mathbf{0.01}}$	—	—
EColi	GRF$_S$	$40.062_{\pm \Delta}$	$\mathbf{40.059}_{\pm \boldsymbol{\Delta}}$	$34.986_{\pm 0.04}$	$\mathbf{34.964}_{\pm \boldsymbol{\Delta}}$
	GRF$_L$	$40.064_{\pm \Delta}$	$\mathbf{40.062}_{\pm \boldsymbol{\Delta}}$	$\mathbf{34.962}_{\pm \boldsymbol{\Delta}}$	$34.963_{\pm \Delta}$
MEHRA	GRF$_S$	$11.665_{\pm 0.01}$	$\mathbf{11.660}_{\pm \mathbf{0.02}}$	—	—
	GRF$_L$	$11.623_{\pm 0.04}$	$\mathbf{11.612}_{\pm \mathbf{0.03}}$	—	—



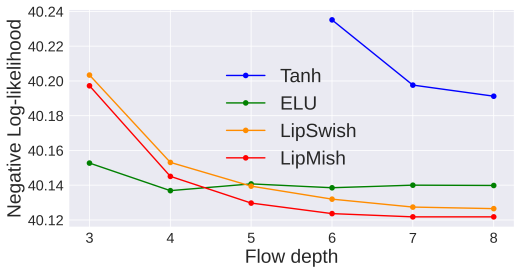
(a) Arithmetic Circuit



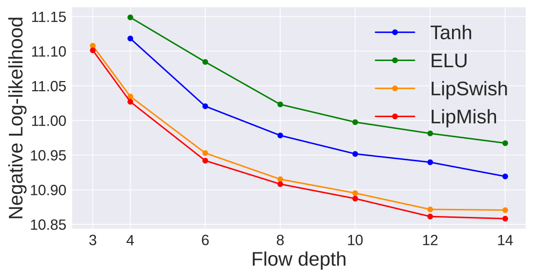
(b) Tree



(c) Protein



(d) EColi



(e) MEHRA

Figure A.1: Negative log-likelihood achieved by GRF models for varying flow depths and different residual block activation functions (tanh, ELU, LipSwish and LipMish) on each of the datasets. Lower is better.

A.1.4.2 Dependency Structure Induced by GRFs

In Section 4.5, we showed that both normalizing and generative GRFs allow additional information to leak from the ancestors of a variable when multiple flow steps are composed. This can result in the final distribution represented by the flow not necessarily respecting the conditional independencies specified by the provided BN structure. If enough steps are composed, additional dependencies between each variable and *all* of its ancestors may be induced, in which case the dependence structure induced by the flow will correspond to the transitive closure of the BN graph $\mathcal{G}$, denoted by $\mathrm{TC}(\mathcal{G})$. However, we argued that the manner in which the BN graph is encoded still provides a strong enough inductive bias to encourage the resulting distribution to respect the provided dependence structure.

Here, we provide preliminary empirical results supporting this claim. If the induced dependence structure fully corresponds to $\mathrm{TC}(\mathcal{G})$, then another BN graph, $\mathcal{G}'$, where $\mathrm{TC}(\mathcal{G}') = \mathrm{TC}(\mathcal{G})$, should provide similar performance. We investigated this for the Tree, Protein and EColi datasets by constructing such alternative graphs and measuring the density estimation performance when encoding either $\mathcal{G}$ or $\mathcal{G}'$. These $\mathcal{G}'$ graphs can be constructed by removing each edge in the original graph that does not change its transitive closure, and adding the same number of new edges that are in the transitive closure of $\mathcal{G}$, but not in $\mathcal{G}$ itself. The type of edge that can be removed without changing the transitive closure, is any edge between a child and one of its parents where this parent is also an ancestor of another one of the child's parents. The above construction ensures that $\mathcal{G}'$ has the same number of edges as $\mathcal{G}$, as well as a matching transitive closure, while specifying a different set of independence assumptions. We also compared using $\mathcal{G}$ and $\mathcal{G}'$ to using a graph $\mathcal{G}'_{\mathrm{min}}$, where *all* possible edges have been removed such that $\mathrm{TC}(\mathcal{G}'_{\mathrm{min}}) = \mathrm{TC}(\mathcal{G})$. Figures A.2 and A.3 show the graphs $\mathcal{G}$, $\mathcal{G}'$ and $\mathcal{G}'_{\mathrm{min}}$ used for the Tree and Protein datasets, respectively. Since the Arithmetic Circuit and MEHRA BNs do not have any such edges that can be used to randomize the dependence structure, we did not consider them here. Table A.4 provides the modelling performance of the small and large GRF models when encoding either $\mathcal{G}$, $\mathcal{G}'$ and $\mathcal{G}'_{\mathrm{min}}$ for density estimation. The architectures of these models were the same as given in Tables A.1 and A.2.

Table A.4: Density estimation performance of GRF_S and GRF_L when encoding either the true BN graph, $\mathcal{G}$, or alternative graphs $\mathcal{G}'$ and $\mathcal{G}'_{\mathrm{min}}$ where $\mathrm{TC}(\mathcal{G}) = \mathrm{TC}(\mathcal{G}') = \mathrm{TC}(\mathcal{G}'_{\mathrm{min}})$ still holds. TC is the transitive closure of a graph. For each BN, we provide the number of edges that were either removed or placed between different child-ancestor pairs to obtain the graphs $\mathcal{G}'_{\mathrm{min}}$ and $\mathcal{G}'$, respectively. Each entry corresponds to the average over the test set for five independent runs, with standard deviation given in the subscript. A standard deviation of less than 0.005 is indicated with Δ. Lower is better in all cases. The best performance in each group is indicated with bold.

BN	#Edges	Flow	Density estimation (NLL)		
			$\mathcal{G}$	$\mathcal{G}'$	$\mathcal{G}'_{\mathrm{min}}$
Protein	9	GRF_S	$\mathbf{-5.26}_{\pm\mathbf{0.01}}$	$-4.87_{\pm 0.04}$	$-4.41_{\pm 0.11}$
		GRF_L	$\mathbf{-6.12}_{\pm\mathbf{0.01}}$	$-5.85_{\pm 0.13}$	$-5.65_{\pm 0.09}$
EColi	12	GRF_S	$\mathbf{40.06}_{\pm\boldsymbol{\Delta}}$	$40.10_{\pm 0.01}$	$40.27_{\pm 0.10}$
		GRF_L	$\mathbf{40.06}_{\pm\boldsymbol{\Delta}}$	$40.07_{\pm\Delta}$	$40.10_{\pm 0.02}$
Tree	2	GRF_S	$\mathbf{8.64}_{\pm\mathbf{0.01}}$	$8.66_{\pm\Delta}$	$8.65_{\pm\Delta}$
		GRF_L	$\mathbf{8.64}_{\pm\boldsymbol{\Delta}}$	$8.66_{\pm\Delta}$	$8.65_{\pm\Delta}$

The results show that models encoding the original dependency structure provided the best modelling performance in all cases. This is most noticeable for the real-world Protein dataset. Even for the simple Gaussian EColi BN and for the Tree BN where only two edges had been changed, the original BN graph still provided the best performance. We suspect therefore that even though information leakage may occur, the direct dependencies encoded by the flow given the BN, still play a more important role in informing the resulting distribution. While the GRF formulation does allow dependencies corresponding to the transitive closure of the encoded BN graph, we expect the forms of the dependencies on variables beyond the underlying graph to be quite constrained, making learning more sophisticated dependencies between vertices not connected in the underlying BN more difficult. Although future work should investigate this issue in more detail, the above results help to motivate our use of GRFs as presented in Chapter 4.

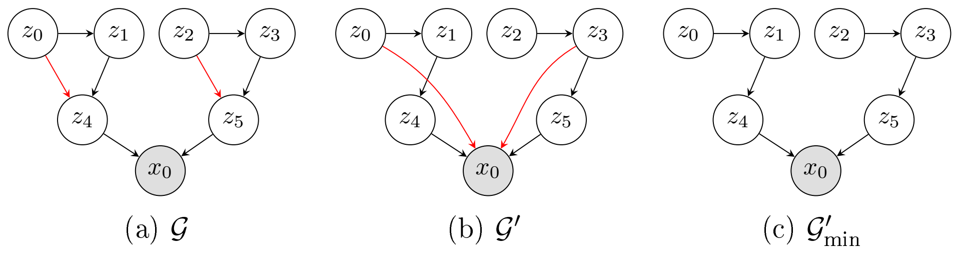


Figure A.2: Illustration of the BN graphs encoded into the GRFs presented in Table A.4 for the Tree dataset. Red edges in (a) the true BN graph were removed and placed between different child-ancestors pairs to create (b) a different graph with the same transitive closure as (a), or removed entirely to obtain (c) a graph with the minimum number of edges that still has the same transitive closure as (a).

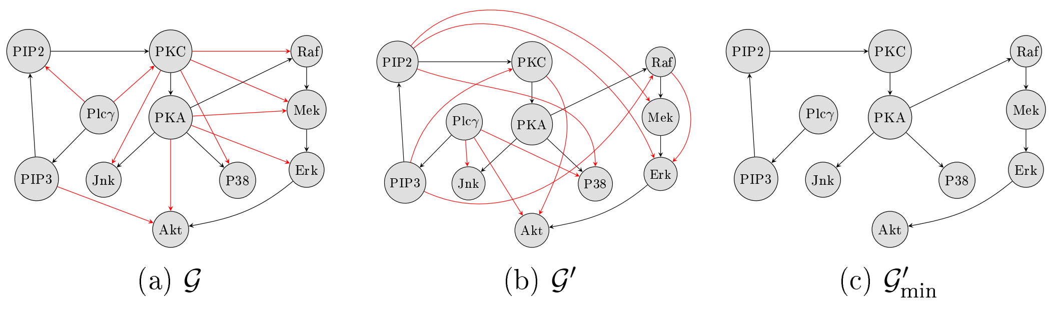
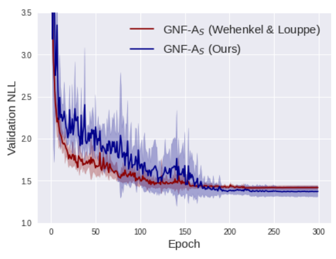


Figure A.3: Illustration of the BN graphs encoded into the GRFs presented in Table A.4 for the Protein dataset.

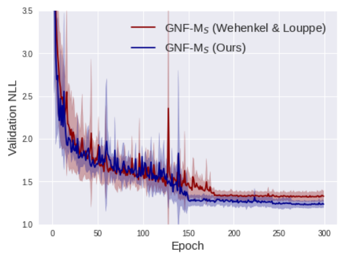
A.1.4.3 Using Different Masking Schemes to Encode Domain Knowledge

In Section 5.2 we compared the density estimation and inference performance of the different graphical flow approaches when using the new masking scheme we presented in Section 4.2 to encode domain knowledge in each of these flows. Below we provide a few examples comparing the performance of GNF-A, GNF-M and SCCNF when using their original masking schemes to encode dependency information versus using our masking scheme. Except for the hidden layer widths, the GNF-A and GNF-M models had the same architectures as given in Table A.1 when using either masking scheme. We reduced the hidden layer widths of the conditioner function neural networks when using the masking scheme of Wehenkel and Louppe (2021), so that all models had approximately the same number of free parameters. When using the masking scheme of Weilbach *et al.* (2020) for SCCNF, the hidden layer widths are required to be the same as the dimensionality of the data. The rest of the continuous flow architecture remained the same as detailed in Table A.1.

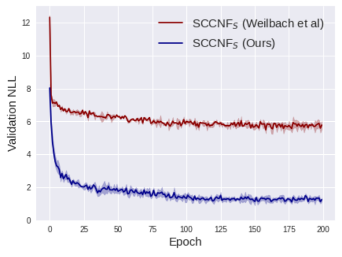
For GNF-A and GNF-M, our approach performed as well or even better than the original masking scheme used by Wehenkel and Louppe (2021). Additionally, it only required a single pass through the neural network. Our masking scheme also significantly outperformed the original scheme used in SCCNF



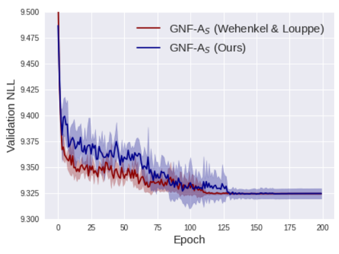
(a) GNF-A$_S$



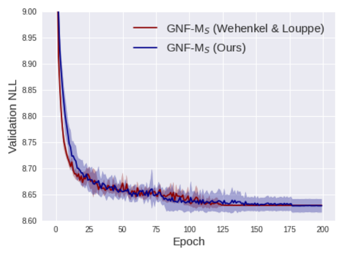
(b) GNF-M$_S$



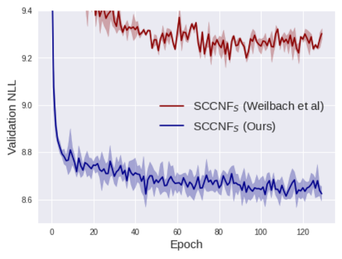
(c) SCCNF$_S$



(d) GNF-A$_S$

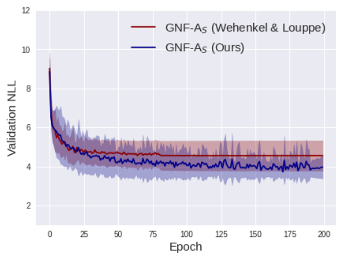


(e) GNF-M$_S$

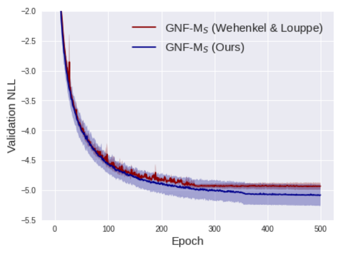


(f) SCCNF$_S$

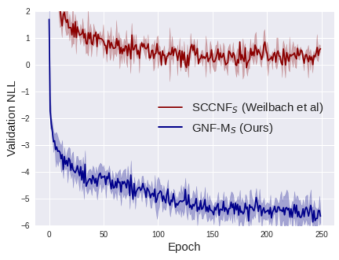
Figure A.4 (Continued on following page): Negative log-likelihood achieved on the Arithmetic Circuit (a)–(c), Tree (d)–(f) and Protein (g)–(i) validation sets during training by GNF-A, GNF-M and SCCNF when using the masking schemes presented in the original works versus using our proposed masking scheme.



(g) GNF-A_S



(h) GNF-M_S



(i) SCCNF$_S$

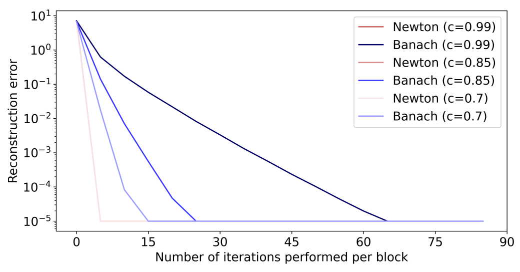
Figure A.4 (Continued): Negative log-likelihood achieved on the Arithmetic Circuit (a)–(c), Tree (d)–(f) and Protein (g)–(i) validation sets during training by GNF-A, GNF-M and SCCNF when using the masking schemes presented in the original works versus using our proposed masking scheme.

when no auxiliary variables were employed. The small fixed size of the hidden layers when using the original masking scheme, is too restrictive and limits the flexibility of this approach.[1]

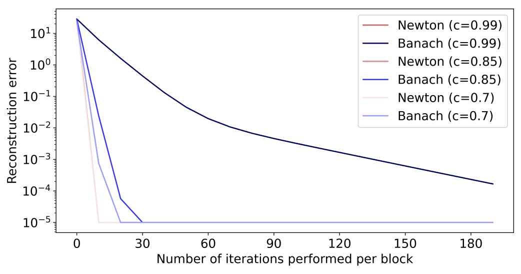
[1]Since we were only interested in the effect of the masking scheme, we did not consider augmenting the input space of SCCNF in this setting. Augmented NFs (Huang *et al.*, 2020; Dupont *et al.*, 2019*b*) provide additional flexibility by allowing the model to learn functions in a higher dimensional space. Augmentation is therefore a general potential improvement that can be applied to the discrete flows as well, independent of the masking scheme.

A.1.4.4 Flow Inversion

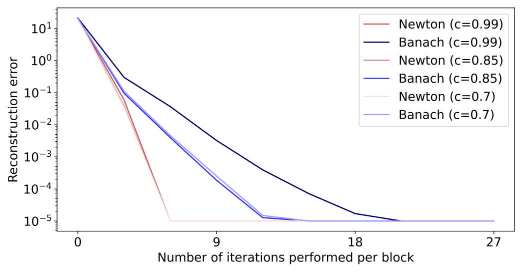
Figure A.5 gives the inversion performance of the Banach and Newton-like fixed-point approaches on the remaining datasets not presented in Section 5.3.



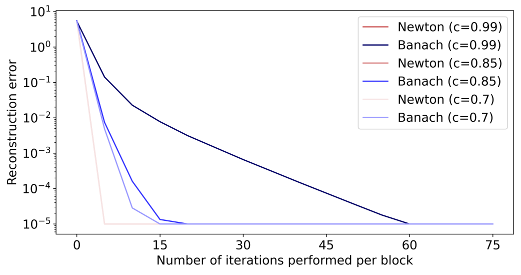
(a) Tree



(b) Protein



(c) EColi



(d) MEHRA

Figure A.5: Using the Newton-like inversion procedure of Equation (4.11) requires far fewer iterations per block to accurately invert a GRF than using the Banach fixed-point approach. The plots show the average reconstruction error (log-scale) for 100 samples from the respective data sets. Note that all the plots for the Newton-like inversion procedure, corresponding to different values of c, overlap in each figure. Also note the change in the range of the x-axis in each figure.

A.2 Empirical Investigation II: SIReN-VAE

A.2.1 Datasets & Bayesian Networks

Since VAEs are typically used to encode information into a lower-dimensional representation, we only considered the datasets from the GRF investigation for which there are fewer latent than observed variables. Thus, in addition to EColi and MEHRA from Appendix A.1.1 we also used the following datasets:

Arithmetic Circuit 2 This Arithmetic Circuit BN is an adaptation of the one described in Appendix A.1.1, where we have added more observed variables. Variables z_0 to z_4 are latent, while x_0 to x_9 are observed. See Figure 7.1b for a diagram of the BN dependency structure. This distribution consists of heavy-tailed densities which are linked through non-linear dependencies:

$$\begin{aligned}
z_0 &\sim \text{Laplace}(5, 1)\\
z_1 &\sim \text{Laplace}(-2, 1)\\
z_2 &\sim \mathcal{N}((z_0 \times z_1)/7.9 - 7, 0.1)\\
z_3 &\sim \mathcal{N}(7, 2)\\
z_4 &\sim \mathcal{N}(\tanh(z_2 + z_3), 0.1)
\end{aligned}
\qquad
\begin{aligned}
x_0 &\sim \mathcal{N}(\tanh(z_0 + z_1 - 2.8), 0.1)\\
x_1 &\sim \mathcal{N}(\tanh(z_1), 1.1)\\
x_2 &\sim \mathcal{N}(\tanh(z_2 + z_3), 0.1)\\
x_3 &\sim \mathcal{N}(z_2 + 8, 0.1)\\
x_4 &\sim \mathcal{N}(\sigma(z_3 - 7), 1.1)\\
x_5 &\sim \mathcal{N}((z_2 \times z_4)/6.1, 0.1)\\
x_6 &\sim \mathcal{N}(z_4, 1.1)\\
x_7 &\sim \mathcal{N}(z_4, 0.1)\\
x_8 &\sim \mathcal{N}(\tanh(z_4), 2.1)\\
x_9 &\sim \mathcal{N}(\sin(z_4), 1.1).
\end{aligned}$$

Arth The Arth dataset was generated using a fully-specified BN from the BN repository of Scutari (2022). All vertices are (conditionally) Gaussian, with means given by a linear combination of the parents. All leaf vertices were considered to be observed, and the rest were taken to be latent. See Figure 7.1d for the corresponding BN graph.

A.2.2 Model Architectures

A.2.2.1 Effect of GRFs on the Latent Distribution

We document here the network architectures used in the experiments of Section 7.2. The models used for the OneHot and MNIST datasets had different architectures, as detailed in Tables A.5 and A.6 below. For each dataset we trained three models: a vanilla VAE (with a factorized Gaussian posterior distribution), a VAE with the posterior augmented by a generative GRF (denoted by VAE+GRF$_g$) and a VAE where both the posterior and prior had been augmented with GRFs (denoted by VAE+GRF$_g$+GRF$_n$). For these experiments we assumed no specific independencies. As such, the GRFs encoded a fully connected structure and no masking was applied to the decoder neural networks (since all observations were dependent on all latent variables). The output of the decoder neural networks of all the models corresponded to the parameters of Bernoulli distributions associated with each of the observed dimensions.

A.2.2.2 Incorporating Graphical Structures

Table A.7 provides the network architectures used in the experiments of Sections 7.3 to 7.6. The models had the same network architectures for all of the datasets, except Arth. For Arth, the hidden layers of the neural networks had a width of 200 units, not 100. The weights of the decoder neural network and all residual blocks were masked according to the given BN structure.

Table A.5: Model architectures used for the OneHot dataset. The input dimension was 5, and all three models used a latent dimension of size 2. All GRFs had 5 transformation steps. For GRF$_g$, the input dimension of the first linear layer of each step was larger to accommodate conditioning on the observation. That is, the first linear layer of each residual block of GRF$_g$ was Linear(2+5, 100), whereas for GRF$_n$, it was Linear(2, 100).

Model		Architecture
VAE	Encoder	Linear(5, 100) → ReLU → Linear(100, 4)
	Decoder	Linear(2, 100) → ReLU → Linear(100, 5) → Sigmoid
VAE+GRF$_g$	Encoder	GRF$_g$ using residual blocks below
	Decoder	Linear(2, 100) → ReLU → Linear(100,5) → Sigmoid
	Residual Block	Linear(2+5, 100) → LipMish → Linear(100, 2)
VAE+GRF$_g$+GRF$_n$	Encoder	GRF$_g$ using residual blocks below
	Decoder	GRF$_n$ → Linear(2, 100) → ReLU → Linear(100, 5) → Sigmoid
	Residual Block	Linear(2(+5), 100) → LipMish → Linear(100, 2)

Table A.6: Model architectures used for the MNIST dataset. The input dimension was $1 \times 28 \times 28$, and all three models used a latent dimension of size 128. All GRFs had 5 transformation steps. For GRF$_g$, the input dimension of the first linear layer of each residual block was larger to accommodate conditioning on the observation.

Model		Architecture
VAE	Encoder	Conv($1 \times 28 \times 28$, $16 \times 14 \times 14$) $\rightarrow$ ReLU $\rightarrow$ Conv($16 \times 14 \times 14$, $32 \times 7 \times 7$) $\rightarrow$ ReLU $\rightarrow$ Linear($32 \times 7 \times 7$, 128)
	Decoder	Linear(128, $32 \times 7 \times 7$) $\rightarrow$ ReLU $\rightarrow$ Deconv($32 \times 7 \times 7$, $16 \times 14 \times 14$)$\rightarrow$ ReLU $\rightarrow$ Deconv($16 \times 14 \times 14$, $1 \times 28 \times 28$) $\rightarrow$ Sigmoid
VAE+GRF$_g$	Encoder	GRF$_g$ using residual blocks below
	Decoder	Linear(128, $32 \times 7 \times 7$) $\rightarrow$ ReLU $\rightarrow$ Deconv($32 \times 7 \times 7$, $16 \times 14 \times 14$)$\rightarrow$ ReLU $\rightarrow$ Deconv($16 \times 14 \times 14$, $1 \times 28 \times 28$) $\rightarrow$ Sigmoid
	Residual Block	Linear(128+784, 500) $\rightarrow$ LipMish $\rightarrow$ Linear(500, 128)
VAE+GRF$_g$+GRF$_n$	Encoder	GRF$_g$ using residual blocks below
	Decoder	GRF$_n$ $\rightarrow$ Linear(128, $32 \times 7 \times 7$) $\rightarrow$ ReLU $\rightarrow$ Deconv($32 \times 7 \times 7$, $16 \times 14 \times 14$)$\rightarrow$ ReLU $\rightarrow$ Deconv($16 \times 14 \times 14$, $1 \times 28 \times 28$) $\rightarrow$ Sigmoid
	Residual Block	Linear(128(+784), 500) $\rightarrow$ LipMish $\rightarrow$ Linear(500, 128)

Table A.7: Model architectures used for the graphical datasets. All GRFs had 5 transformation steps, with each residual block having the same architecture as given below. D indicates the number of observed variables, and K the number of latent variables associated with each dataset's BN. Although not indicated here, the weight matrices of the linear layers in the decoder neural network and residual blocks of SIReN-VAE were masked according to the encoded BN structure. For GRF$_g$, the input dimension of the first linear layer of each residual block was larger to accommodate conditioning on the observation.

Model		Architecture
VAE	Encoder	Linear(D,100) $\rightarrow$ ReLU $\rightarrow$ Linear(100,K$\times$2)
	Decoder	Linear(K,100) $\rightarrow$ ReLU $\rightarrow$ Linear(100,D$\times$2)
SIReN-VAE	Encoder	GRF$_g$
	Decoder	GRF$_n$ $\rightarrow$ Linear(K,100) $\rightarrow$ ReLU $\rightarrow$ Linear(100,D$\times$2)
	Residual Block	Linear(K(+D),100) $\rightarrow$ LipMish $\rightarrow$ Linear(100,K)

Appendix B

Additional Theoretical Background

B.1 Key Mathematical Findings

B.1.1 Leibniz Integral Rule

For an integral of the form:

$$\int_{a(x)}^{b(x)} f(x,t)\,dt \quad \text{where} \quad -\infty < a(x), b(x) < \infty \ , \tag{B.1}$$

The Leibniz integral rule (Protter and Morrey, 2009, Chapter 8) states that the derivative of the integral with respect to x is given by

$$\begin{aligned}\frac{d}{dx}\left(\int_{a(x)}^{b(x)} f(x,t)\,dt\right) = f(x,b(x))\cdot\frac{d}{dx}b(x) - f(x,a(x))\cdot\frac{d}{dx}a(x)\\ + \int_{a(x)}^{b(x)} \frac{\partial}{\partial x} f(x,t)\,dt \ .\end{aligned} \tag{B.2}$$

In the special case where both $a(x)$ and $b(x)$ are constant functions, i.e. $a(x) = a$ and $b(x) = b$, the above expression simplifies to:

$$\frac{d}{dx}\left(\int_{a}^{b} f(x,t)\,dt\right) = \int_{a}^{b} \frac{\partial}{\partial x} f(x,t)\,dt \ . \tag{B.3}$$

B.2 Additional Derivations

B.2.1 Lipschitz Constant of Composition of Functions

Consider the metric space (X, d) and let $f : X \to X$ and $g : X \to X$ be two Lipschitz continuous functions with (smallest) Lipschitz constants equal to k_f and k_g, respectively. We wish to determine the Lipschitz constant of the composition $(g \circ f)$. Per Definition 6, we have that the following holds for any $x, y \in X$:

$$\begin{aligned} d(f(x), f(y)) &\leq k_f \cdot d(x, y) \quad \text{and} \\ d(g(x), g(y)) &\leq k_g \cdot d(x, y) \ . \end{aligned}$$

Based on the above, we have that

$$d(g(f(x)), g(f(y))) \leq k_g \cdot d(f(x), f(y)) \leq k_g \cdot k_f \cdot d(x, y) \ . \tag{B.4}$$

Thus, the Lipschitz constant of $(g \circ f)$ is simply given by $k_g \cdot k_f$.

B.2.2 DReG Derivation

Here, we provide the derivation of the doubly-reparameterized gradient estimator discussed in Section 6.2, viz.[1]

$$\nabla_\phi \, \mathbb{E}_{\mathbf{z}_{1:K}} \left[\log \frac{1}{K} \sum_{i=1}^{K} w_i \right] = \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}} \left[\sum_{i=1}^{K} \left(\frac{w_i}{\sum_j w_j} \right)^2 \nabla_{\mathbf{z}_i} \log w_i \nabla_\phi \, \mathbf{z}_i \right] , \tag{B.5}$$

where $w_i = p_\theta(\mathbf{x}, \mathbf{z}_i)/q_\phi(\mathbf{z}_i|\mathbf{x})$. This exposition closely follows the original derivation given in Tucker *et al.* (2018), and repeats some of the results already provided in Section 6.2 for context.

If q_ϕ is reparameterizable, i.e. $\mathbf{z} = g(\boldsymbol{\epsilon}; \phi)$, where g is some deterministic and differentiable function and $\boldsymbol{\epsilon} \sim p_{\boldsymbol{\epsilon}}$ does not depend on θ or ϕ, then the standard gradient of $\mathcal{L}_K^{\text{IW}}$ with respect to the inference network parameters is given by:

$$\begin{aligned} \nabla_\phi \mathcal{L}_K^{\text{IW}}(\mathbf{x}) &= \nabla_\phi \mathbb{E}_{\mathbf{z}_{1:K} \sim q_\phi(\cdot|x)} \left[\log \frac{1}{K} \sum_{i=1}^{K} w_i \right] && \text{(B.6)} \\ &= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K} \sim p_{\boldsymbol{\epsilon}}} \left[\nabla_\phi \log \frac{1}{K} \sum_{i=1}^{K} w_i \right] && \text{(B.7)} \\ &= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K} \sim p_{\boldsymbol{\epsilon}}} \left[\sum_{i=1}^{K} \frac{1}{\sum_j w_j} \nabla_\phi w_i \right] && \text{(B.8)} \\ &= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K} \sim p_{\boldsymbol{\epsilon}}} \left[\sum_{i=1}^{K} \frac{w_i}{\sum_j w_j} \nabla_\phi \log w_i \right] , && \text{(B.9)} \end{aligned}$$

where we applied the log-derivative (or score function) trick in line (B.9): $\nabla_\phi w_i = w_i \nabla_\phi \log w_i$. The reparameterization trick (Kingma and Welling,

[1]Note that we again abuse notation here and in the rest of this section to simplify the exposition. The path derivative, $\nabla_{\mathbf{z}_i} \log w_i \nabla_\phi \, \mathbf{z}_i$, is a vector-matrix multiplication and is more correctly given by $(\nabla_{\mathbf{z}_i} \log w_i)^T J_g(\phi)$.

2014; Rezende *et al.*, 2014)—see Section 2.4.2—allows us to write the above expectation with respect to $\boldsymbol{\epsilon}_{1:K}$, which allows one to exchange the gradient and expectation operators.

To derive the DReG estimator, we begin by expanding the total derivative of $\log w_i$ with respect to the variational parameters ϕ, into a path derivative and score function component (Roeder *et al.*, 2017):

$$\nabla_\phi^{\mathrm{TD}} \log w_i = \nabla_{\mathbf{z}_i} \log w_i \nabla_\phi \mathbf{z}_i - \nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x}) \ . \tag{B.10}$$

Note that the path derivative measures the dependence of the total derivative on ϕ only through the sample $\mathbf{z}_i = g(\boldsymbol{\epsilon}_i; \phi)$, while the score function measures the dependence on $\log q_\phi(\mathbf{z}_i|\mathbf{x})$ directly and considers $\mathbf{z}_i$ as constant.

We substitute Equation (B.10) back into (B.9):

$$\nabla_\phi \mathcal{L}_K^{\mathrm{IW}}(\mathbf{x}) = \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}\sim p_\epsilon}\left[\sum_{i=1}^K \frac{w_i}{\sum_j w_j}(\nabla_{\mathbf{z}_i} \log w_i \nabla_\phi \mathbf{z}_i - \nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x}))\right] \tag{B.11}$$

For now, we only consider the score function component, $\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})$. First note that:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\epsilon}_{1:K}}\left[\sum_{i=1}^K \frac{w_i}{\sum_j w_j}\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})\right] &= \sum_{i=1}^K \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}}\left[\frac{w_i}{\sum_j w_j}\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})\right] \\ &= \sum_{i=1}^K \mathbb{E}_{\boldsymbol{\epsilon}_{-i}}\mathbb{E}_{\boldsymbol{\epsilon}_i}\left[\frac{w_i}{\sum_j w_j}\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})\right] ,\end{aligned} \tag{B.12}$$

where $\boldsymbol{\epsilon}_{-i} = \boldsymbol{\epsilon}_{1:i-1,i+1:K}$ are all the $\boldsymbol{\epsilon}_{1:K}$ without $\boldsymbol{\epsilon}_i$. It therefore suffices to consider only the inner expectation in (B.12) for each of the K terms.

Since the score function treats $\mathbf{z}_i$ as a constant, we can freely change the random variable that the expectation is taken over from $\boldsymbol{\epsilon}_i$ back to $\mathbf{z}_i$:

$$\mathbb{E}_{\boldsymbol{\epsilon}_i}\left[\frac{w_i}{\sum_j w_j}\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})\right] = \mathbb{E}_{\mathbf{z}_i}\left[\frac{w_i}{\sum_j w_j}\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})\right] \tag{B.13}$$

Next, we note that the right-hand side of Equation (B.13) resembles the REINFORCE gradient term (Williams, 1992, see Section 2.4.2). One can then use the following equivalence between the REINFORCE gradient and the reparameterization trick gradient (for a derivation, see Tucker *et al.* (2018)):

$$\mathbb{E}_{\mathbf{z}\sim q_\phi}\left[f(\mathbf{z})\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x})\right] = \mathbb{E}_{\boldsymbol{\epsilon}\sim p_\epsilon}\left[\nabla_{\mathbf{z}} f(\mathbf{z})\nabla_\phi \mathbf{z}\right] \ , \tag{B.14}$$

where $\mathbf{z} = g(\boldsymbol{\epsilon}; \phi)$. If we let $f(\mathbf{z}) = w_i/\sum_j w_j$, and apply the above identity, i.e. apply the reparameterization trick for a second time, then the inner expectation in (B.12), where we have substituted it with (B.13), becomes:

$$\begin{aligned}
&\mathbb{E}_{\mathbf{z}_i}\left[\frac{w_i}{\sum_j w_j}\nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})\right] \\
&= \mathbb{E}_{\boldsymbol{\epsilon}_i}\left[\nabla_{\mathbf{z}_i}\left(\frac{w_i}{\sum_j w_j}\right)\nabla_\phi\,\mathbf{z}_i\right] \\
&= \mathbb{E}_{\boldsymbol{\epsilon}_i}\left[\left(\frac{1}{\sum_j w_j}-\frac{w_i}{(\sum_j w_j)^2}\right)\nabla_{\mathbf{z}_i} w_i \nabla_\phi\,\mathbf{z}_i\right] \\
&= \mathbb{E}_{\boldsymbol{\epsilon}_i}\left[\left(\frac{w_i}{\sum_j w_j}-\frac{w_i^2}{(\sum_j w_j)^2}\right)\nabla_{\mathbf{z}_i}\log w_i \nabla_\phi\,\mathbf{z}_i\right],
\end{aligned} \tag{B.15}$$

where we have applied the log-derivative trick in the final line. Finally, we substitute (B.15) back into the right-hand side of (B.11):

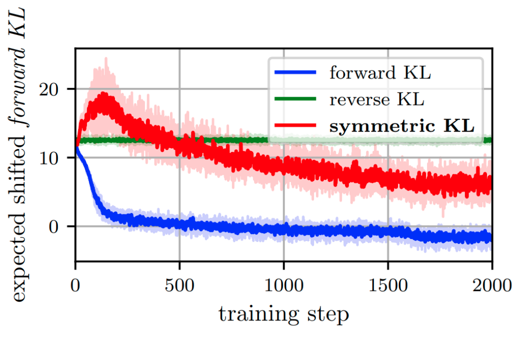
$$\begin{aligned}
&\nabla_\phi \mathcal{L}_K^{\mathrm{IW}}(\mathbf{x}) \\
&= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}}\left[\sum_{i=1}^{K}\frac{w_i}{\sum_j w_j}\left(\nabla_{\mathbf{z}_i}\log w_i \nabla_\phi\,\mathbf{z}_i - \nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{x})\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}}\left[\sum_{i=1}^{K}\left[\cancel{\frac{w_i}{\sum_j w_j}\nabla_{\mathbf{z}_i}\log w_i \nabla_\phi\,\mathbf{z}_i} - \left(\cancel{\frac{w_i}{\sum_j w_j}}-\frac{w_i^2}{(\sum_j w_j)^2}\right)\nabla_{\mathbf{z}_i}\log w_i \nabla_\phi\,\mathbf{z}_i\right]\right] \\
&= \mathbb{E}_{\boldsymbol{\epsilon}_{1:K}}\left[\sum_{i=1}^{K}\left(\frac{w_i}{\sum_j w_j}\right)^2\nabla_{\mathbf{z}_i}\log w_i \nabla_\phi\,\mathbf{z}_i\right].
\end{aligned} \tag{B.16}$$

Appendix C

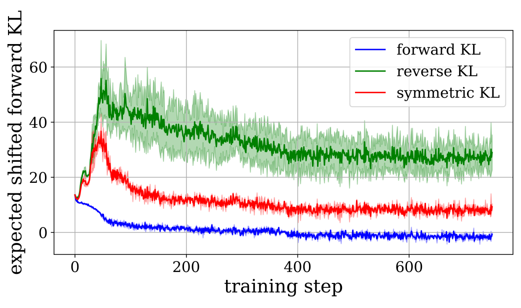
Reproducing SCCNF Article Results

Weilbach *et al.* (2020) propose using the symmetrized KL-divergence, also known as Jeffrey's divergence (Nielsen, 2010), as an objective when training their graphical continuous NFs for the task of variational inference. The symmetrized KL-divergence is a weighted sum of the forward and reverse KL-divergences. They experimentally compare the effect of optimizing the symmetrized KL-divergence instead of either the forward or reverse KL-divergence. The results presented in the original article, for the Arithmetic Circuit BN, are given in Figure C.1a, C.1c and C.1e. Based on the poor performance of the reverse KL-divergence in these experiments, they conclude that it does not provide an adequate learning signal and is therefore not sufficient for training models on BNs such as Arithmetic Circuit.

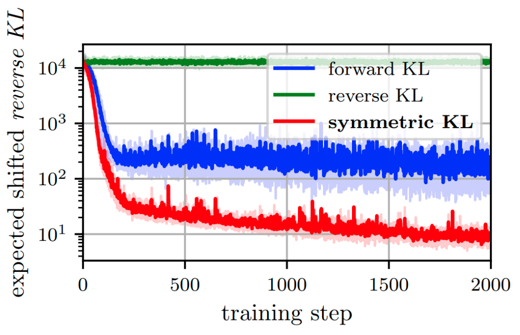
However, we were unable to reproduce these results using the implementation of the technique provided by the authors. Figures C.1b, C.1d and C.1f show the results we obtained when using the same settings as described in the article. It is clear that the reverse KL-divergence objective performs adequately, even outperforming the forward KL-divergence objective in terms of symmetrized KL-divergence obtained. We subsequently confirmed with the authors that their original results were erroneous (C. Weilbach, personal communication, August 21, 2021), and we thus used only the reverse KL-divergence for SCCNF in all our work, in line with the other models used.



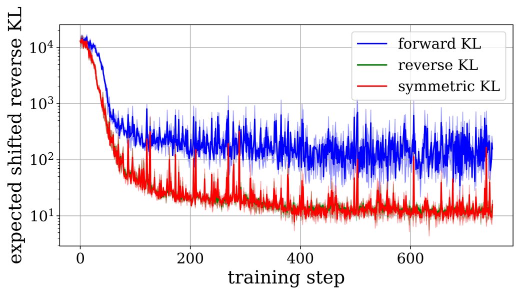
(a) Original



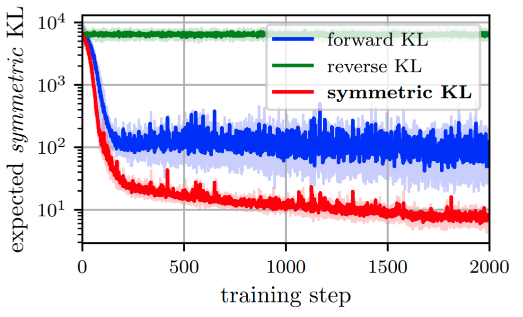
(b) Reproduced



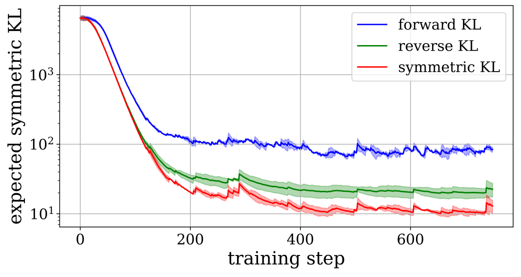
(c) Original



(d) Reproduced



(e) Original



(f) Reproduced

Figure C.1: Reproducing the results presented in Weilbach *et al.* (2020). The top, middle and bottom row show the forward, reverse and symmetric KL-divergence, respectively, when optimizing either the forward, reverse or symmetric KL-divergence for the Arithmetic Circuit BN. Plots (a), (c) and (e) show the original results presented by Weilbach *et al.* (2020), whereas (b), (d) and (f) show the results we were able to produce using the same settings as the original work.

List of References

Ainsworth, S.K., Foti, N.J., Lee, A.K.C. and Fox, E.B. (2018). oi-VAE: Output interpretable VAEs for nonlinear group factor analysis. In: Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 119–128. PMLR.

Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R.A. and Murphy, K. (2018). Fixing a broken ELBO. In: Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 159–168. PMLR.

Ambrogioni, L., Lin, K., Fertig, E., Vikram, S., Hinne, M., Moore, D. and van Gerven, M. (2021*a*). Automatic structured variational inference. In: Banerjee, A. and Fukumizu, K. (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, vol. 130 of *Proceedings of Machine Learning Research*, pp. 676–684. PMLR.

Ambrogioni, L., Silvestri, G. and van Gerven, M. (2021*b*). Automatic variational inference with cascading flows. In: Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, vol. 139 of *Proceedings of Machine Learning Research*, pp. 254–263. PMLR.

Balgi, S., Peña, J.M. and Daoud, A. (2022). Personalized public policy analysis in social sciences using causal-graphical normalizing flows. *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, pp. 11810–11818.

Behrmann, J., Grathwohl, W., Chen, R.T.Q., Duvenaud, D. and Jacobsen, J.-H. (2019). Invertible residual networks. In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 573–582. PMLR.

Behrmann, J., Vicol, P., Wang, K.-C., Grosse, R. and Jacobsen, J.-H. (2021). Understanding and mitigating exploding inverses in invertible neural networks. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pp. 1792–1800. PMLR.

Belghazi, M.I., Baratin, A., Rajeshwar, S., Ozair, S., Bengio, Y., Courville, A. and Hjelm, D. (2018). Mutual information neural estimation. In: Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 531–540. PMLR.

Bengio, Y., LÃ©onard, N. and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bowman, S.R., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R. and Bengio, S. (2016). Generating sentences from a continuous space. In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. Association for Computational Linguistics.

Burda, Y., Grosse, R.B. and Salakhutdinov, R. (2016). Importance weighted autoencoders. In: Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations*.

Chen, R.T.Q., Behrmann, J., Duvenaud, D. and Jacobsen, J.-H. (2019). Residual flows for invertible generative modeling. In: *Advances in Neural Information Processing Systems*, pp. 9916–9926. Curran Associates, Inc.

Chen, R.T.Q., Rubanova, Y., Bettencourt, J. and Duvenaud, D. (2018). Neural ordinary differential equations. In: *Advances in Neural Information Processing Systems*, pp. 6572–6583. Curran Associates, Inc.

Conor Durkan, Artur Bekasov, I.M. and Papamakarios, G. (2019). Cubic-spline flows. In: *ICML Workshop on Invertible Neural Nets and Normalizing Flows*.

Cremer, C., Li, X. and Duvenaud, D. (2018). Inference suboptimality in variational autoencoders. In: *International Conference on Machine Learning*, pp. 1078–1086. PMLR.

Dai, B., Wang, Z. and Wipf, D. (2020). The usual suspects? Reassessing blame for VAE posterior collapse. In: Daumé III, H. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, vol. 119 of *Proceedings of Machine Learning Research*, pp. 2313–2322. PMLR.

Dempster, A.P., Laird, N.M. and Rubin, D.B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–38.

Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142.

Diez, F.J., Mira, J., Iturralde, E. and Zubillaga, S. (1997). DIAVAL, a Bayesian expert system for echocardiography. *Artificial Intelligence in Medicine*, vol. 10, no. 1, pp. 59–73.

Dinh, L., Krueger, D. and Bengio, Y. (2015). NICE: Non-linear independent components estimation. In: *Conference on Learning Representations, Workshop Track Proceedings*.

Dinh, L., Sohl-Dickstein, J. and Bengio, S. (2017). Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*.

Donsker, M.D. and Varadhan, S.R.S. (1983). Asymptotic evaluation of certain Markov process expectations for large time. *Communications on Pure and Applied Mathematics*, vol. 36, no. 2, pp. 183–212.

Dupont, E., Doucet, A. and Teh, Y.W. (2019*a*). Augmented neural ODEs. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc.

Dupont, E., Doucet, A. and Teh, Y.W. (2019*b*). Augmented neural ODEs. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc.

Durkan, C., Bekasov, A., Murray, I. and Papamakarios, G. (2019). Neural spline flows. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc.

Federer, H. (1996). *Geometric Measure Theory*. Springer. ISBN 978-3-540-60656-7.

Fortuin, V., Baranchuk, D., Raetsch, G. and Mandt, S. (2020). GP-VAE: Deep probabilistic time series imputation. In: Chiappa, S. and Calandra, R. (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, vol. 108 of *Proceedings of Machine Learning Research*, pp. 1651–1661. PMLR.

Fraser, A.M. and Swinney, H.L. (1986). Independent coordinates for strange attractors from mutual information. *Physical Review A*, vol. 33, pp. 1134–1140.

Germain, M., Gregor, K., Murray, I. and Larochelle, H. (2015). MADE: Masked autoencoder for distribution estimation. In: *Proceedings of the 32nd International Conference on Machine Learning*, pp. 881–889. PMLR.

Gershman, S. and Goodman, N. (2014). Amortized inference in probabilistic reasoning. In: *Proceedings of the Annual Meeting of the Cognitive Science Society*.

Gouk, H., Frank, E., Pfahringer, B. and Cree, M.J. (2021). Regularisation of neural networks by enforcing Lipschitz continuity. *Machine Learning*, vol. 110, no. 2, pp. 393–416.

Goyal, P., Hu, Z., Liang, X., Wang, C., Xing, E.P. and Mellon, C. (2017). Nonparametric variational auto-encoders for hierarchical representation learning. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5104–5112. IEEE Computer Society. ISSN 2380-7504.

Grathwohl, W., Chen, R.T.Q., Bettencourt, J., Sutskever, I. and Duvenaud, D. (2019). FFJORD: free-form continuous dynamics for scalable reversible generative models. In: *7th International Conference on Learning Representations*.

Hastings, W.K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, vol. 57, no. 1, pp. 97–109. ISSN 00063444.

He, J., Gong, Y., Marino, J., Mori, G. and Lehrmann, A. (2019*a*). Variational autoencoders with jointly optimized latent dependency structure. In: *International Conference on Learning Representations*.

He, J., Spokoyny, D., Neubig, G. and Berg-Kirkpatrick, T. (2019*b*). Lagging inference networks and posterior collapse in variational autoencoders. In: *7th International Conference on Learning Representations*. OpenReview.net.

He, K., Zhang, X., Ren, S. and Sun, J. (2016). Deep residual learning for image recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778. IEEE.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. and Hochreiter, S. (2017). GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6629–6640. Curran Associates Inc., Red Hook, NY, USA.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S. and Lerchner, A. (2017). beta-VAE: Learning basic visual concepts with a constrained variational framework. In: *International Conference on Learning Representations*.

Ho, J., Jain, A. and Abbeel, P. (2020). Denoising diffusion probabilistic models. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851. Curran Associates, Inc.

Hochreiter, S. and Schmidhuber, J. (1994). Simplifying neural nets by discovering flat minima. In: Tesauro, G., Touretzky, D. and Leen, T. (eds.), *Advances in Neural Information Processing Systems*, vol. 7. MIT Press.

Hoffman, M.D. and Johnson, M.J. (2016). ELBO surgery: Yet another way to carve up the variational evidence lower bound. In: *NIPS Workshop: Advances in Approximate Bayesian Inference*.

Huang, C.-W., Dinh, L. and Courville, A. (2020). Augmented normalizing flows: Bridging the gap between generative flows and latent variable models. *arXiv preprint arXiv: 2002.07101*.

Huang, C.-W., Krueger, D., Lacoste, A. and Courville, A. (2018*a*). Neural autoregressive flows. In: *Proceedings of the 35th International Conference on Machine Learning*, pp. 2078–2087. PMLR.

Huang, C.-W., Tan, S., Lacoste, A. and Courville, A.C. (2018*b*). Improving explorability in variational inference with annealed variational objectives. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc.

Huang, C.-W., Touati, A., Dinh, L., Drozdzal, M., Havaei, M., Charlin, L. and Courville, A. (2017). Learnable explicit density for continuous latent space and variational inference. In: *ICML Workshop on Principled Approaches to Deep Learning*.

Hutchinson, M. (1990). A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, vol. 19, no. 2, pp. 433–450.

Jaini, P., Selby, K.A. and Yu, Y. (2019). Sum-of-squares polynomial flow. In: Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, vol. 97 of *Proceedings of Machine Learning Research*, pp. 3009–3018. PMLR.

Johnson, M.J., Duvenaud, D.K., Wiltschko, A., Adams, R.P. and Datta, S.R. (2016). Composing graphical models with neural networks for structured representations and fast inference. In: Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc.

Jordan, M., Ghahramani, Z., Jaakkola, T. and Saul, L. (1999). An introduction to variational methods for graphical models. *Machine Learning*, vol. 37, pp. 183–233.

Khalil, H.K. (2002). *Nonlinear Systems*. Pearson. ISBN 978-0-13-067389-3.

Khemakhem, I., Monti, R., Leech, R. and Hyvarinen, A. (2021). Causal autoregressive flows. In: Banerjee, A. and Fukumizu, K. (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, vol. 130 of *Proceedings of Machine Learning Research*, pp. 3520–3528. PMLR.

Kim, H., Papamakarios, G. and Mnih, A. (2021*a*). The Lipschitz constant of self-attention. In: Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, vol. 139 of *Proceedings of Machine Learning Research*, pp. 5562–5571. PMLR.

Kim, H., Shin, S., Jang, J., Song, K., Joo, W., Kang, W. and Moon, I.-C. (2021*b*). Counterfactual fairness with disentangled causal effect variational autoencoder. *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, pp. 8128–8136.

Kim, M. and Pavlovic, V. (2021). Reducing the amortization gap in variational autoencoders: A Bayesian random function approach. *arXiv preprint arXiv:2102.03151*.

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

Kingma, D.P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 31, pp. 10236–10245. Curran Associates, Inc.

Kingma, D.P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I. and Welling, M. (2016). Improving variational inference with inverse autoregressive flow. In: *Advances in Neural Information Processing Systems*, pp. 4743–4751. Curran Associates, Inc.

Kingma, D.P. and Welling, M. (2014). Auto-encoding variational Bayes. In: *2nd International Conference on Learning Representations*, pp. 14–16.

Kingma, D.P. and Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392. ISSN 1935-8237.

Kleijnen, J.P. and Rubinstein, R.Y. (1996). Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, vol. 88, no. 3, pp. 413–427. ISSN 0377-2217.

Kobyzev, I., Prince, S.J. and Brubaker, M.A. (2021). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 3964–3979.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press. ISBN 978-0-262-01319-3.

Kraskov, A., Stögbauer, H. and Grassberger, P. (2004). Estimating mutual information. *Physical Review E*, vol. 69.

Kreyszig, E. (1989). *Introductory functional analysis with applications*. Wiley. ISBN 0-471-50731-8.

Kullback, S. and Leibler, R.A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86. ISSN 00034851.

Kumar, M., Babaeizadeh, M., Erhan, D., Finn, C., Levine, S., Dinh, L. and Kingma, D. (2019). VideoFlow: A conditional flow-based model for stochastic video generation. In: *ICML Workshop on Invertible Neural Networks and Normalizing Flows*.

Lachapelle, S., Brouillard, P., Deleu, T. and Lacoste-Julien, S. (2020). Gradient-based neural DAG learning. In: *International Conference on Learning Representations*.

Le, T.A., Baydin, A.G. and Wood, F. (2017). Inference compilation and universal probabilistic programming. In: Singh, A. and Zhu, J. (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54 of *Proceedings of Machine Learning Research*, pp. 1338–1348. PMLR.

LeCun, Y., Cortes, C. and Burges, C.J. (2014). The MNIST database of handwritten digits. Data retrieved from `http://yann.lecun.com/exdb/mnist/`.

Louizos, C., Shalit, U., Mooij, J.M., Sontag, D., Zemel, R. and Welling, M. (2017). Causal effect inference with deep latent-variable models. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc.

McCarthy, A.D., Li, X., Gu, J. and Dong, N. (2020). Addressing posterior collapse with mutual information for improved variational neural machine translation. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8512–8525. Association for Computational Linguistics, Online.

Melis, G., György, A. and Blunsom, P. (2022). Mutual information constraints for Monte Carlo objectives to prevent posterior collapse especially in language modelling. *Journal of Machine Learning Research*, vol. 23, no. 75, pp. 1–36.

Mescheder, L., Nowozin, S. and Geiger, A. (2017). Adversarial variational Bayes: Unifying variational autoencoders and generative adversarial networks. In: Precup, D. and Teh, Y.W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 of *Proceedings of Machine Learning Research*, pp. 2391–2400. PMLR.

Minka, T.P. (2001). Expectation propagation for approximate Bayesian inference. In: *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pp. 362–369. Morgan Kaufmann Publishers Inc. ISBN 1558608001.

Misra, D. (2020). Mish: A self regularized non-monotonic activation function. In: *31st British Machine Vision Conference*.

Miyato, T., Kataoka, T., Koyama, M. and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. In: *6th International Conference on Learning Representations*.

Mouton, J. and Kroon, S. (2022*a*). Graphical residual flows. In: *ICLR Workshop on Deep Generative Models for Highly Structured Data*.

Mouton, J. and Kroon, S. (2022*b*). SIReN-VAE: Leveraging flows and amortized inference for Bayesian networks. In: *ICLR Workshop on Deep Generative Models for Highly Structured Data*.

Murphy, K.P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press. ISBN 978-0-262-01802-9.

Murphy, K.P., Weiss, Y. and Jordan, M.I. (1999). Loopy belief propagation for approximate inference: An empirical study. In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 467–475. Morgan Kaufmann Publishers Inc. ISBN 1558606149.

Nielsen, F. (2010). A family of statistical symmetric divergences based on Jensen's inequality. *arXiv preprint arXiv:1009.4004*.

O'Searcoid, M. (2006). *Metric Spaces*. Springer Undergraduate Mathematics Series. Springer London. ISBN 9781846286278.

Paige, B. and Wood, F. (2016). Inference networks for sequential Monte Carlo in graphical models. In: Balcan, M.F. and Weinberger, K.Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48 of *Proceedings of Machine Learning Research*, pp. 3040–3049. PMLR.

Papaconstantinou, C., Theocharous, G. and Mahadevan, S. (1998). An expert system for assigning patients into clinical trials based on Bayesian networks. *Journal of Medical Systems*, vol. 22, no. 3, pp. 189–202.

Papamakarios, G., Nalisnick, E., Rezende, D.J., Mohamed, S. and Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, vol. 22, no. 57, pp. 1–64.

Papamakarios, G., Pavlakou, T. and Murray, I. (2017). Masked autoregressive flow for density estimation. In: *Advances in Neural Information Processing Systems*, pp. 2335–2344. Curran Associates, Inc.

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. (2019). PyTorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc.

Protter, M.H. and Morrey, Jr, C.B. (2009). *Intermediate Calculus*. 2nd edn. Springer. ISBN 978-1-4612-7006-5.

Rainforth, T., Kosiorek, A., Le, T.A., Maddison, C., Igl, M., Wood, F. and Teh, Y.W. (2018). Tighter variational bounds are not necessarily better. In: Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 4277–4285. PMLR.

Ramachandran, P., Zoph, B. and Le, Q.V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Ranganath, R., Tang, L., Charlin, L. and Blei, D. (2015). Deep exponential families. In: Lebanon, G. and Vishwanathan, S.V.N. (eds.), *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, vol. 38 of *Proceedings of Machine Learning Research*, pp. 762–771. PMLR.

Razavi, A., van den Oord, A., Poole, B. and Vinyals, O. (2019). Preventing posterior collapse with delta-VAEs. In: *7th International Conference on Learning Representations*. OpenReview.net.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In: *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1530–1538. PMLR.

Rezende, D.J., Mohamed, S. and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In: Xing, E.P. and Jebara, T. (eds.), *Proceedings of the 31st International Conference on Machine Learning*, vol. 32 of *Proceedings of Machine Learning Research*, pp. 1278–1286. PMLR, Bejing, China.

Roeder, G., Wu, Y. and Duvenaud, D.K. (2017). Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc.

Rosca, M., Lakshminarayanan, B. and Mohamed, S. (2018). Distribution matching in variational inference. *arXiv preprint arXiv:1802.06847*.

Sachs, K., Perez, O., Pe'er, D., Lauffenburger, D.A. and Nolan, G.P. (2005). Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, vol. 308, no. 5721, pp. 523–529.

Samek, W., Montavon, G., Vedaldi, A., Hansen, L.K. and Muller, K.-R. (2019). *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. 1st edn. Springer. ISBN 3030289532.

Scutari, M. (2022). Bayesian network repository.
Available at: `https://www.bnlearn.com/bnrepository/`

Seitzer, M. (2020). pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`. Version 0.2.1.

Shannon, C.E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423.

Sharma, A.K., Kukreja, R., Prasad, R. and Rao, S. (2021). DAGSurv: Directed ayclic graph based survival analysis using deep neural networks. In: Balasubramanian, V.N. and Tsang, I. (eds.), *Proceedings of The 13th Asian Conference on Machine Learning*, vol. 157 of *Proceedings of Machine Learning Research*, pp. 1065–1080. PMLR.

Silvestri, G., Fertig, E., Moore, D. and Ambrogioni, L. (2022). Embedded-model flows: Combining the inductive biases of model-free deep learning and explicit probabilistic modeling. In: *The Tenth International Conference on Learning Representations*. OpenReview.net.

Sohn, K., Lee, H. and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 28. Curran Associates, Inc.

Sønderby, C.K., Raiko, T., Maaløe, L., Sønderby, S.K. and Winther, O. (2016). Ladder variational autoencoders. In: Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc.

Song, Y., Meng, C. and Ermon, S. (2019). MintNet: Building invertible neural networks with masked convolutions. In: *Advances in Neural Information Processing Systems*, pp. 11004–11014. Curran Associates, Inc.

Srivastava, R.K., Greff, K. and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

Stuhlmuller, A., Taylor, J. and Goodman, N. (2013). Learning stochastic inverses. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z. and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, vol. 26. Curran Associates, Inc.

Suzuki, T., Sugiyama, M., Sese, J. and Kanamori, T. (2008). Approximating mutual information by maximum likelihood density ratio estimation. In: Saeys, Y., Liu, H., Inza, I., Wehenkel, L. and van de Pee, Y. (eds.), *Proceedings of the Workshop on New Challenges for Feature Selection in Data Mining and Knowledge Discovery at ECML/PKDD*, vol. 4 of *Proceedings of Machine Learning Research*, pp. 5–20. PMLR.

Tabak, E.G. and Turner, C.V. (2013). A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, vol. 66, no. 2, pp. 145–164.

Takahashi, H., Iwata, T., Yamanaka, Y., Yamada, M. and Yagi, S. (2019). Variational autoencoder with implicit optimal priors. In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI Press. ISBN 978-1-57735-809-1.

Tegnér, G. (2020). mine-pytorch: Mutual Information Neural Estimation. `https://github.com/gtegner/mine-pytorch`.

Tomczak, J. and Welling, M. (2018). VAE with a VampPrior. In: Storkey, A. and Perez-Cruz, F. (eds.), *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, vol. 84 of *Proceedings of Machine Learning Research*, pp. 1214–1223. PMLR.

Tran, D., Vafa, K., Agrawal, K., Dinh, L. and Poole, B. (2019). Discrete flows: Invertible generative models of discrete data. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc.

Tucker, G., Lawson, D., Gu, S. and Maddison, C.J. (2018). Doubly reparameterized gradient estimators for Monte Carlo objectives. In: *1st Symposium on Advances in Approximate Bayesian Inference*, pp. 1–14.

Vahdat, A., Andriyash, E. and Macready, W. (2020 13–18). Undirected graphical models as approximate posteriors. In: III, H.D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, vol. 119 of *Proceedings of Machine Learning Research*, pp. 9680–9689. PMLR.

Vahdat, A. and Kautz, J. (2020). NVAE: A deep hierarchical variational autoencoder. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, vol. 33, pp. 19667–19679. Curran Associates, Inc.

Vaitl, L., Nicoli, K.A., Nakajima, S. and Kessel, P. (2022). Gradients should stay on path: Better estimators of the reverse- and forward KL-divergence for normalizing flows. *Machine Learning: Science and Technology*, vol. 3, no. 4.

van de Meent, J.-W., Paige, B., Yang, H. and Wood, F. (2018). An introduction to probabilistic programming. *Foundations and Trends® in Machine Learning*, pp. 1–209.

van den Berg, R., Hasenclever, L., Tomczak, J. and Welling, M. (2018). Sylvester normalizing flows for variational inference. In: *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*, pp. 393–402. AUAI Press.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K. (2016). WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

van Hulle, M.M. (2005). Edgeworth approximation of multivariate differential entropy. *Neural Computation*, vol. 17, no. 9, pp. 1903–1910. ISSN 0899-7667.

Vitolo, C., Scutari, M., Ghalaieny, M., Tucker, A. and Russell, A. (2018). Modeling air pollution, climate, and health data using Bayesian networks: A case study of the English regions. *Earth and Space Science*, vol. 5, no. 4, pp. 76–88.

Von Mises, R. and Pollaczeck-Geiringer, H. (1929). Praktische Verfahren der Gleichungsauflösung [Practical methods of solving equations]. *Journal of Applied Mathematics and Mechanics*, vol. 9, no. 2, pp. 152–164.

Webb, S., Golinski, A., Zinkov, R., Siddharth, N., Rainforth, T., Teh, Y.W. and Wood, F. (2018). Faithful inversion of generative models for effective amortized inference. In: *Advances in Neural Information Processing Systems*, pp. 3070–3080. Curran Associates, Inc.

Wehenkel, A. and Louppe, G. (2019). Unconstrained monotonic neural networks. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Wehenkel, A. and Louppe, G. (2020). You say normalizing flows I see Bayesian networks. In: *2nd Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models (ICML)*.

Wehenkel, A. and Louppe, G. (2021). Graphical normalizing flows. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pp. 37–45. PMLR.

Weilbach, C., Beronov, B., Wood, F. and Harvey, W. (2020). Structured conditional continuous normalizing flows for efficient amortized inference in graphical models. In: *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, pp. 4441–4451. PMLR.

Weilbach, C., Harvey, W. and Wood, F. (2021). Graphically structured diffusion models. *arXiv preprint arXiv:2210.11633*.

Williams, R.J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, vol. 8, no. 3–4, pp. 229–256. ISSN 0885-6125.

Withers, C.S. and Nadarajah, S. (2010). log det a = tr log a. *International Journal of Mathematical Education in Science and Technology*, vol. 41, no. 8, pp. 1121–1124.

Wyner, A. (1978). A definition of conditional mutual information for arbitrary ensembles. *Information and Control*, vol. 38, no. 1, pp. 51–59. ISSN 0019-9958.

Xu, H., Chen, W., Lai, J., Li, Z., Zhao, Y. and Pei, D. (2019). On the necessity and effectiveness of learning the prior of variational auto-encoder. *arXiv preprint arXiv:1905.13452*.