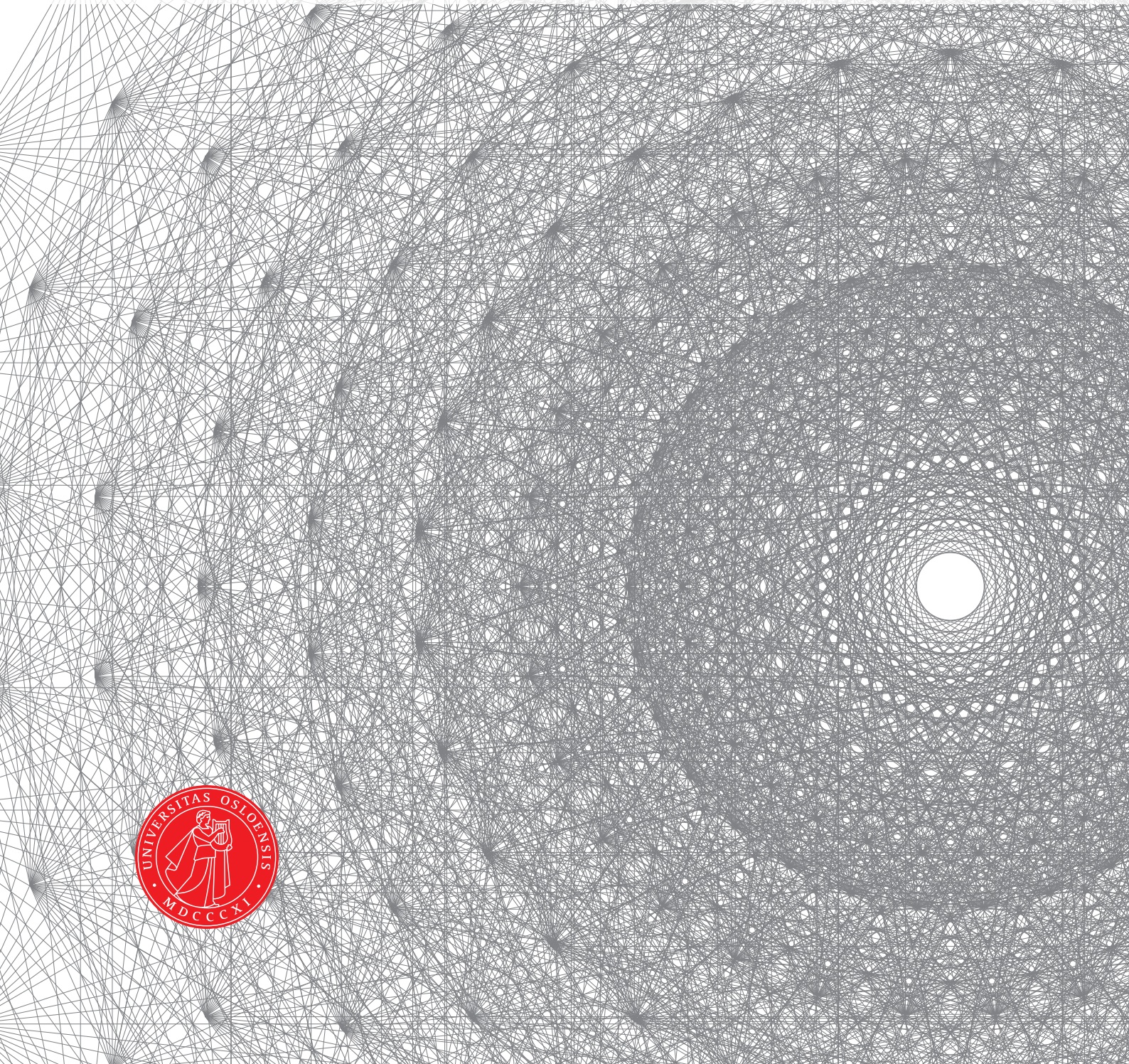
UiO : Department of Mathematics
University of Oslo

Formalising Flows

Introducing Affine Extended Transformations and Flow-Structures

Erik Lien Bolager

Master's Thesis, Autumn 2021



This master's thesis is submitted under the master's programme *Data Science*, with programme option *Statistics and Machine Learning*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

Abstract

Normalizing flows is a promising avenue in both density estimation and variational inference, which promises models that can both generate new samples and evaluate the exact density, both with reasonable computational complexity. In addition, normalizing flows incorporates deep learning, which gives the existence of arbitrarily good approximations of any distribution. This thesis will have two purposes in mind. We first find that normalizing flows contain several components, where each is not as well defined, and which we provide a formalisation of the lay the groundwork for future theoretical work. By formalising, we find both new theoretical results and give an overview of the current literature.

Second purpose is to fill the gap between normalizing flows that are fast computationally and have many attractive properties, but less complex than other flows in the literature. We introduce new normalizing flows that balance the attractive qualities of the less complex flows, but increases the flexibility. We show this through both proving asymptotically behaviour exactly the same as the more complex flows, and then confirm empirically that our proposed flows improve upon the simpler ones, and indeed fills the gap. In addition to this, we find interesting results in terms of variational inference, that shows more complex flows can perform much better in a variational inference setting with increasing dimensions, than what simpler ones shown in the literature can.

Acknowledgements

The work that follows could not have been done without help and support, and many deserve a gratitude for their role, both through my education, this degree in particular, but also through my entire life. I am much obliged.

First and foremost I would like to thank my supervisor, Geir O. Storvik for excellent guidance. You have been very generous with your time and effort, giving me much needed and appreciated feedback on this thesis, but also for the fun discussions we have had. In addition, I express my sincere gratitude towards your trust in me as well, giving me leeway to explore and find my own research topic, while always being there when needed.

I would also express my gratitude towards my parents, Anne and Kai Bolager, for continuous moral support, but also for all the years they spent trying to make a decent man out of me. Hopefully they have achieved it, and that this work can reflect their wisdom and love.

They say it takes a whole village to raise a child, and I am certainly no exception. I wish to express sincere gratitude the environment of students and professors during my years in higher education. In particular for this thesis, I would like to thank Fride Straum, Ingebjørg Sævareid, Marius Havgar, and Åsmund Kvitvang for both support and friendship that have been helpful to take my mind off work, and making this road travelled far from lonely.

Last, but certainly not least, I would like to express gratitude towards Marius Aasan, for his endearing friendship, uncountable many wonderful and interesting academic discussions—which this thesis have befitted of greatly—and unshakeable moral support.

- Erik L. Bolager, 2021

Contents

CHAPTER 1

Introduction

In this thesis, we aim to take a deep dive into normalizing flows and different aspects of them, but first; what are normalizing flows and their use. In both statistics and machine learning, the ability to learn and approximate distributions plays a large role, where there are two tasks to consider. The first task goes under *density estimation*, where the objective is a target distribution $\mathcal{P}$—unknown to us—and a corresponding density p. Having a set of observations $\boldsymbol{x} \sim \mathcal{P}$, the task is then to approximate the distribution, being able to compute the density and preferably generate new observations. Two problems that occur in this field is either to be forced to use too restrictive models, i.e., assumptions that are not necessarily warranted. The other is when the dimension of the distribution grows, and less assumptive models such as kernel density estimations struggle due to the curse of dimensionality (J. Friedman et al. 2001). Normalizing flows provide powerful models that attempts to solve this.

A second task is in the realm of Bayesian statistics, where we have a likelihood $p(\boldsymbol{y} \mid \boldsymbol{z})$, a prior $p(\boldsymbol{z})$, and a set of observations $\boldsymbol{y}$. The task at hand is then to find the posterior distribution—which is in this task the target distribution—$p(\boldsymbol{z} \mid \boldsymbol{y})$, and be able to both generate new observations and evaluate the density of observations. Although there are a few approaches to this task, such as approximating samples through Markov chain Monte Carlo (MCMC), they tend to be too slow for many high dimensional and complex distributions. We concentrate on turning the problem into an optimisation problem, what is known as *variational inference*. In particular, variational inference has been used in many settings with high dimensionality, or difficult posterior distributions where other more complex methods struggle. For instance, the use in machine learning and in particular variational autoencoders (VAE) (Kingma and Welling 2014) have been popular. However, one typically ends up with approximating distributions using quite simple distributions such as independent Gaussian. This is where normalizing flows first saw its introduction.

Normalizing flows can be summarised as follows: start with a distribution which we can easily generate observations from and evaluate its density. We then create a space of bijective functions f, which typically deploy deep learning to create flexible functions f, which then transform the observations from our start distribution. We then find the bijective function f in the corresponding space that output variables, which approximately are from our target distribution, through optimising f's parameters according to some criteria—e.g. maximising log-likelihood.

Relating to our two tasks, normalizing flows can do both, as we can either

find a function that takes observations $\boldsymbol{x}$ and find a function f—which often comprises of a composition of transformations—that transform $\boldsymbol{x}$ through its inverse, to a variable that fits the start distribution. In the second task, we start by generating latent observations $\boldsymbol{z}$ from the start distribution and find f that transforms the variable to fit the posterior distribution. There are, however, a few properties one needs to be aware of when constructing the space of functions f:

- It must be bijective, and hence the introduction of deep learning—in particular neural networks—must be done in a way that preserves bijectivity, as neural networks are, in general, not bijective.
- We ought to be able to compute at least one way—forward or inverse of f—and preferably both fast and efficiently.
- The functions ought to be quite flexible, i.e., the function space ought to be large.
- To evaluate the density of the transformed variable, we must be able to compute the determinant of the Jacobian of f effectively.

Fulfilling these properties is the essence of normalizing flows and its literature.

Problem Description

When we started reading the literature on normalizing flows, we saw that one often were concerned with complete flows/models, while they consisted of several parts that was not completely dependent on each other and deserved to be studied as components, as well as when aggregated to a complete normalizing flow. This also resulted in a lack of terminology for the different parts.

We also noted that there seemed to be a gap of different normalizing flows introduced, between very simple transformations introduced in the early stage, to much more complex later on. We therefore introduce new examples of the components to fill this gap. This also reflect our master project description, which states the two core problems as:

(i) Formalise normalizing flows where the literature lacks,

(ii) Explore new ways to transform data that leans toward the simplistic transformations, yet tries to be as flexible as possible compared to the more complex transformations.

During the formalisation of flows, new definitions spur on new results which we pursue, and our hope is to hopefully make the groundwork that induces more results in the future, with a framework that one can specify and push forward.

However, such formalisation comes at a cost, namely, abstraction. This, as is quite typical when giving rigorous definitions, puts a strain on the reader. We try to alleviate the issue by giving a very concrete example here, which also highlights our approach when formalising it.

Example 1.0.1. The flow we use is one of the first proposed, known as Real NVP (Dinh, Sohl-Dickstein et al. 2017). We assume for this example that the

dimension of the variables we are working with, is $D = 4$. Running through the start by sampling

$$\boldsymbol{z}_0 = \{z_{0,d}\}_{d=1}^4 \sim \mathcal{N}(0, 1),$$

where $\mathcal{N}(0, 1)$ is independent standard Gaussian. We then partition the variable into to parts, $(z_{0,1}, z_{0,2})$ and $(z_{0,3}, z_{0,4})$. We then create a *neural network* (introduced in Section 2.4),

$$\Psi_1 \colon \mathbb{R}^2 \to \mathbb{R}^2.$$

The input of this network is the first subset of the partition $(z_{0,1}, z_{0,2})$, and the output is the parameters that are applied to transform the second subset of the partition $(z_{0,3}, z_{0,4})$. The way it is applied in Real NVP, is through an affine transformation $f_{1,3}$ and $f_{1,4}$, which can be summarised as

$$\begin{aligned}(\boldsymbol{a}_{1,3:4}, \boldsymbol{b}_{1,3:4}) &= \Psi_1(z_{0,1}, z_{0,2}) \\ z_{1,d} = f_{1,d}(z_{0,d}) &= a_{1,d} \cdot z_{1,d} + b_{1,d}, \quad d = 3, 4,\end{aligned}$$

where we constrict the scaling parameter to be strictly positive. We then let $z_{1,1} = z_{0,1}$ and $z_{1,2} = z_{0,2}$. We have then computed the first step in the normalizing flows. Adding another transformation step $t = 2$, but now transforming the first two variables, by again creating a neural network with Ψ_2 in similar fashion. Then

$$\begin{aligned}(\boldsymbol{a}_{2,1:2}, \boldsymbol{b}_{2,1:2}) &= \Psi_2(z_{1,3}, z_{1,4}) \\ z_{2,d} = f_{2,d}(z_{1,d}) &= a_{2,d} \cdot z_{1,d} + b_{2,d}, \quad d = 1, 2.\end{aligned}$$

Applying the identity to $(z_{1,3}, z_{1,4})$. We have then created a small flow, $f(\boldsymbol{z}_0) = \boldsymbol{z}_2$. The density of $\boldsymbol{z}_2$, using the transformation rule, is then

$$q_{\boldsymbol{z}_2}(\boldsymbol{z}_2) = q_{\boldsymbol{z}_0}(f^{-1}(\boldsymbol{z}_2)) \cdot \left| \frac{1}{a_{1,3} \cdot a_{1,4} \cdot a_{2,1} \cdot a_{2,2}} \right|,$$

where $q_{\boldsymbol{z}_0}$ is the density of the independent standard Gaussian. We can then optimise the flow and finding the best f, by changing the functions Ψ_1 and Ψ_2's parameters. To increase the expressiveness of the flow, we can continue transforming $\boldsymbol{z}_2$ in the same manner.

Depending on the size of the two neural networks above, there can be a high number of parameters to optimise—referred to as trainable parameters in this thesis. However, with modern computers and schemes such as backpropogation (LeCun, Bengio et al. 2015), the cost is far from insurmountable in many cases.

This has hopefully given a small insight into flows, and how we can incorporate neural networks, yet preserve the properties we described earlier. Hopefully, the reader will notice that the complete flow have quite a few different components, or choices to be made, which not necessarily require the others to be chosen in similar fashion. In this thesis we have divided the flow into four components/choice to be made:

- The first choice was the partition of $\boldsymbol{z}_0$, or more formally, *which* variables participate in transforming a given variable—which we have defined as *flow-structure* (Section 3.3).

- The second choice is how to compute the variables used in the transformation, e.g. computing (a, b) through neural networks in our example—which we have defined as *conditioner* (Section 3.4).
- The third choice is *how* to transform each variable, which in our example was chosen as $a_{t,d} \cdot z_{t-1,d} + b_{t,d}$—we define this as the *transformation* (Section 3.5).
- The last choice is what to sample from at the start—what is referred to as the *base distribution*.

This will hopefully give the reader some intuition when we start defining the concepts more rigorously.

Overview of Thesis

We give an overview of the different chapters, followed by outlining our biggest contributions.

Chapter 2 introduces some core concept needed, with mostly well known theory. We introduce the tasks at hand, introducing the divergence we use as an optimising criteria, as well as run through variational inference. We continue by introducing neural networks and notation. The last part of Chapter 2 is perhaps a bit less familiar, where we first introduce a type of network well known in the literature, before we generalise it. The generalised version, can be described as neural network, where each dimension of the output y_i can decide which dimensions of the input x_i to be used to calculate y_i. This is essential later on, where neural networks will be used as conditioners.

Chapter 3 is the chapter where we introduce and formalise normalizing flows. We start by giving a very broad definition of flows, before we start defining each component as described earlier. We define first flow-structures, which we then continue investigating, introducing some new theoretical results. The conditioner part is swiftly defined, and a small comment on the different conditioners in the literature, including the ones we define in Chapter 2. Of the different flows introduced in the literature, the component we define that varies the most among the different papers, is the transformation. We therefore take some time to run through the ones existing in the current literature, defining them properly, and discuss the pros and cons of each. After introducing all the concepts, we start putting them together, and define universality, which states what class of probability distributions a given flow are capable of approximating arbitrarily well. We then run through the known universality results in the literature.

Chapter 4 goes on to first recognise gap in what transformations that exist, and introduces several new transformations that tries to be very simple, just as the one we saw in Example 1.0.1, but increase the expressiveness. We then move on to study the conditioner introduced in Chapter 2, and in particular the universal approximator property known from deep learning, which we prove when the conditioner fulfills the universal approximator property. We then move on to proving universality of flows, incorporating our new transformations. We end the chapter by considering how we enforce output of

neural networks to be strictly positive. Considering Example 1.0.1, the $a_{t,d}$ parameters must be strictly positive to enforce inverse and hence make it possible to evaluate the density. As $a_{t,d}$ is calculated by a neural network, the function that enforces positive values is quite important, and can have a lot to say practically, in particular with regard to optimising flows and stability.

Chapter 5 gathers all the theoretical findings and definitions, and send them into practice, by running a few experiments. We start by testing many different flows, giving a overview and some patterns we explore, before choosing a few models to test on more difficult cases, both with density evaluation and variational inference

We end this section by mentioning our contributions. As we are formalising flows, many of the definitions can be seen as a contribution, but we list here the definitions that, as far as we are aware of, does not exist in the literature. When using actual definition, or small rewrite to fit our formalisation, we cite either in the definition, or mention it right above or below. Any theoretical results that is not mine, does not have a proof and the source is cited in the stated result.

In Chapter 2 we have one original contribution, which are conditional neural networks in Section 2.5, which is a generalisation of a existing network MADE. In particular our discussion on Masks and Definition 2.5.2 is an original contribution.

In Chapter 3 starts with a formal definition of flows that is a perhaps not as defined in the literature, but outlines can be found in Kobyzev et al. 2020. In Section 3.3 everything up until the subsection "Autoregressive Structure and Coupling Structure" is original contribution, and the closest we found that had resemblance of our work was in Wehenkel et al. 2020, but which was independently created. In particular, definition on structures Definition 3.3.1, Definition 3.3.2, Definition 3.3.5, and Definition 3.3.6 are original contribution. Corresponding result Proposition 3.3.4 as well. Flow-isomorphic is original contribution with the following Definition 3.3.7, Definition 3.3.9, and Proposition 3.3.10 is our work. The part on triangular structures is an original contribution, namely Definition 3.3.11, Definition 3.3.12, Proposition 3.3.13, Definition 3.3.14, Theorem 3.3.15, Definition 3.3.16, Definition 3.3.17. In Section 3.6 we have two new theoretical results, namely, Lemma 3.6.9 and Lemma 3.6.11, with the corresponding definition Definition 3.6.8 as well. The rest of Chapter 3 is an overview of the current literature, rewriting it into our framework.

In Chapter 4 is a chapter with only original contributions. In Section 4.2 we introduce affine extended transformations, with Definition 4.2.1 and corresponding Proposition 4.2.2 is original. Every transformation following "Piecewise Affine Transformations" is new. In Section 4.3 we give three new results when it comes to CONNs and universality. Namely, Theorem 4.3.2, Proposition 4.3.3, and Theorem 4.3.6. As CONNs are a generalisation of existing MADE, means it is also an original contribution to the theory of MADE. In Section 4.4, we follow similar strategy as to Huang, Krueger et al. 2018, but every result unless otherwise stated in the text, is an original contribution, proving universality for one of our new transformations we proposed in Section 4.2. Finally, in Section 4.5 our definition of a-activation function is new, but is only

to have a name on an existing type of functions. The introduction of Slowplus and Slowabs is an original contribution.

In Chapter 5, unless otherwise specified, the results are computed by us, with our code.

This has hopefully given the reader a scope of what our original contributions are, and what is merely giving a more formal definition and so on. As a last part of the introduction, we give a notation table of the more well known symbols we use.

1.1 Notation

$\mathcal{DI}(\cdot)$	Divergence of distributions, Definition 2.2.1.
$\mathscr{B}(\cdot)$	Borel sigma algebra
$\gamma(\cdot)$	Activation function, Definition 2.4.2.
$\Psi_l(\cdot)$	One hidden layer in a neural network, Definition 2.4.4.
$\Psi(\cdot)$	Fully connected neural network, Definition 2.4.5.
$\mathcal{NN}_{[L,D,\gamma]}$	Space of neural networks with specified properties, Definition 2.4.5.
$\lessapprox$	Used to indicate how many neurons in each hidden layer, Definition 2.4.5
$\mathcal{F}_{width}$	Class of every neural network with arbitrary width.
$\mathcal{F}_{depth}$	Class of every neural network with arbitrary depth.
m_l	Function that informs each node in layer l, which node it can use in previous layer in CONN, Equation (2.10).
M_l	Corresponding binary matrix, mask, to m_l.
$\mathcal{C}_{min}$	Smallest space m_l can map into, Equation (2.11).
$\mathcal{C}$	Largest preferred space m_l can map into, Equation (2.12).
$c(y_d)$	Function that tells which variables in input y_d can use when computed.
Ψ^{CONN}	Conditional neural network, Definition 2.5.2.
$\mathcal{Q}$	Base distribution from Chapter 3 and on.
$\mathcal{D}$	Denotes the set $\{1, 2, \ldots, D\}$.
$\mathcal{D}_\mathcal{T}$	Denotes the set $\{\mathcal{D}_1, \ldots, \mathcal{D}_T\}$.
$\mathcal{T} \otimes \mathcal{D}_\mathcal{T}$	Denotes the set $\{(t, d) \colon t \in \mathcal{T},\, d \in \mathcal{D}_t\}$.
$q_{\boldsymbol{z}_0}$	Density of base distribution.
$q_{\boldsymbol{z}_T}/q_{\boldsymbol{x}}$	Induced density of flows, i.e., the density of $f(\boldsymbol{z}_0)$, Theorem 3.2.4.
$\mathcal{J}_f$	The Jacobian of a function f.
$\mathcal{S}$	Structure, Definition 3.3.1.
$\mathcal{S}_{ext}(t, d)$	The set of influencing variables of (t, d), except for predecessor.
$\mathcal{S}_{int}(t, d)$	The set with only the predecessor of (t, d).
Λ	Function composed of $\Lambda_2 \circ \Lambda_1$, creates forward-local structures, Definition 3.3.12.
π_t	Permutation function used to permute the edges in a structure according to a fixed structure, Definition 3.3.19.
$\mathcal{H}$	Conditioner, Definition 3.4.1, $\mathcal{H}_t$ conditioner constricted to time step t, $\mathcal{H}_{t,d}$ conditioner constricted to time step t and dimension d.
Ψ^+	Neural network transformations using positive weights, Definition 3.5.6.
$\mathcal{NF}$	Class of normalizing flows, Definition 3.6.4.
$\mathbb{R}_*^+$	$\mathbb{R}$ restricted to strictly positive values.
$C(\mathcal{X}, \mathbb{R}^D)$	Space of continuous functions from $\mathcal{X}$ to $\mathbb{R}^D$.
$C(\mathcal{X}, \mathbb{R}^D; c)$	Space of continuous functions from $\mathcal{X}$ to $\mathbb{R}^D$ where each output is constricted to be computed by input given by c, Section 4.3.
$\mathscr{P}$	Space over target distributions.
σ	The Sigmoid function.
g_T	Composition of $\sigma \circ f$, where f is a flow.

CHAPTER 2

Preliminaries

2.1 Introduction

In this chapter, we start by introducing some topics that may be unfamiliar and play an important role in normalizing flows, while also serving as an introduction to the notation. Namely, in Section 2.2 we define what a *divergence* on probability distributions are, and in particular discuss the Kullback-Leibler divergence. In Section 2.3 we introduce *variational inference* (Blei et al. 2017), its pros and cons compared to other sampling schemes such as MCMC, and variational inference using the Kullback-Leibler divergence. Although normalizing flows can be used both in Bayesian inference as well as density estimation, we focus more on the former in this thesis, and therefore focus on variational inference here. Finally, these two sections can easily be skipped for readers who are well versed in the topics at hand. However, we do take the opportunity in this introduction to state the following:

We will for the most part of this text work with continuous distributions, and unless otherwise specified, we let the distributions be implicitly continuous.

Moving onto Section 2.4, we introduce deep learning and neural networks (LeCun, Bengio et al. 2015; Schmidhuber 2015). We define more rigorously *multilayer perceptrons*, and give a recap of the most famous results concerning the flexibility of neural networks. Although it may be familiar to many, we do recommend reading it, as we shall rely heavily on the notation we define.

Finally, in the last section, Section 2.5, we introduce a quintessential tool for normalizing flows, which we also introduce a new concept. We build on neural networks, and introduce a new architecture called *conditional neural networks*. Conditional neural networks are a generalisation of *masked autoencoder for distribution estimation* (MADE) (Germain et al. 2015), which is, as far as we are aware, the first time this is done. The problem put forth for conditional neural networks to solve is the following: create neural networks where each output variable is constrained to a corresponding subset of input variables. That is, any output variable can be computed using only a specific subset of the input variables, where the subset is assumed to be known, and the subset may differ for each output variable. The question then becomes how to enforce neural networks to oblige to the aforementioned restriction. MADE solves this for a subset of the problems, and we generalise this to every problem as specified. The conditional neural networks flexibility is then studied in Chapter 4.

2.2 Divergence

Recall the problems described in the introduction, with density estimation and variational inference. Each model that tries to solve either of the tasks, have a set of parameters/trainable parameters, that is estimated through optimisation. An important question regarding the models underlying distribution is what losses to use to minimise the distance between the true and the approximated distribution. The question is answered by measures of divergence (Bhattacharyya 1946), or simply divergence.

Definition 2.2.1. Let $\mathscr{P}$ be a space of distributions with equal support. A *divergence* is a function $\mathcal{DI}\colon \mathscr{P} \times \mathscr{P} \to \mathbb{R}$ such that $\mathcal{DI}(p, q) \geq 0$ where equality holds if and only if $p = q$, for all $p, q \in \mathscr{P}$. A *dual divergence* is defined as $\mathcal{DI}^*(p, q) = \mathcal{DI}(q, p)$.

Remark 2.2.2. There is a slight abuse of notation, as we are speaking of a space of distributions, but refer to each distribution's density instead, due to the aforementioned assumption in the introduction of this chapter. However, we do need to refer to the actual probability measure as well, and hence let p correspond to a distribution with probability measure μ and equivalently for q and ν.

Although it lacks both symmetry and triangle inequality, and can therefore not be seen as a metric, it can confirm if the distributions are the same, and a space to optimise over.

A myriad of divergences has been proposed, and are actively being deployed or researched. We have added in Appendix A.1, an overview of this and the three major classes one typically sorts divergences in. The Kullback-Leibler divergence is often utilised, and in particular when it comes to the optimisation of models. This thesis is no exception, and we therefore spend some extra time on said divergence.

Kullback-Leibler Divergence

The Kullback-Leibler divergence (KL) (Kullback et al. 1951) has been used in both statistics, statistical learning, and machine learning. Not only in the sense of optimising a model to approximate some distribution, but, for example, as quantifying information gain, that is, how much one learns about a random variable by observing another variable and its value. It has ties to entropy and information theory, which is highlighted when considering Bregman divergences (see Appendix A.1 for more). For our purpose it is a tool to fit models, either in a maximum likelihood fashion or in a Bayesian posterior fashion. We shall denote the divergence as

$$\mathrm{KL}\left(p \mid\mid q\right) = \int_{\mathcal{X}} p(\boldsymbol{x}) \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) d\boldsymbol{x},$$

assuming the corresponding probability measure to the density p is absolutely continuous to the corresponding probability measure of density q, i.e. $\mu \ll \nu$.

Consider the divergence as a loss function optimising a model, where we let q be the induced density of the model and p be the true underlying density. We

have that

$$\mathrm{KL}(p \mid\mid q) = -\int_{\mathcal{X}} p(\boldsymbol{x}) \log q(\boldsymbol{x}) d\boldsymbol{x} + const. = H(p, q) + const.,$$

where $H(p, q)$ is the cross-entropy (and $H(p) = H(p, p)$ is the entropy). We are minimising the cross entropy between our model and the true distribution, as

$$\begin{aligned} H(p, q) &= -\int_{\mathcal{X}} p(\boldsymbol{x}) \log \left(\frac{p(\boldsymbol{x})\, q(\boldsymbol{x})}{p(\boldsymbol{x})} \right) d\boldsymbol{x} \\ &= -\int_{\mathcal{X}} p(\boldsymbol{x}) \log p(\boldsymbol{x}) d\boldsymbol{x} + \int_{\mathcal{X}} p(\boldsymbol{x}) \log \left(\frac{q(\boldsymbol{x})}{p(\boldsymbol{x})} \right) d\boldsymbol{x} \\ &= H(p) - \mathrm{KL}(p \mid\mid q), \end{aligned}$$

means the global minimum of cross entropy is when $\log p(\boldsymbol{x}) = \log q(\boldsymbol{x})$, which agrees with the optimum of KL-divergence (per the definition of divergence). One can typically not compute $H(p, q)$, as we usually do not know $p(\boldsymbol{x})$, and we rather have observations X which we use to estimate $H(p, q)$ through

$$\hat{H}(p, q) = -\frac{1}{|X|} \sum_{\boldsymbol{x} \in X \sim p} \log q(\boldsymbol{x}).$$

When q is the corresponding density of a model with parameters θ, optimising q through minimising $\hat{H}$ gives us the maximum likelihood estimate (MLE) of the models, i.e

$$\hat{\theta}_{MLE} = \arg\min_{\theta} \hat{H}(p, q).$$

Considering the dual divergence and using it as a loss function, we get a different but similar view.

$$\mathrm{KL}(q \mid\mid p) = \int_{\mathcal{X}} q(\boldsymbol{x}) \log q(\boldsymbol{x}) d\boldsymbol{x} - \int_{\mathcal{X}} q(\boldsymbol{x}) \log p(\boldsymbol{x}) d\boldsymbol{x} = -H(q) + H(q, p).$$

If q is an approximation of a posterior, we can estimate the equation above by Monte Carlo (sampling from q),

$$\frac{1}{|X|} \sum_{\boldsymbol{x} \in X \sim q} (\log q(\boldsymbol{x}) - \log p(\boldsymbol{x})). \tag{2.1}$$

It appears that we have more of a compromise between the entropy of the model and the cross entropy between the model and the true distribution. However, ultimately—if the model space includes the true distribution—the global optima is the same for both losses. In some sense, optimising $\mathrm{KL}(p \mid\mid q)$ fits the model to some data, while using $\mathrm{KL}(q \mid\mid p)$ fits the model to the density through changing the parameters and resample X. That is, in the former we have X and want to find the parameters which makes our density fit the data. In the latter, we try to find the parameters θ of q such that the resulting samples from the model fits the true distribution as well. One can think of $\log q(\boldsymbol{x})$ in Equation (2.1) as a bit irrelevant, as the samples will obviously fit the models log-likelihood well, and then be left with a quite similar optimisation where we have swapped the source of data and the log-likelihood we evaluate the data on.

Yet, for any non-zero case, one cannot simply ignore $\log q(\boldsymbol{x})$, as one would end up with an optimisation problem where q is centred at the largest peak of p, with q's mass gathered all at said centre (granted the model can approximate such distribution). This is avoided in Equation (2.1) as $\log q(\boldsymbol{x})$ would become much larger than $\log p(\boldsymbol{x})$, and hence will not be the optimum.

Which of the two divergences to use is typically context based, as one usually has either data from p or can evaluate the likelihood, but not both. If both are the case, one could symmetrize the divergence and simply let the loss function be $\mathrm{KL}(p \,||\, q) + \mathrm{KL}(q \,||\, p)$. Even though it is problem specific, it can be useful to think of the differences between the two versions of KL-divergence. During optimisation, $\mathrm{KL}(p \,||\, q)$ prioritise more that q assign density across p, and rather undershoot the peaks. This can be seen from the fact that, assuming X represents p adequately, low density on a data point is much more detrimental than not maximising log-likelihood for a large chunk of the data, as the former quickly makes the divergence go towards ∞, while the latter results in a finite value of the divergence function. On the other side, $\mathrm{KL}(q \,||\, p)$ prioritise high density on the high density parts of p, and avoid overshooting the density on low density places in p. The reason is that not avoiding the said overshooting leaves $-\log p(\boldsymbol{x})$ rapidly approaching ∞, while overshooting the peaks is not as detrimental. Tad simplified, $\mathrm{KL}(p \,||\, q)$ weights cover the tail area more, while $\mathrm{KL}(q \,||\, p)$ prioritise the mode area. They are opposite in where they overshoot and undershoot p, yet given enough data X, computational power, and that the model space includes p; the models are both capable of approximating p arbitrarily well, using either of the two KL-divergences. That being said, in reality we may experience differences between the two as we rarely acquire optima, and even more crucial is that we are, as mentioned earlier, often forced into using either one due to the availability of data X and density p. In this thesis we are in need of both, one for when density estimation is the problem at hand, and the other when we are applying our models to a variational inference setting (see Section 2.3).

2.3 Variational Inference

When it comes to the task of approximating posterior distributions, as mentioned in the introduction of this thesis, we have some really powerful methods to sample from the posterior, e.g., MCMC. As mentioned already, they do have high computational and time cost for many complex distributions and high dimensional once. A different approach to approximately sample from the posterior is to instead choose a class of distributions which one can easily sample and evaluate the density of, and then find the distribution that best fits the true posterior. Usually finding the best distribution is through choosing the member of the class of distributions which minimises a divergence as we discussed previously, which means we find the distribution that is closest to the true posterior. That is, we turn the problem of generating accurate samples which we in turn can use to estimate the distribution and its quantities (MCMC), to an optimisation problem which gives us a complete distribution that we can generate samples from, evaluate the density of samples, and any statistics we are interested in. The strategy to phrase the problem through optimisation is referred to as *variational inference*.

Definition 2.3.1. Let $\mathscr{Q}$ be a class of continuous distributions, $(\mathcal{Z}, \mathscr{B}(\mathcal{Z}), \mu)$ be a continuous probability distribution with density p, and $\mathcal{DI}$ be a divergence. *Variational inference* is a method that approximate $(\mathcal{Z}, \mathscr{B}(\mathcal{Z}), \mu)$ by optimising

$$q^* = \arg\min_{q \in \mathscr{Q}} \mathcal{DI}(p, q), \tag{2.2}$$

where q^* and q are the densities corresponding to their respective distributions in $\mathscr{Q}$.

Remark 2.3.2. Typically one sees variational inference in the context of approximating Bayesian posterior as we are about to discuss—some also refer to variational Bayesian methods—and where the divergence is the KL-divergence. Yet there are many other combinations of target distribution and divergence that give rise to optimisation. This was also noted by some of the key researchers in this area (Wainwright et al. 2008), in which for something to be referred to as variational inference, one only requires it to be an approximation derived through optimisation. We compromises by defining it through some class of distributions and using a divergence as the criterion for what to optimise for.

Approximating through optimisation instead of sampling will usually give us different results, and there are both pros and cons to this change of strategy. Some of the pros are;

- The result is a fully complete distribution which typically can easily generate new samples if need be, evaluate the density of observations, and—depending on the class of distributions—more interpretable than a collection of samples.
- It is possible to vary the complexity of the distributions we are searching through, and therefore limit the class to allow for faster optimisation when the dimension of the true posterior is large or the distribution is intricate.
- As we are optimising for the closest distribution to the true posterior, means we can set any threshold—although not necessarily trivial to set such thresholds—for what is acceptable for the purpose we have and still obtain a well-defined distribution. Compared to sampling using MCMC, in which the result of stop sampling early can be catastrophic.
- Turning the problem of approximating the true posterior to an optimisation problem allows us to take advantage of a large collection of optimisation schemes such as stochastic gradient descent (SGD).

As one can see, many of the pros were associated with the computational burden and this was indeed the purpose from the start. The cons on the other hand are;

- The class of distributions usually does not contain the distribution we seek to approximate. And to allow for any distribution in the class makes the optimisation harder and at a certain point will be comparable to other sampling techniques.
- This typically means there are also no asymptotically guarantees that the resulting samples we collect through the density q are exact samples from the true distribution.

- There tend to be limitations in terms of what class of distributions to choose from, as it is not straightforward to optimise for a given class of distributions. This is in general for any true distribution, but the true likelihood can also create problems in itself (for example, the posterior distribution of the weights of a neural network). This limitation gives surely an additional weight to the two cons above.

A point worth mentioning, but hard to assign as pro or con, is the fact that variational inference—in theory at least—is a deterministic method contrary to MCMC for example. Stochasticity can add stability and effectiveness to a method, yet it does introduce unpredictability, so there is no clear cut answer to whether such a quality is good or not (we for instance introduce stochasticity into optimisation schemes such as SGD).

There is definitely a place and time for variational inference, but one must be aware of its shortcomings. If the true distribution allows for something akin to MCMC given reasonable computational power and the estimates acquired have a large impact and even detrimental if it is off by much, then there is no doubt that other sampling techniques have the upper edge, and it ought to be preferred over variational inference. However, there are many applications where methods such as MCMC are infeasible and demand an alternative. If not for methods such as variational inference, the options left tends to be point estimates.

Bayesian Inference and KL-divergence

Even though there are many scenarios where variational inference can be applied, there has been quite a surge of interest in regard to approximating true posterior distributions. Some of the explanation for this is simply that their is a need for posterior distributions over models with a large parameter space, instead of opting for point estimates that can often convey false confidence based off limited data/evidence. The divergences of choice have often been the Kullback-Leibler divergence. There happens to be a bound on this divergence that has made it possible to use the divergence, and indeed quite handy to use.

The goal of variational inference under Bayesian inference and KL-divergence can be written as

$$q^* = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}(q(\boldsymbol{z}) \| p(\boldsymbol{z} \mid \boldsymbol{x})) \tag{2.3}$$

$$= \arg\min_{q \in \mathcal{Q}} \int_{\mathcal{Z}} \log\left(\frac{q(\boldsymbol{z})}{p(\boldsymbol{z} \mid \boldsymbol{x})}\right) q(\boldsymbol{z}) d\boldsymbol{z}. \tag{2.4}$$

A problem with optimising using KL-divergence directly is the fact that we are still in need of computing $p(\boldsymbol{x})$, i.e. the evidence. This can easily be seen by rewriting the KL-divergence as follow,

$$\begin{aligned}
\mathrm{KL}(q(\boldsymbol{z}) \| p(\boldsymbol{z} \mid \boldsymbol{x})) &= \int_{\mathcal{Z}} q(\boldsymbol{z}) \log q(\boldsymbol{z}) d\boldsymbol{z} - \int_{\mathcal{Z}} q(\boldsymbol{z}) \log p(\boldsymbol{z} \mid \boldsymbol{x}) d\boldsymbol{z} \\
&= \int_{\mathcal{Z}} q(\boldsymbol{z}) \log q(\boldsymbol{z}) d\boldsymbol{z} - \int_{\mathcal{Z}} q(\boldsymbol{z}) \left(\log p(\boldsymbol{z}, \boldsymbol{x}) - \log p(\boldsymbol{x})\right) d\boldsymbol{z} \\
&= \mathbb{E}_q(\log q(\boldsymbol{z})) - \mathbb{E}_q(\log p(\boldsymbol{z}, \boldsymbol{x})) + \log p(\boldsymbol{x}).
\end{aligned} \tag{2.5}$$

We can therefore not optimise directly with the divergence, but the choice of q does not affect $p(\boldsymbol{x})$, hence we may ignore the term as a constant and optimise without it. As $p(\boldsymbol{x}) \geq 0$, means we are optimising using a lower bound of the KL-divergence as an objective function. The lower bound is known as the *evidence lower bound* and is defined as

$$\begin{aligned}\text{ELBO}(q) &= \mathbb{E}_q(\log p(\boldsymbol{z}, \boldsymbol{x})) - \mathbb{E}_q(\log q(\boldsymbol{z})) \\ &= \mathbb{E}_q(\log p(\boldsymbol{x} \mid \boldsymbol{z})) + \mathbb{E}_q(\log p(\boldsymbol{z})) - \mathbb{E}_q(\log q(\boldsymbol{z})) \\ &= \mathbb{E}_q(\log p(\boldsymbol{x} \mid \boldsymbol{z})) - \text{KL}(q(\boldsymbol{z}) \mid\mid p(\boldsymbol{z})), \end{aligned} \tag{2.6}$$

where we have also flipped the signs of the original lower bound, which means our new objective is

$$q^*(\boldsymbol{z}) = \arg\max_{q \in \mathscr{Q}} \text{ELBO}(q). \tag{2.7}$$

Equation (2.6) reveals the familiar trade-off between the new information represented through the likelihood and our prior belief. The first term wants to have high density around z's that explain the data well, i.e. where the likelihood is high. The second term incentives find a approximation similar to the prior distribution and hence make the divergence small. The more data we acquire, the more concentrated the likelihood is, the more we must tailor our approximation towards the likelihood—and therefore the posterior—due to the aforementioned reasons, putting less weight on the prior. Hence, using the objective function aligns with the Bayesian framework and gives a similar interpretation.

There is clearly no real difference between minimising $\mathbb{E}_q(\log q(\boldsymbol{z})) - \mathbb{E}_q(\log p(\boldsymbol{z}, \boldsymbol{x}))$ or maximising ELBO, but the latter gives us another reason to use the KL-divergence, which is the fact that ELBO is a lower bound of the log evidence, $\log p(\boldsymbol{x})$. This can be seen through

$$\begin{aligned}\log p(\boldsymbol{x}) &= \log \int_{\mathscr{Z}} p(\boldsymbol{z}, \boldsymbol{x}) d\boldsymbol{z} \\ &= \log \int_{\mathscr{Z}} q(\boldsymbol{z}) \frac{p(\boldsymbol{z}, \boldsymbol{x})}{q(\boldsymbol{z})} d\boldsymbol{z} \\ &= \log \mathbb{E}_q \left(\frac{p(\boldsymbol{z}, \boldsymbol{x})}{q(\boldsymbol{z})} \right) \\ &\geq \mathbb{E}_q \left(\log \left(\frac{p(\boldsymbol{z}, \boldsymbol{x})}{q(\boldsymbol{z})} \right) \right) = \text{ELBO}(q), \end{aligned}$$

where the inequality is Jensen's inequality (Jordan et al. 1999). Alternatively, we can see the lower bound of log evidence using Equation (2.5) (Blei et al. 2017). That is,

$$\begin{aligned}\text{KL}(q(\boldsymbol{z}) \mid\mid p(\boldsymbol{z} \mid \boldsymbol{x})) &= \mathbb{E}_q(\log q(\boldsymbol{z})) - \mathbb{E}_q(\log p(\boldsymbol{z}, \boldsymbol{x})) + \log p(\boldsymbol{x}) \\ \log p(\boldsymbol{x}) &= \text{KL}(q(\boldsymbol{z}) \mid\mid p(\boldsymbol{z} \mid \boldsymbol{x})) + \text{ELBO}(q). \end{aligned}$$

As a divergence by definition is greater than or equal to zero implies

$$\log p(\boldsymbol{x}) \geq \text{ELBO}(q).$$

The first approach relates the bound to the Jensen gap, which in our case is

$$\mathbb{E}_q\left(\log\left(\frac{p(\boldsymbol{z},\boldsymbol{x})}{q(\boldsymbol{z})}\right)\right) - \log\left(\mathbb{E}_q\left(\frac{p(\boldsymbol{z},\boldsymbol{x})}{q(\boldsymbol{z})}\right)\right),$$

and how close to zero it is. At the same time, the other approach confirms that our objective in Equation (2.3) gives the tightest bound with respect to the evidence. Regardless, ELBO is a lower bound on the evidence (hence its name *evidence* lower bound), which is why it has also been used for model selection (Cherief-Abdellatif 2019).

Mean-Field Variational Inference

The most common variant of variational inference is the *mean-field* variational family. The core idea behind all the classes of distributions $\mathscr{Q}$ which are included in the mean-field variational family, which is to restrict them to be independent of each other. That is, with $\boldsymbol{z} \in \mathbb{R}^D$,

$$q(\boldsymbol{z}) = \prod_{d=1}^{D} q_d(z_d),\ q \in \mathscr{Q}.$$

This is a quite crude restriction and will in most cases not reflect the true posterior, on the flip side, it allows for fast and cheap optimisation. Its effectiveness computational wise is also reflected by the fact that the earliest variational inference research focused on this particular family (Saul et al. 1996). The independence means that when fixing the value of all other terms than z_d, the ELBO can be rewritten as

$$\begin{aligned}\text{ELBO}(q) &= \mathbb{E}_q\left(\log p(\boldsymbol{z},\boldsymbol{x})\right) - \mathbb{E}_q\left(\log q(\boldsymbol{z})\right) \\ &= \int_{\mathcal{Z}} q(\boldsymbol{z}) \log p(z_d \mid \boldsymbol{x}, \boldsymbol{z}_{-d}) d\boldsymbol{z} - \int_{\mathcal{Z}_d} q_d(z_d) \log q_d(z_d) dz_d + const.\end{aligned}$$

where $-d$ indicates all but dimension d. Maximising w.r.t. to z_d and take the derivative on each side,

$$\begin{aligned}\frac{\partial}{\partial z_d}\left(\int_{\mathcal{Z}_d} q_d(z_d)\log q_d(z_d) dz_d\right) &= \frac{\partial}{\partial z_d}\left(\int_{\mathcal{Z}} q_{\boldsymbol{z}} \log p(z_d \mid \boldsymbol{x}, \boldsymbol{z}_{-d}) d\boldsymbol{z} + const.\right) \\ \log q_d(z_d) &= \int_{\mathcal{Z}_{-d}} q_{-d}(\boldsymbol{z}_{-d}) \log p(z_d \mid \boldsymbol{x}, \boldsymbol{z}_{-d}) d\boldsymbol{z}_{-d}.\end{aligned}$$

Which gives us, taking into account normalisation,

$$q_d^*(z_d) = \exp\left[\mathbb{E}_{-d}\left(\log p(z_d \mid \boldsymbol{x}, \boldsymbol{z}_{-d})\right)\right]. \tag{2.8}$$

Hence, the independence in q bring about a way to optimise, which is to iteratively update each variable following Equation (2.8), repeating the procedure until we meet some stopping criterion based on ELBO. This optimisation scheme is called Coordinate ascent variational inference (CAVI) (Bishop 2006).[1]

[1] There are some striking similarities between CAVI and a sampling technique; Gibbs sampling. Gibbs sampling iteratively samples from its conditional distribution, conditioned on the other dimensions and where the other dimensions are held fixed. This is outside the scope of this text, but for the curious ones, see Blei et al. 2017

In theory, one can use any class of distributions, as long as it includes independence. However, there are practical considerations regarding optimisation and so forth, which limits our choices.

Finally, there are plenty of issues to pertain to regarding mean-field. One is scaling the method to larger and larger data sets, in which lot of work has been done introducing stochastic optimisation (Zhang et al. 2019). The independence requirement is quite strict, and this leads to problems. To illustrate this we can consider a two dimensional independent Gaussian q and highly correlated Gaussian as the true distribution. Using KL-divergence, one can show that the optimal q finds the correct variance to each dimension, but struggle with covering the density induced by the correlation, i.e., it struggles with the tails. Opposite if we reverse the terms in the KL-divergence (sometimes referred to as expectation propagation), as we covers tails, but overshoot the variance (Bishop 2006). Hence, there are considerations to be made on different divergences. These are all important issues, but in this thesis we will expand on the flexibility of the class $\mathscr{Q}$.

2.4 Neural Networks

We now shift gears to introduce neural networks, which is heavily used in the rest of the thesis. Deep learning, and in particular deep neural networks, have sprung up in recent decades as a powerful tool in approximating a diverse set of functions. Although its development can be traced back much further (McCulloch et al. 1943), the advancements, in terms of computational power, made the last half of a century have allowed for high dimensional parameterized models such as neural networks to flourish. As the years have passed, the number of different models that are considered deep learning models have increased (Goodfellow et al. 2014; Hochreiter et al. 1997; LeCun, Boser et al. 1990; Vaswani et al. 2017), and it is hard to give an all-encompassing definition. Vaguely, a common theme is to transform the input through combining linear combinations and simple nonlinearities, which are often elementwise functions.

The different architectures/models can in some sense be seen as adding *inductive biases*. Inductive biases are essentially any assumptions you make and build into the model, typically domain or problem specific. On a macro level it can be assumptions relating to locality in images (LeCun, Boser et al. 1990), the sequential nature of language and relations of words further apart (Hochreiter et al. 1997) etc., or more on a problem specific level. Indeed, one may even see priors as inductive biases. These assumptions are of great value and have made deep learning excel on a diverse field of domains and problems; they can also be seen in the light of the no-free-lunch theorem, namely that inductive biases may help to delegate more weight to specific problems at hand, and decrease performance in other areas (Wolpert et al. 1997). Although it is out of the scope of this thesis to present a comprehensive introduction to different deep learning models, we will define the ones we are in need of. Sometimes it is useful to refer to other models, say convolutional neural networks (CNN), and we assume that the reader is versed in the basics of the most well-known deep learning schemes to follow a comparison or reflection etc.

Thinking generally of neural networks as compositions of linear combinations and rather simple elementwise nonlinearities, one may raise the question of what

the network can approximate. We shall later see the results prompted by this question for specific architectures, but we first take a step back and consider the case of elementwise functions combined with additions. It turns out that one can approximate any continuous function while only allowing elementwise functions and addition. As an answer to part of the thirteenth problem stated by Hilbert 1902, the Kolmogorov-Arnold representation theorem was proven (we use I_D to mean unit cube in D-dimension):

Theorem 2.4.1 (Arnold 2009; Kolmogorov 1957)**.** *We define the continuous functions $\gamma_{d,t}\colon I_1 \to I_1$ and $\gamma_t\colon \mathbb{R} \to \mathbb{R}$. When $D \geq 2$, for any continuous multivariate functions $f\colon I_D \to \mathbb{R}$, there exists functions $\gamma_{d,t}$ and γ_t, such that f can be represented as*

$$f(\boldsymbol{x}) = \sum_{t=1}^{2D+1} \gamma_q \left(\sum_{t=1}^{D} \gamma_{d,t}(x_d) \right).$$

Although the theorem demonstrates the potential expressiveness of linear combinations combined with univariate functions, the functions $\gamma_{d,t}, \gamma_t$ on the right-hand side turns out to be very nonsmooth and complex, hence in reality they are hard to utilise. It turns out that changing the addition to linear combinations, the γ's with simpler nonlinearities, and compensating by increasing the number of summations, leads to neural networks and which have similar results, but now as an approximation of f, as the theorem above (Cybenko 1989).

Construction of Networks

An essential part of a neural network is the activation function, which provides the nonlinearity aspect mentioned previously.[2]

Definition 2.4.2 (Kidger et al. 2020)**.** An *activation function* $\gamma\colon \mathbb{R} \to \mathbb{R}$ is any function that includes the following properties:

- The function is continuous.
- Nonaffine, i.e. $\gamma(x) \neq a \cdot x + b$.
- There exist at least one point in the domain, such that at that point, $\frac{\partial}{\partial x}\gamma(x)$ is continuous and not equal zero.

Remark 2.4.3. There are many ways to define an activation function and many functions have been designed and tested. We restrict ourselves to the requirements above due to the theoretical considerations we discuss later, and this definition is still quite broad and captures all state-of-the-art functions used.

In practice, we often see an activation function that includes some extra properties, which typically are continuously differentiable almost everywhere and monotone. We have included some of the most common functions in Table 2.1.

[2] We refer to a neural network both as a general concept and as a multilayer perceptron—which we define in this section—relying on context to separate the two.

Table 2.1: Some of the most common activation functions used in neural networks.

Name	Function
Sigmoidal logistic	$\gamma(x) = \frac{1}{1+e^{-x}}$
ReLU	$\gamma(x) = \max\{x, 0\}$
Leaky ReLU	$\gamma(x) = \max\{x, 0\} + \alpha \cdot \min\{x, 0\}$
ELU	$\gamma(x) = \max\{x, 0\} + \alpha \cdot \min\{(e^x - 1), 0\}$

Definition 2.4.4. A *hidden layer* is a continuous function $\Psi_l \colon \mathbb{R}^{D_{l-1}} \to \mathbb{R}^{D_l}$ which comprises of an activation function γ, weight matrix $W_l \in \mathbb{R}^{D_l, D_{l-1}}$, and a bias $\boldsymbol{b}_l \in \mathbb{R}^{D_l}$. It is defined as

$$\boldsymbol{x}_l = \Psi_l(\boldsymbol{x}_{l-1}) = \gamma(W_l \boldsymbol{x}_{l-1} + \boldsymbol{b}_l),$$

where the activation function is applied elementwise.

We can now construct a neural network using hidden layers as building blocks.

Definition 2.4.5. An *L-layered neural network* is a continuous function $\Psi \colon \mathbb{R}^{D_0} \to \mathbb{R}^{D_{L+1}}$ of the form

$$\Psi(\boldsymbol{x}_0) = W_{L+1} \left(\Psi_L \circ \Psi_{L-1} \circ \cdots \circ \Psi_1(\boldsymbol{x}_0)\right) + \boldsymbol{b}_{L+1},$$

where $W_{L+1} \in \mathbb{R}^{D_{L+1}, D_L}$ and $\boldsymbol{b}_{L+1} \in \mathbb{R}^{D_{L+1}}$. We denote the space of networks as $\mathcal{NN}_{[L, \boldsymbol{D}, \gamma]}$, where

$$\boldsymbol{D} = D_1 \bowtie D_2 \bowtie \cdots \bowtie D_{L-1} \bowtie D_L,$$

is the dimension of the hidden layers, i.e., D_1 dimension in the first hidden layer and so on. If equal for all layers, we write $\mathcal{NN}_{[L, D, \gamma]}$, where $D \in \mathbb{Z}^+$.

We remind the reader that each $x_{l,d}$, and with slight abuse of notation $\Psi_{l,d}$, for all $d \in D_l$ and $l \in \{1, \ldots, L\}$ is called a neuron or a node. Moreover, the notation $\bowtie$, which granted is not common, is used to represent the number of nodes in each layer. That is, if a network has 256 neurons in the first hidden layer, 512 in the second, and 128 in the third, we can represent it by simply writing $256 \bowtie 512 \bowtie 128$. This comes in handy during empirical testing with different sizes of the network.

We typically find the weights and biases of the neural networks through optimising a loss function $\mathcal{L}$, on a data set $\mathscr{D}$. In an unsupervised setting, e.g. generative models, the data set is $\mathscr{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$. And in a supervised setting, or a generative setting with a latent set for example, the data set is $\mathscr{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$. Importantly, one partition the data set into two, or possibly three parts; a training set, test set, and possibly a validation set if model selection is required (e.g. tuning hyperparameters like amount of regularisation), unless the data set is quite limited, in which case we deploy cross-validation (J. Friedman et al. 2001). Assuming a supervised setting, the weights can then be chosen as

$$\hat{\mathcal{W}} = \arg\min_{\mathcal{W}} \{\mathcal{L}(\boldsymbol{y}_n, \Psi(\boldsymbol{x}_n)) \colon (\boldsymbol{x}_d, \boldsymbol{y}_d) \in \mathscr{D}\},$$

where we let $\mathcal{W} = \{W_1, \boldsymbol{b}_1, \ldots, W_{L+1}, \boldsymbol{b}_{L+1}\}$.

Typically, the connections, i.e., arrows in Figure 2.1, goes from one layer to the next. We can also add connections from a node $\Psi_{l,d}$ to another $\Psi_{l',d'}$, where $l < l' - 1$, in which we refer to it as a *residual connection*.

Definition 2.4.6. A *residual block* is any layer Ψ_l in a neural network Ψ such that every node in Ψ_l contains a residual connection to every node in another layer $\Psi_{l'}$, where $l < l' - 1$. Every residual block is also associated with a weight matrix and bias $W_{res} \in \mathbb{R}^{D_{l'}, D_l}, \boldsymbol{b}_{res} \in \mathbb{R}^{D_{l'}}$, with an intermediate layer $\Psi_{l'}^{res}$ added between l' and $l' + 1$,

$$\Psi_{l'}^{res} = \Psi_{l'} + W_{res}\, \Psi_l + \boldsymbol{b}_{res}.$$

We may have residual blocks where not every node contains a residual connection to another, by simply constricting the relevant weight in W_{res} to be 0. As a final note, it is often assumed that a residual block are done with no affine transformation, i.e. $W_{res} = I$ and $\boldsymbol{b}_{res} = 0$, but this may not be the case and are problem dependent. We have illustrated residual blocks in Figure 2.2, in a vectorized fashion.

Universality

As noted earlier, the combination of linear combinations and univariate functions can represent a large class of functions. Restricting the univariate function, but adding more terms in the summation, can asymptotically—w.r.t. the number of terms in the summations—approximate any real continuous function.

Definition 2.4.7. Let $\mathcal{F}$ be a set of models which approximate a class of functions $\mathcal{G}$. $\mathcal{F}$ is a *universal approximator* for $\mathcal{G}$ if it is dense in $\mathcal{G}$. That is, given any $G \in \mathcal{G}$ and $\epsilon > 0$, there exist a model $F \in \mathcal{F}$ such that

$$\|F(\boldsymbol{x}) - G(\boldsymbol{x})\| < \epsilon,$$

for all $\boldsymbol{x}$ in the given domain and w.r.t. a given norm. Unless otherwise specified, the norm is understood to be the uniform norm $||\cdot||_\infty$.

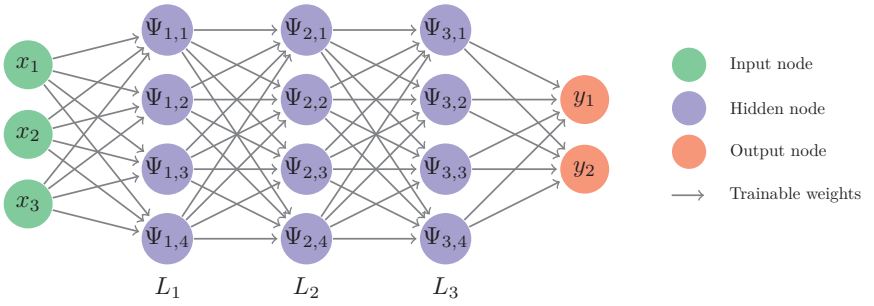


Figure 2.1: A neural network Ψ with $4 \bowtie 4 \bowtie 4$ dimensions of hidden layers. Each hidden node is calculated according to Definition 2.4.4, with the arrows indicating weights and bias omitted from the graph.

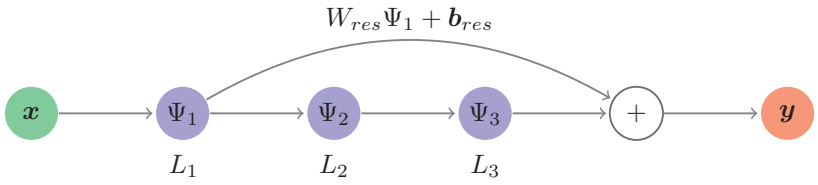


Figure 2.2: Vectorized view of a neural network, with a residual block from hidden layer 1 to the last hidden layer before output.

In a neural network context, we have two different sets of models, one with arbitrary width and bounded depth, and one with arbitrary number of hidden layers and bounded width. That is,

$$\mathcal{F}_{width} = \bigcup_{D=1}^{\infty} \{\mathcal{NN}_{[L,D,\gamma]}\},$$

for some finite L. And equivalently for the arbitrary depth,

$$\mathcal{F}_{depth} = \bigcup_{L=1}^{\infty} \{\mathcal{NN}_{[L,D,\gamma]}\},$$

for some finite number of neurons D in every hidden layer. The original result is with respect to arbitrary width and shows neural networks ability regarding continuous functions.

Theorem 2.4.8 (Cybenko 1989; Hornik 1991; Pinkus 1999)**.** *Let γ be any activation function which is also nonpolynomial. Let $\mathcal{F}_{width}$ have 1 hidden layer, and $\mathcal{X} \subseteq \mathbb{R}^{D_0}$ be compact. Then $\mathcal{F}_{width}$ is a universal approximator for $C(\mathcal{X}, \mathbb{R})$.*

A quite recent result extends the universality to depth and to any activation function (per what we defined). There are other results regarding arbitrary depth (Lu et al. 2017), but we deploy the following result.

Theorem 2.4.9 (Kidger et al. 2020)**.** *Let $\mathcal{F}_{depth}$ have $D_0 + D_{L+1} + 2$ neurons in each hidden layer and $\mathcal{X} \subseteq \mathbb{R}^{D_0}$ be compact. $\mathcal{F}_{depth}$ is then a universal approximator for $C\left(\mathcal{X}, \mathbb{R}^{D_{L+1}}\right)$.*

Hence, we can approximate any continuous function arbitrarily well, as long as we can extend our network with more layers. Additionally, if we constrain our activation function to be ReLU, we can drop the two extra neurons in the theorem above, but prefer the result above as it is more general (Hanin et al. 2017). However, we must stress to interpret the results with some caution. That is to say, this is by no means a guarantee for neural networks trained with finite width and depth, and more importantly, with finite data and any particular optimisation scheme. This does not mean the results are useless by any stretch of the imagination. We reflect more on universality and its usefulness in Section 3.6.

2.5 Conditional Neural Network

In this section we start by introducing a well known type of neural network, called the *masked autoencoder for distribution estimation (MADE)* (Germain et al. 2015). We then generalise the idea of MADE, and introduces a new type of neural network called *conditional neural network*. An important network architecture which is heavily relied on in the following chapters.

MADE started out as a variant of an *autoencoder*.

Definition 2.5.1. An *autoencoder* is a neural network consisting of a *encoder* $\Psi_{enc}\colon \mathcal{X} \to \mathcal{Z}$ and *decoder* $\Psi_{dec}\colon \mathcal{Z} \to \mathcal{X}$, with $\mathcal{Z}$ being the latent representation of $\mathcal{X}$. An autoencoder is the function

$$\Psi(\boldsymbol{x}) = \Psi_{dec} \circ \Psi_{enc}(\boldsymbol{x}).$$

The loss used for autoencoders is typically the reconstruction loss, i.e

$$\mathcal{L}(\boldsymbol{x}) = ||\boldsymbol{x} - \Psi(\boldsymbol{x})||_2^2.$$

Focusing on vectors $\boldsymbol{x} \in \mathcal{X}$ where each component is binary, either one or zero, and adding a logistic function at the output of the decoder, one may consider the cross-entropy loss instead of reconstruction loss. The new loss is then given by

$$\mathcal{L}(\boldsymbol{x}) = \sum_{d=1}^{D_0} -x_d \log \hat{x}_d - (1 - x_d) \log(1 - \hat{x}_d), \tag{2.9}$$

where $\hat{\boldsymbol{x}} = \Psi(\boldsymbol{x})$. However, the expression above does not necessarily constitute a log-likelihood, which is the problem MADE is made for. This is to create a log-likelihood estimated for the binary case, and hence turn the autoencoder into a generative model (one can sample knowing the probability for when it is one and when it is zero). They enforce the loss function to be negative log-likelihood by rewriting Equation (2.9)—applying the autoregressive property—by simply changing $\hat{x}_d = \Psi(x_d \mid \boldsymbol{x}_{<d})$. That is, the autoencoder computes the value for x_d using only itself and dimensions before it. Which means we can interpret the output value as $\hat{x}_d = p(x_d = 1 \mid \boldsymbol{x}_{<d})$, and the loss function is then equal to the negative log-likelihood. Hence, the question that needs an answer is how to construct an autoencoder, or more generally a neural network, which ensures the output follows the autoregressive property.

We generalise the concept given above from the binary autoencoder case used in the original paper, to any neural network with any outputs which wish to compute each output value y_d based on specific dimensions of the input (i.e. not only for the autoregressive case). Letting $\Psi_{l,d}$ represent the node in layer l and component d in the network, we define a function for each hidden layer and output layer $l \in \{1, \ldots, L+1\}$,

$$m_l\colon \{1, \ldots, D_l\} \to \mathcal{P}\left(\{1, \ldots, D_0\}\right), \tag{2.10}$$

where $\mathcal{P}(S)$ is the power set of a set S, i.e., the set with all possible subsets of S. We also add a mapping to input layer $m_0(d) = \{d\}$ for all $d \in \{1, \ldots, D_0\}$. A node $\Psi_{l,d}$ can only be connected to a node in the previous layer $\Psi_{l-1,d'}$ if and only if $m_{l-1}(d') \subseteq m_l(d)$. This is equivalent to restricting weight matrices

to have nonzero values at row d and column d' if and only if $m_{l-1}(d') \subseteq m_l(d)$. Which is equivalent to creating a mask $M_l \in \{0,1\}^{D_l, D_{l-1}}$ for every hidden layer and output layer, such that

$$(M_l)_{d,d'} = \begin{cases} 1, & \text{if } m_{l-1}(d') \subseteq m_l(d) \\ 0, & \text{otherwise.} \end{cases}$$

We can then redefine the hidden layer to be

$$\Psi_l(\boldsymbol{x}_{l-1}) = \gamma\left((W_l \odot M_l)\,\boldsymbol{x}_{l-1} + \boldsymbol{b}_l\right),$$

and the complete network becomes

$$\Psi(\boldsymbol{x}_0) = (W_{L+1} \odot M_{L+1})\,(\Psi_L \circ \Psi_{L-1} \circ \cdots \circ \Psi_1(\boldsymbol{x}_0)) + \boldsymbol{b}_{L+1},$$

where $\odot$ is the Hadamard product. It is typically the latter definition, namely, using masks that are implemented and hence the name *masked* autoencoder for density estimation.

The view above is quite general, and it is often not very interesting to have the image of m_l to be the whole power set. We therefore proceed to specify the problem, which guides us to an adequate range for the mappings m_l, and a fully fledged definition of conditional neural networks.

Masks

We are making models that compute each output variable restricted to a subset of the input variables, in an efficient manner. We therefore have a mapping which informs us of the relationship between input and output for every variable, respectively. Let such a mapping be denoted by $c(y_d) = \{d' \colon y_d \mid x_{d'}\}$ (note that we are not necessarily talking about this condition in terms of probability). With this in mind, we can shrink the range of the functions m_l and define first

$$\mathcal{C}_{min} = \bigcup_{d=1}^{D_{L+1}} c(y_d) \tag{2.11}$$

which is the smallest possible range where every output variable can be computed using all its dependencies. The range is then set to be

$$\mathcal{C} = \left[\bigcup_{d=1}^{D_{L+1}} \mathcal{P}(c(y_d)) \setminus \{\emptyset\}\right] \bigcup \mathcal{C}_{min}, \tag{2.12}$$

where the last union is there to ensure the empty set is included if the output variable is to be calculated by a constant. Otherwise, we exclude the empty set as it enforces neurons to have no connections from the last layer, and hence be a constant. The range also allows for output variables that depend on all input variables, akin to a regular neural network.

The largest and perhaps most impactful difference between $\mathcal{C}_{min}$ and $\mathcal{C}$ is the fact that one can choose sets that are subsets of several $c(y_d)$ sets. If for instance a pair of output variables y_d and $y_{d'}$ depend on the exact same input variables

apart from one each, it may make sense to assign the shared subset to more nodes in the start of the network, and more at the end of the network assign more nodes which lets each node use every possible input variable available to y_d and $y_{d'}$ respectively. However, there are still many sets in $\mathcal{C}$ that is a subset of just one $c(y_d)$, which means that it restricts nodes to use fewer of the input variables available. This can have a regularisation effect, as you are effectively setting some weights that can take any value to zero, yet they may simply be in the way sometimes and make the generating of masks more clouded. It can therefore be beneficial to shrink the range more to

$$\mathcal{C}_s = \left[\bigcup_{D \in \mathcal{P}(\mathcal{D})} \left\{ \bigcap_{d \in D} c(y_d) \right\} \setminus \{\emptyset\} \right] \bigcup \mathcal{C}_{min},$$

where $\mathcal{D} = \{1, 2, \ldots, D_{L+1}\}$. The set $\mathcal{C}_s$ combines the sharing aspects by including all intersections between the different output variables dependencies, but also every set $c(y_d)$. This is one of the most efficient ways to ensure what was previously discussed about sharing information complemented with using all information available for each output variable. In the remaining part of this chapter we will simply use $\mathcal{C}$, but the theoretical results to follow in Chapter 4 holds for any arbitrary set $\mathcal{C}_a$ as long as $\mathcal{C}_{min} \subseteq \mathcal{C}_a$.

CONN

We can now give a reasonable definition of conditional neural networks.

Definition 2.5.2. For all input dimensions we set $m_0(x_d) = \{d\}$, for $l \in \{1, \ldots, L\}$ we set $m_l \colon \{1, \ldots, D_l\} \to \mathcal{C}$, and for every $d \in D_{L+1}$ we have $m_{L+1}(y_d) = c(y_d)$. A *conditional neural network (CONN)* is a L-layered neural network where each hidden layer is of the form

$$\Psi_l^{CONN}(\boldsymbol{x}_{l-1}) = \gamma\left((W_l \odot M_l)\, \boldsymbol{x}_{l-1} + \boldsymbol{b}_l\right)$$

and the network is of the form

$$\boldsymbol{x}_{L+1} = \Psi^{CONN}(\boldsymbol{x}_0) = (W_{L+1} \odot M_{L+1})\left(\Psi_L \circ \Psi_{L-1} \circ \cdots \circ \Psi_1(\boldsymbol{x}_0)\right) + \boldsymbol{b}_{L+1},$$

where all masks M_l are induced by the mappings m_l.

One can easily confirm that the corresponding output y_d of a CONN model is actually computed using only $\{x_d \colon d \in c(y_d)\}$. The first hidden layer has nonzero weights at exactly row d and column d' iff $m_0(d') \subseteq m_1(d)$, by the definition of CONN and how the masks are used. As $m_0(d') = \{d'\}$, means the nonzero weights are exactly the set that connects $\Psi_{1,d}$ to $\{x_{d''} \colon d'' \in m_1(d)\}$. This means $\Psi_{1,d}$ is computed using at most the input variables corresponding to $m_1(d)$.

Assuming every node $\Psi_{l-1,d'}$ is computed using at most $\{x_{d''} \colon d'' \in m_{l-1}(d')\}$, for any arbitrary layer $l - 1 \geq 0$. Nodes in the next layer $\Psi_{l,d}$ has nonzero weights exactly where the corresponding mapping fulfills

$$m_{l-1}(d') \subseteq m_l(d), \tag{2.13}$$

again by the definition of CONN and masks. Due to the assumption made earlier, $\Psi_{l,d}$ is computed using

$$\bigcup_{d':m_{l-1}(d')\subseteq m_l(d)} \{x_{d''} : d'' \in m_{l-1}(d')\}\,.$$

By Equation (2.13), we have that this is at most $m_l(d)$. Hence, by induction, $\Psi_{l,d}$ is computed using $\{x_{d'} : d' \in m_l(d)\}$, for all layers l and dimension d.

As $m_{L+1}(d) = c(y_d)$, by induction, we can safely state that the output y_d is computed using at most $\{x_{d'} : d' \in c(y_d)\}$.

The setting of CONN, in which becomes of utmost importance for our usage later, is where we wish to compute one or more output variables per input variable, which means we need to add a constraint to the output dimension D_{L+1} to be the multiple of the input dimension, i.e. $D_{L+1} \equiv 0 \,(\mathrm{mod}\, D_0)$. We then partition the output vector into $K = D_{L+1}/D_0$ parts, and assign each its unique variable x_d referred to as $\boldsymbol{y}_d$. To further clarify the point and illuminate CONNs generally, we illustrated the CONN model, with said constraint, in Figure 2.3.

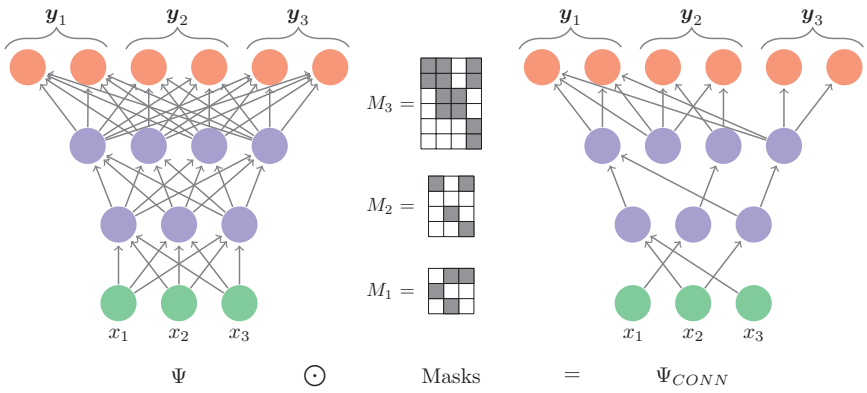


Figure 2.3: Illustration of the application of masks to a 2-layered neural network (left) and the resulting CONN (right). The example has a conditional structure $(c(\boldsymbol{y}_1), c(\boldsymbol{y}_2), c(\boldsymbol{y}_3)) = (\{2,3\},\{1,3\},\{2\})$ and $K = 2$. The mappings are the following: $m_0 = (\{1\},\{2\},\{3\})$, $m_1 = (\{2,3\},\{1\},\{3\})$, $m_2 = (\{2,3\},\{3\},\{1\},\{2\})$, $m_3 = (c(\boldsymbol{y}_1), c(\boldsymbol{y}_2), c(\boldsymbol{y}_3))$.

If we are considering the autoregressive property as with MADE, i.e. $c(y_d) = \{d' : d' < d\}$, the mappings and masks can be rewritten as $\hat{m}_l(d) = \max(m_l(d))$ and

$$(\hat{M}_l)_{d,d'} = \begin{cases} 1, & \text{if } \hat{m}_{l-1}(d') \leq \hat{m}_l(d) \\ 0, & \text{otherwise.} \end{cases}$$

This is similar to the masks in the original paper (Germain et al. 2015), but they have defined the functions m_l slightly differently. This is the main idea behind MADE and CONN, which will become apparent as very useful when we have defined flow-structures Section 3.3 and conditioners Section 3.4.

Agnostic Training

A final part that the original paper (Germain et al. 2015) discussed, and which we include is agnostic training. Shuffling the input variables before running them through the network is known as *order-agnostic training*. This can according to Germain et al. 2015 be beneficial for cases when some value of the input vector is missing in a particular observation, where one can still calculate the output by having the known values first in the vector. Secondly, one can generate some form of ensemble, i.e., a collection of models which can be averaged through simply sending in the input vector with different ordering. This can make sense for conditional structures such as the autoregressive one, as every ordering is a valid one. In the more general setting, one can only consider shuffling variables x_d when the two conditionals are both valid. We will not consider this property when using CONN in our work, but one ought to know of it and its potential usage.

These networks play an important role in a large class of normalizing flows, yet have not been properly studied theoretically, as far as we are aware. In Chapter 4 we investigate the universality of such networks, and we also apply such networks in Chapter 5. We are now ready to introduce normalizing flows.

CHAPTER 3

Normalizing Flows

3.1 Introduction

Normalizing flows are a rather new invention and comes with both promising attributes and many unanswered problems. It is hard to give a detailed definition of normalizing flows, but as an attempt, an all-encompassing definition can be given as sampling from a simple distribution, applying a transformation f, which ought to allow for easy computation as well as easy inversion, and where one can evaluate the exact density—preferably with ease. Such a definition is quite broad and vague, but what it certainly emphasises is that normalizing flow transforms samples and aligns itself with methods such as Whitening transformations (J. H. Friedman 1987; Johnson 1966) and Copulas (Sklar 1959). The requirement for both fast computation, inversion, and density evaluation lets normalizing flows lend itself to both density estimation and variational inference, as alluded to in the first chapter. In this chapter we do not emphasise much the differences between the two, but it is more highlighted empirically in Chapter 5. Specifying the flows we are interested in, roughly speaking, are normalizing flows which exploits the power of neural networks, but without compromising properties such as exact density evaluation. That is not to say we do not consider other flows which do not leverage deep learning, but that most of the flows we study do.

Our main goal in this chapter is to both introduce normalizing flow, but also attempt to make a coherent theory around flows. We therefore attempt to split flows into components, analysing them, before putting them together into flows that we recognise in the literature. We also try to give a better definition of what it means for a flow to be flexible or expressive.

The chapter can be outlined as follows. In Section 3.2 we define normalizing flows in a general manner, discussing what transformations we allow, and ending with a canonical example that hopefully concretise flow and also can be helpful moving onward. In Section 3.3 we start dissecting normalizing flows into components, introducing *structure*. Informally, it tells what variables influences the transformation of another variable. We give a new formal definition of this concept and we both analyse and show neat new results that follows. In Section 3.4 and Section 3.5 we introduce the last parts of our dissection, defining how one transform the variable and give an overview of what transformations exists in the literature. In this part our only contribution is completing our new theory and analyse the different transformations that have been introduced in

the literature. In Section 3.6 we aim to define what it means for a flow to be universal, and then give a comprehensive review of existing results—both in terms of universality and limitations.

In the end we have a fully fledged framework of normalizing flows, while also having introduced the reader for the current literature. We then proceed to develop the theory further in Chapter 4.

3.2 Flows

The first part of a normalizing flow is the *base distribution*. Simply put, any distribution $\mathcal{Q} = (\mathcal{Z}_0, \mathscr{B}(\mathcal{Z}_0), \mu)$, are known as the *base distribution* or *base probability space* if it can efficiently evaluate the density and effectively generate samples. The most common ones in the literature today are the Gaussian distribution, the Student-t distribution, and uniform.

The second part of a normalizing flow are the transformations of a sample $\boldsymbol{z}_0$ from base density to a different and hopefully more complex distribution. The aim of this section is to define normalizing flow formally, which we start by defining a pushforward measure.

Definition 3.2.1 (Kobyzev et al. 2020)**.** If $(\mathcal{Z}_0, \Sigma_{\mathcal{Z}_0})$, $(\mathcal{X}, \Sigma_{\mathcal{X}})$ are measurable spaces, f is a measureable mapping between them, and μ is a measure on $\mathcal{Z}_0$, then one can define a measure on $\mathcal{X}$ as

$$f_*\mu(X) = \mu\left(f^{-1}(X)\right), \text{ for all } X \in \mathcal{X}. \tag{3.1}$$

The measure $f_*\mu(X)$ is known as the *pushforward measure*.

Let $(\mathcal{X}, \Sigma_{\mathcal{X}}, \nu)$ be the measure space we are interested in. Normalizing flows can be seen as a framework that describes classes of functions f and a simpler measure space $(\mathcal{Z}_0, \Sigma_{\mathcal{Z}_0}, \mu)$, such that $f_*\mu = \nu$. When μ is a probability measure implies that the pushforward measure w.r.t f is also a probability measure. This can easily be proven by the fact that

$$f_*\mu(\mathcal{X}) = \mu\left(f^{-1}(\mathcal{X})\right) = \mu(\mathcal{Z}_0) = 1$$

and by letting $\{X_i \colon i = 1, 2, 3, \dots\}$ be sets with pairwise disjoint elements, we have

$$\begin{aligned} f_*\mu\left(\bigcup_{i=1}^{\infty} X_i\right) &= \mu\left(\bigcup_{i=1}^{\infty} f^{-1}(X_i)\right) \\ &= \sum_{i=1}^{\infty} \mu\left(f^{-1}(X_i)\right) \\ &= \sum_{i=1}^{\infty} f_*\mu(X_i). \end{aligned}$$

Hence, we have countable additivity, which means both the requirements for a probability measure are fulfilled. Normalizing flows can therefore be seen as, starting with a simple probability space $(\mathcal{Z}_0, \Sigma_{\mathcal{Z}_0}, \mu)$, applying f on it to achieve a pushforward distribution $(\mathcal{X}, \Sigma_{\mathcal{X}}, f_*\mu)$. The goal being to find f such

that the pushfoward distribution is as close as possible to $(\mathcal{X}, \Sigma_{\mathcal{X}}, \nu)$, w.r.t. a divergence measure.

The view above is quite general and does not lend itself directly to finding an exact density of the pushforward distribution. To achieve this, we need to constrain the class of functions f and the probability space. The probability space is already constricted to continuous distributions and is not a concern.

Limiting the function f will be the other necessary component, such that we can evaluate the density of the pushforward measure.

Definition 3.2.2. A function $f\colon \mathbb{R}^D \to \mathbb{R}^D$ is a diffeomorphism if it is bijective and both itself and its inverse are differentiable. If f and f^{-1} is r times continuous differentiable, we define it as a C^r-diffeomorphism.

Restricting ourselves to f being at least a C^1-diffeomorphism is unnecessary and limiting. We therefore define piecewise diffeomorphisms w.r.t. some distribution.

Definition 3.2.3. Let $(\mathcal{X}, \mathscr{B}(\mathcal{X}), \mu)$ be a probability space with a density p. Let X_i for $i = 0, 1, 2, \ldots, k$ be a partition of $\mathcal{X}$ such that $\mu(x \in X_0) = 0$. A piecewise-diffeomorphism $f\colon \mathcal{X} \to \mathbb{R}^D$ is continuous and restricted to X_i is a diffeomorphism. That is, $f_i\colon X_i \to \mathbb{R}^D$ is a diffeomorphism, for all $i = 1, 2, \ldots, k$. All f_i's are C^r-diffeomorphisms makes f a piecewise C^r-diffeomorphism.

Hence, we shall restrict our choices of f to be piecewise C^1-diffeomorphisms. We can now easily evaluate the density of the pushforward measure, by this well known theorem.

Theorem 3.2.4. *Let $\mathcal{Z}_0 \subseteq \mathbb{R}^D$ and $(\mathcal{Z}_0, \mathscr{B}(\mathcal{Z}_0), \mu)$ be the base probability space. Let f be a C^1-diffeomorphism, where $q_{\boldsymbol{z}_0}$ is the density of the base probability space. Then the density of the pushforward distribution induced by f, is defined as*

$$q_{\boldsymbol{x}}(\boldsymbol{x}) = \sum_{i=1}^{k} q_{\boldsymbol{z}_0}(f_i^{-1}(\boldsymbol{x}))\, |det(J_{f_i^{-1}}(\boldsymbol{x}))|, \tag{3.2}$$

where $J_{f_i^{-1}}(\boldsymbol{x})$ is the Jacobian of the function f_i^{-1} evaluated at $\boldsymbol{x}$.

Hence, we can always evaluate the density of the transformed data, which is one of the major advantages normalizing flows has compared to other popular generative models such as GAN. A special case of the Theorem 3.2.4 is when $k = 1$, which gives us the well-known formula

$$q_{\boldsymbol{x}}(\boldsymbol{x}) = q_{\boldsymbol{z}}(f^{-1}(\boldsymbol{x}))|\det\left(J_{f^{-1}}(\boldsymbol{x})\right)|.$$

From the fact that the transformation is invertible, allows us also to rewrite the Jacobian above to $[J_f(\boldsymbol{z}_0)]^{-1}$, where $\boldsymbol{z}_0$ stems from the base distribution. This follows from the fact that the Jacobian of the identity function $f^{-1}(f(\boldsymbol{z}_0))$ is simply the identity matrix. Applying the chain rule, we have

$$\begin{aligned} I_D = J_{f^{-1}\circ f}(\boldsymbol{z}_0) &= J_{f^{-1}}(f(\boldsymbol{z}_0))\, J_f(\boldsymbol{z}_0) \\ [J_f(\boldsymbol{z}_0)]^{-1} &= J_{f^{-1}}(\boldsymbol{x}), \end{aligned}$$

where the inverse exists as the function is inverse, which means that the determinant of the Jacobian is nonzero, which means the matrix is inverse.

This is a minor point, but is essential in regard to training. As in a maximum likelihood situation, we wish to send the data backwards towards the base density, and we can then calculate the Jacobian of the inverse simultaneously. While in a variational inference situation, we wish to sample the data and transform it so we can evaluate the target likelihood. It is then computationally wise to compute the Jacobian of the forward flow. Hence, the equality can be important in terms of computational speed when implemented.

Strengthening the normalizing flow, we divide into several less complex transformations. That is, using $T \in \mathbb{Z}^+$ transformations, compose the flow

$$\begin{aligned} f(\boldsymbol{z}_0) &= f_T \circ f_{T-1} \circ \cdots \circ f_2 \circ f_1(\boldsymbol{z}_0) \\ &= \bigcirc_{t=1}^{T} f_t(\boldsymbol{z}_0). \end{aligned}$$

We let f be the flow and $z_{t,d}$ be the dth dimension transformed t times. Equivalently, we let $\boldsymbol{z}_t$ be the vector transformed t times.

The benefit of less complex transformations f_t is that we can use rather simple, often computationally faster functions, and often scale linearly in terms of composition, i.e., increasing T. It also allows for sharing of information between the dimensions, while allowing for quick evaluation of the density, which we shall come back to in Section 3.3. Using several transformations, where each transformation may give an easy to calculate Jacobian, means that we can easily find the density of the pushforward measure (as long as we keep the dimension of each time step t equal). Applying the chain rule, we have

$$\det\left(J_{\bigcirc_{t=T}^{1} f_t^{-1}}(\boldsymbol{x})\right) = \det\left(J_{f_1^{-1}}(\boldsymbol{z}_1) \cdots J_{f_{T-1}^{-1}}(\boldsymbol{z}_{T-1}) \cdot J_{f_T^{-1}}(\boldsymbol{x})\right),$$

where $\boldsymbol{z}_t = f_{t+1}^{-1} \circ \cdots \circ f_{T-1}^{-1} \circ f_T^{-1}(\boldsymbol{x})$. Using then the fact that for square matrices A, B we have $\det(A \cdot B) = \det(A) \cdot \det(B)$, we get

$$\det\left(J_{\bigcirc_{t=T}^{1} f_t^{-1}}(\boldsymbol{x})\right) = \prod_{t=1}^{T} \det\left(J_{f_t^{-1}}(\boldsymbol{z}_t)\right).$$

In terms of density, one could have obtained the same result with regard to the determinant of compositions through observing that the input $\boldsymbol{z}_{t-1}$ to f_t also have a density. We can then apply recursively Equation (3.2), and when $k = 1$ for all transformations, we have the base density times $\prod_{t=1}^{T} \det\left(J_{f_t^{-1}}(\boldsymbol{z}_t)\right)$.

We can now define normalizing flows formally for the purposes of this thesis. This will not be all-encompassing, as we are working with a specific probability space and discrete time steps in our flow, i.e., $t \in \mathbb{Z}^+$. There are other flows defined for continuous time and with discrete distributions, but is beyond the scope of this thesis.

Definition 3.2.5. Let $\mathcal{Q} = (\mathcal{Z}_0, \mathscr{B}(\mathcal{Z}_0), \mu)$ be a probability space with $\mathcal{Z}_0 \in \mathbb{R}^D$. Let f_t be a piecewise C^1-diffeomorphism for all $t = 1, 2, \ldots, T$. A *normalizing flow* (NF) is defined by $(\mathcal{Q}, f)$, where $\mathcal{Q}$ is the base probability space and $f = \bigcirc_{t=1}^{T} f_t$ is the flow. If the dimensions after each transformation f_t is

constant, the induced density by letting a sample $\boldsymbol{z}_0$ from $\mathcal{Q}$ flow through f is then given by

$$q_{\boldsymbol{z}_T}(\boldsymbol{z}_T) = \sum_{i_T=1}^{k_T} \cdots \sum_{i_1=1}^{k_1} q_{\boldsymbol{z}_0}\left(\bigcirc_{t=T}^{1} f_{t,i_t}^{-1}(\boldsymbol{z}_T)\right) \prod_{t=1}^{T} \det(J_{f_{t,i_t}^{-1}}(\boldsymbol{z}_t)), \tag{3.3}$$

where f_{t,i_t} is the diffeomorphism of transformation t over partition i_t.

When f is a C^1-diffeomorphism, we get the induced density

$$q_{\boldsymbol{z}_T}(\boldsymbol{z}_T) = q_{\boldsymbol{z}_0}\left(\bigcirc_{t=T}^{1} f_t^{-1}(\boldsymbol{z}_T)\right) \prod_{t=1}^{T} \det(J_{f_t^{-1}}(\boldsymbol{z}_t)).$$

Notice that in the definition of NF we have not included anything regarding the target distribution. Even though we often speak about a flow and a target distribution, the flow is simply defined by transforming samples from a base distribution in such a manner that we can also evaluate the induced density. The application of flows will necessarily be concerned with target distribution and the minimisation of a measurement between target and flow induced density. One can also ask questions about a particular flow and its capability/flexibility w.r.t. target distribution. Ultimately, any combination of $(\mathcal{Q}, f)$ defined as above is a flow, no matter how trivial or impractical the resulting distribution is. We do, however, wish to find transformations f by considering the following points.

- Flexible and expressive, such that we can always transform from $\mathcal{Q}$ to any target distribution as described above.
- Limit the number of parameters to estimate.
- Computation wise, cheap to compute both inverse and the Jacobian determinants.

Clearly, there may be some compromise between the first and the other two points. Our goal is then to construct flows such that one can allow for high expressitivity while remaining computationally feasible.

An Example of a Flow

Before we start deconstructing a normalizing flow into components, we find it useful to introduce an example and deconstructing the example. This gives the reader a certain sense of where we are headed, and a more complete picture, as well as something less abstract. The terms introduced here may be unfamiliar to the reader, but tying the concept to the word will ease the experience through the next sections.

Example 3.2.6. A flow often referred to as an inverse autoregressive flow (Kingma, Salimans et al. 2016), transforms every variable $z_{t-1,d}$ to $z_{t,d}$, by applying two parameters a, b. The two parameters are calculated by something we define as a *conditioner* $\mathcal{H}_{t,d}$. The domain of the conditioner is given by a *structure* $\mathcal{S}$. The structure tells which variables, including $z_{t-1,d}$, to use when transforming $z_{t-1,d}$ into $z_{t,d}$. In IAF, we use

$$\mathcal{S}(t,d) = \{(t-1, d') \colon d' \leq d\}.$$

The conditioner uses all variables given by the structure except for $z_{t-1,d}$ to compute $a_{t,d}, b_{t,d}$, i.e.,

$$a_{t,d}, b_{t,d} = \mathcal{H}_{t,d}(\{(t-1, d') \colon d' < d\}).$$

We then apply these parameters to $z_{t-1,d}$ following a *transformation*

$$z_{t,d} = f_{t,d}(z_{t-1,d}) = a_{t,d} \cdot z_{t-1,d} + b_{t,d}.$$

The form/parametrization of computing $z_{t,d}$, which in this case is an affine transformation, is called a *transformation*. Hence, we have the *structure* that tells us to use the $d-1$ first variables from the last time step $t-1$, to compute the parameters using a *conditioner*, which then is applied to the variable $z_{t-1,d}$ defined by the *transformation*.

We return to this example later on, when it is fruitful. We also take the opportunity to address the overload of the term *transformation*. We both use it when speaking about the transformation of $\boldsymbol{z}_0$, as in applying f, but also for each step $f_{t,d}$. We rely on context to differentiate between the two, but also try to use $f_{t,d}$ consequently when discussing the transformation of each variable. Furthermore, we shall refer to the parameters that actually transform $z_{t-1,d}$, e.g. $(a_{t,d}, b_{t,d})$ in the example above, as *the parameters of the transformations*, and the parameters attached to the conditioner, e.g. the neural network in the example above, as *conditioners parameters* or *trainable parameters*. If the flow does not use an conditioner, which we do encounter, then the two types of parameters are the same. To conclude, it is the trainable parameters that we can optimise, and the parameter of the transformation we apply to a variable to transform it.

We are now ready to start dissecting the flow into different parts, analysing each part by itself, before we finally put them together again to make flows such as the IAF.

3.3 Flow Structure

When constructing transformations f one has to choose the form of the function or the transformation, as well as the structure. By structure we mean which variables $z_{i,j}$ is needed to calculate the transformation. The form of the function is how the variables, given by the structure, are used. That is, when transforming $z_{t,d}$ we apply $f_{t,d}(z_{t-1,d}, \dots)$, where the dots indicate what other variables needed to compute $f_{t,d}$. There are two things we need to define, one is what variables other than $z_{t-1,d}$ are needed, and the other is how we then use these variables to compute $f_{t,d}$. Some of the most popular flows can use a myriad of structures, which we shall formalise here, and give rise to different models. Therefore, while some transformations in the literature only allows for a specific structure, others allow for a larger class of them. It is therefore useful to explore what structures give different properties such as fast computation of the Jacobian determinant. Usually, there are some popular choices that are used, which we define later, which are often very well motivated, but we are looking to generalise this to family of structures that can obtain certain properties, and which we hence try to shed some light on in this section.

We introduce, for ease of readability, $\mathcal{T} = \{1, 2, \dots, T\}$, $\mathcal{T}_0 = \{0\} \cup \mathcal{T}$, and $\mathcal{D} = \{1, 2, \dots, D\}$. Although the flow must start and end with the dimension D,

this does not mean the t'th transformation must oblige to the same constraint. As long as the flows follow Definition 3.2.5, then there is no problem. We therefore define a set of sets, which contains the indices for each transformation $t \in \mathcal{T}$, namely $D_t \in \mathbb{N}$,

$$\begin{aligned}\mathcal{D}_{\mathcal{T}} &\coloneqq \{\mathcal{D}_1, \dots, \mathcal{D}_{T-1}, \mathcal{D}_T\} \\ &\coloneqq \{\{1, 2, \dots, D_1\}, \dots, \{1, 2, \dots, D_{T-1}\}, \{1, 2, \dots, D\}\}\end{aligned}$$

where all $D_t \geq D$ and the last set oblige to the requirement of D dimensions in the first and last transformation. Similarly to $\mathcal{T}$, we also include a set for time step 0,

$$\mathcal{D}_{\mathcal{T}_0} = \mathcal{D}_{\mathcal{T}} \cup \mathcal{D}_0 = \mathcal{D}_{\mathcal{T}} \cup \{1, 2, \dots, D\},$$

where $\mathcal{D}_0 = \mathcal{D}$ to comply with the D dimensions in the first layer.

We also need to define a cross product between $\mathcal{T}$ and $\mathcal{D}_{\mathcal{T}}$, which does not follow the well-known Cartesian product. We therefore define another cross-product,

$$\mathcal{T}_0 \otimes \mathcal{D}_{\mathcal{T}_0} \coloneqq \{(t, d) \colon t \in \mathcal{T}_0,\, d \in \mathcal{D}_t\},$$

and similarly,

$$\mathcal{T} \otimes \mathcal{D}_{\mathcal{T}} \coloneqq \{(t, d) \colon t \in \mathcal{T},\, d \in \mathcal{D}_t\},$$

Now we are ready to define what the structure of a flow is, in which we use $\mathcal{P}(\cdot)$ to mean the power set of the given argument.

Definition 3.3.1. Let $(\mathcal{Q}, f)$ be a normalizing flow. A *flow-structure* is defined as a mapping

$$\mathcal{S} \colon \mathcal{T} \otimes \mathcal{D}_{\mathcal{T}} \to \mathcal{P}(\mathcal{T}_0 \otimes \mathcal{D}_{\mathcal{T}_0}),$$

The output $\mathcal{S}(t, d)$ for any particular (t, d) in the domain, is the set indicating which variables in the flow that are used to calculate $z_{t,d}$ i.e., .

$$z_{t,d} = f_{t,d}(\{z_{i,j} \colon (i, j) \in \mathcal{S}(t, d)\})$$

A structure is also required to have the following properties

- for any $t \in \mathcal{T}$, $d \in \mathcal{D}_t$, and $d \leq D$, then $(t-1, d) \in \mathcal{S}(t, d)$,
- for any $t \in \mathcal{T}$, $d \in \mathcal{D}_t$, and $d > D$, there exists a $d' \in \mathcal{D}_{t-1}$ such that $(t-1, d') \in \mathcal{S}(t, d)$.

The to properties restrict the structure to contain a certain order of the variables, hence we have a variable from last time step that is used when applying $f_{t,d}$. This means we can talk about a variable being transformed, or more formally:

Definition 3.3.2. Let $(\mathcal{Q}, f)$ be a normalizing flow with structure $\mathcal{S}$. For any variable $z_{t,d}$, $t > 0$, its *predecessor* is defined as

- $z_{t-1,d}$ if $d \leq D$.

- $z_{t-1,i}$ for some $i \in \mathcal{D}_{t-1}$ if $d > D$.

Any variable must have one and only one predecessor. We let $\mathcal{S}_{int}(t,d) = (t-1,i)$ where $z_{t-1,i}$ is the predecessor of $z_{t,d}$, and $\mathcal{S}_{ext}(t,d) = \mathcal{S}(t,d) \backslash \{(t-1,i)\}$.[1]

The point with flow structures is to give us information on what variables influence what variables. That is, the transformation of $z_{t-1,d}$ is given by

$$\begin{aligned} z_{t,d} &= f_{t,d}(\{z_{i,j} \colon (i,j) \in \mathcal{S}(t,d)\}) \\ &= f_{t,d}(\{z_{t-1,j} \colon (t-1,j) \in \mathcal{S}_{int}(t,d)\} \cup \{z_{i,j} \colon (i,j) \in \mathcal{S}_{ext}(t,d)\}) \end{aligned}$$

We will often write $f_{t,d}(z_{t-1,j})$, where $z_{t-1,j}$ is the predecessor, as a shortening, knowing that we have a structure to inform which variables are needed to compute the transformation. In addition, we abuse the notation somewhat and allow to write $z_{i,j} \in \mathcal{S}(t,d)$. Finally, we often use *layer* t or *transformation layer* t when referring to all the variables in which have first index t, i.e. $\{z_{t,d} \colon d \in \mathcal{D}_t\}$.

It is very useful to interpret $\mathcal{S}$ as a graph. Let $(\mathcal{Q}, f)$ be a normalizing flow with an accompanied flow structure $\mathcal{S}$. We define a graph G with vertices $V = \mathcal{T}_0 \otimes \mathcal{D}_{\mathcal{T}_0}$ and directed edges E given as

$$E = \{((t',d'),(t,d)) \colon (t',d') \in \mathcal{S}(t,d) \text{ and } (t,d) \in V\}.$$

That is, there is an edge to (t,d) from all vertices in $\mathcal{S}(t,d)$. Equivalent to the set definition above, there are two types of edges. We have E_{int} and E_{ext}, with $E = E_{int} \cup E_{ext}$. These have the same interpretation as for the set definition, i.e., edges from the predecessor and edges from "assisting" variables, respectively. Continuing the Example 3.2.6, we have added a graph of the corresponding structure $\mathcal{S}$ when $D = 4$ and $T = 3$ in Figure 3.1.

We hence allow $\mathcal{S}$ to both be referred to as the mapping in Definition 3.3.1 and also the graph it induces, with the context deciding which one. Typically, if we speak about $\mathcal{S}$ itself, we tend to do it through graph G. When we are talking about a specific variable and what it is dependent on, we refer to the map $\mathcal{S}$ and the set it outputs for a given variable.

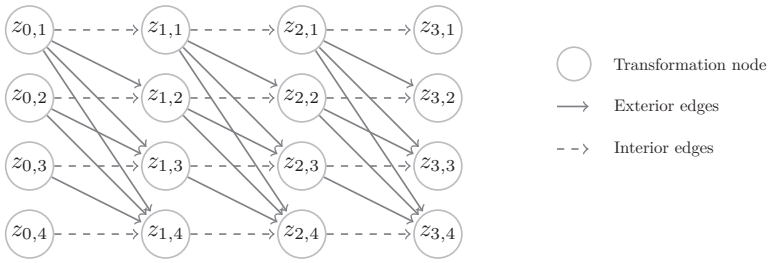


Figure 3.1: The structure corresponding Example 3.2.6, with $T = 3$, $D = 4$, and $D_t = D$ for all $t \in \mathcal{T}$.

[1] *int* is short for interior and refers to the fact that $z_{t,d}$ is directly transformed from the interior variable. While other variables "assist" with the transformation from the "outside", hence exterior (*ext*).

DAGs

There are certain possible structures which give flows that either make little sense, or at least not executable. For instance, structures that need each other to calculate the next transformed variable, that is, $(t, d) \in S(i, j)$ and $(i, j) \in S(t, d)$. This means that one needs to compute $z_{i,j}$ to compute $z_{t,d}$, and vice versa. One could perhaps think of it as repeating the transformation, alternating between updating the two variables, however, this leaves unspecified how many times to repeat and can simply be rewritten into a structure by increasing the number of transformations. The issue does not only pertain to two variables directly dependent on each other, but also chains of variables. If we think of it through a graph perspective, one can easily restrict possible structures such that we avoid the aforementioned issue. By ensuring that the structure induces a direct acyclic graph (DAG), means we completely eradicate the issue at hand. Through this extra requirement, we can ensure that the flow is *computable*. By this we mean that we can compute any variable forward and backwards, as well as evaluate the induced density in finite time. We therefore introduce *valid* structures.

Definition 3.3.3. A flow-structure $\mathcal{S}$ corresponding to a flow $(\mathcal{Q}, f)$ is *valid* if the induced graph of S is a directed acyclic graph (DAG).

We shall assume, if not explicitly stated, that every structure we consider is valid. The validity of the structures allows us to state the following.

Proposition 3.3.4. *Assume the flow f is such that each transformation $f_{t,d}$ can be computed in finite time, where $(t, d) \in \mathcal{T} \otimes \mathcal{D}_{\mathcal{T}}$. A flow f is computable if and only if the structure $\mathcal{S}$ is valid.*

Proof. The proof is quite straightforward. Assume the structure is not valid. Then there must exist at least one cycle in the induced graph of $\mathcal{S}$. This means that one cannot in finite time compute a variable from the base distribution to $\boldsymbol{z}_T$. As there are at least two variables $z_{i,j}$ and $z_{k,l}$, which both require each other to compute.

Assume now that the structure is valid. This means that the induced graph has at least one topological ordering. This means that if we transform each variable in the ordering given, means we transform $z_{t,d}$ after we have transformed all necessary variables needed to compute it. This is because there are no edges to (t, d) that is after the said node in the ordering, per the definition of the topological ordering. As this holds for all $(t, d) \in \mathcal{T}_0 \otimes \mathcal{D}_{\mathcal{T}_0}$, the definition of the transformation is a piecewise diffeomorphism, and the assumption of finite time computing each transformation, implies the flow must be computable. ■

This connects a link between our structure and the work of Wehenkel et al. 2020, as they note the connection between normalizing flows and Bayesian networks, which also depends on the graph being a DAG. Obviously, the Bayesian networks have their own interpretation, while we are merely working on what variables each transformation are using, disconnected from any real dependencies in the target distribution. In addition, the work with structures was done independently from Wehenkel et al. 2020.

We have now established a class of structures which in one sense is all we need, as the goal for normalizing flows is to be able to send any sample forward and backward, and evaluating its density. In another sense there are still problems in terms of speed and computational power, ease of invertibility, and computing the log determinant of the Jacobian. There are some questions that deserve consideration:

- What structure, combined with what transformations, produces fast to invert flows?
- What structures allow for vectorization of many transformations, and the trade-off between these and other structures?
- What structure allows for quick computation of the log determinant of the Jacobian? This will clearly also depend on the transformations we choose. However, we shall see that the structure plays the larger part in this.

When it comes to comparing different structures, a natural interpretation of this is how much information each of the transformation can use, or are available. Many edges directed to a variable means that it can utilise more information and potentially be more flexible. This comes at the cost of increasing the computational burden, both with inversion and with the Jacobian determinant, and potentially making it less stable.

Crossing Paths

Regarding the question of invertibility and how efficient it can be done, how we transform a variable, i.e., how $f_{t,d}$ is defined, will play a big role. There is therefore hard to state any general results based solely on the structure. That being said, one problem that can cause issues when calculating the inverse of the flow, can be defined as *crossing paths*.

This can be intuitively thought of as a graph where the edges form a cross. More formally:

Definition 3.3.5. Let $\mathcal{S}$ be a structure in a flow $(\mathcal{Q}, f)$. A *crossing path* is when at least two variables $z_{t,d}$ and $z_{t,d'}$, where $d \neq d'$, such that there exists $t' < t$ and $t'' < t$, and $z_{t',d} \in \mathcal{S}(t, d')$ and $z_{t'',d'} \in \mathcal{S}(t, d)$.

A crossing path can be seen as transforming variables by using each others predecessors as input. This is by no means inherently negative w.r.t. invertibility, there are indeed many transformations that can calculate the inverse, yet contain one or more crossing paths. However, the lack of them typically give easier computation of the inverse. To conclude, we generally want to avoid crossing paths.

Triangular

Although the invertibility is hard to investigate without a specific group of transformations, we can in contrast guarantee fast computation of the determinant of the Jacobian. When we have computed the Jacobian, there is an upper bound on the time complexity of computing the determinant, as we can

apply LU-decomposition to the matrix and calculate the determinant easily from the two triangular matrices, this method still have an asymptotic complexity of $\mathcal{O}(D^3)$ [2] This is also considering the computation of the determinant after obtaining the Jacobian, which itself can also add an extra cost. Considering the flow typically have several transformations, and we use structures such that the transformation of $\boldsymbol{z}_t$ does not use any variables other than $\boldsymbol{z}_{t-1}$, means we can split the computation of the determinant up in T parts, as described in Section 3.2. We then have a complexity of $\mathcal{O}(T \cdot D^3)$, assuming the calculation of the Jacobian for each time step are less than or equal to $\mathcal{O}(D^3)$, which they tend to be when we divide the transformation into T steps.

One of the most efficient and stable methods we have is to simply ensure that the Jacobian $\mathcal{J}$ is triangular. The determinant $\det(\mathcal{J})$ is then equal to the multiplicative trace $\prod_{d=1}^{D} J_{d,d}$, and we get rid of both evaluating the Jacobian matrix and general computation of determinants. We are therefore interested in finding structures that make the Jacobian, regardless of the transformation, is triangular. This will in turn speed up immensely the density evaluation and increase its stability (speed and stability will of course depend on transformations, but it holds true generally speaking).

Our aim for the rest of this part is twofold. We show that any flow with any corresponding structure can be split up into T parts, where every edge goes from $t-1$ to t transformation step, while the final distribution of the flow remains the same. Secondly, we give the requirements of structures to obtain a triangular Jacobian, when the structures have constant dimension D, i.e., $\mathcal{D}_{\mathcal{T}} = \{\mathcal{D}, \mathcal{D}, \ldots, \mathcal{D}\}$.

Flow-isomorphism

We first start by defining the identity node and the notion of flow-isomorphism, which is useful to alter the structure, but not the output (as we do not transform any variable).

Definition 3.3.6. Let $\mathcal{S}$ be flow-structure. A node (t, d) in the graph $\mathcal{S}$ is an *identity node* $Id_{\mathcal{S}}$ if and only if $\mathcal{S}_{ext}(t, d) = \emptyset$ and the transformation associated with the node is the identity function $f_{t,d}(z_{t,d}) = z_{t,d}$.

In any structure, one can easily remove or add an identity node $z_{t,d}$ and the graph will remain mostly the same. The operations simply reroute edges between two nodes when adding, and the other way around for deleting.

We are about to define an isomorphism between flows, where we want to be exclude identity nodes and therefore create the following restriction on a flow-structure $\mathcal{S}$, by restricting the nodes in the corresponding vertices V,

$$\bar{V} = V \setminus \{(t, d) \colon (t, d) \in V \text{ and } (t, d) \text{ is a identity node } Id_{\mathcal{S}}\}.$$

We also wish to denote, given an identity node, the first ancestor in the graph which is not an identity node. This can be done by the following recursive

[2] We choose to compare with LU-decomposition, as it is commonly used, stable and easy to understand. There are other alternatives, yet most of them run in $\mathcal{O}(D^3)$. Other methods such as Arnoldi iteration is quicker, but is only an approximation.

function: for any node (t, d)

$$\text{pred}((t,d);\mathcal{S}) = \begin{cases} (t,d), & \text{if } (t,d) \text{ is not an identity node} \\ \text{pred}[\mathcal{S}_{int}((t,d));\mathcal{S}], & \text{otherwise.} \end{cases}$$

Finally, we also denote, for any pair of vertices (t, d) and (t', d') in a structure,$(t, d) <_{\text{Top}} (t', d')$ denotes that (t, d) is an ancestor of (t', d') in every topological sorting.

We may now define a *flow-isomorphism*, which we can more informally state as any two flows where one can achieve equality between them by allowing only to add/remove identity nodes.

Definition 3.3.7. Let $(\mathcal{Q}, f)$ be a NF with structure $\mathcal{S} = (V, E)$ and equivalently, $(\mathcal{Q}, f^*)$ with structure $\mathcal{S}^* = (V^*, E^*)$. The two structures are *weak flow-isomorphic* if there exist a bijection $g\colon \bar{V} \to \bar{V}^*$ such that for *all* $(t, d) \in \bar{V}$ and $(t', d') \in \bar{V}$,

(i) $g(0, d) = (0, d)$ for every $d \in \mathcal{D}$ (which is always possible due to the same base distribution).

(ii) $(t, d) <_{\text{Top}} (t', d') \iff g_1(t, d) <_{\text{Top}} g_1(t', d')$, i.e., preserves the topological ordering in some sense.

(iii) There exists a bijection

$$h\colon \mathcal{S}(t, d) \to \mathcal{S}^*(g(t, d))$$

s.t for every node $(i, j) \in \mathcal{S}(t, d)$

$$g[\text{pred}((i, j);\mathcal{S})] = \text{pred}(h((i, j));\mathcal{S}^*).$$

(iv) $f_{t,d}$ is equal to $f^*_{g(t,d)}$, i.e.,

$$f_{t,d}(\boldsymbol{z}) = f^*_{g(t,d)}(\boldsymbol{z}), \quad \boldsymbol{z} = \mathbb{R}^{|\mathcal{S}|}$$

We define two structures to be *flow-isomorphic* if g also fulfills

(v) $g(\text{pred}[(T, d);\mathcal{S}]) = \text{pred}((T, d);\mathcal{S}^*)$ for every $d \in \mathcal{D}$.

We denote the weak flow-isomorphism between two structures as $\mathcal{S} \simeq_W \mathcal{S}^*$ and flow-isomorphism as $\mathcal{S} \simeq \mathcal{S}^*$.

Remark 3.3.8. When the structures have no identity nodes, the definition above is very similar to graph isomorphism, but with the added constraint that the transformations are equal as well. Giving some intuition on the different properties of g, we have that (i) simply preserves the order of $\boldsymbol{z}_0$, and (ii) prevents unnatural mappings such as $g((1, 2)) = (T, 2)$. The reason behind the first two will also become more apparent during the proof of Proposition 3.3.10. The points (iii) makes sure that the mapping of edges is a very natural extension of graph isomorphism, which can be written as $e_i \in E \iff g(e_i) \in E^*$. As g is not a bijection over the whole set of vertices, we cannot simply require $e_i \in E \iff g(e_i) \in E^*$. Roughly speaking, (iii) states that the nonidentity

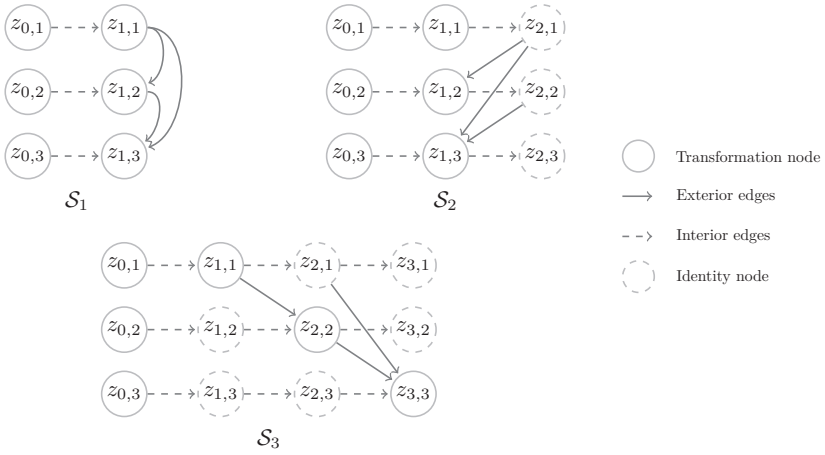


Figure 3.2: Example of three structures that are all flow-isomorphic.

ancestor to any exterior edge exists in $\mathcal{S}$ iff it exists in $\mathcal{S}^*$, and we have the same number of edges. Point (iv) states that the transformation under g is equal, which means that when the input is equal, the transformation is equal. The final point (v) is similar to (i) and preserves the output ordering, where we also handle that case when there are identity transformations as final transformations.

It is trivial to check that (weak) flow-isomorphism constitutes an equivalence relation, as both g and h are bijections. One may also be interested in subset of flows, which in some sense can be interpreted as everything the flow with a subset structure can approximate, so can the superset. More formally:

Definition 3.3.9. Let $\mathcal{S}$ and $\mathcal{S}'$ be two flow-structures. We say $\mathcal{S}$ is an *subset* of $\mathcal{S}'$, if by removing edges in $\mathcal{S}'$ and changing nodes into Identity nodes, the resulting structure is flow-isomorphic to $\mathcal{S}$. We denote this by $\mathcal{S} \subseteq \mathcal{S}'$.

As we are removing edges, and effectively removing transformations, the flow with structure $\mathcal{S}'$ should be able to approximate every distribution $\mathcal{S}$ can, and potentially more distributions—this is not trivially obvious, but we later show for a certain class of flows Lemma 3.6.11, that such interpretation holds. The claims above relies on the fact that two flows that are flow-isomorphic can approximate the same distributions, which we now show.

Proposition 3.3.10. *Let $(\mathcal{Q}, f)$ be a NF with structure $\mathcal{S}$ and $(\mathcal{Q}, f^*)$ be a NF with structure $\mathcal{S}^*$. If $\mathcal{S} \simeq_W \mathcal{S}^*$, then the induced densities, i.e. the densities corresponding to $\boldsymbol{z}_T = f(\boldsymbol{z}_0)$, q_f and q_{f^*} are equal. That is,*

$$q_f(f(\boldsymbol{z}_0)) = q_{f^*}(f^*(\boldsymbol{z}_0)),\ \boldsymbol{z}_0 \sim q_{\boldsymbol{z}_0}$$

If $\mathcal{S} \simeq \mathcal{S}^*$, then we also have $f(\boldsymbol{z}_0) = f^*(\boldsymbol{z}_0)$.

Proof. Assume first that $\mathcal{S} \simeq_W \mathcal{S}^*$. Start by first confirming $z_{0,d} = z_{g(0,d)}$ for $d \in \mathcal{D}$, which holds due to (i) in Definition 3.3.7. Assume inductively that $z_{t',d'} = z_{g(t',d')}$ for every nonidentity node (t', d') up to, but not including, the node (t, d) in an arbitrary topological ordering of $\mathcal{S}$. Due to (ii) we have that every node in $\mathcal{S}(t, d)$ and $\mathcal{S}^*(g(t, d))$ is before the node (t, d) and $g(t, d)$, in

every topological ordering of $\mathcal{S}$ and $\mathcal{S}^*$, respectively. Combining the inductive hypothesis and (ii) with point (iii), we have $z_{i,j} \in \mathcal{S}(t,d) = z_{h(i,j)} \in \mathcal{S}^*(g(t,d))$, for all $(i,j) \in \mathcal{S}(t,d)$. Due to (iv), we know that $z_{t,d} = z_{g(t,d)}$. By induction we know that $z_{t,d} = z_{g(t,d)}$ for all nonidentity nodes $(t,d) \in \bar{V}$.

We can then conclude that for every $(T,d) \in \mathcal{S}$, there exists a $(T,d') \in \mathcal{S}^*$, such that $f(\boldsymbol{z}_0)_d = z_{T,d} = z^*_{T,d'} = f^*(\boldsymbol{z}_0)_{d'}$. This from the fact that $g[\text{pred}((T,d);\mathcal{S})]$ must be mapped to one of the last nonidentity nodes in $\mathcal{S}^*$, due to bijection of g and (ii), and hence must be $\text{pred}((T,d');\mathcal{S}^*)$ for some $d' \in \mathcal{D}$. Using the induction and the fact that any added nodes after $\text{pred}((T,d);\mathcal{S})$ and $\text{pred}((T,d');\mathcal{S}^*)$ are identity transformations, as per the definition of pred, implies $f(\boldsymbol{z}_0) = P\, f^*(\boldsymbol{z}_0)$, for some permutation matrix P. We then have,

$$\begin{aligned} f(\boldsymbol{z}_0) &= P\, f^*(\boldsymbol{z}_0) \\ \frac{\partial}{\partial \boldsymbol{z}_0} f(\boldsymbol{z}_0) &= P\, \frac{\partial}{\partial \boldsymbol{z}_0} f^*(\boldsymbol{z}_0) \\ \mathcal{J}_f(\boldsymbol{z}_0) &= P\, \mathcal{J}_{f^*}(\boldsymbol{z}_0). \end{aligned}$$

We can then write the two densities as, letting $\boldsymbol{x} = f(\boldsymbol{z}_0)$ and $\boldsymbol{x}^* = f^*(\boldsymbol{z}_0)$

$$\begin{aligned} q_f(\boldsymbol{x}) &= q_{\boldsymbol{z}_0}(f^{-1}(\boldsymbol{x})) \cdot |\det \left(\mathcal{J}_{f^{-1}}(\boldsymbol{x})\right)| \\ &= q_{\boldsymbol{z}_0}((f^*)^{-1}(\boldsymbol{x}^*)) \cdot |\det P\, \left(\mathcal{J}_{(f^*)^{-1}}(\boldsymbol{x}^*)\right)| \\ &= q_{\boldsymbol{z}_0}((f^*)^{-1}(\boldsymbol{x}^*)) \cdot |\det(P) \det \left(\mathcal{J}_{(f^*)^{-1}}(\boldsymbol{x}^*)\right)| \\ &= q_{\boldsymbol{z}_0}((f^*)^{-1}(\boldsymbol{x}^*)) \cdot |\det \left(\mathcal{J}_{(f^*)^{-1}}(\boldsymbol{x}^*)\right)| \\ &= q_{f^*}(\boldsymbol{x}^*), \end{aligned}$$

where we used the fact that $\det P = \pm 1$. The densities are equal, and the probability distributions induced by the flows are equal.

If we also have $\mathcal{S} \simeq \mathcal{S}^*$, the proof above holds. Additionally, we also have that $z_{T,d} = z^*_{T,d}$. Using the same argument as earlier in the proof concerning $g[\text{pred}((T,d);\mathcal{S})]$, it must now be mapped to $\text{pred}((T,d);\mathcal{S}^*)]$, due to (v), that is, $d' = d$. Applying the inductive argument leads us to $z_{T,d} = z^*_{T,d}$ for all $d \in \mathcal{D}$. ■

This allows us to rewrite any structure to a flow-isomorphic one, and the corresponding flows have the same distribution, and possibly same output.

Triangular Structures

We can now proceed to the second part, which is to prove some requirements to ensure a triangular Jacobian. First, we define locality in a structure.

Definition 3.3.11. Let $\mathcal{S}$ be a flow-structure. The structure is a *local flow-structure* if every edge in the structure is between neighbouring layers or itself, i.e., there are no edge $((t,d),(t',d'))$ such that $|t - t'| > 1$. We say $\mathcal{S}$ is *forward local flow-structure* if every edge $((t,d),(t',d'))$ goes from the previous layer to the next, i.e., $t' - t = 1$.

We can then create a function which reduces the number of possible structures to a smaller space contains only forward local structures, and then show that every other structure is flow-isomorphic to the aforementioned space. The following transformation might be a bit hard to grasp at first, but keep in mind that it only takes an arbitrary structure $\mathcal{S}$ and create a forward local structure out of it.

Definition 3.3.12. Let $\mathscr{S}$ be the set of all valid flow-structures, and let $\mathscr{S}_{loc}$ be the set of all forward local flow-structures. Let $\Lambda\colon \mathscr{S} \to \mathscr{S}_{loc}$ be a function outputting a new structure $\Lambda(\mathcal{S}) = \hat{\mathcal{S}}$, and is described as a composition of two functions $\Lambda = \Lambda_2 \circ \Lambda_1$ (see Figure 3.3 for visual explanation), as follows,

(i) Λ_1: For every layer $t > 0$:

- Every node in the layer which does not contain edges from other nodes in the same layer; add an identity node between layer t and $t+1$ for that node (there exists at least one such node due to valid structure).
- Every node with an edge from another node in the same layer, replace the node with identity and add the node between t and $t+1$. Repeat until nodes in the new layer have no edges in between layer t.

(ii) Λ_2: Wherever there are edges from node (t, i) to another (t', j), with layers in between:

- If $t' > t$, we expand the next layer $t+1$ with an identity node, where (t, i) is the predecessor. This node serves as storage of $z_{t,i}$. Keep the new node using identity nodes until layer $t'-1$, and then add edge down to (t', j).
- If $t' < t$, i.e., edge from the layer ahead to the previous layer. We know $i \neq j$ due to valid structure, and we simply add identity nodes between t and t' until $t' - t = 1$ for all dimensions except i. This can be done due to the validity of the structure. Repeat the process until no edge exists except from layer t to $t+1$.

Following the function described above, we may therefore turn any structure into a forward local structure. We have illustrated an example of applying Λ in Figure 3.3. A neat little result follows, as a culmination of the definition and flow-isomorphism.

Proposition 3.3.13. *Let $(\mathcal{Q}, f)$ be any normalizing flow with a valid flow-structure $\mathcal{S}$. Then there exists a flow $(\mathcal{Q}, f)$ with a valid structure $\hat{\mathcal{S}}$ which is forward local-structure, and the corresponding flows induce the same distribution.*

Proof. We first apply $\hat{\mathcal{S}} = \Lambda(\mathcal{S})$. It is trivial to check that it is forward local-structure. As the function only changes the structure by adding identity nodes, we have that the two structures are isomorphic, and due to Proposition 3.3.10, that the induced distributions of the two flows are the same. ■

This means that the results regarding structures with edges only between nodes apply to all flows with valid structures, which is an easy consequence of *Proposition* 3.3.13 and the function Λ.

Turning to the second task we wanted to investigate, which is for what structures which gives a triangular structure. As this leads to fast evaluation of

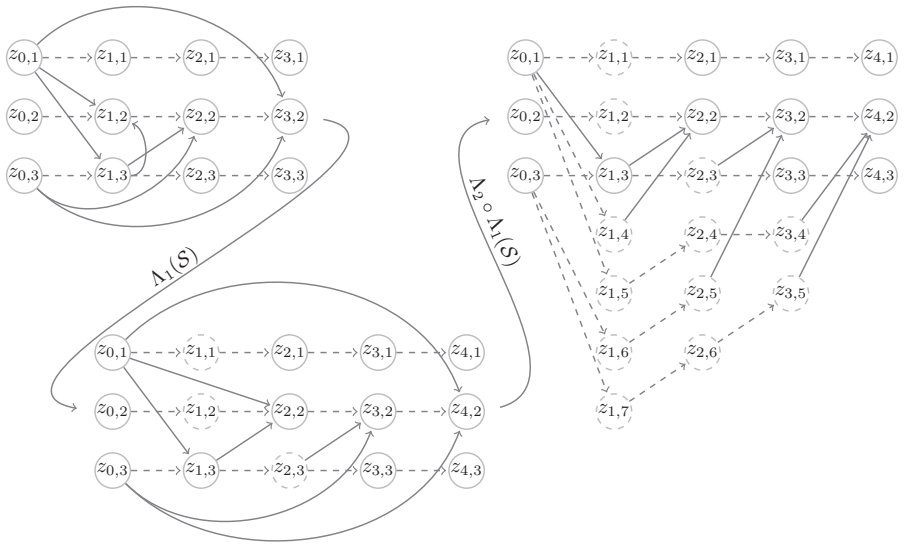


Figure 3.3: Starting with the structure top left, $\mathcal{S}$, we apply Λ_1, and achieve the graph below. Notice how the edge from $z_{1,3}$ to $z_{1,2}$, have been shifted to an edge from layer 1 to layer 2. Applying Λ_2 gives us the graph top right, which extends the graph to use the extra identity nodes as storage. The resulting structure is local with edges from previous layer to next.

density, generally, and is therefore very important when creating flows that are feasible to train and compute. We first define a set of structures which we proof induces a triangular Jacobian, before we discuss a path to find the broadest class of structures, which means any structure outside of this set must give a non-triangular Jacobian.

Definition 3.3.14. Let $\mathcal{S}$ be a valid flow-structure accompanying a normalizing flow $(\mathcal{Q}, f)$, which is local and have D nodes in each layer. The structure is a *triangular flow-structure* if and only if there exists no crossing paths.

Theorem 3.3.15. *Let $\mathcal{S}$ be a triangular flow-structure accompanying a normalizing flow $(\mathcal{Q}, f)$. Then the density induced by the flow is equal to*

$$q(\boldsymbol{x}) = q_{\boldsymbol{z}_0}(f^{-1}(\boldsymbol{x})) \prod_{(t,d)\in\mathcal{T}\times\mathcal{D}} \left| \frac{\partial}{\partial z_{t-1,d}} f_{t,d}^{-1}(z_{t-1,d}) \right|, \tag{3.4}$$

Proof. Let, for an arbitrary time step $t > 0$,

$$\mathcal{Z}_{t-1:t} = \{(t', d') \colon t' \in \{t-1, t\} \text{ and } d' \in \mathcal{D}\}.$$

As $\mathcal{S}$ is a triangular flow-structure, we know that for any time step $t > 0$ we have the following. We start by showing that there must exist a pair (t, d) where $(t-1, d) = \mathcal{Z}_{t-1:t} \cap \mathcal{S}(t, d)$, i.e., $\mathcal{S}_{ext}(t, d) = \emptyset$. Firstly, the only nodes in the graph $\mathcal{S}$ with edge to (t, d) are in $\mathcal{Z}_{t-1:t}$ as the structure is local by assumption. In addition, as there are only D nodes in each time step t by assumption, this

means $(t-1,d) \in \mathcal{S}_{int}(t,d)$ by the definition of flow-structures (Definition 3.3.1) and predecessor (Definition 3.3.2). Furthermore, for any node (t,d), every node with an edge from the same layer (t,d') to (t,d), cannot have an edge from (t,d) to (t,d'), as this would create a cycle and hence the structure would not be valid, which it is by assumption. This allows us to consider only structures with edges from $t-1$ to t, as for any node (t,d) with edge from (t,d'), a transformation

$$f_{t,d}(z_{t-1,d}, z_{t,d'}, \mathcal{S}_{ext}(t,d) \setminus z_{t,d'})$$

can be rewritten as

$$f_{t,d}(z_{t-1,d}, f_{t,d'}(z_{t-1,d'}), \mathcal{S}_{ext}(t,d) \setminus z_{t,d'})$$

hence we only need consider only structures with edges from $t-1$ to t. Now assume that there exists no node (t,d) such that $\mathcal{S}_{ext}(t,d) = \emptyset$. Then, there must exist at least one pair of nodes, $(t-1,d')$ and $(t-1,d)$, where there exists an edge from $((t-1,d),(t,d'))$ and $((t-1,d'),(t,d))$, which means we have a crossing path. This is not possible by assumption. We can conclude that there exists at least one node (t,d) where $\mathcal{S}_{ext}(t,d) = \emptyset$. Then the corresponding row to (t,d) of the Jacobian of $f_t = (f_{t,1}, \dots f_{t,D})$ contains one nonzero value, namely $\frac{\partial f_{t,d}}{z_{t-1,d}}$.

We can then state that there exists at least one node (t,d') which can at most only have edges from $(t-1,d)$ and (t,d), with equivalent argument as above. The row corresponding to (t,d') of the Jacobian of $f_t = (f_{t,1}, \dots f_{t,D})$ contains at most two nonzero values (as we can rewrite the contribution of (t,d) to $f_{t,d}(z_{t-1,d})$, following the previous argument), where one of them is $\frac{\partial f_{t,d'}}{z_{t,d'}}$. Inductively, we can then claim that the Jacobian of f_t can be written as the product of an permutation matrix P and a triangular matrix, with the diagonal containing $\frac{\partial f_{t,d}}{z_{t-1,d}}$. The corresponding determinant is then the multiplicative trace, and as the permutation matrix simply changes the sign, which does not matter as we are interested in the absolute value, we conclude that the determinant of the Jacobian of f_t is

$$\prod_{d \in \mathcal{D}} \left| \frac{\partial}{\partial z_{t-1,d}} f_{t,d}(z_{t-1,d}) \right|.$$

Using that the inverse of the product is equal to

$$\prod_{d \in \mathcal{D}} \left| \frac{\partial}{\partial z_{t-1,d}} f_{t,d}^{-1}(z_{t,d}) \right|,$$

and that we have D nodes in each layer of the structure, as well as the product of triangular matrices is closed, implies by Definition 3.2.5 that Equation (3.4) holds. ■

Definition 3.3.16. Let $\mathcal{S}$ be a flow-structure accompanying a normalizing flow $(\mathcal{Q}, f)$, which is local and have D nodes in each layer. The structure is a *triangular flow-structure* if and only if there exists an flow-isomorphic structure $\mathcal{S}'$ which transform one variable per time step t.

The equivalence follows by applying Λ to the structure with local and D nodes in each layer, and contains no crossing paths.

The foundation we have laid out with flow-isomorphism, Proposition 3.3.10, and the results given with Λ, can also give insight into the requirements to ensure a triangular Jacobian for all DAGs that is also allowing for different dimensions through the structure and edges spanning multiple layers. There are much more details, as we can for instance not rely on separating the flow into t steps, as we cannot split the determinant into product of determinant Jacobians, as the Jacobians may not have equal dimension. We suspect that the class of structures that allows for fast computation of the determinant of the Jacobian, consists of structures that do not contain too many paths from node (t, d) to node (t', d'), as well as the alternative definition Definition 3.3.16, which will rely on our function Λ as well. For now, we leave this as future research

Ending this part with a restatement of the definition of normalizing flow, by also including the structure. This is an equivalent definition to Definition 3.2.5 as the structure was then implicitly defined by f, but we now also want to express the importance of structures.

Definition 3.3.17. Let Q be a probability space, $f_{t,d}$ be piecewise C^1-diffeomorphisms, and $\mathcal{S}$ be a valid flow-structure. Then a *normalizing flow* is defined by the 3-tuple $(Q, \mathcal{S}, f)$.

Autoregressive Structure and Coupling Structure

We introduce three structures that are typically the ones used in the literature. The first two structures are referred to as autoregressive structures, and allow for a triangular Jacobian and every dimension is transformed (i.e., no identity nodes). The structures have also been used in some of the more popular and well-studied flows (Papamakarios et al. 2017, Kingma, Salimans et al. 2016, Huang, Krueger et al. 2018).

Definition 3.3.18. Let $(\mathcal{Q}, \mathcal{S}, f)$ be a NF. $\mathcal{S}$ is an *autoregressive flow-structure* (AR flow-structure) if for every $t \in \mathcal{T}$ and $d \in \mathcal{D}$ we have

$$\mathcal{S}_{ext}(t, d) = \{(t, i) \colon i < d \in \mathcal{D}\}.$$

An *inverse autoregressive flow-structure* (IAR flow-structure) is a structure $\mathcal{S}$ where for every $t \in \mathcal{T}$ and $d \in \mathcal{D}$ we have

$$\mathcal{S}_{ext}(t, d) = \{(t-1, i) \colon i < d \in \mathcal{D}\}.$$

In Figure 3.4 we give example of AR and IAR structures represented as graphs.

Both structures above are triangular structures which gives a triangular Jacobian, and one can also motivate such a structure by the fact that one can always factorise any distribution into a similar form, e.g. $p(x_1, x_2, x_3) = p(x_3 \mid x_2, x_1)p(x_2 \mid x_1)p(x_1)$. This fact makes AR/IAR useful in regard to proving the flexibility of flows later on. They are also flow-isomorphic, which can easily be shown and Figure 3.2 illustrates. However, the two structures differ and the distinction is quite important. When computing the flow

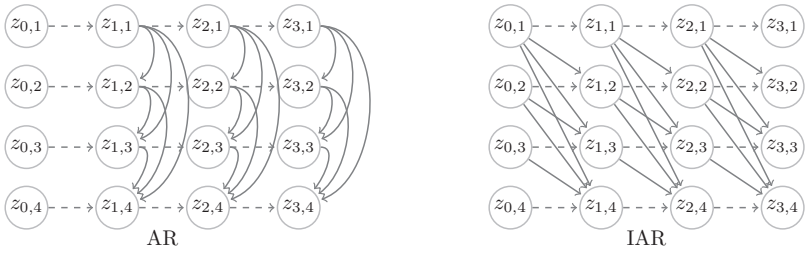


Figure 3.4: Example of an autoregressive flow-structure (left) and an inverse autoregressive flow-structure (right).

forward, all variables needed to compute $\boldsymbol{z}_t$ are already computed when using IAR, while the opposite is true when computing backwards, and vice versa with AR. Hence, one is preferred above the other based on whether forward or backward of the flow is important. In a variational inference case we optimise the flow by sampling latent variables which makes computing f is important, hence IAR is preferred. While in density estimation we optimise by creating a flow that transform observations from target distribution to the base distribution and therefore f^{-1} is needed, hence AR is preferred. It is also worth noting that IAR and AR both uses every variable from $\boldsymbol{z}_{t-1}$ and z_t respectively, and still induces a triangular Jacobian.

There is no problem in using the structures above with different ordering for each transformation t. This can be quite useful in "sharing" information more efficiently, and with more simple transformations it is necessary to achieve a flexible flow (Kingma, Salimans et al. 2016; Papamakarios et al. 2017). One can easily justify the different ordering by adding a transformation between each t, which applies a permutation matrix P_t to $\boldsymbol{z}_{t-1}$. We know that $\det(P_t) = 1$ and is easily invertible, hence, as long as the permutation is fixed, we can use the flow without any added problems. A more complete definition can then be written as:

Definition 3.3.19. Let $(\mathcal{Q}, \mathcal{S}, f)$ be a NF. If there exists a permutation $\pi_t \colon \mathcal{D} \to \mathcal{D}$ for all $t \in \mathcal{T}$ such that

$$\mathcal{S}_{ext}(t, \pi_t(d)) = \{(t, i) \colon i \in \mathcal{D} \text{ and } \pi_t(i) < \pi_t(d)\},$$

then $\mathcal{S}$ is an *autoregressive flow-structure* (AR flow-structure). Equivalently with $\mathcal{S}$ being an *inverse autoregressive flow-structure* (IAR flow-structure) with

$$\mathcal{S}_{ext}(t, \pi_t(d)) = \{(t-1, i) \colon i \in \mathcal{D} \text{ and } \pi_t(i) < \pi_t(d)\},$$

We have included graphs of IAR and AR with permutations in Figure 3.5. Unless specified, when we say a flow with a particular structure, e.g., IAR, we mean including structures with permutations. Whenever we talk about permutations of the flow, we refer to the permutation of the structure π.

When thinking about the difference between IAR and AR as mentioned previously, we have not resolved the issue when both direction of the flows are important to compute fast. It is not possible to vectorize IAR when going backwards, and vice versa with AR and forward. The compromise often

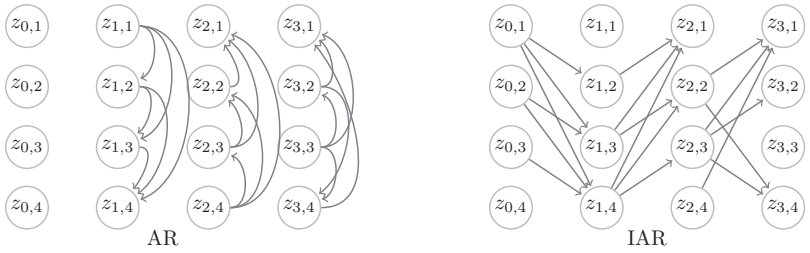


Figure 3.5: Example of AR (left) and IAR (right), with permutations. The permutations are: $\pi_1(i) = i$, $\pi_2 = \{(1,4),(2,3),(3,2),(4,1)\}$, and $\pi_3 = \{(1,4),(2,2),(3,1),(4,3)\}$. We have excluded interior edges for readability.

employed will be referred to as coupling flow-structures, and which we have given an example of in Figure 3.6.[3].

Definition 3.3.20. Let $(\mathcal{Q}, \mathcal{S}, f)$ be a NF. If there exists a permutation $\pi_t \colon \mathcal{D} \to \mathcal{D}$ for all $t \in \mathcal{T}$ such that

$$\mathcal{S}_{ext}(t, \pi_t(d)) = \begin{cases} \{(t-1, i) \colon i \in \mathcal{D} \text{ and } \pi_t(i) \leq \hat{D}\}, & \text{if } \pi_t(d) > \hat{D} \\ Id, & \text{otherwise,} \end{cases}$$

where $\hat{D} \in \mathcal{D}$, then $\mathcal{S}$ is a $\hat{D}$-*coupling flow-structure*.

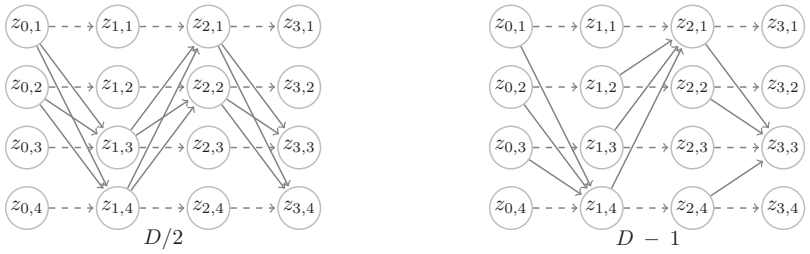


Figure 3.6: Two examples of coupling flow-structure. Left one with $\hat{D} = D/2$ and right one with $\hat{D} = D - 1$.

In short, a coupling structure divides the variables into two parts, one part of size $\hat{D}$ which is simply transformed using the identity function, and the rest is transformed using the $\hat{D}$ part as input. This makes it less efficient than AR/IAR regarding sharing information and transforming variables, as it requires several transformations t to change all variables (that is, excluding identity transformations). However, as we can easily invert the identity transformation, means we can easily vectorize the flow both ways, and hence is more efficient than AR/IAR in that sense.

Finishing up this part by introducing quickly a structure that will be useful later on.

[3]The use of coupling structures were actually introduced before IAR and AR, but as it exploits less information for each t, means we consider it as a compromise.

Definition 3.3.21. Let $(\mathcal{Q}, \mathcal{S}, f)$ be a normalizing flow. The structure $\mathcal{S}$ is a *fully connected* structure if for every $t \in \mathcal{T}$ and $d \in \mathcal{D}$,

$$\mathcal{S}(z_{t,d}) = \{z_{t-1,i} \colon i \in \mathcal{D}\}.$$

We also refer to such structures as $\mathcal{S}_{full}$

This is the most powerful structure, when not considering edges in structures that go from layer t to t' and $|t - t'| > 1$, although it often puts more restraint on the transformation f to achieve fast inverse/density evaluation.

Improvement with Structures

An important question to ask is how different structures affect the flow. One way to approach the question is to look at the difference between the structures, e.g., IAR compared to AR and coupling. This have been studied extensively empirically, often tied together with the transformation. There are also some comparisons theoretically (Papamakarios et al. 2017), but for the most part it comes down to empirical results and practicalities already discussed, e.g., vectorization.

The second approach is to consider the permutations of the structures, π_t. The typical approach is to either use the identity permutation, randomly permute, or reverse, i.e., map one to D, 2 to $D - 1$ and so forth. It may be reasonable to think that, at least between the latter two, there is not much difference unless you have very specific information about the correlation in a dataset. As long as you have a mixing between the variables, one achieves adequately good results relatively. This does not, however, hold true in general, and there are in particular two papers that contradict the notion, where focusing on the structure gives better performance at the task considered in the two papers. We give a recount of their work here.

Kirichenko et al. 2020 points out that, using $D/2$-coupling structures on images, normalizing flows often fail in the task of detecting out-of-distribution data. They propose that the inductive biases in flows are the structure. Hence, picking good structure, even a priori, can have benefits. They also show that a typical split in the coupling structure with every alternating pixel, leads the flow to learn more of the patterns in the image and not the semantic information. Changing the permutation to split the image pixels in the top half and bottom half, i.e., splitting the image horizontally in the middle, results in a significant improvement of out-of-distribution detection.

Another case that highlights the importance of permutation in the structure π_t is done by learning the structure from the observation one has, i.e. training data. One way to use the data as guidance is to add permutation matrices to the flow, and let the matrices be parameterized. However, learning a permutation matrix is quite difficult computationally due to its combinatorial nature. Another approach that both lends itself to continuous optimisation, and not necessarily a particular structure (simply that it is triangular), is to employ NO TEARS (Zheng et al. 2018). Essentially, one can introduce an adjacency matrix A, where the nodes correspond to $\boldsymbol{x}$. Loosening up on the binary part of A, and allow it to take on any real value (turning it more into a structural equation model (SEM)). One can then apply A as a mask, and by minimising a

loss function over the flow and A such that

$$\operatorname{tr}\left[e^{A \odot A}\right] - D = 0,$$

where e is the matrix exponential. Minimising the loss function with the added constraint can be solved relatively efficiently with the augmented Lagrangian method. Without going into too much detail, considering the binary adjacent matrix A, one have that $\operatorname{tr} A^k$, where $k \in \mathbb{N}$, specifies the number of k-closed walks in the graph. As a DAG does not contain any cycles, one must enforce the trace to be zero for any $k > 0$. This can then be developed to the constraint given above, and also holds for $A \in \mathbb{R}^{D \times D}$. Wehenkel et al. 2021 explores this method with normalizing flows, using a similar constraint (Yu et al. 2019), and re-binarize the matrix afterwards (using the Gumbel-Softmax trick (Jang et al. 2017)). Using the re-binarized matrix as a mask, they did experiments with *conditioner transformations* (see Section 3.4), although limited to $T = 1$, it showed promising results that confirm the potential importance of structure.

Conclusion

We have now introduced structures and shown some properties necessary for efficient computation of both inversion and the Jacobian determinant. By decoupling the flow into transformations and structures, it allows us to think more broadly on different structures and how they differ, without muddling it with the efficiency or flexibility of the transformations. It seems reasonable to postulate that having structures which resemble the correlation in target distribution better, can give significant differences, and learning such structures is an interesting research path, including integrating Wehenkel et al. 2021 work with CONNs.

3.4 Conditioner

In many flow architectures, and in particular the ones we shall study, the transformation of $z_{t-1,d}$ to $z_{t,d}$ is done by first computing parameters of the transformation, where the parameters are computed using the variables $\mathcal{S}_{ext}(t, d)$, and then applying these parameters to $z_{t-1,d}$. In Example 3.2.6, we first compute a and b, and then transform $z_{t,d} = a \cdot z_{t-1,d} + b$. The function that computes the parameters is referred to as a *conditioner*, as we are in a certain sense conditioning $z_{t,d}$ on $\mathcal{S}_{ext}(t, d)$.

Definition 3.4.1. Let $(Q, \mathcal{S}, f)$ be a NF, where each $f_{t,d}$ is parameterized by $p_{t,d}$ parameters. A *conditioner* is any function $\mathcal{H}$ where

$$\mathcal{H}_{t,d} \colon \mathcal{S}_{ext}(t, d) \to \mathbb{R}^{p_{t,d}}$$

The transformation can then be written as $f_{t,d}(z_{t-1,d}, \boldsymbol{\theta}_{t,d})$, where $\boldsymbol{\theta}_{t,d} = \mathcal{H}_{t,d}(\mathcal{S}_{ext}(t, d))$.

The positive side of such a separation between $\mathcal{S}_{int}(t, d)$ and $\mathcal{S}_{ext}(t, d)$ is that we can potentially use quite simple transformations, and rather include the complexity through $\mathcal{H}$. In particular, with triangular structures, the transformations given the parameters can be easy to invert and evaluate its derivative, while the conditioner can be as complex as one like with no requirement of its inverse and derivatives. This thanks to Theorem 3.3.15.

Neural Networks as Conditioner

We can observe that $\mathcal{H}_{t,d}$ does not have to be invertible or allow for easy computation of the Jacobian when using triangular structures. This is due to the fact that such structures contain at least one variable $z_{t,d}$ which is transformed through identity or constant parameters. Finding the inverse $z_{t-1,d}$, means any variable that only depends $z_{t-1,d}$ can be computed, as we can find its parameters through the conditioner and input $z_{t-1,d}$. This can then easily be iterated over, and eventually invert $\boldsymbol{z}_t$ without every finding the inverse of the conditioner. As the structure is triangular, means the determinant of the Jacobian does not depend on the conditioner, and hence we can have arbitrary complexity in the conditioner.

Although the conditioner can be any function, it is through the conditioner we introduce deep learning and its flexibility. There are a few ways to do this depending on what type of structure. If one split one part as input to neural network and another part to be transformed, such as coupling structures, we can simply use a feedforward network. In any other case, such as IAR/AR, we have to do things differently. The naive way is to model each $\mathcal{H}_{t,d}$ as its own neural network, as it leads to both an immense amount of parameters, but also a sequential problem, as we need D neural networks for every transformation step t. Hence, even when the structure allows for vectorization, we still end up with sequential computations as we have D networks for every step t.

Another approach which reduces the number of weights required, is to use a recurrent neural network (RNN) (Schmidhuber 2015). The gist of a RNN in our context is as follows: to initiate a state $\boldsymbol{s}_0 \in \mathbb{R}^k$ for some $k \in \mathbb{N}$, and input said state to a feedforward network. The network outputs both the parameters we need to transform $z_{t-1,1}$ and a new state $\boldsymbol{s}_1 \in \mathbb{R}^k$. Next we input both $z_{t-1,1}$ and state $\boldsymbol{s}_1$. The states is carrying the information from $z_{t-1,1:d-2}$. This continues until we have parameters to transform $z_{t-1,D}$ and $\boldsymbol{s}_D$. Note that the state for step d is computed only using $z_{t-1,1:d-1}$, hence we retain the autoregressive structure and typically the RNN will perform as well as D neural networks, but with fewer weights needed. RNN can also be summarised as

$$\boldsymbol{\theta}_d, \boldsymbol{s}_d = \Psi_d(z_{t-1,d-1}, \boldsymbol{s}_{d-1}), \text{ for } d \in \mathcal{D},$$

where $\boldsymbol{\theta}_d$ are the parameters used to transform $z_{t-1,d}$. The solution using RNNs has been studied (Oliva et al. 2018), but it tends to be slow due to the sequential nature, i.e., we have to calculate the first state and parameters, then the second one, and so forth.

The third option, which has become the preferred one, is to use MADE/-CONN. MADE have been used for AR/IAR, but with the introduction of CONN, means we can speedily compute $\mathcal{H}$ for any triangular structure. Obviously, this does not solve for the vectorization problem, which we illustrate in Figure 3.7, where we indicate in what order each node is computed. For example, using CONN with a IAR structure backwards, we simply need to pass a 0-vector to the network to acquire the first set of parameters, then pass in $z_{t-1,1}$ to acquire parameters necessary to invert $z_{t-1,2}$, and so forth, simply ignoring the rest of the output. The use of MADE/CONN does at least fix partially the number of weights and allows for vectorization wherever the structure allows for it.

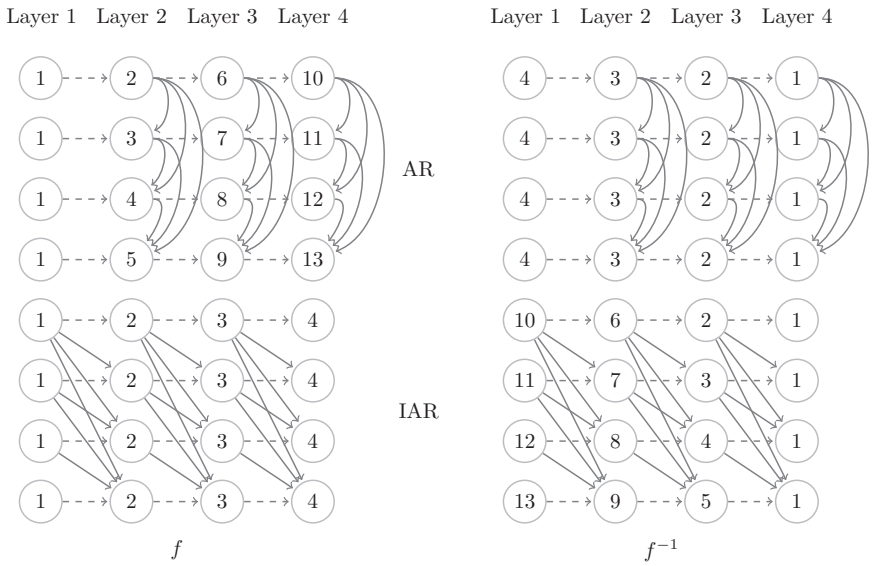


Figure 3.7: The numbers indicate in what order the associated values to the structures can be computed, where equal number implies vectorization/parallel computed. Forward computed on the left and inverse computed on the right, AR-structure above and IAR-structure below. For example, once we have $\boldsymbol{z}_4$, we can in AR compute the parameters needed to compute layer 3.

3.5 Transformations

The second part of a flow is the transformation. It is typically the source of, together with the conditioner, complexity in the flow. This is also what much of the literature focuses on, and many transformations have been proposed over the years, coupled with one of the three structures we defined above. In this section we are covering a few of the most prominent transformations, where we first take a look at the non-conditioner transformations, i.e., the ones that do not employ a conditioner, and then concentrate on the ones using a conditioner. There are too many transformations, such that it would be unproductive to introduce them all, hence we group them in different categories with the most prominent examples of each group. This is to both give the reader an overview of the field and for comparison later on.

Non-conditioner Transformations

We start off with transformations that do not require a conditioner. Typically, these do have quite a few more parameters in the actual transformations (in a conditioner transformation, we offset the heavy number of parameters in the conditioner instead). This also means that the structures often are fully connected structures and implicitly chosen through the parameter space.

Linear Transformations

The first transformation we encounter is the linear transformation. For it to fulfil the requirements of a flow Definition 3.2.5, we constrict the choice of linear transformations, and we therefore remind the reader of the *general linear group* defined as

$$GL(D) = \{A\colon A \in \mathbb{R}^{D\times D} \text{ and } A^{-1} \text{ exists}\}$$

and the operation is matrix multiplication, and the *general orthogonal group* defined as

$$O(D) = \{A\colon A \in GL(D) \text{ and } A'A = AA' = I_D\}.$$

Definition 3.5.1. Let $(\mathcal{Q}, \mathcal{S}, f)$, with $\mathcal{S} \subseteq \mathcal{S}_{full}$. A *linear transformation* is defined as

$$\boldsymbol{z}_t = A\boldsymbol{z}_{t-1}, \quad A \in GL(D).$$

An *orthogonal-linear transformation*, or simply *orthogonal transformation*, is defined as

$$\boldsymbol{z}_t = A\boldsymbol{z}_{t-1}, \quad A \in O(D).$$

The first to consider regarding linear transformations, is that they are quite limited in their capacity. This stems from both its linearity and the fact that the matrices are constant, e.g., the matrix does not change based on what value the variables are—as they would with a conditioner—. Its inadequacy can easily be demonstrated by letting the base distribution be Gaussian, with mean μ and covariance Σ, then we have $A = A_T A_{T-1} \cdots A_1$

$$\boldsymbol{x} = \boldsymbol{z}_T \sim \mathcal{N}(A\mu, A\Sigma A').$$

Allowing for translation does not improve immensely on the transformations capacity. This does not necessarily mean that linear transformations are useless, but flows employing only linear transformations are limited. However, one can imagine mixing transformations, and even think of it as a way to find a good permutation π_t of the structure, as we can rotate the input through such a transformation (another way to look at it is that every single permutation matrix is a subset of the parameter space), and such tactics has given interesting results (Kingma and Dhariwal 2018). Also, linear transformations—depending on its parametrization—offers few parameters to train and can be efficient in computing inverse/the Jacobian determinant.

The other point, aside from capacity, is the parametrization of the matrices. There are a few ways to tackle this, but a straightforward way of optimising over GL is not possible. One approach is to limit the matrices to triangular, making sure that the diagonal contains no zeros. This has a fast evaluation of Jacobian determinant and $\mathcal{O}(D^2)$ inversion. One can also interpret this through Example 3.2.6, where a being a constant and b being a weighted sum of the previous $1 : d-1$ variables, i.e., a very stringent and weak conditioner.

Another option is to use orthogonal transformations. Clearly, a fast inversion by transposing and its Jacobian determinant is simply ± 1 as

$$1 = \Pi_{d=1}^{D} I_{d,d} = \det(I) = \det(AA') = \det(A)\det(A') = (\det(A))^2,$$

using the fact that the sign does not matter as we are interested in the absolute value means no computation is needed. To optimise the flow to approximate some target distribution using orthogonal matrices, we need to choose what parametrization to use. Several has been proposed, such as the Householder matrices (Tomczak et al. 2016) and Cayley transform (Golinski et al. 2019). Without going into too much details, the former spans the complete $O(D)$, but only by using D matrix multiplications, which in turns makes it hard to scale and it introduces potential instabilities. As an upside, one still have orthogonal matrices with $K < D$ multiplications, hence there is a trade-off between performance and computational cost by choosing K. On the other side, Cayley transform only spans the special orthogonal group ($SO(D)$), and when computing the matrix needs one multiplication and one inversion, i.e. $A = (I + A_s)(I - A_s)^{-1}$, where A_s is a skew-symmetric matrix. This adds substantially to the computation during training, with $\mathcal{O}(D^3)$, while Householder matrices uses $\mathcal{O}(K \times D)$. However, as pointed out by Golinski et al. 2019, using Householder matrices in flows can both induce local minimas[4] and also numerical instabilities due to the multiplication of many matrices, e.g., exploding gradients.

Although there are some other approaches such as LU-decomposition, for the sake of brevity, we end it here.

Residual Transformations

Similar to residual connections introduced in Chapter 2, we have residual transformations. They follow the same type of principal as residual connections, that is, the transformation consists of $\boldsymbol{z}_{t-1}$ adding to an output of another function $g(\boldsymbol{z}_{t-1})$.

Definition 3.5.2. Let $(\mathcal{Q}, \mathcal{S}, f)$ be a flow with $\mathcal{S} \subseteq \mathcal{S}_{full}$. A *residual transformation* is defined through a residual function g, which is a C^1-diffeomorphism, and the transformation is of the form

$$\boldsymbol{z}_t = f(\boldsymbol{z}_{t-1}) = \boldsymbol{z}_{t-1} + g(\boldsymbol{z}_{t-1}).$$

In theory, this class of transformations can use every other type of transformation we introduce in this section, as we simply let g be the transformation instead of f. However, there are a few transformations that rely on the residual form given above and we shall present two of them here.

In Chapter 2 g would be a neural network, but this is not necessarily as straightforward to employ here, as we need to both ensure invertibility and preferably a method to invert it as well. In general, if one can make g be a contraction, i.e., Lipschitz constant s less than 1, that is,

$$d_g(g(\boldsymbol{z}_{t-1}), g(\hat{\boldsymbol{z}}_{t-1})) \leq s d_z(\boldsymbol{z}_{t-1}, \hat{\boldsymbol{z}}_{t-1}),$$

[4]This is relevant more so when orthogonal matrices are used in transformations with non-linear elements, such as Sylvester transformation (introduced later).

for all $\boldsymbol{z}_{t-1}, \hat{\boldsymbol{z}}_{t-1} \in \mathcal{Z}_{t-1}$ and metrics $d_z \colon \mathcal{Z}_{t-1} \to \mathbb{R}$ and $d_g \colon \mathcal{Z}_t \to \mathbb{R}$. This implies two things, the residual transformation $\boldsymbol{z}_{t-1} + g(\boldsymbol{z}_{t-1})$ is a contraction, and more importantly $f^{-1}(\boldsymbol{z}_t) = \boldsymbol{z}_t - g(\boldsymbol{z}*_{t-1})$ is a contraction, where $z*_{t-1} \in \mathcal{Z}_{t-1}$ is an arbitrary chosen value. By utilising *Banach's fixed point theorem* (Lindstrøm 2017, Ch. 3, p. 61)—assuming we are working with complete metric space $(\mathcal{Z}_{t-1}, d_z)$ and $\mathcal{Z}_{t-1} = \mathcal{Z}_t$—then f^{-1} has a unique fix point $\tilde{\boldsymbol{z}}_{t-1}$ and regardless of starting point $\boldsymbol{z}*_{t-1}$, iterating

$$\boldsymbol{z}*_{t-1}^{(k)} = f^{-1}\left(\boldsymbol{z}*_{t-1}^{(k-1)}\right),$$

where $k > 0$, converges towards $\tilde{\boldsymbol{z}}_{t-1}$. Also,

$$\tilde{\boldsymbol{z}}_{t-1} = f^{-1}(\tilde{\boldsymbol{z}}_{t-1}) = \boldsymbol{z}_t - g(\tilde{\boldsymbol{z}}_{t-1}),$$

which implies by rearranging

$$\boldsymbol{z}_t = f(\tilde{\boldsymbol{z}}_{t-1}) = f(\boldsymbol{z}_{t-1}),$$

hence f is invertible due to the uniqueness of the fixed point. This also means finding the inverse is equivalent to finding the fixed point and can be done through iterating from a starting point $\boldsymbol{z}*_{t-1}$. Through proving Banach's fixed point theorem, one also discovers the convergence rate through iterations, namely

$$d_z(\boldsymbol{z}*_{t-1}^{(k)}, \tilde{\boldsymbol{z}}_{t-1}) \leq \frac{s^k}{1-s} d_z(\boldsymbol{z}*_{t-1}^{(0)}, \boldsymbol{z}*_{t-1}^{(1)}).$$

This means that if g contracts a lot, we can find the inverse exponentially faster, and vice versa.

This allows us for instance to use a subset of neural networks as g, namely the ones where both the linear layers and each activation function have Lipschitz constant less than or equal to one—where at least one of them has less than one as Lipschitz constant—as contractions are closed under composition. Which means, contracting linear layers and activation functions with aforementioned Lipschitz constant, makes the neural network contractive.

Invertible Residual Networks (Behrmann et al. 2019) deploy such networks, where the weights W_l are constrained to be less than one under the spectral norm, and activation functions are ReLU, ELU, Leaky ReLU, etc. This guarantees that the transformation g is Lipschitz continuous with Lipschitz constant $s < 1$. Even with such constraint on g, it still performs well empirically. However, there are noteworthy drawbacks. The first is the nonanalytical inverse, as we estimate it with a finite number of iterations. The second is concerning neural networks when used as g, and it is high cost when computing the Jacobian determinant. Behrmann et al. 2019 proposes to alleviate the issue stochastically, and ultimately ends up with an unbiased estimate (biased in the original paper, but improved upon by Chen et al. 2019). In addition to this, the computation of the Jacobian relies upon several runs of backpropogation. Hence, both inverse and the density are estimated in the end.

There are several other residual transformations that do not rely on Banach's theorem, such as transformations used in planar flows, radial flows, and Sylvester

flows (Berg et al. 2018; Rezende et al. 2015). Sticking to the last introduced, a *Sylvester transformation* can be written as

$$\boldsymbol{z}_t = \boldsymbol{z}_{t-1} + QR\gamma\left(\tilde{R}Q'\boldsymbol{z}_{t-1} + \boldsymbol{b}\right), \tag{3.5}$$

where $R, \tilde{R}$ are upper triangular $\mathbb{R}^{M\times M}$ matrices, $Q \in \mathbb{R}^{D\times M}$ is an orthonormal matrix, and γ is typically an invertible non-linearity, e.g. leaky ReLU. Here, M is a hyperparameter $M \leq D$. The determinant, using the Sylvester's determinant identity, one can compute the Jacobian determinant in $\mathcal{O}(M)$. Hence, shrinking the parameter space allows for faster computation of the determinant, and vice versa, but in all practicalities it ought not matter too much (unless D is very big). However, to ensure invertibility and nonzero determinant, some constraints are needed.

Theorem 3.5.3 (Berg et al. 2018)**.** *Let γ be a smooth function with bounded positive derivatives. If $\tilde{R}$ is invertible, and the diagonal elements*

$$R_{m,m}\tilde{R}_{m,m} > -\frac{1}{\|\gamma'\|_\infty},$$

then the transformation Equation (3.5) *is invertible.*

The pros of Sylvester transformations are that they allow for full structure, added complexity to linear transformations through γ, and fast evaluation of the Jacobian determinant. The cons are problems with orthonormal matrices and training (as previously discussed), fulfilling the constraints above, which ultimately restricts the parameter space, and no easy method to obtain invertibility (we shall discuss later when finding the inverse and lack thereof may not be a hindrance). An additional point of interest is the fact that the transformation is one hidden layer in a neural network, with a restriction on the weights, similar to previous residual transformations. However, compared to the contractive network, it is bounded by dimension D in number of nodes and constitutes only one hidden layer. Similar pros/cons analysis can be given to planar flows, radial flows, etc.

Conditioner Transformations

Moving onto transformations that employ conditioners. As a reminder; we parameterize the transformation of $z_{t-1,d}$, with the parameters being computed through a function, the conditioner $\mathcal{H}_{t,d}$, with input being $\{z_{i,j} : (i,j) \in \mathcal{S}_{ext}(z_{t,d})\}$. To allow for $\mathcal{H}$ to be as powerful as possible, with easily computable inverse/Jacobian determinant, we constrict ourselves to triangular structures. The Jacobian determinant becomes simply the transformations derivative, as shown in Theorem 3.3.15. If the transformations are easy to invert once we have the parameters, then inverting the flow is easy due to the fact that there must exist an ordering such that a variable $z_{t-1,d}$ can be found directly, and then we can iteratively find the others, as described in Section 3.4. This in turn makes even the simplest of transformation very powerful, as one combine flexible models for $\mathcal{H}$ with permutations in triangular structures. Further explorations of the capacity are presented in the next section.

We start out by extending the definition of flows to include conditioners.

Definition 3.5.4. For every $t \in \mathcal{T}$ and $d \in \mathcal{D}$, let $f_{t,d}$ be a transformation parameterized by a conditioner $\mathcal{H}_{t,d}$. Then a *conditioner normalizing flow* is a flow with such transformation f, and is denoted by the 4-tuple $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$.

As with non-conditioner transformations, there have been a myriad of different transformations proposed in the literature, and we therefore try to group them sensibly and present a varied selection of the most prominent and relevant to our work. We start off with both one of the earliest and simplest form of transformations.

Affine

One of the first transformations that was introduced was a simple affine transformation, componentwise. We introduce the transformation assuming an arbitrary $t \in \mathcal{T}$ and $d \in \mathcal{D}$.

Definition 3.5.5. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a normalizing flow. An *affine transformation* has the form

$$z_{t,d} = f_{t,d}(z_{t-1,d}) = a_{t,d}\, z_{t-1,d} + b_{t,d},$$

with $(a_{t,d}, b_{t,d}) = \mathcal{H}(\mathcal{S}_{ext}(t, d))$, where $a_{t,d} > 0$.

It started out by introducing it strictly through translation, i.e. no $a_{t,d}$ (Dinh, Krueger et al. 2015), which we refer to as *additive transformation*, and is obviously volume preserving. The full affine transformation was then introduced, first using $D/2$-coupling structure (Dinh, Sohl-Dickstein et al. 2017), and then also with IAR and AR structures (Kingma, Salimans et al. 2016; Papamakarios et al. 2017). Due to few parameters used in the actual transformation, makes the transformation incredible easy to invert and find the analytical derivatives, which again have made the transformation very popular.

At the surface level, it may seem as a weak form of a linear transformation, i.e., a matrix with diagonal elements only, but it is certainly not the case. The conditioner, given a flexible one such as neural networks, can quickly give rise to a very nonlinear flow. To understand this, it is enough to point out that $z_{0,1} \neq z'_{0,1}$ means $z_{0,2}$ and $z'_{0,2}$ can be transformed in drastically different ways. They may swap ordering or even become equal even when starting of as unequal. However, in a triangular flow-structure, there is at least one dimension which is transformed through constants a, b, and will of course change linearly. The glaring weakness can easily be resolved by allowing for several transformations T, and permutate the structures.

Although the model space corresponding to flows with affine transformations and neural networks as a conditioner function, using permutations and multiple transformations, is very large (more on that next section), it still has limitations due to its simplicity. It has therefore been natural to suggest different transformations, to improve expressiveness, and which does not rely on permutations nor on multiple transformations.

Neural Network Transformation

Similar to what was done in residual transformations, neural networks have been proposed here as well. That is, each variable $z_{t,d}$ is transformed through a

one dimensional input/output neural network. The weights in said network are computed by the conditioner $\mathcal{H}$. To avoid any confusion, we emphasise the fact that each variable $z_{t,d}$ is transformed through a one input one output neural network, where the weights are computed through the conditioner, which itself can be another neural network that takes as input $\mathcal{S}_{ext}(t,d)$. Before we move on, we remind the reader of the notation $\mathcal{NN}_{[L,\hat{D},\gamma]}$, means the space of neural networks with L hidden layers, $\hat{D}$ dimensions in each hidden layer, and γ as activation function. We then have $\Psi \in \mathcal{NN}_{[L,\hat{D},\gamma]}$ is a neural network with specific weights set.

As with residual transformations, the weights and activation functions we use to transform $z_{t,d}$ must be restricted to guarantee invertibility. One way to assure this was done by Huang, Krueger et al. 2018, where they constrain the model space that transforms $z_{t,d}$ to networks with bijective activation functions and nonnegative weights. We denote networks with an added plus sign, that is, $\mathcal{NN}^{+}_{[L,\hat{D},\gamma]}$ and Ψ^{+}.

Definition 3.5.6. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a normalizing flow. A *neural network transformation* has the form

$$z_{t,d} = f_{t,d}(z_{t-1,d}) = \Psi^{+}(z_{t-1,d}),$$

with the weights of the network$\{W_l, \boldsymbol{b}_l : l \in \{1, 2, \ldots, L\}\} = \mathcal{H}(\mathcal{S}_{ext}(t,d))$, where Ψ^{+} has L layers.

Alternatives have been proposed to eliminate the restrictions we put on Ψ^{+}. Unconstrained Monotonic Neural Networks (Wehenkel et al. 2019) restricts their neural networks only by forcing the output to be positive, and the transformation is then written as

$$z_{t,d} = \int_0^{z_{t-1,d}} \Psi(y)dy + \beta,$$

where $\Psi(y)$ is positive and constant β. The transformation is done through numeric integration, with added tricks to speed up backpropogation. For brevities sake, we focus on Ψ^{+}, with much of the analysis applying to UMNN as well.

Inverting such a transformation is not straightforward, and the preferred way is to do a form of bisection search, which means it both is an estimate and is computationally heavier than analytically inverse transformations such as the affine transformation. Depending on the size of the network, it may be computationally heavy to run backpropagation and find the derivative as well. As the number of weights increases, generally, the larger the neural networks used to compute $\mathcal{H}$ must be. Otherwise, one inevitably tries to preserve information between a low dimensional space and high dimensional one, using only a simple elementwise nonlinearity and an affine transformation, which becomes impossible as the output dimension increases. Hence, increasing the complexity of the transformation, increases the amount of trainable parameters in $\mathcal{H}$, which means the additional capability comes with the need for more computational power. On the other hand, such transformations are indeed quite flexible and performance wise have shown to do better than with affine transformations, and is certainly a viable option if one can bear the brunt of the added computational power.

Spline Transformations

The final class of transformations we include in this section are the ones using any form of splines. Roughly speaking, a one dimensional spline is defined by letting $\kappa_1 < \kappa_2 < \cdots < \kappa_{K+1} \in \mathbb{R}$ be knots, and $P_k \colon [\kappa_k, \kappa_{k+1}] \to \mathbb{R}$ be a polynomial with parameters $\boldsymbol{\alpha}_k$, where $k \in \{1, \ldots, K\}$. As we are applying splines to transform each dimension by itself, we are only interested in one dimensional ones and refer to it as simply splines. We allow for κ_1 and κ_{K+1} to be $-\infty$ and ∞ respectively.

Definition 3.5.7. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a normalizing flow. A *spline transformation* is any transformation on the form

$$z_{t,d} = f(z_{t-1,d}) = \begin{cases} P_1(z_{t-1,d}) & \text{if } \kappa_1 \leq z_{t-1,d} < \kappa_2 \\ P_2(z_{t-1,d}) & \text{if } \kappa_2 \leq z_{t-1,d} < \kappa_3 \\ \vdots & \\ P_K(z_{t-1,d}) & \text{if } \kappa_K \leq z_{t-1,d} \leq \kappa_{K+1} \end{cases}$$

with $\{\boldsymbol{\alpha}_k, \kappa_k, \kappa_{k+1} \colon k \in \{1, \ldots, K\}\} = \mathcal{H}_{t,d}(\mathcal{S}_{ext}(t, d))$.

Splines have been studied extensively, and it has naturally led to proposals of many different splines in the context of flows as well. The focus has been on monotonically increasing splines, although nothing prevents us from using other splines, but to do so, we have to spare extra memory and power to evaluate all terms in the new density as described by Definition 3.2.5. Another argument for using monotonic splines are when evaluating or inverting the transformation, one can do so in $\mathcal{O}(\log K)$ time through binary search.

In many ways, spline transformations can be squeezed in between affine transformations and neural network transformations, its inverse—depending on the spline chosen—is analytical, yet need binary search to find it (although binary search over discrete values). It also has the option of scaling the complexity of each transformation, similar to neural network transformations, by increasing K. The same dynamic with increasing complexity in the transformation, demand change in size of $\mathcal{H}$, is also applicable here, yet provides less dependency on T being larger as is the case with affine transformations.

Examples of monotonic splines proposed in the literature are linear splines (Müller et al. 2019), cubic splines (Durkan et al. 2019a) and rational quadratic splines (Durkan et al. 2019b). We give a brief insight to the latter, which is one of the most flexible ones researched so far. They let the tails outside of $[-B, B]$ be linear and create a polynomial between $[-B, B]$ by using two parameters for each knot, and a third parameter as the knots derivative, with 1 at the boundary points. Hence, one require $3K - 1$ parameters to be output of $\mathcal{H}_{t,d}$, and choose a B. Reason for two parameters for each knot is to use it as width and height of each bin, which can then be used through cumulative sums to find the knots. Using the parameters, one can proceed to find the right bucket for a given value $z_{t-1,d}$ and calculate the rational quadratic. Deriving the actual transformations/inverse/derivative quickly becomes laborious and is avoided for brevity and clarity sake.

Polynomials with a monotonicity requirement may not seem to be immensely flexible, and one may wonder if the transformations are as powerful as, e.g., neural networks. The point which disproves such concerns can be seen through

combining the fact that the polynomials are dense in $C(\mathcal{X}, \mathbb{R})$[5] due to the Stone-Weierstrass Theorem (Lindstrøm 2017, Ch.4, p.127), with the fact that the conditioner allows for a lot of flexibility when considering the transformation of $\boldsymbol{z}_t$. Hence, spline transformations are quite comparable in ability compared to neural network transformations, where both have an added restriction to achieve monotonicity. Empirically, it performs well, but the tests done have been with alternating linear transformations and spline transformations, hence the performance on its own is left unanswered (Kobyzev et al. 2020).

Mixing transformations

Having introduced a bouquet of transformations, we like to point out that we do not necessarily need to use only one transformation in a flow. As alluded to with linear transformations, we may mix them. The transformation f is defined through its components $f_{t,d}$, and we can then let each transformation $f_{t,d}$ be a different transformation (the non-conditioner transformations dictate the same transformation for every $d \in \mathcal{D}$, but the point still stands for different $t \in \mathcal{T}$). When we refer to a flow with a certain transformation, we assume each transformation $f_{t,d}$ has the same transformation type. When we study flows theoretically, we often put one specific transformation under the magnifying glass. However, with the possibility of mixing, we may for example have a more complex transformation for $t = 1$ or $t = T$ and simpler transformations otherwise. The only study we have seen that incorporates this is alternating between linear transformations and conditioner transformations (Durkan et al. 2019b), which we shall return to in Chapter 5.

Conclusion

To conclude the section, we find it useful to consider a few paradigms. Firstly, reliance on complexity through transformations rather than through the conditioner. What we typically gain through the latter is a larger portion of fast computation of the inverse/Jacobian determinant, often incredibly simple transformations that become highly flexible due to the flexibility of the conditioner. On the other hand, non-conditioner transformations allow for interaction between all dimensions at the same transformation, i.e., a fully connected structure, a stark contrast to the AR/IAR structures for example. The lack of fully connected structures in conditioner transformations are something we explore more in Chapter 4.

It is crucial to point out that the number of parameters used in the transformation are typically far fewer with a conditioner, however, this does not mean there are fewer parameters overall. One must count the number of parameters used to transform, but also all trainable parameters such as the ones used to compute the conditioner $\mathcal{H}$. Often, the number of parameters in the conditioner transformations far exceeds the ones in non-conditioners.

Another paradigm worth considering is between transformations requiring more than one transformation $T > 1$ to extract its full potential versus one $T = 1$. Here we find the affine transformation and Sylvester transformation in one corner ($T > 1$) and neural networks/spline/linear transformations in

[5] Assuming $(\mathcal{X}, d)$ is a compact metric space.

the other. Aside from the linear transformation, which is quite limited in its capacity, the key differences between the former and latter groups are the fact that the number of parameters used in the actual transformation are bounded in the former (affine/Sylvester). This paradigm is something we shall explore further in Chapter 4.

Wrapping up transformations, we have now introduced all the important components in a (discrete) flow, namely structures, conditioner, and transformations. We can now proceed to putting them together, and study their capacity/flexibility as flows.

3.6 Universality

It is quite natural to ask questions concerning flexibility. That is, what model space does the flow induce. Until very recently, the question was largely settled for the transformations with unbounded number of parameters, e.g., neural network transformations, and largely unknown for affine transformations. This came to an end with Teshima et al. 2020, and proofs concerning a large number of conditioner based transformations. We do introduce other results given previously as well, as they are fruitful for further discussion in Chapter 4. We are also concerning ourselves with the structure, and any results given specifies both structure and transformation. Obviously, even the most complex transformations struggles if $\mathcal{S}_{ext}(t, d) = \emptyset$, as long as the variables in the target distribution are correlated. However, before we can introduce the fields current understanding when it comes to *universality*, we must first define this concept formally.

As any results we concerns us with, depends on the universality of neural networks, we need to make sure the input space to the neural networks are compact (Kidger et al. 2020). To do this, we need to make sure the input space of flows are compact, and as the flows we concern ourselves with are continuous, means any input space inside the flow—either to a neural network conditioner or the transformation itself—is compact.

Definition 3.6.1. Let $\mathcal{Q}$ be a base distribution and $\mathcal{Z} \in \mathbb{R}^D$ be compact. $\mathcal{Q}$ is a *compact base distribution* if for any $\boldsymbol{z} \in \mathcal{Z}$

$$q_{\boldsymbol{z}_0}(\boldsymbol{z}) > 0$$

and for any $\boldsymbol{z} \notin \mathcal{Z}$

$$q_{\boldsymbol{z}_0}(\boldsymbol{z}) = 0.$$

That is, $\mathcal{Z}_0 = \mathcal{Z}$.

In any universality results we rely on the base distribution to be a compact one. This, however, is not a problem as compactness is not related to the target distribution, and can therefore be set a priori.

Similar to the results concerning neural networks, we want to tie the flows to something akin to universal approximators. However, as we are interested in distributions, it becomes natural to consider what distributions the flow model space covers. To make the definition as broad as possible, we let Φ be the

set of parameters of a flow. This includes both trainable parameters, such as weights in the conditioner, but also hyper parameters. These are typically the number of transformations, the number of knots in splines, the width of hidden layers in neural network transformation, the size of neural networks regarding conditioners, etc. They vary from model to model and are specified in the results we present. We sometimes want to emphasise a flow with a specific set of parameters $\phi \in \Phi$, and denote this by $(\mathcal{Q}, \mathcal{S}, f)_\phi$. We let $\mathcal{NF}$ be a class of normalizing flows, i.e

$$\mathcal{NF} = \bigcup_{\phi \in \Phi} (\mathcal{Q}, \mathcal{S}, f)_\phi.$$

The definition of universality of flows is partially based off Teshima et al. 2020, but in our framework and with the added detail of specifying what set of distribution a particular class of flows can arbitrarily well approximate.

Definition 3.6.2. Let $\mathscr{P}$ be a class of target distributions and μ be the probability measure of a *compact* base distribution $\mathcal{Q}$. $\mathcal{NF}$ is a *universal distribution approximator* (UDA) for $\mathscr{P}$ iff for every distribution $\mathcal{P} \in \mathscr{P}$ with probability measure ν, there exists a sequence of flows $\left[(\mathcal{Q}, \mathcal{S}^{(i)}, f^{(i)})_\phi\right]_{i=1}^{\infty}$ such that $f_*^{(i)}\mu$ converges weakly to ν, i.e

$$\boldsymbol{z}_T = f^{(i)}(\boldsymbol{z}_0) \xrightarrow{d} \boldsymbol{x} \sim \mathcal{P}, \quad \boldsymbol{z}_0 \sim \mathcal{Q},$$

when $i \to \infty$.

Remark 3.6.3. We concern ourselves with weak convergence here, and we are clearly not guaranteed a precise density evaluation. In earlier iterations of this concept, it was referred to as a *universal density approximator* (Huang, Krueger et al. 2018). This can be misleading, exemplified through the density

$$p_n(\boldsymbol{x}) = \begin{cases} 1 - \cos(2\pi n x), & \text{if } 0 < x < 1 \\ 0, & \text{elsewhere} \end{cases}$$

This does clearly not converge to any density, but the CDF of p_n is equal to $x - \frac{\sin(2\pi n x)}{2\pi n}$ which converges to x when $n \to \infty$. Hence, it converges to the uniform distribution between 0 and 1, while the density does not converge to 1. The implication the other way is valid and is proven through Scheffé's Theorem, which states that convergence in density a.e implies convergence in distribution (Scheffe 1947). As the results so far in the literature prove convergence in distribution, we avoid universal density approximator as a term.

Although the results that follow are limited to a weak convergence, the perhaps even bigger limitation is the existence part. There are typically many parameters in normalizing flows and proving the existence of a flow bears no guarantee that we find such a flow (similar complaints can be raised around universal approximators). To systematise it, we can divide such results into three groups.

- Asymptotic results where, by increasing some parameters, one converges guaranteed.

- Asymptotic results are similar to the above, but with the additional information of convergence rate.
- Asymptotic results where, by increasing some parameters, we know there exists a solution which converges.

This divide is of importance when considering the differences between sampling schemes such as MCMC and sampling through normalizing flows, for example. Both can perform bad or good approximating the posterior by samples, in a finite setting. Both can improve by choosing better hyperparameters, but MCMC has a guarantee that no matter what, the samples converge eventually. This is not to say that existence proofs are of no use, it certainly confirms the particular flows capabilities and their potential reach. The reason for using existence proof can mostly be traced back to the use of neural networks. Its flexibility and complexity which makes them popular, also makes it hard to study theoretically.

When we are studying the properties concerning conditioner transformations, one specific type of flow has been central, and therefore given its own definition.

Definition 3.6.4. Let $(\mathcal{Q}, \mathcal{S}, f)$ be a normalizing flow. If the structures for every transformation are IAR-structures without permutation, the flow is a *triangular flow* and denoted by τ. If also, for every $t \in \mathcal{T}$ and $d \in \mathcal{D}$, $f_{t,d}$ is increasing w.r.t. $z_{t-1,d}$ whenever $\boldsymbol{z}_{t-1,i<d}$ is held fixed, it is an *increasing triangular flow*, denoted by $\hat{\tau}$.

Triangular maps (Bogachev et al. 2007) can be seen as the general definition, which we have defined in the scope of flows. Also, any class of normalizing flows $\mathcal{NF}$, where every possible flow in said class is a triangular flow, we denote by $\mathcal{NF}_\tau$ and equivalently $\mathcal{NF}_{\hat{\tau}}$.

Many flows are triangular flows, as the transformations we have introduced are increasing w.r.t. to the variable being transformed, when the parameters used in transforming the variable are held fixed. Combining this with a triangular structure gives us a triangular flow.

One of the reasons why triangular flows are often used in proofs can be illustrated through a result concerning *canonical triangular maps* defined by Bogachev et al. 2007.

Definition 3.6.5 (Bogachev et al. 2007)**.** Let μ and ν be absolutely continuous distributions with CDF F_μ and F_ν respectively and

$$F_\nu^{-1}(u_d \mid \boldsymbol{x}_{i<d}) = \inf\{s \colon F_\nu(s \mid \boldsymbol{x}_{i<d}) \geq u\}$$

A *canonical triangular map* g is an increasing triangular map defined as

$$x_d = g(z_d) = F_\nu^{-1} \circ F_\mu(z_d \mid \boldsymbol{z}_{1:d-1}) \tag{3.6}$$

Remark 3.6.6. The limits of F_ν^{-1} may not exist, and we then adjust the domain of the mapping g to a interval of $\text{supp}(\mu)$—i.e., support of the probability measure.

Theorem 3.6.7 (Bogachev et al. 2007)**.** *Let μ and ν be probability measures defined on $\mathbb{R}^D$. If they are absolutely continuous Borel probability measures,*

there exists a canonical triangular map g such that $\nu = g_*\mu$, where $g_*\mu$ is unique up to null sets of μ.

The definition and result above indicates that triangular flows can be quite expressive. It is also an insightful tool when thinking of proofs, where studying a particular triangular flow's ability to approximate F_μ and F_ν^{-1}. However, it also highlights limitations with regards to flows. We know that flows are continuous, and that the conditioner typically is bounded by continuous functions (e.g., neural networks). The canonical triangular map is not continuous, but by limiting ν to be equivalent[6] to the Lebesgue measure λ, we have continuity in Equation (3.6). Essentially, we require the distribution to have a strictly positive density over its domain. This allows for the inverse of F_ν to be continuous. Hence, the scope of the target distribution is limited to distributions with strictly monotonically increasing CDF. It is also not enough to simply prove that a particular transformation can approximate any *continuous* canonical triangular maps. One also need to show that the parameters used in the transformation change continuously w.r.t. $\boldsymbol{z}_{t-1,i<d}$, to allow for $\mathcal{H}$ to be approximated by neural networks. Yet, Equation (3.6) outlines strategy for proving the universality of flows.

Finally, as many of the results to follow are specified for a particular type of structure, e.g., IAR, it is useful to keep in mind that the results are not limited to the particular structure. By using flow-isomorphism, we can extend the definition of UDA to classes of flows that are flow-isomorphic.

Definition 3.6.8. Let $\mathcal{NF}_\mathcal{S}$ and $\mathcal{NF}_{\mathcal{S}'}$ be two classes of flows, where the only difference between the two classes are the structure they use. We say $\mathcal{S}$ is flow-isomorphic to $\mathcal{S}'$ w.r.t. the classes iff for every $(\mathcal{Q}, \mathcal{S}, f) \in \mathcal{NF}_\mathcal{S}$ and $(\mathcal{Q}, \mathcal{S}', f) \in \mathcal{NF}_{\mathcal{S}'}$ we have $\mathcal{S} \simeq \mathcal{S}'$.

Lemma 3.6.9. *Let $\mathcal{NF}_\mathcal{S}$ and $\mathcal{NF}_{\mathcal{S}'}$ be classes of normalizing flows with structures $\mathcal{S}$ and $\mathcal{S}'$ respectively. If $\mathcal{NF}_\mathcal{S}$ is a UDA for $\mathscr{P}$, $\mathcal{S} \simeq \mathcal{S}'$ w.r.t. $\mathcal{NF}_\mathcal{S}$ and $\mathcal{NF}_{\mathcal{S}'}$, then $\mathcal{NF}_{\mathcal{S}'}$ is also an UDA for $\mathscr{P}$.*

Proof. For any distribution $\mathcal{P} \in \mathscr{P}$, by the definition of UDA, there exists a sequence of flows from $\mathcal{NF}_\mathcal{S}$, $[(\mathcal{Q}, \mathcal{S}, f)_\phi]_{i=1}^\infty$ which converges weakly to $\mathcal{P}$. We know that $\mathcal{NF}_{\mathcal{S}'}$ contains the sequence of flows $[(\mathcal{Q}, \mathcal{S}', f)_\phi]_{i=1}^\infty$, due to the fact that $\mathcal{S} \simeq \mathcal{S}$ w.r.t. $\mathcal{NF}_\mathcal{S}$ and $\mathcal{NF}_{\mathcal{S}'}$. By Proposition 3.3.10, we know that for each i, the corresponding flows in both sequences induce the same density. This also means that the induced probability measures for each flow are equal. By the definition of weak convergence, this must mean that $[(\mathcal{Q}, \mathcal{S}', f)_\phi]_{i=1}^\infty$ converges weakly to $\mathcal{P}$. ■

Remark 3.6.10. For brevity, we say that a sequence of flows converges weakly to a probability distribution. By this we mean that the sequence of pushforward measures corresponding to the sequence of flows, converges weakly to the probability measure corresponding to $\mathcal{P}$.

We also find that any universality results concerning flows with conditioners and structures that are subset of another set of structures, when the transformation $f_{t,d}$ can approximate arbitrarily well the identity function, i.e., there exist a set of parameters such that $f_{t,d}$ is an identity function, and

[6] Two measures μ and ν are equivalent if $\mu \ll \nu$ and $\nu \ll \mu$.

the conditioners are neural networks, implies universality for the latter class of flows.

Lemma 3.6.11. *Let $\mathcal{NF}_{\mathcal{S}}$ and $\mathcal{NF}_{\mathcal{S}'}$ be classes of normalizing flows with conditioner transformations and structures $\mathcal{S}$ and $\mathcal{S}'$ respectively. In addition, every flow in each class uses neural networks as conditioners, and every transformation $f_{t,d}$ can approximate the identity function arbitrarily well. If $\mathcal{NF}_{\mathcal{S}}$ is an UDA for $\mathscr{P}$, $\mathcal{S} \subseteq \mathcal{S}'$ w.r.t. $\mathcal{NF}_{\mathcal{S}}$ and $\mathcal{NF}_{\mathcal{S}'}$, then $\mathcal{NF}_{\mathcal{S}'}$ is also an UDA for $\mathscr{P}$.*

Proof. Assume $\mathcal{NF}_{\mathcal{S}}$ is an UDA for $\mathscr{P}$. By noticing that setting the weights to 0 in the neural network for a particular conditioner $\mathcal{H}_{t,d}$ is equivalent to removing an exterior edge from the structure. Changing a transformation to an Identity node can be done by combining the 0 weights to remove edges and that every transformation can approximate an identity function arbitrarily well by assumption. This means there exists flows $(\mathcal{Q}, \mathcal{S}', \mathcal{H}, f) \in \mathcal{NF}_{\mathcal{S}'}$ that are equivalent to flows obtained by removing edges and changing nodes into Identity nodes. Using the fact that $\mathcal{S} \subseteq \mathcal{S}'$ means for any flow in $\mathcal{NF}_{\mathcal{S}'}$, there exist a flow $(\mathcal{Q}, \mathcal{S}', \mathcal{H}, f) \in \mathcal{NF}_{\mathcal{S}'}$ that are flow-isomorphic to the former flow, and hence by Lemma 3.6.9, we have that $\mathcal{NF}_{\mathcal{S}'}$ is an UDA for $\mathscr{P}$. ■

As every transformation we have encountered can approximate the identity function arbitrarily well, and we rely on neural networks as conditioner, means that the following UDA results in this thesis, for a particular structure $\mathcal{S}$, also holds for structures $\mathcal{S}'$, where $\mathcal{S} \subseteq \mathcal{S}'$. Now before we look at examples of universality of flows, we take a look into its limitations.

Limitations

Before we present universality results, it is worth noticing the limitation of different flows. Linear transformation has already been discussed, and it is quite clear that a flow where all transformations are linear cannot be a UDA for any notable class of distributions.

When $D = 1$, flows with affine transformations, regardless of structure and conditioner, is not an UDA for any notable class either. This can be seen through this observation.

Observation 3.6.12. *Let the target distribution $\mathcal{P}$, with $D = 1$, and base distribution differ in number of modes. Then a flow with only affine transformations cannot approximate the density of $\mathcal{P}$, regardless of number of transformations.*

Proof. This is due to the fact that all the parameters $\{a_{t,1}, b_{t,1}\}_{t=1}^{T}$ in the flow are constants as the set $\mathcal{S}_{ext}(t, 1) = \emptyset$ for all $t \in \mathcal{T}$. This means that the induced density of the flow is

$$q_{\boldsymbol{z}_T} = q_{\boldsymbol{z}_0}(f^{-1}(\boldsymbol{z}_T)) \prod_{t=1}^{T} |a_{t,1}|^{-1}.$$

which shows that the number of modes that we originally have in $q_{\boldsymbol{z}_0}$ is preserved, as we are multiplying by the same constant for all $f^{-1}(\boldsymbol{z}_T)$. This means we need to know a priori the number of modes the target distribution have, so we

can shape our base distribution accordingly, which we clearly cannot in general. Hence, the flow f cannot approximate the density of $\mathcal{P}$. ■

Now this does not necessary imply limitations for $D > 1$, but independence among variables may also seem like an issue. To this one may point out that flows with affine transformations can easily go from independent base distributions to nonindependent distributions—e.g. independent Gaussian as the base distribution transformed to correlated Gaussian—and as the flows are invertible and the inverse makes up a new flow with affine transformations, there must exist flows with affine transformations that decorrelates distributions. Hence, the question that until very recently was unanswered (Teshima et al. 2020), cannot be reduced to issues with independence. However, in practice, when the target distributions have variables independent from the others, intuitively finding flows which do not need other variables is probably easier than correlating/decorrelating them.

Another limitation is whenever the flows are Lipschitz continuous, as Jaini, Kobyzev et al. 2020 demonstrates one such consequence are problems with the tails of target distributions, in particular when the flows are increasing triangular. Starting by defining light and heavy tailed distributions (Foss et al. 2011), extended to multidimensional case (Jaini, Kobyzev et al. 2020).

Definition 3.6.13. Let $\mathcal{P}$ be an arbitrary distribution on $\mathbb{R}^D$ and $\mathcal{P}_{\|\cdot\|}$ be the induced distribution on $\mathbb{R}$ by applying a norm $y = \|\boldsymbol{x}\|$, with $\boldsymbol{x} \sim \mathcal{P}$. $\mathcal{P}$ is *heavy tailed* if

$$\mathbb{E}_{y \sim \mathcal{P}_{\|\cdot\|}} \left[e^{\lambda y} \right] = \infty, \quad \forall \lambda > 0,$$

i.e., no finite higher order moments. Similarly, $\mathcal{P}$ is *light tailed* if there exists a $\lambda > 0$ such that

$$\mathbb{E}_{y \sim \mathcal{P}_{\|\cdot\|}} \left[e^{\lambda y} \right] < \infty.$$

One can extend these to quantify the amount of heaviness of the tail, but this is beyond the scope of this thesis.

We let $\text{Lip}\colon \mathcal{NF} \to \mathbb{R}^+$ denote Lipschitz constant of the flow, which can be unbounded.

Theorem 3.6.14 (Jaini, Kobyzev et al. 2020)**.** *Let $\mathcal{Q}$ be light tailed and $\mathscr{P}$ have at least one heavy tailed distribution $\mathcal{P}$. If $\mathcal{NF}_{\hat{\tau}}$ have*

$$\sup\{Lip(f)\colon f \in \mathcal{NF}_{\hat{\tau}}\} < \infty,$$

then $\mathcal{NF}$ cannot be a UDA for $\mathscr{P}$.

Other subsequent results state similar results for other flows and generative models at large (Wiese et al. 2019). This is not a problem for most flows in general. For instance, a flow with IAR-structure, neural network as the conditioner, and affine transformations with compact base distribution is Lipschitz continuous for a specific set of weights. However, the class of such flows over all possible weights has an unbounded Lipschitz constant. That is, we can find weights for any arbitrarily large Lipschitz constant. The result is still relevant, as many have enforced Lipschitz continuity through bounding parameters when transforming data, e.g., using a sigmoid function on a in

affine transformation. This is done to stabilise the training, but also ruins the possibility of approximating heavy tailed distributions from a light tail one. The solution is either to allow for any parameter value in the actual transformation, or use heavy tail base distributions.

These three results are the only ones, as far as we are aware of, that speaks to the limitation of normalizing flows. The rest of the section is devoted to results of the opposite manner.

Neural network transformations

The earliest result concerning normalizing flow and universality was done by Huang, Krueger et al. 2018. They proved universality for increasing triangular flows with $\mathcal{NN}^{+}_{[L,\hat{D},\gamma]}$ as transformation and neural networks as conditioner $\mathcal{H}$.

Theorem 3.6.15 (Huang, Krueger et al. 2018)**.** *Let $\mathcal{NF}$ be the flow space with*

- *IAR structure,*
- $\mathcal{NN}^{+}_{[L,\hat{D},\gamma]}$ *as transformation,*
- $\mathcal{NN}_{[\tilde{L},\tilde{D},\tilde{\gamma}]}$ *as conditioner space,*
- $\{\hat{D},\tilde{D}\} \subseteq \Phi$*, i.e. the width of hidden layers in both the transformation and the conditioner.*

Let $\mathscr{P} = \{\mathcal{P} : \mathcal{P}$ with the corresponding density $p \in C(\mathbb{R}^D, \mathbb{R}^{+}_{*})\}$, i.e. distributions with positive continuous densities. $\mathcal{NF}$ is then a universal distribution approximator for $\mathscr{P}$.

As $\{\hat{D},\tilde{D}\} \subseteq \Phi$, the universality rely upon extending the number of nodes in the hidden layers of the transformation Ψ^{+}, and the number of nodes in the hidden layers of the networks that approximate the conditioner.

It is worth noting, as alluded to in the previous section, the flow given above does not require more than one transformation ($T = 1$), and therefore no permutation in the structure as well. In return, we have to allow for arbitrarily large hidden layers in the transformation. The second hyperparameter is the size of the network used as the conditioner, which is always a part of conditioner transformation universality results.

The class of distributions $\mathscr{P}$ is quite broad, and can be extended further as we do in Chapter 4. However, with the type of proof that was used, it is difficult to include densities with 0 in arbitrarily many intervals. This comes from the fact that the conditioner cannot estimate non-continuous functions as we are using neural networks, so the weights in Ψ^{+} must change continuous w.r.t. $\mathcal{S}_{ext}(t,d)$. As the crutch of the proof is based on dividing up the conditional CDF of $\boldsymbol{x}_d$, means 0 in its density creates plateaus that makes it difficult to maintain continuity w.r.t. $\mathcal{S}_{ext}(t,d)$, which is essential for neural networks to be able to approximate the conditioner. However, as the proof allows for arbitrary small positive values of its density, it is not detrimental to the flow (also worth mentioning that it does not exclude the possibility of 0 density, it is just not been proven when $T = 1$). To extend it to all continuous target distributions explicitly, the current literature supports the claim that we have

to allow for $T > 1$ and permutations.

Similar result has been shown for polynomials. A particular transformation used by Jaini, Selby et al. 2019 can be written as

$$z_{t,d} = c + \int_0^{z_{t-1,d}} \sum_{k=1}^{K} \left(\sum_{r=0}^{R} a_{l,k} u^l \right)^2 du, \tag{3.7}$$

where $R, K \in \mathbb{N}$. They then state that any univariate real polynomial is increasing iff it can be written as Equation (3.7). Furthermore, they state that the set of increasing real univariate polynomials are dense in the space of real univariate increasing continuous functions. Combining this with continuous increasing triangular functions and Theorem 3.6.7, implies flexibility in flows with Equation (3.7) transformations. Spline transformations have less clearly stated universality results, but typically rely on the same results as increasing polynomials, e.g., relying on Stone-Weierstrass Theorem (Lindstrøm 2017, Ch.4, p.127). Although they do not, as far as we are aware, show directly continuity in the parameters w.r.t. $\mathcal{S}_{ext}(t, d)$, it should still hold, and follows pretty easily from (Huang, Krueger et al. 2018) anyway.

Affine transformations

Let the conditioner be a neural network of some sort (vanilla, CONN, etc.). One may take the view that the normalizing flow is another type of neural network, where two of the components in the network are the conditioner followed by a special type of cell that corresponds to the transformation. Combining these two blocks with invertible matrices, and allowing $\boldsymbol{z}_0$ to not necessarily stem from a base distribution, creates a invertible neural network. This changes the angle from strictly normalizing flows to a special type of invertible neural networks (INN) (Ardizzone et al. 2019). This has then allowed for Teshima et al. 2020 to study the universality of such INNs. To be more specific, we have linear transformations with matrices $W_t \in GL(D)$, and arbitrary conditioner transformation f_t with corresponding structure $\mathcal{S}$, which is a $(D-1)$-coupling structure *without* permutations. The invertible neural network being studied can then be written as a stack of layers T,

$$\Psi_t^{INN}(\boldsymbol{z}_{t-1}) = f_t(W_t \boldsymbol{z}_{t-1}),$$

where the conditioner transformation can be seen as the "non-linearity" γ. From here they consider a myriad of conditioner transformations showing different universality results (not necessarily distributional). For our purpose, we consider the affine transformation and the UDA property.

Theorem 3.6.16 (Teshima et al. 2020)**.** *Let $\mathcal{NF}$ be the flow space with*

- *a $(D-1)$-coupling structure* without *permutation,*
- *each transformation f_t is a composition of a linear transformation with matrix $W_t \in GL_D$, and an affine transformation,*
- *$\mathcal{NN}_{\tilde{L},\tilde{D},\tilde{\gamma}}$ as conditioner space,*

- $\{\tilde{D}, T\} \subseteq \Phi$, *i.e., the width of the hidden layers in the conditioner, and the number of transformations* T.

Let $\mathscr{P} = \{\mathcal{P}\colon \mathcal{P}$ *is any distribution*$\}$, *then* $\mathcal{NF}$ *is a UDA for* $\mathscr{P}$.

The proof of the theorem is quite complex and consists first of a series of reductions from the original space to simpler spaces. The details are beyond the scope of this thesis, but comparing them to the proofs discussed in the last section, we gain less insight into what we are approximating, e.g., the canonical triangular transformation. The final stroke of genius in the paper by Teshima et al. 2020, is that they show the W_t matrices, when using affine transformations, only need to comprise of permutation matrices (Teshima et al. 2020, Lemma 18). This means one can rewrite Theorem 3.6.17 to a more familiar flow.

Theorem 3.6.17 (Teshima et al. 2020)**.** *Let* $\mathcal{NF}$ *be the flow space with*

- *a* $(D-1)$*-coupling structure* with *permutation,*
- *affine transformations,*
- $\mathcal{NN}_{\tilde{L},\tilde{D},\tilde{\gamma}}$ *as conditioner,*
- $\{\tilde{D}, T\} \subseteq \Phi$, *i.e., the width of the hidden layers in the conditioner, and the number of transformations* T.

Let $\mathscr{P} = \{\mathcal{P}\colon \mathcal{P}$ *is either a continuous distribution or discrete distribution*$\}$, *then* $\mathcal{NF}$ *is a UDA for* $\mathscr{P}$.

Hence, affine transformations with the given coupling structure and including permutations, are UDA for practically every distribution. Clearly, as most conditional transformations include the affine transformation (by using a specific set of parameters), and therefore most conditional transformations have the same result.

The second drawback of the proof is the fact that they rely on a very stringent structure. By that we mean we can only transform one variable at a time. The result also holds for structures such as IAR, but where the affine transformation is the identity function for the $D-1$ first dimensions, w.r.t. permutation π_t. Compared to the results for neural network transformation, where one exploit the maximum number of exterior edges in $\mathcal{S}$, while still being a triangular structure, this result is more constricted. Hence, both the limit regarding information sharing for each transformation and the lack of interpretability are drawbacks of such results. In return, it shows that almost all conditional transformations are UDA for practically every distribution, by extending the number of transformations and allowing for permutations in the structure.

CHAPTER 4

Piecewise Affine Flows

4.1 Introduction

In this chapter we use the review of transformations in the literature in the previous chapter, to introduce new transformations that aims to fill the gap between the more complex conditioner transformations such as neural network transformations and spline transformations, and the less complex affine transformation. We start by recognising what traits that make affine transformations attractive. We have also written a small classification of the existing transformations that also emphasis this, but is beyond the scope of the main part, and is added in Appendix A.3. Having pointed out this traits, we introduce a new class of transformations, called *affine extended transformations*. We then move on to creating a few new transformation in under the class of affine extended transformations, which main motive is to be close to affine transformations, yet more expressive.

We then consider CONNs, and implicitly MADE as it is a generalisation, universal approximator properties. We find that CONNs with arbitrary width and one layer are universal approximators, while arbitrary depth are not. Then we show how to remedy the problem for arbitrary depth of the network by enforcing constraints on the masks and sampling, and prove universality under such constraints.

After this we proceed to show that one of the transformations introduced in Section 4.2 are UDA for a large class of distributions, following similar strategy as Theorem 3.6.15 (Huang, Krueger et al. 2018). We both highlight the additional bonus of our transformation compared to the affine transformation, in particular for independency and dimension $D = 1$. We also remark that the transformations and proofs were done before we were aware of the brilliant work of Teshima et al. 2020, but we argue in Section 4.4 that it still has a place and adds to the current literature. We also combine the results we showed for CONNs to our UDA result, which implies that every known UDA result still holds when applying CONN/MADE.

At the final part we consider how to apply a function that can be applied to output of conditioners to enforce strictly positive values, as for instance affine transformations and our new transformations introduced in Section 4.2 requires scale parameters $a_{t,d}$ to be strictly positive, otherwise the resulting model is not a flow, by Definition 3.2.5. The function is rather important when applying normalizing flows, as poor choice leads to unstable training. We review the functions used in the literature, before we introduce two new functions, *Slowplus*

and *Slowabs*.

4.2 Affine Extended Transformations

Affine and additive transformations can be seen as the simplest transformation one can apply in regard to flows that compute parameters through the conditioner. That is, excluding non-conditioner transformations such as linear transformations, etc. Despite their simplicity, they can be quite expressive with certain structures and they can also be quite useful when it comes to dimension and scaling, as both transformations have few parameters and closed form forward, inverse, and derivative. Few parameters also affect the size of the network corresponding to the conditioner, as already noted, the larger the output space is, the larger the network needs to be to get a good approximation. There is also a case to be made regarding interpretation, in as far as that is possible at all, and ease of implementation. That being said, they are quite limited both in a finite context, but also asymptotically under many structures and when the dimension $D = 1$. They also need more than one transformation to produce nonlinearity. This leads us to generalise these transformations to classes that capture some of the scalability, while alleviating flexibility issues.

Affine Extended Transformations

When considering classes of functions which can capture our requirements above, there are some characteristics that in general seem to conflict with the said goal. In particular, transformations where the variable is in multiple terms, i.e., summation. For instance, neural network- and residual transformation must deploy different non $\mathcal{O}(1)$ methods to invert. There are of course exceptions, such as quadratic polynomials or the addition of multiple affine transformations (which obviously can be rewritten as another affine transformation). Another typical problem occurs when piecewise functions are in use, i.e., splines—in particular when there are more than one piece, where one has to search for the right piece or bucket before transforming. Listing all characteristics that can be problematic is of course not fruitful, yet it illustrates typical transformations that are not included in the definition below.

Definition 4.2.1. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow. An *affine extended transformation* is on the form

$$f_{t,d}(z_{t-1,d}) = c_{t,d} \cdot h_{t,d}[a_{t,d} \cdot z_{t-1,d} + b_{t,d}] + d_{t,d}, \tag{4.1}$$

where $h_{t,d}$ are piecewise C^1-diffeomorphisms. Moreover, $h_{t,d}$ must have the following property: when the parameters $(a_{t,d}, b_{t,d}, c_{t,d}, d_{t,d})$ are known, the transformation has analytically and closed form invertible, and where the forward, inverse, and derivative w.r.t. $z_{t-1,d}$ can be computed in $\mathcal{O}(1)$ time. The parameters are computed by the conditioner

$$(a_{t,d}, b_{t,d}, c_{t,d}, d_{t,d}) = \mathcal{H}_{t,d}(\mathcal{S}_{ext}(z_{t,d})).$$

A flow consisting of only affine extended transformations is an *affine extended flow*.

The name *affine extended* refers to the added—potentially nonlinear—function h, separating two affine transformations. Clearly, the affine transformation is included in affine extended transformations, as one can set $h_{t,d}$ to be the identity function, $c_{t,d} = 1$, and $d_{t,d} = 0$. It is also worth noting that for the most part any affine extended transformations are bounded parametrisation- and non-inflection transformations. The following result is mostly included to combine our structure theory and affine extended transformations. We ignore the computational cost of the conditioner, as it is the same for flows with different transformations, but similar conditioners.

Proposition 4.2.2. *Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be an affine extended flow with triangular flow-structure. Then the flow can be computed both ways in $\mathcal{O}(T \cdot D)$ time, including computing the density which is*

$$q_{\boldsymbol{z}_T}(\boldsymbol{z}_T) = q_{\boldsymbol{z}_0}(\boldsymbol{z}_0) \cdot \prod_{t=1}^{T} \prod_{d=1}^{D} \left| \frac{\partial}{\partial z_{t,d}} f_{t,d}(z_{t,d}) \right|^{-1}$$

Proof. This follows from Theorem 3.3.15, the fact that triangular structures allow for knowing the parameters $(a_{t,d}, b_{t,d}, c_{t,d}, d_{t,d})$ for every $t \in \mathcal{T}$ and $d \in \mathcal{D}$, and the definition of an affine extended transformation. ■

The result also emphasises an important fact that the final computation of the forward/inverse flow and its density, is $\mathcal{O}(T \cdot D)$. This means that the computational burden of some more complex transformations, e.g., spline transformations, with only one transformation has better time complexity than affine extended ones with many transformations. However, often the more complex transformations also uses several time steps, time complexity $\mathcal{O}$ does not illuminate the whole picture, and reflections around the size of the conditioner, etc. are still important. It is nevertheless important to point out the full computational burden.

Piecewise Affine Transformations

To make a more fruitful inquiry, both theoretically and empirically later on, we introduce transformations with an explicitly stated $h_{t,d}$. The following transformation—and the ones introduced at the end of this section—can in some sense be seen as adding slight flexibility to some of the more well-known activation functions in the deep learning literature, to create simple yet effective nonlinearities in $h_{t,d}$. This to add extra flexibility to the flow, yet keep it close to affine transformations in terms of the number of parameters and the computational cost associated with it.

Definition 4.2.3. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}f)$ be a normalizing flow. A *piecewise affine transformation* is defined as

$$f_{t,d}(z_{t-1,d}) = h_{t,d}(z_{t-1,d} - b_{t,d}) + b_{t,d} \tag{4.2}$$

where

$$h_{t,d}(x) = \begin{cases} a_{t,d} \cdot x, & \text{if } x > 0 \\ c_{t,d} \cdot x, & \text{else.} \end{cases} \tag{4.3}$$

The parameters are computed $(a_{t,d}, b_{t,d}, c_{t,d}) = \mathcal{H}_{t,d}(\mathcal{S}_{ext}(z_{t,d}))$, where $a_{t,d}, c_{t,d} > 0$. Any flow using only piecewise affine transformations is referred to as an *piecewise affine flow* (PAF).

This small change creates a "breaking point", where the derivative is not defined. However, as this is simply a single point means the transformation is still a piecewise C^1-diffeomorphism and it does not break with the definition of a normalizing flow. However, practically we may run into issues during training due to discontinuity and we return to this point in Chapter 5.

We first show that the transformation we are creating indeed is an affine extended transformation, and therefore follows Proposition 4.2.2 for example. To do this, we need to show that the PAF has a closed form inverse and can be computed in constant time, as it is clear from Definition 4.2.3 that the case holds for the forward transformation. Assume we know the value of $z_{t,d}$ and all the variables $(a_{t,d}, b_{t,d}, c_{t,d}, d_{t,d})$—the latter is, for instance, possible to acquire if the structure is triangular. As

$$\lim_{z_{t,d} \to b_{t,d}^-} f_{t,d}^{-1}(z_{t,d}) = \lim_{z_{t,d} \to b_{t,d}^+} f_{t,d}^{-1}(z_{t,d},$$

means it is continuous. As the derivative when constrained to $z_{t-1,d} < b_{t,d}$ is equal to $c_{t,d} > 0$, and equivalently$z_{t-1,d} > b_{t,d}$ is equal to $a_{t,d} > 0$ means it is a piecewise C^1-diffeomorphism.

The inverse can easily be written in closed form due to the fact that $z_{t-1,d} > b_{t,d} \iff z_{t,d} > b_{t,d}$. Hence, the inverse can be written as

$$f_{t,d}^{-1}(z_{t,d}) = \begin{cases} \frac{z_{t,d} - b_{t,d}}{a_{t,d}} + b_{t,d}, & \text{if } z_{t,d} - b_{t,d} > 0 \\ \frac{z_{t,d} - b_{t,d}}{c_{t,d}} + b_{t,d}, & \text{otherwise.} \end{cases}$$

This shows that the transformation is analytically invertible. Hence, a piecewise affine transformation is an affine extended transformation.

A limited or simplified version of the piecewise affine transformation is when $c_{t,d} = 1$. This version is used to show universality for PAF in Section 4.4 and we also use it in the empirical part in Chapter 5. The reason why we also use it when running experiments are that the limited version only have two parameters and put it very close to affine transformations, where we can see what a very simple nonlinearity $h_{t,d}$ adds to the expressiveness, both theoretically and empirically as well. We therefore find it useful to define it properly.

Definition 4.2.4. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a normalizing flow. A *limited piecewise affine transformation* is defined as

$$f_{t,d}(z_{t-1,d}) = h_{t,d}(z_{t-1,d} - b_{t,d}) + b_{t,d} \tag{4.4}$$

where

$$h_{t,d}(x) = \begin{cases} a_{t,d} \cdot x, & \text{if } x > 0 \\ x, & \text{else.} \end{cases} \tag{4.5}$$

The parameters are computed $(a_{t,d}, b_{t,d}) = \mathcal{H}_{t,d}(\mathcal{S}_{ext}(z_{t,d}))$, where $a_{t,d} > 0$. Any flow using only limited piecewise affine transformations is referred to as an *limited piecewise affine flow* (lPAF).

We end this part by introducing a couple more affine extended transformations to explore empirically in Chapter 5.

Definition 4.2.5. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow. A transformation $f_{t,d}$ is an *affine piecewise affine transformation* if it is of the form

$$f_{t,d}(z_{t-1,d}) = a^{(2)}_{t,d} \cdot h_{t,d}(z_{t-1,d}) + b^{(2)}_{t,d},$$

with $h_{t,d}$ being a piecewise affine transformation and $a^{(2)}_{t,d} > 0$. The parameters comprises of $(a^{(1)}_{t,d}, b^{(1)}_{t,d}, a^{(2)}_{t,d}, b^{(2)}_{t,d})$, where $a^{(1)}_{t,d}$ and $b^{(2)}_{t,d}$ are the parameters used in the piecewise affine transformation $h_{t,d}$. Any flow using only affine piecewise affine transformations is referred to as an *affine-piecewise affine flow* (AfPAF).

Considering the fact that piecewise affine transformations are affine extended easily justifies that affine piecewise affine transformations are as well. A timely question is whether there is a difference—apart from the number of conditioners/size of the conditioner—to alternate between piecewise affine- and affine transformations instead of an affine-piecewise affine transformation, which it turns out to be, and which we explore further in Chapter 5.

Finally, the last transformation we introduce is to eliminate the discontinuity in the derivative of the aforementioned transformations. An analogy can be drawn to what ELU does for ReLU, for example. We construct a function that is still linear for most of $\mathbb{R}$, but non-linear around 0 to make the function have continuity in its derivative. We start by introducing some functions, then defining the transformation, and then explain the different parts and shine some light on the transformation. Firstly, we let $a_{t,d} \in \mathbb{R}$, so not simply greater than 0, and then define $a^+_{t,d} = |a_{t,d}|/2 + 1$. We are creating a transformation which will use a^+ for values outside an area $(-\infty, k]$, for some $k \in \mathbb{R}$ and $k > 0$. That is, we want to have a limited piecewise affine transformation as $z_{t-1,d} \to \infty$, where $a^+_{t,d}$ acts as $a_{t,d}$ in the limited piecewise affine transformation.

Next, we need a function which acts upon the area $[0, k]$, where again $k \in \mathbb{R}$ and $k > 0$. The following two functions is used,

$$g^{(+)}(x) = \beta[e^{\frac{x}{\beta}} - 1]$$
$$g^{(-)}(x) = \beta \ln[\frac{x}{\beta} + 1],$$

where $\beta > 0$ is a hyperparameter discussed later. We can now introduce the $h_{t,d}$ used as according to affine extended transformations (Definition 4.2.1), which we split into two parts $h^{(+)}_{t,d}$ and $h^{(-)}_{t,d}$, and is defined as follows:

$$h^{(+)}_{t,d}(x) = \begin{cases} a^+_{t,d}(x - c_1) + c_2, & x > c_1 \\ g^{(+)}(x), & 0 \leq x \leq c_1 \\ x, & \text{otherwise,} \end{cases}$$

and

$$h^{(-)}_{t,d}(x) = \begin{cases} \frac{(x - c_2)}{a^+_{t,d}} + c_1, & x > c_2 \\ g^{(-)}(x), & 0 \leq x \leq c_2 \\ x, & \text{otherwise,} \end{cases}$$

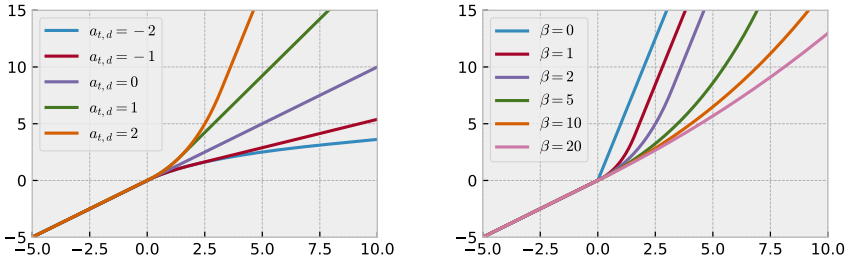


Figure 4.1: continuous piecewise transformations plotted for different $a_{t,d}$, where $a^+_{t,d} = a^2_{t,d} + 1$ and $\beta = 2$, on the left. Equivalently for different βs with $a_{t,d} = 2$, on the right. Towards 0 it we have something smooth, while at the positive end we get something akin to an affine transformation, and identity transformation for $z_{t-1,d} < 0$.

where

$$c_1 = \beta \ln(a^+_{t,d}), \quad c_2 = \beta(a^+_{t,d} - 1).$$

We shall explain why c_1 and c_2 is defined as they are, βs role and so on, but we first introduce the transformation the components above make out.

Definition 4.2.6. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow. A transformation $f_{t,d}$ is an *continuous piecewise transformation* if it is of the form

$$f_{t,d}(z_{t-1,d}) = h_{t,d}(z_{t-1} - b_{t,d}) + b_{t,d}$$

where

$$h_{t,d}(x) = \begin{cases} h^{(+)}_{t,d}(x), & a_{t,d} > 0 \\ h^{(-)}_{t,d}(x), & a_{t,d} \leq 0. \end{cases}$$

Here $(a_{t,d}, b_{t,d}) \in \mathbb{R}^2$ is the two parameters calculated by $\mathcal{H}_{t,d}$. $\beta > 0$ is a hyperparameter.

Although we still have the same amount of parameters—one may also want to consider β as parameter calculated by the conditioner, but for now is a hyperparameter—we have added some complexity and restriction to allow for the transformation to have continuous derivative as well. The effect of the added $g^{(+)}$ and $g^{(+)}$ can be seen in Figure 4.1, where we have plotted the transformations for different $a_{t,d}$ and βs.

To shine some light on the function including c_1, c_2, we first see that the inverse of $h_{t,d}$ is

$$h^{-1}_{t,d}(y) = \begin{cases} h^{(+)}_{t,d}(y), & a \leq 0 \\ h^{(-)}_{t,d}(y). & a > 0. \end{cases}$$

As we see, we simply swap the cases in $h_{t,d}$ to invert it. Hence, if $h_{t,d}$ has continuous derivative, the same applies for the inverse. We therefore concentrate on $h_{t,d}$.

We may start by acknowledging that continuous piecewise transformation is an affine extended transformation, which can be trivially checked. Further, the reason for c_1, c_2 is to make the derivative in $h_{t,d}$ continuous. That is, such that the derivatives are continuous where the different functions in $h_{t,d}^{(\pm)}$ meets. Firstly, we note that we already have continuity in the derivative of $h_{t,d}$ at $x = 0$, regardless of $a_{t,d}$. Finding the derivative w.r.t x for both $h_{t,d}^{(+)}$ and $h_{t,d}^{(-)}$, and also $g^{(+)}$ and $g^{(-)}$ implicitly, we find the derivatives

$$\left(h_{t,d}^{(+)}\right)'(x) = \begin{cases} a_{t,d}^{+}, & x > c_1 \\ e^{x/\beta}, & 0 \leq x \leq c_1 \\ 1, & \text{otherwise,} \end{cases}$$

and

$$\left(h_{t,d}^{(-)}\right)'(y) = \begin{cases} \frac{1}{a_{t,d}^{+}}, & y > c_2 \\ \frac{\beta}{y+\beta}, & 0 \leq y \leq c_2\ t \\ 1, & \text{otherwise.} \end{cases}$$

We then find the appropriate c_1 by solving the equality

$$a_{t,d}^{+} = e^{x/\beta} \implies x = \beta \ln(a_{t,d}^{+}) =: c_1,$$

and similarly for the inverse and c_2,

$$\frac{1}{a_{t,d}^{+}} = \frac{1}{y/\beta + 1} \implies \beta(a_{t,d}^{+} - 1) =: c_2.$$

By setting the two c_1 and c_2 accordingly, we know that $h_{t,d}^{(+)}$ and $h_{t,d}^{(-)}$ have continuous derivative. This implies $h_{t,d}$ has continuous derivative as when $a_{t,d} = 0$, we have $a_{t,d}^{+} = 1$ and $h_{t,d}$ is the identity function, and hence obviously has continuous derivative. We can therefore summarise continuous piecewise transformation as $h_{t,d}$ acts as a piecewise linear transformation for negative number and larger positive ones, but contains forward and inverse continuous derivatives.

We also find it useful to introduce affine continuous piecewise affine transformations, defined similarly to affine piecewise affine transformations.

Definition 4.2.7. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow. A transformation $f_{t,d}$ is an *affine continuous piecewise affine transformation* if it is of the form

$$f_{t,d}(z_{t-1,d}) = a_{t,d}^{(2)} \cdot h_{t,d}(z_{t-1,d}) + b_{t,d}^{(2)},$$

with $h_{t,d}$ being a continuous piecewise affine transformation and $a_{t,d}^{(2)} > 0$. The parameters comprises of $(a_{t,d}^{(1)}, b_{t,d}^{(1)}, a_{t,d}^{(2)}, b_{t,d}^{(2)})$, where $a_{t,d}^{(1)}$ and $b_{t,d}^{(2)}$ are the parameters used in the continuous piecewise affine transformation $h_{t,d}$.

We end with a note on β and its role. It controls, roughly speaking, how much of the nonlinear part in $h_{t,d}^{(+)}$ and $h_{t,d}^{(-)}$ covers, e.g., the $\beta(e^{x/\beta} - 1)$ part is extended longer into the positive part of $h_{t,d}^{(+)}$ when β is larger. However, $a_{t,d}^{+}$ also plays a role of this, as with a small $a_{t,d}^{+}$ we have close to linear transformation for the whole $h_{t,d}$, and thus not much room for the nonlinear part as well. We can therefore only conclude that finding the right β must be explored empirically, and perhaps even include it into the parameter space of the conditioner $\mathcal{H}_{t,d}$.

Related Work

We discuss related work to affine extended transformation, and in particular our new transformations. The only result that we are familiar with is the work of Oliva et al. 2018. They explore the composition of a few transformations. Two of them can be translated to our framework as linear transformations and additive transformations (affine transformation without the scale term). The third transformation they use is of the following form

$$z_{t,d} = r_\alpha(a_{t,d}^{(1)} \cdot z_{t-1,d} + \boldsymbol{v}'_{t,d}\boldsymbol{s}_{1:d-1} + b_{t,d}^{(1)}),$$

where r_α is the Leaky ReLU function, i.e.,

$$r_\alpha(y) = \begin{cases} y & \text{if } y \geq 0 \\ \alpha \cdot y & \text{otherwise,} \end{cases}$$

$a_{t,d}^{(1)}, b_{t,d}^{(1)} \in \mathbb{R}$, and $\boldsymbol{v}_{t,d} \in \mathbb{R}^k$ for some pre-chosen $k \in \mathbb{N}$. Finally, $\boldsymbol{s}_{1:d-1}$ is a state computed by

$$\boldsymbol{s}_{1:d-1} = r(a_{t,d}^{(2)} \cdot z_{t-1,d-1} + \boldsymbol{w}'_{t,d}\boldsymbol{s}_{1:d-2} + b_{t,d}^{(1)}),$$

where r is the ReLU function, $a_{t,d}^{(2)}, b_{t,d}^{(2)} \in \mathbb{R}$, and $\boldsymbol{w} \in \mathbb{R}^k$. Note that $s_{t,0}$ is a known constant. Combining this with linear- and additive transformations has given good empirical results on certain datasets, doing density estimation (Oliva et al. 2018).

There are some differences compared to ours.

(i) $(a_{t,d}^{(1)}, b_{t,d}^{(1)}, a_{t,d}^{(2)}, b_{t,d}^{(2)})$ are all trainable parameters, which means they are not computed through a conditioner. This makes the transformation less complex, as after training the parameters are constant regardless of input.

(ii) The conditioner part is a RNN, and give us a state, $s_{1:d-2}$, which then influences the transformation through addition, as v, w are also trainable parameters. The transformation also enforces the structure to be IAR or AR, and it also suffer computationally due to the inherent sequentiality.

(iii) Piecewise affine transformation allows for the "break point" to be decided by $b_{t,d}$ in $h_{t,d}$, while this is constant at 0 for r_α. To us, the constant 0 works fine due to the combination of linear/additive transformations.

(iv) There are no universality proven around deploying these type of transformations as far as we are aware, while piecewise affine flows are (see Section 4.4).

This was in no way a criticism of the transformations considered by Oliva et al. 2018, but simply to show that there is clearly quite a large difference between these methods. However, the nature of adding nonlinearities seems promising.

4.3 Universality of CONN

Before we move to proving universality for piecewise affine transformations, we turn to CONN and its universal approximator properties. We find that it is a

universal approximator for arbitrary width and one layer, but not for arbitrary depth with finite width.

The question of universality when it comes to MADE, has yet to be discovered, as far as we are aware. In this section we are concerned with the issue of universality in the general sense, i.e. inspecting the CONN model. The results will vary based on how one generate masks, and we concentrate our effort around uniform sampling when initiating the whole network, as it is by far the most widespread one, hence every node which samples does so according to Uniform($\mathcal{C}$). We avoid any agnostic training and comment on them in the end of the section. The results below applies to every model where at least one output dimension is dependent on at least one input dimension. As MADE is simply a special case of CONN fulfilling said assumption, means everything below applies to MADE as well. As we are sampling the masks, we are introducing stochastic elements into the equation, and therefore the question of universality for a space must be considered as the probability of the model space being dense in it, and we assign the model space as a universal approximator for another space, if the probability of it is one, i.e. almost surely.

Firstly, we ensure us that we will always be able to sample all the different masks infinitely many times each, as long as we increase the number of samples.

Lemma 4.3.1. *Let $A = \{1, 2, \ldots, N\}$, where N can be any fixed positive integer. Let $\mathcal{A} \sim$ Uniform(A) be a set with M independently drawn samples. The probability of a finite $k \in \mathbb{N}$ number of samples with value $a \in A$ goes to 0 when $M \to \infty$.*

Proof. We consider the case of $k > 0$. We rewrite every other element in A as a' except for a. We then have a binary case with probabilities $p = 1/N$ for a and $q = (N-1)/N$ for a'. The M samples means we have a binomial distribution $\text{Bin}(M, p)$. Let X be number of a, and consider the limit

$$\begin{aligned}\lim_{M\to\infty} Pr(X = k) &= \lim_{M\to\infty} \binom{M}{k} p^k q^{M-k} \\ &\leq \lim_{M\to\infty} \left(\frac{e\,M}{k}\right) p^k q^{M-k} \\ &\propto \lim_{M\to\infty} M^k\, q^M = 0\end{aligned}$$

The case of $k = 0$ leaves us with $p^k q^{M-k}$ and obviously goes to 0. ■

Let $\mathcal{NN}^C_{[L,\boldsymbol{D},\gamma]}$ be defined similar to earlier, but using CONN instead of neural networks, and equivalently for the two set of models $\mathcal{F}^C_{width}$ and $\mathcal{F}^C_{depth}$. We let $C(\mathcal{X}, \mathbb{R}^{D_{L+1}}; c)$ be the space of continuous functions from $\mathcal{X}$ to $\mathbb{R}^{D_{L+1}}$, where each $d \in D_{L+1}$ is computed using the input dimensions $c(y_d)$.

Theorem 4.3.2. *Let γ be any activation function which is also nonpolynomial. Let $\mathcal{F}^C_{width}$ have 1 hidden layer with every node in the layer using uniform sampling of masks, and $\mathcal{X} \subseteq \mathbb{R}^{D_0}$ be compact. Then $\mathcal{F}^C_{width}$ is almost surely a universal approximator for $C\left(\mathcal{X}, \mathbb{R}^{D_2}; c\right)$.*

Proof. For all $d \in D_2$, let $\Psi^{(d)}$ be a 1 hidden layered regular neural network with input $\{x'_d \colon d' \in c(y_d)\}$ and one dimensional output. For any $\epsilon > 0$ and any $G \in C\left(\mathcal{X}, \mathbb{R}^{D_2}; c\right)$, we have the following: there exists a $D_1^{(d)}$ for every dimension such that there exist a set of weights for each $\Psi^{(d)}$ which gives $|\Psi^{(d)}(c(y_d)) - G_d(c(y_d))| < \epsilon$, due to Theorem 2.4.8.

We then construct a network $\Psi^{CONN} \colon \mathcal{X} \to \mathbb{R}^{D_2}$ with $D_1 = \sum_{d=1}^{D_2} D_1^{(d)}$ number of nodes in hidden layer. We increase the number of hidden nodes until we have $D_1^{(d)}$ nodes with mask $c(y_d)$ for all $d \in D_2$, which has probability equal to one of happening as the number of nodes are increasing, according to Lemma 4.3.1. We simply let weights be 0 for nodes which is not used in any $\Psi^{(d)}$, and have effectively made a network which stacks all the $\Psi^{(d)}$ on top of each other. This implies that with probability equal to one, there exists a network $\Psi^{CONN} \in \mathcal{F}^C_{width}$ with specific weights such that

$$\left\| \Psi^{CONN}(\boldsymbol{x}) - G(\boldsymbol{x}) \right\|_\infty < \epsilon.$$

■

Typically, we are not content with using a 1 hidden layer neural network, and would rather extend it into a bounded number of nodes in each layer, but with arbitrary depth. As it turns out, this is not straightforward, and the results above does not transfer to $\mathcal{F}^C_{depth}$.

Proposition 4.3.3. *Let $\mathcal{F}^C_{depth}$ have some arbitrary fixed width D, and $\mathcal{X} \subseteq \mathbb{R}^{D_0}$ be compact. Regardless of D, $\mathcal{F}^C_{depth}$ is not a universal approximator for $C\left(\mathcal{X}, \mathbb{R}^{D_{L+1}}; c\right)$.*

Proof. For each hidden layer, one can think of sampling uniformly from a set with $D^{|\mathcal{C}|}$ elements, where $|\mathcal{C}|$ is the cardinality of $\mathcal{C}$. Due to Lemma 4.3.1, we know that the number of times a hidden layer l samples results in $m_l(d') \nsubseteq c(y_d)$ for all $d' \in D$ and for an arbitrary output dimension d—assuming not every set in $\mathcal{C}$ is a subset of $c(y_d)$—has no bounds as the number of hidden layers grows. A sample as described results in a node with only a bias term b next time one sample $m_{l'}(d') \subseteq c(y_d)$. Which means we will arbitrarily many times start off by a constant in approximating any output that corresponds y_d. The exception is if every set in $\mathcal{C}$ is a subset of $c(y_d)$. In that case, there are arbitrarily many times where $m_l(d') = c(y_d)$ and $m_{l+1} \neq c(y_d)$, which means layer l will not have a connection to any node in the next layer. Hence, it cannot be a universal approximator for any large class, let alone $C\left(\mathcal{X}, \mathbb{R}^{D_{L+1}}; c\right)$. ■

Remark 4.3.4. In the proof we assume that we can sample uniformly from $\mathcal{C}$, but in MADE one only allow to sample masks that does not allow for constant nodes as above. This does not however improve upon the situation, but rather the opposite. As every time we exclude the set $c(y_d)$ for some d, we assign a constant to the output y_d (or an affine transformation if we include a residual block from input to last hidden layer).

The obvious remedy to the problem is to create distributions which cannot randomly exclude at least one occurrence of $m_l(d') \subseteq c(y_d)$ in each layer. One possibility is to create groups of sets from $\mathcal{C}$ where for each element there exist a larger element of which it is a subset of, e.g. all subsets of $c(y_d)$ in one group

etc. And then sample from each of these groups at least once for each hidden layer. This does not render the issue moot, as there are problems with the smaller subsets, but it might be a reasonable trade-off between the problem described in the last proof, and what we are about to introduce.

For a perhaps more crude way to deal with the issue is to assign a certain number of nodes in each hidden layer to a deterministic mapping m_l, which guarantees that the values computed by $c(y_d)$ can be passed on to the next layer in some form or another. We propose here to let the first $D_0 + D_{L+1}$ nodes in every hidden layer to be chosen deterministically and from here show that it relieves the problems revealed in Proposition 4.3.3.

To show universality for our altered arbitrary depth case, we first introduce a result of identity function and neural networks capability of approximating them. An *enhanced node* refers to an affine transformation, followed by an activation function, and another affine transformation. It basically means we refer to the node as both the weights/bias applied before and after the activation function, and the activation function itself.

Lemma 4.3.5 (Kidger et al. 2020)**.** *Let γ be an activation function and $\mathcal{X} \subseteq \mathbb{R}^{D_0}$ be compact. A single enhanced node $\Psi_{l,d}$ with γ as activation function, can uniformly approximate the identity function arbitrarily well.*

This result means we can treat some of the nodes as storage units, and therefore take a similar approach as Kidger et al. 2020 does when they prove Proposition 4.2 in their paper, i.e., universality of regular neural networks.

Theorem 4.3.6. *Let γ be a nonpolynomial activation function, $\mathcal{X} \subseteq \mathbb{R}^{D_{L+1}}$ be compact. Let $\mathcal{NN}^C_{[L,D_0+D_{L+1}+1,\gamma]}$ be limited to all CONNs where for each hidden layer all but the last node is set accordingly: for each dimension $d \in D_{L+1}$, we require at least one of the nodes d' in every hidden layer l to use the mapping $m_l(d') = c(y_d)$ and similarly for each dimension $d \in D_0$ and a node d' with $m_l(d') = \{d\}$. The final node in each hidden layer is uniformly sampled from $\mathcal{C}$. The resulting $\mathcal{F}^C_{depth}$ is almost surely dense in $C\left(\mathcal{X}, \mathbb{R}^{D_{L+1}}; c\right)$.*

Proof. In the same manner as the proof for the arbitrary width case, we create D_{L+1} single layered arbitrary width networks, $\Psi^{(1)}, \ldots, \Psi^{(L+1)}$, one for each output. Due to Lemma 4.3.5, we can now dedicate D_0 nodes in each hidden layer to simply store the input, the ones with mapping $m_l(d') = d$, and the D_{L+1} nodes with mapping $m_l(d') = c(y_d)$ can store the output. Whenever $c(y_d)$ is sampled in the node which samples masks, we can compute one of the nodes in $\Psi^{(d)}$ using the input storage nodes as input to the sampling node and store the result in the appropriate output storage node. As the output storage nodes has mapping equal to $c(y_d)$ means there is no issue with the masking. At the final layer, the output storage nodes and output nodes both have $c(y_d)$ as mapping and can therefore transfer the values over. As the list with single layered arbitrary width networks can approximate any function in $C\left(\mathcal{X}, \mathbb{R}^{D_{L+1}}; c\right)$ due to Theorem 2.4.8, means the constructed Ψ^{CONN} can also do it, as we can always sample nodes with mapping $c(y_d)$, arbitrarily many times as we extend the depth of the network due to Lemma 4.3.1. Hence, with probability one, $\mathcal{F}^C_{depth}$ is dense in $C\left(\mathcal{X}, \mathbb{R}^{D_{L+1}}; c\right)$. ■

We can therefore achieve the same results compared to regular neural networks, with the same amount of nodes in each hidden layer. One could

of course have made a neural network Ψ for each $d \in D_{L+1}$, but applying Theorem 2.4.9 to each one would require more nodes in the hidden layers, hence the use of CONN is to first and foremost increase speed and memory efficiency, instead of creating D_{L+1} neural networks. This must surely compromise the accuracy of the network some, given finite depth, compared to the naive approach of D_{L+1} networks, but we can now conclude that it still provides the flexibility we often seek when employing neural networks.

Thinking of the Theorem 4.3.6, we may swap the role of the deterministic nodes in the network, with residual connections. That is, wherever a node have no edges from the last layer, we add residual connections from input layer to said node, and add residual connection from a node with no outgoing edges to the output layer. We have explored this with a universality proof, together with a different sampling scheme of masks, in Appendix A.2.

A final note on the role of order-agnostic training and connectivity-agnostic training, as we excluded them previously. It is certainly so, that if one can spare the expenditures, it can improve the results by implementing the aforementioned schemes, but it does not improve on the results above. That is, if the model space is not an universal approximator, the addition of agnostic training does not change that. As already noted, we can look at the result of agnostic training as making many models. However, when the width and depths is finite, there are only a finite number of models one can create through agnostic training. Hence, it can be seen as a model with simply larger hidden layers than the original model, yet finite. This means that Proposition 4.3.3 still applies, as it is arbitrary fixed width. The same argument implies that the theorems given above also holds, as we are simply expanding models that are already universal approximators.

4.4 Universality

In this section we prove universality for flows with limited piecewise transformation, IAR structure without permutation, and neural networks as conditioner, where we first prove using $T \cdot D$ neural networks—i.e. one neural network for each $\mathcal{H}_{t,d}$—and then combine universality results with result from the previous section regarding CONNs.

We preface these results with a note on universality results concerning affine transformations in Teshima et al. 2020. There are obvious differences from the results we are about to present, where both that Teshima et al. 2020 requires $D > 1$, a more stringent structure, and permutations. Other differences has already been discussed in Section 3.6. It is fair however, to also note that the results by Teshima et al. 2020 appeared after we started working on this section, and was not seen by us before we finished the universality proofs. In fact, one of the reasons for us to introduce the transformations we have, was to mimic affine transformations as close as possible yet be able to prove universality. This was rendered somewhat moot with the paper by Teshima et al. 2020, yet the results is still a new and useful contribution through different structure which can transform each variable for time step t, valid for $D = 1$, no permutations in the structure etc.

Before we can outline the proof of universality results and introduce probability classes $\mathscr{P}$ we need to preface with the following. For this section, the compact subset supported by the base distribution—remember the compact distribution assumed in any UDA results—is a cube $[k_0, k_1]^D$, for $k_0 < k_1$ and $k_0, k_1 \in \mathbb{R}$. It makes the proof easier to follow, and we do not lose anything of value by such an assumption (also keep in mind that such an assumption is an a priori assumption about the base distribution, and introduces no problems in that regard). We also remind the reader of the Sigmoid function, denoted by $\sigma\colon (-\infty, \infty) \to (0, 1)$.

For a large portion of the proof, we are interested in showing that the transformation and conditioner combined with the Sigmoid function, can approximate different functions which goes from $[k_0, k_1]^d$ to $[l_0, l_1]$ for $0 \leq l_0 < l_1 \leq 1$ and $d \in \mathcal{D}$. We therefore denote

$$g_T(\boldsymbol{z}_0) = \sigma \circ f(\boldsymbol{z}_0),$$

for flows $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$.

Classes of Target Probabilities

We start by introducing the classes of target distributions we show universality for. We have two different classes of probabilities to concern ourselves with. The first only applies to certain special cases, that is, with independence or dimension $D = 1$. The class is

$$\mathscr{P}_1 = \{\mathcal{P}\colon \mathcal{P} \text{ is a continuous distribution and the variables are independent.}\}$$

Note that for $D = 1$, the distribution above is merely every continuous distribution.

The other distributions do not require independence, but is limiting in other ways. As a comparison, we denote $\mathscr{P}^+$ to be the same class that Huang, Krueger et al. 2018 shows universality for neural network transformations.

$$\mathscr{P}^+ = \{\mathcal{P}\colon \mathcal{P} \text{ with the corresponding density } p \in C(\mathbb{R}^D, \mathbb{R}_*^+)\}.$$

Our class of probabilities further extend this to, what seems to us, as the broadest class we are able to prove directly with the current strategy. It is slightly more intricate to state, and thus is defined as follows.

Definition 4.4.1. Let $\mathcal{X} \subseteq \mathbb{R}^D$ be a connected subset. The *strictly conditionally continuous distribution* is a distributions $\mathscr{P}$ such that the density $p_{\boldsymbol{x}} \in \mathscr{P}$ follows

$$\begin{cases} p(\boldsymbol{x}) > 0, & \text{if } \boldsymbol{x} \in \mathcal{X} \\ p(\boldsymbol{x}) = 0, & \text{if } \boldsymbol{x} \notin \mathcal{X}. \end{cases}$$

Also, the conditional CDF of the density, $F_d(x_d \mid \boldsymbol{x}_{1:d-1})$, is continuous w.r.t. $\boldsymbol{x}_{1:d-1}$.

Using this definition we may specify the third class of distributions we want to prove universality for,

$$\mathscr{P}_2 = \{\mathcal{P}\colon \mathcal{P} \text{ is a strictly conditional continuous distribution.}\}$$

The biggest difference between $\mathscr{P}^+$ and $\mathscr{P}_2$ is that in the latter we include distributions with density 0 in the tails and discontinuity in the density. It also follows that $\mathscr{P}^+ \subset \mathscr{P}_2$. This is in no way saying that the neural network transformation is weaker than the limited piecewise affine transformation. Indeed, it does seem possible to extend the results in Huang, Krueger et al. 2018 to $\mathscr{P}_2$ through the same way we show it for our transformation. Also, even though one do not prove universality for a larger class than the aforementioned ones directly—here directly is for example to construct transformations explicitly as we do—one may argue for universality of larger classes in indirect ways. For instance, argue that universality of $\mathscr{P}_2$ implies universality where the density is allowed to be 0, as we can have distributions in $\mathscr{P}_2$ which take arbitrarily small positive values $\epsilon > 0$ (although we have not seen a proof of this and are hence cautious with our claims). Another case of indirectly showing it is the fact that absolutely continuous distributions (w.r.t Lebesgue measure) are dense in the set of all distributions (Teshima et al. 2020, Lemma 5).

The two main reasons why we choose to specify probability classes in universality results are as follows. The first is to highlight for what distributions the proof is valid for. This also elucidates what the transformations constructed in the proof are able to express, as well as show limitations of different proof strategies. Secondly, specifying the distribution class allows for universality results with added specification, e.g., bound the number of transformations we need to reach a certain precision etc. It is also useful if one want to use more limited conditioners, which may lead to universality results for smaller probability classes, but the resulting flow can have other useful properties—we discuss this particular idea further in Section 6.2.

Outline of Proof

We give a short outline of the proof to both motivate the reader and ease the experience. We use a similar strategy as Huang, Krueger et al. 2018 did, relating closely to convergence of flows to canonical triangular transformations (Bogachev et al. 2007). We wish to show, for any target distribution in $\mathscr{P}_i$ with CDF F, to first show uniform convergence of g_T towards $\sigma \circ F^{-1}$, when $T \to \infty$, using neural networks as conditioner. Then use the fact that we have convergence for $f = \sigma^{-1} \circ g_T$ and applying the following lemma to show we have weak convergence of flow to target distribution:

Lemma 4.4.2 (Lemma 4, Huang, Krueger et al. 2018)**.** *Let $\mathcal{Z} \subseteq \mathbb{R}^D$ and $\mathcal{X} \subseteq \mathbb{R}^D$, with each being the sample space of a probability space, i.e. $(\mathcal{Z}, \mathscr{B}(\mathcal{Z}), \mu)$ and $(\mathcal{X}, \mathscr{B}(\mathcal{X}), \nu)$. Let $J \colon \mathcal{Z} \to \mathcal{X}$ be any function and J_n be a sequence of functions such that J_n converges pointwise to J. Then a transformation of the form $x_n = J_n(z)$ converges in distribution to $x = J(z)$.*

The proof is quite straightforward by introducing a bounded continuous function h, and show convergence in expectation of $h(z_n)$ to $h(z)$ by the dominated convergence theorem. Then simply finish it by applying the Portmanteau's lemma.

The showing of convergence to $\sigma \circ F^{-1}$ takes some steps and we therefore give a small outline of the steps here.

- We start by showing universality for dimension 1. This makes the end goal and the different lemmas for the generalised case more clear, as well as we show universality for $\mathscr{P}_1$.
- We then proceed to introduce a function $G = (G_1, \dots, G_D)$, where each G_d can at the end be substituted with a composition of the canonical triangular maps followed by the Sigmoid function σ.
- We then show uniform convergence of g_T towards G, constructing a f using a non-specific conditioner $\mathcal{H}_{t,d}$.
- Showing some continuity and compact properties of the flow, we can then show that the non-specific conditioner $\mathcal{H}_{t,d}$ can be approximated arbitrarily well by a neural network, and still give uniform convergence to G.
- We can then first show universality for $\mathscr{P}_2$, by showing pointwise convergence towards the canonical triangular map, for any distribution $\mathcal{P} \in \mathscr{P}_2$.

Universality when $D = 1$

We start by showing universality for $D = 1$, where some of the results are needed for the general case, and also goes to show the differentiating strength between limited piecewise affine transformations and affine transformations.

Lemma 4.4.3. *Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ with an IAR-structure, limited piecewise transformation, a neural network as conditioner, and dimension $D = 1$. Let $g\colon [k_0, k_1] \to [l_0, l_1]$ be a monotonically increasing function with $g(k_0) = l_0$ and $g(k_1) = l_1$, and $0 \leq l_0 < l_1 \leq 1$. There exists a flow such that g_T converges uniformly to g when $T \to \infty$, for $z \in [k_0, k_1]$.*

Proof. For any $\epsilon > 0$, we set $M = \lceil \frac{1}{\epsilon} \rceil$. Start by dividing $[l_0, l_1]$ into $M + 1$ subsets

$$\left(l_0, l_0 + \frac{l_1 - l_0}{M+1}\right), \left(l_0 + \frac{l_1 - l_0}{M+1}, l_0 + \frac{2(l_1 - l_0)}{M+1}\right), \dots, \left(l_0 + \frac{M(l_1 - l_0)}{M+1}, l_1\right).$$

We shall denote the boundary points $y_m \coloneqq l_0 + \frac{m(l_1 - l_0)}{M+1}$ for $m \in \{1, 2, \dots, M\}$. We can also find $x_m = g^{-1}(y_m)$, where

$$g^{-1}(y_m) = \inf\{x_m\colon g(x_m) = y_m, \forall x_m \in [k_0, k_1]\}.$$

We cover the case with $l_0 > 0$ and $l_1 < 1$ first, but in the case of $l_0 = 0$ we discard the first transformation defined here, and similarly with $l_1 = 1$ and the last transformation (one may simply think of discarding as letting the first/last transformation to approximate the identity function, i.e., use $a_{t,1} = 1$).

Let the number of transformations be $T = M + 2$. The goal now is to show that there exist a mapping $(a_t, b_t)_{t=1}^T = \mathcal{H}(\mathcal{S}_{ext}(t, 1))$, such that $|g_T(z_0) - g(z_0)| < \epsilon$ for all $z_0 \in [k_0, k_1]$. This means, we need to specify parameters first, and then also show that a neural network can approximate these parameters. The latter can be dealt with quite straight forward. As $D = 1$ means that the conditioner takes no input and output constants $(a_t, b_t)_{t=1}^T$, i.e.,

for all $x \in [k_0, k_1]$ we have $\mathcal{S}_{ext}(t, 1) = \emptyset$. This also means that the function $\mathcal{H}_{t,1}$ is continuous w.r.t $\mathcal{S}_{ext}(t, 1)$, as constants are continuous with input from a compact set, hence we can approximate it arbitrarily well with a neural network following Theorem 2.4.9.

Let $b_{1,1} = \min(\sigma^{-1}(l_0), k_0) - 1$ and

$$a_{1,1} = \frac{\sigma^{-1}(l_0) - b_1}{k_0 - b_1},$$

where we have that $a_1 > 0$, due to $\sigma^{-1}(l_0) > b_{1,1}$ and $k_0 > b_{1,1}$. We can then proceed with setting the next $t \in \{2, \dots, T-1\}$ parameters as

$$b_{t,1} = \begin{cases} \sigma^{-1}(y_{t-2}), & \text{if } t > 2 \\ \sigma^{-1}(l_0), & \text{otherwise.} \end{cases}$$

The scaling parameter is then set to,

$$a_{t,1} = \frac{\sigma^{-1}(y_{t-1}) - b_{t,1}}{\left(\bigcirc_{j=1}^{t-1} f_{j,1}(x_{t-1})\right) - b_{t,1}}.$$

Here $\left(\bigcirc_{j=0}^{t-1} f_{j,1}(x_{t-1})\right)$ applies the previous $t-1$ transformations using the parameters we define, which means $\left(\bigcirc_{j=0}^{t-1} f_j(x_{t-1})\right) > b_{t,1}$. We also have that $\sigma^{-1}(y_{t-1}) > b_{t,1}$ due to how $b_{t,1}$ is defined, $y_m > y_{m-1} > l_0$, and σ is monotonically increasing, which implies that $a_{t,1} > 0$. The last transformation $t = T$ have the parameters

$$b_{T,1} = \sigma^{-1}(y_M)$$

and

$$a_{T,1} = \frac{\sigma^{-1}(l_1) - b_{T,1}}{\left(\bigcirc_{j=0}^{t-1} f_{j,1}(k_1)\right) - b_{T,1}},$$

with $a_{T,1} > 0$ using similar argument as for the other $a_{t,1}$.

We now have the final transformation with the property that for all x_m with $m \in \{1, \dots, M\}$, we have

$$g_T(x_m) = y_m,$$

and if $l_0 > 0$ we have $g_T(k_0) = l_0$ and equivalently with $l_1 < 1$ and $g_T(k_1) = l_1$.

To show convergence of g_T and g, we simply see that $g_T(x_m) - g_T(x_{m-1}) = \frac{l_1 - l_0}{M+1}$ for all $m \in \{1, \dots, M\}$ and $g_T(x_1) - l_0 = l_1 - g_T(x_M) = \frac{l_1 - l_0}{M+1}$. We also know that $l_0 \leq g_T(z_0) \leq l_1$, for all $z_0 \in [k_0, k_1]$. Using the fact that both functions g_T and g are monotonically increasing and $l_1 - l_0 \leq 1$, means that for all $z_0 \in [k_0, k_1]$,

$$|g_T(z_0) - g(z_0)| \leq \frac{l_1 - l_0}{M+1} < \frac{l_1 - l_0}{M} \leq \frac{1}{M} = \frac{1}{\lceil \frac{1}{\epsilon} \rceil} \leq \epsilon$$

■

Following this result we can prove universality for $\mathscr{P}_1$. We remind the reader that the flow space contains all flows parameterized by Φ, which is both hyperparameters and trainable parameters. We also remind the reader of the notation $\mathcal{NN}_{[l,n,\gamma]}$ as the space of neural networks with l hidden layers, n nodes in each hidden layer, and γ as activation function. So any $\Psi \in \mathcal{NN}_{[l,n,\gamma]}$ is a network with l hidden layers, n nodes in each hidden layer, γ as an activation function, and with a specific set of weights.

Theorem 4.4.4. *Let $\mathcal{NF}$ be the flow space with*

- *an IAR structure without permutation,*
- *limited piecewise affine transformations,*
- *$\mathcal{NN}_{[L,\tilde{D},\gamma]}$ as conditioner space,*
- *$\{\tilde{D}, T\} \subseteq \Phi$, i.e., the width of the hidden layers in the conditioner, and the number of transformations T.*

$\mathcal{NF}$ is a UDA for $\mathscr{P}_1$ when $D = 1$.

Proof. For any distribution $\mathcal{P} \in \mathscr{P}_1$, let $F_{\mathcal{P}}$ be the corresponding CDF. We have that $F_{\mathcal{P}}^{-1}(u)$, for $u \in [0,1]$, is monotonically increasing, by definition of the CDF. Similarly, the CDF of the base distribution $F_{\mathcal{Q}}(z_{0,1})$, for $z_{0,1} \in \mathcal{Z}_0$ where $\mathcal{Z}_0$ is the sample space for the base distribution, is monotonically increasing by definition. Let g be the canonical triangular mapping, i.e., $g(z_{0,1}) = F_{\mathcal{P}}^{-1} \circ F_{\mathcal{Q}}(z_{0,1})$. As σ is also monotonically increasing, means $\sigma \circ g$ is monotonically increasing and its image is $[l_0, l_1]$, with $0 \leq l_0 < l_1 \leq 1$ (open set if $l_0 = 0$ and equivalently for $l_1 = 1$). From Lemma 4.4.3, we have that when $T \to \infty$ and the number of neurons in each hidden layer increases $\tilde{D} \to \infty$, there exists a set of T neural networks, due to Lemma 4.4.3,

$$A = \{\Psi^{(t)} \colon \Psi \in \mathcal{NN}_{[L,\tilde{D},\gamma]} \text{ and } t \in \mathcal{T}\}$$

such that $g_T = \sigma \circ f$ uniformly converges towards $\sigma \circ F^{-1}(u)$. This implies f converges towards $F^{-1}(u)$, and due to Lemma 4.4.2, we have that $f(z_{0,1}) = z_{T,1} \xrightarrow{d} x = F_{\mathcal{P}}^{-1} \circ F_{\mathcal{Q}}(z_{0,1})$. ■

This shows that with limited piecewise affine transformation, we can approximate any continuous distribution arbitrarily well—in terms of weak convergence that is. While affine transformations are inherently poor, as we have seen through observations earlier. Hence, adding a small change in the transformation makes a lot of difference in the one dimensional case.

Generalising to Multidimensional Case

We now wish to extend the results above for higher dimensions, while also including the IAR-structure. We define a new function for each $d \in \mathcal{D}$, $G_d(z_{0,d}, \boldsymbol{z}_{0,1:d-1})$, where $\boldsymbol{z}_0 \in [k_0, k_1]^D$. When $\boldsymbol{z}_{1:d-1}$ is fixed, it is assumed the function G_d is a *strictly* monotonically increasing function w.r.t. $z_{0,d}$, where

$l_0^d \leq G_d(z_{0,d}, \boldsymbol{z}_{0,1:d-1}) \leq l_1^d$, with $0 \leq l_0^d < l_1^d \leq 1$. It is also assumed continuous w.r.t. $\boldsymbol{z}_{0,1:d-1}$, i.e. given $\boldsymbol{z}_{0,1:d-1}$, for all $\epsilon > 0$ there exist a $\delta > 0$ such that

$$\begin{aligned} &||\boldsymbol{z}_{0,1:d-1} - \tilde{\boldsymbol{z}}_{0,1:d-1}||_\infty < \delta \\ &\implies |G_d(z_{0,d}, \boldsymbol{z}_{0,1:d-1}) - G_d(z_{0,d}, \tilde{\boldsymbol{z}}_{0,1:d-1})| < \epsilon, \end{aligned} \tag{4.6}$$

where $\tilde{z}_{0,1:d-1} \in [k_0, k_1]^{d-1}$. In addition to this, we allow for the end points l_0^d, l_1^d to change when $z_{0,1:d-1}$ changes. However, they must be between 0 and 1 including, and $l_0^d < l_1^d$ and due to the continuity, the changes must be continuous as well. To be more precise, there must be continuity for the point k_0, i.e. $G_d(k_0, z_{0,1:d-1}) = l_0^d$, and equivalently for k_1, l_1^d. To quickly summarise the properties of G_d:

- For each $\boldsymbol{z}_{0,1:d-1}$:
  - $\exists\, l_0^d, l_1^d \in [0,1]$ such that $l_0^d < l_1^d$ and $G_d \colon [k_0, k_1] \to [l_0^d, l_1^d]$.
  - G_d is *strictly* monotonically increasing.
  - $G_d(k_0, \boldsymbol{z}_{0,1:d-1}) = l_0^d$ and $G_d(k_1, \boldsymbol{z}_{0,1:d-1}) = l_1^d$
- G_d is continuous w.r.t. $\boldsymbol{z}_{0,1:d-1}$, fulfilling Equation (4.6).
- The boundary points in the image of G_d may change when $\boldsymbol{z}_{1:d-1}$ changes, but according to the points above, must do so continuously, with regards to Equation (4.6).

We wish to show g_T can converge towards $G = (G_1, G_2, \ldots, G_D)$ using limited piecewise affine transformation, similarly as in Lemma 4.4.3. We do not assume a neural network as the conditioner yet, as we first need to make sure $(a_t, b_t)_{t=1}^T = \mathcal{H}_d(\boldsymbol{z}_{1:d-1})$ are continuous w.r.t. $\boldsymbol{z}_{1:d-1}$. We then have to show a few properties regarding compactness and continuity before we can confirm that we may approximate $\mathcal{H}$, as specified below, arbitrarily well using a neural network.

Lemma 4.4.5. *Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow with limited piecewise affine transformations and an IAR-structure. Let the function G be defined as the function where each of the D outputs is defined by G_d, i.e. $G = (G_1, G_2, \ldots, G_D)$. Then there exist a flow such that the transformation $g_T = \sigma \circ f$, with continuity in $\mathcal{H}_d(\boldsymbol{z}_{0,1:d-1})$ for all $d \in \mathcal{D}$, converges uniformly to G.*

Proof. For all $\boldsymbol{z}_{0,1:d-1}$ and for all $\epsilon > 0$, setting $M = \lceil \frac{1}{\epsilon} \rceil$ and letting $\mathcal{H}_{t,d}$ output the parameters specified in the proof of Lemma 4.4.3, gives uniform convergence to G_d due to Lemma 4.4.3. Using the same M for all $d \in \mathcal{D}$ and designing $\mathcal{H}_{t,d}$ as mentioned, gives uniform convergence for all G_d's, hence there exist a flow $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ such that the transformation $g_T = \sigma \circ f$ converges uniformly towards G. We therefore only need to show continuity in $\mathcal{H}_{t,d}$ as described, for all $d \in \mathcal{D}$ and $t \in \mathcal{T}$.

We show continuity for an arbitrary $d \in \mathcal{D} \setminus \{1\}$, see Lemma 4.4.3 for continuity when $d = 1$. We remind the reader that we have $M + 2$ transformations, with the first and last transformation depending on whether $0 < l_0^d$ and $l_1^d < 1$ respectively, similar to the proof of Lemma 4.4.3.

Let $\epsilon_1 > 0$. For all $\epsilon_2 > 0$ there exist a $\delta_2 > 0$ such that that whenever

$$||\boldsymbol{z}_{0,1:d-1} - \tilde{\boldsymbol{z}}_{0,1:d-1}||_\infty < \delta_2$$

implies $|l_0^d - \tilde{l}_0^d| < \epsilon_2$, with an equivalent argument for l_1^d, due to continuity in G_d. This means we can choose δ_2 such that $|y_m - \tilde{y}_m| < \epsilon_2$, where y_m is similarly defined as in the proof of Lemma 4.4.3. Combine this with the fact that σ^{-1} is continuous, which means we can choose ϵ_2 such that $|\sigma^{-1}(y_m) - \sigma^{-1}(\tilde{y}_m)| < \epsilon_1$ for all $m \in \{1, 2, \ldots, M\}$. Setting then $\delta_1 = \delta_2$ gives us

$$||\boldsymbol{z}_{0,1:d-1} - \tilde{\boldsymbol{z}}_{0,1:d-1}||_\infty < \delta_1 \implies ||\boldsymbol{b} - \tilde{\boldsymbol{b}}||_\infty < \epsilon_1,$$

where $\boldsymbol{b}$ and $\tilde{\boldsymbol{b}}$ are vectors with the corresponding $(b_{t,d})_{t=2}^{T-1}$ for $G_d(z_{0,d}, \boldsymbol{z}_{0,1:d-1})$ and $G_d(z_{0,d}, \tilde{\boldsymbol{z}}_{0,1:d-1})$ respectively. The same argument holds the two special transformation, i.e., when $0 < l_0 < l_1 < 1$ there is continuity in $b_{1,d}$ and $b_{T,d}$. This is due to the constraint of G_d, that the boundary points may change, but only continuously.

Moving onto the a's. When we know that the $b_{t,d}$ are continuous w.r.t $\boldsymbol{z}_{t,1:d-1}$, it follows quickly that

$$a_{1,d} = \frac{\sigma^{-1}(l_0) - b_{1,d}}{k_0 - b_{1,d}}$$

is continuous, as σ^{-1} is continuous and $k_0 > b_{1,d}$. It then follows inductively that for $t \in \{2, \ldots, T\}$,

$$a_{t,d} = \frac{\sigma^{-1}(y_{t-1}) - b_{t,d}}{\left(\bigcirc_{j=1}^{t-1} f_{j,1}(x_{t-1})\right) - b_{t,d}}.$$

is continuous with similar arguments as for $a_{1,d}$ in addition to the fact that limited piecewise affine transformations are continuous.

We can then conclude that for any $\boldsymbol{z}_{t,1:d-1}$ with $t \in \mathcal{T}$, and for all $\epsilon > 0$, we can find a $\delta > 0$ for each parameter (then simply pick the smallest δ of them), such that

$$||\boldsymbol{z}_{t,1:d-1} - \tilde{\boldsymbol{z}}_{t,1:d-1}||_\infty < \delta \implies ||\mathcal{H}_{t,d}(z_{t,1:d-1}) - \mathcal{H}_{t,d}(\tilde{z}_{t,1:d-1})||_\infty < \epsilon$$

As d was arbitrarily chosen, and $d = 1$ is covered by Lemma 4.4.3, means it holds for all $d \in \mathcal{D}$ and hence $\mathcal{H}$ as specified by Lemma 4.4.3, is continuous. ■

Before we can complete convergence of g_T to G with the additional part of $\mathcal{H}$ being a neural network, we need to show a small result of continuity of $f_{t,d}$ w.r.t $b_{t,d}$.

Lemma 4.4.6. *Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow with limited piecewise affine transformations and an IAR-structure. Then, for all $d \in \mathcal{D}$ and $t \in \mathcal{T}$, the transformation $f_{t,d}$ is continuous w.r.t. $b_{t,d}$.*

This is a minor proof, done quite straightforward using cases. It is, however, a bit long and technical without any real insights, and hence the proof has been relegated to Appendix A.4.

We can now combine Lemma 4.4.5 and the universality of neural networks, to let $\mathcal{H}_{t,d}$ as specified in Lemma 4.4.5 be approximated by a neural network. To differentiate between the conditioner specified in Lemma 4.4.5 and a neural network, we write $g_{T,d}(z_{0,d}; \mathcal{H})$ and $g_{T,d}(z_{0,d}; A)$, where A is a set of $T \cdot D$ neural networks, which indicated the use of the specified conditioner and neural

networks respectively ($g_{T,d}$ is the output after applying σ to $z_{T,d}$). We write equivalently for a transformation, with $f_{t,d}(z_{t-1,d}; \mathcal{H}_{t,d})$ and $f_{t,d}(z_{t-1}; \Psi)$, where Ψ is a neural network.

Lemma 4.4.7. *Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow with limited piecewise affine transformations and an IAR-structure. For every $t \in \mathcal{T}$ and $d \in \mathcal{D}$, let $\mathcal{H}_{t,d}$ be approximated by a neural network $\Psi \in \mathcal{NN}_{[l_d, d+3, \gamma]}$. Then there exist a flow of $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ such that g_T converges uniformly toward G as $T \to \infty$ and $l_d \to \infty$.*

Proof. We start by showing convergence for an arbitrary $d \in \mathcal{D}$ and G_d. Firstly, as the input $\boldsymbol{z}_{0,1:d-1}$ is compact and the function itself is continuous by the definition of an affine extended transformation, the input to any $\mathcal{H}_{t,d}$ is compact. We can then, for any $\delta > 0$ and any $t \in \mathcal{T}$, find a L_d such that whenever $l_d > L_d$ implies there exists a network $\Psi \in \mathcal{NN}_{[l_d, d+3, \gamma]}$ where

$$\|\Psi(\boldsymbol{z}_{0,1:d-1}) - \mathcal{H}_{t,d}(\boldsymbol{z}_{0,1:d-1})\|_\infty < \delta,$$

due to Theorem 2.4.9.

We now wish to show convergence, by picking an arbitrary $t \in \mathcal{T}$, of a transformation $f_{t,d}$ between the two conditioner. We first show that $f_{t,d}$ is uniformly continuous w.r.t. to its parameters $(a_{t,d}, b_{t,d})$, regardless of the conditioner. The line of arguments goes as follows:

1. The derivatives of $f_{t,d}$ (Definition 4.2.3) w.r.t. a_t,

$$\frac{\partial f_{t,d}}{\partial a_{t,d}} = \begin{cases} z_{t,d} - b_{t,d}, & \text{if } z_{t,d} - b_{t,d} > 0 \\ 0, & \text{otherwise.} \end{cases}$$

2. As the derivative of $a_{t,d}$ always exists and using Lemma 4.4.6 for continuity in $b_{t,d}$, means $f_{t,d}$ is continuous w.r.t. its parameters.

3. The input to either $\mathcal{H}_{t,d}$ or Ψ is, as already noted, compact. Both conditioner are continuous, $\mathcal{H}_{t,d}$ due to Lemma 4.4.5 and Ψ follows from Definition 2.4.5.

4. Compactness and continuity of function implies compactness w.r.t. output, hence the parameter space—or output space of the conditioner—-is compact.

5. Continuity in $f_{t,d}$ w.r.t. the parameters combined with the fact that the parameters are compact, implies uniform continuity of $f_{t,d}$ w.r.t. $(a_{t,d}, b_{t,d})$.

Uniform continuity combined with convergence of the network, means there exist for all $z_{0,d} \in [k_0^d, k_1^d]$ and for every $\epsilon/2 > 0$, a $\delta > 0$ (by picking $l_d > L_d$ large enough), there exists a network $\Psi \in \mathcal{NN}_{[l_d, d+3, \gamma]}$ such that

$$\begin{aligned} &\|\Psi(\boldsymbol{z}_{0,1:d-1}) - \mathcal{H}_{t,d}(\boldsymbol{z}_{0,1:d-1})\|_\infty < \delta \\ &\implies |f_{t,d}(z_{0,d}; \Psi) - f_{t,d}(z_{0,d}; \mathcal{H}_{t,d})| < \frac{\epsilon}{2} \end{aligned}$$

where $\mathcal{H}_{t,d}$ is as specified in Lemma 4.4.5.

By combining this with Lemma 4.4.5, we see that for all $z_{0,1:d} \in [k_0, k_1]^d$ and for every $\epsilon > 0$, there exist a $T_d \in \mathbb{N}$ and $L_d \in \mathbb{N}$ such whenever $T > T_d$ and $l_d > L_d$, there exists a set of T networks

$$A = \{\Psi^{(t,d)} \colon \Psi^{(t,d)} \in \mathcal{NN}_{[l_d, d+3, \gamma]} \text{ and } t \in \mathcal{T}\}$$

s.t.

$$\begin{aligned} |g_{T,d}(z_{0,d}; A) - G_d(z_{0,d}, \boldsymbol{z}_{0,1:d-1})| \\ \leq & |g_{T,d}(z_{0,d}; A) - g_{T,d}(z_{0,d}; \mathcal{H}_{t,d})| + \\ & |g_{T,d}(z_{0,d}; \mathcal{H}_{t,d}) - G_d(z_{0,d}, \boldsymbol{z}_{0,1:d-1})| \\ < & \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon. \end{aligned}$$

A small note on A. Each network approximate their own $\mathcal{H}_{t,d}$, which is indicated by the superscript of the networks in A.

Finally, we combine the results above for each $d \in \mathcal{D}$ and acquire the following result. For all $\boldsymbol{z}_0 \in [k_0, k_1]^D$ and for every $\epsilon > 0$ there exists a $L = \max\{L_d\}_{d=1}^D$ and $\tilde{T} = \max\{T_d\}_{d=1}^D$, such that $(l_d > L)_{d=1}^D$ and $T > \tilde{T}$, there exists a set of $T \cdot D$ networks

$$A = \{\Psi^{(t,d)} \colon \Psi^{(t,d)} \in \mathcal{NN}_{[l_d, d+3, \gamma]} \text{ and } t \in \mathcal{T}\}$$

s.t.

$$\|g_T(\boldsymbol{z}_0; A) - G(\boldsymbol{z}_0)\|_\infty < \epsilon.$$

■

Before we comes to the final theorem, we need to make sure the canonical triangular map is continuous and increasing w.r.t $z_{0,d}$, but also continuous w.r.t $\boldsymbol{z}_{0,1:d-1}$. It is fine for the first part of the canonical triangular map, as it is the conditional CDF $F_{\mathcal{Q}}(z_{0,d} \mid \boldsymbol{z}_{0,1:d-1})$. We need to make sure the same holds for the second part, the inverse of $F_{\mathcal{P}}^{-1}$. As the result is more about a class of probabilities, the proof is a bit technical, and does not regard flows and therefore gives no insight to flows, we have relegated the proof to Appendix A.5.

Lemma 4.4.8. *For any probability distribution $\mathcal{P} \in \mathscr{P}_2$, let the conditional CDF be denoted by $F_{\mathcal{P}}(x_d \mid \boldsymbol{x}_{1:d-1})$ for any $d \in \mathcal{D}$. Then the inverse $F_{\mathcal{P}}^{-1}(u_d \mid \boldsymbol{x}_{1:d-1})$, where $u_d \in (0, 1)$ is strictly increasing and continuous w.r.t both u_d and $\boldsymbol{x}_{1:d-1}$.*

Once we have this continuity and strictly increasing, we can prove the main theorem using the canonical triangular map, i.e., we can prove universality for $\mathscr{P}_2$.

Theorem 4.4.9. *Let $\mathcal{NF}$ be the flow space with*

- *an IAR structure without permutation,*
- *limited piecewise affine transformations,*
- *$\mathcal{NN}_{[L, \tilde{D}, \gamma]}$ as conditioner space,*

- $\{\tilde{D}, T\} \subseteq \Phi$, *i.e., the width of the hidden layers in the conditioner, and the number of transformations* T.

Then $\mathcal{NF}$ *is a UDA for* $\mathscr{P}_2$.

Proof. Let $\mathcal{P} \in \mathscr{P}_2$, $F_{\mathcal{Q}}^{(d)}$ be the conditional CDF for $z_{0,d}$ conditioned on $\boldsymbol{z}_{0,1:d-1}$, and equivalently for $F_{\mathcal{P}}^{(d)}$. By Lemma 4.4.8, the definition of a CDF, and the fact that $\boldsymbol{z}_0 \in [k_0, k_1]^D$, we have that the canonical triangular map $[F_{\mathcal{P}}^{(d)}]^{-1} \circ F_{\mathcal{Q}}^{(d)}(z_{0,d} \mid \boldsymbol{z}_{0,1:d-1})$ is continuous w.r.t both $z_{0,d}$ and $\boldsymbol{z}_{0,1:d-1}$, and strictly increasing. Then we have that $G_d = \sigma \circ [F_{\mathcal{P}}^{(d)}]^{-1} \circ F_{\mathcal{Q}}^{(d)}$ fulfills the requirements we specified at the start. From Lemma 4.4.7, we know that $g_T = \sigma \circ f$ can approximate $G = (G_1, \dots, G_D)$ arbitrarily well using neural networks as conditioner, and it converge uniformly towards G. This implies pointwise convergence towards $\sigma^{-1} \circ G = (\sigma^{-1} \circ G_1, \dots, \sigma^{-1} \circ G_D)$, which implies due to Lemma 4.4.2 that

$$f(\boldsymbol{z}_0) = \boldsymbol{z}_T \xrightarrow{d} \boldsymbol{x} = G(\boldsymbol{z}_0) \sim \mathcal{P}.$$

■

With this result we have shown that with a slight modification on the affine transformation, we have universality for broad classes of distributions, for both $D = 1$ and structures without use of permutations. Although the results were of larger importance before Teshima et al. 2020 were published and before we became aware of their excellent results, it still has its place as the first proof using arbitrary many time steps T, rather than number of parameters in transformations (e.g. number of knots in spline transformations), while using no permutations in the structure. Now, as already discussed before, universality of flows does not imply good performance in a finite situation. Both due to the asymptotic nature, but also due to the existence part, i.e., there exists a flow and there exists a set of neural networks.

In the end of this section, we have a couple of results that follows quickly from the theory above, and are neat in their own right. Firstly, we can combine the UDA results above with the universality of CONNs. We implicitly assume that any CONNs we use follows the way they are described in Theorem 4.3.2, Theorem 4.3.6, or Corollary A.2.1.

Corollary 4.4.10. *Let* $\mathcal{NF}$ *be normalizing flow space which is UDA for a distribution space* $\mathscr{P}$ *and the conditioner space is comprised of neural networks. Then the same normalizing flows space where the conditioner space comprises of CONNs is a UDA for* $\mathscr{P}$ *almost surely.*

Another result stems from the fact that flows with limited piecewise affine transformations are UDA for $\mathscr{P}_1$ when $D = 1$, can easily be extended to multidimensional, as $\mathscr{P}_1$ specifies independence between the variables in all distributions $\mathcal{P} \in \mathscr{P}_1$. We can then think of simply letting each dimension in the flow approximate its own one dimensional distribution. This is equivalent to setting the weights in the first layer of the neural network that computes the conditioner, to 0. It then becomes easy to prove that each dimension can approximate its own independent distribution, with similar approach as in Theorem 4.4.4. We therefore have:

Corollary 4.4.11. *Let $\mathcal{NF}$ be the flow space with*

- *an IAR structure without permutation,*
- *limited piecewise affine transformations,*
- *$\mathcal{NN}_{[L,\tilde{D},\gamma]}$ as conditioner space,*
- *$\{\tilde{D}, T\} \subseteq \Phi$, i.e., the width of the hidden layers in the conditioner, and the number of transformations T.*

$\mathcal{NF}$ is a UDA for $\mathscr{P}_1$.

This again highlights the difference between affine and the small change we introduce. If the target distribution contains independent dimensions, there is a more direct way to approximate it with our flows (at least theoretically), compared to first correlating and then de-correlating variables. It also highlight the possibility, if we know something about the dependence of different dimensions—either through learning from observations or a priori—we can use simpler structures such as $\mathcal{S}_{ext}(t, d) = \emptyset$ for all $t \in \mathcal{T}$, when d is independent from the others.

We are now done with the universality part of this thesis. Before we move on to the empirical testing, we take a look at how to enforce positive parameters as output of the conditioner.

4.5 a-activation function

In the last section of this chapter, we turn our focus toward a small, yet important part of transformations, namely when the parameter computed by the conditioner needs to be positive, e.g., $a_{t,d}$ in the affine transformation. There are many valid options here, and many of them are theoretically they are equivalent, but practically can be extremely important. For instance, any function $g_a \colon \mathbb{R} \to \mathbb{R}_*^+$ is theoretically sufficient, but using for instance exp, any small change of the parameters in the conditioner, can drastically alter $a_{t,d}$ in the transformation. We therefore find it useful to investigate and introduce some useful functions.

Definition 4.5.1. Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow, and where $f_{t,d}$ contains parameter $a_{t,d} > 0$. We then define *a-activation function* $g_a \colon \mathbb{R} \to \mathbb{R}_*^+$ as the function applied to outputs of the conditioner $a_{t,d}$.

The activation function part of the name comes from the fact that the conditioner typically is a neural network, and then g_a becomes the last activation function in the network.

The two most prominent a-activation functions when it comes to affine transformation, and the parameter $a_{t,d}$, has been exp (Papamakarios et al. 2017) or σ (Kingma, Salimans et al. 2016). The former often invoke highly unstable flows during training, as small fluctuations in the weights give high values $a_{t,d}$, which result in extreme values in the loss and the gradient, and we end up with exploding gradients. The latter, σ, creates very stable flows during training, but of course limits the output space to $0 < a_{t,d} < 1$, and can have damaging effects, as seen in Section 3.6.

Another of the well known a-activation function is the Softplus, which is defined as

$$\text{Softplus}(a_{t,d}) = \log(1 + \exp(a_{t,d})).$$

This allows for $a_{t,d} \in \mathbb{R}_*^+$, while the growth is a lot slower than exp, hence not as rapid change when optimising weights in the network/conditioner. However, this can still induce exploding gradients, and as another alternative, we introduce an even slower growing function *Slowplus*

$$\text{Slowplus}(a_{t,d}) = \begin{cases} \text{arcsinh}(a_{t,d}) + 2\log(2), & a_{t,d} \geq 0 \\ 2 \cdot \text{Softplus}(a_{t,d}), & a_{t,d} < 0, \end{cases}$$

where arcsinh is the inverse hyperbolic sine $\log(a_{t,d} + \sqrt{a_{t,d}^2 + 1})$. The added constants in Slowplus is there to align the two pieces at $a_{t,d} = 0$, with also same derivative (1). This can be an extra tool to apply before settling for σ.

In continuous piecewise affine, the a-activation function is a little bit different, as we let the value pre a-activation function decide how to apply $a_{t,d}^+$, where we want it to be symmetric for positive and negative side, as the pre$a$-activation function value $a_{t,d}$ uses $a_{t,d}^+$ when $a_{t,d} > 0$ and $1/a_{t,d}^+$ otherwise. Hence, using similar functions as described above, makes it hard to approximate the transformation when the gradient of the linear piece is less than one. However, using simply the absolute value enforce similar exploding gradient problems, and we therefore want to give a smaller growth in the a-activation function for continuous piecewise affine transformation as well ($a_{t,d}^+$). We therefore introduce *Slowabs* as

$$\text{Slowabs}(a_{t,d}) = |\text{arcsinh}(a_{t,d})|.$$

The arcsinh stretches the input out, enforcing smaller growth before we apply the absolute value, while still mapping $\mathbb{R}$ to $\mathbb{R}_*^+$.

Conclusion

We are now at the end of this chapter, where we have seen a few important distinctions between larger groups of transformations. We then took this concept, and introduced some transformations that tries to fill the gap between the simpler ones in one group of transformations, and the more complex ones in the other two groups (unbounded parametrisation/inflection transformations). Also showing both universality to the important group of networks, CONNs, and the UDA of limited piecewise affine transformation, as the first to be shown where we required number of transformations to increase, while using no permutations. At the end we considered a few different parametrisations of $a_{t,d}$, which needs to be carefully considered in the next chapter, which is testing our new introduced parts in empirically.

CHAPTER 5

Empirical Results

5.1 Introduction

In this chapter we take the theory developed, and put it into practice, by running 4 different experiments. We start off by giving an overview of of the implementation used for the experiments, notes on optimising matrices in $SO(D)$, and general information on how the experiments were run.

We then move on to experiment 1 and 2, which is performed by testing 42 different flows on some known target distribution, to find general patterns and broad stroke conclusions. We then find a set of few flows that perform well and continue testing them in experiment 3 and 4, where experiment 3 run our proposed transformation on some benchmark, and we compare it to results from the literature. Experiment 4 contains a Bayesian model, were we test the variational inference side of flows.

In addition to testing our proposed transformations, we also experiment with alternating linear transformations and ours/affine transformations. This have had remarkable results with more complex transformations such as spline transformations (Durkan et al. 2019b), and we are interested in seeing if this have similar effect for simple transformations such as continuous piecewise affine transformations and affine transformations.

5.2 Implementation

In the following experiments we have run, we have written a code base that reflects the formal definitions introduced through this thesis. We found it not only necessary to reflect our work, but as existing code online were very much different complete flows, hard to change or create new flows out of. This means that we have separated the code into several parts, with transformations, structures, conditioners, permutations, and base distributions are separated. With this, one can easily implement new transformations for instance, and pass whatever already implemented structures etc. one wish to use into the main flow class and run. This makes it possible to rapidly create different models, and therefore be able to better test each component, e.g., run similar models but change the permutation scheme. The entirety of the code can be found here **https://github.com/Watakani/Master**, while we run through the implementations components in broad strokes here.

Before we delve into our contribution we run through the use of existing code. The core framework is written on PyTorch v.1.10.0, which we use to compute the underlying computational graphs, parts of the neural networks models, e.g., normal, noisy, MADE/CONN, and functions like exponential etc., as well as the base distribution to sample and evaluate density. We are also using a R package (Vehtari, Gabry et al. 2020) to calculate some variational statistic, by piping the python code into R and run the package, returning the results into python.

We can now move onto what we have contributed to the code. We have a two classes on top, namely a class that is the flow, which then takes as input a list with T structures/transformations, with the classes main task is to compute forward and backward flow—both when updating parameters and after training—and also add base distributions log density to the final log density. The second large part is the training code, which allow user to pass a flow, an updating scheme which are implemented in PyTorch (e.g., Stochastic Gradient Descent, ADAM), and other hyperparameters such as number of epochs, batch sizes etc., and the code then train the flow, and can also save the best model/checkpoints.

The first two parts described above, allows us to implement the different part of flows, and assemble them into a list and pass them on to the parts described above. We are now giving an overview of what exactly is implemented in the code.

- Base distribution: A class that simply takes in any PyTorch implemented distribution, and allow for both sampling and evaluating log density. If nothing is explicitly stated, it assumes a joint independent standard Gaussian distribution.
- Conditioner: We have implemented two conditioners, depending on what structure, and are initiated from the classes of structures. They all take in a dimension D, and number of parameters ρ needed in the transformation, and output $D \cdot \rho$ vector. The following conditioners are implemented:
  - Vanilla neural network to be used in coupling structures, and one dimensional cases.
  - MADE/CONN, one very specific version of MADE with Uniform sampling of the masks at the initialisation, and a CONN class that are more general and can take any mapping c, as described in Section 2.5. The former is a bit faster, and is preferred in the autoregressive case as of now, as it easier to implement and the implementation was based of Kaparthy 2018.
- Transformations: Any transformation must inherit from the Transformation class, which means there is very little that needs to be implemented for each transformation. Only forward and inverse transformation, taking input z or x and parameters, which the structure class takes care of. The following transformations have been implemented:
  - Affine Transformation,
  - limited Piecewise Affine Transformation,

  - Continuous Piecewise Affine Transformation,
  - Affine limited Piecewise Affine Transformation,
  - Affine Continuous Piecewise Transformation,
  - Constant Affine Transformation, simply letting the parameters $(a_{t,d}, b_{t,d})$ be constants, which together with a independent Gaussian base distribution gives a similar model to the standard mean field.
  - Linear Transformation, with translation, and where the matrix A is in $SO(D)$, i.e., orthonormal. We discuss further how we train such matrices later.

- Structure: As the flow class described in the last paragraph as composing of T structures/transformations, we implement structure as taking in a specific transformation for step t, and hyper parameters to the conditioner, and initialise conditioners. The structure takes in z or x, depending on forward or inverse computing, and compute the correct parameters before passing everything on to the specific transformation. The structures implemented are:
  - TwoBlock, is a $D/2$-coupling structure,
  - AR,
  - IAR,
  - Identity/Fully Connected.

- Permutations: We have implemented three different permutation schemes to use in the structures, which are the identity permutation, alternating (the first dimension becomes the last, the last becomes the first, the second becomes the second to last...), and random which generates the permutation through sampling.

This comprises the main parts of our implementation. We need to discuss some details concerning the linear transformation.

Linear Transformation Implementation

The challenge with linear transformations are first to consider what group A should adhere to—as it is unfeasible to optimise over the general linear group (invertible matrices)—and secondly choose how to optimise over the group we pick. We ended up with orthonormal matrices A, that is, optimising over the $SO(D)$ group. Secondly, how to actually optimise over it.

The actual method is beyond the scope of this thesis, however, we give a rough intuition for it here. One can parameterise a matrix B with constant 0 in the lower triangular, constant 0 along the diagonal, and parameters in the rest. One can then calculate a skew-symmetric matrix $B_s = B + -B'$ (where skew-symmetric is when $B_s = -B'_s$, i.e., it is equal to the negative of the transposed). With this skew-symmetric matrix, taking the matrix exponential maps $A = \exp(B_s)$ to the space of $SO(D)$, i.e., space of orthonormal matrices. This means that we can optimise over the $(D^2/2 - D)$ parameters in B, where each parameter can be any real value, and end up with a special orthonormal matrix. Due to recent improvements, the matrix exponential function has

become feasible (Bader et al. 2019). To acquire a well-performing optimisation, one use Riemannian gradient descent on B, which details can be found by Casado 2019. We rely on an implementation from Aasan 2021.

Experiments

Couple of notes on the experiments: Every experiment, and hence every result in tables, was written in a Jupyter Notebook, and are available at the same place as the code. This heightens the experiments reproduceability. The notebooks can be found in the source code, in the first folder, where the start of the name indicates which experiment, e.g., **experiment_1_*.ipynb**.

We trained each model minimising log-likelihood for density estimation, and minimising negative ELBO for variational inference experiment, as described in Section 2.2.

The final model in any training is chosen to be the best one during the run, relying on regularisation/learning rate/number of epochs to avoid overfitting. We use PyTorch's AdamW as an optimiser.

Also, all the experiments were done on the University of Oslo's machine learning nodes (ML nodes).

5.3 Experiments 1 & 2

We start off by testing different flows on toydata, also referred to as generated data. The objective of these two experiments is to be able to test many different flows, and gather a large overview of what properties good vs. worse models have. We therefore get conclusions that are broad, and through this we narrow down the number of flows to a few, which we proceed to test further.

We have chosen two target distributions, one that are relatively easy and one rather difficult distribution. The first target distribution $\mathcal{P}_1$ is a multivariate Gaussian with 25 dimensions, with randomly sampled means between 0 and 8, variance for each dimension of 3, and 0.8 in covariance between every dimension (the covariance matrix have 0.8 everywhere except for diagonal, which contains 3.

The second target distribution $\mathcal{P}_2$ is a 50 dimensional, all independent, with every dimension being an Exponential distribution, where the rate is randomly sampled between 0.5 and 3 for each dimension. These are quite tricky, as the marginal distribution differs much more from the base distribution than in $\mathcal{P}_1$.

A small remark on how we generated the data: With $\mathcal{P}_1$, we sampled both the mean and the dataset of 10 000 samples for each bulk of models. We ought to have sampled one dataset, however, it did not seem to affect the results due to the high number of samples. With $\mathcal{P}_2$, we use the same parameters, but also generate 10 000 new samples for each bulk of models. However, as long as the parameters are the same, with such high number of samples, it seems like it also here have not affected the results. Also, due to the number of models we test here, it was not possible to re-run them all, and we therefore mention it, while it do not seem to affect the results.

We have gathered up the different components we have implemented, i.e., all the structures, transformations, and permutations, and created many

combinations, which we now list Table 5.1, where each model may have several different permutations (permutation π_t in the structure), which means we have tested flows with all every permutation listed. Hence, flow 3 has 3 different flows, one with identity permutation, alternating permutation, and random permutation (i.e., sample π_t uniformly).

When two structures, two transformations or two permutations, are separated by / means we alternate between them. For example, flow 15, when $t = 1$ we apply affine transformation, $t = 2$ applies limited piecewise affine transformation, and so on. Every flow will still have the same number of transformations, regardless of alternating or not.

Table 5.1: List of all models tested in experiment 1 and 2.

	Structure	Transformations	Permutations
Flow 1:	Identity	Affine	Id.
Flow 2:	Fully	Linear ($SO(D)$)	Id.
Flow 3:	$D/2$-coupling	Affine	Id., Alt., Random
Flow 4:	AR	Affine	Id., Alt., Random
Flow 5:	$D/2$-coupling	Piecewise Affine	Id., Alt., Random
Flow 6:	AR	Piecewise Affine	Id., Alt., Random
Flow 7:	$D/2$-coupling	Affine Piecewise Affine	Id., Alt., Random
Flow 8:	AR	Affine Piecewise Affine	Id., Alt., Random
Flow 9:	$D/2$-coupling	Cont. Piecewise Affine	Id., Alt., Random
Flow 10:	AR	Cont. Piecewise Affine	Id., Alt., Random
Flow 11:	$D/2$-coupling	Affine Cont. Piecewise Affine	Id., Alt., Random
Flow 12:	AR	Affine Cont. Piecewise Affine	Id., Alt., Random
Flow 13:	$D/2$-coupling	Affine/Piecewise Affine	Random
Flow 14:	AR	Affine/Piecewise Affine	Random
Flow 15:	ID/($D/2$)-coupling	Linear/Affine	Id., Random
Flow 16:	ID/AR	Linear/Affine	Id.,Random
Flow 17:	ID/($D/2$)-coupling	Linear/Piecewise Affine	Random
Flow 18:	ID/($D/2$)-coupling	Linear/Cont. Piecewise Affine	Random
Flow 19:	ID/($D/2$)-coupling	Linear/Affine Piecewise Affine	Random
Flow 20:	ID/($D/2$)-coupling	Linear/Affine Cont. Piecewise Affine	Random

For each model, we calculate the mean log-likelihood for both the training set and a test set, that is,

$$\hat{\ell}_{tr} = \frac{1}{n_{tr}} \Sigma_{\boldsymbol{x} \in \mathcal{X}_{tr}} q_{\boldsymbol{z}_T}(\boldsymbol{x}),$$

where $\mathcal{X}_{tr}$ is the training set with n_{tr} examples. Similarly with test set,

$$\hat{\ell}_{ts} = \frac{1}{n_{ts}} \Sigma_{\boldsymbol{x} \in \mathcal{X}_{ts}} q_{\boldsymbol{z}_T}(\boldsymbol{x}).$$

We run each model 5 times, and calculate the statistic with a 95% confidence interval, estimated using the t-score, i.e.,

$$[\hat{\ell}_{tr}^{(0.025)}, \hat{\ell}_{tr}^{(0.975)}] = \hat{\ell}_{tr} \pm t_{0.975} \frac{s}{\sqrt{n_{tr}}}$$

where s is the unbiased standard deviation estimate, and $t_{0.975}$ is the 0.975 percentile of a Student-t distribution with $n_{tr} - 1$ degrees of freedom—with similar statistic calculated for test set. This gives us a rough estimation on how the flow is affected by the initiation of weights in the conditioners etc.

Table 5.2: Overview of hyperparameters in experiment 3, for the two datasets

	HEPMASS	BSDS300
N_{train}	10 000	10 000
N_{test}	1000	1000
Dimension	20	50
Learning rate	1e-4	1e-4 (1e-3*)
Weight decay	1e-3	1e-2 (1e-1*)
Size of conditioner	$100 \lessapprox 100 \lessapprox 100$	$200 \lessapprox 200 \lessapprox 200$
T	6	8
Num. Epochs	Until convergence	25

Combining all models and the 5 runs, means we train $42 \cdot 5 = 210$ flows. This is of course way too many to present all, and also too many to use time tuning hyper parameters etc., so we found a standard set of parameters while running the experiments.

Results

We have added the results for every model given in Table 5.1 in Appendix A.6. There are some takeaways to get from the experiments:

- Nearly every flow estimate the multivariate Gaussian in experiment one perfect, in terms of mean log-likelihood. It is in experiment two that we find the real differences.
- Even though we have implemented $D/2$-coupling structures, such that we transform the complete $\boldsymbol{z}_{t-1}$ in one transformation step (first $z_{t-1,D/2:D}$ and then use the new transformed variables tot transform $z_{t-1,1:D/2}$), which means effectively have twice the amount of transformations as AR-structures, yet it do seem that AR generally performs as good or better. There are some flows that did better with $D/2$-coupling layer, but there are other factors that may alter this such as a-activation function, and more testing must be done to conclude. However, generally AR perform as well or better, which is supported in the literature as well (Papamakarios et al. 2017). Taken into account the conditioner have the same number of weights, with AR having in effect fewer weights due to the use of masks, reaffirm that the the structures are important.
- There seem to be no significant difference between the permutations in the structure, however, where we find difference is in affine transformations, which do slightly better with permutations that are non-identity. The lack of difference can also stem from the fact that experiment 1 was too easy, and we have independence in experiment 2 which means permuting the structure from time step to time step is not necessary.
- limited piecewise affine transformations (Flow 5-6) struggles in experiment 2. There are two issues we suspect. The first is the fact that it struggles to move points around, as compared to affine transformations, it relies too much on $a_{t,d}$ to do the heavy lifting. In affine transformations we can

Table 5.3: The best results for experiment 2, in addition to the best among the flows using only affine transformations. Mean log-likelihood, which means higher is better.

	Train exp. 2	True train value	Test exp.2	True test value
Flow 4-Alt	-48.60 ± 0.768	-30.70	-48.86 ± 0.674	-30.77
Flow 11-Alt	$\mathbf{-31.76 \pm 0.442}$	-30.68	$\mathbf{-33.74 \pm 0.134}$	-30.77
Flow 15-Id	-39.84 ± 1.328	-30.78	-44.58 ± 0.202	-30.72
Flow 20	-37.67 ± 0.707	-30.74	-44.12 ± 0.440	-30.84

move points around with $b_{t,d}$ in a much freer way than in lim. piecewise affine. However, as affine lim. piecewise affine transformations (Flow 7-8), as well as alternating with linear transformations (Flow 17 & 19), also struggle seems to imply that there are more than one issue. We suspect it comes down to the discontinuity in the derivative, in particular due to the next point.

- Continuous piecewise affine transformations also struggle alone, however, affine continuous piecewise affine transformations perform much better, and is clearly the best in experiment 2, both w.r.t training set and test set. It therefore seems like the addition of continuity in the transformations derivative helps, as lim. piecewise affine did so poorly.
- Alternating linear layers and conditioner transformations is immensely helpful, even though it has way fewer parameters (half the number of conditioners required and linear layer have fewer than D^2 trainable parameters). In experiment 2, Flow 15-20, performs best except for affine continuous piecewise affine transformations.

The conclusion above must be interpreted as broad conclusions, and that each point deserve further study to really confirm and specify the points above. The experiments still highlight some important points, and in particular how piecewise affine transformations perform poorly, while the continuity added improve the performance a lot. We have added the best results from experiment 2 (only considering one permutation for each group of transformations/structures), as well as the best using only affine transformations in Table 5.3.

When the different flows were trained, we also tested with different a-activation functions, where we tried to use the ones with high growth first, e.g., exp and Softplus, and downgrading to slower growing functions if the training was too unstable (exploding gradients). However, this may also give a skew view of the results, even though one can view the models ability to handle different a-activation functions as a good characteristic. The slower growing functions may also not be worse, even if the faster growing works for a particular flow. We therefore re-run experiment 2 with affine transformations and affine continuous piecewise affine transformations, with the a-activation functions used in the latter, namely, Slowplus (remember that the affine-continuous piecewise transformation is a composition of continuous piecewise affine and affine transformation after. We use Slowplus on the affine transformation part, and we use Slowabs for the scale parameter in the continuous piecewise part).

To allow for better comparison, more one to one comparison, we tested four models, one with affine transformations and $D/2$-coupling structure, and

Table 5.4: Results from experiment 2, with the same a-activation function. Mean log-likelihood, which means higher is better.

	Train exp. 2	True train value	Test exp.2	True test value
Affine-$D/2$	-41.47 ± 0.771	-30.82	-45.41 ± 0.546	-30.97
Affine-AR	-48.51 ± 0.222	-30.81	-48.77 ± 0.2447	-30.54
Aff-Cont.-$D/2$	$\mathbf{-31.47 \pm 0.477}$	-30.82	-34.24 ± 0.232	-30.97
Aff-Cont.-AR	-32.65 ± 0.114	-30.81	$\mathbf{-32.69 \pm 0.440}$	-30.84

one with AR structure. Equivalently with affine-continuous piecewise affine transformation. The permutation used were alternating. The results can be found in Table 5.4. They still show a significant improvement for our proposed transformation compared to affine transformations, and although our proposed transformation have 4 parameters in its transformation, compared to two, the number of parameters in the conditioner are the same, hence the two extra parameters increases the complexity of the flow by very little. This confirms both the Observation 3.6.12 pointing out problems with affine transformations and independence and how our—considering all transformations introduced in Section 4.2 are close to being lim. piecewise affine transformations—transformations UDA properties support independence by Theorem 4.4.4 and Corollary 4.4.11.

As already mentioned, $D/2$-coupling structures essentially have twice the amount of transformations due to how they are implemented, and also seem to overfit slightly, and a bit more unstable to initial flows. Compared to the previous results in Table A.3, the largest difference are between Affine-$D/2$, which used the Sigmoid function as a-activation function—Softplus was too unstable—while Affine-AR is almost exactly the same, which used Softplus in the first run. Hence, Slowplus do not seem to hamper the performance, but can be used also when Softplus is too unstable.

5.4 Experiment 3

In experiment 3 we have run two different datasets from UCI repository, which was chosen as they are used for testing of flows in the literature. There are typically 5 of them, but we managed only to run 2 of them, both due to time constraint and unstable training, which we discuss further in Section 6.2. The dataset, with the appropriate pre-processing according to Papamakarios et al. 2017, can be found by running **get_data.sh** in the first folder of our code, and initialise the class of the corresponding dataset in the folder **NormalizingFlowssrcdata/density**. This will then save the pre-processed data to **NormalizingFlowsdatapreprocessed**.

We run three different flows, running 3 of each to estimate the confidence interval, where we estimate the CI in the same manner as with the previous two experiments. The models are the following:

- Affine continuous piecewise affine with AR structure, and alternate permutation of structures (AffCon-AR).
- Alternating with linear transformation and affine transformations, with AR structure and alternate permutation of structures (AltLinAff-AR).

Table 5.5: Overview of hyperparameters in experiment 3.

	Experiment 1	BSDS300
N_{train}	315 123	1 000 000
N_{test}	174 987	250 000
Dimension	21	63
Learning rate	1e-4	1e-4
Weight decay	0	0
Size of conditioner	$512 \approprox 512$	$512 \approprox 512$
T	10	10
Num. Epochs	20	8
Batch size	32	32

Table 5.6: The best results for experiment 3. Number in parenthesis indicates number of time steps T used, and * indicates results are copied from Huang, Krueger et al. 2018. Mean log-likelihood, which means higher is better.

	Train HEPMASS	Test HEPMASS	Train BSDS300	Test BSDS300
AffCon-AR	-17.02 ± 0.116	-17.58 ± 0.215	164.69 ± 0.295	155.11 ± 0.215
AltLinAff-AR	-18.77 ± 0.141	-19.18 ± 0.191	160.71 ± 0.537	151.56 ± 0.146
AltLinCon-AR	-18.30 ± 0.080	-18.71 ± 0.114	161.33 ± 0.516	151.92 ± 0.308
Aff-AR* (5)	—	-17.70 ± 0.02	—	155.69 ± 0.28
Aff-AR* (10)	—	-17.73 ± 0.02	—	154.93 ± 0.28
Ψ^{+}* (5)	—	$\mathbf{-15.09 \pm 0.40}$	—	$\mathbf{157.73 \pm 0.04}$
Ψ^{+}* (10)	—	-15.32 ± 0.23	—	157.43 ± 0.30

- Alternating with linear transformation and affine continuous piecewise affine transformations, with AR structure and alternate permutation of structures (AltLinCon-AR).

With every scale parameter $a_{t,d}$ in the affine transformation parts uses Slowplus and the scale parameter $a_{t,d}$ in the continuous piecewise affine part uses Slowabs.

The two datasets we are working with, is denoted in the literature as HEPMASS and BSDS300. The hyperparameters used for each dataset, is given in Table 5.5, where most of the hyperparameters were chosen to match Huang, Krueger et al. 2018.

The results are given in Table 5.6, where * indicates results from Huang, Krueger et al. 2018. We included both the best affine transformations results we are aware of, but also neural network transformations as a comparison to the much more complex transformations. We find that affine continuous piecewise affine transformations are working pretty well, with even beating the best affine transformation results in HEPMASS that we are aware of. In addition, the results from the literature indicates that $T = 5$ performs better than with our $T = 10$, and we also find that there are signs of overfitting slightly, and applying some hyperparameter tuning may aid this and improve test results.

The much higher uncertainty, although not every uncertainty is calculated the same way, is concerning. We have similar uncertainty with neural network transformation Ψ^{+}, which can indicate that more complex transformations carries with more uncertainty in how they are initiated. We have, similarly to Huang, Krueger et al. 2018, used no regularisation, which can also be an

explaining factor as the weights in the conditioner is much more free to roam the space induced by the loss function, and hence may lead to quite different flows depending on what weights we start with. However, considering both this and the previous two experiments as well, we find that affine transformations can achieve similar large confidence intervals, and in particular AltLinAff has large intervals as well, which may indicate difference in how the uncertainty estimates are measured. This implies that further investigation is needed, and that we cannot conclude that more complex transformations are prone to more uncertainty w.r.t initialisation and stochasticity in the optimisation.

The much poorer performance when alternating with linear layers is a bit contradictory compared to how well they work with splines (Durkan et al. 2019b). One must add that flows such as AffCon uses more trainable parameters, as each conditioner have 512 neurons in each hidden layer, while each linear layer has only $D^2/2 - D$ trainable parameters in total. It is therefore slightly more difficult to acquire a one-to-one comparison between, for instance, AltLinAff-AR and AffCon-AR. As of now, using the same T favours the latter, while double T for AltLinAff-AR would favour it. There might also be difficulties by working in $SO(D)$, that are too limiting. We thought that combining linear transformation with matrices in $SO(D)$, together with more expressive transformations would render the limitation of $SO(D)$ moot, but perhaps not.

All in all, we see that our proposed affine continuous piecewise affine performs as good as state of the art results for affine transformations, but with room for improvement as well. The results for Aff-AR (Papamakarios et al. 2017) uses faster growing a-activation functions and incorporating batch normalization. Hence, there may be improvements switching our a-activation function that gives us stable training, with Softplus/exp and batch normalization.

5.5 Experiment 4

In this last experiment we run a small variational inference experiment. The flows we test are the following:

- Affine transformations with identity structure (Base)—as a baseline.
- Affine transformations with IAR structure, and alternate permutation of structures (Aff-IAR).
- Affine continuous piecewise affine with IAR structure, and alternate permutation of structures (AffCon-IAR).
- Alternating with linear transformation and affine transformations, with IAR structure and alternate permutation of structures (AltLinAff-IAR).
- Alternating with linear transformation and affine continuous piecewise affine transformations, with IAR structure and alternate permutation of structures (AltLinCon-IAR).

We use IAR structures, and not AR structures, due to the fact that we train by sampling, and therefore want to vectorize the forward flow part to have fast training. Furthermore, with every scale parameter $a_{t,d}$ in the affine transformation parts uses the Sigmoid function σ (except for Base which uses Softplus) and the scale parameter $a_{t,d}$ in the continuous piecewise affine part

Table 5.7: Overview of hyperparameters in experiment 4.

	Experiment 4
N_{train}	10 000
N_{test}	1000
Dimension	101 (100 β and σ)
Learning rate	1e-4
Weight decay	1e-2
Size of conditioner	$300 \nleftarrow 300 \nleftarrow 300$
T	8
Num. Epochs	10
Batches/Batch size	2000/32

uses Slowabs. The Sigmoid function was necessary to stabilise the training—we did attempt with faster growing a-activation function with resulting instabilities.

The model for the experiment is of linear regression following the same setup as section 4.1 in Yao et al. 2018. Let $D = 100$ and $\boldsymbol{x}$ are standard normal random vector (sampled from multivariate Gaussian with 0 mean an identity matrix as covariance)

$$\boldsymbol{\beta} = \{\beta_k\}_{d=1}^{D} \sim \mathcal{N}(0, 1), \quad \sigma \sim \text{gammma}(0.5, 0.5)$$
$$y \sim \mathcal{N}(\boldsymbol{x}^T\boldsymbol{\beta}, \sigma^2)$$

We report two statistics for this experiment. The first is the well-known ELBO. However, as pointed out by Yao et al. 2018, it is hard to interpret too much out of this, as firstly it can drastically change by different parametrisation of models (due to the the unknown marginal probability term in $p(\boldsymbol{\beta}, \sigma, y \mid \boldsymbol{x})$. Secondly, the ELBO is both not an measure, nor is it possible to interpret what values are good or not, i.e., when the approximation is useful to use as an approximation of the posterior. It is therefore suggested (Yao et al. 2018) to use Pareto smoothed importance sampling (PSIS) as a diagnostic. Briefly, importance sampling uses weights $p(\boldsymbol{\beta}, \sigma, y \mid \boldsymbol{x})/q(\boldsymbol{\beta}, \sigma)$, where q is the approximation of the posterior—which is a flow in our case. After finding an approximation q, PSIS adjust the largest weights, by fitting an generalised Pareto distribution to the set of M (empirically set) largest weights, and replaces the original M weights by the expectation of the fitted distribution. This is done to improve the variance of any estimates using importance sampling. The main point in our case is that the shape parameter k in the generalised Pareto distribution can say something of how well the approximation is. Lower $\hat{k}$, where $\hat{k}$ is the *estimated* shape parameter, is better. Vehtari, Simpson et al. 2021 argues that $\hat{k} < 0.5$ is optimal, while $0.5 < \hat{k} < 0.7$ implies useful approximations, but slower convergence rate w.r.t number of samples in the importance weight estimate, and when $\hat{k} > 0.7$ implies that the model ought to be changed to have any usefulness. We used the **loo** package in R (Vehtari, Gabry et al. 2020) to estimate $\hat{k}$. With this in mind, we can move on to the results of the flows.

The results are given in Table 5.8 and the hyperparameters can be found in Table 5.7. The models tested did converge, but had some fluctuations around 12.5 to 13 in ELBO, hence the results must be seen in the light of

Table 5.8: Results for experiment 4, with ELBO (higher is better), and the estimated $\hat{k}$ (lower is better), for both training set and test set. Every statistic is computed by sampling 10 000 new samples from given flow, and each models confidence interval is computed by 3 flows.

	ELBO train	$\hat{k}$ train	ELBO test	$\hat{k}$ test
Base	$\mathbf{-12.56 \pm 0.475}$	0.57 ± 0.534	$\mathbf{-12.34 \pm 0.376}$	0.42 ± 0.216
Aff-IAR	-13.30 ± 0.159	0.64 ± 0.097	-13.11 ± 0.124	0.55 ± 0.132
AffCon-IAR	-12.95 ± 0.108	0.46 ± 0.170	-12.76 ± 0.10	0.47 ± 0.323
AlfLinAff-IAR	-12.60 ± 0.065	$\mathbf{0.27 \pm 0.304}$	-12.41 ± 0.036	$\mathbf{0.30 \pm 0.056}$
AltLinCon-IAR	-12.71 ± 0.156	0.31 ± 0.121	-12.52 ± 0.163	0.397 ± 0.067

this. In addition, the estimated confidence intervals are awfully large for some of the models, where we also should have used more flows to estimate—we used 3 due to time constraints. It is also fair to note that we did little to none hyperparameter tuning. It does however demonstrate a few things worth delving into.

The base flow does reasonably well, which implies that the posterior distribution is not the most complex. It is interesting, however, that the base flow performs best ELBO wise, but not $\hat{k}$ wise, which suggest the analysis of Yao et al. 2018 is important.

The flow with affine transformation performs worse than every other flow tested—although it is likely that they partly do worse than Base due to different a-activation function. This is quite complementary to Dhaka et al. 2021, which looked at flows with affine transformations and $D/2$-coupling structure (Dinh, Sohl-Dickstein et al. 2017). They found that, when the dimension of latent variables increases, the flows struggle with finding a good approximation of posterior, in terms of $\hat{k}$, at least without tuning of hyperparameters. We have also done little to no tuning, and even though this posterior is less complex than theirs, in our results we find that affine transformations struggles with AR structures as well.

What is more surprising is perhaps that the slightly more complex transformations do not have this problem. This is surprising due to Dhaka et al. 2021 findings, as already mentioned. In particular, alternating with linear transformations performs well, with affine transformations performing the best. It is difficult from this to speculate exactly why, but again Dhaka et al. 2021 shows that flows with affine transformations and $D/2$-coupling structure in a 2-dimensional setting where the posterior is similar, but almost rotated— see Figure 3 in Dhaka et al. 2021. We postulate that the linear layers might be helpful in rotating the posterior density, solving the problem demonstrated by Dhaka et al. 2021, however without further testing this is simply conjecture. It is hard to conclude anything concretely from this experiment, but is a promising start into investigating performance of more complex flows, and how they might not suffer as badly as the less complex ones demonstrated both here and in Dhaka et al. 2021.

5.6 Conclusion

In this chapter we have performed 4 experiments, the first two were more broad scope experiments, where we tested many flows, while the last two focused more on giving a better one-to-one comparisons of a few flows. Following the results, we found that piecewise affine transformations struggle, although they are UDA, which simply goes to show that asymptotic behaviour—and in particular the existence of flows with good asymptotic behaviour—does not necessarily imply well functioning flows with finite T and approximating through finite amount of data. We outlined two possible reasons for this: the lack of the ability of translation is one, as we saw similar poor performance with continuous piecewise affine transformations. Another reason for it seems to be the non-smooth derivative of piecewise affine transformations, which is motivated by the very good performance of affine continuous piecewise affine transformations which contains a smooth derivative. However, there are examples of transformations of non-smooth derivatives performing well, e.g. Oliva et al. 2018 which we discussed as Related Work in Section 4.2, hence the explanation behind the poor performance of piecewise affine transformations might be a combination of the two reasons above.

We also saw that affine continuous piecewise affine transformations perform well, in particular for independent distributions such as experiment 2, but also with state of the art results in experiment 3 which indicates that this type of transformations can fill a gap between complex transformations and affine transformations. There are however some additional time added by changing affine transformations with ours. It is hard to measure exactly, but we do find a bump in training time, although each model was trained in reasonable time. The additional time training can also be a bit misleading, as affine transformations are straightforward to implement, while continuous piecewise affine transformations is in total a function of 6 pieces—although the identity part for values less than 0 is practically free—the others are not. We did run some optimisation of code, and made it a lot faster than at the start, however, we find that the biggest time sinker is the sorting of pieces, not the actual computation of each value. There are plenty of piecewise functions in PyTorch that runs quickly and is coded in lower level languages, more directly to the hardware, hence we are confident there is much time to be gained by more optimisation of our code.

Although alternating with linear transformations did not perform as well as we anticipated, it did show remarkable results in experiment 4, which we also saw that more complex transformations may not suffer as badly as less complex ones outlined in Dhaka et al. 2021. There is many more experiments to be done in this are, before we can conclude anything, however interesting and promising results nonetheless.

There are problems with the results produced in this thesis. Firstly, the lack of a reasonable amount of hyperparameter tuning can give quite misleading potential of the different flows, in addition to the fact that each flow had the same hyperparameter that might have favoured some flows over others. The confidence intervals are potentially quite large, although we did see Ψ^+ had similar intervals, but also need higher number of flows trained when estimating the standard deviation, and therefore also the confidence intervals. There is an imbalance and lack of tests on real data, and hence is something discussed

in Section 6.2. Finally, the instabilities we got during training, is problematic as we start to depend on the particular sample of data, and can be favourable to some flows. We have tried to mitigate this by re-running experiments with fewer flows, that differs by only one component, such as the transformation, but it nevertheless is problematic with high instability and we discuss this further in Section 6.2.

CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

We started out this thesis by laying out the description of the problem described in the project description:

(i) Formalise normalizing flows where the literature lacks,

(ii) Explore new ways to transform data that leans toward the simplistic transformations, yet tries to be as flexible as possible compared to the more complex transformations.

We tackled (i) in Chapter 3 by introducing the formalisation of flows through introducing four components: base distribution $\mathcal{Q}$, structure $\mathcal{S}$, conditioner $\mathcal{H}$, and transformations f. In particular, the introduction of structures provided a fruitful new component to describe parts of flows with, which partially lacked in the literature. We further developed structures and defined useful theoretical properties such as flow-isomorphism, which is a useful theoretical tool to analyse and equate different flow-structures. This insight gives us tools to equate larger structures with a smaller class of structures, namely forward-local flow-structures, and these types of proofs can simplify new theoretical proofs regarding structures, as we shrink the space we need to concern ourselves with. In addition, we found requirements for structures, such that we are guaranteed triangular Jacobian. By introducing the concept of structures, together with the generalisation of MADE, i.e. CONN, allows for future work with many more interesting structures. We also adapted current literature into the new formalisation, including conditioner transformations and a new definition of UDA.

In Chapter 4 we address point (ii). We introduce four new transformations that contains the same number of parameters in the transformation – with a maximum addition of two parameters – which keeps the linearity of affine transformations almost everywhere. The proposed transformations thus retains attractive properties such as fast computation, both forward and backward, and with triangular structures, fast evaluation of density, where all three operations scale linearly with the number of dimensions or number of time steps. We then showed showed how such transformations fills the gap between affine transformations and the more complex ones, in terms of expressiveness. In particular, that our proposed transformations are UDA for similar structure as the more complex ones, also for dimension one and under independence in

target distribution, compared to affine transformations. Hence, it compares well asymptotically with more complex transformations, while keeping the attractive practical qualities of affine transformations, which we defined through the class of affine extended transformations. Proving that CONNs can also be universal approximators, means future theoretical and practical work with CONNs and structures can rely on neural networks capabilities.

Finally, in Chapter 5 we perform several experiments to investigate the applicability of our proposed transformations, including our new a-activation functions. We find that our new a-activation functions aid in the stability of training flows, but it is certainly not enough as we still experience instability. We also found that translation and rotation are important elements, both through alternating linear transformations and conditional transformations, but also as both piecewise affine and continuous piecewise affine performs badly. However, it also seems to be that smoothness in the in the derivative of a transformation plays an important role, where affine continuous piecewise affine transformations performed better, even significantly better, in some experiments than both affine transformations and in affine piecewise affine transformations—where the latter has a non-smooth derivative. Confirming that our proposed transformations also fills the gap empirically between affine transformations and more complex ones, as was the goal for (ii). However, there are still gaps in terms of how well our proposed methods work compared to alternating linear transformations and affine transformations, in particular as training linear transformations are not as expensive due to Bader et al. 2019, as we had varying result in which performed best over the experiments. As already noted, there are much improvements to be made empirically, which we come back to in the next section.

All in all, we have formalised flows and laid ground work for new theory to be developed, with a hopefully adequate terminology, where we have already shown several results, and also invite more a more diverse use of structures combined with CONNs. Furthermore, we have started bridging the gaps between complex transformations and simpler ones, that can be useful when the complex ones are too computationally burdensome.

6.2 Future Work

In this section we propose ideas we wish to investigate further, which also highlights shortcomings of the thesis.

Empirical Part

The empirical part of this thesis is by far the part that needs further work. Partly due to time, but also due to difficulties in implementation/instabilities in flows during training. We propose the following points of improvement/future work:

- Tests that involves more complex and difficult target distributions is needed. In particular test our new methods running all of the most common benchmarks for density estimation, as well as many more difficult variational inference problems, to be able to better compare more complex flows with the simpler ones studied in Dhaka et al. 2021. In addition, resample several times sets of parameters $(\boldsymbol{\beta}, \sigma)$ followed by new data

$(\boldsymbol{x}, \boldsymbol{y})$. This to uncover any chance when generating the data that may affect the flows. Also increase number of runs of each flow, to improve the confidence intervals.

- For the experiments we did run, several important statistics lack. First of, in density evaluation, we also need to see how well the new generated samples fits the true model, which is possible when the target distributions density is known. We did run such an experiment for experiment 1, but as the target distribution was too simple, gave us too little information to conclude alone. For experiment 1, the new samples did well however, but more experiments are needed. In addition, we think it would be very fruitful to investigate the tail behaviour for the experiments we ran. Typically the tails are the largest problem in many approximations, and is not necessarily captured perfectly by mean log-likelihood. A good investigation could also give insight into how well they perform compared to Copulas.

- A more comprehensive study of the a-activation functions, as we now tested lots of different models with different a-activation functions, depending on the stability required by the experiment. However, the one-to-one comparison lacks, and hence we can only find general patterns, such as Slowplus seem to be a valid choice when Softplus leads to instability, while performing better than σ. A proper comparison would have to involve several flows, where each is trained with every possible a-activation function we wish to investigate.

- As we did little to no hyperparameter tuning, we need to investigate further how well the models perform under a reasonable amount of tuning. This may give different results and change the view on which transformations in our experiments are better, but also to give a more apt comparison to results reported in the literature.

- Investigate further how linear transformations, where the matrix is in $SO(D)$, can minimise the problems with flows and variational inference which are outlined by Dhaka et al. 2021. In particular how rotating can help when the dimension of the latent space increases.

- We struggled quite a bit with exploding gradients, sometimes even with σ as a-activation function. There are three methods we would like to explore more. First is pre-training, using very restrictive a-activation functions, as many times the exploding gradients happened from the very beginning, which may imply problems with initialisation. The second approach is to do a study over different ways to initialise the weights in the conditioners. Perhaps there are methods suited better to normalizing flows, which is not among the standard methods—this can also include a more theoretical approach in combination. Thirdly, is to clip the values of a-activation functions such as Slowplus, but allow for significantly higher values than Sigmoid σ, as σ often seem to perform worse, finding a better balance between Sigmoid and Slowplus/Softplus can be fruitful. Gaining both the performance part, and yet induce stability during training. We should also explore batch normalization (Papamakarios et al. 2017).

- Another idea that combats instability, which we managed to implement in **flows.py**, but not test, were to train first a certain number of the transformations $t < T$, and then train another part and so on, until the whole flow is trained, where one can then train the whole model for final tuning of the parameters. This makes it less likely to get exploding gradients, as there are fewer transformations stacked, and hence fewer parameters to alter the log-likelihood.

- A comprehensive study of the β parameter in continuous piecewise affine transformations is necessary, as we till now have only used $\beta = 5$ in every experiment.

- We had issues implementing CONN, and was a bit too slow for usage. To properly implement it, and test it empirically is something deeply missed in this thesis. With a well-working CONN, we are able to test interesting structures. In particular, generate data where we know the underlying correlations, testing flows with IAR/AR, and structures that correspond to the underlying correlations, is very interesting prospect.

Theoretical Part

For the more theoretical parts we propose the following as future work:

- To continue working towards finding the class of all valid structures that guarantees triangular Jacobian, which then shows exactly what structures we can explore empirically, without increasing the computational burden when evaluating the density of the flow. Our work so far is to consider only structures that are flow-forward, due to our Proposition 3.3.13, but where $D_t > D$. It seems quite clear that every flow with triangular Jacobian have a structure that only transform one variable at each time step (Identity node for the others), but not all such structures that transform on variable at each time step induces a triangular Jacobian. Finding out when it fails, is the key to finding the complete class that guarantees triangular Jacobian.

- Replacing neural networks with other functions is an interesting prospect. One can then start investigating functions that have much better convergence properties, i.e., some guarantees for finitely many transformations, or asymptotically convergence which is states more than existence, as we commented on previously. This potential functions may be more limited, and to further investigate for which classes $\mathscr{P}$ the class of flows with limited conditioner are UDA for.

- As we proved universality for a class that contains distributions with compact support, and we have a relation between the number of transformations and ϵ (through M in Lemma 4.4.3), where ϵ is w.r.t the canonical triangular map followed by the Sigmoid function. Combining compactness of target distribution and continuous function σ, means we should be able to state number of transformations T, and that there exist a flow approximating the target distribution withing some function of ϵ well.

Valid Structures and Noisy Conditioner

As we were not able to test properly this part, due to time constraints, but the idea was already quite developed and proven, we choose to introduce the idea here, and let the testing be put as future work.

We want to explore structures, and how we can move away from the triangular structures, yet easily compute the density. In return we give up the ease of computing the inverse of the flow, which can be fine especially in a variational inference setting.

In a variational inference setting, we are most concerned with sampling the latent variables, evaluating its density, and evaluating the likelihood. None of these requires us to evaluate the inverse, for as long as we can evaluate the density forward, i.e. while we sample, there are usually no case where we have samples from the latent space that has not been sampled through our flow. That is, we usually do not observe latent observations—hence the name latent—and as long as we have the density of the latent samples generated by our flow, we have everything we need. When we ease on the requirement of a computable inverse of the flow, we find ways to expand on structures outside triangular structures, that still allows for easy density evaluation and complex conditioners.

Even though we have CONNs that can compute quickly the parameters needed for any transformation step t, and still oblige to the triangular structure, it is still limiting. Due to the masks, it cannot use all possible weights in a fully connected neural network, and the structure is also limiting as it cannot allow all the variables to affect each other in any given time step $t \in \mathcal{T}$. We therefore propose a new type of neural network that allows to input the whole $\boldsymbol{z}_{t-1}$ into the conditioner $\mathcal{H}_t = (\mathcal{H}_{t,1}, \ldots, \mathcal{H}_{t,D})$, yet allows for evaluation of density similar to triangular structures.

Definition 6.2.1. Let $\mathcal{NN}_{[L,\tilde{\boldsymbol{D}},\gamma]}$ be a space over fully connected neural networks with L hidden layers, $\tilde{\boldsymbol{D}}$ a vector with number of hidden nodes in each layer $l \in \{1, \ldots, L\}$, activation function γ, and input size K. Then $\mathcal{NN}^{(\text{Noise})}_{[L,\tilde{\boldsymbol{D}},\gamma]}$ is a space of *noisy fully connected neural networks*, where for any neural network $\Psi \in \mathcal{NN}_{[L,\tilde{\boldsymbol{D}},\gamma]}$, we have $\Psi^{(n)} \in \mathcal{NN}^{(\text{Noise})}_{[L,\tilde{\boldsymbol{D}},\gamma]}$, where

$$\Psi^{(n)}(\boldsymbol{z}) = \Psi(\boldsymbol{z} + \boldsymbol{u}).$$

Here $\boldsymbol{u}$ is a vector with $u_k \sim \mathcal{U}(-\epsilon, \epsilon)$, with $\epsilon > 0$ and $\mathcal{U}$ is the Uniform distribution..

The definition of a noisy neural network is simply to add a small uniformly distributed component to each dimension before we apply a regular neural network. The noise can be arbitrarily small, i.e., ϵ can be arbitrarily small, and hence ought to have a very little effect on the networks output. However, it gives us new structures to use.

For the next part we still assume that the structure has the same number of nodes in each layer $t \in \mathcal{T}$, however we allow for every valid structure $\mathcal{S}$, where the fully connected structure $\mathcal{S}_{full}$ is a subset $\mathcal{S}_{full} \subseteq \mathcal{S}$ (see Figure 6.1 for an example of such structures), and where $\mathcal{S}$ has the same number of nodes in

Figure 6.1: Example of a fully connected structure and a structure $\mathcal{S}$, such that $\mathcal{S}_{full} \subseteq \mathcal{S}$.

each layer. Hence, we abandon most of the requirements enforced to create triangular structures. As we want to be able to compute the conditioner for each time step, by using at least $\boldsymbol{z}_{t-1}$, means we need to alter the definition of a conditioner slightly, but which makes it even more expressive.

Definition 6.2.2. Let $(Q, \mathcal{S}, f)$ be a NF, where each $f_{t,d}$ is parameterized by $\rho_{t,d}$ parameters. An *extended conditioner* is any function $\overline{\mathcal{H}}$ where

$$\overline{\mathcal{H}}_{t,d} \colon \mathcal{S}(t,d) \to \mathbb{R}^{\rho_{t,d}}.$$

for all $(t,d) \in \{(t,d) \colon t \in \mathcal{T} \text{ and } d \in \mathcal{D}_{\mathcal{T}}\}$.

The only difference between conditioner and extended conditioner, is that we allow for the variable that is transformed to be input to the extended conditioner as well, i.e., we allow for $\mathcal{S}_{int}(t,d)$ to be part of the input.

We are now ready to prove that using noisy neural networks as an extended conditioner, we allow for structure $\mathcal{S}$, such that $\mathcal{S}_{full} \subseteq \mathcal{S}$ and easy to compute density. We denote U as a set with uniform noise for a specific ϵ, which includes every $u = (\boldsymbol{u}_t)_{t=1}^{T}$. This is the noise used in each neural network that is used to compute the extended conditioner $\overline{\mathcal{H}}_t = (\overline{\mathcal{H}}_{t,1}, \ldots, \overline{\mathcal{H}}_{t,D})$ for every $t \in \mathcal{T}$.

Theorem 6.2.3. *Let $(\mathcal{Q}, \mathcal{S}, \overline{\mathcal{H}}, f)$ be a flow with $\mathcal{S}$ being a valid structure with $D_t = D$ for all $t \in \mathcal{D}$ and $\mathcal{S}_{full} \subseteq \mathcal{S}$. For each $t \in \mathcal{T}$, assume $\overline{\mathcal{H}}_t$ is computed by a noisy fully connected neural network, with conditioner space being $\mathcal{NN}^{(Noise)}_{[L,\tilde{\boldsymbol{D}},\gamma]}$. Then the induced density of the flow can be written as*

$$q_{\boldsymbol{z}_T}(f(\boldsymbol{z}_0, U)) \propto q_{\boldsymbol{z}_0}(\boldsymbol{z}_0) \cdot \prod_{t \in \mathcal{T}} \prod_{t \in \mathcal{D}} \left| \frac{\partial}{\partial z_{t-1,d}} f_{t,d}^{-1}(z_{t-1,d}) \right|$$

Proof. The proof is to rewrite the flow above into a regular flow with neural network as conditioner and regular conditioner (not extended). This shows that we end up with same density, and claim equivalence between the two models.

We start by swapping the noisy neural networks with normal neural networks and apply Λ_1—the first part of the function transforming $\mathcal{S}$ into a forward-local flow-structure—to $\mathcal{S}$, and hence concern ourselves with only edges that move from t to t', where $t < t'$ (the result still stand for other structures, due to

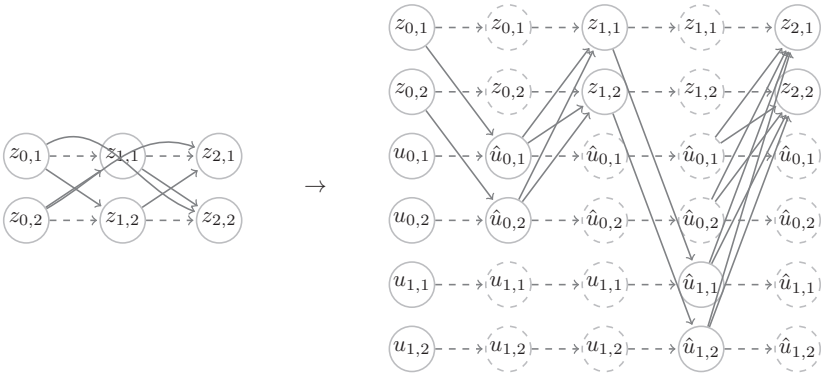


Figure 6.2: To illustrate how we rewrite the flow with noisy neural network/extended conditioner and structure on the left, to the new structure with augmented base distribution and non-extended conditioner using neural networks.

flow-isomorphism). As we are working with a regular neural network, we use the regular conditioner, so we exclude interior edges.

We then make a small adjustment by augmenting the base distribution with $T \cdot D$ uniformly distributed variables $u_{t,d} \sim \mathcal{U}(-\epsilon, \epsilon)$, these corresponds to the noise we use in noisy neural networks. We then alter the flow as follows: before each transformation step t, we add a new layer where every node is an identity transformation except for $u_{t,d}$, which is transformed as $\hat{u}_{t,d} = f(u_{t,d}) = z_{t,d} + u_{t,d}$. Then for the next time step t, we erase every exterior edge from variables $z_{t',d'}$, where $t' < t$, and rather use edges from the corresponding $\hat{u}_{t',d'}$, where the edge goes from the corresponding Uniform node in the last layer to the next. We have illustrated this rewriting in Figure 6.2.

We then achieve a triangular Jacobian for each time step t, the structure has $D + T \cdot D$ nodes in each layer (which means we can split the computing of the determinant Jacobian into T steps), the noise is Uniform, and the transformation of the noise is volume preserving. The density of our new constructed flow is therefore

$$q_{\boldsymbol{z}_T}(f(\boldsymbol{z}_0, U)) \propto q_{\boldsymbol{z}_0}(\boldsymbol{z}_0) \cdot \prod_{t \in \mathcal{T}} \prod_{d \in \mathcal{D}} \left| \frac{\partial}{\partial z_{t-1,d}} f_{t,d}^{-1}(z_{t-1,d}) \right| .$$

As the new flow is equivalent to dropping the augmented base distribution, and rather use noisy neural networks, we are done. ■

Remark 6.2.4. A couple of remarks are in order. Firstly, notice that each variable $z_{T,d}$ is not affected by the noise in any way other than through the parameters in the transformation $f_{t,d}$ for every $t \in \mathcal{T}$. Secondly, we may evaluate exact density and not just proportional, by simply multiplying the density with $(1/2\epsilon)^{T \cdot D}$. Also note that we are not storing the uniform noise as in the proof, we only apply it as input to the neural network, i.e., using noisy neural networks.

A reason behind only caring about the proportional density, is the fact that we are most interested in the likelihood and evaluating posterior through importance sampling, where the term $(1/2\epsilon)^{T \cdot D}$ disappears as it is constant

when D and T is decided.

A few alternatives to the flow above is possible. We may rather use structures similar to AR, i.e., edges from nodes to nodes in the same time step t, to make it usable for density estimation (where the sampling is then not possible, only evaluation). This is not as interesting, as we often want to sample new observations and we want the exact density, not a proportional one with added Uniform variables.

One may also use CONNs with noise added to the input, and use some masks. For example, if you do want to use non-extended conditioner, but fully connected structure, you can use D masks where each include all the dimensions but one, and therefore have output exclude the interior input. This is not something we wish to pursuit, as once the ability to compute the inverse is broken, we find no use in enforcing non-extended transformations.

Finally, there is an opportunity to learn triangular structure with optimal permutations/dependencies among variables. Loosely, in a training setting, optimise trainable parameters using the structures described above, i.e., have fully connected structure as part of it. Then apply pruning of the structure, which can be seen as taking the structure to the right in Figure 6.2, change one of the $\hat{u}_{t,d}]$ node into an identity node. Then any flow-isomorphic structure to the resulting structure after swapping with identity nodes, is a pruned structure. Having the new structure have then implicitly chosen a mask in CONN, which one can enforce, and then continue pruning. Continuing this until some end criteria (e.g. some unacceptable jump in loss), while also ensuring the final pruned structure is a triangular structure as we saw in Section 3.3, in particular methods close to NO TEARS (Zheng et al. 2018). The added bonus of this, is that the training can then find what variables to emphasis on when computing $z_{t,d}$, as it is allowed to use any $\boldsymbol{z}_{t-1}$ as input to the conditioner, i.e., the importance of the different dimensions to compute $z_{t,d}$ can be decided by the conditioner/neural network itself. After finishing pruning, one can then stop adding noise, and run as flows with triangular structure. This was loosely put the idea behind first training with fully connected structure, and then enforce triangular, but need more work to crystallise.

Related Work

In the recent years more research has been done towards exploring flows with augmented space, which is how one can interpret our noisy neural network conditioner (Cornish et al. 2020). The closest to our proposed method is done by Huang, Dinh et al. 2020, where they do similar augmentation as we do in the proof, but they only double the dimension and sample Gaussian distributed variables, which they alternate transforming. First transforming the sample we are interested in, then transforming the augmented part, and so on. Using each other as input to a conditioner, as an augmented $D/2$-coupling structure. This was intended to study universality of affine transformations, and was the closes universality result for affine transformations before Teshima et al. 2020. Our proposed method is different, in that we do not care about transforming the noise, and do not need to store it, only sample it before computing a conditioner $\mathcal{H}_t$. Our method is also the only that have a realistic path to acquire a triangular structure after training, and removing the augmented part.

Appendices

APPENDIX A

Additional Resources

A.1 Classes of Divergences

The three major classes of divergences are the *f-divergence*, *Bregman divergence*, and *integral probability metrics* (IPM), and we quickly run through them. The f-divergence is perhaps the most common of the three, which is any divergence of the form

$$\mathcal{DI}_f(p,q) = \begin{cases} \int_{\mathcal{X}} q(\boldsymbol{x})\, \phi\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) d\boldsymbol{x}, & \text{if } \mu \ll \nu \\ \infty, & \text{otherwise,} \end{cases}$$

where $\phi\colon [0,\infty) \to (-\infty,\infty]$ is convex and $\phi(1) = 0$. The prime example of a f-divergence is when $\phi(r) = r \cdot \log(r)$, which is the well-known Kullback-Leibler divergence, and one we shall return to later on. There are many other known f-divergences such as Hellinger distance $\phi(r) = (\sqrt(r) - 1)^2$, exponential divergence $\phi(r) = \log(r)^{-2}$ etc.

Bregman divergences are any divergence of the form

$$\mathcal{DI}_B(p,q) = \phi(p) - \phi(q) - \langle \nabla\phi(q), p - q\rangle,$$

where $\phi\colon \mathscr{P} \to \mathbb{R}$ is a continuously differentiable, strictly convex function defined on a closed convex set $\mathscr{P}$. The classic example is when $\phi(r) = \|r\|_2^2$ which gives $\mathcal{DI}_B(p,q) = \|p-q\|_2^2$, but perhaps more pertinent is the negative entropy (more specifically the negative differential entropy),

$$\phi(r) = \int_{\mathcal{X}} r(\boldsymbol{x}) \log r(\boldsymbol{x}) d\boldsymbol{x}.$$

This turns out to be connected to the f-divergence, namely,

$$\begin{aligned} \mathcal{DI}_B(p,q) &= \phi(p) - \phi(q) - \langle \nabla\phi(q), p - q\rangle \\ &= \int_{\mathcal{X}} p(\boldsymbol{x}) \log p(\boldsymbol{x}) d\boldsymbol{x} - \int_{\mathcal{X}} q(\boldsymbol{x}) \log q(\boldsymbol{x}) d\boldsymbol{x} \\ &- \int_{\mathcal{X}} p(\boldsymbol{x}) \log q(\boldsymbol{x}) - p(\boldsymbol{x}) + q(\boldsymbol{x}) \log q(\boldsymbol{x}) + q(\boldsymbol{x}) d\boldsymbol{x} \\ &= \int_{\mathcal{X}} p(\boldsymbol{x}) \log p(\boldsymbol{x}) - p(\boldsymbol{x}) \log q(\boldsymbol{x}) d\boldsymbol{x} - \int_{\mathcal{X}} p(\boldsymbol{x}) d\boldsymbol{x} + \int_{\mathcal{X}} q(\boldsymbol{x}) d\boldsymbol{x} \\ &= \int_{\mathcal{X}} p(\boldsymbol{x}) \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) d\boldsymbol{x}, \end{aligned}$$

which one recognize as the KL-divergence. Hence, KL-divergence is a member of both f-divergences and Bregman divergences.

The last one has played a more relevant role when it comes to machine learning, compared to Bregman divergences (except for KL-divergence), namely IPM. Any IPM divergences are on the form

$$\mathcal{DI}_I(p,q) = \sup_{\phi\in\Phi} \left| \int_{\mathcal{X}} \phi(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} - \int_{\mathcal{X}} \phi(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} \right|, \tag{A.1}$$

where Φ is a class of functions $\phi\colon \mathcal{X} \to \mathbb{R}$ which are bounded. Under certain constrains w.r.t separability one can for instance choose Φ such that the divergence is the Wasserstein distance, total variation distance, or maximum mean discrepancy. In a empirical approximation, some of IPM-divergences can have better convergence, that is, converging faster to the true divergence using Equation (A.1) (Sriperumbudur et al. 2009), and recent years some of the divergences have seen rise in popularity (Goodfellow et al. 2014).

A.2 CONN: Non-Independent Sampling and Residual Blocks

Considering the problems described in Section 4.3, we now develop new ways to both sample masks and add residual blocks, in the hope of increasing the CONNs reliability. In this context, reliability refers to how reliably the model are in terms of using its input variables, both in a universal setting, but also in a finite setting. For example, we wish to avoid letting outputs that depends on some input variable, to be computed as a constant.

There are three parts in the new scheme, and can be incorporated by themselves or as a whole. The three parts address the following problems respectively:

1. Nodes that are computed through its bias alone, i.e. no connections from the previous layer.

2. Nodes that have no connections to any node in the next layer.

3. Minimize occurrences of (1) and (2) through sampling.

The first two can be effectively solved through residual connections. Concentrating on (1) first, we add a residual connection from a set of input variables to any node $\Psi_{l,d'}$ which has no connections from previous layers, where the set of input variables is $\{x_d\colon d \in m_l(d')\}$. One could think of it as extending the hidden layer Ψ_{l-1} with nodes that contain the values of the input variables in question. Then simply assign the mapping the new nodes $\{d\}$ for every $d \in \{d\colon d \in m_l(d')\}$. We can solve (2) in a similar manner, where every time a node does not have connection to any node in the next layer, we add a residual connection from said node the the last layer. That is, if a node $\Psi_{l,d}$ does not contain a connection to any nodes in Ψ_{l+1}, we add Ψ_{L,D_L+1} as a node where its mapping and value is equal to $\Psi_{l,d}$.

Another solution to (1) and (2) are to restrict the sampling such that (1) and (2) cannot happen. Not particularly hard to enforce, however, there are two problems with this approach. Firstly, if there are many disjoint sets in

$\mathcal{C}_{min}$, the dimension of the hidden layers must typically be quite high relative to D_0 to avoid many outputs equalling constants. Secondly, the networks, with said restrictions on the sampling, prefers larger sets in $\mathcal{C}$, hence again leaving many outputs to be computed by constants. Both problems stems from not being able to go from one node to another, when the former node's mapping is a superset of the latter node's mapping. We may be able to patch up parts of the problems put forward, but adding more and more restrictions on the sampling reduces it to something close to deterministic, and by that point gains very little than our more crude design in the previous section. We ought, however, not to throw the baby out with the bathwater, as the idea with altering the sampling scheme may have some merit. Indeed, it can be combined with the residual connections discussed above. For example, we may enforce certain nodes to have connections to the next layer, if the node is computed by the last T layers or fewer, with T being a threshold. The idea is in fact embedded somewhat in our proposed sampling scheme.

We now aim to tackle (3) through making the sampling scheme more dynamical. The goal is both to spur on diversity among the mappings, and to avoid too many residual connections. There are two points of interests we have considered. The first is that for every hidden layer, we take into account the sets in $\mathcal{C}$ that are supersets of last layers mappings and adjust the probability for such sets accordingly. That is, for a layer l, we increase the chance of sampling a set $c_i \in \mathcal{C}$ if there exists a $d \in \{1, \dots, D_{l-1}\}$ such that $m_{l-1}(d) \subseteq c_i$. This can then limit the number of residual connections.

Second point of interest concerns long term diversity in samples. When sampling mappings for a new layer, we want to take into consideration which mappings have been sampled in the previous layers, and adjust the probabilities thereafter. In other words, we want to impose more long term memory into the sampling scheme, as well. We want to discourage sampling the same mappings as the one sampled recently, with the relevance of layers diminishing as we add more and more layers. This lead us to the following proposed sampling scheme. We start by defining an order on $\mathcal{C}$ for convenience, with elements $c_i \in \mathcal{C}$ for $i = 1, \dots, M$, and with $M = |\mathcal{C}|$. We then sample sequentially layer-wise. Assuming we have sampled up to layer $l-1$ and each previous layer $l' < l$ is associated with a vector

$$\boldsymbol{v}^{(l')} = [a_1, a_2, \dots, a_M],$$

where a_i is the number of times c_i was sampled in layer l'. We also define a vector with element i being

$$s_i^{(l)} = \begin{cases} 1, & \text{if } \exists d \text{ such that } m_{l-1}(d) \subseteq c_i \\ 0, & \text{otherwise.} \end{cases}$$

We then assign weight to each set in $\mathcal{C}$, as

$$\boldsymbol{p}^{(l)} = \alpha_1 \boldsymbol{s}^{(l)} - \sum_{l'=1}^{l-2} \alpha_2^{(l-l'-2)} \boldsymbol{v}^{(l')},$$

where $\alpha_1, \alpha_2 \in [0, 1)$ are fixed parameters which decides how well we wish to impose connections between layer $l-1$ and l, and how much the past should

influence the current samples. We can then define the probability for any node in hidden layer l to be c_i is equal to

$$Pr(c_i) = \frac{\exp\left(p_i^l\right)}{\sum_{j=1}^{M} \exp\left(p_j^{(l)}\right)}. \tag{A.2}$$

Sampling then becomes generating observations for each layer from a multinomial distribution with the probabilities according to Equation (A.2). This scheme also includes the uniform distribution by setting $\alpha_1 = \alpha_2 = 0$. To avoid storing too much information, we add a cut-off $0 < \delta < 1$ such that

$$p^{(l)} = \alpha_1 \boldsymbol{s}^{(l)} - \sum_{l'=a}^{l-2} \alpha_2^{(l-l'-2)} \boldsymbol{v}^{(l')},$$

where a is the first layer where $\alpha_2^{l-a-2} > \delta$. The described sampling scheme will be referred to as *layer-dependent sampling* and residual connections used in the manner discussed previously are referred to as *connectionless-residuals*.

Corollary A.2.1. *Let γ be a nonpolynomial activation function, $\mathcal{X} \subseteq \mathbb{R}^{D_{L+1}}$ be compact. Let $\mathcal{NN}^C_{[L,1,\gamma]}$ be limited to all models which employs layer-dependent sampling and connectionless-residuals. The resulting $\mathcal{F}^C_{depth}$ is almost surely dense in $C\left(\mathcal{X}, \mathbb{R}^{D_{L+1}}; c\right)$.*

Proof. Let n be maximum number of layers we use, which is finite and depend on what α_2 and δ are. Let D be the maximum number of nodes in a hidden layer, which is also finite, and in the corollary is equal to 1. Then we can bound $p^{(l)}$ for any arbitrary layer l as

$$p^{(l)} < \sum_{i=1}^{n} \alpha_2 \, D.$$

This means Lemma 4.3.1 holds for layer-dependent sampling as well, as the probability for any mapping is always greater than 0. The residual connections given by connectionless-residuals acts in the same manner as the deterministic chosen nodes in Theorem 4.3.6, and hence the results readily follows. ■

Remark A.2.2. One could in theory deploy any sampling scheme that fulfills gives rise to same similar result as in Lemma 4.3.1, as it is the residual connections that does the heavy lifting. The proposed sampling scheme is there simply to avoid too many residual connections. It does not, however, seem to be any foolproof way to use a purely sampling based CONN without connections, and without scaling the width of the network immensely and rendering the sampling method to almost deterministic.

With this we conclude the chapter and introduce normalizing flows next. As we shall see, the usage of CONN models plays a large role in many models concerning normalizing flows, and the new theoretical results gained here can then be incorporated into the theoretical work of normalizing flows.

A.3 Classifying Transformations

In the main part of this thesis, we have introduced specific transformations, which have been a quite narrow classification. They have mostly consisted of a specific parametrization, and the class it has belonged to are the space of values the specific parametrization can take. For example, affine transformations are parameterized by (a, b), applied in a specific way, and the only classification we may recognise is affine transformations with different values (a, b) and $(\tilde{a}, \tilde{b})$. Although there may be some differences other than the parameter values, for instance neural network transformations with Ψ^+ (Huang, Krueger et al. 2018) and Unconstrained Monotonic Neural Networks (Wehenkel et al. 2019), these differences are to aid computationally and not particularly interesting theoretically.

Inflection Transformations

To differentiate between transformations, we begin by addressing their associated inflection points.

Definition A.3.1. Let $(\mathcal{Q}, \mathcal{S}, f)$ be a normalizing flow where each transformation has $\rho_{t,d}$ number of parameters, and is parameterized by $\boldsymbol{\theta}_{t,d} \in \mathbb{R}^{\rho_{t,d}}$. The transformation $f_{t,d}$ with predecessor $z_{t-1,i}$ is an *inflection transformation* if it is a piecewise C^1-diffeomorphism where

$$g_{t,d}(z_{t-1,i}; \boldsymbol{\theta}_{t,d}) = \frac{\partial^2}{\partial z_{t-1,i}^2} f_{t,d}(z_{t-1,i})$$

such that there exists two different parameters $\boldsymbol{\theta}_{t,d}^{(1)}, \boldsymbol{\theta}_{t,d}^{(2)} \in \mathbb{R}^{\rho_{t.d}}$ and a $z_{t-1,i}$, such that

$$g_{t,d}(z_{t-1,i}; \boldsymbol{\theta}_{t,d}^{(1)}) \neq g_{t,d}(z_{t-1,i}; \boldsymbol{\theta}_{t,d}^{(2)}).$$

Informally, inflection transformations are able to change the second derivative through trainable parameters. Restricted to conditioner transformations, we have the additional aspect that $\mathcal{S}_{ext}(t, d)$ affect the second derivative. The reason we concern ourselves with inflection points can be seen through the induced density and its modality. It was first pointed out comparing neural transformations Ψ^+ with affine transformations (Huang, Krueger et al. 2018). We expand on their comment here.

Observation A.3.2. *Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow with the base density $q_{\boldsymbol{z}_0}$, where the derivative exists w.r.t a points predecessor, for all points in its domain and $\mathcal{S}$ is an IAR-structure without permutations Figure 3.4. The derivative of the induced conditional density $q_{\boldsymbol{z}_T}(z_{T,d} \mid z_{T,1:d-1})$ can be written as*

$$\begin{aligned}\frac{\partial}{\partial z_{T,d}} q_{\boldsymbol{z}_T}(z_{T,d} \mid \boldsymbol{z}_{T,1:d-1}) = &\frac{\partial}{\partial z_{T,d}} \left[q_{\boldsymbol{z}_0}(f^{-1}(z_{T,d}) \mid f^{-1}(\boldsymbol{z}_{T,1:d-1})) \right] |\det(\mathcal{J})| \\ &+ q_{\boldsymbol{z}_0}(z_{0,d} \mid \boldsymbol{z}_{0,1:d-1}) | \frac{\partial}{\partial z_{T,d}} \left[\det(\mathcal{J}) \right] |\end{aligned} \tag{A.3}$$

where

$$|\det(\mathcal{J})| = \prod_{t\in\mathcal{T}} \left| \frac{\partial}{\partial z_{t-1,d}} f_{t,d}^{-1}(z_{t-1,d}) \right|,$$

and the Jacobian is w.r.t the composition of functions $f_{t,d}$, with $t \in \mathcal{T}$. We can then see the following, when $\boldsymbol{z}_{T,1:d-1}$ is held fixed and comparing the modes from the base distribution:

(i) *Non-inflection transformations that contains more than one piece in the piecewise diffeomorphism definition, cannot add continuous modes to the density $q_{\boldsymbol{z}_T}(z_{T,d} \mid z_{T,1:d-1})$, through its parametrisation.*

(ii) *If the non-inflection transformation consists of one piece, then it cannot change number of modes to the density $q_{\boldsymbol{z}_T}(z_{T,d} \mid z_{T,1:d-1})$, through its parametrisation.*

Proof. Firstly note that the function, due to no permutation in the structure, is one dimensional once $\boldsymbol{z}_{0,1:d-1}$ is known. The observation then follows from the fact that the Jacobian in the last term of Equation (A.3), is a constant if it is not an inflection transformation, hence the added "through its parametrisation". Without loss of generality, consider when the constants in the last term are 0. Inside any piece of the transformation, we have that $|\det(\mathcal{J})| > 0$, and the piecewise transformation simply multiplies the base distribution with something positive. That means the derivative of the density is still 0 where it is 0 in the the corresponding area of the base distribution. In the boundary points of the pieces, we may add discontinuities, where $|\det(\mathcal{J})|$ are different from one point to next, which opens up for adding more modes. When the transformation consists of one piece, i.e., the whole real line has the same $|\det(\mathcal{J})|$, we see that Equation (A.3) is 0 only when it is 0 in the corresponding point in the base distribution. ■

The observation also highlights the difference between transformations and their complexity that often vanishes in universality claims. We could continue looking at the third derivative of the transformation and its parametrisation, which can be interpreted as being able to influence the change in curvature of the density through learnable parameters. This would elevate transformations such as spline transformations with cubic splines. However, it is not a useful characteristic to classify transformations at this point, as the transformations in the literature do not create new classes with regard to the third derivative, when first classified through inflection transformations. Hence, we leave it as it is.

Bounded and Unbounded Parametrisation Transformations

Moving on to the second property we concern ourselves with.

Definition A.3.3. Let $(\mathcal{Q}, \mathcal{S}, f)$ be a flow.

(i) A transformation $f_{t,d}$ is an *unbounded parametrisation transformation* if the number of parameters needed to transform $z_{t-1,d}$ can be any $\rho_{t,d} \in \mathbb{N}$.

(ii) A transformation where there exists a constant $\hat{\rho}_{t,d} \in \mathbb{N}$ such that the number of parameters do not exceed the constant $\rho_{t,d} < \hat{\rho}_{t,d}$, is called a *bounded parametrisation transformation*.

An example of unbounded parametrisation transformations are spline transformations, which can set the amount of knots, hence amount of parameters, arbitrarily high.

With this distinction, we have spline-, neural network-, and residual transformations as unbounded parametrisation transformations. Likewise, linear- and affine transformations are bounded parametrisation transformations. The unbounded parametrisation transformations may scale the number of parameters to fit the need, and in this specific term, have arbitrary complexity. The $\rho_{t,d}$ can therefore be seen as a hyperparameter that may be tuned according to the flow's performance. The bounded parametrisation transformations can also add complexity, but must do so by adding compositions t.

The difference between bounded and unbounded parametrisation can at first glance seem vacuous, as for unbounded transformations we may simply set the number of transformations to a certain number and achieve transformations that act as a bounded parametrisation transformation. However, there are two points worth consideration. Firstly, although an unbounded parametrisation can act as to be bounded, that does not mean the reverse is true. Hence, there is a separation in complexity, where bounded parametrisation transformations are limited. Secondly, the distinction of unbounded/bounded parametrisation is useful to analyse the difference of simpler bounded transformations and several time steps t, or rather include the complexity through increasing the number of parameters. There are a couple of features to consider:

(i) The added computational burden of more parameters w.r.t. forward/inverse and the Jacobian determinant vs. the added benefits of a more flexible transformation.

(ii) When using a neural network as a conditioner, a larger network to accompany larger output space vs. smaller but several networks using several time steps t.

Point (i) is has partially been dealt with in Chapter 3, as we discussed the added computational burden with the more complex transformations, e.g., searching for bucket in splines, inverting Ψ^+ and computing its Jacobian determinant, etc. The added benefits of added complexity in the transformation itself was partially addressed under universality. Any other comparisons possible is of empirical nature and left for Chapter 5.

For point (ii), we deliberate further and consider conditioner transformations. The question becomes—given that bounded parametrisation transformations are typically fast to compute, invert, and compute the derivative—how does the conditioner handle the added complexity through a number of time steps compared to a number of parameters. Assuming the conditioner is a neural network, when increasing the number of parameters in a transformation, one needs to also extend the size of the network, particularly the number of nodes in each hidden layer. If we wish to still have universal approximation, following Theorem 2.4.9, we need to add a neuron in each hidden layer for each added parameter in the transformation—preferably more due to not having arbitrary

depth. However, the number of weights added, given $\tilde{D}$ number of neurons in each hidden layer with L layers, is $(L-1)(2\tilde{D}+1)$. Adding an extra transformation adds $(L-1)\tilde{D}$. It is also worth considering the strain put on the neural network when the input size is much lower than the output size, in other words, finding mappings from a low-dimensional space to a high-dimensional one. Hence it seems like the added benefit of adding new parameters drops off compared to adding new transformations, which is also supported empirically in the literature—although this also include permutations which play a big role vs. only one transformation—(Kobyzev et al. 2020). A combination seems therefore to be perfect, but of which disregard the cases where bounded parametrisation transformations are the only option due to computational limits (e.g. high dimensional data).

A small summary of the classifications and some of the transformations we have countered so far, is given in Table A.1.

Table A.1: A summary of bounded- or unbounded parametrisation transformations and inflection transformations, with every transformation introduced in Chapter 3 classified accordingly.

	Bounded Param.	Unbounded Param.
Non-Inflection	Affine, Linear	Linear Splines
Inflection	—	Rational Quadratic Splines, Cubic Splines, Neural Networks, Residual

A.4 Proof of Continuity of $b_{t,d}$

We prove the lemma that claims continuity of $f_{t,d}$ w.r.t $b_{t,d}$.

Lemma A.4.1. *Let $(\mathcal{Q}, \mathcal{S}, \mathcal{H}, f)$ be a flow with limited piecewise affine transformations and an IAR-structure. Then, for all $d \in \mathcal{D}$ and $t \in \mathcal{T}$, the transformation $f_{t,d}$ is continuous w.r.t. $b_{t,d}$.*

Proof. We only focus showing continuity for h_t for one t, as the identity function of b_t is continuous, as well as we have preservation of continuity when it comes to addition and function composition. What we are going to show holds is the following. For every z_d, for each $\epsilon > 0$, and for each a, there exist a $\delta > 0$, namely $\delta = \epsilon/2a$, such that

$$|b - \tilde{b}| < \delta \implies |h_d(z_d - b_d) - h_d(z_d - \tilde{b}_d)| < \epsilon.$$

To show that this is true, we consider four different cases.

Case 1: Consider when both $z_d - b_t > 0$ as well as $z_d - (b_t \pm \delta) > 0$ (obviously it might only hold for $+\delta$, in which case we only consider that one). Then we have

$$|a_t(z_d - b_t) - a_t(z_d - (b_t \pm \delta))| = |\pm\delta| = \frac{\epsilon}{2a} < \epsilon,$$

where we use $\delta = \epsilon/2a$.

Case 2: Consider when both $z_d - b_t \leq 0$ and $z_d - (b_t \pm \delta) \leq 0$. Then we have

$$|(z_d - b_t) - (z_d - (b_t \pm \delta))| = |\pm\delta| = \frac{\epsilon}{2a} < \epsilon.$$

Case 3: Consider when $z_d - b_t > 0$ and $z_d - (b_t + \delta) \leq 0$, which also means $b_t < z_d \leq (b_t + \delta)$. We then have

$$|a_t(z_d - b_t) - (z_d - (b_t + \delta))| = |z_d(a_t - 1) - b_t(a_t - 1) + \delta|.$$

We can here consider three subcases. The first is when $a_t = 1$, then obviously have

$$|z_d(a_t - 1) - b_t(a_t - 1) + \delta| < |\delta| = \frac{\epsilon}{2a} < \epsilon.$$

If $a_t > 1$, we have, keeping in mind the bounds on z_d, we have

$$|z_d(a_t - 1) - b_t(a_t - 1) + \delta| \leq |(b_t + \delta)(a_t - 1) - b_t(a_t - 1) + \delta| = |a_t\delta| = \frac{\epsilon}{2} < \epsilon.$$

And finally, if $a_t < 1$, we have

$$|z_d(a_t - 1) - b_t(a_t - 1) + \delta| < |b_t(a_t - 1) - b_t(a_t - 1) + \delta| = |\delta| = \frac{\epsilon}{2a} < \epsilon.$$

Case 4: Finally, consider when $z_d - b_t \leq 0$, while $z_d - (b_t - \delta) > 0$, which gives us the bounds $(b_t - \delta) < z_d \leq b_t$. We then have

$$|(z_d - b_t) - a_t(z_d - (b_t - \delta))| = |z_d(1 - a_t) - b_t(1 - a_t) - a_t\delta|.$$

Considering again three subcases. When $a_t = 1$, we have

$$|z_d(1 - a_t) - b_t(1 - a_t) - a_t\delta| = |-a_t\delta| = \frac{\epsilon}{2} < \epsilon.$$

When $a_t > 1$ we have, keeping in mind the boundaries given above,

$$|z_d(1 - a_t) - b_t(1 - a_t) - a_t\delta| \leq |b_t(1 - a_t) - b_t(1 - a_t) - a_t\delta| = |-a_t\delta| = \frac{\epsilon}{2} < \epsilon.$$

And finally, when $a_t < 1$, we have

$$|z_d(1 - a_t) - b_t(1 - a_t) - a_t\delta| < |(b_t - \delta)(1 - a_t) - b_t(1 - a_t) - a_t\delta| = |-\delta| = \frac{\epsilon}{2a} < \epsilon.$$

Hence, $h_t(z_d - b_t)$ is continuous w.r.t. b_t for all $t \in \mathcal{T}$ and $d \in \mathcal{D}$. By the argument in the start of the proof, it follows that f_d is continuous w.r.t. b_t for all $t \in \mathcal{T}$ and $d \in \mathcal{D}$. ■

A.5 Continuity of Target Inverse CDF

We here prove the following lemma.

Lemma A.5.1. *For any probability distribution $\mathcal{P} \in \mathscr{P}_2$, let the conditional CDF be denoted by $F_{\mathcal{P}}(x_d \mid \boldsymbol{x}_{1:d-1})$ for any $d \in \mathcal{D}$. Then the inverse $F_{\mathcal{P}}^{-1}(\hat{x}_d \mid \boldsymbol{x}_{1:d-1})$, where $\hat{x}_d \in (0, 1)$ is strictly increasing and continuous w.r.t both $\hat{x}_d$ and $\boldsymbol{x}_{1:d-1}$.*

Proof. Let $F \colon \mathbb{R}^D \to [0, 1]^D$ with the dth output defined as the conditional CDF to $p_{\boldsymbol{x}}$, i.e. $\hat{x}_d = F_d(x_d \mid x_{1:d-1}) = Pr(X_d < x_d \mid x_{1:d-1})$. With $\boldsymbol{x} \in [0, 1]^D$. When we are working with $\boldsymbol{x}_{1:d-1}$, we are implicitly restricting possible values such that $p_{\boldsymbol{x}}(x_d \mid \boldsymbol{x}_{1:d-1}) > 0$ for some $x_d \in \mathbb{R}$. Due to $\mathscr{P}$, we have that the set of possible values $\boldsymbol{x}_{1:d-1}$ is a connected subset of $\mathbb{R}^{d-1}$, hence when we have a

$\boldsymbol{x}_{1:d-1}$ and talk about $||\boldsymbol{x}_{1:d-1} - \tilde{\boldsymbol{x}}_{1:d-1}||_\infty$ we talk about the set which fulfill the inequality and also are possible values. They in themselves comprise of a connected subspace which is never empty nor only $\boldsymbol{x}_{1:d-1}$. Going forward we are implictly adding this restriction.

When $x_{1:d-1}$ is fixed, we have two numbers $l_0^d < l_1^d$ (we allow for $\pm\infty$), such that it is strictly monotonically increasing when $x_d \in [l_0^d, l_1^d]$ (obviously the set is open when $\pm\infty$) per the requirement of strictly positive density, 0 when $x_d < l_0^d$ and 1 when $x_d > l_1^d$. We can also see, due to continuity in the conditional, that the boundary points when restricted to the strictly monotonically increasing part can change, but only continuously in the same manner as with G_d. Think of it as the part that is 0 and 1 in the F_d can only change slightly and only the part that is close to the stricly increasing part, otherwise we break the continuity of F_d w.r.t. $\boldsymbol{x}_{1:d-1}$.

Let $F_d^{-1}(\hat{x}_d \mid \boldsymbol{x}_{1:d-1})$ be defined as the inverse of $F_d(x_d \mid \boldsymbol{x}_{1:d-1})$, where the image of the inverse is simply the values mapping to the strictly increasing part. We call this interval for $I \subseteq \mathbb{R}$, so $F_d \colon I \to [0,1]$ is strictly increasing. We now show continuity for the inverse w.r.t. both $\hat{x}_d$, and also w.r.t. $\boldsymbol{x}_{1:d-1}$. When $\boldsymbol{x}_{1:d-1}$ is fixed, means the inverse F_d^{-1} is continuous w.r.t. x_d, as F_d restricted to the interval that is the image of F_d^{-1} is strictly increasing and continuous. This is easy to see, as for any $\epsilon > 0$ and any $x_d \in I$, we have

$$F_d(x_d - \epsilon \mid \boldsymbol{x}_{1:d-1}) < F_d(x_d \mid \boldsymbol{x}_{1:d-1}) < F_d(x_d + \epsilon \mid \boldsymbol{x}_{1:d-1}),$$

which by setting δ to be the minimum of $|F_d(x_d \pm \epsilon \mid \boldsymbol{x}_{1:d-1} - F_d(x_d \mid \boldsymbol{x}_{1:d-1})|$ (some small minor details when x_d is a boundary point in I or if $x_d \pm \epsilon \notin I$, however this is easy to handle by considering left/right continuity in the boundary case and simply picking some points closer towards $F_d(x_d \mid \boldsymbol{x}_{1:d-1})$ in the second case).

Next we look at continuity w.r.t. $\boldsymbol{x}_{1:d-1}$. Let $\epsilon > 0$ and for any $\boldsymbol{x}_{1:d-1}$ we have the following. Let $\delta_1 > 0$ be set so that

$$|\hat{x}_d - \hat{x}'_d| < \delta_1 \implies |F_d^{-1}(\hat{x}_d \mid \boldsymbol{x}_{1:d-1}) - F_d^{-1}(\hat{x}'_d \mid \boldsymbol{x}_{1:d-1})| < \epsilon.$$

Using continuity in conditional CDF, we can find $\delta_2 > 0$ such that whenever $||\boldsymbol{x}_{1:d-1} - \tilde{\boldsymbol{x}}_{1:d-1}||_\infty < \delta_2$ we have

$$|F_d(x_d \mid \boldsymbol{x}_{1:d-1}) - F_d(x_d \mid \tilde{\boldsymbol{x}}_{1:d-1})| < \delta_1. \tag{A.4}$$

Let

$$\tilde{X} = \{\tilde{\boldsymbol{x}}_{1:d-1} \colon ||\tilde{\boldsymbol{x}}_{1:d-1} - x_{1:d-1}||_\infty < \delta_2\}$$

a mapping $\eta \colon \tilde{X} \to (0,1]$ defined as

$$\eta(\tilde{\boldsymbol{x}}_{1:d-1}) = \sup\{\delta \colon \forall \hat{x}'_d : |\hat{x}_d - \hat{x}'_d| < \delta \implies |F_d^{-1}(\hat{x}_d \mid \tilde{\boldsymbol{x}}_{1:d-1}) - F_d^{-1}(\hat{x}'_d \mid \tilde{\boldsymbol{x}}_{1:d-1})| < \epsilon\}.$$

This mapping simply take the larges δ that fulfills continuity w.r.t. $\hat{x}_d$ or 1, if the value can be larger than one. We know at least one such δ exist, as we know there continuity when what we condition on is fixed. Let then $\delta_3 > 0$ be defined as

$$\delta_3 = \inf\{\delta \colon \tilde{\boldsymbol{x}}_{1:d-1} \in \tilde{X} \text{ and } \delta = \eta(\tilde{\boldsymbol{x}}_{1:d-1})\}$$

and set δ_4 equivalently to how we set δ_2 using Equation (A.4), but replacing δ_1 with δ_3. For any $\tilde{\boldsymbol{x}}_{1:d-1}$, let $\hat{x}_d = F_d(x_d \mid \boldsymbol{x}_{1:d-1})$ and $\hat{x}'_d = F_d(x_d \mid \tilde{\boldsymbol{x}}_{1:d-1})$, then whenever $|\boldsymbol{x}_{1:d-1} - \tilde{\boldsymbol{x}}_{1:d-1}| < \delta_4$ we have

$$
\begin{aligned}
&|F_d^{-1}(\hat{x}_d \mid \boldsymbol{x}_{1:d-1}) - F_d^{-1}(\hat{x}_d \mid \tilde{\boldsymbol{x}}_{1:d-1})| \\
\leq &|F_d^{-1}(\hat{x}_d \mid \boldsymbol{x}_{1:d-1}) - F_d^{-1}(\hat{x}'_d \mid \tilde{\boldsymbol{x}}_{1:d-1})| + |F_d^{-1}(\hat{x}'_d \mid \tilde{\boldsymbol{x}}_{1:d-1}) - F_d^{-1}(\hat{x}_d \mid \tilde{\boldsymbol{x}}_{1:d-1})| \\
= &0 + |F_d^{-1}(\hat{x}'_d \mid \tilde{\boldsymbol{x}}_{1:d-1}) - F_d^{-1}(\hat{x}_d \mid \tilde{\boldsymbol{x}}_{1:d-1})|.
\end{aligned}
$$

Due to continuity w.r.t. $\boldsymbol{x}_{1:d-1}$, we have $|\hat{x}_d - \hat{x}'_d| < \delta_3$, hence we have

$$|F_d^{-1}(\hat{x}'_d \mid \tilde{\boldsymbol{x}}_{1:d-1}) - F_d^{-1}(\hat{x}_d \mid \tilde{\boldsymbol{x}}_{1:d-1})| < \epsilon.$$

■

A.6 Experimental Results

Experiments 1 & 2

We include results for every of the 42 models, for both experiment 1 and 2, where every model have calculated mean log-likelihood of the 5 different runs, for training and test set, with confidence intervals. We also include the true mean log-likelihood in each experiment as an extra comparison. As we sampled data for different flows, the true log-likelihood is then placed at the bottom of every group of flows which used the same sample. First, we refresh the reader with the table of the different models in Table A.2. All the results of the 42

Table A.2: List of all models tested in experiment 1 and 2.

	Structure	Transformations	Permutations
Flow 1:	Identity	Affine	Id.
Flow 2:	Fully	Linear (SO_D)	Id.
Flow 3:	$D/2$-coupling	Affine	Id., Alt., Random
Flow 4:	AR	Affine	Id., Alt., Random
Flow 5:	$D/2$-coupling	Piecewise Affine	Id., Alt., Random
Flow 6:	AR	Piecewise Affine	Id., Alt., Random
Flow 7:	$D/2$-coupling	Affine Piecewise Affine	Id., Alt., Random
Flow 8:	AR	Affine Piecewise Affine	Id., Alt., Random
Flow 9:	$D/2$-coupling	Cont. Piecewise Affine	Id., Alt., Random
Flow 10:	AR	Cont. Piecewise Affine	Id., Alt., Random
Flow 11:	$D/2$-coupling	Affine Cont. Piecewise Affine	Id., Alt., Random
Flow 12:	AR	Affine Cont. Piecewise Affine	Id., Alt., Random
Flow 13:	$D/2$-coupling	Affine/Piecewise Affine	Random
Flow 14:	AR	Affine/Piecewise Affine	Random
Flow 15:	ID/$(D/2)$-coupling	Linear/Affine	Id., Random
Flow 16:	ID/AR	Linear/Affine	Id.,Random
Flow 17:	ID/$(D/2)$-coupling	Linear/Piecewise Affine	Random
Flow 18:	ID/$(D/2)$-coupling	Linear/Cont. Piecewise Affine	Random
Flow 19:	ID/$(D/2)$-coupling	Linear/Affine Piecewise Affine	Random
Flow 20:	ID/$(D/2)$-coupling	Linear/Affine Cont. Piecewise Affine	Random

flows are given in Table A.3.

Table A.3: Results of experiment 1 and 2, identity (Id), alternating (Alt) and random (Ra) permutation (higher is better).

	Train exp. 1	Test exp.1	Train exp. 2	Test exp. 2
Flow 1	-49.58 ± 0.190	-49.43 ± 0.194	-51.66 ± 0.000	-51.73 ± 0.001
Flow 2	-60.36 ± 0.008	-59.80 ± 0.011	-65.63 ± 0.003	-65.92 ± 0.012
True value (1,2)	-46.45	-46.45	-30.74	-30.63
Flow 3-Id	-46.32 ± 0.042	-46.74 ± 0.009	-51.92 ± 0.216	-55.78 ± 0.281
Flow 3-Alt	-46.31 ± 0.066	-46.74 ± 0.035	-51.95 ± 0.150	-55.86 ± 0.130
Flow 3-Ra	-46.22 ± 0.023	-46.69 ± 0.056	-52.15 ± 0.306	-55.79 ± 0.171
True value (3)	-46.53	-46.38	-30.68	-30.59
Flow 4-Id	-46.84 ± 0.046	-46.94 ± 0.054	-51.28 ± 0.076	-51.67 ± 0.080
Flow 4-Alt	-46.60 ± 0.020	-46.67 ± 0.013	-48.60 ± 0.768	-48.86 ± 0.674
Flow 4-Ra	-46.53 ± 0.032	-46.61 ± 0.040	-48.48 ± 0.771	-49.03 ± 0.637
True value (4)	-46.46	-46.42	-30.70	-30.77
Flow 5-Id	-47.00 ± 0.140	-47.28 ± 0.125	-57.35 ± 0.847	-59.09 ± 0.965
Flow 5-Alt	-46.96 ± 0.077	-47.24 ± 0.130	-57.86 ± 0.681	-60.30 ± 0.962
Flow 5-Ra	-47.20 ± 0.175	-47.49 ± 0.214	-56.78 ± 0.501	-59.04 ± 0.579
True value (5)	-46.51	-46.47	-30.87	-30.61
Flow 6-Id	-49.23 ± 0.393	-49.17 ± 0.366	-51.04 ± 0.111	-51.72 ± 0.162
Flow 6-Alt	-47.27 ± 0.129	-47.23 ± 0.141	-57.90 ± 0.462	-58.29 ± 0.427
Flow 6-Ran	-47.28 ± 0.065	-47.23 ± 0.069	-57.32 ± 0.191	-57.71 ± 0.145
True value (6)	-46.46	-46.35	-30.62	-30.51
Flow 7-Id	-46.82 ± 0.054	-46.63 ± 0.028	-56.86 ± 0.172	-57.23 ± 0.153
Flow 7-Alt	-46.77 ± 0.067	-46.61 ± 0.036	-56.87 ± 0.145	-57.29 ± 0.158
Flow 7-Ra	-46.72 ± 0.060	-46.56 ± 0.056	-56.82 ± 0.220	-57.18 ± 0.210
True value (7)	-46.56	-46.24	-30.72	-30.90
Flow 8-Id	-46.79 ± 0.020	-46.71 ± 0.307	-51.04 ± 0.111	-51.72 ± 0.162
Flow 8-Alt	-47.14 ± 0.318	-47.02 ± 0.327	-57.90 ± 0.462	-58.29 ± 0.427
Flow 8-Ra	-48.18 ± 0.438	-48.06 ± 0.470	-57.32 ± 0.191	-57.71 ± 0.145
True value (8)	-46.62	-46.43	-30.62	-30.51
Flow 9-Id	-47.26 ± 0.156	-47.52 ± 0.175	-50.79 ± 1.84	-53.56 ± 1.88
Flow 9-Alt	-47.31 ± 0.138	-47.57 ± 0.118	-53.48 ± 1.336	-55.96 ± 1.518
Flow 9-Ra	-47.39 ± 0.157	-47.68 ± 0.160	-50.40 ± 2.057	-53.84 ± 2.053
True value (9)	-46.51	-46.48	-30.76	-30.67
Flow 10-Id	-50.25 ± 0.647	-50.45 ± 0.657	-56.76 ± 1.350	-56.98 ± 1.365
Flow 10-Alt	-46.94 ± 0.134	-47.05 ± 0.157	-55.57 ± 0.49	55.89 ± 0.464
Flow 10-Ra	-46.95 ± 0.082	-47.07 ± 0.087	-55.09 ± 1.005	-55.47 ± 0.987
True value (10)	-46.46	-46.41	-30.82	-30.93
Flow 11-Id	-46.60 ± 0.062	-46.90 ± 0.071	-31.61 ± 0.288	-33.75 ± 0.183
Flow 11-Alt	-46.63 ± 0.062	-46.89 ± 0.055	-31.76 ± 0.442	-33.74 ± 0.134
Flow 11-Ra	-46.52 ± 0.059	-46.83 ± 0.044	-31.45 ± 0.655	-33.90 ± 0.249
True value (11)	-46.51	-46.34	-30.68	-30.77
Flow 12-Id	-47.25 ± 0.088	-47.37 ± 0.077	-49.81 ± 0.77	-49.89 ± 0.773
Flow 12-Alt	-46.74 ± 0.026	-46.87 ± 0.023	-50.50 ± 0.552	-50.48 ± 0.639
Flow 12-Ra	-46.65 ± 0.017	-46.80 ± 0.019	-49.56 ± 0.694	-49.53 ± 0.742
True value (12)	-46.39	-46.45	-30.64	-30.59
Flow 13	-46.55 ± 0.067	-47.12 ± 0.085	-56.67 ± 0.116	-56.91 ± 0.140
Flow 14	-46.93 ± 0.086	-47.18 ± 0.114	-65.61 ± 0.647	-65.40 ± 0.630
True value (13,14)	-46.39	-46.57	-30.94	-30.27
Flow 15-Id	-46.57 ± 0.063	-46.79 ± 0.075	-39.84 ± 1.328	-44.58 ± 0.202
Flow 15-Ran	-46.52 ± 0.055	-46.74 ± 0.0361	-40.29 ± 1.578	-44.84 ± 0.213
Flow 16-Id	-46.94 ± 0.112	-46.94 ± 0.141	-47.99 ± 0.507	-48.47 ± 0.439

Flow 16-Ra	-46.88 ± 0.049	-46.90 ± 0.069	-48.17 ± 0.405	-48.62 ± 0.378
True value (15,16)	-46.47	-46.46	-30.78	-30.72
Flow 17	-46.77 ± 0.070	-47.08 ± 0.070	-58.03 ± 2.286	-59.12 ± 1.972
Flow 18	-47.33 ± 0.085	-47.60 ± 0.129	-40.51 ± 0.189	-45.06 ± 0.198
Flow 19	-46.64 ± 0.044	-47.032 ± 0.066	-45.002 ± 1.645	-47.21 ± 0.95
Flow 20	-46.51 ± 0.033	-46.90 ± 0.029	-37.67 ± 0.707	-44.12 ± 0.440
True value	-46.47	-46.60	30.74	-30.84

Bibliography

Ardizzone, L. et al. (2019). 'Analyzing Inverse Problems with Invertible Neural Networks'. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Arnold, V. I. (2009). 'On functions of three variables'. In: *Collected Works: Representations of Functions, Celestial Mechanics and KAM Theory, 1957–1965*, pp. 5–8.

Bader, P., Blanes, S. and Casas, F. (2019). 'Computing the Matrix Exponential with an Optimized Taylor Polynomial Approximation'. In: *Mathematics* vol. 7, no. 12.

Behrmann, J. et al. (2019). 'Invertible Residual Networks'. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Chaudhuri, K. and Salakhutdinov, R. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 573–582.

Berg, R. van den et al. (2018). 'Sylvester Normalizing Flows for Variational Inference'. In: *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*. Ed. by Globerson, A. and Silva, R. AUAI Press, pp. 393–402.

Bhattacharyya, A. (1946). 'On a Measure of Divergence between Two Multinomial Populations'. In: *Sankhyā: The Indian Journal of Statistics (1933-1960)* vol. 7, no. 4, pp. 401–406.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag.

Blei, D. M., Kucukelbir, A. and McAuliffe, J. D. (2017). 'Variational Inference: A Review for Statisticians'. In: *Journal of the American Statistical Association* vol. 112, no. 518, pp. 859–877.

Bogachev, V., Kolesnikov, A. and Medvedev, K. (Oct. 2007). 'Triangular transformations of measures'. In: *Sbornik: Mathematics* vol. 196, p. 309.

Casado, M. L. (2019). 'Trivializations for Gradient-Based Optimization on Manifolds'. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Wallach, H. M. et al., pp. 9154–9164.

Chen, T. Q. et al. (2019). 'Residual Flows for Invertible Generative Modeling'. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Wallach, H. M. et al., pp. 9913–9923.

Cherief-Abdellatif, B.-E. (Feb. 2019). In: *Proceedings of The 1st Symposium on Advances in Approximate Bayesian Inference*. Vol. 96. Proceedings of Machine Learning Research. PMLR, pp. 11–31.

Cornish, R. et al. (2020). 'Relaxing Bijectivity Constraints with Continuously Indexed Normalising Flows'. In: *ICML*, pp. 2133–2143.

Cybenko, G. (1989). 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of Control, Signals and Systems* vol. 2, no. 4, pp. 303–314.

Dhaka, A. K. et al. (2021). 'Challenges for BBVI with Normalizing Flows'. In: *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*.

Dinh, L., Krueger, D. and Bengio, Y. (2015). 'NICE: Non-linear Independent Components Estimation'. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. Ed. by Bengio, Y. and LeCun, Y.

Dinh, L., Sohl-Dickstein, J. and Bengio, S. (2017). 'Density estimation using Real NVP'. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Durkan, C. et al. (2019a). 'Cubic-Spline Flows'. In: *CoRR* vol. abs/1906.02145. arXiv: 1906.02145.

— (2019b). 'Neural Spline Flows'. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Wallach, H. M. et al., pp. 7509–7520.

Foss, S., Korshunov, D. and Zachary, S. (2011). *An introduction to heavy-tailed and subexponential distributions*. English. Springer Series in Operations Research and Financial Engineering. Springer.

Friedman, J., Hastie, T., Tibshirani, R. et al. (2001). *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York.

Friedman, J. H. (1987). 'Exploratory projection pursuit'. In: *J. Amer. Statist. Assoc.* vol. 82, no. 397, pp. 249–266.

Germain, M. et al. (July 2015). 'MADE: Masked Autoencoder for Distribution Estimation'. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Bach, F. and Blei, D. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 881–889.

Golinski, A., Lezcano-Casado, M. and Rainforth., T. (2019). 'Improving normalizing flows via better orthogonal parameterizations'. In: ICML Workshop on Invertible Neural Networks and Normalizing Flows. ICML.

Goodfellow, I. et al. (2014). 'Generative Adversarial Nets'. In: *Advances in Neural Information Processing Systems*. Ed. by Ghahramani, Z. et al. Vol. 27. Curran Associates, Inc.

Hanin, B. and Sellke, M. (2017). 'Approximating continuous functions by relu nets of minimal width'. In: *arXiv preprint arXiv:1710.11278*.

Hilbert, D. (1902). 'Mathematical problems'. In: *Bulletin of the American Mathematical Society* vol. 8, no. 10, pp. 437–479.

Hochreiter, S. and Schmidhuber, J. (Nov. 1997). 'Long Short-Term Memory'. In: vol. 9, no. 8, pp. 1735–1780.

Hornik, K. (1991). 'Approximation capabilities of multilayer feedforward networks'. In: *Neural Networks* vol. 4, no. 2, pp. 251–257.

Huang, C.-W., Dinh, L. and Courville, A. C. (2020). 'Augmented Normalizing Flows: Bridging the Gap Between Generative Flows and Latent Variable Models'. In: *CoRR* vol. abs/2002.07101. arXiv: 2002.07101.

Huang, C.-W., Krueger, D. et al. (Oct. 2018). 'Neural Autoregressive Flows'. In: ed. by Dy, J. and Krause, A. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, pp. 2078–2087.

Jaini, P., Kobyzev, I. et al. (2020). 'Tails of Lipschitz Triangular Flows'. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 4673–4681.

Jaini, P., Selby, K. A. and Yu, Y. (2019). 'Sum-of-Squares Polynomial Flow'. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Chaudhuri, K. and Salakhutdinov, R. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 3009–3018.

Jang, E., Gu, S. and Poole, B. (2017). 'Categorical Reparameterization with Gumbel-Softmax'. In: *5th International Conference on Learning*

Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net.

Johnson, R. M. (1966). 'The minimal transformation to orthonormality'. In: *Psychometrika* vol. 31, pp. 61–66.

Jordan, M. I. et al. (1999). 'An Introduction to Variational Methods for Graphical Models'. In: *Machine Learning* vol. 37, no. 2, pp. 183–233.

Kaparthy, A. (2018). *pytorch-made*. https://github.com/karpathy/pytorch-made.

Kidger, P. and Lyons, T. (Sept. 2020). 'Universal Approximation with Deep Narrow Networks'. In: ed. by Abernethy, J. and Agarwal, S. Vol. 125. Proceedings of Machine Learning Research. PMLR, pp. 2306–2327.

Kingma, D. P. and Welling, M. (2014). 'Auto-Encoding Variational Bayes'. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Bengio, Y. and LeCun, Y.

Kingma, D. P. and Dhariwal, P. (2018). 'Glow: Generative Flow with Invertible 1x1 Convolutions'. In: *Advances in Neural Information Processing Systems*. Ed. by Bengio, S. et al. Vol. 31. Curran Associates, Inc.

Kingma, D. P., Salimans, T. et al. (2016). 'Improved Variational Inference with Inverse Autoregressive Flow'. In: *Advances in Neural Information Processing Systems 29*. Ed. by Lee, D. D. et al. Curran Associates, Inc., pp. 4743–4751.

Kirichenko, P., Izmailov, P. and Wilson, A. G. (2020). 'Why Normalizing Flows Fail to Detect Out-of-Distribution Data'. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Larochelle, H. et al.

Kobyzev, I., Prince, S. and Brubaker, M. (2020). 'Normalizing Flows: An Introduction and Review of Current Methods'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1.

Kolmogorov, A. N. (1957). 'On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition'. In: *Doklady Akademii Nauk*. Vol. 114. 5. Russian Academy of Sciences, pp. 953–956.

Kullback, S. and Leibler, R. A. (1951). 'On information and sufficiency'. In: *The annals of mathematical statistics* vol. 22, no. 1, pp. 79–86.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). 'Deep Learning'. In: *Nature* vol. 521, no. 7553, pp. 436–444.

LeCun, Y., Boser, B. et al. (1990). 'Handwritten Digit Recognition with a Back-Propagation Network'. In: *Advances in Neural Information Processing Systems*. Ed. by Touretzky, D. Vol. 2. Morgan-Kaufmann.

Lindstrøm, T. L. (2017). *Spaces: An Introduction to Real Analysis*. American Mathematical Society.

Lu, Z. et al. (2017). 'The Expressive Power of Neural Networks: A View from the Width'. In: *Advances in Neural Information Processing Systems*. Ed. by Guyon, I. et al. Vol. 30. Curran Associates, Inc.

McCulloch, W. S. and Pitts, W. (1943). 'A logical calculus of the ideas immanent in nervous activity'. In: *The bulletin of mathematical biophysics* vol. 5, no. 4, pp. 115–133.

Müller, T. et al. (2019). 'Neural Importance Sampling'. In: *ACM Trans. Graph.* vol. 38, no. 5, 145:1–145:19.

Oliva, J. B. et al. (2018). 'Transformation Autoregressive Networks'. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Dy, J. G. and Krause, A. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 3895–3904.

Papamakarios, G., Pavlakou, T. and Murray, I. (2017). 'Masked Autoregressive Flow for Density Estimation'. In: *Advances in Neural Information Processing Systems 30*. Ed. by Guyon, I. et al. Curran Associates, Inc., pp. 2338–2347.

Pinkus, A. (1999). 'Approximation theory of the MLP model'. In: *Acta Numerica 1999: Volume 8* vol. 8, pp. 143–195.

Rezende, D. J. and Mohamed, S. (2015). 'Variational Inference with Normalizing Flows'. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by Bach, F. R. and Blei, D. M. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1530–1538.

Saul, L. K., Jaakkola, T. and Jordan, M. I. (1996). 'Mean field theory for sigmoid belief networks'. In: *Journal of artificial intelligence research* vol. 4, pp. 61–76.

Scheffe, H. (Sept. 1947). 'A Useful Convergence Theorem for Probability Distributions'. In: *Ann. Math. Statist.* vol. 18, no. 3, pp. 434–438.

Schmidhuber, J. (Jan. 2015). 'Deep Learning in Neural Networks'. In: *Neural Netw.* vol. 61, no. C, pp. 85–117.

Sklar, A. (1959). 'Fonctions de répartition à n dimensions et leurs marges'. In: *Publ. Inst. Statist. Univ. Paris* vol. 8, pp. 229–231.

Sriperumbudur, B. K. et al. (2009). 'On integral probability metrics,ϕ-divergences and binary classification'. In: *arXiv preprint arXiv:0901.2698*.

Teshima, T. et al. (2020). 'Coupling-based Invertible Neural Networks Are Universal Diffeomorphism Approximators'. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Larochelle, H. et al.

Tomczak, J. and Welling, M. (Nov. 2016). 'Improving Variational Auto-Encoders using Householder Flow'. In:

Vaswani, A. et al. (2017). 'Attention is All you Need'. In: *Advances in Neural Information Processing Systems*. Ed. by Guyon, I. et al. Vol. 30. Curran Associates, Inc.

Vehtari, A., Gabry, J. et al. (2020). *loo: Efficient leave-one-out cross-validation and WAIC for Bayesian models*. R package version 2.4.1.

Vehtari, A., Simpson, D. et al. (2021). 'Pareto smoothed importance sampling'. In: *arXiv preprint arXiv:1507.02646*.

Wainwright, M. J. and Jordan, M. I. (2008). 'Graphical Models, Exponential Families, and Variational Inference'. In: *Found. Trends Mach. Learn.* vol. 1, no. 1-2, pp. 1–305.

Wehenkel, A. and Louppe, G. (2019). 'Unconstrained Monotonic Neural Networks'. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Wallach, H. M. et al., pp. 1543–1553.

— (2020). 'You say Normalizing Flows I see Bayesian Networks'. In: *arXiv preprint arXiv:2006.00866v2*.

— (2021). 'Graphical Normalizing Flows'. In: *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*. Ed. by Banerjee, A. and Fukumizu, K. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 37–45.

Wiese, M., Knobloch, R. and Korn, R. (2019). *Copula and Marginal Flows: Disentangling the Marginal from its Joint*. arXiv: 1907.03361 [cs.LG].

Wolpert, D. H. and Macready, W. G. (Apr. 1997). 'No Free Lunch Theorems for Optimization'. In: *Trans. Evol. Comp* vol. 1, no. 1, pp. 67–82.

Yao, Y. et al. (2018). 'Yes, but Did It Work?: Evaluating Variational Inference'. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by

Dy, J. G. and Krause, A. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 5577–5586.

Yu, Y. et al. (Sept. 2019). 'DAG-GNN: DAG Structure Learning with Graph Neural Networks'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Chaudhuri, K. and Salakhutdinov, R. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 7154–7163.

Zhang, C. et al. (2019). 'Advances in Variational Inference'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 41, no. 8, pp. 2008–2026.

Zheng, X. et al. (2018). 'DAGs with NO TEARS: Continuous Optimization for Structure Learning'. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Bengio, S. et al., pp. 9492–9503.

Aasan, M. (2021). *Invertible Encoders*. https://github.com/PolterZeit/invertible_encoders.