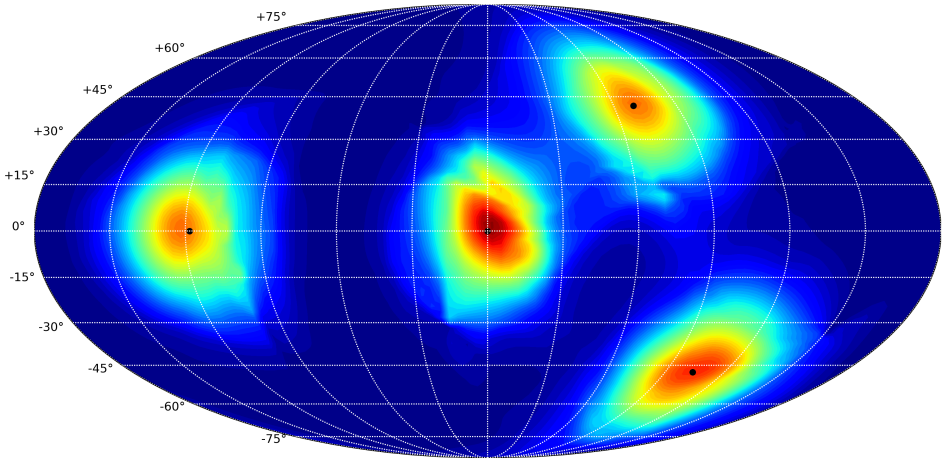
University of Amsterdam

MSc Artificial Intelligence
Master Thesis

Circle Slice Flows and the Variational Determinant Estimator



by
Simon Passenheim
STUDENT ID: 12394130

December 31, 2020

48 ECTS

Supervisor:
Emiel Hoogeboom MSc

Assessor:
Dr. Efstratios Gavves

Contents

Abstract

This thesis introduces *Circle Slice Flows* and the *Variational Determinant Estimator* (VDE). The former is a novel type of normalizing flow, which allows density estimation on the hypersphere by defining a flow on the sphere directly and without breaking the underlying object's geometry. We benchmark the novel flow against the recently introduced spherical flows by Rezende et al. (2020) in terms of density estimation.

The latter is a variational extension of the recently proposed determinant estimator discovered by Sohl-Dickstein (2020). The estimator combines importance-weighted variational inference with spherical normalizing flows, only relies on matrix-vector products, and significantly reduces the variance even for low sample sizes. We benchmark the performance of the VDE in comparison to a naive Monte Carlo approach based on dense matrices and a structured matrix stemming from a linear convolution operator.

1 Introduction

Deep generative models have gained much interest in recent years in Machine Learning and contributed greatly to the success of this field in the last decade. Applications of generative models are diverse and range from high resolution image (Uzunova et al., 2019) and audio (Oord et al., 2016a) synthesis, super-resolution (Ledig et al., 2017), protein folding prediction (Senior et al., 2020), and more can be found in the following Normalizing Flows paragraph in this section.

Initially, we would like to give some context to understand where generative models are placed in Machine Learning. A common problem in statistics and nowadays also in Machine Learning is to have a data set $\{x_i, y_i\}_{i=1}^N$ of observations x_i and associated discrete or continuous labels y_i for which we would like to predict the label of a new observation. For this problem we may distinguish two distinct approaches: *discriminative* and *generative* modeling (Ng, Jordan, 2001). Both approaches share a common assumption: there is a real-world probability distribution p_{data} according to which the data behaves. The former aims to learn a parameterized model $p_\theta(y|x)$ of the true underlying predictive distribution $p_{\text{data}}(y|x)$ by minimizing some notion of distance between both distributions. Well known representatives are for example Support Vector Machines (SVMs) (Cortes, Vapnik, 1995), traditional, fully connected (MLPs) (Nazzal et al., 2008) or convolutional neural networks (CNNs) (LeCun et al., 1999).

Generative Modeling in contrast, is a much more general approach and aims to learn a parameterized model of the joint probability of observations and labels or just of the observations in the case of an unlabeled data set

$$p_\theta(x, y) \quad \text{or} \quad p_\theta(x)$$

In both cases, this is done by solving some notion of distance between the model and the true data distribution. In the former case, one can infer the predictive distribution $p_\theta(y|x)$ with Bayes rule and base the prediction on the argument, which maximizes the model distribution. Why do we consider learning a joint model instead of the predictive distributions directly if that is a more challenging task? The approach offers the following advantages:

(i) **Robustness:** Having a joint model is a fundamentally more general task. Discriminative models only learn a decision boundary between different labels. Joint probability models can capture the relationship and patterns between all components of the data and therefore provide a better understanding of the data and the data generating process. Hence, we may hope that classifiers based on generative models are more robust to unseen data.

(ii) **Novel Sample Generation:** By Bayes rule, we may obtain the marginal $p_\theta(x)$ in the case of a labeled data set or learn the marginal directly. If we also have a sampling procedure available, we can generate novel samples that were not in the data set, either unconditionally or conditioned on a particular label. OpenAI is quoting Richard Feynman in this context: "What I can not create, I do not understand."[1], underlining the implications of having a joint model of the data.

(iii) **Unsupervised Learning** and **Density Estimation:** We note that we may always denote $x' = (x, y)$. Thus the problem of learning the joint distribution of observables and labels is conceptually equal to learn a model of the probability density $p_\theta(x)$ of an unlabeled data set. Therefore, we are provided with a method in cases where no labels are available even in high dimensional cases, which usually lies outside of the field of classical statistics.

As a drawback, generative models usually require more parameters, and if the only ultimate goal is a prediction task, discriminative models might be more suited. However, generative models provide a powerful method to gain insights into the data's underlying hidden structure. For example, in cases where manual label annotation is costly and, therefore, not straightforwardly available or no intrinsic labels are available.

Normalizing Flows Here, we focus on a specific type of generative models called *Normalizing Flows* (Tabak et al., 2010; Rezende, Mohamed, 2015) which have been been successfully applied in

[1] See https://openai.com/blog/generative-models/

Model	Likelihood based	Exact likelihood estimation	Efficient sampling
GANs	✗	✗	✓
VAEs	✓	✗	✓
AR models	✓	✓	✗
Normalizing Flows	✓	✓	✓

Table 1: Properties and comparison of common types of generative models.

various generative tasks such as image generation (Dinh et al., 2014; Hoogeboom et al., 2019; Kingma, Dhariwal, 2018; Ho et al., 2019), audio generation (Kim et al., 2018; Prenger et al., 2019), reinforcement learning (Tang, Agrawal, 2018; Ward et al., 2019; Mazoure et al., 2020) and physics (Köhler et al., 2019; Kanwar et al., 2020). They have appealing properties, see Table 1, compared to other common types of generative models such as Variational Autoencoders (VAEs) Kingma, Welling (2014), Autoregressive Models (AR) (Oord et al., 2016a,b), and Generative Adversarial Networks (GANs) (Goodfellow et al., 2014). In short, GANs do not model an explicit likelihood and only minimize the value function of a zero-sum game, VAEs only optimize a lower bound on the likelihood, and AR models do not admit an efficient sampling procedure due to the sequential structure of the factorized likelihood. In contrast, Normalizing Flows can offer all of these properties. Papamakarios et al. (2019) and Kobyzev et al. (2020) provide a good overview of this type and current methods.

Non-Euclidean Data Spaces Most work has been done under the assumption that the space of observables is Euclidean and therefore has a flat geometry. However, the manifold hypothesis (Fefferman et al., 2016) states that real-world high dimensional data like images or audio often lie on a lower-dimensional manifold embedded in a Euclidean space. Dimensionality reduction techniques or latent space models which learn a lower-dimensional representation of the data are, for example, based on this assumption. We may distinguish two cases: (i) models for a known manifold structure (Rezende et al., 2020; Bose et al., 2020) and (ii) models where the manifold is unknown upfront (Brehmer, Cranmer, 2020). In this thesis, we focus on the former case. When dealing with angular data, for example, each angle lies on a one-dimensional circle, and the Cartesian product of circles forms a torus such that usual methods are not applicable anymore. Other examples of non-trivial manifolds are the special unitary groups SU(2) and SU(3)[2] which play an essential role in particle physics and quantum chemistry, and Normalizing Flows might be applied for spin density estimation in this context. This shows the importance of density estimation on non-Euclidean manifolds, and in this thesis, we will focus on a specific representative.

1.1 Contributions

Here, we focus on the case where the data lies on a D dimensional hypersphere $\mathbb{S}^D$. Besides its simplicity, the sphere has another attractive property, it is compact and therefore allows for a truly uninformative prior in the form of a uniform distribution, which makes it an appealing study object. Our contributions are two-fold:

(i) **Circle Slice Flows** are a novel method to perform density estimation on $\mathbb{S}^D$ by defining a flow directly on the manifold without the need to unnecessarily breaking the geometry of the sphere. This is based on the following observation: any pair of Euclidean coordinates $(x_i, x_j) \in \mathbb{S}^D$ may be interpreted as lying on a circle $r\mathbb{S}^1$ with some radius $r^2 = x_i^2 + x_j^2$. Subsequently, those *Circle Slices* are transformed with flows reviewed in Section 2. By doing this independently for various pairs, we obtain a flexible transformation on the whole sphere, which can learn complicated, multimodal distributions on high-dimensional hyperspheres.

(ii) **Variational Determinant Estimator (VDE):** As an application of the possibility of density estimation on the hypersphere, we introduce the VDE, which is based on work done by Sohl-Dickstein (2020) who connects the absolute determinant of an invertible matrix with the expectation over matrix-vector products where the vectors have unit norm. Our contribution lies

[2]The special unitary group SU(n) is the Lie group of $n \times n$ unitary matrices with determinant 1.

in introducing a variational distribution on the hypersphere modeled by a flow. This distribution acts as a sample generator of the vectors, significantly decreases the variance of the determinant estimator, and achieves extremely accurate estimates even for every low sample size. Furthermore, we discuss in Section 3.2 and 6 possibilities where the VDE may be applied in an online setting for a moving target matrix A and its application in common Machine Learning tasks. The work has been accepted in the form of a condensed paper at the *Advances in Approximate Bayesian Inference* (AABI) workshop 2021 and can be found on arxiv[3].

1.2 Outline

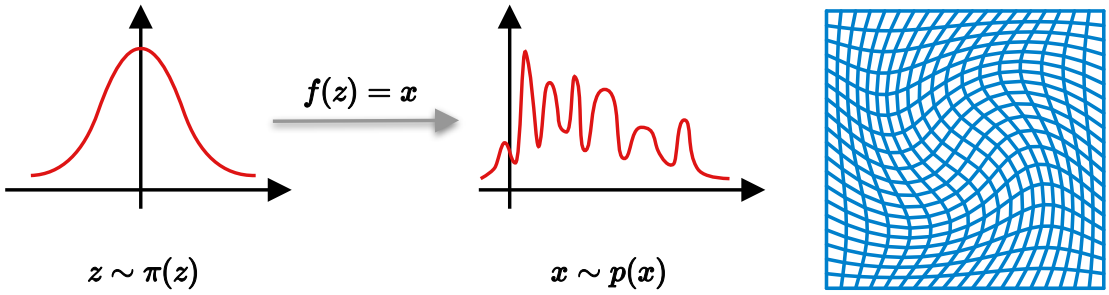
The structure of this thesis is as follows: In Section 2, we review the concept of Normalizing Flows and two novel flows, namely *Moebius Flow* and *Circular Spline Flows*, which act on the one-dimensional circle $\mathbb{S}^1$, introduced by Rezende et al. (2020). Furthermore, we review *Cylindrical Flows*, also introduced by Rezende et al. (2020), which will hold as benchmark line for comparison of our own method. In concluding Section 2, we review the *Power Spherical Distribution* (De Cao, Aziz, 2020), which is similar to the well known von Mises-Fisher distribution function on the hypersphere but admits additionally an efficient sampling procedure, which makes it more convenient to work within higher dimensions. In Section 3, we present our own work in terms of two orthogonal contributions: *Circle Slice Flows* and the *Variational Determinant Estimator*. This is followed by Section 4 in which we discuss the experiments and results of our contributions compared to our benchmark lines. The next Section 5 puts our methods into the greater context and gives a brief overview of other existing methods related to the field of determinant estimation and density estimation on manifolds. Finally, Section 6 concludes the thesis by discussing the results and possible extensions of our work.

[3] https://arxiv.org/abs/2012.13311

2 Background

In this section, we will review the concepts and methods our contributions are based on. In particular, we will first give a brief overview over two flows that act on the circle $\mathbb{S}^1$, *Moebius Flows* (MF) and *Circular Spline Flows* (CSF), and review a flow on $[-1, 1]$, *Interval Spline Flows* (ISF) see (Rezende et al., 2020). This is followed by an introduction of *Cylindrical Flows* (Rezende et al., 2020) which build upon MF, CSF, and ISF to define a flow on $\mathbb{S}^D$ and will hold as a baseline for our contribution in terms of the *Circle Slice Flows*. Finally, we conclude the section by reviewing the recently introduced *Power Spherical Distribution* (De Cao, Aziz, 2020), which is a probability distribution on the hypersphere admitting a fast sampling procedure.

2.1 Normalizing Flows



(a) A flow f transforming a simple density π on Z into a complex density p on X.

(b) A diffeomorphism deforming a square.

Figure 1: Acting mechanism of a diffeomorphism. Left side 1(a): a diffeomorphism transforms a simple base distribution π continuously and differentiably into a complex model distribution p. Right side 1(b): deforming a square $[0, a]^2 \subset \mathbb{R}^2$ of parallel and rectangular lines. The resulting gridlines can not have any sharp corners, which reflects the smoothness criterion.

The idea of Normalizing Flows (Rezende, Mohamed, 2015; Dinh et al., 2016) is to model a real data distribution p_{data} on the space of observables X by transforming a simple base distribution π on a latent space Z into a more complex distribution p on X with a parametrized diffeomorphism $f \colon Z \to X$ which is usually modeled by a neural network, see Figure 1[4] . A diffeomorphism is a differentiable, bijective mapping whose inverse is differentiable as well and is called a *flow* in this context. Note that, if f is a diffeomorphism, the spaces X and Z must have the same dimensionality (Milnor, Weaver, 1997).

Density Update on Euclidean Spaces If both the latent space Z and the space of observables X are Euclidean spaces with flat geometry, the relationship between the complicated density p on X and the simple distribution π on Z is given by the change of variables formula (Rudin, 1987), which connects both densities via the Jacobian determinant of the transformation $f \colon Z \to X$:

$$\log p(x) = \log \pi(f^{-1}(x)) + \log|\det J_{f^{-1}}(x)| = \log \pi(f^{-1}(x)) - \log|\det J_f(f^{-1}(x))| \tag{1}$$

It represents the objective to be optimized. We require $f = f_\lambda$ to depend differentiably on its parameters λ such that gradient based optimization techniques can be applied. Here, we can also see the role of the Jacobian determinant: it acts as a normalizer and ensures that the resulting function p is still a valid probability density with normed area under the curve, hence the name *Normalizing Flows*.

Universal Function Approximator vs. Tractability In principle, any distribution p can be modeled by a flow f, if f is sufficiently complex (Kobyzev et al., 2020), under reasonable assumptions on p and π, such that the notion of a universal function approximator is in general fulfilled. However,

[4]The plot 1(a) is inspired by Lilian Weng and the plot 1(b) is taken from Wikipedia.

calculating the Jacobian determinant of arbitrarily complex functions can be cumbersome and impractical, especially in higher dimensions since this operation is usually of order $\mathcal{O}(D^3)$. However, we can use the fact that the composition of diffeomorphisms is still a diffeomorphism. Therefore, we may model f by a series of simpler flows each with an efficiently computable Jacobian determinant

$$f = f_K \circ f_{K-1} \circ \cdots \circ f_1$$

and still arrive at a flow f which is rich enough to model complex distributions. Then, the log Jacobian determinant of f is simply the sum of the log Jacobian determinants of the individual transformations

$$\log|\det J_f| = \sum_{i=1}^{K} \log|\det J_{f_i}|$$

To ensure that each f_i has an efficiently computable Jacobian determinant, we usually require a triangular structure on the Jacobian J_{f_i}. This is because the log Jacobian determinant is simply the sum of the diagonal elements

$$\log|\det J_g| = \sum_{i=1}^{D} \log \left| \frac{\partial g_i}{\partial x_i} \right|$$

for any differentiable $g\colon \mathbb{R}^D \to \mathbb{R}^D$ such that the Jacobian is triangular. Then, the complexity of calculating the determinant reduces from $\mathcal{O}(D^3)$ to $\mathcal{O}(D)$. We will see in the second next paragraph how to accomplish this.

Efficient Sampling vs. Efficient Density Evaluation First of all, we make one important observation: Since the flow f is a diffeomorphism, we can always model the flow in both directions, that is going from a *simple* space to a *complex* space with $f\colon Z \to X$ or vice versa with $g = f^{-1}\colon X \to Z$. In the following, we consider the implications of both directions.

Forward direction $f\colon Z \to X$: In the former case, sampling is simply done by drawing a sample $z \sim \pi(z)$ in the latent space and subsequently transforming it to generate a sample in the observable space $x = f(z)$ to obtain a sample $x \sim p(x)$. However, we can see from Equation 1 that density evaluation at x requires us to know the inverse f^{-1}.

Backward direction $g\colon X \to Z$: Conversely, if we model the flow in the backward direction, Equation 1 takes the form

$$\log p(x) = \log \pi(g(x)) + \log|\det J_g(x)| \tag{2}$$

which requires us only to know the forward direction of g. Sampling is then done by $z \sim \pi(z)$, followed by $x = g^{-1}(z)$, requiring the inverse of g.

Trade-off: Designing a flow f that allows both directions to be efficiently computed might restrict the complexity of the densities it can model such that in real-world applications we are often confronted with the situation that the other direction needs numerical methods or several iteration steps. Therefore, often a choice in the model direction has to be made depending on the application and if efficient sampling or efficient density evaluation is desired.

Coupling Layers introduced by (Dinh et al., 2014, 2016) are a family of transformations who have a triangular Jacobian and allow in principle an efficient forward and backward pass at the same time, if the flow τ, which transforms the individual components, can also model both directions efficiently. The transformer τ depends on its own parameter set, that is $\tau(\cdot|\psi_i)$, and ψ_i is usually obtained by a neural net. Ideally, we would like to let the flow f, which transforms the whole data vector $(x_1, \ldots, x_D) \in \mathbb{R}^D$, act elementwise on the components via τ, that is

$$f(x_1, \ldots, x_D|\psi_1, \ldots, \psi_D) = (\tau(x_1|\psi_1), \ldots, \tau(x_D|\psi_D)) = (z_1, \ldots, z_D) \tag{3}$$

such that the whole transformation is parallelizable. Note, that we will denote in future Sections 2.2 and 2.3 also the flow acting on the individual components by f for simplicity.

The idea of a coupling layer is to transform half of the data conditioned on the other half of the data, see Figure 2 for an illustration. In terms of equations, the transformation of a single coupling layer

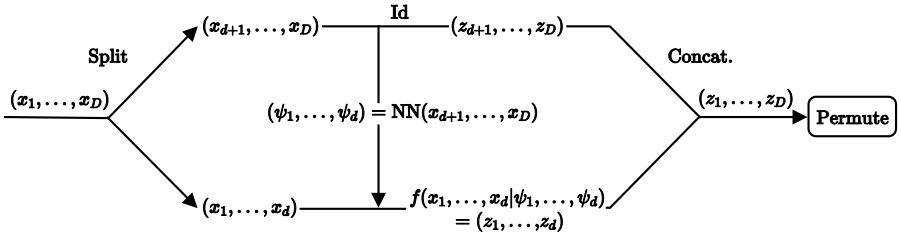


Figure 2: A Coupling Layer: Half of the data is transformed conditioned on the other half and the latter is mapped via the identity with Id = Identity. In the end, a permutation is done such that the other half is transformed in the next layer.

can be expressed as

$$\begin{aligned} z_i &= \tau(x_i|\psi_i) \quad \text{for all} \quad i \leq d \\ z_{d+1:D} &= x_{d+1:D} \end{aligned}$$

where the parameters $(\psi_1,\dots,\psi_d) = \mathrm{NN}\,(x_{d+1},\dots,x_D)$ are obtained with a neural network, $d{+}1{:}D$ denotes all indices from $d+1$ to D and the integer d is close or equal to $D/2$. The neural network is suited to the problem and can be, for example, implemented as a convolutional or fully connected network. Usually, the parameters ψ_i for $i > d$ in Equation 3 are chosen such that τ_i are the identity for $i > d$. As a result of the conditioning structure, the Jacobian of this transformation is triangular and takes the form

$$J_f = \begin{bmatrix} \mathrm{diag}\left(\frac{\partial \tau_i}{\partial x_i}\right)_{i=1}^{d} & \frac{\partial \tau_{1:d}}{\partial x_{d+1:D}} \\ 0 & \mathbb{1}_{d+1:D} \end{bmatrix}$$

where $\mathbb{1}_{d+1:D}$ is an identity matrix of corresponding shape. Therefore, the Jacobian determinant is efficiently to compute it is the product of the diagonal elements. Furthermore, coupling layers are straightforward to invert via

$$\begin{aligned} x_i &= \tau^{-1}(z_i|\psi_i) \quad \text{for all} \quad i \leq d \\ x_{d+1:D} &= z_{d+1:D} \end{aligned}$$

where the parameters are again obtained by $(\psi_1,\dots,\psi_d) = \mathrm{NN}\,(x_{d+1},\dots,x_D)$. When stacking multiple flows, we have to ensure by a permutation of the data that in the subsequent layer the other half is transformed. Dinh et al. (2016) proposed checkerboard masking and channel wise masking whereas the latter is only suitable for spatial data like images. Kingma, Dhariwal (2018) proposed learnable 1×1 convolutions which can be interpreted as a generalization of permutations. In our implementation, we will use rotations which act as a learnable permutation of the data.

The expressivity of a single coupling layer is rather limited since one half of the data always remains unchanged. However, they offer an efficient evaluation conversely in both directions and a tractable Jacobian. It remains still an open question if and under which conditions coupling layers are universal function approximators[5]. Recently Teshima et al. (2020) have claimed that they are indeed universal function approximators, but this has not been widely accepted in the research community according to the best of our knowledge.

Autoregressive Flows also admit a triangular Jacobian and are on top universal function approximator, see Huang et al. (2018). However, for autoregressive flows, only one direction can be efficiently implemented, and the other direction always needs an iteration over all dimensions due to a sequential sampling procedure. Papamakarios et al. (2019) provide an adequate overview of flows based on coupling layers and autoregressive implementation and their relationship.

[5]Except for the (trivial) case where D coupling layers are stacked. In that case, the coupling layer can be constructed such that it is equivalent to an autoregressive flow, see Papamakarios et al. (2019)

Flows on Manifolds So far, we have only considered the case where X and Z are Euclidean spaces with flat geometry and the same dimensionality. However, in this thesis, we will be confronted with two situations:

(i) We would like to calculate the density update for a coordinate chart φ of a manifold embedded in a Euclidean space, which can be seen as a mapping between spaces of different extrinsic dimensionality.

(ii) We would like to calculate the density update of a mapping between manifolds directly.

In both cases, the update rules from Equation 1 and 2, respectively, are not valid anymore.

Let us initially consider the first case. A coordinate chart of a manifold $\mathcal{M}$ is a bijective, continuous mapping with continuous inverse φ^{-1} and

$$\varphi \colon U \subset \mathcal{M} \to \varphi(U) \subset \mathbb{R}^n$$

where U is an open subset of $\mathcal{M}$, $\varphi(U)$ is an open subset in $\mathbb{R}^n$, n is the dimension of the manifold, and $\mathcal{M}$ is embedded in a higher dimensional Euclidean space, that is $\mathcal{M} \subset \mathbb{R}^m$ with $m > n$. Note that φ as $U \to \varphi(U)$ is bijective, even though the spaces they are embedded in have different dimensions. An illustration of a coordinate chart can be seen in Figure 3[6] where we take a manifold $\mathcal{M}$ similar to the sphere as an example. One important point to notice here is that by calculating the Jacobian of φ, one basically considers the extended continuous map of the embedding spaces $\mathbb{R}^m \to \mathbb{R}^n$ with $m > n$ and therefore the normal update rule does not work anymore. Ben-Israel (1999, 2000) has shown that the update rule for a density $p \colon \varphi(U) \to \mathbb{R}^+$ on the Euclidean space in this case is

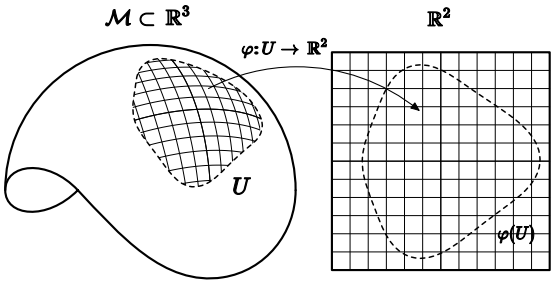


Figure 3: Coordinate chart for $\mathcal{M} \subset \mathbb{R}^3$. The subset U is bijectively mapped to an open subset of $\mathbb{R}^2$. Since U is embedded in $\mathbb{R}^3$, the usual density update rule is not valid.

$$\log p(x) = \log \pi(\varphi^{-1}(x))) + \log \sqrt{\det J_{\varphi^{-1}}^{\top}(x) J_{\varphi^{-1}}(x)} \tag{4}$$

where π is a simple density on $\mathcal{M}$. This update rule is analogous to Equation 1. The **key observation** to make here is that the update rule is defined in terms of the inverse φ^{-1} and that this inverse maps from an open subset $\varphi(U)$ of an Euclidean space. For this open subset, any sufficently small open neighborhood[7] of some point in $\varphi(U)$, is contained in the subset itself. This is essential such that the metric on the image space induced by the Jacobian is correct[8]. Gemici et al. (2016) have used the relationship in Equation 4 to define a general flow on Riemannian manifolds, see more about this in the related work Section 5.

In contrast, if we consider a mapping f between a manifold $\mathcal{M}$ embedded in $\mathbb{R}^m$ and $\mathcal{N}$ embedded in $\mathbb{R}^n$, that is

$$f \colon \mathcal{M} \subset \mathbb{R}^m \to \mathcal{N} \subset \mathbb{R}^n$$

then we are confronted with the following situation: any open neighbourhood of a point $x \in \mathcal{M}$ might contain points y that are *on* the manifold $y \in \mathcal{M}$ but also points that are *off* the manifold, that is $z \in \mathbb{R}^m \setminus \mathcal{M}$. In this case, the update rule in Equation 4 is also not valid anymore. Instead, we have to consider how the spanned volume of the set of tangent vectors at the tangent space $T_x\mathcal{M}$ changes under the transformation f. Rezende et al. (2020) have shown that the spanned volume of the transformed tangent vectors on the image space is $\sqrt{\det(J_f E)^{\top}(J_f E)}$ where E is a matrix with

[6] Image is based on: http://kahrstrom.com/mathematics/illustrations.php

[7] We define an open neighbourhood of a point $x \in U$ in an open subset U as the set $B_r(x) = \{y \in U \,|\, \|x - y\| < r\}$.

[8] The Jacobian is basically the set of derivatives with respect to each basis vector. If we would like to take the derivative at a point x, that is $J_f(x)$, and an infinitesimal line segment in the direction of one basis vector is not contained in an open neighbourhood of x, then the notion of a derivative is not well defined anymore.

the tangent vectors as columns. Therefore, the update rule for a distribution $p(x)$ on $\mathcal{N}$ takes the form

$$\log p(x) = \log \pi(z) - \log \sqrt{\det E(z)^\top J_f(z)^\top J_f(z) E(z)} \tag{5}$$

where $\pi(z)$ is a simple distribution on $\mathcal{M}$. By writing $z = f^{-1}(x)$, the similarity of Equation 5 to the update rule in Equation 1 becomes obvious. Note that in the case of an Euclidean preimage, that is $\mathcal{M} = \mathbb{R}^m$, the tangent space $T_x\mathcal{M}$ is $\mathbb{R}^m$ itself, E becomes the identity and Equation 5 simplifies to Equation 4. In the remainder we will write

$$\det G_f = \det J_f^\top J_f \quad \text{or} \quad \det G_f = \det E^\top J_f^\top J_f E$$

shorthand for the induced metric and infer from the context which of the two is referred to.

2.2 Moebius Flow

In the following, we review the work of Rezende et al. (2020) who used the Moebius transformation to define a flow on the one dimensional circle $\mathbb{S}^1$ and has been previously used by Kato, McCullagh (2015); Wang, Gelfand (2013) to define a distribution on the sphere $\mathbb{S}^D$. In our work, we generalize the results of Rezende et al. (2020) to objects with arbitrary radius r since we will use this for our own method, which is presented in Section 3.1. Throughout the thesis, we will denote a sphere of dimension D and radius r by $r\mathbb{S}^D$.

The Moebius Transformation is a diffeomorphism on the sphere $r\mathbb{S}^D$ (Hertrich-Jeromin, 2003) and is given in terms of Cartesian coordinates for $z \in \mathbb{R}^{D+1}$ by

$$h_\omega(z) = \frac{r^2 - \|\omega\|^2}{\|z - \omega\|^2}(z - \omega) - \omega \tag{6}$$

with inverse

$$h_\omega^{-1}(z) = h_{-\omega}(z)$$

see Appendix A.3 for details of the inverse. However, we will use the Moebius transformation only to define a flow on the circle $r\mathbb{S}^1$ and see later why that is the case. The center parameter $\omega \in \mathbb{R}^2$ lies inside the circle, that is $\|\omega\| < r$, and takes the role of a projection center: a point z is mapped through ω to an antipodal point on the circle $g_\omega(z)$ and subsequently mapped back through the origin. The result of this operation is the Moebius transformation, that is $h_\omega(z) = -g_\omega(z)$, see Figure 4 for an illustration. The unnormalized parameter $\omega \in \mathbb{R}^2$ is initially obtained with a neural net and then normalized via $\omega = \frac{0.99}{1+\|\omega'\|}\omega'$.

Increasing the Expressivity The Moebius transformation h_ω has the following property: For any two ω_1 and ω_2, there exists another ω_3 such that $h_{\omega_1} \circ h_{\omega_2} = h_{\omega_3}$ which means in mathematical terms that the set of Moebius transformations forms a group under function composition, see Kato, McCullagh (2015) theorem 2. Therefore, in order to increase the expressivity of the Moebius transformation, we consider convex combinations of Moebius transformations but convex combinations of Euclidean coordinates on the sphere don't lay on the sphere anymore.

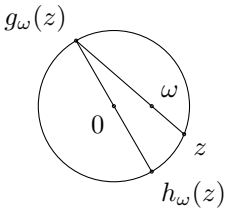


Figure 4: Moebius transformation on the circle $\mathbb{S}^1$, plot is taken from Rezende et al. (2020).

Thus, we concentrate on the circle $r\mathbb{S}^1$ and identify each pair of Euclidean coordinates $(z_1, z_2) \in r\mathbb{S}^1$ with a parametrized angle $\theta \in [0, 2\pi)$, due to the manifold property of $r\mathbb{S}^1$, where 0 and 2π are identified as the same point. This allows us to calculate convex combinations of angles. The bijective mappings between the two representations are given by

$$T_1(\theta, r) = (r\cos\theta, r\sin\theta) = (z_1, z_2)$$
$$T_2(z) = \operatorname{arctan2}(z_2, z_1) \mod 2\pi$$

where the image of arctan2 is usually $[-\pi, \pi]$ such that the modulo operator ensures that T_2 maps to $[0, 2\pi)$. The flow $g\colon [0, 2\pi) \to [0, 2\pi)$ is then given by

$$g(\theta) = (T_2 \circ h_\omega \circ T_1)(\theta)$$

such that convex combinations $\sum_i \rho_i g_i$ with $\sum_i \rho_i = 1$ and $\rho_i > 0$ are valid angles again. As a drawback, the convex combination of analytically invertible functions is not analytically invertible and needs numerical methods like bisection search.

Fixed Point Property for Diffeomorphism There is one important thing to notice. In its current form, g might have a single discontinuity as a function from $[0, 2\pi)$ onto itself, because it is a bijection. For a single Moebius transformation, this is not problematic because the resulting density is then not defined at a point of measure 0 and is therefore still well defined almost everywhere, see Figure 16(b) in the Appendix. However, if we compose functions with discontinuities at different points via convex combination, the resulting function is not bijective anymore and, therefore, not a valid diffeomorphism which could act as flow, see Figure 16(c).

Since it already holds that $\nabla g(\theta) > 0$[9], see Rezende et al. (2020), one sufficient condition is that g leaves the points 0 and 2π invariant. In that case, g is a continuous and monotonic increasing function, see Figure 16(a). Then the convex combination of continuous, monotonic increasing function is a continuous and monotonic increasing as well[10]. Furthermore, a continuous, monotonic increasing function is a invertible (Binmore, 1982) and therefore the convex combination is a valid diffeomorphism, see Figure 16(d). The figures and a further explanation are provided in the Appendix A.3. To ensure the sufficient fixed point condition, we denote the image of the zero angle as $\varphi = g(0)$ and define the final flow as

$$f(\theta) = g(\theta) - \varphi \mod 2\pi$$

such that 0 is a fixed point of f. This flow f can be used as a building block of convex combinations such that $\sum_i \rho_i f_i$ is a valid diffeomorphism, where each of the f_i has its own projection center ω_i. Leaving a point invariant limits the expressivity of the flow. However, we will overcome this limitation by composing the flow with volume preserving rotations, which act as phase translation in the angle space. See Section 2.4 for details concerning Cylindrical flows and Section 3.1 for Circle Slice flows.

Now we can see why we do not use the Moebius transformation to define a flow on $\mathbb{S}^D$ directly. We can not make the flow more expressive via function composition but only via convex combination. However, the notion of monotonicity of the flows f_i, which ensured the invertibility of the convex combination $\sum \rho_i f_i$, does not generalize to higher dimensions. Thus, we can not ensure that the convex combination of Moebius transformations in arbitrary dimensions is a valid diffeomorphism.

Jacobian of Moebius First, we note that the Jacobians of f and g are equal, that is

$$J_f = J_g$$

The Jacobian of g is the product of the Jacobians of the compositioned functions

$$J_g = \frac{\partial T_2}{\partial h}(h) \cdot \frac{\partial h_\omega}{\partial z}(z) \cdot \frac{\partial T_1}{\partial \theta}(\theta)$$

where $z = T_1(\theta)$ and $h = h_\omega(z)$. The individual derivatives are given by

$$\frac{\partial}{\partial z} h_\omega(z) = -\frac{2\left(r^2 - \|w\|^2\right)}{\|z - w\|^4}(z - w)(z - w)^\top + \frac{r^2 - \|\omega\|^2}{\|z - \omega\|^2} I$$

and

$$\frac{\partial T_1}{\partial \theta}(\theta) = r \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} \quad \text{and} \quad \frac{\partial T_2}{\partial h}(h) = \frac{1}{h_1^2 + h_2^2}\begin{bmatrix} -h_2, & h_1 \end{bmatrix}$$

[9] Except at a point where g has a discontinuity. At this point, the derivative is not defined.
[10] https://math.stackexchange.com/questions/1501539/sum-of-monotone-functions

with $\frac{\partial}{\partial\theta}T_1 \in \mathbb{R}^{2\times 1}$, $\frac{\partial}{\partial h}T_2 \in \mathbb{R}^{1\times 2}$ and $\frac{\partial}{\partial z}h_\omega(z) \in \mathbb{R}^{2\times 2}$ such that J_g is a scalar as expected, see Appendix A.3 for details. Since f is real valued, the Jacobian determinant of the convex combinations is simply the convex combination of the Jacobian determinants. Finally, for the Jacobian of the inverse the shift operation has to be undone such that:

$$J_{f^{-1}}(\theta) = J_{g^{-1}}(\theta+\varphi) = \frac{\partial T_2}{\partial h}(h) \cdot \frac{\partial h_{-\omega}}{\partial z}(z) \cdot \frac{\partial T_1}{\partial \theta}(\theta+\varphi)$$

2.3 Neural Spline Flows

Neural Spline flows were initially introduced by Müller et al. (2019), further developed to rational quadratic spline flows by Durkan et al. (2019) and can be used to define a flow on $[-1,1]$ and on the circle in terms of $[0,2\pi]$. In what follows, we strictly review the explanation of Rezende et al. (2020).

Interval Spline Flow (ISF) The idea is to divide the support of the spline flow into K bins defined by a set of $K+1$ knot points $\{x_k, y_k\}_{k=0}^K$ and $K+1$ slopes $\{s_k\}_{k=0}^K$ such that $x_{k-1} < x_k$, $y_{k-1} < y_k$ for all $k \geq 1$ and $s_k > 0$ for all k. Each interval is then transformed via a rational quadratic function

$$f(\theta) = \frac{a_{k2}\theta^2 + a_{k1}\theta + a_{k0}}{b_{k2}\theta^2 + b_{k1}\theta + b_{k0}}$$

where the coefficients are chosen such that f is monotonically increasing and $f(x_k) = y_k$, $\nabla f(\theta)|_{\theta=x_k} = s_k$ for all k. This construction can be done for any closed, real interval and we concentrate here on $[-1,1]$.

Circular Spline Flow (CSF) To adapt ISF to be a valid diffeomorphism on the circle as a function on $[0,2\pi]$, we simply have to set $x_0 = y_0 = 0$, $x_K = y_K = 2\pi$ and set $s_1 = s_K$ because 0 and 2π identify as the same point such that the derivatives have to coincide at the boundaries.

Implementation In the actual implementation we work with the (unnormalized) widths $w_k = x_k - x_{k-1}$ and heights $h_k = y_k - y_{k-1}$ instead of the knot points and normalize them with softmax for width and heights, and softplus functions for derivatives. The implementation is based on[11], and only a few adjustments were necessary to extend ISF to Circular Spline flows. Finally, ISF and CSF leave the boundaries of $[-1,1]$ and $[0,2\pi]$ invariant, which is a limitation of the model in the latter case. Therefore, we compose CSF with rotations, as in the case of the Moebius flow. The analytic Jacobian can be found in Durkan et al. (2019).

2.4 Cylindrical Flows

In this section, we review the model introduced in Rezende et al. (2020) which allows density estimation on a hypersphere. This is done by iteratively transforming $\mathbb{S}^D$ into a cylinder $\mathbb{S}^1 \times [-1,1]^{D-1}$ and we will call these type of flows *Cylindrical Flows* because of the resulting geometry. Subsequently, the circle part $\mathbb{S}^1$ is transformed with the previously introduced Moebius flow (MF), see Section 2.2, or Circular Spline (CSF) and the Euclidean part $[-1,1]^{D-1}$ with Interval Spline Flows (ISF), see Section 2.3, respectively. As already stated in Section 2.2 and 2.3, we compose CSF and MF with rotations to increase their expressivity.

The parameters of the CSF, ISF and MF depend on the implementation of the conditioners, and we used in most of our experiments coupling layers, except for a density fit in the lowest dimensional case of $\mathbb{S}^2 \subset \mathbb{R}^3$ where we implemented the transformation autoregressively. The procedure is illustrated in Figure 5 with an autoregressive flow. After unfolding, first, the radius with an ISF is transformed and, subsequently, the angle θ conditioned on r' with an MF or CSF.

[11] https://github.com/bayesiains/nflows/tree/master/nflows/transforms/splines .

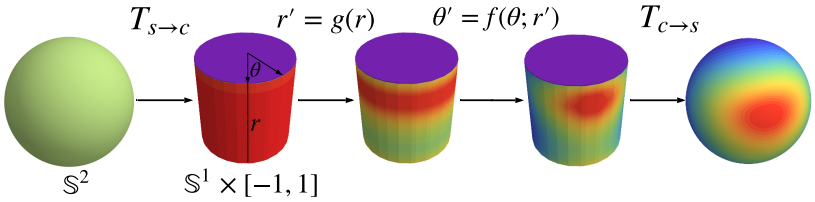


Figure 5: Cylindrical flow acting on $\mathbb{S}^2$ autoregressively. From left to right: (i) the sphere is unfolded, (ii) the radius $r \in [-1, 1]$ is transformed with an Interval Spline flow, (iii) the angle $\theta \in \mathbb{S}^1$ conditioned on the new radius r' is transformed with a Circular Spline or Moebius flow, (iv) the cylinder is folded back to the sphere. Plot is taken from Rezende et al. (2020).

Unfolding the Sphere The unfolding, as described in Rezende et al. (2020), of the D-dimensional sphere $\mathbb{S}^D$ into a cylinder $\mathbb{S}^1 \times [-1, 1]^{D-1}$ is done via iteratively applying the transformation

$$\begin{aligned} U &: \mathbb{S}^D \to \mathbb{S}^{D-1} \times [-1, 1] \\ U &: (x_{1:D+1}) \mapsto \left(\frac{x_{1:D}}{\sqrt{1 - x_{D+1}^2}}, x_{D+1} \right) \end{aligned} \tag{7}$$

on the remaining sphere part of the image. The inverse is straightforwardly available and can be found in Rezende et al. (2020). Repeatedly applying U on the remaining spheres by iterating over all dimensions until the sphere is completely rolled out becomes inefficient in high dimensions. Therefore, we derived the equivalent but vectorized transformation

$$\begin{aligned} T_{s\to c} &: \mathbb{S}^D \to \mathbb{S}^1 \times [-1, 1] \\ T_{s\to c} &: (x_{1:D+1}) \mapsto \left(\frac{x_{1:2}}{\sqrt{1 - \sum_{i=3}^{D+1} x_i^2}}, \ldots, \frac{x_k}{\sqrt{1 - \sum_{i=k+1}^{D+1} x_i^2}}, \ldots, x_{D+1} \right) \end{aligned} \tag{8}$$

which can be proven by applying two steps of U and via induction, see the Appendix A.2. However, this transformation might cause numerical instabilities if the dimensionality is high since the denominator can get close to 0 or even negative, due to the limits of computational precision. We will examine this in more detail with an experiment in Section 4. We were not able to find a closed form for a vectorized version of the inverse.

Jacobian The transformation U is a mapping between manifolds and as such the geometry of the spaces in form of the tangent vectors at the preimage space have to be taken into account by Equation 5 to calculate the density update. Rezende et al. (2020) have shown by a rotation argument that the log volume change inflicted by U has the form

$$\log \sqrt{\det G_U} = \left(\frac{D}{2} - 1 \right) \log \left(1 - x_{D+1}^2 \right) \tag{9}$$

A consequence of Equation 9 is that the log volume change inflicted by the transformation $\mathbb{S}^2 \to \mathbb{S}^1 \times [-1, 1]$ is 0. It is obvious that the volume change of Equation 9 might cause singularities at the boundaries $\{-1, 1\}$. However, Rezende et al. (2020) have shown that if the flow g, which transforms the interval part $[-1, 1]$, has positive derivatives at the boundaries, that is $g'(-1) > 0$ and $g'(1) > 0$, this singularity can theoretically never occur. This can easily be achieved by the ISF, which we will use to transform the Euclidean intervals. Furthermore, let us write shorthand T for $T_{s\to c}$. Then, the volume change inflicted by T is the sum of the volume changes inflicted by Equation 9 for all components and takes the vectorized form

$$\log \sqrt{\det G_T} = \sum_{k=3}^{D+1} \left(\frac{k-1}{2} - 1 \right) \log \left(1 - \frac{x_k^2}{1 - \sum_{i=k+1}^{D+1} x_i^2} \right)$$

This is also how we implemented the volume change.

2.5 Power Spherical Distribution

The Power Spherical Distribution, introduced in De Cao, Aziz (2020), is a distribution on the sphere and can be seen as a more tractable generalization of the von Mises-Fisher (vMF) distribution (Mardia, Jupp, 2009). The latter is usually known as the Gaussian on the hypersphere and has the probability density function

$$p_{\text{vMF}}(x \,|\, \mu, \kappa) = \mathcal{C}_p(\kappa) \exp(\kappa \mu^\top x)$$

where $\mathcal{C}_p(\kappa)$ is a normalization constant, and μ and κ are location and scale parameters. Besides its easy interpretability with two easy to understand parameters, it suffers from the essential drawback to require rejection sampling (Ulrich, 1984; Davidson et al., 2018). This is because sampling of $x \in \mathbb{S}^D$ is based on the tangent normal decomposition (Mardia, Jupp, 2009), whereas one vector stems from a uniform distribution (which is trivial to sample from) and the other vector from a univariate marginal distribution for which no known cumulative inverse exists and therefore it is not straightforward to sample from. Imposing different marginal distributions result in different distributions on the sphere.

As an alternative, De Cao, Aziz (2020) propose a marginal distribution function (df) with known inverse cumulative df and arrive by doing so at a distribution following a power law on the sphere

$$p(x \,|\, \mu, \kappa) = \left(2^{\alpha+\beta}\pi^\beta \frac{\Gamma(\alpha)}{\Gamma(\alpha+\beta)}\right)^{-1} \left(1 + \mu^\top x\right)^\kappa$$

with $\alpha = \frac{D}{2} + \kappa$, $\beta = \frac{D}{2}$, $x \in \mathbb{S}^D$, Γ the gamma function, and μ, κ having the same property of a location and scale parameter as in the vMF distribution. As a result, a reparametrization trick can be applied (Rezende et al., 2014; Kingma, Welling, 2014) and the distribution enables fast sampling since no rejection sampling for the marginal of the tangent-normal composition is required, see De Cao, Aziz (2020) for details. A PyTorch implementation from the original author can be found here[12] and was used in our implementation.

In our experiments we used a 4 component mixture of power spherical distributions with equal weights, that is $p(x) = \frac{1}{4}\sum_i^4 p(x \,|\, \mu_i, \kappa_i)$. Although De Cao, Aziz (2020) provide a closed form of the entropy of a single power spherical distribution, this does not generalize to a mixture of distribution. Therefore, we estimate the entropy of $X \sim \text{MixturePowerSpherical}(\{\mu_i\}_i, \{\kappa_i\}_i)$ stochastically with Monte Carlo estimation in our experiments

$$\hat{H}(X) = -\frac{1}{N}\sum_{i=1}^{N} \log p(x_i) \quad \text{with} \quad x_i \sim p(x)$$

with $N = 10^6$ samples if the dimensionality is lower than 100 and $N = 5 \cdot 10^6$ samples otherwise. Note that, sampling from a mixture distribution with n equal weights is a two stepg procedure: First, we sample uniformly an index $j \sim \mathcal{U}\{1, \dots, n\}$ and secondly, we sample from the j-th distribution $x_i \sim p(x|\mu_j, \kappa_j)$. In order to achieve consistent results for the same parameter set $\{\mu_i\}_i, \{\kappa_i\}_i$, we used the same random seed of 42 both in numpy and PyTorch for the sampling procedure which becomes crucial when comparing the performance across different models and different runs in terms of the KL divergence.

[12] https://github.com/nicola-decao/power_spherical

3 Method

In this section, we present the two main and orthogonal contributions of our thesis: *Circle Slice Flows* and the *Variational Determinant Estimator* (VDE). The former is an alternative method to Cylindrical flows introduced in Section 2.4 by defining a flow directly on the sphere without breaking the geometry and without the need for a numerically unstable unfolding procedure. The latter extends the work of Sohl-Dickstein (2020) by the introduction of a variational distribution modeled by a flow on the hypersphere. The VDE accurately estimates the determinant of a full rank matrix, even with very low sample sizes.

3.1 Circle Slice Flows

The key idea of the first of our two main contributions, namely *Circle Slice Flows*, is to slice the sphere $\mathbb{S}^D$ into lower-dimensional sub spheres $r\mathbb{S}^n$ and, in particular, also one-dimensional circles $r\mathbb{S}^1$ with varying radii. These slices are then transformed either via rotations in the former case or by the flows introduced in Section 2.2 and 2.3, respectively.

The architecture to transform a data sample $(x_1, \dots, x_{D+1}) \in \mathbb{S}^D$ consists of the following modules:

(i) Global rotation.

(ii) Split of the data $(x_1, \dots, x_{D+1})$ into two halves $x_{1:d}$ and $x_{d+1:D+1}$ such that d is an even number and close to $(D+1)/2$.

(iii) Identification of Cartesian pairs with their associated angles, the circle slices.

(iv) Circle slice transformation.

(v) Mapping of circle slices angles to Cartesian coordinates.

(vi) Conditional rotation.

(vii) Concatenation of the transformed vector with the rest,

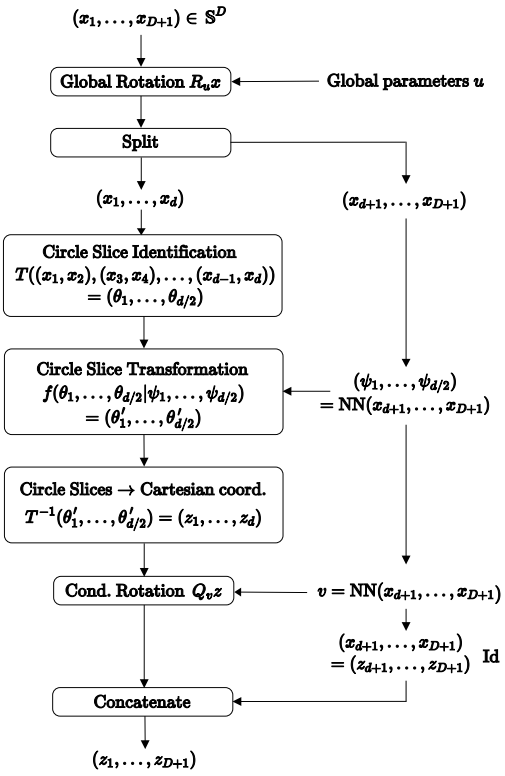


Figure 6: Overview of the architecture of Circle Slice flows with the different modules. Id means here the identity mapping.

which is illustrated in Figure 6. The architecture is a coupling layer where half of the data is transformed conditioned on the other half, see Section 2.1. We start this section with a profound explanation and illustration of the concept of sphere and circle slices, given in the next three paragraphs. This is followed by an explanation of the corresponding modules. Note that we do not do this in a fully linear order, but by starting with module (ii), group the global and conditional rotation within one paragraph for a high-level explanation, and also dedicate an own paragraph to the cond. rotation due to its special role, and indicate the module numbers in the respective paragraphs.

Slices of the Sphere *Circle Slice Flows* use the observation that any subset of n Cartesian coordinates $(x_{i_1}, \dots, x_{i_n}) \subset (x_1, \dots, x_{D+1}) \in \mathbb{S}^D$ of a D dimensional sphere lies itself on a lower dimensional $n-1$ sphere $r\mathbb{S}^{n-1}$ with radius

$$r^2 = x_{i_1}^2 + x_{i_2}^2 + \dots + x_{i_n}^2 = 1 - \sum_{k \neq i_1, i_2, \dots, i_n} x_k^2$$

if the complement set of dimensions $(x_1, \dots, x_{D+1}) \setminus (x_{i_1}, \dots, x_{i_n})$ is fixed. In the case of $n = 2$, we refer to these as *Circle Slices* and to *Sphere Slices* for $n \geq 3$. Note that the subset of Cartesian

coordinates must consist of at least two components such that the concept of slices is well defined for our purpose.

Circle Slices In the case of a slice with intrinsic dimension one, the concept is relatively easy to understand by considering a subset (x_1, x_2) as part of $\mathbb{S}^2$ embedded in $\mathbb{R}^3$ which is visualized in Figure 7. Each pair lies on a circle $r\mathbb{S}^1$ which are indicated by dashed ellipses in the illustration and have an associated radius $r^2 = 1 - x_3^2$. Furthermore, we may identify each pair of Cartesian coordinates with the associated angle

$$\theta = \text{arctan2}(x_2, x_1) \mod 2\pi \in [0, 2\pi)$$

and subsequently transform it with the Moebius (MF) or Circular Spline flows (CSF) reviewed in Section 2.2 and 2.3, respectively, by basically moving probability mass along the circle conditioned on the remaining value x_3.

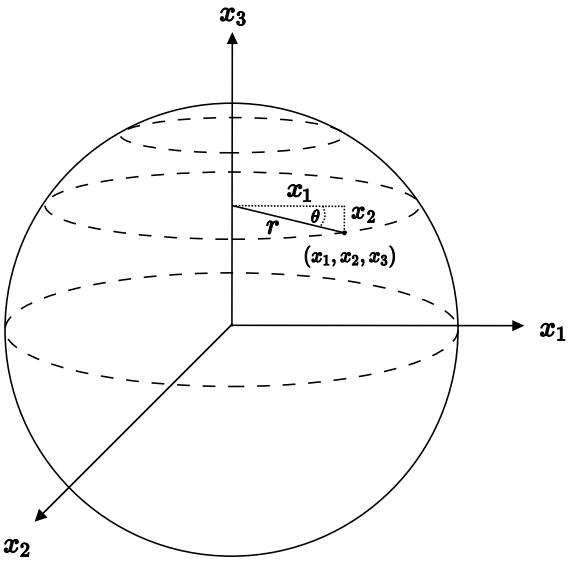


Figure 7: Different $r\mathbb{S}^1$ circle slices (dashed lines) of $\mathbb{S}^2$ (solid sphere) with varying radius. The point (x_1, x_2) lies on a circle with radius $r^2 = 1 - x_3^2$ and has an associated angle θ.

Slices in Higher Dimension The concept of sphere slices also generalizes to the higher dimensional case, but it is not straightforward to visualize this in the head. For didactic reasons, we chose in Figure 8 to visualize a subset of three components (x_1, x_2, x_3) as part of $\mathbb{S}^3$ embedded in $\mathbb{R}^4$. Analogously to the previous case, the radius of the sphere slice depends on the associated value of x_4, but this time the slices are two-dimensional spheres. These spheres are embedded in a four-dimensional Euclidean space and vary in diameter, depending on the value of x_4. Subsequently, these sphere slices are transformed with volume preserving rotations discussed in the next paragraphs. Note that we may also consider Cartesian pairs of two (x_1, x_2) embedded in $\mathbb{R}^4$ and arrive at circles $r\mathbb{S}^1$ where the relationship of the radius and the other coordinates is given by $r^2 = 1 - x_3^2 - x_4^2$. However, this is equivalent to the case considered in Figure 7 but with an additional degree of freedom. By considering slices of different dimensionality and transforming them with either rotations or MF, and CSF, *Circle Slice Flows* can model expressive distributions on $\mathbb{S}^D$.

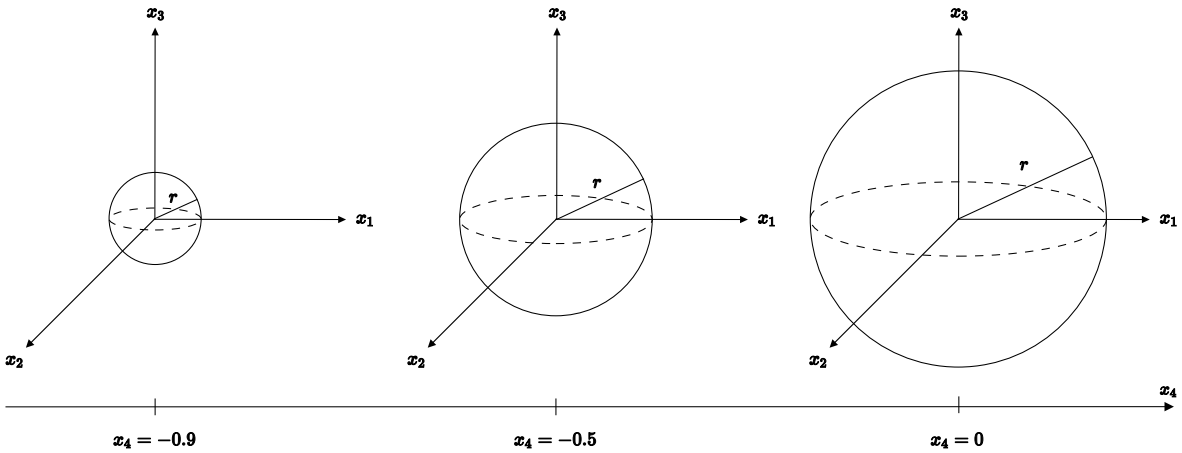


Figure 8: Illustration of $\mathbb{S}^2$ sphere slices through $\mathbb{S}^3$ with varying radius corresponding to different values of x_4. The relationship between the different radii and the fourth dimension is given by $r^2 = 1 - x_4^2$. From left to right: A $\mathbb{S}^2$ sphere with small radius corresponding to a value of $x_4 = -0.9$, a sphere with intermediate radius, and a unit sphere with radius $r = 1$ corresponding to $x_4 = 0$.

(ii): Split of the Data For sake of completeness we would like to state that initially a global rotation of the whole sphere $\mathbb{S}^D$ is performed which is reviewed in the next paragraph. However, we start with module (ii): as usual for a coupling layer, the data is split into two halves. The key point of our concept is that we may interpret this split as a slice S_1 of the sphere $\mathbb{S}^D$ into two separate spheres of not necessarily equal dimension with their own radii by applying the symbolic function

$$\begin{aligned} S_1 \colon \mathbb{S}^D &\to r\mathbb{S}^{d-1} \times r'\mathbb{S}^{k-1} \\ S_1 \colon (x_1, \dots, x_{D+1}) &\mapsto (x_1, \dots, x_d) \times (x_{d+1}, \dots, x_{D+1}) \end{aligned} \tag{10}$$

where $k = D + 1 - d$ is the dimension of the remaining sphere, which only holds as a conditioner for the coupling layer. Note that $r\mathbb{S}^{d-1} \times r'\mathbb{S}^{k-1}$ is actually an abuse of notation[13] and that the radius r of the first sphere slice is only fixed if the other data vector $(x_{d+1:D+1})$ is fixed. However, due to our coupling layer implementation, this is the case and we will use this notation in the remainder of this section.

One Word in Advance The first half is then transformed conditioned on the other half by dividing the first sphere slice $r\mathbb{S}^{d-1}$ further into a number of $d/2$ circle slices $\left\{r_i\mathbb{S}^1\right\}_{i=1}^{d/2}$ followed by an independent, parallelized transformation with a Moebius or Circular Spline flow, and finally, an identification with Cartesian coordinates again. This procedure corresponds to the points (iii) - (v) of the module list and is discussed in detail in the corresponding paragraphs.

Back to module (ii): since the transformation is achieved by considering Cartesian pairs of two, we have to ensure that d is an even number. We choose the breaking point index d as the closest, even number to $(D + 1)/2$ which is

$$d = 2\left\lceil \frac{D+1}{4} \right\rceil$$

where $\lceil \cdot \rceil$ is the ceiling function. The split of the data is only well defined for our purposes if $D \geq 2$.

(i), (vi): Rotation of the Sphere and Sphere Slices As first building block of transformations, we consider global rotations for the whole sphere $\mathbb{S}^D$ and conditional rotations for the previously introduced sphere slice $r\mathbb{S}^{d-1}$. Strictly speaking, our rotations are more general, orthogonal transformations with determinant in $\{-1, 1\}$ which do not preserve the orientation. This is because the Jacobian determinant $\det J_R$ of a an orthogonal operator R equals the determinant of the corresponding orthogonal matrix since

$$\det J_R = \det\left[\frac{\partial}{\partial x} Rx\right] = \det R$$

and therefore they do not inflict a volume change, see the density update Equation 1. However, we will still refer to this class of transformation as rotations.

The global rotation acts as an arbitrary permutation of the dimensions of the data and is inspired by the 1×1 convolution in Kingma, Dhariwal (2018) and the checkerboard and channel-wise masking respectively in Dinh et al. (2016). The same holds for the conditional rotation but more **importantly**: rotating the sphere slice $r\mathbb{S}^{d-1}$, after the univariate Moebius or Circular Spline flows transformed the circle slices $\left\{r_i\mathbb{S}^1\right\}_{i=1}^{d/2}$, acts as a learnable phase translation (mod 2π) in the angle space of each circle slice. Therfore, it overcomes the limitation that 0 and 2π are fixed points of the univariate flows, which, has been remarked in the respective Sections 2.2 and 2.3 of the flows. Furthermore, there is another implication which we will discuss in an own paragraph dedicated to the conditional rotation.

Since it becomes inconvenient to work with rotation angles in arbitrary high dimensions, we express rotations in terms of the product of Householder reflections (Householder, 1958) and utilize the fact that any orthogonal matrix $R \in \mathrm{O}(D + 1)$ can be expressed in terms of the composition of up to

[13] Since the Cartesian product of two circles is a two dimensional torus, that is $\mathbb{S}^1 \times \mathbb{S}^1 = \mathbb{T}^2$, if the data vector is not subject to any boundary conditions.

$D+1$ householder reflections, see Bischof, Sun (1994). In particular, a single Householder reflection is obtained via

$$R_i = I - 2\frac{u_i u_i^\top}{\|u_i\|^2}$$

where u_i are parameter vectors in $\mathbb{R}^{D+1}$, I is the identity matrix, and the parameter set $u = \{u_i\}_{i=1}^{D+1}$ is either learned but fixed in the unconditional case or obtained by a neural net in the conditional case. An orthogonal matrix is then given by

$$R = R_{D+1} \circ \cdots \circ R_1$$

As already stated, rotations are only matrices $R \in \mathrm{SO}(D+1)$ which would be achieved by only compositing an even number of reflections, since $\det R_i = -1$ (Householder, 1958) but since we allow also not orientation-preserving rotations, this does not have to be enforced.

The transformations are then

1. Point (i) of the module list: an initial global rotation of the whole sphere for $x \in \mathbb{S}^D$ by

   $$R_u x = x' \quad \text{with} \quad u \text{ being global parameters}$$

   is performed.

2. Point (vi) of the module list: after the steps (iii) - (v) were applied, a conditional rotation of the first sphere slice for $z \in r\mathbb{S}^{d-1}$ by

   $$Q_v z = z' \quad \text{with} \quad v = \mathrm{NN}\left(x_{d+1:D+1}\right)$$

   is performed. More about this in the conditional rotation paragraph.

For simplicity, we will denote in further sections x' and z' by x and z, respectively, but it should be clear from the context at which point of the whole architecture we are. Here, the matrices R and Q area already the composition of Householder reflections. As orthogonal matrices, the inverses of R and Q are easily obtained by matrix transposition, that is $R^\top x' = x$ and both transformations do not incur any volume change.

Implementation Details for Rotation To avoid unnecessary clutter, we denote only in this paragraph the extrinsic dimension of the Euclidean space by D. We note that the total number of parameters for both rotation modules is roughly $D^2 + D^2/4$ since we need D vectors each of dimension D to obtain $R \in \mathrm{O}(D)$ in the unconditional and approximately $(D/2)^2$ parameters[14] in total for the conditional case. This can be problematic in high dimensions. Therefore, we added the possibility to restrict the number of reflections to 64 such that we trade flexibility of the model against tractability and memory space on the GPU. Empirically, this restriction has not hurt the performance of the model.

Furthermore, one possible, naive implementation would loop over each Householder reflection, construct R_i at each step, and multiply it with the predecessor to obtain R at the end of the loop. However, matrix multiplication becomes expensive in high dimensions and is even with the fastest known algorithm, the Coppersmith–Winograd algorithm, of order $\mathcal{O}(D^{2.37})$ for two $D \times D$ matrices. The previously described approach performs D such matrix multiplications followed by a multiplication with an input vector x and is therefore computationally expensive.

A less expensive implementation makes use of the associative property of matrix multiplication. Let us consider an example in three dimensions:

$$Rx = \left(R_3 R_2 R_1\right) x = \left(R_3 (R_2 (R_1 x))\right)$$

Furthermore, let us denote a unit vector as $w = u/\|u\|$ and check that at each step we have

$$R_i x = \left(I - 2 w_i w_i^\top\right) x = x - 2 w_i \left(w_i^\top x\right) = y$$

[14] a number of $4\left\lceil \frac{D}{4} \right\rceil^2$ parameters to be precise.

By doing so, we reduce the expensive matrix multiplication to a calculation of a dot product $w_i^\top x$ and scalar multiplication since $w_i^\top x \in \mathbb{R}$. In the next step, the same calculation can be applied to $R_{i+1}y$. The inverse is easily obtained by the following observation: a single Householder reflection is symmetric, that is $R_i^\top = R_i$. Then, $R^\top$ is simply given by going through the iteration in reversed order since

$$R^\top = (R_D R_{D-1} \cdot \ldots \cdot R_1)^\top = R_1^\top R_2^\top \cdot \ldots \cdot R_D^\top = R_1 R_2 \cdot \ldots \cdot R_D$$

(iii): Circle Slice Identification After slicing the sphere $\mathbb{S}^D$ in two halves $r\mathbb{S}^{d-1} \times r'\mathbb{S}^{k-1}$ via the mapping from Equation 10, we further slice the first part into a set of circle slices by applying the symbolic map

$$\begin{aligned} S_2 \colon r\mathbb{S}^{d-1} &\to r_1\mathbb{S}^1 \times r_2\mathbb{S}^1 \times \ldots \times r_{d/2}\mathbb{S}^1 \\ S_2 \colon (x_1, x_2, \ldots, x_d) &\mapsto (x_1, x_2), (x_3, x_4), \ldots, (x_{d-1}, x_d) \end{aligned}$$

where r_i are the radii corresponding to the Cartesian pairs (x_j, x_k). This is possible because we chose the breaking point d as an even number. As next step, we may identify independent Cartesian pairs with their associated angles

$$\begin{aligned} T \colon r_1\mathbb{S}^1 \times r_2\mathbb{S}^1 \times \ldots \times r_{d/2}\mathbb{S}^1 &\to [0, 2\pi)^{d/2} \\ T \colon (x_1, x_2), (x_3, x_4), \ldots, (x_{d-1}, x_d) &\mapsto (\theta_1, \ldots, \theta_{d/2}) \end{aligned}$$

Each pair is transformed with the mapping T_2 from Section 2.2, that is

$$\begin{aligned} T_2 \colon r_i\mathbb{S}^1 &\to [0, 2\pi) \\ T_2 \colon (x_j, x_k) &\mapsto \operatorname{arctan2}(x_k, x_j) \mod 2\pi = \theta_l \end{aligned}$$

The function T_2 is basically a coordinate chart, which bijectively maps the circle to an Euclidean space. Therefore, we are confronted with the situation as described in the *Flows on Manifolds* paragraph in Section 2.1 and have to take Equation 4 into account as update rule. Hence, we calculate the Jacobian of the inverse $T_1(r_i, \theta_l) = (r_i \cos\theta_l, r_i \sin\theta_l)^\top$ which is

$$J_{T_1} = \frac{\partial T_1}{\partial \theta} = r_i \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

such that the induced metric is then

$$\sqrt{\det G_{T_1}} = \sqrt{\frac{\partial T_1}{\partial \theta}^\top \frac{\partial T_1}{\partial \theta}} = r_i \tag{11}$$

where r_i is the radius of the circle slice associated to the pair (x_j, x_k). Our transformation of the circle slices leaves the radius invariant and therefore this identification with the coordinate chart does not inflict a volume change since we will transform the angles back to Cartesian pairs and the densities updates cancel out each other. However, we will discuss in Section 6 possible extensions of this architecture where this density update will become important.

(iv): Circle Slice Transformation The angles can subsequently be transformed with the diffeomorphic mappings in form of the Moebius Flow (MF) and Circular Spline Flow (CSF) introduced in Section 2.2 and 2.3, respectively. Since the circle slices are independent and their radii are left invariant by MF and CSF, we may transform all circles independently and parallelized

$$\begin{aligned} f \colon r_1\mathbb{S}^1 \times r_2\mathbb{S}^1 \times \cdots \times r_{d/2}\mathbb{S}^1 &\to r_1\mathbb{S}^1 \times r_2\mathbb{S}^1 \times \cdots \times r_{d/2}\mathbb{S}^1 \\ f \colon \big(\theta_1, \ldots, \theta_{d/2}\big) &\mapsto f(\theta_1, \ldots, \theta_{d/2} \,|\, \psi_1, \ldots, \psi_{d/2}) = \big(f\left(\theta_1|\psi_1\right), \ldots, f(\theta_{d/2}|\psi_{d/2})\big) \end{aligned}$$

where $(\psi_1, \ldots, \psi_{d/2}) = \mathrm{NN}(x_{d+1}, \ldots x_D)$ are the parameters of the flow and obtained by a neural net. Since half of the data is transformed conditioned on the other half, the Jacobian admits a triangular structure, and the log Jacobian determinant is simply the sum of the log Jacobians of the individual transformations.

(v): Cartesian Coordinates Identification As subsequent building block of *Circle Slice Flows*, we reverse the circle slice identification by transforming the angles back to Cartesian coordinates via

$$T^{-1}\colon [0, 2\pi)^{d/2} \to r_1\mathbb{S}^1 \times r_2\mathbb{S}^1 \times \ldots \times r_{d/2}\mathbb{S}^1$$
$$T^{-1}\colon (\theta'_1, \ldots, \theta'_{d/2}) \mapsto (z_1, z_2), (z_3, z_4), \ldots, (z_{d-1}, z_d)$$

which in sum does not inflict a volume change since it cancels out with the term from Equation 11. This is followed by applying the symbolic map

$$S_2^{-1}\colon r_1\mathbb{S}^1 \times r_2\mathbb{S}^1 \times \ldots \times r_{d/2}\mathbb{S}^1 \to r\mathbb{S}^{d-1}$$
$$S_2^{-1}\colon (z_1, z_2), (z_3, z_4), \ldots, (z_{d-1}, z_d) \mapsto (z_1, z_2, \ldots, z_d)$$

which identifies the circle slices with the first sphere slice.

(vi): Conditional Rotation To highlight the implications of the conditional rotation we dedicate an own paragraph to it. The conditional rotation is applied to the first sphere slice, that is

$$Q_v\colon r\mathbb{S}^{d-1} \to r\mathbb{S}^{d-1}$$
$$Q_v\colon (z_1, z_2, \ldots, z_d) \mapsto (z'_1, z'_2, \ldots, z'_d)$$

and acts, as mentioned in the first rotation paragraph, as a phase translation in the angle space of the circle slices to overcome the limitations of the fixed point property of MF and CSF.

However, there is another interesting and important observation to make. Remember, that the map S_2 and S_2^{-1} are symbolic, so we may also write $(z'_1, z'_2), (z'_3, z'_4), \ldots, (z'_{d-1}, z'_d) = (z'_1, z'_2, \ldots, z'_d)$. Thus, the conditional rotation acts also as a transformation of the radii of the circle slices, that is

$$Q_v\colon r_1\mathbb{S}^1 \times r_2\mathbb{S}^1 \times \ldots \times r_{d/2}\mathbb{S}^1 \to r'_1\mathbb{S}^1 \times r'_2\mathbb{S}^1 \times \ldots \times r'_{d/2}\mathbb{S}^1 \tag{12}$$
$$Q_v\colon (z_1, z_2), (z_3, z_4), \ldots, (z_{d-1}, z_d) \mapsto (z'_1, z'_2), (z'_3, z'_4), \ldots, (z'_{d-1}, z'_d)$$

under the boundary condition that the the radius r of the sphere slice $r\mathbb{S}^{d-1}$ remains the same, that is $\sum_i r_i^2 = \sum_i r_i'^2 = r^2$. The new radii r'_i are then determined by the transformed pairs (z'_j, z'_k). In the next paragraph we will denote the transformed vector z' also by z for better readability. Since Q is orthogonal, the whole transformation is volume preserving. We will discuss in in the conclusion Section 6 possible extensions how a similar transformation might be achieved with non-volume preserving transformations.

(vii): Concatenation Finally and for sake of completeness, the part that was taken out of the coupling layer is mapped via the identity

$$\mathrm{Id}\colon (x_{d+1}, \ldots, x_{D+1}) \mapsto (z_{d+1}, \ldots, z_{D+1})$$

and concatenated with the transformed part $(z_{1:d})$ via

$$S_1^{-1}\colon r\mathbb{S}^{d-1} \times r'\mathbb{S}^{k-1} \to \mathbb{S}^D$$
$$S_1^{-1}\colon (z_1, \ldots, z_d) \times (z_{d+1}, \ldots, z_{D+1}) \mapsto (z_1, \ldots, z_{D+1})$$

By following this whole procedure, we obtain an expressively transformed sample on the whole sphere $z \in \mathbb{S}^D$.

3.2 Variational Determinant Estimation

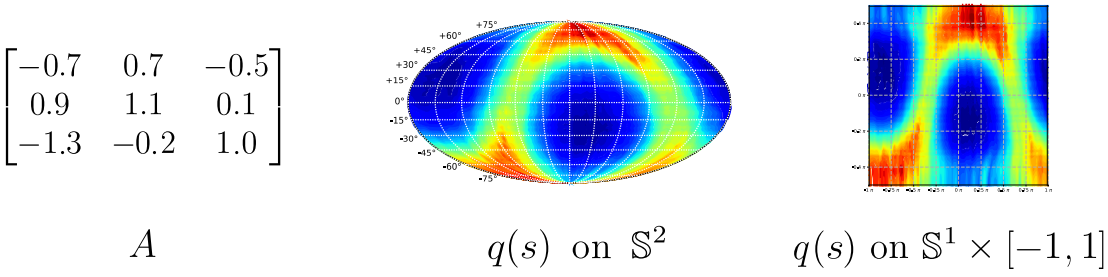


Figure 9: Overview of the variational distribution from the VDE. A variational distribution $q(s)$ on the sphere is trained to estimate the absolute determinant of a matrix 3×3 matrix A.

In this section, we introduce the *Variational Determinant Estimator* (VDE), which is a variational extension of the determinant estimator recently introduced in Sohl-Dickstein (2020). The computation of the (log) absolute determinant of matrices is an omnipresent problem that is encountered in Machine Learning in areas such as spatial models (Aune et al., 2014; Rue, Held, 2005), kernel-based models (Davis et al., 2007; Rasmussen, 2003), normalizing flows (Rezende, Mohamed, 2015; Dinh et al., 2016) and convolutional neural networks (LeCun et al., 1995). The usual approach of calculating the log determinant of a $D \times D$ matrix A involves a Cholesky decomposition, which is of cost $\mathcal{O}(D^3)$ and therefore might be unsuited if D is large.

Naive Stochastic Determinant Estimation This motivates us to explore methods based on stochastic estimation and specifically, Sohl-Dickstein (2020) connects the inverse absolute determinant of a $D \times D$ matrix A with an expectation over matrix-vector products

$$|A|^{-1} = \mathbb{E}_{s \sim \mathcal{U}(\mathbb{S}^{D-1})} \left[\|As\|^{-D} \right] \tag{13}$$

where samples are uniformly drawn from a $D-1$ dimensional hypersphere. For better readability, we will write shorthand $\mathcal{U}(s)$ for the uniform spherical distribution in the following. A common unbiased estimator for Equation 13 is then given via Monte Carlo (MC) integration:

$$\mathbb{E}_{s \sim \mathcal{U}(s)} \left[\|As\|^{-D} \right] \approx \sum_{i=1}^{N} \|As_i\|^{-D} \quad \text{with} \quad s_i \sim \mathcal{U}(s) \tag{14}$$

Sohl-Dickstein (2020) has empirically shown that the naive MC approach of Equation 14 can have problematic high variance, meaning that we need around 10^6 samples to correctly estimate even a 10×10 matrix, see Figure 10.

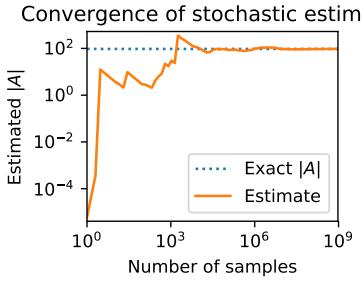


Figure 10: Determinant estimate development with the MC estimator from Equation 14. The matrix A is of dimension 10×10 and randomly sampled with unit Gaussian entries. Plot from Sohl-Dickstein (2020).

Variational Determinant Estimation In this section, we extend the MC determinant estimator by the introduction of a variational distribution $q(s)$ on the hypersphere modeled by *Cylindrical Flows* introduced in Section 2.4 and call this novel method *The Variational Determinant Estimator*. The variational distribution will act as a sample generator and we chose Cylindrical flows to model $q(s)$ over Circle Slice flows since their performance was slightly better at the time of the implementation in terms of the KL divergence and their numerical instabilities mainly arise in higher dimensions. This new estimator achieves lower variance and as a result requires fewer samples for accurate estimates. Then, the *Variational Determinant Estimator* is given by:

$$|A|^{-1} = \mathbb{E}_{s \sim \mathcal{U}(s)} \left[\|As\|^{-D} \right] = \mathbb{E}_{s \sim q(s)} \left[\frac{\mathcal{U}(s)}{q(s)} \|As\|^{-D} \right]$$

which according to Owen (2013) has the least variance when

$$q(s) \propto \mathcal{U}(s)\|As\|^{-D} \tag{15}$$

An example of such a trained variational distribution $q(s)$ in the case of a 3×3 matrix A can be seen in Figure 9 and the optimal variational distribution in the Appendix A.1.

Motivation Here, we would like to give some motivation for going through a training process of a variational distribution $q(s)$ instead of calculating the log determinant directly. Firstly, in Machine Learning, we are often confronted with the situation that only matrix-vector products are available or obtaining the matrix A requires a disproportionately high effort. Thus, the variance estimator provides a convenient way to obtain the determinant in these cases. Secondly, the determinant is a continuous function with respect to the entries of the matrix since the determinant is simply a polynomial of the entries. Therefore, small changes of A with respect to the Euclidean norm result only in small changes of the determinant. Hence, once we obtained an optimal proposal distribution $q(s)$, we might either (i) use the same distribution if the perturbation of the matrix A is sufficiently small or, (ii) only very few new training steps are required to obtain the new optimal distribution for a slightly perturbated matrix $A + \epsilon$. Thus, the VDE might be applied in an online, moving target setting, where only few iterations are required to adjust $q(s)$ for a slightly changed matrix when only matrix-vector products are available. We discuss in Section 6 possible applications.

Objective An example to achieve the desired proportionality, in Equation 15, is to minimize the divergence

$$\mathrm{KL}(q(s)\,;\,\mathcal{U}(s)\|As\|^{-D}/Z)$$

where $\mathcal{U}(s)\|As\|^{-D}$ is treated as an (unnormalized) probability distribution and Z is an unknown normalization constant which does not influence the gradient. To avoid clutter with unnecessary constants, we drop Z in the following and note that the resulting KL divergence is an abuse of notation because the well-known properties such as non-negativity do not necessarily hold anymore for $\mathrm{KL}(q(s)\,;\,\mathcal{U}(s)\|As\|^{-D})$. Using this divergence has the additionally desired effect that:

$$\log|A|^{-1} = \log \mathbb{E}_{s\sim q(s)}\left[\frac{\mathcal{U}(s)}{q(s)}\|As\|^{-D}\right] \geq \mathbb{E}_{s\sim q(s)}\left[\log\frac{\mathcal{U}(s)}{q(s)}\|As\|^{-D}\right] = \mathrm{KL}(q(s)\,;\,\mathcal{U}(s)\|As\|^{-D})$$

and thus the objective $\mathrm{KL}(q(s)\,;\,\mathcal{U}(s)\|As\|^{-D})$ directly gives a lower bound on the log absolute determinant of A^{-1} due to Jensen's inequality. Consequently, this gives a direct method to estimate the log absolute determinant of A using an upper bound which holds with equality when $q(s) \propto \mathcal{U}(s)\|As\|^{-D}$. In this ideal case, the VDE becomes a zero variance estimator, see Goliński et al. (2019).

The proposal distribution q is modeled by a spherical flow f introduced in Section 2.4 or 3.1. However, we chose in our experiment, see Section 4.2, a Cylindrical flow. Since f is a flow on the sphere directly, that is $f\colon \mathbb{S}^{D-1} \to \mathbb{S}^{D-1}$, the update Equation 5 from the *Flows on Manifold* paragraph has to be taken into account. If we substitute q in the objective term KL using the respective change of variables formula and if we choose a uniform base distribution $\pi(s) = \mathcal{U}(s)$, the objective simplifies:

$$\begin{aligned}\mathrm{KL}(q(s)\,;\,\mathcal{U}(s)\|As\|^{-D}) &= \mathbb{E}_{s\sim q(s)}\left[\log q(s) - \log\|As\|^{-D} - \log\mathcal{U}(s)\right] \\ &= \mathbb{E}_{s_0\sim\mathcal{U}(s)}\left[-\log\sqrt{\det G_f(s_0)} + D\log\|Af(s_0)\|\right]\end{aligned} \tag{16}$$

since both uniform densities cancel out each other. Note that Equation 16 also only contains matrix-vector products and does not require to know the matrix A itself. Cylindrical flows, as already explained in Section 2.4, unfold the sphere into a cylinder $\mathbb{S}^D \to \mathbb{S}^1 \times [-1,1]^{D-1}$ and transform the remaining parts with the flows reviewed in Section 2.2 and 2.3. The induced metric $\sqrt{\det G_f}$ from Equation 16 accumulates therefore the volume change inflicted by the folding/unfolding procedure and the respective transformations of the circle and interval parts. Since these volume changes are known analytically, the flow can be trained via gradient based optimization techniques whereas the objective is evaluated via single sample Monte Carlo integration.

4 Results

In this section, we perform four experiments in two orthogonal areas. First, we benchmark the Circle Slice flow introduced in Section 3.1 against the Cylindrical flows introduced in Rezende et al. (2020) and reviewed in Section 2.4 based on a synthetic data set, see Figure 11. Secondly, we quantify the numerical instabilities resulting from the unfolding procedure in Equation 8. Thirdly and fourthly, as an orthogonal experiment, we use Cylindrical flows to model the proposal distribution from Equation 15 and apply the VDE on dense matrices and a structured matrix stemming from a linear convolution operation. We used a uniform base distribution in all experiments, since $\mathbb{S}^D$ is compact.

4.1 Density Estimation with Circle Slice and Cylindrical Flows

In order to compare the performance of the Circle Slice flows against the Cylindrical flows, we constructed a synthetic target density consisting of a four-component mixture of Power Spherical distributions, see Figure 11, with randomly sampled but fixed parameters, see Appendix A.6 for details of the parameters. Furthermore, the unfolding Equation 8 of Cylindrical flows might be problematic and cause numerical instabilities. To quantify this, we counted the number of numerical instabilities when unfolding the sphere and discuss in Section 6 how these problems were tackled.

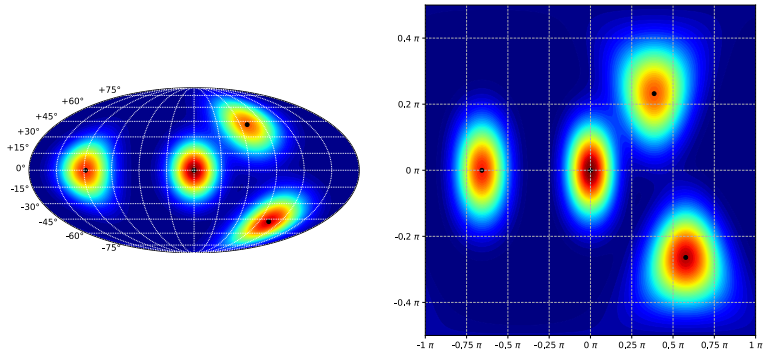


Figure 11: Synthetic target density consisting of a four component mixture of power spherical distributions. Parameters can be found in the Appendix A.6.

Model and Training Parameters Rezende et al. (2020) measured the performance of the Cylindrical flows only up to a dimension of 6. Therefore, we chose in our experiment powers of 2, namely 32, 64, and 128 dimensions. Additionally, we also performed an experiment in dimension 3 to get a feeling for the quality of the density fits and show the model densities with the Mollweide projection in Figure 16.

Model specific parameters corresponding to the different dimensions can be found in Table 2 whereas training specific parameters can be found in Table 3. We used a coupling layer implementation for all models except for the Cylindrical Flows in dimension 3, where we used an autoregressive model. The reason is found in the organic development of the codebase, and a subsequent adjustment would have been a disproportionately high effort. However, in the low dimension of 3, both implementations do not differ too much. The parameter networks for the Moebius Flow, the Spline Flows, and the rotation module were implemented with an MLP consisting of a single hidden layer of dimension $h_{\text{dim}} = 128$. We trained each model for 20 epochs with 400 iterations, and each iteration consists of a batch of 256 samples. Every 100 iterations we evaluated the model on a test set consisting of 100k samples. As optimizer, we used the AdamW optimizer (Loshchilov, Hutter, 2017) with the standard learning rate

Dim	N_C	N_B	N_F	h_{dim} MLP
3	8	8	3	128
32	8	12	3	128
64	8	12	4	128
128	8	12	5	128

Table 2: Model parameters, both for Circle Slice and Cylindrical flows. N_C = number of centers for Moebius flow, N_B = number of bins for Spline flows, N_F = number of stacked flows, and h_{dim} MLP = dimension of hidden layer of parameter MLP.

and weight decay, which is an extension of the Adam optimizer (Kingma, Ba, 2014) with true weight decay.

Learning rate	Weight decay	Batch size	Iters per epoch	Epochs	Iters till eval	Test set size
10^{-3}	10^{-2}	256	400	20	100	100k

Table 3: Training parameters for all models.

Furthermore, we capped the number of Householder reflections for the Circle Slice flow for the experiment in dimension 128 and found no substantial difference in the performance. Then, Durkan et al. (2019) and Rezende et al. (2020) use a relatively high number of bins for the Interval and Circular Spline flows. However, we found that a high number is rather disadvantageous for the performance, so we chose a low number of 8 and 12, respectively. Note that the performance might equalize or even surpass ours if we train the model longer with a lower learning rate.

Results The development of the KL divergence on the test set measured in nats is illustrated in Figure 12 and minimum KL and the KL at the end of the training are summarized in Table 4. It can be seen that for $D = 3$, the performance of the Circle Slice flow with Circular Spline flow (CSF) is on par with the Cylindrical flow with CSF and that both Moebius flows (MF) perform significantly worse. This observation is confirmed by the density fit plots in Figure 16. Inversely, in higher dimensions, the MF perform consistently better than the CSF for the Circle Slice flows. Both Cylindrical flows perform equally well for $D \in \{32, 64, 128\}$, which is not surprising since the difference lies only in the transformation of the first two components, which is neglectable if the dimension is high.

When it comes to the comparison between Circle Slice flows with MF and Cylindrical flows with MF, an inconclusive picture arises. It can be seen that for $D \in \{32, 128\}$, the Cylindrical flows MF perform better than the Circle Slice flows MF. However, for $D = 64$, the performance is on par. To investigate this behavior further, we trained both models in the same configuration for $D \in \{32, 64\}$ on two other four-component densities and illustrate their training development this time in terms of the negative log-likelihood in the Appendix A.7 for a better comparison between different target densities. Here, the same picture arises with a similar performance gap for $D = 32$. Therefore, we suggest for future work to test the performance on different real-world data sets with varying dimensionality. For the additional experiment, we left out $D = 128$ because of computational capacity reasons.

We had the initial intuition that breaking the geometry by Cylindrical flows and Equation 8, especially in higher dimensions, would be rather disadvantageous for the performance. However, this intuition is not confirmed by our experiments since, in none of them, Cylindrical flows perform significantly worse. We also performed several experiments with synthetic data sets where most of the probability mass was set on the poles, but this has not led to a drop in performance of the Cylindrical flows.

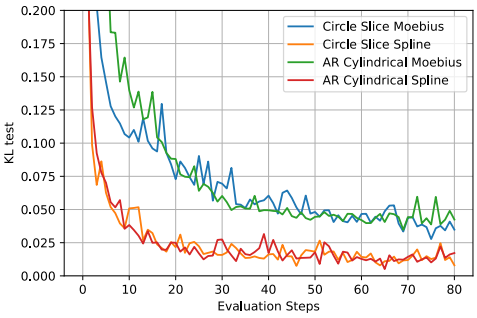
Numerical Instabilities Still, Cylindrical flows suffer from numerical instabilities which is illustrated in Table 5. We count the number of instabilities when unfolding uniform samples on the sphere via Equation 8 for $50k$ data points in terms of three metrics:

(i) The number of NaN's that occur in the first two dimensions. This happens when the argument of the square root operation in Equation 8 becomes negative due to computational precision limits.

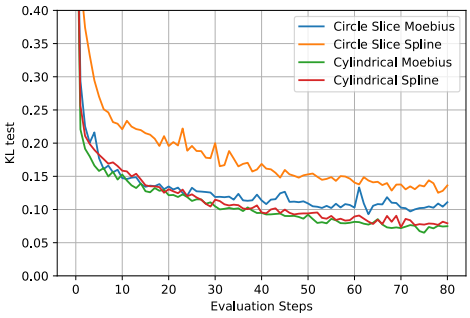
(ii) How often the heights of the cylinder $\mathbb{S}^1 \times [-1, 1]^{D-1}$ are 'unfolded' to a value outside of the interval.

(iii) How often the circle part $(z_1, z_2) \in \mathbb{S}^1$ of the the unfolded cylinder $\mathbb{S}^1 \times [-1, 1]^{D-1}$ are outside of $[-1, 1]$.

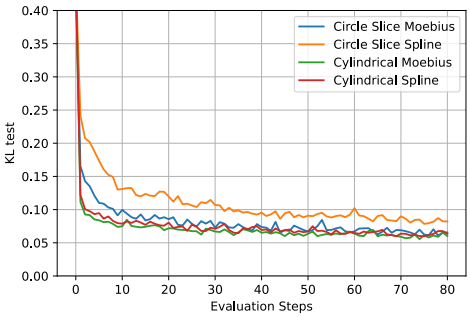
It can be seen that substantial numerical rounding errors occur and specifically that the occurrence of metric (i) is at least superlinear as a function of the dimension. We will discuss in Section 6 how the numerical instabilities were addressed. In contrast, Circle Slice flows do not admit any of these problems and offer an appealing alternative for density estimation on hyperspheres without the need for tedious adjustments.



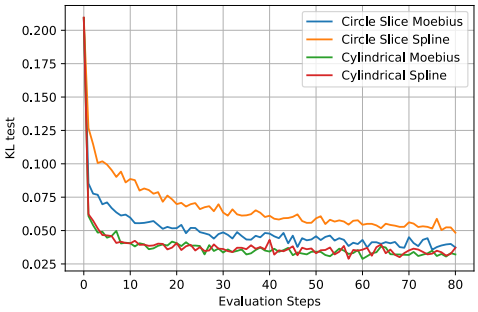
(a) KL in dimension $D = 3$



(b) KL in dimension $D = 32$

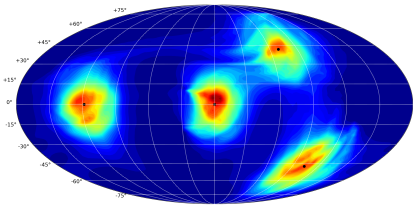


(c) KL in dimension $D = 64$

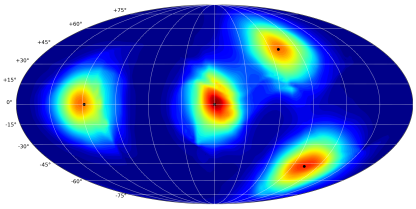


(d) KL in dimension $D = 128$

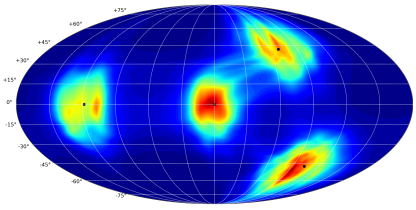
Figure 12: Development of KL divergence in nats evaluated on a 10^5 sample-sized test set over 80 evaluation steps. All models were trained on the same synthetic, four-component, mixture of Power Spherical distributions data set with random but fixed parameters.



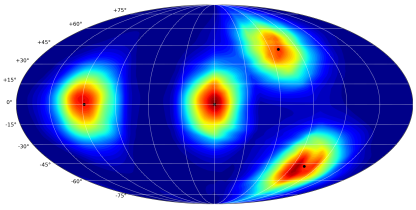
(a) Circle Slice flow with Moebius flow



(b) Circle Slice flow with Sircular Spline flow



(c) Cylindrical flow with Moebius flow



(d) Cylindrical flow with Circular Spline flow

Figure 13: Mollweide projection of the model density in Figure (a) - (d) at the end of the training process after 20 epochs.

Dim	Model	KL [nats] at last iter	Min. KL [nats]	Iter. step of min. KL
3	Circle Slice Moebius (ours)	0.035	0.028	75
	Circle Slice Spline (ours)	0.008	0.008	46
	AR Cylindrical Moebius	0.042	0.035	69
	AR Cylindrical Spline	0.017	**0.005**	65
32	Circle Slice Moebius (ours)	0.111	0.093	63
	Circle Slice Spline (ours)	0.136	0.125	78
	Cylindrical Moebius	0.075	**0.065**	75
	Cylindrical Spline	0.079	0.073	70
64	Circle Slice Moebius (ours)	0.066	0.061	78
	Circle Slice Spline (ours)	0.082	0.078	75
	Cylindrical Moebius	0.06	**0.055**	74
	Cylindrical Spline	0.064	0.06	69
128	Circle Slice Moebius (ours)	0.037	0.036	75
	Circle Slice Spline (ours)	0.048	0.048	80
	Cylindrical Moebius	0.032	0.029	60
	Cylindrical Spline	0.037	**0.029**	57

Table 4: KL divergence at the end of training process and min KL divergence. AR stands for autoregressive implementation, whereas the rest was implemented via coupling layers.

	Cylindrical Flows			Circle Slice Flows
Dim	NaN in $\mathbb{S}^1$	Heights outside $[-1,1]^{D-1}$	$(z_1, z_2) \in \mathbb{S}^1$ outside $[-1,1]^2$	Num. Instability
64	6	15	3k	**0**
128	14	33	6k	**0**
256	41	85	12k	**0**
512	105	196	24k	**0**

Table 5: Occurrences of numerical instabilities when unfolding $\mathbb{S}^D \to \mathbb{S}^1 \times [-1,1]^{D-1}$ by applying Equation 8 for $50k$ uniform samples. Numbers are rounded to full thousands in the last column.

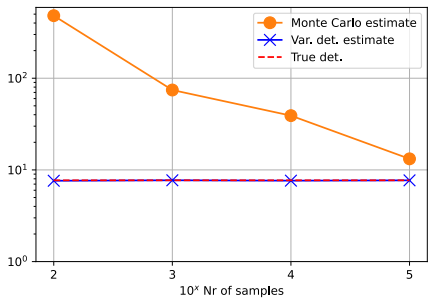
4.2 Variational Determinant Estimation

In this section, we demonstrate the performance of the Variational Determinant Estimator (VDE). We consider two cases: first, we estimate the determinant of randomly sampled 10×10 dense matrices, and secondly, we estimate the determinant of a convolutional layer. For our experiments in this section, we utilized Cylindrical Flows with MF, since at the time of performing the experiment, the architecture of Circle Slice flows was slightly different and our model choice performed best.

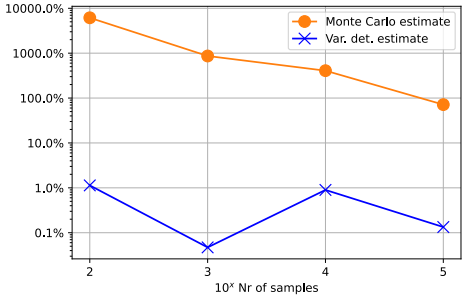
Dense Matrix The determinant is estimated for five 10×10 matrices where the entries are sampled from unit Gaussians, see the Appendix A.4 for the specific matrices. The Cylindrical flow utilizes a Moebius transformation, Section 2.2, for the circle part with $N_C = 12$ number of centers and an Interval Spline flow, Section 2.3, with $N_B = 16$ number of bins for the interval part. Furthermore, we stacked $N_F = 8$ flows on top, used coupling layers, and trained the models for $10k$ iterations with a batch size of 1024.

Nr. of samples	10^2	10^3	10^4	10^5
VDE det. (ours)	**3.4** $\pm$ 2.1 %	**1.7** $\pm$ 0.6 %	**1.6** $\pm$ 1.3 %	**0.3** $\pm$ 0.3 %
MC det.	533 $\pm$ 660 %	348 $\pm$ 262 %	104 $\pm$ 30 %	59 $\pm$ 43 %

Table 6: Comparison of determinant estimates. Results are in mean absolute relative difference for 5 unit Gaussian sampled 10×10 matrices. Deviations are given in one standard deviation.



(a) Absolute determinant estimates.



(b) Relative variation of determinant esitmates

Figure 14: Determinant estimates of a structured 16×16 matrix. Left: absolute determinant estimated values. Right: the absolute relative difference to the true determinant value. Both plots are on log-scale.

Nr. of samples	10^2	10^3	10^4	10^5	true (log) determinant
VDE det. (ours)	7.62	7.71	7.64	7.70	7.71
MC det.	481.09	74.37	39.08	13.21	
VDE log det. (ours)	2.03	2.04	2.03	2.04	2.04
MC log det.	6.18	4.31	3.67	2.58	
VDE Rel. diff of det. (ours)	**1.1** %	**0.05** %	**0.9** %	**0.1** %	0 %
MC Rel. diff of det.	6144 %	865 %	407 %	71 %	

Table 7: Comparison of variational and Monte Carlo determinant estimates of a structured 16×16 matrix. First four rows show results in absolute numbers and log-space. The last two rows illustrate the relative difference of the estimates to the true determinant.

The results can be seen in Table 6 where we present the mean of the relative absolute differences of the estimated determinant in comparison to the true absolute determinant. The variational determinant estimator achieves even for a low sample size of 10^2 low relative differences whereas in contrast the naive Monte Carlo estimate still has high variance throughout all sample sizes.

Convolutional Layer In this experiment the (log) determinant of a convolutional layer is estimated. The reason for this experiment is that these types of sparse linear transformations often occur in deep learning, and they typically have cheap matrix-vector products. We manually reconstructed the 16×16 equivalent matrix W of a convolution of a 3×3 filter with an 4×4 image, see Appendix A.5 for the filter. The determinant of W equals the determinant of the convolution operation. We chose the same architecture as in our previous experiment but run the experiment this time for $40k$ iterations with the same batch size. Note that the parametrization of our Cylindrical flow is fully connected, and better optimization behaviour is expected when the parametrization would be convolutional. Figure 14 and Table 7 show the results of the experiment in terms of absolute and log abs. determinant estimates. We observe the same behaviour as in the previous experiment: The VDE achieves low errors with already low sample sizes whereas the MC determinant estimator is not able to capture the determinant correctly with even high sample sizes.

5 Related Work

In this section, we discuss related work concerning the Variational Determinant Estimator and flows that are able to capture densities on spaces with non-flat geometry.

5.1 Variational Inference and Determinant Estimation

Variational Inference & Importance Sampling Importance sampling is one of the underlying concepts to reduce the variance with the VDE and has a long history as a study object. Hesterberg (1988) introduced extensions for the importance weights such as regression or non-linear exponential estimates to allow the method to be effectively applied in a wider range of settings such as multivariate outputs. Kingma, Welling (2014); Rezende et al. (2014) have introduced deep learning based variational inference and Burda et al. (2015) have shown tighter bounds with importance-weighted variational inference. Although these works were originally aimed at estimating log probabilities, they can be more generally applied to marginalize a probabilistic latent variable. Müller et al. (2019) utilize flows to learn a proposal distribution for importance sampling in a Euclidean space. Goliński et al. (2019) introduce amortized Monte Carlo integration, which combines different proposal distributions for better estimates.

Determinant Estimation Furthermore, log determinant estimation is an omnipresent problem in Machine Learning and the standard approach uses a Cholesky decomposition (Golub, Van Loan, 1996), which is of order $\mathcal{O}(D^3)$ for a $D \times D$ matrix and becomes infeasible for large scale matrices. Many common stochastic methods rely on the identity $\log\det A = \mathrm{Tr}\,(\log A)$[15] and a stochastic trace estimator. Fitzsimons et al. (2017); Ubaru et al. (2017) examine different types of the latter, such as mutually unbiased bases estimator and Lanczos quadrature, their asymptotic behaviour and error bounds.

Another common type of such estimator is Hutchinson's trace estimator (Hutchinson, 1989) and Han et al. (2015) use this type in combination with Chebyshev polynomial expansion (Mason, Handscomb, 2002) of the power series expression of the matrix logarithm and rely only on efficient matrix-vector products. Furthermore, their algorithm offers linear time complexity with respect to the number of non-zero entries of the matrix and admits additive and multiplicative error bounds for the approximation of the log-determinant. A similar approach with a Taylor expansion instead of Chebyshev polynomials is done by Zhang, Leithead (2007); Boutsidis et al. (2017).

Finally, Granziol et al. (2018) recast the log determinant calculation as a variational inference problem. They do this by relating the log determinant to an integral with respect to the spectral density[16] of the matrix. Subsequently, a variational posterior as an approximation of the true spectral density is introduced and optimized in terms of the KL divergence.

5.2 Normalizing Flows on Manifolds

Normalizing Flows (Tabak, Turner, 2013; Rezende, Mohamed, 2015) are attractive generative models to learn distributions because they admit exact likelihood evaluation, and they are fast to sample from as opposed to purely autoregressive models. There have been many advances for (ordinary) flows on Euclidean spaces (Dinh et al., 2016; Kingma et al., 2016; Chen et al., 2019; Perugachi-Diaz et al., 2020). Extensions in a different direction were made by Chen et al. (2018) who observed that stacking multiple flows can be interpreted as an Euler discretization of a continuous-time transformation. Based on that, they define *continuous normalizing flows* where (in the limit) uncountably many flows are stacked instead of a discrete number and whose dynamics are determined by an ordinary differential equation (ODE). As a welcome consequence, the continuous change of variable formula only requires the trace and not the determinant of the Jacobian, which is computationally less expensive. Grathwohl et al. (2018) develop this further by estimating the trace with Hutchinson's trace estimator.

[15] Where $\log A$ is the matrix logarithm, which has the following power series expansion $\log A = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{(A-I)^k}{k}$.

[16] For a non-singular matrix $n \times n$ matrix A, this is given by $\log\det A = n \int_0^1 p(\lambda) \log \lambda d\lambda$, where $p(\lambda)$ is the spectral density of the operator A.

Flows on Manifolds Recently, Gemici et al. (2016); Falorsi et al. (2019); Rezende et al. (2020); Brehmer, Cranmer (2020); Bose et al. (2020); Mathieu, Nickel (2020) made advances in defining expressive normalizing flows on manifolds by following different approaches. Gemici et al. (2016) project a base distribution on a manifold $\mathcal{N}$ first to a Euclidean space via its coordinate chart, enrich it with well established normalizing flows and then map it back onto $\mathcal{N}$. However, this approach might cause singularities if the whole space $\mathcal{N}$ is not diffeomorphic to $\mathbb{R}^n$, which is for many manifolds of interest the case.

The work of Falorsi et al. (2019) enables to learn reparameterizable densities on Lie groups, which allow them to be applied in a VAE setting to establish low variance gradient estimators. Lie groups are algebraic groups with a manifold structure, often occur in physics, and well-known representatives are the Lorentz group, the Poincaré group, and SU(2). The idea is to define a flow on the *Lie Algebra*, which is the tangent space at the identity of the Lie group and isomorphic to $\mathbb{R}^n$. Therefore, well-established flows on Euclidean spaces can be applied, and the density is subsequently mapped onto the group via the exponential map (Kobayashi, Nomizu, 1963). However, in general, the exponential map is not bijective and, therefore, problematic to compose with other flows, and its Jacobian does not inhibit a triangular structure.

Rezende et al. (2020) contributions to the field of learning densities on manifolds are restricted to the case of spheres $\mathbb{S}^D$ and tori $\mathbb{T}^D$, but they propose several distinct approaches. The first is to unfold the sphere to a cylinder $\mathbb{S}^D \to \mathbb{S}^1 \times [-1, 1]^{D-1}$ and apply circle flows, and Euclidean flows on the circle part and interval part respectively, which is the method described in Section 2.4. Next to their contribution of the Moebius flow and Circular Spline flow, presented in Section 2.2 and 2.3, they pick up the idea of Gemici et al. (2016) and map $\mathbb{S}^1$ (almost) bijectively to $\mathbb{R}$, apply a Euclidean flow on $\mathbb{R}$ and map it back onto $\mathbb{S}^1$. They show that the gradient of this transformation is well defined everywhere, even though the projection itself is not. Finally, they combine ideas of Falorsi et al. (2019) and Sei (2013) and define a diffeomorphism on $\mathbb{S}^D$ as the exponential map of a gradient field similar to the one defined in Sei (2013). However, this approach suffers from the inefficient Jacobian determinant calculation as previously described and is thus unsuited for higher dimensions.

Brehmer, Cranmer (2020) introduced manifold-modeling flows (MFMFs). They do this by combining the manifold learning aspects of GANs with the tractable density estimation of flows on manifolds and claim to simultaneously learn the shape of an unknown manifold and the density populating this manifold. The key idea is based on Gemici et al. (2016), and initially, they assume that the manifold is of known dimensionality and can be described with a single coordinate chart, which is learned with a flow while later possible relaxations are discussed.

Mathieu, Nickel (2020) combine the idea of continuous normalizing flows specified by an ODE with the aspect of density estimation on Riemannian manifolds and call the resulting model *Riemaniannian continuous normalizing flows*. In their work, they focus on constant curvature manifolds such as the hypersphere and the hyperbolic Poincaré ball and do this by a manifold 'aware' numerical solver for the ODE, which describes the dynamics of the flow. Finally, Bose et al. (2020) extend the work of Dinh et al. (2016) to hyperbolic spaces.

As a result, it is now also possible to learn expressive distributions on specific manifolds like hyperspheres or hyperbolic spaces with exact likelihood estimates and efficient sampling, but there is still much work to be done for manifolds that e.g. occur in theoretical physics such as $\mathrm{U}(1), \mathrm{SU}(2)$, SU(3) or in cases where the manifold is not known upfront.

6 Conclusion

In this work, we presented *Circle Slice flows* and the *Variational Determinant Estimator*, two orthogonal contributions to the field of density estimation on hyperspheres. In the following, we will review our experience with both concepts and give suggestions for future work.

Circle Slice Flows divide a data set on the hypersphere into pairs of independent circle slices which lie on one-dimensional circles $r\mathbb{S}^1$ with some radius r and transform the associated angles. They do this without changing the geometry of the underlying space and without the occurrence of numerical instabilities. In our experiments, we found that our method is on par at the dimension of 3 and 64 and got slightly outperformed by our benchmark line in form the Cylindrical flows in the other dimensions.

However, this performance was by no means achieved out of the box, and we would like to highlight that the implementation of the Cylindrical Flows was troublesome and that, in particular, the unfolding procedure needed extra care implementation wise. First of all, the iterative unfolding procedure in Equation 7 is suboptimal for high dimensional data since it requires to loop over each of the individual dimensions and to perform a costly division and square root operation, eventually slowing down the density estimation and sampling process. This can be overcome by applying the vectorized unfolding transformation in Equation 8. Nonetheless, after inspecting the denominator in form of $\sqrt{1 - \sum_{i \geq 3} x_i^2}$ this opens new problems. It is clear that for a high number of dimensions, the argument of the normalizing term becomes small and, in some cases, indistinguishable from 0 or even negative due to the limits of computational precision.

Furthermore, by dividing the components of the sphere by the normalizer to arrive at a component that populates the whole interval $[-1, 1]$, we often observed that the resulting value lies outside of the interval $[-1, 1]$ and therefore could not be processed anymore by the following Interval Spline transformation. We tackled these problems by first clipping all values that got mapped outside of $[-1, 1]$ to $[-1 + \epsilon, 1 - \epsilon]$ with a straight-through gradient. Otherwise, the clipping operation zeros the gradient, which makes the data sample redundant. Secondly, we replaced all occurring NaN values with a small value obtained by empirical observations also with a straight-through gradient. However, clipping and replacing values opens new problems since then the squares of the components do not necessarily sum up to one anymore.

Finally, all these problems apparently did not significantly hurt the performance, and we were able to stabilize the unfolding procedure. Since the number of numerical instabilities rises over proportionately with the number of dimensions as it can be seen in Table 5, we still consider Cylindrical flows as not optimal for higher-dimensional problems but were lacking the computational capacity to quantify this.

Therefore, to increase the expressivity of the Circle Slice flows, we suggest the following: We have already seen in Equation 12 that the conditional rotation might be interpreted as a volume-preserving transformation of the radii of the circle slices. As an extension, we suggest this in a non-volume preserving way. By taking $\mathbb{S}^2$ as an illustrative example and recalling Figure 7, this would mean that the circle slices are moved along the x_3 axis, and as a result, the density update of Equation 11 has to be taken into account. We propose to do this sequentially by first transforming the angles of the circle slices in parallel and subsequently transforming the radii by obeying the boundary conditions such that the overall radius remains identical. Another application, besides density estimation, would be to benchmark the performance of the Circle Slices flows in a hyperspherical VAE (Davidson et al., 2018) to model the variational distribution which has the sphere as support. Code for this purpose as a starting point can be found in the git repository of the author.

Variational Determinant Estimator On the other hand, the introduced VDE achieves with low sample sizes high accuracy in estimating a determinant of a linear operator. Interestingly, the estimator in its original variant and the VDE allow estimation if only matrix-vector products are available. In our experiments, we considered an offline setting where a large number of samples are required to first optimize the model. However, in future work, the VDE could also be applied in an

online, moving target settings. In this case, small updates to the matrix A would only require small updates to the density model $q(s)$.

A perpendicular direction for VDE would be to estimate the Jacobian determinant of a function f. The Jacobian would depend on the input x and the density model $q(s|x)$ can be amortized. Additionally, not the entire Jacobian but only Jacobian-vector products would be required to estimate the determinant. An example of such a Jacobian can be found in the exponential map flow setting, which we mentioned in the related work section, developed by Falorsi et al. (2019) and further developed by Rezende et al. (2020) to define a flow on $\mathbb{S}^D$ directly. The Jacobian does not admit a functional form that allows an efficient Jacobian determinant calculation and is therefore suited for the problem of determinant estimation.

A direct related application might be found in the recently published self normalizing flow framework (Keller et al., 2020), which allows efficient density updates without functional constraints on the network architecture such as a coupling layer. Here, two separate flows f and g are trained, each with its own parameter set where the objective consists of the likelihoods with respect to both flows and a reconstruction constraint, which ensures that both flows are approximate inverses of each other, that is $f \approx g^{-1}$. By modeling both directions simultaneously, only the 'forward' directions of either one of the flows during training are required for the evaluation of the density and the gradient update. Therefore, the framework avoids the need to know the inverse of the log Jacobian determinant, which might not have a closed-form in the case of unconstrained networks. However, with the Variational Determinant Estimator, we are still able to evaluate the density fit quality of the flows separately via estimating the respective inverse Jacobian determinants by only relying on matrix-vector products, which are given by the self-normalizing flow framework for free. The VDE could then be trained in parallel to the self normalizing flow. It would be interesting to see how the VDE performs in such a high dimensional setting.

Conclusively, we hope that both of our contributions, *Circle Slice Flows* and the *Variational Determinant Estimator*, will enable new and exciting research directions. The codebase to reproduce the results and enabling further work can be found in the git repository[17] of the author.

[17] https://github.com/P4ppenheimer

References

Aune Erlend, Simpson Daniel P, Eidsvik Jo. Parameter estimation in high dimensional Gaussian distributions // Statistics and Computing. 2014. 24, 2. 247–263.

Ben-Israel Adi. The change-of-variables formula using matrix volume // SIAM Journal on Matrix Analysis and Applications. 1999. 21, 1. 300–312.

Ben-Israel Adi. An application of the matrix volume in probability // Linear Algebra and its Applications. 2000. 321, 1-3. 9–25.

Binmore Kenneth George. Mathematical Analysis: a straightforward approach. 1982.

Bischof Christian H, Sun Xiaobai. On orthogonal block elimination // Preprint MCS-P450-0794, Mathematics and Computer Science Division, Argonne National Laboratory. 1994.

Bose Avishek Joey, Smofsky Ariella, Liao Renjie, Panangaden Prakash, Hamilton William L. Latent Variable Modelling with Hyperbolic Normalizing Flows // arXiv preprint arXiv:2002.06336. 2020.

Boutsidis Christos, Drineas Petros, Kambadur Prabhanjan, Kontopoulou Eugenia-Maria, Zouzias Anastasios. A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix // Linear Algebra and its Applications. 2017. 533. 95–117.

Brehmer Johann, Cranmer Kyle. Flows for simultaneous manifold learning and density estimation // arXiv preprint arXiv:2003.13913. 2020.

Burda Yuri, Grosse Roger, Salakhutdinov Ruslan. Importance weighted autoencoders // arXiv preprint arXiv:1509.00519. 2015.

Chen Ricky TQ, Rubanova Yulia, Bettencourt Jesse, Duvenaud David K. Neural ordinary differential equations // Advances in neural information processing systems. 2018. 6571–6583.

Chen Tian Qi, Behrmann Jens, Duvenaud David, Jacobsen Jörn-Henrik. Residual Flows for Invertible Generative Modeling // Neural Information Processing Systems, NeurIPS. 2019. 9913–9923.

Cortes Corinna, Vapnik Vladimir. Support-vector networks // Machine learning. 1995. 20, 3. 273–297.

Davidson Tim R, Falorsi Luca, De Cao Nicola, Kipf Thomas, Tomczak Jakub M. Hyperspherical variational auto-encoders // arXiv preprint arXiv:1804.00891. 2018.

Davis Jason V, Kulis Brian, Jain Prateek, Sra Suvrit, Dhillon Inderjit S. Information-theoretic metric learning // Proceedings of the 24th international conference on Machine learning. 2007. 209–216.

De Cao Nicola, Aziz Wilker. The Power Spherical distribution // arXiv preprint arXiv:2006.04437. 2020.

Dinh Laurent, Krueger David, Bengio Yoshua. Nice: Non-linear independent components estimation // arXiv preprint arXiv:1410.8516. 2014.

Dinh Laurent, Sohl-Dickstein Jascha, Bengio Samy. Density estimation using real nvp // arXiv preprint arXiv:1605.08803. 2016.

Durkan Conor, Bekasov Artur, Murray Iain, Papamakarios George. Neural spline flows // Advances in Neural Information Processing Systems. 2019. 7511–7522.

Falorsi Luca, Haan Pim de, Davidson Tim R, Forré Patrick. Reparameterizing distributions on Lie groups // arXiv preprint arXiv:1903.02958. 2019.

Fefferman Charles, Mitter Sanjoy, Narayanan Hariharan. Testing the manifold hypothesis // Journal of the American Mathematical Society. 2016. 29, 4. 983–1049.

Fitzsimons Jack, Cutajar Kurt, Osborne Michael, Roberts Stephen, Filippone Maurizio. Bayesian inference of log determinants // arXiv preprint arXiv:1704.01445. 2017.

Gemici Mevlana C., Rezende Danilo Jimenez, Mohamed Shakir. Normalizing Flows on Riemannian Manifolds // CoRR. 2016. abs/1611.02304.

Goliński Adam, Wood Frank, Rainforth Tom. Amortized Monte Carlo Integration // arXiv preprint arXiv:1907.08082. 2019.

Golub Gene H, Van Loan Charles F. Matrix Computations Johns Hopkins University Press // Baltimore and London. 1996.

Goodfellow Ian, Pouget-Abadie Jean, Mirza Mehdi, Xu Bing, Warde-Farley David, Ozair Sherjil, Courville Aaron, Bengio Yoshua. Generative adversarial nets // Advances in neural information processing systems. 2014. 2672–2680.

Granziol Diego, Wagstaff Edward, Ru Bin Xin, Osborne Michael, Roberts Stephen. VBALD-Variational Bayesian approximation of log determinants // arXiv preprint arXiv:1802.08054. 2018.

Grathwohl Will, Chen Ricky TQ, Bettencourt Jesse, Sutskever Ilya, Duvenaud David. Ffjord: Free-form continuous dynamics for scalable reversible generative models // arXiv preprint arXiv:1810.01367. 2018.

Han Insu, Malioutov Dmitry, Shin Jinwoo. Large-scale log-determinant computation through stochastic Chebyshev expansions // International Conference on Machine Learning. 2015. 908–917.

Hertrich-Jeromin Udo. Introduction to Möbius differential geometry. 300. 2003.

Hesterberg Timothy Classen. Advances in importance sampling. 1988.

Ho Jonathan, Chen Xi, Srinivas Aravind, Duan Yan, Abbeel Pieter. Flow++: Improving flow-based generative models with variational dequantization and architecture design // arXiv preprint arXiv:1902.00275. 2019.

Hoogeboom Emiel, Peters Jorn, Berg Rianne van den, Welling Max. Integer discrete flows and lossless compression // Advances in Neural Information Processing Systems. 2019. 12134–12144.

Householder Alston S. Unitary triangularization of a nonsymmetric matrix // Journal of the ACM (JACM). 1958. 5, 4. 339–342.

Huang Chin-Wei, Krueger David, Lacoste Alexandre, Courville Aaron. Neural autoregressive flows // arXiv preprint arXiv:1804.00779. 2018.

Hutchinson Michael F. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines // Communications in Statistics-Simulation and Computation. 1989. 18, 3. 1059–1076.

Kanwar Gurtej, Albergo Michael S, Boyda Denis, Cranmer Kyle, Hackett Daniel C, Racanière Sébastien, Rezende Danilo Jimenez, Shanahan Phiala E. Equivariant flow-based sampling for lattice gauge theory // arXiv preprint arXiv:2003.06413. 2020.

Kato Shogo, McCullagh Peter. Moeobius transformation and a Cauchy family on the sphere // arXiv preprint arXiv:1510.07679. 2015.

Keller T Anderson, Peters Jorn WT, Jaini Priyank, Hoogeboom Emiel, Forré Patrick, Welling Max. Self Normalizing Flows // arXiv preprint arXiv:2011.07248. 2020.

Kim Sungwon, Lee Sang-gil, Song Jongyoon, Kim Jaehyeon, Yoon Sungroh. FloWaveNet: A generative flow for raw audio // arXiv preprint arXiv:1811.02155. 2018.

Kingma Diederik P, Ba Jimmy. Adam: A method for stochastic optimization // arXiv preprint arXiv:1412.6980. 2014.

Kingma Diederik P, Salimans Tim, Jozefowicz Rafal, Chen Xi, Sutskever Ilya, Welling Max. Improved variational inference with inverse autoregressive flow // Advances in Neural Information Processing Systems. 2016. 4743–4751.

Kingma Diederik P., Welling Max. Auto-Encoding Variational Bayes // 2nd International Conference on Learning Representations, ICLR. 2014.

Kingma Durk P, Dhariwal Prafulla. Glow: Generative flow with invertible 1x1 convolutions // Advances in neural information processing systems. 2018. 10215–10224.

Kobayashi Shoshichi, Nomizu Katsumi. Foundations of differential geometry. 1, 2. 1963.

Kobyzev Ivan, Prince Simon, Brubaker Marcus. Normalizing flows: An introduction and review of current methods // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020.

Köhler Jonas, Klein Leon, Noé Frank. Equivariant flows: sampling configurations for multi-body systems with symmetric energies // arXiv preprint arXiv:1910.00753. 2019.

LeCun Yann, Bengio Yoshua, others . Convolutional networks for images, speech, and time series // The handbook of brain theory and neural networks. 1995. 3361, 10. 1995.

LeCun Yann, Haffner Patrick, Bottou Léon, Bengio Yoshua. Object recognition with gradient-based learning // Shape, contour and grouping in computer vision. 1999. 319–345.

Ledig Christian, Theis Lucas, Huszár Ferenc, Caballero Jose, Cunningham Andrew, Acosta Alejandro, Aitken Andrew, Tejani Alykhan, Totz Johannes, Wang Zehan, others . Photo-realistic single image super-resolution using a generative adversarial network // Proceedings of the IEEE conference on computer vision and pattern recognition. 2017. 4681–4690.

Loshchilov Ilya, Hutter Frank. Decoupled weight decay regularization // arXiv preprint arXiv:1711.05101. 2017.

Mardia Kanti V, Jupp Peter E. Directional statistics. 494. 2009.

Mason John C, Handscomb David C. Chebyshev polynomials. 2002.

Mathieu Emile, Nickel Maximilian. Riemannian continuous normalizing flows // Advances in Neural Information Processing Systems. 2020. 33.

Mazoure Bogdan, Doan Thang, Durand Audrey, Pineau Joelle, Hjelm R Devon. Leveraging exploration in off-policy algorithms via normalizing flows // Conference on Robot Learning. 2020. 430–444.

Milnor John, Weaver David W. Topology from the differentiable viewpoint. 1997.

Müller Thomas, McWilliams Brian, Rousselle Fabrice, Gross Markus, Novák Jan. Neural importance sampling // ACM Transactions on Graphics (TOG). 2019. 38, 5. 1–19.

Multilayer perceptron neural network (MLPs) for analyzing the properties of Jordan Oil Shale 1. // . 2008.

Ng Andrew, Jordan Michael. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes // Advances in neural information processing systems. 2001. 14. 841–848.

Oord Aaron van den, Dieleman Sander, Zen Heiga, Simonyan Karen, Vinyals Oriol, Graves Alex, Kalchbrenner Nal, Senior Andrew, Kavukcuoglu Koray. Wavenet: A generative model for raw audio // arXiv preprint arXiv:1609.03499. 2016a.

Oord Aaron van den, Kalchbrenner Nal, Kavukcuoglu Koray. Pixel recurrent neural networks // arXiv preprint arXiv:1601.06759. 2016b.

Owen Art B. Monte Carlo Theory // Methods and Examples. 2013. 665.

Papamakarios George, Nalisnick Eric, Rezende Danilo Jimenez, Mohamed Shakir, Lakshminarayanan Balaji. Normalizing flows for probabilistic modeling and inference // arXiv preprint arXiv:1912.02762. 2019.

Perugachi-Diaz Yura, Tomczak Jakub M., Bhulai Sandjai. i-DenseNets // CoRR. 2020. abs/2010.02125.

Prenger Ryan, Valle Rafael, Catanzaro Bryan. Waveglow: A flow-based generative network for speech synthesis // ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2019. 3617–3621.

Rasmussen Carl Edward. Gaussian processes in machine learning // Summer School on Machine Learning. 2003. 63–71.

Rezende Danilo Jimenez, Mohamed Shakir. Variational inference with normalizing flows // arXiv preprint arXiv:1505.05770. 2015.

Rezende Danilo Jimenez, Mohamed Shakir, Wierstra Daan. Stochastic Backpropagation and Approximate Inference in Deep Generative Models // Proceedings of the 31th International Conference on Machine Learning, ICML. 2014.

Rezende Danilo Jimenez, Papamakarios George, Racanière Sébastien, Albergo Michael S, Kanwar Gurtej, Shanahan Phiala E, Cranmer Kyle. Normalizing flows on tori and spheres // arXiv preprint arXiv:2002.02428. 2020.

Rudin Walter. Real and Complex Analysis. 1987 // Cited on. 1987. 156.

Rue Havard, Held Leonhard. Gaussian Markov random fields: theory and applications. 2005.

Sei Tomonari. A Jacobian inequality for gradient maps on the sphere and its application to directional statistics // Communications in Statistics-Theory and Methods. 2013. 42, 14. 2525–2542.

Senior Andrew W, Evans Richard, Jumper John, Kirkpatrick James, Sifre Laurent, Green Tim, Qin Chongli, Žídek Augustin, Nelson Alexander WR, Bridgland Alex, others . Improved protein structure prediction using potentials from deep learning // Nature. 2020. 577, 7792. 706–710.

Sohl-Dickstein Jascha. Two equalities expressing the determinant of a matrix in terms of expectations over matrix-vector products // arXiv preprint arXiv:2005.06553. 2020.

Tabak Esteban G, Turner Cristina V. A family of nonparametric density estimation algorithms // Communications on Pure and Applied Mathematics. 2013. 66, 2. 145–164.

Tabak Esteban G, Vanden-Eijnden Eric, others . Density estimation by dual ascent of the log-likelihood // Communications in Mathematical Sciences. 2010. 8, 1. 217–233.

Tang Yunhao, Agrawal Shipra. Boosting trust region policy optimization by normalizing flows policy // arXiv preprint arXiv:1809.10326. 2018.

Teshima Takeshi, Ishikawa Isao, Tojo Koichi, Oono Kenta, Ikeda Masahiro, Sugiyama Masashi. Coupling-based Invertible Neural Networks Are Universal Diffeomorphism Approximators // arXiv preprint arXiv:2006.11469. 2020.

Ubaru Shashanka, Chen Jie, Saad Yousef. Fast Estimation of tr(f(A)) via Stochastic Lanczos Quadrature // SIAM Journal on Matrix Analysis and Applications. 2017. 38, 4. 1075–1099.

Ulrich Gary. Computer Generation of Distributions on the M-Sphere // Journal of the Royal Statistical Society: Series C (Applied Statistics). 1984. 33, 2. 158–163.

Uzunova Hristina, Ehrhardt Jan, Jacob Fabian, Frydrychowicz Alex, Handels Heinz. Multi-scale GANs for Memory-efficient Generation of High Resolution Medical Images // International Conference on Medical Image Computing and Computer-Assisted Intervention. 2019. 112–120.

Wang Fangpo, Gelfand Alan E. Directional data analysis under the general projected normal distribution // Statistical methodology. 2013. 10, 1. 113–127.

Ward Patrick Nadeem, Smofsky Ariella, Bose Avishek Joey. Improving exploration in soft-actor-critic with normalizing flows policies // arXiv preprint arXiv:1906.02771. 2019.

Zhang Yunong, Leithead William E. Approximate implementation of the logarithm of the matrix determinant in Gaussian process regression // Journal of Statistical Computation and Simulation. 2007. 77, 4. 329–348.

A Appendix

A.1 Cover Density

The Figure 9 illustrates the learned proposal distribution $q(s)$ corresponding to the matrix

$$A = \begin{bmatrix} -0.7056 & 0.6741 & -0.5454 \\ 0.9107 & 1.0682 & 0.1424 \\ -1.2754 & -0.1769 & 1.0084 \end{bmatrix}$$

which is created with `torch.randn(3,3)` and torch manual seed 15. The optimal proposal distribution $q^* \propto \|As\|^{-n}$ is illustrated in Figure 15. We trained the model for 10k iterations and in contrast to the architecture in Section 4.2, we used $N_F = 6$ flows with autoregressive masking and Neural Spline flows for both the spherical part and the interval part of $\mathbb{S}^1 \times [-1, 1]$ with $N_B = 32$ bins, see again Rezende et al. (2020) for details.

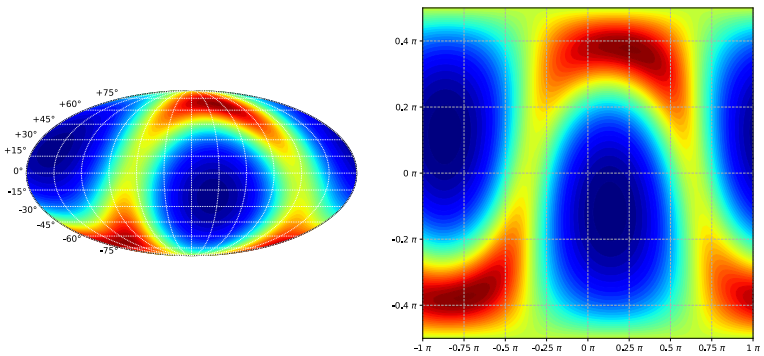


Figure 15: Optimal proposal distribution corresponding to A.

A.2 Vectorizing Recursive Unfolding

Two steps of applying U yields

$$\mathbb{S}^D \xrightarrow{U} \mathbb{S}^{D-1} \times [-1, 1] \xrightarrow{U} \mathbb{S}^{D-2} \times [-1, 1]^2$$

with components

$$(x_{1:D+1}) \mapsto \left(\frac{x_{1:D}}{\sqrt{1 - x_{D+1}^2}}, x_{D+1} \right) \mapsto \left(\frac{x_{1:D-1}}{\sqrt{1 - x_{D+1}^2}\sqrt{1 - \frac{x_D^2}{1 - x_{D+1}^2}}}, \frac{x_D}{\sqrt{1 - x_{D+1}^2}}, x_{D+1} \right)$$

The first component of the resulting vector can be simplified via

$$\frac{1}{\sqrt{1 - a^2}} \frac{1}{\sqrt{1 - \frac{b^2}{1 - a^2}}} = \frac{\sqrt{1 - a^2}}{\sqrt{1 - a^2}} \frac{1}{\sqrt{1 - a^2 - b^2}} = \frac{1}{\sqrt{1 - a^2 - b^2}}$$

Thus, by applying the concept of mathematical induction, the function $T_{s \to c} \colon \mathbb{S}^D \to \mathbb{S}^1 \times [-1, 1]^{D-1}$ takes the form

$$T_{s \to c} \colon (x_1, x_2, \ldots, x_{D+1}) \mapsto \left(\frac{x_1}{\sqrt{1 - \sum_{i=3}^{D+1} x_i^2}}, \frac{x_2}{\sqrt{1 - \sum_{i=3}^{D+1} x_i^2}}, \ldots, \frac{x_k}{\sqrt{1 - \sum_{i=k+1}^{D+1} x_i^2}}, \ldots, x_{D+1} \right)$$

A.3 Details on Moebius Transformation

Inverse First, we proof the that the inverse of the Moebius transformation is given by $h_\omega^{-1}(z) = h_{-w}(z)$. In order to see this, let us denote

$$\lambda = \frac{r^2 - \|\omega\|^2}{\|z - \omega\|^2}$$

Then

$$h_\omega(z) = \frac{r^2 - \|\omega\|^2}{\|z - \omega\|^2}(z - \omega) - \omega = \lambda(z - \omega) - \omega = z' \tag{17}$$

and

$$h_{-\omega}(z') = \frac{r^2 - \|\omega\|^2}{\|z' + \omega\|^2}(z' + \omega) + \omega$$

Then, rearranging Equation 17 gives

$$z' + \omega = \lambda(z - \omega)$$

If we insert the last term into $h_{-\omega}$, we obtain

$$h_{-\omega}(z') = \frac{r^2 - \|\omega\|^2}{\|\lambda(z - \omega)\|^2}\lambda(z - w) + w = \frac{\lambda^2}{\lambda^2}(z - \omega) + \omega = z$$

which shows that $h^{-1}(z) = h_{-w}(z)$.

Jacobian The Moebius transformation is given by

$$h_\omega(z) = \frac{r^2 - \|\omega\|^2}{\|z - \omega\|^2}(z - \omega) - \omega$$

In order to make the derivative of h_ω more accessible, we denote

$$p(z) = \frac{r^2 - \|\omega\|^2}{\|z - \omega\|^2} \qquad \text{and} \qquad q(z) = (z - \omega)$$

Then, the derivative of h_ω calculates as

$$\frac{\partial}{\partial z}h_\omega(z) = \frac{\partial}{\partial z}\left[p(z)q(z)\right] = q(z)\frac{\partial}{\partial z}p(z) + \left[\frac{\partial}{\partial z}q(z)\right]p(z)$$

with

$$\frac{\partial}{\partial z}p(z) = -\frac{2\left(r^2 - \|w\|^2\right)}{\|z - w\|^4}(z - w)^\top \qquad \frac{\partial}{\partial z}q(z) = I$$

which yields

$$\frac{\partial}{\partial z}h_\omega(z) = -\frac{2\left(r^2 - \|w\|^2\right)}{\|z - w\|^4}(z - w)(z - w)^\top + \frac{r^2 - \|\omega\|^2}{\|z - \omega\|^2}I$$

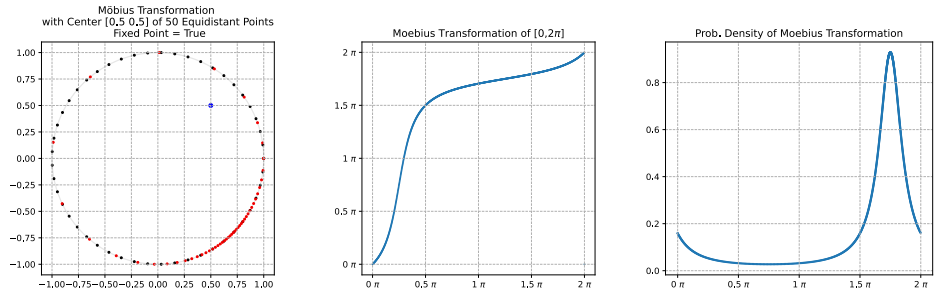
Fixed Point Property To illustrate the necessity of the fixed point property for a convex combination of Moebius transformation, we transform a uniform base distribution into a more complex distribution, that is we go from a simple space Z to a more complex space X. To stay in the terminology of Section 2.2, let f be the flow with fixed point property and g the flow without. Furthermore, we recall the relationship $J_f^{-1}(x) = J_g^{-1}(x + \varphi)$. Then the update rule for $f\colon Z \to X$ is:

$$\log p(x) = \log \pi(z) + \log|\det J_{f^{-1}}(x)| \tag{18}$$

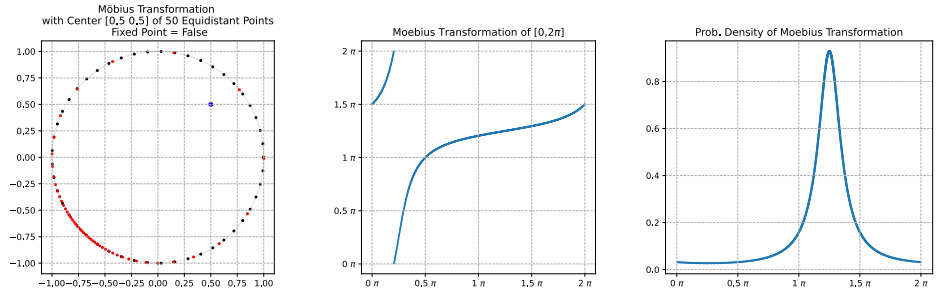
and for $g\colon Z \to X$ is

$$\log p(x) = \log \pi(z) + \log|J_g^{-1}(x + \varphi)| \tag{19}$$

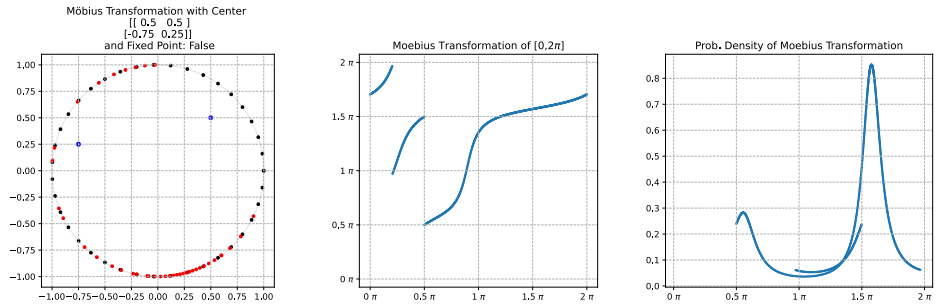
The Figure 16(a) shows f with density, 16(b) shows g with density, 16(c) shows $\sum_i \rho_i g_i$ and 16(d) shows $\sum_i \rho_i f_i$



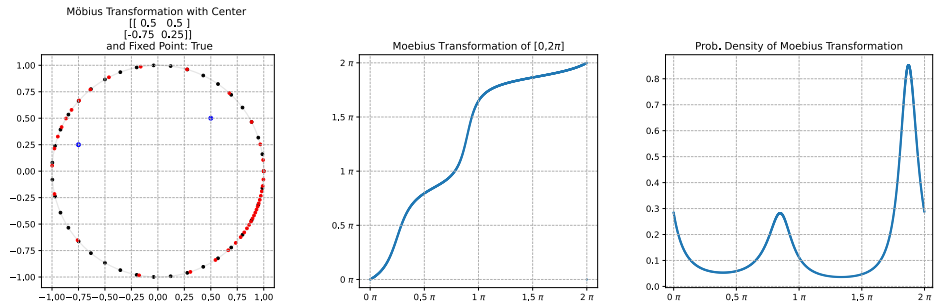
(a) Illustration of f: a diffeomorphism with well defined density via Equation 18.



(b) Illustration of g: is discontinuous but derivatives at discontinuity coincide. Has proper density via Equation 19.



(c) Illustration of $\sum \rho_i g_i$: a convex combination of discontinuous functions is not a bijection anymore. The density is degenerate.



(d) Illustration of $\sum \rho_i f_i$; convex combination of diffeomorphisms is a diffeomorphism. The density is well defined.

Figure 16: Left column: The black dots are uniform samples, the blue dots the projection centers ω, and the red dots are the image of the Moebius transformation. Mid column: The flow as a function on $[0, 2\pi)$. Right: The density of the Moebius transformation according to the update rules of Equation 18 and 19.

	A_1	A_2	A_3	A_4	A_5
Absolute det.	520.36	748.68	945.02	3000.5	252.29

Table 8: Absolute determinants of matrices of the experiment in Section 4.2.

A.4 Dense 10×10 Matrices

In this section, we present the absolute determinants in Table 8 and the corresponding 10×10 matrices can be found here which were used in the experiment in Section 4.2. Numbers are rounded to two decimals.

A.5 Filter of Convolutional Layer

Here, we present the 3×3 filter k which was used to create the 16×16 matrix W of our experiment in Section 4.2. The matrix W is the equivalent matrix of the convolution $k \star x$ where x is an arbitrary 4×4 image and can be found here under `matrix.npy`.

$$k = \begin{bmatrix} -0.107 & -0.689 & -0.027 \\ 0.226 & 1.393 & -0.544 \\ -0.28 & -0.467 & 0.024 \end{bmatrix}$$

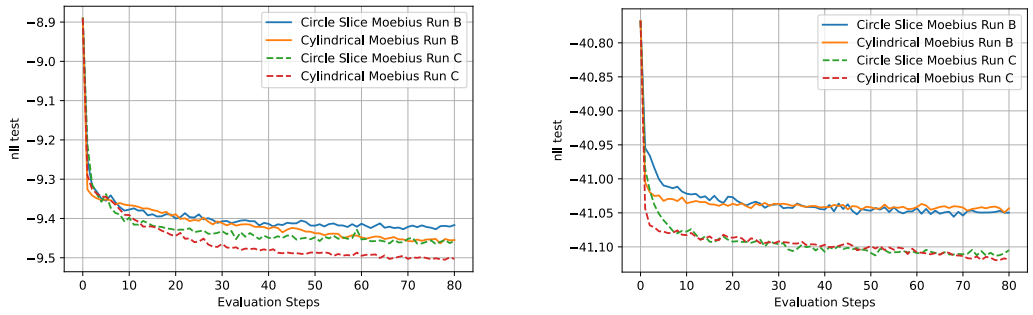
A.6 Parameters of Mixture of Power Spherical Distributions

In this section, we present the parameters μ and κ of our experiment in Section 3.1 for the dimensions $D = 3$. The parameters for the higher dimensions can be found in the git repository. The parameter μ is a $N_m \times D$ matrix and $\kappa \in \mathbb{R}^{N_m}$ where N_m is the number of mixture components and D is the dimension. All parameters were sampled randomly but with a fixed seed of 42 both for numpy and PyTorch to ensure the same parameter set across different runs. All digits are rounded to two decimals. For $D = 3$, the parameters are:

$$\mu = \begin{bmatrix} -0.48 & -0.88 & 0 \\ -0.16 & 0.66 & -0.74 \\ 0.26 & 0.7 & 0.67 \\ 1 & 0 & 0 \end{bmatrix} \qquad \kappa = \begin{bmatrix} 19.5 & 21 & 18 & 22.5 \end{bmatrix}$$

A.7 Additional Density Fit Experiment

In Figure 17 we present the training development in terms of the negative log likelihood (nll) for two further runs, B and C, respectively. The same picture arises as in the result Section 4.1. In $D = 64$ both models perform equally well, but in $D = 32$, the Cylindrical flows perform better. The parameters of the synthetic target density for run B and C can be found in the git repository of the author.



(a) nll in dimension $D = 32$

(b) nll in dimension $D = 64$

Figure 17: Development of the negative log likelihood in nats evaluated on a 10^5 sample sized test set over 80 evaluation steps.