INAUGURAL–DISSERTATION
zur
Erlangung der Doktorwürde
der
Gesamtfakultät für Mathematik, Ingenieur- und Naturwissenschaften
der
Ruprecht-Karls-Universität
Heidelberg

vorgelegt von

Draxler, Felix Matthias, MSc.
aus München

Tag der mündlichen Prüfung: _____

Architectural Constraints of Normalizing Flows

Betreuer: Prof. Dr. Ullrich Köthe
Zweitbetreuer: Prof. Dr. Christoph Schnörr

Abstract

In this thesis, we consider Normalizing Flows, a class of models that leverage neural networks to represent probability distributions, enabling efficient sampling and density estimation. The focus of our work is to develop *versatile normalizing flows*, that is flexible methods readily applicable to arbitrary problems.

We therefore theoretically examine the expressivity of existing architectures. We find that volume-preserving flows are fundamentally biased and identify a fix. We improve on universality guarantees for coupling-based flows, showing that well-conditioned affine coupling flows are universal. We find that the latter scale favorably with dimension in comparison to Gaussianization flows.

We then proceed to lift architectural restrictions from normalizing flows altogether via the introduction of Free-Form Flows. This framework trains arbitrary neural network architectures as normalizing flows. This allows for the first time, among others, cheap rotation-equivariant normalizing flows, normalizing flows on arbitrary Riemannian manifolds, and injective flows based on feed-forward autoencoders. This model is significantly more flexible to adapt to novel problems, performs comparable or better than existing normalizing flows and competitive with methods with iterative inference such as diffusion models and flow matching.

Zusammenfassung

In dieser Dissertation betrachten wir Normalizing Flows, eine Klasse von Methoden, die neuronale Netzwerke nutzen, um Wahrscheinlichkeitsverteilungen darzustellen, die effizientes Sampling und Dichteschätzung ermöglichen. Der Schwerpunkt dieser Arbeit liegt auf der Entwicklung von *flexiblen Normalizing Flows*, die problemlos auf beliebige Probleme anwendbar sind.

Daher untersuchen wir zunächst die Eigenschaften bestehender Architekturen theoretisch. Wir stellen fest, dass volumenerhaltende Flows eine verzerrte Dichte lernen und entwickeln eine Lösung dafür. Wir verbessern existierende Universalitätsgarantien für coupling-basierte Flows, indem wir die Universalität von wohlkonditionierten affinen Coupling Flows bestätigen. Außerdem stellen wir fest, dass letztere im Vergleich zu Gaussianization Flows günstig mit der Dimension skalieren.

Anschließend heben wir die architektonischen Beschränkungen existierender Normalizing Flows auf, indem wir Free-Form Flows einführen. Dieses Framework ermöglicht es, beliebige neuronale Netzwerkarchitekturen als Normalizing Flows zu trainieren. Das ermöglicht erstmals unter anderem kostengünstige rotationsequivariante Normalizing Flows, Normalizing Flows auf beliebigen Riemannschen Mannigfaltigkeiten und injektive Flows basierend auf Autoencodern. Free-Form Flows sind erheblich flexibler in der Anpassung an neue Probleme, zeigen vergleichbare oder bessere Performanz als bestehende Normalizing Flows und sind wettbewerbsfähig mit Methoden mit iterativer Inferenz wie Diffusionsmodellen und Flow Matching.

Acknowledgments

An immense thank-you goes to my PhD supervisors Ullrich Köthe and Christoph Schnörr, who have supported and guided this project through helpful, positive, critical and results-oriented feedback and pointers.

I would like to extend my gratitude to all my colleagues at the Computer Vision and Learning Lab (CVL) for the collaborative and friendly environment, the interesting discussions and the exciting projects. I am especially thankful to Armand Rousselot, Peter Sorrenson, Jens Müller, Sander Hummerich, Stefan Wahl, Lars Kühmichel, Robert Schmier, Stefan Radev, Lynton Ardizzone, Jakob Kruse, The-Gia Leo Nguyen, Carsten Rother, and Bogdan Savchynskyy. A big thank you also goes to my colleagues in the Image & Pattern Analysis Group (IPA) for the helpful feedback and friendly atmosphere, especially to Jonathan Schwarz, Bastian Boll, Dmitrij Sitenko, Fabrizio Savarino, Ruben Hühnerbein, Alexander Zeilmann, Matthias Zisler, and Artjom Zern.

Thank you, Mel, for always being so supportive and helping me every step of the way. And thank you, Timba, for enduring late evening walks after paper deadlines and lifting the mood in the office.

Finally, thanks to Jens, Peter, Christoph, Fiona and Gabi for proofreading parts of this thesis.

This work is supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC-2181/1 - 390900948 (the Heidelberg STRUCTURES Cluster of Excellence). It is also supported by the Vector Stiftung in the project TRINN (P2019-0092). The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG. We thank the Center for Information Services and High Performance Computing (ZIH) at TU Dresden for its facilities for high throughput calculations.

Contents

1. Motivation

Machine learning is a field at the intersection of computer science and statistics. It aims to develop algorithms that enable computers to learn from and make predictions based on data (Bishop, 2006). Typically, data is provided through a set of example data points called the training data. The task of a machine learning system is to infer a mathematical computation that, when applied to the training inputs x, yields the corresponding training outputs y (*supervised learning*), or reproduces the structure of data points x without a specified output (*unsupervised learning*). The system should be set up in a way behaves as intended even for values it has not seen during training (*generalization*). Contrast this to a classical algorithm where an expert devises a sequence of mathematical operations that compute outputs; in machine learning, the algorithm is derived by example.

Machine learning systems have achieved remarkable success in various areas. The following is a non-exhaustive list, and the references each point to a central breakthrough: In computer vision, they have surpassed human-level performance on tasks such as object recognition and classification in images (Krizhevsky et al., 2012). In natural language processing, they have significantly improved text translation and are able to perceive and generate natural text (Vaswani et al., 2017). In drug discovery and protein folding, they have achieved unprecedented accuracy in predicting protein folding, a complex problem with significant implications for understanding diseases and developing new therapies (Senior et al., 2020). *Generative models* can generate highly realistic images, videos, and audio, as well as solve inverse problems in scientific applications, such as reconstructing high-resolution images from low-resolution counterparts, and in simulating complex systems (Goodfellow et al., 2014).

The backbone of modern machine learning, including these examples, are *artificial neural networks* or simply *neural networks*. They consist of connected artificial neurons that each process input from other neurons to compute some output. Neurons are organized into subsequent layers and each layer of neurons receives the output from the previous layer as input. The input to the first layer is the input to the entire neural network, and the network's output is collected from the last layer. All values are represented by scalar real-valued numbers. To train a neural network, one fixes the number of neurons and their arrangement into layers and then the computation of each neuron is gradually adapted via gradient descent on a loss function that measures the discrepancy between the target imposed by the training data and the actual behavior the network (Rumelhart et al., 1986). An arrangement of neurons into layers and their connectivity is called a neural network *architecture*.

The resulting systems are highly flexible and generic: Only little structural information about the data at hand must be available in order to successfully train a neural network. This has also been shown theoretically: If a neural network is large enough, it is *universal* in the sense that it is able to represent arbitrary target functions as well as is required (Hornik, 1991). This is related to Turing completeness in computability theory which states that any Turing machine can be simulated by a given system. It is a strong statement in the sense that there are no fundamental barriers preventing to represent any target function of interest.

However, real data often contains noise and uncertainty, leading to no unique mapping between input and output in supervised learning. For example, predicting the outcome of a quantum experiment or using inputs from sensors with finite resolution introduces intrinsic randomness. A natural approach to make predictions under uncertainty is to model the *answer via a probability distribution* over possible outcomes. Similarly, in unsupervised learning, the structure of the data can be represented as a probability distribution. This can be done by discretizing the output space and predicting a

histogram of the possible outcomes. This explicit representation fails when predicting many variables jointly, as the total number of outcomes scales exponentially with the number of dimensions (*curse of dimensionality*).

To address this, several approaches have been proposed to model probability distributions using neural networks. They map simple noise sources, such as samples from a (multivariate) standard normal distribution, to *samples* of the target probability distribution. This avoids the curse of dimensionality as the neural network yields a single sample from the target distribution rather than fully representing the distribution explicitly. This effectively captures the underlying probability distribution without the computational infeasibility of dealing with exponentially growing outcome spaces.

Such a neural network is called a *deep generative model* (Goodfellow et al., 2014). There are two fundamental ways to query a generative model: First, the model can yield *samples* from the distribution. In supervised learning, this yields the plausible predictions given an input; in unsupervised learning, it produces data points similar to the training data. Second, many deep generative models can evaluate the probability density at points of interest (*density estimation*). Besides the practical applications mentioned, we argue that properly communicating the uncertainty underlying a prediction is essential for deploying neural networks in critical and large-scale contexts, such as autonomous driving, medicine, and climate change.

A plethora of architecture and training methods have been proposed, most notably variational autoencoders (VAEs, (Kingma & Welling, 2014)), generative adversarial networks (GANs, (Goodfellow et al., 2014)), normalizing flows (Rezende & Mohamed, 2015) as well as their continuous variant, continuous normalizing flows (Chen et al., 2018b), as well as diffusion models (Sohl-Dickstein et al., 2015). These variants share the approach of computing samples via a transport from random noise, but differ in the way they employ neural networks and how they are trained.

The goal of this thesis is to develop versatile generative model architectures that can model arbitrary probability distributions of interest. In particular, we focus on normalizing flows since they are deeply rooted in information theoretic principles, promising to be agnostic to the data modality. Normalizing flows do not simply inherit their universality from neural networks: They learn a function from data to latent space such that the data is mapped to latent codes distributed like a normal distribution. Then, samples can be generated by passing random noise into the *inverse of the learned function*, reversing the above process. This means that the learned function must be invertible, requiring the use of an *invertible neural network*. This kind of neural networks modifies the architecture of neural networks in terms of how neurons are connected, and thus does not fall under the classical universality guarantees mentioned above.

We approach the task of developing versatile normalizing flows from two perspectives:

First, we establish a theoretical framework grounded in the rigorous formulation of the considered models. In particular, we focus on distributional universality in chapter 5, that is given an architecture, can we learn arbitrary target distributions? Also, we address how the required computational resources scale with the problem at hand in chapter 6.

Second, the architectural constraints in invertible neural network architectures make adopting them to problems with special structure difficult. For example, only expensive invertible neural network architectures are known that learn rotation invariant distributions, a useful prior for modeling the distribution of molecules (Köhler et al., 2020), or when the data lies on a known Riemannian manifold (Chen & Lipman, 2024). Developing invertible neural networks for these cases is complex at best and may be impossible at worst. We overcome this limitation via a new scheme to train arbitrary neural network architectures as normalizing flows in chapter 7. This allows us to leverage the significant advancements in versatile neural network architecture research for efficient and powerful generative models.

2. Related Work

This chapter gives a high-level overview of the most prominent related work to each considered topic. We refer to the reader to the related work in the corresponding part of this thesis for a detailed discussion of the existing literature.

Applications Normalizing flows have been used to solve various problems. A prominent branch is solving inverse problems in domains such as astrophysics (Ardizzone et al., 2018b), mechanical engineering (Noever-Castelos et al., 2022), image colorization (Ardizzone et al., 2020a), medicine (Adler et al., 2019), epidemiology (Radev et al., 2021), among others. Another branch considers modeling the distribution of images (Kingma & Dhariwal, 2018) with applications to trustworthy generative classifiers (Ardizzone et al., 2020b; Mackowiak et al., 2021). Other applications lie in causality (Müller et al., 2021) and computational biology (Noé et al., 2019).

Universality theorems Standard neural networks that give a single prediction for each input are known to be universal function approximators: Given a target mapping between inputs and outputs that should be modeled by a neural network, if the neurons in a neural network are coordinated in the right way, they can approximate this mapping to arbitrary precision. The classical results show universality of vanilla neural networks (Cybenko, 1989; Hornik, 1991), but there are variants for different architectures such as convolutional networks (Zhou, 2020; Heinecke et al., 2020) or for data represented in special geometry (Kratsios & Bilokopytov, 2020).

In the context of normalizing flows, we are interested in whether the models can transform the random latent codes into the right structured noise into the target probability distribution, despite the constraints introduced to make the underlying neural networks invertible. The predominant approach tackles the problem by showing that invertible neural networks can approximate any invertible mapping, which thus also applies to the transformation between random and structured noise (Teshima et al., 2020a; Koehler et al., 2021). However, these works require the involved neural networks to become arbitrarily ill-conditioned, among other shortcomings.

We also consider the special case of volume-preserving normalizing flows, a family of architectures whose additional constraints can be beneficial in some applications. To the best of our knowledge, no previous analysis showed that they are fundamentally biased and how to fix this shortcoming.

Complexity guarantees The universality theorems above are useful as they assure that there are no fundamental barriers from learning certain functions. However, they do not guarantee or only loosely bound the computational complexity required to achieve a given quality. A bound of practical relevance would limit how many layers or neurons are required to achieve a certain quality, since in practice only limited resources are available.

Some complexity estimates are available for vanilla neural networks (Poggio et al., 2017). For invertible neural networks, an important complexity trade-off lies between the expressive power of each individual block in a normalizing flow with the total number of layers. Many such variants have been proposed, see (Kobyzev et al., 2021) for a review. The only theoretical guarantee is that three affine coupling layers are enough to fit arbitrary distributions (Koehler et al., 2021). However, as mentioned before, this result has limited practical applicability due to the unrealistic assumptions it makes.

Incorporating prior knowledge If there is existing knowledge about the structure of the data, then it is advantageous to incorporate it into the architecture of the neural network. For example, to process data in images, it is helpful to first process the image in small patches before combining the results into a global prediction (Fukushima, 1980), which has led to a breakthrough in classifying real-world images (Krizhevsky et al., 2012). Introducing such knowledge is known as an *inductive bias*. In addition, properties of the data at hand might be known that should be strictly incorporated into the model. For example, molecular data should often be treated identically regardless of global orientation and location, and some data, in particular in scientific applications, is known to live on a manifold whose topology and geometry should be obeyed. Below, we provide more details on how generative model architectures, particularly normalizing flows, can incorporate prior knowledge.

A broad literature exists building neural network architectures with different inductive biases (Zhang et al., 2023a). However, integrating these advances into normalizing flow architectures is nontrivial, as the network must also be invertible. Such an architecture has been achieved for images (Kingma & Dhariwal, 2018), but not for rotation equivariance, among others. One alternative is restricting each block to an infinitesimal action, essentially parameterizing an ordinary differential equation from latent codes to data (Chen et al., 2018b). However, this increases costs at inference time, since the number of steps can grow large to accurately solve the differential equation.

A particularly prevalent prior knowledge about data is that real-world data such as images, sound or complex sensor data is usually represented in raw form such as pixel or time-varying waveform values, but they can be described efficiently in a low-dimensional representation through lossy or even lossless compression algorithms that faithfully preserve the corresponding content (manifold hypothesis, Bengio et al. (2013)). For a generative model, this has the implication that modeling such data in its raw format wastes resources on modeling the negligible noise in addition to the variation of the meaningful content in a prediction. This can also lead to badly conditioned transformations between data and latent noise.

Generative autoencoders and *injective (normalizing) flows* tackle this issue by learning both a low-dimensional representation and a distribution on it. Existing work either follow a two-stage approach which first learns a low-dimensional representation and then its distribution, or learn a distribution in the high-dimensional space together with a core set of latent variables that span the low-dimensional manifold. To avoid ill-conditioned functions, the data is usually augmented with small-variance noise (Horvat & Pfister, 2021). *Rectangular flows* jointly learn the low-dimensional representation and a low-dimensional distribution on it, but this comes with the expensive use of several conjugate gradient steps (Caterini et al., 2021).

Similarly, the case where the data is known to lie on a manifold requires special attention. Examples are normed vectors on a (hyper)sphere, angles on a torus, points on a surface in an embedding space, or manifolds with non-Euclidean geometry such hyperbolic space. These manifolds are also not naturally represented by full-dimensional generative models, as this is incompatible with the topology, for example at the poles of a sphere. Intuitively, a model respects the topology if it treats data points similarly whenever they are close according to the geometry.

Riemannian generative models address this challenge of learning distributions on a known manifold. To the best of our knowledge, Riemannian variants of normalizing flows are only known in special cases, as they are nontrivial to adapt to arbitrary manifolds while maintaining expressivity. Like for incorporating inductive biases, multistep methods offer a viable approach to learn generative models on arbitrary manifolds, but this makes them expensive at inference and, in some cases, also at training time (Mathieu & Nickel, 2020; Lou et al., 2020).

3. Contribution

Normalizing flows are a well-established generative model for learning multivariate probability densities based on neural networks. In this thesis, we analyze and extend their versatility.

Therefore, we theoretically underpin the universality and efficiency of normalizing flows based on invertible neural networks. However, such architectures are not known for all data modalities. We address this by lifting all architectural constraints through the introduction of free-form flows, allowing arbitrary neural network architectures to be trained as normalizing flows efficiently for the first time.

In the remainder of this chapter, we provide details on the contributions of this thesis and where they were originally published. If not specified otherwise, the first author takes credit for the majority of the contributions.

3.1. Theoretical framework for normalizing flows

Characterizing the Role of a Single Coupling Layer in Affine Normalizing Flows.
Felix Draxler, Jonathan Schwarz, Christoph Schnörr, and Ullrich Köthe.
In German Conference on Pattern Recognition (GCPR), 2020, best paper honorable mention.

Many normalizing flows rely on invertible neural networks consisting of a sequence of affine coupling blocks, each of which is restricted to a limited class of invertible functions that is particularly easy to invert analytically. Empirically, a deep sequence of such blocks can represent complicated probability distributions such as those of images. To understand their success, we characterize the effect of a single coupling block for learning a distribution in a deep sequence. We derive that a single block estimates and then normalizes conditional moments of the data distribution. We give a tight lower bound for the maximum possible loss improvement by a single coupling, depending on the orthogonal transformation of the data. This bound can be used to identify the optimal orthogonal transform. Together, this yields a block-wise training algorithm for deep affine coupling flows. Toy examples confirm our findings and stimulate further research by highlighting the remaining gap between block-wise and end-to-end training of deep affine flows. See section 5.3 for details, based on Draxler et al. (2020).

This work lays the foundation to the subsequent theoretical contributions that extrapolate the effect of a single block to a deep coupling flow.

On the Universality of Volume-Preserving and Coupling-Based Normalizing Flows.
Felix Draxler, Stefan Wahl, Christoph Schnörr, and Ullrich Köthe.
In International Conference on Machine Learning (ICML), 2024.

In practice, no fundamental restrictions have been identified as to which probability distributions can be represented by a coupling-based normalizing flow, even in the simplest variant of an affine coupling block such as RealNVP (Dinh et al., 2017). Despite their prevalence in scientific applications, a comprehensive understanding explaining their success remains elusive. Existing theorems fall short as they require the use of arbitrarily ill-conditioned neural networks, limiting practical applicability. We propose a distributional universality theorem for well-conditioned coupling-based normalizing flows for the first time. In addition, we show that volume-preserving normalizing flows are not universal, what distribution they learn instead, and how to fix their expressivity. Our results support the general wisdom that affine and related couplings are expressive and in general outperform volume-preserving

flows, bridging a gap between empirical results and theoretical understanding. Finally, we show that if coupling is more expressive than affine, each block can reduce a loss component that is inaccessible to affine couplings. See sections 5.2 and 5.4 for details, based on Draxler et al. (2024b).

Individual contributions: FD derived the theorems and performed the experiments, except: SW trained the volume-preserving flow and derived a counter-example showing the non-universality of volume-preserving flow.

Whitening Convergence Rate of Coupling-based Normalizing Flows.
Felix Draxler, Christoph Schnörr, and Ullrich Köthe.
In Annual Conference on Neural Information Processing (NeurIPS), 2022, oral presentation.

One key hyperparameter of neural network architectures and coupling flows in particular is the choice of architecture parameters, such as the number of blocks. It is resource-intensive to tune, given that the network has to be retrained for each new configuration. Theoretical guidance for choosing parameters such as depth is therefore of great practical relevance. While existing work shows that three layers are sufficient for representing any distribution, their proof construction for this statement is severely limited as argued above. In particular, they make no statement about the strict convergence criterion used in practice, the Kullback-Leibler divergence. For the first time, we make a quantitative statement about this kind of convergence: We prove that all coupling-based normalizing flows perform whitening of the data distribution (i.e. diagonalize the covariance matrix) and derive corresponding convergence bounds that show a linear convergence rate in the depth of the flow. The convergence rate is independent of the dimension of the data in late training. Numerical experiments confirm our theory. See section 5.3 for details, based on Draxler et al. (2020).

On the Convergence Rate of Gaussianization with Random Rotations.
Felix Draxler, Lars Kühmichel, Armand Rousselot, Jens Müller, Christoph Schnörr, and Ullrich Köthe.
In International Conference on Machine Learning (ICML), 2023.

While coupling blocks are a common choice for normalizing flows on high-dimensional data, the structurally simpler Gaussianization block has also achieved compelling performance in low dimensions or little training data. Empirically, however, the convergence speed of Gaussianization slows down as the dimension increases. We confirm this analytically, showing that the number of required layers scales linearly with the dimension for Gaussian input in late training. We compare this to the previously derived result on coupling flows, where the number of required layers is largely unaffected by dimension. This suggests that the inferior scaling of Gaussianization occurs because the model struggles at capturing dependencies between dimensions. Empirically, we find the same linear increase in cost for arbitrary input in early training, but observe favorable scaling for some distributions. We explore potential speed-ups and formulate challenges for further research. See section 6.3 for details, based on Draxler et al. (2022).

Individual contributions: FD derived the theorems and suggested the experiments. LK performed the experiments, with support by AR and JM.

3.2. Removing architectural constraints from normalizing flows

Free-form flows: Make any architecture a normalizing flow.
Felix Draxler[1], Peter Sorrenson[1], Lea Zimmermann, Armand Rousselot, and Ullrich Köthe.
In International Conference on Artificial Intelligence and Statistics (AISTATS), 2024.

Building versatile invertible neural networks has so far been largely constrained by the need to build architectures that are analytically and tractably invertible and whose Jacobian determinant is

[1]These authors contributed equally.

efficient to compute. We overcome these constraints by a novel training procedure for normalizing flows, which allows any dimension-preserving neural network to serve as a generative model through maximum likelihood training. It is enabled by an efficient estimator for the gradient of the change of variables formula, fundamental to maximum likelihood training. This allows placing the emphasis on incorporating prior knowledge about the data into the architecture. Specifically, we achieve competitive results in molecule generation benchmarks utilizing $E(n)$-equivariant networks. Moreover, our method is competitive in an inverse problem benchmark, while employing off-the-shelf ResNet architectures. See section 7.1 for details, based on Draxler et al. (2024a).

Individual contributions: PS derived the theoretical guarantees for the combined loss. FD coordinated and performed the majority of the experiments. AR and LZ contributed individual experiments.

Learning Distributions on Manifolds with Free-form Flows.
Peter Sorrenson[1], Felix Draxler[1], Armand Rousselot[1], Sander Hummerich, and Ullrich Köthe.
Preprint, under review, 2023.

Many real-world data, particularly in the natural sciences and computer vision, lie on known Riemannian manifolds such as spheres, tori or the group of rotation matrices. When building normalizing flows for such data, one is faced with the challenge that coupling blocks are not directly compatible to work with this data, requiring special constructions that make adaptation to a new manifold difficult or resorting to Euclidean models which violate the topology at hand. A competing approach is based on learning a distribution on such a manifold that involves learning a differential equation. While this is readily applicable to arbitrary Riemannian manifolds, sampling from the model and evaluating densities requires solving the differential equation, significantly increasing the cost for downstream tasks. Based on the free-form flow framework proposed above, we develop an alternative approach which is directly compatible with arbitrary Riemannian manifolds with a known projection function. For sampling, the resulting model only requires a single function evaluation followed by a projection to the manifold. We outperform normalizing flows specialized to the considered manifolds and achieve competitive performance to multistep methods at typically two orders of magnitude faster inference. See section 7.2 for details, based on Sorrenson et al. (2023).

Individual contributions: FD implemented the modified gradient estimator based on the network architecture proposed by PS, who then confirmed the estimator theoretically. AR and FD coordinated and performed experiments, with the help of SH.

Lifting Architectural Constraints of Injective Flows.
Peter Sorrenson[1], Felix Draxler[1], Armand Rousselot, Sander Hummerich, Lea Zimmermann, and Ullrich Köthe.
In International Conference on Learning Representations (ICLR), 2024.

Normalizing Flows, including free-form flows, maximize a full-dimensional likelihood on the training data. However, real data such as real-world images is often best represented on a lower-dimensional manifold, leading the model to expend significant compute on modeling noise. Injective Flows fix this by jointly learning a manifold and the distribution of the data projected to that manifold. So far, these approaches have been limited by restrictive architectures and/or high computational cost. We lift both constraints by adapting the free-form flow estimator for the maximum likelihood loss to the case of free-form bottleneck architectures. We further show that naively learning both the data manifold and the distribution on it can lead to divergent solutions, since the maximum likelihood loss comes with pathologies that negatively affect what projection is learned. We use these insights to motivate a stable maximum likelihood training objective. We perform extensive experiments on toy, tabular and image data, demonstrating the competitive performance of the resulting model. See section 7.3 for details, based on Sorrenson et al. (2024).

Individual contributions: PS came up with the original idea, derived and implemented the efficient gradient estimator. FD implemented the experimental framework, and identified the stability of the final gradient estimator. AR, SH and LZ contributed individual experiments.

4. Background

4.1. Generative modeling

As motivated in chapter 1, we are interested in learning probability densities with neural networks, termed a *generative model* over continuous variables. Discrete distributions can be modeled by adding small amounts of real-valued noise to the data (Uria et al., 2013).

Generative models can model both conditional $p(x|c)$ and unconditional probability densities $p(x)$. For simplicity, we consider the unconditional case, since we are interested in how probability distributions can be constructed using neural networks. From a conceptual perspective, it is easy to make an unconditional model conditional via some function that takes the condition as input and yields the parameters of the generative model representing the conditional distribution as output. Practically, the condition can be introduced by feeding the condition as an additional input into all involved neural networks, see e.g. (Ardizzone et al., 2020a).

We denote all ground truth probability densities as $p(\cdot)$ and a probability density that depends on (learnable) parameters θ by $p_\theta(\cdot)$. The argument to p respectively p_θ specifies the random variable it concerns. We explicitly specify such a random variable x via $p(x = \cdot)$ if it is unclear from context.

4.2. Normalizing flows

Normalizing flows are a class of generative models that represent a distribution $p_\theta(x)$ with parameters θ by learning an invertible function $z = f_\theta(x)$ so that the latent codes $z \in \mathbb{R}^D$ obtained from the data $x \in \mathbb{R}^D$ are distributed like a standard normal distribution $p(z) = \mathcal{N}(z; 0, I)$. Via the change of variables formula, see Köthe (2023) for an overview, this invertible function yields an explicit form for the density $p_\theta(x)$:

$$p_\theta(x) = p\Big(z = f_\theta(x)\Big)|f'_\theta(x)| = \Big((f_\theta^{-1})_\sharp p(z)\Big)(x), \tag{4.1}$$

where $f'_\theta(x) = \frac{\partial}{\partial x} f_\theta(x)$ is the Jacobian matrix of f_θ at x and $|f'_\theta(x)|$ is its absolute determinant. The last form denotes the *push-forward* of $p(z)$ through f_θ^{-1}, mapping from the latent distribution to the model distribution. We will later also consider generalizations of the change of variables formula to non-Euclidean geometry in equation (7.14) and injective functions in equation (7.29).

Equation (4.1) allows easily evaluating the model density at points of interest. Obtaining samples from $p_\theta(x)$ can be achieved by sampling from the latent standard normal and applying the inverse $f_\theta^{-1}(z)$ of the learned transformation:

$$x = f_\theta^{-1}(z) \sim p_\theta(x) \text{ for } z \sim p(z). \tag{4.2}$$

The change of variables formula in equation (4.1) can be used directly to train a normalizing flow. The corresponding loss minimizes the Kullback-Leibler divergence between the true data distribution $p(x)$ and the learned distribution, which can be optimized via a Monte-Carlo estimate of the involved expectation:

$$\mathcal{L} = \mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(x)) = \mathbb{E}_{x\sim p(x)}[\log p(x) - \log p_\theta(x)] \tag{4.3}$$

$$= \mathbb{E}_{x\sim p(x)}[-\log p_\theta(x)] + \text{const.} \tag{4.4}$$

This last variant makes clear that minimizing this loss is the same as maximizing the log-likelihood the model assigns to the data. For training, the expectation value is approximated using (batches of) training samples $x_{1,\dots,N}$.

Changing perspective, consider the push-forward of the data distribution to the latent space, that is the distribution of latent codes obtained from mapping the data distribution $p(x)$ through the invertible neural network f_θ:

$$p_\theta(z) = \Big((f_\theta)_\sharp p(x)\Big)(z) = p\Big(x = f_\theta^{-1}(z)\Big)|f_\theta^{-1\prime}(z)| \tag{4.5}$$

The above KL divergence can then be rewritten in the latent space:

$$\begin{aligned}
\mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(x)) &= \mathbb{E}_{x\sim p(x)}[\log p(x) - \log p_\theta(x)] && (4.6)\\
&= \mathbb{E}_{x\sim p(x)}[\log p(x) - \log p(z = f_\theta(x)) - \log|f_\theta'(x)|] && (4.7)\\
&= \mathbb{E}_{z\sim p_\theta(z)}[\log p(x = f_\theta^{-1}(z)) + \log|f_\theta^{-1\prime}(z)| - \log p(z)] && (4.8)\\
&= \mathbb{E}_{z\sim p_\theta(z)}[\log p_\theta(z) - \log p(z)] && (4.9)\\
&= \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|p(z)) && (4.10)
\end{aligned}$$

In other words, approximating $p(x)$ with $p_\theta(x)$ in the data space is directly tied to $p_\theta(z)$ approximating the target latent distribution $p(z)$. We will heavily use this fact in chapters 5 and 6.

For sampling from $p_\theta(x)$, draw a latent code $z \sim p(z) = \mathcal{N}(0, I)$ and apply the inverse of the learned function:

$$x = f_\theta^{-1}(z) \sim p_\theta(x). \tag{4.11}$$

A common alternative scenario is that the target distribution is known $p(x) \propto e^{-u(x)}$ up to its normalization constant. In this case, we can again use the change of variables to train with the reverse KL divergence:

$$\mathcal{D}_{\mathrm{KL}}(p_\theta(x)\|p(x)) = \mathbb{E}_{x\sim p_\theta(z)}[\log p_\theta(x) + u(x)] + \text{const.} \tag{4.12}$$

While this training objective is not limited by the size of the training set, training with reverse KL can be unstable and exhibit mode collapse (Nicoli et al., 2022). Throughout this thesis, we consider universality with respect to forward KL divergence.

4.3. Invertible neural networks

The change of variables formula equation (4.1) is only valid for invertible functions f_θ. This is, in general, not fulfilled by arbitrarily parameterized feed-forward neural networks, but requires special constructions so that the network is invertible. These constructions are called *invertible neural networks* (INN).

A useful invertible architecture for f_θ has to (i) be computationally easy to invert, (ii) be able to represent complex transformations, and (iii) have a tractable Jacobian determinant $|\det J|$ (Ardizzone et al., 2018b). A plethora of such architectures called have been suggested, see e.g. (Kobyzev et al., 2021) for an overview.

Constructing an architecture fulfilling these requirements is nontrivial; a major part of the literature is based on composing L invertible *blocks*:

$$f_\theta(x) = (f_{L,\theta_L} \circ \cdots \circ f_{1,\theta_1})(x). \tag{4.13}$$

The parameters collect the parameters of the individual layers $\theta = (\theta_1, \dots, \theta_L)$, but we usually drop this explicit dependence on the parameters for simplicity. It is easy to see that if each of the blocks is

easy to invert and the Jacobian determinant of each block is tractable, these properties transfer to the composition:

$$f_\theta^{-1}(z) = (f_1^{-1} \circ \cdots \circ f_L^{-1})(z), \tag{4.14}$$

$$|f'_\theta(x)| = |f'_L(x_{L-1}) \cdots f'_2(x_1) f'_1(x)| = |f'_L(x_{L-1})| \cdots |f'_2(x_1)||f'_1(x)|. \tag{4.15}$$

Here, we have denoted $x_1 = f_1(x)$ and $x_{i+1} = f_{i+1}(x_i)$. The expressivity of the overall transformation causes the expressivity of the resulting distribution $p_\theta(x)$. It depends on the expressivity and count of the blocks, which are the main concern of chapters 5 and 6.

The majority of invertible blocks is based on one-dimensional parameterized invertible functions $c_\theta : \mathbb{R} \to \mathbb{R}$, which we write as $z = c(x; \theta)$. In the following, we first explain how to parameterize one-dimensional invertible functions. We then show how to combine one-dimensional invertible functions to high-dimensional invertible blocks fulfilling the above requirements for an invertible neural network.

With the introduction of free-form flows in chapter 7, we overcome all of these requirements by using two separate networks for the forward f_θ and inverse maps $g_\varphi \approx f_\theta^{-1}$ and replacing the volume change term by an efficient gradient estimator.

4.3.1. One-dimensional invertible functions

In this section, we focus on how to represent one-dimensional invertible functions $c(x; \theta)$, a cornerstone of building high-dimensional invertible neural networks. In fact, much of the research on normalizing flow architectures is concerned with tuning these one-dimensional invertible functions, trading off expressivity with inversion cost.

- A particularly simple function is an **affine-linear function** with positive slope s:

  $$c(x; \theta) = sx + t. \tag{4.16}$$

  as proposed in NICE (Dinh et al., 2015) with $s = 1$ (which is volume-preserving, see section 4.3.3, and RealNVP (Dinh et al., 2017) with $s > 0$. Here, $\theta = [s; t] \in \mathbb{R}_+ \times \mathbb{R}$.

- **Nonlinear squared flow** extend the above to a transformation that is still uniquely invertible (Ziegler & Rush, 2019):

  $$c(x; \theta) = ax + b + \frac{c}{1 + (dx + h)^2}, \tag{4.17}$$

  for $\theta = [a, b, c, d, h] \in \mathbb{R}^5$ with $a > \frac{9}{8\sqrt{3}}|c|d$ and $d > 0$. The inverse is obtained by solving a third degree polynomial equation.

- **SOS polynomial flows** (Jaini et al., 2019):

  $$c(x; \theta) = \int_0^x \sum_{\kappa=1}^{k} \left(\sum_{l=0}^{r} a_{l,\kappa} u^l \right)^2 du + t. \tag{4.18}$$

  Here, $\theta = [t; (a_{l,\kappa})_{l,\kappa}] \in \mathbb{R} \times \mathbb{R}^{rk}$. Since the integrand is positive, the solution to the integration is invertible. The inverse can be found using bisection.

- **Flow++** (Ho et al., 2019):

  $$c(x; \theta) = s\sigma^{-1}\left(\sum_{j=1}^{K} \pi_j \sigma\left(\frac{x - \mu_j}{\sigma_j} \right) \right) + t. \tag{4.19}$$

Here, $\theta = [s; t; (\pi_j, \mu_j, \sigma_j)_{j=1}^K] \in \mathbb{R}_+ \times \mathbb{R} \times (\mathbb{R} \times \mathbb{R} \times \mathbb{R}_+)^K$ and σ is the logistic function. Again, it can be inverted using bisection.

- **Spline flows** in the form of piecewise-linear, monotone quadratic (Müller et al., 2019), cubic (Durkan et al., 2019), and rational quadratic (Durkan et al., 2019) splines. Here, c is a spline of the corresponding type, parameterized by knots θ. For these splines, analytically tractable inverses are known.
- **Neural autoregressive flow** (Huang et al., 2018) parameterize $c(x; \theta)$ by a feed-forward neural network, which can be shown to be bijective if all weights are positive and all activation functions are strictly monotone. Again, it can be inverted using bisection.
- **Unconstrained monotonic neural networks** (Wehenkel & Louppe, 2019) parameterize $c(x; \theta)$ as the integral of a neural network with positive output:

$$c(x; \theta) = \int_0^x c(u; \theta) du + t. \tag{4.20}$$

This integral is computed numerically, and it can be inverted using bisection. The volume change can be efficiently computed via the Leibniz integral rule.

4.3.2. High-dimensional invertible functions

Gaussianization layer Chen & Gopinath (2000) propose building a high-dimensional invertible block by applying a one-dimensional function $c(x; \theta_i)$ to each dimension $i = 1, \ldots, D$ separately:

$$\tilde{x} = f_{\text{gzn}} \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix} = \begin{pmatrix} c(x_1; \theta_1) \\ \vdots \\ c(x_D; \theta_D) \end{pmatrix}. \tag{4.21}$$

This is easy to invert by applying $x_i = c^{-1}(z_i; \theta_i)$ to each dimension individually. Since $\partial_i c(x_j; \theta_j) = 0$ for $i \neq j$, the Jacobian is diagonal and the determinant of the block reads:

$$\log |f'_{\text{gzn}}(x)| = \sum_{i=1}^{D} \log |c'(x_i; \theta_i)|. \tag{4.22}$$

This requires computing the derivative c', which can be computed analytically or via automatic differentiation.

Coupling layer Dinh et al. (2015, 2017) proposed to make Gaussianization layers more expressive by adding dependencies between dimensions. Just like the Gaussianization layers above, *coupling layers* apply one-dimensional invertible functions, but with a twist: the parameters of the one-dimensional transforms of the second half of the dimensions $a = x_{D/2+1,\ldots,D}$ (*active dimensions*) are predicted by a function $\psi(b)$ called the *conditioner* that takes the first half of dimensions $b = x_{1,\ldots,D/2}$ (*passive dimensions*) as input:

$$\tilde{x} = f_{\text{cpl}} \begin{pmatrix} x_1 \\ \vdots \\ x_{D/2} \\ x_{D/2+1} \\ \vdots \\ x_D \end{pmatrix} = \begin{pmatrix} c(x_1; \theta_1) \\ \vdots \\ c(x_{D/2}; \theta_{D/2}) \\ c(x_{D/2+1}; \psi_1(x_{1\ldots D/2})) \\ \vdots \\ c(x_D; \psi_{D/2}(x_{1\ldots D/2})) \end{pmatrix}. \tag{4.23}$$

This allows dimensions to interact between one another, since the transformation of x_i for $i > D/2$ can depend on the value of all $x_{1\dots D/2}$:

$$\frac{\partial}{\partial x_j} c(x_i; \psi_i(x_{1\dots D/2})) = \frac{\partial c}{\partial \psi_i} \frac{\partial \psi_i}{\partial x_j}, \qquad \text{for } j = 1 \dots D/2. \tag{4.24}$$

Throughout this thesis, we will for simplicity assume that the dimension D is even. If D is odd, use $\lfloor D/2 \rfloor$ or $\lceil D/2 \rceil$ active dimensions and the remaining dimensions are passive.

Inversion of a coupling layer takes three steps: (1) Compute $x_{1\dots D/2}$ from $z_{1\dots D/2}$, (2) determine $\psi_{1\dots D/2}(x_{1\dots D/2})$, and (3) use these parameters to obtain $x_{D/2+1\dots D}$ from $z_{D/2+1\dots D}$.

Computing $\log |f'_{\text{coupling}}(x)|$ is still feasible since the Jacobian is triangular, i.e. $\partial_j z_i(x) = 0$ if $j > i$, so that we find:

$$\log |f'_{\text{cpl}}(x)| = \sum_{i=1}^{D/2} \log |c'(x_i; \theta_i)| + \sum_{i=1}^{D/2} \log |c'(x_{i+D/2}; \psi_i(x_{1\dots D/2}))|. \tag{4.25}$$

Usually, the conditioning is implemented via a single feed-forward neural network $\psi_\varphi : \mathbb{R}^{D/2} \to \mathbb{R}^{D/2}$ with parameters φ that takes the passive dimensions as input and jointly predicts the transformation parameters of all active dimensions. Together, the parameters transformations of the passive dimensions $\theta_{1\dots D/2}$ and the parameters of the conditioner φ form the parameters of the coupling layer.

It is common to choose the identity as the transform for the first half of dimensions $i = 1, \dots, D/2$. This is consistent with our theory on the convergence rate of invertible architectures for the special case of Gaussian data in chapter 6, suggesting that modeling dimensions individually is outperformed by modeling dependencies.

A similar scheme is used in Feistel cipher (Feistel, 1973), applying the coupling structure to encryption. Each round of the cipher splits data in two blocks. The first block fed together with a secret key into a "round function" that produced a bit pattern of the same length as the second block. This pattern is combined via XOR with the second block to replace the second block in the output of the round. The first block is passed through as is. The parameters of the neural network $\theta(x_{1\dots D/2})$ correspond to the secret key of the Feistel cipher.

Another variant uses the identity transform for the first half of dimensions and additionally transforms all dimensions with an affine-linear layer called the activation normalization (ActNorm), which reportedly stabilizes training (Kingma & Dhariwal, 2018). ActNorm rescales and shifts each dimension:

$$f_{\text{act}}(x) = r \odot x + u, \tag{4.26}$$

given parameters $r \in \mathbb{R}^D_+$ and $u \in \mathbb{R}^D$.

Recursive coupling layers The Hierarchical Invertible Neural Transport (HINT) uses a recursive scheme to model more dependencies between dimensions in a single block (Kruse et al., 2021). The idea is to concatenate two additional coupling layers, one restricted to the passive dimensions (i.e. the second half of the passive dimensions is transformed conditioned on the first half) and the other to the active dimensions (i.e. the second half of dimensions is transformed conditioned on the first half). This is repeated by again using recursive couplings for the additional coupling layers. Each hierarchical level can be executed in parallel on the GPU.

Autoregressive layer We can build even more expressive invertible functions by letting the transformations of later dimensions depend on all previous dimensions:

$$f_{\text{ar}}(x)_i = c(x_i, \theta_i(x_{<i})). \tag{4.27}$$

One can show that if $c(x_i, \theta_i(x_{<i}))$ are arbitrary diffeomorphisms, then any continuous distribution $p(x)$ can be constructed by a single autoregressive layer. This is called a Knothe-Rosenblatt rearrangement (Rosenblatt, 1952; Knothe, 1957) and it is unique given data $p(x)$ and latent distributions $p(z)$.

While being expressive, autoregressive layers come with an important disadvantage: Inverting $f_{\text{ar}}^{-1}(\tilde{x})$ is sequential in the dimensions, since $\theta_i(x_{<i})$ requires the values of $x_{<i}$ to be known. Gaussianization layers can invert all dimensions in parallel since they are independent, and coupling layers require only two steps.

Rotation layer If several Gaussianization or coupling layers are repeated, the transformation would be strictly limited in expressivity as all respectively a subset of the dimensions do not get information about one another. This means that they would be modeled as independent by p_θ; for the Feistel cipher, it would mean that the first part of the data would not be encrypted. This is unrealistic for general data, which motivates prepending the above layers with rotation layers that mix dimensions before each layer:

$$f_{\text{rot}}(x) = Qx, \tag{4.28}$$

where $Q \in SO(D) \subset \mathbb{R}^{D\times D}$ is a matrix with orthonormal columns/rows and positive determinant.

On this choice of matrices, note that we could, in principle, use an arbitrary invertible matrix $M \in \mathbb{R}^{D\times D}$ with $\det M > 0$. However, any invertible matrix M can be written in the form AQB, where A, B are diagonal and Q orthogonal. Absorbing A and B in the subsequent respectively previous layer, we are left again with an orthogonal matrix Q. Also, we could choose $Q \in O(D)$ by allowing $\det Q = -1$, but this has no effect since the latent distribution is point symmetric.

In the case of coupling layers, it is common to choose permutations instead of free-form rotations. They are cheaper to sample for initializing the network and execute, avoiding a matrix-vector product. In fact, often just active and passive dimensions are swapped by the rotation layer. By (Koehler et al., 2021, Theorem 2), a constant number of coupling layer together with such swapping rotation layers is enough to represent any invertible linear transform, indicating that these variants are equivalent. For Gaussianization layers, free-form rotations are necessary.

Invertible block Combining a Gaussianization or coupling with a rotation layer yields an *invertible block*. An *invertible neural network* is then composed of several of the above blocks. We analyze the expressivity of the resulting architectures in chapters 5 and 6.

Neural ODEs An orthogonal approach is to parameterize an ordinary differential equation (ODE) with a neural network (Chen et al., 2018b):

$$\frac{\partial}{\partial t} x(t) = f_\theta(x(t), t). \tag{4.29}$$

By integrating the ODE from $t = 0$ (data space) to 1 (latent space) it can be used as an invertible neural network. Neural ODEs do not require the special structures introduced by Gaussianization or coupling blocks, allowing for more flexible architectures such as graph neural networks (Köhler et al., 2020) incompatible with coupling blocks. However, they come with high inference compute costs, as the ODE has to be solved using a numerical solver. We lift this constraint with free-form flows in chapter 7.

Training Neural ODEs can be achieved via equation (4.4), which also requires solving the underlying ODE, or by flow matching (Liu et al., 2023a; Lipman et al., 2023; Albergo & Vanden-Eijnden, 2023), which learns a fixed process between data and latent codes and does not require integrating the ODE at training time.

Invertible Residual Networks Another family of invertible neural networks consist of a feed-forward neural network in a skip connection:

$$x_{l+1} = x_l + f_{\theta_l}(x_l). \tag{4.30}$$

Here, the f_{θ_l} are restricted such that the transformation in each block is invertible by ensuring that the spectral radius $\rho(f'_{\theta_l}) < 1$ (Behrmann et al., 2019; Chen et al., 2019; Perugachi-Diaz et al., 2021). Again, inference is expensive since the inversion is based on iterative root finding. Free-form flows in chapter 7 lift this restriction by jointly learning a decoder network that samples in a single function evaluation.

4.3.3. Volume-preserving flows

Volume-Preserving Normalizing Flows or sometimes *incompressible flows* are a special variant of normalizing flows that have a constant Jacobian determinant $|f'_\theta(x)| = \text{const}$. This simplifies the change of variables formula in equation (4.1) above, where $C = |f'_\theta(x)|$:

$$p_\theta(x) = p(z = f_\theta(x))C. \tag{4.31}$$

Strictly speaking, "volume-preserving" is a misnomer when $C \neq 1$, but the term is commonly used also in this more general case and in lemma A.3, we show that non-unit volume change can be absorbed into a single scaling layer as previously implemented by Dinh et al. (2015).

Volume-preserving flows have been demonstrated to have useful properties in certain applications such as disentanglement (Sorrenson et al., 2019), or temperature-scaling in Boltzmann generators (Dibak et al., 2022) or to preserve volume in physical state-space (Toth et al., 2020). However, we show that the volume-preserving change of variables in equation (4.31) does not allow for universal normalizing flows regardless of the architecture in section 5.2.

For one-dimensional functions, a constant volume change implies that $f_\theta(x) = sx + t$ is linear. For multivariate functions, $f_\theta(x)$ can be nonlinear, but any volume change in one dimension must be compensated by an inverse volume change in the remaining dimensions.

Below, we list the ways to construct volume-preserving flows we are aware of. Our non-universality results theorems 5.2 and 5.3 and proposition 5.4 hold for all of them:

- **Nonlinear independent components estimation** (NICE) (Dinh et al., 2015) is a volume-preserving flow based on a coupling block:

$$c(x; \psi) = x + t. \tag{4.32}$$

  Here, $\psi = t \in \mathbb{R}$.

- **General Incompressible-flow Networks** (GIN) (Sorrenson et al., 2019) generalize NICE by allowing the individual dimensions to change volume, only the overall volume change is normalized:

$$c_i(x_i; \psi) = \frac{s_i}{\prod_{j=1}^{D/2} s_j} x_i + t_i, \tag{4.33}$$

  Here, $\psi = [s, t] \in \mathbb{R}_+^{D/2} \times \mathbb{R}^{D/2}$ is jointly predicted for all active dimensions and then normalized as above.

- **Neural Hamiltonian Flows** (Toth et al., 2020) parameterize a Neural ODE as a Hamiltonian system:

$$\frac{dq}{dt} = \frac{\partial \mathcal{H}}{\partial p}, \qquad \frac{dp}{dt} = -\frac{\partial \mathcal{H}}{\partial q} \tag{4.34}$$

The Hamiltonian $\mathcal{H}(p, q)$ is a real-valued function that is parameterized by a neural network. Its derivatives are obtained via automatic differentiation. The variant **Fixed-kinetic Neural Hamiltonian Flows** (Souveton et al., 2024) fixes the kinetic term of the Hamiltonian to $K(p) = \frac{1}{2}p^{\mathrm{T}}M^{-1}p$, where the positive definite matrix M is learned, and learns the potential $V(q)$ via a neural network to obtain $\mathcal{H}(p, q) = K(p) + V(q)$. The solution to the above ODE is always volume-preserving on $x = (p, q)$.

Note that some works employing volume-preserving flows such as Dibak et al. (2022); Souveton et al. (2024) consider *augmented flows* (Huang et al., 2020), where additional noise dimensions are padded $p(a|x)$ to the data distribution of interest $p(x)$. Then, the flow learns the joint distribution $p(x, a) = p(x)p(a|x)$. Depending on how $p(a|x)$ is constructed, this can positively or negatively impact the expressivity of the considered volume-preserving flow. For example, if $a \perp x$, then the joint distribution $p(x, a)$ has at least the same number of modes as $p(x)$, but the learned joint distribution $p_\theta(x, a)$ can only have a single mode by proposition 5.4, inducing a bias. To derive the universality in terms of KL, apply theorem 5.2 to the joint distribution at hand. On the positive side, Souveton et al. (2024) find that having $p(a|x) = \mathcal{N}(\mu(x), \sigma(x)^2 I)$ brings the obtained $p_\theta(x) = \int p_\theta(x, a)da$ closer to the target. This can be seen having an independent augmentation $a \sim \mathcal{N}(0, I) \perp x$ plus a single RealNVP coupling shifting and scaling the augmented dimensions. This effectively breaks the volume-preservation of the flow in the joint space. It is unclear, whether this trick removes all biases from the volume-preserving flow.

4.4. Kullback-Leibler divergence Pythagorean identities

To analyze the expressivity and efficiency of different normalizing flow architectures, it will be instructive to split the KL divergence $\mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(x)) = \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0, I))$ into several contributions. The following identities are due to our Draxler et al. (2022) as well as Cardoso (2003):

Theorem 4.1 (Pythagorean Identities, proof in appendix A.1.1)**.** *Given a probability density $p_\theta(z)$ with mean m and covariance Σ. Then, the Kullback-Leibler divergence to a standard normal distribution decomposes as follows:*

$$\mathcal{L} = \mathcal{D}_{KL}(p_\theta(z)\|\mathcal{N}(0, I)) = \underbrace{\mathcal{D}_{KL}(p_\theta(z)\|\mathcal{N}(m, \Sigma))}_{\text{non-Gaussianity } G[p_\theta(z)]} + \underbrace{\mathcal{D}_{KL}(\mathcal{N}(m, \Sigma)\|\mathcal{N}(0, I))}_{\text{non-Standardness } S(m,\Sigma)}, \tag{4.35}$$

and the non-Standardness again decomposes:

$$S(m, \Sigma) = \underbrace{\mathcal{D}_{KL}(\mathcal{N}(m, \Sigma)\|\mathcal{N}(m, \mathrm{Diag}(\Sigma)))}_{\text{Correlation } C(m,\Sigma)} + \underbrace{\mathcal{D}_{KL}(\mathcal{N}(m, \mathrm{Diag}(\Sigma))\|\mathcal{N}(0, I))}_{\text{Diagonal non-Standardness}}. \tag{4.36}$$

Also,

$$\mathcal{L} = \mathcal{D}_{KL}(p_\theta(z)\|\mathcal{N}(0, I)) = \underbrace{\mathcal{D}_{KL}(p_\theta(z)\|p(x_1)\cdots p(x_D))}_{\text{Mutual Information } I[p_\theta(z)]} + \sum_i \underbrace{\mathcal{D}_{KL}(p_\theta(z_i)\|\mathcal{N}(0, 1))}_{\text{marginal loss } \mathcal{L}[p_\theta(z_i)]}. \tag{4.37}$$

The mutual information $I[p_\theta(z)]$ is also called the *dependence* (Cardoso, 2003). The non-Gaussianity (Cardoso, 2003) is also called the *negentropy* (Comon, 1994). Note how equation (4.37) generalizes equation (4.36) to non-Gaussian distributions.

These identities are called Pythagorean identities in analogy to the classical Pythagorean theorem on inner product spaces that $\|a + b\|^2 = \|a\|^2 + \|b\|^2$ when a and b are orthogonal. They are special cases of a general result from information geometry (Amari & Nagaoka, 2007, Theorem 3.8). We use

them because different function classes are limited in which parts of the loss they can affect. For example, an *affine-linear* transport map $f_\theta(x) = Ax + b$ only affects the non-Standardness S, but not the non-Gaussianity G. This has already been argued by (Comon, 1994), and we provide an explicit proof in appendix A.7.1:

Lemma 4.2 (Proof in appendix A.1.2)**.** *Given a probability density $p(x)$ and an affine-linear function*

$$z = f_\theta(x) = Ax + b \tag{4.38}$$

for some invertible $A \in \mathbb{R}^{D\times D}$ and $b \in \mathbb{R}^D$. Then:

$$G[p_\theta(z)] = G[p(x)]. \tag{4.39}$$

Similarly, an element-wise function only affects the marginal losses, but not the mutual information:

Lemma 4.3. *Given a probability density $p(x)$ and an element-wise homeomorphism*

$$z = f_\theta(x) = \begin{pmatrix} f_{\theta,1}(x_1) \\ \vdots \\ f_{\theta,D}(x_D) \end{pmatrix} \tag{4.40}$$

Then:

$$I[p_\theta(z)] = I[p(x)]. \tag{4.41}$$

See for example the appendix of (Kraskov et al., 2004) for a proof of this statement.

5. Distributional universality of normalizing flows

What probability distributions can a given generative model architecture represent? In this chapter, we present a novel theoretical framework for understanding the expressive power of existing normalizing flows.

We first analyze volume-preserving flows in section 5.2. They have a restricted change-of-variables (equation (4.31) instead of equation (4.1)). While this form provides advantages in some applications, we show that they are not universal in terms of what distributions they can learn. We propose a fix that can be applied after training to recover universality.

We then enhance the proofs showing universality of coupling flows with non-volume-preserving coupling functions in section 5.4. Despite their prevalence in scientific applications, a comprehensive understanding of coupling flows remains elusive due to their restricted architectures. Existing theorems fall short as they require the use of arbitrarily ill-conditioned neural networks, limiting practical applicability. We propose a new distributional universality theorem for well-conditioned coupling-based normalizing flows such as RealNVP (Dinh et al., 2017) that overcomes problems of previous constructions. This result is based on an analysis of the universality of an individual affine coupling block in section 5.3.

Our results support the general wisdom that affine and related couplings are expressive and in general outperform volume-preserving flows, bridging a gap between empirical results and theoretical understanding.

5.1. Distributional universality

The question about whether a generative model can learn arbitrary distributions $p(x)$ stands in close analogy to the classical universal approximation theorems that show that fully-connected neural networks can approximate arbitrary continuous functions (Cybenko, 1989; Hornik, 1991). Due to the nature of neural networks, we cannot hope for generative model based on them to *exactly* represent a given $p(x)$, that is $p(x) = p_\theta(x)$ everywhere, just like for regression: Here, a neural network with ReLU activations always models piecewise linear functions, and as such it can never *exactly* regress a parabola $y = x^2$. However, for every finite value of $\epsilon > 0$ and given more and more linear pieces, it can follow the parabola ever so closer, so that the average distance between x^2 and $f_\theta(x)$ vanishes: $\mathbb{E}_{x\sim p(x)}[|x^2 - f_\theta(x)|^2] < \epsilon$. To characterize the expressivity of a class of neural networks, it is thus instructive to call a class of networks universal if the error between the model and any continuous target function can be reduced arbitrarily.

In terms of representing distributions $p(x)$, the following definition captures universality of a class of model distributions, similar to Teshima et al. (2020a, Definition 3):

Definition 5.1. *A set of probability distributions $\mathcal{P}$ is called a* distributional universal approximator *if for every possible target distribution $p(x)$ there is a sequence of distributions $p_n(x) \in \mathcal{P}$ such that $p_n(x) \xrightarrow{n\to\infty} p(x)$.*

The formulation of universality as a convergent series is useful as it (i) captures that the distribution in question $p(x)$ may not lie in $\mathcal{P}$, and (ii) the series index n usually reflects a hyperparameter of

the underlying model corresponding to computational requirements (for example, the depth of the network).

We have left the exact definition of the limit "$p_n(x) \xrightarrow{n\to\infty} p(x)$" open as we may want to consider different definitions of convergence. The existing literature on affine coupling-based normalizing flows considers weak convergence (Teshima et al., 2020a) respectively convergence in Wasserstein distance (Koehler et al., 2021) (the construction in the most recent proof by Koehler et al. (2021) is fundamentally tied to this relatively weak convergence metric as it constructs a volume-preserving flow, see section 5.4.2). Many metrics of convergence have been proposed, see Gibbs & Su (2002) for a systematic overview.

In this chapter, we consider continuous target distributions $p(x)$ that have infinite support and finite moments, which covers distributions of practical interest.

5.2. Non-universality of volume-preserving flows

We first consider volume-preserving normalizing flows. They are based on invertible neural networks with constant Jacobian determinant $|f'_\theta(x)| = \text{const}$, see section 4.3.

We contribute:

- We first show that **volume-preserving flows are not universal in KL divergence**.
- We propose how universality can be recovered by adding a single non-volume-preserving block which learns a one-dimensional function.

We are not aware of previous work considering the universality of volume-preserving flows. The closest result known to us states that Hamiltonian Monte Carlo (HMC) requires resampling the momentum distribution in every step to sample from the correct target distribution (Toth et al., 2020).

The rest of this section is adapted from Draxler et al. (2024b).

Let us derive the minimizer of the loss $\mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(x))$ for a volume-preserving flow by using the volume-preserving change-of-variables in equation (4.31), first in the special case of $C = 1$:

$$\mathcal{L} = \int p(x) \log \frac{p(x)}{p_\theta(x)} dx = -H[p(x)] - \int p(x) \log p(z = f_\theta(x)) dx. \tag{5.1}$$

Only the last term depends on f_θ. To derive the minimizer, consider the data $p(x)$ and latent distribution $p(z)$ on a regular grid over $\mathbb{R}^D$ with some spacing $a > 0$. Then, define a volume-preserving flow with $C = 1$ that permutes the grid cells $B_i \mapsto B_{f(i)}$ (within the cells, keep the relative positions). Then, discretize the above integral on the grid by approximating the latent probability by the average density in each cell, that is $p(z) \approx \frac{1}{a^D} p(B_i : z \in B_i)$:

$$-\int p(x) \log p(z = f_\theta(x)) dx \approx -\sum_i p(x \in B_{s_x(i)}) \log(p(z \in B_{f(i)})/|a^D|). \tag{5.2}$$

This is minimized by a bijective $f^* : \mathbb{N} \to \mathbb{N}$ that permutes the grid cells such that the cell with the highest probability $p(x \in B_i)$ in the data space aligns with the cell with the highest (logarithmic) probability in latent space, and so on:

$$f^*(i) = s_z(s_x^{-1}(i)), \tag{5.3}$$

where $s_v(i)$ is a sorting of the grid cells, determined by probability mass $p(v \in B_i)$ for $v = x$ respectively $v = z$.

The following theorem makes the above argument continuous and determines the optimal volume change $C > 0$:

Theorem 5.2 (Best possible volume-preserving flow, proof in appendix A.2.1)**.** *Given a continuous bounded probability density $p(x)$ with $(D-1)$-dimensional level sets. Then, the minimal achievable KL divergence by a volume-preserving normalizing flow $p^*_\theta(x)$ whose underlying map f^*_θ is continuous almost everywhere and with a standard normal latent distribution reads:*

$$\mathcal{D}_{KL}(p(x)\|p^*_\theta(x)) = \mathcal{D}_{KL}(p^*(z)\|\mathcal{N}(0, |\Sigma_{p^*(z)}|^{\frac{1}{D}} I)), \tag{5.4}$$

where $p^*(z)$ is constructed by decreasingly sorting the probability densities $p(x)$ from the origin with unit volume change, and $\Sigma_{p^*(z)}$ is its covariance matrix. This minimal loss is achieved for $C = |\Sigma_{p^*(z)}|^{-\frac{1}{2D}}$.

The optimal $p^*(x)$ and its latent counterpart $p^*(z)$ are constructed by sorting both the data and latent space by density and progressively assigning regions of decreasing density to each other (see lemma A.1 in appendix A.2.1). The theorem considers the more general class of almost everywhere continuous volume-preserving flows, and everywhere continuous flows can only achieve at most as good. Figure 5.1 shows how this optimal distribution $p^*(x)$ differs from the target $p(x)$ for a bimodal toy distribution in 2D.

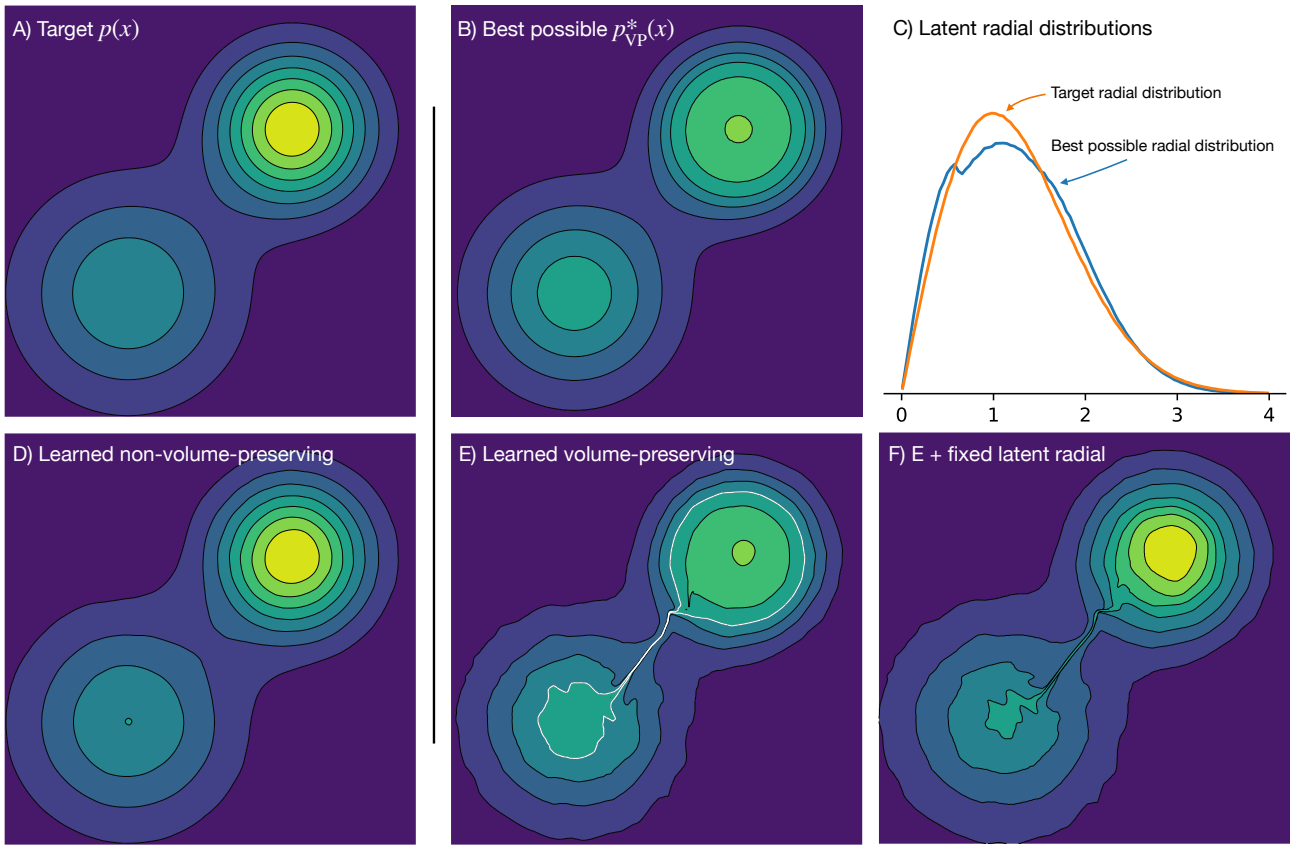


Figure 5.1.: We reveal two limitations of volume-preserving flows: First, a 2D bimodal distribution *(A)* cannot be represented by a volume-preserving flow, the theoretic optimum predicted by theorem 5.2 assigns wrong densities to both modes *(B)*. This is because the radial part of the latent distribution $p^*(z)$ does not match the radial part of the standard normal *(C)*. In practice, learning a volume-preserving flow comes very close to the biased solution *(E)*. A normalizing flow with variable Jacobian determinant does not have this issue *(D)*. Our proposed fix corrects the densities at the modes *(F)* by correcting the latent radials, see appendix A.2.2. Second, since the flow is continuous in practice, it cannot represent multi-modal distributions by proposition 5.4, but a vanishing density bridge connecting the modes remains *(E, white level set)*.

The following result formalizes that volume-preserving flows are not universal:

Theorem 5.3 (Non-universality of volume-preserving flows, proof in appendix A.2.3)**.** *The family of volume-preserving normalizing flows is* not *a universal distribution approximator under KL divergence.*

To see this, we choose a concrete $p(x)$, compute the optimal latent distribution $p^*(z)$ and use theorem 5.2 to derive the best possible KL divergence of a volume-preserving flow.

How can we recover universality? The construction underlying theorem 5.2 shows a clear path to construct a universal volume-preserving flow: The best achievable latent distribution $p^*(z)$ (the push-forward of $p(x)$ through f^*) is rotationally symmetric due to the sorting procedure. Now transform both $p^*(z)$ and the target standard normal $p(z)$ into hyperspherical coordinates (r, Ω). As both distributions are rotationally symmetric, only their radial parts $p^*(r)$ and $p(r)$ need to be matched, which can be achieved via the addition of a single one-dimensional *non*-volume-preserving transformation (see appendix A.2.2). As this fix is one-dimensional, unique, and can be applied after training, we think that it is compatible with retaining beneficial properties of volume-preserving flows.

Figure 5.1 also reveals a shortcoming of volume-preserving flows as they are practically implemented: There is a thin bridge of density between the modes with roughly constant height, so that the lower mode in the ground truth is not a local maximum of the learned density. The reason is that flows are implemented as continuous invertible functions (as opposed to theorems 5.2 and 5.3, which only required almost everywhere continuous functions). This makes the learned distribution $p_\theta(x)$ inherit the mode structure of the latent $p(z)$:

Proposition 5.4 (Mode preservation of volume-preserving flows, proof in appendix A.2.4)**.** *A normalizing flow $p_\theta(x)$ based on a volume-preserving diffeomorphism $f_\theta(x)$ has the same number of modes as the latent distribution $p(z)$.*

The proof uses that diffeomorphisms map open sets to open sets, and thus the neighborhoods of density maxima in the latent space remain neighborhoods of density maxima in the data space. Note that the thin bridge connecting the modes can be made arbitrarily small by an expressive enough volume-preserving flow, so that the shortcoming in proposition 5.4 does not manifest in a bias in the KL divergence in addition to theorem 5.2.

Together, we identify a fundamental limitation for applications based on volume-preserving flows. It explains why RealNVP significantly outperforms NICE in practice (Dinh et al., 2017). Work using volume-preserving flows must take this limited expressivity and the resulting biases in the learned distributions into account. In section 5.4.2, we show that this problem also applies to the most recent universality proof for coupling-based normalizing flows by Koehler et al. (2021).

5.3. Expressivity of a single affine coupling block

In this section, we derive the minimizer of the loss if a normalizing flow consists of an isolated affine coupling block. It is clear that a single block cannot fit arbitrary distributions, even in approximation. However, our results yield several valuable insights:

- The minimizer of an isolated affine coupling block normalizes the first two moments of each active dimension conditioned on all passive dimensions, that is of $p(a_i|b)$.
- It terms of the loss, the amount by which the loss is improved can be identified to a KL divergence in the Pythagorean identity in theorem 4.1.
- We find that the rotation layer in an affine coupling block plays an important role in bringing the performance of iterative training closer to end-to-end training.

In addition, we will heavily use the above results in our analysis of deep coupling flows in section 5.4 and their convergence rate in section 6.3. We present illustrative examples and questions for further research with each result.

The remainder of this section is an adapted version of Draxler et al. (2020).

5.3.1. Related work

The connection between affine transformations and the first two moments of a distribution is well-known in the Optimal Transport literature. When the function space of an Optimal Transport (OT) problem with quadratic ground cost is reduced to affine maps, the best possible transport matches mean and covariance of the involved distributions (Tabak & Trigila, 2018). In the case of conditional distributions, affine maps become conditional affine maps (Trigila & Tabak, 2016). We show such maps to have the same minimizer under maximum likelihood loss (KL divergence) as under OT costs.

It has been argued before that a single coupling or autoregressive block (Papamakarios et al., 2019) can capture the moments of conditional distributions. This is one of the motivations for the SOS flow (Jaini et al., 2019), based on a classical result on degree-3 polynomials by Fleishman (1978). However, they do not make this connection explicit. We can give a direct correspondence between the function learned by an affine coupling and the first two moments of the distribution to be approximated.

Rotations in affine flows are typically chosen at random at initialization and left fixed during training (Dinh et al., 2015, 2017). Others have tried training them via some parameterization like a series of Householder reflections (Putzky & Welling, 2019). The stream of work most closely related to ours explores the idea to perform layer-wise training. This allows an informed choice of the rotation based on the current estimate of the latent normal distribution. Most of these works propose to choose the least Gaussian dimensions as the active subspace (Bigoni et al., 2019; Meng et al., 2019). We argue that this is inapplicable to affine flows due to their limited expressivity when the passive dimensions are not informative. To the best of our knowledge, our approach is the first to take the specific structure of the coupling layer into account and derive a tight lower bound on the loss as a function of the rotation.

5.3.2. KL divergence minimizer

We now derive how much an infinitely expressive affine coupling block as given in section 4.3 can reduce the loss. By infinitely expressive, we mean that we consider arbitrary functions $s(b), t(b)$ as the conditioner ψ instead of practically realizable neural networks. We analyze finitely sized neural networks in section 5.4.

We choose to fix the transformation of the first half of dimensions $i = 1, \dots, D/2$ to the identity (see section 4.3.2) in this section.

We start by deriving the exact form of the maximum likelihood loss in equation (4.4) for such an isolated affine coupling layer. Inserting the above definitions, we find the following optimization problem for the parameters of the isolated coupling block with fixed rotation Q, where the degrees of freedom of the coupling layer are captured by the conditioner functions $s(b), t(b)$:

$$\min_{\theta=[s(b),t(b)]} \mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(x)) = \min_{\theta=[s(b),t(b)]} \mathbb{E}_{b,a\sim Q_\sharp p}\left[\frac{1}{2}\|b\|^2 + \frac{1}{2}\|a \odot s(b) + t(b)\|^2 - \sum_{i=1}^{D/2} \log s_i(b)\right]. \quad (5.5)$$

As before, we collected the unchanged variables as the *passive dimensions* $b = (Qx)_{1,\dots,D/2}$ and the variables transformed by the coupling layers as $a = (Qx)_{D/2+1,\dots,D}$ (*active dimensions*).

We now derive the minimizer of this loss in the case:

Lemma 5.5 (Optimal single affine coupling, proof in appendix A.3.1)**.** *Given a probability density $p(x)$ with finite first and second moments, and a single affine coupling layer f_{cpl} after a fixed rotation Q so*

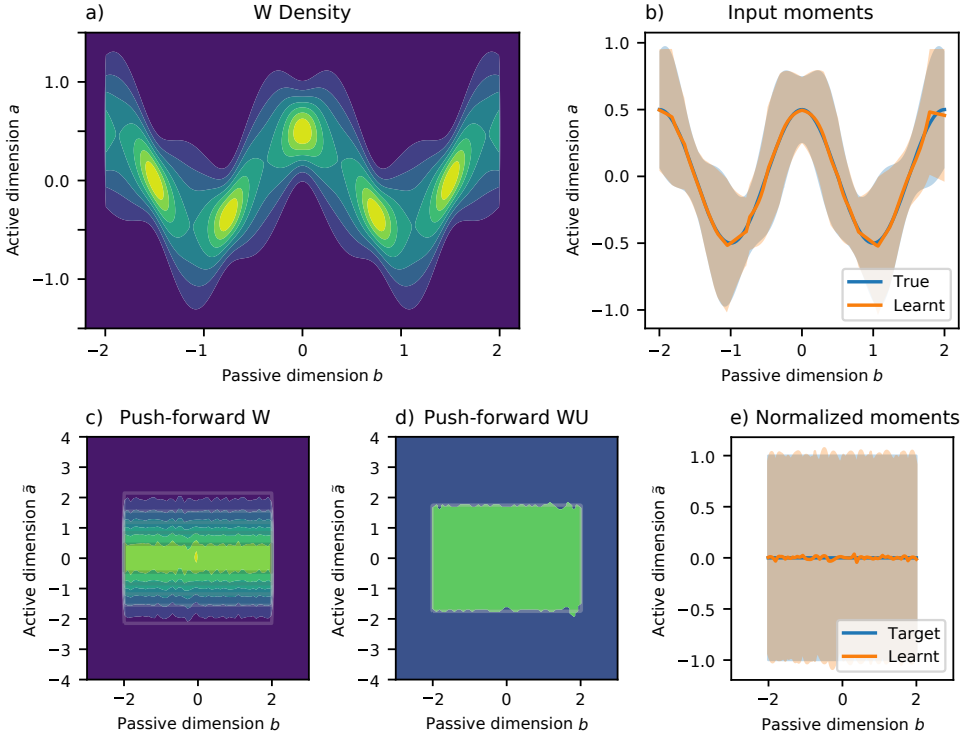


Figure 5.2.: *(a)* W density contours. *(b)* The conditional moments are well approximated by a single affine layer. *(c, d)* The learned push-forwards of the W (example 5.6) and WU (example 5.7) densities remain normal respectively uniform distributions. *(e)* The moments of the transported distributions are close to zero mean and unit variance, shown for the layer trained on the W density.

that $(b, a) = Qx$*. Then, equation* (5.5) *is uniquely minimized by the following scaling and translations as a function of the passive dimensions* b*:*

$$s_i^*(b) = \frac{1}{\sqrt{\mathrm{Var}_{a_i|b}[a_i]}} =: \frac{1}{\sigma_i(b)}, \tag{5.6}$$

$$t_i^*(b) = -\mathbb{E}_{a_i|b}[a_i] s_i(b) =: -\frac{m_i(b)}{\sigma_i(b)}. \tag{5.7}$$

Here we denote by $m_i(b)$ and $\sigma_i(b)$ the mean and standard deviation of a_i given b. The above result means that the conditioner $\psi(b) = [s^*(b), t^*(b)]$ learns the first two moments of each conditional distribution $p(a_i|b)$ for each value of b and the coupling acts on the data by normalizing these moments:

$$\tilde{a}_i = a_i \cdot s_i(b) + t_i(b) = \frac{1}{\sigma_i(b)} \cdot (a_i - m_i(b)). \tag{5.8}$$

This shifts the mean of $p(a|b)$ to zero and rescales the standard deviations to one:

$$\mathbb{E}_{\tilde{a}_i|\tilde{b}}[\tilde{a}_i] = 0, \quad \mathrm{Var}_{\tilde{a}_i|\tilde{b}}[\tilde{a}_i] = 0. \tag{5.9}$$

We derive this by optimizing the optimal value of equation (5.5) for each value of b separately.

Example 5.6. Consider a distribution where the first variable b is uniformly distributed on the interval $[-2, 2]$. The distribution of the second variable a is normal, but its mean $m(b)$ and standard deviation

$\sigma(b)$ are varying depending on b:

$$p(b) = \mathcal{U}([-2, 2]), \quad p(a|b) = \mathcal{N}(m(b), \sigma(b)). \tag{5.10}$$

$$m(b) = \frac{1}{2}\cos(\pi b), \quad \sigma(b) = \frac{1}{8}(3 - \cos(8\pi/3\, b)). \tag{5.11}$$

We call this distribution "W density". It is shown in figure 5.2a.

From lemma 5.5, we expect that after the coupling the passive dimensions are unchanged and the active dimensions are now independent of b and follow a standard normal distribution:

$$p_\theta(\tilde{b}) = p(b), \quad p_\theta(\tilde{a}|\tilde{b}) = \mathcal{N}(0, 1) = p_\theta(\tilde{a}). \tag{5.12}$$

Figure 5.2 confirms that this is approximated empirically if we train a single affine coupling block by minimizing the equation (5.5), fixing $Q = I$. As hyperparameters, we choose a joint neural network for s, t with one hidden layer with 256 hidden units. The output of the network is split to form s and t. The learning rate is 10^{-1} with a learning rate decay with factor 0.9 every 100 epochs. We train for 4096 epochs with 4096 i.i.d. samples from $p(x)$ each using the Adam optimizer without weight decay.

We solve s, t in lemma 5.5 for the estimated mean $\hat{m}(b)$ and standard deviation $\hat{\sigma}(b)$ as predicted by the learned $\hat{s}$ and $\hat{t}$. Upon convergence of the model, they closely follow their true counterparts $m(b)$ and $\sigma(b)$ as shown in figure 5.2b.

Example 5.7. This example modifies the previous one to illustrate that the learned latent conditional density $p_\theta(\tilde{a}|\tilde{b})$ is not necessarily Gaussian at the minimum of the loss, but still, the moments are normalized.

We modify the W density from above to the "WU density" by replacing the conditional normal distribution $p(a|b)$ by a conditional uniform distribution with the same conditional mean $m(b)$ and standard deviation $\sigma(b)$ as before:

$$p(b) = \mathcal{U}([-2, 2]), \tag{5.13}$$

$$p(a|b) = \mathcal{U}([m(b) - \sqrt{3}\sigma(b), m(b) + \sqrt{3}\sigma(b)]). \tag{5.14}$$

One might wrongly believe that the KL divergence favors building a distribution that is *marginally* normal while ignoring the conditionals, for example that the resulting marginal over $\tilde{a}$ becomes normal $p_\theta(\tilde{a}) = \mathcal{N}(\tilde{a}; 0, 1)$.

Lemma 5.5 predicts the correct result, resulting in the following uniform push-forward density depicted in figure 5.2d:

$$p_\theta(\tilde{b}) = p(b), \quad p_\theta(\tilde{a}|\tilde{b}) = \mathcal{U}([-\sqrt{3}, \sqrt{3}]), \tag{5.15}$$

where the latter is the uniform distribution with unit standard deviation. Note how $p_\theta(\tilde{a}|\tilde{b})$ does not depend on $\tilde{b}$.

Knowing that a single affine layer learns the mean and standard deviation of $p(a_i|b)$ for each b, we can insert this minimizer into the KL divergence. This yields a tight lower bound on the loss after training. Even more, it allows us to compute a tight upper bound on the loss improvement by the layer, which we denote as $\Delta^*_{\text{affine}}(Q) \geq 0$.

Theorem 5.8 (Improvement by single affine layer, proof in appendix A.3.2)**.** *Given a probability density $p(x)$ with finite first and second moments, and a single affine coupling layer f_{cpl} after a fixed rotation Q so that $(b, a) = Qx$. Let f_{cpl} be the minimizer from lemma 5.5 and $\tilde{x} = f_{\text{cpl}}(Qx) = (\tilde{b}, \tilde{a})$. Then, the loss has the following minimal value:*

$$\mathcal{D}_{KL}(p_\theta(\tilde{x})\|p(z)) = \mathcal{D}_{KL}(p(b)\|\mathcal{N}(0, I)) + \mathbb{E}_{b\sim p(b)}\Big[G[p(a|b)] + C(m_{a|b}, \Sigma_{a|b})\Big] \tag{5.16}$$

$$= \mathcal{D}_{KL}(p(b)\|\mathcal{N}(0, I)) - \Delta^*_{\text{affine}}(Q). \tag{5.17}$$

Here, the non-Gaussianity $G[p(\cdot)]$ and correlation $C(m, \Sigma)$ are defined in theorem 4.1 and $m_{a|b}$ and $\Sigma_{a|b}$ are the mean and covariance of $p(a|b)$. The loss improvement can explicitly be computed from the conditional moments of $p(a|b)$:

$$\Delta^*_{\text{affine}}(Q) = \sum_{i=1}^{D/2} \mathbb{E}_b[S(m_i(b), \sigma_i(b))] = \frac{1}{2} \sum_{i=1}^{D/2} \mathbb{E}_b[m_i^2(b) + \sigma_i^2(b) - 1 - \log \sigma_i^2(b)]. \tag{5.18}$$

Here, $S(m, \sigma) = \mathcal{D}_{KL}(\mathcal{N}(m, \sigma) \| \mathcal{N}(0, 1))$ is the univariate non-Standardness like in theorem 4.1.

Theorem 5.8 says that the loss reduction by a single affine layer depends solely on the moments of the distribution of the active dimensions conditioned on the passive subspace. Higher order moments are ignored by this coupling design (as well as correlations between active dimensions, and the loss in the passive dimensions). Together with lemma 5.5, this paints the following picture of an affine coupling layer: It fits a Gaussian distribution to each conditional $p(a_i|b)$ and normalizes this Gaussian's moments. This gap constitutes the remaining KL divergence in equation (5.16). We will complement the above result with a more in-depth analysis on the reducible and remaining loss contributions in section 5.4.5 via the use of theorem 4.1.

To prove, insert the minimizer s, t from lemma 5.5 into equation (4.4). Then evaluate $\Delta^*_{\text{affine}}(Q) = \mathcal{D}_{\text{KL}}(p(x) \| p(z)) - \mathcal{D}_{\text{KL}}(p_\theta(\tilde{x}) \| p(z))$ to obtain the statement.

We now make the connection explicit, that a single affine layer can achieve zero loss on the active subspace if and only if the conditional distribution is Gaussian with diagonal covariance:

Corollary 5.9 (Exact representation by single affine coupling layer, proof in appendix A.3.3)**.** *If and only if $p(a|b)$ is normally distributed for all b with diagonal covariance, that is:*

$$p(a|b) = \prod_{i=1}^{D/2} \mathcal{N}(a_i | m_i(b), \sigma_i(b)), \tag{5.19}$$

a single affine block can reduce the KL divergence on the active subspace to zero:

$$\mathcal{D}_{KL}(p(x) \| p_\theta(x)) = 0. \tag{5.20}$$

This shows how limited a single-block normalizing flow is, in particular if compared to fully-connected neural networks, where a single hidden layer is enough for the classical universality proofs in the regression setting (Hornik, 1991).

Example 5.10. We revisit the examples 5.6 and 5.7 and confirm that the minimal loss achieved by a single affine coupling layer on the W-shaped densities matches the predicted lower bound. This is the case for both densities. Figure 5.3 shows the contribution of the conditional part of the KL divergence $\mathcal{D}_{\text{KL}}(p_\theta(\tilde{a}|b) \| p(z_2))$ as a function of b:

For the W density, the conditional $p(a|b)$ is normally distributed. This is the situation of corollary 5.9 and the remaining conditional KL divergence is zero. The remaining loss for the WU density is the non-Gaussianity of a uniform distribution with unit variance.

5.3.3. Determining the optimal rotation

The rotation Q of the isolated coupling layer determines the splitting into active and passive dimensions and the axes of the active dimensions (the rotation within the passive subspace only rotates the input into s, t and is therefore irrelevant). The bounds in theorem 5.8 explicitly depends on these choices and thus depend on the chosen rotation Q, as visualized in figure 5.4.

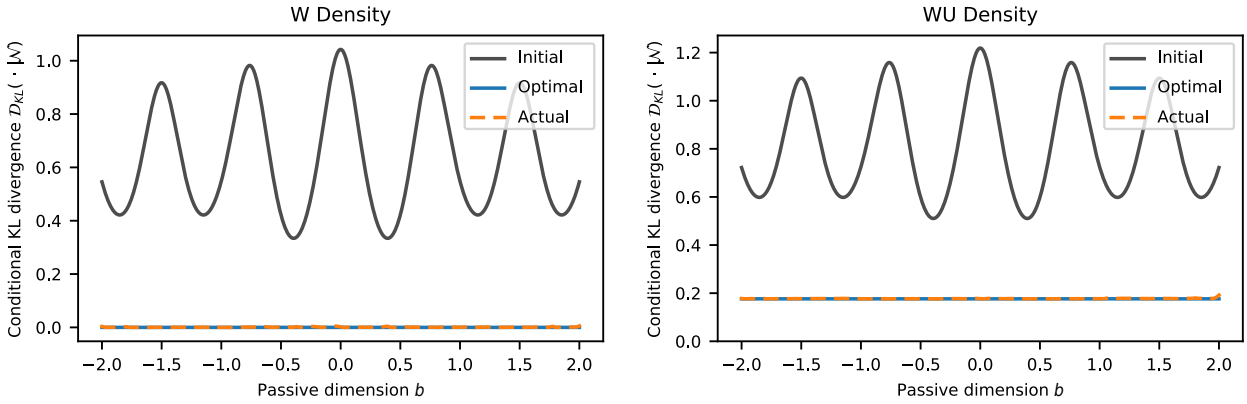


Figure 5.3.: Conditional KL divergence before *(gray)* and after *(orange)* training for W-shaped densities confirms lower bound *(blue, coincides with orange)*. The plots show the W density from example 5.6 *(left)* and the WU density from example 5.7 *(right)*.

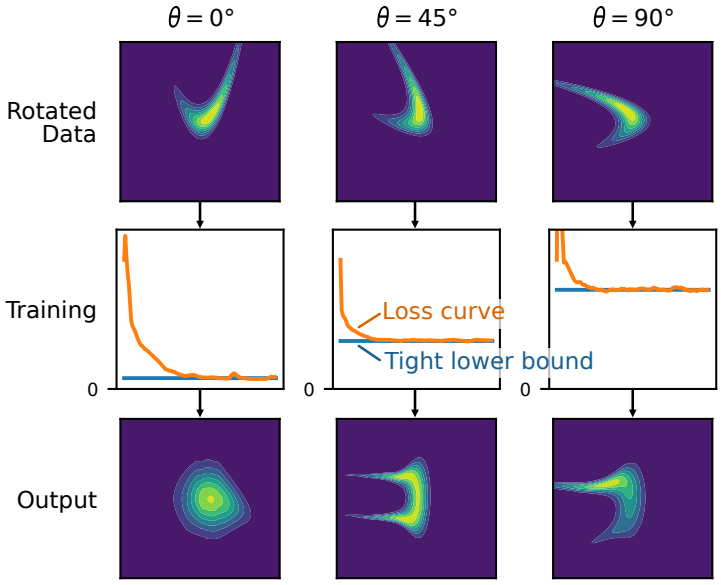


Figure 5.4.: An affine coupling layer pushes the input density closer to the latent standard normal. Its success depends on the rotation of the input *(top row)*. Theorem 5.8 yields a lower bound for the error that is actually attained empirically *(center row, blue and orange curves)*. The solution with the lowest error is clearly closest to standard normal *(bottom row, left)*.

This makes it natural to consider the loss improvement as a function of the rotation: $\Delta^*_{\text{affine}}(Q)$. When aiming to maximally reduce the loss with a single affine layer, one should choose the subspace maximizing this tight upper bound in equation (5.18):

$$\max_{Q \in SO(D)} \Delta^*_{\text{affine}}(Q). \tag{5.21}$$

We propose approximating this maximization by evaluating the loss improvement for a finite set of candidate rotations in algorithm 1 "Optimal Affine Subspace (OAS)"

Note that Step 5 requires approximating $\Delta^*_{\text{affine}}(Q)$ from samples, which involves estimation of 1D moments conditioned on high-dimensional p. In the regime of low $D/2$, one can discretize this by binning samples by their passive coordinate b. Then, one computes mean and variance empirically for each bin. We leave the general solution of equation (5.21) for future work.

Algorithm 1 Optimal Affine Subspace (OAS).

1: **Input**: $\mathcal{Q} = \{Q_1, \dots, Q_C\} \subset SO(D)$, $(x_j)_{j=1}^N$ i.i.d. samples from p.
2: **for** candidate $Q_c \in \mathcal{Q}$ **do**
3: Rotate samples: $y_j = Q_c x_j$.
4: **for** each active dimension $i = 1, \dots, D/2$ **do**
5: Use $(y)_{j=1}^N$ to estimate the conditional mean $m_i(b)$ and variance $\sigma_i(b)$ as a function of b. {Example implementation in example 5.11}
6: **end for**
7: Compute $\hat{\Delta}_{\text{affine}}(Q_c) := \frac{1}{2N} \sum_{j=1}^N \sum_{i=1}^{D/2} (m_i(b_j)^2 + \sigma_i(b_j)^2 - 1 - \log \sigma_i(b_j)^2)$ {equation (5.18)}.
8: **end for**
9: **Return**: $\arg\max_{Q_c \in \mathcal{Q}} \hat{\Delta}_{\text{affine}}(Q_c)$.

Example 5.11. Consider the following two-component 2D Gaussian Mixture Model:

$$p(x) = \frac{1}{2}\Big(\mathcal{N}([-\delta; 0], \sigma) + \mathcal{N}([\delta; 0], \sigma)\Big). \tag{5.22}$$

We choose $\delta = 0.95, \sigma = \sqrt{1 - \delta^2} = 0.3122...$ so that the mean is zero and the standard deviation along the first axis is one. We now evaluate the loss improvement $\Delta^*_{\text{affine}}(\theta)$ in equation (5.18) as a function of the rotation angle θ by which we rotate the above distribution:

$$(a, b) = Qx, \quad x \sim p(x). \tag{5.23}$$

Since we are considering a Gaussian mixture distribution, we can compute $m(b)$ and $\sigma(b)$ analytically for each b and then integrate numerically to obtain $\hat{\Delta}_{\text{affine}}(\theta)$. This will not be possible for applications where only samples are available. As a proof of concept, we employ the previously mentioned binning approach. It groups N samples from p by their b value into B bins. Then, we compute $m(b_i)$ and $\sigma(b_i)$ using the samples in each bin $i = 1, \dots, B$.

Figure 5.5 shows how much the loss can be improved by a single affine coupling layer as a function of the rotation angle θ, as determined from the numerical integration and the estimation from binning samples into $B = 32$ bins. Around $N = 256$ samples are sufficient for a good agreement between the analytic and empiric bound on the loss improvement and the corresponding maximizer angle.

In this 2D example, we find that the rotation influences how much a single coupling block reduces the loss. If we naively or by chance decide for $\theta = 90°$, the distribution is left unchanged.

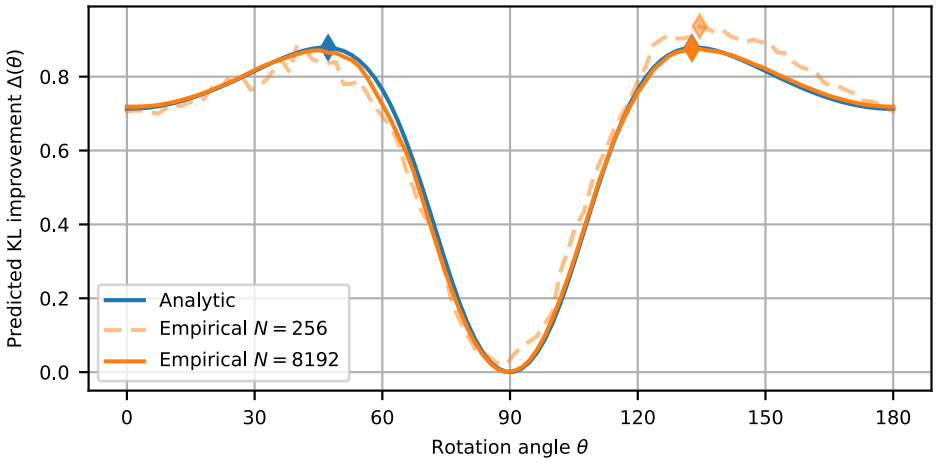


Figure 5.5.: Tight upper bound given by equation (5.18) for two-component Gaussian mixture as a function of rotation angle θ, determined analytically *(blue)* and empirically *(orange)* for different numbers of samples. The diamonds mark the equivalent outputs of the OAS algorithm 1.

5.3.4. Gap to universality

What do we need to make a single affine coupling block universal? Here, we identify one sufficient condition that pushes an input to the standard normal via single additional Gaussianization block.

A necessary step towards pushing a multivariate distribution to a normal distribution is making the dimensions independent of one another. In fact, theorem 4.1 shows that the loss $\mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(x))$ can be decomposed into two parts, where one of the two exactly measures the mutual information between all dimensions x_i.

Then, the residual to a global latent normal distribution can be solved with one sufficiently expressive 1D flow per dimension, pushing each distribution independently to a normal distribution. The following lemma shows for which data sets a single affine layer can make the active and passive dimensions exactly independent.

Lemma 5.12 (Independence by single layer, proof in appendix A.3.4)**.** *Given a distribution p and a single affine coupling layer f with a fixed rotation Q. Call $(a, b) = Qx$ the rotated versions of $x \sim p$. Then, the following are equivalent:*

1. *$\tilde{a} \perp b$ for $(\tilde{a}, \tilde{b}) = f_{\mathrm{cpl}}(a, b)$ minimizing the ML loss in equation (5.5),*
2. *there exists $n \perp b$ such that $a = f(b) + n \odot g(b)$, where $f, g : \mathbb{R}^{D/2} \to \mathbb{R}^{D/2}$.*

This result relates a data generation process to the previously derived minimizer. It suggests that affine coupling flows can learn distributions by reverting the above generating process, possibly by iteratively applying lemma 5.12. While mathematically precise, this iteration is trivial and does not capture well which distributions can be approximated in terms of some convergence metric. We will provide such a universality result in section 5.4.

Example 5.13. Consider again the W-shaped densities from the previous examples 5.6 and 5.7. After optimizing the single affine layer, the two variables $b, \tilde{a}$ are independent (compare figure 5.2c, d):

$$\text{Example 5.6: } \tilde{a} \sim \mathcal{N}(0, 1) \perp b, \tag{5.24}$$

$$\text{Example 5.7: } \tilde{a} \sim \mathcal{U}([-\sqrt{3}, \sqrt{3}]) \perp b, \tag{5.25}$$

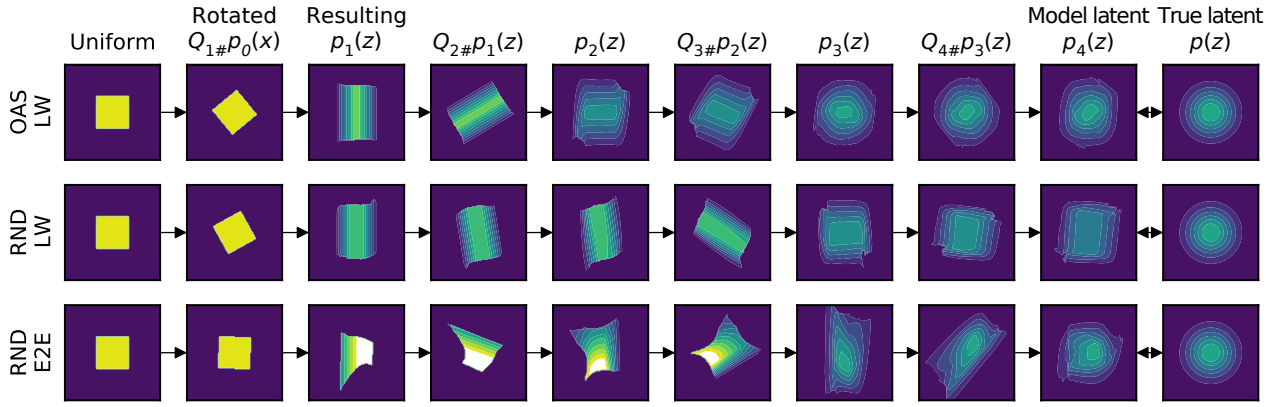


Figure 5.6.: Affine flow trained layer-wise "LW", using optimal affine subspaces "OAS" *(top)* and random subspaces "RND" *(middle)*. After a lucky start, the random subspaces do not yield a good split and the flow approaches the latent normal distribution significantly slower. End-to-end training "E2E" *(bottom)* chooses a substantially different mapping, yielding a similar quality to layer-wise training with optimal subspaces.

5.3.5. Layer-wise training

By learning only a single coupling block, we have modified the usual end-to-end training procedure of normalizing flows. While this allows optimizing for Q, the model cannot use potential symbiotic effects of a sequence of layers. In this section, we explore constructing a deep flow layer by layer using the optimal affine subspace (OAS) algorithm in algorithm 1. Each layer l being added to the flow is trained to minimize the residuum between the current push-forward $p_{l-1}(x_{l-1})$ and the latent $p(z)$. The corresponding rotation Q_l is chosen by maximizing $\hat{\Delta}_{\text{affine}}(Q_l)$ and the coupling layers are trained by gradient descent.

How does this ansatz compare to the quality of end-to-end affine flows? As an analytic answer is out of the scope of this work, we perform an ablation on toy data:

Example 5.14. We consider a uniform 2D distribution $p = \mathcal{U}([-1,1]^2)$. Figure 5.6 compares the flow learned layer-wise using to flows learned layer-wise and end-to-end, the latter two with fixed random rotations. Visually, our proposed layer-wise algorithm performs on-par with end-to-end training despite optimizing only the respective last layer in each iteration, and beats layer-wise with random rotations.

In appendix B.1.2, we provide examples on additional toy distributions.

5.3.6. Conclusion

In this section, we have derived what transformation an affine coupling block can learn if the conditioner neural network is replaced by an arbitrary continuous function. We find that the affine coupling block learns to normalize the first two conditional moments of each active dimension conditioned on all passive dimensions when the rotation layer is fixed. From the perspective of the Pythagorean identities in section 4.4, this removes the non-Standardness in each active dimension $p(a_i|b)$ from the overall loss.

We find that choosing the optimal rotation layer strongly influences how much loss can be removed for the toy distributions considered. This brings iteratively training affine coupling flows significantly closer to end-to-end training. This training scheme is attractive as the number of blocks can be chosen dynamically, and we also propose a hybrid training procedure. Our experiments highlight that end-to-end training is capable of approximating distributions even when rotations are chosen at random. We leave scaling the iterative training with optimized rotations to larger problems open

to future work (note that learning interesting directions from samples in high-dimensional spaces is difficult, compare section 6.2.3).

The above lemma 5.5 and theorem 5.8 lay the foundation of all our other other theoretical guarantees on coupling-based normalizing flows in sections 5.4 and 6.3.

5.4. Coupling flow universality

We now extrapolate the results in the previous section to understand the expressive power of deep coupling-based normalizing flows such as RealNVP (Dinh et al., 2017) and neural spline flows (Durkan et al., 2019) in terms of distributional universality (see definition 5.1). Coupling blocks impose a strong architectural constraint on invertible neural networks. Most strikingly, half of the dimensions are left unchanged in each block, and the transformation of the remaining dimensions is restricted in order to ensure invertibility. At the same time, even the simple affine coupling-based normalizing flows can learn high-dimensional distributions such as images (Kingma & Dhariwal, 2018).

Theoretical explanations for this architecture's ability to fit complex distributions are limited. Existing proofs make assumptions that are not valid in practice, as the involved constructions rely on ill-conditioned neural networks (Teshima et al., 2020a; Koehler et al., 2021) or construct a volume-preserving flow (Koehler et al., 2021). We introduce a new proof for the distributional universality of coupling-based normalizing flows that does not require ill-conditioned neural networks to converge. This proof is constructive, showing that training affine coupling blocks sequentially converges to the correct target (compare figure 5.7).

In summary, we contribute:

- We show that whenever the target distribution is not perfectly learned, there is an affine coupling block that reduces the loss (section 5.4.3).
- We use this result to give a new universality proof for coupling-based normalizing flows that is not volume-preserving, considers the full support of the distribution, and is well-conditioned in section 5.4.4.

Our results validate insights previously observed only empirically: Affine coupling blocks are an effective foundation for normalizing flows, and volume-preserving flows have limited expressive power. We also show that the most recent distributional universality proof for affine coupling-based normalizing flows by Koehler et al. (2021) constructs a volume-preserving flow in section 5.4.2.

The code to our experiments can be found at the following public repository: `https://github.com/vislearn/Coupling-Universality`.

5.4.1. Related work

That coupling-based normalizing flows work well in practice despite their restricted architecture has sparked the interest of several papers analyzing their distributional universality. Teshima et al. (2020a) showed that that coupling flows are universal approximators for invertible functions, which results in distributional universality. Koehler et al. (2021) demonstrated that affine coupling-based normalizing flows can approximate any distribution with arbitrary precision using just three coupling blocks. However, these works assume neural networks with exploding derivatives for couplings, an unrealistic condition in practical scenarios. Our work addresses this limitation by showing that training a normalizing flow layer by layer yields universality. We additionally demonstrate that the latter proof constructs a volume-preserving transformation in section 5.4.2, an additional important limitation.

Some works show distributional universality of *augmented* affine coupling-based normalizing flows, which add at least one additional dimension usually filled with exact zeros (Huang et al., 2020; Koehler

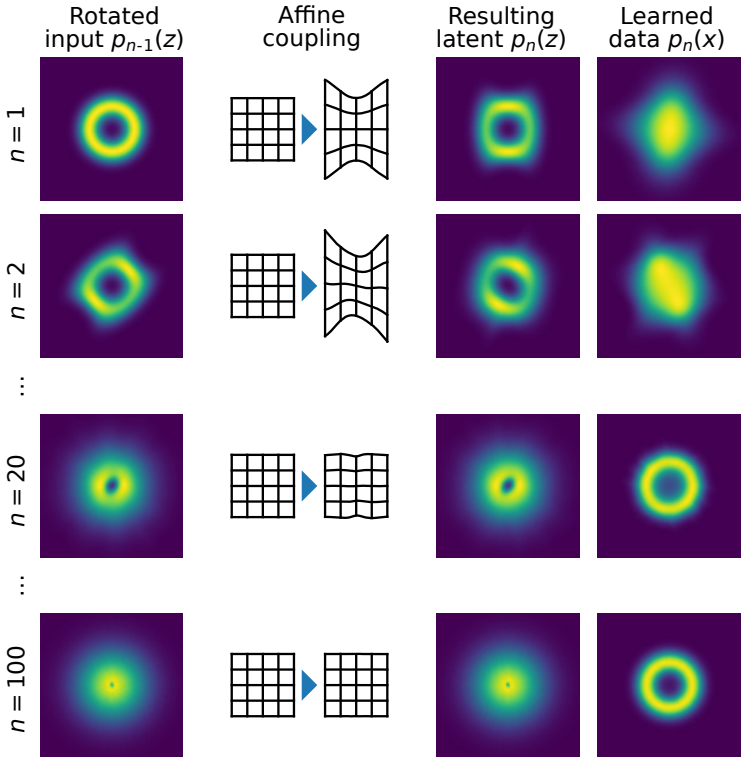


Figure 5.7.: Our universality proof constructs a normalizing flow by iteratively adding affine coupling blocks. We illustrate this by constructing such a flow from real data. Each block first rotates the distribution $p_{n-1}(z)$ from the previous step *(first column)*, then applies an affine coupling layer that transforms the active dimensions to zero mean and unit variance for each passive coordinate b *(second column)*. The resulting latent distribution converges step by step *(third column)* to a standard normal distribution, where the learned additional layers essentially learn the identity *(last row)*. The data distribution $p_\theta(x)$ converges in parallel *(right)*.

et al., 2021; Lyu et al., 2022). The problem with adding additional zeros is that the flow is not exactly invertible anymore in the data domain and usually loses tractability of the change of variables formula (equation (4.1)). Lee et al. (2021) add i.i.d. Gaussians as additional dimensions, which again allows density estimation, but they only show how to approximate the limited class of log-concave distributions. Our universality proof does not rely on such a construction.

Other theoretical work on the expressivity of normalizing flows considers more expressive invertible neural networks, including SoS polynomial flows, Neural ODEs and Residual Neural Networks (Jaini et al., 2019; Zhang et al., 2020a; Teshima et al., 2020b; Ishikawa et al., 2022).

5.4.2. Problems with existing constructions

The existing proofs that affine and more expressive coupling flows are distributional universal approximators (Teshima et al., 2020a; Koehler et al., 2021) come with several limitations. In particular, their constructions use ill-conditioned coupling blocks as the translations $t(b)$ approximate step functions, as noted by Koehler et al. (2021). Also, they give guarantees only on a compact subspace $K \subset \mathbb{R}^D$, and Teshima et al. (2020a) use only one active dimension per coupling. This limits their practical applicability.

In addition, the flow constructed in Koehler et al. (2021) is volume-preserving and thus not universal in KL divergence by our theorem 5.3. Their proof is technically accurate, but they only show convergence under Wasserstein distance W_2 in (Koehler et al., 2021, Theorem 1), which does not imply

convergence in KL (see the counterexample in appendix A.4.4).

It is easy to see that their flow is volume-preserving by looking at the scaling functions $s(b)$ in the three affine coupling layers (equation (4.16)) they use. They read: $s^{(1)} = \epsilon'$ and $s^{(2)} = s^{(3)} = \epsilon''$. This means that the overall flow has a Jacobian determinant $|f'_\theta| = (\epsilon'\epsilon''^2)^{\frac{D}{2}}$. This volume change is independent of the input, making the flow volume-preserving, which is not universal per our theorem 5.3.

Also note how the volume change is directly tied to the guaranteed Wasserstein distance, since they guarantee that $W_2(p_\theta(x), p(x)) < \epsilon$, and the above scalings fulfill $\epsilon'' \ll \epsilon' \ll \epsilon$. Thus, the volume change $|f'_\theta| \ll \epsilon^{\frac{3D}{2}}$ vanishes, and its inverse $|(f_\theta^{-1})'| \gg \epsilon^{-\frac{3D}{2}}$ explodes as ϵ is reduced, rendering the flow ill-conditioned regardless of the distribution at hand. This is additional to the ill-condition of the translation terms $t(b)$ approximating step functions.

Together, this calls for a new universality guarantee that is based on a new coupling flow construction. Our new construction, presented in the following sections, uses a flow that is neither arbitrarily ill-conditioned nor volume-preserving. Also, it converges globally and considers vanilla affine coupling blocks.

5.4.3. Affine coupling blocks have a unique fixed point

To construct our new universality theorem, we first analyze the effect of a single affine coupling on learning a target distribution $p(x)$. Our main result is that an affine coupling block always reduces the loss when it has not yet perfectly learned the target distribution.

To derive this, remember that the loss $\mathcal{L} = \mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(x))$ can be measured both in data and latent space by equation (4.10):

$$\mathcal{L} = \mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(z)) = \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|p(z)). \tag{5.26}$$

Here, the pushforward of $p(x)$ through the flow $f_\theta(x)$ is the latent distribution the flow actually creates by mapping $x \mapsto z = f_\theta(x)$.

Let us now consider what happens if we append one more affine coupling block $f_{\mathrm{blk};\theta_+}$ to an existing invertible neural network $f_\theta(x)$, resulting in a flow which we call $p_{\theta\cup\theta_+}(x)$. Let us choose the parameters of the additional coupling block θ_+ such that it maximally reduces the loss without changing the previous parameters like in section 5.3.5:

$$\min_{\theta_+} \mathcal{D}_{\mathrm{KL}}(p_{\theta\cup\theta_+}(z)\|p(z)). \tag{5.27}$$

Let us formalize this construction for later use:

Definition 5.15. *Given a normalizing flow $p_\theta(x)$ on a continuous probability distribution $p(x)$ with finite first and second moment and $p(x) > 0$ everywhere. Then, we define the* loss improvement by an affine coupling block *as:*

$$\Delta_{\mathrm{affine}}[p_\theta(z)] := \mathcal{D}_{KL}(p_\theta(z)\|p(z)) - \min_{\theta_+} \mathcal{D}_{KL}(p_{\theta\cup\theta_+}(z)\|p(z)), \tag{5.28}$$

where $\theta_+ = (Q, \varphi)$ parameterizes a single L-bi-Lipschitz affine coupling block whose conditioner neural network ψ_φ has at least two hidden layers of finite width and ReLU activations and $p_\theta(z) = (f_{\theta\sharp}p)(z)$.

The coupling block is restricted to be bi-Lipschitz to be well-conditioned. This means that we can choose $L > 1$ such that $L^{-1} < \|f_{\mathrm{cpl}}(x) - f_{\mathrm{cpl}}(y)\|/\|x - y\| < L$, making both forward and inverse passes through each coupling well-conditioned. Choosing a smaller L will result in more coupling blocks, each more numerically stable.

Note that unlike in section 5.3, where we assumed the conditioner function to be any function, here we restrict ourselves to practical neural networks. We assume ReLU networks for mathematical convenience, but think that the definition is equivalent to versions with different activation functions.

Considering this loss improvement $\Delta_{\text{affine}}[p_\theta(z)]$ is a useful quantity, since we can show that is directly related to convergence of the flow:

Theorem 5.16 (Unique fixed point of affine couplings, proof in appendix A.4.1)**.** *With the definitions from definition 5.15:*

$$p_\theta(z) = \mathcal{N}(z; 0, I) \iff \Delta_{\text{affine}}[p_\theta(z)] = 0. \tag{5.29}$$

This is a nontrivial result: One might have thought that it is possible to end up in distributions such that an affine coupling block gets stuck and cannot improve on the loss. Instead, **if adding another coupling layer has no effect, the latent distribution has converged to a standard normal**. This unique fixed point allows using $\Delta_{\text{affine}}[p_\theta(z)]$ as a convergence metric for our universality theorem in section 5.4.4.

In the remainder of this section, we give a sketch of the proof of theorem 5.16, with technical details moved to appendix A.4.1.

We proceed as follows: First, we use the explicit form of the maximal loss improvement $\Delta^*_{\text{affine}}[p_\theta(z)]$ for infinitely expressive affine coupling blocks from theorem 5.8, modified to include optimizing over Q. Then, we show in lemma 5.17 below that convergence of free-form function conditioners is equivalent to convergence of finite ReLU networks. Finally, we show that $\Delta_{\text{affine}}[p_\theta(z)] = 0$ implies $p_\theta(z) = \mathcal{N}(z; 0, I)$. The other direction is trivial, since by $p_\theta(z) = \mathcal{N}(0, I)$, no loss improvement is possible.

If we assume for a moment that neural networks can exactly represent arbitrary continuous functions, then we computed the learned optimal affine transformation in lemma 5.5 to be:

$$\tilde{a}_i(a_i; b) = \frac{1}{\sigma_i(b)}(a_i - m_i(b)). \tag{5.30}$$

Per theorem 5.8, this reduces the loss by:

$$\Delta^*_{\text{affine}}[p_\theta(z)] = \max_Q \mathbb{E}_b[S(b)], \tag{5.31}$$

where Q enters through $(b, a) = Qx$, and the expectation averages the contribution from normalizing the component-wise conditional *non-Standardness* (compare theorem 4.1):

$$S(b) = \mathbb{E}_b\left[\sum_{i=1}^{D/2} \mathcal{D}_{\text{KL}}(\mathcal{N}(m_i(b), \sigma_i(b) \| \mathcal{N}(0, 1))\right] = \frac{1}{2}\sum_{i=1}^{D/2} \mathbb{E}_b\Big[\underbrace{m_i^2(b)}_{\text{(A)}} + \underbrace{\sigma_i^2(b) - 1 - \log \sigma_i^2(b)}_{\text{(B)}}\Big]. \tag{5.32}$$

With the asterisk, we denote that this improvement cannot necessarily be reached in practice with finitely sized and well-conditioned neural networks. More expressive coupling functions can reduce the loss stronger, see section 5.4.5.

What loss improvement can be achieved if we go back to finite neural networks? It turns out that $\Delta^*_{\text{affine}}[p_\theta(z)] > 0$ is equivalent to the existence of a well-conditioned coupling block as in definition 5.15 with $\Delta_{\text{affine}}[p_\theta(z)] > 0$:

Lemma 5.17 (Proof in appendix A.4.1)**.** *Given a continuous probability density $p_\theta(z)$. Then,*

$$\Delta^*_{\text{affine}}[p_\theta(z)] > 0 \iff \Delta_{\text{affine}}[p_\theta(z)] > 0. \tag{5.33}$$

This says that the events $\Delta_{\text{affine}}[p_\theta(z)] = 0$ and $\Delta^*_{\text{affine}}[p_\theta(z)] = 0$ can be used interchangeably. The equivalence comes from the fact that if $\Delta^*_{\text{affine}}[p_\theta(z)] > 0$, then we can always construct a conditioner neural network that scales the conditional standard deviations closer to one or the conditional means closer to zero, reducing the loss. In the detailed proof in appendix A.4.1 we also make use of a classical regression universal approximation theorem (Hornik, 1991) and ensure the additional coupling block is well-conditioned.

Finally, if the first two *conditional* moments of any latent distribution $p(z)$ are normalized for all rotations Q:

$$\mathbb{E}_{a_i|b}[a_i] = m_i(b) = 0, \quad \text{Var}_{a_i|b}[a_i] = \sigma_i(b) = 1, \tag{5.34}$$

then the distribution must be the standard normal distribution: $p(z) = \mathcal{N}(z; 0, I)$: Equation (5.32) enforces two characteristics of $p_\theta(z)$ that uniquely identify the standard normal distribution: (A) It must be rotationally symmetric, since $m_i(b) = 0$ for all Q holds only for rotationally symmetric distributions (Eaton, 1986). (B) This term is non-negative and zero only for $\sigma_i(b) = 1$ for all Q, which uniquely identifies the standard normal from all rotationally symmetric distributions (Bryc, 1995).

This concludes the proof sketch of theorem 5.16 and we are now ready to present our universality result, employing $\Delta_{\text{affine}}[p_\theta(z)]$ as a convergence metric.

5.4.4. Affine coupling flow universality

We now confirm that affine coupling flows are a distributional universal approximator in terms of the convergence metric we derived in section 5.4.3:

Theorem 5.18 (Affine coupling universality, proof in appendix A.4.2)**.** *For every continuous $p(x)$ with finite first and second moment with infinite support, there is a sequence of normalizing flows $p_n(x)$ consisting of n L-bi-Lipschitz affine coupling blocks such that their latent distributions converge to the standard normal:*

$$p_n(z) \xrightarrow{n\to\infty} \mathcal{N}(z; 0, I), \tag{5.35}$$

in the sense that $\Delta_{\text{affine}}[p_n(z)] \xrightarrow{n\to\infty} 0$.

This means that with increasing depth, the latent distribution of the flow converges to the standard normal. The use of $\Delta_{\text{affine}}[p_n(z)]$ as a convergence metric is justified by theorem 5.16 that $p_n(x) = p(x) \Leftrightarrow p_n(z) = \mathcal{N}(0, I) \Leftrightarrow \Delta_{\text{affine}}[p_n(z)] = 0$.

The proof of theorem 5.18 explicitly constructs a normalizing flow by following an iterative scheme. We start with the data distribution as our original guess for the latent distribution: $p_0(z) = p(x = z)$. Then, we repeatedly append individual affine coupling blocks $f_{\text{blk}}(x)$ consisting of a rotation Q and a coupling f_{cpl}, optimizing the new parameters to maximally reduce the loss as in equation (5.27).

This series of coupling blocks converges: $\Delta_{\text{affine}}[p_\theta(z)]$ measures how much adding each affine coupling block reduces the loss, but the total loss that can be reduced by the concatenation of many blocks is bounded. Since improvements $\Delta_{\text{affine}}[p_\theta(z)]$ are also non-negative, they must converge to zero for the sum to be finite (Rudin, 1976, Theorems 3.14 and 3.23). By theorem 5.16, the fixed point of this procedure is a standard normal distribution in the latent space.

Figure 5.7 shows an example for how theorem 5.18 constructs the coupling flow in order to learn a toy distribution. The affine coupling flow can learn the distribution well, despite its difficult topology. Empirically, this is also true in terms of KL divergence: Figure B.1 in appendix B.1.1 shows the relation between $\Delta_{\text{affine}}[p_\theta(z)]$ and the KL divergence for the flow, both of which decrease over the course of training. There we also provide experimental details.

5.4.5. Expressive coupling flow universality

The above theorem 5.18 shows that affine coupling functions $c(a_i;\theta) = sa_i + t$ are sufficient for universal distribution approximation. As listed in section 4.3.1, a plethora of more expressive coupling functions have been suggested, for example neural spline flows (Durkan et al., 2019) that use monotone rational-quadratic splines as the coupling function. It turns out that by choosing the parameters in the right way, all coupling functions we are aware of can exactly represent an affine coupling, except for the volume-preserving variants, see section 4.3.1 for a list. For example, a rational quadratic spline can be parameterized as an affine function by using equidistant knots $(a_k, \tilde{a}_k)$ such that $\tilde{a}_k = sa_k + t$ and fixing the derivative at each knot to s.

Thus, the universality of more expressive coupling functions follows immediately from theorem 5.18, just like Ishikawa et al. (2022) extended their results from affine to more expressive couplings:

Corollary 5.19 (Expressive coupling universality)**.** *For every continuous $p(x)$ with finite first and second moment with infinite support, there is a sequence of normalizing flows $f_n(x)$ consisting of n coupling blocks* with coupling functions at least as expressive as affine couplings *such that their latent distributions converge to the standard normal:*

$$p_n(z) \xrightarrow{n\to\infty} \mathcal{N}(z; 0, I), \tag{5.36}$$

in the sense that $\Delta_{\text{affine}}[p_n(z)] \xrightarrow{n\to\infty} 0$.

Proof. The function class of a coupling layer more expressive than affine parameterizes a superset of the functions that can be represented by an affine coupling layer, and therefore the solution constructed in theorem 5.18 is also in more expressive function classes. □

A similar result can also be stated about recursive coupling layer HINT as well as autoregressive layers (compare section 4.3), since they also parameterize larger classes of invertible layers than affine coupling layers.

In addition, our construction of a universal flow in theorem 5.18 through layer-wise training reveals how more expressive coupling functions can outperform affine functions using the same number of blocks. Similar to the loss improvement for an affine coupling in equation (5.28), let us compute the maximal loss improvement for an arbitrarily flexible coupling:

$$\Delta^*_{\text{universal}}[p_\theta(z)] = \max_Q \mathbb{E}_b \left[\sum_{i=1}^{D/2} (G_i(b) + S_i(b)) \right] \geq \Delta^*_{\text{affine}}[p_\theta(z)], \tag{5.37}$$

where the expectation again goes over the passive coordinate $b = (Qz)_{1,\dots,D/2}$ and $z \sim p_\theta(z)$.

Here, the loss improvement additional to the non-Standardness $S_i(b)$ in the ith active dimension as given in equation (5.32) is the conditional non-Gaussianity $G_i(b) = \mathcal{D}_{\text{KL}}(p_\theta(a_i|b) \| \mathcal{N}(m_i(b), \sigma_i(b)))$, which measures the deviation of each active dimension from a Gaussian distribution with matching mean and variance. An affine coupling function $c(a_i;\theta) = sa_i + t$ doesn't influence this term, due to its symmetrical effect on both sides of the KL in $G(b)$ by lemma 4.2. More expressive coupling blocks, however, can tap on this loss component if the conditional distributions $p(a_i|b)$ are non-Gaussian, see figure 5.8 for an example. Note that while a single affine coupling does not affect $G_i(b)$, subsequent blocks can because the overall loss is redistributed over the loss terms.

The impact of this gain likely varies with the dataset. For instance, in images, the distribution of one color channel of one pixel conditioned on the other color channels in the entire image, often shows a simple unimodal pattern with low non-Gaussianity. This may explain why affine coupling blocks are enough to learn the distribution of images (Kingma & Dhariwal, 2018). We give additional technical details on equation (5.37) and the subsequent arguments in appendix A.5.

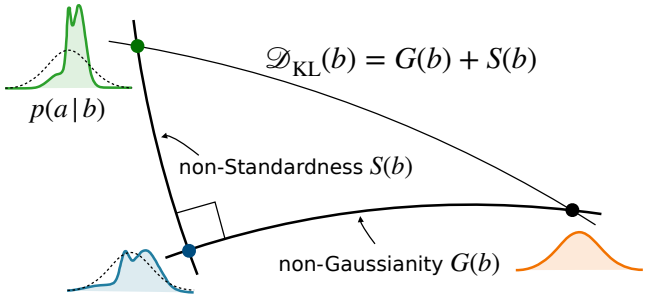


Figure 5.8.: Information-geometric view of couplings more expressive than affine: The conditional KL divergence $\mathcal{D}_{\mathrm{KL}}(p(a|b)\|\mathcal{N}(0,I))$ can be split into two orthogonal KL divergences, the non-Standardness $S(b)$ sensitive to the first two moments, and the non-Gaussianity $G(b)$ sensitive to non-Gaussianity. Affine couplings only reduce $S(b)$, more expressive coupling also affect $G(b)$.

5.5. Conclusion

In this chapter, we considered the universality of existing invertible neural networks. We derived that volume-preserving flows are not distributional universal approximators under KL divergence and cannot represent multi-modal distirbutions. Through identifying the distribution a volume-preserving flow constructs in the latent space, we propose a fix to volume-preserving flows that can be applied after training to make restore universality. Since the fix consists of learning a one-dimensional transform, we expect that the beneficial properties of volume-preserving flows are easily transferable to our construction.

Regarding the widespread coupling-based normalizing flows, we derived the minimizer and minimum achievable loss value of a single affine coupling block trained with maximum likelihood. The central insight is that such a block learns to normalize the first two moments of the active half of dimensions conditioned on the passive half of dimensions. The rotation Q determines which subspaces of the data end up being active or passive, and we give an explicit formula for the possible loss improvement for a fixed rotation. We use this to approximate the best rotation and find that this makes layer-wise training competitive to end-to-end training, while transforming the data more naturally to a Gaussian latent distribution.

Using these results on a single layer, we construct a novel universality proof for coupling-based normalizing flows.

Together, this reveals an intriguing hierarchy of the expressivity of invertible neural networks:

1. *Volume-preserving normalizing flows* are not universal in KL divergence, meaning that there is a bias in what distributions they learn in practice. We propose a simple fix to restore universality.

2. *Affine coupling flows* such as RealNVP (Dinh et al., 2017) are distributional universal approximators in terms of Δ_{affine} despite their seemingly restrictive architecture.

3. Coupling flows with *more expressive coupling functions* are also universal approximators, but they converge with fewer blocks by tapping on an additional loss component in layer-wise training.

Our work theoretically grounds why coupling blocks are the standard choice for practical applications with normalizing flows, combined with their easy implementation and speed in training and inference. We remove spurious constructions present in previous proofs and use a simple principle instead: Construct a flow layer by layer until no more loss improvement can be achieved.

Table 5.1 summarizes how our universal construction is closer to practice than previous work (Teshima et al., 2020a; Koehler et al., 2021): We only use well-conditioned L-bi-Lipschitz couplings, allow variable volume change $|f'_\theta(x)|$ (as evidenced by the rescaling term in equation (5.30)) and

Table 5.1.: Our construction of a universal coupling flow overcomes important limitations of the previous work on arbitrary input $p(x)$ (Teshima et al., 2020a; Koehler et al., 2021): We use well-conditioned coupling blocks, consider convergence on the full space and allow variable volume change $|f'_\theta(x)| \neq \text{const}$, which is necessary for universality in KL divergence by theorem 5.3.

	Teshima et al. (2020a)	Koehler et al. (2021)	Theorem 5.18 (ours)
Well-conditioned	✗	✗	✓
Variable $\lvert f'_\theta(x)\rvert$	✓	✗	✓
Global support	✗	✗	✓

consider the entire support of $p(x)$. We give further details on the sensitivity of $\Delta_{\text{affine}}[p_\theta(z)]$ to volume-preserving transformations in appendix A.2.1.

Directions for Future work Despite these advances, there are some properties we hope can be improved in the future: First, our construction shows that we can build a deep enough flow with arbitrary precision, but we have not exploited that blocks can be jointly optimized. Thus, while our construction shows universality of end-to-end training, we expect a flow trained this way to require fewer blocks than our iterative proof for the same performance.

Secondly, it is unclear how the convergence metric $\Delta_{\text{affine}}[\cdot]$ in definition 5.15 is related to convergence in the loss used in practice, the KL divergence given in equation (4.3). In practice, we find that constructing a coupling flow through iterative training converges in KL divergence (see figure B.1 in appendix B.1.1), so we conjecture that our way of constructing a universal coupling flow converges in KL divergence. The reverse holds: We show in corollary A.10 in appendix A.4.3 that convergence in KL implies convergence under our new metric.

Finally, our proof gives no guarantee on the number of required coupling blocks to achieve a certain performance.

We will partly address the last two points in the next chapter on the convergence rate of normalizing flows, where we consider the KL divergence, among others. We hope that our contribution paves the way towards a full understanding of affine coupling-based normalizing flows.

6. Convergence rates of invertible neural network blocks

In the previous chapter, and in particular theorem 5.18 and corollary 5.19, we established that generative models $p_\theta(x)$ based on non-volume-preserving coupling blocks are universal distributional approximators. Similarly, Gaussianization is in theory able to represent arbitrary distributions to arbitrary precision (Chen & Gopinath, 2000; Meng et al., 2020). However, these results come with two important shortcomings. They only show that there exists some finite number of blocks L to achieve a given performance, but do not state how large this number is. In addition, they do not give a convergence guarantee for the loss used in practice, but consider weaker convergence metrics.

In this chapter, we take a step towards addressing these shortcomings: We derive explicit convergence rates of the practical loss, the Kullback-Leibler divergence in equation (4.4). We focus our derivations on the efficiency of different normalizing flow architectures at learning the first two moments of the incoming data distribution, most notably the covariance matrix Σ. To motivate this special case, we first show in section 6.1 that fitting these moments to any distribution is a separable task that any generative model has to solve to accurately approximate a distribution $p(x)$. We thus expect that any difficulties arising in learning the first two moments also apply in learning the full distribution.

In particular, we compare the three basic invertible block layouts that have been proposed in the literature: Gaussianization blocks, which modify all dimensions independently in each block (equation (4.21)), coupling blocks, which allow the first half of dimensions to influence the second half (equation (4.23)), and autoregressive blocks (equation (4.27)), where every dimension is influenced by all previous.

6.1. Setting

To compare the different architectures, we choose to focus on comparing the approaches in their ability to learn the covariance $\Sigma = \mathrm{Cov}_{x\sim p(x)}[x]$ of a given distribution $p(x)$. This is interesting because (i) we will see that the considered models do differ in the resources required for this task, (ii) the convergence rates are analytically tractable and (iii) learning the covariance is a necessary condition for convergence of maximum likelihood training in equation (4.4). To see the third point, we again make use of the Pythagorean identities in theorem 4.1, of which we repeat equation (4.35) for convenience:

$$\mathcal{L} = \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0,I)) = \underbrace{\mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(m,\Sigma))}_{\text{non-Gaussianity } G[p_\theta(z)]} + \underbrace{\mathcal{D}_{\mathrm{KL}}(\mathcal{N}(m,\Sigma)\|\mathcal{N}(0,I))}_{\text{non-Standardness } S(m,\Sigma)}. \tag{4.35}$$

Remember from section 4.2 that learning a normalizing flow means learning a function that maps data points $x \sim p(x)$ to latent codes such that these codes follow the standard normal $p(z) = \mathcal{N}(0, I)$. The above decomposition splits the transport from the data distribution to the latent standard normal into two parts: (i) From the data to the nearest Gaussian distribution $\mathcal{N}(m, \Sigma)$, measured by the non-Gaussianity $G[p_\theta(z)]$ (Cardoso, 2003) (also called negentropy (Comon, 1994)). (ii) From that nearest Gaussian to the standard normal in the latent space, for which a linear transformation suffices.

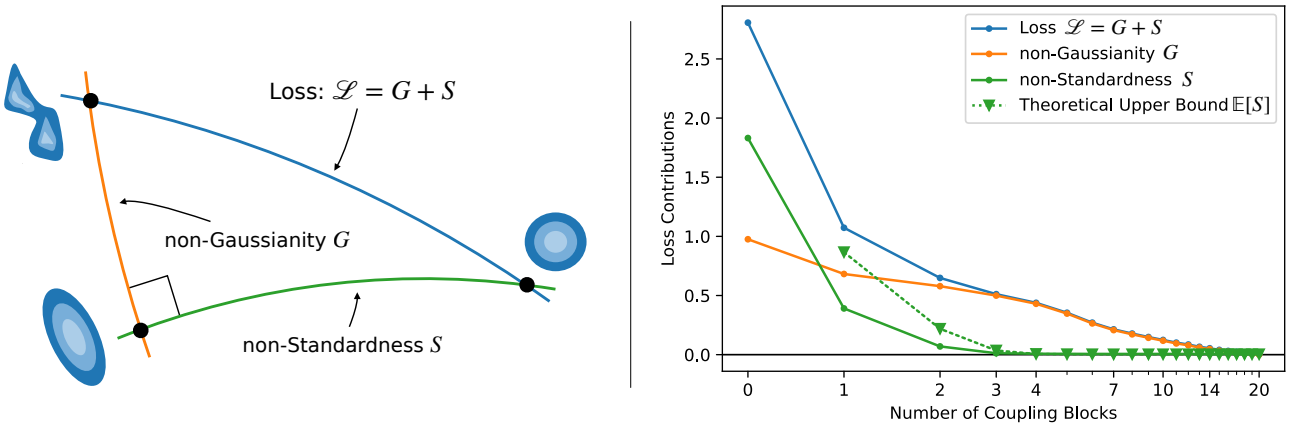


Figure 6.1.: *(Left)* The Maximum Likelihood Loss $\mathcal{L}$ (blue) can be split into the *non-Gaussianity* G (orange) (Cardoso, 2003) and the *non-Standardness* S (green) of the latent code $z = f_\theta(x)$: $\mathcal{L} = G + S$ (theorem 4.1). For the latter, we give explicit guarantees as one more coupling block is added in theorems 6.7 and 6.8 and show a global convergence rate in theorem 6.9. *(Right)* Typical fit of EMNIST digits by a standard affine coupling flow for various depths. Our theory (theorem 6.7) upper bounds the average S for $L + 1$ coupling blocks given a trained model with L coupling blocks (dotted green). We observe that our bound is predictive for how much end-to-end training reduces S.

This split is visualized in figure 6.1. In an experiment, we fit a series of affine coupling flows of increasing depths to the EMNIST digit dataset (Cohen et al., 2017) using maximum likelihood loss and measure the capability of each flow in decreasing G and S (details in appendix B.4.1).

The form of the non-Standardness S is given by the well-known KL divergence between the involved normal distributions (see appendix A.1.1). It only depends on the first two moments of $p_\theta(z)$, its mean $m = \mathbb{E}_{z\sim p_\theta(z)}[z]$ and its covariance matrix $\Sigma = \mathrm{Cov}_{z\sim p_\theta(z)}[z]$:

$$S(m, \Sigma) = \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(m, \Sigma) \| \mathcal{N}(0, I)) = \frac{1}{2}(\|m\|^2 + \operatorname{tr}\Sigma - D - \log\det\Sigma). \tag{6.1}$$

The non-Standardness $S(m, \Sigma)$ will be our measure of how far the covariance and mean have approached the standard normal in the latent space.

Since a linear transformation suffices to map the mean and covariance of any input distribution to zero respectively the identity matrix, we restrict the action of each invertible block to be linear. This also ensures that the construction does not increase the non-Gaussianity, as it is left invariant under linear transformations by lemma 4.2. This leaves room for a complimentary theory to derive a convergence rate in non-Gaussianity.

For each block architecture, we consider two constructions to derive a scaling of the number of blocks with the dimensionality of the problem: First, if the flow is trained layer by layer, how many blocks are required to reduce the non-Standardness to a given target value? We derive the scaling laws in the limit of low loss. Second, if the flow is trained end-to-end, how many blocks are required at least to exactly match the first two moments.

To formulate the scaling laws, we adopt the notation for asymptotic convergence behavior with dimension D by Knuth (1997), replacing $O \to \mathcal{O}$ to avoid name collision with the orthogonal group $O(D)$:

$$\mathcal{O}(f(D)) := \{g(D) \leq Cf(D) \text{ for } D > D^*\}, \tag{6.2}$$
$$\Omega(f(D)) := \{g(D) \geq Cf(D) \text{ for } D > D^*\}, \tag{6.3}$$
$$\Theta(f(D)) := \mathcal{O}(f(D)) \cap \Omega(f(D)). \tag{6.4}$$

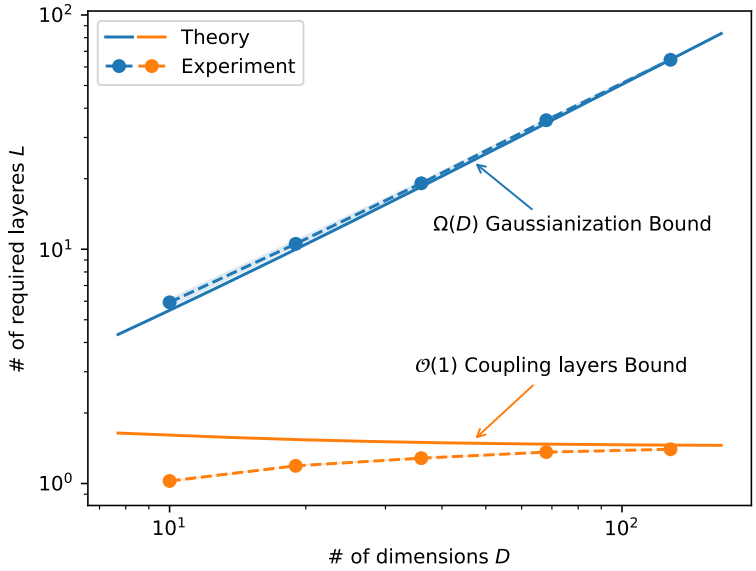


Figure 6.2.: Empirical scaling of learning Gaussian distributions as a function of dimension D in the limit of low loss. Gaussianization requires at least $\Omega(D)$ layers (theorem 6.3), while only at most constant (i.e. $\mathcal{O}(1)$) number of coupling layers are needed (theorem 6.9). The solid lines are the exact values predicted by the theories, the dots indicate experimental measurements. The shades show the interquartile range over various initial covariance matrics.

Figure 6.2 visualizes the high-level perspective on our theoretical results alongside an experimental confirmation: To reduce the non-Standardness by a given amount, Gaussianization requires at least $\Omega(D)$ blocks, while coupling blocks require a constant number $\mathcal{O}(1)$ of blocks regardless of the dimension. These results hold in late training when $S \ll 1$ and for layer-wise training, which is state-of-the-art for Gaussianization (so this is a useful lower bound) and it indicates that jointly trained coupling blocks may perform even better (so this is a useful upper bound).

In the remainder of this chapter, we derive these convergence rates and give additional details. We then summarize our results and combine them with existing knowledge on other architectures.

6.2. Gaussianization

This section is an adapted version of Draxler et al. (2023).

The Gaussianization layer in equation (4.21) was proposed by Chen & Gopinath (2000). In this section, we contribute the following:

- We analytically derive that the number of Gaussianization blocks grows linearly with the dimensionality of the problem for Gaussian input, in section 6.2.3 in terms of KL divergence for iterative training with random rotations, and in section 6.2.4 for end-to-end training.
- For iterative training, we demonstrate limits of determining better-than-random rotations from finite training data.
- We empirically determine the scaling behavior for non-Gaussian data, where we find a similar linear increase in complexity with dimension, and favorable scaling for some distributions (see section 6.2.5).

The code to our experiments can be found at the following public repository: `https://github.com/vislearn/Gaussianization-Bound`.

6.2.1. Related work

Several works consider iteratively transforming between distributions via rotations and element-wise transformations. Originally, the idea of iteratively transporting input data to standard normal latent codes has been proposed by Chen & Gopinath (2000). Laparra et al. (2011) extended the idea with rotation-based iterative Gaussianization (RBIG), by also learning the reverse transport from latent codes to the data distribution. Other variants like Iterative Distribution Transfer (IDT) replace the standard normal by an arbitrary other distribution (Pitié et al., 2007). Meng et al. (2020) leave the iterative scheme and train the flow end-to-end, which performs favorable in situations of little data.

An important part of all these works is to find meaningful non-Gaussian projections of the data. Originally, random orthogonal matrices, ICA and PCA were suggested (Chen & Gopinath, 2000). Meng et al. (2020) learn rotations jointly with the dimension-wise transforms. Sliced Iterative Normalizing Flow (SINF) uses max K-SWD, which optimizes for the K most non-Gaussian directions in terms of Sliced Wasserstein Distance (SWD) (Dai & Seljak, 2021). Sliced Wasserstein Flows (SWF) had previously suggested utilizing SWD, but used this to iteratively solve a PDE (Liutkus et al., 2019). We give limitations in section 6.2.3.

The existing work on the universality of Gaussianization (Chen & Gopinath, 2000; Meng et al., 2020) relies on the concept of weak convergence, ensuring convergence of expectation values $\mathbb{E}_{x\sim p_\theta(x)}[f(x)] \to \mathbb{E}_{x\sim p(x)}[f(x)]$ for f under mild regularity conditions. This does not show convergence of the corresponding densities "$q \to p$" (Gibbs & Su, 2002). We fill this gap by considering the KL divergence in the form of the non-Standardness, a stronger notion of convergence. We also give explicit convergence rates instead of asymptotic guarantees. Our theoretical derivations are limited to Gaussian distributions close to convergence, however.

6.2.2. Architecture details

Before we come to our convergence guarantees, we give some additional context on Gaussianization, as it is often constructed differently than other normalizing flows.

As mentioned in section 4.3.2, a Gaussianization layer splits an incoming data vector $x \in \mathbb{R}^D$ into its D dimensions and then applies a one-dimensional invertible function to each dimension. This is motivated by an exact decomposition of the loss $\mathcal{L}$ in theorem 4.1, which we repeat for convenience:

$$\mathcal{L} = \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0,I)) = \underbrace{\mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|p(x_1)\cdots p(x_D))}_{\text{Mutual Information } I[p_\theta(z)]} + \sum_i \underbrace{\mathcal{D}_{\mathrm{KL}}(p_\theta(z_i)\|\mathcal{N}(0,1))}_{\text{marginal loss } \mathcal{L}[p_\theta(z_i)]}. \tag{4.37}$$

This identity makes clear that the loss is composed of two parts: The *mutual information* $I := I[p_\theta(z)]$, which measures how far $p(x)$ deviates from the product of its marginal $p(z_1)\cdots p(z_D)$, and the *marginal loss* $\mathcal{L}_i := \mathcal{L}[p_\theta(z_i)]$ for each dimension, which measure the deviation of each marginal $p(z_i)$ from the univariate standard normal.

A single layer can reduce the marginal losses in all dimensions (close) to zero if each one-dimensional transformation is sufficiently rich. Since a single-dimensional transformation does not affect the mutual information by lemma 4.3, this leaves only the mutual information I as the remaining loss. As mentioned in section 4.3.2, this is where the rotation layer $f_{\mathrm{rot}}(x) = Qx$ comes into play.

Rotating the data does not affect the sum of the loss contributions $\mathcal{L} = I + \sum_i \mathcal{L}_i$, but it redistributes it between the mutual information and the marginal losses, see below. Note that unlike for coupling blocks, permutation matrices are not sufficient for mixing the dimensions as they do not re-distribute loss from the mutual information to the marginal losses.

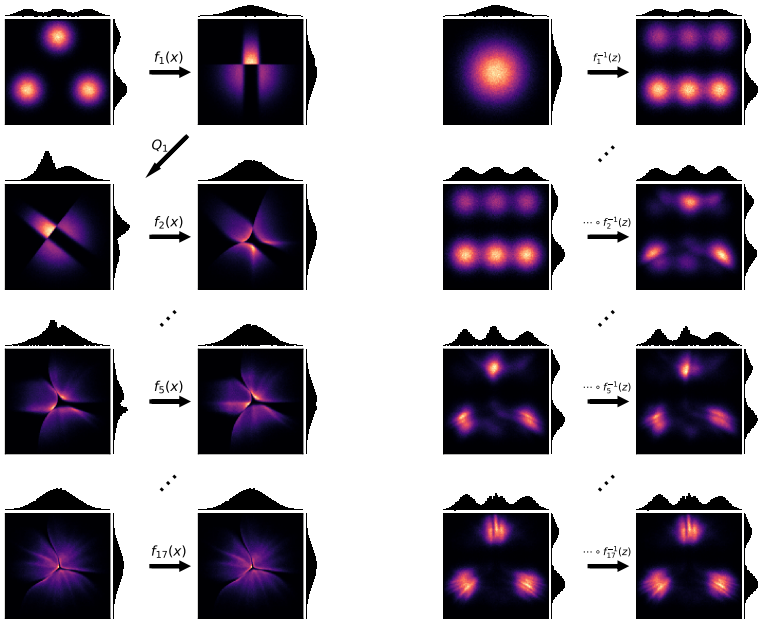


Figure 6.3.: Gaussianization learns a Gaussian mixture with three modes. *(Left)* Gaussianization makes marginals normal and rotates randomly. Iterating makes the latent distribution gradually Gaussian. *(Right)* The first layer approximates $p(x)$ via the product of its marginals, see first row: $q_1(x) \approx p(x_1)p(x_2)$. Subsequent rows show the effect of additional later layers. The rows show the effect of layer $1, 2, 5$, and 17.

Training

Gaussianization is typically trained layer-by-layer (*iterative*) or by performing gradient descent with equation (4.3) on all blocks jointly (*end-to-end*).

Iterative training In this training paradigm, one adds layers one by one: The data set is used to train the first block f_{block} such that it transforms each marginal close to a univariate standard normal. The second block is then trained with the data transformed by the first layer. This is visualized in figure 6.3.

This approach has two advantages: First, there are fast approaches for learning the one-dimensional transforms that do not require backpropagation. One approach is to estimate the cumulative distribution function (CDF) on each 1D slice using quantiles of the data. Dai & Seljak (2021) use rational-quadratic splines to fit the CDF due to their flexibility and analytical invertibility. Second, the approach allows choosing rotations that promise maximum loss improvement by shifting loss from the mutual information I to the marginal losses $\mathcal{L}_i$ in equation (4.37) (see below).

End-to-end training In this approach, one concatenates a prespecified number of blocks at initialization. The parameters of each block are then trained jointly using negative log-likelihood in equation (4.4) (Meng et al., 2020). The advantage of end-to-end training is that blocks can collaborate, as the training signal is the gradient of the entire pipeline and not just of a single block.

In practice, end-to-end training requires fewer layers than iterative training when learning a given distribution in low dimensions (Meng et al., 2020). As the dimension increases, however, the convergence of the training saturates. State-of-the-art Gaussianization results in high dimensions (on MNIST and CIFAR10 image datasets) are currently held by iterative training with SINF (Dai & Seljak, 2021).

Choosing rotations

As mentioned above, the KL divergence $\mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0,I))$ itself is symmetric under rotations, but the rotation Q distributes the loss between the mutual information I and the marginal losses $\mathcal{L}_i$ (see equation (4.37)). The more loss ends up in the $\mathcal{L}_i$, the more loss can be reduced by the block. To illustrate this, consider two special examples:

First, take a distribution $p(x) \neq \mathcal{N}(0,I)$ which can be written as the product of marginals in some rotation of the data Q^*:

$$p(Q^*x) = p((Q^*x)_1)\dots p((Q^*x)_D). \tag{6.5}$$

If we evaluate the loss in this orientation, the mutual information $I = 0$ becomes zero and all loss is contained in the marginals: $\mathcal{L} = \sum_i \mathcal{L}_i$. This allows Gaussianization to fit $p(x)$ in one block: Parameterize the rotation layer as $Q = Q^*$ and let the element-wise transforms fit the $p((Q^*x)_i)$.

Now consider a Gaussian distribution $p(x) = \mathcal{N}(0,\Sigma)$ where $\operatorname{tr}\Sigma = D$. Then, there exists a rotation Q^+ for which the standard deviations along the axis are one, i.e. $(\tilde{\Sigma})_{ii} = ((Q^+)^{\mathrm{T}}\Sigma Q^+)_{ii} = 1$. Then, $\mathcal{L}_i = 0$ and all the loss is contained in the mutual information: $\mathcal{L} = I$. The element-wise transformation layer cannot make any progress in this situation.

For the two examples given, it is in principle known how to obtain the optimal choice for Q: In the case of the first distribution, independent component analysis (ICA, (Hyvärinen et al., 2001)) aims to find Q^* such that the marginals are independent. For the case of Gaussian distributions, principal component analysis (PCA, (Pearson, 1901)) yields the optimal Q^* such that $\tilde{\Sigma}$ is diagonal and can be fit using one block.

For most real-world distributions, however, such an orientation Q^* does not exist where the data dimensions become independent.

The Cramér-Wold theorem (Cramér & Wold, 1936) guarantees that the learned latent codes are exactly Gaussian if and only if there is no orientation Q with a non-Gaussian marginal. To visualize this, imagine projecting the data set along a unit vector and looking at the histogram:

$$z_{\mathrm{proj}} = w^{\mathrm{T}}z. \tag{6.6}$$

If the codes z are distributed like a multivariate standard normal distribution, then each projection z_{proj} will be distributed like a univariate standard normal distribution. If, however, z is not normally distributed, then there must be some projection for which the data is also not distributed like a normal distribution. Equivalently, if all marginal losses $\mathcal{L}_i = 0$ for all possible rotations Q, then $p_\theta(z) = \mathcal{N}(0,I)$. Note that our theorem 5.16 is a variant of the Cramér-Wold theorem for coupling flows.

This implies that for a single layer, we want to choose Q such that the marginal projections are as non-Gaussian as possible. Then, as much loss as possible is contained in the $\mathcal{L}_i$, which can then be removed by the Gaussianization layer f_{gzn} acting on each dimension individually.

There is a rich history in identifying interesting marginal directions in high-dimensional data. These are the most common choices for computing Q from data:

- Random rotations are randomly sampled as $Q \in O(D)$. We give several guarantees in section 6.2.3 for this case.
- Principal Component Analysis (PCA) transforms any distribution with nontrivial covariance Σ such that its principal axes coincide with the coordinate axes, i.e. such that the resulting covariance matrix is diagonal $Q^{\mathrm{T}}\Sigma Q = \mathrm{Diag}(\Lambda)$ with eigenvalues Λ.
- Independent Component Analysis (ICA) identifies the space in which a $p(u) = p(u_1)\cdots p(u_D)$, if $x = Au$ and u can be factorized in this form.

- max K-SWD identifies the directions in which the sliced Wasserstein distance can be maximally reduced. The sliced Wasserstein distances can be a proxy for the marginal KL divergences, as both measure a divergence between distributions.

6.2.3. Iterative training

We now derive scaling laws for normalizing flows based on Gaussianization blocks if trained in the iterative training scheme.

Single Gaussianization layer minimizer

As explained in section 6.1, we restrict ourselves to linear Gaussianization blocks, as this is sufficient to learn a Gaussian distribution, and as this does not interfere with non-Gaussianity. Restricted to the family of affine-linear mappings, the Gaussianization layer in equation (4.21) takes the following form:

$$f_{\text{gzn,lin}}(x) = r \odot x + u, \tag{6.7}$$

where $r \in \mathbb{R}_+^D$ and $u \in \mathbb{R}^D$ are to be determined.

The following result characterizes the first two moments of the data after applying a single Gaussianization block with a fixed rotation trained to optimality:

Proposition 6.1 (Moments after single Gaussianization layer, proof in appendix A.6.1)**.** *Given D-dimensional data with mean m and covariance Σ and a rotation matrix Q. Then, the moments $\tilde{m}, \tilde{\Sigma}$ that can be reached by a linear Gaussianization layer as in equation* (6.7) *are:*

$$\tilde{m} = 0, \qquad \tilde{\Sigma}(Q) = M(Q\Sigma Q^{\mathrm{T}}). \tag{6.8}$$

This minimizes $S(\tilde{m}, \tilde{\Sigma})$ as given in equation (6.1)*, and G does not change.*

The function M takes a square matrix A and rescales the diagonal elements to 1 as follows. It is a well-known operation in numerics called Diagonal scaling or Jacobi preconditioning (Shewchuk et al., 1994) so that $M(A)_{ii} = 1$:

$$M(A)_{ij} = \sqrt{A_{ii}A_{jj}}^{-1} A_{ij} = (\operatorname{Diag}(A)^{-1/2} A \operatorname{Diag}(A)^{-1/2})_{ij}. \tag{6.9}$$

This means that a single Gaussianization layer will center the data and scale the standard deviations of each marginal to one.

What is the effect of proposition 6.1 on the non-Standardness? First note that for mean $m = 0$ and a covariance Σ with $\operatorname{tr}\Sigma = D$, the non-Standardness in equation (6.1) simplifies:

$$S(0, \Sigma) = -\frac{1}{2} \log\det\Sigma. \tag{6.10}$$

Thus, the non-Standardness after the layer in proposition 6.1 reads:

$$S(\tilde{m}, \tilde{\Sigma}(Q)) = -\frac{1}{2} \log\det\tilde{\Sigma} = -\frac{1}{2}(\log\det\Sigma - \log\det\operatorname{Diag}(Q\Sigma Q^T)), \tag{6.11}$$

where $\operatorname{Diag}(Q\Sigma Q^T)^{-1/2}$ is the scaling matrix of the Jacobi preconditioning. To derive this, insert equation (6.8) and use that $\det(AB) = \det(A)\det(B)$ for square matrices A, B.

Expected single Gaussianization block performance

We now consider the role of the rotation Q in the Gaussianization block. In particular, we derive expected non-Standardness after a Gaussianization block, averaged over rotations.

To simplify the arguments, we assume that the covariance has normalized trace already before the block:

Assumption 6.1. The covariance is normalized: $\operatorname{tr}\Sigma = D$.

This is part of typical data preprocessing, it can be achieved by scaling all data points by the average standard deviation over all dimensions. Also, it is achieved after a single Gaussianization layer, since $\operatorname{tr}\tilde{\Sigma}(Q) = D$ as all its diagonal entries are 1 by proposition 6.1.

We then find:

Theorem 6.2 (Non-Standardness after Gaussianization block, proof in appendix A.6.2)**.** *Given D-dimensional data with covariance Σ fulfilling assumption 6.1 with covariance eigenvalues $\lambda_1, \dots \lambda_D$. Then:*

$$S(m, \Sigma) \geq \mathbb{E}_Q[S(\tilde{m}, \tilde{\Sigma}(Q))] \geq S(0, \Sigma) - \frac{1}{2\lambda_{\min}^2} \frac{2}{D+2} \operatorname{Var}[\lambda] \tag{6.12}$$

$$\geq S(0, \Sigma) - \frac{2}{D+2} \frac{2 - g^D}{1 - \sqrt{1 - g^D}} (1 - g), \tag{6.13}$$

where $\operatorname{Var}[\lambda]$ *is the empirical variance of the covariance eigenvalues, g is their geometric mean and* $\lambda_{\min}$ *is the minimal eigenvalue.*

These guarantees make a negative statement: The non-Standardness after the layer will be at least as large as given (but not worse than before the block). The last variant in equation (6.13) is particularly useful, since we can rewrite the geometric mean of the covariance eigenvalues via the non-Standardness:

$$g = \prod_{i=1}^{D} \lambda_i^{D/2} = \exp(-2S(\Sigma)/D) < 1. \tag{6.14}$$

Deep Gaussianization guarantee near convergence

We now extrapolate theorem 6.2 to a deep network, that is can we bound the non-Standardness after L_{gzn} Gaussianization blocks.

It will be helpful to assume:

Assumption 6.2. The data is centered: $m = 0$.

Like assumption 6.1, centering the data is part of usual preprocessing and achieved by a single Gaussianization block by proposition 6.1.

Theorem 6.3 (Deep Gaussianization bound, proof in appendix A.6.3)**.** *Given D-dimensional data with covariance Σ fulfilling assumptions 6.1 and 6.2, and $S(0, \Sigma) \ll 1$. Then, the expected non-Standardness after L_{gzn} Gaussianization blocks as in proposition 6.1 can be bounded as follows:*

$$\mathbb{E}_{Q_1,\dots,Q_{L_{\text{gzn}}}}[S(0, \Sigma^{(L_{\text{gzn}})}(Q_1, \dots Q_{L_{\text{gzn}}}))] \geq \left(1 - \frac{2}{D+2}\right)^{L_{\text{gzn}}} S(0, \Sigma). \tag{6.15}$$

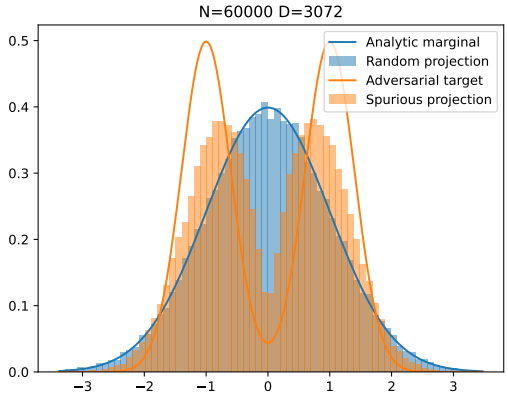


Figure 6.4.: Spurious projection of standard normal data. The plot shows $N = 60,000$ samples from a $D = 3072$-dimensional standard normal distribution projected to a single dimension. The blue projection is selected randomly, and the resulting histogram is close to standard normal. The orange projection is optimized so that the dataset has a spurious bimodal histogram. The histograms coincide with the marginal distribution Gaussianization would learn, producing a bimodal distribution from Gaussian data in the second case.

By solving equation (6.15) for the number of blocks L_{gzn}, we find that **the number of required blocks increases** (at least) **linearly with dimension** to decrease the non-Standardness from $S^{(0)}$ to $\mathbb{E}[S^{(L_{\text{gzn}})}]$:

$$L_{\text{gzn}} \geq \frac{\log(1/\gamma)}{\log\left(1 - \frac{2}{D+2}\right)} \approx \log(\gamma)\frac{D+1}{2} = \Omega(D), \tag{6.16}$$

where we write the target loss ratio as $\gamma = S^{(0)}/\mathbb{E}[S^{(L_{\text{gzn}})}])$. Note that we consider the case of late training by assuming $S(0, \Sigma) \ll 1$. We explore early training in experiments in section 6.2.5 and leave extending theorems 6.2 and 6.3 to early training open for future work.

Limitations of learned rotations

The previous sections assumed that the rotations are drawn at random. However, the bulk of the literature focuses on rotating the data such that the marginals deviate as much as possible from the target normal distribution $\mathcal{N}(0, 1)$, see section 6.2.2. Alternative methods differ mainly by their measure of marginal non-Gaussianity.

Unfortunately, learning rotations has an inherent tendency to overfit on finite training sets: In this non-asymptotic regime, there is a high probability that some marginal projections exhibit considerable spurious non-Gaussianity even when the data are sampled from a perfect standard normal, $x \sim \mathcal{N}(0, I)$. In other words, although training has converged, the iterative training algorithm will identify a rotation Q that "improves" the training loss but does not generalize since spurious orientations stem from the concrete realization of the finite training data. In sufficiently high dimensions D, this still happens for large datasets with $N \gg D$, a typical situation in computer vision. Inspired by Bickel et al. (2018), we illustrate the phenomenon at $D = 3072 = 32 \times 32 \times 3$ and $N = 60000$, the dimension and size of the CIFAR10 dataset.

In figure 6.4, we show the histogram of a spurious non-Gaussian projection

$$x_{\text{proj}} = w^{\mathrm{T}} x, \qquad |w| = 1, x \sim \mathcal{N}(0, I) \tag{6.17}$$

of N fixed samples from a D-dimensional standard normal. We construct w such that the projection of the fixed data is as close as possible to the adversarial bimodal distribution shown in the plot.

Although in the asymptotic limit $N \to \infty$ no such w exists, the optimization readily finds a bimodal projection in the finite dataset. Details on the experiment can be found in appendix B.3.3.

A similar experiment is reported in Dai & Seljak (2021, Appendix D.1). They show that max K-SWD can identify spurious non-Gaussian projections. We extend their experiment by demonstrating the effect of such projections on the learned distribution. In our example, Gaussianization would fit a bimodal distribution to a standard normal.

The theoretical analysis in Wainwright (2019, Chapter 8) shows that phenomena like this are fundamental in finite datasets. Specifically, they prove that *no* method can reliably estimate the eigenvectors of the *empirical* covariance in high dimensions if the ratio D/N is bounded away from zero, unless additional assumptions (e.g. sparsity of the covariance) are made. Consequently, even the straightforward idea of defining the optimal Q via PCA can fail and must be used with caution.

In practice, we expect Gaussianization with learned rotations to work well in the initial blocks, where the intermediate latent distributions are strongly non-Gaussian. Deeper in the network, however, intermediate distributions are already close to standard normal, and spurious projections will appear. The resulting overfit of the rotation Q will fool the subsequent marginal transformation, making the data less rather than more Gaussian. This can only be fully avoided by random orthogonal matrices Q, which almost surely do not result in spurious non-Gaussian projections (Bickel et al., 2018), compare the random projection in figure 6.4. Random rotations and late training are exactly the regime of the previous theoretical results.

6.2.4. End-to-end training

The previous results assumed iterative training, that is, we construct the Gaussianization blocks one by one. However, as mentioned in section 6.2.2, Gaussianization can also be trained end-to-end, that is the blocks are adapted jointly. Does this yield a better scaling behavior? Below, we provide a simple parameter counting argument for the scaling behavior that confirms the previous scaling in equation (6.16) that $L_{\text{gzn}} = \Omega(D)$.

We consider both the case of random as well as learned rotations.

Random rotations

We will assume the following, which is typically satisfied when working with real data that are in 'general position':

Assumption 6.3. The eigenvalues of the covariance matrix Σ are distinct: $\lambda_i \neq \lambda_j$ for $i \neq j$.

We then find the following result:

Theorem 6.4 (Exact Gaussian representation using Gaussianization blocks, proof in appendix A.6.4)**.** *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ under assumption 6.3. To exactly represent $p(x)$ with* random *rotations Q, at least*

$$L_{\text{gzn}} \geq \frac{1}{2}(D+1) \tag{6.18}$$

Gaussianization blocks are required almost surely.

This is a lower bound on the required number of blocks regardless of the training approach: The number of Gaussianization blocks required grows (at least) linearly with the number of dimensions. The proof is a simple parameter-counting argument: The covariance Σ has $D(D+1)/2$ degrees of freedom, but linear Gaussianization with random rotations only has D parameters per layer. Dividing $D(D+1)/2$ by D yields the result. This is visualized in figure 6.5.

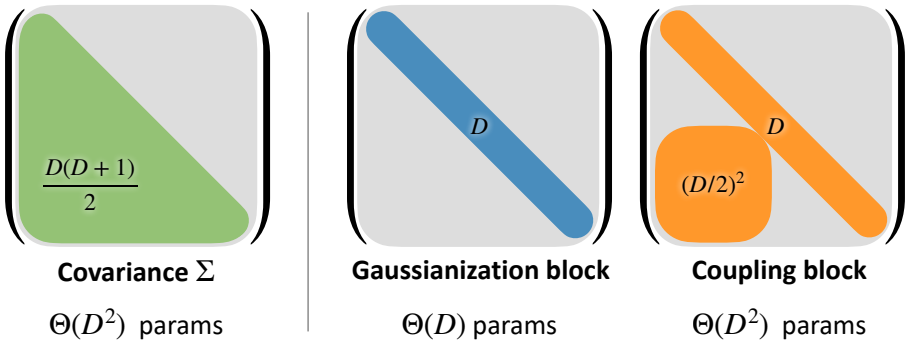


Figure 6.5.: Parameter counting argument: The goal is to transform the covariance matrix Σ to the unit matrix I. The covariance has $D(D+1)/2$ degrees of freedom, of which Gaussianization can learn D per layer, and couplings $D^2/4 + D$ per layer.

Learned rotations

Theorem 6.4 showed the scaling behavior of Gaussianization for random rotations. In this section, we show that jointly training parameterized rotations has the same scaling behavior with dimension.

End-to-end training with *learned* rotations may outperform theorem 6.4 in terms of the number of required layers. In fact, rotations exist such that arbitrary Σ can be fit with a single layer. However, state-of-the-art rotation learning typically does not train on the full orthogonal group $O(D)$, as this becomes prohibitively expensive with increasing D. Instead, one considers subsets typically spanned by $k \cdot D$ independent parameters, such as spanned by Householder transforms $I - vv^{\mathrm{T}}$ $(k = 1)$ or block-diagonal orthogonal matrices (with $k = (b-1)/2$ for block size b) (Meng et al., 2020). Even with these parameterized rotations, however, the number of required layers scales with the dimension D:

Corollary 6.5 (Exact representation with learned rotations, proof in appendix A.6.5)**.** *Given a multivariate non-degenerate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ under assumption 6.3. To exactly represent $p(x)$ with* learned *rotations Q with $k \cdot D$ parameters each, at least*

$$L_{\mathrm{gzn}} \geq \frac{1}{2(k+1)} D \tag{6.19}$$

Gaussianization layers are required almost surely.

The scaling of L_{gzn} with D can only be avoided when $k = \Omega(D)$, which does not hold for the parameterizations mentioned above.

6.2.5. Empirical extrapolation to non-Gaussian data and early training

Theorem 6.3 predicts the convergence of Gaussianization on Gaussian input $p(x) = \mathcal{N}(0, \Sigma)$ at late training. The core result is that the number of blocks increases linearly with the dimension.

We now lift these restrictions and consider $p(x) \neq \mathcal{N}(0, \Sigma)$ at early training. We find that as the dimension increases, the number of required layers L_{gzn} to reduce the loss by a factor increases with dimension D, but favorable scaling can be achieved depending on the properties of the data. We base our implementation of Gaussianization blocks on the code provided by SINF (Dai & Seljak, 2021), see appendix B.3.4 for all details.

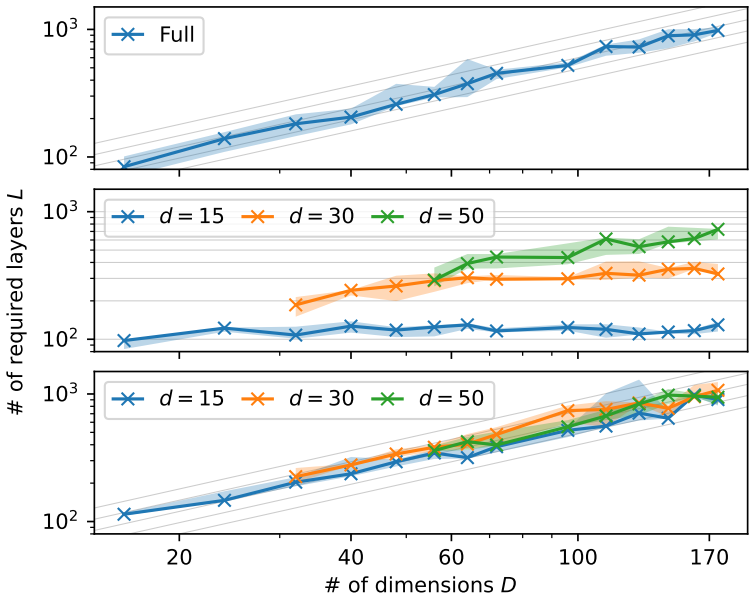


Figure 6.6.: Required layers of Gaussianization on toy data. *(Top)* If all dimensions depend on one another, the number of required layers increases linearly with dimension. *(Middle)* If trailing dimensions $i > d$ are pairwise independent given the core d dimensions, only about a constant number of layers is sufficient for fixed d. *(Bottom)* If the trailing dimensions $i > d$ are independent Gaussian noise, the number of layers increases linearly with dimension. Shaded regions indicate 100% of the training runs. Gray lines indicate $\Theta(D)$ resp. $\Theta(1)$.

Toy scaling experiment

For determining the scaling behavior of Gaussianization we consider a family of distributions of varying dimension D. We propose to build such a toy distribution autoregressively:

$$p(x) = p(x_1) \prod_{i=2}^{D} p(x_i|A_i), \tag{6.20}$$

where the set $A_i \subseteq \{x_1, \dots, x_{i-1}\}$ collects the random variables that x_i depend upon. This allows adding new dimensions by specifying their dependencies.

We consider the following three variants: (1) Let every variable depend on all previous variables: $A_i^{(1)} = \{x_1, \dots, x_{i-1}\}$. (2) We only make a subset of d variables depend on all previous, and let the remaining dimensions depend on this fixed subset of dimensions: $A_{i\leq d}^{(2)} = \{x_1, \dots, x_{i-1}\}$ (core) and $A_{i>d}^{(2)} = \{x_1, \dots, x_d\}$ (remainder). (3) Like the second case, but the remaining dimensions $i > d$ are independent Gaussian noise: $A_{i\leq d}^{(2)} = A_{i\leq d}^{(3)}$ (core) and $A_{i>d}^{(3)} = \emptyset$ (noise).

In particular, we choose $p(x_i|A_i)$ as a continuous mixture of Gaussians

$$p(x_1) = \mathcal{N}(m_1, \sigma_1^2), \quad p(x_i|A_i) = \mathcal{N}(m_i(A_i), \sigma_2^2) \tag{6.21}$$

where the dependencies are introduced through $m_i(A_i)$:

$$m_i(A_i) = m_0 + 5 \tanh\left(\frac{1}{10} \sum_{x_j \in A_i} s_{ij} x_j^2\right). \tag{6.22}$$

The values $m_1, m_0 \in \mathbb{R}; \sigma_1, \sigma_2 \in \mathbb{R}_+, s_{ij} \in \{-1, 1\}$ are parameters to the distribution.

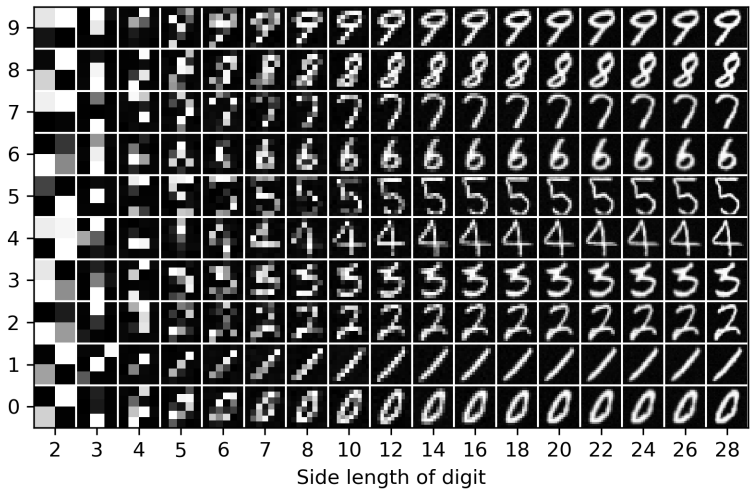


Figure 6.7.: Our multiscale EMNIST digits dataset.

Figure 6.6 shows how many layers L_{gzn} are needed to reduce the loss by a fixed ratio $\gamma = \tilde{\mathcal{L}}/\mathcal{L} < 1$ for each case as a function of dimension D. We find that for cases (1) and (3), the number of required Gaussianization layers increases linearly with dimension, which is consistent with our theoretical result on Gaussian data in equation (6.16). In case (2), however, the number of required layers remains roughly constant with dimension (but it does depend on the number of dependent dimensions).We show in figure B.6 in appendix B.3.5 that random projections are less Gaussian in this case, as additional variables carry information about the core dimensions. This makes it easier for Gaussianization to fit the data, efficiently removing loss by fitting the non-Gaussian marginals.

This toy experiment indicates that linear increase in required layers holds for some distributions, and a favorable scaling behavior may be obtained for certain input if random projections are non-Gaussian. Note that these experiments estimate the number of layer from initial training. This does not contradict our theoretical results that consider Gaussian marginals and late training.

Real dataset experiment

We now consider the scaling behavior of Gaussianization on a real dataset, the EMNIST digits (Cohen et al., 2017). To measure the scaling with dimensions, we construct variants of the data with different dimensions. We therefore rescale the images to scales between 2×2 and the original 28×28, see figure 6.7.

Figure 6.8 shows how many layers are required as a function of dimension, extrapolated from training 64 Gaussianization layers. We find that up to a scale of $D = 10 \times 10$, the estimated number of required layers roughly increases as $\Theta(D)$, like in equation (6.16), and then remains about constant.

Consistently, around a side length $l = 10$, the main characteristics of each digit become clear. Afterwards, only local details are filled in. We identify the corresponding scaling of Gaussianization with case (2) in section 6.2.5, where additional dimensions are highly correlated with others, so that random marginal distributions become less Gaussian.

Note that for computing the absolute KL divergence as defined in equation (4.4), we need to evaluate the entropy of the data $H[p(x)]$. This is an unknown value in general, and for rescaled EMNIST digits in particular. We therefore replace the ground truth dataset by generative models trained on each respective scale. We use a coupling-based normalizing flow for our ground truth distribution, which achieves better density estimates in general than Gaussianization. This removes bias from our convergence rate estimates: If we were to use the trained model itself as ground truth,

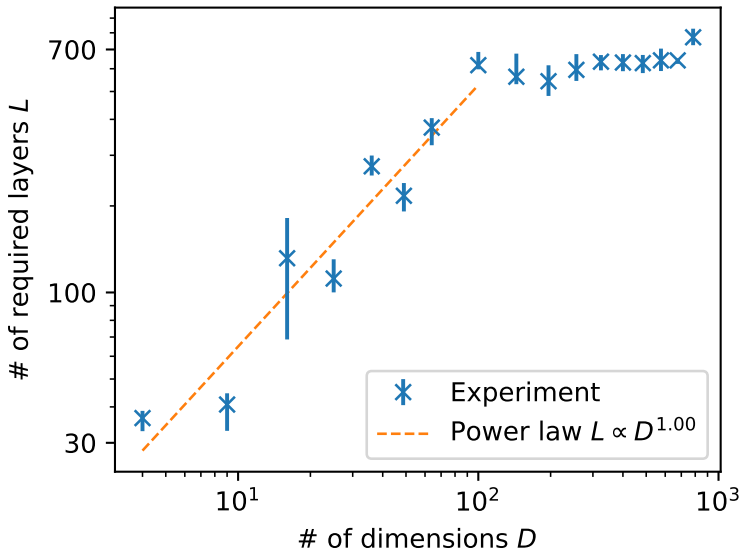


Figure 6.8.: Gaussianization requires more layers for higher resolution datasets. From 2×2 to 10×10, fitting a power law yields a linear function. For larger images, the difficulty does not increase further. We think that this is due to an increasing number of pixels being largely determined from few other dimensions. Data points show the median and error bars cover 90% of the training runs.

we would find spuriously fast convergence even if the model does not converge – as we might compute the convergence rate to a suboptimal optimum. See appendix B.3.6 for all experimental details.

6.2.6. Conclusion

Gaussianization is a simple generative model, both trained end-to-end and iteratively (Meng et al., 2020; Dai & Seljak, 2021). Differently from other approaches, it can be trained without backpropagation and neural networks, yet provides useful density estimates and samples in low and moderate dimensions.

Scaling Gaussianization to high dimensions remains a major challenge, and we confirm this rigorously. We show analytically for the Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ that the number of required layers in late training typically scales with the dimensionality of the problem if the rotations Q are chosen at random. On non-Gaussian distributions, we find that convergence can be favorable for some distributions in early training.

This yields a first data point on the tradeoff between few unconstrained yet expensive layers and constrained yet cheap layers. Gaussianization requires many layers, being limited to learning marginals in each step.

An important point remains open, however: Improving Gaussianization to high dimensions may be possible by constructing better rotations Q. Our work points out a fundamental limitation of this approach: There exist spurious non-Gaussian directions that may be spuriously identified, yielding overfitting distributions. It remains open to theoretically describe how many Gaussianization blocks are required to fit real-world distributions.

6.3. Coupling blocks

Unless otherwise noted, this section is adapted from Draxler et al. (2022).

Coupling-based normalizing flows are the backbone of many of the successful applications of normalizing flows listed in chapter 2. Here, we complement the universality theory on coupling flows in

section 5.4 with an explicit convergence rate for learning the first two moments of the input distribution.

Specifically, we make the following contributions:

- The contribution of a single coupling layer on the non-Standardness is analyzed in terms of matrix operations (Schur complement and scaling).
- We derive explicit bounds for the non-Standardness after a single coupling block in expectation over all possible rotations.
- We use these results to prove that a sequence of coupling blocks learns the first two moments of the data and to derive linear convergence rates for this process. We find that rate to be largely independent of dimension in late training.

We confirm our theoretical findings experimentally. They hold for all coupling functions we are aware of (see section 4.3.1) and give an indication why coupling flows are a popular choice for applications as opposed to Gaussianization. We provide the code to our experiments at `https://github.com/vislearn/Coupling-Flow-Bound`.

6.3.1. Related work

Analyzing the universality of coupling-based normalizing flows is an active area of research, and we list the bulk of the literature in section 5.4.1. We give additional details here as far as they concern the resources required to learn a given distribution.

The universality guarantee by Koehler et al. (2021) states that three coupling blocks are enough to learn an arbitrary input distribution in terms of Wasserstein divergence. While technically correct, their construction comes with strong restrictions preventing their results to being translated into a practical recommendation: The underlying normalizing flow learns a volume-preserving transformation, which is not universal in terms of KL divergence by our theorem 5.3. Also, it relies heavily on ill-conditioned networks and the derivatives explode as the accuracy increases, regardless of the input distribution. We provided details on these points in section 5.4.2 and appendix A.4.4.

In fact, none of the existing universality guarantees for coupling flows concern the KL divergence (Huang et al., 2020; Teshima et al., 2020a; Zhang et al., 2020a; Ishikawa et al., 2022). While our results only consider fitting the first two moments of the data, they do give rigorous guarantees in this strong convergence metric.

Closely related to our work, at most 48 linear affine coupling blocks are required to represent any invertible linear function $Ax + b$ with $\det(A) > 0$ (Koehler et al., 2021, Theorem 2). This also allows mapping any Gaussian distribution $\mathcal{N}(m, \Sigma)$ to the standard normal $\mathcal{N}(0, I)$. We complement their result with a lower bound on the number of layers and give several statements in terms of the part of the non-Standardness, the part of KL divergence that sensitive to the first two moments. We then make strong statements about the convergence of the latter.

6.3.2. Convergence of non-Standardness

Like for Gaussianization, we derive the scaling behavior of reducing the non-Standardness via coupling blocks trained iteratively. Since end-to-end training can also converge to the solution we construct below, the number of layers we find for iterative training is an upper bound for the number of layers for end-to-end training.

Single coupling layer minimizer

We refer to section 4.3 for an introduction to normalizing flows and invertible neural networks based on coupling layers. We decide to also apply the coupling function on the passive dimensions, as this simplifies derivations. For learning the first two moments, this is equivalent to not transforming the passive dimensions and instead appending an ActNorm layer, see equation (4.26).

As mentioned in section 6.1, we restrict the couplings to affine-linear functions, which ensures that we reduce the non-Standardness S without increasing the non-Gaussianity G in turn. Under this constraint, the coupling layer transforms an incoming vector $x \in \mathbb{R}^D$ as follows:

$$\begin{pmatrix} \tilde{b} \\ \tilde{a} \end{pmatrix} = f_{\text{cpl}}(Qx) = r \odot \begin{pmatrix} I & 0 \\ T & I \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} + u. \tag{6.23}$$

This is consistent for all architectures listed in section 4.3.1, and thus our **results apply to all non-volume-preserving coupling functions**.

Our first result shows which mean $\tilde{m}$ and covariance $\tilde{\Sigma}$ a single affine-linear coupling as in equation (6.23) yields to minimize $S(\tilde{m}, \tilde{\Sigma})$ given data with mean m and covariance Σ, rotated by Q:

Proposition 6.6 (Moments after single coupling layer, roof in appendix A.7.1)**.** *Given D-dimensional data with mean m and covariance Σ and a rotation matrix Q. Split the covariance of the rotated data into four blocks, corresponding to the passive and active dimensions of the coupling layer:*

$$Q\Sigma Q^{\mathrm{T}} = \begin{pmatrix} \Sigma_{bb} & \Sigma_{ba} \\ \Sigma_{ab} & \Sigma_{aa} \end{pmatrix} \tag{6.24}$$

Then, the moments $\tilde{m}, \tilde{\Sigma}$ that can be reached by a linear coupling layer as in equation (6.23) *are:*

$$\tilde{m} = 0, \qquad \tilde{\Sigma}(Q) = \begin{pmatrix} M(\Sigma_{bb}) & 0 \\ 0 & M(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}) \end{pmatrix}. \tag{6.25}$$

This minimizes the non-Standardness $S(\tilde{m}, \tilde{\Sigma})$ as given in equation (6.1)*, and G does not change.*

The function M scales the marginal standard deviations to one, as defined as in equation (6.9).

Proposition 6.6 shows how the covariance can be brought closer to the identity. The new covariance has passive and active dimensions uncorrelated. In the active subspace, the covariance is the Schur complement $\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$. This coincides with the covariance of the Gaussian $\mathcal{N}(0, \Sigma)$ as it is conditioned on any passive value p. Afterward, the diagonal is rescaled to one, matching the standard deviations of all dimensions with the desired latent code. Proposition 6.6 is the variant of lemma 5.5 if the coupling block is restricted to be linear and the passive dimensions are normalized.

Figure 6.9 shows an experiment in which a single affine-linear layer was trained to bring the covariance of EMNIST digits (Cohen et al., 2017) as close to I as possible (details in appendix B.4.2). The experimental result coincides with the prediction by proposition 6.6. Due to the finite batch-size, a small difference between theory and experiment remains.

Single coupling block guarantees

Proposition 6.6 allows the computation of the minimum non-Standardness after a single coupling block given its rotation Q, by evaluating $S(\tilde{m}, \tilde{\Sigma}(Q))$. Just like for Gaussianization, if we were to choose $Q = Q^*$ such that the data is rotated so that principal components lie on the axes (i.e. obtain Q^* using PCA), a single coupling block suffices to reduce the covariance to the identity: The rotated covariance

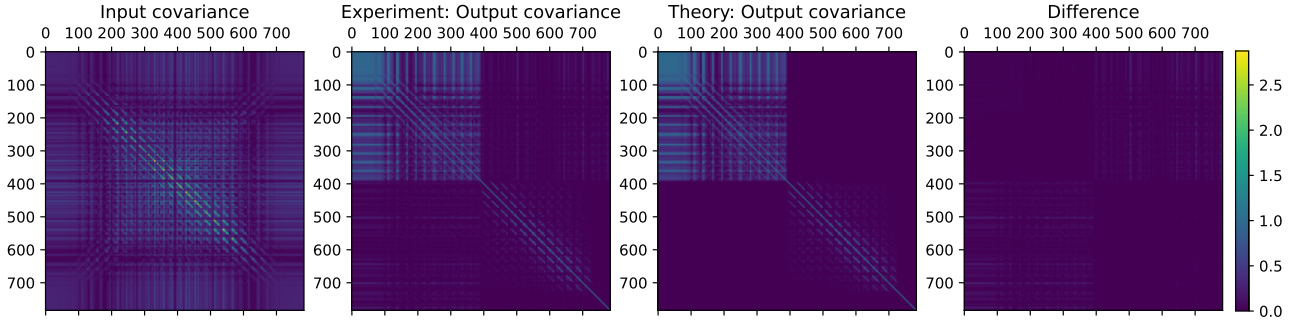


Figure 6.9.: How a single coupling layer can whiten the covariance at the example of the EMNIST digits covariance matrix *(first panel)*. The covariance after a single layer trained experimentally to minimize non-Standardness $S(\tilde{m}, \tilde{\Sigma})$ *(second panel)*, which matches closely the prediction of proposition 6.6 *(third panel)*. The difference between theory and experiment vanishes *(last panel)*.

$Q^* \Sigma Q^{*,\mathrm{T}}$ would be a diagonal matrix and $\tilde{\Sigma}(Q^*) = I$. This is not the case in practice, where this optimal orientation has zero probability: Q is chosen uniformly at random before training from all orthogonal matrices. One could argue that one should whiten the data before passing it to the flow, reducing S to zero from the start. However, we are about to show that coupling-based normalizing flows are already well-equipped to bring the non-Standardness to zero making such modifications unnecessary.

We make essentially the same mild assumptions on the considered input as for Gaussianization. They are part of usual data-preprocessing, when the mean is subtracted from the data $m = 0$ (assumption 6.2) and all data points are divided by the scalar $\sqrt{\operatorname{tr} \Sigma / D}$ so that $\operatorname{tr} \Sigma = D$ (assumption 6.1, not to be confused with diagonal preconditioning, which acts dimension-wise). The assumptions simplify the non-Standardness in equation (6.1) to only depend on the determinant of Σ, see equation (6.10): We aim to compute the average non-Standardness after a single block $\mathbb{E}_{Q \in p(Q)}[S(\tilde{\Sigma}(Q))]$. For any Q, $S(\tilde{m}, \tilde{\Sigma}(Q))$ is again given by the determinant of the covariance $\tilde{\Sigma}(Q)$ as assumptions 6.1 and 6.2 remain fulfilled: By proposition 6.6, $\tilde{m} = 0$, and the diagonal preconditioning M ensures that the trace of $\tilde{\Sigma}$ is D. We write $\det(\tilde{\Sigma})$ via M_a and M_p, the diagonal matrices that make up the diagonal preconditioning in equation (6.9), and use the Schur determinantal formula for the determinant of block matrices: $\det(\Sigma_{bb}) \det(\Sigma_{aa} - \Sigma_{ab} \Sigma_{bb}^{-1} \Sigma_{ba}) = \det(Q \Sigma Q^{\mathrm{T}}) = \det(\Sigma)$ (Horn & Johnson, 2012). By using that $\det(AB) = \det(A) \det(B)$ for square matrices A, B, we get:

$$\det(\tilde{\Sigma}) = \det(M_p \Sigma_{bb} M_p) \det(M_a (\Sigma_{aa} - \Sigma_{ab} \Sigma_{bb}^{-1} \Sigma_{ba}) M_a) = \det(M_p^2) \det(M_a^2) \det(\Sigma). \tag{6.26}$$

Inserting this into equation (6.10), we find:

$$S(0, \tilde{\Sigma}) = -\tfrac{1}{2} \Big(\log\det \Sigma + \log\det M_p^2 + \log\det M_a^2 \Big) \leq S(\Sigma). \tag{6.27}$$

The inequality $S(\tilde{\Sigma}) \leq S(\Sigma)$ holds because $\tilde{\Sigma} = Q^{\mathrm{T}} \Sigma Q$ is an admissible solution of the coupling layer optimization, and $\tilde{\Sigma}$ as given by proposition 6.6 is a minimizer of $S(\tilde{\Sigma})$.

We average this quantity over training runs, i.e. over rotations Q:

$$\mathbb{E}_{Q \sim p(Q)}[S(\tilde{\Sigma})] = -\tfrac{1}{2} \Big(\log\det \Sigma + \mathbb{E}_{Q \sim p(Q)}[\log\det M_p^2] + \mathbb{E}_{Q \sim p(Q)}[\log\det M_a^2] \Big). \tag{6.28}$$

The main difficulty lies in the computation of $\mathbb{E}_{Q \sim p(Q)}[\log\det M_a^2]$. Here, we contribute the two strong statements theorems 6.7 and 6.8 below.

Precise guarantee The first result is computed using projected orbital measures (Olshanski, 2013). This theory describes the eigenvalues of submatrices of matrices in a random basis. We require such a result for integrating over $p(Q)$ in $\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2]$. In contrast to the typical choice of picking Q from the Haar measure over $O(D)$, the theory to the best of our knowledge only covers data rotated by unitary matrices $U(D)$.[1] To comply with (Olshanski, 2013), we thus make two more assumptions. The first assumption is that the eigenvalues are pairwise distinct (assumption 6.3), which is easily satisfied when working with real data in 'general position'. In addition, we switch to unitary rotations:

Assumption 6.4. The distribution of rotations is the Haar measure over *unitary* matrices $U(D)$.

One could think that the step from orthogonal to unitary rotations takes us far away from the scenario we want to consider. We will later observe empirically that the difference between averaging over unitary and orthogonal matrices is negligible. Technically, the covariance matrix remains positive definite, so the non-Standardness S is always real (see appendix A.7.2). We will write $\mathbb{E}_{Q\sim U(D)}[\,\cdot\,]$ to denote expectations over the Haar measure over the unitary group.

We are now ready to **compute the expected non-Standardness** after a single coupling block:

Theorem 6.7 (Non-Standardness after coupling block, proof in appendix A.7.2)**.** *Given D-dimensional data with covariance Σ with eigenvalues $\lambda_1,\dots\lambda_D$. Assume that assumptions 6.1, 6.3 and 6.4 hold. Then, after a single coupling block, the expected non-Standardness is bounded from above:*

$$\mathbb{E}_{Q\in U(D)}[S(\tilde{m},\tilde{\Sigma}(Q))] < S(0,\Sigma) + \frac{D}{2}\log\left((-1)^{\frac{D}{2}+1}\sum_{i=1}^{D}\lambda_i^{1-\frac{D}{2}}\log(\lambda_i)R(\lambda_i^{-1};\lambda_{\neq i}^{-1})e_{\frac{D}{2}-1}(\lambda_{\neq i}^{-1})\right). \quad (6.29)$$

Here, $\lambda_{\neq i} := \{\lambda_1,\dots,\lambda_{i-1},\lambda_{i+1},\dots,\lambda_D\}$ and R, e_K are given by:

$$R(a;\{b_i\}_{i=1}^N) = \prod_{i=1}^{N}\frac{1}{a-b_i} \quad \textit{and} \quad e_K(\{b_i\}_{i=1}^N) = \sum_{0<i_1<\dots<i_K\leq N} b_{i_1}\cdots b_{i_K}. \quad (6.30)$$

Inequality (6.29) sharply bounds the expected non-Standardness that can be achieved by a single block. The only approximation made is an inequality which comes close to equality as the dimension D increases due to the concentration of the corresponding probability distribution.

Figure 6.10 shows an experiment confirming theorem 6.7 (details in appendix B.4.3). We start with covariance matrices using parameterized eigenvalue spectra. To each initial matrix, we first apply a single coupling block with random Q and determine the coupling that maximally reduces S using proposition 6.6. Then we iteratively append 32 additional blocks in the same manner, building a flow of that depth. We average the resulting empirical ratio $S(0,\tilde{\Sigma}(Q))/S(0,\Sigma)$ over several *orthogonal* orientations Q of the rotation layer for each input covariance matrix. Then, we compare this to (i) experimentally averaging over *unitary* rotations and (ii) to the prediction by theorem 6.7 and confirm that it is a valid and close upper bound. Details for replication and more examples can be found in appendix B.4.3.

The proof explicitly integrates $\mathbb{E}[M_a^2]$ using (Olshanski, 2013), see appendix A.7.2. Numerically evaluating equation (6.29) can be hard even for small D as the summands scale as $\mathcal{O}(\exp(D))$, but the overall sum scales as $\mathcal{O}(D)$. High values cancel due to R alternating in sign, and one requires arbitrary-precision floating-point software to evaluate equation (6.29). We use `mpmath` for computations (Johansson et al., 2010).

[1] The only result known to us on the orthogonal group would only yield predictions for $D = 2$ (Faraut, 2015), whereas we are interested in large D.

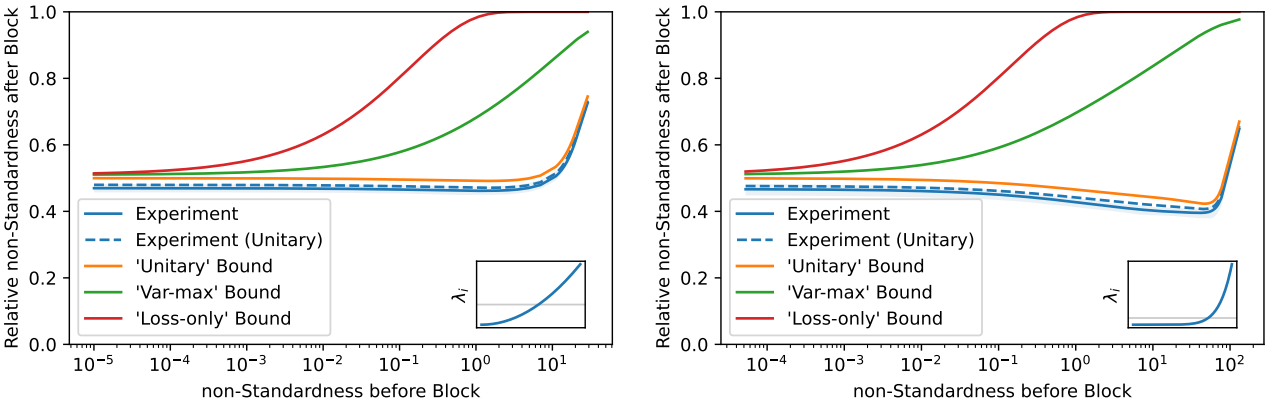


Figure 6.10.: Comparison between predicted non-Standardness and experiment for 48-dimensional parametrized eigenvalue spectra *(insets)*, varied over a parameter which controls the spread of the spectrum and thus changes S. The experimental average over *orthogonal* rotations matrices (blue, shaded by Interquartile Range IQR) is closely matched by the experimental average over *unitary* matrices (dotted blue). The prediction by theorem 6.7 is a close upper bound that closely matches the experimental behavior (orange). The predictions by theorem 6.8 are less precise, but converge to the same value as the precise bound for covariances close to the identitiy: 'Var-max' is equation (6.31) (green) and 'Loss-only' is equation (6.32) (red). More details and examples in appendix B.4.3.

Interpretable guarantee The guarantee in theorem 6.7 yields useful predictions, but it does not lend itself to further analysis: How does the bound behave over several coupling blocks? What is the behavior for varying dimension D? Also, assumption 6.4 restricts formal reasoning as we are interested in averaging over orthogonal and not unitary rotations. Our second single-block guarantee depends only on simple metrics of the covariance. Moreover, we drop assumptions 6.3 and 6.4, averaging over orthogonal, not unitary, Q:

Theorem 6.8 (Interpretable non-Standardness after coupling block, proof in appendix A.7.3)**.** *Given D-dimensional data with covariance Σ fulfilling assumption 6.1 with covariance eigenvalues $\lambda_1, \dots \lambda_D$. Then, after a single coupling block, the expected loss can be bounded from above:*

$$\mathbb{E}_{Q\in O(D)}[S(0,\tilde{\Sigma}(Q))] \leq S(0,\Sigma) + \frac{D}{4}\log\left(1 - \frac{D^2}{2(D-1)(D+2)}\frac{\mathrm{Var}[\lambda]}{\lambda_{\max}}\right) \tag{6.31}$$

$$\leq S(0,\Sigma) + \frac{D}{4}\log\left(1 - \frac{D^2}{(D-1)(D+2)}\frac{1-\sqrt{1-g^D}}{1+\sqrt{1-g^D}}(1-g)\right) \tag{6.32}$$

$$< S(m,\Sigma). \tag{6.33}$$

Here, g is the geometric mean of the eigenvalues: $g = \prod_{i=1}^{D} \lambda_i^{1/D} = \exp(-2S(0,\Sigma)/D) < 1$ which is a bijection of $S(0,\Sigma)$.

These two new bounds on the average achievable non-Standardness S after a single block are also depicted in figure 6.10. They make useful predictions, but are less precise than theorem 6.7. The second bound will be especially useful in what follows because it only depends on the non-Standardness before the block $S(0,\Sigma)$.

The full proof is given in appendix A.7.3. It relies on the integration of monomials of entries of random orthogonal matrices as described by (Gorin, 2002) and the arithmetic mean-geometric mean inequality by (Cartwright & Field, 1978).

The first bound hints at the beneficial scaling behavior of coupling blocks: The performance only weakly depends on the dimension. To see this, divide equation (6.31) by D to obtain a statement

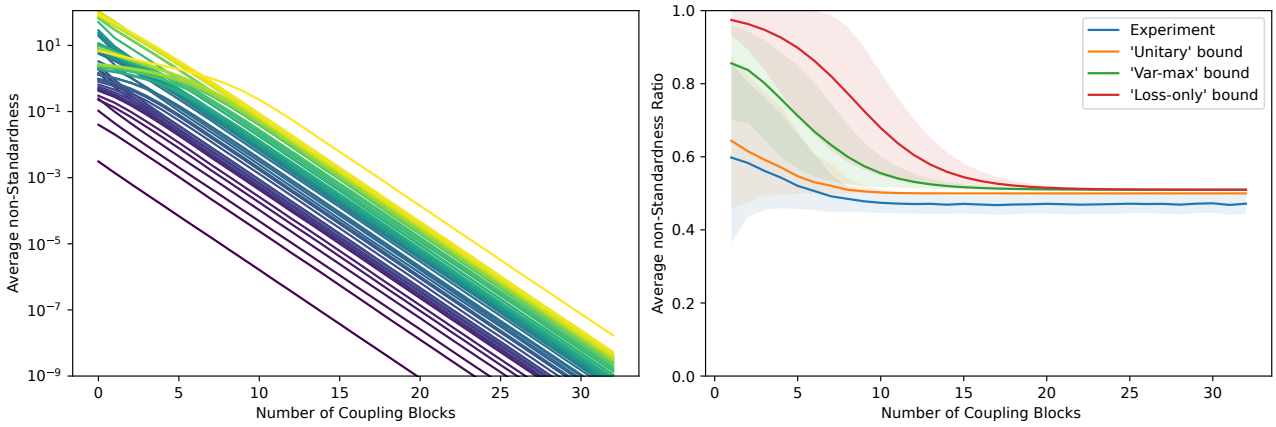


Figure 6.11.: Deep network convergence of covariance on toy dataset. *(Left)* Each line shows the experimental convergence of S via the repeated application of proposition 6.6, averaged over 32 runs with different rotations Q. *(Right)* The empirical convergence rate (blue), i.e. the ratio of S before and after a block, is correctly bounded from above by our predictions in theorem 6.7 (orange), and the bounds in theorem 6.8: Equation (6.31) (green) and equation (6.32) (red). The solid lines show the ratio (bounds) averaged over the toy dataset and rotations, the shade is the IQR. The experiment suggests that a convergence rate like theorem 6.9 can also be derived for the remaining bounds.

about the non-Standardness per dimension S/D. Then keep the variance and the maximum of covariance eigenvalues fixed. The guarantee is then approximately constant in D, it varies slightly with $D^2/(D^2 + D - 2)$, which is always close to 1.

Deep network guarantee

The previous results were concerned with determining how much a *single* coupling block can typically contribute towards reducing the non-Standardness S to zero. Now, we extend this result to compute the expected non-Standardness after a *deep* coupling-based normalizing flow as an explicit function of the number of blocks. We again treat the rotation layer of each block as a random variable, as it is randomly determined before training.

We find that the **convergence rate** of the covariance to the identity is (at least) **linear**:

Theorem 6.9 (Deep coupling flow bound, proof in appendix A.7.4)**.** *Given D-dimensional data with covariance Σ Given D-dimensional data with covariance Σ fulfilling assumptions 6.1 and 6.2. Then, after L_{cpl} coupling blocks, the expected loss is smaller than:*

$$\mathbb{E}_{Q_1,\ldots,Q_{L_{\text{cpl}}} \in O(D)}[S(0, \Sigma_{L_{\text{cpl}}})] \leq \gamma(S(0,\Sigma))^{L_{\text{cpl}}} S(0,\Sigma), \tag{6.34}$$

where the convergence rate depends on the non-Standardness before training:

$$\gamma(S) = 1 + \frac{1}{4S/D} \log\left(1 - \frac{D^2}{(D-1)(D+2)} \frac{1 - \sqrt{1-g(S)^D}}{1 + \sqrt{1-g(S)^D}} \Big(1 - g(S)\Big)\right) < 1. \tag{6.35}$$

The non-Standardness decreases at least exponentially fast in the number of blocks. The convergence rate that holds for a deep network is computed using the non-Standardness of the input data $S(0,\Sigma)$. This rate comes from equation (6.32). The proof uses that $\gamma(S)$ improves from block to block as S decreases (see appendix A.7.4). Again, $g(S) = \exp(-2S/D) < 1$ is the geometric mean of eigenvalues of Σ, which increases from block to block.

Figure 6.11 shows the convergence of the non-Standardness to zero in an experiment. We build a toy dataset of various covariances, where we aim to capture a plethora of possible cases (see appendix B.4.4). We apply a single coupling block with random Q and the coupling that maximally reduces S via proposition 6.6. We iteratively add such blocks 32 times, building a flow of that depth. The resulting convergence of S as a function of depth is averaged over 32 runs with different rotations. The measured curve confirms theorem 6.9. We find that the rate γ in equation (6.35) is correct, but several experiments show even faster convergence in practice. Indeed, the experiments suggest that dividing all upper bounds for $\mathbb{E}[S(0,\tilde{\Sigma})]$ in theorems 6.7 and 6.8 by $S(0,\Sigma)$ also bounds the non-Standardness ratio for subsequent blocks. Formally, we conjecture that $\mathbb{E}[S(0,\Sigma_{L_{\text{cpl}}})]/S(0,\Sigma) \leq (B/S(0,\Sigma))^{L_{\text{cpl}}}$ where B is the rhs. of equations (6.29) and (6.31) (theorem 6.9 shows exactly this for equation (6.32)). We leave a proof or falsification of this conjecture open to future work.

The bounds evaluated for the experimental covariances also suggest that all bounds agree after a few blocks, leaving only a small gap to the actual experiment. We can explicitly compute this limit value of $\gamma(S)$ by taking $S \to 0$:

$$\gamma(S) \xrightarrow{S\to 0} \frac{D(D+2)-4}{2(D-1)(D+2)} \in \Big[1/2, 5/9\Big]. \tag{6.36}$$

The two experimental observations together with this limit value suggest the heuristic that **a single additional coupling block typically reduces the non-Standardness S by a factor of approximately 50%** if previous blocks are left unchanged, and possibly faster if cooperations between blocks are considered.

This following result was derived in Draxler et al. (2023).

In terms of how many layers are required to reduce the non-Standardness by a prescribed factor, we can condense theorem 6.9 to have the same format as theorem 6.3:

Corollary 6.10 (Coupling guarantee for low loss, proof in appendix A.6.6)**.** *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0,\Sigma)$. The initial loss S is given by equation* (6.10)*. Then, in the case $S \ll 1, 1 \ll D$, the loss after L_{cpl} iterative coupling blocks with random rotations is at most:*

$$\mathbb{E}_{Q_{1\dots L}\in O(D)}[S^{(L_{\text{cpl}})}] \lesssim \left(\frac{1}{2}\right)^{L_{\text{cpl}}} S. \tag{6.37}$$

In parallel to the previous result for Gaussianization in equation (6.16), we derive how many coupling blocks are required to reduce the loss by a factor $\gamma = S/\mathbb{E}[\tilde{S}^{(L_{\text{cpl}})}]$:

$$L_{\text{cpl}} \lesssim \frac{\log(\gamma)}{\log(2)} = \mathcal{O}(1). \tag{6.38}$$

6.3.3. Minimum number of layers

Like for Gaussianization, we are interested in deriving how many layers are required to fit the covariance and in particular how this number changes as the dimension of the data varies.

Applying the same parameter counting argument as in theorem 6.4 for Gaussianization blocks to coupling blocks yields a first glance on how many coupling blocks are required to fit $p(x) = \mathcal{N}(0,\Sigma)$:

Corollary 6.11 (Exact representation with coupling blocks, proof in appendix A.7.5)**.** *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0,\Sigma)$ under assumption 6.3. To exactly represent $p(x)$, at least*

$$L_{\text{cpl}} \geq 2 \in \Omega(1) \tag{6.39}$$

coupling blocks with random rotations $Q \in O(D)$ are required almost surely.

This result says that the number of coupling blocks required to represent $\mathcal{N}(0, \Sigma)$ is independent of dimension, or worse.

Just like corollary 6.11 gives a lower bound on the number of blocks required, Koehler et al. (2021, Theorem 2) provide an upper bound, which we condense for simplicity:

Theorem 6.12 (Koehler et al. (2021))**.** *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$. To exactly represent $p(x)$, at most $L_{\text{cpl}} \leq 48 \in \mathcal{O}(1)$ coupling blocks with block permutations are required.*

This ensures that at most 48 blocks or $\mathcal{O}(1)$ are required to fit a Gaussian exactly $p(x) = \mathcal{N}(0, \Sigma)$. While this may seem like a large number, the result crucially ensures that the number of required blocks is independent of the dimension. In the statement, block permutations refer to all rotations switching all active and passive dimensions:

$$Q = \begin{bmatrix} 0 & I_{D/2} \\ I_{D/2} & 0 \end{bmatrix}. \tag{6.40}$$

Odd and even layers each modify one half of the dimensions respectively. This is a popular choice in practice.

Together, corollary 6.11 and theorem 6.12 state that for exactly representing $\mathcal{N}(0, \Sigma)$, $\Theta(1)$ coupling layers are required.

6.3.4. Conclusion

To the best of our knowledge, this is the first work on coupling-based normalizing flows that provides a quantitative convergence analysis in terms of the KL divergence. Specifically, a minimal convergence rate is established at which flows whiten the covariance of the input data under this strong measure of discrepancy of probability distributions. Splitting the loss into the non-Gaussianity G and the non-Standardness S, we show that this whitening is a necessary condition for the flow to converge and give explicit guarantees. Our derivations suggest the rule of thumb that S can typically be reduced by about 50% per coupling block.

Our central idea was to separate out the contribution a single isolated block can make to reduce the loss. Then, we combine the effect of many isolated blocks, disregarding potential further improvements to S due to joint, cooperative learning of all blocks. This simplifies the theoretical analysis, but it is not a restriction on the model: Any function that is achieved in block-wise training could also be the solution of end-to-end training.

6.4. Architecture comparison

Together, sections 6.2 and 6.3 draw a compelling picture of the behavior of different invertible block architectures. Theorems 6.3, 6.4 and 6.12 and corollaries 6.10 and 6.11 imply that the number of the different invertible layers required to learn the first two moments of the input scales as follows with dimension D:

$$\text{Gaussianization:} \quad L_{\text{gzn}} = \Omega(D), \tag{6.41}$$

$$\text{Coupling:} \quad L_{\text{cpl}} = \Theta(1), \tag{6.42}$$

$$\text{Autoregressive:} \quad L_{\text{ar}} = 1. \tag{6.43}$$

We did not explicitly derive the scaling of the autoregressive layer, but the result immediately follows from the Knothe-Rosenblatt rearrangement that any distribution can be represented by an expressive enough autoregressive layer (Rosenblatt, 1952; Knothe, 1957).

This suggests that coupling blocks lie in a sweet spot between expressivity and computational efficiency – only a constant factor away from autoregressive blocks, despite the greatly improved inference speed by removing the sequential sampling. The central advantage over Gaussianization is that they can model dependencies between dimensions in a constant number of blocks.

The same $L_{\text{rec}} = \Theta(1)$ holds for recursive coupling layers such as HINT (Kruse et al., 2021), and they can achieve $L_{\text{rec}} = 1$ if the dimension is a power of 2. However, this again comes at the cost of a sequential sampling scheme in each layer and the maximum number of hierarchies $K = \log_2 D$ has to be used.

In terms of wall clock time, let's estimate the number of required floating-point operations (FLOPs). Both Gaussianization and coupling layers have to evaluate the coupling functions $c(a_i; \theta_i)$, so evaluating them once for each dimension yields $\mathcal{O}(D)$ FLOPs. A coupling block requires the computation of the active parameters $\theta = \psi(b)$ by a neural network for a coupling block. This scales as $\mathcal{O}(D^2)$ from the scaling of a matrix-vector product if the number of hidden dimensions is scaled linearly with dimension (a common choice). However, both layers are concatenated with a rotation layer, which involves multiplication of a vector with a $D \times D$ matrix, so both kinds of invertible block overall scale as $\mathcal{O}(D^2)$. Overall, this makes coupling flows scale a power of dimension D faster with dimension, as they require fewer blocks.

We hope that the results developed here and leveraged from existing work will be helpful for the generative model community at large. In particular, deriving useful quantities for arbitrary distributions $p(x)$ is often difficult. Given that the above results correlate with empirical findings for Gaussianization (Dai & Seljak, 2021) and coupling blocks (Kingma & Dhariwal, 2018), a statement on a simple distribution such as a Gaussian can be a useful indicator for the general behavior of the model in terms such as the sample complexity and generalization, and its relation to learning the underlying neural network.

7. Free-form neural networks as normalizing flows

In the previous chapters, we were concerned with the expressivity and efficiency of existing neural networks invertible by design. However, being able to represent the distribution of interest is not the only requirement for a successful application. As argued in chapter 2, it is often helpful to incorporate prior knowledge in the form of inductive biases and strict constraints into the neural network architecture.

For example, convolutional neural networks or patched-based approaches are essential for successfully training on image data. However, this is not apparent from a representational perspective: A large enough fully-connected network can represent any convolutional network by carefully copying the weights, but this is not the case vice versa (dAscoli et al., 2019). The solution to this paradox is that architecture biases guide the learning process, ensuring that the model captures relevant patterns.

In addition, some useful architectures are incompatible to be used with coupling layers, for example if the learned distribution should be invariant under rotations of the input: $p(Qx) = p(x)$. Even if some constructions can be implemented as invertible neural network blocks, developing specialized architectures is time-consuming and increases complexity.

This strongly suggests that there is a need for streamlining the process of building normalizing flows based on arbitrary architectures. This has been achieved previously by parameterizing an invertible function via an ordinary differential equation (ODE), see section 4.3.2. However, this approach makes sampling from the resulting $p_\theta(x)$ more expensive because it involves solving the ODE in equation (4.29).

In this chapter, we overcome this limitation and allow training arbitrary neural network architectures as normalizing flows for the first time. We call the approach free-form flows (FFF) and provide all details in section 7.1. We then consider two variants of free-form flows in the context of data on manifolds: If the manifold of the data is known, we adapt free-form flows to manifold free-form flows (M-FFF, section 7.2), which learn a distribution on that known manifold while respecting its topology. To the best of our knowledge, M-FFFs are the first non-specialized generative model on manifolds that sample in a single evaluation of the underlying neural network. If that manifold is unknown, free-form injective flows (FIF, section 7.3) jointly learn a subspace and the distribution of the data projected to it. The code to reproduce the experiments is publicly available at `https://github.com/vislearn/FFF`.

7.1. Full-dimensional free-form flows (FFF)

This section is adapted from Draxler et al. (2024a), see the list of individual contributions in chapter 3.

In this section, we contribute an approach that frees full-dimensional normalizing flows from their conventional architectural confines, simplifying the development of normalizing flows. The key methodological innovations are twofold: We derive a novel efficient estimator for computing the *gradient* of the volume change $\log|f'_\theta(x)|$ in the change-of-variables in equation (4.1). This allows training normalizing flows based on architectures whose volume change is not readily available. We then replace invertible neural networks (INNs), which jointly parameterize functions $f_\theta(x)$ and their analytical inverse $f_\theta^{-1}(z)$, by a pair of arbitrary networks $f_\theta(x)$ and $g_\varphi(z)$, where $g_\varphi(z)$ approximates the inverse function $f_\theta^{-1}(z)$. These networks can be freely chosen, depending on the task at hand.

In molecule generation, where rotational invariance of the learned distribution has proven to be

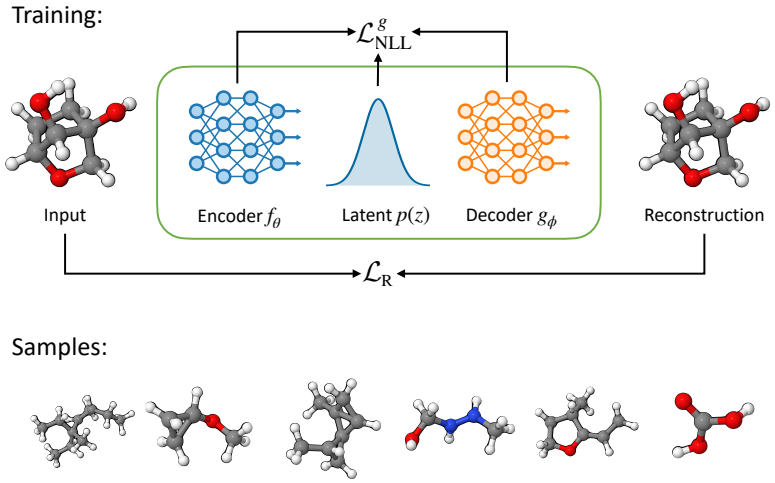


Figure 7.1.: Free-form flows (FFF) train a pair of encoder and decoder neural networks with a fast maximum likelihood estimator $\mathcal{L}^g_{\text{NLL}}$ and reconstruction loss $\mathcal{L}_{\text{R}}$. This enables training any dimension-preserving architecture as a one-step generative model. For example, an equivariant graph neural network can be trained on the QM9 dataset to generate molecules by predicting atom positions and properties in a single decoder evaluation. *(Bottom)* Stable molecules sampled from our $E(3)$-FFF trained on the QM9 dataset for several molecule sizes.

a crucial inductive bias, our approach outperforms traditional normalizing flows and generates valid samples more than an order of magnitude faster than previous approaches. Further, experiments in simulation-based inference (SBI) underscore the model's versatility. We find that our training method achieves competitive performance with minimal fine-tuning requirements.

In summary, our contributions are as follows:

- We remove all architectural constraints from normalizing flows by introducing maximum-likelihood training for free-form architectures. We call our model the free-form flow (FFF), see figure 7.1 and section 7.1.3.
- We demonstrate competitive performance with minimal fine-tuning on inverse problems and molecule generation benchmarks, outperforming ODE-based models in the latter. Compared to a diffusion model, our model produces stable molecules more than two orders of magnitude faster. See section 7.1.4.

7.1.1. Related Work

Normalizing flows traditionally rely on specialized architectures that are invertible and have a manageable Jacobian determinant. We listed the predominant approaches in section 4.3 and here give a short overview over their properties in comparison to free-form flows.

One body of work builds invertible architectures by concatenating simple layers (coupling blocks) each of which are easy to invert analytically and have a triangular Jacobian, which makes computing the volume change easy (Dinh et al., 2015). Many choices for coupling blocks have been proposed, see section 4.3.1. For computing the gradient of the volume change in the change-of-variables, we propose a novel estimator that is also compatible with previous architectures, albeit exhibiting a larger variance. Instead of analytical invertibility, free-form flows rely on a reconstruction loss to enforce approximate invertibility between two networks that do not share weights.

Another line of work ensures invertibility by using a ResNet structure and limiting the Lipschitz constant of each residual layer (Behrmann et al., 2019; Chen et al., 2019). Neural ODEs or continuous

normalizing flows (Chen et al., 2018b; Grathwohl et al., 2019) take the continuous limit of ResNets, guaranteeing invertibility under mild conditions. These models can become expensive due to the restriction of each layer. In addition, the Jacobian determinant must be estimated, adding overhead. Like these methods, we must estimate the gradient of the Jacobian determinant, but we propose a more efficient estimator. Flow Matching (Lipman et al., 2023; Liu et al., 2023a; Albergo & Vanden-Eijnden, 2023) avoids these limitations at training time, but still involves an expensive multistep sampling process at inference time. By construction, our approach consists of a single model evaluation, and we put no constraints on the architecture apart from inductive biases indicated by the task at hand.

Two interesting methods (Gresele et al., 2020; Keller et al., 2021) compute or estimate gradients of the Jacobian determinant but are severely limited to architectures with exclusively square weight matrices and no residual blocks. We have no architectural limitations besides preserving dimension. Intermediate activations and weight matrices may have any dimension and any network topology is permitted.

The gradient estimator closest to our method uses conjugate gradient to estimate the inverse Jacobian Caterini et al. (2021), which we replace with the Jacobian of the inverse. We also generalize to combining the gradient estimator with arbitrary feed-forward neural networks.

7.1.2. Gradient trick

We first derive how to efficiently estimate the *gradient* of the maximum-likelihood loss in equation (4.4), even if the architecture does not yield an efficient way to compute the change of variables term $\log|f'_\theta(x)|$. We avoid this computation by estimating the *gradient* of $\log|f'_\theta(x)|$ via a pair of vector-Jacobian and Jacobian-vector products, which are readily available in standard automatic differentiation software libraries.

Gradient via trace estimator The crucial insight enabling fast gradient computation of the volume change $\log|f'_\theta(x)|$ is that its gradient can be computed via a trace:

Theorem 7.1 (Free-form flow volume change gradient, proof in appendix A.8)**.** *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$ be a diffeomorphism parameterized by θ. Then, for all $x \in \mathbb{R}^D$ and $z = f_\theta(x)$:*

$$\nabla_\theta \log|f'_\theta(x)| = \operatorname{tr}\Big((\nabla_\theta f'_\theta(x))(f_\theta^{-1\prime}(z))\Big) = \mathbb{E}_v\Big[v^{\mathrm{T}}(\nabla_\theta f'_\theta(x))(f_\theta^{-1\prime}(z))v\Big], \tag{7.1}$$

The proof follows directly by applying Jacobi's formula and using that the matrix inverse of the Jacobian is the Jacobian of the inverse function $(f'_\theta)^{-1} = f_\theta^{-1\prime}$. We then replace the trace via the Hutchinson trace estimator, where the random vector $v \in \mathbb{R}^D$ must have unit covariance (Hutchinson, 1989).

Efficient computation via automatic differentiation We approximate the expectation over v by a Monte Carlo sample (omitting the dependence on x and z for simplicity):

$$\mathbb{E}_v\Big[v^{\mathrm{T}}(\nabla_\theta f'_\theta)(f_\theta^{-1\prime})v\Big] \approx \frac{1}{K}\sum_{k=1}^{K} v_k^{\mathrm{T}}(\nabla_\theta f'_\theta) f_\theta^{-1\prime} v_k. \tag{7.2}$$

Now all we require is computing dot products of the form $v_k^{\mathrm{T}}(\nabla_\theta f'_\theta)$ and $f_\theta^{-1\prime}v_k$.

Since v_k^{T} is independent of θ, we can draw it into the gradient $v_k^{\mathrm{T}}(\nabla_\theta f'_\theta) = \nabla_\theta(v_k^{\mathrm{T}} f'_\theta)$. The term to take the gradient of is a vector-Jacobian product. It can be readily computed via backward automatic differentiation, for example via PyTorch's `torch.func.vjp` or `torch.autograd.grad` (Paszke et al.,

2019). Similarly, the Jacobian-vector product $f_\theta'^{-1} v_k$ is readily available via forward automatic differentiation, for example using `torch.func.jvp`.

To implement the final gradient with respect to the flow parameters θ, we draw the derivative with respect to parameters out of the trace, making sure to prevent gradient from flowing to $f_\theta'^{-1}$ by wrapping it in a stop-gradient operation SG, for example via PyTorch's `tensor.detach()`:

$$\frac{1}{K}\sum_{k=1}^{K} v_k^{\mathrm{T}}(\nabla_\theta f_\theta')f_\theta^{-1\prime} v_k = \nabla_\theta \frac{1}{K}\sum_{k=1}^{K} v_k^{\mathrm{T}} f_\theta' \mathtt{SG}(f_\theta^{-1\prime} v_k). \tag{7.3}$$

Summary The above argument shows that

$$\nabla_\theta \log|f_\theta'(x)| \approx \nabla_\theta \frac{1}{K}\sum_{k=1}^{K} v_k^{\mathrm{T}} f_\theta' \mathtt{SG}(f_\theta^{-1\prime} v_k), \tag{7.4}$$

Instead of computing the full Jacobian $f_\theta'(x)$, which involved as many backpropagation steps as dimensions, we are left with computing just one vector-Jacobian product and one Jacobian-vector product for each k. In practice, we find that setting $K = 1$ is sufficient, and we drop the summation over k in the following. We provide an ablation study on the effect of K in appendix B.5.3.

This yields the following maximum likelihood training objective, whose gradients are an unbiased estimator for the true gradients from exact maximum likelihood as in equation (4.4):

$$\nabla_\theta \mathcal{L}_{\mathrm{NLL}}^{f^{-1}} = \nabla_\theta \mathbb{E}_{x,v}[-\log p(f_\theta(x)) - v^{\mathrm{T}} f_\theta' \mathtt{SG}(f_\theta'^{-1} v)]. \tag{7.5}$$

This result enables training normalizing flow architectures with a tractable inverse function, but whose Jacobian determinant is not easily accessible. We now move on to show how this gradient estimator can be used to train any neural network architecture as a normalizing flow.

7.1.3. Free-Form Flows (FFF)

The previous section assumed that we have access to both f_θ and its analytic inverse f_θ^{-1}. We now drop the assumption that f_θ is invertible and replace its inverse f_θ^{-1} by a separate neural network $g_\varphi \approx f_\theta^{-1}$. Instead, we regularize these properties via a reconstruction loss:

$$\mathcal{L}_{\mathrm{R}} = \tfrac{1}{2}\mathbb{E}_x[\|x - g_\varphi(f_\theta(x))\|^2]. \tag{7.6}$$

This removes all architectural constraints from f_θ and g_φ except from preserving the dimension.

The replacement $g_\varphi \approx f_\theta^{-1}$ leads to a modification of $\mathcal{L}_{\mathrm{NLL}}^{f^{-1}}$, where we replace $f_\theta'^{-1}$ by g_φ', where g_φ' is shorthand for the Jacobian of g_φ evaluated at $f_\theta(x)$:

$$\nabla_\theta \mathcal{L}_{\mathrm{NLL}}^{g} = \nabla_\theta \mathbb{E}_{x,v}[-\log p(f_\theta(x)) - v^{\mathrm{T}} f_\theta' \mathtt{SG}(g_\varphi' v)] \tag{7.7}$$

Combining this with the reconstruction loss leads to the following optimization:

$$\nabla_{\theta,\varphi} \mathcal{L}^{g} = \nabla_{\theta,\varphi}(\mathcal{L}_{\mathrm{NLL}}^{g} + \beta \mathcal{L}_{\mathrm{R}}) \tag{7.8}$$

where the two terms are traded off by a hyperparameter β. See algorithm 2 for a reference implementation.

Algorithm 2 FFF loss function. Vector-Jacobian product = `vjp`; Jacobian-vector product = `jvp`. Time and space complexity are $\mathcal{O}(D)$.

Function LOSS($x, f_\theta, g_\varphi, \beta$)
$v \sim p(v)$ $\{\mathbb{E}[vv^{\mathrm{T}}] = I\}$
$z, v_f \leftarrow \texttt{vjp}(f_\theta, x, v)$ $\{z = f_\theta(x), v_f^{\mathrm{T}} = v^{\mathrm{T}} f'_\theta(x)\}$
$\hat{x}, v_g \leftarrow \texttt{jvp}(g_\varphi, z, v)$ $\{\hat{x} = g_\varphi(z), v_g = g'_\varphi(z)v\}$
$\mathcal{L}^g_{\mathrm{NLL}} \leftarrow \frac{1}{2}\|z\|^2 - v_f^{\mathrm{T}}\texttt{SG}(v_g)$ {stop gradient to v_g}
$\mathcal{L}_{\mathrm{R}} \leftarrow \|\hat{x} - x\|^2$
$\mathcal{L}^g \leftarrow \mathcal{L}^g_{\mathrm{NLL}} + \beta\mathcal{L}_{\mathrm{R}}$
return $\mathcal{L}^g$

Likelihood Calculation

Once training is completed, our generative model involves sampling from the latent distribution and passing the samples through the decoder g_φ.

To calculate the likelihoods induced by g_φ, we can use the change of variables formula:

$$p_\varphi(X = x) = p(Z = g_\varphi^{-1}(x))|g'_\varphi(g_\varphi^{-1}(x))| \approx p(Z = f_\theta(x))|g'_\varphi(f_\theta(x))| \tag{7.9}$$

where the approximation is due to $g_\varphi^{-1} \approx f_\theta$. See algorithm 3 for a reference implementation. In the paper underlying this section, Peter Sorrenson derived additional justification for using free-form architectures and the combination of maximum likelihood with a reconstruction loss (Draxler et al., 2024a).

Algorithm 3 FFF likelihood calculation: returns an approximation of $\log p_\varphi(x)$. Time complexity is $\mathcal{O}(D^3)$ and space complexity is $\mathcal{O}(D^2)$.

Function LOGLIKELIHOOD(x, f_θ, g_φ)
$z \leftarrow f_\theta(x)$
$g'_\varphi \leftarrow \texttt{jacobian}(g_\varphi, z)$ {where $g'_\varphi = \frac{\partial}{\partial z} g_\varphi(z)$}
$\ell \leftarrow -\frac{1}{2}\|z\|^2 - \frac{D}{2}\log(2\pi) + \log|g'_\varphi|$ {see change of variables in equation (4.1)}
return ℓ

7.1.4. Experiments

In this section, we demonstrate the practical capabilities of free-form flows (FFF). We compare the performance against normalizing flows based on architectures which are invertible by construction. First, on an inverse problem benchmark, we show that using free-form architectures offers competitive performance to recent spline-based and ODE-based normalizing flows. This is achieved despite minimal tuning of hyperparameters, demonstrating that FFFs are easy to adapt to a new task. Second, on two molecule generation benchmarks, we demonstrate that specialized networks can now be used in a normalizing flow. In particular, we employ the equivariant graph neural networks $E(n)$-GNN (Satorras et al., 2021b). This $E(n)$-FFF outperforms ODE-based equivariant normalizing flows in terms of likelihood, and generates stable molecules significantly faster than a diffusion model.

Simulation-Based Inference

One popular application of generative models is in solving inverse problems. Here, the goal is to estimate hidden parameters from an observation. As inverse problems are typically ambiguous, a

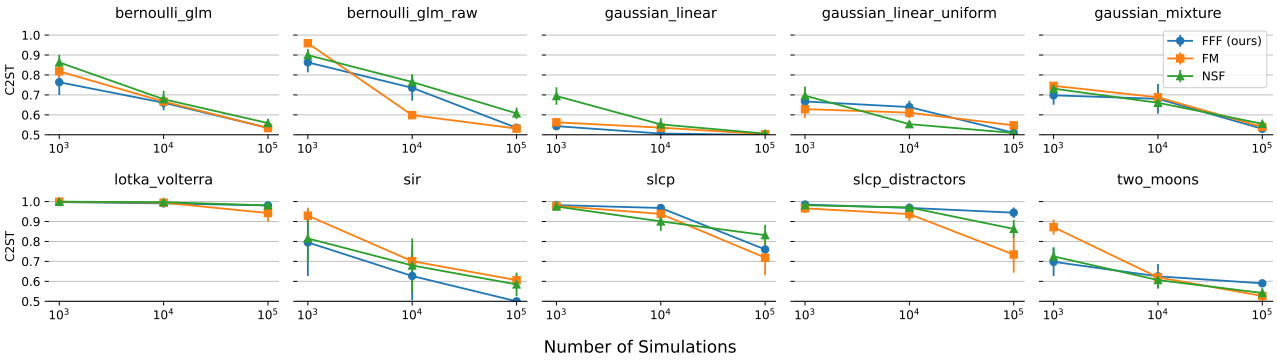


Figure 7.2.: C2ST accuracy on the SBI benchmark datasets *(lower is better)*. We compare our method (FFF) against flow matching (FM) (Wildberger et al., 2023) and the neural spline flow (NSF) baseline in the benchmark dataset (Lueckmann et al., 2021). The accuracy is averaged over ten different observations, with error bars indicating the standard deviation. Our performance is comparable to the competitors across all datasets, with no model being universally better or worse.

probability distribution represented by a generative model is a suitable solution. From a Bayesian perspective, this probability distribution is the posterior of the parameters given the observation. We learn this posterior via a conditional generative model.

In particular, we focus on simulation-based inference (SBI, (Radev et al., 2022, 2021; Bieringer et al., 2021)), where we want to predict the parameters of a simulation. The training data consists of pairs of parameters (inputs) and measurements (outputs) generated from the simulation.

We train FFF models on the benchmark proposed in (Lueckmann et al., 2021), which comprises ten inverse problems of varying difficulty at three different simulation budgets (i.e. sizes of the training set) each. The models are evaluated via a classifier 2-sample test (C2ST) (Lopez-Paz & Oquab, 2017; Friedman, 2003), where a classifier is trained to discern samples from the trained generative model and the true parameter posterior. The model performance is then reported as the classifier accuracy, where 0.5 demonstrates a distribution indistinguishable from the true posterior. We average this accuracy over ten different observations. In figure 7.2, we report the C2ST of our model and compare it against the baselines based on neural spline flows (Durkan et al., 2019) and flow matching for SBI (Wildberger et al., 2023). Our method performs competitively, especially providing an improvement over existing methods in the regime of low simulation budgets. Regarding tuning of hyperparameters, we find that a simple fully-connected architecture with skip connections works across datasets with minor modifications to increase capacity for the larger datasets. We identify the reconstruction weight β large enough such that training becomes stable. We give all dataset and more training details in appendix B.5.1.

Molecule Generation

Free-form flows (FFF) do not make any assumptions about the underlying networks f_θ and g_φ, except that they preserve dimension. We can leverage this flexibility for tasks where explicit constraints *should* be built into the architecture, as opposed to constraints that originate from the need for tractable invertibility (such as coupling blocks).

As a showcase, we apply FFF to molecule generation. Here, the task is to learn the joint distribution of a number of atoms $x_1, \dots, x_N \in \mathbb{R}^n$. Each prediction of the generative model should yield a physically valid position for each atom: $x = (x_1, \dots, x_N) \in \mathbb{R}^{N \times n}$.

The physical system of atoms in space have an important symmetry: if a molecule is shifted or rotated in space, its properties do not change. This means that a generative model for molecules should

yield the same probability regardless of orientation and translation:

$$p_\varphi(Qx + t) = p_\varphi(x). \tag{7.10}$$

Here, the rotation $Q \in \mathbb{R}^{n\times n}$ acts on x by rotating or reflecting each atom $x_i \in \mathbb{R}^n$ about the origin, and $t \in \mathbb{R}^n$ applies the same translation to each atom. Formally, (Q, t) are realizations of the Euclidean group $E(n)$. The above equation (7.10) means that the distribution $p_\varphi(x)$ is invariant under the Euclidean group $E(n)$.

If the latent distribution $p(z)$ is *invariant* under a group G, and a generative model $g_\varphi(z)$ is *equivariant* to G, then the resulting distribution is also invariant to G (Köhler et al., 2020). Equivariance means that applying any group action to the input (e.g. rotation and translation) and then applying g_φ should give the same result as first applying g_φ and then applying the group. For example, for the Euclidean group:

$$Qg_\varphi(z) + t = g_\varphi(Qz + t). \tag{7.11}$$

This implies that we can learn a distribution invariant to the Euclidean group by making the normalizing flows equivariant to the Euclidean group, as in equation (7.11). Previous work has demonstrated that this inductive bias is more effective than data augmentation, where random rotations and translations are applied to each data point at train time (Köhler et al., 2020; Hoogeboom et al., 2022).

We therefore choose an $E(n)$ equivariant network as the networks $f_\theta(x)$ and $g_\varphi(z)$ in our FFF. We employ the $E(n)$-GNN proposed by Satorras et al. (2021b). We call this model the $E(n)$-free-form flow ($E(n)$-FFF). We give the implementation details in appendix B.5.2.

The $E(n)$-GNN has also been the backbone for previous normalizing flows on molecules. However, to the best of our knowledge, all realizations of such architectures have been based on neural ODEs, where the flow is parameterized as a differential equation $\frac{\mathrm{d}x}{\mathrm{d}t} = f_\theta(x(t), t)$. While training, one can avoid solving the ODE by using the flow matching objective (Liu et al., 2023a; Lipman et al., 2023; Albergo & Vanden-Eijnden, 2023). However, they still have the disadvantage that they require integrating the ODE for sampling. Our model, in contrast, only calls $g_\varphi(z)$ once for sampling.

Boltzmann Generator We test our $E(n)$-FFFs in learning a Boltzmann distribution:

$$p(x) \propto e^{-\beta u(x)}, \tag{7.12}$$

where $u(x) \in \mathbb{R}$ is an energy function that takes the positions of atoms $x = (x_1, \ldots, x_N)$ as an input. A generative model $p_\varphi(x)$ that approximates $p(x)$ can be used as a Boltzmann generator (Noé et al., 2019). The idea of the Boltzmann generator is that having access to $u(x)$ allows re-weighting samples from the generator after training even if $p_\varphi(x)$ is different from $p(x)$. This allows computing expectation values $\mathbb{E}_{x\sim p(x)}[O(x)] = \mathbb{E}_{x\sim p_\varphi(x)}[\frac{p(x)}{p_\varphi(x)}O(x)]$ from samples of the generative model $p_\varphi(x)$ if $p_\varphi(x)$ and $p(x)$ have the same support.

We evaluate the performance of free-form flows (FFF) as a Boltzmann generator on the benchmark tasks DW4, LJ13, and LJ55 (Köhler et al., 2020; Klein et al., 2023). Here, pairwise potentials $v(x_i, x_j)$ are summed as the total energy $u(x)$:

$$u(x) = \sum_{i,j} v(x_i, x_j). \tag{7.13}$$

DW4 uses a double-well potential $v_{\mathtt{DW}}$ and considers four particles in 2D. LJ13 and LJ55 both employ a Lennard-Jones potential $v_{\mathtt{LJ}}$ between 13 respectively 55 particles in 3D space (see appendix B.5.2 for

Table 7.1.: Equivariant free-form flows ($E(n)$-FFF) sample significantly faster than previous models, and achieve comparable or better negative log-likelihood (NLL, lower is better). More details in table B.5.

	NLL (↓)	Sampling time (↓)
	DW4	
$E(2)$-NF (Satorras et al., 2021a)	1.72 ± 0.01	**0.024** ms
OT-FM (Klein et al., 2023)	**1.70** ± 0.02	0.034 ms
E-OT-FM (Klein et al., 2023)	**1.68** ± 0.01	0.033 ms
$E(2)$-FFF (ours)	**1.68** ± 0.01	**0.026** ms
	LJ13	
$E(3)$-NF	-16.28 ± 0.04	0.27 ms
OT-FM	-16.54 ± 0.03	0.77 ms
E-OT-FM	-16.70 ± 0.12	0.72 ms
$E(3)$-FFF (ours)	**-17.09** ± 0.16	**0.11** ms
	LJ55	
OT-FM	-88.45 ± 0.04	40 ms
E-OT-FM	**-89.27** ± 0.04	40 ms
$E(3)$-FFF (ours)	-88.72 ± 0.16	**2.1** ms

details). We make use of the datasets presented by Klein et al. (2023), which obtained samples from $p(x)$ via MCMC.[1]

In table 7.1, we compare our model against (i) the equivariant ODE normalizing flow trained with maximum likelihood $E(n)$-NF (Satorras et al., 2021a), and (ii) two equivariant ODEs trained via optimal transport (equivariant) flow matching (Klein et al., 2023). We find our model to have comparable or better negative log-likelihood than competitors. In addition, $E(n)$-FFFs sample significantly faster than competitors because our model needs to evaluate the learned network only once, as opposed to the multiple evaluations required to integrate an ODE.

QM9 Molecules As a second molecule generation benchmark, we test the performance of $E(3)$-FFF in generating novel molecules. We therefore train on the QM9 dataset (Ruddigkeit et al., 2012; Ramakrishnan et al., 2014), which contains molecules of varying atom count, with the largest molecules counting 29 atoms. The goal of the generative model is not only to predict the positions of the atoms in each molecule $x = (x_1, \dots, x_N) \in \mathbb{R}^3$, but also each atom's properties h_i (atom type (categorical), and atom charge (ordinal)). We dequantize atom properties via an argmax flow (Hoogeboom et al., 2021) and learn a distribution $p(x, h)$.

We again employ the $E(3)$-GNN (Satorras et al., 2021b). The part of the network that acts on coordinates $x_i \in \mathbb{R}^3$ is equivariant to rotations, reflections and translations (Euclidean group $E(3)$). The network leaves the atom properties h invariant under these operations.

We show samples from our model in figure 7.1. Because free-form flows only need one network evaluation to sample, they generate two orders of magnitude more stable molecules than the $E(3)$-diffusion model (Hoogeboom et al., 2022) and one order of magnitude more than the $E(3)$-normalizing flow (Satorras et al., 2021a) in a fixed time window, see table 7.2. This includes the time to generate unstable samples, which are discarded. A molecule is called stable if each atom has the correct number of bonds, where bonds are determined from inter-atomic distances. $E(3)$-FFF also outperforms $E(3)$-NF trained with maximum likelihood both in terms of likelihood and in how many of the sampled molecules are stable. See appendix B.5.2 for implementation details.

[1]Datasets available at: `https://osf.io/srqg7/?view_only=28deeba0845546fb96d1b2f355db0da5`

Table 7.2.: $E(3)$-FFF trained on QM9 generates a stable molecule faster than previous models because a sample is obtained via a single function evaluation. $E(3)$-DM is the $E(3)$-diffusion model, $E(3)$-NF the $E(3)$-normalizing flow. The latter is also trained explicitly using maximum likelihood, yet outperformed by $E(3)$-FFF in terms of negative log-likelihood (NLL) and what ratio of generated molecules is stable.

	NLL (↓)	Stable (↑)	Sampling time (↓) Raw	Stable
$E(3)$-NF (Satorras et al., 2021a)	-59.7	4.9 %	13.9 ms	309.5 ms
$E(3)$-DM (Hoogeboom et al., 2022)	**-110.7**	**82.0** %	1580.8 ms	1970.6 ms
$E(3)$-FFF (ours)	-76.2	8.7 %	**0.6** ms	**8.1** ms
Data	-	95.2 %	-	-

7.1.5. Conclusion

In this section, we presented free-form flows (FFF), a new paradigm for normalizing flows that signficantly broadens what architectures can be trained efficiently as normalizing flows: First, $\mathcal{L}_{\text{NLL}}^{f^{-1}}$ enables training architectures that are invertible yet lack a tractable Jacobian determinant. Second, learning a separate generator network allows training arbitrary architectures as normalizing flows via $\mathcal{L}_{\text{NLL}}^{g}$.

On the practical side, we first confirm that free-form flows with a vanilla architecture is able to solve SBI tasks effectively, with no clear winner compared to expressive normalizing flows based on rational-quadratic splines and flow matching. FFF and flow matching both do not require special architectures to solve the task. Compared to flow matching, FFF has the advantage that it yields samples in a single step whereas flow matching requires calling the underlying neural network several times in order to solve the learned ordinary differential equation.

For the first time, we learn a normalizing flow equivariant to the Euclidean group sampling in a single function evaluation. The resulting model outperforms previous models trained with maximum likelihood and is competitive with a model based on diffusion, again at greatly improved inference speed.

7.2. Manifold free-form flows on a known Riemannian manifold (M-FFF)

This section is adapted from Sorrenson et al. (2023), see the list of individual contributions in chapter 3.

Generative models such as normalizing flows have achieved remarkable success where the data $x \in \mathbb{R}^D$ is described in Euclidean geometry. However, the approaches are not directly applicable when dealing with data inherently structured in non-Euclidean spaces, which is common in fields such as the natural sciences, computer vision, and robotics. Examples include earth science data on a sphere, the orientation of real-world objects given as a rotation matrix in $SO(3)$, or data on special geometries modeled by meshes or signed distance functions. Representing such data naively using internal coordinates, such as angles, can lead to topological issues, causing discontinuities or artifacts.

While many generative models can be adapted to handle data on arbitrary manifolds, the predominant methods compatible with arbitrary Riemannian manifolds involve solving differential equations—stochastic (SDEs) or ordinary (ODEs)—for sampling and density estimation (Rozen et al., 2021; Mathieu & Nickel, 2020; Huang et al., 2022; De Bortoli et al., 2022; Chen & Lipman, 2024). These methods are computationally intensive due to the need for numerous function evaluations during integration, slowing down inference.

In this section, we address these challenges by generalizing the free-form flow framework from section 7.1 to data on arbitrary Riemannian manifolds. This novel approach for modeling distributions

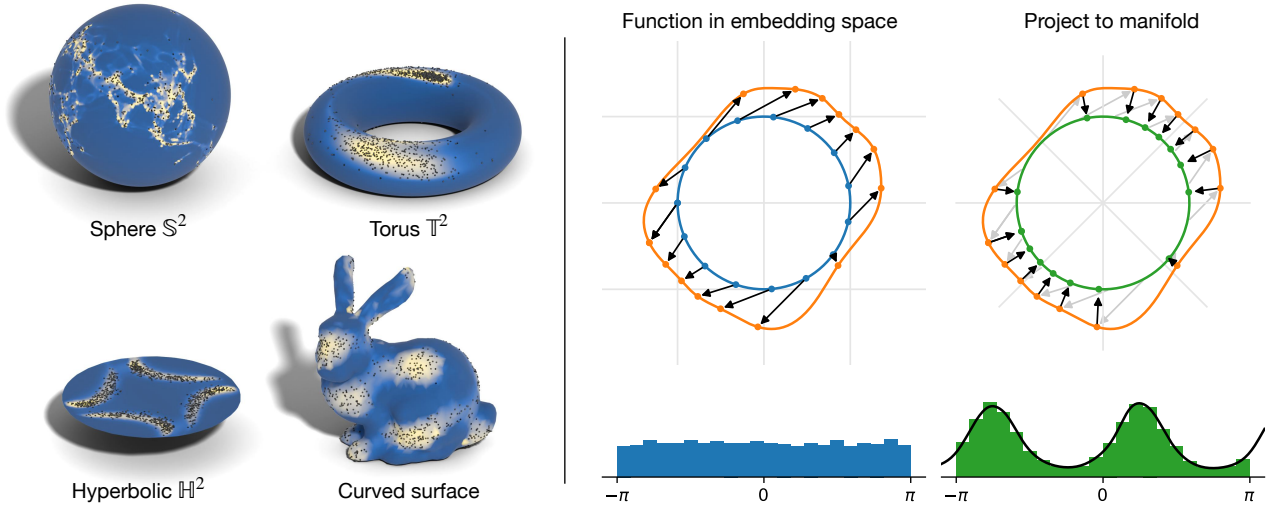


Figure 7.3.: Manifold Free-Form Flows (M-FFF) learn generative models on various manifolds. *(Left)* The learned distributions *(colored surface)* accurately match the test points *(black dots)*. *(Right)* We parameterize M-FFF using a neural network in an embedding space, whose outputs are projected to the manifold. This enables simulation-free training and inference, and naturally respects the corresponding geometry, yielding fast sampling and continuous distributions regardless of the manifold.

on arbitrary Riemannian manifolds circumvents the computational burden of previous methods and is remarkably simple to adapt to novel manifolds. Again, we use a single feed-forward neural network on an embedding space as a generator, with outputs projected to the manifold (figure 7.3). The generalization of the volume change gradient estimator to manifolds is achieved by estimating the gradient of the negative log-likelihood within the tangent space of the manifold.

In summary, we make the following contributions:

- We extend free-form flows to Riemannian manifolds, yielding manifold free-form flows (M-FFF) in section 7.2.3. It can easily be adapted to arbitrary Riemannian manifolds, requiring only a projection function from an embedding space.
- M-FFF only relies on a single function evaluation during training and sampling, speeding up inference over multistep methods typically by two orders of magnitude.
- M-FFF consistently matches or outperforms previous single-step methods on several benchmarks on spheres, tori, rotation matrices, hyperbolic space and curved surfaces (see figure 7.3 and section 7.2.4). In addition, it is competitive with multistep methods.

Together, manifold free-form flows offer a novel and efficient approach for learning distributions on manifolds, applicable to any Riemannian manifold with a known embedding and projection.

7.2.1. Related work

Existing work on learning distributions on manifolds can be broadly categorized as follows: (i) leveraging Euclidean generative models; (ii) building specialized architectures that respect one particular kind of geometry; and (iii) learning a continuous time process on the manifold. We compare our method to these approaches in table 7.3 and give additional detail below.

Euclidean generative models. One approach maps the n-dimensional manifold to $\mathbb{R}^n$ and learns the resulting distribution (Gemici et al., 2016). Another approach generalizes the reparameterization trick to Lie groups by sampling on the Lie algebra, which can be parameterized in Euclidean space

Table 7.3.: Feature comparison of generative models on manifolds. We give a "✓" if any method in a category meets this requirement.

	Respects topology	Single step sampling	Arbitrary manifolds
Euclidean generative models	✗	✓	✓
Specialized architectures	✓	✓	✗
Continuous time models	✓	✗	✓
M-FFF (ours)	✓	✓	✓

(Falorsi et al., 2019). These approaches come with the downside that a Euclidean representation may not respect the geometry of the manifold sufficiently, e.g. mapping the earth to a plane causes discontinuities at the poles. This can be overcome by learning distributions on overlapping charts that together span the full manifold (Kalatzis et al., 2021). An orthogonal solution is to embed the data and add noise to it in the off-manifold directions so that the distribution can be learned directly in an embedding space $\mathbb{R}^m$ (Brofos et al., 2021); this only provides access to a variational approximation instead of the exact density. Our method also works in the embedding space so that it respects the geometry of the manifold, but directly optimizes the likelihood on the manifold.

Specialized architectures take advantage of the specific geometry of a certain kind of manifold to come up with special coupling blocks for building normalizing flows such as $SO(3)$ (Liu et al., 2023b), $SU(d), U(d)$ (Boyda et al., 2021; Kanwar et al., 2020); hyperbolic space (Bose et al., 2020); tori and spheres (Rezende et al., 2020). Manifold free-form flows are not restricted to one particular manifold, but can be easily applied to any manifold for which an embedding and a projection to the manifold is known, see table 7.4. As such, our model is an alternative to all of the above specialized architectures.

Continuous time models build a generative model based on parameterizing an ODE or SDE on any Riemannian manifold, meaning that they specify the (stochastic) differential equation in the tangent space (Rozen et al., 2021; Falorsi, 2021; Falorsi & Forré, 2020; Huang et al., 2022; Mathieu & Nickel, 2020; De Bortoli et al., 2022; Chen & Lipman, 2024; Lou et al., 2020; Ben-Hamu et al., 2022). These methods come with the disadvantage that sampling and density evaluation integrates the ODE or SDE, requiring many function evaluations. Our manifold free-form flows do not require repeatedly evaluating the model, a single function call followed by a projection is sufficient.

At its core, our method generalizes the free-form flow (FFF) framework from section 7.1 to manifolds.

7.2.2. Riemannian manifolds

A manifold is a fundamental concept in mathematics, providing a framework for describing and analyzing spaces that locally resemble Euclidean space, but may have different global structure. For example, a small region on a sphere is similar to the Euclidean plane, but walking in a straight line on the sphere in any direction will return to the starting point, unlike on a plane.

Mathematically, an n-dimensional manifold, denoted as $\mathcal{M}$, is a space where every point has a neighborhood that is topologically equivalent to $\mathbb{R}^n$. A Riemannian manifold $(\mathcal{M}, G)$ extends the concept of a manifold by adding a Riemannian metric G, which introduces a notion of distances and angles. At each point x on the manifold, there is an associated tangent space $\mathcal{T}_x\mathcal{M}$ which is an n-dimensional Euclidean space, characterizing the directions in which you can travel and still stay on the manifold. The metric G acts in this space, defining an inner product between vectors. From this inner product, we can compute the length of paths along the manifold, distances between points as well as volumes (see next section).

Here, we consider Riemannian manifolds globally embedded into an m-dimensional Euclidean space $\mathbb{R}^m$, with $n \leq m$. Embedding means that we represent a point on the manifold $x \in \mathcal{M}$ as a vector

Table 7.4.: Manifolds, a global embedding and corresponding projections considered in this work.

Manifold	Dimension n	Embedding	Projection
Generic	$\text{rank}(\text{proj}'(\text{proj}(x)))$	$\{x \in \mathbb{R}^m : \text{proj}(x) = x\}$	$x \mapsto \text{proj}(x)$
Rotations $SO(d)$	$(d-1)d/2$	$\{Q \in \mathbb{R}^{d\times d} : QQ^\text{T} = I, \det Q = 1\}$	$R \mapsto \arg\min_{Q\in SO(d)} \|Q - R\|_F$; see equation (B.41)
Sphere $\mathbb{S}^n$	n	$\{x \in \mathbb{R}^{n+1} : \|x\| = 1\}$	$x \mapsto x/\|x\|$
Torus $\mathbb{T}^n = (\mathbb{S}^1)^n$	n	$\{X \in \mathbb{R}^{n\times 2} : \|X_i\| = 1 \text{ for } i = 1...n\}$	$X_i \mapsto X_i/\|X_i\| \text{ for } i = 1...n$
Hyperbolic $\mathbb{H}^n$	n	$\{x \in \mathbb{R}^n : \|x\| < 1\}$	$x \mapsto x \min\{1, (1-\epsilon)/\|x\|\}$

in $\mathbb{R}^m$ confined to an n-dimensional subspace; we write $x \in \mathcal{M} \subseteq \mathbb{R}^m$ and denote by $\pi : \mathbb{R}^m \to \mathcal{M}$ a projection from the embedding space to the manifold. A global embedding is a smooth, injective mapping of the entire manifold into $\mathbb{R}^m$, its smoothness preserving the topology.

In most cases, we work with, but are not limited to, isometrically embedded manifolds, meaning that the metric is inherited from the ambient Euclidean space. Intuitively, this means that the length of a path on the manifold is just the length of the path in the embedding space. We note that due to the Nash embedding theorem (Nash, 1956), every Riemannian manifold has a smooth isometric embedding into Euclidean space of some finite dimension, so in this sense using isometric embeddings is not a limitation. Nevertheless, for some manifolds (especially with negative curvature, e.g. hyperbolic space) there may not be a sensible isometric embedding.

7.2.3. Generalization of free-form flows to manifolds

The free-form flow (FFF) framework allows training any pair of parameterized encoder $f_\theta(x)$ and decoder $g_\varphi(z)$ as a generative model. In this section, we demonstrate how to generalize the steps in section 7.1 to arbitrary Riemannian manifolds. Note that for simplicity, we choose the same manifold in data and latent spaces, i.e. $\mathcal{M}_X = \mathcal{M}_Z = \mathcal{M}$, but the method readily applies to $\mathcal{M}_X \neq \mathcal{M}_Z$ or $G_X \neq G_Z$ as long as they are topologically compatible, like a sphere and a closed surface in 3D without holes. The detailed derivations in the appendix consider this generalization.

Manifold change of variables The volume change on manifolds generalizes the Euclidean variant in equation (4.1) by (a) considering the change of volume in the tangent space and (b) accounting for volume change due to changes in the metric:

Theorem 7.2 (Manifold change of variables, generalized proof in appendix A.9.1)**.** *Let $(\mathcal{M}, G)$ be a n-dimensional Riemannian manifold embedded in $\mathbb{R}^m$, i.e., $\mathcal{M} \subseteq \mathbb{R}^m$. Let $p(x)$ be a probability distribution on $\mathcal{M}$ and let $f_\theta : \mathcal{M} \to \mathcal{M}$ be a diffeomorphism. Let $p_\theta(z)$ be the pushforward of $p(x)$ under f_θ.*

Let $x \in \mathcal{M}$. Define $Q \in \mathbb{R}^{m\times n}$ as an orthonormal basis for $\mathcal{T}_x\mathcal{M}$ and $R \in \mathbb{R}^{m\times n}$ as an orthonormal basis for $\mathcal{T}_{f_\theta(x)}\mathcal{M}$.

Then, the probability densities $p_\theta(x)$ and $p(z)$ are related under the following change of variables:

$$p_\theta(x) = p(z = f_\theta(x))|R^\text{T} f'_\theta(x) Q| \sqrt{\frac{|R^\text{T} G(f_\theta(x)) R|}{|Q^\text{T} G(x) Q|}}. \tag{7.14}$$

where Q and R depend on x and $f_\theta(x)$, respectively, although this dependency is omitted for brevity.

To give an intuition for this result, figure 7.4 shows how the volume change is computed for an isometrically embedded manifold, that is $G = I$ so that $|R^\text{T} G R| = |Q^\text{T} G Q| = 1$. This simplifies equation (7.14) to:

$$p_\theta(x) = p(z = f_\theta(x))|R^\text{T} f'_\theta(x) Q|. \tag{7.15}$$

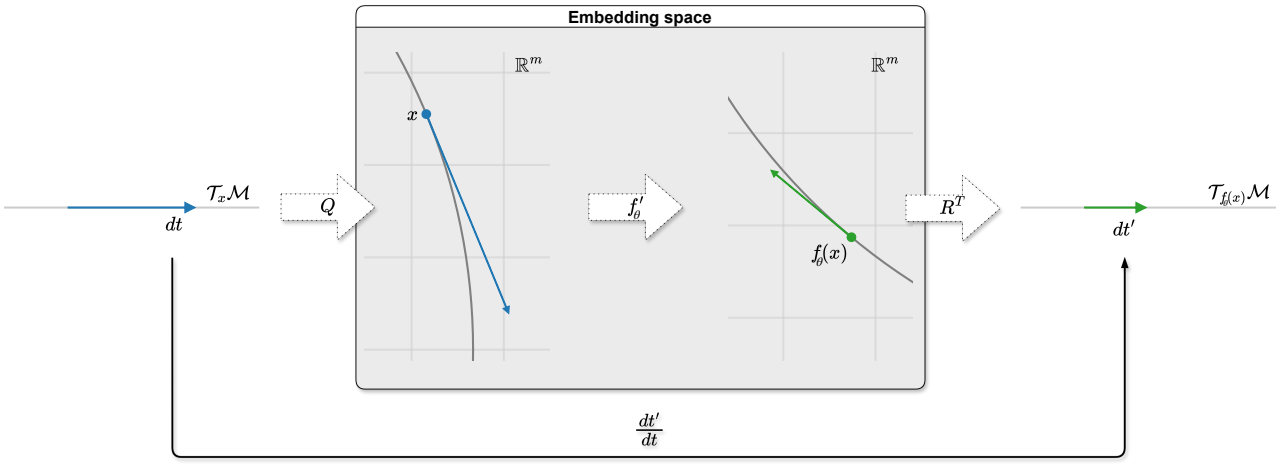


Figure 7.4.: Computation of the volume change in the tangent space of the manifold: The manifold change of variables formula in equation (7.15) requires to compute the change of a volume element in the tangent spaces under f_θ, which in this example is given by the ratio of lengths of dt and dt'. Since f is a map in the embedding space, $f'_\theta(x)$ defines a linear map between vectors from the embedding space. To correctly compute the change in volume, we use Q and R to change coordinates to the intrinsic tangent spaces, resulting in the linear map $R^{\mathrm{T}} f'_\theta(x) Q : \mathcal{T}_x\mathcal{M} \to \mathcal{T}_{f_\theta(x)}\mathcal{M}$, which maps dt to dt'.

This is very similar to the familiar change of variables formula in the Euclidean case in equation (4.1), the only difference being that the determinant is evaluated on the $n \times n$ projection of $f'_\theta(x)$ into the tangent spaces. These projections are necessary as the Jacobian of f is singular in the embedding space, since its action is restricted to the local tangent spaces.

Manifold gradient estimator We now generalize the volume change gradient estimator in section 7.1.2 to an invertible function on a manifold $f_\theta : \mathcal{M} \to \mathcal{M}$. We find that taking the gradient of the manifold change of variables in equation (7.14) results in essentially the same computation as in the Euclidean case, but the trace in is now evaluated in the local tangent space:

Theorem 7.3 (Manifold volume change gradient, generalized proof in appendix A.9.2)**.** *Under the assumptions of theorem 7.2. Let $v \in \mathbb{R}^m$ be a random variable with zero mean and covariance RR^{T}. Then, the derivative of the change of variables term has the following trace expression, where $z = f_\theta(x)$:*

$$\nabla_\theta \log |R^{\mathrm{T}} f'_\theta(x) Q| = \mathrm{tr}\left(R^{\mathrm{T}} (\nabla_\theta f'_\theta(x)) f_\theta^{-1\prime}(z) R \right) = \mathbb{E}_v \left[v^{\mathrm{T}} (\nabla_\theta f'_\theta(x)) f_\theta^{-1\prime}(z) v \right]. \tag{7.16}$$

This shows that the adaptation of free-form flows for an invertible function f_θ to isometrically embedded manifolds is remarkably simple (if the manifold is not isometrically embedded, add the corresponding term in equation (7.14)):

$$\nabla_\theta \mathcal{L}^{f^{-1}}_{\text{M-NLL}} = \nabla_\theta \mathbb{E}_{x,v} \left[-\log p(z) - v^{\mathrm{T}} f'_\theta(x) \mathtt{SG}[f_\theta^{-1\prime}(z) v] \right]. \tag{7.17}$$

The only change from equation (7.5) is that v must have covariance RR^{T} rather than the identity. We achieve this by sampling standard normal vectors $\tilde{v} \in \mathbb{R}^m$ in the embedding space and then projecting them into the tangent space using the Jacobian of the projection function:

$$v = \mathrm{proj}'(f_\theta(x)) \tilde{v}. \tag{7.18}$$

Constructing v like this fulfills the conditions of theorem 7.3 because $\mathbb{E}_v[v] = 0$, and:

$$\mathrm{Cov}[v] = \mathbb{E}_{\tilde{v}}\Big[\mathrm{proj}'(f_\theta(x))\tilde{v}\tilde{v}^{\mathrm{T}}\,\mathrm{proj}'(f_\theta(x))^{\mathrm{T}}\Big] = \mathrm{proj}'(f_\theta(x))\,\mathrm{proj}'(f_\theta(x))^{\mathrm{T}} = RR^{\mathrm{T}}. \tag{7.19}$$

Equation (7.17) allows training invertible architectures on manifolds even if the volume change $\log|R^{\mathrm{T}} f'_\theta(x)Q|$ is not tractable.

Free-form manifold-to-manifold neural networks As discussed in the related work in section 7.3.1, invertible architectures have to be specially constructed for each manifold. To overcome this limitation, we now soften the constraint that the learned model be analytically invertible. Instead, we learn a pair of free-form manifold-to-manifold neural networks, an encoder $f_\theta(x)$ and a decoder $g_\varphi(z)$ as arbitrary functions on the manifold:

$$f_\theta(x) : \mathcal{M} \to \mathcal{M}, \quad g_\varphi(z) : \mathcal{M} \to \mathcal{M}. \tag{7.20}$$

We choose to fulfill equation (7.20) using feed-forward neural networks $\tilde{f}_\theta, \tilde{g}_\varphi : \mathbb{R}^m \to \mathbb{R}^m$ working in the embedding space $\mathbb{R}^m$ of $\mathcal{M}$, but ensure that their outputs lie on the manifold by appending a projection $\mathrm{proj} : \mathbb{R}^m \to \mathcal{M}$, mapping points from the embedding space $\mathbb{R}^m$ to the manifold $\mathcal{M}$:

$$f_\theta(x) = \mathrm{proj}(\tilde{f}_\theta(x)), \quad g_\varphi(z) = \mathrm{proj}(\tilde{g}_\varphi(z)). \tag{7.21}$$

Figure 7.3 illustrates this for an example on a circle $\mathcal{M} = \mathbb{S}^1$.

Just like in the Euclidean case, we employ a reconstruction loss to make f_θ and g_φ approximately inverse to one another:

$$\mathcal{L}_{\mathrm{R}} = \mathbb{E}_{p_{\mathrm{data}}}[\|g_\varphi(f_\theta(x)) - x\|^2]. \tag{7.22}$$

We measure the distance in the embedding space; one can modify this to use an on-manifold distance (e.g. great circle distance for the sphere) but we find that ambient Euclidean distance works well in practice, since it is almost identical for small distances in an isometric embedding.

We now substitute ${f_\theta^{-1}}'(z) \approx g'_\varphi(z)$ in equation (7.17):

$$\nabla_{\theta,\varphi}\mathcal{L}^g_{\text{M-NLL}} \approx \nabla_{\theta,\varphi}\mathbb{E}_{x,v}\Big[-\log p(z) - v^{\mathrm{T}} f'_\theta(x)\mathtt{SG}[{g_\varphi}'(z)v]\Big]. \tag{7.23}$$

Regularization and final loss We find that the following two regularizations to the loss improve the stability and performance of our models. Firstly, the reconstruction loss on points sampled uniformly from the data manifold:

$$\mathcal{L}_{\mathrm{U}} = \mathbb{E}_{x\sim\mathcal{U}(\mathcal{M})}[\|g_\varphi(f_\theta(x)) - x\|^2], \tag{7.24}$$

helps ensure that we have a globally consistent mapping between the data and latent manifolds in low data regions. Secondly, the squared distance between the output of $\tilde{f}_\theta$ and its projection to the manifold:

$$\mathcal{L}_{\mathrm{P}} = \mathbb{E}_{p_{\mathrm{data}}(x)}[\|\tilde{f}_\theta(x) - f_\theta(x)\|^2] \tag{7.25}$$

discourages the output of $\tilde{f}_\theta$ from entering unprojectable regions, for example the origin when the manifold is $\mathbb{S}^n$. The same regularizations can be applied starting from the latent space.

The full loss is:

$$\mathcal{L} = \mathcal{L}_{\mathcal{M}-\mathrm{NLL}} + \beta_{\mathrm{R}}\mathcal{L}_{\mathrm{R}} + \beta_{\mathrm{U}}\mathcal{L}_{\mathrm{U}} + \beta_{\mathrm{P}}\mathcal{L}_{\mathrm{P}} \tag{7.26}$$

where the gradient of $\mathcal{L}_{\mathcal{M}-\mathrm{NLL}}$ is computed using equation (7.23), and β_{R}, β_{U} and β_{P} are hyperparameters. We give our choices in appendix B.6.

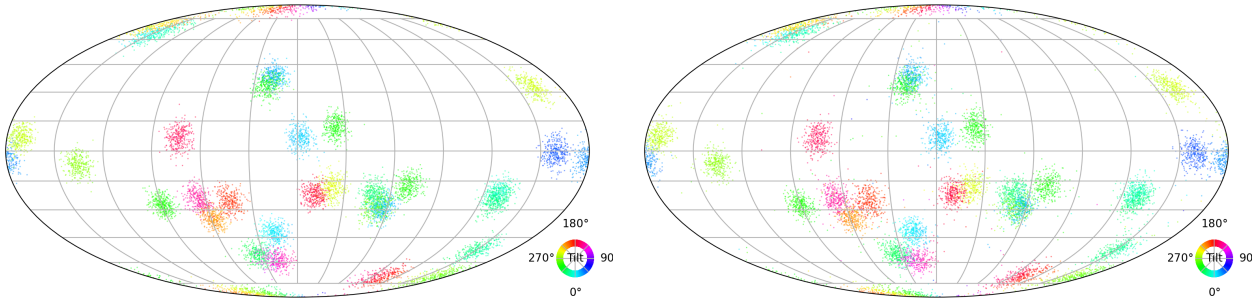


Figure 7.5.: Manifold free-form flows on a synthetic $SO(3)$ mixture distribution with $M = 64$ mixture components proposed by De Bortoli et al. (2022). *(Left)* 10,000 samples each from the ground truth distribution and *(right)* our model. This visualization computes three Euler angles, which fully describe a rotation matrix, and then plot the first two angles on the projection of a sphere and the last by color (Murphy et al., 2021). We find that our model nicely samples from the distribution with few outliers between the modes.

7.2.4. Experiments

We now demonstrate the practical performance of manifold free-form flows on various manifolds. We choose established experiments to ensure comparability with previous methods, and find:

- M-FFF matches or outperforms previous single-step methods. M-FFF uses a simple ResNet architecture, whereas previous methods were specialized to the given manifolds, hindering adoption to novel manifolds.
- M-FFF is competitive with methods sampling in several steps at significantly faster inference speeds.

In our result tables, we mark as bold (a) the best method overall (both single- and multistep), and (b) the best single-step method. We provide all details necessary to reproduce the experiments in appendix B.6. We run each experiment multiple times with different data splits and report the mean and standard deviation of those runs.

Synthetic distribution over rotation matrices The group of 3D rotations $SO(3)$ can be represented by rotations matrices with positive determinant, i.e., all $Q \in \mathbb{R}^{3\times3}$ with $Q^{\mathrm{T}}Q = I$ and $\det Q = 1$. We choose $\mathbb{R}^{3\times3}$ as our embedding space and project to the manifold by solving the constrained Procrustes problem via SVD (Lawrence et al., 2019), see appendix B.6.2.

We evaluate M-FFF on synthetic mixture distributions proposed by De Bortoli et al. (2022) with M mixture components for $M = 16, 32$ and 64. Samples from one of the distributions and samples from our model are depicted in figure 7.5.

Table 7.5 shows that M-FFF outperforms the normalizing flow developed for $SO(3)$ by Liu et al. (2023b), as well as the diffusion-based approaches for the mixtures $M = 32$ and 64.

Earth data on the sphere We evaluate manifold free-form flows on spheres with datasets from the domain of earth sciences. We use four established datasets compiled by Mathieu & Nickel (2020) for density estimation on $\mathbb{S}^2$: Volcanic eruptions (National Geophysical Data Center / World Data Service (NGDC/WDS), 2022b), earthquakes (National Geophysical Data Center / World Data Service (NGDC/WDS), 2022a), floods (Brakenridge, 2017) and wildfires (EOSDIS, 2020).

Figure 7.3 shows an example for a model trained on flood data. As the reconstruction error sometimes does not drop to a satisfactory level, we employ the method described in appendix B.6.1 to ensure that the measured likelihoods are accurate. Table 7.6 shows that M-FFF again outperforms the

Table 7.5.: Test negative log-likelihood (NLL, ↓) on $SO(3)$ for multistep and single-step methods. M-FFF consistently outperforms the specialized normalizing flow by Liu et al. (2023b) on synthetic distributions of $SO(3)$ matrices, and outperforms multistep methods in the cases with more mixture components. Multistep baseline values are due to De Bortoli et al. (2022).

	$M = 16$	$M = 32$	$M = 64$	Fast inference?
Moser flow (Rozen et al., 2021)	-0.85 ± 0.03	-0.17 ± 0.03	0.49 ± 0.02	✗: 1000 steps
Exp-wrapped SGM (De Bortoli et al., 2022)	-0.87 ± 0.04	-0.16 ± 0.03	0.58 ± 0.04	✗: 500 steps
Riemannian SGM (De Bortoli et al., 2022)	**-0.89 ± 0.03**	-0.20 ± 0.03	0.49 ± 0.02	✗: 100 steps
$SO(3)$-NF (Liu et al., 2023b)	-0.81 ± 0.01	-0.12 ± 0.004	0.61 ± 0.01	✓
M-FFF (ours)	**-0.87 ± 0.02**	**-0.21 ± 0.02**	**0.45 ± 0.02**	✓

Table 7.6.: M-FFF significantly outperforms the previous single-step density estimator (Peel et al., 2001) on the sphere on real-world earth datasets in terms of negative log-likelihood (lower is better). Baseline values are collected from De Bortoli et al. (2022); Huang et al. (2022); Chen & Lipman (2024).

	Volcano	Earthquake	Flood	Fire	Fast inference?
Riemannian CNF (Mathieu & Nickel, 2020)	-6.05 ± 0.61	0.14 ± 0.23	1.11 ± 0.19	-0.80 ± 0.54	✗: ∼100 steps
Moser flow (Rozen et al., 2021)	-4.21 ± 0.17	-0.16 ± 0.06	0.57 ± 0.10	-1.28 ± 0.05	✗: ∼100 steps
Stereographic score-based (De Bortoli et al., 2022)	-3.80 ± 0.27	-0.19 ± 0.05	0.59 ± 0.07	-1.28 ± 0.12	✗: ∼100 steps
Riemannian score-based (De Bortoli et al., 2022)	-4.92 ± 0.25	-0.19 ± 0.07	0.45 ± 0.17	-1.33 ± 0.06	✗: ∼100 steps
Riemannian diffusion (Huang et al., 2022)	-6.61 ± 0.97	**-0.40 ± 0.05**	0.43 ± 0.07	-1.38 ± 0.05	✗: >100 steps
Riemannian flow matching (Chen & Lipman, 2024)	**-7.93 ±1.67**	-0.28 ± 0.08	**0.42 ± 0.05**	**-1.86 ± 0.11**	✗: 1000 steps
Mixture of Kent (Peel et al., 2001)	-0.80 ± 0.47	0.33 ± 0.05	0.73 ± 0.07	-1.18 ± 0.06	✓
M-FFF (ours)	**-2.25 ± 0.02**	**-0.23 ± 0.01**	**0.51 ± 0.01**	**-1.19 ± 0.03**	✓
Datset size	827	6120	4875	12809	

specialized single-step model; the performance compared to multistep methods is mixed. We think that multistep models have an advantage on the considered data, as there are large regions of empty space between highly concentrated data points that have to be separated by the neural network while achieving a high reconstruction accuracy everywhere (see density and sample plots in appendix B.6.3).

Table 7.7.: M-FFF consistently outperforms normalizing flows specialized to tori (Rezende et al., 2020) on torus datasets, without requiring the development of a specialized architecture. In addition, our method comes close to the performance of the multi-step methods and outperforms them on the Glycine dataset. Baseline values are due to Huang et al. (2022); Chen & Lipman (2024).

	General	Glycine	Proline	Pre-Pro	RNA	Fast inference?
Riemannian diffusion (Huang et al., 2022)	1.04 ± 0.012	1.97 ± 0.012	**0.12** ± 0.011	1.24 ± 0.004	-3.70 ± 0.592	✗: ∼1000 steps
Riemannian flow matching (Chen & Lipman, 2024)	**1.01 ± 0.025**	1.90 ± 0.055	0.15 ± 0.027	**1.18 ± 0.055**	**-5.20 ± 0.067**	✗: 1000 steps
Mixture of power spherical (Huang et al., 2022)	1.15 ± 0.002	2.08 ± 0.009	0.27 ± 0.008	1.34 ± 0.019	4.08 ± 0.368	✓
Circular Spline Coupling Flows (Rezende et al., 2020)	**1.03 ± 0.01**	1.91 ± 0.04	0.21 ± 0.08	1.24 ± 0.04	-4.01 ± 0.24	✓
M-FFF (ours)	**1.03 ± 0.02**	**1.89 ± 0.05**	**0.17 ± 0.08**	**1.23 ± 0.04**	**-4.27 ± 0.09**	✓

Torsion angles of molecules on tori To benchmark manifold free-form flows on tori $\mathbb{T}^n$, we follow (Huang et al., 2022) and evaluate our model on two datasets from structural biology. We consider the torsion (dihedral) angles of the backbone of protein and RNA substructures, respectively.

We represent a tuple of angles $(\varphi_1, \dots, \varphi_n) \in [0, 2\pi]^n$ by mapping each angle to a position on a circle: $X_i = (\cos\varphi_i, \sin\varphi_i) \in \mathbb{S}^1$. Then we stack all X_i into a matrix $X \in \mathbb{R}^{n\times 2}$, compare table 7.4.

The first dataset comprises 500 proteins assembled by (Lovell et al., 2003) and is located on $\mathbb{T}^2$. The three-dimensional arrangement of a protein backbone can be described by the so called

Table 7.8.: Test NLL on Stanford bunny data proposed by (Chen & Lipman, 2024), living on a manifold with nontrivial curvature (see figure 7.3). M-FFF outperforms the multistep model for datasets with more modes.

	$k = 10$	$k = 50$	$k = 100$	Fast inference?
Riemannian Flow Matching (w/ diffusion) (Chen & Lipman, 2024)	1.16 ± 0.02	1.48 ± 0.01	1.53 ± 0.01	✗: 1000 steps
Riemannian Flow Matching (w/ biharmonic) (Chen & Lipman, 2024)	**1.06** ± **0.05**	1.55 ± 0.01	1.49 ± 0.01	✗: 1000 steps
M-FFF (ours)	**1.21** ± **0.01**	**1.34** ± **0.01**	**1.28** ± **0.01**	✓

Ramachandran angles (Ramachandran et al., 1963) Φ and Ψ, which represent the torsion of the protein backbone around the N-C_α and C_α-C bonds. The data is split into four distinct subsets *General*, *Glycine*, *Proline* and *Pre-Proline*, depending on the residue of each substructure.

The second dataset is extracted from a subset of RNA structures introduced by Murray et al. (2003). As the RNA backbone structure can be characterized by seven torsion angles, in this case we are dealing with data on $\mathbb{T}^7$.

We report negative log-likelihoods in table 7.7, finding that M-FFF outperforms a circular spline coupling flow, a normalizing flow particularly developed for data on tori (Rezende et al., 2020), as well as the multistep methods on one of the datasets. In addition to the quantitative results, we show the log densities of the M-FFF models for the four protein datasets infigure B.12 in appendix B.6.4 .

Toy distributions on hyperbolic space We apply M-FFF to the Poincaré ball model, which embeds the 2-dimensional hyperbolic space $\mathbb{H}^2$ of constant negative curvature -1 in the 2-dimensional Euclidean space $\mathbb{R}^2$, as specified in table 7.4. As this embedding is not isometric, and distances between points grow when moving away from the origin, we must include the last term of equation (7.14) when changing variables under a map on this embedded manifold.

We show that M-FFF can be applied to non-isometric embeddings using equation (7.14) and visualize learned densities in figure 7.3 and in figure B.13 in appendix B.6.5 for several toy datasets defined on the 2-dimensional Poincaré ball model. Further details can be found in appendix B.6.5.

Manifold with non-trivial curvature Finally, we follow Chen & Lipman (2024) and train M-FFF on synthetic distributions on the Stanford bunny (Turk & Levoy, 1994) on the data provided with their paper, see figure 7.3. The natural embedding of this mesh is $\mathbb{R}^3$, and we train a separate neural network to project from the embedding space to the mesh. This ensures that the projection is continuously differentiable, which we identify to be important for stable gradients.

Table 7.8 shows that M-FFF performs well on this manifold, outperforming Riemannian flow matching in two out of three cases. This experiment underlines the flexibility of our model: We only need a projection function to the manifold to train a generative model.

7.2.5. Conclusion

In this section, we present Manifold Free-Form Flows (M-FFF), a generative model designed for manifold data. To the best of our knowledge, it is the first generative model on manifolds with single-step sampling and density estimation readily applicable to arbitrary Riemannian manifolds. This significantly accelerates inference and allows for deployment on edge devices.

M-FFF matches or outperforms single-step architectures specialized to particular manifolds. It also surpasses multistep methods in some cases, despite the greatly reduced inference compute.

Adapting M-FFF to new manifolds is straightforward and only requires selecting an embedding space and a projection to the manifold. In contrast, competing multistep methods are more challenging to adapt, as they require implementing a diffusion process or computing distances on the manifold.

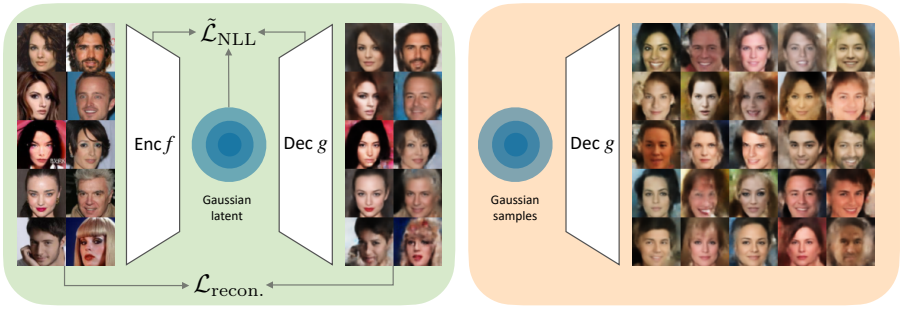


Figure 7.6.: Free-form injective flow (FIF) training and inference. *(Left)* We combine a reconstruction loss $\mathcal{L}_{\text{recon.}}$ with a novel maximum likelihood loss $\tilde{\mathcal{L}}_{\text{NLL}}$ to obtain an injective flow without architectural constraints. *(Right)* We generate novel samples by decoding standard normal latent samples with our best-performing models on CelebA and MNIST. The reconstructions shown are on CelebA validation data, the samples are uncurated samples from our models.

7.3. Free-form injective flows for learning compressed representations and distributions (FIF)

This section is adapted from Sorrenson et al. (2024), see the list of individual contributions in chapter 3.

The manifold hypothesis suggests that realistic data lies on a low-dimensional manifold embedded into a high-dimensional data space (Bengio et al., 2013). This implies that it is more efficient to model distributions on a low-dimensional representation and regard deviations from that representation as uninformative noise. Prior works such as Caterini et al. (2021); Brehmer & Cranmer (2020) have restricted normalizing flows to low-dimensional subspaces via specially constructed bottleneck architectures (known as "invertible autoencoders" (Teng & Choromanska, 2019) or "injective flows" (Kothari et al., 2021)) where encoder and decoder share parameters. These injective flows are optimized by some version of maximum likelihood training. This is not an ideal design decision, as the architectures used in such models were originally designed for tractable change of variables calculations in normalizing flows, but such calculations are not possible in the presence of a bottleneck (Brehmer & Cranmer, 2020). As a result, we propose to drop the restrictive constructions usually coming with invertible neural networks (see section 4.3), instead use an unconstrained encoder and decoder, and generalize the free-form flow gradient estimator to the *injective* change of variables formula. This greatly simplifies the design of the model and makes it more expressive.

The resulting gradient estimator is an improvement to the one used by rectangular flows (Caterini et al., 2021). We simplify their estimator considerably by replacing iterative conjugate gradient with an efficient single-step estimator. This is fast: a batch can be processed in about twice the time (or less) as an autoencoder trained on reconstruction loss only. In addition, we make a novel observation about injective flows: naively training with maximum likelihood is ill-defined due to the possibility of diverging curvature in the decoding function. To resolve this problem, we propose a modification to our maximum likelihood estimator which counteracts the possibility of diverging curvature. We call our model the *free-form injective flow* (FIF).

To summarize, we make the following contributions:

- We introduce an efficient gradient estimator for the injective change-of-variables and use it to train an unconstrained injective flow for the first time (section 7.3.3).
- We identify pathological behavior in the naive application of maximum likelihood training in the presence of a bottleneck, and offer a solution to avoid this behavior while maintaining computational efficiency (section 7.3.3).

- We outperform previous injective flows and demonstrate competitive performance to generative autoencoders on toy, tabular and image data (section 7.3.4).

7.3.1. Related work

Injective flows jointly learn a manifold and maximize likelihood on that manifold. The latter requires estimating the Jacobian determinant of the transformation to calculate the change of variables. Efficient computation of this determinant traditionally imposed two major restrictions on normalizing flow architectures: Firstly, the latent space has to match in dimension with the data space, ruling out bottleneck architectures. Secondly, normalizing flows are restricted to certain functional forms, such as coupling and autoregressive blocks. Below, we outline the existing approaches to overcome these problems and how our solution compares.

Lower-dimensional latent spaces One set of methods attempts to use full-dimensional normalizing flows, with some additional regularization or architectural constraints, such that a subspace of the latent space corresponds to the manifold. One strategy adds noise to the data to make it a full-dimensional distribution, then denoises to the manifold (Horvat & Pfister, 2021; Loaiza-Ganem et al., 2022). Another restricts the non-manifold latent dimensions to have small variance (Beitler et al., 2021; Silvestri et al., 2023; Zhang et al., 2023b).

Other methods sidestep the problem by making training into a two-step procedure. First, an autoencoder is trained on reconstruction loss, then a normalizing flow is trained to learn the resulting latent distribution. In this line of work, (Brehmer & Cranmer, 2020) and (Kothari et al., 2021) use an injective flow, while (Böhm & Seljak, 2022) use unconstrained networks as autoencoder. Ghosh et al. (2020) additionally regularize the decoder.

Conformal embedding flows (Ross & Cresswell, 2021) ensure the decomposition of the determinant into the contribution from each block by further restricting the architecture to exclusively conformal transformations. Cramer et al. (2022) uses an isometric autoencoder such that the change of variables is trivial. However, the resulting transformations are quite restrictive and cannot represent arbitrary manifolds.

The most similar work to ours is the rectangular flow (Caterini et al., 2021) which estimates the gradient of the log-determinant via an iterative, unbiased estimator. The resulting method is quite slow to train, and uses injective flows, which are restrictive.

Unconstrained normalizing flow architectures Several works attempt to reduce the constraints imposed by typical normalizing flow architectures, allowing the use of free-form networks. However, all of these methods only apply to full-dimensional architectures. FFJORD (Grathwohl et al., 2019) is a type of continuous normalizing flow (Chen et al., 2018b) which estimates the change of variables stochastically. Residual flows (Behrmann et al., 2019; Chen et al., 2019) make residual networks invertible, but require expensive iterative estimators to train via maximum likelihood. Self-normalizing flows (Keller et al., 2021) and relative gradient optimization (Gresele et al., 2020) estimate maximum likelihood gradients for the matrices used in neural networks, but restrict the architecture to use exclusively square weight matrices without skip connections.

Approximating maximum likelihood Many methods optimize some bound on the full-dimensional maximum likelihood, notably the variational autoencoder (Kingma & Welling, 2014) and its variants. (Cunningham et al., 2020) also optimizes a variational lower bound to the likelihood. Other methods fit into the injective flow framework by jointly optimizing a reconstruction loss and some approximation to maximum likelihood on the manifold: Kumar et al. (2020); Zhang et al. (2020b) approximate

the log-determinant of the Jacobian by its Frobenius norm. The entropic AE (Ghose et al., 2020) maximizes the entropy of the latent distribution by a nearest-neighbor estimator while constraining its variance, resulting in a Gaussian latent space. In addition, there are other ways to regularize the latent space of an autoencoder which are not based on maximum likelihood, e.g. adversarial methods (Makhzani et al., 2015).

In contrast to the above, our approach jointly learns the manifold and maximizes the likelihood on it with an unconstrained architecture, which easily accommodates a lower-dimensional latent space.

7.3.2. Injective change-of-variables

We now generalize normalizing flows in section 4.2 to bottleneck architectures. In particular, let $f_\theta : \mathbb{R}^D \to \mathbb{R}^d$ be an encoder which compresses data to a latent space and a decoder $g_\varphi : \mathbb{R}^d \to \mathbb{R}^D$ which decompresses the latent representation. A full-dimensional model has $d = D$ while a bottleneck model has $d < D$.

Normalizing flows in full-dimensional space were constructed using *invertible* functions. If $d < D$, f_θ and g_φ cannot be invertible. Instead, we consider an *injective* decoder g_φ, that is $g_\varphi(z_1) = g_\varphi(z_2)$ implies $z_1 = z_2$. We call the image of the decoder the *learned manifold* $\mathcal{M}_\varphi = g_\varphi(\mathbb{R}^d) = \{g_\varphi(z) : z \in \mathbb{R}^d\}$.

We call a pair f_θ and g_φ *consistent* if $f_\theta \circ g_\varphi : \mathbb{R}^d \to \mathbb{R}^d$ is the identity. For a fixed decoder g_φ, consistency does not fully determine f_θ: The encoder f_θ has additional degrees of freedom in where to map points that do not lie on the learned manifold. Similarly, given a fixed encoder, g_φ can map a latent code z to any element in $f^{-1}(\{z\})$ and still be consistent. This is why unlike in the full-dimensional case, where the inverse function is unique, we directly consider separate functions f_θ and g_φ.

The change of variables in equation (4.1) transfers to an injective decoder for $x \in \mathcal{M}_\varphi$ via a result from differential geometry (Krantz & Parks, 2008):

$$p_\varphi(x) = p(z)\sqrt{|g'_\varphi(f_\theta(x))^{\mathrm{T}} g'_\varphi(f_\theta(x))|}^{-1} \tag{7.27}$$

Here, $z = g_\varphi^{-1}(x)$, which exists when $x \in \mathcal{M}_\varphi$.

We can also write this change of variables equation in terms of the encoder if it holds that the encoder Jacobian $f'_\theta(x)$ is the *pseudoinverse* of the decoder Jacobian $g'_\varphi(z)$ for $x \in \mathcal{M}_\varphi$ and $z = f_\theta(x)$:

$$f'_\theta(x) = (g'_\varphi(z))^\dagger \quad \text{where } A^\dagger = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}. \tag{7.28}$$

In this case, we can rewrite equation (7.27) via the encoder Jacobian:

$$p_\varphi(x) = p_\theta(x) = p(z = f_\theta(x))\sqrt{|f'_\theta(x) f'_\theta(x)^{\mathrm{T}}|}. \tag{7.29}$$

This change of variables only holds for $x \in \mathcal{M}_\varphi$, as points outside of $\mathcal{M}_\varphi$ are never generated by g_φ. We can still perform density estimation for points outside of $\mathcal{M}_\varphi$, by first projecting points $x \notin \mathcal{M}_\varphi$ via $\hat{x} = g_\varphi(f_\theta(x)) \in \mathcal{M}_\varphi$ and then computing $p_\theta(\hat{x})$. For example, this can be useful if $g_\varphi \circ f_\theta$ denoises the data and then a denoised density is computed.

7.3.3. Free-form injective flow (FIF)

If we want to perform maximum likelihood training using equation (7.29) with gradient descent for a point $x \in \mathcal{M}_\varphi$, we need to compute the gradient of the negative logarithm of the injective change of variables formula. Naively, we need $O(d)$ computations to compute $f'_\theta(x)$, which can quickly become prohibitively expensive. Again, we construct an efficient gradient estimator just like for free-form flows:

Theorem 7.4 (Injective volume change gradient, proof in appendix A.10.1)**.** *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^d$ be the pseudoinverse to a C^1 and injective $g_\varphi : \mathbb{R}^d \to \mathbb{R}^D$. Then, for all $x \in \mathcal{M}_\varphi$:*

$$\frac{1}{2}\nabla_\theta \log |f'_\theta(x) f'_\theta(x)^{\mathrm{T}}| = \mathrm{tr}\Big((\nabla_\theta f'_\theta(x)) g'_\varphi(z)\Big) \tag{7.30}$$

$$= \mathbb{E}_v\Big[v^{\mathrm{T}}(\nabla_\theta f'_\theta(x)) g'_\varphi(z) v\Big], \tag{7.31}$$

where $v \in \mathbb{R}^d$ is a random variable with zero mean and unit covariance and $z = f_\theta(x)$.

This estimator has the same form as for the full-dimensional change-of-variables in theorem 7.1, only the dimension of v is adapted to the latent space. Equation (7.31) improves over a related gradient estimator, that is based on conjugate gradient (Caterini et al., 2021):

$$\frac{1}{2}\nabla_\theta \log |f'_\theta f'^{\mathrm{T}}_\theta| = \frac{1}{2}\mathbb{E}_v\left[v^{\mathrm{T}}(f'_\theta f'^{\mathrm{T}}_\theta)^{-1}\Big(\nabla_\theta(f'_\theta f'^{\mathrm{T}}_\theta)\Big) v\right] \tag{7.32}$$

$$= \frac{1}{2}\mathbb{E}_v\left[\mathtt{CG}(f'_\theta f'^{\mathrm{T}}_\theta; v)^{\mathrm{T}}\Big(\nabla_\theta(f'_\theta f'^{\mathrm{T}}_\theta)\Big) v\right], \tag{7.33}$$

where conjugate gradient $x = \mathtt{CG}(A, b)$ iteratively constructs the solution to $Ax = b$ for a non-singular A. Structurally, both equations (7.31) and (7.33) only rely on Jacobian-vector respectively vector-Jacobian products. However, the latter variant is computationally more expensive by a factor since conjugate gradient may require several iterations for sufficient convergence and the Jacobian appears twice in each product.

This suggests that we are now ready to train models with the efficient gradient estimator. However, we first need to fix an important pathology, arising because the maximum likelihood term influences what $\mathcal{M}_\varphi$ is learned.

Problems with maximum likelihood in the presence of a bottleneck

Remember that equation (7.29) only holds for data $x \in \mathcal{M}_\varphi$. In this section, we consider what goes wrong if we choose to optimize with projected data $\hat{x} = g_\varphi(f_\theta(x))$ instead. A variant of this issue has already been raised by Brehmer & Cranmer (2020). We expand on their argument here, identifying an additional pathology that cannot be overcome by adding a reconstruction loss. Consider using the volume change in equation (7.29) as a per-sample injective maximum likelihood on projected data:

$$\mathcal{L}_{\text{I-NLL}} = \mathbb{E}_{\hat{x}\sim p_{\theta,\varphi}(\hat{x})}\Big[-\log p_\theta(\hat{x})\Big] = \mathbb{E}_{\hat{x}\sim p_{\theta,\varphi}(\hat{x})}\left[-\log p(z) - \frac{1}{2}\log\det\Big[f'_\theta(\hat{x})^{\mathrm{T}} f'_\theta(\hat{x})\Big]\right], \tag{7.34}$$

where we denote by $p_{\theta,\varphi}(\hat{x})$ the distribution obtained from projecting $p(x)$ via $g_\varphi \circ f_\theta$. Now consider that the negative log-likelihood loss is one part of a KL divergence, and KL divergences are always non-negative:

$$\mathcal{D}_{\mathrm{KL}}(p_{\theta,\varphi}(\hat{x}) \| p_\theta(\hat{x})) = -H[p_{\theta,\varphi}(\hat{x})] - E_{p_{\theta,\varphi}(\hat{x})}[\log p_\theta(\hat{x})] \geq 0. \tag{7.35}$$

As a result, the loss is lower bounded by the entropy of the data projected onto the manifold:

$$\mathcal{L}_{\text{I-NLL}} \geq H[p_{\theta,\varphi}(\hat{x})]. \tag{7.36}$$

Unlike in standard normalizing flow optimization, where the right hand side would be fixed, here the entropy depends on the projection learned by the model, as indicated by the parameters (φ, θ). Thus, the model can modify the projection such that entropy is as low as possible. We break this pathology down into two cases:

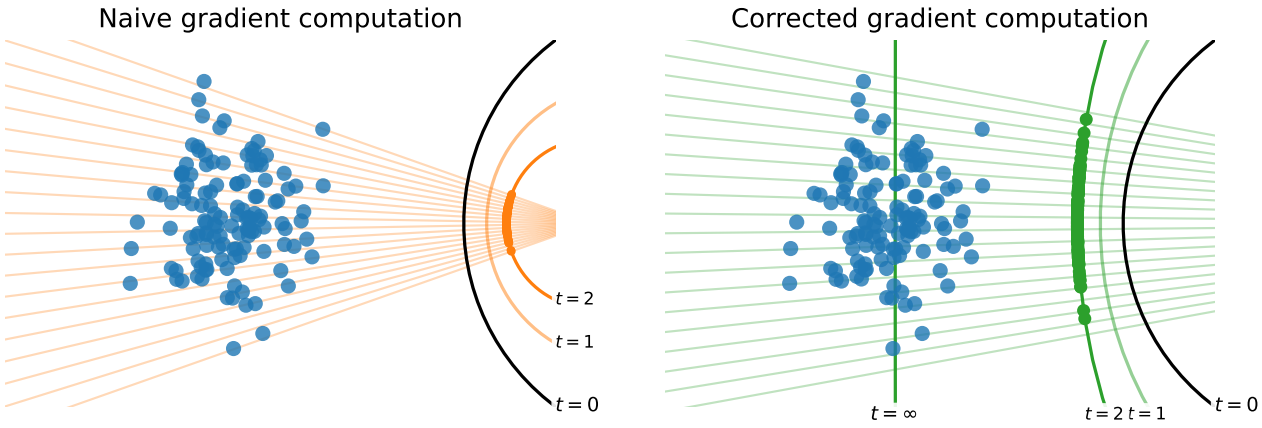


Figure 7.7.: Naive training of autoencoders with negative log-likelihood (NLL, see section 7.3.3) leads to pathological solutions *(left)*. Starting with the initialization ($t = 0$, black), gradient steps increase the curvature of the learnt manifold ($t = 1, 2$, orange). This reduces NLL because the entropy of the projected data is reduced, by moving the points closer to one another. This effect is stronger than the reconstruction loss. We fix this problem by evaluating the volume change off-manifold *(right)*. This moves the manifold closer to the data and reduces the curvature ($t = 1, 2$, green), until it eventually centers the manifold on the data with zero curvature ($t = \infty$, green). Light lines show the set of points which map to the same latent point. Data is projected onto the $t = 2$ manifold.

1. *A model manifold which does not align with the data manifold but instead intersects it*. For example, Brehmer & Cranmer (2020) discuss a case where a linear model learns to project a data distribution to a single point on the manifold, thus reducing its entropy to $-\infty$, the lowest possible value. To the best of our knowledge, this can be fixed by adding noise and a reconstruction loss with sufficiently high weight. In appendix A.10.2 we prove as much for linear models and characterize the solutions, which are the same as PCA if the weight of a reconstruction loss $\beta \geq 1/2\sigma^2$ where σ^2 is the smallest eigenvalue of the data covariance matrix.

2. *A model manifold which concentrates the data by use of high curvature,* see figure 7.7 *(left)*. This newly identified pathological case only occurs in nonlinear models, hence Brehmer & Cranmer (2020) did not notice this effect in their linear example. Importantly, this is **not** fixed by adding a reconstruction loss.

Most existing injective flows avoid this by a two-stage training, which first learns a projection and then the distribution of the projected data in the latent space. To enable jointly learning a manifold and a maximum-likelihood density on it, we need to find a fix for the pathology.

Towards a well-behaved loss The term which leads to pathological behavior in the likelihood loss is the log-determinant. When using the change of variables with f'_θ evaluated at $\hat{x}$, all that matters is the change of volume from the projected data to the latent space, so the model can decrease the loss by choosing a manifold which concentrates the projected data more tightly (the more possibility it has to expand the data, the lower the loss will be). We can counteract this effect by introducing a factor inversely proportional to the concentration. This can be achieved by the fairly simple modification of evaluating f'_θ in our estimator at x rather than $\hat{x}$, that is *removing* the projection operation and directly working with the training data (plus optionally added noise). Namely, we modify equation (7.31) to estimate the gradient of the log-determinant term by:

$$v^\mathrm{T} f'_\theta(x) \mathtt{SG}\Big(g'_\varphi(z)v\Big). \tag{7.37}$$

See appendix A.10.1 for a detailed explanation.

In this way, we discourage pathological solutions involving high curvature. In figure 7.7 (*right*) we can see the effect of the modified estimator: the manifold now moves towards the data since the optimization is not dominated by diverging curvature. We note that the modified estimator is also computationally cheaper, since we do not need to compute the projection through the network before computing the likelihood terms.

We add a reconstruction loss to train the projection, whose minimizer yields a pair of pseudoinverse functions (see appendix A.10.1 for a derivation). This results in training with the following gradient:

$$\nabla_{\theta,\varphi}\mathcal{L}_{\text{FIF}}(x) = \nabla_{\theta,\varphi}\mathbb{E}_{x\sim p(x)}\Big[-\log p(z) - v^{\mathrm{T}} f'_\theta(x)\,\texttt{SG}\Big(g'_\varphi(f_\theta(x))v\Big) + \beta\|g_\varphi(f_\theta(x)) - x\|^2\Big]. \tag{7.38}$$

Phase transition Figure 7.8 shows that when using this loss, if β is large enough, the dominant manifold direction is identified. In appendix B.7.2, we show a similar experiment on MNIST.

Architectures

In the existing literature, *injective flows* (Brehmer & Cranmer, 2020), also called *invertible autoencoders* (Teng & Choromanska, 2019), adapt invertible neural network architectures as the ones given in section 4.3 to a bottleneck setting. They parameterize f_θ and g_φ as the composition of two invertible functions, w_{θ_w} defined in $\mathbb{R}^D$ and h_{θ_h} defined in $\mathbb{R}^d$, with a slicing/padding operation in between:

$$f_\theta = h_{\theta_h}^{-1} \circ \texttt{slice} \circ w_{\theta_w}^{-1} \quad \text{and} \quad g_\varphi = w_{\theta_w} \circ \texttt{pad} \circ h_{\theta_h}, \tag{7.39}$$

where $\texttt{slice}(x)$ selects the first d elements of x and $\texttt{pad}(z)$ concatenates $D-d$ zeros to the end of z. Since `slice` and `pad` are *consistent*, so too are f_θ and g_φ. Note that injective flows share weights between encoder and decoder, that is $\theta = \varphi$. Note that unlike the full-dimensional normalizing flows, injective in most cases do not provide a computational advantage in computing the volume change. In addition, while consistent, the pair of encoder and decoder as constructed above are (in general) not pseudoinverses, so that only the decoder change of variables formula in equation (7.27) yields accurate densities.

Like for (manifold) free-form flows in sections 7.1 and 7.2, we again drop all architectural constraints and instead make use of arbitrary neural network architectures for f_θ and g_φ. In our experiments, we use off-the-shelf autoencoders. In some cases, we find that adding additional residual blocks on the bottleneck dimensions helps performance.

7.3.4. Experiments

In this section, we test the empirical performance of the proposed model. First, we show that our model is much faster than rectangular flows on tabular data. Second, we show that it outperforms previous SOTA injective flows on generating images. Finally, we compare against other generative autoencoders on the Pythae image generation benchmark (Chadebec et al., 2022), achieving the best FID score in some categories.

Implementation details In implementing the trace estimator, we have to make a number of choices. Briefly, i) we chose to formulate the log-determinant gradient in terms of the encoder rather than decoder as it was more stable in practice, ii) we performed traces in the order $f'_\theta(x)g'_\varphi(z)$ as this reduces variance (both orderings are valid due to the cyclic property of the trace but since $f'_\theta(x)g'_\varphi(z)$ is a $d \times d$ matrix whereas $g'_\varphi(z)f'_\theta(x)$ is $D \times D$, the former is typically easier to estimate), iii) we used a mixture of forward- and backward-mode automatic differentiation as this was compatible with our

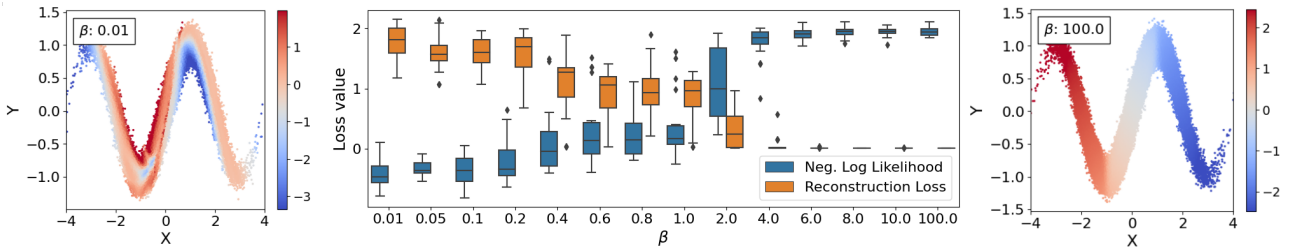


Figure 7.8.: Learning a noisy 2-D sinusoid with a 1-D latent space for different reconstruction weights β. Color codes denote the value of the latent variable at each location. When the reconstruction term has low weight (*left*), the autoencoder learns to throw away information about the position along the sinusoid and only retains the orthogonal noise. Only sufficiently high weights (*right*) result in the desired solution, where the decoder spans the sinusoid manifold. The middle plot shows the tradeoff between reconstruction error and NLL as we transition between these regimes (box plots indicate variability across runs).

Table 7.9.: Free-form injective flows (FIF) are significantly faster than rectangular flows (RF) with superior performance in FID-like metric on 3 out of 4 tabular datasets (Papamakarios et al., 2017). Both methods use $K = 1$. The results for RF are taken directly from (Caterini et al., 2021).

Method	POWER	GAS	HEPMASS	MINIBOONE
RF (Caterini et al., 2021)	0.083 ± 0.015	**0.110 ± 0.021**	0.779 ± 0.191	1.001 ± 0.051
FIF (ours)	**0.041 ± 0.007**	0.281 ± 0.031	**0.541 ± 0.034**	**0.598 ± 0.024**
Training Time Speedup	**3.9** ×	**2.2** ×	**6.1** ×	**1.5** ×

estimator, and iv) we used orthogonalized Gaussian noise in the trace estimator, to reduce variance. Full justification for these choices is given in appendix B.7.1.

Tabular Data

We evaluate our method on four of the tabular datasets used by Papamakarios et al. (2017), using the same data splits, and make a comparison to the published rectangular flow results (Caterini et al., 2021), see table 7.9. We adopt the "FID-like metric" from that work, which computes the Wasserstein-2 distance between the Gaussian distributions with equal mean and covariance as the test data and the data generated by the model. This is a measure of the difference of the means and covariance matrices of the generated and test datasets. We outperform rectangular flows on all datasets except GAS. In addition, we see a speedup in training time of between 1.5 and 6 times between FIF and a rerun of rectangular flows (using the published code) on the same hardware. Full experimental details are in appendix B.7.2.

In appendix B.7.2, we also perform an ablation comparing the different changes in the gradient computation in table B.16. It shows that the on-manifold loss computation is unstable if combined with a free-form autoencoder, but using an injective flow based on an invertible neural network can stabilize training. This may be explained by the restrictive architecture. We notice that a free-form architecture with our off-manifold gradient computation yields the best results overall.

Comparison to injective flows

We compare FIF against previous injective flows on CelebA images (Liu et al., 2015) in table 7.10. Our models significantly improve the quality of the generated images in terms of the Fréchet inception distance (FID) (Heusel et al., 2017) and Inception Score (IS) (Salimans et al., 2016). The former compares generated samples to a set of reference samples, by computing the Wasserstein-2 distance

between two Gaussian distributions fit to some embedding of the respective sets of samples. The latter measures diversity by the entropy of the distribution of class labels in the generated samples, where the class labels are provided by some pre-trained classifier. Samples from this model are depicted in figure 7.6.

For a fair comparison, we train each model on the same hardware for equal wall clock time with the code provided by the authors. The architectures of previous works were dominated by the need that most layers are invertible and have a tractable Jacobian determinant. Our loss in equation (7.38) does not impose these constraints on the architecture, and we can use an off-the-shelf convolutional auto-encoder with additional fully-connected layers in the latent space. Details can be found in appendix B.7.2.

Comparison to generative autoencoders

As free-form injective flows (FIF) do not require any specific architecture, we expand our comparison to the much broader range of *generative autoencoders*. This is a general class of bottleneck architectures that encode the training data to a standard normal distribution so that the decoder can be used as a generator after training.

Recently, Chadebec et al. (2022) proposed the Pythae benchmark for comparing generative autoencoders on image generation. They evaluate different training methods using two different architectures on MNIST (LeCun et al., 2010) (data $D = 784$, latent $d = 16$), CIFAR10 (Krizhevsky, 2009) ($D = 3072, d = 256$), and CelebA (Liu et al., 2015) ($D = 12288, d = 64$). All models are trained with the same limited computational budget. The goal of the benchmark is to provide a fair comparison of different models, not to achieve SOTA image generation results, as this would require significantly more compute.

As shown in table 7.11, our model performs strongly on the benchmark, achieving SOTA on CelebA in Fréchet Inception Distance (FID) (Heusel et al., 2017) on the ResNet architecture with latent codes sampled from a standard normal, and on both architectures when sampling from a Gaussian Mixture Model fit using training data. At the same time, the Inception Scores (IS) (Salimans et al., 2016) are high, indicating a high diversity. In the one combination where our model does not outperform the competitors, FIF still achieves a comparable FID and high Inception Score. FIF also performs strongly on the other datasets, see appendix B.7.2.

For each method in the benchmark, ten hyperparameter configurations are trained and the best model according to FID is reported. For our method, we choose to vary $\beta = 5, 10, 15, 20, 25$ and the number of Hutchinson samples $K = 1, 2$. We find the performance to be robust against these choices, and provide all details on the training procedure in appendix B.7.2.

7.3.5. Conclusion

Free-form injective flows offer a computationally efficient solution to learning a compressed representation of data together with its distribution. We i) significantly improve an existing estimator for the

Table 7.10.: Comparison of injective flows on CelebA under equal computational budget. Free-form Injective Flows (FIF) outperform previous work significantly in terms of FID.

Model	# parameters	$\mathcal{N}$ sampler		GMM sampler	
		FID ↓	IS ↑	FID ↓	IS ↑
DNF (Horvat & Pfister, 2021)	39.4M	55.6 ± 0.59	1.9	52.7 ± 0.33	2.0
Trumpet (Kothari et al., 2021)	19.1M	56.2 ± 1.39	1.8	47.7 ± 2.24	1.9
FIF	34.3M	**47.3 ± 1.39**	1.7	**37.4 ± 1.35**	2.0

Table 7.11.: Pythae benchmark results on CelebA, following (Chadebec et al., 2022). We train their architectures (ConvNet and ResNet) with our new training objective, achieving SOTA FID on ResNet. We draw latent samples from standard normal "$\mathcal{N}$" or a GMM fit using training data "GMM". Models with multiple variants (indicated in brackets) have been merged to indicate only the best result across variants. We mark the best FID in each column in **bold** and underline the second best.

Model	ConvNet + $\mathcal{N}$		ResNet + $\mathcal{N}$		ConvNet + GMM		ResNet + GMM	
	FID ↓	IS ↑	FID	IS	FID	IS	FID	IS
VAE (Kingma & Welling, 2014)	<u>54.8</u>	1.9	66.6	1.6	52.4	1.9	63.0	1.7
IWAE (Burda et al., 2015)	55.7	1.9	67.6	1.6	52.7	1.9	64.1	1.7
VAE-lin NF (Rezende & Mohamed, 2015)	56.5	1.9	67.1	1.6	53.3	1.9	62.8	1.7
VAE-IAF (Kingma et al., 2016)	55.4	1.9	66.2	1.6	53.6	1.9	62.7	1.7
β-(TC) VAE (Higgins et al., 2017; Chen et al., 2018a)	55.7	1.8	65.9	1.6	51.7	1.9	59.3	1.7
FactorVAE (Kim & Mnih, 2018)	**53.8**	1.9	66.4	1.7	52.4	2.0	63.3	1.7
InfoVAE - (RBF/IMQ) (Zhao et al., 2017)	55.5	1.9	66.4	1.6	52.7	1.9	62.3	1.7
AAE (Makhzani et al., 2015)	59.9	1.8	<u>64.8</u>	1.7	53.9	2.0	58.7	1.8
MSSSIM-VAE (Snell et al., 2017)	124.3	1.3	119.0	1.3	124.3	1.3	119.2	1.3
Vanilla AE	327.7	1.0	275.0	2.9	55.4	2.0	<u>57.4</u>	1.8
WAE - (RBF/IMQ) (Tolstikhin et al., 2018)	64.6	1.7	67.1	1.6	51.7	2.0	57.7	1.8
VQVAE (van den Oord et al., 2017)	306.9	1.0	140.3	2.2	<u>51.6</u>	2.0	57.9	1.8
RAE - (L2/GP) (Ghosh et al., 2020)	86.1	2.8	168.7	3.1	52.5	1.9	58.3	1.8
FIF (ours)	56.9	2.1	**62.3**	1.7	**47.3**	1.9	**55.0**	1.8

gradient of the change of variables across dimensions, ii) note that it can be applied to unconstrained architectures, iii) analyze problems with joint manifold and maximum-likelihood training and offer a solution, and iv) implement and test our model on toy, tabular and image datasets. We find that the model is practical and scalable, outperforming comparable injective flows, and showing similar or better performance to other autoencoder generative models on the Pythae benchmark.

Several theoretical and practical questions remain for future work: We identified a problem with jointly learning a manifold and maximum likelihood. We propose a fix in section 7.3.3 that provides high-quality models, but further investigation is needed for a thorough understanding. Fitting a GMM to the latent space after training improves performance on image data, suggesting that our latent distributions are not perfectly Gaussian. We generally find that architectures with more fully-connected layers in the latent space have a more Gaussian latent distribution, suggesting that larger models suffer less from this problem. We leave potential theoretical or practical improvements to future work.

7.4. Conclusion

Using architectures that incorporate prior knowledge about the data at hand yields to superior performance. Developing coupling architectures respecting beneficial properties increases the complexity of modeling the data, and in some cases has not been possible, such as for rotation-equivariant networks and for arbitrary Riemannian manifolds.

Free-form flows and its variants overcome this limitation by allowing the training of arbitrary neural networks as normalizing flows. The two essential ingredients are the efficient estimator for the change of variables term and the usage of two separate neural network for the forward and inverse mapping in a normalizing flows, coupled by a reconstruction loss.

We show the versatility of the free-form flow framework by adapting it to the following three classes of applications:

- Standard Euclidean geometry, where data is modeled as a full-dimensional distribution, by mapping it to a latent distribution with equal dimension. We develop an estimator for the gradient estimator in theorem 7.1.

- Non-Euclidean geometry, where the data is modeled in an embedding space, but the mapping between data and latent space is restricted to the manifold and respects its geometry. Theorem 7.3 generalizes the gradient estimator to the tangent space of Riemannian manifolds.
- Learning the data distribution in terms of a compressed representation. Theorem 7.4 modifies the gradient estimator to a latent space with reduced dimension.

The resulting gradient estimators are remarkably similar, and we hypothesize that the estimator can also be applied in more scenarios such as missing data, time series, and energy-based training.

In practice, free-form flows perform comparable or better than previous normalizing flows. They are competitive with multistep methods at typically two orders of magnitude faster inference.

However, there are several points that can be improved: Free-form architectures are not guaranteed to fulfill the invertibility/injectivity assumptions underlying the change of variables equations. This may influence the training dynamics, and it needs to be considered when reporting negative log-likelihood as a performance metric, which we address in section 7.1.3. The gradient estimator can exhibit a high variance, potentially slowing down training and preventing the model from sticking at the true optimum. On the positive side, this may partially prevent overfitting. Because of the use of a stop-gradient operation, the training dynamics are governed by a gradient field that is not necessarily the gradient of a loss.

8. Conclusion

This thesis contributes significantly to modeling arbitrary probability distributions with normalizing flows. We propose a new theoretical framework of coupling-based normalizing flows, and we introduce free-form flows, which allow training arbitrary neural network architectures as normalizing flows.

Our theoretical contributions can best be summarized by the following two rigorous hierarchies of normalizing flow architectures.

The first hierarchy considers the *universality* of different constructions, that is, their ability to approximate arbitrary probability distributions of interest:

1. While *volume-preserving normalizing flows* may have useful properties for practical applications, we show in section 4.3.3 that they are not universal, meaning that they are fundamentally limited in what distributions they can approximate. We also provide a fix that consists of a single, non-volume-preserving layer that can be concatenated after training.

2. *Affine coupling normalizing flows* are universal. Unlike previous work, our construction in section 5.4.4 does not contain ill-conditioned building blocks. Note that this universality is only measured in a premetric derived from the loss used in practice of training normalizing flows.

3. *More expressive coupling flows* inherit their universality from affine coupling flows. In section 5.4.5 we identify a loss term that they can leverage within a single block, which enhances their expressivity compared to affine couplings.

This has several practical implications: It underpins the practical insight that affine coupling flows are a universal generative model. It raises a warning for practical applications relying on volume-preserving flows such as temperature steerable flows (Dibak et al., 2022), in that they are fundamentally limited in what distributions they can learn. It equips the search for more expressive coupling flows with a novel metric, the loss improvement by a single layer.

While the above guarantees are an important step towards guaranteeing that normalizing flows can accurately capture the uncertainty in a prediction, they leave several important questions open for future work: First, showing that the target distribution can in principle be represented is not sufficient for successful application. It requires to additionally understand the practical training of networks using gradient descent and a finite set of samples. This includes successful generalization and an understanding of adversarial samples. We think that our constructive proofs can serve as a basis for a more general guarantee in the practical setting of learning from a finite training set. Second, the derived guarantees do not directly consider the practical loss, the Kullback-Leibler divergence, but a derived convergence metric. On the positive side, our experiments suggest that this practical loss vanishes with our construction, unlike that in previous work. Finally, the above guarantees do not make a precise statement on the number of layers required for a certain model quality. See section 5.5 for additional details.

Our second hierarchy of architectures addresses the latter two open questions. It concerns the resources different architectures require in terms of the loss used in practice, depending on the problem dimension:

1. A *Gaussianization blocks* treats dimensions independent of one another. It requires a number of blocks that is at least linear in the number of dimensions: $\Omega(D)$ required blocks, compare section 6.2.3.

2. By modeling a part of dependencies with a neural network between dimensions in each layer, at most a constant number of *coupling blocks* is sufficient regardless of dimension: $\mathcal{O}(1)$ required blocks, compare section 6.3.2.

3. Finally, the Knothe-Rosenblatt rearrangement realized as an *autoregressive model* (Rosenblatt, 1952; Knothe, 1957) demonstrates that if dependencies between all dimensions can be modeled in a single block, then only that single block is needed. This comes at the disadvantage that either training or sampling has to be performed dimension by dimension, slowing down the computation.

Combined with their practical efficiency, this shows that coupling blocks lie in a sweet spot of being expressive yet cheap. These results confirm empirical observations, e.g. found for images by Gaussianization (Dai & Seljak, 2021) and coupling flows (Kingma & Dhariwal, 2018). Our proofs rely on the assumption that the data is Gaussian and the limit model is already close to convergence. This is a weakness since real-world data is usually far off from a Gaussian, in particular if it should be modeled with a generative model. However, we think that our statement still describes a practical scenario, namely when the model is close to convergence, and leave closing this gap open to future work. To facilitate this extension, we also show that the remaining task of learning any distribution can be isolated from learning the closest Gaussian distribution.

In the standard setting of full-dimensional probability distributions under Euclidean geometry, we identify no theoretical shortcomings of coupling-based normalizing flows. However, as argued in chapter 1, some problems require incorporating prior knowledge such as inductive biases, symmetries and special geometry. Previously, building invertible neural networks for these cases was time-consuming at best and impossible at worst. Chapter 7 achieves a breakthrough in this challenge via the introduction of free-form flows. They allow using unconstrained, not necessarily analytically invertible neural networks as the transform between latent and data space in a normalizing flow. This allows easily incorporating architectural improvements achieved in the more general research community into normalizing flows. Training is possible by an efficient way to optimize the change in volume from data to latent space, an essential part of the normalizing flow loss.

Overall, free-form flows perform better or comparable to existing normalizing flows. In comparison to other generative models built on dimension-preserving free-form architectures, free-form flows have the advantage that they greatly reduce the time to produce a single sample: While competitors such as diffusion models (Sohl-Dickstein et al., 2015) require calling the learned neural network on the order of 100 to 1000 times to get a sample, free-form flows require only a single network evaluation at competitive quality. This is achieved using neural networks of comparable size, so that the speedup is reflected in greatly reduced wall clock times.

We confirm the power and flexibility of the framework in experiments in solving inverse problems, finding comparable performance to previous state of the art normalizing flows. In addition, free-form flows greatly broaden the spectrum of applications normalizing flows can be used for, which we explore in three orthogonal directions:

First, we consider the generation of molecules in a Euclidean space using a neural network that is equivariant to rotations and translations. This had not been achieved using invertible neural networks before. The resulting distribution is invariant to these transformations, an important prior knowledge helping the model generalize.

Second, if the data lies on a known Riemannian manifold, we derive that the free-form flow framework is readily applicable by computing the change in volume only in the tangent space of the manifold. This yields a simple recipe to learn a normalizing flow on arbitrary Riemannian manifolds. We demonstrate successful application to standard datasets on spheres, tori and rotation matrices. We find our model to outperform normalizing flows specialized to the respective manifolds, and observe competitive quality to multistep models while reducing the wall clock times for sampling by two to

three orders of magnitude.

Finally, many real-world data such as images is generally viewed to be supported on a manifold, but no analytic expression for that manifold is known. In this case, we propose free-form injective flows that jointly learn a low-dimensional manifold to project the data to, as well as the distribution of the data on that manifold. We contribute the identification of a pathology that can occur in this joint training, in addition to pathologies already described in previous work (Brehmer & Cranmer, 2020). Again, we find our model to outperform previous injective flows and comparable performance to alternative generative autoencoders on a benchmark.

Together, this makes free-form flows a strong generative model that is readily applicable to novel problems. The gradient estimator for maximum likelihood training was easy to adapt in the considered cases, and we expect that this will transfer to other modalities.

We leave several important research directions open to future work: On the theoretical side, since encoder and decoder are only regularized to be inverse of one another, the training dynamics are currently not understood and what minimizer is achieved in practice. On the practical side, computing the value of learned density requires computing the exact Jacobian matrix, whose costs scale linearly with dimension. Easy fixes could be estimating the volume change (Chen et al., 2019) or training a surrogate model. In addition, the experiments have only scratched the surface of what applications are possible with free-form flows. Given the plethora of available neural network architectures, we expect to see applications in various settings.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

Adler, T. J., Ardizzone, L., Vemuri, A., Ayala, L., Gröhl, J., Kirchner, T., Wirkert, S., Kruse, J., Rother, C., Köthe, U., and Maier-Hein, L. Uncertainty-aware performance assessment of optical imaging modalities with invertible neural networks. *International Journal of Computer Assisted Radiology and Surgery*, 14(6):997–1007, 2019. ISSN 1861-6410, 1861-6429. doi: 10.1007/s11548-019-01939-9.

Albergo, M. S. and Vanden-Eijnden, E. Building normalizing flows with stochastic interpolants. In *International Conference on Learning Representations*, 2023.

Amari, S.-i. and Nagaoka, H. *Methods of Information Geometry*. American Mathematical Society, 2007. ISBN 978-0-8218-4302-4 978-1-4704-4605-5. doi: 10.1090/mmono/191.

Ardizzone, L., Bungert, T., Draxler, F., Köthe, U., Kruse, J., Schmier, R., and Sorrenson, P. Framework for Easily Invertible Architectures (FrEIA), 2018a.

Ardizzone, L., Kruse, J., Rother, C., and Köthe, U. Analyzing Inverse Problems with Invertible Neural Networks. In *International Conference on Learning Representations*, 2018b.

Ardizzone, L., Kruse, J., Lüth, C., Bracher, N., Rother, C., and Köthe, U. Conditional Invertible Neural Networks for Diverse Image-to-Image Translation. In *German Conference on Pattern Recognition*, 2020a. doi: 10.1007/978-3-030-71278-5_27.

Ardizzone, L., Mackowiak, R., Rother, C., and Köthe, U. Training Normalizing Flows with the Information Bottleneck for Competitive Generative Classification. In *Advances in Neural Information Processing Systems*, 2020b.

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. In *International Conference on Machine Learning*, 2019.

Beitler, J. J., Sosnovik, I., and Smeulders, A. PIE: Pseudo-invertible encoder. *arXiv:2111.00619*, 2021.

Ben-Hamu, H., Cohen, S., Bose, J., Amos, B., Nickel, M., Grover, A., Chen, R. T., and Lipman, Y. Matching normalizing flows and probability paths on manifolds. In *International Conference on Machine Learning*, 2022.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Bickel, P. J., Kur, G., and Nadler, B. Projection Pursuit in High Dimensions. *PNAS*, 115(37):9151–9156, 2018.

Bieringer, S., Butter, A., Heimel, T., Höche, S., Köthe, U., Plehn, T., and Radev, S. T. Measuring QCD splittings with invertible networks. *SciPost Physics*, 10(6):126, 2021.

Bigoni, D., Zahm, O., Spantini, A., and Marzouk, Y. Greedy inference with layers of lazy maps. *arXiv preprint arXiv:1906.00031*, 2019.

Bishop, C. M. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, 2006. ISBN 978-0-387-31073-2.

Böhm, V. and Seljak, U. Probabilistic auto-encoder. *Transactions on Machine Learning Research*, 2022.

Bose, J., Smofsky, A., Liao, R., Panangaden, P., and Hamilton, W. Latent variable modelling with hyperbolic normalizing flows. In *International Conference on Machine Learning*, 2020.

Boyda, D., Kanwar, G., Racanière, S., Rezende, D. J., Albergo, M. S., Cranmer, K., Hackett, D. C., and Shanahan, P. E. Sampling using SU(N) gauge equivariant flows. *Physical Review D: Particles and Fields*, 103(7):074504, April 2021. doi: 10.1103/PhysRevD.103.074504.

Brakenridge, G. Global active archive of large flood events, 2017.

Brehmer, J. and Cranmer, K. Flows for simultaneous manifold learning and density estimation. *Advances in Neural Information Processing Systems*, 33:442–453, 2020.

Brofos, J. A., Brubaker, M. A., and Lederman, R. R. Manifold density estimation via generalized dequantization. *arXiv preprint arXiv:2102.07143*, 2021.

Bryc, W. *The Normal Distribution*, volume 100 of *Lecture Notes in Statistics*. Springer New York, New York, NY, 1995. ISBN 978-0-387-97990-8 978-1-4612-2560-7. doi: 10.1007/978-1-4612-2560-7.

Burda, Y., Grosse, R., and Salakhutdinov, R. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Cambanis, S., Huang, S., and Simons, G. On the theory of elliptically contoured distributions. *Journal of Multivariate Analysis*, 11(3):368–385, September 1981. ISSN 0047259X. doi: 10.1016/0047-259X(81)90082-8.

Cardoso, J.-F. Dependence, Correlation and Gaussianity in Independent Component Analysis. *Journal of Machine Learning Research*, 4:1177–1203, 2003. ISSN 1532-4435.

Cartwright, D. I. and Field, M. J. A refinement of the arithmetic mean-geometric mean inequality. *Proceedings of the American Mathematical Society*, 71(1):36–38, 1978. ISSN 0002-9939, 1088-6826. doi: 10.1090/S0002-9939-1978-0476971-2.

Caterini, A. L., Loaiza-Ganem, G., Pleiss, G., and Cunningham, J. P. Rectangular flows for manifold learning. *Advances in Neural Information Processing Systems*, 2021.

Chadebec, C., Vincent, L. J., and Allassonnière, S. Pythae: Unifying generative autoencoders in python – a benchmarking use case. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, 2022.

Chen, R. T. and Lipman, Y. Flow Matching on General Geometries. In *International Conference on Learning Representations*, 2024.

Chen, R. T., Li, X., Grosse, R. B., and Duvenaud, D. K. Isolating sources of disentanglement in variational autoencoders. *Advances in Neural Information Processing Systems*, 31, 2018a.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018b.

Chen, R. T., Behrmann, J., Duvenaud, D. K., and Jacobsen, J.-H. Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*, 2019.

Chen, S. and Gopinath, R. Gaussianization. In Leen, T., Dietterich, T., and Tresp, V. (eds.), *Advances in Neural Information Processing Systems*, 2000.

Cohen, G., Afshar, S., Tapson, J., and Van Schaik, A. EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Comon, P. Independent component analysis, A new concept? *Signal Processing*, 36(3):287–314, April 1994. ISSN 01651684. doi: 10.1016/0165-1684(94)90029-9.

Costarelli, D. and Spigler, R. How sharp is the Jensen inequality? *Journal of Inequalities and Applications*, 2015(1):69, December 2015. ISSN 1029-242X. doi: 10.1186/s13660-015-0591-x.

Cramer, E., Rauh, F., Mitsos, A., Tempone, R., and Dahmen, M. Isometric manifold learning for injective normalizing flows. *arXiv preprint arXiv:2203.03934*, 2022.

Cramér, H. and Wold, H. Some Theorems on Distribution Functions. *Journal of the London Mathematical Society*, s1-11(4):290–294, October 1936. ISSN 00246107. doi: 10.1112/jlms/s1-11.4.290.

Cunningham, E., Zabounidis, R., Agrawal, A., Fiterau, M., and Sheldon, D. Normalizing flows across dimensions. *International Conference on Machine Learning, Workshop Track*, 2020.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, December 1989. ISSN 0932-4194, 1435-568X. doi: 10.1007/BF02551274.

Dai, B. and Seljak, U. Sliced iterative normalizing flows. In Meila, M. and Zhang, T. (eds.), *International Conference on Machine Learning*, 2021.

dAscoli, S., Sagun, L., Biroli, G., and Bruna, J. Finding the Needle in the Haystack with Convolutions: On the benefits of architectural bias. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Davidson, T. R., Falorsi, L., De Cao, N., Kipf, T., and Tomczak, J. M. Hyperspherical variational auto-encoders. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

De Bortoli, V., Mathieu, E., Hutchinson, M., Thornton, J., Teh, Y. W., and Doucet, A. Riemannian score-based generative modelling. *Advances in Neural Information Processing Systems*, 35:2406–2422, 2022.

Dibak, M., Klein, L., Krämer, A., and Noé, F. Temperature steerable flows and Boltzmann generators. *Phys. Rev. Res.*, 4(4):L042005, October 2022. doi: 10.1103/PhysRevResearch.4.L042005.

Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear Independent Components Estimation. In *International Conference on Learning Representations, Workshop Track*, 2015.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. In *International Conference on Learning Representations*, 2017.

Draxler, F., Schwarz, J., Schnörr, C., and Köthe, U. Characterizing the Role of a Single Coupling Layer in Affine Normalizing Flows. In *German Conference on Pattern Recognition*, 2020.

Draxler, F., Schnörr, C., and Köthe, U. Whitening Convergence Rate of Coupling-based Normalizing Flows. In *Advances in Neural Information Processing Systems*, 2022.

Draxler, F., Kühmichel, L., Rousselot, A., Müller, J., Schnoerr, C., and Koethe, U. On the Convergence Rate of Gaussianization with Random Rotations. In *International Conference on Machine Learning*, 2023.

Draxler, F., Sorrenson, P., Zimmermann, L., Rousselot, A., and Köthe, U. Free-form Flows: Make Any Architecture a Normalizing Flow. In *Artificial Intelligence and Statistics*, 2024a.

Draxler, F., Wahl, S., Schnörr, C., and Köthe, U. On the Universality of Coupling-based Normalizing Flows. *arXiv:2402.06578*, 2024b.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural Spline Flows. In *Advances in Neural Information Processing Systems*, 2019.

Eaton, M. L. A characterization of spherical distributions. *Journal of Multivariate Analysis*, 20(2): 272–276, December 1986. ISSN 0047259X. doi: 10.1016/0047-259X(86)90083-7.

EOSDIS. Land, atmosphere near real-time capability for EOS (LANCE) system operated by NASA's earth science data and information system (ESDIS), 2020.

Falcon, W. and The PyTorch Lightning team. PyTorch lightning, March 2019.

Falorsi, L. Continuous normalizing flows on manifolds. *arXiv preprint arXiv:2104.14959*, 2021.

Falorsi, L. and Forré, P. Neural ordinary differential equations on manifolds. *arXiv preprint arXiv:2006.06663*, 2020.

Falorsi, L., de Haan, P., Davidson, T. R., and Forré, P. Reparameterizing distributions on lie groups. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

Faraut, J. Rayleigh theorem, projection of orbital measures and spline functions. *Advances in Pure and Applied Mathematics*, 6(4):261–283, 2015. ISSN 1867-1152, 1869-6090. doi: 10.1515/apam-2015-5012.

Feistel, H. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, 1973. ISSN 00368733, 19467087.

Fleishman, A. I. A method for simulating non-normal distributions. *Psychometrika*, 43(4):521–532, 1978.

Friedman, J. H. On multivariate goodness–of–fit and two–sample testing. *Statistical Problems in Particle Physics, Astrophysics, and Cosmology*, 1:311, 2003.

Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980. ISSN 0340-1200, 1432-0770. doi: 10.1007/BF00344251.

Gemici, M. C., Rezende, D., and Mohamed, S. Normalizing flows on riemannian manifolds. *arXiv:1611.02304*, 2016.

Ghose, A., Rashwan, A., and Poupart, P. Batch norm with entropic regularization turns deterministic autoencoders into generative models. In *Conference on Uncertainty in Artificial Intelligence*, 2020.

Ghosh, P., Sajjadi, M. S., Vergari, A., Black, M., and Schölkopf, B. From variational to deterministic autoencoders. *International Conference on Learning Representations*, 2020.

Gibbs, A. L. and Su, F. E. On Choosing and Bounding Probability Metrics. *International Statistical Review / Revue Internationale de Statistique*, 70(3):419–435, 2002. ISSN 03067734, 17515823.

Girard, A. A fast 'Monte-Carlo cross-validation' procedure for large least squares problems with noisy data. *Numerische Mathematik*, 56:1–23, 1989.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Golub, G. H. and Pereyra, V. The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. *SIAM Journal on numerical analysis*, 10(2):413–432, 1973.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Gorin, T. Integrals of monomials over the orthogonal group. *Journal of Mathematical Physics*, 43(6): 3342–3351, 2002. ISSN 0022-2488, 1089-7658. doi: 10.1063/1.1471367.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*, 2019.

Gresele, L., Fissore, G., Javaloy, A., Schölkopf, B., and Hyvarinen, A. Relative gradient optimization of the jacobian term in unsupervised deep learning. In *Advances in Neural Information Processing Systems*, 2020.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

Heinecke, A., Ho, J., and Hwang, W.-L. Refinement and universal approximation via sparsely connected ReLU convolution nets. *IEEE Signal Processing Letters*, 27:1175–1179, 2020.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in Neural Information Processing Systems*, 30, 2017.

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

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design. In *International Conference on Machine Learning*, 2019.

Hoogeboom, E., Nielsen, D., Jaini, P., Forré, P., and Welling, M. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 2021.

Hoogeboom, E., Satorras, V. G., Vignac, C., and Welling, M. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, 2022.

Horn, R. A. and Johnson, C. R. *Matrix Analysis*. Cambridge University Press, 2012.

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2): 251–257, 1991. ISSN 08936080. doi: 10.1016/0893-6080(91)90009-T.

Horvat, C. and Pfister, J.-P. Denoising normalizing flow. *Advances in Neural Information Processing Systems*, 34:9099–9111, 2021.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural Autoregressive Flows. In *International Conference on Machine Learning*, 2018.

Huang, C.-W., Dinh, L., and Courville, A. Augmented Normalizing Flows: Bridging the Gap Between Generative Flows and Latent Variable Models. In *International Conference on Learning Representations, Workshop Track*, 2020.

Huang, C.-W., Aghajohari, M., Bose, J., Panangaden, P., and Courville, A. C. Riemannian diffusion models. *Advances in Neural Information Processing Systems*, 35:2750–2761, 2022.

Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007.

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

Hyvärinen, A., Karhunen, J., and Oja, E. *Independent Component Analysis*. John Wiley & Sons, Ltd, 2001. ISBN 978-0-471-22131-9.

Ishikawa, I., Teshima, T., Tojo, K., Oono, K., Ikeda, M., and Sugiyama, M. Universal approximation property of invertible neural networks. *arXiv preprint arXiv:2204.07415*, 2022.

Jaini, P., Selby, K. A., and Yu, Y. Sum-of-Squares Polynomial Flow. In *International Conference on Machine Learning*, 2019.

Johansson, F. et al. *Mpmath: A Python Library for Arbitrary-Precision Floating-Point Arithmetic (Version 0.14)*, February 2010.

Jost, J. *Riemannian Geometry and Geometric Analysis*, volume 42005. Springer, 2008.

Kalatzis, D., Ye, J. Z., Pouplin, A., Wohlert, J., and Hauberg, S. Density estimation on smooth manifolds with normalizing flows. *arXiv preprint arXiv:2106.03500*, 2021.

Kanwar, G., Albergo, M. S., Boyda, D., Cranmer, K., Hackett, D. C., bastien Racanière, S., Rezende, D. J., and Shanahan, P. E. Equivariant flow-based sampling for lattice gauge theory. *Physical Review Letters*, 125(12), September 2020. doi: 10.1103/physrevlett.125.121601.

Keller, T. A., Peters, J. W., Jaini, P., Hoogeboom, E., Forré, P., and Welling, M. Self normalizing flows. In *International Conference on Machine Learning*, 2021.

Kim, H. and Mnih, A. Disentangling by factorising. In *International Conference on Machine Learning*, 2018.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization, January 2017.

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

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, 29, 2016.

Klein, L., Krämer, A., and Noé, F. Equivariant Flow Matching. In *Advances in Neural Information Processing Systems*, 2023.

Knothe, H. Contributions to the theory of convex bodies. *Michigan Mathematical Journal*, 4(1):39–52, 1957.

Knuth, D. E. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1997.

Kobyzev, I., Prince, S. J., and Brubaker, M. A. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11): 3964–3979, 2021.

Koehler, F., Mehta, V., and Risteski, A. Representational aspects of depth and conditioning in normalizing flows. In *International Conference on Machine Learning*, 2021.

Köhler, J., Klein, L., and Noé, F. Equivariant flows: Exact likelihood generative learning for symmetric densities. In *International Conference on Machine Learning*, 2020.

Kothari, K., Khorashadizadeh, A., de Hoop, M. V., and Dokmanic, I. Trumpets: Injective flows for inference and inverse problems. In *Conference on Uncertainty in Artificial Intelligence*, 2021.

Köthe, U. A review of change of variable formulas for generative modeling. *arXiv preprint arXiv:2308.02652*, 2023.

Krantz, S. G. and Parks, H. R. *Geometric Integration Theory*. Springer Science & Business Media, 2008.

Kraskov, A., Stögbauer, H., and Grassberger, P. Estimating mutual information. *Physical Review E*, 69(6):066138, June 2004. ISSN 1539-3755, 1550-2376. doi: 10.1103/PhysRevE.69.066138.

Kratsios, A. and Bilokopytov, I. Non-euclidean universal approximation. *Advances in Neural Information Processing Systems*, 33:10635–10646, 2020.

Krishnaiah, P. Some recent developments on complex multivariate distributions. *Journal of Multivariate Analysis*, 6(1):1–30, 1976. ISSN 0047259X. doi: 10.1016/0047-259X(76)90017-8.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Kruse, J., Detommaso, G., Köthe, U., and Scheichl, R. HINT: Hierarchical Invertible Neural Transport for Density Estimation and Bayesian Inference. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):8191–8199, May 2021. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v35i9.16997.

Kühmichel, L. E. and Draxler, F. Lightning trainable, 2023.

Kumar, A., Poole, B., and Murphy, K. Regularized autoencoders via relaxed injective probability flow. In *International Conference on Artificial Intelligence and Statistics*, 2020.

Laparra, V., Camps-Valls, G., and Malo, J. Iterative Gaussianization: From ICA to Random Rotations. *IEEE Transactions on Neural Networks*, 22(4):537–549, April 2011. ISSN 1045-9227, 1941-0093. doi: 10.1109/TNN.2011.2106511.

Lawrence, J., Bernal, J., and Witzgall, C. A purely algebraic justification of the Kabsch-Umeyama algorithm. *Journal of research of the National Institute of Standards and Technology*, 124:1, 2019.

LeCun, Y., Cortes, C., and Burges, CJ. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Lee, H., Pabbaraju, C., Sevekari, A. P., and Risteski, A. Universal approximation using well-conditioned normalizing flows. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 12700–12711. Curran Associates, Inc., 2021.

Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., and Le, M. Flow Matching for Generative Modeling. In *International Conference on Learning Representations*, 2023.

Liu, X., Gong, C., and Liu, Q. Learning to Generate and Transfer Data with Rectified Flow. In *International Conference on Learning Representations*, 2023a.

Liu, Y., Liu, H., Yin, Y., Wang, Y., Chen, B., and Wang, H. Delving into discrete normalizing flows on SO (3) manifold for probabilistic rotation modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21264–21273, 2023b.

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

Liutkus, A., Simsekli, U., Majewski, S., Durmus, A., and Stöter, F.-R. Sliced-Wasserstein flows: Nonparametric generative modeling via optimal transport and diffusions. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Loaiza-Ganem, G., Ross, B. L., Wu, L., Cunningham, J. P., Cresswell, J. C., and Caterini, A. L. Denoising deep generative models. In *Advances in Neural Information Processing Systems, Workshop Track*, 2022.

Lopez-Paz, D. and Oquab, M. Revisiting classifier two-sample tests. In *International Conference on Learning Representations*, 2017.

Lou, A., Lim, D., Katsman, I., Huang, L., Jiang, Q., Lim, S. N., and De Sa, C. M. Neural manifold ordinary differential equations. *Advances in Neural Information Processing Systems*, 33:17548–17558, 2020.

Lovell, S. C., Davis, I. W., Arendall III, W. B., de Bakker, P. I. W., Word, J. M., Prisant, M. G., Richardson, J. S., and Richardson, D. C. Structure validation by cα geometry: φ,ψ and cβ deviation. *Proteins: Structure, Function, and Bioinformatics*, 50(3):437–450, 2003. doi: 10.1002/prot.10286.

Lueckmann, J.-M., Boelts, J., Greenberg, D., Goncalves, P., and Macke, J. Benchmarking simulation-based inference. In *International Conference on Artificial Intelligence and Statistics*, 2021.

Lyu, J., Chen, Z., Feng, C., Cun, W., Zhu, S., Geng, Y., Xu, Z., and Chen, Y. Universality of parametric Coupling Flows over parametric diffeomorphisms. *arXiv preprint arXiv:2202.02906*, 2022.

Mackowiak, R., Ardizzone, L., Kothe, U., and Rother, C. Generative classifiers as a basis for trustworthy image classification. In *IEEE / CVF Computer Vision and Pattern Recognition Conference*, 2021.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. Adversarial autoencoders. In *International Conference on Learning Representations*, 2015.

Mathieu, E. and Nickel, M. Riemannian continuous normalizing flows. *Advances in Neural Information Processing Systems*, 33:2503–2515, 2020.

McKinney, W. Data Structures for Statistical Computing in Python. In van der Walt, S. and Jarrod Millman (eds.), *9th Python in Science Conference*, 2010.

Meng, C., Ke, Y., Zhang, J., Zhang, M., Zhong, W., and Ma, P. Large-scale optimal transport map estimation using projection pursuit. In Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8116–8127. Curran Associates, Inc., 2019.

Meng, C., Song, Y., Song, J., and Ermon, S. Gaussianization Flows. In *AISTATS*, 2020.

Mezzadri, F. How to generate random matrices from the classical compact groups. *Notices of the American Mathematical Society*, 54(5):592–604, May 2007. ISSN 0002-9920.

Miolane, N., Guigui, N., Brigant, A. L., Mathe, J., Hou, B., Thanwerdas, Y., Heyder, S., Peltre, O., Koep, N., Zaatiti, H., Hajri, H., Cabanes, Y., Gerald, T., Chauchat, P., Shewmake, C., Brooks, D., Kainz, B., Donnat, C., Holmes, S., and Pennec, X. Geomstats: A python package for riemannian geometry in machine learning. *Journal of Machine Learning Research*, 21(223):1–9, 2020.

Miolane, N., Utpala, S., Guigui, N., Pereira, L. F., Brigant, A. L., Hzaatiti, Cabanes, Y., Mathe, J., Koep, N., elodiemaignant, ythanwerdas, xpennec, tgeral68, Christian, Nguyen, T. M., Peltre, O., pchauchat, Jules-Deschamps, Harvey, J., mortenapedersen, Maya95assal, Barthélemy, Q., Abdellaoui-Souhail, Myers, A., Ambellan, F., Florent-Michel, Talbar, S., Heyder, S., de Mont-Marin, Y., and Marius. Geomstats/geomstats: Geomstats v2.7.0. Zenodo, August 2023.

Müller, J., Schmier, R., Ardizzone, L., Rother, C., and Köthe, U. Learning Robust Models Using The Principle of Independent Causal Mechanisms. In *German Conference on Pattern Recognition*, 2021.

Müller, T., Mcwilliams, B., Rousselle, F., Gross, M., and Novák, J. Neural Importance Sampling. *ACM Transactions on Graphics*, 38(5):1–19, 2019. ISSN 0730-0301. doi: 10.1145/3341156.

Murphy, K. A., Esteves, C., Jampani, V., Ramalingam, S., and Makadia, A. Implicit-PDF: Non-parametric representation of probability distributions on the rotation manifold. In *International Conference on Machine Learning*, 2021.

Murray, L., Arendall, W., Richardson, D., and Richardson, J. RNA backbone is rotameric. *Proceedings of the National Academy of Sciences of the United States of America*, 100:13904–9, December 2003. doi: 10.1073/pnas.1835769100.

Nash, J. The imbedding problem for Riemannian manifolds. *Annals of mathematics*, 63(1):20–63, 1956.

National Geophysical Data Center / World Data Service (NGDC/WDS). NCEI/WDS global significant earthquake database, 2022a.

National Geophysical Data Center / World Data Service (NGDC/WDS). NCEI/WDS global significant volcanic eruptions database, 2022b.

Nicoli, K. A., Anders, C., Funcke, L., Hartung, T., Jansen, K., Kessel, P., Nakajima, S., and Stornati, P. Machine Learning of Thermodynamic Observables in the Presence of Mode Collapse. In *Proceedings of The 38th International Symposium on Lattice Field Theory — PoS(LATTICE2021)*, pp. 338, Zoom/Gather@Massachusetts Institute of Technology, May 2022. Sissa Medialab. doi: 10.22323/1.396.0338.

Noé, F., Olsson, S., Köhler, J., and Wu, H. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019.

Noever-Castelos, P., Ardizzone, L., and Balzani, C. Model updating of wind turbine blade cross sections with invertible neural networks. *Wind Energy*, 25(3):573–599, 2022. ISSN 1095-4244, 1099-1824. doi: 10.1002/we.2687.

Olshanski, G. Projections of orbital measures, Gelfand-Tsetlin polytopes, and splines. *Journal of Lie Theory*, 23(4):1011–1022, 2013.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. *Advances in Neural Information Processing Systems*, 2017.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Pearson, K. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, November 1901. ISSN 1941-5982, 1941-5990. doi: 10.1080/14786440109462720.

Peel, D., Whiten, W. J., and McLachlan, G. J. Fitting mixtures of Kent distributions to aid in joint set identification. *Journal of the American Statistical Association*, 96(453):56–63, 2001.

Perugachi-Diaz, Y., Tomczak, J. M., and Bhulai, S. Invertible DenseNets with concatenated LipSwish. In *Advances in Neural Information Processing Systems*, 2021.

Pitié, F., Kokaram, A. C., and Dahyot, R. Automated colour grading using colour distribution transfer. *Computer Vision and Image Understanding*, 107(1-2):123–137, July 2007. ISSN 10773142. doi: 10.1016/j.cviu.2006.11.011.

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, 14(5):503–519, October 2017. ISSN 1476-8186, 1751-8520. doi: 10.1007/s11633-017-1054-2.

Putzky, P. and Welling, M. Invert to learn to invert. In *Advances in Neural Information Processing Systems*, pp. 446–456, 2019.

Radev, S. T., Graw, F., Chen, S., Mutters, N. T., Eichel, V. M., Bärnighausen, T., and Köthe, U. OutbreakFlow: Model-based Bayesian inference of disease outbreak dynamics with invertible neural networks and its application to the COVID-19 pandemics in Germany. *PLoS computational biology*, 17(10):e1009472, 2021.

Radev, S. T., Mertens, U. K., Voss, A., Ardizzone, L., and Köthe, U. BayesFlow: Learning Complex Stochastic Models With Invertible Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1452–1466, April 2022. ISSN 2162-2388. doi: 10.1109/TNNLS.2020.3042395.

Ramachandran, G. N., Ramakrishnan, C., and Sasisekharan, V. Stereochemistry of polypeptide chain configurations. *Journal of Molecular Biology*, 7(1):95–99, 1963. doi: 10.1016/S0022-2836(63)80023-6.

Ramakrishnan, R., Dral, P. O., Rupp, M., and von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1:140022, 2014.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International Conference on Machine Learning*, 2015.

Rezende, D. J., Papamakarios, G., Racaniere, S., Albergo, M., Kanwar, G., Shanahan, P., and Cranmer, K. Normalizing flows on tori and spheres. In *International Conference on Machine Learning*, 2020.

Rosenblatt, M. Remarks on a multivariate transformation. *The annals of mathematical statistics*, 23 (3):470–472, 1952.

Ross, B. and Cresswell, J. Tractable density estimation on learned manifolds with conformal embedding flows. *Advances in Neural Information Processing Systems*, 34:26635–26648, 2021.

Rozen, N., Grover, A., Nickel, M., and Lipman, Y. Moser flow: Divergence-based generative modeling on manifolds. *Advances in Neural Information Processing Systems*, 34:17669–17680, 2021.

Ruddigkeit, L., van Deursen, R., Blum, L. C., and Reymond, J.-L. Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. *Journal of Chemical Information and Modeling*, 52:2864–2875, 2012.

Rudin, W. *Principles of Mathematical Analysis*. International Series in Pure and Applied Mathematics. McGraw-Hill, New York St. Louis San Francisco [etc.], third ed edition, 1976. ISBN 978-0-07-054235-8.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. ISSN 0028-0836, 1476-4687. doi: 10.1038/323533a0.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

Satorras, V. G., Hoogeboom, E., Fuchs, F., Posner, I., and Welling, M. E (n) equivariant normalizing flows. *Advances in Neural Information Processing Systems*, 2021a.

Satorras, V. G., Hoogeboom, E., and Welling, M. E(n) equivariant graph neural networks. In *International Conference on Machine Learning*, 2021b.

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., and Hassabis, D. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, January 2020. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-019-1923-7.

Shewchuk, J. R. et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

Silvestri, G., Roos, D., and Ambrogioni, L. Deterministic training of generative autoencoders using invertible layers. In *The Eleventh International Conference on Learning Representations*, 2023.

Snell, J., Ridgeway, K., Liao, R., Roads, B. D., Mozer, M. C., and Zemel, R. S. Learning to generate images with perceptual similarity metrics. In *IEEE International Conference on Image Processing*, pp. 4277–4281. IEEE, 2017.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2015.

Sorrenson, P., Rother, C., and Köthe, U. Disentanglement by nonlinear ICA with general incompressible-flow networks (GIN). In *International Conference on Learning Representations*, 2019.

Sorrenson, P., Draxler, F., Rousselot, A., Hummerich, S., and Köthe, U. Learning distributions on manifolds with free-form flows. *arXiv preprint arXiv:2312.09852*, 2023.

Sorrenson, P., Draxler, F., Rousselot, A., Hummerich, S., Zimmermann, L., and Köthe, U. Lifting architectural constraints of injective flows. In *International Conference on Learning Representations*, 2024.

Souveton, V., Guillin, A., Jasche, J., Lavaux, G., and Michel, M. Fixed-kinetic Neural Hamiltonian Flows for enhanced interpretability and reduced complexity. In *International Conference on Artificial Intelligence and Statistics*, 2024.

Tabak, E. G. and Trigila, G. Conditional expectation estimation through attributable components. *Information and Inference: A Journal of the IMA*, 7(4):727–754, 2018.

Teng, Y. and Choromanska, A. Invertible autoencoder for domain adaptation. *Computation*, 7(2):20, 2019.

Teshima, T., Ishikawa, I., Tojo, K., Oono, K., Ikeda, M., and Sugiyama, M. Coupling-based Invertible Neural Networks Are Universal Diffeomorphism Approximators. In *Advances in Neural Information Processing Systems*, 2020a.

Teshima, T., Tojo, K., Ikeda, M., Ishikawa, I., and Oono, K. Universal Approximation Property of Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems, Workshop Track*, 2020b.

The pandas development team. Pandas-dev/pandas: Pandas, February 2020.

Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. Wasserstein auto-encoders. In *International Conference on Learning Representations*, 2018.

Toth, P., Rezende, D. J., Jaegle, A., Racanière, S., Botev, A., and Higgins, I. Hamiltonian generative networks. In *International Conference on Learning Representations*, 2020.

Trigila, G. and Tabak, E. G. Data-driven optimal transport. *Communications on Pure and Applied Mathematics*, 69(4):613–648, 2016.

Turk, G. and Levoy, M. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pp. 311–318, 1994.

Uria, B., Murray, I., and Larochelle, H. RNADE: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, volume 26, 2013.

van den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in Neural Information Processing Systems*, 30, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wainwright, M. *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge University Press, 2019.

Wehenkel, A. and Louppe, G. Unconstrained Monotonic Neural Networks. In *Advances in Neural Information Processing Systems*, 2019.

Wildberger, J., Dax, M., Buchholz, S., Green, S. R., Macke, J. H., and Schölkopf, B. Flow Matching for Scalable Simulation-Based Inference. In *Advances in Neural Information Processing Systems*, 2023.

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. *Dive into Deep Learning*. Cambridge University Press, 2023a.

Zhang, H., Gao, X., Unterman, J., and Arodz, T. Approximation Capabilities of Neural ODEs and Invertible Residual Networks. In *International Conference on Machine Learning*, 2020a.

Zhang, M., Sun, Y., Zhang, C., and Mcdonagh, S. Spread flows for manifold modelling. In *International Conference on Artificial Intelligence and Statistics*, 2023b.

Zhang, Z., Zhang, R., Li, Z., Bengio, Y., and Paull, L. Perceptual generative autoencoders. In *International Conference on Machine Learning*, 2020b.

Zhao, S., Song, J., and Ermon, S. Infovae: Information maximizing variational autoencoders. *arXiv:1706.02262*, 2017.

Zhou, D.-X. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794, 2020.

Ziegler, Z. and Rush, A. Latent Normalizing Flows for Discrete Sequences. In *International Conference on Machine Learning*, 2019.

A. Proofs

A.1. Proofs on Pythagorean identities

This appendix is adapted from Draxler et al. (2022).

A.1.1. Proof of theorem 4.1

The explicit form of the non-Standardness is given by the KL divergence between the two multivariate Gaussians $\mathcal{N}(m, \Sigma)$ and $\mathcal{N}(0, I)$:

$$
\begin{aligned}
&S(m, \Sigma) = \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(m, \Sigma) \| \mathcal{N}(0, I)) && \text{(A.1)}\\
&= \mathbb{E}_{x \sim \mathcal{N}(m,\Sigma)}[\log \mathcal{N}(x; m, \Sigma) - \log \mathcal{N}(x; 0, I)] && \text{(A.2)}\\
&= \mathbb{E}_{x \sim \mathcal{N}(m,\Sigma)}[-\tfrac{1}{2} \log\det(2\pi\Sigma) - \tfrac{1}{2}(x-m)^{\mathrm{T}}\Sigma^{-1}(x-m) + \tfrac{1}{2}\log\det(2\pi I_D) + \tfrac{1}{2}\|x\|^2] && \text{(A.3)}\\
&= \tfrac{1}{2}\Big(-\log\det(\Sigma) + \mathbb{E}_{x \sim \mathcal{N}(m,\Sigma)}[-\tfrac{1}{2}(x-m)^{\mathrm{T}}\Sigma^{-1}(x-m) + \tfrac{1}{2}\|x\|^2]\Big) && \text{(A.4)}\\
&= \tfrac{1}{2}(\|m\|^2 + \operatorname{tr}\Sigma - D - \log\det\Sigma). && \text{(A.5)}
\end{aligned}
$$

Theorem 4.1. *Given data with distribution $p_\theta(z)$ with mean m and covariance Σ. Then, the Kullback-Leibler divergence to a standard normal distribution decomposes as follows:*

$$
\mathcal{L} = \mathcal{D}_{KL}(p_\theta(z)\|\mathcal{N}(0,I)) = \underbrace{\mathcal{D}_{KL}(p_\theta(z)\|\mathcal{N}(m,\Sigma))}_{\text{non-Gaussianity } G[p_\theta(z)]} + \underbrace{\mathcal{D}_{KL}(\mathcal{N}(m,\Sigma)\|\mathcal{N}(0,I))}_{\text{non-Standardness } S(m,\Sigma)}, \tag{A.6}
$$

and the non-Standardness again decomposes:

$$
S(m,\Sigma) = \underbrace{\mathcal{D}_{KL}(\mathcal{N}(m,\Sigma)\|\mathcal{N}(m,\mathrm{Diag}(\Sigma)))}_{\text{Correlation } C(m,\Sigma)} + \underbrace{\mathcal{D}_{KL}(\mathcal{N}(m,\mathrm{Diag}(\Sigma))\|\mathcal{N}(0,I))}_{\text{Diagonal non-Standardness}}. \tag{A.7}
$$

Also,

$$
\mathcal{L} = \mathcal{D}_{KL}(p_\theta(z)\|\mathcal{N}(0,I)) = \underbrace{\mathcal{D}_{KL}(p_\theta(z)\|p(x_1)\cdots p(x_D))}_{\text{Mutual Information } I[p_\theta(z)]} + \sum_i \underbrace{\mathcal{D}_{KL}(p_\theta(z_i)\|\mathcal{N}(0,1))}_{\text{marginal loss } \mathcal{L}[p_\theta(z_i)]}. \tag{A.8}
$$

Proof. We start with the first decomposition in equation (4.35).

$$
\begin{aligned}
&\mathcal{D}_{\mathrm{KL}}(p\|\mathcal{N}(0,I)) - \mathcal{D}_{\mathrm{KL}}(p\|\mathcal{N}(m,\Sigma)) && \text{(A.9)}\\
&= \mathbb{E}_{x\sim p(x)}[\log p(x) - \log\mathcal{N}(x;0,I) - \log p(x) + \log\mathcal{N}(x;m,\Sigma)] && \text{(A.10)}\\
&= \mathbb{E}_{x\sim p(x)}[-\log\mathcal{N}(x;0,I) + \log\mathcal{N}(x;m,\Sigma)] && \text{(A.11)}\\
&= \frac{1}{2}\mathbb{E}_{x\sim p(x)}[D\log(2\pi) + \|x\|^2 - D\log(2\pi) - \log\det\Sigma - (x-m)^{\mathrm{T}}\Sigma^{-1}(x-m)] && \text{(A.12)}\\
&= \frac{1}{2}\mathbb{E}_{x\sim p(x)}[\|x\|^2 - \log\det\Sigma - (x-m)^{\mathrm{T}}\Sigma^{-1}(x-m)] && \text{(A.13)}\\
&= \frac{1}{2}\Big(\mathbb{E}_{x\sim p(x)}[\|x\|^2] - \log\det\Sigma - \mathbb{E}_{x\sim p(x)}[(x-m)^{\mathrm{T}}\Sigma^{-1}(x-m)]\Big). && \text{(A.14)}
\end{aligned}
$$

The open expectation values read:

$$\mathbb{E}_{x\sim p(x)}\Big[\|x\|^2\Big] = \mathbb{E}_{x\sim p(x)}\left[\sum_{i=1}^{D} x_i^2\right] = \sum_{i=1}^{D}\mathbb{E}_{x\sim p(x)}[x_i^2] = \sum_{i=1}^{D}(m_i^2+\Sigma_{ii}) = \|m\|^2 + \operatorname{tr}\Sigma, \tag{A.15}$$

and interpreting $(x-m)$ as a $\mathbb{R}^{D\times 1}$ matrix, we can re-write using the trace. Then use the cyclic property and linearity of the trace:

$$\begin{aligned}
\mathbb{E}_{x\sim p(x)}[(x-m)^{\mathrm{T}}\Sigma^{-1}(x-m)] &= \mathbb{E}_{x\sim p(x)}[\operatorname{tr}((x-m)^{\mathrm{T}}\Sigma^{-1}(x-m))] && \text{(A.16)}\\
&= \mathbb{E}_{x\sim p(x)}[\operatorname{tr}((x-m)(x-m)^{\mathrm{T}}\Sigma^{-1})] && \text{(A.17)}\\
&= \operatorname{tr}(\mathbb{E}_{x\sim p(x)}[(x-m)(x-m)^{\mathrm{T}}]\Sigma^{-1}) && \text{(A.18)}\\
&= \operatorname{tr}(\Sigma\Sigma^{-1}) && \text{(A.19)}\\
&= D. && \text{(A.20)}
\end{aligned}$$

Inserting the two expectation values, we identify:

$$\begin{aligned}
\mathcal{D}_{\mathrm{KL}}(p\|\mathcal{N}(0,I)) - \mathcal{D}_{\mathrm{KL}}(p\|\mathcal{N}(m,\Sigma)) &= \frac{1}{2}\Big(\|m\|^2 + \operatorname{tr}\Sigma - \log\det\Sigma - D\Big) && \text{(A.21)}\\
&= \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(m,\Sigma)\|\mathcal{N}(0,I)), && \text{(A.22)}
\end{aligned}$$

and obtain equation (4.35).

Now we move on to show equation (4.36):

$$\begin{aligned}
C(m,\Sigma) &= \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(m,\Sigma)\|\mathcal{N}(m,\operatorname{Diag}(\Sigma))) && \text{(A.23)}\\
&= \frac{1}{2}\left(\operatorname{tr}\Big((\operatorname{Diag}\Sigma)^{-1}\Sigma\Big) - D + \log\frac{\det(\operatorname{Diag}(\Sigma))}{\det\Sigma}\right) && \text{(A.24)}\\
&= \frac{1}{2}\log\frac{\det(\operatorname{Diag}(\Sigma))}{\det\Sigma}, && \text{(A.25)}
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{D}_{\mathrm{KL}}(\mathcal{N}(m,\operatorname{Diag}(\Sigma))\|\mathcal{N}(0,I)) &= \frac{1}{2}\Big(\operatorname{tr}\operatorname{Diag}(\Sigma) - D - \log(\det(\operatorname{Diag}(\Sigma)))\Big) && \text{(A.26)}\\
&= \frac{1}{2}\Big(\operatorname{tr}\Sigma - D - \log(\det(\operatorname{Diag}(\Sigma)))\Big). && \text{(A.27)}
\end{aligned}$$

Adding the two divergences yields equation (4.36).

Equation (4.37) is due to (Cardoso, 2003, Section 2.1). □

A.1.2. Proof of lemma 4.2

Lemma 4.2. *Given a D-dimensional distribution $p_(x)$ and an affine-linear function*

$$z = f_\theta(x) = Ax + b \tag{A.28}$$

for some invertible $A \in \mathbb{R}^{D\times D}$ and $b \in \mathbb{R}^D$. Then:

$$G[p_\theta(z)] = G[p(x)]. \tag{A.29}$$

Proof. The non-Gaussianity G is given by:

$$G[p(x)] = \mathcal{D}_{\mathrm{KL}}(p(x) \| \mathcal{N}(m, \Sigma)). \tag{A.30}$$

Mean and covariance of the push-forward of p through f_θ read:

$$\mathbb{E}_{x\sim p(x)}[f_\theta(x)] = \mathbb{E}_{x\sim p(x)}[Ax + b] = Am + b = \tilde{m}, \tag{A.31}$$

$$\mathrm{Cov}_{x\sim p(x)}[f_\theta(x)] = \mathrm{Cov}_{x\sim p(x)}[Ax + b] = A\Sigma A^{\mathrm{T}} = \tilde{\Sigma}. \tag{A.32}$$

Thus, the non-Gaussianity after applying f_θ reads:

$$G[g_\sharp p] = \mathcal{D}_{\mathrm{KL}}(g_\sharp p \| \mathcal{N}(\tilde{m}, \tilde{\Sigma})). \tag{A.33}$$

The push-forward of $\mathcal{N}(m, \Sigma)$ via f_θ is identical to the normal distribution that occurs in the non-Gaussianity of $g_\sharp p$:

$$(f_\theta)_\sharp \mathcal{N}(m, \Sigma) = \mathcal{N}(\tilde{m}, \tilde{\Sigma}), \tag{A.34}$$

Now, we make use of the fact that the KL divergence is invariant if both arguments are transformed by any invertible function f_θ:

$$\mathcal{D}_{\mathrm{KL}}(p_1(x) \| p_2(x)) = \mathcal{D}_{\mathrm{KL}}(((f_\theta)_\sharp p_1)(x) \| ((f_\theta)_\sharp p_2)(x)). \tag{A.35}$$

Together,

$$G[p_\theta(z)] = G[p(x)]. \tag{A.36}$$

□

A.2. Proofs on Volume-preserving Flows

This appendix is adapted from Draxler et al. (2024b).

A.2.1. Minimizer of Volume-Preserving Normalizing Flows

In this section, we consider what distribution a volume-preserving flow converges to instead. We first construct the latent distribution a sufficiently rich volume-preserving flows converges to when trained with KL divergence and then show that this actually minimizes the KL.

We also demonstrate in what sense $\Delta_{\mathrm{affine}}[p_\theta(z)]$ is sensitive to volume-preserving transformations. We therefore show that this flow converges under our convergence measure $\Delta_{\mathrm{affine}}[p_\theta(z)]$ if and only if that volume-preserving flow converges to a standard normal in the latent space under KL divergence.

Rotationally Symmetric Distribution with Same Level Set Structure

Let us repeat the change-of-variables formula for a volume-preserving flow from equation (4.31):

$$p_\theta(x) = p(z = f_\theta(x)) \cdot C, \tag{A.37}$$

where $C = |f'_\theta(x)|$ is constant with respect to x. Intuitively, this means that such a flow can *permute* the probability mass at all locations x, and apply a single global factor to scale all probability values by spreading out the distribution.

We now construct the best possible distribution learned by a volume-preserving flow to an arbitrary input $p(x)$ in terms of KL divergence. We therefore split the input space into the into level sets of $p(x)$:

$$L_v(p(\cdot)) := \{x \in \mathbb{R}^D : p(x) = v\}. \tag{A.38}$$

Acting on the input distribution, a volume-preserving flow yields a latent distribution $p_\theta(z)$ whose level set structure is closely related to that of the input distribution, in the following sense:

$$|L_v(p(x))| = \frac{|L_{v/C}(p_\theta(z))|}{C}. \tag{A.39}$$

Here, with $|\cdot|$ we denote the $(D-1)$-dimensional volume, thereby assuming:

Assumption A.1. The level sets of the data distribution $L_v(p(x))$ are $(D-1)$-dimensional.

Intuitively, this captures that a volume-preserving flow maps the input space to the latent space such that the level set in data space for the density value v is mapped to the level set in the latent space of level v/C – and the volumes are scaled by the factor C.

In the following, we use these level sets to construct the distribution $p^*(z)$ a volume-preserving flow converges to in the latent space. This allows us to specify the best solution a volume-preserving flow can converge to. We first consider a fixed C and then later consider choosing C.

Lemma A.1. *Let $p(x)$ be a bounded continuous probability density under assumption A.1. Then, a unique continuous probability density $p^*(z)$ with the following properties exists:*

1. *Its level sets have equal volume: $|L_v(p)| = |L_v(p^*)|$,*
2. *p^* is rotationally symmetric: $p^*(z) = p^*(Qz)$ for all $Q \in SO(D)$,*
3. *$p^*(z_1, 0, \ldots, 0)$ is strictly monotonically decreasing in $0 \leq z_1 < \infty$.*

Proof. We write $p^*(x) = p^*(r)p^*(\Omega|r)$, where (r, Ω) are hyper-spherical coordinates. Since $p^*(x)$ should be rotationally symmetric, the distribution of the solid angle Ω is isotropic, and equal to one over the surface $A_{D-1}(r)$ of the $(D-1)$-dimensional hypersphere:

$$p^*(\Omega|r) = \frac{1}{A_{D-1}(r)} = \frac{\Gamma(D/2)}{2\pi^{\frac{D}{2}} r^{D-1}} \tag{A.40}$$

This makes p^* rotationally symmetric and leaves us with constructing $p^*(r)$.

Define the superlevel sets v for $p(x)$ as follows:

$$L_v^+(p) = \{x \in \mathbb{R}^D : p(x) \geq v\}. \tag{A.41}$$

Their volume $|L_v^+(p)|$ as measured in D dimensions is monotonically decreasing in v and its derivative yields the volume of the $(D-1)$-dimensional level set:

$$\frac{\partial}{\partial v}|L_v^+(p)| = |L_v(p)|. \tag{A.42}$$

We now demand that

$$|L_v^+(p)| = |L_v^+(p^*)| \tag{A.43}$$

and integrate out Ω:

$$|L_v^+(p^*)| = \int \mathbf{1}[p^*(x) \geq v]dx \tag{A.44}$$

$$= \int d\Omega \int r^{D-1}\mathbf{1}[p^*(r) \geq v]dr \tag{A.45}$$

$$= A_{D-1}(1) \int r^{D-1}\mathbf{1}[p^*(r) \geq v]dr. \tag{A.46}$$

Since $p^*(r)$ should decrease monotonically with r, we can replace the indicator function by integral boundaries, where the upper limit depends on the target density value v. We identify $\max_x p(x)$ (exists because $p(x)$ is bounded) with $R = 0$:

$$|L_v^+(p^*)| = A_{D-1}(1) \int_0^{R(v)} r^{D-1} dr \tag{A.47}$$

$$= A_{D-1}(1) \frac{1}{D} R(v)^D \tag{A.48}$$

Rearranging yields $R(v)$ from $|L_v^+(p)|$:

$$R(v) = \left(\frac{D|L_v^+(p)|}{A_{D-1}(1)} \right)^{\frac{1}{D}} \tag{A.49}$$

As $R(v)$ is monotonous in $|L_v^+(p)|$, $|L_v^+(p)|$ is continuous and monotonous in v, and $R(v)$ is invertible, its inverse can be used to define $p^*(r)$:

$$p^*(r) = R^{-1}(r). \tag{A.50}$$

By choosing $R(v)$ to fulfill equation (A.43), their derivatives also match:

$$|L_v(p)| = \frac{\partial}{\partial v} |L_v^+(p)| = \frac{\partial}{\partial v} |L_v^+(p^*)| = |L_v^+(p^*)|. \tag{A.51}$$

The other properties of p^* follow directly from the construction above. The density is unique (up to zero sets) since a rotationally symmetric distribution is uniquely defined by a one-dimensional ray (Eaton, 1986). □

We now show that the latent distribution $p^*(z)$ is actually attainable by a volume-preserving flow:

Lemma A.2. *Under the assumptions of lemma A.1. There exists a bijective function $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$ that is continuous and volume-preserving with unit volume change ($|f'_\theta(x)| = 1$) almost everywhere and $p_\theta(z) = p^*(x = z)$.*

This means that there is a volume-preserving flow that exactly pushes $p(x)$ to its respective $p^*(x)$. Note that this volume-preserving flow has $|f'_\theta(x)| = 1$.

Proof. Divide the space into the level sets $L_p(v)$ of $p(x)$. Since $p(x)$ is continuous, there is a countable sequence of thresholds v_i at which the number of connected components in the level set $L_p(v)$ jumps: The first jump is at $v_{\max} = \max_x p(x)$, below which find as many connected components as there are maximal modes in $p(x)$. The number of connected components changes whenever there is a saddle point or maximum in $p(x)$. Between each subsequent pair of jumps (v_i, v_{i+1}), each connected component can be continuously assigned to a countable cluster number. This yields two tessellations of the entire space: One into level sets, and one into continuous connected components. To construct f_θ, we assign the highest points of $x^* \in \mathbb{R}^D$ to $f_\theta(x^*) = 0$. Then, we continuously arrange the finite number of components until the next jump in $(\max_x p(x), v_1)$ around the origin, such that the resulting level sets are concentric circles. This constructs f_θ that pushes $p(x)$ to $p^*(x)$ and fulfills the above constraints. □

Best Volume-Preserving Normalizing Flow under KL Divergence

We are going to make use of the following identity that a volume-preserving flow with $|f'_\theta(x)| = C \neq 1$ with a standard normal in the latent space can be written as a volume-preserving flow with $|\tilde{f}'_\theta(x)| = 1$ and an alternative latent distribution $\tilde{p}(z) = \mathcal{N}(0, C^{-2/D}I)$:

Lemma A.3. *Given a volume-preserving bijection f_θ that is diffeomorphic almost everywhere, and with $|f'_\theta(x)| = C$ for some $C > 0$. Then, there exists a volume-preserving bijection $\tilde{f}_\theta$ with $|\tilde{f}'_\theta(x)| = 1$ such that:*

$$((f_\theta^{-1})_\sharp \mathcal{N}(0, I))(x) = ((\tilde{f}_\theta^{-1})_\sharp \mathcal{N}(0, C^{-2/D}I))(x) \tag{A.52}$$

In other words, the global volume change of a volume-preserving flow with standard normal latent space can be absorbed into a single scaling layer at the latent end of the flow.

Proof. Let $\tilde{f}_\theta(x) = C^{-1/D} f_\theta(x)$, which has $|\tilde{f}'_\theta(x)| = C/C = 1$ and write $\mathcal{N}(0, C^{-2/D}I)$ as the push-forward of $\mathcal{N}(0, I)$ through $z \mapsto C^{-1/D}z$. □

Now we show our main theorem 5.2 on volume-preserving flows, formalized in terms of the learned latent distribution as constructed in lemma A.1:

Theorem A.4 (Best possible volume-preserving flow, formalized)**.** *Given a continuous bounded probability density $p(x)$ with $(D-1)$-dimensional level sets. Then, the minimal achievable KL divergence by a volume-preserving normalizing flow $p^*_\theta(x)$ whose underlying map f^*_θ is continuous almost everywhere and with a standard normal latent distribution reads:*

$$\mathcal{D}_{KL}(p(x)\|p^*_\theta(x)) = \mathcal{D}_{KL}(p^*(z)\|\mathcal{N}(0, |\Sigma_{p^*(z)}|^{\frac{1}{D}} I)), \tag{A.53}$$

where $p^*(z)$ is constructed as in lemma A.1, and $\Sigma_{p^*(z)}$ is its covariance matrix. The minimal loss is achieved for $C = |\Sigma_{p^*(z)}|^{-\frac{1}{2D}}$.

Proof. By equation (4.10), $\mathcal{D}_{\mathrm{KL}}(p(x)\|p_\theta(x)) = \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0, I))$. The variant in the latent space can be rewritten using the entropy of $p_\theta(z)$ as:

$$\mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0, I)) = \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0, I)) = -H[p_\theta(x)] - \int p_\theta(z) \log \mathcal{N}(z; 0, I) dz. \tag{A.54}$$

The entropy of the latent distribution of a volume-preserving flow only depends on the volume change constant C, but not on the exact choice of f_θ:

$$\begin{aligned} H[p_\theta(z)] &= -\int p_\theta(z) \log p_\theta(z) dz && \text{(A.55)} \\ &= -\int p(x) \log(p_\theta(z = f_\theta(x))C) dx - \log C && \text{(A.56)} \\ &= -\int p(x) \log p(x) dx - \log C && \text{(A.57)} \\ &= H[p(x)] - \log C. && \text{(A.58)} \end{aligned}$$

Inserting into equation (A.54):

$$\mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0, I)) = \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0, I)) = -H[p(x)] + \log C - \int p_\theta(z) \log \mathcal{N}(z; 0, I) dz. \tag{A.59}$$

Using lemma A.3, rewrite the last term in equation (A.59) as an integral over x:

$$-\int p_\theta(z)\log\mathcal{N}(z;0,C^{-2}I)dz = -\int p(x)\log\mathcal{N}(\tilde{f}_\theta(x);0,C^{-2}I)dx. \tag{A.60}$$

This reveals the KL is minimized by assigning the highest values of $p(x)$ to the highest values of $\mathcal{N}(0,C^{-2}I)$. Since the order of densities $\mathcal{N}(z;0,C^{-2}I)$ is the same regardless of C, the assignment $\tilde{f}_\theta(x)$ does not depend on C, which can be estimated separately via

$$\mathcal{D}_{\mathrm{KL}}(p^*(z)\|\mathcal{N}(0,C^{-2}I)). \tag{A.61}$$

By theorem 4.1, the KL divergence equation (A.61) can be decomposed as follows:

$$\mathcal{D}_{\mathrm{KL}}(p^*(z)\|\mathcal{N}(0,C^{-2}I)) = \mathcal{D}_{\mathrm{KL}}(p^*(z)\|\mathcal{N}(0,|\Sigma_z|^{\frac{1}{D}}I)) + \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(0,|\Sigma_z|^{\frac{1}{D}}I)\|\mathcal{N}(0,C^{-2}I)), \tag{A.62}$$

where $|\Sigma_z|$ is the determinant of the covariance matrix of the latent codes of $p^*(z)$: $\Sigma_z = \mathrm{Cov}_{z\sim p^*(z)}[z] = |\Sigma_z|^{\frac{1}{D}}I$ due to the rotational symmetry. The first term is invariant under scaling the latent codes, and the second term is minimal for $C^{-2} = |\Sigma_z|^{\frac{1}{D}}$.

□

The remaining loss $\mathcal{D}_{\mathrm{KL}}(p^*(z)\|\mathcal{N}(0,|\Sigma_{p^*(z)}|^{\frac{1}{D}}I))$ can be reduced to zero by switching to spherical coordinates and learning $p^*(r)$ via a non-volume-preserving one-dimensional distribution, see appendix A.2.2.

Sensitivity of Δ_{affine} to Volume-Preserving Flows

We now confirm that if a volume-preserving flow converges in KL divergence to $p^*(x)$ instead of $p(x)$, this is also noted by Δ_{affine}.

Lemma A.5. *For a volume-preserving normalizing flow such that $\mathcal{D}_{KL}(p_\theta(z)\|p^*(z)) \to 0$, it holds that $\Delta_{\text{affine}}[p_\theta(z)] \to 0$ if and only if $\mathcal{D}_{KL}(p^*(z)\|\mathcal{N}(0,I)) = 0$.*

Proof. By lemma 5.17, we can use $\Delta^*_{\text{affine}} > 0$ and $\Delta_{\text{affine}} > 0$ interchangeably. By its definition in equation (5.32),

$$\Delta^*_{\text{affine}} = \max_Q \tfrac{1}{2}\sum_i^{D/2}\mathbb{E}_b\Big[m_i^2(b) + \sigma_i^2(b) - 1 - \log\sigma_i^2(b)\Big] \tag{A.63}$$

According to theorem 5.2, the latent distribution $p_\theta(z) = p^*(z = x)$ minimizes the KL in the latent space: $\mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0,I))$ (the exact assignment between data and latent codes is not unique, but all lead to the same latent estimate).

At this minimum, $p_\theta(z) = p^*(x = z)$. Since $p^*(x)$ is symmetric under rotations, it holds that, it holds that $m_i(b) = 0$ for $(a,b) = Qx$ in all rotations Q. However, since $\mathrm{Var}_{a\sim(Q\sharp p^*)(a)[a_i|b]} \neq 1$ for all Q, it holds that

$$\Delta^*_{\text{affine}} = \tfrac{1}{2}\sum_i^{D/2}\mathbb{E}_b\Big[\sigma_i^2(b) - 1 - \log\sigma_i^2(b)\Big], \tag{A.64}$$

which evaluates to the same value regardless of Q since the distribution is rotationally symmetric.

While this minimum may not be exactly achieved by a continuous volume-preserving flow, a sufficiently rich architecture is able to achieve universality $\mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0,I)) \to \mathcal{D}_{\mathrm{KL}}(p^*(z)\|\mathcal{N}(0,I))$. Since the KL divergence implies the convergence of expectation values (Gibbs & Su, 2002), it holds that $\Delta^*_{\text{affine}} \to \frac{1}{2}\sum_i^{D/2}\mathbb{E}_b\Big[\sigma_i^2(b) - 1 - \log\sigma_i^2(b)\Big] \geq 0$. Equality holds if and only if $\mathcal{D}_{\mathrm{KL}}(p^*(z)\|\mathcal{N}(0,I)) = 0$. □

Note that the same argument also applies to Wasserstein distance and weak convergence, so this does not indicate that $\Delta_{\text{affine}}[p_\theta(z)]$ is more informative about convergence than these convergence metrics.

A.2.2. Fixing Volume-Preserving Flows by Learning the Latent Radial Distribution

By Theorem 5.2, the global minimizer of a volume-preserving flow is given by $p^*(z)$ in the latent space, as characterized by lemma A.1. Both $p^*(z)$ and the standard normal distribution $p(z) = \mathcal{N}(0, I)$ are rotationally symmetric, so it is useful to make a change of variables to hyperspherical coordinates (r, Ω):

$$p^*(z) = p^*(r(z))p^*(\Omega(z)|r(z))\left|\frac{d(r,\Omega)}{dz}\right|, \tag{A.65}$$

$$p(z) = p(r(z))p(\Omega(z)|r(z))\left|\frac{d(r,\Omega)}{dz}\right|. \tag{A.66}$$

The rotational symmetry implies that:

$$p^*(\Omega|r) = p(\Omega|r) = \frac{1}{A_{D-1}(r)}, \tag{A.67}$$

where $A_{D-1}(r)$ is the surface of the $(D-1)$-dimensional sphere of radius r.

This means that we only need to match $p^*(r)$ and $p(r)$. This can be achieved by a one-dimensional transformation of r.

Fixing the latent distribution can even be done after training, even if training had fixed the volume change $|f'_\theta(x)| = 1$: Training the volume-preserving flow with a standard normal in the latent space sorts the data such that $p(x_a) \geq p(x_b)$ implies that $r_a \leq r_b$, that is points of higher ground truth density are mapped to points closer to the origin. If we now replace the latent distribution with another distribution that fulfills the same constraint by fitting $p^*(r)$, the minimizer with this latent distribution remains the same. Thus, we can train with a standard normal and later fix the one-dimensional distribution.

A.2.3. Proof of theorem 5.3

Theorem 5.3. *The family of volume-preserving normalizing flows is* not *a universal distribution approximator under KL divergence.*

Proof. Consider the data distribution

$$p(x) = \frac{\alpha^2}{2\pi}e^{-\alpha\sqrt{x_1^2+x_2^2}} \tag{A.68}$$

This distribution is rotationally symmetric, and it is monotonically decreasing in x_1, so it fulfills the conditions of $p^*(z)$ in lemma A.1, meaning that the best possible latent distribution is $p^*(z) = p(x = z)$.

Let us compute the KL divergence between this optimal latent distribution and a normal distribution with covariance $\Sigma = \sigma^2 I$ in polar coordinates:

$$\mathcal{D}_{\text{KL}}(p^*(z)\|\mathcal{N}(0, \sigma^2 I)) \tag{A.69}$$

$$= \alpha^2 \int re^{-\alpha r}\left(2\log(\alpha) - \log(2\pi) - \alpha r + \log(2\pi) + 2\log(\sigma) + \log\frac{1}{2\sigma^2}r^2\right)dr \tag{A.70}$$

$$= 2\log(\alpha\sigma) - 2 + \frac{3}{\alpha^2\sigma^2} \tag{A.71}$$

$$= \log \kappa - 2 + \frac{3}{\kappa}, \tag{A.72}$$

where we have defined $\kappa := \alpha^2\sigma^2$.

To find the minimum, differentiate with respect to κ:

$$\frac{\partial}{\partial \kappa}\mathcal{D}_{\mathrm{KL}}(p^*(z)\|\mathcal{N}(0, \sigma^2 I)) = \frac{1}{\kappa} - \frac{3}{\kappa^2}, \tag{A.73}$$

which is zero for $\kappa_{\mathrm{crit}} = 3$. This critical point is a minimizer since the KL goes to ∞ for $\kappa \to 0$ and $\kappa \to \infty$ and κ_{crit} is the only critical point.

We find at the extremum:

$$\mathcal{D}_{\mathrm{KL}}(p^*(z)\|\mathcal{N}(0, \sigma^2 I))\Big|_{\kappa=3} = -1 + \log 3 = 0.09861... > 0. \tag{A.74}$$

Thus, there exists a distribution for which by theorem 5.2 the KL divergence achievable by a volume-preserving flow is bounded from below by a nonzero value. This makes volume-preserving flows not universal under KL divergence. □

A.2.4. Proof of proposition 5.4

Definition A.6. *Given a probability density $p(x)$ and a connected set $M \subset \mathbb{R}^D$. Then, M is called a* mode *of $p(x)$ if*

$$p(x) = p(y) \quad \forall x, y \in M, \tag{A.75}$$

and there is a neighborhood U of M such that:

$$p(x) > p(y) \quad \forall x \in M, y \in U \setminus M. \tag{A.76}$$

With this definition of a mode, let us characterize the correspondence between modes of $p_\theta(x)$ and $p(z)$ for a volume-preserving flow:

Lemma A.7. *Given a latent probability density $p(z)$, a diffeomorphism $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$ with constant Jacobian determinant $|f'_\theta(x)| = C$ and a mode $M \subset \mathbb{R}^D$ of $p(z)$. Then, $f_\theta^{-1}(M)$ is a mode of $p_\theta(x)$.*

Proof. We show that $f_\theta^{-1}(M)$ fulfills definition A.6. First, for every $x, y \in f_\theta^{-1}(M)$: Then, by construction: $f_\theta(x), f_\theta(y) \in M$. As M is a mode:

$$p(z = f_\theta(x)) = p(z = f_\theta(y)). \tag{A.77}$$

We follow:

$$p_\theta(x) = p(z = f_\theta(x))C = p(z = f_\theta(y))C = p_\theta(y), \tag{A.78}$$

where we have used the volume-preserving change of variables formula in equation (4.31).

Second, let U be a neighborhood of M such that equation (A.76) is fulfilled. As f_θ is a diffeomorphism, there is a neighborhood V of $f_\theta^{-1}(M)$ such that $V \subseteq f_\theta^{-1}(U)$. Consider $x \in f_\theta^{-1}(M), y \in V \setminus f_\theta^{-1}(M)$. As M is a mode:

$$p(z = f_\theta(x)) > p(z = f_\theta(y)). \tag{A.79}$$

Multiplying both sides by C, we find:

$$p_\theta(x) = p(f_\theta(x))C > p(z = f_\theta(y))C = p_\theta(y). \tag{A.80}$$

Thus, $f^{-1}(M)$ is a mode of $p_\theta(x)$ by definition A.6. □

This makes us ready for the proof:

Proposition 5.4. *A normalizing flow $p_\theta(x)$ based on a volume-preserving diffeomorphism $f_\theta(x)$ has the same number of modes as the latent distribution $p(z)$.*

Proof. By lemma A.7, every mode of $p(z)$ implies a mode of $p_\theta(x)$. Applied reversely, every mode of $p_\theta(x)$ implies a mode of $p(z)$. Therefore, there is a one-to-one correspondence of modes between $p(z)$ and $p_\theta(x)$. □

A.3. Single non-linear affine layer

This appendix is adapted from Draxler et al. (2020).

A.3.1. Proof of lemma 5.5

Lemma 5.5. *Given a probability density $p(x)$ with finite first and second moments, and a single affine coupling layer f_{cpl} after a fixed rotation Q so that $(b, a) = Qx$. Then, equation (5.5) is uniquely minimized by the following scaling and translations as a function of the passive dimensions b:*

$$s_i(b) = \frac{1}{\sqrt{\text{Var}_{a_i|b}[a_i]}} =: \frac{1}{\sigma_i(b)}, \tag{A.81}$$

$$t_i(b) = -\mathbb{E}_{a_i|b}[a_i] s_i(b) =: -\frac{m_i(b)}{\sigma_i(b)}. \tag{A.82}$$

Proof. The affine nonlinearity c leaves the passive dimensions b unchanged. This leaves us with the minimization problem in equation (5.5):

$$\min_{s,t:\mathbb{R}^{D/2}\to\mathbb{R}^{D/2}} \mathbb{E}_{b,a}\left[\frac{1}{2}\|a \odot e^{s(b)} + t(b)\|^2 - \sum_{i=1}^{D/2} s_i(b)\right], \tag{A.83}$$

where $\mathbb{E}_{b,a}$ is shorthand for $\mathbb{E}_{b,a\sim Q_\sharp p}$.

Under the assumption that s, t are arbitrary functions without smoothness constraints, the above minimization problem decouples into one for each value of b. We fix b for what follows and write $s = s(b), t = t(b) \in \mathbb{R}^{D/2}$ instead of the corresponding functions and obtain:

$$\min_{s,t\in\mathbb{R}^{D/2}} \mathbb{E}_{a_i|b}\left[\frac{1}{2}\left\|a \odot e^{s} + t\right\|^2 - \sum_{i=1}^{D/2} s_i(b)\right]. \tag{A.84}$$

This can be decoupled into $D/2$ independent minimization problems, indexed by $i = 1, \ldots, D/2$:

$$\min_{s_i,t_i\in\mathbb{R}} \mathbb{E}_{a_i|b}\left[\frac{1}{2}(a_i e^{s_i} + t_i)^2 - s_i\right]. \tag{A.85}$$

At an extremal point, we find

$$\partial_{s_i}\mathbb{E}_{a_i|b}\left[\frac{1}{2}(e^{s_i}a_i + t_i)^2 - s_i\right] = \mathbb{E}_{a_i|b}\Big[(e^{s_i}a_i + t_i)e^{s_i}a_i - 1\Big] = 0, \tag{A.86}$$

$$\partial_{t_i}\mathbb{E}_{a_i|b}\left[\frac{1}{2}(e^{s_i}a_i + t_i)^2 - s_i\right] = \mathbb{E}_{a_i|b}\Big[(e^{s_i}a_i + t_i)\Big] = 0. \tag{A.87}$$

We can solve the second equation for t_i:

$$t_i = -\mathbb{E}_{a_i|b}[a_i]e^{s_i}. \tag{A.88}$$

Insert this into the condition on s_i:

$$\mathbb{E}_{a_i|b}\Big[2(e^{s_i}a_i - \mathbb{E}_{a_i|b}[a_i]e^{s_i})e^{s_i}a_i - 1\Big] = e^{2s_i}\operatorname{Var}_{a_i|b}[a_i] - 1 = 0, \tag{A.89}$$

and find:

$$\begin{aligned} e^{2s_i}\operatorname{Var}_{a_i|b}[a_i] &= 1, \\ e^{s_i} &= \frac{1}{\sqrt{\operatorname{Var}_{a_i|b}[a_i]}}. \end{aligned} \tag{A.90}$$

This is the statement, written component-wise and for a fixed b. □

A.3.2. Proof of theorem 5.8

We will use the chain rule for the KL divergence for $x = (y, z)$:

$$\mathcal{D}_{\mathrm{KL}}(p(y,z)\|q(y,z)) = \mathcal{D}_{\mathrm{KL}}(p(y)\|q(y)) + \mathbb{E}_y[\mathcal{D}_{\mathrm{KL}}(p(z|y)\|q(z|y)]. \tag{A.91}$$

Theorem 5.8. *Given a probability density $p(x)$ with finite first and second moments, and a single affine coupling layer f_{cpl} after a fixed rotation Q so that $(b, a) = Qx$. Let f_{cpl} be the minimizer from lemma 5.5 and $\tilde{x} = f_{\mathrm{cpl}}(Qx) = (\tilde{b}, \tilde{a})$. Then, the loss has the following minimal value:*

$$\begin{aligned} \mathcal{D}_{KL}(p_\theta(\tilde{x})\|p(z)) &= \mathcal{D}_{KL}(p(b)\|\mathcal{N}(0,I)) + \mathbb{E}_{b\sim p(b)}\Big[G[p(a|b)] + C(m_{a|b}, \Sigma_{a|b})\Big] && \text{(A.92)} \\ &= \mathcal{D}_{KL}(p(b)\|\mathcal{N}(0,I)) - \Delta^*_{\mathrm{affine}}(Q). && \text{(A.93)} \end{aligned}$$

Here, the non-Gaussianity $G[p(\cdot)]$ and correlation $C(m, \Sigma)$ are defined in theorem 4.1 and $m_{a|b}$ and $\Sigma_{a|b}$ are the mean and covariance of $p(a|b)$. The loss improvement can explicitly be computed from the conditional moments of $p(a|b)$:

$$\Delta^*_{\mathrm{affine}}(Q) = \sum_{i=1}^{D/2}\mathbb{E}_b[S(m_i(b), \sigma_i(b))] = \frac{1}{2}\sum_{i=1}^{D/2}\mathbb{E}_b[m_i^2(b) + \sigma_i^2(b) - 1 - \log\sigma_i^2(b)]. \tag{A.94}$$

Here, $S(m, \sigma) = \mathcal{D}_{KL}(\mathcal{N}(m, \sigma)\|\mathcal{N}(0, 1))$ is the univariate non-Standardness like in in theorem 4.1.

Proof. By the chain rule of the KL divergence in equation (A.91), we can write the KL divergence before the transport as:

$$\mathcal{D}_{\mathrm{KL}}(p(x)\|\mathcal{N}(0,I)) = \mathcal{D}_{\mathrm{KL}}(p(b)\|\mathcal{N}(0,I)) + \mathbb{E}_b[\mathcal{D}_{\mathrm{KL}}(p(a|b)\|\mathcal{N}(0,I))], \tag{A.95}$$

where we take the dimension of I to match the distribution in the other argument of the KL divergence. By theorem 4.1,

$$\mathcal{D}_{\mathrm{KL}}(p(a|b)\|\mathcal{N}(0,I)) = G[p(a|b)] + C(m_{a|b}, \Sigma_{a|b}) + \sum_{i=1}^{D/2} S(m_i(b), \sigma_i(b)). \tag{A.96}$$

Applying the transport in equation (5.8), we find the outgoing moments:

$$\tilde{m}_i(b) = 0, \quad \tilde{\sigma}_i(b) = 1. \tag{A.97}$$

Thus, $S(\tilde{m}_i(b), \tilde{\sigma}_i(b)) = 0$ and by lemmas 4.2 and 4.3 the non-Gaussianity G and the correlation C (which is a mutual information) are left unchanged. Also, $\tilde{b} = b$, and so the passive KL contribution is also left unchanged.

Taking the difference, we find:

$$\Delta^*_{\text{affine}}(Q) = \mathcal{D}_{\text{KL}}(p(x)\|\mathcal{N}(0,I)) - \mathcal{D}_{\text{KL}}(p_\theta(\tilde{x})\|\mathcal{N}(0,I)) = \sum_{i=1}^{D/2} S(m_i(b), \sigma_i(b)), \tag{A.98}$$

which concludes the proof.

□

A.3.3. Proof of corollary 5.9

Corollary 5.9. *If and only if $p(a|b)$ is normally distributed for all b with diagonal covariance, that is:*

$$p(a|b) = \prod_{i=1}^{D/2} \mathcal{N}(a_i|m_i(b), \sigma_i(b)), \tag{A.99}$$

a single affine block can reduce the KL divergence on the active subspace to zero:

$$\mathcal{D}_{KL}(p(x)\|p_\theta(x)) = 0. \tag{A.100}$$

Proof. "⇒": Insert this particular choice of $p(x)$ into equation (5.16) to obtain the result.

"⇐": The push-forward $p_\theta(\tilde{a}|b)$ can only be written as a product of its marginals if $p(a|b)$ was a product distribution for each b. Then, equation (5.16) decouples into contributions from each $p(a_i|b)$. Each contribution is the non-Gaussianity of $p(a_i|b)$ which is only zero if $p(a_i|b)$ is Gaussian. □

A.3.4. Proof of lemma 5.12

Lemma 5.12. *Given a distribution p and a single affine coupling layer f with a fixed rotation Q. Call $(a, b) = Qx$ the rotated versions of $x \sim p$. Then, the following are equivalent:*

1. *$\tilde{a} \perp \tilde{b}$ for $(\tilde{a}, \tilde{b}) = f_{\text{cpl}}(a, b)$ minimizing the ML loss in equation (5.5),*
2. *there exists $n \perp b$ such that $a = f(b) + n \odot g(b)$, where $f, g : \mathbb{R}^{D/2} \to \mathbb{R}^{D/2}$.*

Proof. $1 \Rightarrow 2$: Rewriting equation (5.8), we find:

$$a = m(b) + \sigma(b) \odot \tilde{a} =: f(b) + g(b) \odot n. \tag{A.101}$$

By assumption, $n = \tilde{a} \perp b$ and and we obtain the statement.

$2 \Rightarrow 1$: In the following, we omit "$\odot$" and all multiplications are element-wise.

We first identify the solution as in equation (5.8) and then show that the resulting variable is independent of b. In the following, we write $f = f(b)$ and $g = g(b)$.

$$\mathbb{E}_n[a] = f + \mathbb{E}_n[n]g, \tag{A.102}$$

$$\mathbb{E}_n[a^2] = f^2 + 2fg\mathbb{E}_n[n] + g^2\mathbb{E}[n^2]. \tag{A.103}$$

We combine:

$$m(b) = f + \mathbb{E}_n[n]g, \tag{A.104}$$

$$\sigma(b) = \sqrt{f^2 + 2fg\mathbb{E}_n[n] + g^2\mathbb{E}[n^2] - (f^2 + 2fg\mathbb{E}_n[n] + g^2\mathbb{E}_n[n]^2)} \tag{A.105}$$

$$= g\sigma_n. \tag{A.106}$$

The resulting $\tilde{a}$ from this transport reads:

$$\tilde{a} = f_{\text{cpl}}(a|b) = \frac{1}{\sigma(b)}(a - m(b)) \tag{A.107}$$

$$= \frac{1}{g\sigma_n}(f + ng - (f + \mathbb{E}_n[n]g)) \tag{A.108}$$

$$= \frac{1}{\sigma_n}(n - \mathbb{E}_{\hat{n}}[\hat{n}]). \tag{A.109}$$

This is independent of $\tilde{b}$. □ □

A.4. Proofs on affine coupling flow universality

This appendix is adapted from Draxler et al. (2024b).

A.4.1. Proof of unique fixed point of affine coupling blocks

Underlying results from previous work

Here, we restate the results from the literature that our proof is based on:

First, for some vector-valued random variable X and every pair of orthogonal projections the mean of one projection conditioned on the other is zero, then X follows a spherical distribution:

Theorem A.8 (Eaton (1986))**.** *Suppose the random vector $X \in \mathbb{R}^D$ has a finite mean vector. Assume that for each vector $v \neq 0$ and for each vector u perpendicular to v (i.e. $u \cdot v = 0$):*

$$\mathbb{E}[u \cdot X | v \cdot X] = 0. \tag{A.110}$$

Then X is spherical and conversely.

Secondly, Cambanis et al. (1981, Corollary 8a) identifies the Gaussian from all elliptically contoured (which includes spherical) distributions. We write it in the form of Bryc (1995, Theorem 4.1.4):

Theorem A.9 (Bryc (1995))**.** *Let $p(x)$ be radially symmetric with $\mathbb{E}[\|x\|^\alpha] < \infty$ for some $\alpha > 0$. If*

$$\mathbb{E}[\|x_{1,\dots,m}\|^\alpha | x_{m+1,\dots,n}] = \text{const}, \tag{A.111}$$

for some $1 \leq m < n$, then $p(x)$ is Gaussian.

Relation to practical neural networks (lemma 5.17)

Before moving to the proof of theorem 5.16, we show the helper statement lemma 5.17.

To relate equations (5.6) and (5.18) to actually realizable networks, which cannot *exactly* follow the arbitrary continuous functions $s_i^*(b), t_i^*(b)$, the following statement asserts that the fixed point of adding coupling layers with infinitely expressive conditioner functions is the same as for actually realizable and well-conditioned coupling blocks:

Lemma 5.17. *Given a continuous probability density $p_\theta(z)$. Then,*

$$\Delta^*_{\text{affine}}[p_\theta(z)] > 0 \tag{A.112}$$

if and only if:

$$\Delta_{\text{affine}}[p_\theta(z)] > 0. \tag{A.113}$$

Proof. First, note that $\Delta^*_{\text{affine}}[p_\theta(z)] \geq \Delta_{\text{affine}}[p_\theta(z)] \geq 0$ since no practically realizable coupling block can achieve better than equation (5.18). Thus, if $\Delta^*_{\text{affine}}[p_\theta(z)] = 0$, so is $\Delta_{\text{affine}}[p_\theta(z)] = 0$.

Denote by $\Delta^*_{\text{affine}}(Q)$ and $\Delta_{\text{affine}}(Q)$ the best possible loss improvement for a fixed rotation Q, omitting the argument $p_\theta(z)$ for brevity, as in theorem 5.8.

Now choose the rotation $Q^* = \arg\max_Q \Delta^*_{\text{affine}}(Q)$ (pick any if not unique). Below, we show that then $\Delta_{\text{affine}}(Q^*) > 0$, and thus $\Delta_{\text{affine}}[p_\theta(z)] \geq \Delta_{\text{affine}}(Q^*) > 0$.

Without loss of generalization, we consider one single active dimension a_i in the following, but the construction can then be repeated for each other active dimension.

If we apply any affine coupling layer $c_i(a_i; b) = s_i(b) \odot a_i + t_i(b)$, the loss change by this layer can be computed from the theoretical maximal improvement $\Delta^*_{\text{affine}}(Q^*)$ before and after adding this layer $\tilde{\Delta}^*_{\text{affine}}(I)$:

$$\Delta_{\text{affine}}(Q^*) = \Delta^*_{\text{affine}}(Q^*) - \tilde{\Delta}^*_{\text{affine}}(I) \tag{A.114}$$

$$= \tfrac{1}{2}\mathbb{E}_b\Big[m_i(b)^2 + \sigma_i(b)^2 - 1 - \log \sigma_i(b)^2\Big] - \tfrac{1}{2}\mathbb{E}_b\Big[\tilde{m}_i(b)^2 + \tilde{\sigma}_i(b)^2 - 1 - \log \tilde{\sigma}_i(b)^2\Big]. \tag{A.115}$$

The moments after the affine coupling layers read:

$$\tilde{m}_i(b) = s_\varphi(b) m_i(b) + t_\varphi(b), \qquad \tilde{\sigma}_i(b) = s_\varphi(b)\sigma_i(b). \tag{A.116}$$

Case 1: $\mathbb{E}_b[\sigma_i(b)^2 - 1 - \log \sigma_i(b)^2] > 0$:

Then, without loss of generality, by continuity and positivity of $p_\theta(z)$ and consequential continuity of $\sigma_i(b)$ in b, there is a convex open set $A \subset \mathbb{R}^{D/2}$ with non-zero measure $p(A) > 0$ where $\sigma_i(b) > 1$. If $\sigma_i(b) < 1$ everywhere, apply the following argument flipped around $\sigma_i(b) = 1$.

Denote by $\sigma_{\max} = \max_{b \in A} \sigma_i(b)$. Then, by continuity of $\sigma_i(b)$ there exists $B \subset A$ so that $\sigma_i(b) > (\sigma_{\max} - 1)/2 + 1 =: \sigma_{\max/2}$ for all $b \in B$. Let $C \subset B$ be a multidimensional interval $[l_1, r_1] \times \cdots \times [l_{D/2}, r_{D/2}]$ with $p(C) > 0$ inside B.

Now, we construct a ReLU neural network with two hidden layers with the following property, where $F \subset E \subset C$ are specified later with $p(F) > p(E) > 0$:

$$\begin{cases} g_\varphi(x) = \frac{1}{\sigma_{\max/2}} & x \in E \subset D \\ \frac{1}{\sigma_{\max/2}} \leq g_\varphi(x) < 1 & x \in D \\ g_\varphi(x) = 0 & \text{else.} \end{cases} \tag{A.117}$$

To do so, we make four neurons for each dimension $i = 1, \ldots, D/2$:

$$\text{ReLU}(x_i - l_i), \text{ReLU}(x_i - l_i - \delta), \text{ReLU}(x_i - r_i), \text{ReLU}(x_i - r_i + \delta), \tag{A.118}$$

where $0 < \delta < \min_i(r_i - l_i)/4$. If we add these four neurons with weights $1, -1, -1, 1$, we find the following piecewise function:

$$\begin{cases} 0 & x \leq l_i \\ x - l_i & l_i < x < l_i + \delta \\ \delta & l_i + \delta \leq x \leq r_i - \delta \\ r_i - x & r_i - \delta < x < r_i \\ 0 & r_i \leq x. \end{cases} \tag{A.119}$$

If we repeat this for each dimension and add together all neurons with the corresponding weights into a single neuron in the second layer, then only inside $D = (l_1+\delta, r_1-\delta)\times\cdots\times(l_{D/2}+\delta, r_{D/2}-\delta) \subset C$ the weighted sum would equal $\delta D/2$. By choosing δ as above, this region has nonzero volume. We thus equip the single neuron in the second layer with a bias of $-\delta D/2 + \epsilon$ for some $\epsilon < \delta$, so that it is constant with value ϵ inside $E = (l_1 + \delta - \epsilon, r_1 - \delta + \epsilon) \times \cdots \times (l_{D/2} + \delta + \epsilon, r_{D/2} - \delta - \epsilon) \subset D$ and smoothly interpolates to zero in the rest of D.

For the output neuron of our network, we choose weight $(\sigma_{\max/2} - 1)/\epsilon$ and bias 1. By inserting the above construction, we find the network specified in equation (A.117).

Now, for all $b \in D$,

$$1 < \tilde{\sigma}_i(b) < \sigma_i(b), \tag{A.120}$$

so that

$$\tilde{m}_i(b)^2 + \tilde{\sigma}_i(b)^2 - 1 - \log\tilde{\sigma}_i(b)^2 < m_i(b)^2 + \sigma_i(b)^2 - 1 - \log\sigma_i(b)^2. \tag{A.121}$$

Thus, parameters φ exist that improve on the loss. (Note that this construction can be made more effective in practice by identifying the sets where $\sigma > 1$ resp. $\sigma < 1$ and then building neural networks that output one or scale towards $\tilde{\sigma}(b) = 1$ everywhere. Because we are only interested in identifying improvement, the above construction is sufficient.)

Now, regrading t_φ, we focus on $\mathbb{E}_b[m_i(b)^2] > 0$ (otherwise choose $t_\varphi = 0$ as a constant, which corresponds to a ReLU network with all weights and biases set to zero):

$$\mathbb{E}_b[m_i(b)^2] > \mathbb{E}_b[(s_\varphi(b)m_i(b) + t_\varphi(b))^2]. \tag{A.122}$$

By Hornik (1991, Theorem 1) there always is a t_φ that fulfills this relation.

Case 2: $\mathbb{E}_b[\sigma_i(b)^2 - 1 - \log\sigma_i(b)^2] = 0$. Then, choose the neural network $s_\varphi(b) = 1$ as a constant. As $\Delta^*_{\text{affine}}[p_\theta(z)] > 0$, $\mathbb{E}_{b\sim p(a,b)}[m_i(b)^2] > 0$ and we can use the same argument for the existence of t_φ as before.

We now show that a L-bi-Lipschitz coupling block can be constructed. To achieve this, replace the action of the coupling block $\tilde{a}_i = s(b)a_i + t(b)$ by $\tilde{a}_i = \alpha(s(b)a_i + t(b)) + (1-\alpha)a_i$. Since $s(b)$ and $t(b)$ above were constructed to move the data in the right direction, we obtain a finite loss improvement, since $\alpha > 0 \Leftrightarrow \Delta^*_{\text{affine}} > 0$. The Jacobian f'_{cpl} of the restricted coupling block is $|f'_{\text{cpl}}| = \alpha|f'_{\text{original}}| + (1-\alpha)I$. Since the eigenvectors are unchanged, all eigenvalues $\lambda^{(i)}_{\text{original}}$ of $|f'_{\text{original}}|$ are modified to $\lambda^{(i)}_\alpha = \alpha\lambda^{(i)}_{\text{original}} + (1-\alpha)$. This moves all eigenvalues closer to 1. Choose $\alpha > 0$ such that $\min_i \lambda^{(i)}_\alpha \geq L^{-1}$ and $\max_i \lambda^{(i)}_\alpha \leq L$ to achieve L-bi-Lipschitzness. □

Proof of theorem 5.16

The following theorem based on lemma 5.17 shows that $\Delta_{\text{affine}}[p_\theta(z)]$ is a useful measure of convergence to the standard normal distribution:

Theorem 5.16. *With the definitions from definition 5.15:*

$$p_\theta(z) = \mathcal{N}(z; 0, I) \iff \Delta_{\text{affine}}[p_\theta(z)] = 0. \tag{A.123}$$

Proof. The forward direction is trivial: $p_\theta(z) = \mathcal{N}(0, I)$ and therefore $\mathcal{D}_{\text{KL}}(p_\theta(z)\|\mathcal{N}(0, I)) = 0$. As adding an identity layer is a viable solution to equation (5.27), there is a θ_+ with $\mathcal{D}_{\text{KL}}(p_{\theta\cup\theta_+}(z)\|\mathcal{N}(0, I)) = 0$, and thus $\Delta_{\text{affine}}[p_\theta(z)] = 0$.

For the reverse direction, we have $\Delta_{\text{affine}}[p_\theta(z)] = 0$. Then, by lemma 5.17 also $\Delta^*_{\text{affine}}[p_\theta(z)] = 0$.

The maximally achievable loss improvement for any rotation Q is then given by equation (5.32):

$$\Delta^*_{\text{affine}}[p_\theta(z)] = \max_Q \frac{1}{2}\sum_{i=1}^{D/2} \mathbb{E}_b\Big[m_i(b)^2 + \sigma_i(b)^2 - 1 - \log\sigma_i(b)^2\Big] = 0. \tag{A.124}$$

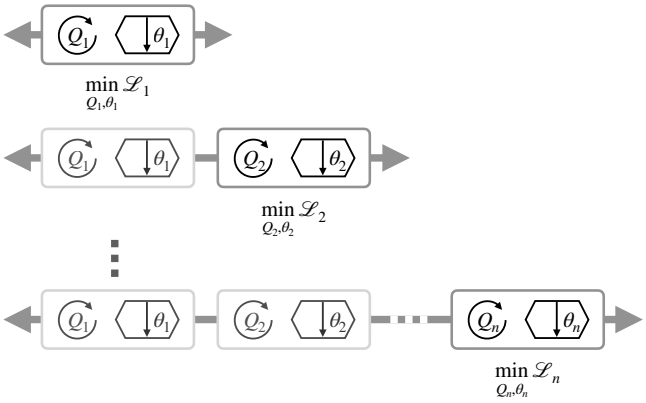


Figure A.1.: The normalizing flow we construct in our proof is remarkably simple: We iteratively add coupling blocks, optimizing the parameters of the new block while keeping previous parameters fixed. Theorem 5.16 shows that if adding another block shows no improvement in the loss, the flow has converged to a standard normal distribution in the latent space. Since the total loss that can be removed is finite, the flow converges.

It holds that both $x^2 \geq 0$ and $x^2 - 1 - \log x^2 \geq 0$. Thus, the following two summands are zero:

$$0 = \tfrac{1}{2}\mathbb{E}_b\Big[m_i(b)^2\Big], \tag{A.125}$$

$$0 = \tfrac{1}{2}\mathbb{E}_b\Big[\sigma_i(b)^2 - 1 - \log \sigma_i(b)^2\Big]. \tag{A.126}$$

This holds for all Q since the maximum over Q is zero.

By continuity of $p(b)$ and $m_1(b)$ in b, this implies for all b:

$$\mathbb{E}_{a_1|b}[a_1] = 0. \tag{A.127}$$

Fix b_1 and marginalize out the remaining dimensions $b_{2,\dots D/2}$ to compute the mean of a_1 conditioned on b_1:

$$m_{a_1|b} = \mathbb{E}_{a_1|b_1}[a_1] = \mathbb{E}_{b_{1,\dots,D/2}}[\mathbb{E}_{a_1|b}[a_1]] = \mathbb{E}_{b_{1,\dots,D/2}}[0] = 0. \tag{A.128}$$

As a_1 and b_1 are arbitrary orthogonal directions since the above is valid for any Q, we can employ theorem A.8 to follow that $p_\theta(z)$ is spherically symmetric.

We are left with showing that for a spherically symmetric $p_\theta(z)$, if for all Q there is no improvement $\Delta_{\text{affine}}(Q)$, then $p_\theta(z) = \mathcal{N}(0, I)$.

Choose $Q = I$ and write $x = (b; a)$. As $\Delta_{\text{affine}} = 0$, we can follow $\sigma_i(b) = 1$ like above. This implies that:

$$\mathbb{E}_{a|b}[\|a\|^2] = \sum_{i=1}^{D/2}(m_i(b)^2 + \sigma_i(b)^2) = D/2. \tag{A.129}$$

In particular, this is independent of b and we can thus apply theorem A.9 with $\alpha = 2$.

Finally, $m_i(b) = 0$ and $\sigma_i(b) = 1$ for all Q imply that $p_\theta(x) = \mathcal{N}(0, I)$. □

A.4.2. Proof of theorem 5.18

Theorem 5.18. *For every continuous $p(x)$ with finite first and second moment with infinite support, there is a sequence of normalizing flows $p_n(x)$ consisting of n L-bi-Lipschitz affine coupling blocks such that their latent distributions converge to the standard normal:*

$$p_n(z) \xrightarrow{n\to\infty} \mathcal{N}(z; 0, I), \tag{A.130}$$

in the sense that $\Delta_{\text{affine}}[p_n(z)] \xrightarrow{n\to\infty} 0$.

Proof. The proof idea of iteratively adding new layers which are trained without changing previous layers is visualized in figure A.1.

Let us consider a coupling-based normalizing flow of depth n and call the corresponding latent distributions $p_n(z)$, where $n = 0$ corresponds to the initial data distribution $p(x)$. Denote by $\mathcal{L}_n = \mathcal{D}_{\text{KL}}(p_n(z)\|p(z))$ the corresponding loss. Then, if we add another layer to the flow, we achieve a difference in loss of: $\Delta_{\text{affine}}[p_n(z)] = \mathcal{L}_n - \mathcal{L}_{n+1}$.

Without loss of generality, we may assume that the rotation layer Q of each block can be chosen freely. Otherwise, add 48 coupling blocks with fixed rotations that together exactly represent the Q we want, as shown by Koehler et al. (2021, Theorem 2).

We construct the blocks of the flow iteratively: Choose the rotation and subnetwork parameters φ_{n+1} of each additional block such that the block maximally reduces the loss, keeping the parameters of the previous blocks $\varphi_{1,\dots,n}$ fixed. Then, $\Delta_{\text{affine}}[p_n(z)]$ attains the value given in equation (5.28):

$$\Delta_{\text{affine}}[p_n(z)] = \mathcal{L}_n - \mathcal{L}_{n+1} = \mathcal{D}_{\text{KL}}(p_{\varphi_{1,\dots,n}}(z)\|p(z)) - \min_{\varphi_{n+1}} \mathcal{D}_{\text{KL}}(p_{\varphi_{1,\dots,n}\cup\varphi_{n+1}}(z)\|p(z)) \geq 0, \quad \text{(A.131)}$$

Each layer contributes a non-negative improvement in the loss which can at most sum up to the initial loss:

$$\sum_{k=0}^{n-1} \Delta_{\text{affine}}[p_k(z)] = \mathcal{L}_0 - \mathcal{L}_n \leq \mathcal{L}_0 \quad \text{for all } n \geq 1, \quad \text{(A.132)}$$

and the inequality is due to $\mathcal{L}_n \geq 0$. For a non-negative series that is bounded above, the terms of the series must converge to zero (Rudin, 1976, Theorems 3.14 and 3.23), which shows convergence in terms of section 5.4.3:

$$\sum_{n=0}^{\infty} \Delta_{\text{affine}}[p_n(z)] \leq \mathcal{L}_0 < \infty \Rightarrow \Delta_{\text{affine}}[p_n(z)] \to 0. \quad \text{(A.133)}$$

□

A.4.3. Relation to convergence in KL

Corollary A.10. *Given a series of probability distributions $p_n(z)$. Then, convergence in KL divergence*

$$\mathcal{D}_{KL}(p_n(z)\|\mathcal{N}(0,1)) \xrightarrow{n\to\infty} 0 \quad \text{(A.134)}$$

implies convergence in the loss improvement by a single affine coupling as in definition 5.15:

$$\Delta_{\text{affine}}[p_n(z)] \xrightarrow{n\to\infty} 0. \quad \text{(A.135)}$$

Proof. By assumption, for every $\epsilon > 0$ there exists $N \in \mathbb{N}$ such that:

$$\mathcal{D}_{\text{KL}}(p_n(z)\|\mathcal{N}(0,1)) < \epsilon \quad \forall n > N. \quad \text{(A.136)}$$

This implies convergence of $\Delta_{\text{affine}}[p_n(z)]$, by the following upper bound via the sum of all possible future improvements which is bounded from above by the total loss:

$$\Delta_{\text{affine}}[p_n(z)] \leq \sum_{m=n}^{\infty} \Delta_{\text{affine}}[p_m(z)] \leq \mathcal{D}_{\text{KL}}(p_n(z)\|\mathcal{N}(0,1)) < \epsilon \quad \forall n > N. \quad \text{(A.137)}$$

□

A.4.4. Convergence in Wasserstein but not in KL Divergence

In section 5.4.2 we argued that convergence under Wasserstein distance $W_2(p, p_n) \xrightarrow{n\to\infty} 0$ does not imply convergence under KL divergence $\mathcal{D}_{\mathrm{KL}}(p\|p_n) \xrightarrow{n\to\infty} 0$. We illustrate that via an example:

Take a standard normal target $p(x) = \mathcal{N}(0, 1)$ in 1D and approximate by a mixture of δ-distributions:

$$p_n(x) = \sum_{i=-\infty}^{\infty} p([i\, a_n, (i+1)a_n))\delta(x - a_n\, i). \tag{A.138}$$

This mixture splits the input space into bins of width a_n, and positions a δ-distribution at the left of each bin, weighted by the amount of mass in the bin in the target distribution.

The optimal transport plan underlying the Wasserstein distance redistributes the weight from the left edge of each bin over the entire bin. This means that the total distance any point has to travel under the optimal transport plan is a_n. It thus holds that $W_2(p, p_n) \leq a_n$. If we choose $a_n \to 0$, so does $W_2(p, p_n) \to 0$.

The KL divergence can be lower bounded by the total variation via Pinsker's inequality:

$$\delta_{\mathrm{TV}}(p, p_n) = \sup_{A \text{ measurable}} \Big|P(A) - P_n(A)\Big| \leq \sqrt{\frac{1}{2}\mathcal{D}_{\mathrm{KL}}(p(x)\|p_n(x))} \tag{A.139}$$

The set of all bin interiors $I = \cup_{i=-\infty}^{\infty}(i\, a_n, (i+1)a_n)$ is measurable. It holds that $P(I) = 1$ (since we only exclude the zero-set of bin edges to get I from $\mathbb{R}$). Also, $P_n(I) = 0$ since all the mass is concentrated at the bin edges in p_n, and so $1 \leq \delta_{\mathrm{TV}}(p, p_n) \leq \sqrt{\frac{1}{2}\mathcal{D}_{\mathrm{KL}}(p\|p_n)}$ regardless of a_n.

Thus, $\mathcal{D}_{\mathrm{KL}}(p\|p_n) \geq 2 > 0$ regardless of n, but $W_2(p, p_n) \to 0$.

Intuitively, the construction in Koehler et al. (2021) is related to the mixture above. The vanishing scaling terms from latent to data space in their universality proof squeeze the distribution to a thin wall of high probability. The translation terms ensure that this squeezed distribution is distributed over the space such that the error in terms of Wasserstein distance is bounded by the grid length ϵ.

A.5. Benefits of more expressive coupling blocks

This appendix is adapted from Draxler et al. (2024b).

To see what the best improvement for an infinite capacity coupling function can ever be for a fixed rotation Q, we make use of the chain rule for KL divergences in equation (A.91) and the Pythagorean identities in theorem 4.1:

$$\mathcal{L} = \mathcal{D}_{\mathrm{KL}}(p_\theta(z)\|\mathcal{N}(0, I)) = B + \mathbb{E}_{b\sim p(a,b)}\left[I(b) + \sum_{i=1}^{D/2}\Big(G_i(b) + S_i(b)\Big)\right]. \tag{A.140}$$

The symbols $B, I(b), G_i(b), S_i(b)$ all denote KL divergences:

The first two terms remain unchanged under a coupling layer: The KL divergence to the standard normal in the passive dimensions $B = \mathcal{D}_{\mathrm{KL}}(p_\theta(b)\|\mathcal{N}(0, I_{D/2}))$, which are left unchanged. The *mutual information* between active dimensions $I(b) = \mathcal{D}_{\mathrm{KL}}(p_\theta(a|b)\|p_\theta(a_1|b)\cdots p_\theta(a_{D/2}|b))$ the dependence between active dimensions conditioned on the passive vales. It is unchanged because each dimension a_i is treated conditionally independent of the others by lemma 4.3.

The remaining terms measure how far each dimension $p_\theta(a_i|b)$ differs from the standard normal: The non-Gaussianity measures the divergence to the Gaussian with the same first moments as $p_\theta(a_i|b)$: $G(b) = \mathcal{D}_{\mathrm{KL}}(p_\theta(a_i|b)\|\mathcal{N}(m_i(b), \sigma_i(b)))$. Finally, the non-Standardness measures how far each 1d Gaussian is away from the standard normal distribution: $S_i(b) = \mathcal{D}_{\mathrm{KL}}(\mathcal{N}(m_i(b), \sigma_i(b))\|\mathcal{N}(0, 1))$.

Note that the total loss $\mathcal{L}$ is invariant under a rotation of the data. The rotation does, however, affect how that loss is distributed into the different components in equation (A.140), which we have used in section 5.3.3.

If we restrict the coupling function to be affine-linear $c(a_i;\theta) = sa_i + t$ (i.e. a RealNVP coupling), then this means that also $G(b)$ is left unchanged per lemma 4.2. Only a nonlinear coupling function $c(a_i;\theta)$ can thus affect $G(b)$ and reduce it to $G(\tilde{b}) \leq G(b)$.

Taking the loss difference between two layers in both cases, we find equation (5.37).

A.6. Convergence rate of Gaussianization blocks

This appendix is adapted from Draxler et al. (2023).

A.6.1. Proof of proposition 6.1

Proposition 6.1. *Given D-dimensional data with mean m and covariance Σ and a rotation matrix Q. Then, the moments $\tilde{m}, \tilde{\Sigma}$ that can be reached by a linear Gaussianization layer as in equation (6.7) are:*

$$\tilde{m} = 0, \qquad \tilde{\Sigma}(Q) = M(Q\Sigma Q^{\mathrm{T}}). \tag{A.141}$$

This minimizes $S(\tilde{m}, \tilde{\Sigma})$ as given in equation (6.1), and G does not change.

Proof. By equation (4.37), we only need to consider the marginal losses $\mathcal{L}[p_\theta(z_i)]$, as Gaussianization does not affect the mutual information $I[p_\theta(z)]$. The marginal losses, by equation (4.35), decompose into the non-Gaussianity $G[p_\theta(z_i)]$, which is not affected by the linear layer we consider, and the one-dimensional non-Standardness $S(\tilde{m}_i, \tilde{\Sigma}_{ii} I_1)$. The latter is minimal for $\tilde{m}_i = 0$ and $\tilde{\Sigma}_{ii} = 1$. Solving $(\tilde{\Sigma})_{ii} = (\mathrm{Diag}(r) Q \Sigma Q^{\mathrm{T}} \mathrm{Diag}(r))_{ii}$ for r and then choosing u to shift the mean to zero yields the result. □

A.6.2. Proof of theorem 6.2

Theorem 6.2. *Given D-dimensional data with covariance Σ fulfilling assumption 6.1, and $S(0,\Sigma) \ll 1$. Then, the expected non-Standardness after L_{gzn} Gaussianization blocks as in proposition 6.1 can be bounded as follows:*

$$\mathbb{E}_{Q_1,\dots,Q_{L_{\mathrm{gzn}}}}[S(0, \Sigma^{(L_{\mathrm{gzn}})}(Q_1, \dots Q_{L_{\mathrm{gzn}}}))] \geq \left(1 - \frac{2}{D+2}\right)^{L_{\mathrm{gzn}}} S(0,\Sigma). \tag{A.142}$$

Proof. By proposition 6.1, the output moments read:

$$\tilde{m} = 0, \quad \tilde{\Sigma} = M(Q\Sigma Q^{\mathrm{T}}). \tag{A.143}$$

We define $M = \mathrm{Diag}(Q\Sigma Q^{\mathrm{T}})^{-1/2}$. Inserting into the loss in equation (6.10), we find:

$$\tilde{S} = -\tfrac{1}{2} \log\det \tilde{\Sigma} = -\tfrac{1}{2} \log\det(MQ\Sigma Q^{\mathrm{T}} M) = \tfrac{1}{2}(\log\det M^2 - \log\det\Sigma) \tag{A.144}$$

$$= S + \tfrac{1}{2} \log\det M^2. \tag{A.145}$$

As we consider random rotations $Q \in O(D)$, we compute the expected loss over rotations:

$$\mathbb{E}_{Q\in O(D)}[\tilde{S}] = S + \tfrac{1}{2}\mathbb{E}_{Q\in O(D)}[\log\det M^2] = S + \tfrac{1}{2}\sum_{i=1}^{D} \mathbb{E}_{Q\in O(D)}[\log(Q\Sigma Q^{\mathrm{T}})_{ii}] \tag{A.146}$$

$$\leq S + \tfrac{1}{2}\sum_{i=1}^{D} \log \mathbb{E}_{Q\in O(D)}[(Q\Sigma Q^{\mathrm{T}})_{ii}] = S. \tag{A.147}$$

Here, we have used Jensen's inequality and that $\mathbb{E}_{Q\in O(D)}[(Q\Sigma Q^{\mathrm{T}})_{ii}] = \operatorname{tr}\Sigma/D = 1$ by lemma A.11. (This is a vacuous bound: The expected loss after the layer is at least as good as before the layer.)

We now estimate Jensen gap, that is the error of Jensen's inequality, to get an estimate for a lower bound on $\mathbb{E}_{Q\in O(D)}[\tilde{S}]$. Here, we make use of the result in (Costarelli & Spigler, 2015) applied to log:

$$\mathbb{E}[\varphi(x)] - \varphi(\mathbb{E}[x]) \leq \frac{1}{2}\max_{x\in I}\varphi''(x)\operatorname{Var}[x]. \tag{A.148}$$

In our case, $\varphi(x) = -\log x$, and $x \in [\lambda_{\min}, \lambda_{\max}]$, the extremal eigenvalues of Σ (which are invariant under rotation).

Identifying terms in equation (A.267):

$$\operatorname{Var}[(Q\Sigma Q^{\mathrm{T}})_{ii}] = \frac{2}{(D+2)}\operatorname{Var}[\lambda], \tag{A.149}$$

for the variance of the eigenvalues of Σ, given by $\operatorname{Var}_i[\lambda_i]$. This means:

$$S - \tilde{S} \leq \frac{1}{2\lambda_{\min}^2}\frac{2}{(D+2)}\operatorname{Var}[\lambda]. \tag{A.150}$$

We make use of the following arithmetic mean-geometric mean (AM-GM) inequality by (Cartwright & Field, 1978):

$$\frac{\operatorname{Var}[\lambda]}{2\lambda_{\max}} \leq \bar{\lambda} - g \leq \frac{\operatorname{Var}[\lambda]}{2\lambda_{\min}}, \tag{A.151}$$

where g is the geometric mean of the eigenvalues:

$$g := \prod_{i=1}^{D} \lambda_i^{1/D}, \tag{A.152}$$

and find:

$$S - \tilde{S} \leq \frac{1}{2\lambda_{\min}^2}\frac{2}{(D+2)} 2\lambda_{\max}(1-g). \tag{A.153}$$

As $\lambda_{\max} > 1$, we can multiply the right-hand side by $\lambda_{\max}$:

$$S - \tilde{S} \leq \frac{1}{2\lambda_{\min}^2}\frac{2}{(D+2)} 2\lambda_{\max}^2(1-g). \tag{A.154}$$

Then, rewrite using the conditioning number $\kappa = \lambda_{\max}/\lambda_{\min}$:

$$S - \tilde{S} \leq \frac{2}{(D+2)}\kappa^2(1-g). \tag{A.155}$$

Note that one can write the loss S directly via g and vice versa:

$$S = -\frac{1}{2}\log g^D = -\frac{D}{2}\log g, \tag{A.156}$$

$$g = \exp(-2S/D). \tag{A.157}$$

We upper bound κ using a function of the loss (equation (A.276)):

$$\max_{\substack{\lambda_1,\dots,\lambda_D \\ \sum_i \lambda_i = D \\ \prod_i \lambda_i^{1/D} = g}} \kappa = \frac{1+\sqrt{1-g^D}}{1-\sqrt{1-g^D}}. \tag{A.158}$$

Then,

$$S - \tilde{S} \leq \frac{2}{(D+2)} \frac{1+\sqrt{1-g^D}^2}{1-\sqrt{1-g^D}} (1-g). \tag{A.159}$$

□

A.6.3. Proof of theorem 6.3

Theorem 6.3. *Under the assumptions of theorem 6.2 and assumption 6.2, and $S(0,\Sigma) \ll 1$. Then, the expected non-Standardness after L_{gzn} Gaussianization blocks as in proposition 6.1 can be bounded as follows:*

$$\mathbb{E}_{Q_1,\dots,Q_{L_{\text{gzn}}}}[S(0,\Sigma^{(L_{\text{gzn}})}(Q_1,\dots Q_{L_{\text{gzn}}}))] \geq \left(1 - \frac{2}{D+2}\right)^{L_{\text{gzn}}} S(0,\Sigma). \tag{A.160}$$

Proof. By assumption, we have the limit $S \ll 1$. Rewriting equation (6.13) in terms of S and taking the limit, we find:

$$\mathbb{E}_Q[\tilde{S}] \geq \left(1 - \frac{2}{D+2}\right) S - |\mathcal{O}(S)|. \tag{A.161}$$

Repeat over L_{gzn} blocks to obtain the statement. □

A.6.4. Proof of theorem 6.4

Theorem 6.4. *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0,\Sigma)$ under assumption 6.3. To exactly represent $p(x)$ with* random *rotations Q, at least*

$$L \geq \frac{1}{2}(D+1) \tag{A.162}$$

Gaussianization layers are required almost surely.

Proof. We need to exclude the special case that the accumulated rotation $Q^{(l)} = Q_l \cdots Q_1$ (partially) aligns with the eigenspace of Σ for some $l = 1,\dots,L$. Then, the Σ is (partially) diagonal in the input to this block l and f_{dim} can map $\Sigma \to I$ already earlier. However, this alignment has probability mass zero under random Q.

To perfectly map the input Gaussian $\mathcal{N}(0,\Sigma)$ to the latent distribution $\mathcal{N}(0,I)$, we need to learn a linear function A such that the latent covariance becomes the identity: $A^{\mathrm{T}}\Sigma A = I$. As a lower bound on how many layers we need to represent a suitable A, our learned function needs to have at least as many degrees of freedom as the covariance matrix $\Sigma \in \mathbb{R}^{D\times D}$: As it is symmetric, Σ has $D(D+1)/2$ independent degrees of freedom. The D linear single-dimensional transforms in each block $f_{\text{dim},i}$ have a total of D degrees of freedom. Thus, we need more than $D(D+1)/(2D) = (D+1)/2$ layers to represent Σ. □

A.6.5. Proof of corollary 6.5

Corollary 6.5. *Given a multivariate non-degenerate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ under assumption 6.3. To exactly represent $p(x)$ with* learned *rotations Q with $k \cdot D$ parameters each, at least*

$$L \geq \frac{1}{2(k+1)} D \tag{A.163}$$

Gaussianization layers are required almost surely.

Proof. The proof follows by replacing the number of parameters per layer in appendix A.6.4 by $(k+1)D$ instead of D. We find at least $D(D+1)/(2(k+1)D) = (D+1)/(2(k+1)) \geq D/(2(k+1))$ layers to represent Σ. □

A.6.6. Proof of corollary 6.10

Corollary 6.10. *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$. The initial loss $\mathcal{L}$ is given by equation* (6.10)*. Then, in the case $\mathcal{L} \ll 1, 1 \ll D$, the loss after L_{cpl} iterative coupling blocks with random rotations is at most:*

$$\mathbb{E}_{Q_{1\ldots L} \in O(D)}[\mathcal{L}^{(L_{\text{cpl}})}] \lesssim \left(\frac{1}{2}\right)^{L_{\text{cpl}}} \mathcal{L}. \tag{A.164}$$

Proof. We start from theorem 6.9. We first take the limit of $\gamma(\mathcal{L})$ for $\mathcal{L} \ll 1$ and then $D \gg 1$. We use the computer algebra system `sympy` to take the limits:

$$\gamma(\mathcal{L}) \xrightarrow{\mathcal{L}\to 0} \frac{D(D+2)-4}{2(D-1)(D+2)} + \mathcal{O}(\mathcal{L}) \xrightarrow{D\to\infty} \frac{1}{2} + \mathcal{O}(\mathcal{L}) + \mathcal{O}(D^{-1}). \tag{A.165}$$

□

To further justify the usage of

$$\gamma(\mathcal{L}) \xrightarrow{\mathcal{L}\to 0, D\to\infty} \frac{1}{2}, \tag{A.166}$$

note that

$$\gamma(\mathcal{L}) \xrightarrow{\mathcal{L}\to 0} \frac{D(D+2)-4}{2(D-1)(D+2)} \in \left[1/2, 5/9\right] \leq 0.555\ldots \tag{A.167}$$

A.7. Convergence rate of coupling blocks

This appendix is adapted from Draxler et al. (2022).

A.7.1. Proof of proposition 6.6

Proposition 6.6. *Given D-dimensional data with mean m and covariance Σ and a rotation matrix Q. Split the covariance of the rotated data into four blocks, corresponding to the passive and active dimensions of the coupling layer:*

$$Q\Sigma Q^{\mathrm{T}} = \begin{pmatrix} \Sigma_{bb} & \Sigma_{ba} \\ \Sigma_{ab} & \Sigma_{aa} \end{pmatrix} \tag{A.168}$$

Then, the moments $\tilde{m}, \tilde{\Sigma}$ that can be reached by a linear coupling layer as in equation (6.23) *are:*

$$\tilde{m} = 0, \qquad \tilde{\Sigma}(Q) = \begin{pmatrix} M(\Sigma_{bb}) & 0 \\ 0 & M(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}) \end{pmatrix}. \tag{A.169}$$

This minimizes the non-Standardness $S(\tilde{m}, \tilde{\Sigma})$ as given in equation (6.1), and G does not change.

Proof. We aim to find the affine-linear coupling layer f_{cpl} minimizing $S(\tilde{\Sigma})$. By lemma 4.2, the non-Gaussianity G does not change.

The affine-linear coupling has the form in equation (6.23), which we repeat here for convenience:

$$\begin{pmatrix} \tilde{b} \\ \tilde{a} \end{pmatrix} = f_{\text{cpl}}(Qx) = r \odot \begin{pmatrix} I & 0 \\ T & I \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} + u. \tag{6.23}$$

It has the form of $f_{\text{cpl}}(x) = AQx + u$ with an appropriately constrained A. To make the coupling affine-linear, $r \in \mathbb{R}_+^D$ is positive, $u \in \mathbb{R}^{D/2}$ and $T \in \mathbb{R}^{D/2 \times D/2}$ is the matrix describing the linear dependence of the active on the passive dimensions.

By linearity of expectation, the mean of $\tilde{x}$ reads:

$$\tilde{m} = AQm + u. \tag{A.170}$$

Write $M_b := \text{Diag}(r_{1\dots D/2})$ and $M_a := \text{Diag}(r_{D/2+1\dots D})$, so that the covariance of $\tilde{x}$ is given by:

$$\tilde{\Sigma} := \text{Cov}[\tilde{x}] = AQ\Sigma Q^{\mathrm{T}} A^{\mathrm{T}} \tag{A.171}$$

$$= \begin{pmatrix} M_b & 0 \\ T & M_a \end{pmatrix} \begin{pmatrix} \Sigma_{bb} & \Sigma_{ba} \\ \Sigma_{ab} & \Sigma_{aa} \end{pmatrix} \begin{pmatrix} M_b & T^{\mathrm{T}} \\ 0 & M_a \end{pmatrix} \tag{A.172}$$

$$= \begin{pmatrix} M_b \Sigma_{bb} M_b & M_b(\Sigma_{ba} M_a + \Sigma_{bb} T^{\mathrm{T}}) \\ (T\Sigma_{bb} + M_a \Sigma_{ab}) M_b & (T\Sigma_{ba} + M_a \Sigma_{aa}) M_a + (T\Sigma_{bb} + M_a \Sigma_{ab}) T^{\mathrm{T}} \end{pmatrix} \tag{A.173}$$

Together, the non-Standardness of $\tilde{x}$ is given from equation (6.1)

$$S(\tilde{m}, \tilde{\Sigma}) = \tfrac{1}{2} \left(\left\| \tilde{m} \right\|^2 + \operatorname{tr} \tilde{\Sigma} - D - \log\det \tilde{\Sigma} \right) \tag{A.174}$$

$$= \tfrac{1}{2} \Big(\left\| Am + b \right\|^2 + \operatorname{tr}(M_b^2 \Sigma_{bb}) + \operatorname{tr}(T \Sigma_{ba} M_a) + \operatorname{tr}(M_a^2 \Sigma_{aa}) + \operatorname{tr}(T \Sigma_{bb} T^{\mathrm{T}}) \tag{A.175}$$

$$+ \operatorname{tr}(M_a \Sigma_{ab} T^{\mathrm{T}}) - D - \log\det \Sigma - \log\det M_b - \log\det M_a \Big). \tag{A.176}$$

To find the minimum of $S(\tilde{m}, \tilde{\Sigma})$, minimize the above over r, T and u:

$$\arg\min_{r,T,u} S(\tilde{m}, \tilde{\Sigma}). \tag{A.177}$$

It is easy to see that $u = -Am$ minimizes equation (A.176) as in this case $\tilde{m} = 0$.

At the minimum, we find for r:

$$0 = \frac{\partial S(\tilde{m}, \tilde{\Sigma})}{\partial r_l} = -\frac{1}{r_l} + r_l (\Sigma_{bb})_{ll}, \tag{A.178}$$

for some $l = 1, \dots, D/2$. We read off that $r_l = (\Sigma_{bb})_{ll}^{-1/2}$. In matrix notation:

$$M_b = \text{Diag}(\Sigma_{bb})^{-1/2}. \tag{A.179}$$

For $r_{D/2+1\dots D}, T$, we find the system:

$$0 = \frac{\partial S(\tilde{m}, \tilde{\Sigma})}{\partial s_{n+D/2}} = -\frac{1}{s_{n+D/2}} + \sum_{j=1}^{D/2} T_{nj} (\Sigma_{ba})_{jn} + s_{n+D/2} (\Sigma_{ab})_{nn} \tag{A.180}$$

$$0 = \frac{\partial S(\tilde{m}, \tilde{\Sigma})}{\partial T_{op}} = s_p(\Sigma_{ba})_{po} + \sum_{k=1}^{D/2} T_{pk}(\Sigma_{aa})_{ko}. \tag{A.181}$$

Multiplying the first equation by $s_{n+D/2}$, we find in matrix notation:

$$I = \mathrm{Diag}(T\Sigma_{ba}M_a + M_a^2\Sigma_{aa}) \tag{A.182}$$
$$0 = M_a\Sigma_{ba} + T\Sigma_{bb}. \tag{A.183}$$

We solve the second equation for T (we use that Σ_{bb} is invertible as it is positive definite):

$$T = -M_a\Sigma_{ab}\Sigma_{bb}^{-1}, \tag{A.184}$$

and insert into the first:

$$I = \mathrm{Diag}(-M_a\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}M_a + M_a^2\Sigma_{aa}) \tag{A.185}$$
$$= \mathrm{Diag}(-M_a^2\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba} + M_a^2\Sigma_{aa}). \tag{A.186}$$

The last step is due to $\mathrm{Diag}(\cdot)$ linear and M_a diagonal. We read off that:

$$M_a = \mathrm{Diag}(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})^{-1/2}. \tag{A.187}$$

Alternative solutions with negative signs are discarded because we assumed $r > 0$, that is the diagonal of the Jacobian is positive (note that this would not have an effect on the covariance).

Inserting into equation (A.173), we find:

$$\tilde{\Sigma}_{pp} = M_b\Sigma_{bb}M_b = \mathrm{Diag}(\Sigma_{bb})^{-1/2}\Sigma_{bb}\,\mathrm{Diag}(\Sigma_{bb})^{-1/2} = M(\Sigma_{bb}), \tag{A.188}$$
$$\tilde{\Sigma}_{pa} = M_b(\Sigma_{ba}M_a + \Sigma_{bb}T^{\mathrm{T}}) = M_b(\Sigma_{ba}M_a - \Sigma_{bb}\Sigma_{bb}^{-1}\Sigma_{ba}M_a) = 0, \tag{A.189}$$
$$\tilde{\Sigma}_{ap} = \tilde{\Sigma}_{pa}^{\mathrm{T}} = 0, \tag{A.190}$$
$$\tilde{\Sigma}_{aa} = (T\Sigma_{ba} + M_a\Sigma_{aa})M_a + (T\Sigma_{bb} + M_a\Sigma_{ab})T^{\mathrm{T}} \tag{A.191}$$
$$= (-M_a\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba} + M_a\Sigma_{aa})M_a + (M_a\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{bb} - M_a\Sigma_{ab})\Sigma_{bb}^{-1}\Sigma_{ba}M_a \tag{A.192}$$
$$= M_a(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})M_a = M(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}). \tag{A.193}$$

This concludes the proof:

$$\tilde{m} = 0, \qquad \tilde{\Sigma} = \begin{pmatrix} M(\Sigma_{bb}) & 0 \\ 0 & M(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}) \end{pmatrix}. \tag{A.194}$$

□

A.7.2. Proof of theorem 6.7

The following statement will help us along the way:

Lemma A.11. *For $A \in \mathbb{C}^D$, $Q \in \{O(D), U(D)\}$ with the corresponding Haar measure $p(Q)$:*

$$\mathbb{E}_{Q\in p(Q)}[(QAQ^*)_{ii}] = \frac{1}{D}\,\mathrm{tr}(A). \tag{A.195}$$

Proof. By symmetry, $\mathbb{E}_Q[(QAQ^*)_{11}] = \mathbb{E}_Q[(QAQ^*)_{ii}]$ for $i = 1, \ldots, D$. Thus, $\mathbb{E}_Q[(QAQ^*)_{11}] = \frac{1}{D}\sum_{i=1}^{D}\mathbb{E}_Q[(QAQ^*)_{ii}] = \frac{1}{D}\mathbb{E}_Q[\mathrm{tr}(QAQ^*)] = \frac{1}{D}\,\mathrm{tr}\,A$. □

When we write Q^*, we mean conjugate transpose if Q is sampled from the unitary group $U(D)$, and transpose if Q is from the orthogonal group $O(D)$. Whenever we only consider orthogonal Q, we will resort back to writing Q^{T}.

This allows us to directly estimate $\mathbb{E}_{Q\sim p(Q)}[\log\det M_p^2]$:

Lemma A.12. *With the definitions in section 6.3.2, $p(Q)$ either the Haar measure of orthogonal or unitary matrices, and assumption 6.1. Then:*

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_p^2] \geq 0. \tag{A.196}$$

Proof. M_p is given by:

$$M_p^2 = \mathrm{Diag}(\Sigma_{bb})^{-1}. \tag{A.197}$$

The corresponding expectation value can be estimated via Jensen's inequality:

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_p^2] = \mathbb{E}_{Q\sim p(Q)}[\log\det \mathrm{Diag}(\Sigma_{bb})^{-1}] \tag{A.198}$$

$$= -\mathbb{E}_{Q\sim p(Q)}\left[\log\prod_{i=1}^{D/2}(\Sigma_{bb})_{ii}\right] = -\sum_{i=1}^{D/2}\mathbb{E}_{Q\sim p(Q)}[\log(\Sigma_{bb})_{ii}] \tag{A.199}$$

$$\geq -\sum_{i=1}^{D/2}\log\mathbb{E}_{Q\sim p(Q)}[(\Sigma_{bb})_{ii}] = -\frac{D}{2}\log(\mathrm{tr}\,\Sigma/D) \tag{A.200}$$

$$= 0. \tag{A.201}$$

By assumption 6.1, $\mathrm{tr}\,\Sigma = D$. We have used lemma A.11 for evaluating $\mathbb{E}_{Q\sim p(Q)}[(\Sigma_{bb})_{ii}]$. □

As mentioned in section 6.3.2, the main difficulty in estimating $\mathbb{E}_{Q\sim p(Q)}[S(\tilde{\Sigma}(Q))]$ lies in estimating $\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2]$. The followings show a path to do so.

Problem reformulation

In a first step, we reformulate this expectation so that it can be computed with the help of projected orbital measures (Olshanski, 2013).

We split the expectation over the Haar measure $p(Q)$ in two parts: One that defines which eigenvalues the $(D/2)\times(D/2)$ block Σ_{aa} has (denote this as Q_{ab}) and, conditioned on this, another which rotates Σ_{aa} into all possibles bases (denote this as Q_a). Formally, write Q as:

$$Q = \begin{pmatrix} I & 0 \\ 0 & Q_a \end{pmatrix} Q_{ab}. \tag{A.202}$$

We will replace the Schur complement $\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$ appearing in proposition 6.6 by the corresponding block of the precision matrix $P_0 := (Q\Sigma^{-1}Q^*)^{-1} = Q\Sigma^{-1}Q^*$ (e.g. (Horn & Johnson, 2012, Section (0.7.3)):

$$(P_{0,aa})^{-1} = ((\Sigma_0^{-1})_{aa})^{-1} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}. \tag{A.203}$$

We give more details in the proof of the following lemma, which formalizes this step:

Lemma A.13. *Given the definitions in section 6.3.2 and assumption 6.1. It holds that*

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2] \geq -\sum_{i=1}^{D/2}\log\mathbb{E}_{Q_a\sim p(Q_a|Q_{ab})}[((P_{aa})^{-1})_{ii}]. \tag{A.204}$$

Proof. By proposition 6.6, M_a^2 is given by:

$$M_a^2 = \mathrm{Diag}(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})^{-1}. \tag{A.205}$$

Being diagonal, its determinant is given by the product of its diagonal entries:

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2] = \mathbb{E}_{Q\sim p(Q)}[\log\prod_{i=1}^{D/2}(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})_{ii}^{-1}] \tag{A.206}$$

$$= \sum_{i=1}^{D/2}\mathbb{E}_{Q\sim p(Q)}[\log((\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})_{ii}^{-1})] \tag{A.207}$$

$$= -\sum_{i=1}^{D/2}\mathbb{E}_{Q\sim p(Q)}[\log((\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})_{ii})]. \tag{A.208}$$

Evaluating this expression is hard mainly because the $\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$ involves the inverse of $\Sigma_{bb} = (Q\Sigma Q^*)_{bb}$, which depends on Q.

To circumvent this, note the following property of any nonsingular matrix M (Horn & Johnson, 2012, Section (0.7.3)). Split M into blocks as:

$$M = \begin{pmatrix} A & B \\ B^* & C \end{pmatrix}, \tag{A.209}$$

and do the same for its inverse:

$$M^{-1} = \begin{pmatrix} A' & B' \\ B'^* & C' \end{pmatrix} \tag{A.210}$$

Then, $(A')^{-1} = A - BC^{-1}B^*$, which is called the Schur complement M/C. This means we can rewrite

$$\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba} = (P_{aa})^{-1}, \tag{A.211}$$

where $P_Q = \Sigma_Q^{-1}$ is the *precision matrix* of the rotated data. Given a rotation Q, it can easily be obtained from the precision matrix of the data in its original rotation:

$$\Sigma_Q = Q\Sigma Q^*, \qquad P_Q = Q\Sigma^{-1}Q^* = \begin{pmatrix} P_{bb} & P_{ba} \\ P_{ab} & P_{aa} \end{pmatrix}. \tag{A.212}$$

Inserting this, we find the expectation value:

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2] = -\sum_{i=1}^{D/2}\mathbb{E}_{Q\sim p(Q)}[\log(((P_{aa})^{-1})_{ii})]. \tag{A.213}$$

The logarithm can be drawn out via Jensen's inequality:

$$-\sum_{i=1}^{D/2}\mathbb{E}_{Q\sim p(Q)}[\log(((P_{aa})^{-1})_{ii})] \geq -\sum_{i=1}^{D/2}\log(\mathbb{E}_{Q\sim p(Q)}[((P_{aa})^{-1})_{ii}]). \tag{A.214}$$

This concludes the statement. □

Projected orbit expectation

The theory of projected orbital measures describes the distribution of eigenvalues of a randomly projected submatrix of some given matrix. Let us formalize this:

Fix a diagonal matrix $A = \mathrm{Diag}(a_1, \ldots, a_N)$. Then, then the *orbit* of A is defined as:

$$\mathcal{O}_A := \{QAQ^* : Q \in U(D)\}. \tag{A.215}$$

(The same definition also exists for orthogonal $Q \in O(D)$, but we keep it to the level we require here). All matrices in the orbit of $\mathcal{O}_A$ have the same eigenvalues.

The natural measure (probability distribution) on the orbit $\mathcal{O}_A$ is given by the image of the Haar measure on the unitary group $U(D)$. This can be thought of as the uniform measure on the group of unitary rotations. We call this measure the *orbital measure*.

We now cut out the $K \times K$ top-left corner out of every matrix in $\mathcal{O}_A$:

$$P_K\mathcal{O}_A := \{P_K Y : Y \in \mathcal{O}_A\}. \tag{A.216}$$

We call this the *projected orbit*. The matrix P_K projects a matrix to its upper-left corner:

$$P_K = (I_K; 0_{K\times(N-K)}). \tag{A.217}$$

The distribution of matrices in the projected orbit $P_K\mathcal{O}_A$ induced by the orbital measure is denoted as the *projected orbital measure* $\mu_{A,K}$. We are now interested in the eigenvalues of matrices in the projected orbit $P_K\mathcal{O}_A$.

Let spectrum be the function that assigns a matrix $Y \in \mathbb{C}^{K\times K}$ its eigenvalues $y_1, \ldots, y_K$. We will make use of a result that gives the distribution of eigenvalues of matrices in the projected orbit $P_K\mathcal{O}_A$. This is called the *radial part of the projected orbital measure* and is denoted as $\nu_{A,K}(x_1, \ldots, x_K)$.

$$\nu_{A,K}(x_1, \ldots, x_K) = \mathbb{P}_{X\sim\mu_{A,K}}[\mathrm{spectrum}(X) = (x_1, \ldots, x_K)]. \tag{A.218}$$

In other words, $\nu_{A,K}(x_1, \ldots, x_K)$ gives the probability density that a random matrix from the projected orbit of A has exactly eigenvalues $(x_1, \ldots, x_K)$. Its functional form was shown by (Olshanski, 2013):

Theorem A.14 (Radial part of projected orbital measure (Olshanski, 2013))**.** *Fix $A = (a_1, \ldots, a_D)$ with $a_1 < \cdots < a_D$. For any $K = 1, \ldots, D-1$, the density of eigenvalues of*

$$\nu_{A,K}(x_1, \ldots, x_K) = c_{D,K}\frac{V(x_1, \ldots, x_K)\det[M(a_j; x_i, \ldots, x_{D-K+i})]_{i,j=1}^{K}}{\prod_{j-i\geq D-K+1}(x_j - x_i)}. \tag{A.219}$$

Here, the constant is given by:

$$c_{D,K} = \prod_{i=1}^{K-1}\binom{D-K+i}{i}, \tag{A.220}$$

and $M(a; y_1, \ldots, y_N)$ is the B-spline:

$$M(a; y_1, \ldots, y_n) := (N-1)\sum_{i:y_i>a}\frac{(y_i - a)^{n-2}}{\prod_{r:r\neq i}(y_i - y_r)}, \tag{A.221}$$

and V is the Vandermonde polynomial:

$$V(y_1, \ldots, y_n) = \prod_{i<j}(y_j - y_i). \tag{A.222}$$

We will make use of the following variant of the Vandermonde determinant where all powers greater or equal to some k are increased by one:

Lemma A.15. *For all $n \in \mathbb{N}$, $k = 1, \dots, n-1$ and distinct $a_i, i = 1, \dots, n$:*

$$\det \begin{pmatrix} 1 & \cdots & a_1^{k-1} & a_1^{k+1} & \cdots & a_1^n \\ \vdots & & \vdots & \vdots & & \vdots \\ 1 & \cdots & a_n^{k-1} & a_n^{k+1} & \cdots & a_n^n \end{pmatrix} = V(a_1, \dots, a_n) e_{n-k}(a_1, \dots, a_n). \tag{A.223}$$

with the elementary symmetric polynomial e_K given by equation (6.30).

Lemma A.16. *Fix $A = (a_1, \dots, a_N)$ with $a_1 < \cdots < a_N$. For any $K = 1, \dots, N-1$, it holds that:*

$$\mathbb{E}_{a_1,\dots,a_K \sim \nu_{A,K}(x_1,\dots,x_K)}[x_1^{-1} + \cdots + x_K^{-1}] \tag{A.224}$$

$$= (N-K)(-1)^{N-K} \sum_{j=1}^{N} a_j^{N-K-1} \log(a_j) R(a_j; a_{\neq j}) e_{K-1}(a_{\neq j}). \tag{A.225}$$

Here, R is defined in equation (6.30).

Proof. We use the Andreief identity in the form of (Krishnaiah, 1976, Lemma 2.1)

$$\mathbb{E}_{x_1,\dots,x_k \sim \nu_{A,K}}[\tfrac{1}{x_1} + \cdots + \tfrac{1}{x_k}] \tag{A.226}$$

$$= Z^{-1} \int_{(\mathbb{R}^K)_+} (\tfrac{1}{x_1} + \cdots + \tfrac{1}{x_k}) \det(x_i^{j-1}) \det(M(x_i; a_j, \dots, a_{j+N-K})) dx_1 \cdots dx_k \tag{A.227}$$

$$= Z^{-1} \sum_{k=1}^{K} \det \int_{\mathbb{R}} x^{-\delta_{jk}} x^{j-1} M(x; a_i, \dots, a_{i+N-K}) dx \tag{A.228}$$

$$= Z^{-1} \det \int_{\mathbb{R}} x^{j-1-\delta_{j1}} M(x; a_i, \dots, a_{i+N-K}) dx \tag{A.229}$$

$$= Z^{-1} \det \begin{cases} \mu_{-1}(a_i, \dots, a_{i+N-K}) & j = 1 \\ \mu_{j-1}(a_i, \dots, a_{i+N-K}) & j > 1 \end{cases} \tag{A.230}$$

where $\mu_k(t_1, \dots, t_n)$ the kth moment of the B-spline with knots $t_1, \dots, t_n$:

$$\mu_k(t_1, \dots, t_n) = \int x^k M(x; t_1, \dots, t_n) dx. \tag{A.231}$$

We can now make use of the Hermite–Genocchi formula (Faraut, 2015, Proposition 6.3):

$$\int f^{(n-1)}(x) M(x; t_1, \dots, t_n) dx = (n-1)! f[t_1, \dots, t_n], \tag{A.232}$$

so we can rewrite

$$\mu_k(t_1, \dots, t_n) = f_k[t_1, \dots, t_n], \tag{A.233}$$

with

$$f_{-1}(x) = (n-1) x^{n-2} \log x, \tag{A.234}$$

$$f_k(x) = \binom{n+k-1}{k}^{-1} x^{n+k-1}. \tag{A.235}$$

Together, we find

$$\mathbb{E}_{x_1,\dots,x_k\sim\nu_{A,K}}[\tfrac{1}{x_1}+\cdots+\tfrac{1}{x_k}] = Z^{-1}\det(f_{i-1-\delta_{1i}}[a_j,\dots,a_{j+N-K}]). \tag{A.236}$$

The right-hand side can be identified with the right-hand side of (Faraut, 2015, Proposition 6.4). It is equal to:

$$Z^{-1}\det(f_i[a_j,\dots,a_{j+N-K}]) \tag{A.237}$$

$$= Z^{-1}\left(\prod_{0<j-i\leq N-K}(a_j-a_i)^{-1}\right)\begin{vmatrix} 1 & \cdots & 1 \\ a_1 & \cdots & a_N \\ \vdots & & \vdots \\ a_1^{N-K-1} & \cdots & a_N^{N-K-1} \\ f_1(a_1) & \cdots & f_1(a_N) \\ \vdots & & \vdots \\ f_K(a_1) & \cdots & f_K(a_N) \end{vmatrix} \tag{A.238}$$

$$= Z^{-1}\left(\prod_{0<j-i\leq N-K}(a_j-a_i)^{-1}\prod_{k=1}^{K}\binom{N-K+k}{k}^{-1}\right)(N-K)\begin{vmatrix} 1 & \cdots & 1 \\ a_1 & \cdots & a_N \\ \vdots & & \vdots \\ a_1^{N-K-1} & \cdots & a_N^{N-K-1} \\ a_1^{N-K-1}\log a_1 & \cdots & a_N^{N-K-1}\log a_N \\ a_1^{N-K+1} & \cdots & a_N^{N-K+1} \\ \vdots & & \vdots \\ a_1^{N-1} & \cdots & a_N^{N-1} \end{vmatrix} \tag{A.239}$$

$$=: C_2\det(M_{ij}) \tag{A.240}$$

Here, C_2 reduces to:

$$C_2 = \frac{N-K}{V(a_1,\dots a_n)}. \tag{A.241}$$

Then, the determinant of M_{ij} reads:

$$\det M_{ij} = \sum_{j=1}^{N}(-1)^{N-K+1+j}a_j^{N-K-1}\log(a_j)V(a_{\neq j})\sum_{\substack{i_1<\cdots<i_{K-1}\\ i_\dots\neq j}} a_{i_1}\cdots a_{i_{K-1}} \tag{A.242}$$

$$= V(a)(-1)^{N-K}\sum_j a_j^{N-K-1}\log(a_j)R(a_j;a_{\neq j})e_{K-1}(a_{\neq j}), \tag{A.243}$$

where $R(a_j;a_{\neq j})$ collects all the terms in $V(a)$ that were not contained in $V(a_{\neq j})$ up to sign:

$$R(a_j;a_{\neq j}) = \prod_{\substack{i=1\\ i\neq j}}^{n}\frac{1}{a_i-a_j} = (-1)^{j-1}V(a_{\neq j})/V(a). \tag{A.244}$$

Note that the sign of $R(a_j;a_{\neq j})$ flips from $j\to j+1$, so the alternating nature of the above series remains.

Together, the desired expectation value reads:

$$\mathbb{E}_{x_1,\dots,x_k\sim\nu_{A,K}}[\tfrac{1}{x_1}+\dots+\tfrac{1}{x_k}] \tag{A.245}$$

$$= (N-K)(-1)^{N-K}\sum_{j=1}^{N} a_j^{N-K-1}\log(a_j)R(a_j;a_{\neq j})e_{K-1}(a_{\neq j}), \tag{A.246}$$

which concludes the proof. □

We now connect this result to our situation. This paves the path from the reformulation in lemma A.13 to theorem 6.7.

Corollary A.17. *For the definitions in section 6.3.2 and when assumptions 6.3 and 6.4 are fulfilled, it holds that:*

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2] \geq \tfrac{D}{2}\log\left((-1)^{\frac{D}{2}+1}\sum_{i=1}^{D}\lambda_i^{1-\frac{D}{2}}\log(\lambda_i)R(\lambda_i^{-1};\lambda_{\neq i}^{-1})e_{\frac{D}{2}-1}(\lambda_{\neq i}^{-1})\right). \tag{A.247}$$

Proof. Lemma A.11 tells us how to integrate over Q_a.

$$\mathbb{E}_{Q_a\sim p(Q_a|Q_{ab})}[((P_{aa})^{-1})_{ii}] = \operatorname{tr}((Q_{ab}PQ_{ab}^*)^{-1}) = \sum_{i=1}^{D/2} a_i(Q_{ab})^{-1}. \tag{A.248}$$

Here, we denote by $a_i(Q_{ab})$ the ith eigenvalue of $P_0 = Q_{ab}PQ_{ab}^*$, which depends on the "outer" rotation Q_{ab}.

We substitute the expectation over Q_{ab} with an expectation over the projected eigenvalues of the rotated precision matrix:

$$\mathbb{E}_{Q_{ab}\sim p(Q)}[\mathbb{E}_{Q_a\sim p(Q_a|Q_{ab})}[((P_{aa})^{-1})_{ii}] = \operatorname{tr}((Q_{ab}PQ_{ab}^*)^{-1})] \tag{A.249}$$

$$= \mathbb{E}_{a_1,\dots,a_{D/2}\sim\nu_{A,D/2}(a_1,\dots,a_{D/2}|\lambda_1^{-1},\dots,\lambda_D^{-1})}[a_1^{-1}+\dots+a_{D/2}^{-1}]. \tag{A.250}$$

Here $X = (\lambda_1^{-1},\dots\lambda_D^{-1})$ contains the eigenvalues of the precision matrix P, the inverse of the covariance Σ. Lemma A.16 with $K = D/2$ tells us how to evaluate the above expression. Insert the result into lemma A.13 to obtain the result. □

Summary

We can now collect the above pieces to build the **proof of theorem 6.7**:

Theorem 6.7. *Given D-dimensional data with covariance Σ with eigenvalues $\lambda_1,\dots\lambda_D$. Assume that assumptions 6.1 to 6.4 hold. Then, after a single coupling block, the expected non-Standardness is bounded from above:*

$$\mathbb{E}_{Q\in U(D)}[S(\tilde{\Sigma}(Q))] < S(\Sigma) + \frac{D}{2}\log\left((-1)^{\frac{D}{2}+1}\sum_{i=1}^{D}\lambda_i^{1-\frac{D}{2}}\log(\lambda_i)R(\lambda_i^{-1};\lambda_{\neq i}^{-1})e_{\frac{D}{2}-1}(\lambda_{\neq i}^{-1})\right). \tag{A.251}$$

Here, $\lambda_{\neq i} := \{\lambda_1,\dots,\lambda_{i-1},\lambda_{i+1},\dots,\lambda_D\}$ and R, e_K are given by:

$$R(a;\{b_i\}_{i=1}^N) = \prod_{i=1}^{N}\frac{1}{a-b_i} \quad \textit{and} \quad e_K(\{b_i\}_{i=1}^N) = \sum_{0<i_1<\dots<i_K\leq N} b_{i_1}\cdots b_{i_K}. \tag{A.252}$$

Proof. Equation (6.28) is the version of the non-Standardness after a single layer when assumptions 6.1 and 6.2 are fulfilled. Insert lemma A.12 (passive part) and corollary A.17 (active part) to obtain the result. The former required assumption 6.1 and the latter assumptions 6.3 and 6.4 to hold. □

Handling of imaginary part

If we allow for unitary rotations $Q \in U(D)$, real-valued data is typically rotated into imaginary space. In fact, the case that the input remains real even has probability zero:

$$\mathbb{P}[Qx \in \mathbb{R}^D] = 0. \tag{A.253}$$

This does not pose a problem for our theory: The covariance matrix is positive definite also for complex data and so it has a positive determinant and trace, which are the only quantities entering the non-Standardness S (see equation (6.1)).

A.7.3. Proof of theorem 6.8

Lemma A.18. *With the definitions in section 6.3.2 and $p(Q)$ the Haar measure over the orthogonal group $O(D)$:*

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2] \geq D/2 \log\left(1 - \frac{DD}{2(D+1)(D-1)}\frac{\mathrm{Var}[\lambda]}{\lambda_{\max}}\right). \tag{A.254}$$

Proof. By proposition 6.6, M_a^2 is given by:

$$M_a^2 = \mathrm{Diag}(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})^{-1}. \tag{A.255}$$

The determinant of a diagonal matrix is equal to the product of the entries on the diagonal. By the permutation symmetry of $p(Q)$, we can pick the entry in the upper-left corner:

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2] = D/2\mathbb{E}_{Q\sim p(Q)}[\log((M_a^{-2})_{11})] \leq -D/2 \log \mathbb{E}_{Q\sim p(Q)}[(M_a^2)_{11}]. \tag{A.256}$$

The last step is due to the Jensen inequality.

We are left with computing $\mathbb{E}_{Q\sim p(Q)}[(M_a^2)_{11}]$:

$$\begin{aligned}
\mathbb{E}_{Q\sim p(Q)}[(M_a^2)_{11}] &= \mathbb{E}_{Q\sim p(Q)}[(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})_{11}] && \text{(A.257)}\\
&= \mathbb{E}_{Q\sim p(Q)}[(\Sigma_{aa})_{11}] - \mathbb{E}_{Q\sim p(Q)}[(\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})_{11}] && \text{(A.258)}\\
&= \frac{1}{D}\operatorname{tr}\Sigma - \mathbb{E}_{Q\sim p(Q)}[(\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})_{11}]. && \text{(A.259)}
\end{aligned}$$

The first expectation can be exactly computed via lemma A.11.

The average trace of the second matrix is not so easy to evaluate. As Σ_{bb}^{-1} is positive definite, we can replace it with the worst case in the expectation:

$$(\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})_{11} \geq (\Sigma_{ab}\lambda_{\max}^{-1} I \Sigma_{ba})_{11} = (\Sigma_{ab}\Sigma_{ba})_{11}\lambda_{\max}^{-1}. \tag{A.260}$$

$\lambda_{\max}$ is the largest eigenvalue of Σ, which does not depend on Q.

The expectation value can now be computed exactly:

$$\begin{aligned}
\mathbb{E}_{Q\sim p(Q)}[(\Sigma_{ab}\Sigma_{ba})_{11}] &= \sum_{i=1}^{D/2} \mathbb{E}_{Q\sim p(Q)}[(\Sigma_{ab})_{1i}(\Sigma_{ba})_{i1}] && \text{(A.261)}\\
&= D/2\mathbb{E}_{Q\sim p(Q)}[(\Sigma_{ab})_{11}^2]. && \text{(A.262)}
\end{aligned}$$

The last step is because each summand will have the same contribution. Writing the matrix multiplication out explicitly:

$$(\Sigma_{ab})_{11}^2 = (Q\,\mathrm{Diag}(\lambda_1,\ldots,\lambda_D)Q^{\mathrm{T}})_{11}^2 = \left(\sum_{j=1}^{D} Q_{1j}\lambda_j Q_{(D/2+1)j}\right)^2 \tag{A.263}$$

Again by symmetry, we can exchange axes and write 2 instead of $D/2+1$ in what follows:

$$\left(\sum_{j=1}^{D} Q_{1j}\lambda_j Q_{(D/2+1)j}\right)^2 = \left(\sum_{j=1}^{D} Q_{1j}\lambda_j Q_{2j}\right)^2 = \sum_{j,k=1}^{D} \lambda_j\lambda_k Q_{1j}Q_{2j}Q_{1k}Q_{2k}. \tag{A.264}$$

Taking the expectation, we use the linearity of the expectation and are left with the following monomials of elements of Q:

1. $j=k$: $\mathbb{E}_{Q\sim p(Q)}[Q_{1j}^2 Q_{2j}^2] = \mathbb{E}_{Q\sim p(Q)}[Q_{11}^2 Q_{21}^2]$ as we can exchange axes,
2. $j\neq k$: $\mathbb{E}_{Q\sim p(Q)}[Q_{1j}Q_{2j}Q_{1k}Q_{2k}] = \mathbb{E}_{Q\sim p(Q)}[Q_{11}Q_{21}Q_{12}Q_{22}]$ as we can exchange axes.

By (Gorin, 2002), these amount to the following integrals of monomials of entries of random orthogonal matrices and the corresponding values:

$$1.\ \left\langle 2 \quad 2 \right\rangle = \frac{1}{D(D+2)}, \tag{A.265}$$

$$2.\ \left\langle \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \right\rangle = -\frac{1}{D(D-1)(D+2)}. \tag{A.266}$$

Together, we find:

$$\mathbb{E}_{Q\sim p(Q)}[(M_a^2)_{11}] = 1 - \frac{1}{2(D+2)\lambda_{\max}}\left(\sum_{j=1}^{D}\lambda_j^2 - \frac{1}{D-1}\sum_{j\neq k}\lambda_j\lambda_k\right) \tag{A.267}$$

$$= 1 - \frac{D^2}{2(D-1)(D+2)}\frac{\mathrm{Var}[\lambda]}{\lambda_{\max}}. \tag{A.268}$$

Here, $\mathrm{Var}[\lambda] = \frac{1}{D}\operatorname{tr}\Sigma^2 - (\frac{1}{D}\operatorname{tr}\Sigma)^2$ is the variance of the eigenvalues of Σ.

Insert this to obtain the result. □

Lemma A.19. *With the definitions in section 6.3.2:*

$$\mathbb{E}_{Q\sim p(Q)}[\log\det M_a^2] \geq D/2\log\left(1 - \frac{DD}{2(D+1)(D-1)}\frac{\mathrm{Var}[\lambda]}{\lambda_{\max}}\right). \tag{A.269}$$

Proof. The idea is to lower bound

$$\frac{\mathrm{Var}[\lambda]}{\lambda_{\max}} \tag{A.270}$$

by some function of L. We make use of following arithmetic mean-geometric mean (AM-GM) inequality by (Cartwright & Field, 1978):

$$\frac{\mathrm{Var}[\lambda]}{2\lambda_{\max}} \leq \bar{\lambda} - g \leq \frac{\mathrm{Var}[\lambda]}{2\lambda_{\min}}, \tag{A.271}$$

where g is the geometric mean of the eigenvalues:

$$g := \left(\prod_{i=1}^{D}\lambda_i\right)^{1/D}. \tag{A.272}$$

We can write the loss L directly via g and vice versa:

$$L = -\frac{1}{2}\log g^D = -\frac{D}{2}\log g, \tag{A.273}$$

$$g = \exp(-2L/D). \tag{A.274}$$

Rewrite equation (A.271) to our needs:

$$\frac{\mathrm{Var}[\lambda]}{\lambda_{\max}} = \frac{\mathrm{Var}[\lambda]\lambda_{\min}}{\lambda_{\max}\lambda_{\min}} = \frac{2}{\kappa}\frac{\mathrm{Var}[\lambda]}{2\lambda_{\min}} \geq \frac{2}{\kappa}(1-g), \tag{A.275}$$

with κ the condition number of the covariance Σ.

As we want a bound that merely depends on the loss, we upper bound κ using a function of the loss, yielding a lower bound on $\mathrm{Var}[\lambda]/\lambda_{\max}$ that merely depends on the loss. The maximum of the condition value is given by:

$$\max_{\substack{\lambda_1,\dots,\lambda_D \\ \sum_i \lambda_i = D \\ \prod_i \lambda_i^{1/D} = g}} \kappa = \frac{1+\sqrt{1-g^D}}{1-\sqrt{1-g^D}}. \tag{A.276}$$

This yields the required lower bound:

$$\frac{\mathrm{Var}[\lambda]}{\lambda_{\max}} \geq 2\frac{1-\sqrt{1-g^D}}{1+\sqrt{1-g^D}}(1-g), \tag{A.277}$$

which results in an overall upper bound:

$$\mathbb{E}_{Q\in O(D)}[S(\tilde{\Sigma}(Q))] \leq S(\Sigma) + \frac{D}{4}\log\left(1 - \frac{D^2}{(D-1)(D+2)}\frac{1-\sqrt{1-g^D}}{1+\sqrt{1-g^D}}(1-g)\right). \tag{A.278}$$

Replacing the expression in equation (A.274) for g yields the statement. □

We summarize to obtain the **proof of theorem 6.8**:

Theorem 6.8. *Given D-dimensional data with covariance Σ fulfilling assumption 6.1 with covariance eigenvalues $\lambda_1, \dots \lambda_D$. Then, after a single coupling block, the expected loss can be bounded from above:*

$$\mathbb{E}_{Q\in O(D)}[S(0,\tilde{\Sigma}(Q))] \leq S(0,\Sigma) + \frac{D}{4}\log\left(1 - \frac{D^2}{2(D-1)(D+2)}\frac{\mathrm{Var}[\lambda]}{\lambda_{\max}}\right) \tag{A.279}$$

$$\leq S(0,\Sigma) + \frac{D}{4}\log\left(1 - \frac{D^2}{(D-1)(D+2)}\frac{1-\sqrt{1-g^D}}{1+\sqrt{1-g^D}}(1-g)\right) \tag{A.280}$$

$$< S(m,\Sigma). \tag{A.281}$$

Here, g is the geometric mean of the eigenvalues: $g = \prod_{i=1}^D \lambda_i^{1/D} = \exp(-2S(0,\Sigma)/D) < 1$ which is a bijection of $S(0,\Sigma)$.

Proof. Equation (6.28) is the form of non-Standardness $S(\tilde{\Sigma})$ (see equation (6.1)) we need to evaluate when assumptions 6.1 and 6.2 hold. Into this equation, insert lemma A.12 together with lemma A.18 for the first bound. For the second bound, insert lemmas A.12 and A.19. □

A.7.4. Proof of theorem 6.9

Theorem 6.9. *Given D-dimensional data fulfilling assumptions 6.1 and 6.2 with covariance Σ. Then, after L_{cpl} coupling blocks, the expected loss is smaller than:*

$$\mathbb{E}_{Q_1,\ldots,Q_{L_{\text{cpl}}}\in O(D)}[S(\Sigma_{L_{\text{cpl}}})] \leq \gamma(S(\Sigma))^{L_{\text{cpl}}} S(\Sigma), \tag{A.282}$$

where the convergence rate depends on the non-Standardness before training:

$$\gamma(S) = 1 + \frac{1}{4S/D} \log\left(1 - \frac{D^2}{(D-1)(D+2)} \frac{1-\sqrt{1-g(S)^D}}{1+\sqrt{1-g(S)^D}}\Big(1-g(S)\Big)\right) < 1. \tag{A.283}$$

Proof. The non-Standardness will not increase by the action of a single layer given in proposition 6.6 (compare equation (6.27)). This holds regardless of the rotations of the individual blocks $Q_1, \ldots Q_{L_{\text{cpl}}}$, so $S(\Sigma) \geq S(\Sigma_1) \geq \cdots \geq S(\Sigma_{L_{\text{cpl}}})$. It is easy to see that γ decreases as S decreases by using $S > 0$ to check that

$$\frac{\partial \gamma}{\partial S} > 0. \tag{A.284}$$

Together, we have:

$$\gamma\Big(S(\Sigma_{L_{\text{cpl}}-1})\Big) \leq \cdots \leq \gamma\Big(S(\Sigma)\Big). \tag{A.285}$$

Rewrite theorem 6.8 as follows:

$$\mathbb{E}_{Q\in O(D)}[S(\Sigma_1(Q))] \leq \gamma\Big(S(\Sigma)\Big) S(\Sigma), \tag{A.286}$$

and apply repeatedly:

$$\begin{aligned}
\mathbb{E}_{Q_1,\ldots,Q_{L_{\text{cpl}}}\in O(D)}[S(\Sigma_{L_{\text{cpl}}})] &\leq \mathbb{E}_{Q_1,\ldots,Q_{L_{\text{cpl}}-1}\in O(D)}[\gamma(S(\Sigma_{L_{\text{cpl}}-1}))S(\Sigma_{L_{\text{cpl}}-1})] && \text{(A.287)}\\
&\leq \gamma(S(\Sigma))\mathbb{E}_{Q_1,\ldots,Q_{L_{\text{cpl}}-1}\in O(D)}[S(\Sigma_{L_{\text{cpl}}-1})] && \text{(A.288)}\\
&\leq \cdots \leq \gamma(S(\Sigma))^{L_{\text{cpl}}} S(\Sigma) && \text{(A.289)}
\end{aligned}$$

This shows the statement. □

A.7.5. Proof of corollary 6.11

This proof was published in (Draxler et al., 2023).

Corollary 6.11. *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ under assumption 6.3. To exactly represent $p(x)$, at least*

$$L_{\text{cpl}} \geq 2 \in \Omega(1) \tag{A.290}$$

coupling blocks with random rotations $Q \in O(D)$ are required almost surely.

Proof. The proof follows by replacing the number of parameters per layer in appendix A.6.4 by $(D/2)^2 + D/2$ instead of D. We find at least $2 > D(D+1)/(2((D/2)^2 + D/2)) = 2(D+1)/(D+2) > 1$ layers to represent Σ. □

A.8. Free-form flows

This appendix is adapted from Draxler et al. (2024a).

Theorem 7.1. *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$ be a diffeomorphism parameterized by θ. Then, for all $x \in \mathbb{R}^D$:*

$$\nabla_\theta \log |f'_\theta(x)| = \operatorname{tr}\left((\nabla_\theta f'_\theta(x))(f_\theta^{-1\prime}(x))\right) = \mathbb{E}_v\left[v^{\mathrm{T}}(\nabla_\theta f'_\theta(x))(f_\theta^{-1\prime}(x))v\right]. \tag{A.291}$$

Proof. Jacobi's formula states that, for a matrix $A(t)$ parameterized by t, the derivative of the determinant is

$$\frac{\mathrm{d}}{\mathrm{d}t}|A(t)| = |A(t)| \operatorname{tr}\left(A(t)^{-1}\frac{\mathrm{d}A(t)}{\mathrm{d}t}\right) \tag{A.292}$$

and hence

$$\frac{\mathrm{d}}{\mathrm{d}t}\log|A(t)| = |A(t)|^{-1}\frac{\mathrm{d}}{\mathrm{d}t}|A(t)| = \operatorname{tr}\left(A(t)^{-1}\frac{\mathrm{d}A(t)}{\mathrm{d}t}\right) = \operatorname{tr}\left(\frac{\mathrm{d}A(t)}{\mathrm{d}t}A(t)^{-1}\right) \tag{A.293}$$

using the cyclic property of the trace in the last step. Applying this formula with $A = f'_\theta(x)$ and $t = \theta$, and noting that $(f'_\theta(x))^{-1} = (f_\theta^{-1})'(x)$ for a diffeomorphism yields the result. □

A.9. Free-form flows on Riemannian manifolds

This appendix is adapted from Sorrenson et al. (2023).

In this appendix, we will focus on intuitive definitions of concepts from topology and differential geometry. For a more rigorous treatment of these concepts, see (Jost, 2008).

An n-dimensional manifold $\mathcal{M}$ is a space where every point x has a neighborhood which is homeomorphic to an open subset of $\mathbb{R}^n$. Intuitively, this means that there is a small region of $\mathcal{M}$ containing x which can be bent and stretched in a continuous way to map onto a small region in $\mathbb{R}^n$. This is what is meant when we say that the manifold locally resembles $\mathbb{R}^n$. If all these maps from $\mathcal{M}$ to $\mathbb{R}^n$ are also differentiable, then the manifold itself is differentiable, as long as there is a way to connect the local neighborhoods in a differentiable and consistent way.

The tangent space of the manifold at x, denoted $\mathcal{T}_x\mathcal{M}$, is an n-dimensional Euclidean space, which is a linearization of the manifold at x: if we zoom in to a very small region around x the manifold looks flat, and this flat Euclidean space is aligned with the tangent space. Because the tangent space is a linearization of the manifold, this is where derivatives on the manifold live, e.g. if $f_\theta : \mathcal{M}_X \to \mathcal{M}_Z$ is a map between two manifolds, then the Jacobian $f'_\theta(x)$ is a linear map from $\mathcal{T}_x\mathcal{M}_X$ to $\mathcal{T}_{f_\theta(x)}\mathcal{M}_Z$.

A Riemannian manifold $(\mathcal{M}, G)$ is a differentiable manifold which is equipped with a Riemannian metric $G : \mathcal{T}_x\mathcal{M} \times \mathcal{T}_x\mathcal{M} \to \mathbb{R}$ which defines an inner product on the tangent space, which allows us to calculate lengths and angles in this space. The length of a smooth curve $\gamma : [0, 1] \to \mathcal{M}$ is given by the integral of the length of its velocity vector $\gamma'(t) \in \mathcal{T}_{\gamma(t)}\mathcal{M}$. This ultimately allows us to define a notion of distance on the manifold, as the curve of minimal length connecting two points.

In the remainder of the appendix, we only consider Riemannian manifolds.

A.9.1. Manifold change of variables

Embedded manifolds We define an n-dimensional manifold embedded in $\mathbb{R}^m$ via a projection function

$$\mathrm{proj} : \mathbb{P} \to \mathbb{R}^m \tag{A.294}$$

where $\mathbb{P} \subseteq \mathbb{R}^m$ is the projectable set. We require the projection to have the following properties (the first is true of all projections, the others are additional requirements):

1. $\text{proj} \circ \text{proj} = \text{proj}$
2. proj is smooth on $\mathbb{P}$
3. $\text{rank}(\text{proj}'(\text{proj}(x))) = n$ for all $x \in \mathbb{P}$

Given such a projection, we define a manifold by

$$\mathcal{M} = \{x \in \mathbb{R}^m : \text{proj}(x) = x\} \tag{A.295}$$

with the tangent space

$$\mathcal{T}_x\mathcal{M} = \text{col}(\text{proj}'(x)) \tag{A.296}$$

where col denotes the column space. Since the rank of $\text{proj}'(x)$ with $x \in \mathcal{M}$ is n, the tangent space is n-dimensional and $\mathcal{M}$ is an n-dimensional manifold. To avoid clutter, we denote the Riemannian metric and its $m \times m$ matrix representation with G interchangeably. If $\mathcal{M}$ is isometrically embedded, then $G(x)$ is just the identity matrix.

The Jacobian of the projection is a projection matrix, meaning $\text{proj}'(x)\,\text{proj}'(x) = \text{proj}'(x)$ for $x \in \mathcal{M}$. For any v in the column space of $\text{proj}'(x)$, there is a u such that $v = \text{proj}'(x)u$ and due to the projection property, $\text{proj}'(x)v = \text{proj}'(x)u = v$. Similarly, for any w in the row space of $\text{proj}'(x)$, $w\,\text{proj}'(x) = w$. If proj is an orthogonal projection, proj' is symmetric by definition and hence the row and column spaces are identical.

Integration on embedded manifolds In order to perform integration on the manifold, we cannot work directly in the m-dimensional coordinates of the embedding space, instead we have to introduce some local n-dimensional coordinates. This means that the domain of integration has to be diffeomorphic to an open set in $\mathbb{R}^n$. Since this might not be the case for the whole region of integration, we might need to partition it into such regions and perform integration on each individually (each such region, together with its map to $\mathbb{R}^n$, is known as a chart and a collection of charts is an atlas). For example, if we want to integrate a function on the sphere, we could split the sphere into two hemispheres and integrate each separately. A hemisphere can be continuously stretched and flattened into a 2-dimensional region, whereas the whole sphere cannot without creating discontinuities.

Given an open set U in $\mathbb{R}^n$, and a diffeomorphic local embedding function $\varphi : U \to \mathcal{M}$, the integral of a scalar function $p : \mathcal{M} \to \mathbb{R}$ on $\varphi(U) \subseteq \mathcal{M}$ is

$$\int_{\varphi(U)} p dV = \int_U (p \circ \varphi)\sqrt{|\varphi'(u)^{\mathrm{T}} G(\varphi(u))\varphi'(u)|}du^1 \cdots du^n. \tag{A.297}$$

The integral on the right is an ordinary integral in $\mathbb{R}^n$. The quantity inside the determinant is known as the pullback metric.

Below, we generalize theorem 7.2 to the case where x- and z-space may have different manifolds and metrics:

Theorem A.20 (Manifold change of variables)**.** *Let $(\mathcal{M}_X, G_X)$ and $(\mathcal{M}_Z, G_Z)$ be n-dimensional Riemannian manifolds embedded in $\mathbb{R}^m$, i.e., $\mathcal{M}_X, \mathcal{M}_Z \subseteq \mathbb{R}^m$, and assume they have the same global topological structure. Let $p_\theta(x)$ be a probability distribution on $\mathcal{M}_X$ and let $f_\theta : \mathcal{M}_X \to \mathcal{M}_Z$ be a diffeomorphism. Let $p(z)$ be the pushforward of $p_\theta(x)$ under f_θ.*

Let $x \in \mathcal{M}_X$. Define $Q \in \mathbb{R}^{m \times n}$ as an orthonormal basis for $\mathcal{T}_x\mathcal{M}_X$ and $R \in \mathbb{R}^{m \times n}$ as an orthonormal basis for $\mathcal{T}_{f_\theta(x)}\mathcal{M}_Z$.

Then, the probability densities $p_\theta(x)$ and $p(z)$ are related under the change of variables $x \mapsto f_\theta(x)$ by the following equation:

$$p_\theta(x) = p(z = f_\theta(x))|R^{\mathrm{T}} f'_\theta(x) Q| \sqrt{\frac{|R^{\mathrm{T}} G_Z(f_\theta(x))R|}{|Q^{\mathrm{T}} G_X(x) Q|}}. \tag{A.298}$$

where Q and R depend on x and $f_\theta(x)$, respectively, although this dependency is omitted for brevity.

Below, we provide two versions of the proof, the second being a less rigorous and more geometric variant of the first.

Proof. Let $\varphi : \mathbb{R}^n \to \mathcal{M}_X$ be defined by $\varphi(u) = \mathrm{proj}_X(x + Qu)$. Let U be an open subset of $\mathbb{R}^n$ containing the origin which is small enough so that φ is bijective. Let $\psi : \mathbb{R}^n \to \mathcal{M}_Z$ be defined by $\psi(w) = \mathrm{proj}_Z(f_\theta(x) + Rw)$. Define $\varphi = \psi^{-1} \circ f_\theta \circ \varphi$ and let $W = \varphi(U)$.

Note that $\varphi'(u) = \mathrm{proj}'_X(x + Qu) \cdot Q$ and hence $\varphi'(0) = \mathrm{proj}'_X(x)Q = Q$ (since each column of Q is in $\mathcal{T}_x\mathcal{M}_X = \mathrm{col}(\mathrm{proj}'_X(x))$).

Similarly, $\psi'(0) = R$. Since ψ is a map from n to m dimensions, there is not a unique function from $\mathbb{R}^m$ to $\mathbb{R}^n$ which is ψ^{-1} on the manifold and there are remaining degrees of freedom in the off-manifold behavior which can result in different Jacobians. For our purposes, we define the inverse ψ^{-1} such that $\psi \circ \psi^{-1}$ is an orthogonal projection onto $\mathcal{M}_Z$. This means $\psi'(\psi^{-1}(f_\theta(x)))(\psi^{-1})'(f_\theta(x)) = RR^{\mathrm{T}}$ and hence $(\psi^{-1})'(f_\theta(x)) = R^{\mathrm{T}}$.

Since $p(z)$ is the pushforward of $p_\theta(x)$ under f_θ, the amount of probability mass contained in $\varphi(U)$ is the same as that contained in $f_\theta(\varphi(U)) = \psi(W)$:

$$\int_{\varphi(U)} p_\theta(x) dV_X = \int_{\psi(W)} p(z) dV_Z \tag{A.299}$$

and therefore:

$$\int_U p_\theta(\varphi(u)) \sqrt{|\varphi'(u)^{\mathrm{T}} G_X(\varphi(u))\varphi'(u)|} du^1 \cdots du^n = \int_W p(\psi(w)) \sqrt{|\psi'(w)^{\mathrm{T}} G_Z(\psi(w))\psi'(w)|} dw^1 \cdots dw^n. \tag{A.300}$$

Changing variables of the RHS with $w = \varphi(u)$ gives us

$$\int_U p_\theta(\varphi(u)) \sqrt{|\varphi'(u)^{\mathrm{T}} G_X(\varphi(u))\varphi'(u)|} du^1 \cdots du^n = \int_U p(f_\theta(\varphi(u))) \sqrt{|\psi'(\varphi(u))^{\mathrm{T}} G_Z(f_\theta(\varphi(u)))\psi'(\varphi(u))|} \cdot \left|\frac{\partial w}{\partial u}\right| du^1 \cdots du^n. \tag{A.301}$$

Since U was arbitrary, we can make it arbitrarily small, demonstrating that the integrands must be equal for $u = 0$:

$$p_\theta(x) \sqrt{|Q^{\mathrm{T}} G_X(x) Q|} = p(f_\theta(x)) \sqrt{|R^{\mathrm{T}} G_Z(f_\theta(x))R|} \cdot \left|\frac{\partial w}{\partial u}\right|. \tag{A.302}$$

Since $w = \psi^{-1}(f_\theta(\varphi(u)))$, the Jacobian has the following form when evaluated at the origin (note $\varphi(0) = x$):

$$\frac{\partial w}{\partial u} = (\psi^{-1})'(f_\theta(x)) \cdot f'_\theta(x) \cdot \varphi'(0) \tag{A.303}$$

$$= R^{\mathrm{T}} f'_\theta(x) Q. \tag{A.304}$$

Substituting this into the equality, rearranging and taking the logarithm gives the result:

$$\log p_\theta(x) = \log p(f_\theta(x)) + \log |R^{\mathrm{T}} f'_\theta(x) Q| + \tfrac{1}{2} \log \frac{|R^{\mathrm{T}} G_Z(f_\theta(x)) R|}{|Q^{\mathrm{T}} G_X(x) Q|}. \tag{A.305}$$

□

Alternative proof Here is a less rigorous and more geometric proof, which may be more intuitive for some readers.

Proof. Let x be a point on $\mathcal{M}_X$. Consider a small square region $U \subseteq \mathcal{M}$ around x (hypercubic region in higher dimensions). If the sides of the square are small enough, the square is approximately tangent to the manifold since the manifold looks very flat if we zoom in. Suppose Q is a basis for the tangent space at x and $q^1, \ldots, q^n$ are the columns of Q. Suppose that the sides of the square (or hypercube) are spanned by $u^i = \epsilon q^i$ for a small ϵ. The volume spanned by a parallelotope (higher-dimensional analog of a parallelogram) is the square root of the determinant of the Gram matrix of inner products:

$$\mathrm{vol}(u^1, \ldots, u^n) = \sqrt{|\langle u^i, u^j \rangle|}. \tag{A.306}$$

The inner product is given by G, namely $\langle u, v \rangle = u^{\mathrm{T}} G v$. We can therefore write the volume of U as

$$\mathrm{vol}(U) \approx \epsilon^n \sqrt{|Q^{\mathrm{T}} G Q|}. \tag{A.307}$$

Now consider how U is transformed under f_θ. It will be mapped to a region $f_\theta(U)$ on $\mathcal{M}_z$ with approximately straight edges, forming an approximate parallelotope in the tangent space at $z = f_\theta(x)$. This region will be spanned by the columns of $f'_\theta(x)\epsilon Q$ (since $f_\theta(x + u^i) \approx f_\theta(x) + f'_\theta(x) u^i$) and hence will have a volume of

$$\mathrm{vol}(f_\theta(U)) \approx \epsilon^n \sqrt{|Q^{\mathrm{T}} f'_\theta(x)^{\mathrm{T}} G_Z(z) f'_\theta(x) Q|} \tag{A.308}$$

$$= \epsilon^n \sqrt{|Q^{\mathrm{T}} f'_\theta(x)^{\mathrm{T}} R R^{\mathrm{T}} G_Z(z) R R^{\mathrm{T}} f'_\theta(x) Q|} \tag{A.309}$$

$$= \epsilon^n |R^{\mathrm{T}} f'_\theta(x) Q| \sqrt{|R^{\mathrm{T}} G_Z(z) R|} \tag{A.310}$$

where R is a basis for the tangent space at $f_\theta(x)$. We can introduce RR^{T} into the expression since it is a projection in the tangent space at $f_\theta(x)$ and is essentially the identity within that space. Since the RHS of $G_Z(z)$ and the LHS of $f'_\theta(x)$ both live in this tangent space, we can introduce RR^{T} between them without changing the expression. Then in the last step, we use that $|AB| = |A||B|$ for square matrices.

The probability density in U and $f_\theta(U)$ should be roughly constant since both regions are very small. Since the probability mass in both regions should be the same, we can write

$$p_\theta(x) \,\mathrm{vol}(U) \approx p(f_\theta(x)) \,\mathrm{vol}(f_\theta(U)) \tag{A.311}$$

and therefore

$$p_\theta(x) = p(f_\theta(x)) |R^{\mathrm{T}} f'_\theta(x) Q| \frac{\sqrt{|R^{\mathrm{T}} G_Z(f_\theta(x)) R|}}{\sqrt{|Q^{\mathrm{T}} G_X(x) Q|}} \tag{A.312}$$

where the approximation becomes exact by taking the limit of infinitesimally small ϵ. Taking the logarithm, we arrive at the result of theorem 7.2. □

A.9.2. Loss function

We prove theorem 7.3 in the slightly more general variant of considering potentially different manifolds in x- and z-space, just like theorem A.20.

Theorem A.21. *Under the assumptions of theorem A.20. Let $v \in \mathbb{R}^m$ be a random variable with zero mean and covariance RR^{T}. Then, the derivative of the change of variables term has the following trace expression, where $z = f_\theta(x)$:*

$$\nabla_\theta \log |R^{\mathrm{T}} f'_\theta(x) Q| = \operatorname{tr}(R^{\mathrm{T}} (\nabla_\theta f'_\theta(x)) f_\theta^{-1\prime}(z) R) \tag{A.313}$$

$$= \mathbb{E}_v \left[v^{\mathrm{T}} (\nabla_\theta f'_\theta(x)) f_\theta^{-1\prime}(z) v \right]. \tag{A.314}$$

Proof. For brevity, we drop the index θ and denote $g = f^{-1}$. First, a reminder that $\varphi'(u) = R^{\mathrm{T}} f'(x) Q$ with $\varphi = \psi^{-1} \circ f \circ \varphi$. Let $\chi = \varphi^{-1}$, i.e. $\chi = \varphi^{-1} \circ g \circ \psi$. Jacobi's formula tells us that:

$$\frac{d}{dt} \log |A(t)| = \operatorname{tr}\left(\frac{dA(t)}{dt} A(t)^{-1} \right). \tag{A.315}$$

Note also that since $\chi(\varphi(u)) = u$, therefore $\chi'(\varphi(u))\varphi'(u) = I$ and $\chi'(\varphi(u)) = \varphi'(u)^{-1}$. Applying Jacobi's formula to $\varphi'(u)$:

$$\nabla_\theta \log |\varphi'(u)| = \operatorname{tr}((\nabla_\theta \varphi'(u)) \varphi'(u)^{-1}) \tag{A.316}$$

$$= \operatorname{tr}((\nabla_\theta \varphi'(u)) \chi'(\varphi(u))) \tag{A.317}$$

and substituting in f and g:

$$\nabla_\theta \log |R^{\mathrm{T}} f'(x) Q| = \operatorname{tr}(\nabla_\theta (R^{\mathrm{T}} f'(x) Q) Q^{\mathrm{T}} g'(f(x)) R). \tag{A.318}$$

Q does not depend on θ, but R depends on $f(x)$ and hence θ, so it must be considered in the derivative. However,

$$\nabla_\theta \operatorname{tr}(RR^{\mathrm{T}}) = \operatorname{tr}((\nabla_\theta R) R^{\mathrm{T}} + R \nabla_\theta R^{\mathrm{T}}) = 2 \operatorname{tr}(R \nabla_\theta R^{\mathrm{T}}) \tag{A.319}$$

and since $\operatorname{tr}(RR^{\mathrm{T}}) = \operatorname{tr}(I)$ is a constant, $\operatorname{tr}(R \nabla_\theta R^{\mathrm{T}}) = 0$. Expanding equation (A.318):

$$\nabla_\theta \log |R^{\mathrm{T}} f'(x) Q| = \operatorname{tr}(\nabla_\theta (R^{\mathrm{T}}) f'(x) Q Q^{\mathrm{T}} g'(f(x)) R) + \operatorname{tr}(R^{\mathrm{T}} \nabla_\theta (f'(x)) Q Q^{\mathrm{T}} g'(f(x)) R). \tag{A.320}$$

Since Q is an orthonormal basis for $\mathcal{T}_x \mathcal{M}_X$, QQ^{T} is a projection matrix onto $\mathcal{T}_x \mathcal{M}_X$. This is because $(QQ^{\mathrm{T}})^2 = QQ^{\mathrm{T}} QQ^{\mathrm{T}} = QQ^{\mathrm{T}}$, using $Q^{\mathrm{T}} Q = I$. As a result, $QQ^{\mathrm{T}} \operatorname{proj}'(x) = \operatorname{proj}'(x)$. Since g can also be written inside a projection: $g(z) = \operatorname{proj}_Z(g(z))$, therefore $g'(z) = \operatorname{proj}'_Z(\tilde{g}(z)) \tilde{g}'(z)$, so $QQ^{\mathrm{T}} g'(z) = g'(z)$. Note also that $f'(x) g'(f(x)) = I$ since $f \circ g = \mathrm{id}$. This simplifies the equation:

$$\nabla_\theta \log |R^{\mathrm{T}} f'(x) Q| = \operatorname{tr}(\nabla_\theta (R^{\mathrm{T}}) R) + \operatorname{tr}(R^{\mathrm{T}} \nabla_\theta (f'(x)) g'(f(x)) R) \tag{A.321}$$

and finally

$$\nabla_\theta \log |R^{\mathrm{T}} f'(x) Q| = \operatorname{tr}(R^{\mathrm{T}} \nabla_\theta (f'(x)) g'(f(x)) R). \tag{A.322}$$

□

In the above proof we used the fact that $QQ^{\mathrm{T}} g'(z) = g'(z)$, where we dropped the index θ and use $g := f^{-1}$ for brevity. Can we use $RR^{\mathrm{T}} f'(x) = f'(x)$ to simplify the equation further? No, we cannot, since the expression involving f' is actually its derivative with respect to parameters, which may not have the same matrix structure as f'. Is it instead possible to use $g'(z) RR^{\mathrm{T}} = g'(z)$ for simplification?

If g is implemented as $\text{proj}_Z(\tilde{g}(z))$, this is not necessarily true, as $g'(z)$ might not be a map from the tangent space at z to the tangent space at $g(z)$. For example, if we add a small deviation v to z, where v is orthogonal to the tangent space at z, then $g(z+v)$ might not equal $g(z)$. However, this would mean that derivatives in the off-manifold direction can be non-zero, meaning that $g'(z)v \neq g'(z)RR^{\mathrm{T}}v = 0$ (since RR^{T} will project v to 0). We can change this by prepending g by a projection:

$$g = \text{proj}_X \circ \tilde{g} \circ \text{proj}_Z \,. \tag{A.323}$$

If proj_Z is an orthogonal projection, meaning that proj'_Z is symmetric, the column space and row space of proj_Z will both be the same as those of RR^{T}, meaning $\text{proj}'_Z(z)RR^{\mathrm{T}} = \text{proj}'_Z$ and hence $g'(z)RR^{\mathrm{T}} = g'(z)$. This leads to the following corollary:

Corollary A.22. *Suppose the assumptions of theorem A.20 hold and the following implementation:*

$$f_\theta^{-1} = \text{proj}_X \circ f_\theta^{-1} \circ \text{proj}_Z \tag{A.324}$$

where proj_Z *is an orthogonal projection. Then the derivative of the change of variables term has the following trace expression, where* $z = f_\theta(x)$*:*

$$\nabla_\theta \log |R^{\mathrm{T}} f'_\theta(x) Q| = \text{tr}((\nabla_\theta f'(x))(f_\theta^{-1})'(z)). \tag{A.325}$$

Proof. Again, we drop the index θ and let $g = f^{-1}$ for brevity. Take the result of theorem A.21 and use the cyclic property of the trace and the properties of g' discussed above:

$$\begin{aligned} \text{tr}(R^{\mathrm{T}} \nabla_\theta (f'(x)) g'(f(x)) R) &= \text{tr}(\nabla_\theta (f'(x)) g'(f(x)) RR^{\mathrm{T}}) && \text{(A.326)} \\ &= \text{tr}(\nabla_\theta (f'(x)) g'(f(x))). && \text{(A.327)} \end{aligned}$$

□

We use Hutchinson-style trace estimators to approximate the traces given above. This uses the property that, for a matrix $A \in \mathbb{R}^{n \times n}$ and a distribution $p(v)$ in $\mathbb{R}^n$ with unit second moment (meaning $\mathbb{E}[vv^{\mathrm{T}}] = I$),

$$\begin{aligned} \mathbb{E}_{p(v)}[v^{\mathrm{T}} A v] &= \text{tr}(\mathbb{E}_{p(v)}[v^{\mathrm{T}} A v]) && \text{(A.328)} \\ &= \text{tr}(\mathbb{E}_{p(v)}[vv^{\mathrm{T}}] A) && \text{(A.329)} \\ &= \text{tr}(A) && \text{(A.330)} \end{aligned}$$

meaning that $v^{\mathrm{T}} A v \approx \text{tr}(A)$ is an unbiased estimate of the trace of A.

We have two variants of the trace estimate derived above, one evaluated in $\mathbb{R}^n$, the other in $\mathbb{R}^m$. The first can be estimated using the following equality (again dropping the index θ and using $g = f^{-1}$):

$$\begin{aligned} &\text{tr}(R^{\mathrm{T}} \nabla_\theta (f'(x)) g'(f(x)) R) \\ &\quad = \mathbb{E}_{p(u)}[u^{\mathrm{T}} R^{\mathrm{T}} \nabla_\theta (f'(x)) g'(f(x)) Ru] && \text{(A.331)} \\ &\quad = \mathbb{E}_{p(v)}[v^{\mathrm{T}} \nabla_\theta (f'(x)) g'(f(x)) v] && \text{(A.332)} \\ &\quad = \nabla_\theta \mathbb{E}_{p(v)}[v^{\mathrm{T}} \nabla_\theta (f'(x)) \texttt{SG}[g'(f(x))] v] && \text{(A.333)} \end{aligned}$$

where $p(u)$ has unit second moment in $\mathbb{R}^n$ and $p(v)$ is the distribution of Ru, which lies in the tangent space at x and has unit second moment in that space by which we mean $\mathbb{E}[vv^{\mathrm{T}}] = RR^{\mathrm{T}}$. An example of such a distribution is the standard normal projected to the tangent space, i.e. $v = RR^{\mathrm{T}}\tilde{v}$ where $\tilde{v}$ is standard normal.

In the second case, we can just sample from a distribution with unit second moment in the embedding space $\mathbb{R}^m$:

$$\operatorname{tr}(\nabla_\theta(f'(x))g'(f(x))) = \nabla_\theta \mathbb{E}_{p(v)}[v^\mathrm{T}\nabla_\theta(f'(x))\mathtt{SG}[g'(f(x))]v]. \tag{A.334}$$

For this reason, we choose the first estimator, sampling v in the tangent space at x. This also simplifies the definition of g, meaning that we don't have to prepend it with a projection.

A.10. Free-form injective flows

This appendix is adapted from Sorrenson et al. (2024).

A.10.1. Derivation of gradient estimator

Theorem 7.4. *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^d$ be the pseudoinverse to a C^1 and injective $g_\varphi : \mathbb{R}^d \to \mathbb{R}^D$. Then, for all $x \in \mathcal{M}_\varphi$:*

$$\frac{1}{2}\nabla_\theta \log |f'_\theta(x) f'_\theta(x)^\mathrm{T}| = \operatorname{tr}\Big((\nabla_\theta f'_\theta(x)) g'_\varphi(z)\Big) \tag{A.335}$$

$$= \mathbb{E}_v\Big[v^\mathrm{T}(\nabla_\theta f'_\theta(x)) g'_\varphi(z) v\Big], \tag{A.336}$$

where $v \in \mathbb{R}^d$ is a random variable with zero mean and unit covariance and $z = f_\theta(x)$.

Proof. Just like for theorem 7.1, we use Jacobi's formula. Then, we find with $J = f'_\theta(x)$ for $x \in \mathcal{M}_\varphi$:

$$\frac{\partial}{\partial\theta_j}\frac{1}{2}\log\det JJ^\mathrm{T} = \frac{1}{2}\operatorname{tr}\left((JJ^\mathrm{T})^{-1}\frac{\partial}{\partial\theta_j}(JJ^\mathrm{T})\right) \tag{A.337}$$

$$= \frac{1}{2}\operatorname{tr}\left((JJ^\mathrm{T})^{-1}\left(\frac{\partial}{\partial\theta_j}J\right)J^\mathrm{T}\right) + \frac{1}{2}\operatorname{tr}\left((JJ^\mathrm{T})^{-1}J\left(\frac{\partial}{\partial\theta_j}J^\mathrm{T}\right)\right) \tag{A.338}$$

$$= \frac{1}{2}\operatorname{tr}\left(J^\mathrm{T}(JJ^\mathrm{T})^{-1}\frac{\partial}{\partial\theta_j}J\right) + \frac{1}{2}\operatorname{tr}\left(\left((JJ^\mathrm{T})^{-1}J\right)^\mathrm{T}\frac{\partial}{\partial\theta_j}J\right) \tag{A.339}$$

$$= \operatorname{tr}\left(J^\mathrm{T}(JJ^\mathrm{T})^{-1}\frac{\partial}{\partial\theta_j}J\right) \tag{A.340}$$

$$= \operatorname{tr}\left(\left(\frac{\partial}{\partial\theta_j}J\right)J^\dagger\right) \tag{A.341}$$

where we used the cyclic property of the trace, that $\operatorname{tr}(AB) = \operatorname{tr}(A^\mathrm{T}B^\mathrm{T})$ and that $J^\dagger = J^\mathrm{T}(JJ^\mathrm{T})^{-1}$ since J is surjective. Since g_φ is the pseudoinverse of f_θ, we can replace $J^\dagger = g'_\varphi(z = f_\theta(x))$. □

Note on optimality with respect to reconstruction loss

Our estimator relies on the approximation $f'_\theta(\hat{x}) \approx g'_\varphi(f_\theta(x))^\dagger$. If f_θ and g_φ are consistent ($f_\theta \circ g_\varphi$ is the identity) this guarantees that $g'_\varphi(f_\theta(x))f'_\theta(\hat{x}) = I$, but not that $f'_\theta(\hat{x})$ is the Moore-Penrose inverse of $g'_\varphi(f_\theta(x))$ and vice versa. A sufficient requirement is that f_θ and g_φ are optimal with respect to the

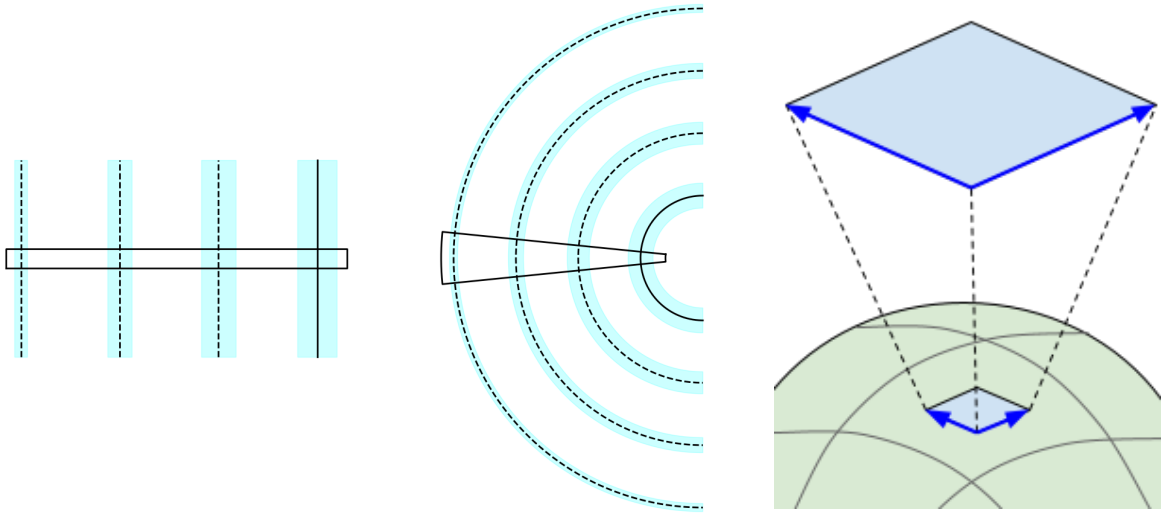


Figure A.2.: Representation of ill-defined probability density $\tilde{p}(x) \propto p(\hat{x})e^{-\beta\|\hat{x}-x\|^2}$ (*left* and *center*). Solid black lines denote the manifold, dashed lines are a constant distance from the manifold. The probability density is constant along the manifold. The width of the cyan bands is proportional to $e^{-\beta\|\hat{x}-x\|^2}$ and represents the probability density along the on- and off-manifold contours. While the density is reasonable for a flat manifold (*left*), note that the amount of probability mass associated with a region of the manifold (bounded by solid lines) is *larger* at some points off the manifold than on it when the manifold has curvature (*center*). This behavior can lead to divergent solutions when optimizing for likelihood and should be compensated for. The appropriate compensation factor is the ratio of the volume of a small region on the manifold (small blue square embedded in green manifold, *right*) to the equivalent region off the manifold (large blue square, *right*). The blue arrows represent an orthonormal frame on the manifold, and the equivalent frame in the off-manifold region.

reconstruction loss, that is, any variation in the functions would lead to a higher reconstruction. With calculus of variations, it is possible to show that such f_θ and g_φ are consistent, and

$$(\hat{x} - x)^{\mathrm{T}} g'_\varphi(f_\theta(x)) = 0 \tag{A.342}$$

for all x. By taking the derivative with respect to x and evaluating at some x in the image of g_φ (so $\hat{x} = x$) we have that

$$f'_\theta(\hat{x})^{\mathrm{T}} g'_\varphi(f_\theta(x))^{\mathrm{T}} g'_\varphi(f_\theta(x)) - g'_\varphi(f_\theta(x)) = 0 \tag{A.343}$$

and hence

$$f'_\theta(\hat{x}) = g'_\varphi(f_\theta(x))^{\mathrm{T}} (g'_\varphi(f_\theta(x))^{\mathrm{T}} g'_\varphi(f_\theta(x)))^{-1} = g'_\varphi(f_\theta(x))^\dagger. \tag{A.344}$$

In the remainder of the appendix, given an encoder-decoder pair f_θ and g_φ which are optimal with respect to the reconstruction loss, we refer to g_φ as the pseudoinverse of f_θ, and f_θ as the pseudoinverse of g_φ.

Modified estimator

As stated in 7.3.3, we modify the log-determinant estimator (equation (7.31)) by replacing $f'_\theta(\hat{x})$ by $f'_\theta(x)$. The loss we are trying to optimize (sending gradient from the log-determinant to the encoder) is:

$$\mathcal{L}(x) = -\log p(f_\theta(x)) - \log \mathrm{vol}\Big(f'_\theta(\hat{x})\Big) + \beta\|\hat{x} - x\|^2 \tag{A.345}$$

with $\operatorname{vol}(f'_\theta(\hat{x})) = \sqrt{\det(f'_\theta(\hat{x}) f'_\theta(\hat{x})^{\mathrm{T}})}$. Consider the probability density $\tilde{p}$ implied by interpreting this loss as a negative log-likelihood:

$$\tilde{p}(x) \propto p(\hat{x}) e^{-\beta\|\hat{x}-x\|^2} = p(z) \operatorname{vol}\Big(f'_\theta(\hat{x})\Big) e^{-\beta\|\hat{x}-x\|^2} \tag{A.346}$$

where $p(\hat{x})$ is the on-manifold density. Unfortunately, this density is ill-defined and leads to pathological behavior (see figure A.2). To provide a correction to this density, we need a term which compensates for the volume increase or decrease of off-manifold regions in comparison to the on-manifold region they are projected to. This is depicted in figure A.2 (*right*). The blue arrows in the on-manifold region will span the same latent-space volume as the blue arrows in the off-manifold region. The change in volume between the depicted on-manifold region and the latent space is $\operatorname{vol}(f'_\theta(\hat{x}))$ and $\operatorname{vol}(f'_\theta(x))$ between the off-manifold region and the latent space. Combining these facts means

$$\operatorname{vol}(f'_\theta(\hat{x})) \times (\text{volume of on-manifold region}) = \operatorname{vol}(f'_\theta(x)) \times (\text{volume of off-manifold region}) \tag{A.347}$$

and hence the ratio of the volume of the on-manifold to the off-manifold region is $\operatorname{vol}(f'_\theta(x))/\operatorname{vol}(f'_\theta(\hat{x}))$. Multiplying $\tilde{p}(x)$ by this factor leads to:

$$\tilde{p}(x) \frac{\operatorname{vol}(f'_\theta(x))}{\operatorname{vol}(f'_\theta(\hat{x}))} = p(z) \operatorname{vol}\Big(f'_\theta(x)\Big) e^{-\beta\|\hat{x}-x\|^2} \tag{A.348}$$

and the corresponding negative log-likelihood loss is:

$$\mathcal{L}(x) = -\log p(f_\theta(x)) - \log \operatorname{vol}\Big(f'_\theta(x)\Big) + \beta\|\hat{x}-x\|^2 \tag{A.349}$$

The surrogate for the log-determinant term is therefore:

$$-\operatorname{tr}(f'_\theta(x)\mathtt{SG}(f'_\theta(x)^\dagger)) \tag{A.350}$$

In order to maintain computational efficiency, we approximate $f'_\theta(x)^\dagger$ by $g'_\varphi(f_\theta(x))$:

$$-\operatorname{tr}(f'_\theta(x)\mathtt{SG}(g'_\varphi(f_\theta(x)))) \tag{A.351}$$

giving the stated correction.

A.10.2. Linear model trained on maximum likelihood alone

Consider a linear model, trained on data with zero mean and covariance Σ. Let the encoder function be $f_\theta(x) = Ax$ and suppose that A has positive singular values, meaning that AA^{T} is positive definite. Let the decoder function be $g_\varphi(z) = A^\dagger z$, where $A^\dagger = A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1}$. We want to minimize a combination of negative log-likelihood and a reconstruction loss (here we use $1/2\sigma^2$ instead of β as prefactor):

$$\begin{aligned}
\mathcal{L} &= \mathcal{L}_{\mathrm{NLL}} + \mathcal{L}_{\mathrm{recon.}} && \text{(A.352)}\\
&= E_x\left[\frac{1}{2}\|Ax\|^2 - \frac{1}{2}\log\det(AA^{\mathrm{T}}) + \frac{1}{2\sigma^2}\|A^\dagger Ax - x\|^2\right] && \text{(A.353)}\\
&= \frac{1}{2}E_x[x^{\mathrm{T}}A^{\mathrm{T}}Ax] - \frac{1}{2}\log\det(AA^{\mathrm{T}}) + \frac{1}{2\sigma^2}E_x[x^{\mathrm{T}}(A^\dagger A - \mathbb{I})^2 x] && \text{(A.354)}\\
&= \frac{1}{2}\operatorname{tr}(AE_x[xx^{\mathrm{T}}]A^{\mathrm{T}}) - \frac{1}{2}\log\det(AA^{\mathrm{T}}) + \frac{1}{2\sigma^2}\operatorname{tr}(E_x[xx^{\mathrm{T}}](\mathbb{I} - A^\dagger A)) && \text{(A.355)}\\
&= \frac{1}{2}\operatorname{tr}(A\Sigma A^{\mathrm{T}}) - \frac{1}{2}\log\det(AA^{\mathrm{T}}) + \frac{1}{2\sigma^2}\operatorname{tr}(\Sigma(\mathbb{I} - A^\dagger A)) && \text{(A.356)}
\end{aligned}$$

where A is a full rank $d \times D$ matrix with $d \leq D$.

Before solving for the minimum, let's review some matrix calculus identities. It is often convenient to consider A as a function of a single variable x, differentiate with respect to x, and then choose x to be A_{ij}. Then the derivative is

$$\frac{d}{dA_{ij}} A = E^{(ij)} \tag{A.357}$$

where $E^{(ij)}$ is a matrix of zeros, except for the ij entry which is a one. We can write this as $E^{(ij)}_{kl} = \delta_{ik}\delta_{jl}$ where δ is the Kronecker delta. When evaluating $E^{(ij)}$ inside a trace, we get the simple expression:

$$\mathrm{tr}(E^{(ij)}B) = E^{(ij)}_{kl} B_{lk} = \delta_{ik}\delta_{jk}B_{lk} = B_{ji} \tag{A.358}$$

using Einstein notation. The additional matrix identities we will need are Jacobi's formula for a square invertible matrix B:

$$\frac{d}{dx}\det(B) = \det(B)\,\mathrm{tr}\left(B^{-1}\left(\frac{d}{dx}B\right)\right) \tag{A.359}$$

and hence

$$\frac{d}{dx}\log\det(B) = \mathrm{tr}\left(B^{-1}\left(\frac{d}{dx}B\right)\right) \tag{A.360}$$

and we will prove the following lemma.

Lemma A.23. *Suppose the matrix A depends on a variable x. Then we have the following expression for the derivative of the projection operator $A^\dagger A$:*

$$\frac{d}{dx}(A^\dagger A) = A^\dagger\left(\frac{d}{dx}A\right)(\mathbb{I} - A^\dagger A) + \left(A^\dagger\left(\frac{d}{dx}A\right)(\mathbb{I} - A^\dagger A)\right)^{\mathrm{T}} \tag{A.361}$$

Proof. The following is based on the proof to lemma 4.1 in (Golub & Pereyra, 1973). Define the projection operator $P_A = A^\dagger A$ and its complement $P_A^\perp = \mathbb{I} - P_A$. Then, since $P_A P_A = P_A$,

$$\left(\frac{d}{dx}P_A\right) = \left(\frac{d}{dx}P_A P_A\right) = \left(\frac{d}{dx}P_A\right)P_A + P_A\left(\frac{d}{dx}P_A\right) \tag{A.362}$$

In addition, since $P_A A^{\mathrm{T}} = A^{\mathrm{T}}$

$$\left(\frac{d}{dx}P_A A^{\mathrm{T}}\right) = \left(\frac{d}{dx}P_A\right)A^{\mathrm{T}} + P_A\left(\frac{d}{dx}A\right)^{\mathrm{T}} = \left(\frac{d}{dx}A\right)^{\mathrm{T}} \tag{A.363}$$

and therefore

$$\left(\frac{d}{dx}P_A\right)P_A = \left(\frac{d}{dx}P_A\right)A^{\mathrm{T}}A^{\dagger\mathrm{T}} = \left(\frac{d}{dx}A\right)^{\mathrm{T}}A^{\dagger\mathrm{T}} - P_A\left(\frac{d}{dx}A\right)^{\mathrm{T}}A^{\dagger\mathrm{T}} = P_A^\perp\left(\frac{d}{dx}A\right)^{\mathrm{T}}A^{\dagger\mathrm{T}} \tag{A.364}$$

By similar steps but using $AP_A = A$, we can derive

$$P_A\left(\frac{d}{dx}P_A\right) = A^\dagger\left(\frac{d}{dx}A\right)P_A^\perp \tag{A.365}$$

Putting it all together gives

$$\left(\frac{d}{dx}P_A\right) = A^\dagger\left(\frac{d}{dx}A\right)P_A^\perp + \left(A^\dagger\left(\frac{d}{dx}A\right)P_A^\perp\right)^{\mathrm{T}} \tag{A.366}$$

Note that the second term is just the transpose of the first. □

Now we are ready to find the derivative of the loss and set it to zero.

Lemma A.24. *The derivative of the loss with respect to A takes the form:*

$$\frac{d}{dA}\mathcal{L} = \left(\Sigma A^{\mathrm{T}} - A^\dagger - \frac{1}{\sigma^2}(\mathbb{I} - A^\dagger A)\Sigma A^\dagger\right)^{\mathrm{T}} \tag{A.367}$$

Proof. Let's apply the above identities to the first term in the loss:

$$\frac{d}{dx}\frac{1}{2}\operatorname{tr}(A\Sigma A^{\mathrm{T}}) = \frac{1}{2}\operatorname{tr}\left(\left(\frac{d}{dx}A\right)\Sigma A^{\mathrm{T}} + A\Sigma\left(\frac{d}{dx}A\right)^{\mathrm{T}}\right) \tag{A.368}$$

$$= \operatorname{tr}\left(\left(\frac{d}{dx}A\right)\Sigma A^{\mathrm{T}}\right) \tag{A.369}$$

since the trace is invariant under transposition and hence

$$\frac{d}{dA_{ij}}\frac{1}{2}\operatorname{tr}(A\Sigma A^{\mathrm{T}}) = \operatorname{tr}(E^{(ij)}\Sigma A^{\mathrm{T}}) = (\Sigma A^{\mathrm{T}})_{ji} \tag{A.370}$$

Applying Jacobi's formula to the second term in the loss gives:

$$\frac{d}{dx}\frac{1}{2}\log\det(AA^{\mathrm{T}}) = \frac{1}{2}\operatorname{tr}\left((AA^{\mathrm{T}})^{-1}\left(\frac{d}{dx}(AA^{\mathrm{T}})\right)\right) \tag{A.371}$$

$$= \frac{1}{2}\operatorname{tr}\left((AA^{\mathrm{T}})^{-1}\left(\left(\frac{d}{dx}A\right)A^{\mathrm{T}} + A\left(\frac{d}{dx}A\right)^{\mathrm{T}}\right)\right) \tag{A.372}$$

and therefore

$$\frac{d}{dA_{ij}}\frac{1}{2}\log\det(AA^{\mathrm{T}}) = \frac{1}{2}\operatorname{tr}\left((AA^{\mathrm{T}})^{-1}\left(E^{(ij)}A^{\mathrm{T}} + AE^{(ji)}\right)\right) \tag{A.373}$$

$$= \frac{1}{2}\operatorname{tr}\left(E^{(ij)}A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1} + E^{(ij)}A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1}\right) \tag{A.374}$$

$$= \left(A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1}\right)_{ji} \tag{A.375}$$

$$= A^\dagger_{ji} \tag{A.376}$$

where we used the cyclic and transpose properties of the trace and that $E^{(ji)\mathrm{T}} = E^{(ij)}$.

The final term requires a derivative of $\operatorname{tr}(\Sigma(\mathbb{I} - A^\dagger A))$, which is equal to a derivative of $-\operatorname{tr}(\Sigma A^\dagger A)$. We use the formula for the derivative of the projection operator to get

$$\frac{d}{dx}\frac{1}{2}\operatorname{tr}(\Sigma A^\dagger A) = \frac{1}{2}\operatorname{tr}\left(\Sigma\left(\frac{d}{dx}(A^\dagger A)\right)\right) \tag{A.377}$$

$$= \operatorname{tr}\left(\Sigma A^\dagger\left(\frac{d}{dx}A\right)(\mathbb{I} - A^\dagger A)\right) \tag{A.378}$$

again using the transpose property of the trace, and therefore

$$\frac{d}{dA_{ij}}\frac{1}{2}\operatorname{tr}(\Sigma A^\dagger A) = \operatorname{tr}(E^{(ij)}(\mathbb{I} - A^\dagger A)\Sigma A^\dagger) = ((\mathbb{I} - A^\dagger A)\Sigma A^\dagger)_{ji} \tag{A.379}$$

Putting the three expressions together, we have that

$$\frac{d}{dA}\mathcal{L} = \left(\Sigma A^{\mathrm{T}} - A^\dagger - \frac{1}{\sigma^2}(\mathbb{I} - A^\dagger A)\Sigma A^\dagger\right)^{\mathrm{T}} \tag{A.380}$$

□

Lemma A.25. *The critical points of $\mathcal{L}$ satisfy the following properties:*

1. *$A = U\Sigma^{-\frac{1}{2}}$ with $UU^{\mathrm{T}} = \mathbb{I}$*
2. *$U^{\mathrm{T}}U$ commutes with Σ*

Proof. Using lemma A.24, the critical points satisfy

$$\Sigma A^{\mathrm{T}} - A^\dagger - \frac{1}{\sigma^2}(\mathbb{I} - A^\dagger A)\Sigma A^\dagger = 0 \tag{A.381}$$

By multiplying by A from the left we have

$$A\Sigma A^{\mathrm{T}} = \mathbb{I}_d \tag{A.382}$$

meaning that $U = A\Sigma^{\frac{1}{2}}$ must have orthonormal rows (since $UU^{\mathrm{T}} = \mathbb{I}$). With this definition, we can write $A = U\Sigma^{-\frac{1}{2}}$.

If we now multiply by A from the right, we get

$$\Sigma A^{\mathrm{T}}A - A^\dagger A - \frac{1}{\sigma^2}\Sigma A^\dagger A + \frac{1}{\sigma^2}A^\dagger A\Sigma A^\dagger A = 0 \tag{A.383}$$

Noting that the second and fourth terms are symmetric (since $A^\dagger A$ is symmetric), this means that the remaining terms must be symmetric:

$$\Sigma A^{\mathrm{T}}A - \frac{1}{\sigma^2}\Sigma A^\dagger A = A^{\mathrm{T}}A\Sigma - \frac{1}{\sigma^2}A^\dagger A\Sigma \tag{A.384}$$

Since $A^{\mathrm{T}}A$ commutes with $A^\dagger A$, they are simultaneously diagonalizable, and since they are both symmetric, they share an orthonormal basis of eigenvectors. Clearly $A^{\mathrm{T}}A - A^\dagger A/\sigma^2$ has the same basis. Since this matrix commutes with Σ, it must share a basis with Σ and hence Σ has the same basis as $A^{\mathrm{T}}A$. This means that Σ commutes with $A^{\mathrm{T}}A$.

Expanding A in terms of U, this means that

$$\Sigma A^{\mathrm{T}}A = \Sigma^{\frac{1}{2}}U^{\mathrm{T}}U\Sigma^{-\frac{1}{2}} = \Sigma^{-\frac{1}{2}}U^{\mathrm{T}}U\Sigma^{\frac{1}{2}} = A^{\mathrm{T}}A\Sigma \tag{A.385}$$

and therefore $\Sigma U^{\mathrm{T}}U = U^{\mathrm{T}}U\Sigma$, meaning that Σ and $U^{\mathrm{T}}U$ commute. □

Consider as an example the case where Σ is diagonal. $U^\mathrm{T}U$ is a projection matrix, and in this case, must be diagonal due to commuting with Σ. As a result, it must have exactly d ones and $D-d$ zeros along the diagonal. This means that the rows of U are a basis for the d dimensional axis-aligned subspace corresponding to the d nonzero entries. In the case of a non-diagonal Σ, this generalizes to the rows of U spanning the same subspace as some subset of d eigenvectors of Σ. This leads to the expression of the loss function in the next theorem.

Theorem A.26. *Let Σ have the eigen-decomposition $V\Lambda V^\mathrm{T}$ with $\Lambda = \operatorname{diag}(\lambda)$. Let $U^\mathrm{T}U$ have the eigen-decomposition VEV^T, with $E = \operatorname{diag}(\alpha)$. Then the minimum of the loss is satisfied by α such that*

$$\mathcal{L}_\alpha = \sum_{i=1}^{D} \frac{1}{2}\alpha_i \left(\log \lambda_i - \frac{1}{\sigma^2}\lambda_i \right) \tag{A.386}$$

is minimal, subject to the constraint $\alpha_i \in \{0,1\}$ with $\sum_{i=1}^{D} \alpha_i = d$.

Proof. Let's note a couple of properties. First, we have $(U\Sigma U^\mathrm{T})^k = U\Sigma^k U^\mathrm{T}$ due to Σ commuting with $U^\mathrm{T}U$, so we can say that $f_\theta(U\Sigma U^\mathrm{T}) = U f_\theta(\Sigma)U^\mathrm{T}$ for any matrix function f_θ with a Taylor series. Next, $U^\mathrm{T}U$ is an orthogonal projection matrix, so E is a diagonal matrix with ones or zeros on the diagonal. We know that the rank of U is d, hence E has exactly d ones and $D-d$ zeros along the diagonal. Therefore, we have the constraint $\alpha_i \in \{0,1\}$ with $\sum_{i=1}^{D} \alpha_i = d$. Next, note that $A^\dagger A = U^\mathrm{T}U$.

Now we substitute back into the loss in terms of U:

$$\mathcal{L} = \frac{1}{2}\operatorname{tr}(UU^\mathrm{T}) - \frac{1}{2}\log\det(U\Sigma^{-1}U^\mathrm{T}) + \frac{1}{2\sigma^2}\operatorname{tr}(\Sigma(\mathbb{I} - U^\mathrm{T}U)) \tag{A.387}$$

$$= \frac{1}{2}\operatorname{tr}(U\log(\Sigma)U^\mathrm{T}) - \frac{1}{2\sigma^2}\operatorname{tr}(U\Sigma U^\mathrm{T}) + \text{const.} \tag{A.388}$$

where we used that

$$\log\det(U\Sigma^{-1}U^\mathrm{T}) = \operatorname{tr}\log(U\Sigma^{-1}U^\mathrm{T}) = -\operatorname{tr}(U\log(\Sigma)U^\mathrm{T}) \tag{A.389}$$

Note that $\operatorname{tr}(UU^\mathrm{T})$ is constant. Consider that

$$\operatorname{tr}(U\Sigma U^\mathrm{T}) = \operatorname{tr}(U^\mathrm{T}U\Sigma U^\mathrm{T}U) = \operatorname{tr}(VEDEV^\mathrm{T}) = \operatorname{tr}(EDE) = \alpha \cdot \lambda \tag{A.390}$$

The same logic holds for the term with $\log(\Sigma)$. Therefore, dropping constant terms, the loss can be written in terms of α and λ:

$$\mathcal{L}_\alpha = \sum_{i=1}^{D} \frac{1}{2}\alpha_i \left(\log \lambda_i - \frac{1}{\sigma^2}\lambda_i \right) \tag{A.391}$$

□

The loss will take different values depending on which elements of α are nonzero. Define $f_\theta(\lambda_i) = \log\lambda_i - \lambda_i/\sigma^2$. The loss will be minimized when the nonzero α_i correspond to those values of $f_\theta(\lambda_i)$ which are minimal. Clearly $f''_\theta(\lambda_i) < 0$, so f_θ has only one maximum at $\lambda_i = \sigma^2$ and is unbounded below on either side of this maximum (see figure A.3). Consider the two extreme cases:

1. All eigenvalues λ are smaller than σ^2. The minimal values of $f_\theta(\lambda_i)$ will occur for the smallest values of λ_i. Hence the d smallest eigenvalues of Σ will be selected.

2. All eigenvalues λ are larger than σ^2. The minimal values of $f_\theta(\lambda_i)$ will occur for the largest values of λ_i. Hence the d largest eigenvalues of Σ will be selected.

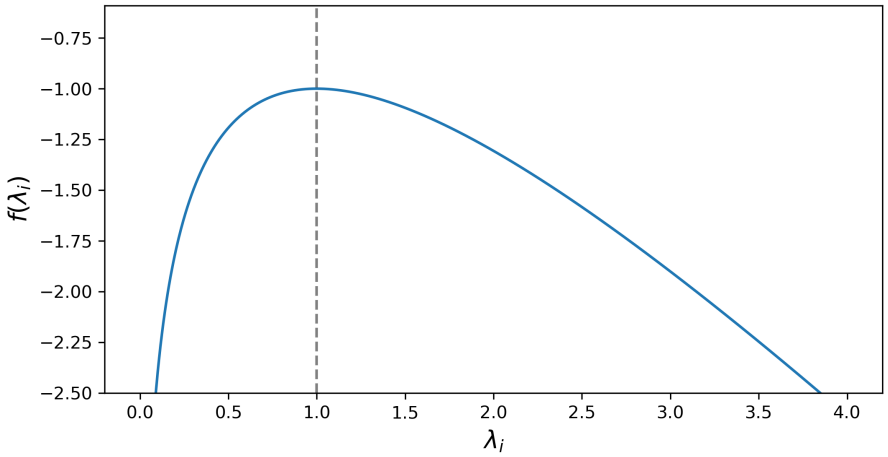


Figure A.3.: Plot of $f_\theta(\lambda_i) = \log \lambda_i - \lambda_i/\sigma^2$ with $\sigma = 1$, showing maximum value at $\lambda_i = \sigma^2$ and unbounded behavior on either side.

In the intermediate regime, there will be a phase transition between these two extremes.

In the first case, the reconstructed manifold will be a projection onto the d-dimensional subspace with the lowest variance, exactly the opposite result to PCA. In the second case, the reconstructed manifold will be a projection onto the d-dimensional subspace with the highest variance, the same result as PCA. If we maximize the likelihood on the manifold without any reconstruction loss, corresponding to the $\sigma^2 \to \infty$ limit, we actually learn the lowest entropy manifold. It makes more sense to learn the highest entropy manifold as in PCA. We can ensure this is the case by adding Gaussian noise of variance σ^2 to the data, ensuring that the minimum eigenvalue of the covariance matrix is at least σ^2, even if the original data is degenerate.

B. Experimental details

B.1. Layer-wise flow

B.1.1. Deep universal layer-wise flow

This appendix is adapted from Draxler et al. (2024b).

In an experiment on a toy dataset for figure 5.7, we demonstrate that a coupling flow constructed layer by layer as in equation (5.30) learns a target distribution. We proceed as follows:

We construct a data distribution on a circle as a Gaussian mixture of M Gaussians with means $m_i = (r\cos\varphi_i, r\sin\varphi_i)$, where $\varphi_i = 0, \frac{1}{M}2\pi, \ldots, \frac{M-1}{M}2\pi$ are equally spaced, and $\sigma_i = 0.3$. The advantage of approximating the ring with this construction is that this yields a simple to evaluate data density, which we need for accurately plotting $p_\theta(z)$:

$$p(x) = \frac{1}{M}\sum_{i=1}^{M}\mathcal{N}(x; m_i, \sigma^2 I). \tag{B.1}$$

We then fit a total 100 layers in the following way: First, treat $p(x)$ as the initial guess for the latent distribution. Then, we build the affine coupling block that maximally reduces the loss using equation (5.30). We therefore need to know the conditional mean $m(b)$ and standard deviation $\sigma(b)$ for each b. We approximate this from a finite number of samples N which are grouped by the passive coordinate b into B bins so that N/B samples are in each bin. We then compute the empirical mean m_i and standard deviation σ_i over the active dimension in each bin $i = 1, \ldots, B$. According to equation (5.30), we define $s_i = \frac{1}{\sigma_i}$ and $t_i = -\frac{1}{\sigma_i}m_i$ at the bin centers and interpolate between bins using a cubic spline. Outside the domain of the splines, we extrapolate with constants s, t with the value of the closest bin. We do not directly optimize over Q, but choose the Q that reduces the loss most out of N_Q random 2D rotation matrices.

We limit the step size of each layer to avoid artifacts from finite training data, by mapping:

$$\tilde{x} = \alpha x + (1-\alpha) f_{\text{blk}}(x). \tag{B.2}$$

In addition, we resample the training data from the ground truth distribution after every step to avoid overfitting. We do not explicitly control for the bi-Lipschitz constant of our coupling blocks because we do not encounter any numerical problems.

We choose $N = 2^{26}$, $B = 64$, $M = 20$, $\alpha = 0.5$, $N_Q = 10$. The resulting flow has $64 \cdot 2 \cdot 100 = 12,800$ learnable parameters. Figure B.1 shows how the KL divergence vanishes for our layer-wise training, together with Δ_{affine}.

B.1.2. Comparison of training methods on toy distributions

This appendix is adapted from Draxler et al. (2020).

In addition, we provide more examples on a set of toy distributions. As before, we train layer-wise using OAS and randomly fixed rotations, and end-to-end. Additionally, we train a mixed variant of OAS and end-to-end: New layers are still added one by one, but in iteration l the parameters of *all*

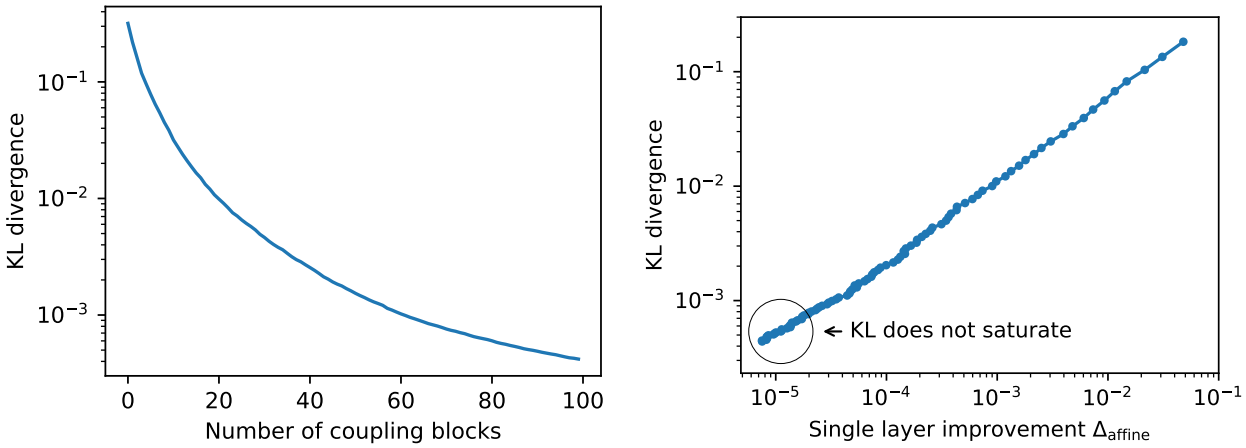


Figure B.1.: Empirically, the KL divergence decreases as more coupling blocks are added for the toy distribution considered in figure 5.7 *(left)*. At the same time, there is a strong correlation between the loss improvement by a single coupling block $\Delta_{\text{affine}}[p_\theta(z)]$ and the KL divergence *(right)*. Crucially, the KL divergence does not saturate as the loss improvements become smaller. The coupling flow considered is trained according to the greedy layer-wise training in our proof construction.

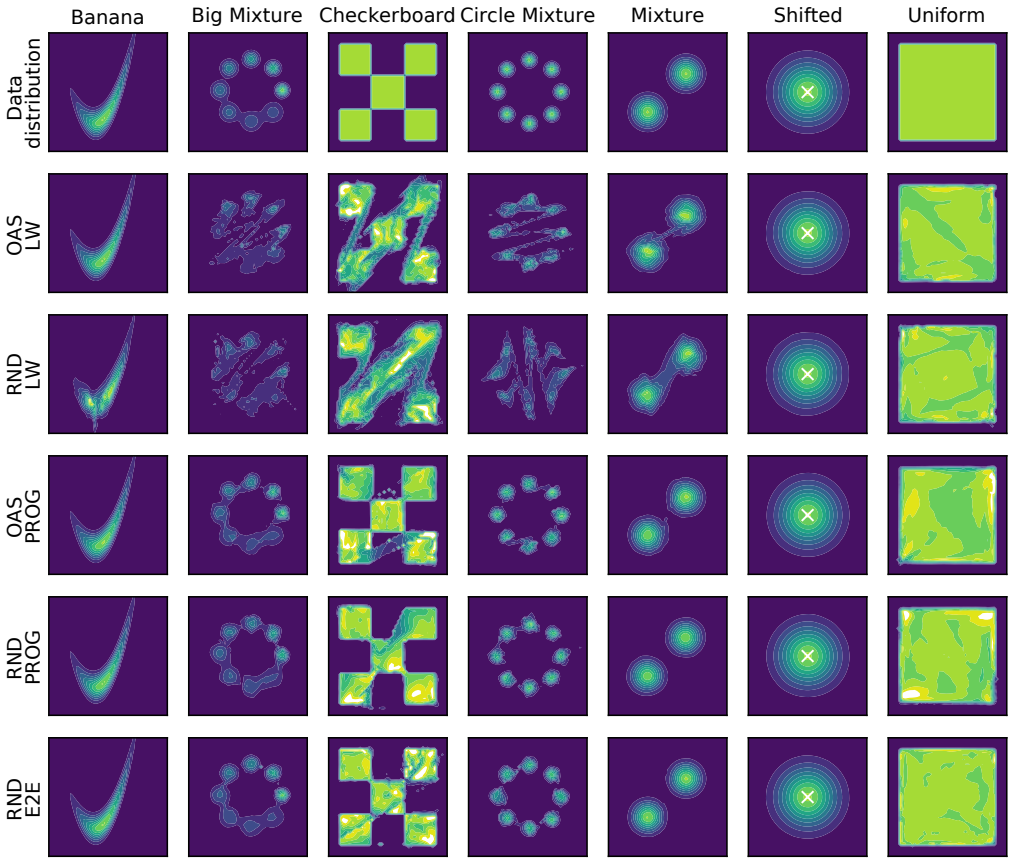


Figure B.2.: Twelve-block affine coupling normalizing flows trained on different toy problems *(top row)*. The following rows depict different training methods: layer-wise "LW" *(rows 2 and 3)*, progressively "PROG" *(rows 4-5)* and end-to-end "E2E" *(last row)*. Rotations are "OAS" when determined by algorithm 1 *(rows 2 and 4)* or randomly fixed "RND" *(rows 3, 5 and 6)*.

layers 1 through l are adapted in an end-to-end fashion. We call this training "progressive" as layers are progressively activated and never turned off again.

We find the following results: Optimal rotations always outperform random rotations in layer-wise training. With only a few layers, they also outperform end-to-end training, but are eventually overtaken as the network depth increases. Progressive training continues to be competitive also for deep networks.

Figure B.2 shows the density estimates after twelve layers. At this point, none of the methods show a significant improvement by adding additional layers. Hyperparameters were optimized for each setup to obtain a fair comparison. Densities obtained by layer-wise training exhibit significant spurious structure for both optimal and random rotations, with an advantage for optimally chosen subspaces.

B.2. Volume-preserving flows

This appendix is adapted from Draxler et al. (2024b).

The target distribution is a two-dimensional Gaussian Mixture Model with two modes. The two modes have the same relative weight but different covariance matrices ($\Sigma_1 = I \cdot 0.2$, $\Sigma_2 = I \cdot 0.1$) and means ($m_1 = [-0.5, -0.5]$, $m_2 = [0.5, 0.5]$).

The normalizing flow with a constant Jacobian determinant consists of 15 GIN coupling blocks as introduced in Sorrenson et al. (2019). This type of coupling block has a Jacobian determinant of one. To allow for a global volume change, a layer with a learnable global scaling is added after the final coupling block. This learnable weight is initialized as one. For the normalizing flow with variable Jacobian determinant, the GIN coupling is modified by removing the normalization of the scaling factors in the affine couplings. This allows the normalizing flow to have variable Jacobian determinants. In this case, the global scaling block is omitted. To implement the normalizing flow, we use the FrEIA package (Ardizzone et al., 2018a) implementation of the GIN coupling blocks.

In both normalizing flows, the two subnetworks used to compute the parameters of the affine couplings are fully-connected neural networks with two hidden layers and a hidden dimensionality of 128. ReLU activations are used. The weights of the linear layers of the subnetworks are initialized by applying the PyTorch implementation of the Xavier initialization (Glorot & Bengio, 2010). In addition, the weights and biases of the final layer of each subnetwork are set to zero.

The networks are trained using the Adam (Kingma & Ba, 2017) with PyTorch's default settings and an initial learning rate of $1 \cdot 10^{-3}$ which is reduced by a factor of ten after $5000, 10000$ and 15000 training iterations. In total, the training ran for 25000 iterations. In each iteration, a batch of size 128 was drawn from the target distribution to compute the negative log likelihood objective. We use a standard normal distribution as the latent distribution.

For obtaining the optimal distribution $p^*(x)$, we follow the grid procedure in section 5.2 and compute the probabilities on a regular 400×400 grid of grid spacing 0.01. The covariance of $p^*(z)$ is computed for the latent scaling layer by sampling 4096 points from the mixture model, moving them according to the volume-preserving flow learned using the grid and computing their empirical covariance matrix. This yields essentially the same scaling as obtained from training the volume-preserving flow.

To learn $p^*(r)$ as in appendix A.2.2, we sample $5 \cdot 10^8$ samples from ground truth data distribution. These samples are passed through the volume-preserving flow and afterward the L_2 norm is applied to the latent codes to obtain the latent radii r. We subtract the smallest observed radius from all radii to ensure that the distribution we construct is supported for all $r \geq 0$ and fit a histogram with 4200 bins to the radii. To obtain a smoother distribution, we use the left bin edges of the histogram and the corresponding density values to fit a cubic spline using SciPy's `interpolate` package (Virtanen et al., 2020). We choose the partition function of the distribution $p_r(r)$ defined by the spline such that it integrates to one. For a given latent code z the latent density can be computed by evaluating p_r at

$r = \|z\|_2$ and correcting for the volume at the given radius (see equation (B.3)).

$$p(z) = \frac{1}{2\pi\|z\|_2} \cdot p_r(r = \|z\|_2) \tag{B.3}$$

B.3. Iterative Gaussianization

This appendix is adapted from Draxler et al. (2023).

B.3.1. Measuring number of required layers

We proceed as follows to predict the number of required layers to reduce the loss by a factor $\tilde{\mathcal{L}}/\mathcal{L}$ from training L_{train} layers in all experiments:

1. Determine the entropy $H[p(x)]$ of the data distribution $p(x)$. This is required for computing the KL divergence in equation (4.3), and restricts us to distributions $p(x)$ which we can sample from (for training) and where we can evaluate the entropy via:

$$H[p(x)] = -\mathbb{E}_{x\sim p(x)}[\log p(x)] \tag{B.4}$$

2. Train a fixed number of layers L_{train} iteratively.

3. In general, we observe the loss as a function of depth L to decrease in a geometric series:

$$\mathcal{L}_{L_{\text{train}}} := \mathcal{L}_0 \gamma^{L_{\text{train}}}. \tag{B.5}$$

   We determine γ from the initial loss $\mathcal{L}_0$ and the loss after L_{train} layers:

$$\gamma := \left[\mathcal{L}_{L_{\text{train}}}/\mathcal{L}_0\right]^{1/L_{\text{train}}} \tag{B.6}$$

4. We then extrapolate equation (B.5) to predict the number of layers required for an arbitrary loss ratio $\tilde{\mathcal{L}}/\mathcal{L}$:

$$L = \frac{\log(\tilde{\mathcal{L}}/\mathcal{L})}{\log\gamma} \tag{B.7}$$

We evaluate equation (B.7) throughout the experiments for a loss ratio of $\log(\tilde{\mathcal{L}}/\mathcal{L}) = 1$, that is $\tilde{\mathcal{L}} = e^{-1}\mathcal{L} = 36.8\%\mathcal{L}$, but this is arbitrary as we are only interested in the scaling with dimension which is independent of $\tilde{\mathcal{L}}/\mathcal{L}$.

B.3.2. Gaussian data

As our dataset, we construct the same dataset of Gaussians as in appendix B.4.4. We choose D in 10 geometrically spaced values from 10 to 128. We choose $\lambda_{\min} = 10^{-3}$. The case where all eigenvalues are equal to 1, $\lambda_i = 1$, is excluded, as Gaussianization has converged at this point.

For each Σ_i in the resulting data set, we create N_{rot} differently rotated variants $\Sigma_i^r = Q_r^{\mathrm{T}}\Sigma_i Q_r$, where we choose $N_{\text{rot}} = 8$, whatever is larger. This corresponds to the initial unknown rotation of the data.

We then apply the analytic solution of a Gaussianization block on the covariance, given by equation (A.143). Then, another N_{rot} rotations are drawn, which rotate each resulting covariance. This procedure is repeated until the specified number of layers is reached. We use $L_{\text{train}} = 8D$ layers as we expect the number of required layers to increase with $\Omega(D)$.

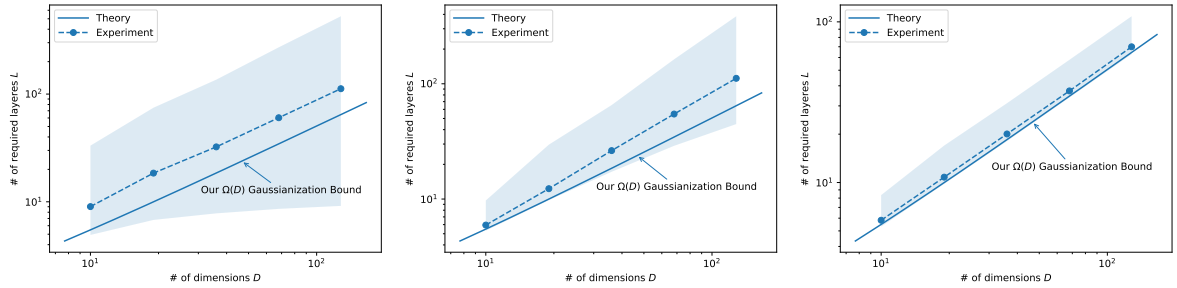


Figure B.3.: For larger loss $\mathcal{L} \gg 0$, the median number of required layers over the data set is larger than predicted, and some cases scale faster. *(Left)* If we predict the number of required layers from the first $L_{\text{train}} = D$ layers, the majority of cases show slower convergence than predicted. Some cases show faster convergence, see figure B.4. *(Medium)* After the first $L_{\text{train}} = 3D$ layers, most cases show the linear scaling behavior with dimension. *(Right)* The bound is valid for at least 90% of the data after $L_{\text{train}} = 10D$ layers. All averaging is performed via the median, and shaded regions cover 90% of the cases.

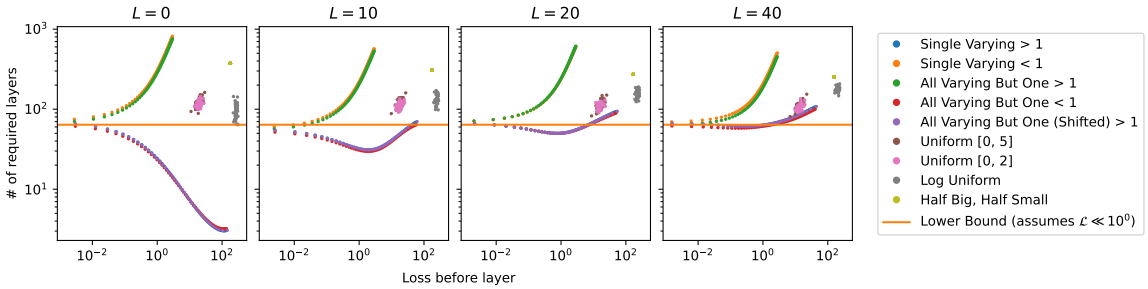


Figure B.4.: Our analytic lower bound on the number of layers gives a good estimate even outside the valid regime (where $\mathcal{L} \not\ll 1$) for several considered cases. *(Left)* The cases with faster initial convergence than equation (6.16) (orange) originate from "All eigenvalues varying but one" and "Single eigenvalue varying" with one eigenvalue bigger than the others *(left)*. After a finite number of layers ($L = 40$ here), the number of required layers predicted at this depth is close to the number predicted by the theory in all cases. The plot considers $D = 128$. Note that at this depth, the loss for the fastest configuration is still greater than 10, far from convergence. The suffixes "> 1" and "< 1" separate $\alpha \lessgtr 1$.

To evaluate theorem 6.3, we compare equation (6.16) with the empirical result. To estimate the required number of layers for a fixed loss ratio, we proceed as in appendix B.3.1 so that we end up in the regime where $\mathcal{L} \ll 1$. For figure 6.2, we fit γ from the loss ratio of the last two layers: $\gamma = \sqrt{\mathcal{L}_{L_{\text{train}}} / \mathcal{L}_{L_{\text{train}}-2}}$ instead of equation (B.6). The maximum loss in this case is 10^{-2}, so we are in the regime of $\mathcal{L} \ll 1$.

In figure B.3, we observe the same linear scaling behavior of required layers with dimension for $\mathcal{L} \gg 0$, i.e. at the beginning of training. However, in this case, the bound in equation (6.16) is violated in few scenarios: They require less layers than predicted. However, after a small number of layers, also these cases fulfill equation (6.16), see figure B.4.

B.3.3. Finding spurious dimensions in standard normal data

We randomly sample $N = 60,000$ normal samples of dimension $D = 3072$, which corresponds to the size of the CIFAR10 dataset. We optimize $w \in \mathbb{R}^D, \|w\| = 1$ to minimize the 1-dimensional sample-based

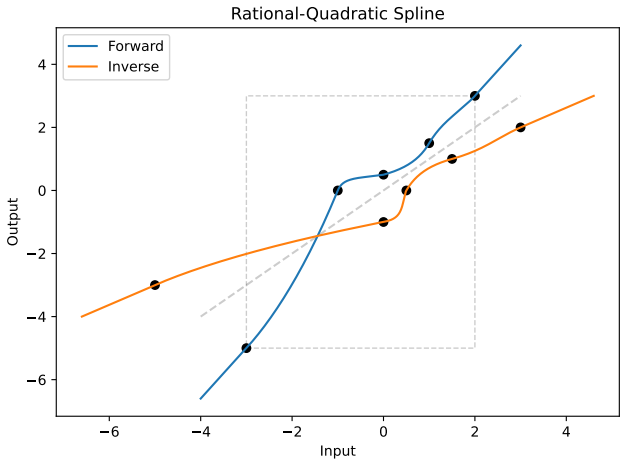


Figure B.5.: Rational-Quadratic Spline. Each pair of knots (black) is interpolated with a rational-quadratic polynomial. The outside is extrapolated with linear tails.

2-Wasserstein distance to a predefined bimodal distribution $p_{\text{adv}} = \Big(\mathcal{N}(-d/2, \sigma) + \mathcal{N}(d/2, \sigma)\Big)/2$:

$$W_2^2 = \sum_{i=1}^{N} \Big((w^{\mathrm{T}} x)_{\pi_w(i)} - y_i \Big)^2 . \tag{B.8}$$

Here, y_i are sorted samples from p_{adv}. The permutation $\pi_w : [N] \to [N]$ sorts the projected values $(w^{\mathrm{T}} x)$.

We choose the spread of the bimodal distribution to be $d = 2$ and the standard deviation of each mode as $\sigma = 0.4$.

We optimize w for 64 steps using SGD with a learning rate of 10 and momentum .8. After each update, we rescale $\|w\| = 1$. The final Wasserstein distance we obtain reads 0.03, down from 0.1 for a random w.

For the visualization in figure 6.4, we project the data once with a random $w_{\text{rand}}^{\mathrm{T}} x$ and once with $w^{\mathrm{T}} x$ into 70 bins.

B.3.4. Gaussianization implementation

Following equation (4.21), we construct f_{rot} as a random rotation $f_{rot}(x) = Qx$ where $Q \sim O(D)$ for data dimensionality D. For f_{dim}, we choose rational-quadratic splines, which allow for approximation of arbitrary functions by separating their domain into b bins. Given bin edges (or knots) $x^{(k)}, y^{(k)}, x^{(k+1)}, y^{(k+1)}$ and derivatives $\delta^{(k)}, \delta^{(k+1)}$ at those edges for bin $k = 1, ..., b$, they can be interpolated with a rational-quadratic polynomial as described in (Durkan et al., 2019). Beyond the outermost bin edges, the spline is extrapolated with linear tails.

We use an implementation of RQ splines based on (Dai & Seljak, 2021), where $\psi(x, \alpha) = (1 - \alpha)RQ(x) + \alpha x$ with a scalar regularization parameter α. We choose $b = 128$ bins, as well as $\alpha_1 = 0.9$ for the spline and $\alpha_2 = 0.99$ for the linear extrapolation, such that

$$f_{dim,\theta_i}(x_i) = \begin{cases} \psi(x_i, \alpha_1) & x_i^{(1)} \leq x_i \leq x_i^{(b+1)} \\ \psi(x_i, \alpha_2) & \text{otherwise} \end{cases} . \tag{B.9}$$

The $\alpha_{1,2}$ significantly slow down training, but increase performance (Dai & Seljak, 2021). It should

not alter the scaling behavior of the number of required layers with dimension, up to a constant factor independent of dimension.

Splines are fit to the CDF of the data by evenly distributing bin knots on the quantiles of the data and applying the inverse Gaussian CDF:

$$x_i^{(k)} = q\left(\frac{k}{b+2}\right), \tag{B.10}$$

$$y_i^{(k)} = G^{-1}\left(x_i^{(k)}\right), \tag{B.11}$$

where $q(p) = \sqrt{2}\,\mathrm{erf}^{-1}(2p-1)$ is the quantile function and $G(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$ is the standard normal CDF. The inner derivatives $\delta^{(k)}$ can then be estimated by finite differences, following (Durkan et al., 2019):

$$h_i^{(k)} = x_i^{(k+1)} - x_i^{(k)}, \tag{B.12}$$

$$s_i^{(k)} = \frac{y_i^{(k+1)} - y_i^{(k)}}{x_i^{(k+1)} - x_i^{(k)}}, \tag{B.13}$$

$$\delta_i^{(k+1)} = \frac{s_i^{(k)} h^{(k+1)} + s^{(k+1)} h^{(k)}}{h^{(k+1)} + h^{(k)}}, \quad k = 1, ..., b-1. \tag{B.14}$$

We choose identity tails, i.e. $\delta^{(1)} = \delta^{(b+1)} = 1$.

B.3.5. Toy scaling experiment

Toy data distribution Our goal is to create a family of distribution $p(x)$ which naturally extend over different dimensions D by the continuous mixture of Gaussians equation (6.21) where the mean of each dimension is conditioned on the previous via $m_i(x_j \in A_i)$. In particular, we choose:

$$m_1 = \frac{1}{2}, \quad m_0 = 0, \quad \sigma_1^2 = 0.8, \quad \sigma_2^2 = 0.2. \tag{B.15}$$

The $s_{ij}(D)$ are drawn randomly from $\{-1, +1\}$ for each dimension D and for each seed. This is the main source of noise between runs.

Training We employ the architecture described in appendix B.3.4 for $L_{\text{train}} = 64$ layers. We use $N = 60,000$ training samples in each case and run a total of four runs per case (except for case 1, for which we average over eight runs). Before training, the data is normalized to zero mean and unit standard deviation in the original rotation.

In principle, measuring convergence rates would be more accurate by using different batch of training data for fitting each layer, as this avoids overfitting and yields slower convergence on test data. We find this not to be a problem for the considered 64 layers and high α (see appendix B.3.4).

Evaluation We compute the number of required layers for each run via the procedure in appendix B.3.1. This number is then averaged over all runs for each case and dimension.

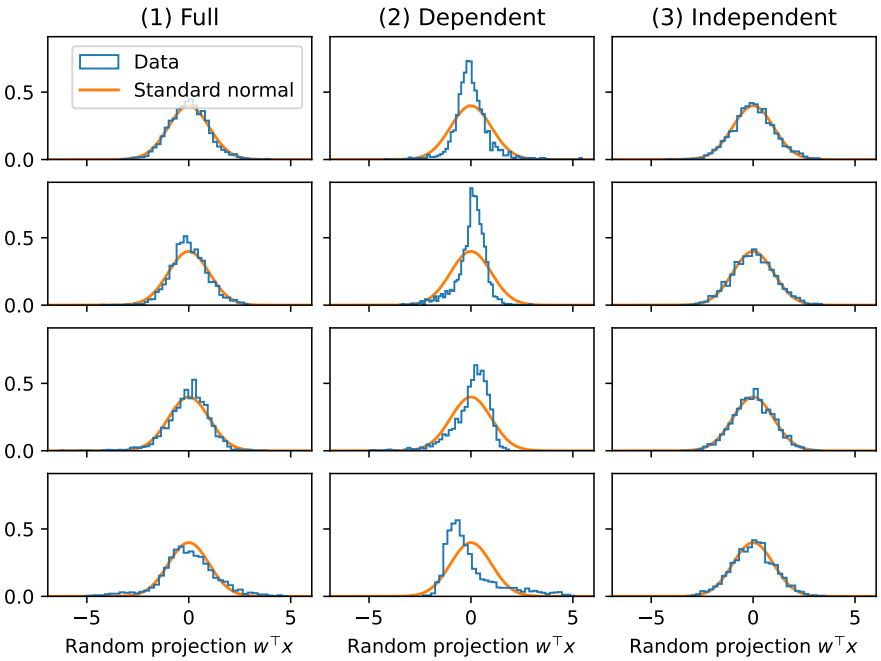


Figure B.6.: Random projections of data from the different versions of toy data in $D = 128$. Columns show the different cases, and each row depicts a different random projection $w \in \mathbb{R}^D, \|w\| = 1$. In case (2, *center*), most dimensions depend on the same dimensions, making random projections deviate strongly from a standard normal. For case (1) and (3), random projections are very close to a Gaussian, preventing a fast fit using Gaussianization.

B.3.6. Multiscale EMNIST experiment

Data distribution As described in section 6.2.5, we make use of a normalizing flow as our data density $p(x)$. Each flow architecture consists of a fixed normalization layer, followed by 20 affine coupling blocks. We use purely full-connected networks for odd scales, and convolutional networks for even scales. We use wavelet downsampling before the first. If the image side length scale is a power of 4, we add a second wavelet downsampling after the eighth affine coupling layer. Each convolutional subnetwork uses two hidden layers with 16 channels respectively 32 channels after the second downsampling each, and a kernel size of 3. The final 4 coupling blocks are fully-connected. Each fully-connected subnetwork has two hidden layers with hidden width equal to the total number of dimensions D. The details for each architecture are given in table B.1.

We train the normalizing flows for 30 epochs for $D = 28 \times 28$, and 20 for the other scales using negative log-likelihood, see equation (6.10). We use Adam with a learning rate of 10^{-3} and a batch size of 256.

After training, we replace the latent distribution with a normal distribution $\mathcal{N}(0, \hat{\sigma}^2 I_D)$ with $\hat{\sigma} = 0.8$ to achieve a better sample quality. Note that this does not influence the ability to compute density estimates $p(x)$ from our model, but it does reduce the entropy of the data.

Traininig Like in the toy experiment, we use the implementation from appendix B.3.4. We again choose $N = 60,000$ training samples without resampling. We average each case over 10 runs.

Evaluation We use the same procedure as in appendix B.3.5.

Table B.1.: Normalizing Flow architecture as a function of image size. A purely fully-connected network is labeled by "fc", "conv" networks are partially convolutional.

Scale	Network type	# of downsamplings	# of parameters
$4 = 2 \times 2$	conv	1	149k
$9 = 3 \times 3$	fc	0	7k
$16 = 4 \times 4$	conv	2	194k
$25 = 5 \times 5$	fc	0	38k
$36 = 6 \times 6$	conv	1	164k
$49 = 7 \times 7$	fc	0	134k
$64 = 8 \times 8$	conv	2	235k
$100 = 10 \times 10$	conv	1	254k
$144 = 12 \times 12$	conv	2	406k
$196 = 14 \times 14$	conv	1	544k
$256 = 16 \times 16$	conv	2	861k
$324 = 18 \times 18$	conv	1	1M
$400 = 20 \times 20$	conv	2	2M
$484 = 22 \times 22$	conv	1	3M
$576 = 24 \times 24$	conv	2	4M
$676 = 26 \times 26$	conv	1	5M
$784 = 28 \times 28$	conv	2	6M

B.3.7. Compute and libraries

Experiments were performed on three workstations, each with a single high-end consumer GPU and CPU each. We build our code upon the following Python libraries: PyTorch (Paszke et al., 2019), PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019), Lightning Trainable (Kühmichel & Draxler, 2023), Tensorflow (Abadi et al., 2015) for FID score evaluation, Numpy (Harris et al., 2020), Matplotlib (Hunter, 2007) for plotting and Pandas (McKinney, 2010; The pandas development team, 2020) for data evaluation.

B.4. Deep coupling bound

This appendix is adapted from Draxler et al. (2022).

All experiments were carried out on a single AMD Ryzen 7 3700X 8-Core Processor together with

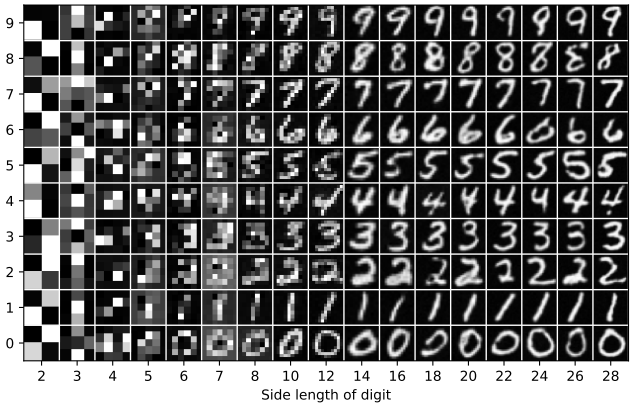


Figure B.7.: Samples from our down-scaled EMNIST normalizing flows which we use as ground truth distributions $p(x)$. Normalizing Flows are sampled with reduced temperature 0.8.

a NVIDIA GeForce RTX 2080.

B.4.1. Deep network on EMNIST

In this experiment, we estimate the capability of affine normalizing flows in reducing the non-Standardness S (see equation (6.1)) as a function of the number of layers. We compare this to the theoretic bound in theorem 6.7.

To this end, we train affine normalizing flows on EMNIST digits (Cohen et al., 2017). We leverage a 20-block Glow architecture. To measure the effect of depth $L = 1, \dots, 20$ of the flow on S, we truncate the architecture to L layers.

The architecture is built as follows: We start by down-sampling the input image from gray scale $1 \times 28 \times 28$ to $4 \times 14 \times 14$: Each group of four neighboring pixels is reordered into one pixel with four times the channels in a checkerboard-like pattern. Then, eight convolutional coupling blocks with 16 hidden channels are applied. They are followed by another down-sampling to $16 \times 7 \times 7$ and eight convolutional coupling blocks with 32 hidden channels. After flattening the input, four fully-connected affine coupling blocks are added with 392 hidden dimensions.

When truncating this architecture, we remove blocks *from the left*. For example, when one block is present ($L = 1$), only the last coupling block with the fully-connected subnetwork remains. This makes our theory applicable, as proposition 6.6 assumes that the neural networks s and t are fully-connected (otherwise, the whitening operation cannot always be represented).

We train each depth from scratch for 300 epochs using Adam with a learning rate of $3 \cdot 10^{-3}$ which is reduced by a factor of .1 after 100 and 200 epochs. The batch size is 240 which implies 1000 iterations per epoch.

Given the 20 networks of different depth, we split the loss into the non-Gaussianity $\mathcal{G}$ and non-Standardness S as suggested by theorem 4.1. To do so, we compute the empirical covariances Σ_l of 10'000 test samples pushed through each flow.

To relate this experiment to our theory, we take the covariance matrices obtained using the trained flows Σ_l and apply theorem 6.7 to each. This yields an upper bound on the expected non-Standardness after training another network with depth increased by one. In other words, given Σ_l, theorem 6.7 predicts an upper bound on the expected $\mathbb{E}_{Q_{l+1} \sim p(Q)}[S(\Sigma_{l+1})]$. We observe that the experimentally observed non-Standardness behaves similar to the upper bound. We do not expect this to be the case in general: There might be a trade-off between reducing S and $\mathcal{G}$, so the optimization might actually decide for reducing $\mathcal{G}$ at the cost of increasing S. We only show that with the covariance in proposition 6.6, $\mathcal{G}$ does not increase. On the other end, an affine flow might actually be able to reduce the non-Standardness stronger than predicted, as our theory does not take potentially useful cooperation between layers into account.

We average all results over eight runs per depth (i.e. $8 \cdot 20 = 160$ networks in total). Despite different random orientations in each run, the results are very concentrated: We find error bars so small that they are not visible in figure 6.1.

We observe that after four blocks, the non-Standardness is close to zero. Here, the flow consists of four coupling blocks with fully-connected subnetworks. This justifies the use of convolutional networks for s and t in the remaining blocks; they can only reduce correlations between pixels locally, thus not reducing non-Standardness as strongly as predicted. However, the non-Standardness is reduced enough by only four coupling blocks.

figure B.8 shows samples from one networks trained for each depth (sampling temperature $T = 0.7$).

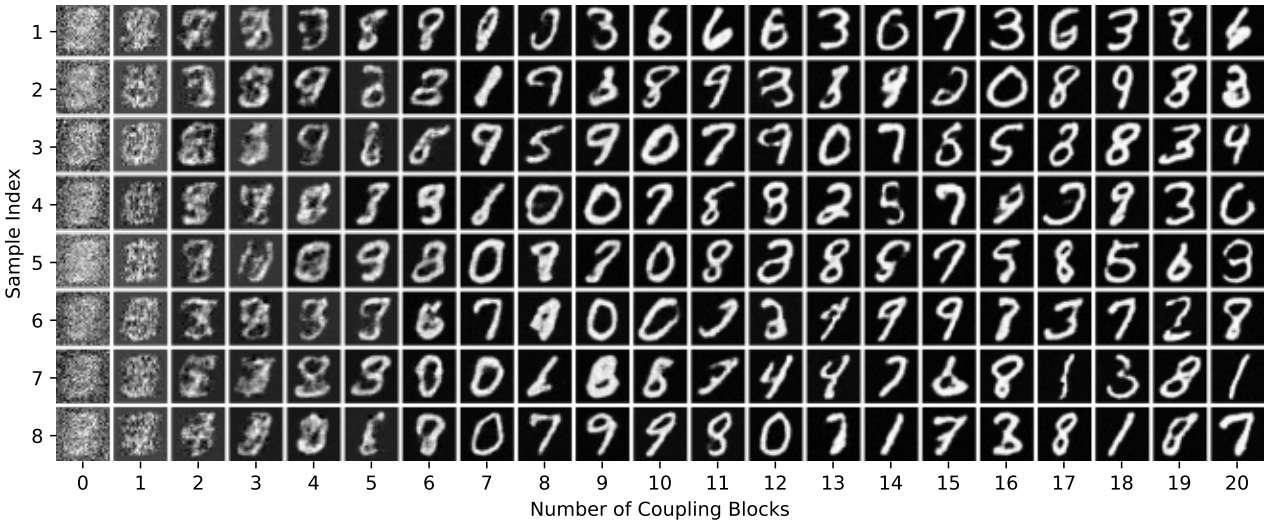


Figure B.8.: Samples generated by the affine coupling flows with varying depth trained for figure 6.1. Each column shows eight samples by a network of the corresponding depth.

B.4.2. Single layer on EMNIST digit covariance

This experiment confirms that the covariance minimizing the non-Standardness $S(\tilde{\Sigma})$ after a single layer is correctly predicted by proposition 6.6.

To get an interesting covariance matrix, we flatten the EMNIST digits training data and compute its covariance matrix Σ, as depicted in figure 6.9 on the left. We then sample a multivariate Gaussian with this covariance matrix and train a single affine coupling layer. As the data is Gaussian, we can train with the standard maximum likelihood loss as it is equivalent to the non-Standardness S. We use Adam with a learning rate 0.05, a batch size of 2048 and train for 512 iterations.

B.4.3. Single block on toy data

This experiment explores the average non-Standardness that can be reached by a single layer by modifying the covariance as given by proposition 6.6. It also aims to confirm the upper bounds shown in theorems 6.7 and 6.8.

We build a family of toy covariance matrices to work with. As the data will be randomly rotated anyway, we choose the matrices to be diagonal w.l.o.g., i.e. we directly design the eigenvalue spectrum of each covariance. We prescribe this spectrum by a continuous function $\mu : [0, 1] \to \mathbb{R}_+$. It is chosen bijective to ensure that the eigenvalues are distinct. We then define the eigenvalues as follows:

$$\mu_i = \mu\Big(\tfrac{i}{D-1}\Big) \quad i = 0, \dots, D-1. \tag{B.16}$$

With this approach, we can systematically modify eigenvalue/noise spectra.

Given a vector of eigenvalues $(\mu_i)_i$, we need to ensure that its mean is one. We do so by dividing by the mean:

$$\nu_i := \frac{\mu_i}{\sum_{i=1}^{D} \mu_i / D}. \tag{B.17}$$

Finally, we add a scaling parameter $s > 0$ that defines how close the spectrum is to the identity:

$$\lambda_i^{(s)} := (\nu_i - 1) \cdot s + 1. \tag{B.18}$$

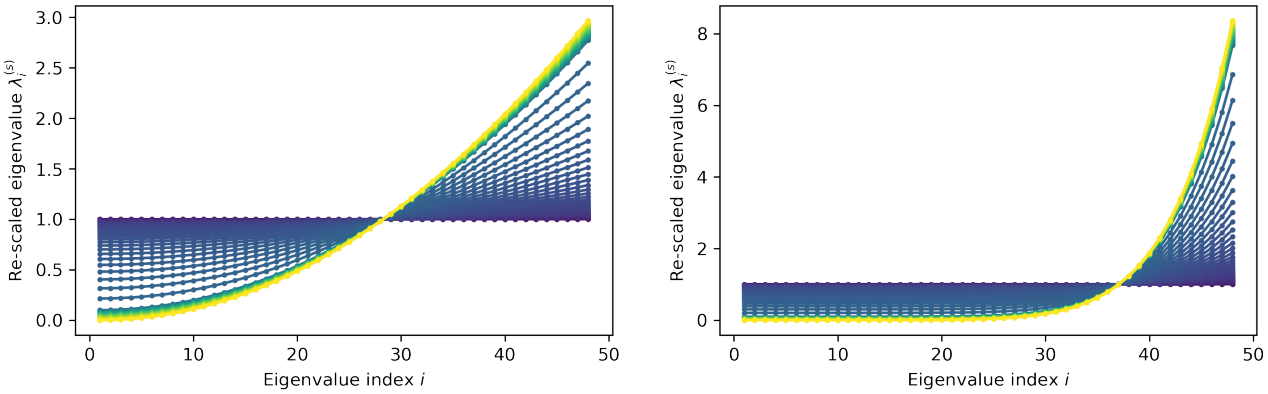


Figure B.9.: Eigenvalue spectra used for experiment depicted in figure 6.10. *(Left)* $\mu(x) = x^2$ and *(right)* $\mu(x) = x^8$. Each line corresponds to a different scaling s.

The non-Standardness strictly decreases as s comes closer to 0. As the eigenvalues always have to be positive, s must be chosen smaller than:

$$s < \frac{1}{1 - \lambda_{\min}} =: s_{\max}. \tag{B.19}$$

Given a spectrum $\lambda_i^{(s)}$, we build a diagonal covariance matrix

$$\Sigma = \mathrm{Diag}(\lambda_i^{(s)})_{i=1}^{D}. \tag{B.20}$$

For the experimental baseline, we sample N_{rot} orthogonal and unitary rotation matrices $Q \sim p(Q)$ from the corresponding Haar measure over $O(D)$ and $U(D)$. We employ `scipy.stats.ortho_group` respectively `scipy.stats.unitary_group` (Virtanen et al., 2020). This yields the covariance of the rotated data:

$$\Sigma_0 = Q\Sigma Q^{\mathrm{T}}. \tag{B.21}$$

(Or, Q^* instead of Q^{T} if we average over unitary matrices).

We do not train affine coupling layers directly. Instead, we make use of the single layer output covariance $\tilde{\Sigma}$ from proposition 6.6.

We choose the following numerical values for s: To get a close look at the case where $s \to 0$ and correspondingly $S \to 0$, we take $N_{\text{scale}}/3$ geometrically spaced points in $[0.001 s_{\max}, 0.9 s_{\max}]$. To accurately capture the off-minimum behavior, we add to that $2N_{\text{scale}}/3$ linearly spaced points between $[0.9 s_{\max}, .999 s_{\max}]$.

We choose $N_{\text{rot}} = 100$ and $N_{\text{scale}} = 150$ for all experiments. To save computational resources, we re-use the rotations sampled for the first scale for the remaining.

In figure 6.10, we showed the experiment for the parameterized spectra $\mu(x) = x^2$ and $\mu(x) = x^8$. For both, figure B.9 shows which rescaled eigenvalue spectra were used in this experiment. In figure B.10, we give examples for more spectra.

B.4.4. Layer-wise training on toy data

In this experiment, we track the non-Standardness as layers are added, check theorem 6.9, and compare the convergence rate equation (6.35) to the other bounds in theorems 6.7 and 6.8.

This experiment uses a different set of toy covariances than appendix B.4.3. This time, we build a plethora of different initial covariances (eigenvalue spectra) that include extreme cases:

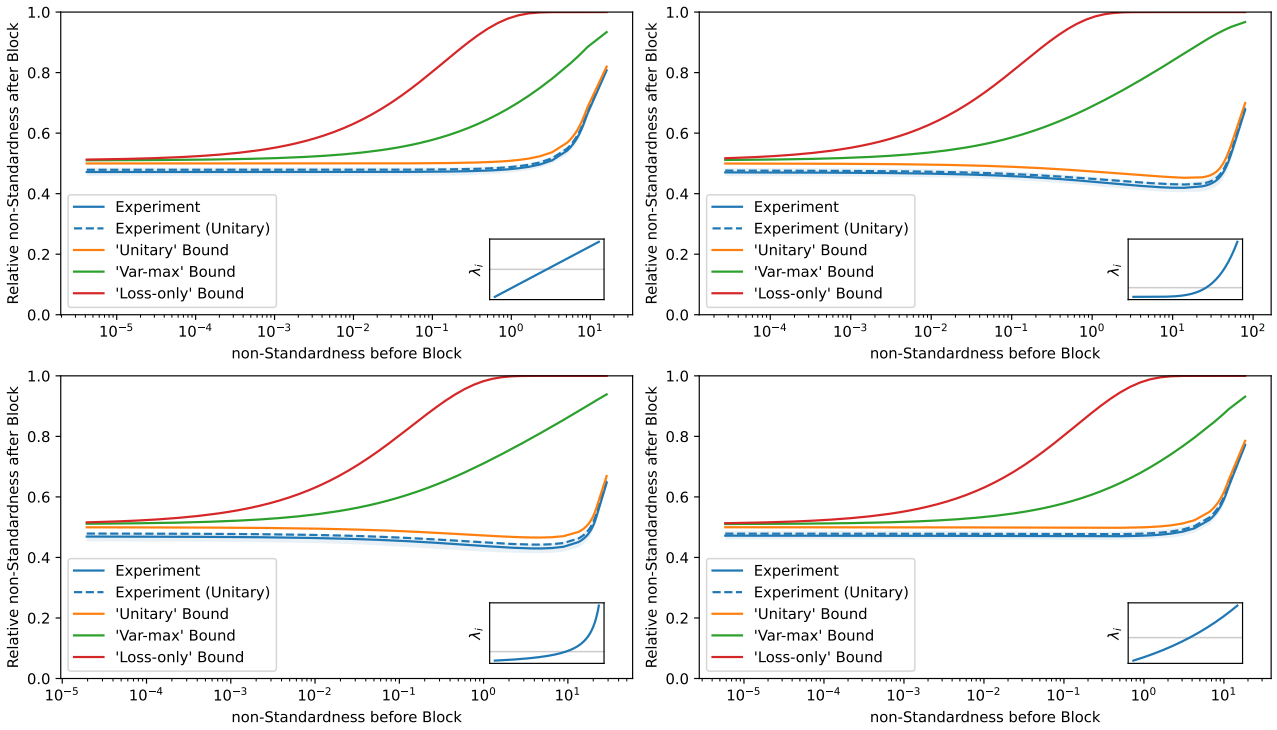


Figure B.10.: Examples for single layer relative non-Standardness on more eigenvalue spectra: *(Top left)* $\mu(x) = x$, *(top right)* $\mu(x) = x^5$, *(bottom left)* $\mu(x) = \frac{1}{1.1-x}$, *(bottom right)* $\mu(x) = \exp(x)$. More details in figure 6.10.

1. All eigenvalues are set to 1 except for one that is varying.
2. All eigenvalues have the same value that is varied, except for one that is set to 1.
3. Split the eigenvalues into two halves, respectively having the same value: The first half is varied, the second half assume the inverse value of the first half.
4. Randomly sample all eigenvalues uniformly from $[0, 2]$.
5. Randomly sample all eigenvalues between such that the logarithm is uniformly distributed over $[1/v_{\max}, v_{\max}]$.

Whenever we vary the value of any eigenvalue, we take N_{vary} scalars geometrically spaced between $1/v_{\max}$ and $v_{\max}$. We exclude the case where all eigenvalues are equal to 1, implying a non-Standardness of 0.

To fulfill assumption 6.3, we do not actually assign the same value to eigenvalues, but multiply them each with a linearly increasing factor in $(1-\epsilon, 1+\epsilon)$. We do not observe any change in experimental behavior from this, but this allows us evaluating theorem 6.7.

Given the dataset of eigenvalues, we build diagonal covariances, repeatedly apply random rotations and the whitening procedure in proposition 6.6. The details are given in algorithm 4. For each input covariance, we obtain N_{rot} trajectories of covariances.

We evaluate the non-Standardness of each covariance matrix $S(\Sigma_l^{(i,r)})$ and average over rotations. This is shown in the left plot in figure 6.11.

In addition, we compute the relative non-Standardness between layers:

$$S(\Sigma_l^{(i,r)})/S(\Sigma_{l-1}^{(i,r)}). \tag{B.22}$$

Algorithm 4 Multi-layer non-Standardness experiment

Require: Input covariances $\Sigma^{(i)}, i = 1, \dots, N$, number of rotations N_{rot}, number of layers L.
  $\Sigma_0^{(i,r)} \leftarrow \Sigma^{(i)}$ **for** $i = 1, \dots, N; r = 1, \dots, N_{\text{rot}}$ {Copy each input covariance N_{rot} times}
  for $l = 1, \dots, L$ **do**
    $Q^{(r)} \sim O(D)$ **for** $r = 1, \dots, N_{\text{rot}}$ {Sample rotations}
    $\Sigma_{l-1}^{(i,r)\prime} \leftarrow Q^{(r)} \Sigma_{l-1}^{(i,r)} (Q^{(r)})^{\mathrm{T}}$ **for** $i = 1, \dots, N; r = 1, \dots, N_{\text{rot}}$ {Apply rotations}
    $\Sigma_l^{(i,r)} \leftarrow$ proposition 6.6 on $\Sigma_{l-1}^{(i,r)\prime}$ {Apply whitening step}
  end for
Ensure: $\{\Sigma_l^{(i,r)}\}_{l=1}^{L}$ **for** $i = 1, \dots, N; r = 1, \dots, N_{\text{rot}}$.

This quantity is averaged over rotations r and instances i. It is depicted together with the corresponding interquartile range (IQR) in the right half of figure 6.11.

We also evaluate each of the bounds on $\mathbb{E}_Q[S(\Sigma_{l+1}^{(i,r)})]$ in theorems 6.7 and 6.8 given $\Sigma_l^{(i,r)}$ and divide it by the non-Standardness $S(\Sigma_l^{(i,r)})$. Again, we average over rotations and iterations.

Averaging over rotations might be counter-intuitive as the bounds explicitly calculate a value that is an average: It is necessary because for each initial covariance, we have N_{rot} trajectories with different convergence behavior. Let us make this explicit. Denote by B any of the bounds in theorems 6.7 and 6.8:

$$\frac{\mathbb{E}_{Q_{l+1} \sim p(Q)}[S(\Sigma_{l+1}^{(i,r)}(Q_{l+1}))]}{S(\Sigma_l^{(i,r)})} \leq \frac{B(\Sigma_l^{(i,r)})}{S(\Sigma_l^{(i,r)})}. \tag{B.23}$$

We average the quantity on the right over the different trajectories, i.e. over i, r. It only depends on the covariances in the lth layer in contrast to the expression on the left.

As hyperparameters to the experiment, we choose $D = 48, L = 32, N_{\text{vary}} = 128, N_{\text{rot}} = 32, v_{\max} = 1000, \epsilon = 10^{-5}$. We stop a trajectory once the non-Standardness falls below 10^{-9} to avoid numerical instabilities.

B.5. Free-form flows

This appendix is adapted from Draxler et al. (2024a).

We sample the vectors in theorem 7.1 from $v \sim \mathcal{U}(\mathbb{S}_{D/2}^{D-1})$, reducing the variance of the trace in comparison to $v \sim \mathcal{N}(0, I)$, see the experimental details on FIF in appendix B.7.1 for more details.

B.5.1. Simulation-Based Inference

Our models for the SBI benchmark use the same ResNet architecture as the neural spline flows (Durkan et al., 2019) used as the baseline. It consists of 10 residual blocks of hidden width 50 and ReLU activations. Conditioning is concatenated to the input and additionally implemented via GLUs at the end of each residual block. We also define a simpler, larger architecture which consists of 2x256 linear layers followed by 4x256 residual blocks without GLU conditioning. We denote the architectures in the following as ResNet S and ResNet L. To find values for architecture size, learning rate, batch size and β we follow (Wildberger et al., 2023) and perform a grid search to pick the best value for each dataset and simulation budget. As opposed to (Wildberger et al., 2023) we run the full grid, but with greatly reduced search ranges, which are provided in Table B.2, which amounts to a similar budget. The best hyperparameters for each setting are shown in Table B.3. Notably, this table shows that our method oftentimes works well on the same datasets for a wide range different β values. The entire grid search

Table B.2.: Hyperparameter ranges for the grid search on the SBI benchmark. *We only perform the search over architecture size for the 100k simulation budget scenarios.

	Range
Reconstruction weight β	$10, 25, 100, 500$
Learning rate	$\{1, 2, 5, 10\} \times 10^{-4}$
Batch size	$2^2, ..., 2^8$
Architecture size	S, L*

Table B.3.: Hyperparamters found by the grid search for the SBI benchmark. Cells are split into the hyperparameters for all three simulation budgets, unless we use the same setting across all of them.

Dataset	Batch size	Learning rate	β	ResNet size
Bernouli glm	8/32/128	5×10^{-4}	25/25/500	S/S/L
Bernouli glm raw	16/64/32	$5/10/10 \times 10^{-4}$	25/25/50	S/S/L
Gaussian linear	8/128/128	$5/10/1 \times 10^{-4}$	25/500/500	S
Gaussian linear uniform	8/8/32	$5/2/5 \times 10^{-4}$	500/10/100	S/S/L
Gaussian mixture	4/16/32	$5/2/10 \times 10^{-4}$	10/500/25	S/S/L
Lotka Volterra	4/32/64	$10/10/5 \times 10^{-4}$	500/500/25	S
SIR	8/32/64	$10/10/5 \times 10^{-4}$	500/25/25	S
SLCP	8/32/32	5×10^{-4}	10/25/25	S/S/L
SLCP distractors	4/32/256	$5/10/5 \times 10^{-4}$	25/10/10	S
Two moons	4/16/32	$5/5/1 \times 10^{-4}$	500	S/S/L

was performed exclusively on compute nodes with "AMD Milan EPYC 7513 CPU" resources and took $\sim 14.500\text{h} \times 8$ cores total CPU time for a total of 4480 runs.

B.5.2. Molecule Generation

$E(n)$-GNN

For all experiments, we make use of the $E(n)$ equivariant graph neural network proposed by (Satorras et al., 2021b) in the stabilized variant in (Satorras et al., 2021a). It is a graph neural network that takes a graph (V, E) as input. Each node $v_i \in V$ is the concatenation of a vector in space $x_i \in \mathbb{R}^n$ and some additional node features $h_i \in \mathbb{R}^h$. The neural network consists of L layers, each of which performs an operation on $v^l = [x_i^l; h_i^l \to x_i^{l+1}; h_i^{l+1}]$. Spatial components are transformed *equivariant* under the Euclidean group $E(n)$ and feature dimensions are transformed *invariant* under $E(n)$.

$$\mathbf{m}_{ij} = \varphi_e\Big(\boldsymbol{h}_i^l, \boldsymbol{h}_j^l, d_{ij}^2, a_{ij}\Big), \tag{B.24}$$

$$\tilde{e}_{ij} = \varphi_{inf}(m_{ij}), \tag{B.25}$$

$$\boldsymbol{h}_i^{l+1} = \varphi_h\left(\boldsymbol{h}_i^l, \sum_{j \neq i} \tilde{e}_{ij}\mathbf{m}_{ij}\right), \tag{B.26}$$

$$\boldsymbol{x}_i^{l+1} = \boldsymbol{x}_i^l + \sum_{j \neq i} \frac{\boldsymbol{x}_i^l - \boldsymbol{x}_j^l}{d_{ij} + 1} \varphi_x\Big(\boldsymbol{h}_i^l, \boldsymbol{h}_j^l, d_{ij}^2, a_{ij}\Big) \tag{B.27}$$

Here, $d_{ij} = \|x_i^l - x_j^l\|$ is the Euclidean distance between the spatial components, a_{ij} are optional edge features that we do not use. The $\tilde{e}_{ij}$ are normalized for the input to φ_h. The networks $\varphi_e, \varphi_{inf}, \varphi_h, \varphi_x$ are learned fully-connected neural networks applied to each edge or node respectively.

Table B.4.: Hyperparameters used for the Boltzmann generator tasks. The format "A / B" specifies a two-step training.

	DW4	LJ13	LJ55
Layer count	20	8	10
Reconstruction weight β	10	200	500
Learning rate	0.001	0.001	0.001
Learning rate scheduler	One cycle	-	- / Exponential $\gamma = 0.99999$
Gradient clip	1	1	0.1
Batch size	256	256	48
Duration	50 epochs	400 epochs	300k / 450k steps

Latent distribution

As mentioned in section 7.1.4, the latent distribution must be invariant under the Euclidean group. While rotational invariance is easy to fulfill, a normalized translation invariant distribution does not exist. Instead, we adopt the approach in (Köhler et al., 2020) to consider the subspace where the mean position of all atoms is at the origin: $\sum_{i=1}^{N} x_i = 0$. We then place a normal distribution over this space. By enforcing the output of the $E(n)$-GNN to be zero-centered as well, this yields a consistent system. See (Köhler et al., 2020) for more details.

Boltzmann Generators on DW4, LJ13 and LJ55

We consider the two potentials

$$\text{Double well (DW):} \qquad v_{\texttt{DW}}(x_1, x_2) = \frac{1}{2\tau}\Big(a(d-d_0) + b(d-d_0)^2 + c(d-d_0)^4\Big), \tag{B.28}$$

$$\text{Lennard-Jones (LJ):} \qquad v_{\texttt{LJ}}(x_1, x_2) = \frac{\epsilon}{2\tau}\left(\left(\frac{r_m}{d}\right)^{12} - 2\left(\frac{r_m}{d}\right)^{6}\right). \tag{B.29}$$

Here, $d = \|x_1 - x_2\|$ is the Euclidean distance between two particles. The DW parameters are chosen as $a = 0, b = -4, c = 0.9, d_0 = 4$ and $\tau = 1$. For LJ, we choose $r_m = 1, \epsilon = 1$ and $\tau = 1$. This is consistent with (Klein et al., 2023) and we use their MCMC samples as data, of which we use 400k samples for validation and 500k for testing the final model.

We give hyperparameters for training the models in table B.4. We consistently use the Adam optimizer. While we use the $E(n)$-GNN as our architecture, we do not make use of the features h because the Boltzmann distributions in question only concern positional information. Apart from the varying layer count, we choose the following $E(n)$-GNN model parameters as follows: fully-connected node and edge networks (which are invariant) have one hidden layers of hidden width 64 and SiLU activations. Two such invariant blocks are executed sequentially to parameterize the equivariant update. We compute the edge weights $\tilde{e}_{ij}$ via attention. Detailed choices for building the network can be determined from the code in (Hoogeboom et al., 2022).

QM9 molecule generation

For the QM9 (Ruddigkeit et al., 2012; Ramakrishnan et al., 2014) experiment, we again employ a $E(3)$-GNN. This time, the dimension of node features h is composed of a one-hot encoding for the atom type and an ordinal value for the atom charge. Like (Satorras et al., 2021a), we use variational dequantization for the ordinal features (Ho et al., 2019), and argmax flows for the categorical features

Table B.5.: Boltzmann generator negative log-likelihood and sampling times, including the time to compute the density from the network Jacobians. Note that in all cases, the log prob could be distilled by a $E(3)$-invariant network with scalar output for faster density estimation. NLLs are due to (Klein et al., 2023). Errors are the standard deviations over runs. The other models are based on an ODE trained via maximum likelihood ($E(n)$-NF, (Satorras et al., 2021a)), and trained via Optimal Transport Flow Matching with (OT-FM) or without (E-OT-FM) equivariance-aware matching (Klein et al., 2023). $E(n)$-NF is too memory intensive to train on LJ55 efficiently. Expands table 7.1 in main text.

	NLL (↓)	Sampling time (↓)	
		Raw	incl. $\log q_\theta(x)$
		DW4	
$E(n)$-NF	1.72 ± 0.01	0.024 ms	0.10 ms
OT-FM	1.70 ± 0.02	0.034 ms	0.76 ms
E-OT-FM	1.68 ± 0.01	0.033 ms	0.75 ms
FFF	1.68 ± 0.01	0.026 ms	0.74 ms
		LJ13	
$E(n)$-NF	-16.28 ± 0.04	0.27 ms	**1.2** ms
OT-FM	-16.54 ± 0.03	0.77 ms	38 ms
E-OT-FM	-16.70 ± 0.12	0.72 ms	38 ms
FFF	**-17.09 ± 0.16**	**0.11** ms	3.5 ms
		LJ55	
OT-FM	-88.45 ± 0.04	40 ms	6543 ms
E-OT-FM	**-89.27 ± 0.04**	40 ms	6543 ms
FFF	-88.72 ± 0.16	**2.1** ms	**311** ms

(Hoogeboom et al., 2021). For QM9, the number of atoms may differ depending on the input. We represent the distribution of molecule sizes as a categorical distribution.

We again employ the $E(3)$-GNN with the same settings as for the Boltzmann generators. We use 16 equivariant blocks, train with Adam with a learning rate of 10^{-4} for 700 epochs. We then decay the learning rate by a factor of $\gamma = 0.99$ per epoch for another 100 epochs. We set reconstruction weight to $\beta = 2000$. We use a batch size of 64.

For both molecule generation tasks together, we used approximately 6,000 GPU hours on an internal cluster of NVIDIA A40 and A100 GPUs. A full training run on QM9 took approximately ten days on a single such GPU.

B.5.3. Ablation studies

We study the effect of different modifications to our method in an ablation study on the MINIBOONE dataset shown in Table B.6. Firstly, we train a classical normalizing flow, implemented as a coupling flow (Dinh et al., 2015) (Classical NF) and compare it to a model where the exact likelihood loss of equation (4.4) has been replaced by the trace estimator of equation (7.5), but still using the exact inverse (NF + trace estimator). The resulting NLL shows that using equation (7.5) provides a good estimate for the maximum likelihood objective, with only small deterioration due to the increased stochasticity. Next, we train two identical coupling flow networks as FFF, meaning we no longer use their invertibility and instead rely on the reconstruction loss of equation (7.8) to learn an inverse (INN as FFF). This shows that the coupling flow architecture is suboptimal compared to FFF, despite its guarantee of invertibility. Finally, we show that with the right architecture (ResNet as FFF/Transformer as FFF) we can reach and even outperform coupling flows. We also include an ablation study on the

Table B.6.: Ablation in terms of NLL on MINIBOONE: We start with a classical normalizing flow trained with exact likelihood, then add the trace estimator. Next, we compare different architectures trained as free-form flows. Finally, we compare using more Hutchinson samples K and varying batch size bs.

Setup	NLL
Classical NF (equation (4.4))	10.55
NF + trace estimator (equation (7.5))	10.60
INN as FFF	16.42
Transformer as FFF	10.10
ResNet as FFF, $K = 1, bs = 256$	10.60
ResNet as FFF, $K = 2, bs = 256$	10.18
ResNet as FFF, $K = 1, bs = 64$	12.96
ResNet as FFF, $K = 2, bs = 64$	12.30

effect of increasing the number of Hutchinson samples K vs. increasing batch size bs. Both measures reduce stochasticity of the optimizer and can lead to better performance, but we find that the effect of increasing batch size is more pronounced, and can lead to more improvements than increasing K at the same cost.

B.5.4. Software libraries

We build our code upon the following Python libraries: PyTorch (Paszke et al., 2019), PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019), Tensorflow (Abadi et al., 2015) for FID score evaluation, Numpy (Harris et al., 2020), Matplotlib (Hunter, 2007) for plotting and Pandas (McKinney, 2010; The pandas development team, 2020) for data evaluation.

B.6. Manifold free-form flows

This appendix is adapted from Sorrenson et al. (2023).

In accordance with the details provided in section 7.2.3, our approach incorporates multiple regularization loss components in addition to the negative log-likelihood objective. This results in the final loss expression:

$$\mathcal{L} = \mathcal{L}_{\text{NLL}} + \beta_{\text{R}}^{x/z}\mathcal{L}_{\text{R}}^{x/z} + \beta_{\text{U}}^{x/z}\mathcal{L}_{\text{U}}^{x/z} + \beta_{\text{P}}^{x/z}\mathcal{L}_{\text{P}}^{x/z}. \tag{B.30}$$

For each of the terms, there is a variant in x- and in z-space, as indicated by the superscript. In detail:

The first loss $\mathcal{L}_{\text{R}}$ represents the reconstruction loss:

$$\mathcal{L}_{\text{R}}^{x} = \mathbb{E}_{x\sim p_{\text{data}}}[d(x, g_\varphi(f_\theta(x))], \tag{B.31}$$
$$\mathcal{L}_{\text{R}}^{z} = \mathbb{E}_{x\sim p_{\text{data}}}[d(f_\theta(x), f_\theta(g_\varphi(f_\theta(x)))]. \tag{B.32}$$

Here $d(x, y) = \|x - y\|^2$ is the standard reconstruction loss in the embedding space. This could be replaced with a distance on the manifold. However, this would be more expensive to compute and since we initialize networks close to identity, the distance in the embedding space is almost equal to the shortest path length on the manifold.

To have accurate reconstructions outside of training data, we also add reconstruction losses for data uniformly sampled on the manifold, both for x and z:

$$\mathcal{L}_{\text{U}}^{x} = \mathbb{E}_{x\sim\mathcal{U}(\mathcal{M})}[d(x, g_\varphi(f_\theta(x))], \tag{B.33}$$

$$\mathcal{L}_{\mathrm{U}}^{z} = \mathbb{E}_{x\sim\mathcal{U}(\mathcal{M})}[d(z, f_\theta(g_\varphi(z))]. \tag{B.34}$$

Finally, we make sure that the function learned by the neural networks is easy to project by regularizing the distance between the raw outputs by the neural networks in the embedding space and the subsequent projection to the manifold (compare equation (7.21)):

$$\mathcal{L}_{\mathrm{P}}^{x} = \mathbb{E}_{x\sim p_{\mathrm{data}}}[\|g_\varphi(f_\theta(x)) - \tilde{g}_\varphi(f_\theta(x))\|^2], \tag{B.35}$$

$$\mathcal{L}_{\mathrm{P}}^{z} = \mathbb{E}_{x\sim p_{\mathrm{data}}}[\|f_\theta(x) - \tilde{f}_\theta(x)\|^2]. \tag{B.36}$$

If these superscripts are not specified explicitly in the following summary of experimental details, we mean $\beta^x = \beta^z$.

In all cases, we selected hyperparameter using the performance on the validation data.

B.6.1. Likelihood evaluation

Sampling from a trained model can be easily achieved by sampling from the latent distribution and performing a single pass through the decoder g_φ. However, to evaluate the likelihood of the test set under our model, as in section 7.1.3, we need to calculate the change of variable formula w.r.t. g_φ^{-1}

$$\log p_\varphi(x) = \log p(z = g_\varphi^{-1}(x)) + \log|R^{\mathrm{T}} g_\varphi^{-1\prime}(x) Q| + \tfrac{1}{2}\log\frac{|R^{\mathrm{T}} G_Z(g_\varphi^{-1}(x))R|}{|Q^{\mathrm{T}} G_X(x) Q|} \tag{B.37}$$

$$\approx \log p(z = f_\theta(x)) + \log|R^{\mathrm{T}} g_\varphi'(f_\theta(x))^{-1} Q| + \tfrac{1}{2}\log\frac{|R^{\mathrm{T}} G_Z(f_\theta(x))R|}{|Q^{\mathrm{T}} G_X(x) Q|}. \tag{B.38}$$

Here we used the approximation $f_\theta \approx g_\varphi^{-1}$. While it is expensive to compute for training, it is reasonably fast to compute during inference time. However, if the assumption of $f_\theta \approx g_\varphi^{-1}$ is not sufficiently fulfilled, the measured likelihoods might be inaccurate. We try to identify such cases by sampling points from $\mathcal{M}$ around the proposed latent point with small noise strength σ

$$\tilde{z} = \mathrm{proj}(f_\theta(x) + \mathcal{N}(0, \sigma)). \tag{B.39}$$

As inverse of the decoder g_φ we use the sample $\tilde{z}$ which results in the lowest reconstruction loss:

$$g_\varphi^{-1}(x) \approx \arg\min_{\tilde{z}} ||g_\varphi(\tilde{z}) - x||_2^2. \tag{B.40}$$

We also do this with samples drawn around $\tilde{x} = \mathrm{proj}(x + \mathcal{N}(0, \sigma))$ and uniformly from the manifold $\tilde{z} = \mathcal{U}(\mathcal{M})$. We note that in most cases, except for the earth datasets, the likelihoods computed in this way agree with $f_\theta \approx g_\varphi^{-1}$. In the case of the earth datasets, we note that the newly computed likelihoods now agree with observed model quality. Specifically, whenever reconstruction loss is low, we also see agreement between the likelihoods computed via f_θ and our approximation of g_φ^{-1} via sampling. Otherwise, the disagreement between f_θ and g_φ^{-1} and the resulting drop in model quality are correctly diagnosed. Therefore, for the all $\mathbb{S}^2$ experiments, we report the likelihoods computed by our approximation.

B.6.2. Special orthogonal group

To apply manifold free-form flows, we project an output matrix $R \in \mathbb{R}^{3\times3}$ from the encoder/decoder to the subspace of special orthogonal matrices by finding the matrix $Q \in SO(3)$ minimizing the Frobenius norm $\|Q - R\|_F$. This is known as the constrained Procrustes problem and the solution Q can be determined via the SVD $R = U\Sigma V^{\mathrm{T}}$ (Lawrence et al., 2019):

$$Q = USV^{\mathrm{T}}, \tag{B.41}$$

Table B.7.: Hyperparameter choices for the rotation experiments. β_{U} and β_{P} are the same for both the sample and latent space.

Hyperparameter	Value
Layer type	ResNet
Residual blocks	2
Inner depth	5
Inner width	512
Activation	ReLU
β_{R}^{x}	500
β_{R}^{z}	0
β_{U}	10
β_{P}	10
Latent distribution	uniform
Optimizer	Adam
Learning rate	5×10^{-3}
Scheduler	Exponential w/ $\gamma = 1 - 10^{-5}$
Gradient clipping	1.0
Weight decay	3×10^{-5}
Batch size	1,024
Step count	585,600
#Repetitions	3

Table B.8.: Dataset overview for the earth data experiments. Each dataset is split into 80% for training, 10% for validation and 10% for testing.

Dataset	Number of instances	Noise strength
Volcano	827	0.008
Earthquake	6120	0.0015
Flood	4875	0.0015
Fire	12809	0.0015

where the diagonal entries of Σ were sorted from largest to smallest and $S = \mathrm{Diag}(1, \ldots, 1, \det(UV^{\mathrm{T}}))$.

The special orthogonal group data set is synthetically generated. We refer to (De Bortoli et al., 2022) for a description of the data generation process. They use an infinite stream of samples. To emulate this, we generate a data set of 10^7 samples from their code, of which we reserve 1,000 for validation during training and 5,000 for testing. We vectorize the 3×3 matrices before passing them into the fully-connected networks. All training details are given in table B.7, one training run takes approximately 7 hours on a NVIDIA A40.

The data set is synthetically generated; of the $N = 100,000$ data points, we use 1% of for validation and hyperparameter selection and 5% for test NLL. Each run uses a different random initialization of weights.

B.6.3. Earth data

We follow previous works and use a dataset split of 80% for training, 10% for validation and 10% for testing. For the earth datasets, we use a mixture of 5 learnable Von-Mises-Fisher distributions for the target latent distribution. We base our implementation on the `hyperspherical_vae` library (Davidson et al., 2018). In order to stabilize training, we apply a small amount of Gaussian noise to every batch (see table B.8) and project the resulting data point back onto the sphere. Other training

Table B.9.: Hyperparameter choices for the earth data experiments. β_U and β_P are the same for both the sample and latent space.

Hyperparameter	Value
Layer type	ResNet
residual blocks	4
Inner depth	2
Inner width	256
Activation	sin
β_R^x	10^5
β_R^z	0
β_U	2×10^2
β_P	0
Latent distribution	VMF-Mixture ($n_\mathrm{comp} = 5$)
Optimizer	Adam
Learning rate	2×10^{-4}
Scheduler	onecyclelr
Gradient clipping	10.0
Weight decay	5×10^{-5}
Batch size	32
Step count	$\sim$ 1.2M
#Repetitions	5

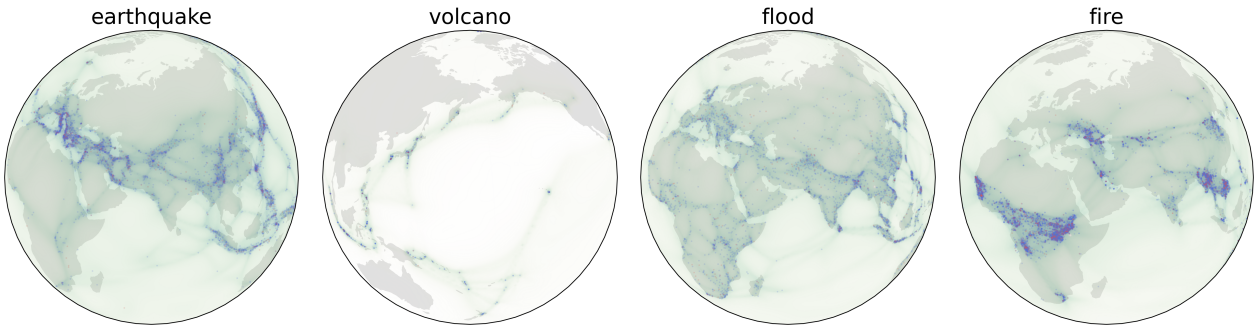


Figure B.11.: Density estimates of our model on the earth datasets. Blue points show the training dataset, red points the test dataset.

Table B.10.: Details on the torus datasets. Each dataset is randomly split into a train dataset (80%), validation dataset (10%) and test dataset (10%). During training, we add Gaussian noise with mean zero and standard deviation given by 'noise strength' to the data, to counteract overfitting.

Dataset	Number of instances	Noise strength
General	138208	0
Glycine	13283	0
Proline	7634	0
Pre-Proline	6910	0
RNA	9478	1×10^{-2}

Table B.11.: Details on the model architecture, loss weights and optimizer parameters for the torus datasets. We use the same configuration for all protein datasets on $\mathbb{T}^2$.

Hyperparameter	Value ($\mathbb{T}^2$)	Value ($\mathbb{T}^7$)
Layer type	ResNet	ResNet
residual blocks	6	2
Inner depth	3	2
Inner width	256	256
Activation	SiLU	SiLU
β_{R}^{x}	100	1000
β_{R}^{z}	100	100
β_{U}^{x}	100	100
β_{U}^{z}	0	1000
β_{P}	0	0
Latent distribution	uniform	uniform
Optimizer	Adam	Adam
Learning rate	1×10^{-3}	1×10^{-3}
Scheduler	oneclyclelr	onecyclelr
Gradient clipping	-	-
Weight decay	1×10^{-3}	1×10^{-3}
Batch size	512	512
Step count	$\sim$ 120k	$\sim$ 120k
#Repetitions	5	

hyperparameters can be found in table B.9. Each model trained around 20h on a compute cluster using a single NVIDIA A40.

B.6.4. Tori

The torus datasets are randomly split into a train dataset (80%), validation dataset (10%) and test dataset (10%). To counteract overfitting, we augment the RNA dataset with random Gaussian noise. The noise strength and total number of instances is reported in table B.10. We use a uniform latent distribution. We train for 120k steps with a batch size of 512 which takes 2.5 to 3 hours on a NVIDIA GeForce RTX 2070 graphics card. Further hyperparameters used in training can be found in table B.11.

B.6.5. Hyperbolic space

A straightforward way to define distributions on hyperbolic space (but also other Riemannian manifolds) is, to define a probability density p_{tangent} in the tangent space at the origin and use the exponential

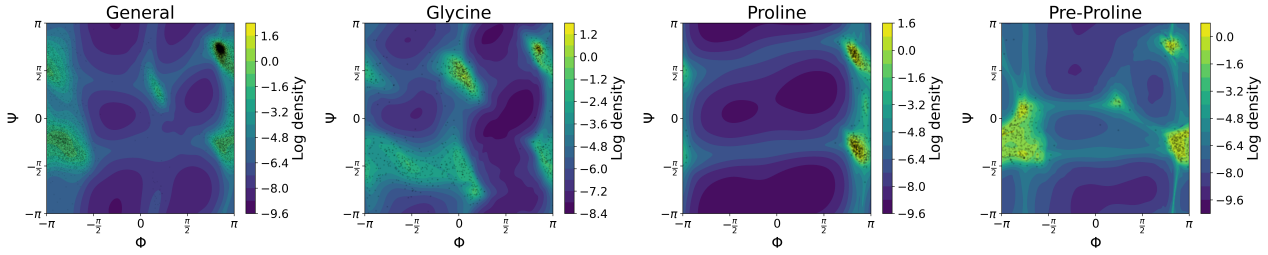


Figure B.12.: Log density of M-FFF models in the (Φ, Ψ)-plane of protein backbone dihedral angles (known as Ramachandran plot(Ramachandran et al., 1963)). The learned density matches the true density indicated by the test dataset (*black dots*) very well. Note also that the learned distribution obeys the periodic boundary conditions.

map $\exp_0$ to pushforward this distribution onto the manifold using equation (7.14):

$$\log p_{\text{manifold}}(\exp_0(v)) = \log p_{\text{tangent}}(v) - \log |J_{\exp_0}(v)| - \frac{1}{2} \log \frac{|G_{\text{manifold}}(\exp_0(v))|}{|G_{\text{tangent}}(v)|}, \tag{B.42}$$

where G_{manifold} denotes the metric tensor of the embedded manifold and G_{tangent} the metric tensor of the tangent space. This is also known as a 'wrapped' distribution.

We use the Poincaré ball model, which embeds the n-dimensional hyperbolic space $\mathbb{H}^n$ in the n-dimensional Euclidean space $\mathbb{R}^n$ as defined in table 7.4. The exponential map at the origin of this embedding and its Jacobian determinant are simply given by:

$$\exp_0(v) = \tanh(\|v\|) \frac{v}{\|v\|} \qquad \text{and} \qquad |J_{\exp_0}(v)| = \frac{\tanh(\|v\|)}{\|v\| \cosh^2(\|v\|)}. \tag{B.43}$$

The metric tensor at some point $p \in \mathbb{H}$ is defined by: $G^{ij}_{\mathbb{H}}(p) = \lambda_p^2 \delta_{ij}$ with $\lambda_p = 2/(1 - \|x\|^2)$. The metric tensor of the tangent space is the usual Euclidean metric tensor: $G^{ij}_{\mathbb{R}} = \delta_{ij}$.

With this at hand, we can define latent and toy distributions as wrapped distributions at the origin, as depicted in figure B.13.

For training, we sample 100k data points from each distribution. Hyperparameters for each model can be found in table B.12. Training takes approximately 2.5, 16, 8 and 16 hours on a NVIDIA A40 graphics card for the one Gaussian, five Gaussians, swish and checkerboard dataset respectively. The resulting model densities are shown in figure B.13.

B.6.6. 3D mesh

We base our experiment on the manifold and data provided by (Chen & Lipman, 2024) using 80% for training, 10% for validation and hyperparameter tuning. We report test NLL on the remaining 10% of the data. Each run starts from different parameter initialization. They give the manifold as a triangular mesh, consisting of vertices $v_i \in \mathbb{R}^3, i = 1, \dots N_v$ and triangular faces $f_j \in \{1, \dots, N_v\}^3$.

Since the projection to the nearest point on the mesh has zero gradient in parts of $\mathbb{R}^3$, we instead project to the manifold using a separately trained auto encoder with a spherical latent space. This autoencoder consists of an encoder $e : \mathbb{R}^3 \to \mathbb{R}^3$ consisting of five hidden layers with 256 neurons each, SiLU activations and an overall skip connection. The latent codes are computed by projecting the encoder outputs $e(x)$ to a sphere as $z(x) = e(x)/\|e(x)\|$, so that the latent space has the same the topology as the input mesh. Then, a decoder $d(z)$ with the same structure as the encoder is trained to reconstruct the original points by minimizing $\|x - d(e(x)/\|e(x)\|\|^2$. We train it for $2^{18} = 262,144$ steps, with each batch consisting of all $N_v = 2,502$ vertices and an additional $N = 2,502$ uniformly

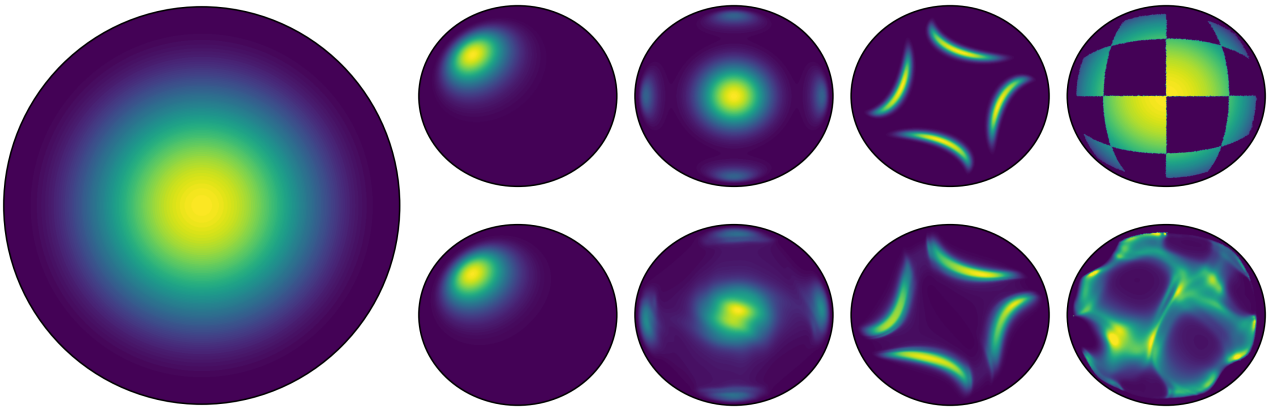


Figure B.13.: Density estimation on the Poincaré ball model. As the latent distribution we use a wrapped normal distribution with standard deviation 0.5 (*Left*). As target distributions (*top row*) we define several toy distributions in the tangent space at the origin and use equation (B.42) to push forward to the manifold. We will reference each distribution from left to right as 'one Gaussian', 'five Gaussians', 'swish' and 'checkerboard'. We train M-FFF on these target distributions using the full expression in equation (7.14) to compute the change in variables and evaluate the densities of the models (*bottom row*). M-FFF are capable to adapting to non-isometrically embedded manifolds. The learned densities on the one Gaussian, five Gaussians and swish dataset closely follow the target densities. On the checkerboard dataset, M-FFF cannot fully reproduce the sharp edges and density of the dataset.

Table B.12.: Details on the model architecture, loss weights and optimizer parameters for the Poincaré ball experiments.

Hyperparameter	Value (one wrapped)	Value (five gaussians)	Value (swish)	Value (checkerboard)
Layer type	ResNet	ResNet	ResNet	ResNet
residual blocks	2	6	6	6
Inner depth	2	3	3	3
Inner width	128	256	256	256
Activation	SiLU	SiLU	SiLU	SiLU
β_{R}^{x}	1000	1000	1000	1000
β_{R}^{z}	100	100	100	100
β_{U}^{x}	100	100	100	100
β_{U}^{z}	0	0	0	0
β_{P}	1	1	1	1
Latent distribution	Wrapped normal	Wrapped normal	Wrapped normal	Wrapped Normal
Optimizer	Adam	Adam	Adam	Adam
Learning rate	2×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}
Scheduler	Exponential w/ $\gamma = 0.9986$	onecyclelr	onecyclelr	onecyclelr
Gradient clipping	-	-	-	-
Weight decay	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}
Batch size	4096	4096	4096	4096
Step count	$\sim$ 84k	$\sim$ 485k	$\sim$ 240k	$\sim$ 495k

Table B.13.: Hyperparameter choices for the bunny experiments.

Hyperparameter	Value
Layer type	ResNet
Residual blocks	2
Inner depth	5
Inner width	512
Activation	ReLU
β_{R}^{x}	1000
β_{R}^{z}	0
β_{U}	10
β_{P}^{x}	100
β_{P}^{z}	10
Latent distribution	uniform
Optimizer	Adam
Learning rate	5×10^{-4}
Scheduler	Exponential w/ $\gamma = 1 - 0.0039$
Gradient clipping	1.0
Weight decay	3×10^{-5}
Batch size	1,024
Step count	469.199
#Repetitions	3

random points on the original mesh. We find that for successful training, it is helpful to filter out data with $x_2 > 0.5 + n/10,000$, where n is the step number. This prevents the long bunny ears from collapsing as the ears are slowly grown, allowing the model to adapt.

We then train M-FFF with the hyperparameters given in table B.13, using the pretrained autoencoder as our projection to the manifold. Note that we do not train the distribution on the latent sphere of the encoder, but directly on the manifold spanned by it. Training takes approximately 14 hours on a NVIDIA A40.

B.6.7. Libraries

We base our code on `PyTorch` (Paszke et al., 2019), PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019), Lightning Trainable (Kühmichel & Draxler, 2023), Numpy (Harris et al., 2020), Matplotlib (Hunter, 2007) for plotting and Pandas (McKinney, 2010; The pandas development team, 2020) for data evaluation. We use the `geomstats` (Miolane et al., 2020, 2023) package for embeddings and projections.

B.7. Free-form injective flows

This appendix is adapted from Sorrenson et al. (2024).

B.7.1. Implementation Choices

In implementing the trace estimator, we have to make a number of choices, each elaborated below. The main reasons for each choice are given first, with more technical details deferred to later in the appendix.

Gradient to encoder or decoder An equivalent variant to theorem 7.4 can be derived for the decoder, see the original publication for the derivation (Sorrenson et al., 2024). Empirically, we find that formulating it in terms of the encoder Jacobian leads to more stable training. Since the training also minimizes the squared norm of $f_\theta(x)$, we speculate that having gradient from this term and the log-determinant term both being sent to the encoder allows the encoder to more efficiently shape the latent space distribution. We note the similarity of this formulation to the standard change-of-variables loss used to train normalizing flows. If we instead send the gradient of the log-determinant to the decoder, the information about how the encoder can change can only reach it via the reconstruction term, which doesn't allow the encoder to deviate significantly from being the pseudoinverse of the decoder. A change in the decoder will therefore lead to a corresponding change in the encoder, but this is a less direct process than sending gradient to the encoder directly. In addition, this formulation means the decoder is optimized only to minimize reconstruction loss, meaning that it will likely be an approximate pseudoinverse for the encoder, a condition we require for the accuracy of the surrogate estimator.

Space in which trace is performed Considering equation (A.341), the central component of the surrogate is a trace (estimator). Making use of the cyclic property of the trace, i.e. $\mathrm{tr}(A^\mathrm{T}B) = \mathrm{tr}(BA^\mathrm{T})$ for any $A, B \in \mathbb{R}^{D\times d}$, we can choose which expansion of the trace to estimate:

$$\sum_{i=1}^{d}(A^\mathrm{T}B)_{ii} = \mathrm{tr}(A^\mathrm{T}B) = \mathrm{tr}(BA^\mathrm{T}) = \sum_{i=1}^{D}(BA^\mathrm{T})_{ii}. \tag{B.44}$$

The variance of a stochastic trace estimator depends on the noise used but in general is roughly proportional to the squared Frobenius norm of the matrix (see appendix B.7.1). Given two matrices $A, B \in \mathbb{R}^{D\times d}$ with $d < D$, it is likely that $\|A^\mathrm{T}B\|_F^2 < \|BA^\mathrm{T}\|_F^2$. This statement is not true for all A and B, but is almost always fulfilled when $d \ll D$.

Transferred to our context: In general the matrices $f'_\theta(x) \in \mathbb{R}^{d\times D}$ and $g'_\varphi(z) \in \mathbb{R}^{D\times d}$ are rectangular and can be multiplied together in either the $f'_\theta(x)g'_\varphi(z) \in \mathbb{R}^{d\times d}$ order or $g'_\varphi(z)f'_\theta(x) \in \mathbb{R}^{D\times D}$ order. This matters for applying the trace since generally $d < D$.

A more precise statement (proven in appendix B.7.1) is that if the entries of A and B are sampled from standard normal distributions, then $E[\|A^\mathrm{T}B\|_F^2] = Dd^2$ versus $E[\|BA^\mathrm{T}\|_F^2] = D^2d$. For $d \ll D$ the difference becomes significant. The difference between the two estimators may not be large if the two matrices have special structure, in particular if they share a basis. However, since the terms being multiplied in our case are a Jacobian matrix and the derivative of another Jacobian matrix with respect to a parameter θ_j or φ_j, it is unlikely that any such particular structure is present.

As a result, when the latent space is smaller than the data space, the preferable estimator is the one that performs the trace in the latent space, meaning that products in the estimator have the order $f'_\theta(x)g'_\varphi(z)$ (see table B.14). In appendix B.7.1, we experimentally test the convergence of trace estimators with increasing Hutchinson samples, performed in both data and latent space, confirming that convergence is much faster when performing the trace in latent space.

Type of gradient Consider the estimator:

$$\mathrm{tr}\left(\left(\frac{\partial}{\partial\theta_j}f'_\theta(x)\right)g'_\varphi(z)\right) = \frac{\partial}{\partial\theta_j}E_\epsilon\left[\epsilon^\mathrm{T}f'_\theta(x)\mathtt{SG}(g'_\varphi(z))\epsilon\right] \tag{B.45}$$

Ignoring the stop gradient operation for now, this requires computing terms of the form $\epsilon^\mathrm{T}f'_\theta(x)g'_\varphi(z)\epsilon$. In order to avoid calculating full Jacobian matrices, we can implement the calculation using some

Table B.14.: Different possible estimators for the gradient of the log-determinant term.

	gradient to encoder	gradient to decoder
trace in data space	$-\operatorname{tr}\left(g'_\varphi(z)\left(\frac{\partial}{\partial\theta_j} f'_\theta(x)\right)\right)$	$\operatorname{tr}\left(\left(\frac{\partial}{\partial\varphi_j} g'_\varphi(x)\right) f'_\theta(x)\right)$
trace in latent space	$-\operatorname{tr}\left(\left(\frac{\partial}{\partial\theta_j} f'_\theta(x)\right) g'_\varphi(z)\right)$	$\operatorname{tr}\left(f'_\theta(x)\left(\frac{\partial}{\partial\varphi_j} g'_\varphi(z)\right)\right)$

combination of vector-Jacobian (`vjp`) or Jacobian-vector (`jvp`) products, which are efficient to compute with backward-mode respectively forward-mode automatic differentiation. Note that we can use the result from one product as the vector for another `vjp` or `jvp`. For example, $v_1 := (\epsilon^\mathrm{T} f'_\theta(x))^\mathrm{T} \in \mathbb{R}^D$ yields a vector, so we can compute $\epsilon^\mathrm{T} f'_\theta(x) g'_\varphi(z)\epsilon = v_1^\mathrm{T} g'_\varphi(z)\epsilon$ via two vector-Jacobian products.

This gives us three choices: i) backward mode only (two `vjp`), ii) forward mode only (two `jvp`) or iii) a mix of both (one `jvp` and one `jvp`). We opt to use mixed mode (see appendix B.7.1 for further details).

Trace estimator noise Trace estimators rely on the identity $E_\epsilon[\epsilon^\mathrm{T} A\epsilon] = \operatorname{tr}(AE_\epsilon[\epsilon\epsilon^\mathrm{T}]) = \operatorname{tr}(A)$, meaning that we require only $E_\epsilon[\epsilon\epsilon^\mathrm{T}] = \mathbb{I}$ for the noise variable. The choice comes down mainly to the variance of the estimator. Among all noise vectors whose entries are sampled independently, Rademacher noise has the lowest variance (Hutchinson, 1989). However, if the entries are sampled from a standard normal distribution and then scaled to have length $\sqrt{d}$ where d is the dimension of ϵ, the entries are no longer independent and the variance of the estimator is comparable to Rademacher noise (Girard, 1989). When using a single Hutchinson sample, we choose to use scaled Gaussian noise for its low variance, and since it covers more directions than Rademacher noise (covering the hypersphere uniformly, rather than at a fixed 2^d points). When we have more than one Hutchinson sample, we additionally orthogonalize the vectors, as this further reduces variance. More details are in appendix B.7.1.

Number of noise samples We can choose to use between 1 and d noise samples in the trace estimator (with d samples we already can calculate the exact trace, so more samples are not necessary). Denote the number of samples by K. We find that in general $K = 1$ is enough for good performance, especially if the batch size is sufficiently high.

Variance of trace estimator

Theorem B.1. *Let $A, B \in \mathbb{R}^{D\times d}$ where the entries of both matrices are sampled from a standard normal distribution. Then*

$$E\left[\|A^\mathrm{T} B\|_F^2\right] = d^2 D \qquad \textit{and} \qquad E\left[\|BA^\mathrm{T}\|_F^2\right] = dD^2 \tag{B.46}$$

Proof. Consider first $\|A^\mathrm{T} B\|_F^2$. We can write this as

$$\|A^\mathrm{T} B\|_F^2 = \sum_{i=1}^{d}\sum_{j=1}^{d}\left(\sum_{k=1}^{D} A_{ki}B_{kj}\right)^2 = \sum_{i,j}^{d}\sum_{k,l}^{D} A_{ki}A_{li}B_{kj}B_{lj} \tag{B.47}$$

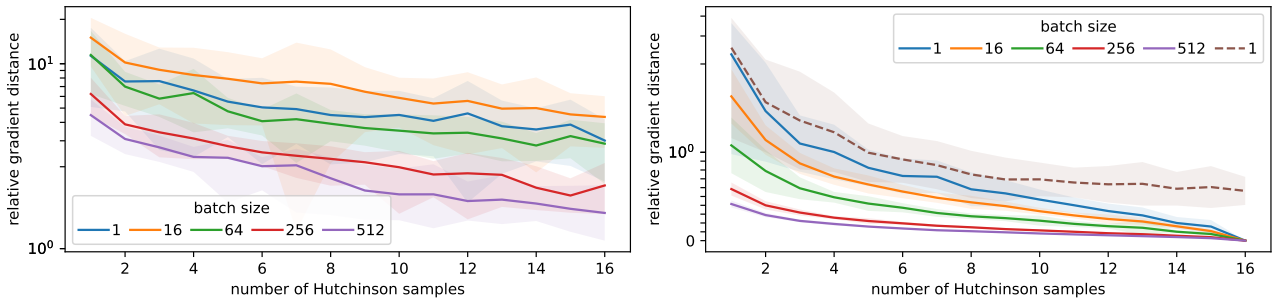


Figure B.14.: Relative gradient distance to the exact surrogate gradient as a function of Hutchinson samples for varying batch sizes. *(Left)* Trace estimation in data space. *(Right)* Trace estimation in latent space. We estimate the trace using orthogonalized Gaussian noise (see appendix B.7.1) (solid lines), but also feature non-orthogonalized samples for batch size 1 in latent space (dashed line). Note the different scales on the y-axes and the use of `symlog` in the right-hand plot. We evaluate the trace estimation on the surrogate estimator with gradient to encoder as specified in table B.14 for a converged model trained on conditional MNIST ($d = 16$, $D = 784$).

Taking an expectation over this expression, the only nonzero contributions will be from terms where the A and B terms are both quadratic, since if not, the term will be multiplied by $E[X] = 0$ where X is standard normal. This requires $k = l$, giving

$$E\left[\|A^{\mathrm{T}}B\|_F^2\right] = \sum_{i,j}^{d}\sum_{k}^{D} E\left[A_{ki}^2 B_{kj}^2\right] = \sum_{i,j}^{d}\sum_{k}^{D} E\left[A_{ki}^2\right] E\left[B_{kj}^2\right] = d^2 D \tag{B.48}$$

since A_{ki} and B_{kj} are independent and the expectation of the square of a standard normal variable is its variance, i.e. 1. The equivalent expressions for $\|BA^{\mathrm{T}}\|_F^2$ can be obtained by swapping d and D in these expressions. □

Experimental confirmation To evaluate the convergence behavior of the trace estimation, which is exact for d and D Hutchinson samples in latent and data space respectively, we compute the relative gradient distance of the resulting surrogate gradient with respect to the exact solution as a function of Hutchinson samples K:

$$\text{relative gradient distance}(K) = \frac{\|\nabla\text{surrogate}(K) - \nabla\text{exact}\|_2}{\|\nabla\text{exact}\|_2}. \tag{B.49}$$

Here, $\nabla\text{surrogate}(K)$ denotes the gradient of the surrogate loss term after K Hutchinson samples and ∇exact the gradient of the exact surrogate loss term, i.e. after d or D samples.

In figure B.14 one can see a clear decrease in gradient distance and its variance when computing the trace in latent space instead of data space. Furthermore, we note that increasing the batch size also contributes to fast and steady convergence, which is a result of sampling an independent noise sample per batch instance.

Forward/backward automatic differentiation

Two of the basic building blocks of automatic differentiation (autodiff) libraries are the vector-Jacobian product (`vjp`) and the Jacobian-vector product (`jvp`). The `vjp` is implemented by backward-mode autodiff and computes a vector multiplied with a Jacobian matrix from the left, along with the output of the function being used:

$$f_\theta(x),\ \epsilon^{\mathrm{T}} f'_\theta(x) = \texttt{vjp}(f_\theta, x, \epsilon) \tag{B.50}$$

In PyTorch, this is implemented by the `torch.autograd.functional.vjp` function, or by first computing $f_\theta(x)$, then using `torch.autograd.grad`.

The `jvp` is implemented by forward-mode autodiff and computes a vector multiplied by a Jacobian matrix from the right, along with the output of the function being used:

$$f_\theta(x),\ f'_\theta(x)\epsilon = \texttt{jvp}(f_\theta, x, \epsilon) \tag{B.51}$$

In PyTorch, this is implemented by the `torch.autograd.forward_ad` package.

Our preferred estimator for the log-determinant is the following (see section 7.3.3):

$$-\frac{1}{K}\sum_{k=1}^{K} \epsilon_k^{\mathrm{T}} f'_\theta(x)\texttt{SG}\Big(g'_\varphi(z)\epsilon_k\Big) \tag{B.52}$$

Therefore, we need to compute terms of the form $\epsilon^{\mathrm{T}} f'_\theta(x) g'_\varphi(z)\epsilon$, with a `SG` operation. The `SG` operation is implemented by applying the `.detach()` method to a tensor in PyTorch. We have the following options. Note that we can use a product obtained from `vjp` or `jvp` as a vector input to a subsequent product.

Backward mode This mode uses only vector-Jacobian products, requiring backward-mode autodiff.

1. $v_1^{\mathrm{T}} = \epsilon^{\mathrm{T}} f'_\theta(x)$ $\qquad z,\ v_1^{\mathrm{T}} = \texttt{vjp}(f, x, \epsilon)$
2. $v_2^{\mathrm{T}} = v_1^{\mathrm{T}} g'_\varphi(z)$ $\qquad \hat{x},\ v_2^{\mathrm{T}} = \texttt{vjp}(g, z, v_1)$
3. $\epsilon^{\mathrm{T}} f'_\theta(x) g'_\varphi(z)\epsilon = v_2^{\mathrm{T}}\epsilon$

Forward mode This mode uses only Jacobian-vector products, requiring forward-mode autodiff.

1. $v_1 = g'_\varphi(z)\epsilon$ $\qquad \hat{x},\ v_1 = \texttt{jvp}(g, z, \epsilon)$
2. $v_2 = f'_\theta(x) v_1$ $\qquad z,\ v_2 = \texttt{jvp}(f, x, v_1)$
3. $\epsilon^{\mathrm{T}} f'_\theta(x) g'_\varphi(z)\epsilon = \epsilon^{\mathrm{T}} v_2$

Mixed mode This mode uses one vector-Jacobian and one Jacobian-vector product, requiring both forward- and backward-mode autodiff.

1. $v_1^{\mathrm{T}} = \epsilon^{\mathrm{T}} f'_\theta(x)$ $\qquad z,\ v_1 = \texttt{vjp}(f, x, \epsilon)$
2. $v_2 = g'_\varphi(z)\epsilon$ $\qquad \hat{x},\ v_2 = \texttt{jvp}(g, z, \epsilon)$
3. $\epsilon^{\mathrm{T}} f'_\theta(x) g'_\varphi(z)\epsilon = v_1^{\mathrm{T}} v_2$

We prefer using backward mode autodiff where possible, since we find that it is slightly faster than forward mode in PyTorch. However, for our estimator of choice, we use mixed mode, since this is most easily implemented. Using backward mode would require a `SG` operation to be introduced after the second step, but in a way that allows gradient to flow back to $f'_\theta(x)$. While we believe this is possible if implemented carefully, we did not pursue this option. In mixed mode, we can easily detach the gradient of v_2 without affecting the first step of the calculation.

Properties of trace estimator noise

Hutchinson style trace estimators (Hutchinson, 1989) have the form $E_\epsilon[\epsilon^\mathrm{T} A\epsilon]$ and equal $\mathrm{tr}(A)$ in expectation. If A is skew-symmetric ($A^\mathrm{T} = -A$), then $(\epsilon^\mathrm{T} A\epsilon)^\mathrm{T} = \epsilon^\mathrm{T} A^\mathrm{T}\epsilon = -\epsilon^\mathrm{T} A\epsilon$ and hence $\epsilon^\mathrm{T} A\epsilon = 0$ with zero variance. Since any matrix can be decomposed into a symmetric and skew-symmetric part, the variance in the estimator comes only from the symmetric part of A, namely $A_s = (A + A^\mathrm{T})/2$. From now on, suppose A is symmetric and if not, substitute A_s for A.

Rademacher noise If the entries of ϵ are sampled independently from a distribution with zero mean and unit variance, then the variance of the estimator is minimized by the Rademacher distribution which samples the values -1 and 1 each with probability half. This estimator achieves the following variance for symmetric A (see proposition 1 in (Hutchinson, 1989)):

$$V_\epsilon[\epsilon^\mathrm{T} A\epsilon] = 2\sum_{i\neq j} A_{ij}^2 \tag{B.53}$$

Gaussian noise With standard normal noise, the estimator is unbiased, but the variance is higher (see again (Hutchinson, 1989)):

$$V_\epsilon[\epsilon^\mathrm{T} A\epsilon] = 2\sum_{i,j} A_{ij}^2 = 2\|A\|_F^2 \tag{B.54}$$

i.e. twice the Frobenius norm.

Scaled Gaussian noise By contrast:

$$E_\epsilon\left[\frac{\epsilon^\mathrm{T} A\epsilon}{\epsilon^\mathrm{T}\epsilon}\right] = \frac{1}{d}\,\mathrm{tr}(A) \tag{B.55}$$

where ϵ is a standard normal variable in $\mathbb{R}^d$. The variance of this estimator for symmetric A (see theorem 2.2 in (Girard, 1989)) is:

$$V_\epsilon\left[\frac{\epsilon^\mathrm{T} A\epsilon}{\epsilon^\mathrm{T}\epsilon}\right] = \frac{2}{d+2}\sigma^2(\lambda(A)) \tag{B.56}$$

where $\sigma^2(\lambda(A))$ denotes the variance of the eigenvalues of A.

We can write this estimator in the "Hutchinson" form by sampling ϵ from a standard normal distribution, then normalizing it such that its length is $\sqrt{d}$. Then we have

$$E_\epsilon[\epsilon^\mathrm{T} A\epsilon] = \mathrm{tr}(A) \tag{B.57}$$

and

$$V_\epsilon[\epsilon^\mathrm{T} A\epsilon] = \frac{2d^2}{d+2}\sigma^2(\lambda(A)) \tag{B.58}$$

Comparison When the dimension of A becomes large, the variance of Rademacher and scaled Gaussian estimators are comparable. Suppose that the eigenvalues of A have zero mean (e.g. the entries are independent normal samples). Then

$$d\sigma^2(\lambda(A)) = \sum_i \lambda_i^2 = \mathrm{tr}(A^2) = \|A\|_F^2 \tag{B.59}$$

If we further assume that all entries of A have roughly equal magnitude, we have that:

$$\sum_{i \neq j} A_{ij}^2 \approx \|A\|_F^2 \tag{B.60}$$

since the sum in the Frobenius norm is dominated by the $d(d-1)$ off-diagonal terms. Similarly,

$$\frac{d^2}{d+2}\sigma^2(\lambda(A)) \approx d\sigma^2(\lambda(A)) = \|A\|_F^2 \tag{B.61}$$

meaning that the two estimators have approximately the same variance.

If the matrix has special structure, we might choose one estimator over the other. For example, if the standard deviation of the eigenvalues of A is small in comparison to the mean eigenvalue, the scaled Gaussian estimator is preferable and if A is dominated by its diagonal then the Rademacher estimator is preferable. We don't expect either type of special structure in our matrices, so we consider the estimators interchangeable. We decided to use scaled Gaussian noise since it produces noise which points in all possible directions in $\mathbb{R}^d$ whereas Rademacher noise is restricted to a finite 2^d points. We assume that there is no reason to prefer this set of 2^d directions and therefore sampling from all possible directions is better.

Reducing variance when sampling more than 1 Hutchinson sample When the number of Hutchinson samples K are greater than 1, it is more favorable to sample the noise vectors in a dependent way than independently. Consider the case of a $d \times d$ matrix A with $K = d$. Then we can get an exact estimate of the trace via

$$\sum_{i=1}^{d} q_i^{\mathrm{T}} A q_i = \mathrm{tr}(Q^{\mathrm{T}} A Q) = \mathrm{tr}(A) \tag{B.62}$$

with orthogonal Q and q_i the i-th column of Q. If the q_i were sampled independently, we almost certainly wouldn't achieve this exact result. We therefore sample our noise vectors as the first K columns of a randomly sampled $d \times d$ orthogonal matrix and scale each column by $\sqrt{d}$. We show below that this reduces variance compared with sampling independently and make an experimental comparison (see figure B.14). If the resulting noise vectors are denoted ϵ_i, we estimate $\mathrm{tr}(A)$ by

$$\widehat{\mathrm{tr}}(A) = \frac{1}{K}\sum_{i=1}^{K} \epsilon_i^{\mathrm{T}} A \epsilon_i \tag{B.63}$$

This estimator is unbiased:

$$E_{\epsilon_1,\ldots,\epsilon_K}[\widehat{\mathrm{tr}}(A)] = \frac{1}{K}\sum_{i=1}^{K} E_{\epsilon_i}[\epsilon_i^{\mathrm{T}} A \epsilon_i] = \frac{1}{K}\sum_{i=1}^{K} \mathrm{tr}\Big(A E_{\epsilon_i}[\epsilon_i \epsilon_i^{\mathrm{T}}]\Big) = \frac{1}{K}\sum_{i=1}^{K} \mathrm{tr}(A) = \mathrm{tr}(A) \tag{B.64}$$

since $E_{\epsilon_i}[\epsilon_i \epsilon_i^{\mathrm{T}}] = \mathbb{I}$ for all ϵ_i.

This procedure is equivalent to using scaled Gaussian noise when $K = 1$ and is what we implement in practice for all values of K. Note that it is not necessary to use $K > d$ since we already achieve the exact value with $K = d$.

A note on our sampling strategy: in practice we sample by taking the Q matrix of the QR decomposition of a $d \times K$ matrix with entries sampled from a standard normal. Since the QR decomposition performs Gram-Schmidt orthogonalization, the Q matrix is uniformly sampled from the group of orthogonal matrices if Q is square (Mezzadri, 2007). The same logic applies to the QR decomposition of non-square matrices, yielding a $d \times K$ matrix made up of the first K columns of a

uniformly sampled orthogonal $d \times d$ matrix. Strictly speaking, the QR decomposition is only unique if the R matrix has a positive diagonal, and uniqueness is required for uniform sampling. Let $X = QR$ and define by D the sign of the diagonal of R. D is diagonal with 1 or -1 on the diagonal and $D^2 = \mathbb{I}$. Uniqueness can be achieved by multiplying Q by D from the right and multiplying R by D from the left: $X = QDDR = QR$. The resulting uniformly sampled orthogonal matrix is QD, meaning the columns of Q are multiplied by either 1 or -1. In our setting, we have terms of the form $\epsilon_i^{\mathrm{T}} A \epsilon_i$ where the ϵ_i are the columns of Q, so multiplying the columns by -1 has no effect on the trace estimate. As a result we opt not to multiply by D. Therefore, although we do not sample uniformly from the orthogonal group, the final result is equivalent to sampling uniformly.

Variance derivation Using the formula for the variance of a sum of random variables, we have that

$$V_{\epsilon_1,\ldots,\epsilon_K}[\widehat{\mathrm{tr}}(A)] = \frac{1}{K^2} \sum_{i,j=1}^{K} C_{\epsilon_i,\epsilon_j}[\epsilon_i^{\mathrm{T}} A \epsilon_i, \epsilon_j^{\mathrm{T}} A \epsilon_j] \tag{B.65}$$

where C denotes the covariance between two random variables. Note that since the permutation of the columns of a randomly sampled orthogonal matrix is arbitrary (permuting the columns results in another randomly sampled orthogonal matrix of equal probability), all columns are equivalent and we only have to distinguish between the cases $i = j$ and $i \neq j$. This leads to[1]

$$V_{\epsilon_1,\ldots,\epsilon_K}[\widehat{\mathrm{tr}}(A)] = \frac{1}{K^2}(Kv + K(K-1)c) = \frac{1}{K}(v + (K-1)c) \tag{B.66}$$

with $v = V_{\epsilon_i}[\epsilon_i^{\mathrm{T}} A \epsilon_i]$ and $c = C_{\epsilon_i,\epsilon_j}[\epsilon_i^{\mathrm{T}} A \epsilon_i, \epsilon_j^{\mathrm{T}} A \epsilon_j]$ when $i \neq j$. Each column viewed individually is a randomly sampled Gaussian vector, scaled to have length $\sqrt{d}$, hence the value of v is equal to the scaled Gaussian noise case above, namely

$$v = \frac{2d^2}{d+2}\sigma^2(\lambda(A)) \tag{B.67}$$

where $\sigma^2(\lambda(A))$ denotes the variance of the eigenvalues of A. We also know that the variance of the estimator reduces to zero when $K = d$ and hence $v + (d-1)c = 0$ leading to

$$c = -\frac{v}{d-1} \tag{B.68}$$

Putting it all together leads to

$$V_{\epsilon_1,\ldots,\epsilon_K}[\widehat{\mathrm{tr}}(A)] = \frac{1}{K}(v - \frac{K-1}{d-1}v) \tag{B.69}$$

$$= \frac{1}{K}\frac{d-K}{d-1}v \tag{B.70}$$

$$= \frac{2d^2(d-K)}{K(d-1)(d+2)}\sigma^2(\lambda(A)) \tag{B.71}$$

valid for $d > 1$. The comparable quantity for independently sampled scaled Gaussian noise is

$$V_{\epsilon_1,\ldots,\epsilon_K}[\widehat{\mathrm{tr}}(A)] = \frac{2d^2}{K(d+2)}\sigma^2(\lambda(A)) \tag{B.72}$$

[1] This argument is inspired by `https://math.stackexchange.com/questions/1081345/finding-variance-of-the-sample-mean-of-a-random-sample-of-size-n-without-replace`

i.e. $1/K$ times the $K = 1$ result. The orthogonalized noise strategy always has lower variance since the ratio between the variances is

$$\frac{d-K}{d-1} \leq 1 \tag{B.73}$$

which even reduces to zero when $K = d$. If d is large, the difference is not great for small K, which aligns with the fact that randomly sampled directions in $\mathbb{R}^d$ are close to orthogonal for large d.

B.7.2. Experimental Details

Role of reconstruction weight

Toy data To analyze the model behavior depending on the reconstruction weight β, we train the same architecture on a simple sinusoid data set with β varied between 0.01 and 100.

For the generation of data points, we draw x positions from a 1D standard normal distribution and calculate the respective y positions by $y = \sin(\pi x/2)$. Then, isotropic Gaussian noise with $\sigma = 0.1$ is added. We train an autoencoder architecture built with four residual blocks and a 1D latent space for 50 epochs with learning rate 0.001 until convergence. Each residual block is made up of a feed-forward network with one hidden layer of width 256. For each value of β, 20 models are trained.

To visualize the dimension of the data that is captured by the model, we project samples from the data distribution to the (1D) latent space and color the data points using the respective latent code as color value. Figure 7.8 illustrates that low reconstruction weight β values result in learning the dimension with the lowest entropy (noise) and higher values are required to learn the manifold that spans the sinusoid.

Additionally, we repeat the procedure with higher noise levels ($\sigma = 0.2, 0.3$). We observe that the point at which the model transitions from learning the noise to representing the manifold is not fixed, but depends on features of the data set such as the noise (figure B.15).

Conditional MNIST To measure the structure of the conditional MNIST dataset learned by the generating model g, we compute the full decoder Jacobian matrix J by calculating d Jacobian-vector products (one per column of the Jacobian). We compute the singular values $\Sigma = \mathrm{diag}(s_1, \ldots, s_d)$ of J, which – roughly speaking – indicate the stretching or shrinking of the latent manifold by g_φ. Hence, the number of non-vanishing singular values suggest the dimension of the data manifold and the sum of the log singular values is equal to the change in entropy between the latent space and data space, where a higher entropy indicates that more of the data manifold is spanned by the decoder. We can see this from the formula:

$$H[p_\varphi] = H[p(z)] + E_{z\sim p(z)}\left[\frac{1}{2}\log\det(g'_\varphi(z)^{\mathrm{T}} g'_\varphi(z))\right] \tag{B.74}$$

with H the differential entropy, and noting that $2\,\mathrm{tr}\log\Sigma = \log\det(g'_\varphi(z)^{\mathrm{T}} g'_\varphi(z))$.

We train multiple FIF models on conditional MNIST with reconstruction weights β ranging from 0.1 to 100, and evaluate their singular value spectra. The models are trained for 400 epochs, which is sufficient for convergence. The architecture used is the same as in table B.21, except that it has four times as many channels in each convolutional layer.

In figure B.16 it is clear that a higher reconstruction weight gives rise to a higher number of non-vanishing singular values. Hence, the reconstruction weight contributes towards learning structure of the true data manifold. This observed additional structure for higher reconstruction weights is reflected in an increasing diversity of samples (see figure B.17). Nevertheless, for high reconstruction weights we note the trade-off between sample diversity and properly learned latent distributions, which might result in out of distribution samples.

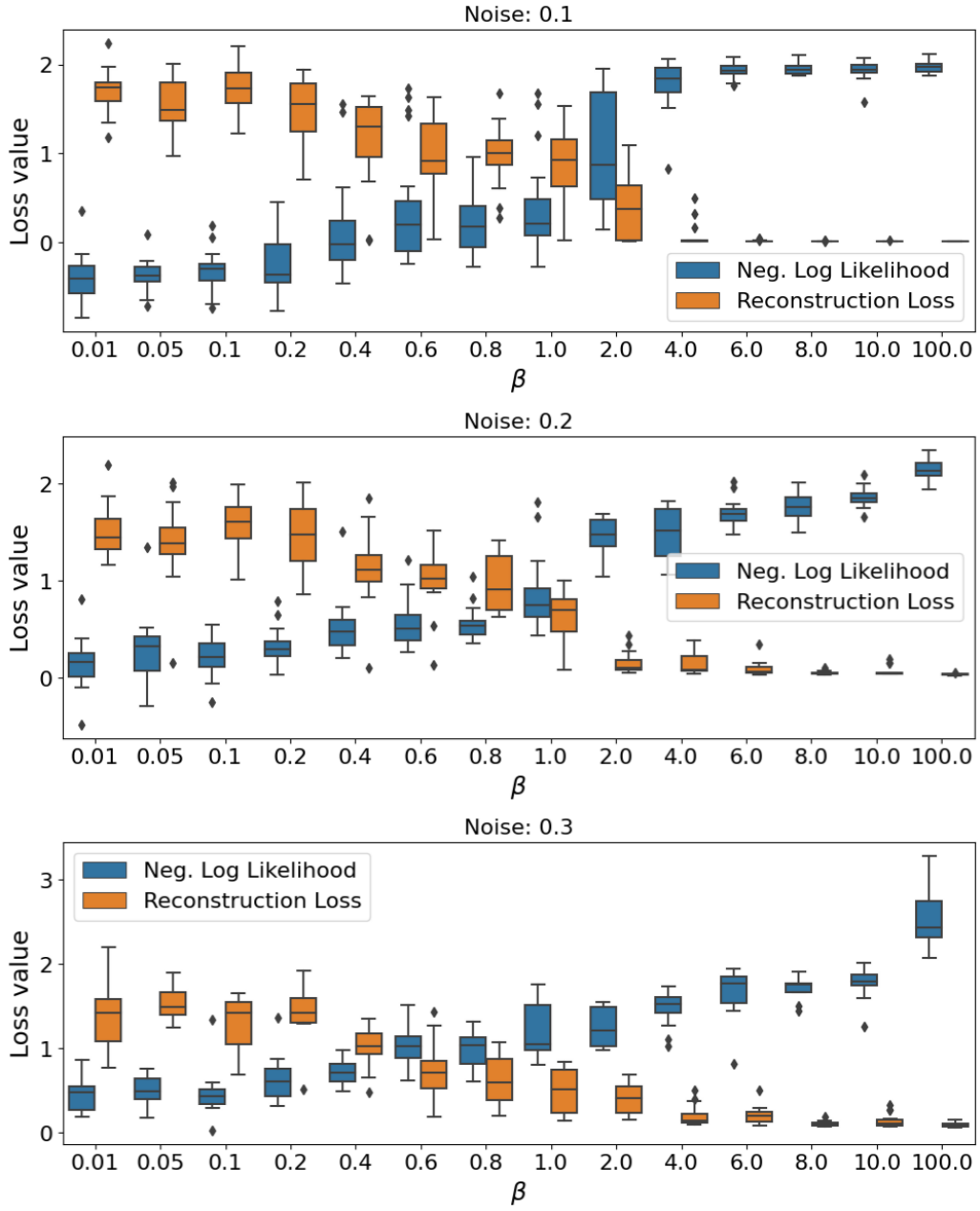


Figure B.15.: The position of the transition point depends on the data set. The plots show the tradeoff between reconstruction error and NLL with reconstruction weight β (box plots summarize 20 runs per condition). The point at which β becomes sufficiently large (transition point) shifts to lower values with increased Gaussian noise added to the data points.

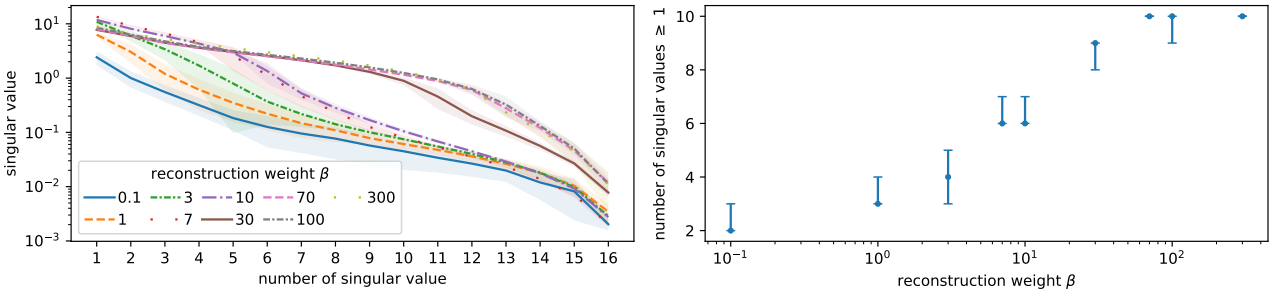


Figure B.16.: *(Left)* Singular value spectra for varying reconstruction weight. *(Right)* Number of singular values greater or equal to one as a function of reconstruction weight. The error bars show the span of the intersection of the shaded region with the line $y = 1$ in the left-hand plot, rounded down to the nearest integer. For each trained model, we generate 1024 samples per condition, compute the singular value spectra and average over all samples regardless of condition. The mean spectra and their standard deviation are evaluated across five trained models per reconstruction weight.

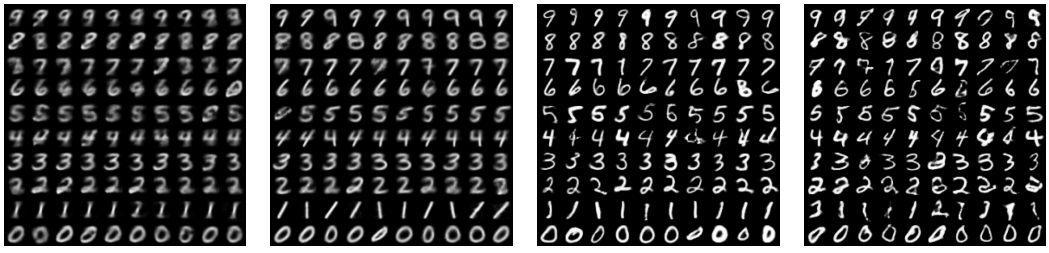


Figure B.17.: Conditional MNIST samples for varying reconstruction weight β. For each value of $\beta \in [0.1, 1, 10, 100]$ (from left to right), and each condition (rows) we generate ten samples (columns) at temperature $T = 1$.

Table B.15.: Dataset-dependent hyperparameters and average total runtime for tabular data experiments. We compare our runtime against the published rectangular flow (Caterini et al., 2021) runtimes (RNFs-ML ($K = 1$) model), as well as rerunning their code on our hardware.

Hyperparameter	POWER	GAS	HEPMASS	MINIBOONE
Latent dimension	3	2	10	21
Training epochs	15	30	85	875
Model	Training time (minutes)			
FIF (*ours*)	38	39	41	49
Rectangular flow (published)	113	75	138	34
Rectangular flow (our hardware)	147	86	249	75
Training time speedup (same hardware)	3.9 ×	2.2 ×	6.1 ×	1.5 ×

Tabular data

We compare to the tabular data experiments in (Caterini et al., 2021), using the same datasets and data splits, as well as the same latent space dimensions. We train models with roughly the same number of parameters. Our main architectural difference is that we use an unconstrained autoencoder rather than an injective flow. Our encoder consists of two parts: i) a feed-forward network with two hidden layers of dimension 256 and ReLU activations (no normalization layers), which maps from the input dimension to the latent dimension ii) a ResNet with two blocks, each with two hidden layers of dimension 256 and ReLU activations. The ResNet has input and output dimension equal to the latent space dimension. The decoder is the inverse: i) an identical ResNet to the encoder (but with separate parameters) followed by ii) a feed-forward network with two hidden layers of dimension 256 mapping from the latent space to the data space dimension.

We use a batch size of 512, add isotropic Gaussian noise with standard deviation 0.01, use $K = 1$ Hutchinson samples and a reconstruction weight $\beta = 10$ for all experiments. We use the Adam optimizer with the onecycle LR scheduler with LR of 10^{-4} (except for HEPMASS which has LR of 3×10^{-4}) and weight decay of 10^{-4}. The number of epochs was chosen such that all experiments had approximately the same number of training iterations. We ran the model 5 times per dataset. The dataset-dependent parameters and average training times are given in table B.15.

We compare our training times against the published rectangular flow training times for their RNFs-ML ($K = 1$) model, as well as rerunning their code on our hardware (a single RTX 2070 card). We find comparable FID-like scores on our rerun (except on GAS where we could not reproduce the score, see section 7.3.4), but our hardware is slower, with runs consistently taking at least 15% longer and more than twice as long on MINIBOONE. We find that our model runs in half the time or less of the rectangular flow on the same hardware, except for MINIBOONE (about 2/3 the time).

Comparison to existing injective flows

We compare against Trumpets (Kothari et al., 2021) and Denoising Normalizing Flows (DNF) (Horvat & Pfister, 2021), as they are the best-performing injective flows to our knowledge, and report performance on CelebA in table 7.11. Note that Trumpets default to $d = 192$, DNF to $d = 512$, whereas we can reduce the bottleneck dimension to $d = 64$ (consistent with the Pythae benchmark in appendix B.7.2).

Both models differ in the recommended wall clock time, and we therefore fix the wall clock time available to each model to five hours on a single NVIDIA A40. Trumpets train the manifold and the distribution on it in two sequential steps. To accommodate both steps in the reduced training time, we vary the fraction of the five hours spent in training manifold and distribution and report the best

Table B.16.: Ablation study on the effect of each component of our proposed improvement to rectangular flows (RF). By NLL estimator, we denote how the loss in equation 7.38 is approximated. For this experiment, we used our reimplementation of RF. The off-manifold estimator with a free-form architecture performs best in all cases.

Hyperparameters & Model	NLL estimator (on-/off-manifold)	POWER	GAS	HEPMASS	MINIBOONE
FIF & free-form net	off manifold (eq. 7.34)	**0.041 ± 0.007**	**0.281 ± 0.031**	**0.541 ± 0.034**	**0.598 ± 0.024**
FIF & free-form net	on manifold (eq. 7.34)	19.54 ± 20.81	7.48 ± 5.40	29.03 ± 5.42	77.23 ± 16.55
FIF & coupling flow	off manifold (eq. 7.37)	0.11 ± 0.06	0.45 ± 0.09	1.30 ± 0.14	1.55 ± 0.04
RF & coupling flow	off manifold (eq. 7.37)	0.98 ± 0.69	6.16 ± 4.20	2.02 ± 0.74	1.80 ± 0.10
FIF & coupling flow	on manifold (eq. 7.34)	3.71 ± 2.19	0.40 ± 0.22	0.71 ± 0.05	3.13 ± 0.42
RF & coupling flow	on manifold (eq. 7.34)	0.33 ± 0.22	0.33 ± 0.17	0.82 ± 0.07	1.84 ± 0.11

Table B.17.: Reconstruction losses of FIF with a free-form architecture on the POWER, GAS HEPMASS and MINIBOONE datasets. The reconstruction error is always much higher for on-manifold training compared to off-manifold, demonstrating the instability caused by on-manifold NLL evaluation in free-form networks. Note: the large standard deviations in on-manifold runs are typically the result of a single large outlier. We remove the largest outlier where applicable ("On Manifold (outliers removed)" row).

	POWER	GAS	HEPMASS	MINIBOONE
On manifold	237 ± 498	5835 ± 13006	119 ± 34	300 ± 160
On manifold (outliers removed)	14 ± 27	18 ± 31	119 ± 34	229 ± 23
Off manifold	0.072 ± 0.002	0.188 ± 0.012	2.569 ± 0.098	1.077 ± 0.011

FID among the variants tried. We vary the number of manifold epochs as $n_{\text{manifold}} = 2, 5, 10$, with 10 performing best.

Our free-form injective flows (FIF) are not restricted in their architecture, and we choose an off-the-shelf convolutional autoencoder, followed by a total of four fully-connected ResNet blocks, see table B.21. The fully-connected blocks are important, as can be seen when comparing to the architecture used in the Pythae benchmark (see appendix B.7.2). We note that the Pythae benchmark could benefit from a modified architecture, but leave this modification open for future work.

For Trumpets and DNF, we point to the training details provided by the respective authors. For FIF, we choose these training hyperparameters: We train with the Adam optimizer with a LR of 10^{-3} and a weight decay of 10^{-4}, linearly increase β from 20 at initialization to 40 at the end of training, a single Hutchinson sample $K = 1$ and a student-t distribution on the latent space. We set the batch size to 256.

We conclude that the Pythae benchmark could benefit from an optimized architecture, as this change probably also improves the other methods. From the data at hand, we further conclude that the full potential of FIF has not yet been exploited, and that easy gains can be made by improving the architecture and other hyperparameters.

Pythae benchmark on generative autoencoders

We compare our method to existing generative autoencoder paradigms using the benchmark from (Chadebec et al., 2022). We use the provided open-source pipeline and follow the training setup described by the authors. For MNIST and CIFAR10 this means training for 100 epochs with the Adam optimizer at a starting LR of 10^{-4}, reserving the last 10k images of the training sets as validation sets. CelebA trains for 50 epochs with a starting LR of 10^{-3}. All experiments are performed with a batch size of 100 and LR is reduced by half when the loss plateaus for 10 epochs. In accordance with the original benchmark, we pick 10 sets of hyper-parameters, compute their validation FID (see

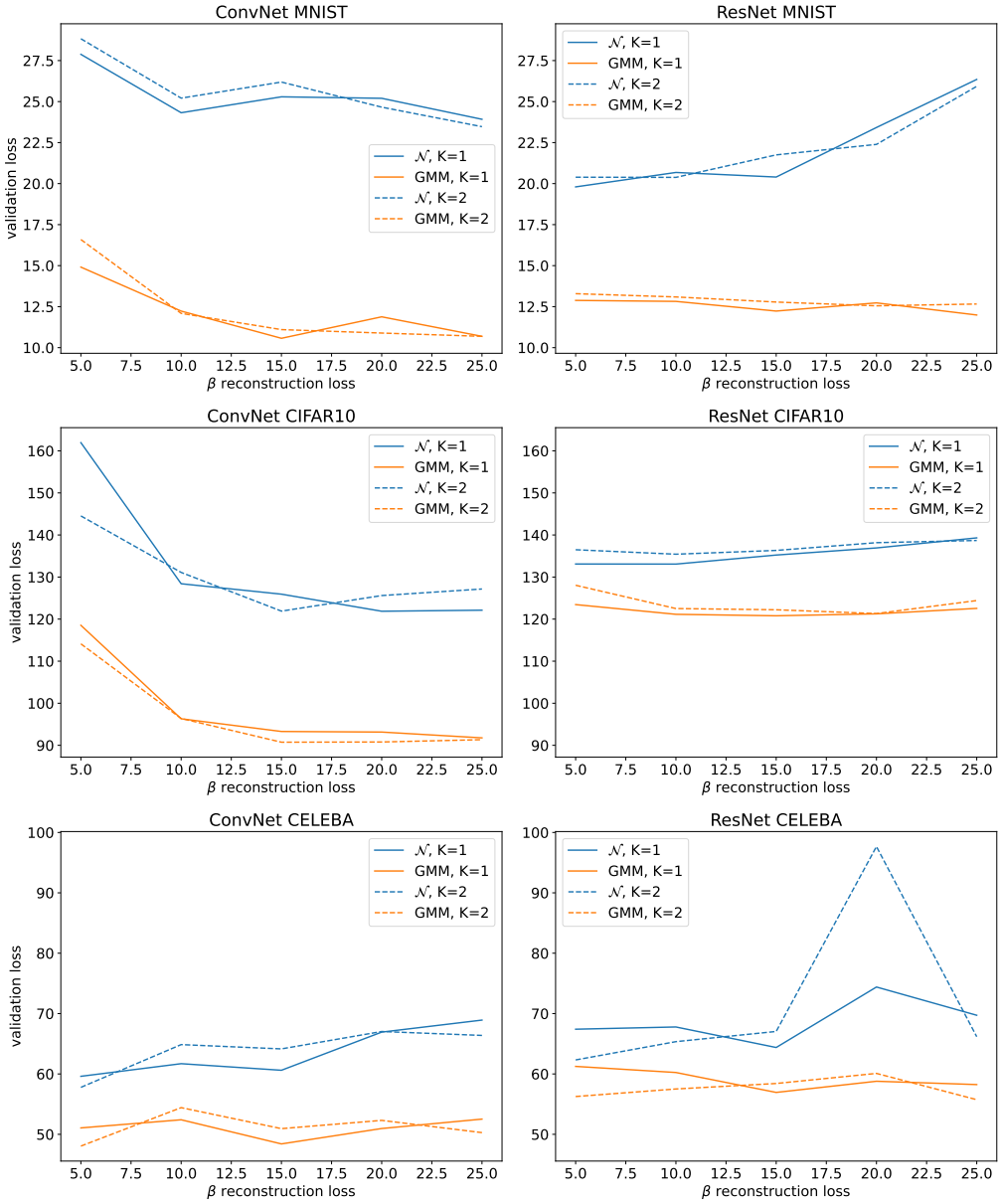


Figure B.18.: Validation FID of our 10 benchmark model setups on the different datasets and architectures used by Pythae. We report results for reconstruction weights $\beta = 5, 10, 15, 20, 25$ and number of Hutchinson samples $K = 1$ in the solid lines and $K = 2$ in the dashed lines. We show the performance of standard normal sampling ($\mathcal{N}$) and a Gaussian mixture model (GMM). We see that GMM sampling always improves performance. In contrast, the reconstruction weight and number of Hutchinson samples have no noticeable effect on performance, except that increasing β improves performance on ConvNet MNIST and ConvNet CIFAR10, and decreases performance on ResNet MNIST (normal sampling only).

figure B.18) and use the model which achieves the best FID on the validation set as the final model. In table B.18 we report the FID and IS of this model on the test set. To complement the metrics, we show samples from all models in table B.22, demonstrating convincing quality. We exclude the VAEGAN from the FID comparison, as the model trains more than double the time required for FIF and goes beyond fitting a transformation of the training data to a standard normal distribution.

As described in section 7.3.4, we use the architectures from (Chadebec et al., 2022) for the benchmark, which we replicate in tables B.19 and B.20.

Compute and dependencies

We used approximately 150 GPU hours for computing the Pythae benchmark, and an additional 800 GPU hours for model exploration and testing. The majority of the experiments were performed on an internal cluster of A100s. The majority of compute time was spent on image datasets.

We build our code upon the following Python libraries: PyTorch (Paszke et al., 2019), PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019), Tensorflow (Abadi et al., 2015) for FID score evaluation, Numpy (Harris et al., 2020), Matplotlib (Hunter, 2007) for plotting and Pandas (McKinney, 2010; The pandas development team, 2020) for data evaluation.

Table B.18.: Table taken from (Chadebec et al., 2022) with our results added at the top. We report Inception Score (IS) and Fréchet Inception Distance (FID) computed with 10k samples on the test set. The best model per dataset and sampler is highlighted in **bold**, the second best is underlined.

		ConvNet						ResNet					
Model	Sampler	MNIST		CIFAR10		CELEBA		MNIST		CIFAR10		CELEBA	
		FID ↓	IS ↑	FID	IS	FID	IS ↑	FID ↓	IS ↑	FID	IS	FID	IS
FIF (ours)	$\mathcal{N}$	23.8	2.2	121.0	3.0	56.9	2.1	19.5	2.1	132.6	2.9	**62.3**	1.7
	GMM	11.0	2.2	90.6	4.0	**47.3**	1.9	11.7	2.1	119.2	3.4	**55.0**	1.8
VAE	$\mathcal{N}$	28.5	2.1	241.0	2.2	54.8	1.9	31.3	2.0	181.7	2.5	66.6	1.6
	GMM	26.9	2.1	235.9	2.3	52.4	1.9	32.3	2.1	179.7	2.5	63.0	1.7
VAMP	VAMP	64.2	2.0	329.0	1.5	56.0	1.9	34.5	2.1	181.9	2.5	67.2	1.6
IWAE	$\mathcal{N}$	29.0	2.1	245.3	2.1	55.7	1.9	32.4	2.0	191.2	2.4	67.6	1.6
	GMM	28.4	2.1	241.2	2.1	52.7	1.9	34.4	2.1	188.8	2.4	64.1	1.7
VAE-lin NF	$\mathcal{N}$	29.3	2.1	240.3	2.1	56.5	1.9	32.5	2.0	185.5	2.4	67.1	1.6
	GMM	28.4	2.1	237.0	2.2	53.3	1.9	33.1	2.1	183.1	2.5	62.8	1.7
VAE-IAF	$\mathcal{N}$	27.5	2.1	236.0	2.2	55.4	1.9	30.6	2.0	183.6	2.5	66.2	1.6
	GMM	27.0	2.1	235.4	2.2	53.6	1.9	32.2	2.1	180.8	2.5	62.7	1.7
β-VAE	$\mathcal{N}$	21.4	2.1	**115.4**	3.6	56.1	1.9	**19.1**	2.0	**124.9**	3.4	65.9	1.6
	GMM	9.2	2.2	92.2	3.9	51.7	1.9	11.4	2.1	112.6	3.6	59.3	1.7
β-TC VAE	$\mathcal{N}$	21.3	2.1	116.6	2.8	55.7	1.8	20.7	2.0	125.8	3.4	65.9	1.6
	GMM	11.6	2.2	89.3	4.1	51.8	1.9	13.3	2.1	**106.5**	3.7	59.3	1.7
FactorVAE	$\mathcal{N}$	27.0	2.1	236.5	2.2	**53.8**	1.9	31.0	2.0	185.4	2.5	66.4	1.7
	GMM	26.9	2.1	234.0	2.2	52.4	2.0	32.7	2.1	184.4	2.5	63.3	1.7
InfoVAE - RBF	$\mathcal{N}$	27.5	2.1	235.2	2.1	55.5	1.9	31.1	2.0	182.8	2.5	66.5	1.6
	GMM	26.7	2.1	230.4	2.2	52.7	1.9	32.3	2.1	179.5	2.5	62.8	1.7
InfoVAE - IMQ	$\mathcal{N}$	28.3	2.1	233.8	2.2	56.7	1.9	31.0	2.0	182.4	2.5	66.4	1.6
	GMM	27.7	2.1	231.9	2.2	53.7	1.9	32.8	2.1	180.7	2.6	62.3	1.7
AAE	$\mathcal{N}$	**16.8**	2.2	139.9	2.6	59.9	1.8	**19.1**	2.1	164.9	2.4	64.8	1.7
	GMM	9.3	2.2	92.1	3.8	53.9	2.0	11.1	2.1	118.5	3.5	58.7	1.8
MSSSIM-VAE	$\mathcal{N}$	26.7	2.2	279.9	1.7	124.3	1.3	28.0	2.1	254.2	1.7	119.0	1.3
	GMM	27.2	2.2	279.7	1.7	124.3	1.3	28.8	2.1	253.1	1.7	119.2	1.3
VAEGAN	$\mathcal{N}$	*8.7*	2.2	199.5	2.2	*39.7*	1.9	*12.8*	2.2	198.7	2.2	122.8	2.0
(not compared)	GMM	*6.3*	2.2	197.5	2.1	*35.6*	1.8	*6.5*	2.2	188.2	2.6	84.3	1.7
AE	$\mathcal{N}$	26.7	2.1	201.3	2.1	327.7	1.0	221.8	1.3	210.1	2.1	275.0	2.9
	GMM	9.3	2.2	97.3	3.6	55.4	2.0	11.0	2.1	120.7	3.4	57.4	1.8
WAE - RBF	$\mathcal{N}$	21.2	2.2	175.1	2.0	332.6	1.0	21.2	2.1	170.2	2.3	69.4	1.6
	GMM	9.2	2.2	97.1	3.6	55.0	2.0	11.2	2.1	120.3	3.4	58.3	1.7
WAE - IMQ	$\mathcal{N}$	18.9	2.2	164.4	2.2	64.6	1.7	20.3	2.1	150.7	2.5	67.1	1.6
	GMM	**8.6**	2.2	96.5	3.6	51.7	2.0	11.2	2.1	119.0	3.5	57.7	1.8
VQVAE	$\mathcal{N}$	28.2	2.0	152.2	2.0	306.9	1.0	170.7	1.6	195.7	1.9	140.3	2.2
	GMM	9.1	2.2	95.2	3.7	51.6	2.0	**10.7**	2.1	120.1	3.4	57.9	1.8
RAE - L2	$\mathcal{N}$	25.0	2.0	156.1	2.6	86.1	2.8	63.3	2.2	170.9	2.2	168.7	3.1
	GMM	9.1	2.2	**85.3**	3.9	55.2	1.9	11.5	2.1	122.5	3.4	58.3	1.8
RAE - GP	$\mathcal{N}$	27.1	2.1	196.8	2.1	86.1	2.4	61.5	2.2	229.1	2.0	201.9	3.1
	GMM	9.7	2.2	96.3	3.7	52.5	1.9	11.4	2.1	123.3	3.4	59.0	1.8

Table B.19.: ConvNet, neural network architecture used for the convolutional networks, adapted from Chadebec et al. (2022).

	MNIST	CIFAR10	CelebA
Encoder	(1, 28, 28)	(3, 32, 32)	(3, 64, 64)
Layer 1	Conv(128, 4, 2), BN, ReLU	Conv(128, 4, 2), BN, ReLU	Conv(128, 4, 2), BN, ReLU
Layer 2	Conv(256, 4, 2), BN, ReLU	Conv(256, 4, 2), BN, ReLU	Conv(256, 4, 2), BN, ReLU
Layer 3	Conv(512, 4, 2), BN, ReLU	Conv(512, 4, 2), BN, ReLU	Conv(512, 4, 2), BN, ReLU
Layer 4	Conv(1024, 4, 2), BN, ReLU	Conv(1024, 4, 2), BN, ReLU	Conv(1024, 4, 2), BN, ReLU
Layer 5	Linear(1024, latent_dim)*	Linear(4096, latent_dim)*	Linear(16384, latent_dim)*
Decoder			
Layer 1	Linear(latent_dim, 16384)	Linear(latent_dim, 65536)	Linear(latent_dim, 65536)
Layer 2	ConvT(512, 3, 2), BN, ReLU	ConvT(512, 4, 2), BN, ReLU	ConvT(512, 5, 2), BN, ReLU
Layer 3	ConvT(256, 3, 2), BN, ReLU	ConvT(256, 4, 2), BN, ReLU	ConvT(256, 5, 2), BN, ReLU
Layer 4	Conv(1, 3, 2), Sigmoid	Conv(3, 4, 1), Sigmoid	ConvT(128, 5, 2), BN, ReLU
Layer 5	-	-	ConvT(3, 5, 1), Sigmoid
#Parameters	17.2M	39.4M	33.5M

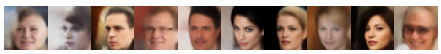
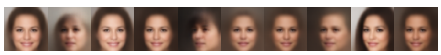
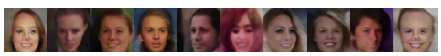
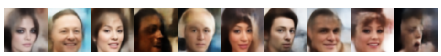
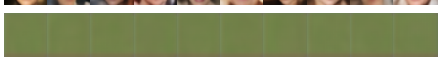
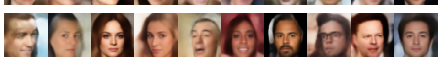
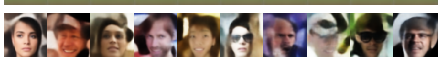
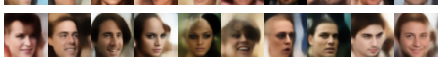
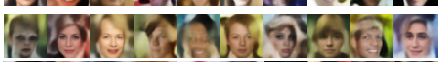
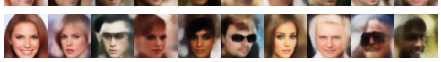
Table B.20.: ResNet, neural network architecture used for the residual networks, adapted from Chadebec et al. (2022). *The ResBlocks are composed of one Conv(32, 3, 1) followed by Conv(128, 1, 1) with ReLU.

	MNIST	CIFAR10	CelebA
Encoder	(1, 28, 28)	(3, 32, 32)	(3, 64, 64)
Layer 1	Conv(64, 4, 2)	Conv(64, 4, 2)	Conv(64, 4, 2)
Layer 2	Conv(128, 4, 2)	Conv(128, 4, 2)	Conv(128, 4, 2)
Layer 3	Conv(128, 3, 2)	Conv(128, 3, 1)	Conv(128, 3, 2)
Layer 4	ResBlock*	ResBlock*	Conv(128, 3, 2)
Layer 5	ResBlock*	ResBlock*	ResBlock*
Layer 6	Linear(2048, latent_dim)*	Linear(8192, latent_dim)*	ResBlock*
Layer 7	-	-	Linear(2048, latent_dim)*
Decoder			
Layer 1	Linear(latent_dim, 2048)	Linear(latent_dim, 8192)	Linear(latent_dim, 2048)
Layer 2	ConvT(128, 3, 2)	ResBlock*	ConvT(128, 3, 2)
Layer 3	ResBlock*	ResBlock*	ResBlock*
Layer 4	ResBlock*, ReLU	ConvT(64, 4, 2)	ResBlock*
Layer 5	ConvT(64, 3, 2), ReLU	ConvT(3, 4, 2), Sigmoid	ConvT(128, 5, 2), Sigmoid
Layer 6	ConvT(1, 3, 2), Sigmoid	-	ConvT(64, 5, 2), Sigmoid
Layer 6	-	-	ConvT(3, 4, 2), Sigmoid
#Parameters	0.73M	4.8M	1.6M

Table B.21.: FIF-ConvNet, neural network architecture used for comparison to Trumpet and Denoising Normalizing Flow. *The ResBlocks(inner_dim) are composed of Linear(latent_dim, inner_dim), SiLU, Linear(inner_dim, inner_dim), SiLU, Linear(inner_dim, latent_dim) with a skip connection.

	MNIST	CelebA
Encoder	(1, 28, 28)	(3, 64, 64)
Layer 1	Conv(32, 4, 2), BN, ReLU	Conv(128, 4, 2), BN, ReLU
Layer 2	Conv(64, 4, 2), BN, ReLU	Conv(256, 4, 2), BN, ReLU
Layer 3	Conv(128, 4, 2), BN, ReLU	Conv(512, 4, 2), BN, ReLU
Layer 4	Conv(256, 4, 2), BN, ReLU	Conv(*1024*, 4, 2), BN, ReLU
Layer 5	Linear(256, latent_dim)*	Linear(16384, latent_dim)*
Layer 6-9	4xResBlock(512)	4xResBlock(256)
Decoder		
Layer 1-4	*4xResBlock(512)*	*4xResBlock(256)*
Layer 5	Linear(latent_dim, *4096*)	Linear(latent_dim, 65536)
Layer 6	ConvT(256, 3, 2), BN, ReLU	ConvT(512, 5, 2), BN, ReLU
Layer 7	ConvT(128, 3, 2), BN, ReLU	ConvT(256, 5, 2), BN, ReLU
Layer 8	ConvT(64, 3, 2), BN, ReLU	ConvT(128, 5, 2), BN, ReLU
Layer 9	Conv(1, 3, 2), Sigmoid	ConvT(3, 5, 1), Sigmoid
#Parameters	3.3M	34.3M

Table B.22.: Uncurated samples from the CelebA ConvNet experiments in the PythAE benchmark. Our model is shown at the bottom, samples from the other models have been taken from (Chadebec et al., 2022).

	CelebA - $\mathcal{N}$	CelebA - GMM
IWAE		
VAE-lin-NF		
VAE-IAF		
β-VAE		
β-TC-VAE		
Factor-VAE		
InfoVAE - IMQ		
InfoVAE - RBF		
AAE		
MSSSIM-VAE		
VAEGAN		
AE		
WAE-IMQ		
WAE-RBF		
VQVAE		
RAE-L2		
RAE-GP		
FIF (ours)		

B.7.3. Details on pathology induced by curvature

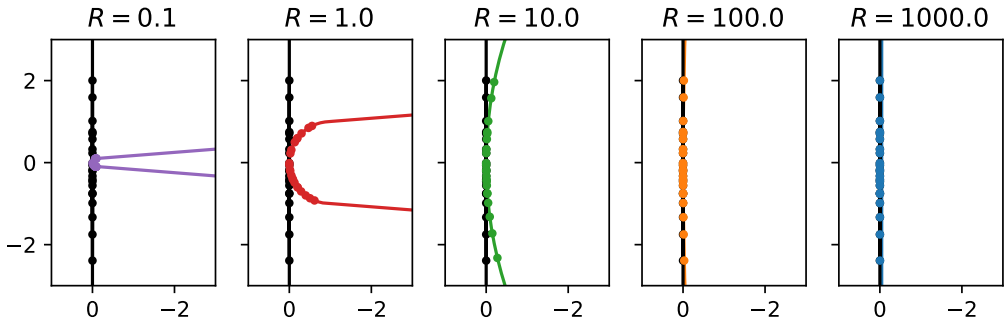


Figure B.19.: Possible learned manifolds of varying curvature $1/R$ for data supported on a subspace, where $d = 1$ and $D = 2$.

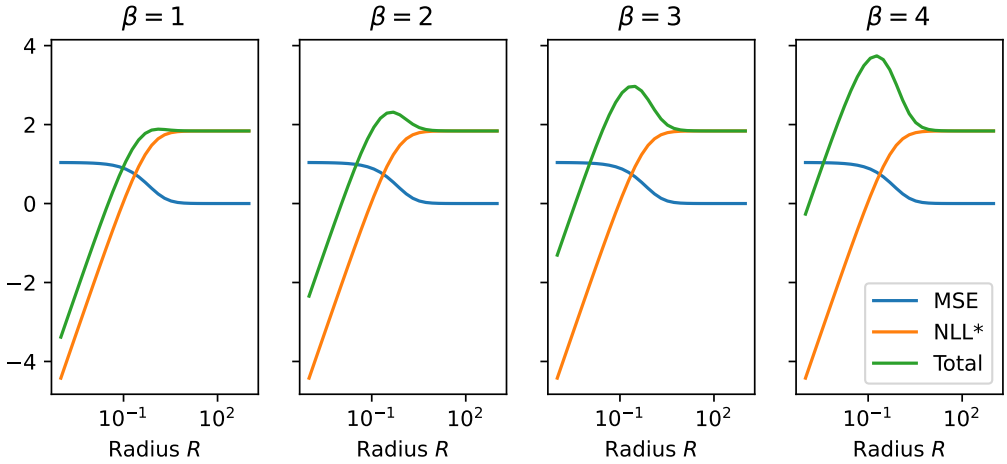


Figure B.20.: Weighting the reconstruction loss higher does not lead to stable training. The plots show different reconstruction weights β. In all settings, highly curved manifolds (i.e. low radius) achieve the lowest loss.

As described in section 7.3.3, gradients from the on-manifold loss in equation (7.34) cause the learned manifold to increase curvature. This is visualized in the main text in figure 7.7, where the left plot shows that this loss leads to ever-increasing curvature. The reason is that the entropy of data projected to a curved manifold is smaller than the entropy of data projected to a flat manifold.

Here, we provide intuition for why this happens for synthetic data where d is known, and the data could in principle be perfectly reconstructed. Figure B.19 depicts projections of the data to possible model manifolds of varying curvature κ. We parameterize the curvature by varying the radius $R = 1/\kappa$. One can observe that for with increasing curvature (i.e. decreasing radius), the data is projected to an increasingly small region. Correspondingly, the entropy $H[\hat{p}_{\text{data}}(\hat{x})]$ of the projected data becomes arbitrarily negative (just like a Gaussian with low standard deviation has arbitrarily negative entropy), lowering the achievable negative log-likelihood.

Adding reconstruction loss alone does not fix this pathology, which we illustrate in figure B.20: The reconstruction loss saturates for small radii, but the best achievable negative log-likelihood (i.e. the entropy of the data) continues to decrease with the radius. Thus, even when increasing β, the minimal possible value of the total loss is still achieved by a spuriously curved manifold.