Normalizing Flows: An Introduction and Review of Current Methods

Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker, *Member, IEEE*

Abstract—Normalizing Flows are generative models which produce tractable distributions where both sampling and density evaluation can be efficient and exact. The goal of this survey article is to give a coherent and comprehensive review of the literature around the construction and use of Normalizing Flows for distribution learning. We aim to provide context and explanation of the models, review current state-of-the-art literature, and identify open questions and promising future directions.

Index Terms—Generative models, normalizing flows, density estimation, variational inference, invertible neural networks

1 Introduction

A major goal of statistics and machine learning has been to model a probability distribution given samples drawn from that distribution. This is an example of unsupervised learning and is sometimes called generative modelling. Its importance derives from the relative abundance of unlabelled data compared to labelled data. Applications include density estimation, outlier detection, prior construction, and dataset summarization.

Many methods for generative modeling have been proposed. Direct analytic approaches approximate observed data with a fixed family of distributions. Variational approaches and expectation maximization introduce latent variables to explain the observed data. They provide additional flexibility but can increase the complexity of learning and inference. Graphical models [59] explicitly model the conditional dependence between random variables. Recently, generative neural approaches have been proposed including generative adversarial networks (GANs) [33] and variational auto-encoders (VAEs) [54].

GANs and VAEs have demonstrated impressive performance results on challenging tasks such as learning distributions of natural images. However, several issues limit their application in practice. Neither allows for exact evaluation of the probability density of new points. Furthermore, training can be challenging due to a variety of phenomena including mode collapse, posterior collapse, vanishing gradients and training instability [11], [82].

Normalizing Flows (NF) are a family of generative models with tractable distributions where both sampling and density evaluation can be efficient and exact. Applications include image generation [41], [57], noise modelling [1], video generation [60], audio generation [27], [53], [77], graph generation [65], reinforcement learning [67], [70], [93], computer graphics [69], and physics [51], [58], [71], [104], [105].

There are several survey papers for VAEs [55] and GANs [17], [100]. This article aims to provide a comprehensive review of the literature around Normalizing Flows for distribution learning. Our goals are to 1) provide context and explanation to enable a reader to become familiar with the basics, 2) review the current literature, and 3) identify open questions and promising future directions. Since this article was first made public, an excellent complementary treatment has been provided by Papamakarios *et al.* [75]. Their article is more tutorial in nature and provides many details concerning implementation, whereas our treatment is more formal and focuses mainly on the families of flow models.

In Section 2, we introduce Normalizing Flows and describe how they are trained. In Section 3 we review constructions for Normalizing Flows. In Section 4 we describe datasets for testing Normalizing Flows and discuss the performance of different approaches. Finally, in Section 5 we discuss open problems and possible research directions.

2 Background

Normalizing Flows were popularised by Rezende and Mohamed [78] in the context of variational inference and by Dinh *et al.* [19] for density estimation. However, the framework was previously defined in Tabak and Vanden- Eijnden [89] and Tabak and Turner [88], and explored for clustering and classification [2], and density estimation [61], [80].

A Normalizing Flow is a transformation of a simple probability distribution (e.g., a standard normal) into a more complex distribution by a sequence of invertible and differentiable mappings. The density of a sample can be evaluated by transforming it back to the original simple distribution and then computing the product of i) the density of the inverse-transformed sample under this distribution and ii) the associated change in volume induced by the sequence of inverse transformations. The change in volume is the product of the absolute values of the determinants of

• *The authors are with Borealis AI, Montreal H2S 3H1, Canada. E-mail: {ivan.kobyzev, simon.prince}@borealisai.com, mab@eecs.yorku.ca.*

Manuscript received 8 Dec. 2019; revised 21 Apr. 2020; accepted 1 May 2020. Date of publication 7 May 2020; date of current version 1 Oct. 2021. (Corresponding author: Ivan Kobyzev.) Recommended for acceptance by B. Kingsbury. Digital Object Identifier no. 10.1109/TPAMI.2020.2992934

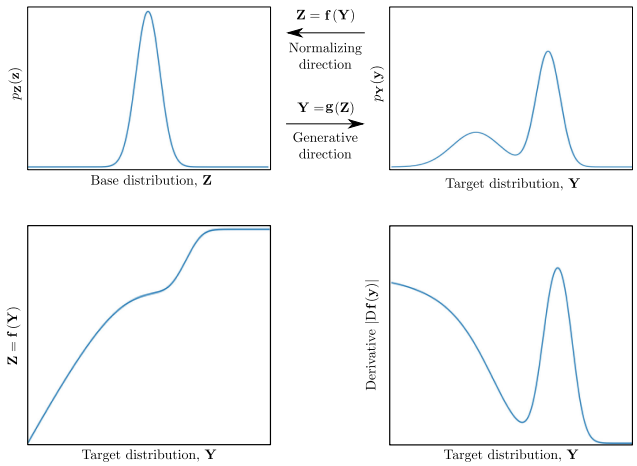

Fig. 1. Change of variables (Equation (1)). Top-left: the density of the source $p_{\mathbf{Z}}$. Top-right: the density function of the target distribution $p_{\mathbf{Y}}(\mathbf{y})$. There exists a bijective function $\mathbf{g}$, such that $p_{\mathbf{Y}} = \mathbf{g}_* p_{\mathbf{Z}}$, with inverse $\mathbf{f}$. Bottom-left: the inverse function $\mathbf{f}$. Bottom-right: the absolute Jacobian (derivative) of $\mathbf{f}$.

the Jacobians for each transformation, as required by the change of variables formula.

The result of this approach is a mechanism to construct new families of distributions by choosing an initial density and then chaining together some number of parameterized, invertible and differentiable transformations. The new density can be sampled from (by sampling from the initial density and applying the transformations) and the density at a sample (i.e., the likelihood) can be computed as above.

2.1 Basics

Let $\mathbf{Z} \in \mathbb{R}^D$ be a random variable with a known and tractable probability density function $p_{\mathbf{Z}} : \mathbb{R}^D \to \mathbb{R}$. Let $\mathbf{g}$ be an invertible function and $\mathbf{Y} = \mathbf{g}(\mathbf{Z})$. Then using the change of variables formula, one can compute the probability density function of the random variable $\mathbf{Y}$

$$\begin{aligned} p_{\mathbf{Y}}(\mathbf{y}) &= p_{\mathbf{Z}}(\mathbf{f}(\mathbf{y}))|\det \mathrm{D}\mathbf{f}(\mathbf{y})| \\ &= p_{\mathbf{Z}}(\mathbf{f}(\mathbf{y}))|\det \mathrm{D}\mathbf{g}(\mathbf{f}(\mathbf{y}))|^{-1}, \end{aligned} \tag{1}$$

where $\mathbf{f}$ is the inverse of $\mathbf{g}$, $\mathrm{D}\mathbf{f}(\mathbf{y}) = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ is the Jacobian of $\mathbf{f}$ and $\mathrm{D}\mathbf{g}(\mathbf{z}) = \frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ is the Jacobian of $\mathbf{g}$. This new density function $p_{\mathbf{Y}}(\mathbf{y})$ is called a *pushforward* of the density $p_{\mathbf{Z}}$ by the function $\mathbf{g}$ and denoted by $\mathbf{g}_* p_{\mathbf{Z}}$ (Fig. 1).

In the context of generative models, the above function $\mathbf{g}$ (a generator) "pushes forward" the base density $p_{\mathbf{Z}}$ (sometimes referred to as the "noise") to a more complex density. This movement from base density to final complicated density is the *generative direction*. Note that to generate a data point $\mathbf{y}$, one can sample $\mathbf{z}$ from the base distribution, and then apply the generator: $\mathbf{y} = \mathbf{g}(\mathbf{z})$.

The inverse function $\mathbf{f}$ moves (or "flows") in the opposite, *normalizing direction*: from a complicated and irregular data distribution towards the simpler, more regular or "normal" form, of the base measure $p_{\mathbf{Z}}$. This view is what gives rise to the name "normalizing flows" as $\mathbf{f}$ is "normalizing" the data distribution. This term is doubly accurate if the base measure $p_{\mathbf{Z}}$ is chosen as a Normal distribution as it often is in practice.

Intuitively, if the transformation $\mathbf{g}$ can be arbitrarily complex, one can generate any distribution $p_{\mathbf{Y}}$ from any base distribution $p_{\mathbf{Z}}$ under reasonable assumptions on the two distributions. This has been proven formally [9], [68], [99]. See Section 3.4.3.

Constructing arbitrarily complicated non-linear invertible functions (bijections) can be difficult. By the term *Normalizing Flows* people mean bijections which are convenient to compute, invert, and calculate the determinant of their Jacobian. One approach to this is to note that the composition of invertible functions is itself invertible and the determinant of its Jacobian has a specific form. In particular, let $\mathbf{g}_1, \ldots, \mathbf{g}_N$ be a set of N bijective functions and define $\mathbf{g} = \mathbf{g}_N \circ \mathbf{g}_{N-1} \circ \ldots \circ \mathbf{g}_1$ to be the composition of the functions. Then it can be shown that $\mathbf{g}$ is also bijective, with inverse

$$\mathbf{f} = \mathbf{f}_1 \circ \ldots \circ \mathbf{f}_{N-1} \circ \mathbf{f}_N, \tag{2}$$

and the determinant of the Jacobian is

$$\det \mathrm{D}\mathbf{f}(\mathbf{y}) = \prod_{i=1}^{N} \det \mathrm{D}\mathbf{f}_i(\mathbf{x}_i), \tag{3}$$

where $\mathrm{D}\mathbf{f}_i(\mathbf{y}) = \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}}$ is the Jacobian of $\mathbf{f}_i$. We denote the value of the ith intermediate flow as $\mathbf{x}_i = \mathbf{g}_i \circ \ldots \circ \mathbf{g}_1(\mathbf{z}) = \mathbf{f}_{i+1} \circ \ldots \circ \mathbf{f}_N(\mathbf{y})$ and so $\mathbf{x}_N = \mathbf{y}$. Thus, a set of nonlinear bijective functions can be composed to construct successively more complicated functions.

2.1.1 More Formal Construction

In this section we explain normalizing flows from more formal perspective. Readers unfamiliar with measure theory can safely skip to Section 2.2. First, let us recall the general definition of a pushforward.

Definition 1. *If $(\mathcal{Z}, \Sigma_{\mathcal{Z}})$, $(\mathcal{Y}, \Sigma_{\mathcal{Y}})$ are measurable spaces, $\mathbf{g}$ is a measurable mapping between them, and μ is a measure on $\mathcal{Z}$, then one can define a measure on $\mathcal{Y}$ (called the pushforward measure and denoted by $\mathbf{g}_*\mu$) by the formula*

$$\mathbf{g}_*\mu(U) = \mu(\mathbf{g}^{-1}(U)), \quad \text{for all } U \in \Sigma_{\mathcal{Y}}. \tag{4}$$

This notion gives a general formulation of a generative model. Data can be understood as a sample from a measured "data" space $(\mathcal{Y}, \Sigma_{\mathcal{Y}}, \nu)$, which we want to learn. To do that one can introduce a simpler measured space $(\mathcal{Z}, \Sigma_{\mathcal{Z}}, \mu)$ and find a function $\mathbf{g} : \mathcal{Z} \to \mathcal{Y}$, such that $\nu = \mathbf{g}_*\mu$. This function $\mathbf{g}$ can be interpreted as a "generator", and $\mathcal{Z}$ as a latent space. This view puts generative models in the context of transportation theory [99].

In this survey we will assume that $\mathcal{Z} = \mathbb{R}^D$, all sigma-algebras are Borel, and all measures are absolutely continuous with respect to Lebesgue measure (i.e., $\mu = p_{\mathbf{Z}}\mathbf{dz}$).

Definition 2. *A function $\mathbf{g} : \mathbb{R}^D \to \mathbb{R}^D$ is called a diffeomorphism, if it is bijective, differentiable, and its inverse is differentiable as well.*

The pushforward of an absolutely continuous measure $p_{\mathbf{Z}}\mathbf{dz}$ by a diffeomorphism $\mathbf{g}$ is also absolutely continuous with a density function given by Equation (1). Note that this more general approach is important for studying generative models on non-euclidean spaces (see Section 5.2).

Remark 3. It is common in the normalizing flows literature to simply refer to diffeomorphisms as "bijections" even though this is formally incorrect. In general, it is not necessary that $\mathbf{g}$ is everywhere differentiable; rather it is sufficient that it is differentiable only almost everywhere with respect to the Lebesgue measure on $\mathbb{R}^D$. This allows, for instance, piecewise differentiable functions to be used in the construction of $\mathbf{g}$.

2.2 Applications

2.2.1 Density Estimation and Sampling

The natural and most obvious use of normalizing flows is to perform density estimation. For simplicity assume that only a single flow, $\mathbf{g}$, is used and it is parameterized by the vector θ. Further, assume that the base measure, $p_{\mathbf{Z}}$ is given and is parameterized by the vector ϕ. Given a set of data observed from some complicated distribution, $\mathcal{D} = \{\mathbf{y}^{(i)}\}_{i=1}^{M}$, we can then perform likelihood-based estimation of the parameters $\Theta = (\theta, \phi)$. The data likelihood in this case simply becomes

$$\begin{aligned}\log p(\mathcal{D}|\Theta) &= \sum_{i=1}^{M} \log p_{\mathbf{Y}}(\mathbf{y}^{(i)}|\Theta) \\ &= \sum_{i=1}^{M} \log p_{\mathbf{Z}}(\mathbf{f}(\mathbf{y}^{(i)}|\theta)|\phi) + \log \left|\det \mathrm{D}\mathbf{f}(\mathbf{y}^{i}|\theta)\right|,\end{aligned} \tag{5}$$

where the first term is the log likelihood of the sample under the base measure and the second term, sometimes called the log-determinant or volume correction, accounts for the change of volume induced by the transformation of the normalizing flows (see Equation (1)). During training, the parameters of the flow (θ) and of the base distribution (ϕ) are adjusted to maximize the log-likelihood.

Note that evaluating the likelihood of a distribution modelled by a normalizing flow requires computing $\mathbf{f}$ (i.e., the normalizing direction), as well as its log determinant. The efficiency of these operations is particularly important during training where the likelihood is repeatedly computed. However, sampling from the distribution defined by the normalizing flow requires evaluating the inverse $\mathbf{g}$ (i.e., the generative direction). Thus sampling performance is determined by the cost of the generative direction. Even though a flow must be theoretically invertible, computation of the inverse may be difficult in practice; hence, for density estimation it is common to model a flow in the normalizing direction (i.e., $\mathbf{f}$).[1]

Finally, while maximum likelihood estimation is often effective (and statistically efficient under certain conditions) other forms of estimation can and have been used with normalizing flows. In particular, adversarial losses can be used with normalizing flow models (e.g., in Flow-GAN [36]).

2.2.2 Variational Inference

Consider a latent variable model $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{y}) d\mathbf{y}$ where $\mathbf{x}$ is an observed variable and $\mathbf{y}$ the latent variable. The posterior distribution $p(\mathbf{y}|\mathbf{x})$ is used when estimating the parameters of the model, but its computation is usually intractable in practice. One approach is to use variational inference and introduce the approximate posterior $q(\mathbf{y}|\mathbf{x}, \theta)$ where θ are parameters of the variational distribution. Ideally this distribution should be as close to the real posterior as possible. This is done by minimizing the KL divergence $D_{KL}(q(\mathbf{y}|\mathbf{x}, \theta)||p(\mathbf{y}|\mathbf{x}))$, which is equivalent to maximizing the evidence lower bound $\mathcal{L}(\theta) = \mathbb{E}_{q(\mathbf{y}|\mathbf{x},\theta)}[\log(p(\mathbf{y}, \mathbf{x})) - \log(q(\mathbf{y}|\mathbf{x}, \theta))]$. The latter optimization can be done with gradient descent; however for that one needs to compute gradients of the form $\nabla_\theta \mathbb{E}_{q(\mathbf{y}|\mathbf{x},\theta)}[h(\mathbf{y})]$, which is not straightforward.

As was observed by Rezende and Mohamed [78], one can reparametrize $q(\mathbf{y}|\mathbf{x}, \theta) = p_{\mathbf{Y}}(\mathbf{y}|\theta)$ with normalizing flows. Assume for simplicity, that only a single flow $\mathbf{g}$ with parameters θ is used, $\mathbf{y} = \mathbf{g}(\mathbf{z}|\theta)$ and the base distribution $p_{\mathbf{Z}}(\mathbf{z})$ does not depend on θ. Then

$$\mathbb{E}_{p_{\mathbf{Y}}(\mathbf{y}|\theta)}[h(\mathbf{y})] = \mathbb{E}_{p_{\mathbf{Z}}(\mathbf{z})}[h(\mathbf{g}(\mathbf{z}|\theta))], \tag{6}$$

and the gradient of the right hand side with respect to θ can be computed. This approach generally to computing gradients of an expectation is often called the "reparameterization trick".

In this scenario evaluating the likelihood is only required at points which have been sampled. Here the sampling performance and evaluation of the log determinant are the only relevant metrics and computing the inverse of the mapping may not be necessary. Indeed, the planar and radial flows introduced in Rezende and Mohamed [78] are not easily invertible (see Section 3.3).

3 Methods

Normalizing Flows should satisfy several conditions in order to be practical. They should:

- be invertible; for sampling we need $\mathbf{g}$ while for computing likelihood we need $\mathbf{f}$,
- be sufficiently expressive to model the distribution of interest,
- be computationally efficient, both in terms of computing $\mathbf{f}$ and $\mathbf{g}$ (depending on the application) but also in terms of the calculation of the determinant of the Jacobian.

In the following section, we describe different types of flows and comment on the above properties. An overview of the methods discussed can be seen in Fig. 2.

3.1 Elementwise Flows

A basic form of bijective non-linearity can be constructed given any bijective scalar function. That is, let $h : \mathbb{R} \to \mathbb{R}$ be a scalar valued bijection. Then, if $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$,

$$\mathbf{g}(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_D))^T, \tag{7}$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the absolute values of the derivatives of h. This can be generalized by allowing each element to have its own distinct bijective function which might be useful if we wish to only modify portions of our parameter vector. In deep learning terminology, h, could be viewed as an "activation function". Note that the

1. To ensure both efficient density estimation and sampling, van den Oord *et al*. [98] proposed an approach called Probability Density Distillation which trains the flow $\mathbf{f}$ as normal and then uses this as a teacher network to train a tractable student network $\mathbf{g}$.

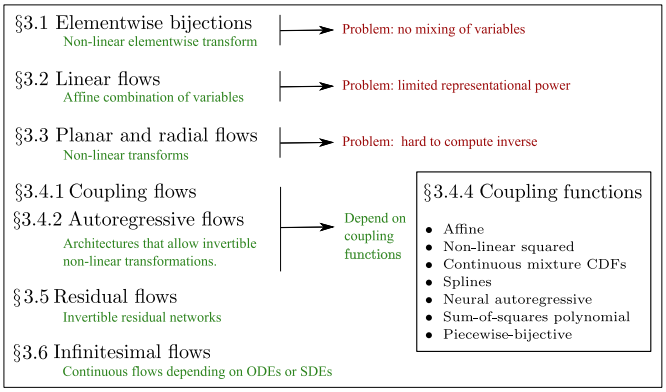

Fig. 2. Overview of flows discussed in this review. We start with elementwise bijections, linear flows, and planar and radial flows. All of these have drawbacks and are limited in utility. We then discuss two architectures (coupling flows and autoregressive flows) which support invertible non-linear transformations. These both use a coupling function, and we summarize the different coupling functions available. Finally, we discuss residual flows and their continuous extension infinitesimal flows.

most commonly used activation function ReLU is not bijective and can not be directly applicable, however, the (Parametric) Leaky ReLU [39], [64] can be used instead among others. Note that recently spline-based activation functions have also been considered [24], [25] and will be discussed in Section 3.4.4.4.

3.2 Linear Flows

Elementwise operations alone are insufficient as they cannot express any form of correlation between dimensions. Linear mappings can express correlation between dimensions

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}, \tag{8}$$

where $\mathbf{A} \in \mathbb{R}^{D\times D}$ and $\mathbf{b} \in \mathbb{R}^D$ are parameters. If $\mathbf{A}$ is an invertible matrix, the function is invertible.

Linear flows are limited in their expressiveness. Consider a Gaussian base distribution: $p_{\mathbf{Z}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, \boldsymbol{\mu}, \Sigma)$. After transformation by a linear flow, the distribution remains Gaussian with distribution $p_{\mathbf{Y}} = \mathcal{N}(\mathbf{y}, \mathbf{A}\mu + \mathbf{b}, \mathbf{A}^T\Sigma\mathbf{A})$. More generally, a linear flow of a distribution from the exponential family remains in the exponential family. However, linear flows are an important building block as they form the basis of affine coupling flows (Section 3.4.4.1).

Note that the determinant of the Jacobian is simply $\det(\mathbf{A})$, which can be computed in $\mathcal{O}(D^3)$, as can the inverse. Hence, using linear flows can become expensive for large D. By restricting the form of $\mathbf{A}$ we can avoid these practical problems at the expense of expressive power. In the following sections we discuss different ways of limiting the form of linear transforms to make them more practical.

3.2.1 Diagonal

If $\mathbf{A}$ is diagonal with nonzero diagonal entries, then its inverse can be computed in linear time and its determinant is the product of the diagonal entries. However, the result is an elementwise transformation and hence cannot express correlation between dimensions. Nonetheless, a diagonal linear flow can still be useful for representing normalization transformations [20] which have become a ubiquitous part of modern neural networks [46].

3.2.2 Triangular

The triangular matrix is a more expressive form of linear transformation whose determinant is the product of its diagonal. It is non-singular so long as its diagonal entries are non-zero. Inversion is relatively inexpensive requiring a single pass of back-substitution costing $\mathcal{O}(D^2)$ operations.

Tomczak and Welling [91] combined K triangular matrices $\mathbf{T}_i$, each with ones on the diagonal, and a K-dimensional probability vector ω to define a more general linear flow $\mathbf{y} = (\sum_{i=1}^{K} \omega_i \mathbf{T}_i)\mathbf{z}$. The determinant of this bijection is one. However finding the inverse has $\mathcal{O}(D^3)$ complexity, if some of the matrices are upper- and some are lower-triangular.

3.2.3 Permutation and Orthogonal

The expressiveness of triangular transformations is sensitive to the ordering of dimensions. Reordering the dimensions can be done easily using a permutation matrix which has an absolute determinant of 1. Different strategies have been tried, including reversing and a fixed random permutation [20], [57]. However, the permutations cannot be directly optimized and so remain fixed after initialization which may not be optimal.

A more general alternative is the use of orthogonal transformations. The inverse and absolute determinant of an orthogonal matrix are both trivial to compute which make them efficient. Tomczak and Welling [92] used orthogonal matrices parameterized by the *Householder transform*. The idea is based on the fact from linear algebra that any orthogonal matrix can be written as a product of reflections. To parameterize a reflection matrix H in $\mathbb{R}^D$ one fixes a non-zero vector $\mathbf{v} \in \mathbb{R}^D$, and then defines $H = \mathbb{1} - \frac{2}{\|\mathbf{v}\|^2}\mathbf{v}\mathbf{v}^T$.

3.2.4 Factorizations

Instead of limiting the form of $\mathbf{A}$, Kingma and Dhariwal [57] proposed using the LU factorization

$$\mathbf{g}(\mathbf{x}) = \mathbf{P}\mathbf{L}\mathbf{U}\mathbf{x} + \mathbf{b}, \tag{9}$$

where $\mathbf{L}$ is lower triangular with ones on the diagonal, $\mathbf{U}$ is upper triangular with non-zero diagonal entries, and $\mathbf{P}$ is a permutation matrix. The determinant is the product of the diagonal entries of $\mathbf{U}$ which can be computed in $\mathcal{O}(D)$. The inverse of the function $\mathbf{g}$ can be computed using two passes of backward substitution in $\mathcal{O}(D^2)$. However, the discrete permutation $\mathbf{P}$ cannot be easily optimized. To avoid this, $\mathbf{P}$ is randomly generated initially and then fixed. Hoogeboom *et al.* [42] noted that fixing the permutation matrix limits the flexibility of the transformation, and proposed using the QR decomposition instead where the orthogonal matrix Q is described with Householder transforms.

3.2.5 Convolution

Another form of linear transformation is a convolution which has been a core component of modern deep learning architectures. While convolutions are easy to compute their inverse and determinant are non-obvious. Several approaches have been considered. Kingma and Dhariwal [57] restricted themselves to "1×1" convolutions for flows which are simply a full linear transformation but applied only across channels. Zheng *et al.* [107] used 1D

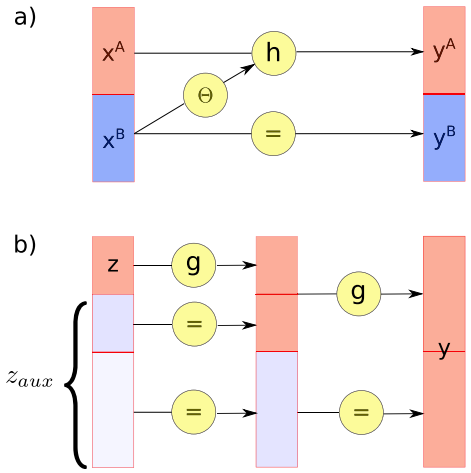

Fig. 3. Coupling architecture. a) A single coupling flow described in Equation (15). A coupling function $\mathbf{h}$ is applied to one part of the space, while its parameters depend on the other part. b) Two subsequent multi-scale flows in the generative direction. A flow is applied to a relatively low dimensional vector $\mathbf{z}$; its parameters no longer depend on the rest part $\mathbf{z}_{aux}$. Then new dimensions are gradually introduced to the distribution.

convolutions (*ConvFlow*) and exploited the triangular structure of the resulting transform to efficiently compute the determinant. However Hoogeboom *et al.* [42] have provided a more general solution for modelling $d \times d$ convolutions, either by stacking together masked autoregressive convolutions (referred to as Emerging Convolutions) or by exploiting the Fourier domain representation of convolution to efficiently compute inverses and determinants (referred to as Periodic Convolutions).

3.3 Planar and Radial Flows

Rezende and Mohamed [78] introduced planar and radial flows. They are relatively simple, but their inverses aren't easily computed. These flows are not widely used in practice, yet they are reviewed here for completeness.

3.3.1 Planar Flows

Planar flows expand and contract the distribution along certain specific directions and take the form

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} + \mathbf{u}h(\mathbf{w}^T\mathbf{x} + b), \tag{10}$$

where $\mathbf{u}, \mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are parameters and $h : \mathbb{R} \to \mathbb{R}$ is a smooth non-linearity. The Jacobian determinant for this transformation is

$$\begin{aligned} \det\left(\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right) &= \det(\mathbb{1}_D + \mathbf{u}h'(\mathbf{w}^T\mathbf{x} + b)\mathbf{w}^T) \\ &= 1 + h'(\mathbf{w}^T\mathbf{x} + b)\mathbf{u}^T\mathbf{w}, \end{aligned} \tag{11}$$

where the last equality comes from the application of the matrix determinant lemma. This can be computed in $\mathcal{O}(D)$ time. The inversion of this flow isn't possible in closed form and may not exist for certain choices of $h(\cdot)$ and certain parameter settings [78].

The term $\mathbf{u}h(\mathbf{w}^T\mathbf{x} + b)$ can be interpreted as a multilayer perceptron with a bottleneck hidden layer with a single unit [56]. This bottleneck means that one needs to stack many planar flows to get high expressivity. Hasenclever *et al.* [38] and van den Berg *et al.* [97] introduced *Sylvester flows* to resolve this problem

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} + \mathbf{U}\mathbf{h}(\mathbf{W}^T\mathbf{x} + \mathbf{b}), \tag{12}$$

where $\mathbf{U}$ and $\mathbf{W}$ are $D \times M$ matrices, $\mathbf{b} \in \mathbb{R}^M$ and $\mathbf{h} : \mathbb{R}^M \to \mathbb{R}^M$ is an elementwise smooth nonlinearity, where $M \leq D$ is a hyperparameter to choose and which can be interpreted as the dimension of a hidden layer. In this case the Jacobian determinant is

$$\begin{aligned} \det\left(\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right) &= \det(\mathbb{1}_D + \mathbf{U}\,\mathrm{diag}(\mathbf{h}'(\mathbf{W}^T\mathbf{x} + b))\mathbf{W}^T) \\ &= \det(\mathbb{1}_M + \mathrm{diag}(\mathbf{h}'(\mathbf{W}^T\mathbf{x} + b))\mathbf{W}\mathbf{U}^T), \end{aligned} \tag{13}$$

where the last equality is Sylvester's determinant identity (which gives these flows their name). This can be computationally efficient if M is small. Some sufficient conditions for the invertibility of Sylvester transformations are discussed in Hasenclever *et al.* [38] and van den Berg *et al.* [97].

3.3.2 Radial Flows

Radial flows instead modify the distribution around a specific point so that

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} + \frac{\beta}{\alpha + \|\mathbf{x} - \mathbf{x}_0\|}(\mathbf{x} - \mathbf{x}_0), \tag{14}$$

where $\mathbf{x}_0 \in \mathbb{R}^D$ is the point around which the distribution is distorted, and $\alpha, \beta \in \mathbb{R}$ are parameters, $\alpha > 0$. As for planar flows, the Jacobian determinant can be computed relatively efficiently. The inverse of radial flows cannot be given in closed form but does exist under suitable constraints on the parameters [78].

3.4 Coupling and Autoregressive Flows

In this section we describe coupling and auto-regressive flows which are the two most widely used flow architectures. We first present them in the general form, and then in Section 3.4.4 we give specific examples.

3.4.1 Coupling Flows

[19] introduced a coupling method to enable highly expressive transformations for flows (Fig. 3a). Consider a disjoint partition of the input $\mathbf{x} \in \mathbb{R}^D$ into two subspaces: $(\mathbf{x}^A, \mathbf{x}^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$ and a bijection $\mathbf{h}(\cdot\,; \theta) : \mathbb{R}^d \to \mathbb{R}^d$, parameterized by θ. Then one can define a function $\mathbf{g} : \mathbb{R}^D \to \mathbb{R}^D$ by the formula

$$\begin{aligned} \mathbf{y}^A &= \mathbf{h}(\mathbf{x}^A; \Theta(\mathbf{x}^B)) \\ \mathbf{y}^B &= \mathbf{x}^B, \end{aligned} \tag{15}$$

where the parameters θ are defined by *any* arbitrary function $\Theta(\mathbf{x}^B)$ which only uses $\mathbf{x}^B$ as input. This function is called a *conditioner*. The bijection $\mathbf{h}$ is called a *coupling function*, and the resulting function $\mathbf{g}$ is called a *coupling flow*. A coupling flow is invertible if and only if $\mathbf{h}$ is invertible and has inverse

$$\begin{aligned} \mathbf{x}^A &= \mathbf{h}^{-1}(\mathbf{y}^A; \Theta(\mathbf{x}^B)) \\ \mathbf{x}^B &= \mathbf{y}^B. \end{aligned} \tag{16}$$

The Jacobian of $\mathbf{g}$ is a block triangular matrix where the diagonal blocks are $D\mathbf{h}$ and the identity matrix respectively.

Hence the determinant of the Jacobian of the coupling flow is simply the determinant of D**h**.

Most coupling functions are applied to $\mathbf{x}^A$ element-wise

$$\mathbf{h}(\mathbf{x}^A;\theta) = (h_1(x_1^A;\theta_1),\ \ldots,h_d(x_d^A;\theta_d)), \tag{17}$$

where each $h_i(\cdot;\theta_i):\mathbb{R}\to\mathbb{R}$ is a scalar bijection. In this case a coupling flow is a triangular transformation (i.e., has a triangular Jacobian matrix). See Section 3.4.4 for examples.

The power of a coupling flow resides in the ability of a conditioner $\Theta(\mathbf{x}^B)$ to be arbitrarily complex. In practice it is usually modelled as a neural network. For example, Kingma and Dhariwal [57] used a shallow ResNet architecture.

Sometimes, however, the conditioner can be constant (i.e., not depend on $\mathbf{x}^B$ at all). This allows for the construction of a "*multi-scale flow*" [20] which gradually introduces dimensions to the distribution in the generative direction (Fig. 3b). In the normalizing direction, the dimension reduces by half after each iteration step, such that most of semantic information is retained. This reduces the computational costs of transforming high dimensional distributions and can capture the multi-scale structure inherent in certain kinds of data like natural images.

The question remains of how to partition $\mathbf{x}$. This is often done by splitting the dimensions in half [19], potentially after a random permutation. However, more structured partitioning has also been explored and is common practice, particularly when modelling images. For instance, Dinh *et al.* [20] used "masked" flows that take alternating pixels or blocks of channels in the case of an image in non-volume preserving flows (*RealNVP*). In place of permutation Kingma and Dhariwal [57] used 1×1 convolution (*Glow*). For the partition for the multi-scale flow in the normalizing direction, Das *et al.* [18] suggested selecting features at which the Jacobian of the flow has higher values for the propagated part.

3.4.2 Autoregressive Flows

Kingma *et al.* [56] used autoregressive models as a form of normalizing flow. These are non-linear generalizations of multiplication by a triangular matrix (Section 3.2.2).

Let $h(\cdot;\theta):\mathbb{R}\to\mathbb{R}$ be a bijection parameterized by θ. Then an autoregressive model is a function $\mathbf{g}:\mathbb{R}^D\to\mathbb{R}^D$, which outputs each entry of $\mathbf{y}=\mathbf{g}(\mathbf{x})$ conditioned on the previous entries of the input

$$y_t = h(x_t;\Theta_t(\mathbf{x}_{1:t-1})), \tag{18}$$

where $\mathbf{x}_{1:t}=(x_1,\ \ldots,x_t)$. For $t=2,\ \ldots,D$ we choose arbitrary functions $\Theta_t(\cdot)$ mapping $\mathbb{R}^{t-1}$ to the set of all parameters, and Θ_1 is a constant. The functions $\Theta_t(\cdot)$ are called *conditioners*.

The Jacobian matrix of the autoregressive transformation $\mathbf{g}$ is triangular. Each output y_t only depends on $\mathbf{x}_{1:t}$, and so the determinant is just a product of its diagonal entries

$$\det(\mathrm{D}\mathbf{g}) = \prod_{t=1}^{D}\frac{\partial y_t}{\partial x_t}. \tag{19}$$

In practice, it's possible to efficiently compute all the entries of the direct flow (Equation (18)) in one pass using a single network with appropriate masks [31]. This idea was used by Papamakarios *et al.* [74] to create masked autoregressive flows (*MAF*).

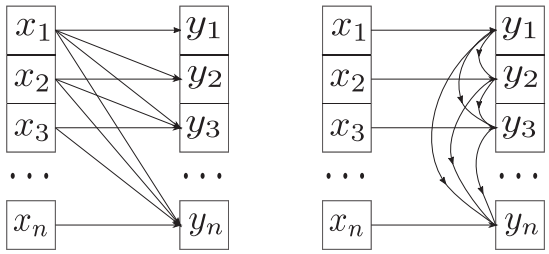

Fig. 4. Autoregressive flows. On the left, is the direct autoregressive flow given in Equation (18). Each output depends on the current and previous inputs and so this operation can be easily parallelized. On the right, is the inverse autoregressive flow from Equation (20). Each output depends on the current input and the previous outputs and so computation is inherently sequential and cannot be parallelized.

However, the computation of the inverse is more challenging. Given the inverse of h, the inverse of $\mathbf{g}$ can be found with recursion: we have $x_1 = h^{-1}(y_1;\Theta_1)$ and for any $t=2,\ \ldots,D$, $x_t = h^{-1}(y_t;\Theta_t(\mathbf{x}_{1:t-1}))$. This computation is inherently sequential which makes it difficult to implement efficiently on modern hardware as it cannot be parallelized.

Note that the functional form for the autoregressive model is very similar to that for the coupling flow. In both cases a bijection h is used, which has as an input one part of the space and which is parameterized conditioned on the other part. We call this bijection a *coupling function* in both cases. Note that Huang, Krueger, Lacoste, and Courville [45] used the name "transformer" (which has nothing to do with transformers in NLP).

Alternatively, Kingma *et al.* [56] introduced the "inverse autoregressive flow" (*IAF*), which outputs each entry of $\mathbf{y}$ conditioned the previous entries of $\mathbf{y}$ (with respect to the fixed ordering). Formally,

$$y_t = h(x_t;\theta_t(\mathbf{y}_{1:t-1})). \tag{20}$$

One can see that the functional form of the inverse autoregressive flow is the same as the form of the inverse of the flow in Equation (18), hence the name. Computation of the IAF is sequential and expensive, but the inverse of IAF (which is a direct autoregressive flow) can be computed relatively efficiently (Fig. 4).

In Section 2.2.1 we noted that papers typically model flows in the "normalizing flow" direction (i.e., in terms of $\mathbf{f}$ from data to the base density) to enable efficient evaluation of the log-likelihood during training. In this context one can think of IAF as a flow in the generative direction: i.e.in terms of $\mathbf{g}$ from base density to data. Hence Papamakarios *et al.* [74] noted that one should use IAFs if fast sampling is needed (e.g., for stochastic variational inference), and MAFs if fast density estimation is desirable. The two methods are closely related and, under certain circumstances, are theoretically equivalent [74].

3.4.3 Universality

For several autoregressive flows the universality property has been proven [45], [49]. Informally, universality means that the flow can learn any target density to any required

precision given sufficient capacity and data. We will provide a formal proof of the universality theorem following [49]. This section requires some knowledge of measure theory and functional analysis and can be safely skipped.

First, recall that a mapping $T = (T_1, \ldots, T_D) : \mathbb{R}^D \to \mathbb{R}^D$ is called triangular if T_i is a function of $\mathbf{x}_{1:i}$ for each $i = 1, \ldots, D$. Such a triangular map T is called increasing if T_i is an increasing function of x_i for each i.

Proposition 4 ([9], Lemma 2.1). *If μ and ν are absolutely continuous Borel probability measures on $\mathbb{R}^D$, then there exists an increasing triangular transformation $T : \mathbb{R}^D \to \mathbb{R}^D$, such that $\nu = T_*\mu$. This transformation is unique up to null sets of μ. A similar result holds for measures on $[0,1]^D$.*

Proposition 5. *If μ is an absolutely continuous Borel probability measures on $\mathbb{R}^D$ and $\{T_n\}$ is a sequence of maps $\mathbb{R}^D \to \mathbb{R}^D$ which converges pointwise to a map T, then a sequence of measures $(T_n)_*\mu$ weakly converges to $T_*\mu$.*

Proof. See [45], Lemma 4. The result follows from the dominated convergence theorem. □

As a corollary, to claim that a class of autoregressive flows $\mathbf{g}(\cdot, \theta) : \mathbb{R}^D \to \mathbb{R}^D$ is universal, it is enough to demonstrate that a family of coupling functions h used in the class is dense in the set of all monotone functions in the pointwise convergence topology. In particular, [45] used neural monotone networks for coupling functions, and [49] used monotone polynomials. Using the theory outlined in this section, universality could also be proved for spline flows [24], [25] with splines for coupling functions (see Section 3.4.4.4).

3.4.4 Coupling Functions

As described in the previous sections, coupling flows and autoregressive flows have a similar functional form and both have coupling functions as building blocks. A coupling function is a bijective differentiable function $\mathbf{h}(\cdot, \theta) : \mathbb{R}^d \to \mathbb{R}^d$, parameterized by θ. In coupling flows, these functions are typically constructed by applying a scalar coupling function $h(\cdot, \theta) : \mathbb{R} \to \mathbb{R}$ elementwise. In autoregressive flows, $d = 1$ and hence they are also scalar valued. Note that scalar coupling functions are necessarily (strictly) monotone. In this section we describe the scalar coupling functions commonly used in the literature.

3.4.4.1 Affine coupling. Two simple forms of coupling functions $h : \mathbb{R} \to \mathbb{R}$ were proposed by Dinh *et al.* [19] in **NICE** (nonlinear independent component estimation). These were the *additive coupling function*

$$h(x\,; \theta) = x + \theta, \quad \theta \in \mathbb{R}, \tag{21}$$

and the *affine coupling function*

$$h(x; \theta) = \theta_1 x + \theta_2, \quad \theta_1 \neq 0,\ \theta_2 \in \mathbb{R}. \tag{22}$$

Affine coupling functions are used for coupling flows in NICE [19], RealNVP [20], Glow [57] and for autoregressive architectures in IAF [56] and MAF [74]. They are simple and computation is efficient. However, they are limited in expressiveness and many flows must be stacked to represent complicated distributions.

3.4.4.2 Nonlinear squared flow. Ziegler and Rush [108] proposed an invertible non-linear squared transformation defined by

$$h(x\,; \theta) = ax + b + \frac{c}{1 + (dx + h)^2}. \tag{23}$$

Under some constraints on parameters $\theta = [a, b, c, d, h] \in \mathbb{R}^5$, the coupling function is invertible and its inverse is analytically computable as a root of a cubic polynomial (with only one real root). Experiments showed that these coupling functions facilitate learning multimodal distributions.

3.4.4.3 Continuous mixture CDFs. Ho *et al.* [41] proposed the *Flow++* model, which contained several improvements, including a more expressive coupling function. The layer is almost like a linear transformation, but one also applies a monotone function to x

$$h(x; \theta) = \theta_1 F(x, \theta_3) + \theta_2, \tag{24}$$

where $\theta_1 \neq 0$, $\theta_2 \in \mathbb{R}$ and $\theta_3 = [\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}] \in \mathbb{R}^K \times \mathbb{R}^K \times \mathbb{R}_+^K$. The function $F(x, \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s})$ is the CDF of a mixture of K logistics, postcomposed with an inverse sigmoid

$$F(x, \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}) = \sigma^{-1}\left(\sum_{j=1}^{K} \pi_j \sigma\left(\frac{x - \mu_j}{s_j}\right)\right). \tag{25}$$

Note, that the post-composition with $\sigma^{-1} : [0,1] \to \mathbb{R}$ is used to ensure the right range for h. Computation of the inverse is done numerically with the bisection algorithm. The derivative of the transformation with respect to x is expressed in terms of PDF of logistic mixture (i.e., a linear combination of hyperbolic secant functions), and its computation is not expensive. An ablation study demonstrated that switching from an affine coupling function to a logistic mixture improved performance slightly.

3.4.4.4 Splines. A spline is a piecewise-polynomial or a piecewise-rational function which is specified by $K + 1$ points $(x_i, y_i)_{i=0}^K$, called *knots*, through which the spline passes. To make a useful coupling function, the spline should be monotone which will be the case if $x_i < x_{i+1}$ and $y_i < y_{i+1}$. Usually splines are considered on a compact interval.

Piecewise-linear and piecewise-quadratic. Müller *et al.* [69] used linear splines for coupling functions $h : [0,1] \to [0,1]$. They divided the domain into K equal bins. Instead of defining increasing values for y_i, they modeled h as the integral of a positive piecewise-constant function

$$h(x; \theta) = \alpha\theta_b + \sum_{k=1}^{b-1} \theta_k, \tag{26}$$

where $\theta \in \mathbb{R}^K$ is a probability vector, $b = \lfloor Kx \rfloor$ (the bin that contains x), and $\alpha = Kx - b$ (the position of x in bin b). This map is invertible, if all $\theta_k > 0$, with derivative: $\frac{\partial h}{\partial x} = \theta_b K$.

Müller *et al.* [69] also used a monotone quadratic spline on the unit interval for a coupling function and modeled this as the integral of a positive piecewise-linear function. A monotone quadratic spline is invertible; finding its inverse map requires solving a quadratic equation.

Cubic Splines. Durkan *et al.* [24] proposed using monotone cubic splines for a coupling function. They do not restrict the domain to the unit interval, but instead use the form: $h(\cdot;\theta) = \sigma^{-1}(\hat{h}(\sigma(\cdot);\theta))$, where $\hat{h}(\cdot;\theta) : [0,1] \to [0,1]$ is a monotone cubic spline and σ is a sigmoid. Steffen's method is used to construct the spline. Here, one specifies $K+1$ knots of the spline and boundary derivatives $\hat{h}'(0)$ and $\hat{h}'(1)$. These quantities are modelled as the output of a neural network.

Computation of the derivative is easy as it is piecewise-quadratic. A monotone cubic polynomial has only one real root and for inversion, one can find this either analytically or numerically. However, the procedure is numerically unstable if not treated carefully. The flow can be trained by gradient descent by differentiating through the numerical root finding method. However Durkan *et al.* [25] noted numerical difficulties when the sigmoid saturates for values far from zero.

Rational Quadratic Splines. Durkan *et al.* [25] model a coupling function $h(x;\theta)$ as a monotone rational-quadratic spline on an interval as the identity function otherwise. They define the spline using the method of Gregory and Delbourgo [35], by specifying $K+1$ knots $\{h(x_i)\}_{i=0}^{K}$ and the derivatives at the inner points: $\{h'(x_i)\}_{i=1}^{K-1}$. These locations of the knots and their derivatives are modelled as the output of a neural network.

The derivative with respect to x is a quotient derivative and the function can be inverted by solving a quadratic equation. Durkan, Bekasov, Murray, and Papamakarios [25] used this coupling function with both a coupling architecture *RQ-NSF(C)* and an auto-regressive architecture *RQ-NSF(AR)*.

3.4.4.5 Neural autoregressive flow. Huang *et al.* [45] introduced Neural Autoregressive Flows (*NAF*) where a coupling function $h(\cdot;\theta)$ is modelled with a deep neural network. Typically such a network is not invertible, but they proved a sufficient condition for it to be bijective:

Proposition 6. *If* $\mathrm{NN}(\cdot) : \mathbb{R} \to \mathbb{R}$ *is a multilayer percepton, such that all weights are positive and all activation functions are strictly monotone, then* $\mathrm{NN}(\cdot)$ *is a strictly monotone function.*

They proposed two forms of neural networks: the deep sigmoidal coupling function (*NAF-DSF*) and deep dense sigmoidal coupling function (*NAF-DDSF*). Both are MLPs with layers of sigmoid and logit units and non-negative weights; the former has a single hidden layer of sigmoid units, whereas the latter is more general and does not have this bottleneck. By Proposition 6, the resulting $h(\cdot;\theta)$ is a strictly monotone function. They also proved that a DSF network can approximate any strictly monotone univariate function and so NAF-DSF is a universal flow.

Wehenkel and Louppe [102] noted that imposing positivity of weights on a flow makes training harder and requires more complex conditioners. To mitigate this, they introduced unconstrained monotonic neural networks (*UMNN*). The idea is in order to model a strictly monotone function, one can describe a strictly positive (or negative) function with a neural network and then integrate it numerically. They demonstrated that UMNN requires less parameters than NAF to reach similar performance, and so is more scalable for high-dimensional datasets.

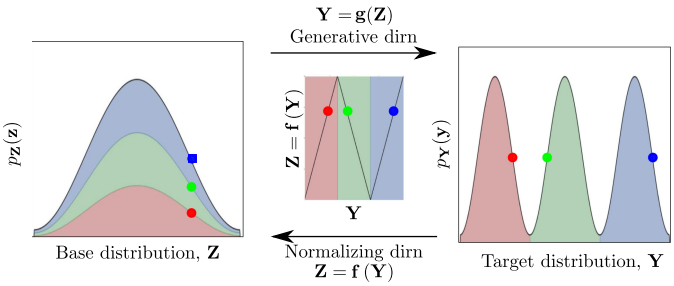

Fig. 5. Piecewise bijective coupling. The target domain (right) is divided into disjoint sections (colors) and each mapped by a monotone function (center) to the base distribution (left). For inverting the function, one samples a component of the base distribution using a gating network.

3.4.4.6 Sum-of-Squares polynomial flow. Jaini *et al.* [49] modeled $h(\cdot;\theta)$ as a strictly increasing polynomial. They proved such polynomials can approximate any strictly monotonic univariate continuous function. Hence, the resulting flow (*SOS* - sum of squares polynomial flow) is a universal flow.

The authors observed that the derivative of an increasing single-variable polynomial is a positive polynomial. Then they used a classical result from algebra: all positive single-variable polynomials are the sum of squares of polynomials. To get the coupling function, one needs to integrate the sum of squares

$$h(x;\theta) = c + \int_0^x \sum_{k=1}^{K} \left(\sum_{l=0}^{L} a_{kl} u^l \right)^2 du \,, \tag{27}$$

where L and K are hyperparameters (and, as noted in the paper, can be chosen to be 2).

SOS is easier to train than NAF, because there are no restrictions on the parameters (like positivity of weights). For L=0, SOS reduces to the affine coupling function and so it is a generalization of the basic affine flow.

3.4.4.7 Piecewise-bijective coupling. Dinh *et al.* [21] explore the idea that a coupling function does not need to be bijective, but just piecewise-bijective (Fig. 5). Formally, they consider a function $h(\cdot;\theta) : \mathbb{R} \to \mathbb{R}$ and a covering of the domain into K disjoint subsets: $\mathbb{R} = \bigsqcup_{i=1}^{K} A_i$, such that the restriction of the function onto each subset $h(\cdot;\theta)|_{A_i}$ is injective.

Dinh *et al.* [21] constructed a flow $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$ with a coupling architecture and piecewise-bijective coupling function in the normalizing direction - from data distribution to (simpler) base distribution. There is a covering of the data domain, and each subset of this covering is separately mapped to the base distribution. Each part of the base distribution now receives contributions from each subset of the data domain. For sampling, [21] proposed a probabilistic mapping from the base to data domain.

More formally, denote the target y and base z, and consider a lookup function $\phi : \mathbb{R} \to [K] = \{1, \ldots, K\}$, such that $\phi(y) = k$, if $y \in A_k$. One can define a new map $\mathbb{R} \to \mathbb{R} \times [K]$, given by the rule $y \mapsto (h(y), \phi(y))$, and a density on a target space $p_{Z,[K]}(z,k) = p_{[K]|Z}(k|z)p_Z(z)$. One can think of this as an unfolding of the non-injective map h. In particular, for each point z one can find its pre-image by sampling from $p_{[K]|Z}$, which is called a *gating network*. Pushing forward

along this unfolded map is now well-defined and one gets the formula for the density p_Y

$$p_Y(y) = p_{Z,[K]}(h(y), \phi(y))|\mathrm{D}h(y)|. \tag{28}$$

This real and discrete (*RAD*) flow efficiently learns distributions with discrete structures (multimodal distributions, distributions with holes, discrete symmetries etc).

3.5 Residual Flows

Residual networks [40] are compositions of the function of the form

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} + F(\mathbf{x}). \tag{29}$$

Such a function is called *a residual connection*, and here the *residual block* $F(\cdot)$ is a feed-forward neural network of any kind (a CNN in the original paper).

The first attempts to build a reversible network architecture based on residual connections were made in *RevNets* [32] and *iRevNets* [47]. Their main motivation was to save memory during training and to stabilize computation. The central idea is a variation of additive coupling functions: consider a disjoint partition of $\mathbb{R}^D = \mathbb{R}^d \times \mathbb{R}^{D-d}$ denoted by $\mathbf{x} = (\mathbf{x}^A, \mathbf{x}^B)$ for the input and $\mathbf{y} = (\mathbf{y}^A, \mathbf{y}^B)$ for the output, and define a function

$$\begin{aligned} \mathbf{y}^A &= \mathbf{x}^A + F(\mathbf{x}^B) \\ \mathbf{y}^B &= \mathbf{x}^B + G(\mathbf{y}^A), \end{aligned} \tag{30}$$

where $F : \mathbb{R}^{D-d} \to \mathbb{R}^d$ and $G : \mathbb{R}^d \to \mathbb{R}^{D-d}$ are residual blocks. This network is invertible (by re-arranging the equations in terms of $\mathbf{x}^A$ and $\mathbf{x}^B$ and reversing their order) but computation of the Jacobian is inefficient.

A different point of view on reversible networks comes from a dynamical systems perspective via the observation that a residual connection is a discretization of a first order ordinary differential equation (see Section 3.6 for more details). [12], [13] proposed several architectures, some of these networks were demonstrated to be invertible. However, the Jacobian determinants of these networks cannot be computed efficiently.

Other research has focused on making the residual connection $\mathbf{g}(\cdot)$ invertible. A sufficient condition for the invertibility was found in [7]. They proved the following statement:

Proposition 7. *A residual connection (29) is invertible, if the Lipschitz constant of the residual block is* $\mathrm{Lip}(F) < 1$.

There is no analytically closed form for the inverse, but it can be found numerically using fixed-point iterations (which, by the Banach theorem, converge if we assume $\mathrm{Lip}(F) < 1$).

Controlling the Lipschitz constant of a neural network is not simple. The specific architecture proposed by Behrmann *et al.* [7], called *iResNet*, uses a convolutional network for the residual block. It constrains the spectral radius of each convolutional layer in this network to be less than one.

The Jacobian determinant of the iResNet cannot be computed directly, so the authors propose to use a (biased) stochastic estimate. The Jacobian of the residual connection $\mathbf{g}$ in Equation (29) is: $\mathrm{D}\mathbf{g} = I + \mathrm{D}F$. Because the function F is assumed to be Lipschitz with $\mathrm{Lip}(F) < 1$, one has: $|\det(I + \mathrm{D}F)| = \det(I + \mathrm{D}F)$. Using the linear algebra identity, $\ln\det\mathbf{A} = \mathrm{Tr}\ln\mathbf{A}$ we have

$$\ln|\det \mathrm{D}\mathbf{g}| = \ln\det(I + \mathrm{D}F) = \mathrm{Tr}(\ln(I + \mathrm{D}F)), \tag{31}$$

Then one considers a power series for the trace of the matrix logarithm

$$\mathrm{Tr}(\ln(I + \mathrm{D}F)) = \sum_{k=1}^{\infty}(-1)^{k+1}\frac{\mathrm{Tr}(\mathrm{D}F)^k}{k}. \tag{32}$$

By truncating this series one can calculate an approximation to the log Jacobian determinant of $\mathbf{g}$. To efficiently compute each member of the truncated series, the Hutchinson trick was used. This trick provides a stochastic estimation of of a matrix trace $\mathbf{A} \in \mathbb{R}^{D\times D}$, using the relation: $\mathrm{Tr}\mathbf{A} = \mathbb{E}_{p(\mathbf{v})}[\mathbf{v}^T\mathbf{A}\mathbf{v}]$, where $\mathbf{v} \in \mathbb{R}^D$, $\mathbb{E}[\mathbf{v}] = 0$, and $\mathrm{cov}(\mathbf{v}) = I$.

Truncating the power series gives a biased estimate of the log Jacobian determinant (the bias depends on the truncation error). An unbiased stochastic estimator was proposed by Chen *et al.* [16] in a model they called a *Residual flow*. The authors used a *Russian roulette* estimator instead of truncation. Informally, every time one adds the next term a_{n+1} to the partial sum $\sum_{i=1}^{n} a_i$ while calculating the series $\sum_{i=1}^{\infty} a_i$, one flips a coin to decide if the calculation should be continued or stopped. During this process one needs to re-weight terms for an unbiased estimate.

3.6 Infinitesimal (Continuous) Flows

The residual connections discussed in the previous section can be viewed as discretizations of a first order ordinary differential equation (ODE) [26], [37]

$$\frac{d}{dt}\mathbf{x}(t) = F(\mathbf{x}(t), \theta(t)), \tag{33}$$

where $F : \mathbb{R}^D \times \Theta \to \mathbb{R}^D$ is a function which determines the dynamic (the *evolution function*), Θ is a set of parameters and $\theta : \mathbb{R} \to \Theta$ is a parameterization. The discretization of this equation (Euler's method) is

$$\mathbf{x}_{n+1} - \mathbf{x}_n = \varepsilon F(\mathbf{x}_n, \theta_n), \tag{34}$$

and this is equivalent to a residual connection with a residual block $\varepsilon F(\cdot, \theta_n)$.

In this section we consider the case where we do not discretize but try to learn the continuous dynamical system instead. Such flows are called *infinitesimal* or *continuous*. We consider two distinct types. The formulation of the first type comes from ordinary differential equations, and of the second type from stochastic differential equations.

3.6.1 ODE-Based Methods

Consider an ODE as in Equation (33), where $t \in [0, 1]$. Assuming uniform Lipschitz continuity in $\mathbf{x}$ and continuity in t, the solution exists (at least, locally) and, given an initial condition $\mathbf{x}(0) = \mathbf{z}$, is unique (Picard-Lindelöf-Lipschitz-Cauchy theorem [5]). We denote the solution at each time t as $\Phi^t(\mathbf{z})$.

Remark 8. At each time t, $\Phi^t(\cdot) : \mathbb{R}^D \to \mathbb{R}^D$ is a diffeomorphism and satisfies the group law: $\Phi^t \circ \Phi^s = \Phi^{t+s}$. Mathematically speaking, an ODE (33) defines a one-parameter group of diffeomorphisms on $\mathbb{R}^D$. Such a group is called

a smooth flow in dynamical systems theory and differential geometry [52].

When $t = 1$, the diffeomorphism $\Phi^1(\cdot)$ is called a *time one map*. The idea to model a normalizing flow as a time one map $\mathbf{y} = \mathbf{g}(\mathbf{z}) = \Phi^1(\mathbf{z})$ was presented by Chen *et al.* [15] under the name *Neural ODE (NODE)*. From a deep learning perspective this can be seen as an "infinitely deep" neural network with input $\mathbf{z}$, output $\mathbf{y}$ and continuous weights $\theta(t)$. The invertibility of such networks naturally comes from the theorem of the existence and uniqueness of the solution of the ODE.

Training these networks for a supervised downstream task can be done by the *adjoint sensitivity method* which is the continuous analog of backpropagation. It computes the gradients of the loss function by solving a second (*augmented*) ODE backwards in time. For loss $L(\mathbf{x}(t))$, where $\mathbf{x}(t)$ is a solution of ODE (33), its sensitivity or adjoint is $\mathbf{a}(t) = \frac{dL}{d\mathbf{x}(t)}$. This is the analog of the derivative of the loss with respect to the hidden layer. In a standard neural network, the backpropagation formula computes this derivative: $\frac{dL}{d\mathbf{h}_n} = \frac{dL}{d\mathbf{h}_{n+1}} \frac{d\mathbf{h}_{n+1}}{d\mathbf{h}_n}$. For "infinitely deep" neural network, this formula changes into an ODE

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t) \frac{dF(\mathbf{x}(t), \theta(t))}{d\mathbf{x}(t)}. \tag{35}$$

For density estimation learning, we do not have a loss, but instead seek to maximize the log likelihood. For normalizing flows, the change of variables formula is given by another ODE

$$\frac{d}{dt} \log\left(p(\mathbf{x}(t))\right) = -\mathrm{Tr}\left(\frac{dF(\mathbf{x}(t))}{d\mathbf{x}(t)}\right). \tag{36}$$

Note that we no longer need to compute the determinant. To train the model and sample from $p_{\mathbf{Y}}$ we solve these ODEs, which can be done with any numerical ODE solver.

Grathwohl *et al.* [34] used the Hutchinson estimator to calculate an unbiased stochastic estimate of the trace-term. This approach which they termed *FFJORD* reduces the complexity even further. Finlay, Jacobsen *et al.* [29] added two regularization terms into the loss function of FFJORD: the first term forces solution trajectories to follow straight lines with constant speed, and the second term is the Frobenius norm of the Jacobian. This regularization decreased the training time significantly and reduced the need for multiple GPUs. An interesting side-effect of using continuous ODE-type flows is that one needs fewer parameters to achieve the similar performance. For example, Grathwohl *et al.* [34] show that for the comparable performance on CIFAR10, FFJORD uses less than 2 percent as many parameters as Glow.

Not all diffeomorphisms can be presented as a time one map of an ODE (see [3], [52]). For example, one necessary condition is that the map is *orientation preserving* which means that the Jacobian determinant must be positive. This can be seen because the solution Φ^t is a (continuous) path in the space of diffeomorphisms from the identity map $\Phi^0 = Id$ to the time one map Φ^1. Since the Jacobian determinant of a diffeomorphism is nonzero, its sign cannot change along the path. Hence, a time one map must have a positive Jacobian determinant. For example, consider a map $f : \mathbb{R} \to \mathbb{R}$, such that $f(x) = -x$. It is obviously a diffeomorphism, but it can not be presented as a time one map of any ODE, because it is not orientation preserving.

Dupont *et al.* [23] suggested how one can improve Neural ODE in order to be able to represent a broader class of diffeomorphisms. Their model is called Augmented Neural ODE (*ANODE*). They add variables $\hat{\mathbf{x}}(t) \in \mathbb{R}^p$ and consider a new ODE

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}(t)\\ \hat{\mathbf{x}}(t)\end{bmatrix} = \hat{F}\left(\begin{bmatrix}\mathbf{x}(t)\\ \hat{\mathbf{x}}(t)\end{bmatrix}, \theta(t)\right), \tag{37}$$

with initial conditions $\mathbf{x}(0) = \mathbf{z}$ and $\hat{\mathbf{x}}(0) = 0$. The addition of $\hat{\mathbf{x}}(t)$ in particular gives freedom for the Jacobian determinant to remain positive. As was demonstrated in the experiments, ANODE is capable of learning distributions that the Neural ODE cannot, and the training time is shorter. Zhang *et al.* [106] proved that any diffeomorphism can be represented as a time one map of ANODE and so this is a universal flow.

A similar ODE-base approach was taken by Salman *et al.* [83] in Deep Diffeomorphic Flows. In addition to modelling a path $\Phi^t(\cdot)$ in the space of all diffeomorphic transformations, for $t \in [0, 1]$, they proposed geodesic regularisation in which longer paths are punished.

3.6.2 SDE-Based Methods (Langevin Flows)

The idea of the Langevin flow is simple; we start with a complicated and irregular data distribution $p_{\mathbf{Y}}(\mathbf{y})$ on $\mathbb{R}^D$, and then mix it to produce the simple base distribution $p_{\mathbf{Z}}(\mathbf{z})$. If this mixing obeys certain rules, then this procedure can be invertible. This idea was explored by Chen *et al.* [87], Jankowiak and Obermeyer [103], Rezende and Mohamed [78], Salimans *et al.* [84], Sohl-Dickstein *et al.* [81], Suykens *et al.* [50], Welling and Teh [14]. We provide a high-level overview of the method, including the necessary mathematical background.

A stochastic differential equation (SDE) or Itô process describes a change of a random variable $\mathbf{x} \in \mathbb{R}^D$ as a function of time $t \in \mathbb{R}_+$

$$d\mathbf{x}(t) = b(\mathbf{x}(t), t)dt + \sigma(\mathbf{x}(t), t)dB_t, \tag{38}$$

where $b(\mathbf{x}, t) \in \mathbb{R}^D$ is the *drift coefficient*, $\sigma(\mathbf{x}, t) \in \mathbb{R}^{D\times D}$ is the *diffusion coefficient*, and B_t is D-dimensional *Brownian motion*. One can interpret the drift term as a deterministic change and the diffusion term as providing the stochasticity and mixing. Given some assumptions about these functions, the solution exists and is unique [72].

Given a time-dependent random variable $\mathbf{x}(t)$ we can consider its density function $p(\mathbf{x}, t)$ and this is also time dependent. If $\mathbf{x}(t)$ is a solution of Equation (38), its density function satisfies two partial differential equations describing the forward and backward evolution [72]. The forward evolution is given by Fokker-Plank equation or Kolmogorov's forward equation

TABLE 1
List of Normalizing Flows for Which We Show Performance Results

Architecture	Coupling function	Flow name
Coupling, 3.4.1	Affine, 3.4.4.1	RealNVP Glow
	Mixture CDF, 3.4.4.3	Flow++
	Splines, 3.4.4.4	quadratic (C) cubic RQ-NSF(C)
	Piecewise Bijective, 3.4.4.7	RAD
Autoregressive, 3.4.2	Affine	MAF
	Polynomial, 3.4.4.6	SOS
	Neural Network, 3.4.4.5	NAF UMNN
	Splines	quadratic (AR) RQ-NSF(AR)
Residual, 3.5		iResNet
		Residual flow
ODE, 3.6.1		FFJORD

$$\frac{\partial}{\partial t}p(\mathbf{x},t) = -\nabla_{\mathbf{x}} \cdot (b(\mathbf{x},t)p(\mathbf{x},t)) + \sum_{i,j} \frac{\partial^2}{\partial x_i \partial x_j} D_{ij}(\mathbf{x},t)p(\mathbf{x},t), \tag{39}$$

where $D = \frac{1}{2}\sigma\sigma^T$, with the initial condition $p(\cdot,0) = p_{\mathbf{Y}}(\cdot)$. The reverse is given by Kolmogorov's backward equation

$$-\frac{\partial}{\partial t}p(\mathbf{x},t) = b(\mathbf{x},t) \cdot \nabla_{\mathbf{x}}(p(\mathbf{x},t)) + \sum_{i,j} D_{ij}(\mathbf{x},t) \frac{\partial^2}{\partial x_i \partial x_j} p(\mathbf{x},t), \tag{40}$$

where $0 < t < T$, and the initial condition is $p(\cdot,T) = p_{\mathbf{Z}}(\cdot)$.

Asymptotically the Langevin flow can learn any distribution if one picks the drift and diffusion coefficients appropriately [87]. However this result is not very practical, because one needs to know the (unnormalized) density function of the data distribution.

One can see that if the diffusion coefficient is zero, the Itô process reduces to the ODE (33), and the Fokker-Plank equation becomes a Liouville's equation, which is connected to Equation (36) (see [15]). It is also equivalent to the form of the transport equation considered in [50] for stochastic optimization.

Sohl-Dickstein *et al.* [84] and Salimans *et al.* [81] suggested using MCMC methods to model the diffusion. They considered discrete time $t = 0, \ldots, T$. For each time t, $\mathbf{x}^t$ is a random variable where $\mathbf{x}^0 = \mathbf{y}$ is the data point, and $\mathbf{x}^T = \mathbf{z}$ is the base point. The forward transition probability $q(\mathbf{x}^t|\mathbf{x}^{t-1})$ is taken to be either normal or binomial distribution with trainable parameters. Kolmogorov's backward equation implies that the backward transition $p(\mathbf{x}^{t-1}|\mathbf{x}^t)$ must have the same functional form as the forward transition (i.e., be either normal or binomial). Denote: $q(\mathbf{x}^0) = p_{\mathbf{Y}}(\mathbf{y})$, the data distribution, and $p(\mathbf{x}^T) = p_{\mathbf{Z}}(\mathbf{z})$, the base distribution. Applying the backward transition to the base distribution, one obtains a new density $p(\mathbf{x}^0)$, which one wants to match with $q(\mathbf{x}^0)$. Hence, the optimization objective is the log likelihood $L = \int d\mathbf{x}^0 q(\mathbf{x}^0) \log p(\mathbf{x}^0)$. This is intractable, but one can find a lower bound as in variational inference.

Several papers have worked explicitly with the SDE [14], [62], [63], [76], [96]. Chen *et al.* [14] use SDEs to create an interesting posterior for variational inference. They sample a latent variable $\mathbf{z}_0$ conditioned on the input $\mathbf{x}$, and then evolve $\mathbf{z}_0$ with SDE. In practice this evolution is computed by discretization. By analogy to Neural ODEs, *Neural Stochastic Differential Equations* were proposed [76], [96]. In this approach coefficients of the SDE are modelled as neural networks, and black box SDE solvers are used for inference. To train Neural SDE one needs an analog of backpropagation, Tzen and Raginsky [96] proposed the use of Kunita's theory of stochastic flows. Following this, Li *et al.* [62] derived the adjoint SDE whose solution gives the gradient of the original Neural SDE.

Note, that even though Langevin flows manifest nice mathematical properties, they have not found practical applications. In particular, none of the methods has been tested on baseline datasets for flows.

TABLE 2
Tabular Datasets: Data Dimensionality and Number of Training Examples

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Dims	6	8	21	43	63
#Train	$\approx$ 1.7M	$\approx$ 800K	$\approx$ 300K	$\approx$ 30K	$\approx$ 1M

4 Datasets and Performance

In this section we discuss datasets commonly used for training and testing normalizing flows. We provide comparison tables of the results as they were presented in the corresponding papers. The list of the flows for which we post the performance results is given in Table 1.

4.1 Tabular Datasets

We describe datasets as they were preprocessed in [74] (Table 2).[2] These datasets are relatively small and so are a reasonable first test of unconditional density estimation models. All datasets were cleaned and de-quantized by adding uniform noise, so they can be considered samples from an absolutely continuous distribution.

We use a collection of datasets from the UC Irvine machine learning repository [22].

1) POWER: a collection of electric power consumption measurements in one house over 47 months.
2) GAS: a collection of measurements from chemical sensors in several gas mixtures.
3) HEPMASS: measurements from high-energy physics experiments aiming to detect particles with unknown mass.
4) MINIBOONE: measurements from MiniBooNE experiment for observing neutrino oscillations.

In addition we consider the Berkeley segmentation dataset [66] which contains segmentations of natural images. [74] extracted 8×8 random monochrome patches from it.

In Table 3 we compare performance of flows for these tabular datasets. For experimental details, see the following papers: RealNVP [20] and MAF [74], Glow [57] and FFJORD

2. See https://github.com/gpapamak/maf

TABLE 3
Average Test Log-Likelihood (in Nats) for Density Estimation on Tabular Datasets (Higher the Better)

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
MAF(5)	0.14 ± 0.01	9.07 ± 0.02	-17.70 ± 0.02	-11.75 ± 0.44	155.69 ± 0.28
MAF(10)	0.24 ± 0.01	10.08 ± 0.02	-17.73 ± 0.02	-12.24 ± 0.45	154.93 ± 0.28
MAF MoG	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	-11.68 ± 0.44	156.36 ± 0.28
RealNVP(5)	-0.02 ± 0.01	4.78 ± 1.8	-19.62 ± 0.02	-13.55 ± 0.49	152.97 ± 0.28
RealNVP(10)	0.17 ± 0.01	8.33 ± 0.14	-18.71 ± 0.02	-13.84 ± 0.52	153.28 ± 1.78
Glow	0.17	8.15	-18.92	-11.35	155.07
FFJORD	0.46	8.59	-14.92	-10.43	157.40
NAF(5)	0.62 ± 0.01	11.91 ± 0.13	-15.09 ± 0.40	**-8.86** ± 0.15	157.73 ± 0.04
NAF(10)	0.60 ± 0.02	11.96 ± 0.33	-15.32 ± 0.23	-9.01 ± 0.01	157.43 ± 0.30
UMNN	0.63 ± 0.01	10.89 ± 0.70	**-13.99** ± 0.21	-9.67 ± 0.13	**157.98** ± 0.01
SOS(7)	0.60 ± 0.01	11.99 ± 0.41	-15.15 ± 0.10	-8.90 ± 0.11	157.48 ± 0.41
Quadratic Spline (C)	0.64 ± 0.01	12.80 ± 0.02	-15.35 ± 0.02	-9.35 ± 0.44	157.65 ± 0.28
Quadratic Spline (AR)	**0.66** ± 0.01	12.91 ± 0.02	-14.67 ± 0.03	-9.72 ± 0.47	157.42 ± 0.28
Cubic Spline	0.65 ± 0.01	**13.14** ± 0.02	-14.59 ± 0.02	-9.06 ± 0.48	157.24 ± 0.07
RQ-NSF(C)	0.64 ± 0.01	13.09 ± 0.02	-14.75 ± 0.03	-9.67 ± 0.47	157.54 ± 0.28
RQ-NSF(AR)	**0.66** ± 0.01	13.09 ± 0.02	-14.01 ± 0.03	-9.22 ± 0.48	157.31 ± 0.28

A number in parenthesis next to a flow indicates number of layers. MAF MoG is MAF with mixture of Gaussians as a base density.

[34], NAF [45], UMNN [102], SOS [49], Quadratic Spline flow and RQ-NSF [25], Cubic Spline Flow [24].

Table 3 shows that universal flows (NAF, SOS, Splines) demonstrate relatively better performance.

4.2 Image Datasets

These datasets summarized in Table 4. They are of increasing complexity and are preprocessed as in [20] by dequantizing with uniform noise (except for Flow++).

Table 5 compares performance on the image datasets for unconditional density estimation. For experimental details, see: RealNVP for CIFAR-10 and ImageNet [20], Glow for CIFAR-10 and ImageNet [57], RealNVP and Glow for MNIST, MAF and FFJORD [34], SOS [49], RQ-NSF [25], UMNN [102], iResNet [7], Residual Flow [16], Flow++ [41].

As of this writing Flow++ [41] is the best performing approach. Besides using more expressive coupling layers (see Section 3.4.4.3) and a different architecture for the conditioner, variational dequantization was used instead of uniform. An ablation study shows that the change in dequantization approach gave the most significant improvement.

5 Discussion and Open Problems

5.1 Inductive Biases

5.1.1 Role of the Base Measure

The base measure of a normalizing flow is generally assumed to be a simple distribution (e.g., uniform or Gaussian). However this doesn't need to be the case. Any distribution where we can easily draw samples and compute the log probability density function is possible and the parameters of this distribution can be learned during training.

Theoretically the base measure shouldn't matter: any distribution for which a CDF can be computed, can be simulated by applying the inverse CDF to draw from the uniform distribution. However in practice if structure is provided in the base measure, the resulting transformations may become easier to learn. In other words, the choice of base measure can be viewed as a form of prior or inductive bias on the distribution and may be useful in its own right. For example, a trade-off between the complexity of the generative transformation and the form of base measure was explored in [48] in the context of modelling tail behaviour.

TABLE 4
Image Datasets: Data Dimensionality and Number of Training Examples for MNIST, CIFAR-10, ImageNet32 and ImageNet64 Datasets

	MNIST	CIFAR-10	ImNet32	ImNet64
Dims	784	3072	3072	12288
#Train	50K	90K	≈ 1.3M	≈ 1.3M

5.1.2 Form of Diffeomorphisms

The majority of the flows explored are triangular flows (either coupling or autoregressive architectures). Residual networks and Neural ODEs are also being actively investigated and applied. A natural question to ask is: are there other ways to model diffeomorphisms which are efficient for computation? What inductive bias does the architecture impose? For instance, Spantini, Bigoni, and Marzouk [85] investigate the relation between the sparsity of the triangular flow and Markov property of the target distribution.

TABLE 5
Average Test Negative Log-Likelihood (in Bits per Dimension) for Density Estimation on Image Datasets (Lower is Better)

	MNIST	CIFAR-10	ImNet32	ImNet64
RealNVP	1.06	3.49	4.28	3.98
Glow	1.05	3.35	4.09	3.81
MAF	1.89	4.31		
FFJORD	0.99	3.40		
SOS	1.81	4.18		
RQ-NSF(C)		3.38		3.82
UMNN	1.13			
iResNet	1.06	3.45		
Residual Flow	0.97	3.28	4.01	3.76
Flow++		**3.08**	**3.86**	**3.69**

A related question concerns the best way to model conditional normalizing flows when one needs to learn a conditional probability distribution. Trippe and Turner [95] suggested using different flows for each condition, but this approach doesn't leverage weight sharing, and so is inefficient in terms of memory and data usage. Atanov, Volokhova, Ashukha, Sosnovik, and Vetrov [6] proposed using affine coupling layers where the parameters θ depend on the condition. Conditional distributions are useful in particular for time series modelling, where one needs to find $p(y_t|\mathbf{y}_{<t})$ [60].

5.1.3 Loss Function

The majority of the existing flows are trained by minimization of KL-divergence between source and the target distributions (or, equivalently, with log-likelihood maximization). However, other losses could be used which would put normalizing flows in a broader context of optimal transport theory [99]. Interesting work has been done in this direction including Flow-GAN [36] and the minimization of the Wasserstein distance as suggested by [4], [90].

5.2 Generalisation to Non-Euclidean Spaces

5.2.1 Flows on Manifolds

Modelling probability distributions on manifolds has applications in many fields including robotics, molecular biology, optics, fluid mechanics, and plasma physics [30], [79]. How best to construct a normalizing flow on a general differentiable manifold remains an open question. One approach to applying the normalizing flow framework on manifolds, is to find a base distribution on the euclidean space and transfer it to the manifold of interest. There are two main approaches: 1) embed the manifold in the euclidean space and "restrict" the measure, or 2) induce the measure from the tangent space to the manifold. We will briefly discuss each in turn.

One can also use differential structure to define measures on manifolds [86]. Every differentiable and orientable manifold M has a volume form ω, then for a Borel subset $U \subset M$ one can define its measure as $\mu_\omega(U) = \int_U \omega$. A Riemannian manifold has a natural volume form given by its metric tensor: $\omega = \sqrt{|g|}dx_1 \wedge \ldots \wedge dx_D$. Gemici *et al.* [30] explore this approach considering an immersion of an D-dimensional manifold M into a euclidean space: $\phi : M \to \mathbb{R}^N$, where $N \geq D$. In this case, one pulls-back a euclidean metric, and locally a volume form on M is $\omega = \sqrt{\det((\mathrm{D}\phi)^T \mathrm{D}\phi)}dx_1 \wedge \ldots \wedge dx_D$, where $\mathrm{D}\phi$ is the Jacobian matrix of ϕ. Rezende *et al.* [79] pointed out that the realization of this method is computationally hard, and proposed an alternative construction of flows on tori and spheres using diffeomorphisms of the one-dimensional circle as building blocks.

As another option, one can consider exponential maps $\exp_x : T_x M \to M$, mapping a tangent space of a Riemannian manifold (at some point x) to the manifold itself. If the manifold is geodesic complete, this map is globally defined, and locally is a diffeomorphism. A tangent space has a structure of a vector space, so one can choose an isomorphism $T_x M \cong \mathbb{R}^D$. Then for a base distribution with the density $p_{\mathbf{Z}}$ on $\mathbb{R}^D$, one can push it forward on M via the exponential map. Additionally, applying a normalizing flow to a base measure before pushing it to M helps to construct multimodal distributions on M. If the manifold M is a hyperbolic space, the exponential map is a global diffeomorphism and all the formulas could be written explicitly. Using this method, Ovinnikov [73] introduced the Gaussian reparameterization trick in a hyperbolic space and Bose *et al.* [10] constructed hyperbolic normalizing flows.

Instead of a Riemannian structure, one can impose a Lie group structure on a manifold G. In this case there also exists an exponential map $\exp : \mathfrak{g} \to G$ mapping a Lie algebra to the Lie group and one can use it to construct a normalizing flow on G. Falorsi *et al.* [28] introduced an analog of the Gaussian reparameterization trick for a Lie group.

5.2.2 Discrete Distributions

Modelling distributions over discrete spaces is important in a range of problems, however the generalization of normalizing flows to discrete distributions remains an open problem in practice. Discrete latent variables were used by Dinh *et al.* [21] as an auxiliary tool to pushforward continuous random variables along piecewise-bijective maps (see Section 3.4.4.7). However, can we define normalizing flows if one or both of our distributions are discrete? This could be useful for many applications including natural language modelling, graph generation and others.

To this end Tran *et al.* [94] model bijective functions on a finite set and show that, in this case, the change of variables is given by the formula: $p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{Z}}(\mathbf{g}^{-1}(\mathbf{y}))$, i.e., with no Jacobian term (compare with Definition 1). For backpropagation of functions with discrete variables they use the straight-through gradient estimator [8]. However this method is not scalable to distributions with large numbers of elements.

Alternatively Hoogeboom *et al.* [43] models bijections on $\mathbb{Z}^D$ directly with additive coupling layers. Other approaches transform a discrete variable into a continuous latent variable with a variational autoencoder, and then apply normalizing flows in the continuous latent space [101], [108].

A different approach is dequantization, (i.e., adding noise to discrete data to make it continuous) which can be used with ordinal variables, e.g., discretized pixel intensities. The noise can be uniform but other forms are possible and this dequantization can even be learned as a latent variable model [41], [44]. Hoogeboom *et al.* [44] analyzed how different choices of dequantization objectives and dequantization distributions affect the performance.

Acknowledgments

The authors would like to thank Matt Taylor and Kry Yik-Chau Lui for their insightful comments.

References

[1] A. Abdelhamed, M. A. Brubaker, and M. S. Brown, "Noise flow: Noise modeling with conditional normalizing flows," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 3165–3173.

[2] J. Agnelli, M. Cadeiras, E. Tabak, T. Cristina, and E. Vanden-Eijnden, "Clustering and classification through normalizing flows in feature space," *Multiscale Model. Simul.*, vol. 8, pp. 1784–1802, 2010.

[3] J. Arango and A. Gómez, "Diffeomorphisms as time one maps," *Aequationes Math.*, vol. 64, pp. 304–314, 2002.

[4] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein Generative Adversarial Networks," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 214–223.

[5] V. Arnold, *Ordinary Differential Equations*. Cambridge, MA, USA: The MIT Press, 1978.

[6] A. Atanov, A. Volokhova, A. Ashukha, I. Sosnovik, and D. Vetrov, "Semi-conditional normalizing flows for semi-supervised learning," in *Workshop Invertible Neural Nets Normalizing Flows (ICML)*, 2019.

[7] J. Behrmann, D. Duvenaud, and J.-H. Jacobsen, "Invertible residual networks," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019.

[8] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.

[9] V. Bogachev, A. Kolesnikov, and K. Medvedev, "Triangular transformations of measures," *Sbornik Math.*, vol. 196, no. 3/4, pp. 309–335, 2005.

[10] A. J. Bose, A. Smofsky, R. Liao, P. Panangaden, and W. L. Hamilton, "Latent variable modelling with hyperbolic normalizing flows," 2020, *arXiv: 2002.06336*.

[11] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio, "Generating sentences from a continuous space," in *Proc. 20th SIGNLL Conf. Comput. Natural Lang. Learn.*, 2015, pp. 10–21.

[12] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham, "Reversible architectures for arbitrarily deep residual neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2811–2818.

[13] B. Chang, M. Chen, E. Haber, and E. H. Chi, "AntisymmetricRNN: A dynamical system view on recurrent neural networks," in *Proc. Int. Conf. Learn. Representations*, 2019.

[14] C. Chen, C. Li, L. Chen, W. Wang, Y. Pu, and L. Carin, "Continuous-time flows for efficient inference and density estimation," in *Proc. Int. Conf. Mach. Learn.*, 2018.

[15] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6572–6583.

[16] R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J.-H. Jacobsen, "Residual flows for invertible generative modeling," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019.

[17] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 53–65, Jan. 2018.

[18] H. P. Das, P. Abbeel, and C. J. Spanos, "Dimensionality reduction flows," 2019, *arXiv: 1908.01686*.

[19] L. Dinh, D. Krueger, and Y. Bengio, "NICE: Non-linear independent components estimation," in *Proc. Int. Conf. Learn. Representations Workshop*, 2015.

[20] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *Proc. Int. Conf. Learn. Representations*, 2017.

[21] L. Dinh, J. Sohl-Dickstein, R. Pascanu, and H. Larochelle, "A RAD approach to deep mixture models," in *Proc. Int. Conf. Learn. Representations Workshop*, 2019.

[22] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[23] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented neural ODEs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 3140–3150.

[24] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Cubic-spline flows," in *Workshop Invertible Neural Networks Normalizing Flows (ICML)*, 2019.

[25] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural spline flows," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 7511–7522.

[26] Weinan E, "A proposal on machine learning via dynamical systems," *Commun. Math. Statist.*, vol. 5, pp. 1–11, 2017.

[27] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos, "Universal audio synthesizer control with normalizing flows," 2019, *arXiv: 1907.00971*.

[28] L. Falorsi, P. de Haan, T. R. Davidson, and P. Forré, "Reparameterizing distributions on lie groups," 2019, *arXiv: 1903.02958*.

[29] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman, "How to train your neural ODE," 2020, *arXiv: 2002.02798*.

[30] M. C. Gemici, D. Rezende, and S. Mohamed, "Normalizing flows on riemannian manifolds," 2016, *arXiv:1611.02304*.

[31] M. Germain, K. Gregor, I. Murray, and H. Larochelle, "MADE: Masked autoencoder for distribution estimation," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 881–889.

[32] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2211–2221.

[33] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[34] W. Grathwohl, R. T. Q Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "FFJORD: Free-form continuous dynamics for scalable reversible generative models," in *Proc. Int. Conf. Learn. Representations*, 2019.

[35] J. Gregory and R. Delbourgo, "Piecewise rational quadratic interpolation to monotonic data," *IMA J. Numer. Anal.*, vol. 2, no. 2, pp. 123–130, 1982.

[36] A. Grover, M. Dhar, and S. Ermon, "Flow-GAN: Combining maximum likelihood and adversarial learning in generative models," in *Proc. AAAI Conf. Artif. Intell.*, 2018.

[37] E. Haber, L. Ruthotto, and E. Holtham, "Learning across scales - A multiscale method for convolution neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2018.

[38] L. Hasenclever, J. M. Tomczak, R. Van Den Berg, and M. Welling, "Variational inference with orthogonal normalizing flows," in *Workshop Bayesian Deep Learn. (NeurIPS)*, 2017.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[41] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving flow-based generative models with variational dequantization and architecture design," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 2722–2730.

[42] E. Hoogeboom, R. V. D. Berg, and M. Welling, "Emerging convolutions for generative normalizing flows," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 2771–2780.

[43] E. Hoogeboom, J. W. Peters, R. van den Berg, and M. Welling, "Integer discrete flows and lossless compression," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019.

[44] E. Hoogeboom, T. S. Cohen, and J. M. Tomczak, "Learning discrete distributions by dequantization," 2020, *arXiv: 2001.11235*.

[45] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, "Neural autoregressive flows," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 2078–2087.

[46] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[47] J.-H. Jacobsen, A. W. Smeulders, and E. Oyallon, "i-RevNet: Deep invertible networks," in *Proc. Int. Conf. Learn. Representations*, 2018.

[48] P. Jaini, I. Kobyzev, M. Brubaker, and Y. Yu, "Tails of triangular flows," 2019, *arXiv: 1907.04481*.

[49] P. Jaini, K. A. Selby, and Y. Yu, "Sum-of-squares polynomial flow," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 3009–3018.

[50] M. Jankowiak and F. Obermeyer, "Pathwise derivatives beyond the reparameterization trick," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 2235–2244.

[51] G. Kanwar *et al.*, "Equivariant flow-based sampling for lattice gauge theory," 2020, *arXiv: 2003.06413*.

[52] A. Katok and B. Hasselblatt, *Introduction to the Modern Theory of Dynamical Systems*. New York, NY, USA: Cambridge Univ. Press, 1995.

[53] S. Kim, S. Gil Lee, J. Song, J. Kim, and S. Yoon, "FloWaveNet: A generative flow for raw audio," in *Proc. 36th Int. Conf. Mach. Learn.*, 2018, pp. 3370–3378.

[54] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. 2nd Int. Conf. Learn. Representations*, 2014.

[55] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," 2019, *arXiv: 1906.02691*.

[56] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4743–4751.

[57] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 10 215–10 224.

[58] J. Köhler, L. Klein, and F. Noé, "Equivariant flows: Sampling configurations for multi-body systems with symmetric energies," in *Workshop Mach. Learn. Physical Sciences (NeurIPS)*, 2019.

[59] D. Koller and N. Friedman, *Probabilistic Graphical Models*. Cambridge, MA, USA: MIT Press, 2009.

[60] M. Kumar *et al.*, "VideoFlow: A flow-based generative model for video," in *Workshop Invertible Neural Nets Normalizing Flows (ICML)*, 2019.

[61] P. M. Laurence, R. J. Pignol, and E. G. Tabak, "Constrained density estimation," in *Proc. Wolfgang Pauli Inst. Conf. Energy Commodity Trading*, 2014, pp. 259–284.

[62] X. Li, T.-K. L. Wong, R. T. Q. Chen, and D. Duvenaud, "Scalable gradients for stochastic differential equations," 2020, *arXiv: 2001.01328*.

[63] A. Liutkus, U. Simsekli, S. Majewski, A. Durmus, and F.-R. Stöter, "Sliced-wasserstein flows: Nonparametric generative modeling via optimal transport and diffusions," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 4104–4113.

[64] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. Int. Conf. Mach. Learn.*, 2013.

[65] K. Madhawa, K. Ishiguro, K. Nakago, and M. Abe, "GraphNVP: An invertible flow model for generating molecular graphs," 2019, *arXiv: 1905.11600*.

[66] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th Int. Conf. Comput. Vis.*, 2001, pp. 416–423.

[67] B. Mazoure, T. Doan, A. Durand, J. Pineau, and R. D. Hjelm, "Leveraging exploration in off-policy algorithms via normalizing flows," in *Proc. 3rd Conf. Robot Learn.*, 2019.

[68] K. V. Medvedev, "Certain properties of triangular transformations of measures," *Theory Stochastic Processes*, vol. 14, no. 30, pp. 95–99, 2008.

[69] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novak, "Neural importance sampling," *ACM Trans. Graph.*, vol. 38, 2018, Art. no. 145.

[70] P. Nadeem Ward, A. Smofsky, and A. Joey Bose, "Improving exploration in soft-actor-critic with normalizing flows policies," in *Workshop Invertible Neural Nets Normalizing Flows (ICML)*, 2019.

[71] F. Noé, S. Olsson, J. Köhler, and H. Wu, "Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning," *Science*, vol. 365, 2019, Art. no. eaaw1147.

[72] B. Oksendal, *Stochastic Differential Equations (3rd Ed.): An Introduction With Applications*. Berlin, Germany: Springer, 1992.

[73] I. Ovinnikov, "Poincaré wasserstein autoencoder," in *Workshop on Bayesian Deep Learning, NeurIPS*, 2018.

[74] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2335–2344.

[75] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," 2019, *arXiv: 1912.02762*.

[76] S. Peluchetti and S. Favaro, "Neural stochastic differential equations," 2019, *arXiv: 1905.11065*.

[77] R. Prenger, R. Valle, and B. Catanzaro, "Waveglow: A flow-based generative network for speech synthesis," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2019, pp. 3617–3621.

[78] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 1530–1538.

[79] D. J. Rezende *et al.*, "Normalizing flows on tori and spheres," 2020, *arXiv: 2002.02428*.

[80] O. Rippel and R. P. Adams, "High-dimensional probability estimation with deep density models," 2013, *arXiv:1302.5125*.

[81] T. Salimans, A. Diederik, D. P. Kingma, and M. Welling, "Markov chain Monte Carlo and variational inference: Bridging the gap," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 1218–1226.

[82] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2234–2242.

[83] H. Salman, P. Yadollahpour, T. Fletcher, and N. Batmanghelich, "Deep diffeomorphic normalizing flows," 2018, *arXiv: 1810.03256*.

[84] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 2256–2265.

[85] A. Spantini, D. Bigoni, and Y. Marzouk, "Inference via low-dimensional couplings," *J. Mach. Learn. Res.*, vol. 19, pp. 2639–2709, Mar. 2017.

[86] M. Spivak, *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*. San Francisco, CA, USA: Print, 1965.

[87] J. Suykens, H. Verrelst, and J. Vandewalle, "On-line learning Fokker-Planck machine," *Neural Process. Lett.*, vol. 7, pp. 81–89, 1998.

[88] E. G. Tabak and C. V. Turner, "A family of nonparametric density estimation algorithms," *Commun. Pure Appl. Math.*, vol. 66, no. 2, pp. 145–164, 2013.

[89] E. G. Tabak and E. Vanden-Eijnden, "Density estimation by dual ascent of the log-likelihood," *Commun. Math. Sci.*, vol. 8, no. 1, pp. 217–233, 2010.

[90] I. O. Tolstikhin, O. Bousquet, S. Gelly, and B. Schölkopf, "Wasserstein auto-encoders," in *Proc. Int. Conf. Learn. Representations*, 2018.

[91] J. Tomczak and M. Welling, "Improving variational auto-encoders using convex combination linear inverse autoregressive flow," *Benelearn*, 2017.

[92] J. M. Tomczak and M. Welling, "Improving variational auto-encoders using householder flow," 2016, *arXiv:1611.09630*.

[93] A. Touati, H. Satija, J. Romoff, J. Pineau, and P. Vincent, "Randomized value functions via multiplicative normalizing flows," in *Proc. Conf. Uncertainty Artif. Intell.*, 2019.

[94] D. Tran, K. Vafa, K. Agrawal, L. Dinh, and B. Poole, "Discrete flows: Invertible generative models of discrete data," in *Proc. Int. Conf. Learn. Representations Workshop*, 2019.

[95] B. L. Trippe and R. E. Turner, "Conditional density estimation with Bayesian normalising flows," in *Workshop Bayesian Deep Learn. (NeurIPS)*, 2017.

[96] B. Tzen and M. Raginsky, "Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit," 2019, *arXiv: 1905.09883*.

[97] R. van den Berg, L. Hasenclever, J. M. Tomczak, and M. Welling, "Sylvester normalizing flows for variational inference," in *Proc. 34th Conf. Uncertainty Artif. Intell.*, 2018.

[98] A. van den Oord *et al.*, "Parallel wavenet: Fast high-fidelity speech synthesis," in *Proc. 35th Int. Conf. Mach. Learn.*, 2017, pp. 3918–3926.

[99] C. Villani, *Topics in Optimal Transportation (Graduate Studies in Mathematics 58)*. Providence, RI, USA: American Mathematical Society, 2003.

[100] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, and F. Yue Wang, "Generative adversarial networks: Introduction and outlook," *IEEE/CAA J. Automatica Sinica*, vol. 4, no. 4, pp. 588–598, 2017.

[101] P. Z. Wang and W. Y. Wang, "Riemannian normalizing flow on variational wasserstein autoencoder for text modeling," 2019, *arXiv: 1904.02399*.

[102] A. Wehenkel and G. Louppe, "Unconstrained monotonic neural networks," 2019, *arXiv: 1908.05164*.

[103] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 681–688.

[104] P. Wirnsberger *et al.*, "Targeted free energy estimation via learned mappings," 2020, *arXiv: 2002.04913*.

[105] K. W. K. Wong, G. Contardo, and S. Ho, "Gravitational wave population inference with deep flow-based generative network," 2020, *arXiv: 2002.09491*.

[106] H. Zhang, X. Gao, J. Unterman, and T. Arodz, "Approximation capabilities of neural ordinary differential equations," 2019, *arXiv: 1907.12998*.

[107] G. Zheng, Y. Yang, and J. Carbonell, "Convolutional normalizing flows," in *Workshop Theoretical Foundations Applications Deep Generative Models (ICML)*, 2018.

[108] Z. M. Ziegler and A. M. Rush, "Latent normalizing flows for discrete sequences," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 7673–7682.

Ivan Kobyzev received the master's degree in mathematical physics from St Petersburg State University, Russia, in 2011, and the PhD degree in mathematics from Western University, Canada, in 2016. He did two postdocs in mathematics and in computer science at the University of Waterloo, Canada. Currently he is a researcher at Borealis AI, Canada. His research interests include algebra, generative models, cognitive computing, and natural language processing.

Simon J.D. Prince received the master's degree from University College London, United Kingdom and the doctorate degree from the University of Oxford, United Kingdom. He has a diverse research background and has published in wide-ranging areas including Computer Vision, Neuroscience, HCI, Computer Graphics, Medical Imaging, and Augmented Reality. He is also the author of a popular textbook on Computer Vision. From 2005–2012, he was a tenured faculty member with the Department of Computer Science, University College London, where he taught courses in Computer Vision, Image Processing and Advanced Statistical Methods. During this time, he was director of the MSc in Computer Vision, Graphics and Imaging. He worked in industry applying AI to computer graphics software. Currently he is a research director of Borealis AI's Montreal office.

Marcus A. Brubaker (Member, IEEE) received the PhD degree from the University of Toronto, Canada, in 2011. He did postdocs at the Toyota Technological Institute, Chicago, Toronto Rehabilitation Hospital and the University of Toronto, Canada. His research interests include computer vision, machine learning and statistics. He is currently an assistant professor at York University, Toronto, Canada, an adjunct professor at the University of Toronto, Canada and a faculty affiliate of the Vector Institute. He is also an academic advisor to Borealis AI, Canada where he previously worked as the research director of the Toronto office. He is also an associate editor for the journal *IET Computer Vision* and has served as a reviewer and an area chair for many computer vision and machine learning conferences.