Normalizing Flows: An Introduction and Review of Current Methods

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

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 [Koller and Friedman, 2009] explicitly model the conditional dependence between random variables. Recently, generative neural approaches have been proposed including generative adversarial networks (GANs) [Goodfellow et al., 2014] and variational auto-encoders (VAEs) [Kingma and Welling, 2014].

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 [Bowman et al., 2015; Salimans et al., 2016].

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 [Ho et al., 2019; Kingma and Dhariwal, 2018], video generation [Kumar et al., 2019], audio generation [Esling et al., 2019; Kim et al., 2018; Prenger et al., 2019], graph generation [Madhawa et al., 2019], and reinforcement learning [Nadeem Ward et al., 2019].

• *Borealis AI, Canada.*
ivan.kobyzev@borealisai.com
simon.prince@borealisai.com
marcus.brubaker@borealisai.com

There are several survey papers for VAEs [Kingma and Welling, 2019] and GANs [Creswell et al., 2018; Wang et al., 2017]. 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. [2019]. Their article is more tutorial in nature and provides many implementation details, 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 [2015] in the context of variational inference and by Dinh et al. [2015] for density estimation. However, the framework was previously defined in Tabak and Vanden-Eijnden [2010] and Tabak and Turner [2013] and explored for density estimation by Rippel and Adams [2013].

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 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

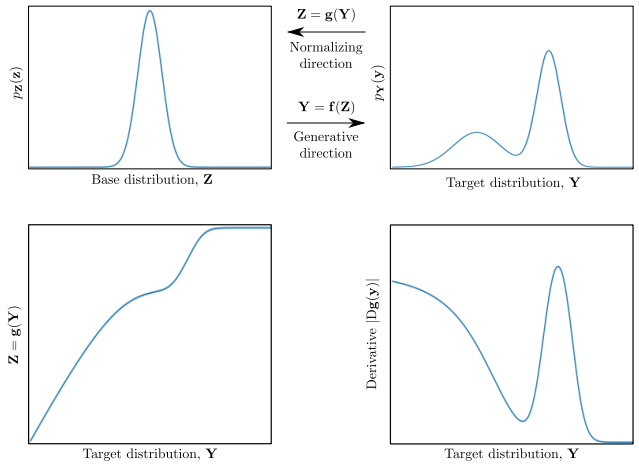

Fig. 1. Change of variables (Equation (1)). Top-left: the density of the source $p_{\mathbf{Z}}$. Top-right: the density function of the target distribution $p_{\mathbf{Y}}(\mathbf{y})$. There exists a bijective function $\mathbf{f}$, such that $p_{\mathbf{Y}} = \mathbf{f}_* p_{\mathbf{Z}}$, with inverse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

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 $\mathbf{f}$ be an invertible function and $\mathbf{Y} = \mathbf{f}(\mathbf{Z})$. Then using the change of variables formula, one can compute the probability density function of the random variable $\mathbf{Y}$:

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

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

In the context of generative models, the above function $\mathbf{f}$ (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: $\mathbf{y} = \mathbf{f}(\mathbf{z})$.

The inverse function $\mathbf{g}$ moves 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 $\mathbf{g}$ 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 $\mathbf{f}$ can be arbitrarily complex, one can generate any distribution $p_{\mathbf{Y}}$ from any base distribution $p_{\mathbf{Z}}$.[1] This has been proven formally [Bogachev et al., 2005; Medvedev, 2008; Villani, 2003]. See Section 3.5.1.

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 $\mathbf{f}_1, \ldots, \mathbf{f}_N$ be a set of N bijective functions and define $\mathbf{f} = \mathbf{f}_N \circ \mathbf{f}_{N-1} \circ \cdots \circ \mathbf{f}_1$ to be the composition of the functions. Then it can be shown that $\mathbf{f}$ is also bijective, with inverse:

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

and the determinant of the Jacobian is

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

where $\mathrm{D}\mathbf{g}_i(\mathbf{y}) = \frac{\partial \mathbf{g}_i}{\partial \mathbf{x}}$ is the Jacobian of $\mathbf{g}_i$. We denote the value of the i-th intermediate flow as $\mathbf{x}_i = \mathbf{f}_i \circ \cdots \circ \mathbf{f}_1(\mathbf{z}) = \mathbf{g}_{i+1} \circ \cdots \circ \mathbf{g}_N(\mathbf{y})$ 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, $\mathbf{f}$ 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 $\mathbf{f}_*\mu$) by the formula

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

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 $\mathbf{f} : \mathcal{Z} \to \mathcal{Y}$, such that $\nu = \mathbf{f}_*\mu$. This function $\mathbf{f}$ can be interpreted as a "generator", and $\mathcal{Z}$ as a latent space. This view puts generative models in the context of transportation theory [Villani, 2003].

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 $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$ 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 $\mathbf{f}$ 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).

1. Subject to some reasonable conditions on the base density, such as that $p_{\mathbf{Z}}$ has non-zero support everywhere.

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 $\mathbf{f}$ 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 $\mathbf{f}$.

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, $\mathbf{f}$, 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

$$\begin{aligned} \log p(\mathcal{D}|\Theta) &= \sum_{i=1}^{M} \log p_{\mathbf{Y}}(\mathbf{y}_i|\Theta) \qquad (5) \\ &= \sum_{i=1}^{M} \log p_{\mathbf{Z}}(\mathbf{g}(\mathbf{y}_i|\theta)|\phi) + \log|\det \mathbf{Dg}(\mathbf{y}_i|\theta)| \end{aligned}$$

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 the inverse mapping, $\mathbf{g}$ (normalizing direction), as well as the 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 only requires evaluating the forward mapping of the flow $\mathbf{f}$ (generative direction). Thus sampling performance is largely determined by the cost of the forward mapping. 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.*, $\mathbf{g}$). [2]

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 [Grover et al., 2018]).

2. To ensure both efficient density estimation and sampling, van den Oord et al. [2017] proposed an approach called Probability Density Distillation which trains the flow $\mathbf{g}$ as normal and then uses this as a teacher network to train a tractable student network $\mathbf{f}$.

2.2.2 Variational Inference

Consider a latent variable model $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{y}) d\mathbf{y}$. Let $\mathbf{x}$ be 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 solution is to use variational inference and introduce the approximate posterior $q_\phi(\mathbf{y}|\mathbf{x})$ which should be as close to the real posterior as possible. This can be achieved by minimizing the KL divergence $D_{KL}(q_\phi(\mathbf{y}|\mathbf{x})||p(\mathbf{y}|\mathbf{x}))$, which is equivalent to maximizing the evidence lower bound $\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\mathbf{y}|\mathbf{x})}[\log(p(\mathbf{y}, \mathbf{x})) - \log(q_\phi(\mathbf{y}|\mathbf{x}))]$. The latter optimization can be done with gradient descent; however for that one needs to compute gradients of the form $\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{y}|\mathbf{x})}[h(\mathbf{y})]$, which is not straightforward.

As was observed in Rezende and Mohamed [2015], one can reparametrize $q_\phi(\mathbf{y}|\mathbf{x}) = p_{\mathbf{Y}}(\mathbf{y})$ with normalizing flows. Assume for simplicity, that only a single flow $\mathbf{f}$ with parameterization ϕ is used, $\mathbf{y} = \mathbf{f}(\mathbf{z})$ and the base distribution $p_{\mathbf{Z}}(\mathbf{z})$ does not depend on ϕ. Note that this is a generative direction. Then

$$\mathbb{E}_{p_{\mathbf{Y}}(\mathbf{y})}[h(\mathbf{y})] = \mathbb{E}_{p_{\mathbf{Z}}(\mathbf{z})}[h(\mathbf{f}(\mathbf{z}))], \qquad (6)$$

and the gradient of the right hand side with respect to ϕ can be computed.

In this scenario evaluating the likelihood is only required at points which have been sampled. Here the sampling performance and evaluation 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 [2015] 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 to know their inverse,
- be expressive enough to model real distributions,
- be computationally efficient: calculation of the Jacobian determinant, sampling from the base distribution, and application of the forward and (for sampling) inverse functions should be tractable.

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 figure 2.

3.1 Elementwise bijections

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

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

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

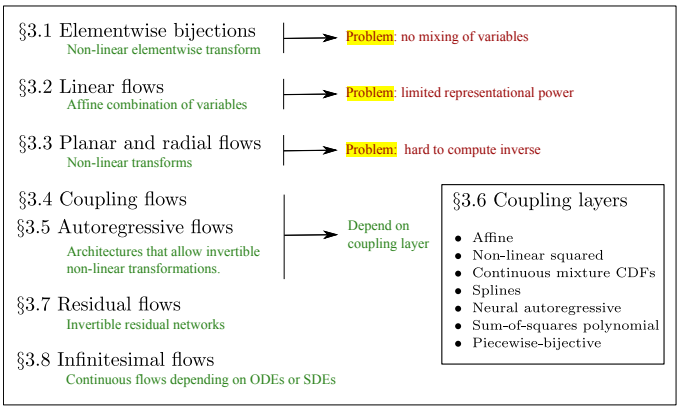

Fig. 2. Overview of flows discussed in this review. We start with elementwise bijections, linear flows, and planar and radial flows. All of these have drawbacks and are limited in utility. We then discuss two architectures (coupling flows and autoregressive flows) which support invertible non-linear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

portions of our parameter vector. In deep learning terminology, h, could be viewed as an "activation function". Note that the most commonly used activation function ReLU is not bijective and can not be directly applicable, however, the (Parametric) Leaky ReLU [He et al., 2015; Maas et al., 2013] can be used instead.

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:

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

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: $p_{\mathbf{Z}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, \mu, \Sigma)$. 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 3.6.1).

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 layers [Dinh et al., 2017] which have become a ubiquitous part of modern neural networks [Ioffe and Szegedy, 2015].

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 [2017] 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 [Dinh et al., 2017; Kingma and Dhariwal, 2018]. 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 [2016] 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 nonzero 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 [2018] proposed using the LU factorization:

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

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 $\mathbf{f}$ 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. [2019a] 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. A general convolution is easy to compute but it can be difficult to efficiently calculate the determinant or ensure invertibility. Kingma and Dhariwal [2018] restricted themselves to "1×1" convolutions for flows, and Zheng et al. [2018] used 1D convolutions (**ConvFlow**). Hoogeboom et al. [2019a] provided a more general solution for modelling $d \times d$ convolutions by stacking together autoregressive convolutions.

3.3 Planar and Radial Flows

Rezende and Mohamed [2015] 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

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

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

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

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 [Rezende and Mohamed, 2015].

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 [Kingma et al., 2016]. This bottleneck means that one needs to stack many planar flows to get high expressivity. Hasenclever et al. [2017] and van den Berg et al. [2018] introduced **Sylvester flows** to resolve this problem:

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

where $\mathbf{U}$ and $\mathbf{W}$ are $D \times M$ matrices, $\mathbf{b} \in \mathbb{R}^M$ and $h : \mathbb{R}^M \to \mathbb{R}^M$ 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:

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

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. [2017] and van den Berg et al. [2018].

3.3.2 Radial flows

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

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

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 [Rezende and Mohamed, 2015].

3.4 Coupling Flows

In this section and the following one 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 3.6 we give specific examples.

Dinh et al. [2015] introduced a coupling method to enable highly expressive transformations for flows (Figure 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 $\hat{\mathbf{f}}(\cdot\,; \theta) : \mathbb{R}^d \to \mathbb{R}^d$, parametrized by θ. Then one can define a function $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$ by the formula:

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

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 $\hat{\mathbf{f}}$ is called a *coupling layer*, and the resulting function $\mathbf{f}$ is called a *coupling flow*. A coupling flow is invertible if and only if $\hat{\mathbf{f}}$ is invertible and has inverse:

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

The Jacobian of $\mathbf{f}$ is a block triangular matrix where the diagonal blocks are $\mathrm{D}\hat{\mathbf{f}}$ and the identity matrix respectively. Hence the determinant of the Jacobian of the coupling flow is simply the determinant of $\mathrm{D}\hat{\mathbf{f}}$.

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

$$\hat{\mathbf{f}}(\mathbf{x}^A; \theta) = (\hat{\mathbf{f}}_1(x_1^A; \theta_1), \ldots, \hat{\mathbf{f}}_d(x_d^A; \theta_d)), \tag{17}$$

where each $\hat{\mathbf{f}}_i(\cdot; \theta_i) : \mathbb{R} \to \mathbb{R}$ is a scalar bijection. In this case a coupling flow is a triangular transformation (*i.e.*, has a triangular Jacobian matrix). See section 3.6 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 [2018] 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*" Dinh et al. [2017] which gradually introduces dimensions to the distribution in the generative direction (Figure 3b). In the normalizing direction, the layer 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 [Dinh et al., 2015], 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. [2017] used "masked" flows take alternating pixels or blocks of channels in the case of an image in non-volume preserving flows (**NVP**). In place of permutation Kingma and Dhariwal [2018] used 1×1 convolution (**Glow**). For the partition for the multi-scale flow in the normalizing direction, Das et al. [2019] suggested selecting features at which the Jacobian of the flow has higher values for the propagated part.

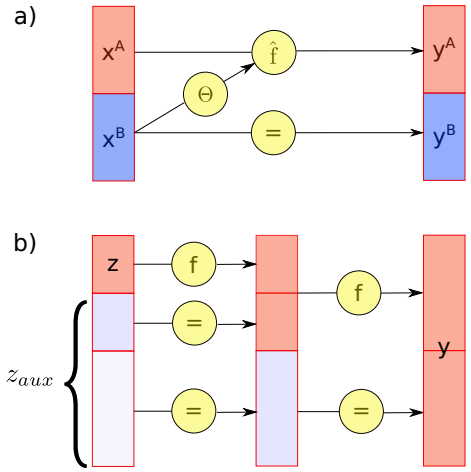

Fig. 3. Coupling architecture. a) A single layer of coupling flow described in Equation (15). A coupling layer $\hat{\mathbf{f}}$ is applied to one part of the space, while its parameters depend on the other part. b) Two subsequent layers of multi-scale flow in the generative direction. A flow is applied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

3.5 Autoregressive Flows

Kingma et al. [2016] 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 $\hat{\mathbf{f}}(\cdot;\theta) : \mathbb{R} \to \mathbb{R}$ be a bijection parameterized by θ. Then an autoregressive model is a function $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$, which outputs each entry of $\mathbf{y} = \mathbf{f}(\mathbf{x})$ conditioned on the previous entries of the input:

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

where $\mathbf{x}_{1:t}$ denotes the tuple $(x_1, \dots, x_t)$. For each $t = 2, \dots, D$ 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 $\mathbf{f}$ 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:

$$\det(\mathbf{Df}) = \prod_{t=1}^{D} \frac{\partial \hat{\mathbf{f}}}{\partial x_t}. \tag{19}$$

In practice, it possible to compute all the entries of the direct flow (Equation (18)) in one pass using a single network with appropriate masks []Germain2015. This idea was used by Papamakarios et al. [2017] to create masked autoregressive flows (**MAF**).

However, the computation of the inverse is more challenging. Given the inverse of $\hat{\mathbf{f}}$, the inverse of $\mathbf{f}$ can be found with recursion: we have $x_1 = \hat{\mathbf{f}}^{-1}(y_1; \Theta_1)$ and for any $t = 2, \dots, D$, $x_t = \hat{\mathbf{f}}^{-1}(y_t; \Theta_t(\mathbf{x}_{1:t-1}))$. 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 $\hat{\mathbf{f}}$ 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 layer* in both cases.

Note that Huang et al. [2018] used the name "transformer" (which has nothing to do with transformers in NLP).

Alternatively, Kingma et al. [2016] 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,

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

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 (Figure 4).

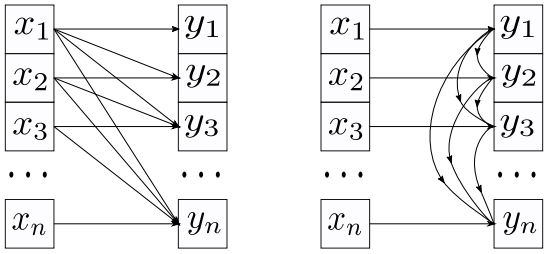

Fig. 4. Autoregressive flows. On the left, is the direct autoregressive flow given in Equation (18). Each output depends on the current and previous inputs and so this operation can be easily parallelised. On the right, is the inverse autoregressive flow from Equation (20). Each output depends on the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

In Section 2.2.1 we noted that papers typically model flows in the "normalizing" direction (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: from base density to data. Hence Papamakarios et al. [2017] 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. However, the two methods are theoretically equivalent and can learn the same distribution [Papamakarios et al., 2017].

3.5.1 Universality

For several autoregressive flows the universality property has been proven [Huang et al., 2018; Jaini et al., 2019b]. 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 Jaini et al. [2019b]. This section requires some knowledge of measure theory and functional analysis and can be safely skipped.

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

Proposition 4 ([Bogachev et al., 2005], 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 Huang et al. [2018], Lemma 4. The result follows from the dominated convergence theorem. ■

As a corollary, to claim that a class of autoregressive flows $\mathbf{f}(\cdot, \theta) : \mathbb{R}^D \to \mathbb{R}^D$ is universal, it is enough to demonstrate that a family of coupling layers $\hat{\mathbf{f}}$ used in the class is dense in the set of all monotone functions in the pointwise convergence topology. In particular, Huang et al. [2018] used neural monotone networks for coupling layers, and Jaini et al. [2019b] used monotone polynomials. Using the theory outlined in this section, universality could also be proved for spline flows [Durkan et al., 2019a,b] with splines for coupling layers (see Section 3.6.4).

3.6 Coupling Layers

As described in the previous sections, coupling flows and autoregressive flows have a similar functional form and both have coupling layers as building blocks. A coupling layer is just a univariate bijective differentiable function $\hat{\mathbf{f}}(\cdot, \theta) : \mathbb{R} \to \mathbb{R}$, parameterized by θ. Note that such a function is necessarily (strictly) monotone. In this section we describe the coupling layers used in the literature.

3.6.1 Affine coupling

Two simple forms of coupling layers $\hat{\mathbf{f}} : \mathbb{R} \to \mathbb{R}$ were proposed by [Dinh et al., 2015] in **NICE** (nonlinear independent component estimation). These were the *additive coupling layer*:

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

and the *affine coupling layer*:

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

Affine coupling layers are used for coupling flows in NICE [Dinh et al., 2015], RealNVP [Dinh et al., 2017], Glow [Kingma and Dhariwal, 2018] and for autoregressive architectures in IAF [Kingma et al., 2016] and MAF [Papamakarios et al., 2017]. They are simple and computation is efficient. However, they are limited in expressiveness and many layers must be stacked to represent complicated distributions.

3.6.2 Nonlinear squared flow

Ziegler and Rush [2019] proposed an invertible non-linear squared transformation defined by:

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

Under some constraints on parameters $\theta = [a, b, c, d, h] \in \mathbb{R}^5$, the coupling layer 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 layers facilitate learning multimodal distributions.

3.6.3 Continuous mixture CDFs

Ho et al. [2019] proposed the **Flow++** model, which contained several improvements, including a more expressive coupling layer. The layer is almost like a linear transformation, but one also applies a monotone function to x:

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

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 $\hat{\mathbf{f}}$. Computation of the inverse of the coupling function 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 layer to a logistic mixture improved performance slightly.

3.6.4 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 layer, 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.

3.6.4.1 Piecewise-linear and piecewise-quadratic: Müller et al. [2018] used linear splines for coupling layers $\hat{\mathbf{f}} : [0, 1] \to [0, 1]$. They divided the domain into K equal bins. Instead of defining increasing values for y_i, they used the integral of a positive piecewise-constant function:

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

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: $\frac{\partial \hat{\mathbf{f}}}{\partial x} = \theta_b K$.

Müller et al. [2018] also used a monotone quadratic spline on the unit interval for a coupling layer 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.

3.6.4.2 Cubic Splines: Durkan et al. [2019a] proposed using monotone cubic splines for a coupling layer. They do not restrict the domain to the unit interval, but instead use the form: $\hat{\mathbf{f}}(\cdot; \theta) = \sigma^{-1} \circ \hat{\mathbf{h}}(\cdot; \theta) \circ \sigma$, where $\hat{\mathbf{h}}(\cdot; \theta) : [0, 1] \to [0, 1]$ 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 $\hat{\mathbf{h}}'(0)$ and $\hat{\mathbf{h}}'(1)$. 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. Since Steffen's method is differentiable, the flow can be trained by gradient descent. However, Durkan et al. [2019b], noted numerical difficulties when the sigmoid saturates for values far from zero.

3.6.4.3 Rational quadratic splines: Durkan et al. [2019b] model a coupling layer $\hat{\mathbf{f}}(x\,;\theta)$ as a monotone rational-quadratic spline on the interval $[-B, B]$, and outside of the interval as the identity function. They define the spline using the method of Gregory and Delbourgo, by specifing $K+1$ knots, where boundary points are $(x_0, y_0) = (-B, -B)$ and $(x_K, y_K) = (B, B)$), and also specify derivatives at the inner points: $\{\hat{\mathbf{f}}'(x_i)\}_{i=1}^{K-1}$. These parameters 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 et al. [2019b] used this coupling layer with both a coupling architecture **RQ-NSF(C)** and an autoregressive architecture **RQ-NSF(AR)**.

3.6.5 Neural autoregressive flow

Huang et al. [2018] introduced Neural Autoregressive Flows (**NAF**) where a coupling layer $\hat{\mathbf{f}}(\cdot\,;\theta)$ 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 layer (**NAF-DSF**) and deep dense sigmoidal coupling layer (**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 $\hat{\mathbf{f}}(\cdot\,;\theta)$ is a strictly monotonic function. They also proved that a DSF network can approximate any strictly monotonic univariate function and so NAF-DSF is a universal flow.

Wehenkel and Louppe [2019] 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 simple: 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.

3.6.6 Sum-of-Squares polynomial flow

Jaini et al. [2019b] modeled $\hat{\mathbf{f}}(\cdot\,;\theta)$ 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 layer, one needs to integrate the sum of squares:

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

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 layer and so it is a generalization of the basic affine flow.

3.6.7 Piecewise-bijective coupling

Dinh et al. [2019] explore the idea that a coupling layer does not need to be bijective, but just piecewise-bijective (figure 5). Formally, they consider a function $\hat{\mathbf{f}}(\cdot\,;\theta) : \mathbb{R} \to \mathbb{R}$ 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 $\hat{\mathbf{f}}(\cdot\,;\theta)|_{A_i}$ is injective.

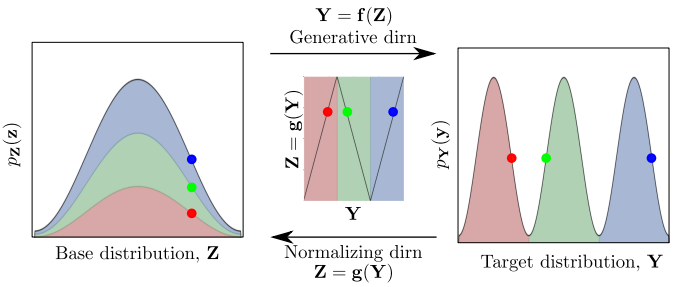

Fig. 5. Piecewise bijective coupling. The target domain (right) is divided into disjoint sections (colors) and each mapped by a monotone function (center) to the base distribution (left). For inverting the function, one samples a component of the base distribution using a gating network.

Dinh et al. [2019] constructed a flow $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$ with a coupling architecture and piecewise-bijective coupling layer 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, Dinh et al. [2019] 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 $y \mapsto (\hat{\mathbf{f}}(y), \phi(y))$, 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 $\hat{\mathbf{f}}$. 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:

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

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

3.7 Residual Flows

Residual networks [He et al., 2016] are compositions of the function of the form

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

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** [Gomez et al., 2017] and **iRevNets** [Jacobsen et al., 2018]. Their main motivation was to save memory during training and to stabilize computation. The central idea is a variation of additive coupling layers: 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 3.8 for more details). Chang et al. [2018, 2019] 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 $\mathbf{f}(\cdot)$ invertible. A sufficient condition for the invertibility was found in [Behrmann et al., 2019]. 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. [2019], 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 $\mathbf{f}$ in Equation (29) is: $\mathrm{D}\mathbf{f} = I + \mathrm{D}F$. 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:

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

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 $\mathbf{f}$. To efficiently compute each member of the truncated series, the Hutchinson trick was used. This provides stochastic estimation of the trace 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. [2019] in **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.

3.8 Infinitesimal (Continuous) Flows

The residual connections discussed in the previous section can be viewed as discretizations of a first order ordinary differential equation (ODE) [E, 2017; Haber et al., 2018]:

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

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.

3.8.1 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 [Arnold, 1978]). 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 [Katok and Hasselblatt, 1995].

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 $\mathbf{y} = \mathbf{f}(\mathbf{z}) = \Phi^1(\mathbf{z})$ was presented by [Chen et al., 2018b] under the name **Neural ODE (NODE)**. From a deep learning perspective this can be seen as an "infinitely deep" neural network with the input layer $\mathbf{z}$, the output layer

$\mathbf{y}$ 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:

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

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. [2019] 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.

An interesting side-effect of using continuous ODE-type flows is that one needs fewer parameters to achieve the same performance. For example, Grathwohl et al. [2019] show that for the comparable performance on CIFAR10, FFJORD uses less than 2% as many parameters as Glow.

Not all diffeomorphisms can be presented as a time one map of an ODE (see [Arango and Gómez, 2002; Katok and Hasselblatt, 1995]). 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. [2019] 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 $\mathbf{a}(t) \in \mathbb{R}^p$ and considered a new ODE:

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

with initial conditions $\mathbf{x}(0) = \mathbf{z}$ and $\mathbf{a}(0) = 0$. The addition of $\mathbf{a}(t)$ 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. [2019] 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. [2018] 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.

3.8.2 *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. [2018a]; Jankowiak and Obermeyer [2018]; Rezende and Mohamed [2015]; Salimans et al. [2015]; Sohl-Dickstein et al. [2015]; Suykens et al. [1998]; Welling and Teh [2011]). 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 [Oksendal, 1992].

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 [Oksendal, 1992]. The forward evolution is given by Fokker-Plank equation or Kolmogorov's forward equation:

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

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 [Suykens et al., 1998]. 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 Chen et al. [2018b]). It is also equivalent to the form of the transport equation considered in Jankowiak and Obermeyer [2018] for stochastic optimization.

Sohl-Dickstein et al. [2015] and Salimans et al. [2015] 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 [Chen et al., 2018a; Liutkus et al., 2019; Peluchetti and Favaro, 2019; Tzen and Raginsky, 2019]. Chen et al. [2018a] 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 [Peluchetti and Favaro, 2019; Tzen and Raginsky, 2019]. 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 [2019] proposed to use Kunita's theory of stochastic flows.

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.

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 Papamakarios et al. [2017] (Table 2)[3]. 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 [Dua and Graff, 2017].

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

TABLE 1
List of Normalizing Flows for which we show performance results.

Architecture	Coupling layer	Flow name
Coupling, 3.4	affine, 3.6.1	realNVP Glow
	mixture CDF, 3.6.3	Flow++
	splines, 3.6.4	quadratic (C) cubic RQ-NSF(C)
	piecewise-bijective, 3.6.7	RAD
Autoregressive, 3.5	affine	MAF
	polynomial, 3.6.6	SOS
	neural network, 3.6.5	NAF UMNN
	splines	quadratic (AR) RQ-NSF(AR)
Residual, 3.7		iResNet
		Residual flow
ODE, 3.8.1		FFJORD

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 [Martin et al., 2001] which contains segmentations of natural images. Papamakarios et al. [2017] 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 and MAF [Papamakarios et al., 2017], Glow and FFJORD [Grathwohl et al., 2019], NAF [Huang et al., 2018], UMNN [Wehenkel and Louppe, 2019], SOS [Jaini et al., 2019b], Quadratic Spline flow and RQ-NSF [Durkan et al., 2019b], Cubic Spline Flow [Durkan et al., 2019a].

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 Dinh et al. [2017] 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 [Dinh et al., 2017], Glow for CIFAR-10 and ImageNet [Kingma and Dhariwal, 2018], realNVP and Glow for MNIST, MAF and FFJORD [Grathwohl et al., 2019], SOS [Jaini et al., 2019b], RQ-NSF [Durkan et al., 2019b], UMNN [Wehenkel and Louppe, 2019], iResNet [Behrmann et al., 2019], Residual Flow [Chen et al., 2019], Flow++ [Ho et al., 2019].

Flow++ produces the best results. Besides using expressive coupling layers (Section 3.6.3), Ho et al. [2019] changed the architecture of the conditioner to self-attention and used variational dequantization instead of uniform. The ablation study in the paper shows that the latter modification gave the most significant improvement. It would be interesting to test other flow models with variational dequantization.

TABLE 2
Tabular datasets: data dimensionality and number of training examples for MNIST, CIFAR-10, ImageNet32 and ImageNet64 datasets.

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

TABLE 3
Average test log-likelihood (in nats) for density estimation on tabular datasets (higher the better). A number in parenthesis next to a flow indicates number of layers. MAF MoG is MAF with mixture of Gaussians as a base density.

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

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	$\approx$ 1.3M	$\approx$ 1.3M

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**

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 be: 1) less complex and 2) 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 [Jaini et al., 2019a] in the context of modelling tail behaviour.

5.1.2 Form of diffeomorphisms

The majority of the flows explored are triangular flows (either coupling or autoregressive architecture). 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? A related question concerns the best way to model conditional normalizing flows when one needs to learn a conditional probability distribution. Trippe and Turner [2017] 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 et al. [2019] 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 $p(z_t|z_{<t})$ [Kumar et al., 2019].

5.1.3 Loss function

The majority of the existing flows are trained by minimization of KL-divergence between source and the target distri-

butions (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 [Villani, 2003]. Interesting work has been done in this direction including Flow-GAN [Grover et al., 2018] and the minimization of the Wasserstein distance as suggested by [Arjovsky et al., 2017; Tolstikhin et al., 2018].

5.2 Generalisation to non-Euclidean spaces

5.2.1 *Flows on manifolds.*

[Falorsi et al., 2019] explored warping probability distributions on a manifold, where the analog of the Gaussian reparameterization trick was given for a Lie group. In particular, for a D-dimensional Lie group G, one considers its Lie algebra $\mathfrak{g}$ and chooses an isomorphism $\mathfrak{g} \cong \mathbb{R}^D$. Then for a base distribution with the density $p_{\mathbf{Z}}$ on $\mathbb{R}^D$, one can push it forward on G via the exponential map. Then, the analog of shifting by an element $g \in G$ is by left multiplication. Additionally applying a normalizing flow to a base measure before pushing it to G helps to construct multimodal distributions on G. Another approach was proposed in [Ovinnikov, 2018], where the Gaussian reparameterization trick in a hyperbolic space was investigated. How best to define a normalizing flow on a differentiable manifold remains an open question.

5.2.2 *Discrete distributions*

Discrete latent variables were used in Dinh et al. [2019] as an auxiliary tool to pushforward continuous random variables along piecewise-bijective maps (see Section 3.6.7). However, can we define normalizing flows if one or both of our distributions are discrete? This could be useful for many applications including NLP, graph generation and others.

To this end Tran et al. [2019] model bijective functions on a finite set and show that, in this case, the change of variables is given by the formula: $p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{Z}}(\mathbf{f}^{-1}\mathbf{y})$, *i.e.*, with no Jacobian term (compare with Definition 1). For backpropagation of functions with discrete variables they use the straight-through gradient estimator [Bengio et al., 2013]. However this method is not scalable to distributions with large numbers of elements.

Alternatively Hoogeboom et al. [2019b] 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 [Wang and Wang, 2019; Ziegler and Rush, 2019].

A different approach is de-quantization, (*i.e.*, adding noise to discrete data to make it continuous) which can be used on for ordinal variables, *e.g.*, discretized intensities. The noise can be uniform but other forms are possible and this dequantization can even be learned as part of a variational model [Ho et al., 2019]. 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.

Acknowledgments

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

References

Jaime Arango and Adriana Gómez. Diffeomorphisms as time one maps. *Aequationes Math.*, 64:304–314, 2002.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. In *ICML*, 2017.

Vladimir Arnold. *Ordinary Differential Equations*. The MIT Press, 1978.

Andrei Atanov, Alexandra Volokhova, Arsenii Ashukha, Ivan Sosnovik, and Dmitry Vetrov. Semi-Conditional Normalizing Flows for Semi-Supervised Learning. In *Workshop on Invertible Neural Nets and Normalizing Flows, ICML*, 2019.

Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint, arXiv:1308.3432*, 2013.

V.I. Bogachev, A.V. Kolesnikov, and K.V. Medvedev. Triangular transformations of measures. *Sbornik Math.*, 196(3-4):309–335, 2005.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. In *CoNLL*, 2015.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible Architectures for Arbitrarily Deep Residual Neural Networks. In *AAAI*, 2018.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *ICLR*, 2019.

Changyou Chen, Chunyuan Li, Liqun Chen, Wenlin Wang, Yunchen Pu, and Lawrence Carin. Continuous-Time Flows for Efficient Inference and Density Estimation. In *ICML*, 2018a.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018b.

Ricky T. Q. Chen, Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Residual Flows for Invertible Generative Modeling. *Advances in Neural Information Processing Systems*, 2019.

Antonia Creswell, Tom White, Vincent Dumoulin, Kailash Arulkumaran, Biswa Sengupta, and Anil Anthony Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35:53–65, 2018.

Hari Prasanna Das, Pieter Abbeel, and Costas J. Spanos. Dimensionality Reduction Flows. *arXiv preprint, arXiv:1908.01686*, 2019.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation. In *ICLR Workshop*, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density Estimation using Real NVP. In *ICLR*, 2017.

Laurent Dinh, Jascha Sohl-Dickstein, Razvan Pascanu, and Hugo Larochelle. A RAD approach to deep mixture models. In *ICLR Workshop*, 2019.

Dheeru Dua and Casey Graff. UCI Machine Learning

Repository, 2017.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs. *Advances in Neural Information Processing Systems*, 2019.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows. In *Workshop on Invertible Neural Networks and Normalizing Flows, ICML*, 2019a.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural Spline Flows. *Advances in Neural Information Processing Systems*, 2019b.

Weinan E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5: 1–11, 2017.

Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla-Romeu-Santos. Universal audio synthesizer control with normalizing flows. *arXiv preprint, arXiv:1907.00971*, 2019.

Luca Falorsi, Pim de Haan, Tim R. Davidson, and Patrick Forré. Reparameterizing Distributions on Lie Groups. *arXiv preprint, arXiv:1903.02958*, 2019.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The Reversible Residual Network: Backpropagation Without Storing Activations. *Advances in Neural Information Processing Systems*, 2017.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, 2014.

Will Grathwohl, Ricky T Q Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. In *ICLR*, 2019.

Aditya Grover, Manik Dhar, and Stefano Ermon. FlowGAN: Combining Maximum Likelihood and Adversarial Learning in Generative Models. In *AAAI*, 2018.

Eldad Haber, Lars Ruthotto, and Elliot Holtham. Learning across scales - a multiscale method for convolution neural networks. In *AAAI*, 2018.

Leonard Hasenclever, Jakub M Tomczak, Rianne Van Den Berg, and Max Welling. Variational Inference with Orthogonal Normalizing Flows. In *Workshop on Bayesian Deep Learning, NIPS*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019.

Emiel Hoogeboom, Rianne Van Den Berg, and Max Welling. Emerging Convolutions for Generative Normalizing Flows. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019a.

Emiel Hoogeboom, Jorn W.T. Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. In *NeurIPS*, 2019b.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural Autoregressive Flows. In *ICML*, 2018.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, 2015.

Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. i-RevNet: Deep Invertible Networks. In *ICLR*, 2018.

Priyank Jaini, Ivan Kobyzev, Marcus Brubaker, and Yaoliang Yu. Tails of Triangular Flows. *arXiv preprint, arXiv:1907.04481*, 2019a.

Priyank Jaini, Kira A. Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 5 2019b.

Martin Jankowiak and Fritz Obermeyer. Pathwise derivatives beyond the reparameterization trick. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

Anatole Katok and Boris Hasselblatt. *Introduction to the modern theory of dynamical systems*. Cambridge University Press, New York, 1995.

Sungwon Kim, Sang gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. FloWaveNet: A Generative Flow for Raw Audio. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2018.

Diederik P Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. Technical report, 2018.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations, ICLR*, 2014.

Diederik P. Kingma and Max Welling. An Introduction to Variational Autoencoders. *arXiv preprint, arXiv:1906.02691*, 2019.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved Variational Inference with Inverse Autoregressive Flow. In *NIPS*, 2016.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. Massachusetts: MIT Press, 2009.

Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. VideoFlow: A Flow-Based Generative Model for Video. In *Workshop on Invertible Neural Nets and Normalizing Flows, ICML*, 2019.

Antoine Liutkus, Umut Simsekli, Szymon Majewski, Alain Durmus, and Fabian-Robert Stöter. Sliced-Wasserstein Flows: Nonparametric Generative Modeling via Optimal Transport and Diffusions. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *ICML*, 2013.

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. GraphNVP: An Invertible Flow Model for Generating Molecular Graphs. *arXiv preprint, arXiv:1905.11600*, 2019.

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 *Proceedings of the 8th International*

Conference on Computer Vision, ICCV, 2001.

Kirill V. Medvedev. Certain properties of triangular transformations of measures. *Theory Stoch. Process.*, 14(30):95–99, 2008.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novak. Neural Importance Sampling. *ACM Transactions on Graphics (TOG)*, 38, 2018.

Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving exploration in soft-actor-critic with normalizing flows policies. In *Workshop on Invertible Neural Networks and Normalizing Flows, ICML*, 2019.

Bernt Oksendal. *Stochastic Differential Equations (3rd Ed.): An Introduction with Applications*. Springer-Verlag, Berlin, Heidelberg, 1992.

Ivan Ovinnikov. Poincaré Wasserstein Autoencoder. In *Bayesian Deep Learning Workshop, NeurIPS*, 2018.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation. In *NIPS*, 2017.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for Probabilistic Modeling and Inference. *arXiv preprint, arXiv:1912.02762*, 2019.

Stefano Peluchetti and Stefano Favaro. Neural Stochastic Differential Equations. *arXiv preprint, arXiv:1905.11065*, 2019.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP*, 2019.

Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *ICML*, 2015.

Oren Rippel and Ryan Prescott Adams. High-Dimensional Probability Estimation with Deep Density Models. Technical report, 2013.

Tim Salimans, Algoritmica Diederik, Diederik P Kingma, and Max Welling. Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. In *ICML*, 2015.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. In *NIPS*, 2016.

Hadi Salman, Payman Yadollahpour, Tom Fletcher, and Nematollah Batmanghelich. Deep diffeomorphic normalizing flows. *arXiv preprint, arXiv:1810.03256*, 2018.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015.

Johan Suykens, Herman Verrelst, and Joos Vandewalle. On-Line Learning Fokker-Planck Machine. *Neural Processing Letters*, 7:81–89, 1998.

Esteban G. Tabak and Cristina V. Turner. A Family of Nonparametric Density Estimation Algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Esteban G. Tabak and Eric Vanden-Eijnden. Density Estimation by Dual Ascent of the Log-Likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

Ilya O. Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schölkopf. Wasserstein Auto-Encoders. In *ICLR*, 2018.

Jakub Tomczak and Max Welling. Improving Variational Auto-Encoders using convex combination linear Inverse Autoregressive Flow. *Benelearn*, 2017.

Jakub M Tomczak and Max Welling. Improving Variational Auto-Encoders using Householder Flow. Technical report, 2016.

Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. Discrete Flows: Invertible Generative Models of Discrete Data. In *ICLR Workshop*, 2019.

Brian Loeber Trippe and Richard E. Turner. Conditional Density Estimation with Bayesian Normalising Flows. In *Workshop on Bayesian Deep Learning, NIPS*, 2017.

Belinda Tzen and Maxim Raginsky. Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit. *arXiv preprint, arXiv:1905.09883*, 2019.

Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence, UAI*, 2018.

Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. In *ICML*, 2017.

Cédric Villani. *Topics in optimal transportation (Graduate Studies in Mathematics 58)*. American Mathematical Society, Providence, RI, 2003.

Kunfeng Wang, Chao Gou, Yanjie Duan, Yilun Lin, Xinhu Zheng, and Fei yue Wang. Generative adversarial networks: introduction and outlook. *IEEE/CAA Journal of Automatica Sinica*, 4:588–598, 2017.

Prince Zizhuang Wang and William Yang Wang. Riemannian Normalizing Flow on Variational Wasserstein Autoencoder for Text Modeling. *arXiv preprint, arXiv:1904.02399*, 2019.

Antoine Wehenkel and Gilles Louppe. Unconstrained Monotonic Neural Networks. *arXiv preprint, arXiv:1908.05164*, 2019.

Max Welling and Yee Whye Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *ICML*, 2011.

Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation Capabilities of Neural Ordinary Differential Equations. *arXiv preprint, arXiv:1907.12998*, 2019.

Guoqing Zheng, Yiming Yang, and Jaime Carbonell. Convolutional Normalizing Flows. In *Workshop on Theoretical Foundations and Applications of Deep Generative Models, ICML*, 2018.

Zachary M. Ziegler and Alexander M. Rush. Latent Normalizing Flows for Discrete Sequences. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019.

Ivan Kobyzev Ivan Kobyzev received his Masters degree in Mathematical Physics from St Petersburg State University, Russia, in 2011 and his PhD in Mathematics from Western University, Canada, in 2016. He did two postdocs in Matematics and in Computer Science at the University of Waterloo. Currently he is a researcher in Borealis AI.

Simon Prince Simon Prince holds a Masters by Research from University College London and a doctorate from the University of Oxford. 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 Dr. Prince was a tenured faculty member in the Department of Computer Science at University College London, where he taught courses in Computer Vision, Image Processing and Advanced Statistical Methods. During this time, he was Director of the M.Sc. in Computer Vision, Graphics and Imaging. Dr. Prince worked in industry applying AI to computer graphics software. Currently he is a Research Director of Borealis AI's Montreal office.

Marcus Brubaker Marcus Brubaker received his PhD in 2011 at the University of Toronto. He also did postdocs at the Toyota Technological Institute at Chicago, Toronto Rehabilitation Hospital and the University of Toronto. His research interests span computer vision, machine learning and statistics. Dr. Brubaker is a member of the Centre for Vision Research and core member of the Vision: Science to Application (VISTA) program at York University. He is also currently serving as an Associate Editor for the journal IET Computer Vision, an Area Chair for ECCV 2018 and Student Volunteer Chair for CVPR 2018. Currently he is a Research Director of Borealis AIs Toronto office, Assistant Professor of Computer Science at York University and Adjunct Professor in the University of Toronto Department of Computer Science.