Normalizing Flows: Introduction and Ideas

Ivan Kobyzev IVAN.KOBYZEV@BOREALISAI.COM
Simon Prince SIMON.PRINCE@BOREALISAI.COM
Marcus A. Brubaker MARCUS.BRUBAKER@BOREALISAI.COM

"Be still like a mountain and flow like a great river."

Lao Tzu

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 a data distribution with a fixed family of distributions; these include traditional parameter estimation in statistics. Variational approaches and expectation maximization introduce latent variables which aim to explain observed variables and provide additional flexibility but can increase the complexity of learning and inference. Graphical models [Koller and Friedman, 2009] aim to explicitly model the conditional dependence between random variables. Recently, generative neural approaches have been proposed including generative adversarial networks (GAN) [Goodfellow et al., 2014] and variational auto-encoders (VAE)[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.[1] 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].

A Normalizing Flow (NF) is family of generative models which produces tractable distributions where both sampling and density evaluation can be efficient and exact. Successful applications include: image generation [Kingma and Dhariwal, 2018; Ho et al., 2019], video generation [Kumar et al., 2019], audio generation [Kim et al., 2018; Prenger et al., 2019; Esling et al., 2019], graph generation [Madhawa et al., 2019], and reinforcement learning [Nadeem Ward et al., 2019].

1. Approximations of the likelihood for GANs and VAEs are possible but can be difficult and computationally expensive.

There are several survey papers for VAEs [Kingma and Welling, 2019] and GANs [Creswell et al., 2018; Wang et al., 2017], but to the best of our knowledge, no equivalent survey articles for Normalizing Flows. This article aims to provide a coherent and comprehensive review of the literature around the construction and use of 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 current the state-of-the-art literature, and 3) identify open questions and promising future directions.

In Section 2, we introduce Normalizing Flows and describe how they are trained. In Section 3 we review existing constructions for Normalizing Flows. In Section 4 we describe datasets used for testing Normalizing Flows and illustrate the performance of existing methods. Finally, in Section 5 we discuss open problems and possible future 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 in a preprint 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 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 described 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 $\mathbf{Y}$:

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

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).

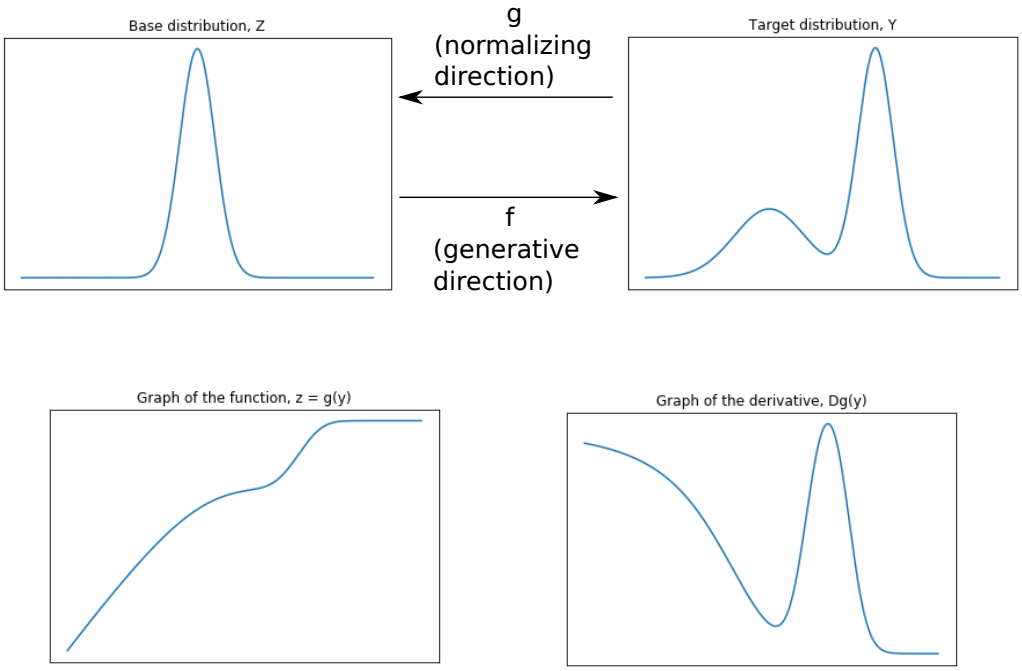

Figure 1: Illustration of the change of variables (Equation (1)). On the top left, is the density function of the source $p_{\mathbf{Z}}$. On the top right, is the density function of the target distribution $p_{\mathbf{Y}}(\mathbf{y})$. There exists a bijective function $\mathbf{f}$, such that $p_{\mathbf{Y}} = \mathbf{f}_* p_{\mathbf{Z}}$, and its inverse $\mathbf{g}$. On the bottom left, is the graph of the inverse function $\mathbf{g}$. On the bottom right, is the graph of the Jacobian of $\mathbf{g}$.

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 complex 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 complex and irregular data distribution towards a simpler, more regular or "normal" form, specifically 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 a base distribution $p_{\mathbf{Z}}$[2] as the pushforward $p_{\mathbf{Y}} = \mathbf{f}_* p_{\mathbf{Z}}$. This has been proven formally [Villani, 2003; Bogachev et al., 2005; Medvedev, 2008]. See Section 3.5.1 for more details.

2. There should be some reasonable condition on the base density, like $p_{\mathbf{Z}}$ has non-zero support everywhere.

Constructing arbitrarily complex 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 $\mathbf{f}_1, \dots, \mathbf{f}_N$ 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 given by

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

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 $\mathbf{g}_i$, $\mathbf{x}_i = \mathbf{f}_i \circ \cdots \circ \mathbf{f}_1(\mathbf{z}) = \mathbf{g}_{i+1} \circ \cdots \circ \mathbf{g}_N(\mathbf{y})$ is the value of the i-th intermediate flow and $\mathbf{x}_N = \mathbf{y}$. Thus, given a set of nonlinear bijective functions, the above result allows them to be composed to construct successively more complex 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 it and move 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*

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

This notion gives a general formulation of a generative modelling. 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].

For the rest of 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.*

It is well known that the pushforward of an absolutely continuous measure $p_{\mathbf{Z}}\mathbf{dz}$ by a diffeomorphism $\mathbf{f}$ is also absolutely continuous and its density function is calculated by Equation (1). Note that this more general approach is important if one wants to study generative models on non-Euclidean spaces (see Section 5.2 for discussion).

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 continuous 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 complex distribution, $\mathcal{D} = \{\mathbf{y}_i\}_{i=1}^M$, we can then perform likelihood-based estimation of the parameters $\Theta = (\theta, \phi)$. The data likelihood in this case simply becomes

$$\begin{aligned} \log p(\mathcal{D}|\Theta) &= \sum_{i=1}^{M} \log p_{\mathbf{Y}}(\mathbf{y}_i|\Theta) \\ &= \sum_{i=1}^{M} \log p_{\mathbf{Z}}(\mathbf{g}(\mathbf{y}_i|\theta)|\phi) + \log|\det \mathrm{D}\mathbf{g}(\mathbf{y}_i|\theta)| \end{aligned} \tag{4}$$

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 normalizing flow requires computing the inverse mapping, $\mathbf{g}$ (normalizing direction), as well as the log determinant. This occurs at test time, but will occur repeatedly during likelihood-based training. Thus, efficiency of the inverse mapping and log determinant will determine the efficiency of the training.

However, sampling from the distribution defined by the normalizing flow requires only 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.*, $\mathbf{g}$).

If one wants to have both efficient density estimation and sampling, van den Oord et al. [2017] proposed an approach called Probability Density Distillation which trains the flow $\mathbf{g}$ with log likelihood maximization, and then fixes it and uses it as a teacher network. The teacher network is used to train a student network $\mathbf{f}$ by minimizing the divergence between the distribution of sampled points and the modeled data distribution by the teacher network (Figure 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]).

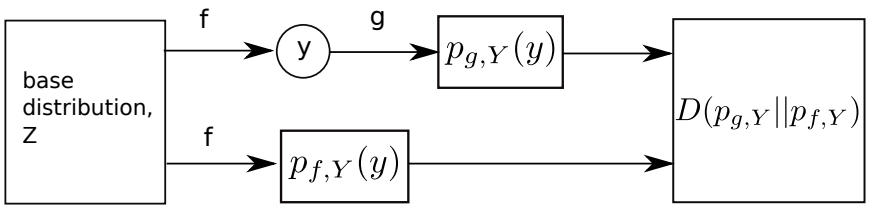

Figure 2: A schematic diagram of a Probability Density Distillation method. The teacher network g (flow in normalizing direction) is trained to produce the likelihood for a data point. A student network f (flow in generative direction) is able to efficiently generate samples with their likelihood value. One trains the student network to minimize the divergence between distributions of samples and the data.

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 $\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\mathbf{y}|\mathbf{x})}[\log(p(\mathbf{y}, \mathbf{x}) - q_\phi(\mathbf{y}|\mathbf{x})]$. 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

$$\mathbb{E}_{p_{\mathbf{Y}}(\mathbf{y})}[h(\mathbf{y})] = \mathbb{E}_{p_{\mathbf{Z}}(\mathbf{z})}[h(\mathbf{f}(\mathbf{z}))], \tag{5}$$

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

As described in the previous section, Normalizing Flows should satisfy several conditions in order to be practical:

- They should be invertible; for sampling we need to know their inverse.
- They should be expressive enough to model real data distributions.

- They should 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 this section we will describe different flows that have been proposed in the literature.

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

$$\mathbf{f}(\mathbf{x}) = (h(x_1), h(x_2), \ldots, h(x_D))^T \tag{6}$$

is also a bijection whose inverse simply requires computing h^{-1} and whose Jacobian is the product of the derivatives of h. This can be generalized by allowing each element to have its own distinct bijective function; this might be useful, for instance, if we wish to only modify portions of our parameter vector. In deep learning terminology, h, could be viewed 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 [Maas et al., 2013; He et al., 2015] can be used instead.

Elementwise operations on their own are inherently insufficient as they cannot express any form of correlation between dimensions; more bijections are necessary if dependencies between dimensions are to be captured.

3.2 Linear Flows

Linear mappings take the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{7}$$

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 on their own are not very expressive. For example, consider a Gaussian base distribution: $p_{\mathbf{Z}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, \mu, \Sigma)$. After transformation by a linear flow, the distribution will remain Gaussian: if $\mathbf{y} = \mathbf{f}(\mathbf{z})$, then its distribution will be $p_{\mathbf{Y}} = \mathcal{N}(\mathbf{y}, \mathbf{A}\mu + \mathbf{b}, \mathbf{A}^T\Sigma\mathbf{A})$. More generally, a linear flow of a distribution from the exponential family will stay 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 directly in $\mathcal{O}(D^3)$. The inverse map can also be computed in $\mathcal{O}(D^3)$. Hence, using general form of the linear flow 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. For example, if $\mathbf{A}$ is diagonal with nonzero diagonal entries, then its inverse can be computed in linear time and its determinant is simply the product of the diagonal entries. However, such a transformation is extremely limited in expressivity; it is an elementwise transformation. Nonetheless, this can still be useful for representing normalization layers [Dinh et al., 2017] which have become a ubiquitous part of modern CNNs [Ioffe and Szegedy, 2015].

3.2.1 Triangular

A somewhat more expressive form of linear transformation is the triangular matrix. Its determinant is the product of its diagonal and 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 [Stewart, 1998] versus $\mathcal{O}(D^3)$ for a general matrix. This can be viewed as a linear analog of autoregressive models considered in Section 3.5.

A slight generalization was done in Tomczak and Welling [2017]. Consider K triangular matrices $\mathbf{T}_i$ with ones on the diagonal and a K-dimensional probability vector ω. Then one can define a linear transformation $\mathbf{y} = (\sum_{i=1}^{K} \omega_i \mathbf{T}_i)\mathbf{z}$. 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.

3.2.2 Permutation and Orthogonal

The expressiveness of triangular transformations is sensitive to the ordering of dimensions. Reordering the dimensions can be done easily and 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, in these cases the permutations are set initially and remain fixed during training which might not be optimal for a given problem.

An alternative to permutations is the use of orthogonal transformations which is a generalization of the permutation operation because the set of permutation matrices is a subset of the set of orthogonal matrices. The inverse and absolute determinant of an orthogonal matrices are both trivial which make them efficient to use. Tomczak and Welling [2016] used orthogonal matrices parameterized by the *Householder transform*. The idea is based on the observation that any orthogonal matrix can be written as a product of reflections. To parameterize a reflection matrix H in $\mathbb{R}^D$ one just needs to fix a nonzero vector $\mathbf{v} \in \mathbb{R}^D$, and then define $H = \mathbb{1} - \frac{2}{||\mathbf{v}||^2}\mathbf{v}\mathbf{v}^T$. To model an orthogonal matrix one can consider a linear flow of reflections.[3]

3.2.3 Factorizations

Instead of limiting the form of $\mathbf{A}$, Kingma and Dhariwal [2018] proposed to parameterize it using the LU factorization. Specifically,

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

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, one issue is that $\mathbf{P}$ is a discrete permutation and cannot be easily optimized as a parameter of the model. To avoid this, $\mathbf{P}$ is randomly generated initially and then fixed. Hoogeboom et al. [2019a]

3. Another natural choice of parameterization is to use the tangent space of $SO(n)$ (*i.e.*, the set of skew-symmetric matrices) to parameterize $SO(n)$ and the exponential map as the transformation. This has yet to be explored in the literature but is an interesting alternative as it is directly differentiable and may have a more natural geometry for optimization.

noted that fixing the permutation matrix limits the flexibility of the transformation, and proposed using the QR decomposition instead. They proposed modeling the orthogonal matrix for that decomposition with Householder transforms (Section 3.2.2)

3.2.4 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 we review them here for completeness.

3.3.1 Planar flows

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

$$\mathbf{f}(\mathbf{x}) = \mathbf{x} + \mathbf{u}h(\mathbf{w}^T\mathbf{x} + b), \tag{9}$$

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

$$\det\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right) = \det(\mathbb{1}_D + \mathbf{u}h'(\mathbf{w}^T\mathbf{x} + b)\mathbf{w}^T) = 1 + h'(\mathbf{w}^T\mathbf{x} + b)\mathbf{u}^T\mathbf{w}, \tag{10}$$

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. This is explored in more detail in the Appendix of Rezende and Mohamed [2015].

As was speculated in Kingma et al. [2016], 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. 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. Here, the transformation is given by the similar formula as before:

$$\mathbf{f}(\mathbf{x}) = \mathbf{x} + \mathbf{U}h(\mathbf{W}^T\mathbf{x} + \mathbf{b}), \tag{11}$$

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:

$$\det(\mathrm{D}\mathbf{f}) = \det(\mathbb{1}_D + \mathbf{u}\,\mathrm{diag}(h'(\mathbf{w}^T\mathbf{x} + b))\mathbf{w}^T) = \det(\mathbb{1}_M + \mathrm{diag}(h'(\mathbf{w}^T\mathbf{x} + b))\mathbf{w}\mathbf{u}^T), \tag{12}$$

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

$$\mathbf{f}(\mathbf{x}) = \mathbf{x} + \frac{\beta}{\alpha + \|\mathbf{x} - \mathbf{x}_0\|}(\mathbf{x} - \mathbf{x}_0) \tag{13}$$

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, however, it exists under suitable constraints on the parameters. Details can be found in Rezende and Mohamed [2015].

In the next two sections we will describe coupling and auto-regresssive flows. These are currently the two most widely used flow architectures. First, we will present them in the general form, and then in Section 3.6 we will give specific examples.

3.4 Coupling Flows

Dinh et al. [2015] introduced a coupling method to enable highly expressive transformations for flows. 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:

$$\begin{aligned} \mathbf{y}^A &= \hat{\mathbf{f}}(\mathbf{x}^A; \Theta(\mathbf{x}^B)) \\ \mathbf{y}^B &= \mathbf{x}^B, \end{aligned} \tag{14}$$

where $\Theta(\mathbf{x}^B)$ is *any* arbitrary function which only uses $\mathbf{x}^B$ as input, which is called a *conditioner*. A bijection $\hat{\mathbf{f}}$ is called a *coupling layer*, and the resulting function $\mathbf{f}$ is called a *coupling flow*. The coupling flow is invertible if and only if $\hat{\mathbf{f}}$ is invertible and its inverse is given by:

$$\begin{aligned} \mathbf{x}^A &= \hat{\mathbf{f}}^{-1}(\mathbf{y}^A; \Theta(\mathbf{x}^B)) \\ \mathbf{x}^B &= \mathbf{y}^B. \end{aligned} \tag{15}$$

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 $\mathrm{D}\hat{\mathbf{f}}$.

The majority of coupling layers used in the literature are in the decomposed form (*i.e.*, they apply to $\mathbf{x}^A$ element-wise):

$$\hat{\mathbf{f}}(\mathbf{x}^A; \theta) = (\hat{\mathbf{f}}_1(x_1^A; \theta_1), \ldots, \hat{\mathbf{f}}_d(x_d^A; \theta_d)), \tag{16}$$

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). We give a list of coupling layers in Section 3.6.

The power of a coupling flow resides, largely, in the ability of $\Theta(\mathbf{x}^B)$ to be arbitrarily complex. In practice it is usually modelled as a neural network, for instance, Kingma and Dhariwal [2018] used a shallow ResNet [He et al., 2016] architecture. Alternatively, the conditioner can be constant (*i.e.*, not depend on $\mathbf{x}^B$ at all). This provides a form of factorization which allows for the construction of a "*multi-scale flow*" which gradually introduces dimensions to the distribution in the generative direction. In the normalizing direction, the dimension of the layer reduces by half after each iteration step, such that the final output contains the most of semantic information about the input. This was proposed by Dinh et al. [2017] to reduce the computational costs of transforming high dimensional distributions as well as to capture the multi-scale structure inherent in natural images. The coupling architecture and its use for a multi-scale structure is demonstrated in Figure 3.

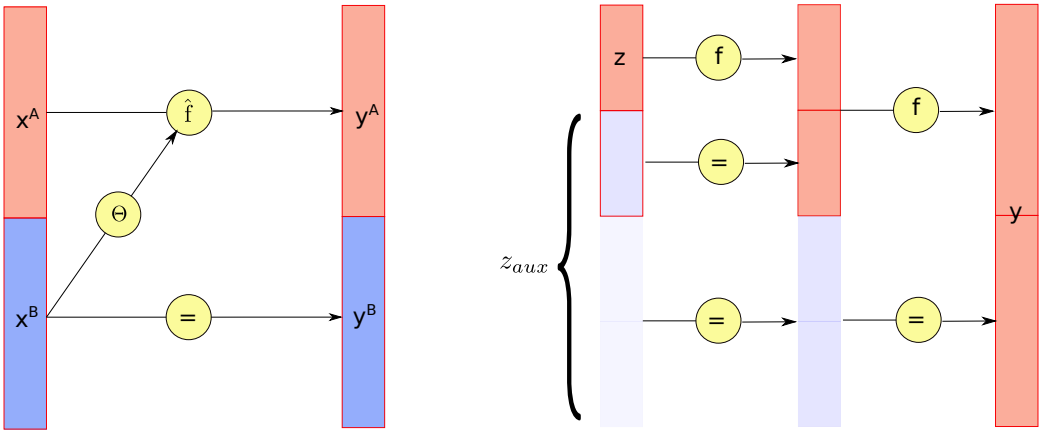

Figure 3: Coupling architecture. On the left, is a single layer of coupling flow described in Equation (14). A coupling layer $\hat{\mathbf{f}}$ is applied to one part of the space, while its parameters depend on the other part. On the right, is two subsequent layers of multi-scale flow in the generative 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.

A 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.

3.5 Autoregressive Flows

Kingma et al. [2016] used autoregressive models as a form of normalizing flow. One can consider these as nonlinear versions of multiplication by a triangular matrix (Section 3.2.1).

Consider an ordering on the basis of $\mathbb{R}^D$, so that the order of entries in the variable $\mathbf{x} = (x_1, \ldots, x_D)$ is fixed. Denote by $\mathbf{x}_{1:t}$ the tuple $(x_1, \ldots, x_t)$. 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 (with respect to the fixed ordering). Formally,

$$y_t = \hat{\mathbf{f}}(x_t; \Theta_t(\mathbf{x}_{1:t-1})), \tag{17}$$

where for each $t = 2, \ldots, D$ we choose arbitrary functions $\Theta_t(\cdot)$ mapping $\mathbb{R}^{t-1}$ to the set of all parameters, and Θ_1 is a constant. The functions $\Theta_t(\cdot)$ use $\mathbf{x}_{1:t-1}$ as input 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:

$$\det(\mathrm{D}\mathbf{f}) = \prod_{t=1}^{D} \frac{\partial \hat{\mathbf{f}}}{\partial x_t}. \tag{18}$$

Given the inverse of $\hat{\mathbf{f}}$, the inverse of $\mathbf{f}$ can be found with recursion. For that $x_1 = \hat{\mathbf{f}}^{-1}(y_1; \Theta_1)$ and for any $t = 2, \ldots, D$, $x_t = \hat{\mathbf{f}}^{-1}(y_t; \Theta_t(\mathbf{x}_{1:t-1}))$. However 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,

$$y_t = \hat{\mathbf{f}}(x_t; \theta_t(\mathbf{y}_{1:t-1})). \tag{19}$$

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 (17), 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).

Germain et al. [2015] (in **MADE**) suggested to use masks for efficient autoregressive density estimation. This idea was used by Papamakarios et al. [2017] for masked autoregressive flows (**MAF**): masking makes it possible to compute all the entries of the direct flow (Equation (17)) in one pass.

As mentioned before (Section 2.2.1), papers typically model flows in the "normalizing" direction (from data to the base density) to enable efficient evaluation of the log-likelihood for training with density estimation. In this context one can think of IAF as a flow in the generative direction: from base density to data. As noted in Papamakarios et al. [2017], one

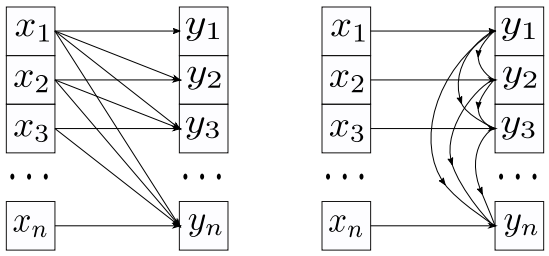

Figure 4: Autoregressive flows. On the left, is the direct autoregressive flow given in Equation (17). Each output depends on the the current and previous inputs. On the right, is the inverse autoregressive flow from Equation (19). Each output depends on the current input and the previous outputs. This cannot easily be parallelized.

should use IAF if fast sampling is needed (*e.g.*, for stochastic variational inference), and use MAF if fast density estimation is desirable. However, these two methods are theoretically equivalent, *i.e.*, they can learn the same distribution [Papamakarios et al., 2017]. For both efficient density estimation and sampling one can also use Probability Density Distillation [van den Oord et al., 2017] (Section 2.2.1).

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 the clean exposition of Jaini et al. [2019b]. This section requires some knowledge of measure theory and functional analysis and can be safely skipped without jeopardizing comprehension of the rest of the document.

First, recall that a mapping $T = (T_1, \ldots, T_D) : \mathbb{R}^D \to \mathbb{R}^D$ is called triangular if T_i is a function of $\mathbf{x}_{1:i}$ for each $i = 1, \ldots, D$. Such 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. 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, for details. Basically 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, the universality could be proved for spline flows [Durkan et al., 2019a,b] with splines for coupling layers (it was not done in the papers). See the next section for further details on each architecture.

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. Basically, 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.[4] 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): the *additive coupling layer*:

$$\hat{\mathbf{f}}(x\,; \theta) = x + \theta, \quad \theta \in \mathbb{R}, \tag{20}$$

and the *affine coupling layer*:

$$\hat{\mathbf{f}}(x; \theta) = \theta_1 x + \theta_2, \quad \theta_1 \neq 0,\ \theta_2 \in \mathbb{R}. \tag{21}$$

Affine coupling layers are used for coupling architectures 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]. It is simple and facilitates efficient computation, however it is limited in its expressiveness and many such layers must be stacked.

3.6.2 Nonlinear squared flow

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

$$\hat{\mathbf{f}}(x\,; \theta) = ax + b + \frac{c}{1 + (dx + h)^2}. \tag{22}$$

Under some constraints on the 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). Qualitative experiments demonstrated that such coupling layer facilitates learning multimodal distributions.

4. On the other hand, every continuous univariate function which is surjective and strictly monotone, must be bijective.

3.6.3 Expressive coupling with continuous mixture CDFs

Ho et al. [2019] proposed **Flow++** model, where among several improvements a more expressive coupling layer was suggested. The layer is almost like a linear transformation, but one also applies a monotone function to x. Formally,

$$\hat{\mathbf{f}}(x;\theta) = \theta_1 F(x,\theta_3) + \theta_2, \tag{23}$$

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

$$F(x, \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}) = \sigma^{-1}\left(\sum_{j=1}^{K} \pi_j \sigma\left(\frac{x-\mu_j}{s_j}\right)\right). \tag{24}$$

Note, that the postcomposition 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 this 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 logistic mixture gives slightly better performance.

3.6.4 Splines

A spline is a piecewise-polynomial or a piecewise-rational function. One can model a coupling layer as a monotone spline. To define a spline one specifies $K+1$ points $(x_i, y_i)_{i=0}^K$, called *knots*, through which the spline passes. To define a monotone spline one has a restriction: $x_i < x_{i+1}$ and $y_i < y_{i+1}$.

Usually splines are considered on a compact interval.

Piecewise-linear coupling Müller et al. [2018] introduced a linear spline for a coupling layer $\hat{\mathbf{f}} : [0,1] \to [0,1]$. They divided the domain into K bins of equal length. Instead of defining increasing values for y_i, the authors suggested to model a monotone piecewise-linear function as the integral of a positive piecewise-constant function. Formally:

$$\hat{\mathbf{f}}(x;\theta) = \alpha\theta_b + \sum_{k=1}^{b-1} \theta_k, \tag{25}$$

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

$$\frac{\partial \hat{\mathbf{f}}}{\partial x} = \theta_b K. \tag{26}$$

Piecewise-quadratic coupling Müller et al. [2018] also consider a monotone quadratic spline on the unit interval for a coupling layer. Analogously to the linear case, they propose to model it as an integral of a positive piecewise-linear function. A monotone quadratic spline is invertible; finding its inverse map requires solving a quadratic equation.

Cubic Splines Durkan et al. [2019a] proposed using monotone cubic splines for a coupling layer. Unlike [Müller et al., 2018], they do not restrict the domain to the unit interval, but they model a coupling layer in 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.

For construction of a monotone cubic spline the authors use Steffen's method: one needs to specify $K+1$ knots of the spline and also boundary derivatives $\hat{\mathbf{h}}'(0)$ and $\hat{\mathbf{h}}'(1)$. All of these quantities are modelled as the output of a neural network.

Computation of the derivative of a cubic spline is not hard as it is a piecewise-quadratic polynomial. To invert a cubic spline, one needs to find a root of a cubic polynomial[5], which can be done either analytically or numerically. However, the procedure is numerically unstable if not treated carefully. Because Steffen's method is differentiable, the flow can be trained by gradient descent. However, the problem with this model, as was observed in [Durkan et al., 2019b], is numerical difficulty with the sigmoid which saturates for input values far from zero.

Rational quadratic splines Durkan et al. [2019b] model a coupling layer $\hat{\mathbf{f}}(x\,;\theta)$ as a monotone rational-quadratic splines on the compact interval $[-B, B]$, and outside of the interval as the identity function. To define such a spline (with the method of Gregory and Delbourgo), one needs to specify $K+1$ 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: $\{\hat{\mathbf{f}}'(x_i)\}_{i=1}^{K-1}$. All these parameters are modelled as the output of a neural network.

The derivative of such layer with respect to x is simply a quotient derivative. The function can be inverted by solving a quadratic equation. The authors suggested to use this layer either with a coupling architecture **RQ-NSF(C)** or with auto-regressive architecture **RQ-NSF(AR)**.

3.6.5 Neural autoregressive flow

A different approach was taken by Huang et al. [2018] in their Neural Autoregressive Flows (**NAF**). They modeled $\hat{\mathbf{f}}(\cdot\,;\theta)$ with a deep neural network. The authors proved the following statement giving a sufficient condition for a neural network to be bijective:

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

Two forms of neural networks for $\hat{\mathbf{f}}$ were suggested: the deep sigmoidal coupling layer (DSF) and deep dense sigmoidal coupling layer (DDSF); the resulting autoregressive flows are called **NAF-DSF** and **NAF-DDSF** respectively. In both cases the coupling layers are MLPs with layers of sigmoid and logit units and nonnegative weights - see Huang et al. [2018] for details. By Proposition 6, the resulting $\hat{\mathbf{f}}(\cdot\,;\theta)$ is a strictly monotonic function. The paper further proved that any strictly monotonic univariate function can be approximated with a DSF network[6]. Hence, NAF-DSF is a universal flow (Section 3.5.1).

5. Note that a monotone cubic polynomial has only one real root.
6. More formally, DSF networks are dense in the set of all strictly monotonic univariate continuous functions in the topology of compact convergence.

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: every strictly monotone function has a nonzero derivative. To model a strictly monotone function one has to model a strictly positive (or negative) function with a neural network and then integrate it using any numerical integration method. Authors demonstrated that UMNN requires less parameters than NAF to reach similar performance, and hence, it is more scalable for higher-dimensional datasets.

3.6.6 Sum-of-Squares polynomial flow

Jaini et al. [2019b] proposed to model $\hat{\mathbf{f}}(\cdot\,;\theta)$ as a strictly increasing polynomial. They proved that any strictly monotonic univariate continuous function can be approximated with an increasing univariate polynomial[7]. Hence, the resulting flow (called **SOS** - sum of squares polynomial flow) is a universal flow (see Section 3.5.1).

To construct all possible increasing single-variable polynomials the authors observed that the derivative of such a polynomial is a positive polynomial. Then a classical result from algebra was used: all positive single-variable polynomials are the sum of squares of polynomials. Hence, to get a 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).

As Jaini et al. [2019b] stated, SOS is easier to train than NAF, because there are no restrictions on the parameters (like positivity of weights). Further, if one takes L=0, SOS reduces to the affine coupling layer and as such can be viewed as a generalization of the basic affine coupling 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. 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 (see Figure 5).

Dinh et al. [2019] constructed a flow $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$ with a coupling architecture with piecewise-bijective coupling layer in the normalizing direction - from data distribution to (simpler) base distribution. This means that there is a covering of the data domain, and each subset of this covering is separately mapped to the base distribution. It follows that Equation 1 is not applicable for flows with piecewise-bijective coupling layers; each part of the base distribution receives contributions from each subset of the data domain. Dinh et al. [2019] suggested how to correct this by using a probabilistic mapping from the base domain to the data domain; for each part of the base domain there is some probability that the mapping is to each subset of the data domain.

7. More formally, a set of increasing univariate polynomials is dense in the set of all strictly monotonic univariate continuous functions in the topology of compact convergence.

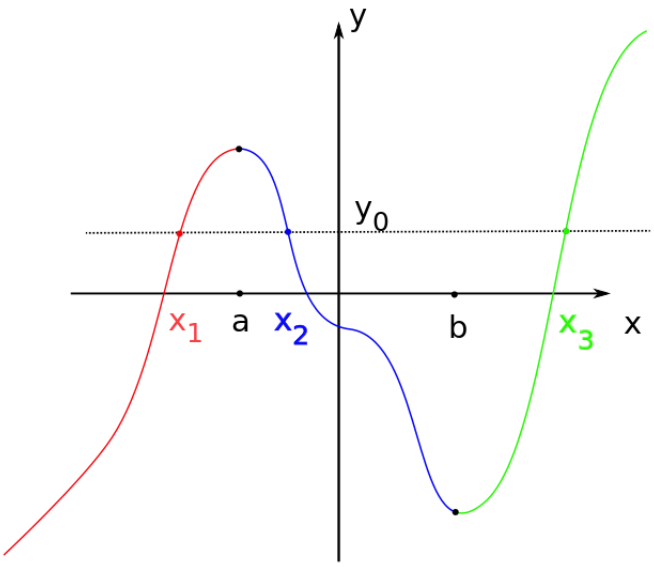

Figure 5: Piecewise-bijective function $\hat{\mathbf{f}}$. Consider the covering $\mathbb{R} = A_1 \bigsqcup A_2 \bigsqcup A_3$, where $A_1 = (-\infty, a)$, $A_2 = [a, b)$, and $A_3 = [b, \infty)$. Then the restriction of $\hat{\mathbf{f}}$ onto each A_i is a strictly monotone function. The point y_0 has three pre-images x_1, x_2 and x_3. For inverting the function, one samples the pre-image of y_0 from the gating network $p_{[3]|Y}(k|y)$.

More formally, let us denote the input with x and the output with y, and consider a lookup function $\phi : \mathbb{R} \to [K] = \{1, \dots, K\}$, such that $\phi(x) = k$, if $x \in A_k$. One can define a new map $\mathbb{R} \to \mathbb{R} \times [K]$, given by the rule $x \mapsto (\hat{\mathbf{f}}(x), \phi(x))$, and a density on a target space $p_{Y,[K]}(y, k) = p_{[K]|Y}(k|y)p_Y(y)$. One can think of this as an unfolding of the non-injective map $\hat{\mathbf{f}}$. In particular, for each point y one can find its pre-image by sampling from $p_{[K]|Y}$, 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_X:

$$p_X(x) = p_{Y,[K]}(\hat{\mathbf{f}}(x), \phi(x))|\mathrm{D}\hat{\mathbf{f}}(x)|. \tag{28}$$

The resulting flow (called **RAD** - for "real and discrete") was demonstrated to efficiently learn distributions with discrete structures (multimodal distributions, distributions with holes, discrete symmetries etc).

3.7 Residual Flows

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

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

(such a function is called a *residual connection*), where 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 are 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 clearly invertible. However, the computation of the Jacobian determinant is not efficient.

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 [Haber et al., 2017; E, 2017]. Consider an ODE

$$\frac{d}{dt}\mathbf{x}(t) = F(\mathbf{x}(t), \theta(t)), \tag{31}$$

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

$$\mathbf{x}_{n+1} - \mathbf{x}_n = \varepsilon F(\mathbf{x}_n, \theta_n), \tag{32}$$

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

Guided by the rich and elaborated theory of ODEs, one can look for improvements of residual blocks. For example, Chang et al. [2018, 2019] proposed several architectures for stable networks, which are robust to the input noise and easier to train, and some of these networks were demonstrated to be invertible. However the Jacobian determinant of these network cannot be computed efficiently.

A sufficient condition for the invertibility was found in [Behrmann et al., 2018]. They proved the following statement:

Proposition 7 *The residual connection* (29) *is invertible, if the Lipschitz constant of the residual block is* $Lip(F) < 1$.

There is no analytically closed form of the inverse of the network, 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. [2018], 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 estimation . They considered the residual connection $\mathbf{f}$ in Equation (29); its Jacobian 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: $|\det \mathrm{D}\mathbf{f}| = |\det(I + \mathrm{D}F)| = \det(I + \mathrm{D}F) = \det \mathrm{D}\mathbf{f}$. One can write the following chain of equalities:

$$\ln |\det \mathrm{D}\mathbf{f}| = \ln \det \mathrm{D}\mathbf{f} = \mathrm{Tr}(\ln \mathrm{D}\mathbf{f}) = \mathrm{Tr}(\ln (I + \mathrm{D}F)), \tag{33}$$

where the second equality is a linear algebra identity: $\ln \det \mathbf{A} = \mathrm{Tr} \ln \mathbf{A}$ (see *loc.cit* for the proof). Then one considers a power series for the trace of the matrix logarithm:

$$\mathrm{Tr}(\ln (I + \mathrm{D}F)) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{\mathrm{Tr}(\mathrm{D}F)^k}{k}. \tag{34}$$

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

Truncating the power series in Equation (34) gives a biased estimation of the log Jacobian determinant of the flow (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 at this step. During this process one needs to reweight terms for an unbiased estimation.[8]

3.8 Infinitesimal (Continuous) Flows

In the previous section we discussed a discretization of ODEs. But what if one does not discretize and tries 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

First let us set up the mathematical formulation. One will need Definition 2 for this section.

Consider an ODE as in Equation (31), where $t \in [0, 1]$. Under some assumptions on the evolution function $F(\mathbf{x}, t)$ (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 (this fact is known as 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 it satisfies the group law: $\Phi^t \circ \Phi^s = \Phi^{t+s}$. Mathematically speaking, an ODE* (31) *defines a one-parameter group of diffeomorphisms on $\mathbb{R}^D$. Such group is called a smooth flow in the dynamical systems theory and differential geometry - see Katok and Hasselblatt [1995] for details.*

8. More formally, for a series $\sum_{i=1}^{\infty} a_i$ one has an equality $\sum_{i=1}^{\infty} a_i = \mathbb{E}_{n \sim p(N)}[\sum_{i=1}^{n} \frac{a_i}{\mathbb{P}(N \geq k)}]$, where N is a random variable, which support is natural numbers (e.g, geometric distribution).

In particular, when $t = 1$, a 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 the 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 $\theta(t)$. As mentioned, the invertibility of such networks naturally comes from the theorem of the existence and uniqueness of the solution of the ODE.

Training these types of networks for a supervised downstream task could be done by continuous analog of backpropagation, the so called *adjoint sensitivity method*, introduced by Pontryagin. It computes gradients of the loss function (whatever is more relevant for the task) by solving a second (*augmented*) ODE backwards in time.[9] For density estimation learning, the change of variables formula is given by another ODE:

$$\frac{d}{dt}\log(p(\mathbf{x}(t))) = -\mathrm{Tr}\left(\frac{dF(\mathbf{x}(t))}{d\mathbf{x}(t)}\right). \tag{35}$$

Note that one no longer needs determinant computation (compare with Equation (1)). To train the model and to sample from $p_{\mathbf{Y}}$ one needs to solve these ODEs, which can be done with any numerical ODE solver.

Grathwohl et al. [2018] proposed a slightly modified version (called **FFJORD**), where they provided an unbiased stochastic estimation of the trace-term, using the Hutchinson estimator. This reduced the complexity even further.

An interesting side-effect of considering continuous ODE-type flows is that one needs much fewer parameters to achieve the same performance. For example, Grathwohl et al. [2018] 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. This is a well-studied question in mathematics - see [Katok and Hasselblatt, 1995; Arango and Gómez, 2002]. Without going too deep into the theory of dynamical systems, one can easily find a necessary condition. Note that 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. Because the Jacobian determinant of a diffeomorphism is nonzero, the sign of it can not change along the path. Hence a time one map must have a positive Jacobian determinant (such maps are called orientation preserving). 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 as time one maps. Their model is called Augmented Neural ODE (**ANODE**). They simply added variables $\mathbf{a}(t) \in \mathbb{R}^p$ and considered a new ODE:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}(t)\\ \mathbf{a}(t)\end{bmatrix} = \hat{F}\left(\begin{bmatrix}\mathbf{x}(t)\\ \mathbf{a}(t)\end{bmatrix}, \theta(t)\right) \tag{36}$$

9. If one has the loss $L(\mathbf{x}(t))$, where $\mathbf{x}(t)$ is a solution of ODE (31), one can consider its sensitivity $\mathbf{a}(t) = \frac{dL}{d\mathbf{x}(t)}$, also called adjoint. It is the analog of the derivative of the loss with respect to the hidden layer. In a standard neural network, one uses the backpropagation formula to find 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: $\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)\frac{dF(\mathbf{x}(t),\theta(t))}{d\mathbf{x}(t)}$. For a more formal description, see Chen et al. [2018b].

with initial condition $\begin{bmatrix}\mathbf{x}(0)\\ \mathbf{a}(0)\end{bmatrix} = \begin{bmatrix}\mathbf{z}\\ 0\end{bmatrix}$. Note that addition of free variables, in particular, gives freedom to keep the Jacobian determinant 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, hence this is a universal flow.

Salman et al. [2018] take a similar approach and presented Deep Diffeomorphic Flows, where the authors modelled a path in the space of diffeomorphisms as a solution of an ODE in that space. They also 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: assume that we have a complicated and irregular distribution $p_{\mathbf{Y}}(\mathbf{y})$ on $\mathbb{R}^D$, then we can mix it well enough to produce a simple base distribution $p_{\mathbf{Z}}(\mathbf{z})$ (either normal or uniform on some compact support). If the mixing obeys certain rules (stochastic equations), then this procedure can be invertible, and we obtain a bijection this way. This idea was explored in the literature by several authors (Suykens et al. [1998]; Welling and Teh [2011]; Rezende and Mohamed [2015]; Sohl-Dickstein et al. [2015]; Salimans et al. [2015]; Jankowiak and Obermeyer [2018]; Chen et al. [2018a]). We will provide a high-level overview of the method, including the necessary mathematical background.

A stochastic differential equation (SDE), also called Itô process, is an equation describing a change of a random variable in time. It is written as:

$$d\mathbf{x}(t) = b(\mathbf{x}(t), t)dt + \sigma(\mathbf{x}(t), t)dB_t, \tag{37}$$

where $t \in \mathbb{R}_+$, $\mathbf{x}(t) \in \mathbb{R}^D$, $b(\mathbf{x}, t) \in \mathbb{R}^D$ - *drift coefficient*, $\sigma(\mathbf{x}, t) \in \mathbb{R}^{D\times D}$ - *diffusion coefficient*, and B_t is D-dimensional *Brownian motion*. Under some assumption on the functions b and σ, the solution $\mathbf{x}(t)$ exists and is unique [Oksendal, 1992]. Informally, one can interpret the drift term as a deterministic dynamic and the diffusion term as providing all the stochasticity and mixing. In particular, the diffusion process satisfies this equation.

For a time-dependent random variable $\mathbf{x}(t)$ one can consider its density function $p(\mathbf{x}, t)$, which is also time dependent. If $\mathbf{x}(t)$ is a solution of Equation (37), its density function satisfies two partial differential equations describing the forward and backward evolution (in the positive and negative direction of time, respectively) [Oksendal, 1992]. The forward evolution is given by Fokker-Plank equation (or Kolmogorov's forward equation):

$$\frac{\partial}{\partial t}p(\mathbf{x}, t) = -\nabla_{\mathbf{x}} \cdot (b(\mathbf{x}, t)p(\mathbf{x}, t)) + \sum_{i,j} \frac{\partial^2}{\partial x_i \partial x_j} D_{ij}(\mathbf{x}, t)p(\mathbf{x}, t), \tag{38}$$

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

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

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

As observed by Suykens et al. [1998], asymptotically the Langevin flow can learn any distribution, if one picks the drift and diffusion coefficients appropriately. However this result is not very practical, because one needs to know the (unnormalized) density function of the data distribution for that.

One can see that if the diffusion coefficient is zero, the Itô process reduces to an ODE, and the Fokker-Plank equation becomes a Liouville's equation. The connection between this and Equation (35), which describes the log density changes, was observed in 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 consider discrete time $t = 0, \ldots, T$. For each time shot t, $\mathbf{x}^t$ is a random variable at each time shot, $\mathbf{x}^0 = \mathbf{y}$ is a data point, and $\mathbf{x}^T = \mathbf{z}$ a base point. The forward transition probability $q(\mathbf{x}^t|\mathbf{x}^{t-1})$ is taken to be either normal or binomial distribution (whose parameters are trainable parameters of the model). As was noticed in [Sohl-Dickstein et al., 2015], it follows from Kolmogorov's backward equation 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, respectively, with trainable parameters). 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 in practice, but one can find a lower bound on L (similarly to in variational inference).

Several papers have worked explicitly with the SDE (37) [Chen et al., 2018a; Peluchetti and Favaro, 2019; Tzen and Raginsky, 2019; Liutkus et al., 2019]. In particular, Chen et al. [2018a] discuss how to create an interesting posterior for variational inference. For that they sampled a latent variable $\mathbf{z}_0$ conditioned on the input $\mathbf{x}$, and then evolved $\mathbf{z}_0$ with SDE. In practice this evolution can be computed by discretization. Also, by analogy to Neural ODEs, **Neural Stochastic Differential Equations** were proposed [Peluchetti and Favaro, 2019; Tzen and Raginsky, 2019]. Here coefficients of the equation are modelled as neural networks, and black box SDE solvers are used for the inference.[10]

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 also 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.

Note that for preliminary qualitative analysis of flow performance people usually use synthetic 2-dimensional datasets. Such datasets (grid Gaussian mixture, ring Gaussian mixture, two moons, two circles, spiral, many moons, checkboard etc.) exhibit multimodal-

10. To train Neural SDE one needs an analog of backpropagation. Tzen and Raginsky [2019] used Kunita's theory of stochastic flows.

Architecture	Coupling layer	Flow
Coupling flows, 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 flows, 3.5	affine	MAF
	polynomial, 3.6.6	SOS
	neural network, 3.6.5	NAF UMNN
	splines	quadratic (AR) RQ-NSF(AR)
Residual flows, 3.7		iResNet
		Residual Flow
ODE-based, 3.8.1		FFJORD

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

ity, strong curvature and other peculiarities which an expressive flow should be able to model.

4.1 Tabular datasets

We describe datasets as they were preprossessed in Papamakarios et al. [2017][11]. The relatively small size makes these datasets reasonable for a first test of unconditional density estimation models.

UCI datasets A collection of datasets from University of California, 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. HEPMASS: measurements from high-energy physics experiments aiming to detect particles with unknown mass.
4. MINIBOONE: measurements from MiniBooNE experiment for observing neutrino oscillations.

BSDS300 Berkeley segmentation dataset [Martin et al., 2001] contains segmentations of natural images. Papamakarios et al. [2017] extracted 8×8 random monochrome patches from it.

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

All these datasets were cleaned and dequantized by adding uniform noise, so they can be considered samples from an absolutely continuous distribution. In Table 2 we provide a summary of these datasets.

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Dimensions	6	8	21	43	63
Examples in train set	$\approx$ 1.7M	$\approx$ 800K	$\approx$ 300K	$\approx$ 30K	$\approx$ 1M

Table 2: Summary of the tabular datasets.

In Table 3 we present performance of flows on tabular datasets. For the details of the experiments see the following papers: realNVP and MAF [Papamakarios et al., 2017], Glow and FFJORD [Grathwohl et al., 2018], 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].

	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)	$0.62_{\pm 0.01}$	$11.91_{\pm 0.13}$	$-15.09_{\pm 0.40}$	$\mathbf{-8.86}_{\pm 0.15}$	$157.73_{\pm 0.04}$
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	$0.63_{\pm 0.01}$	$10.89_{\pm 0.70}$	$\mathbf{-13.99}_{\pm 0.21}$	$-9.67_{\pm 0.13}$	$\mathbf{157.98}_{\pm 0.01}$
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)	$\mathbf{0.66}_{\pm 0.01}$	$12.91_{\pm 0.02}$	$-14.67_{\pm 0.03}$	$-9.72_{\pm 0.47}$	$157.42_{\pm 0.28}$
Cubic Spline	$0.65_{\pm 0.01}$	$\mathbf{13.14}_{\pm 0.02}$	$-14.59_{\pm 0.02}$	$-9.06_{\pm 0.48}$	$157.24_{\pm 0.07}$
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)	$\mathbf{0.66}_{\pm 0.01}$	$13.09_{\pm 0.02}$	$-14.01_{\pm 0.03}$	$-9.22_{\pm 0.48}$	$157.31_{\pm 0.28}$

Table 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.

One can see that universal flows (NAF, SOS, Splines) demonstrate relatively better performance.

4.2 Image datasets

These are datasets of increasing complexity. They are used preprocessed as in Dinh et al. [2017] by dequantizing with uniform noise (except for Flow++). The most common image datasets used for evaluating normalizing flows are MNIST and CIFAR-10, however other (Omniglot, LSUN, CIFAR-100, CelebA HQ, SVHN, etc) could be used too. In Table 4 we give the summary of the image datasets.

	MNIST	CIFAR-10	ImageNet32	ImageNet64
Dimensions	784	3072	3072	12288
Examples in train set	50K	90K	$\approx$ 1.3M	$\approx$ 1.3M

Table 4: Summary of image datasets.

Performance of flows on image datasets is given in Table 5 for unconditional density estimation. For the details of the experiments see the following papers: 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., 2018], SOS [Jaini et al., 2019b], RQ-NSF [Durkan et al., 2019b], UMNN [Wehenkel and Louppe, 2019], iResNet [Behrmann et al., 2018], Residual Flow [Chen et al., 2019], Flow++ [Ho et al., 2019].

	MNIST	CIFAR-10	ImageNet32	ImageNet64
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**

Table 5: Average test negative log-likelihood (in bits per dimension) for density estimation on image datasets (lower the better).

One can see that Flow++ is the overall winner. 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. Ablation study in the paper shows that the latter twist gave the most significant improvement. It would be interesting to test other flow models with variational dequantization.

5. Discussion and open problems

5.1 Inductive biases

Role of the base measure The base measure of a normalizing flow is (generally) assumed to be a simple distribution such as a uniform or Gaussian. However as noted in several places this does not need to be the case. Any distribution for which we can easily sample from and compute the log probability density function is possible and the parameters of this distribution can also be learned during training.

One important thing to note is that theoretically a 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 need to 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.

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. The natural question to ask: 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 conditional probability distribution. Trippe and Turner [2018] suggested to use different flows for each condition, but this approach doesn't leverage weight sharing, and as a result may be inefficient in terms of memory and data usage. Atanov et al. [2019] proposed to use 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 $p(z_t|z_{<t})$. This context was considered in [Kumar et al., 2019].

Loss function The majority of the existing flows are trained by minimization of KL-divergence between the source and the target distributions (or, equivalently, with log-likelihood maximization). However, other losses could be used which would put normalizing flows in a broader context of optimal transport theory [Villani, 2003]. Interesting work has been done in this direction including the Flow-GAN [Grover et al., 2018] paper mentioned earlier 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

Flows on manifolds. A mathematically interesting problem is warping probability distributions on a manifold. This idea was explored in [Falorsi et al., 2019], 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. In general, the question is how best to define a normalizing flow on a differentiable manifold.

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, how one can define normalizing flows if one or both of our distributions are discrete? Answers to this could be useful for many applications including natural language processing, graph generation and others.

Several approaches have been suggested. Tran et al. [2019] models 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. Further, several approaches transform a discrete variable into a continuous latent variable with a variational autoencoder, and then apply normalzing flows in the continuous latent space [Wang and Wang, 2019; Ziegler and Rush, 2019].

Finally, another approach is dequantization, *i.e.*, adding noise to discrete data to make it continuous, 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.

References

Jaime Arango and Adriana Gómez. Diffeomorphisms as time one maps. *Aequationes Math.*, 64:304–314, 2002. URL `https://doi.org/10.1007/PL00013195`.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *ArXiv*, 2017. URL `https://arxiv.org/abs/1701.07875`.

Vladimir Arnold. *Ordinary Differential Equations*. The MIT Press, 1978. ISBN 0-262-51018-9.

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. URL `https://arxiv.org/abs/1905.00505`.

Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. 11 2018. URL `https://arxiv.org/pdf/1811.00995`.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. 08 2013. URL https://arxiv.org/abs/1308.3432.

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. URL https://arxiv.org/abs/1511.06349.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible Architectures for Arbitrarily Deep Residual Neural Networks. In *AAAI*, 2018. URL https://docs.wixstatic.com/ugd/a28d7e_6b02dd4fe34a4a32b7a08f47ddf69f50.pdf.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryxepo0cFX.

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. URL https://arxiv.org/pdf/1709.01179.pdf.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018b. URL https://arxiv.org/abs/1806.07366.

Ricky T. Q. Chen, Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Residual Flows for Invertible Generative Modeling. 6 2019. URL https://arxiv.org/abs/1906.02735.

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. URL https://arxiv.org/abs/1710.07035.

Hari Prasanna Das, Pieter Abbeel, and Costas J. Spanos. Dimensionality Reduction Flows. 8 2019. URL https://arxiv.org/abs/1908.01686.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation. In *ICLR Workshop*, 2015. URL https://arxiv.org/abs/1410.8516.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density Estimation using Real NVP. In *ICLR*, 2017. URL https://arxiv.org/abs/1605.08803.

Laurent Dinh, Jascha Sohl-Dickstein, Razvan Pascanu, and Hugo Larochelle. A RAD approach to deep mixture models. page 9, 3 2019. URL https://arxiv.org/abs/1903.07714.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs. 4 2019. URL https://arxiv.org/abs/1904.01681.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows. In *Workshop on Invertible Neural Networks and Normalizing Flows, International Conference on Machine Learning*, 2019a. URL https://arxiv.org/abs/1906.02145.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural Spline Flows. 6 2019b. URL https://arxiv.org/abs/1906.04032.

Weinan E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5:1–11, 02 2017. doi: 10.1007/s40304-017-0103-z.

Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla-Romeu-Santos. Universal audio synthesizer control with normalizing flows. 07 2019. URL https://arxiv.org/abs/1907.00971.

Luca Falorsi, Pim de Haan, Tim R. Davidson, and Patrick Forré. Reparameterizing Distributions on Lie Groups. 3 2019. URL https://arxiv.org/abs/1903.02958.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked Autoencoder for Distribution Estimation. In *ICML*, 2015. URL http://proceedings.mlr.press/v37/germain15.pdf.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The Reversible Residual Network: Backpropagation Without Storing Activations. In *NIPS*, 2017. URL https://github.com/renmengye/revnet-public.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative Adversarial Nets. In *NIPS*, 2014.

Will Grathwohl, Ricky T Q Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. Technical report, 2018. URL https://arxiv.org/abs/1810.01367.

Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-GAN: Combining Maximum Likelihood and Adversarial Learning in Generative Models. In *AAAI*, 2018. URL https://arxiv.org/abs/1705.08868.

Eldad Haber, Lars Ruthotto, and Elliot Holtham. Learning across scales - a multiscale method for convolution neural networks. 03 2017. URL https://arxiv.org/abs/1703.02009.

Leonard Hasenclever, Jakub M Tomczak, Rianne Van Den Berg, and Max Welling. Variational Inference with Orthogonal Normalizing Flows. In *NIPS Workshop on Bayesian Deep Learning*, 2017. URL http://bayesiandeeplearning.org/2017/papers/51.pdf.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV*, 2015. URL https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. URL https://arxiv.org/abs/1512.03385.

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*, 2019. URL https://openreview.net/forum?id=Hyg74h05tX.

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*, 2019a. URL http://proceedings.mlr.press/v97/hoogeboom19a.html.

Emiel Hoogeboom, Jorn W.T. Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression, 5 2019b. URL https://arxiv.org/abs/1905.07376.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural Autoregressive Flows. In *ICML*, 2018. URL https://arxiv.org/abs/1804.00779.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, 2015. URL http://proceedings.mlr.press/v37/ioffe15.pdf.

Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. i-RevNet: Deep Invertible Networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJsjkMb0Z.

Priyank Jaini, Ivan Kobyzev, Marcus Brubaker, and Yaoliang Yu. Tails of Triangular Flows. 7 2019a. URL https://arxiv.org/abs/1907.04481.

Priyank Jaini, Kira A. Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. In *Proceedings of the 36th International Conference on Machine Learning*, 5 2019b. URL https://arxiv.org/abs/1905.02325.

Martin Jankowiak and Fritz Obermeyer. Pathwise derivatives beyond the reparameterization trick. In *Proceedings of the 35th International Conference on Machine Learning, PMLR*, volume 8, pages 2235–2244, 2018. URL https://arxiv.org/abs/1806.01851.

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, Long Beach, California, PMLR 97*, 11 2018. URL https://arxiv.org/abs/1811.02155.

Diederik P Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. Technical report, 2018. URL https://github.com/openai/glow.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014. URL https://arxiv.org/abs/1312.6114.

Diederik P. Kingma and Max Welling. An Introduction to Variational Autoencoders. 06 2019. URL https://arxiv.org/abs/1906.02691.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved Variational Inference with Inverse Autoregressive Flow. In *NIPS*, 2016. URL http://papers.nips.cc/paper/6581-improved-variational-inference-with-inverse-autoregressive-flow.pdf.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. Massachusetts: MIT Press, 2009. ISBN 978-0-262-01319-2.

Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. VideoFlow: A Flow-Based Generative Model for Video. 3 2019. URL https://arxiv.org/abs/1903.01434.

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 ICML, Long Beach, California, PMLR 97*, 2019. URL https://arxiv.org/abs/1806.08141.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *ICML*, 2013. URL https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. GraphNVP: An Invertible Flow Model for Generating Molecular Graphs. 05 2019. URL https://arxiv.org/abs/1905.11600.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 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. 2018. URL https://tom94.net/data/publications/mueller18neural/mueller18neural-v2.pdf.

Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving exploration in soft-actor-critic with normalizing flows policies. In *ICML workshop on Invertible Neural Networks and Normalizing Flows*, 06 2019. URL https://arxiv.org/abs/1906.02771.

Bernt Oksendal. *Stochastic Differential Equations (3rd Ed.): An Introduction with Applications*. Springer-Verlag, Berlin, Heidelberg, 1992. ISBN 3-387-53335-4.

Ivan Ovinnikov. Poincaré Wasserstein Autoencoder. *Bayesian Deep Learning Workshop (NeurIPS 2018)*, 2018. URL https://arxiv.org/abs/1901.01427.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation. In *NIPS*, 2017. URL https://arxiv.org/abs/1705.07057.

Stefano Peluchetti and Stefano Favaro. Neural Stochastic Differential Equations. 5 2019. URL https://arxiv.org/abs/1905.11065.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019*, pages 3617–3621, 05 2019. doi: 10.1109/ICASSP.2019.8683143. URL https://arxiv.org/abs/1811.00002.

Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *ICML*, 2015. URL https://arxiv.org/abs/1505.05770.

Oren Rippel and Ryan Prescott Adams. High-Dimensional Probability Estimation with Deep Density Models. Technical report, 2013. URL https://arxiv.org/abs/1302.5125.

Tim Salimans, Algoritmica Diederik, Diederik P Kingma, and Max Welling. Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. In *ICML*, 2015. URL https://arxiv.org/abs/1410.6460.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NIPS*, 2016. URL https://arxiv.org/abs/1606.03498.

Hadi Salman, Payman Yadollahpour, Tom Fletcher, and Nematollah Batmanghelich. Deep diffeomorphic normalizing flows. *CoRR*, abs/1810.03256, 2018. URL https://arxiv.org/abs/1810.03256.

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, Lille, France, 6-11 July 2015*, pages 2256–2265, 2015. URL http://jmlr.org/proceedings/papers/v37/sohl-dickstein15.html.

G. W. Stewart. *Matrix Algorithms: Volume 1, Basic Decompositions*. Society for Industrial and Applied Mathematic, 1998. ISBN 9780898714142.

Johan Suykens, Herman Verrelst, and Joos Vandewalle. On-Line Learning Fokker-Planck Machine. *Neural Processing Letters*, 7:81–89, 04 1998. doi: 10.1023/A:1009632428145.

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. ISSN 00103640. doi: 10.1002/cpa.21423. URL http://doi.wiley.com/10.1002/cpa.21423.

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. URL https://projecteuclid.org/download/pdf_1/euclid.cms/1266935020.

Ilya O. Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schölkopf. Wasserstein Auto-Encoders. *ArXiv*, 2018. URL https://arxiv.org/abs/1711.01558.

Jakub Tomczak and Max Welling. Improving Variational Auto-Encoders using convex combination linear Inverse Autoregressive Flow. *Benelearn*, 06 2017. URL https://arxiv.org/abs/1706.02326.

Jakub M Tomczak and Max Welling. Improving Variational Auto-Encoders using Householder Flow. Technical report, 2016. URL https://arxiv.org/pdf/1611.09630.pdf.

Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. Discrete Flows: Invertible Generative Models of Discrete Data . In *ICLR 2019 Workshop*, 2019. URL https://openreview.net/pdf?id=rJlo4UIt_E.

Brian Loeber Trippe and Richard E. Turner. Conditional Density Estimation with Bayesian Normalising Flows. 2018. URL https://arxiv.org/abs/1802.04908.

Belinda Tzen and Maxim Raginsky. Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit. 5 2019. URL https://arxiv.org/abs/1905.09883.

Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 393–402, 2018. URL https://arxiv.org/abs/1803.05649.

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. URL https://arxiv.org/pdf/1711.10433.pdf.

Cédric Villani. *Topics in optimal transportation (Graduate Studies in Mathematics 58)*. American Mathematical Society, Providence, RI, 2003. ISBN 0-8218-3312-X.

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. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8039016.

Prince Zizhuang Wang and William Yang Wang. Riemannian Normalizing Flow on Variational Wasserstein Autoencoder for Text Modeling. 04 2019. URL https://arxiv.org/abs/1904.02399.

Antoine Wehenkel and Gilles Louppe. Unconstrained Monotonic Neural Networks. 8 2019. URL https://arxiv.org/abs/1908.05164.

Max Welling and Yee Whye Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *ICML*, 2011. URL https://www.ics.uci.edu/~welling/publications/papers/stoclangevin_v6.pdf.

Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation Capabilities of Neural Ordinary Differential Equations. 7 2019. URL https://arxiv.org/abs/1907.12998.

Guoqing Zheng, Yiming Yang, and Jaime Carbonell. Convolutional Normalizing Flows. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018. URL https://arxiv.org/pdf/1711.02255.pdf.

Zachary M. Ziegler and Alexander M. Rush. Latent Normalizing Flows for Discrete Sequences. In *Proceedings of the 36th ICML, Long Beach, California, PMLR 97*, 2019. URL https://arxiv.org/abs/1901.10548.