Normalizing Flows based on Diffeomorphic Coupling Functions

Iñigo Martinez[1], Elisabeth Viles[2,3], and Igor G. Olaizola[1]

[1]Vicomtech Foundation, Basque Research and Technology Alliance (BRTA), San Sebastian, Spain
[2]TECNUN School of Engineering, University of Navarra, San Sebastian, Spain
[3]Institute of Data Science and Artificial Intelligence, University of Navarra, Pamplona, Spain

Abstract

Normalizing flows based on coupling layers require a bijective one-dimensional function and the derivative of the function with respect the input variable. Related flows based on coupling layers such as NICE and RealNVP have an analytic one-pass inverse, but are often less flexible than their autoregressive counterparts. Based on these limitations, this article proposes to implement the coupling function using the integration of continuous piecewise-affine (CPA) velocity functions as a building block. The module acts as a drop-in replacement for the affine or additive transformations commonly found in coupling and autoregressive transforms. When combined with alternating invertible linear transformations, the resulting class of normalizing flows is referred to as closed-form Diffeomorphic Spline Flows (DIFW-NF), which may feature coupling layers, DIFW-NF (C), or autoregressive layers, DIFW-NF (AR). Experiments demonstrate that this module significantly enhances the flexibility of both classes of flows, obtaining competitive results in a variety of high-dimensional datasets. Unlike the additive and affine transformations, which have limited flexibility, the proposed differentiable monotonic function with sufficiently many intervals can approximate any differentiable monotonic function, yet has a closed-form, tractable Jacobian determinant, and can be inverted analytically. Our parameterization is fully-differentiable, which allows for training by gradient methods.

1 Introduction

Generative models have become increasingly popular in recent years due to their ability to capture complex data distributions and generate high-quality, realistic samples. Not all model families permit efficient and accurate inference on both density estimation and sampling tasks. Indeed, the trade-offs in the inference capabilities of the current generative models have led to the development of very diverse approaches: Generative adversarial networks (GANs) (Goodfellow et al., 2014), Variational Autoencoders (VAEs) (Kingma & Welling, 2013), Energy-based models (LeCun & Huang, 2005) and Normalizing flows (Rezende & Mohamed, 2015).

Normalizing flows transform a simple probability distribution into a more complex one through a series of invertible transformations f. The key defining property of flow-based generative models is that transformations f must be **invertible** and both f and f^{-1} must be **differentiable**. Normalizing flows based on coupling functions require a bijective one-dimensional function $h(z)$ and the derivative of the function with respect the input variable z (also called the spatial dimension), i.e. $\frac{\partial h}{\partial z}$. Related flows based on coupling layers such as NICE (Dinh et al., 2014) and RealNVP (Dinh et al., 2016), have an analytic one-pass inverse, but are often less flexible than their autoregressive counterparts.

Based on these limitations, this work proposes to implement the **coupling function h using the integration of continuous piecewise-affine (CPA) velocity functions** as a building block. A fully-differentiable module based on the integration of CPA velocity functions is presented, which yield diffeomorphic curves. Computing the inverse of such curves is equivalent to computing the forward curve backward in time or with opposite sign of the parameters. These diffeomorphic curves are used as the coupling function $h(z)$.

The module acts as a **drop-in replacement for the affine or additive transformations** commonly found in coupling and autoregressive transforms. When combined with alternating invertible linear transformations, the resulting class of normalizing flows is referred to as **closed-form diffeomorphic spline flows** (DIFW-NF), which may feature coupling layers, DIFW-NF (C), or autoregressive layers, DIFW-NF (AR). Overall, DIFW-NF resembles a traditional feed-forward neural network architecture, alternating between linear transformations and element-wise non-linearities, while retaining an exact, analytic inverse. Experiments demonstrate that this module significantly enhances the flexibility of both classes of flows, obtaining competitive results in a variety of high-dimensional datasets.

The article is structured as follows: in Section 2 an extensive review of the coupling-based normalizing flows is presented. The proposed model is detailed in Section 3 and the experiments applied to 1D, 2D and ND data are included in Section 4. Finally, conclusions are stated in Section 5.

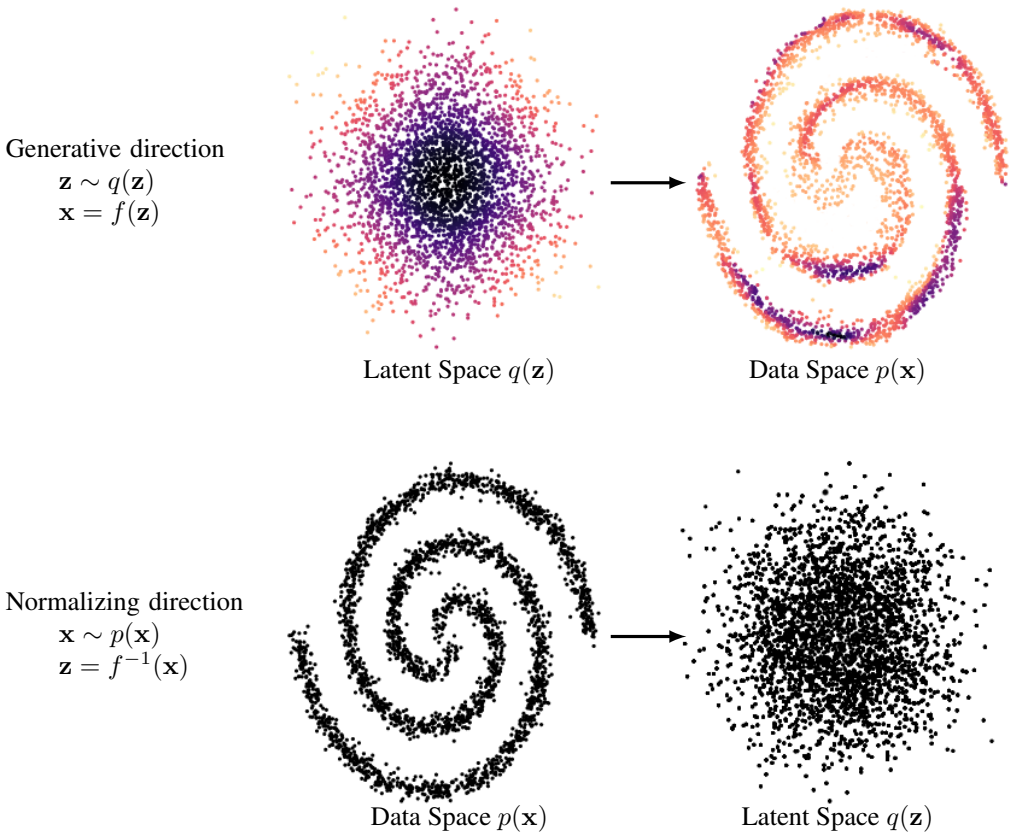


Figure 1. Normalizing flow on a toy two-dimensional dataset. The function $f(\mathbf{z})$ maps samples $\mathbf{z}$ from the latent distribution in the upper left into approximate samples $\mathbf{x}$ from the data distribution in the upper right. This corresponds to exact generation of samples from the model. The inverse function $f^{-1}(\mathbf{x})$ maps samples $\mathbf{x}$ from the data distribution in the lower left into approximate samples $\mathbf{z}$ from the latent distribution in the lower right. This corresponds to the exact inference of the latent state given the data.

2 Related Work

A normalizing flow is a transformation that converts a simple probability distribution $q(\mathbf{z})$ into a more complex one $p(\mathbf{x})$ by applying a sequence of invertible and differentiable mappings (Papamakarios et al., 2021) (see Figure 1). There has been a growing interest in normalizing flows in the deep learning community, driven by successful applications and structural advantages they have over alternatives: **model flexibility** and **generation speed**. Normalizing flows can represent a richer

family of distributions without requiring approximations. Flows have been explored both to increase the flexibility of the variational posterior in the context of VAEs, and directly as a generative model.

Autoregressive models such as MAF (Papamakarios et al., 2017) achieve state-of-the-art density estimation performance on many challenging real-world datasets, but generally suffer from slow sampling time due to their autoregressive structure. These flows are d times slower to invert than to evaluate, where d is the dimensionality of $\mathbf{x}$. **Inverse autoregressive models** such as IAF (Kingma et al., 2016) can sample quickly and potentially have strong modeling capacity, but they cannot be trained efficiently by maximum likelihood. Subsequent work which enhances the flexibility of autoregressive flows has resulted in models which do not have an analytic inverse, and require numerical optimization to invert. For instance, inverting the non-affine transformations used by NAF (Huang et al., 2018) and block-NAF (De Cao et al., 2020) would require numerical optimization. Transformations that are equally fast to invert and evaluate do exist. **Continuous flows** such as Neural ODEs (Chen et al., 2018) and FFJORD (Grathwohl et al., 2019) are equally fast in both directions, but they require numerically integrating a differential equation in each direction, which can be slower than a single neural-network pass.

Coupling flows and autoregressive flows have a similar functional form and both have coupling functions as building blocks. A coupling function is a bijective differentiable function $\mathbf{h}(\cdot, \theta) : \mathcal{R}^d \to \mathcal{R}^d$, parametrized by θ. In coupling flows, these functions are typically constructed by applying a scalar coupling function $h(\cdot, \theta) : \mathcal{R} \to \mathcal{R}$ element-wise. In autoregressive flows, $d = 1$ and hence they are also scalar valued. **Additive and affine** coupling functions are used for coupling flows in NICE (Dinh et al., 2014), RealNVP (Dinh et al., 2016), Glow (Kingma & Dhariwal, 2018) and for autoregressive architectures in IAF and MAF. Both the affine and additive transformations are easy to invert, but they lack flexibility. (Ziegler & Rush, 2019) proposed an invertible **non-linear squared** transformation that adds an inverse-quadratic perturbation to an affine transformation in an autoregressive flow. The **Flow++ model** (Ho et al., 2019) uses the cumulative distribution function (CDF) of a mixture of logistic distributions as a monotonic transformation. In this case the computation of the inverse is done numerically with the bisection algorithm since a closed form is not available. The derivative of the transformation with respect to z is expressed in terms of probability density function of logistic mixture. Also related, (Jaini et al., 2019) parametrize the monotonic transformation as a strictly increasing **sum of squares polynomial** (SOS). For low-degree polynomials, an analytic inverse may be available, but the method would require an iterative solution in general. SOS is easier to train than NAF, because there are no restrictions on the parameters (like positivity of weights).

Regarding **linear splines**, (Müller et al., 2019) divided the domain into k equal bins and modeled h as the integral of a positive piecewise-constant function. (Müller et al., 2019) also used a **monotone quadratic spline** on the unit interval and modeled it as the integral of a positive piecewise-linear function. Such spline is invertible but finding its inverse requires solving a quadratic equation. (Durkan et al., 2019a) proposed using **monotone cubic splines** defined only on the interval $[0, 1]$. To ensure that the input is always between 0 and 1, a sigmoid transformation was placed before each coupling layer, and a logit transformation after each coupling layer. A monotone cubic polynomial has only one real root and for inversion, one can find this either analytically or numerically. The flow can be trained by gradient descent by differentiating through the numerical root finding method. However, the procedure is numerically unstable if not treated carefully, as noted by (Durkan et al., 2019b) when the sigmoid saturates for values far from zero, Also related, (Durkan et al., 2019b) model a coupling function as a **monotone rational-quadratic spline** on an interval and as the identity in the rest of the domain. The derivative is a quotient derivative and the inverse is obtained by solving a quadratic equation. The RQ-NSF (C) coupling architecture and RQ-NSF(AR) autoregressive architectures used these coupling functions.

In the hope of creating an ideal likelihood-based generative model that simultaneously has fast sampling, fast inference, and strong density estimation performance, this article proposes replacing the conditional affine transformation (Kingma et al., 2016) with a more rich family of transformations, and note the requirements for doing so. In particular, we propose a novel normalizing-flow model based on a coupling function h using the integration of continuous piecewise-affine velocity functions as a building block. Approaches like SOS (Jaini et al., 2019), cubic splines (Durkan et al., 2019a) and Flow++ (Ho et al., 2019) present couplings that are similar in spirit to our approach.

3 Diffeomorphic Coupling Functions

Normalizing flows based on coupling functions require a **bijective one-dimensional function** $h(z)$ and the derivative of the function with respect the input variable z (also called the spatial dimension), i.e. $\frac{\partial h}{\partial z}$. This work proposes a fully-differentiable module based on the **integration of continuous piecewise-affine (CPA) velocity functions**, which yield diffeomorphic curves $\phi^\theta(x,t)$ (differentiable, invertible and with differentiable inverse). As a matter of fact, computing the inverse is exactly the same as computing the forward curve backward in time or by flipping the sign of the parameters. These diffeomorphic curves will be used as the coupling function, so $h(z) = \phi^\theta(z,t)$. The diffeomorphic function itself maps an interval $[-B, B]$ to $[-B, B]$, as illustrated on Figure 2. The transformation outside this range is defined as the identity, resulting in linear tails, so that the overall transformation can take unconstrained inputs.

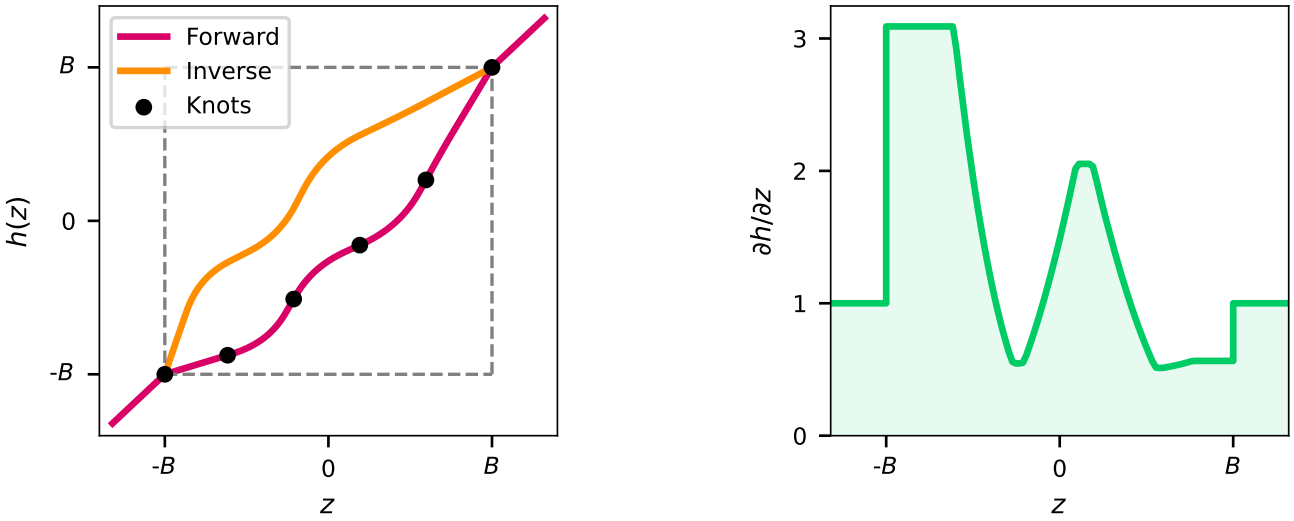


Figure 2. The proposed diffeomorphic transforms are drop-in replacements for additive or affine transformations in coupling or autoregressive layers, greatly enhancing their flexibility while retaining exact invertibility. **Left**: Forward and inverse transformer of a random diffeomorphic monotonic function with tessellation size of 5 bins and linear tails, which is parametrized by a series of 6 points in the plane, and the 5 derivatives at the internal knots. **Right**: Transformer derivative with respect to z.

The module acts as a **drop-in replacement for the affine or additive transformations** commonly found in coupling and autoregressive transforms. Unlike the additive and affine transformations, which have limited flexibility, the proposed differentiable monotonic function with sufficiently many intervals can approximate any differentiable monotonic function on the specified interval $[-B, B]$, yet has a closed-form, tractable Jacobian determinant, and can be inverted analytically. The proposed parameterization is fully-differentiable, which allows for training by gradient methods.

The proposed formulation can also easily be adapted for autoregressive transforms; each θ_k can be computed as a function of $\mathbf{z}_{1:i}$ using an autoregressive neural network, and then all elements of $\mathbf{z}$ can be transformed at once. When combined with alternating invertible linear transformations, the resulting class of normalizing flows is referred to as **closed-form diffeomorphic spline flows (DIFW-NF)**, which may feature coupling layers, DIFW-NF (C), or autoregressive layers, DIFW-NF (AR). Experiments demonstrate that this module significantly **enhances the flexibility** of both classes of flows, and in some cases brings the performance of coupling transforms on par with the best-known autoregressive flows. DIFW-NF only requires a single neural-network pass in either the forward or the inverse direction, but in practice is as flexible as state-of-the-art autoregressive flows. Overall, DIFW-NF resembles a traditional feed-forward neural network architecture, alternating between linear transformations and element-wise non-linearities, while retaining an exact, analytic inverse. Like Real NVP or Glow, DIFW-NF flows can represent either a transformation from data to noise, or from noise to data. In both cases, the transformation requires only a single pass of the neural network defining the flow. Note that for the rest of the section the spatial dimension of the transformation will be referred to as x instead of z.

3.1 CPA-based Diffeomorphic Transformations

Let's recall the CPA-based diffeomorphic transformations introduced in (Martinez et al., 2022). A diffeomorphism can be obtained, via integration, from uniformly continuous stationary velocity fields $T^\theta(x) = \phi^\theta(x, 1)$ where $\phi^\theta(x,t) = x + \int_0^t v^\theta(\phi^\theta(x,\tau))d\tau$ for uniformly continuous $v : \Omega \to \mathbb{R}$ and integration time t. Here x refers to the spatial dimension and t the integration time. The solution for the integral equation is a composition of a finite number of solutions ψ, given by: $\phi^\theta(x,t) = \left(\psi^{t_m}_{\theta,c_m} \circ \psi^{t_{m-1}}_{\theta,c_{m-1}} \circ \cdots \circ \psi^{t_2}_{\theta,c_2} \circ \psi^{t_1}_{\theta,c_1}\right)(x)$, where m is the number of cells visited and $\psi^t_{\theta,c}$ is the solution of a basic ODE $\frac{d\psi}{dt} = v^\theta(\psi)$ with an $\mathbb{R} \to \mathbb{R}$ affine velocity field: $v^\theta(\psi) = a^\theta\psi + b^\theta$ and an initial condition $\psi(x, 0) = x$.

During the iterative process of integration, several cells are crossed, starting from c_1 at integration time $t_1 = 1$, and finishing at c_m at integration time t_m. The integration time t_m of the last cell c_m can be calculated by subtracting from the initial integration time the accumulated boundary hitting times t_{hit}: $t_m = t_1 - \sum_{i=1}^{m-1} t^\theta_{hit}(c_i, x_i)$. The final integration point x_m is the boundary of the penultimate cell c_{m-1}: $x_m = x_{c_{m-1}}$. In case only one cell is visited, both time and space remain unchanged: $t_m = 1$ and $x_m = x$. Taking this into consideration, the trajectory can be calculated as follows:

$$\phi^\theta(x,t) = \psi^\theta(x = x_m, t = t_m) = \left(xe^{ta_c} + \left(e^{ta_c} - 1\right)\frac{b_c}{a_c}\right)_{\substack{x=x_m \\ t=t_m}} \tag{1}$$

Gradient-based optimization algorithms require the derivatives of the transformation with respect to the model parameters θ and x. That is, we need to obtain:

- $\boxed{\frac{\partial\phi^\theta(x,t)}{\partial\theta}}$: partial derivative of transformation $\phi^\theta(x,t)$ w.r.t. θ. Presented at (Martinez et al., 2022).
- $\boxed{\frac{\partial\phi^\theta(x,t)}{\partial x}}$: partial derivative of transformation $\phi^\theta(x,t)$ w.r.t. x.

The closed-form derivative $\partial\phi^\theta(x,t)/\partial\theta$ was presented in (Martinez et al., 2022). In addition, a normalizing flow model involves computing the Jacobian determinant, so under gradient-based optimization techniques, second order derivatives are also required:

- $\boxed{\frac{\partial^2\phi^\theta(x,t)}{\partial x^2}}$: partial second derivative of transformation $\phi^\theta(x,t)$ w.r.t. x.
- $\boxed{\frac{\partial^2\phi^\theta(x,t)}{\partial x\partial\theta}}$: partial second derivative of transformation $\phi^\theta(x,t)$ w.r.t. x and θ.

3.2 Closed-Form Derivatives of $\phi^\theta(x,t)$ w.r.t. x

The derivative can be calculated by going backwards in the integration direction. We focus on the partial derivative w.r.t. the spatial dimension x, and derive each of the terms of this derivative:

$$\boxed{\frac{\partial\phi^\theta(x,t)}{\partial x}} = \left(\frac{\partial\psi^\theta(x,t)}{\partial x} + \frac{\partial\psi^\theta(x,t)}{\partial t^\theta} \cdot \frac{\partial t^\theta}{\partial x}\right)_{\substack{x=x_m \\ t=t_m}} \tag{2}$$

3.2.1 Expression for $\dfrac{\partial \psi^\theta(x,t)}{\partial x}$

Based on the expression for $\psi(x,t)$ from Equation (1), we can obtain the derivative w.r.t. x:

$$\phi^\theta(x,t) = \psi^\theta(x = x_m, t = t_m) = \left(x e^{t a_c} + \left(e^{t a_c} - 1 \right) \frac{b_c}{a_c} \right)_{\substack{x=x_m \\ t=t_m}} \tag{3}$$

It is important to note that the variable x only affects this expression when $m = 1$, otherwise is zero. That is, when the integration process does not traverse to adjacent cells the value of x is present on $\phi^\theta(x,t)$, otherwise is x_m. Here $\mathbb{1}_C$ is the indicator function that takes value 1 when the condition C is true.

$$\frac{\partial \psi^\theta(x,t)}{\partial x} = e^{t a_c} \cdot \mathbb{1}_{m=1} \tag{4}$$

3.2.2 Expression for $\dfrac{\partial \psi^\theta(x,t)}{\partial t^\theta}$

Similarly, from the same expression for $\psi(x,t)$, we explicitly get the derivative w.r.t t^θ:

$$\frac{\partial \psi^\theta(x,t)}{\partial t^\theta} = x\, a_c\, e^{t a_c} + a_c\, e^{t a_c} \frac{b_c}{a_c} = e^{t a_c} \left(a_c x + b_c \right) \tag{5}$$

3.2.3 Expression for $\dfrac{\partial t^\theta}{\partial x}$

After visiting m cells, the integration time t^θ can be expressed as:

$$t^\theta = t_1 - \sum_{i=1}^{m-1} t^\theta_{hit}(c_i, x_i) \tag{6}$$

In this case, the variable x only affects Equation (6) if we move away from the first cell, i.e. $m > 1$, and on the first cell $c = 1$, which results the following expresion:

$$\frac{\partial t^\theta}{\partial x} = - \sum_{i=1}^{m-1} \frac{\partial t^\theta_{hit}(c_i, x_i)}{\partial x} = - \frac{\partial t^\theta_{hit}(c, x)}{\partial x} \cdot \mathbb{1}_{m>1} \tag{7}$$

where

$$t^\theta_{hit}(c, x) = \frac{1}{a_c} \log \left(\frac{a_c x_c + b_c}{a_c x + b_c} \right) \tag{8}$$

and x_c is the boundary for cell index c. Now, apply the chain rule operation to the hitting time $t^\theta_{hit}(c, x)$ expression:

$$\frac{\partial t^\theta_{hit}(c, x)}{\partial x} = \frac{1}{a_c} \frac{-a_c \dfrac{a_c x_c + b_c}{(a_c x + b_c)^2}}{\dfrac{a_c x_c + b_c}{a_c x + b_c}} = \frac{-1}{a_c x + b_c} \tag{9}$$

Then, based on Equation (7), the derivative of the integration time t^θ can be expressed as:

$$\frac{\partial t^\theta}{\partial x} = - \frac{\partial t^\theta_{hit}(c, x)}{\partial x} \cdot \mathbb{1}_{m>1} = \frac{1}{a_c x + b_c} \cdot \mathbb{1}_{m>1} \tag{10}$$

3.2.4 Final Expression for $\boxed{\frac{\partial \phi^\theta(x,t)}{\partial x}}$

Joining all the terms together and evaluating the derivative at $x = x_m$ and $t = t_m$ yields the expression for the partial derivative w.r.t. x:

$$\begin{aligned}\frac{\partial \phi^\theta(x,t)}{\partial x} &= \left(\frac{\partial \psi^\theta(x,t)}{\partial x} + \frac{\partial \psi^\theta(x,t)}{\partial t^\theta} \cdot \frac{\partial t^\theta}{\partial x}\right)_{\substack{x=x_m \\ t=t_m}} = \\ &= e^{t_m a_{c_m}} \cdot \mathbb{1}_{m=1} + e^{t_m a_{c_m}} \frac{a_{c_m} x_m + b_{c_m}}{a_c x + b_c} \cdot \mathbb{1}_{m>1} = \\ &= \begin{cases} e^{t_m a_{c_m}} \dfrac{a_{c_m} x_m + b_{c_m}}{a_c x + b_c} & \text{if} \quad m > 1 \\ e^{t_m a_{c_m}} & \text{otherwise} \end{cases}\end{aligned} \tag{11}$$

3.3 Closed-Form Derivatives of $\dfrac{\partial \phi^\theta(x,t)}{\partial x}$ w.r.t. x

In this case, we start from the previous Equation (11) and derive again w.r.t. x:

$$\boxed{\frac{\partial^2 \phi^\theta(x,t)}{\partial x^2}} = \left(\frac{\partial^2 \psi^\theta(x,t)}{\partial x^2} + \frac{\partial^2 \psi^\theta(x,t)}{\partial x\, \partial t^\theta} \cdot \frac{\partial t^\theta}{\partial x} + \frac{\partial \psi^\theta(x,t)}{\partial t^\theta} \cdot \frac{\partial^2 t^\theta}{\partial x^2}\right)_{\substack{x=x_m \\ t=t_m}} \tag{12}$$

3.3.1 Expression for $\dfrac{\partial^2 \psi^\theta(x,t)}{\partial x^2}$

$$\frac{\partial^2 \psi^\theta(x,t)}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial \psi^\theta(x,t)}{\partial x}\right) = \frac{\partial}{\partial x}\left(e^{t a_c} \cdot \mathbb{1}_{m>1}\right) = 0 \tag{13}$$

3.3.2 Expression for $\dfrac{\partial^2 \psi^\theta(x,t)}{\partial t^\theta \partial x}$

$$\frac{\partial^2 \psi^\theta(x,t)}{\partial t^\theta \partial x} = \frac{\partial}{\partial x}\left(\frac{\partial \psi^\theta(x,t)}{\partial t^\theta}\right) = \frac{\partial}{\partial x}\left(e^{t a_c}\left(a_c x + b_c\right)\right) = e^{t a_c} a_c \tag{14}$$

3.3.3 Expression for $\dfrac{\partial^2 t^\theta}{\partial x^2}$

Again, in this case the variable x only affects Equation (6) if we move away from the first cell, i.e. $m > 1$, and on the first cell $c = 1$:

$$\frac{\partial^2 t^\theta}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial t^\theta}{\partial x}\right) = \frac{\partial}{\partial x}\frac{1}{a_c x + b_c} \cdot \mathbb{1}_{m>1} = -\frac{a_c}{(a_c x + b_c)^2} \cdot \mathbb{1}_{m>1} \tag{15}$$

3.3.4 Final Expression for $\dfrac{\partial^2 \phi^\theta(x,t)}{\partial x^2}$

Joining all the terms together and evaluating the derivative at $x = x_m$ and $t = t_m$ yields the expression for the partial derivative w.r.t. x:

$$\begin{aligned}\frac{\partial^2 \phi^\theta(x,t)}{\partial x^2} &= \left(\frac{\partial^2 \psi^\theta(x,t)}{\partial x^2} + \frac{\partial^2 \psi^\theta(x,t)}{\partial t^\theta \partial x} \cdot \frac{\partial t^\theta}{\partial x} + \frac{\partial \psi^\theta(x,t)}{\partial t^\theta} \cdot \frac{\partial^2 t^\theta}{\partial x^2}\right)_{\substack{x=x_m \\ t=t_m}} = \\ &= e^{t_m a_{c_m}} a_{c_m} \frac{1}{a_c x + b_c} \cdot \mathbb{1}_{m>1} - e^{t_m a_{c_m}} \left(a_{c_m} x_m + b_{c_m}\right) \frac{a_c}{(a_c x + b_c)^2} \cdot \mathbb{1}_{m>1}\end{aligned} \tag{16}$$

3.4 Closed-Form Derivatives of $\dfrac{\partial \phi^\theta(x,t)}{\partial x}$ w.r.t. θ

We start from Equation (11) and derive w.r.t. one of the coefficients of θ, i.e., θ_k:

$$\frac{\partial^2 \phi^\theta(x,t)}{\partial \theta_k \, \partial x} = \left(\frac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \, \partial x} + \frac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \partial t^\theta} \cdot \frac{\partial t^\theta}{\partial x} + \frac{\partial \psi^\theta(x,t)}{\partial t^\theta} \cdot \frac{\partial^2 t^\theta}{\partial \theta_k \partial x}\right)_{\substack{x=x_m \\ t=t_m}} \tag{17}$$

Note that the slope a_c and intercept b_c are a linear combination of the orthogonal basis B of the constraint matrix L, with θ as coefficients.

$$vec(\mathbf{A}) = \mathbf{B} \cdot \boldsymbol{\theta} = \sum_{j=1}^{d} \theta_j \cdot \mathbf{B}_j \tag{18}$$

If we define one of the components from the orthogonal basis $\mathbf{B}_j$ as:

$$\mathbf{B}_j = \begin{bmatrix} a_1^{(j)} & b_1^{(j)} & \cdots & a_c^{(j)} & b_c^{(j)} & \cdots & a_{N_\mathcal{P}}^{(j)} & b_{N_\mathcal{P}}^{(j)} \end{bmatrix}^T \tag{19}$$

Then,

$$vec(\mathbf{A}) = \sum_{j=1}^{d} \theta_j \cdot \mathbf{B}_j = \begin{bmatrix} \cdots & \sum_{j=1}^{d} \theta_j a_c^{(j)} & \sum_{j=1}^{d} \theta_j b_c^{(j)} & \cdots \end{bmatrix}^T \tag{20}$$

Thus, the slope a_c and intercept b_c (parameters of the affine transformation) and their derivatives w.r.t. one of the coefficients of θ, i.e., θ_k, are denoted as follows:

$$a_c = \sum_{j=1}^{d} \theta_j a_c^{(j)} \qquad b_c = \sum_{j=1}^{d} \theta_j b_c^{(j)} \tag{21}$$

$$\frac{\partial a_c}{\partial \theta_k} = a_c^{(k)} \qquad \frac{\partial b_c}{\partial \theta_k} = b_c^{(k)} \tag{22}$$

3.4.1 Expression for $\dfrac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \, \partial x}$

Given that $\dfrac{\partial \psi^\theta(x,t)}{\partial x} = e^{t a_c} \cdot \mathbb{1}_{m=1}$, we derive w.r.t. one of the coefficients of θ, i.e., θ_k:

$$\frac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \, \partial x} = \frac{\partial^2 \psi^\theta(x,t)}{\partial a_c \, \partial x} \cdot \frac{\partial a_c}{\partial \theta_k} = t \cdot e^{t a_c} \cdot \mathbb{1}_{m=1} \cdot a_c^{(k)} \tag{23}$$

3.4.2 Expression for $\frac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \partial t^\theta}$

Given that $\frac{\partial \psi^\theta(x,t)}{\partial t^\theta} = e^{ta_c}\big(a_c x + b_c\big)$, we derive w.r.t. one of the coefficients of θ, i.e., θ_k:

$$\frac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \, \partial t^\theta} = \frac{\partial^2 \psi^\theta(x,t)}{\partial t^\theta \, \partial t^\theta} \cdot \frac{\partial t^\theta}{\partial \theta_k} + \frac{\partial^2 \psi^\theta(x,t)}{\partial a_c \, \partial t^\theta} \cdot \frac{\partial a_c}{\partial \theta_k} + \frac{\partial^2 \psi^\theta(x,t)}{\partial b_c \, \partial t^\theta} \cdot \frac{\partial b_c}{\partial \theta_k} \tag{24}$$

We reutilize $\frac{\partial t^\theta}{\partial \theta_k}$ from (Martinez et al., 2022), and derive each remaining term.

$$\frac{\partial^2 \psi^\theta(x,t)}{\partial t^\theta \, \partial t^\theta} = e^{ta_c}(a_c x + b_c) \cdot a_c \tag{25}$$

$$\frac{\partial^2 \psi^\theta(x,t)}{\partial a_c \, \partial t^\theta} = \frac{\partial}{\partial a_c}\left(e^{ta_c}(a_c x + b_c)\right) = te^{ta_c}(a_c x + b_c) + e^{ta_c} x = e^{ta_c}(t(a_c x + b_c) + x) \tag{26}$$

$$\frac{\partial^2 \psi^\theta(x,t)}{\partial b_c \, \partial t^\theta} = \frac{\partial}{\partial b_c}\left(e^{ta_c}(a_c x + b_c)\right) = e^{ta_c} \tag{27}$$

Therefore, joining the expressions from Equations (25) to (27) into Equation (24):

$$\frac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \, \partial t^\theta} = e^{ta_c}(a_c x + b_c) a_c \cdot \frac{\partial t^\theta}{\partial \theta_k} + e^{ta_c}(t(a_c x + b_c) + x) \cdot a_c^{(k)} + e^{ta_c} \cdot b_c^{(k)} \tag{28}$$

3.4.3 Expression for $\frac{\partial^2 t^\theta}{\partial \theta_k \partial x}$

Given that $\frac{\partial t^\theta}{\partial x} = \frac{1}{a_c x + b_c} \cdot \mathbb{1}_{m>1}$, we derive w.r.t. one of the coefficients of θ, i.e., θ_k:

$$\begin{aligned} \frac{\partial^2 t^\theta}{\partial \theta_k \, \partial x} &= \frac{\partial^2 t^\theta}{\partial a_c \, \partial x} \cdot \frac{\partial a_c}{\partial \theta_k} + \frac{\partial^2 t^\theta}{\partial b_c \, \partial x} \cdot \frac{\partial b_c}{\partial \theta_k} = \\ &= \frac{-x}{(a_c x + b_c)^2} \cdot a_c^{(k)} + \frac{-1}{(a_c x + b_c)^2} \cdot b_c^{(k)} \end{aligned} \tag{29}$$

3.4.4 Final Expression for $\boxed{\frac{\partial^2 \phi^\theta(x,t)}{\partial \theta_k \, \partial x}}$

Joining all the terms together and evaluating the derivative at $x = x_m$ and $t = t_m$ yields the expression for the partial derivative w.r.t. x :

$$\begin{aligned} \frac{\partial^2 \phi^\theta(x,t)}{\partial \theta_k \, \partial x} &= \left(\frac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \, \partial x} + \frac{\partial^2 \psi^\theta(x,t)}{\partial \theta_k \partial t^\theta} \cdot \frac{\partial t^\theta}{\partial x} + \frac{\partial \psi^\theta(x,t)}{\partial t^\theta} \cdot \frac{\partial^2 t^\theta}{\partial \theta_k \partial x} \right)_{\substack{x=x_m \\ t=t_m}} = \\ &= t \cdot e^{ta_c} \cdot \mathbb{1}_{m=1} \cdot a_c^{(k)} + \\ &\quad \left(e^{ta_c}(a_c x + b_c) a_c \cdot \frac{\partial t^\theta}{\partial \theta_k} + e^{ta_c}(t(a_c x + b_c) + x) \cdot a_c^{(k)} + e^{ta_c} \cdot b_c^{(k)} \right) \cdot \frac{1}{a_c x + b_c} \cdot \mathbb{1}_{m>1} + \\ &\quad e^{ta_c}(a_c x + b_c) \cdot \left(\frac{-x}{(a_c x + b_c)^2} \cdot a_c^{(k)} + \frac{-1}{(a_c x + b_c)^2} \cdot b_c^{(k)} \right) \end{aligned} \tag{30}$$

4 Experiments and Results

The performance of DIFW-NF flows were evaluated on both synthetic and real-world datasets for density estimation, and compare it to several alternative autoregressive models and flow based methods. For these experiments, we define the forward and backward operators of the proposed normalizing flow model. The forward (generative) direction samples data $\mathbf{z}$ from a known distribution $p(\mathbf{z})$ and applies a transformation f to generate data samples $\mathbf{x} = f(\mathbf{z})$. Similarly, the backward (normalizing) direction takes real data samples $\mathbf{x} \sim p(\mathbf{x})$ and applies the inverse transformation f^{-1}. Model's loss function is simply the negative log-likelihood.

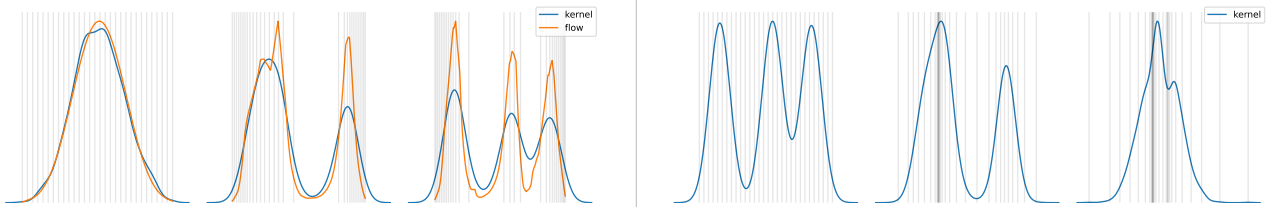
4.1 1-dimensional Data

First, we performed a host of experiments on one-dimensional simulated data to gain in-depth understanding of DIFW-NF. Five datasets are tested: BLOBS generates isotropic one-dimensional Gaussian blobs, GAUSSIAN creates an unimodal Gaussian distribution, GAUSSIANMIX generates multimodal data from a mixture of Gaussians, POWER applies the power transformation 5^x and UNIFORM samples from a uniform 0-1 distribution.

A grid-hyperparameter search is conducted for each dataset. Flows can be composed after 1,2 or 3 steps and the transformation tessellation size is chosen among $\{4, 8, 16, 32\}$. The Adam optimizer (Kingma & Ba, 2014) is used with default hyperparameters and an initial learning rate of $\{1e^{-4}, 1e^{-3}, 5e^{-3}\}$ over 500 training epochs with batch size 256. For training and test, 5000 and 2000 data points are used respectively. The final hyperparameters are shown in Table 1, along with the log-likelihood of the generated data.

Table 1. Hyperparameters and log-probability for density-estimation results in 1D-datasets

Dataset	BLOBS	GAUSSIAN	GAUSSIANMIX	POWER	UNIFORM
Train Size	5000	5000	5000	5000	5000
Test Size	2000	2000	2000	2000	2000
Batch Size	256	256	256	256	256
Tessellation Size	32	32	32	8	32
Flow Steps	2	2	2	3	2
Epochs	500	500	500	500	500
Learning Rate	0.005	0.005	0.001	0.005	0.005
$\log p(\mathbf{x})$	-3.70 $\pm$ 0.22	-1.49 $\pm$ 0.08	-1.22 $\pm$ 0.14	0.80 $\pm$ 0.02	0.01 $\pm$ 0.01



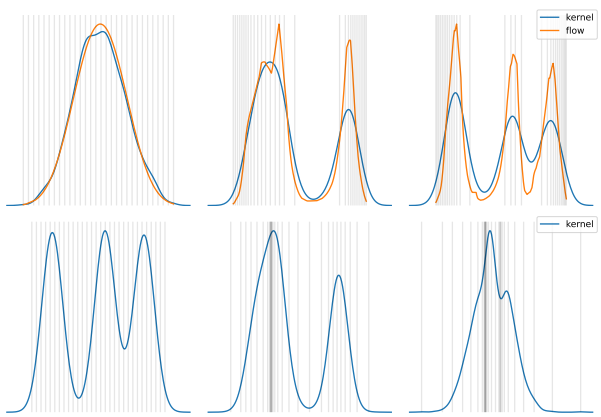
(a) Generative flow: noise ⇒ data

(b) Normalizing flow: data ⇒ noise

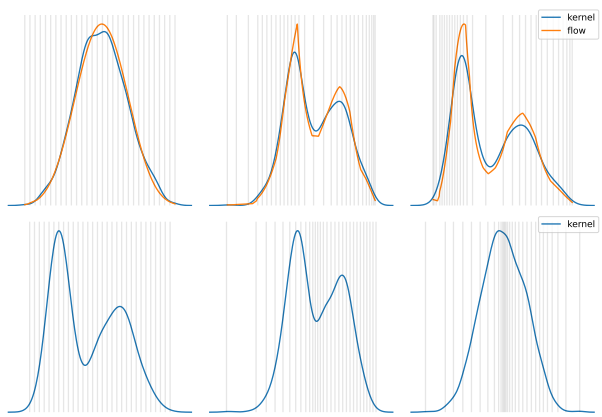
Figure 3. DIFW-NF model applied to 1D BLOBS dataset: kernel probability density estimate in blue and inferred probability using the change-of-variable formula in orange.

Figure 3 shows a detailed example of a one-dimensional flow using the BLOBS dataset. Figure 3a shows the generative direction, in which the function $f(\mathbf{z})$ maps samples $\mathbf{z}$ from the latent distribution into approximate samples $\mathbf{x}$ from the data distribution. This corresponds to exact generation of samples from the model. Figure 3b shows the normalizing direction, in which the inverse function $f^{-1}(\mathbf{x})$ maps samples $\mathbf{x}$ from the data distribution into approximate samples $\mathbf{z}$ from the latent distribution. This corresponds to the exact inference of the latent state given the data.

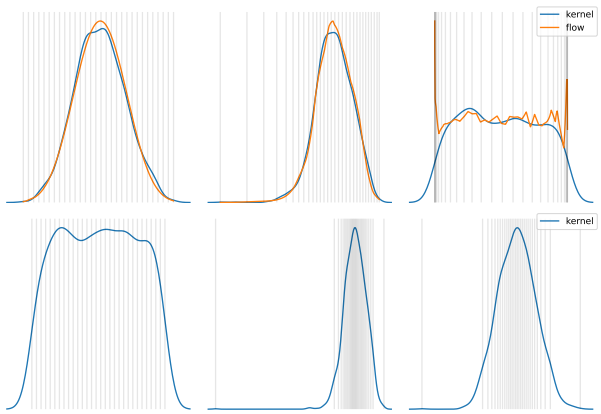
More results on other one-dimensional datasets are presented in Figure 4. The visual comparison between the empirical kernel density function (blue) and the probability density value inferred using the change-of-variable formula orange indicates that the proposed model is flexible enough to appropriately capture the target data distribution. Based on these results we can proceed to 2D and high-dimensional data.



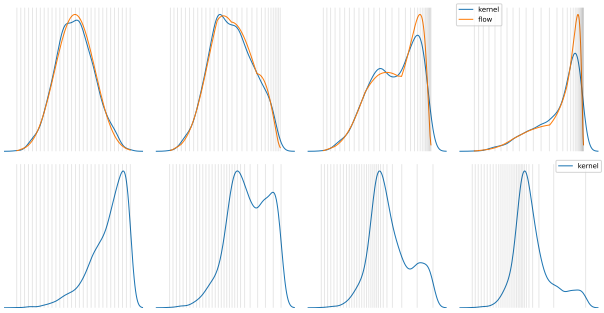
(a) BLOBS dataset



(b) GAUSSIANMIX dataset



(c) UNIFORM dataset



(d) POWER dataset

Figure 4. 1D normalizing flows on multiple datasets. **Top**: Generative flow from Gaussian distribution to data. Kernel probability density estimate in blue and inferred probability using the change-of-variable formula in orange. **Bottom**: Normalizing flow from data back to the Gaussian distribution.

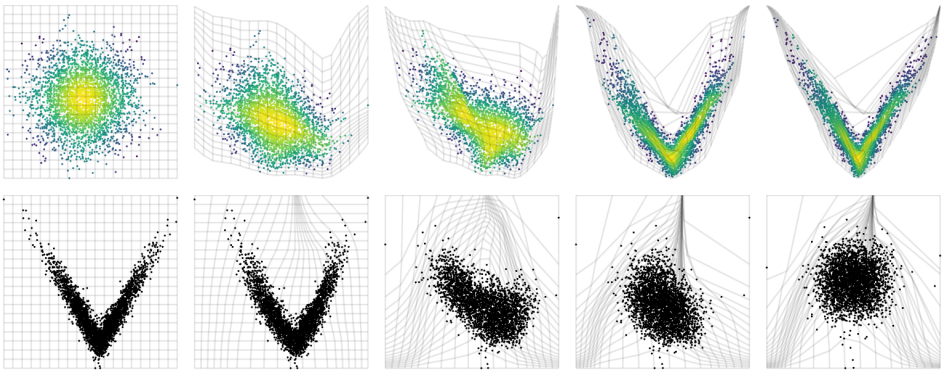
4.2 2-dimensional Data

In this section the flexibility of DIFW-NF is demonstrated on synthetic two-dimensional datasets. A fully-connected neural network with ReLu activation function computes the parameters of the element-wise transformations. A grid-hyperparameter search is conducted for each dataset. Flows can be composed after 1,2,3,4 or 8 steps and the transformation tessellation size is chosen among $\{4, 8, 16, 32\}$. In this case, a grid-search over network architectures was also performed; we searched over models with 1,2 or 4 layers with 8 or 16 hidden layers per flow. The Adam optimizer (Kingma & Ba,

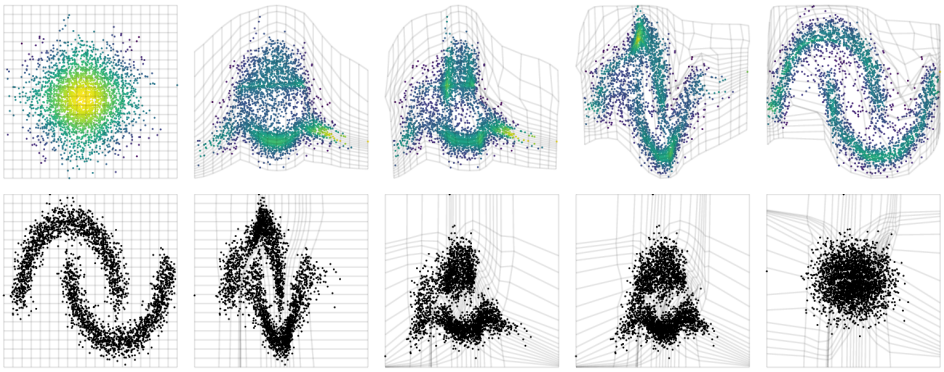
2014) is used with default hyperparameters and an initial learning rate of $1e^{-4}$ over 500 training epochs with batch size 256. For training and test, 5000 and 2000 data points are used respectively. The final hyperparameters are shown in Table 2, along with the log-likelihood of the generated data.

Table 2. Hyperparameters and log-probability for density-estimation results in 2D-datasets

Dataset	# Neurons per Layer	# Hidden Layers	Tessellation Size $N_{\mathcal{P}}$	Flow Steps	$\log p(\mathbf{x})$
ABS	16	1	32	2	-1.15 $\pm$ 0.06
CHECKERBOARD	16	4	32	1	-3.71 $\pm$ 0.06
CIRCLES	16	2	32	1	-1.24 $\pm$ 0.06
CRESCENT	16	2	4	1	-1.79 $\pm$ 0.06
CRESCENTCUBED	8	4	4	1	-1.75 $\pm$ 0.09
DIAMOND	16	4	32	1	-3.40 $\pm$ 0.06
FOURCIRCLES	16	4	8	2	-2.90 $\pm$ 0.06
MOONS	16	4	32	2	-1.10 $\pm$ 0.06
SIGN	16	2	32	3	-1.43 $\pm$ 0.07
SINEWAVE	16	2	32	3	-1.96 $\pm$ 0.05
TWOSPIRALS	16	2	4	8	-2.81 $\pm$ 0.05

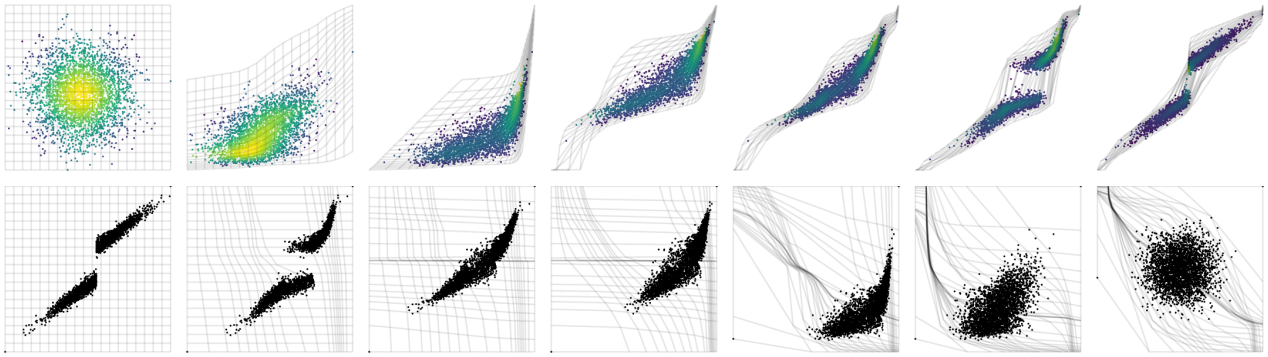


(a) ABS dataset

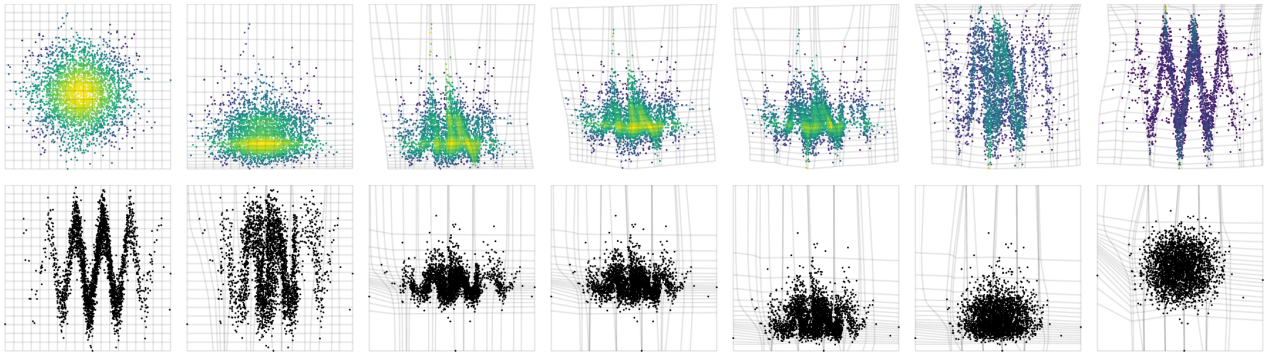


(b) MOONS dataset

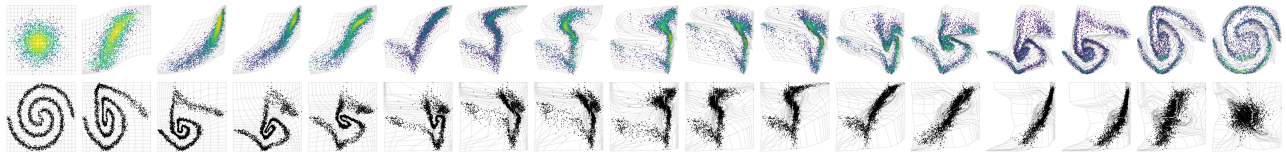
Figure 5. Density estimation for two-dimensional synthetic datasets, including multi-modal and discontinuous densities. Flow transforms samples from a standard-normal base density to the target density. The top row shows the generative direction, visualizing the transformation from noise to data: $\mathbf{z} \sim p(\mathbf{z}) \rightarrow \mathbf{x} = f(\mathbf{z})$. The color scale represents the probability density function. Normalizing flows are reversible, so one can train on a density estimation task and still be able to sample from the learned density efficiently. The bottom row shows the normalizing direction from data to noise: $\mathbf{x} \sim p(\mathbf{x}) \rightarrow \mathbf{z} = f^{-1}(\mathbf{x})$



(c) SIGN dataset



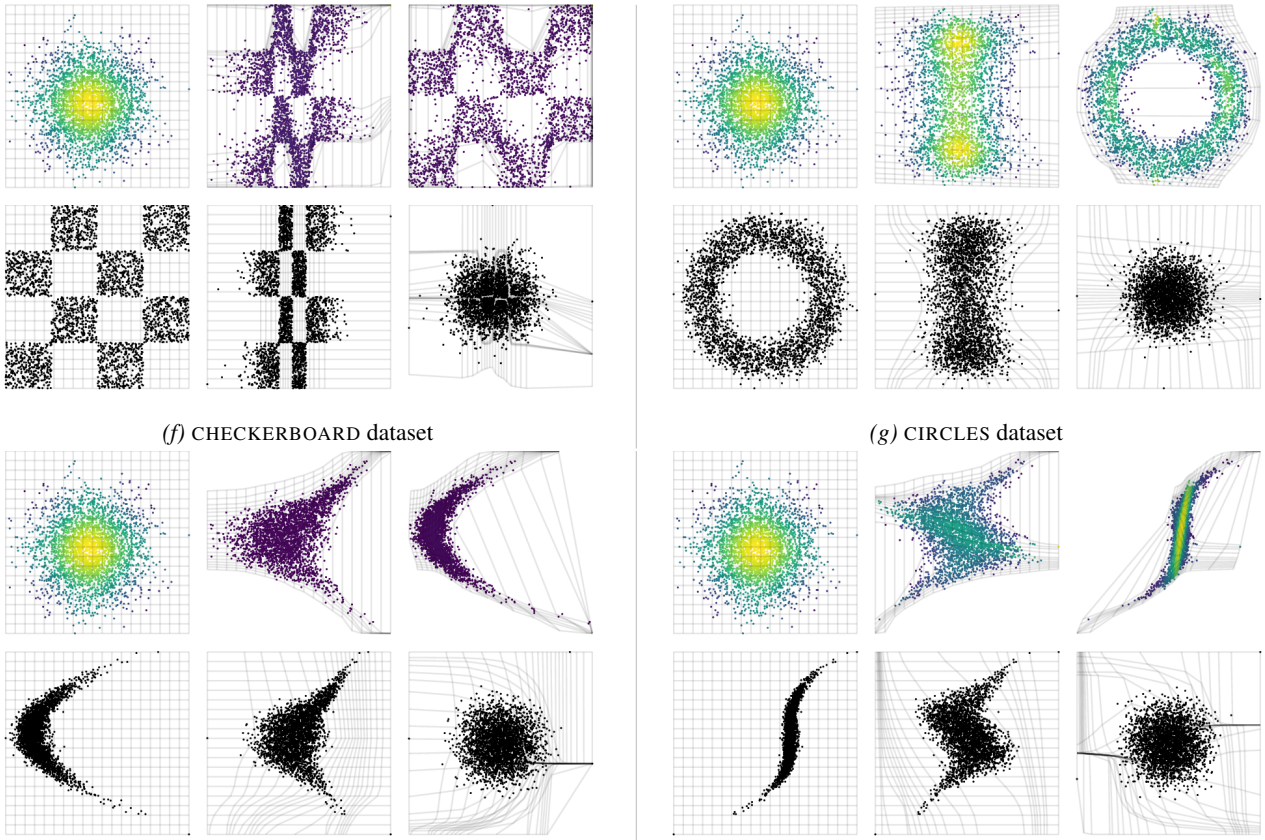
(d) SINEWAVE dataset



(e) TWOSPIRALS dataset

Figure 5. (Cont.) Density estimation for two-dimensional synthetic datasets, including multi-modal and discontinuous densities. Flow transforms samples from a standard-normal base density to the target density. The top row shows the generative direction, visualizing the transformation from noise to data: $\mathbf{z} \sim p(\mathbf{z}) \rightarrow \mathbf{x} = f(\mathbf{z})$. The color scale represents the probability density function. Normalizing flows are reversible, so one can train on a density estimation task and still be able to sample from the learned density efficiently. The bottom row shows the normalizing direction from data to noise: $\mathbf{x} \sim p(\mathbf{x}) \rightarrow \mathbf{z} = f^{-1}(\mathbf{x})$.

Figure 5 shows qualitative results for two-dimensional synthetic datasets. The comparison of the generated points with the original data shows that the normalizing flows can accurately replicate multi-modal and discontinuous distributions. DIFW-NF is able to achieve this with a limited number of flow layers and small neural network architecture, i.e., small number of hidden layers and neurons per layer. It is worth noting that the TWOSPIRALS dataset, which has a complex and non-linear distribution, requires more than three flow steps to generate accurate results. Considering other models, comparisons are avoided in this instance because these datasets are small, simple and not representative of real-world data. Models that perform well on toy datasets may not necessarily perform well on larger, more complex datasets. Hence, the next section focuses on experiments on N-dimensional real-world data.



(f) CHECKERBOARD dataset

(g) CIRCLES dataset

(h) CRESCENT dataset

(i) CRESCENTCUBED dataset

Figure 5. (Cont.) Density estimation for two-dimensional synthetic datasets, including multi-modal and discontinuous densities. Flow transforms samples from a standard-normal base density to the target density. The top row shows the generative direction, visualizing the transformation from noise to data: $\mathbf{z} \sim p(\mathbf{z}) \rightarrow \mathbf{x} = f(\mathbf{z})$. The color scale represents the probability density function. Normalizing flows are reversible, so one can train on a density estimation task and still be able to sample from the learned density efficiently. The bottom row shows the normalizing direction from data to noise: $\mathbf{x} \sim p(\mathbf{x}) \rightarrow \mathbf{z} = f^{-1}(\mathbf{x})$

4.3 N-dimensional Data

We evaluate our proposed flows using a selection of datasets from the UCI machine-learning repository (Asuncion & Newman, 2007) and BSDS300 collection of natural images (Martin et al., 2001). This is the standard suite of benchmarks for density estimation of tabular data. The experimental setup and pre-processing of (Papamakarios et al., 2017) is followed, who make their data available online.

Model selection is performed using the standard validation splits for these datasets. The norm of gradients is clipped to the range $[-5, 5]$, and find this helps stabilize training. In this case a fully-connected neural network of 4 layers with 64 neurons per layer with ReLu activation function was chosen to compute the parameters of the element-wise transformations. A grid-search is not carried out to optimize the neural network architecture.

On the contrary, a grid-hyperparameter search is conducted for the transformation of each dataset. Flows can be composed after 3,5,8,10,15 or 20 steps and the transformation tessellation size is chosen among $\{5, 10, 20, 50\}$. In addition, a grid-search over network architectures was also performed; we searched over models with 1,2 or 4 layers with 8 or 16 hidden layers per flow. The Adam optimizer (Kingma & Ba, 2014) was used with default hyperparameters and an initial learning rate of $3e^{-4}$ or $5e^{-4}$ over 128, 256 or 512 training epochs with batch size 512.

Hyperparameter settings are shown for coupling flows in Table 3. The dimensionality and number of training data points are included in each table for reference.

Table 3. Hyperparameters and log-probability for density-estimation results in ND-datasets

Dataset	BSDS300	GAS	HEPMASS	MINIBOONE	POWER
Train Points	1000000	852174	315123	29556	1615917
Dimension	63	8	21	43	6
Batch Size	512	512	512	512	512
# Neurons per Layer	64	64	64	64	64
# Hidden Layers	4	4	4	4	4
Tessellation Size	10	10	10	10	50
Flow Steps	15	15	10	10	3
Epochs	256	256	256	512	128
Learning Rate	0.0005	0.0005	0.0005	0.0003	0.0005

Table 4. Test log-likelihood (in nats) for UCI datasets and BSDS300, with error bars corresponding to two standard deviations. Higher is better.

Dataset	BSDS300	GAS	HEPMASS	MINIBOONE	POWER
DIFW (AR)	155.95 $\pm$ 0.39	11.60 $\pm$ 0.02	-14.18 $\pm$ 0.04	-9.06 $\pm$ 0.07	0.45 $\pm$ 0.01
DIFW (CL)	**160.25** $\pm$ 0.40	10.79 $\pm$ 0.03	-15.32 $\pm$ 0.03	**-8.02** $\pm$ 0.06	0.30 $\pm$ 0.01
BLOCK-NAF	157.36 $\pm$ 0.03	12.06 $\pm$ 0.09	-14.71 $\pm$ 0.38	-8.95 $\pm$ 0.07	0.61 $\pm$ 0.01
FFJORD	157.40 $\pm$ 0.19	8.59 $\pm$ 0.12	-14.92 $\pm$ 0.08	-10.43 $\pm$ 0.04	0.46 $\pm$ 0.01
GLOW	156.95 $\pm$ 0.28	12.24 $\pm$ 0.03	-16.99 $\pm$ 0.02	-10.55 $\pm$ 0.45	0.52 $\pm$ 0.01
MAF	156.95 $\pm$ 0.28	12.35 $\pm$ 0.02	-17.03 $\pm$ 0.02	-10.92 $\pm$ 0.46	0.45 $\pm$ 0.01
NAF	157.73 $\pm$ 0.04	11.96 $\pm$ 0.33	-15.09 $\pm$ 0.40	-8.86 $\pm$ 0.15	0.62 $\pm$ 0.01
Q-NSF (AR)	157.42 $\pm$ 0.28	12.91 $\pm$ 0.02	-14.67 $\pm$ 0.03	-9.72 $\pm$ 0.47	**0.66** $\pm$ 0.01
Q-NSF (C)	157.65 $\pm$ 0.28	12.80 $\pm$ 0.02	-15.35 $\pm$ 0.02	-9.35 $\pm$ 0.44	0.64 $\pm$ 0.01
RQ-NSF (AR)	157.31 $\pm$ 0.28	**13.09** $\pm$ 0.02	**-14.01** $\pm$ 0.03	-9.22 $\pm$ 0.48	**0.66** $\pm$ 0.01
RQ-NSF (C)	157.54 $\pm$ 0.28	**13.09** $\pm$ 0.02	-14.75 $\pm$ 0.03	-9.67 $\pm$ 0.47	0.64 $\pm$ 0.01
SOS	157.48 $\pm$ 0.41	11.99 $\pm$ 0.41	-15.15 $\pm$ 0.10	-8.90 $\pm$ 0.11	0.60 $\pm$ 0.01

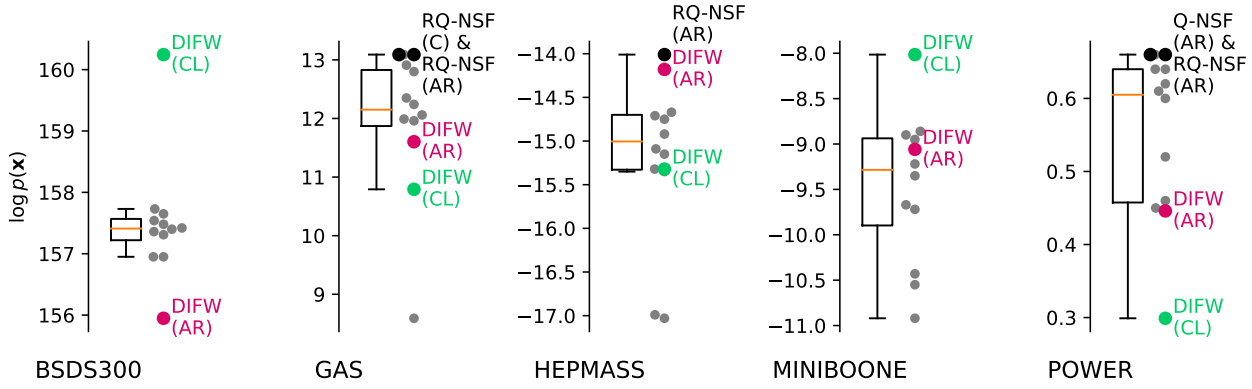


Figure 6. Boxplot visualization. Each point represents the test log-likelihood (in nats) of each model for UCI datasets and BSDS300, higher is better.

Our results are shown in Table 4 and Figure 6. DIFW (CL) achieve state-of-the-art results for BSDS300 and MINIBOONE, while Both RQ-NSF (C) and RQ-NSF (AR) do so on the POWER, GAS, and HEPMASS datasets, tied with Q-NSF (AR) on

[0]The average log-likelihood (relative entropy) is usually reported in units of nats. When one uses the natural logarithm to compute entropy, it takes on the "natural units of information", or nats.

the POWER dataset. Moreover, DIFW (CL) and DIFW (AR) achieve competitive scores with the best autoregressive models (Glow and FFJORD). These results close the gap between autoregressive flows and flows based on coupling layers, and demonstrate that, in some cases, it may not be necessary to sacrifice one-pass sampling for density-estimation performance.

5 Conclusions

On the whole, DIFW-NF shows that upgrading the commonly-used affine transformations in coupling and autoregressive layers can significantly improve performance without compromising analytic invertibility. Through the use of monotonic transforms based on the integration of continuous piecewise-affine velocity functions, coupling-layer-based models can achieve density-estimation performance on par with the best autoregressive flows while maintaining exact one-pass sampling. These models find a unique compromise between versatility and flexibility, serving as a powerful off-the-shelf tool for enhancing architectures like the variational autoencoder and boosting parameter efficiency in generative modeling.

The proposed transforms scale to high-dimensional problems, as demonstrated empirically in Section 4. Moreover, due to the increased flexibility of the proposed transformations, it requires fewer steps to build flexible flows, reducing the computational cost. A potential drawback of the proposed method is a more involved implementation. This is alleviated by providing an extensive Section 3 with technical details, and a reference implementation in PyTorch.

The proposed transformations are also a useful differentiable and invertible module in their own right, which could be included in many models that can be trained end-to-end. For instance, monotonic warping functions with a tractable Jacobian determinant are useful for supervised learning. More generally, invertibility can be useful for training very large networks, since activations can be recomputed on-the-fly for backpropagation, meaning gradient computation requires memory which is constant instead of linear in the depth of the network. Monotonic transforms based on the integration of continuous piecewise-affine velocity functions are one way of constructing invertible element-wise transformations, but there may be others. The benefits of research in this direction are clear, and so we look forward to future work in this area.

References

Asuncion, A. and Newman, D. Uci machine learning repository, 2007.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

De Cao, N., Aziz, W., and Titov, I. Block neural autoregressive flow. In *Uncertainty in artificial intelligence*, pp. 1263–1273. PMLR, 2020.

Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Cubic-spline flows. *arXiv preprint arXiv:1906.02145*, 2019a.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. *Advances in neural information processing systems*, 32, 2019b.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2014.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–13, 2019.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pp. 2722–2730. PMLR, 2019.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. In *International Conference on Machine Learning*, pp. 2078–2087. PMLR, 2018.

Jaini, P., Selby, K. A., and Yu, Y. Sum-of-squares polynomial flow. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:5335–5344, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

LeCun, Y. and Huang, F. J. Loss functions for discriminative training of energy-based models. In *International workshop on artificial intelligence and statistics*, pp. 206–213. PMLR, 2005.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pp. 416–423. IEEE, 2001.

Martinez, I., Viles, E., and Olaizola, I. G. Closed-form diffeomorphic transformations for time series alignment. In *International Conference on Machine Learning*, pp. 15122–15158. PMLR, 2022.

Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. Neural importance sampling. *ACM Transactions on Graphics (TOG)*, 38(5):1–19, 2019.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22:1–64, 2021. ISSN 15337928.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Ziegler, Z. and Rush, A. Latent normalizing flows for discrete sequences. In *International Conference on Machine Learning*, pp. 7673–7682. PMLR, 2019.