Neural Autoregressive Flows

Chin-Wei Huang [1 2 *] **David Krueger** [1 2 *] **Alexandre Lacoste** [2] **Aaron Courville** [1 3]

Abstract

Normalizing flows and autoregressive models have been successfully combined to produce state-of-the-art results in density estimation, via Masked Autoregressive Flows (MAF) (Papamakarios et al., 2017), and to accelerate state-of-the-art WaveNet-based speech synthesis to 20x faster than real-time (Oord et al., 2017), via Inverse Autoregressive Flows (IAF) (Kingma et al., 2016). We unify and generalize these approaches, replacing the (conditionally) affine univariate transformations of MAF/IAF with a more general class of invertible univariate transformations expressed as monotonic neural networks. We demonstrate that the proposed **neural autoregressive flows (NAF)** are universal approximators for continuous probability distributions, and their greater expressivity allows them to better capture multimodal target distributions. Experimentally, NAF yields state-of-the-art performance on a suite of density estimation tasks and outperforms IAF in variational autoencoders trained on binarized MNIST. [1]

1. Introduction

Invertible transformations with a tractable Jacobian, also known as **normalizing flows**, are useful tools in many machine learning problems, for example: (1) In the context of **deep generative models**, training necessitates evaluating data samples under the model's inverse transformation (Dinh et al., 2016). Tractable density is an appealing property for these models, since it allows the objective of interest to be directly optimized; whereas other mainstream methods rely on alternative losses, in the case of intractable density models (Kingma & Welling, 2013; Rezende et al., 2014), or implicit losses, in the case of adversarial models (Goodfellow et al., 2014). (2) In the context of **variational inference** (Rezende & Mohamed, 2015), they can be used to improve the variational approximation to the posterior by parameterizing more complex distributions. This is important since a poor variational approximation to the posterior can fail to reflect the right amount of *uncertainty*, and/or be biased (Turner & Sahani, 2011), resulting in inaccurate and unreliable predictions. We are thus interested in improving techniques for normalizing flows.

Recent work by Kingma et al. (2016) reinterprets autoregressive models as invertible transformations suitable for constructing normalizing flows. The inverse transformation process, unlike sampling from the autoregressive model, is not sequential and thus can be accelerated via parallel computation. This allows multiple layers of transformations to be stacked, increasing expressiveness for better variational inference (Kingma et al., 2016) or better density estimation for generative models (Papamakarios et al., 2017). Stacking also makes it possible to improve on the sequential conditional factorization assumed by autoregressive models such as PixelRNN or PixelCNN (Oord et al., 2016), and thus define a more flexible joint probability.

We note that the normalizing flow introduced by Kingma et al. (2016) only applies an affine transformation of each scalar random variable. Although this transformation is conditioned on preceding variables, the resulting flow can still be susceptible to bad local minima, and thus failure to capture the multimodal shape of a target density; see Figure 1 and 2.

1.1. Contributions of this work

We propose replacing the conditional affine transformation of Kingma et al. (2016) with a more rich family of transformations, and note the requirements for doing so. We determine that very general transformations, for instance parametrized by deep neural networks, are possible. We then propose and evaluate several specific monotonic neural network architectures which are more suited for learning multimodal distributions. Concretely, our method amounts to using an autoregressive model to output the weights of multiple independent transformer networks, each of which operates on a single random variable, replacing the affine

[*]Equal contribution [1]MILA, University of Montreal [2]Element AI [3]CIFAR fellow. Correspondence to: Chin-Wei Huang <chin-wei.huang@umontreal.ca>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]Implementation can be found at https://github.com/CW-Huang/NAF/

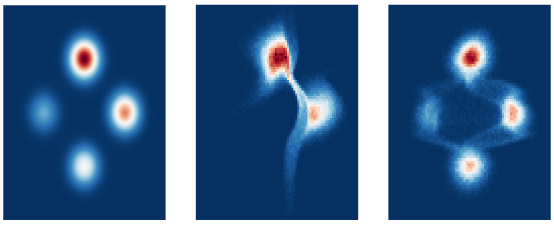


Figure 1. Energy function fitting using IAF.
Left: true distribution. Center: IAF-affine. Right: IAF-DSF.

Figure 2. Density estimation using MAF.
Left: true distribution. Center: MAF-affine. Right: MAF-DSF.

transformations of previous works.

Empirically, we show that our method works better than the state-of-the-art affine autoregressive flows of Kingma et al. (2016) and Papamakarios et al. (2017), both as a sample generator which captures multimodal target densities with higher fidelity, and as a density model which more accurately evaluates the likelihood of data samples drawn from an unknown distribution.

We also demonstrate that our method is a universal approximator on proper distributions in real space, which guarantees the expressiveness of the chosen parameterization and supports our empirical findings.

2. Background

A (finite) **normalizing flow (NF)**, or **flow**, is an invertible function $f_\theta : \mathcal{X} \to \mathcal{Y}$ used to express a transformation between random variables [2]. Since f is invertible, the change of variables formula can be used to translate between densities $p_Y(\mathbf{y})$ and $p_X(\mathbf{x})$:

$$p_Y(\mathbf{y}) = \left| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|^{-1} p_X(\mathbf{x}) \tag{1}$$

The determinant of f's Jacobian appears on the right hand side to account for the way in which f can (locally) expand or contract regions of X, thereby lowering or raising the resulting density in those regions' images in Y. Since the composition of invertible functions is itself invertible, complex NFs are often formed via function composition (or "stacking") of simpler NFs.

Normalizing flows are most commonly trained to produce an output distribution $p_Y(\mathbf{y})$ which matches a target distribution (or, more generally, energy function) $p_{\text{target}}(\mathbf{y})$ as measured by the KL-divergence $KL(p_Y(\mathbf{y})||p_{\text{target}}(\mathbf{y}))$. When X or Y is distributed by some simple distribution, such as uniform or standard normal, we call it an unstructured noise; and we call it a structured noise when the distribution is complex and correlated. Two common settings are maximum likelihood and variational inference. Note that these two settings are typically viewed as optimizing different directions of the KL-divergence, whereas we provide a unified view in terms of different input and target distributions. A detailed derivation is presented in the appendix (See Section A).

For maximum likelihood applications (Dinh et al., 2016; Papamakarios et al., 2017), $p_{\text{target}}(\mathbf{y})$ is typically a simple prior over latent variable $\mathbf{y}$, and f attempts to disentangle the complex empirical distribution of the data, $p_X(\mathbf{x})$ into a simple latent representation $p_Y(\mathbf{y})$ matching the prior (*structured to unstructured*) [3].

In a typical application of variational inference (Rezende & Mohamed, 2015; Kingma et al., 2016), $p_{\text{target}}(\mathbf{y})$ is a complex posterior over latent variables $\mathbf{y}$, and f transforms a simple input distribution (for instance a standard normal distribution) over $\mathbf{x}$ into a complex approximate posterior $p_Y(\mathbf{y})$ (*u*nstructured to structured). In either case, since p_X does not depend on θ, the gradients of the KL-divergence are typically estimated by Monte Carlo:

$$\begin{aligned} &\nabla_\theta \mathcal{D}_{KL}\big(p_Y(\mathbf{y})||p_{\text{target}}(\mathbf{y})\big) \\ &\qquad = \nabla_\theta \int_{\mathcal{Y}} p_Y(\mathbf{y}) \log \frac{p_Y(\mathbf{y})}{p_{\text{target}}(\mathbf{y})} \mathrm{d}\,\mathbf{y} \\ &\qquad = \int_{\mathcal{X}} p_X(\mathbf{x}) \nabla_\theta \log \frac{p_Y(\mathbf{y})}{p_{\text{target}}(\mathbf{y})} \mathrm{d}\,\mathbf{x} \end{aligned} \tag{2}$$

Applying the change of variables formula from Equation 1 to the right hand side of Equation 2 yields:

$$\mathbb{E}_{\substack{\mathbf{x}\sim p_X(\mathbf{x}) \\ \mathbf{y}=f_\theta(\mathbf{x})}} \left[\nabla_\theta \log \left| \frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} \right|^{-1} p_X(\mathbf{x}) - \nabla_\theta \log p_{\text{target}}(\mathbf{y}) \right] \tag{3}$$

[2] We use $\mathbf{x}$ and $\mathbf{y}$ to denote inputs and outputs of a function, *not* the inputs and targets of a supervised learning problem.

[3] It may also be possible to form a generative model from such a flow, by passing samples from the prior $p_{\text{target}}(\mathbf{y})$ through f^{-1}, although the cost of doing so may vary. For example, RealNVP (Dinh et al., 2016) was devised as a generative model, and its inverse computation is as cheap as its forward computation, whereas MAF (Papamakarios et al., 2017) is designed for density estimation and is much more expensive to sample from. For the NAF architectures we employ, we do not have an analytic expression for f^{-1}, but it is possible to approximate it numerically.

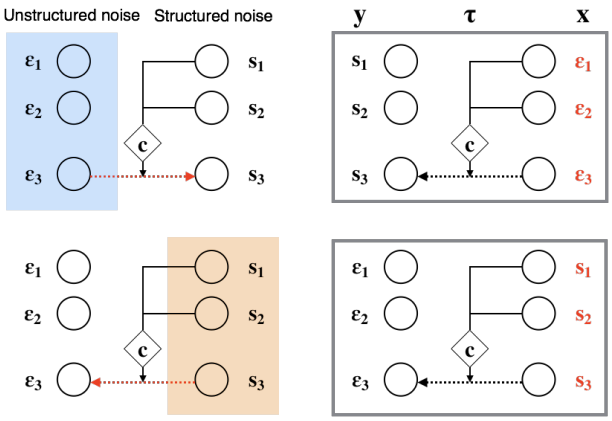


Figure 3. Difference between autoregressive and inverse autoregressive transformations (left), and between IAF and MAF (right). **Upper left**: sample generation of an autoregressive model. Unstructured noise is transformed into structured noise. **Lower left**: inverse autoregressive transformation of structured data. Structured variables are transformed into unstructured variables. **Upper right**: IAF-style sampling. **Lower right**: MAF-style evaluation of structured data. ϵ represents unstructured noise and s represents structured noise.

Thus for efficient training, the following operations must be tractable and cheap:

1. Sampling $\mathbf{x} \sim p_X(\mathbf{x})$
2. Computing $\mathbf{y} = f(\mathbf{x})$
3. Computing the gradient of the log-likelihood of $\mathbf{y} = f(\mathbf{x})$; $\mathbf{x} \sim p_X(\mathbf{x})$ under both $p_Y(\mathbf{y})$ and $p_{\text{target}}(\mathbf{y})$
4. Computing the gradient of the log-determinant of the Jacobian of f

Research on constructing NFs, such as our work, focuses on finding ways to parametrize flows which meet the above requirements while being maximally flexible in terms of the transformations which they can represent. Note that some of the terms of of Equation 3 may be constant with respect to θ [4] and thus trivial to differentiate, such as $p_X(\mathbf{x})$ in the maximum likelihood setting.

Affine autoregressive flows (AAFs) [5], such as inverse autoregressive flows (IAF) (Kingma et al., 2016), are one particularly successful pre-existing approach. Affine autoregressive flows yield a triangular Jacobian matrix, so that the log-determinant can be computed in linear time, as the sum of the diagonal entries on log scale. In AAFs, the components of $\mathbf{x}$ and $\mathbf{y}$ are given an order (which may be chosen arbitrarily), and y_t is computed as a function of $x_{1:t}$. Specifically, this function can be decomposed via an autoregressive **conditioner**, c, and an invertible **transformer**, τ, as [6]:

$$y_t = f(x_{1:t}) = \tau(c(x_{1:t-1}), x_t) \tag{4}$$

It is possible to efficiently compute the output of c for all t in a single forward pass using a model such as **MADE** (Germain et al., 2015), as pointed out by Kingma et al. (2016).

In previous work, τ is taken to be an affine transformation with parameters $\mu \in \mathbb{R}, \sigma > 0$ output from c. For instance Dinh et al. (2016) use:

$$\tau(\mu, \sigma, x_t) = \mu + \sigma x_t \tag{5}$$

with σ produced by an exponential nonlinearity. Kingma et al. (2016) use:

$$\tau(\mu, \sigma, x_t) = \sigma x_t + (1 - \sigma)\mu \tag{6}$$

with σ produced by a sigmoid nonlinearity. Such transformers are trivially invertible, but their relative simplicity also means that the expressivity of f comes entirely from the complexity of c and from stacking multiple AAFs (potentially using different orderings of the variables) [7]. However, the only requirements on τ are:

1. The transformer τ must be invertible as a function of x_t.
2. $\frac{dy_t}{dx_t}$ must be cheap to compute.

This raises the possibility of using a more powerful transformer in order to increase the expressivity of the flow.

3. Neural Autoregressive Flows

We propose replacing the affine transformer used in previous works with a neural network, yielding a more rich family of distributions with only a minor increase in computation and memory requirements. Specifically,

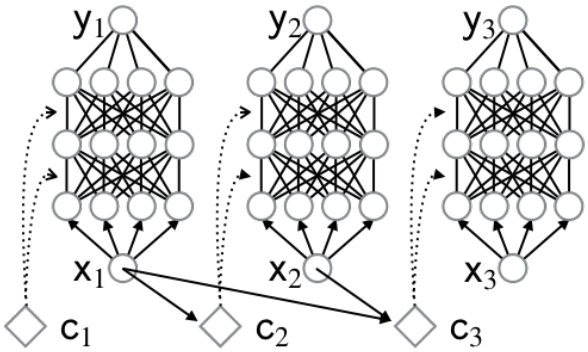
$$\tau(c(x_{1:t-1}), x_t) = \text{DNN}(x_t;\ \phi = c(x_{1:t-1})) \tag{7}$$

[4] There might be some other parameters other than θ that are learnable, such as parameters of p_X and p_{target} in the variational inference and maximum likelihood settings, respectively.

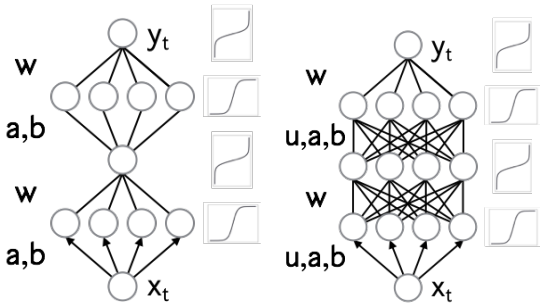
[5] Our terminology differs from previous works, and hence holds the potential for confusion, but we believe it is apt. Under our unifying perspective, NAF, IAF, AF, and MAF all make use of the same principle, which is an invertible transformer conditioned on the outputs of an autoregressive (and emphatically *not* an *inverse* autoregressive) conditioner.

[6] Dinh et al. (2014) use m and g^{-1} to denote c and τ, and refer to them as the "coupling function" and "coupling law", respectively.

[7] Permuting the order of variables is itself a normalizing flow that does not expand or contract space and can be inverted by another permutation.



(a) Neural autoregressive flows (NAF)



(b) DSF (c) DDSF

Figure 4. **Top:** In neural autoregressive flows, the transformation of the current input variable is performed by an MLP whose parameters are output from an autoregressive conditioner model, $c_t \doteq c(x_{1:t-1})$, which incorporates information from previous input variables. **Bottom:** The architectures we use in this work: deep sigmoidal flows (DSF) and deep dense sigmoidal flows (DDSF). See section 3.1 for details.

is a deep neural network which takes the scalar x_t as input and produces y_t as output, and its weights and biases are given by the outputs of $c(x_{1:t-1})$[8] (see Figure 4(a)). We refer to these values ϕ as **pseudo-parameters**, in order to distinguish them from the statistical parameters of the model.

We now state the condition for NAF to be strictly monotonic, and thus invertible (as per requirement 1):

Proposition 1. *Using strictly positive weights and strictly monotonic activation functions for τ_c is sufficient for the entire network to be strictly monotonic.*

Meanwhile, $\frac{dy_t}{dx_t}$ and gradients wrt the pseudo-parameters [9] can all be computed efficiently via backpropagation (as per requirement 2).

[8] We'll sometimes write τ_c for $\tau(c(x_{1:t-1}), \cdot)$.

[9] Gradients for pseudo-parameters are backpropagated through the conditioner, c, in order to train its parameters.

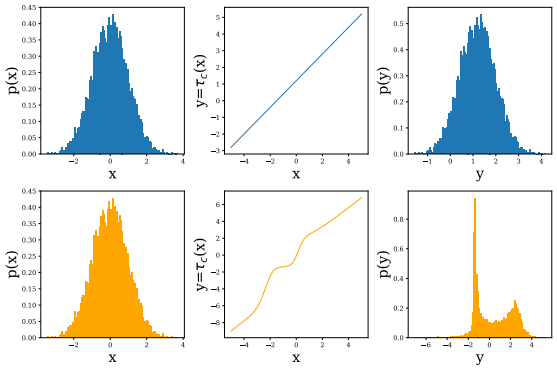


Figure 5. Illustration of the effects of traditional IAF (top), and our proposed NAF (bottom). Areas where the slope of the transformer τ_c is greater/less than 1, are compressed/expanded (respectively) in the output distribution. Inflection points in $\tau_c(x_t)$ (middle) can transform a unimodal $p(x_t)$ (left) into a multimodal $p(y_t)$ (right); NAF allows for such inflection points, whereas IAF does not.

Whereas affine transformers require information about multimodality in y_t to flow through $x_{1:t-1}$, our **neural autoregressive flows (NAFs)** are able to induce multimodality more naturally, via inflection points in τ_c, as shown in Figure 5. Intuitively, τ_c can be viewed as analogous to a cumulative distribution function (CDF), so that its derivative corresponds to a PDF, where its inflection points yield local maxima or minima.

3.1. Transformer Architectures

In this work, we use two specific architectures for τ_c, which we refer to as **deep sigmoidal flows (DSF)** and **deep dense sigmoidal flows (DDSF)** (see Figure 4(b), 4(c) for an illustration). We find that small neural network transformers of 1 or 2 hidden layers with 8 or 16 sigmoid units perform well across our experiments, although there are other possibilities worth exploring (see Section 3.3). Sigmoids contain inflection points, and so can easily induce inflection points in τ_c, and thus multimodality in $p(y_t)$. We begin by describing the DSF transformation, which is already sufficiently expressive to form a universal approximator for probability distributions, as we prove in section 4.

The DSF transformation resembles an MLP with a single hidden layer of sigmoid units. Naive use of sigmoid activation functions would restrict the range of τ_c, however, and result in a model that assigns 0 density to sufficiently large or small y_t, which is problematic when y_t can take on arbitrary real values. We address this issue by applying the inverse sigmoid (or "logit") function at the output layer. To ensure that the output's preactivation is in the domain of the logit (that is, $(0, 1)$), we combine the output of the sigmoid units via an attention-like (Bahdanau et al., 2014)

softmax-weighted sums:

$$y_t = \sigma^{-1}(\underbrace{w^T}_{1\times d} \cdot \sigma(\underbrace{a}_{d\times 1} \cdot \underbrace{x_t}_{1\times 1} + \underbrace{b}_{d\times 1}))) \tag{8}$$

where $0 < w_{i,j} < 1$, $\sum_i w_{i,j} = 1$, $a_{s,t} > 0$, and d denotes the number of hidden units [10].

Since all of the sigmoid activations are bounded between 0 and 1, the final preactivation (which is their convex combination) is as well. The complete DSF transformation can be seen as mapping the original random variable to a different space through an activation function, where doing affine/linear operations is non-linear with respect to the variable in the original space, and then mapping it back to the original space through the inverse activation.

When stacking multiple sigmoidal transformation, we realize it resembles an MLP with bottleneck as shown by the bottom left of Figure 4. A more general alternative is the **deep dense sigmoidal flow (DDSF)**, which takes the form of a fully connected MLP:

$$h^{(l+1)} = \sigma^{-1}(\underbrace{w^{(l+1)}}_{d_{l+1}\times d_{l+1}} \cdot \sigma(\underbrace{a^{(l+1)}}_{d_{l+1}} \odot \underbrace{u^{(l+1)}}_{d_{l+1}\times d_l} \cdot \underbrace{h^{(l)}}_{d_l} + \underbrace{b^{(1+1)}}_{d_{l+1}})) \tag{9}$$

for $1 \leq l \leq L$ where $h_0 = x$ and $y = h_L$; $d_0 = d_L = 1$. We also require $\sum_j w_{ij} = 1$, $\sum_j u_{kj} = 1$ for all i, k, and all parameters except b to be positive.

We use either DSF (Equation 8) or DDSF (Equation 9) to define the transformer function τ in Equation 4. To compute the log-determinant of Jacobian in a numerically stable way, we need to apply log-sum-exp to the chain rule

$$\nabla_x y = \left[\nabla_{h^{(L-1)}} h^{(L)}\right]\left[\nabla_{h^{(L-2)}} h^{(L-1)}\right], \cdots, \left[\nabla_{h^{(0)}} h^{(1)}\right] \tag{10}$$

We elaborate more on the numerical stability in parameterization and computation of logarithmic operations in the supplementary materials.

3.2. Efficient Parametrization of Larger Transformers

Multi-layer NAFs, such as DDSF, require c to output $\mathcal{O}(d^2)$ pseudo-parameters, where d is the number of hidden units in each layer of τ. As this is impractical for large d, we propose parametrizing τ with $\mathcal{O}(d^2)$ statistical parameters, but only $\mathcal{O}(d)$ pseudo-parameters which modulate the computation on a per-unit basis, using a technique such as conditional batch-normalization (CBN) (Dumoulin et al., 2016). Such an approach also makes it possible to use minibatch-style matrix-matrix products for the forward and backwards propagation through the graph of τ_c. In particular, we use a technique similar to *conditional weight normalization (CWN)* (Krueger et al., 2017) in our experiments with DDSF; see appendix for details.

[10] Constraints on the variables are enforced via activation functions; w and a are outputs of a $\mathrm{softmax}$, and $\mathrm{softplus}$ or $\exp$, respectively.

3.3. Possibilities for Alternative Architectures

While the DSF and DDSF architectures performed well in our experiments, there are many alternatives to be explored. One possibility is using other (strictly) monotonic activation functions in τ_c, such as leaky ReLUs (LReLU) (Xu et al., 2015) or ELUs (Clevert et al., 2015). Leaky ReLUs in particular are bijections on $\mathbb{R}$ and so would not require the softmax-weighted summation and activation function inversion tricks discussed in the previous section.

Finally, we emphasize that in general, τ need not be expressed as a neural architecture; it only needs to satisfy the requirements of invertibility and differentiability given at the end of section 2.

4. NAFs are Universal Density Approximators

In this section, we prove that NAFs (specifically DSF) can be used to approximate any probability distribution over real vectors arbitrarily well, given that τ_c has enough hidden units output by generic neural networks with autoregressive conditioning. Ours is the first such result we are aware of for finite normalizing flows.

Our result builds on the work of Huang et al. (2017), who demonstrate the general universal representational capability of inverse autoregressive transformations parameterized by an autoregressive neural network (that transform uniform random variables into any random variables in reals). However, we note that their proposition is weaker than we require, as there are no constraints on the parameterization of the transformer τ, whereas we've constrained τ to have strictly positive weights and monotonic activation functions, to ensure it is invertible throughout training.

The idea of proving the universal approximation theorem for DSF (1) in the IAF direction (which transforms unstructured random variables into structured random variables) resembles the concept of the **inverse transform sampling**: we first draw a sample from a simple distribution, such as uniform distribution, and then pass the sample though DSF. If DSF converges to any inverse conditional CDF, the resulting random variable then converges in distribution to any target random variable as long as the latter has positive continuous probability density everywhere in the reals. (2) For the MAF direction, DSF serves as a solution to the nonlinear independent component analysis problem (Hyvärinen & Pajunen, 1999), which disentangles structured random variables into uniformly and independently distributed ran-

dom variables. (3) Combining the two, we further show that DSF can transform any structured noise variable into a random variable with any desired distribution.

We define the following notation for the pre-logit of the DSF transformation (compare equation 8):

$$\mathcal{S}(x_t, \mathcal{C}(x_{1:t-1})) = \sum_{j=1}^{n} w_j(x_{1:t-1}) \cdot \sigma\left(\frac{x_t - b_j(x_{1:t-1})}{\tau_j(x_{1:t-1})}\right) \tag{11}$$

where $\mathcal{C} = (w_j, b_j, \tau_j)_{j=1}^n$ are functions of $x_{1:1-t}$ parameterized by neural networks. Let b_j be in (r_0, r_1); τ_j be bounded and positive; $\sum_{j=1}^n w_j = 1$ and $w_j > 0$. See Appendix F and G for the proof.

Proposition 2. (DSF universally transforms uniform random variables into any desired random variables) *Let Y be a random vector in $\mathbb{R}^m$ and assume Y has a strictly positive and continuous probability density distribution. Let $X \sim \text{Unif}((0,1)^m)$. Then there exists a sequence of functions $(G_n)_{n\geq 1}$ parameterized by autoregressive neural networks in the following form*

$$G(\mathbf{x})_t = \sigma^{-1}\left(\mathcal{S}\left(x_t; \mathcal{C}_t(x_{1:t-1})\right)\right) \tag{12}$$

where $\mathcal{C}_t = (a_{tj}, b_{tj}, \tau_{tj})_{j=1}^n$ are functions of $x_{1:t-1}$, such that $Y_n \doteq G_n(X)$ converges in distribution to Y.

Proposition 3. (DSF universally transforms any random variables into uniformly distributed random variables) *Let X be a random vector in an open set $\mathcal{U} \subset \mathbb{R}^m$. Assume X has a positive and continuous probability density distribution. Let $Y \sim \text{Unif}((0,1)^m)$. Then there exists a sequence of functions $(H_n)_{n\geq 1}$ parameterized by autoregressive neural networks in the following form*

$$H(\mathbf{x})_t = \mathcal{S}\left(x_t; \mathcal{C}_t(x_{1:t-1})\right) \tag{13}$$

where $\mathcal{C}_t = (a_{tj}, b_{tj}, \tau_{tj})_{j=1}^n$ are functions of $x_{1:t-1}$, such that $Y_n \doteq H_n(X)$ converges in distribution to Y.

Theorem 1. (DSF universally transforms any random variables into any desired random variables) *Let X be a random vector in an open set $\mathcal{U} \subset \mathbb{R}^m$. Let Y be a random vector in $\mathbb{R}^m$. Assume both X and Y have a positive and continuous probability density distribution. Then there exists a sequence of functions $(K_n)_{n\geq 1}$ parameterized by autoregressive neural networks in the following form*

$$K(\mathbf{x})_t = \sigma^{-1}\left(\mathcal{S}\left(x_t; \mathcal{C}_t(x_{1:t-1})\right)\right) \tag{14}$$

where $\mathcal{C}_t = (a_{tj}, b_{tj}, \tau_{tj})_{j=1}^n$ are functions of $x_{1:t-1}$, such that $Y_n \doteq K_n(X)$ converges in distribution to Y.

5. Related work

Neural autoregressive flows are a generalization of the affine autoregressive flows introduced by Kingma et al. (2016) as **inverse autoregressive flows (IAF)** and further developed by Chen et al. (2016) and Papamakarios et al. (2017) as **autoregressive flows (AF)** and **masked autoregressive flows (MAF)**, respectively; for details on their relationship to our work see Sections 2 and 3. While Dinh et al. (2014) draw a particular connection between their **NICE** model and the **Neural Autoregressive Density Estimator (NADE)** (Larochelle & Murray, 2011), (Kingma et al., 2016) were the first to highlight the general approach of using autoregressive models to construct normalizing flows. Chen et al. (2016) and then Papamakarios et al. (2017) subsequently noticed that this same approach could be used efficiently in reverse when the key operation is evaluating, as opposed to sampling from, the flow's learned output density. Our method increases the expressivity of these previous approaches by using a neural net to output pseudo-parameters of another network, thus falling into the hypernetwork framework (Ha et al., 2016; Bertinetto et al., 2016; Brabandere et al., 2016).

There has been a growing interest in normalizing flows (NFs) in the deep learning community, driven by successful applications and structural advantages they have over alternatives. Rippel & Adams (2013), Rezende & Mohamed (2015) and Dinh et al. (2014) first introduced normalizing flows to the deep learning community as density models, variational posteriors and generative models, respectively. In contrast to traditional variational posteriors, NFs can represent a richer family of distributions without requiring approximations (beyond Monte Carlo estimation of the KL-divergence). The NF-based **RealNVP**-style generative models (Dinh et al., 2016; 2014) also have qualitative advantages over alternative approaches. Unlike **generative adversarial networks (GANs)** (Goodfellow et al., 2014) and **varational autoencoders (VAEs)** (Kingma & Welling, 2013; Rezende et al., 2014), computing likelihood is cheap. Unlike autoregressive generative models, such as **pixelCNNs** (Oord et al., 2016), sampling is also cheap. Unfortunately, in practice RealNVP-style models are not currently competitive with autoregressive models in terms of likelihood, perhaps due to the more restricted nature of the transformations they employ.

Several promising recent works expand the capabilities of NFs for generative modeling and density estimation, however. Perhaps the most exciting example is Oord et al. (2017), who propose the **probability density distillation** technique to train an IAF (Kingma et al., 2016) based on the autoregressive **WaveNet** (van den Oord et al., 2016) as a generative model using another pretrained WaveNet model to express the target density, thus overcoming the slow sequential sampling procedure required by the original WaveNet (and characteristic of autoregressive models in general), and reaching super-real-time speeds suitable for production. The previously mentioned MAF technique (Pa-

Table 1. Using DSF to improve variational inference. We report the number of affine IAF with our implementation. We note that the log likelihood reported by Kingma et al. (2016) is 78.88. The average and standard deviation are carried out with 5 trials of experiments with different random seeds.

Model	ELBO	$\log p(x)$
VAE	85.00 ± 0.03	81.66 ± 0.05
IAF-affine	82.25 ± 0.05	80.05 ± 0.04
IAF-DSF	81.92 ± 0.04	79.86 ± 0.01

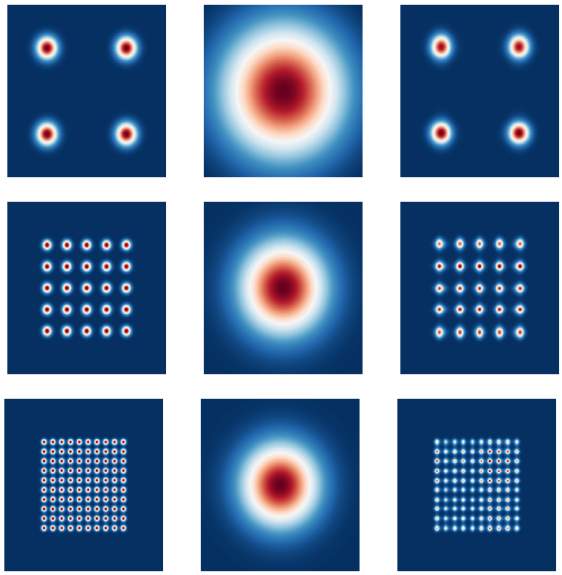


Figure 6. Fitting grid of Gaussian distributions using maximum likelihood. Left: true distribution. Center: affine autoregressive flow (AAF). Right: neural autoregressive flow (NAF)

pamakarios et al., 2017) further demonstrates the potential of NFs to improve on state-of-the-art autoregressive density estimation models; such highly performant MAF models could also be "distilled" for rapid sampling using the same procedure as in Oord et al. (2017).

Other recent works also find novel applications of NFs, demonstrating their broad utility. Loaiza-Ganem et al. (2017) use NFs to solve maximum entropy problems, rather than match a target distribution. Louizos & Welling (2017) and Krueger et al. (2017) apply NFs to express approximate posteriors over parameters of neural networks. Song et al. (2017) use NFs as a proposal distribution in a novel Metropolis-Hastings MCMC algorithm.

Finally, there are also several works which develop new techniques for constructing NFs that are orthogonal to ours (Tomczak & Welling, 2017; 2016; Gemici et al., 2016; Duvenaud et al., 2016; Berg et al., 2018).

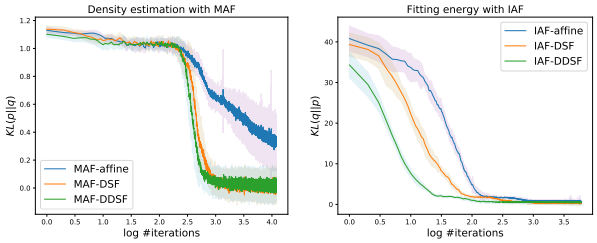


Figure 7. Learning curve of MAF-style and IAF-style training. q denotes our trained model, and p denotes the target.

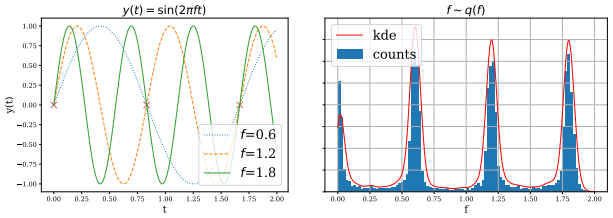


Figure 8. The DSF model effectively captures the true posterior distribution over the frequency of a sine wave. Left: The three observations (marked with red x's) are compatible with sine waves of frequency $f \in 0.0, 0.6, 1.2, 1.8$. Right: a histogram of samples from the DSF approximate posterior ("counts") and a Kernel Density Estimate of the distribution it represents (KDE).

6. Experiments

Our experiments evaluate NAFs on the classic applications of variational inference and density estimation, where we outperform IAF and MAF baselines. We first demonstrate the qualitative advantage NAFs have over AAFs in energy function fitting and density estimation (Section 6.1). We then demonstrate the capability of NAFs to capture a multimodal Bayesian posterior in a limited data setting (Section 6.2). For larger-scale experiments, we show that using NAF instead of IAF to approximate the posterior distribution of latent variables in a variational autoencoder (Kingma & Welling, 2013; Rezende et al., 2014) yields better likelihood results on binarized MNIST (Larochelle & Murray, 2011) (Section 6.3). Finally, we report our experimental results on density estimation of a suite of UCI datasets (Section 6.4).

6.1. Toy energy fitting and density estimation

6.1.1. Expressiveness

First, we demonstrate that, in the case of marginally independent distributions, affine transformation can fail to fit the true distribution. We consider a mixture of Gaussian density estimation task. We define the modes of the Gaussians to be laid out on a 2D meshgrid within the range [-5,5], and consider 2, 5 and 10 modes on each dimension. While the affine flow only produces a single mode, the neural flow

Table 2. Test log-likelihood and error bars of 2 standard deviations on the 5 datasets (5 trials of experiments). Neural autoregressive flows (NAFs) produce state-of-the-art density estimation results on all 5 datasets. The numbers (5 or 10) in parantheses indicate the number of transformations which were stacked; for TAN (Oliva et al., 2018), we include their best results, achieved using different architectures on different datasets. We also include validation results to give future researchers a fair way of comparing their methods with ours during development.

Model	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
MADE MoG	0.40 ± 0.01	8.47 ± 0.02	-15.15 ± 0.02	-12.27 ± 0.47	153.71 ± 0.28
MAF-affine (5)	0.14 ± 0.01	9.07 ± 0.02	-17.70 ± 0.02	-11.75 ± 0.44	155.69 ± 0.28
MAF-affine (10)	0.24 ± 0.01	10.08 ± 0.02	-17.73 ± 0.02	-12.24 ± 0.45	154.93 ± 0.28
MAF-affine MoG (5)	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	-11.68 ± 0.44	156.36 ± 0.28
TAN (various architectures)	0.48 ± 0.01	11.19 ± 0.02	-15.12 ± 0.02	-11.01 ± 0.48	157.03 ± 0.07
MAF-DDSF (5)	$\mathbf{0.62 \pm 0.01}$	11.91 ± 0.13	$\mathbf{-15.09 \pm 0.40}$	$\mathbf{-8.86 \pm 0.15}$	$\mathbf{157.73 \pm 0.04}$
MAF-DDSF (10)	0.60 ± 0.02	$\mathbf{11.96 \pm 0.33}$	-15.32 ± 0.23	-9.01 ± 0.01	157.43 ± 0.30
MAF-DDSF (5) valid	0.63 ± 0.01	11.91 ± 0.13	15.10 ± 0.42	-8.38 ± 0.13	172.89 ± 0.04
MAF-DDSF (10) valid	0.60 ± 0.02	11.95 ± 0.33	15.34 ± 0.24	-8.50 ± 0.03	172.58 ± 0.32

matches the target distribution quite well even up to a 10x10 grid with 100 modes (see Figure 5).

6.1.2. Convergence

We then repeat the experiment that produces Figure 1 and 2 16 times, smooth out the learning curve and present average convergence result of each model with its corresponding standard deviation. For affine flow, we stack 6 layers of transformation with reversed ordering. For DSF and DDSF we used one transformation. We set $d = 16$ for both, $L = 2$ for DDSF.

6.2. Sine Wave experiment

Here we demonstrate the ability of DSF to capture multimodal posterior distributions. To do so, we create a toy experiment where the goal is to infer the posterior over the frequency of a sine wave, given only 3 datapoints. We fix the form of the function as $y(t) = \sin(2\pi f \cdot t)$ and specify a Uniform prior over the frequency: $p(f) = U([0, 2])$. The task is to infer the posterior distribution $p(f|T, Y)$ given the dataset $(T, Y) = ((0, 5/6, 10/6), (0, 0, 0))$, as represented by the red crosses of Figure 8 (left). We assume the data likelihood given the frequency parameter to be $p(y_i|t_i, f) = \mathcal{N}(y_i; y_f(t_i), 0.125)$, where the variance $\sigma^2 = 0.125$ represents the inherent uncertainty of the data. Figure 8 (right) shows that DSF learns a good posterior in this task.

6.3. Amortized Approximate Posterior

We evaluate NAF's ability to improve variational inference, in the context of the binarized MNIST (Larochelle & Murray, 2011) benchmark using the well-known variational autoencoder (Kingma & Welling, 2013; Rezende et al., 2014) (Table 1). Here again the DSF architecture outperforms both standard IAF and the traditional independent Gaussian posterior by a statistically significant margin.

6.4. Density Estimation with Masked Autoregressive Flows

We replicate the density estimation experiments of Papamakarios et al. (2017), which compare MADE (Germain et al., 2015) and RealNVP (Dinh et al., 2016) to their proposed MAF model (using either 5 or 10 layers of MAF) on BSDS300 (Martin et al., 2001) as well as 4 UCI datasets (Lichman, 2013) processed as in Uria et al. (2013). Simply replacing the affine transformer with our DDSF architecture in their best performing architecture for each task (keeping all other settings fixed) results in substantial performance gains, and also outperforms the more recent Transformation Autoregressive Networks (TAN) Oliva et al. (2018), setting a new state-of-the-art for these tasks. Results are presented in Table 2.

7. Conclusion

In this work we introduce the neural autoregressive flow (NAF), a flexible method of tractably approximating rich families of distributions. In particular, our experiments show that NAF is able to model multimodal distributions and outperform related methods such as inverse autoregressive flow in density estimation and variational inference. Our work emphasizes the difficulty and importance of capturing multimodality, as previous methods fail even on simple toy tasks, whereas our method yields significant improvements in performance.

Acknowledgements

We would like to thank Tegan Maharaj, Ahmed Touati, Shawn Tan and Giancarlo Kerg for helpful comments and advice. We also thank George Papamakarios for providing details on density estimation task's setup.

References

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Berg, Rianne van den, Hasenclever, Leonard, Tomczak, Jakub M, and Welling, Max. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Bertinetto, Luca, Henriques, João F., Valmadre, Jack, Torr, Philip H. S., and Vedaldi, Andrea. Learning feed-forward one-shot learners. *CoRR*, 2016.

Brabandere, Bert De, Jia, Xu, Tuytelaars, Tinne, and Gool, Luc Van. Dynamic filter networks. *CoRR*, 2016.

Chen, Xi, Kingma, Diederik P, Salimans, Tim, Duan, Yan, Dhariwal, Prafulla, Schulman, John, Sutskever, Ilya, and Abbeel, Pieter. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Cybenko, George. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 1989.

Dinh, Laurent, Krueger, David, and Bengio, Yoshua. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Dinh, Laurent, Sohl-Dickstein, Jascha, and Bengio, Samy. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Dumoulin, Vincent, Shlens, Jonathon, and Kudlur, Manjunath. A learned representation for artistic style. *CoRR*, 2016.

Duvenaud, David, Maclaurin, Dougal, and Adams, Ryan. Early stopping as nonparametric variational inference. In *Artificial Intelligence and Statistics*, 2016.

Gemici, Mevlana C, Rezende, Danilo, and Mohamed, Shakir. Normalizing flows on riemannian manifolds. *arXiv preprint arXiv:1611.02304*, 2016.

Germain, Mathieu, Gregor, Karol, Murray, Iain, and Larochelle, Hugo. MADE: masked autoencoder for distribution estimation. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in neural information processing systems*, 2014.

Ha, David, Dai, Andrew, and Le, Quoc V. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Huang, Chin-Wei, Touati, Ahmed, Dinh, Laurent, Drozdzal, Michal, Havaei, Mohammad, Charlin, Laurent, and Courville, Aaron. Learnable explicit density for continuous latent space and variational inference. *arXiv preprint arXiv:1710.02248*, 2017.

Hyvärinen, Aapo and Pajunen, Petteri. Nonlinear independent component analysis: Existence and uniqueness results. *Neural Networks*, 1999.

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

Kingma, Diederik. P., Salimans, T., and Welling, M. Variational Dropout and the Local Reparameterization Trick. *arXiv e-prints*, June 2015.

Kingma, Diederik P, Salimans, Tim, Jozefowicz, Rafal, Chen, Xi, Sutskever, Ilya, and Welling, Max. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, 2016.

Krueger, David, Huang, Chin-Wei, Islam, Riashat, Turner, Ryan, Lacoste, Alexandre, and Courville, Aaron. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

Larochelle, Hugo and Murray, Iain. The neural autoregressive distribution estimator. In *The Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15 of *JMLR: W&CP*, 2011.

Lichman, M. UCI machine learning repository, 2013.

Loaiza-Ganem, Gabriel, Gao, Yuanjun, and Cunningham, John P. Maximum entropy flow networks. *arXiv preprint arXiv:1701.03504*, 2017.

Louizos, Christos and Welling, Max. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

Martin, David, Fowlkes, Charless, Tal, Doron, and Malik, Jitendra. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2. IEEE, 2001.

Oliva, J. B., Dubey, A., Póczos, B., Schneider, J., and Xing, E. P. Transformation Autoregressive Networks. *ArXiv e-prints*, January 2018.

Oord, Aaron van den, Kalchbrenner, Nal, and Kavukcuoglu, Koray. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

Oord, Aaron van den, Li, Yazhe, Babuschkin, Igor, Simonyan, Karen, Vinyals, Oriol, Kavukcuoglu, Koray, Driessche, George van den, Lockhart, Edward, Cobo, Luis C, Stimberg, Florian, et al. Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433*, 2017.

Papamakarios, George, Murray, Iain, and Pavlakou, Theo. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, 2017.

Polyak, Boris T and Juditsky, Anatoli B. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 1992.

Reddi, Sashank J, Kale, Satyen, and Kumar, Sanjiv. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

Rezende, Danilo Jimenez and Mohamed, Shakir. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Rippel, Oren and Adams, Ryan Prescott. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

Salimans, Tim and Kingma, Diederik P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, 2016.

Song, Jiaming, Zhao, Shengjia, and Ermon, Stefano. A-nice-mc: Adversarial training for mcmc. In *Advances in Neural Information Processing Systems*, 2017.

Tomczak, Jakub M and Welling, Max. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.

Tomczak, Jakub M and Welling, Max. Improving variational auto-encoders using convex combination linear inverse autoregressive flow. *arXiv preprint arXiv:1706.02326*, 2017.

Turner, Richard E and Sahani, Maneesh. Two problems with variational expectation maximisation for time-series models. *Bayesian Time series models*, 2011.

Uria, Benigno, Murray, Iain, and Larochelle, Hugo. Rnade: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, 2013.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio. *ArXiv e-prints*, September 2016.

Xu, Bing, Wang, Naiyan, Chen, Tianqi, and Li, Mu. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

A. Exclusive KL View of the MLE

Lets assume a change-of-variable model $p_Z(\mathbf{z})$ on the random variable $Z \in \mathbb{R}^m$, such as the one used in Dinh et al. (2016): $\mathbf{z_0} \sim p_0(\mathbf{z_0})$ and $\mathbf{z} = \psi(\mathbf{z_0})$, where ψ is an invertible function and density evaluation of $\mathbf{z_0} \in \mathbb{R}^m$ is tractable under p_0. The resulting density function can be written

$$p_Z(\mathbf{z}) = p_0(\mathbf{z_0}) \left| \frac{\partial \psi(\mathbf{z_0})}{\partial \mathbf{z_0}} \right|^{-1}$$

The maximum likelihood principle requires us to minimize the following metric:

$$\begin{aligned}
&\mathcal{D}_{\mathrm{KL}}\big(p_{data}(\mathbf{z}) || p_Z(\mathbf{z})\big) \\
&\quad = \mathbb{E}_{p_{data}}\left[\log p_{data}(\mathbf{z}) - \log p_Z(\mathbf{z})\right] \\
&\quad = \mathbb{E}_{p_{data}}\left[\log p_{data}(\mathbf{z}) - \log p_0(\mathbf{z_0}) \left| \frac{\partial \psi^{-1}(\mathbf{z})}{\partial \mathbf{z}} \right| \right] \\
&\quad = \mathbb{E}_{p_{data}}\left[\log p_{data}(\mathbf{z}) \left| \frac{\partial \psi^{-1}(\mathbf{z})}{\partial \mathbf{z}} \right|^{-1} - \log p_0(\mathbf{z_0}) \right]
\end{aligned}$$

which coincides with exclusive KL divergence; to see this, we take $X = Z$, $Y = Z_0$, $f = \psi^{-1}$, $p_X = p_{data}$, and $p_{target} = p_0$

$$\begin{aligned}
&= \mathbb{E}_{p_X}\left[\log p_X(\mathbf{x}) \left| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|^{-1} - \log p_{target}(\mathbf{y}) \right] \\
&= \mathcal{D}_{\mathrm{KL}}\left(p_Y(\mathbf{y}) || p_{target}(\mathbf{y})\right)
\end{aligned}$$

This means we want to transform the empirical distribution p_{data}, or p_X, to fit the target density (the usually unstructured, base distribution p_0), as explained in Section 2.

B. Monotonicity of NAF

Here we show that using *strictly positive weights* and *strictly monotonically increasing activation functions* is sufficient to ensure strict monotonicity of NAF. For brevity, we write *monotonic*, or *monotonicity* to represent the strictly monotonically increasing behavior of a function. Also, note that a continuous function is strictly monotonically increasing exactly when its derivative is greater than 0.

Proof. (Proposition 1)

Suppose we have an MLP with $L + 1$ layers: $h_0, h_1, h_2, ..., h_L$, $x = h_0$ and $y = h_L$, where x and y are scalar, and $h_{l,j}$ denotes the j-th node of the l-th layer.For $1 \leq l \leq L$, we have

$$p_{l,j} = w_{l,j}^T h_{l-1} + b_{l,j} \tag{15}$$
$$h_{l,j} = A_l(p_{l,j}) \tag{16}$$

for some monotonic activation function A_l, positive weight vector $w_{l,j}$, and bias $b_{l,j}$. Differentiating this yields

$$\frac{\mathrm{d}h_{l,j}}{\mathrm{d}h_{l-1,k}} = \frac{\mathrm{d}A_l(p_{l,j})}{\mathrm{d}p_{l,j}} \cdot w_{l,j,k}, \tag{17}$$

which is greater than 0 since A is monotonic (and hence has positive derivative), and $w_{l,j,k}$ is positive by construction. Thus any unit of layer l is monotonic with respect to any unit of layer $l-1$, for $1 \leq l \leq L$.

Now, suppose we have J monotonic functions f_j of the input x. Then the weighted sum of these functions $\sum_{j=1}^J u_j f_j$ with $u_j > 0$ is also monotonic with respect to x, since

$$\frac{\mathrm{d}}{\mathrm{d}x} \sum_{j=1}^J u_j f_j = \sum_{j=1}^J u_j \frac{\mathrm{d}f_j}{\mathrm{d}x} > 0 \tag{18}$$

Finally, we use induction to show that all $h_{l,j}$ (including y) are monotonic with respect to x.

1. The base case ($l = 1$) is given by Equation 17.

2. Suppose the inductive hypothesis holds, which means $h_{l,j}$ is monotonic with respect to x for all j of layer l. Then by Equation 18, $h_{l+1,k}$ is also monotonic with respect to x for all k.

Thus by mathematical induction, monotonicity of $h_{l,j}$ holds for all l and j. □

C. Log Determinant of Jacobian

As we mention at the end of Section 3.1, to compute the log-determinant of the Jacobian as part of the objective function, we need to handle the numerical stability. We first derive the Jacobian of the DDSF (note that DSF is a special case of DDSF), and then summarize the numerically stable operations that were utilized in this work.

C.1. Jacobian of DDSF

Again defining $x = h_0$ and $y = h_L$, the Jacobian of each DDSF transformation can be written as a sequence of dot products due to the chain rule:

$$\nabla_x y = \left[\nabla_{h^{(L-1)}} h^{(L)}\right] \left[\nabla_{h^{(L-2)}} h^{(L-1)}\right], \cdots, \left[\nabla_{h^{(0)}} h^{(1)}\right] \tag{19}$$

For notational convenience, we define a few more interme-

diate variables. For each layer of DDSF, we have

$$\begin{aligned}\underbrace{C^{(l+1)}}_{d_{l+1}} &= \underbrace{a^{(l+1)}}_{d_{l+1}} \odot (\underbrace{u^{(l+1)}}_{d_{l+1}\times d_l} \cdot \underbrace{h^{(l)}}_{d_l}) + \underbrace{b^{(1+1)}}_{d_{l+1}} \\ \underbrace{D^{(l+1)}}_{d_{l+1}} &= \underbrace{w^{(l+1)}}_{d_{l+1}\times d_{l+1}} \cdot \sigma(\underbrace{C^{(l+1)}}_{d_{l+1}}) \\ \underbrace{h^{(l+1)}}_{d_{l+1}}) &= \sigma^{-1}(\underbrace{D^{(l+1)}}_{d_{l+1}})\end{aligned}$$

The gradient can be expanded as

$$\begin{aligned}\nabla_{h^{(l)}}\left(h^{(l+1)}\right) = \Bigg(& \nabla_D \left(\sigma^{-1}(D^{(l+1)})\right)_{[:,\bullet]} \odot \\ & \nabla_{\sigma(C)}\left(D^{(l+1)}\right) \odot \\ & \nabla_C \left(\sigma(C^{(l+1)})\right)_{[\bullet,:]} \odot \\ & \nabla_{(u^{(l+1)}h^{(l)})}\left(C^{(l+1)}\right)_{[\bullet,:]}\Bigg)_{[:,:,\bullet]}^{\times_{-1}} \\ & \nabla_{h^{(l)}}\left(u^{(l+1)}h^{(l)}\right)_{[\bullet,:,:]}\end{aligned}$$

where the bullet $\bullet$ in the subscript indicates the dimension is broadcasted, $\odot$ denotes element-wise multiplication, and $\times_{-1}$ denotes summation over the last dimension after element-wise product,

$$\begin{aligned}= \Bigg(& \left(\frac{1}{D^{(l+1)}(1-D^{(l+1)})}\right)_{[:,\bullet]} \odot \\ & \left(w^{(l+1)}\right) \odot \\ & \left(\sigma(C^{(l+1)}) \odot (1-\sigma(C^{(l+1)}))\right)_{[\bullet,:]} \odot \\ & \left(a^{(l+1)}\right)_{[\bullet,:]}\Bigg)_{[:,:,\bullet]}^{\times_{-1}} \\ & \left(u^{(l+1)}\right)_{[\bullet,:,:]}\end{aligned} \tag{20}$$

C.2. Numerically Stable Operations

Since the Jacobian of each DDSF transformation is chain of dot products (Equation 19), with some nested multiplicative operations (Equation 20), we calculate everything in the log-scale (where multiplication is replaced with addition) to avoid unstable gradient signal.

C.2.1. Log Activation

To ensure the summing-to-one and positivity constraints of u and w, we let the autoregressive conditioner output pre-activation $u_{\text{-}}$ and $w_{\text{-}}$, and apply softmax to them. We do the same for a by having the conditioner output $a_{\text{-}}$ and apply softplus to ensure positivity. In Equation 20, we have

$$\begin{aligned}\log w &= \text{logsoftmax}(w_{\text{-}}) \\ \log u &= \text{logsoftmax}(u_{\text{-}}) \\ \log\sigma(C) &= \text{logsigmoid}(C) \\ \log 1-\sigma(C) &= \text{logsigmoid}(-C)\end{aligned}$$

where

$$\begin{aligned}\text{logsoftmax}(x) &= x - \text{logsumexp}(x) \\ \text{logsigmoid}(x) &= -\text{softplus}(-x) \\ \text{logsumexp}(x) &= \log(\sum_i \exp(x_i - x^*)) + x^* \\ \text{softplus}(x) &= \log(1+\exp(x)) + \delta\end{aligned}$$

where $x^* = \max_i\{x_i\}$ and δ is a small value such as 10^{-6}.

C.2.2. Logarithmic dot product

In both Equation 19 and 20, we encounter matrix/tensor product, which is achived by summing over one dimension after doing element-wise multiplication. Let $\tilde{M}_1 = \log M_1$ and $\tilde{M}_2 = \log M_2$ be $d_0 \times d_1$ and $d_1 \times d_2$, respectively. The logarithmic matrix dot product $\star$ can be written as:

$$\tilde{M}_1 \star \tilde{M}_2 = \text{logsumexp}_{\text{dim}=1}\left(\left(\tilde{M}_1\right)_{[:,:,\bullet]} + \left(\tilde{M}_2\right)_{[\bullet,:,:]}\right)$$

where the subscript of logsumexp indicates the dimension (index starting from 0) over which the elements are to be summed up. Note that $\tilde{M}_1 \star \tilde{M}_2 = \log(M_1 \cdot M_2)$.

D. Scalability and Parameter Sharing

As discussed in Section 3.3, a multi-layer NAF such as DDSF requires the autoregressive conditioner c to output many pseudo-parameters, on the order of $\mathcal{O}(Ld^2)$, where L is the number of layers of the transformer network (τ), and d is the average number of hidden units per layer. In practice, we reduce the number of outputs (and thus the computation and memory requirements) of DDSF by instead endowing τ with some learned (*non-conditional*) statistical parameters. Specifically, we decompose $w_{\text{-}}$ and $u_{\text{-}}$ (the preactivations of τ's weights, see section C.2.1) into pseudo-parameters and statistical parameters. Take $u_{\text{-}}$ for example:

$$u^{(l+1)} = \text{softmax}_{\text{dim}=1}(v^{(l+1)} + \eta^{(l+1)}_{[\bullet,:]})$$

where $v^{(l+1)}$ is a $d_{l+1} \times d_l$ matrix of statistical parameters, and η is output by c. See figure D for a depiction.

The linear transformation before applying sigmoid resembles conditional weight normalization (CWN) (Krueger

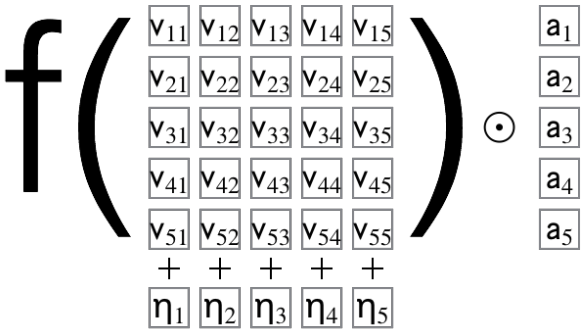


Figure 9. Factorized weight of DDSF. The v_i, j are learned parameters of the model; only the pseudo-parameters η and a are output by the conditioner. The activation function f is softmax, so adding η yields an element-wise rescaling of the inputs to this layer of the transformer by $\exp(\eta)$.

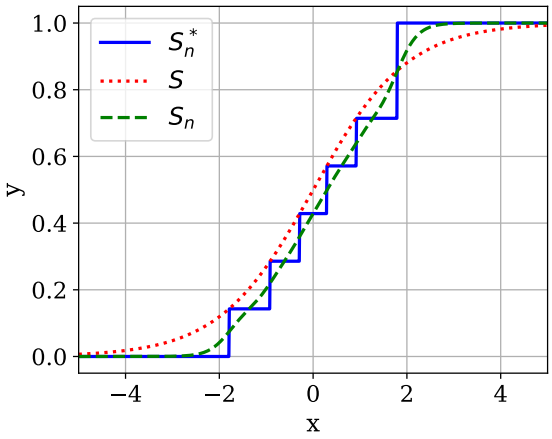


Figure 10. Visualization of how sigmoidal functions can universally approximate an monotonic function in $[0, 1]$. The red dotted curve is the target monotonic function (S), and blue solid curve is the intermediate superposition of step functions (S_n^*) with the parameters chosen in the proof. In this case, n is 6 and $|S_n^* - S| \leq \frac{1}{7}$. The green dashed curve is the resulting superposition of sigmoids (S_n), that intercepts with each step of S_n^* with the additionally chosen τ.

et al., 2017). While CWN rescales the weight vectors normalized to have unit L2 norm, here we rescale the weight vector normalized by softmax such that it sums to one and is positive. We call this *conditional normalized weight exponentiation*. This reduces the number of pseudo-parameters to $\mathcal{O}(Ld)$.

E. Identity Flow Initialization

In many cases, initializing the transformation to have a minimal effect is believed to help with training, as it can be thought of as a warm start with a simpler distribution. For instance, for variational inference, when we initialize the normalizing flow to be an identity flow, the approximate posterior is at least as good as the input distribution (usually a fully factorized Gaussian distribution) before the transformation. To this end, for DSF and DDSF, we initialize the pseudo-weights a to be close to 1, the pseudo-biases b to be close to 0.

This is achieved by initializing the conditioner (whose outputs are the pseudo-parameters) to have small weights and the appropriate output biases. Specifically, we initialize the output biases of the last layer of our MADE (Germain et al., 2015) conditioner to be zero, and add $\text{softplus}^{-1}(1) \approx 0.5413$ to the outputs of which correspond to a before applying the softplus activation function. We initialize all conditioner's weights by sampling from from $\text{Unif}(-0.001, 0.001)$. We note that there might be better ways to initialize the weights to account for the different numbers of active incoming units.

F. Lemmas: Uniform Convergence of DSF

We want to show the convergence result of Equation 11. To this end, we first show that DSF can be used to universally approximate any strictly monotonic function. The is the case where $x_{1:t-1}$ are fixed, which means $\mathcal{C}(x_{1:t-1})$ are simply constants. We demonstrate it using the following two lemmas.

Lemma 1. (Step functions universally approximate monotonic functions) *Define:*

$$S_n^*(x) = \sum_{j=1}^{n} w_j \cdot s\left(x - b_j\right)$$

where $s(z)$ is defined as a step function that is 1 when $z \geq 0$ and 0 otherwise. For any continuous, strictly monotonically increasing $S : [r_0, r_1] \to [0, 1]$ where $S(r_0) = 0$, $S(r_1) = 1$ and $r_0, r_1 \in \mathbb{R}$; and given any $\epsilon > 0$, there exists a positive integer n, real constants w_j and b_j for $j = 1, ..., n$, where $\sum_j^n w_j = 1$, $w_j > 0$ and $b_j \in [r_0, r_1]$ for all j, such that $|S_n^*(x) - S(x)| < \epsilon \quad \forall x \in [r_0, r_1]$.

Proof. (Lemma 1)

For brevity, we write $s_j(x) = s(x - b_j)$. For any $\epsilon > 0$, we choose $n = \lceil \frac{1}{\epsilon} \rceil$, and divide the range $(0, 1)$ into $n + 1$ evenly spaced intervals: $(0, y_1)$, (y_1, y_2), ..., $(y_n, 1)$. For each y_j, there is a corresponding inverse value since S is strictly monotonic, $x_j = S^{-1}(y_j)$. We want to set

$S_n^*(x_j) = y_j$ for $1 \leq j \leq n-1$ and $S_n^*(x_n) = 1$. To do so, we set the bias terms b_j to be x_j. Then we just need to solve a system of n linear equations $\sum_{j'=1}^{n} w_{j'} \cdot s_{j'}(x_j) = t_j$, where $t_j = y_j$ for $1 \leq j < n$, $t_0 = 0$ and $t_n = 1$. We can express this system of equations in the matrix form as $\mathbf{Sw} = \mathbf{t}$, where:

$$\mathbf{S}_{j,j'} = s_{j'}(x_j) = \delta_{x_j \geq b_{j'}} = \delta_{j \geq j'} \,,$$
$$\mathbf{w}_j = w_j \,, \quad \mathbf{t}_j = t_j$$

where $\delta_{\omega \geq \eta} = 1$ whenever $\omega \geq \eta$ and $\delta_{\omega \geq \eta} = 0$ otherwise. Then we have $\mathbf{w} = \mathbf{S}^{-1}\mathbf{t}$. Note that $\mathbf{S}$ is a lower triangular matrix, and its inverse takes the form of a Jordan matrix: $(\mathbf{S}^{-1})_{i,i} = 1$ and $(\mathbf{S}^{-1})_{i+1,i} = -1$. Additionally, $t_j - t_{j-1} = \frac{1}{n+1}$ for $j = 1, ..., n-1$ and is equal to $\frac{2}{n+2}$ for $j = n$. We then have $S_n^*(x) = \mathbf{t}^T \mathbf{S}^{-T} \mathbf{s}(x)$, where $\mathbf{s}(x)_j = s_j(x)$; thus

$$\begin{aligned}
|S_n^*(x) - S(x)| = \; & |\sum_{j=1}^{n} s_j(x)(t_j - t_{j-1}) - S(x)| \\
= \; & |\frac{1}{n+1} \sum_{j=1}^{n-1} s_j(x) + \frac{2 s_n(x)}{n+1} - S(x)| \\
= \; & |\frac{C_{y_{1:n-1}}(x)}{n+1} + \frac{2\delta_{x \geq y_n}}{n+1} - S(x)| \\
\leq \; & \frac{1}{n+1} < \frac{1}{\lceil 1/\epsilon \rceil} \leq \epsilon \qquad (21)
\end{aligned}$$

where $C_v(z) = \sum_k \delta_{z \geq v_k}$ is the count of elements in a vector that z is no smaller than. □

Note that the additional constraint that $\mathbf{w}$ lies on an $n-1$ dimensional simplex is always satisfied, because

$$\sum_j w_j = \sum_j t_j - t_{j-1} = t_n - t_0 = 1$$

See Figure 10 for a visual illustration of the proof. Using this result, we now turn to the case of using sigmoid functions instead of step functions.

Lemma 2. (Superimposed sigmoids universally approximate monotonic functions) *Define:*

$$S_n(x) = \sum_{j=1}^{n} w_j \cdot \sigma\left(\frac{x - b_j}{\tau_j}\right)$$

With the same constraints and definition in Lemma 1, given any $\epsilon > 0$, there exists a positive integer n, real constants w_j, τ_j and b_j for $j = 1, ..., n$, where additionally τ_j are bounded and positive, such that $|S_n(x) - S(x)| < \epsilon \quad \forall x \in (r_0, r_1)$.

Proof. (Lemma 2)

Let $\epsilon_1 = \frac{1}{3}\epsilon$ and $\epsilon_2 = \frac{2}{3}\epsilon$. We know that for this ϵ_1, there exists an n such that $|S_n^* - S| < \epsilon_1$.

We chose the same w_j, b_j for $j = 1, ..., n$ as the ones used in the proof of Lemma 1, and let $\tau_1, ..., \tau_n$ all be the same value denoted by τ.

Take $\kappa = \min_{j \neq j'} |b_j - b_{j'}|$ and $\tau = \frac{\kappa}{\sigma^{-1}(1-\epsilon_0)}$ for some $\epsilon_0 > 0$. Take Γ to be a lower triangular matrix with values of 0.5 on the diagonal and 1 below the diagonal.

$$\begin{aligned}
& \max_{j=1,...,n} |S_n(b_j) - \Gamma_j \cdot w| \\
& = \max \left| \sum_{j'} w_{j'} \sigma\left(\frac{b_j - b_{j'}}{\tau}\right) - \sum_{j'} w_{j'} \Gamma_{jj'} \right| \\
& = \max \left| \sum_{j'} w_{j'} \left(\sigma\left(\frac{b_j - b_{j'}}{\tau}\right) - \Gamma_{jj'} \right) \right| \\
& < \max \sum_{j'} w_{j'} \epsilon_0 = \epsilon_0
\end{aligned}$$

The inequality is due to

$$\sigma\left(\frac{b_j - b_{j'}}{\gamma}\right) = \sigma\left(\frac{b_j - b_{j'}}{\min_{k \neq k'} b_k - b_{k'}} \sigma^{-1}(1-\epsilon_0)\right)$$
$$\begin{cases} = 0.5 & \text{if } j = j' \\ \geq 1 - \epsilon_0 & \text{if } j > j' \\ \leq \epsilon_0 & \text{if } j < j' \end{cases}$$

Since the product $\Gamma \cdot w$ represents the half step points of S_n^* at $x = b_j$'s, the result above entails $|S_n(x) - S_n^*(x)| < \epsilon_2 = 2\epsilon_1$ for all x. To see this, we choose $\epsilon_0 = \frac{1}{2(n+1)}$. Then S_n intercepts with all segments of S_n^* except for the ends. We choose $\epsilon_2 = 2\epsilon_1$ since the last step of S_n^* is of size $\frac{2}{n+1}$, and thus the bound also holds true in the vicinity where S_n intercepts with the last step of S_n^*.

Hence,

$$\begin{aligned}
& |S_n(x) - S(x)| \\
& \quad \leq |S_n(x) - S_n^*(x)| + |S_n^*(x) - S(x)| < \epsilon_1 + \epsilon_2 = \epsilon
\end{aligned}$$

□

Now we show that (the pre-logit) DSF (Equation 11) can universally approximate monotonic functions. We do this by showing that the well-known universal function approximation properties of neural networks (Cybenko, 1989) allow us to produce parameters which are sufficiently close to those required by Lemma 2.

Lemma 3. *Let $x_{1:m} \in [r_0, r_1]^m$ where $r_0, r_1 \in \mathbb{R}$. Given any $\epsilon > 0$ and any multivariate continuously differentiable*

function [11] $S(x_{1:m})_t = S_t(x_t, x_{1:t-1})$ *for* $t \in [1, m]$ *that is strictly monotonic with respect to the first argument when the second argument is fixed, where the boundary values are* $S_t(r_0, x_{1:t-1}) = 0$ *and* $S_t(r_1, x_{1:t-1}) = 1$ *for all* $x_{1:t-1}$ *and* t*, then there exists a multivariate function* $\mathcal{S}$ *such that* $\| \mathcal{S}(x_{1:m}) - S(x_{1:m}) \|_\infty < \epsilon$ *for all* $x_{1:m}$*, of the following form:*

$$\begin{aligned}\mathcal{S}(x_{1:m})_t &= \mathcal{S}_t(x_t, \mathcal{C}_t(x_{1:t-1})) \\ &= \sum_{j=1}^{n} w_{tj}(x_{1:t-1}) \cdot \sigma\left(\frac{x_t - b_{tj}(x_{1:t-1})}{\tau_{tj}(x_{1:t-1})}\right)\end{aligned}$$

where $t \in [1, m]$*, and* $\mathcal{C}_t = (w_{tj}, b_{tj}, \tau_{tj})_{j=1}^n$ *are functions of* $x_{1:1-t}$ *parameterized by neural networks, with* τ_{tj} *bounded and positive,* $b_{tj} \in [r_0, r_1]$*,* $\sum_{j=1}^n w_{tj} = 1$*, and* $w_{tj} > 0$

Proof. (Lemma 3)

First we deal with the univariate case (for any t) and drop the subscript t of the functions. We write $\mathcal{S}_n$ and $\mathcal{C}_k$ to denote the sequences of univariate functions. We want to show that for any $\epsilon > 0$, there exist (1) a sequence of functions $\mathcal{S}_n(x_t, \mathcal{C}_k(x_{1:t-1}))$ in the given form, and (2) large enough N and K such that when $n \geq N$ and $k \geq K$, $|\mathcal{S}_n(x_t, \mathcal{C}_k(x_{1:t-1})) - S(x_t, x_{1:t-1})| \leq \epsilon$ for all $x_{1:t} \in [r_0, r_1]^t$.

The idea is first to show that we can find a sequence of parameters, $\mathcal{C}_n(x_{1:t-1})$, that yield a good approximation of the target function, $S(x_t, x_{1:t-1})$. We then show that these parameters can be arbitrarily well approximated by the outputs of a neural network, $\mathcal{C}_k(x_{1:t-1})$, which in turn yield a good approximation of S.

From Lemma 2, we know that such a sequence $\mathcal{C}_n(x_{1:t-1})$ exists, and furthermore that we can, for any ϵ, and independently of S and $x_{1:t-1}$ choose an N large enough so that:

$$|\mathcal{S}_n(x_t, \mathcal{C}_n(x_{1:t-1})) - S(x_t, x_{1:t-1})| < \frac{\epsilon}{2} \tag{22}$$

To see that we can further approximate a given $\mathcal{C}_n(x_{1:t-1})$ well by $\mathcal{C}_k(x_{1:t-1})$ [12], we apply the classic result of Cybenko (1989), which states that a multilayer perceptron can approximate any continuous function on a compact subset of $\mathbb{R}^m$. Note that specification of $\mathcal{C}_n(x_{1:t-1}) = (w_{tj}, b_{tj}, \tau_{tj})_{j=1}^n$ in Lemma 2 depends on the quantiles of $S_t(x_t, \cdot)$ as a function of $x_{1:t-1}$; since the quantiles are continuous functions of $x_{1:t-1}$, so is $\mathcal{C}_n(x_{1:t-1})$, and the theorem applies.

Now, $\mathcal{S}$ has bounded derivative wrt $\mathcal{C}$, and is thus uniformly continuous, so long as τ is greater than some positive constant, which is always the case for any fixed $\mathcal{C}_n$, and thus can be guaranteed for $\mathcal{C}_k$ as well (for large enough k). Uniform continuity allows us into translate the convergence of $\mathcal{C}_k \to \mathcal{C}_n$ to convergence of $\mathcal{S}_n(x_t, \mathcal{C}_k(x_{1:t-1})) \to \mathcal{S}_n(x_t, \mathcal{C}_n(x_{1:t-1}))$, since for any ϵ, there exists a $\delta > 0$ such that

$$\begin{aligned}&\| \mathcal{C}_k(x_{1:t-1}) - \mathcal{C}_n(x_{1:t-1}) \|_\infty < \delta \\ &\implies |\mathcal{S}_n(x_t, \mathcal{C}_k(x_{1:t-1})) - \mathcal{S}_n(x_t, \mathcal{C}_n(x_{1:t-1}))| < \frac{\epsilon}{2}\end{aligned} \tag{23}$$

Combining this with Equation 22, we have for all x_t and $x_{1:t-1}$, and for all $n \geq N$ and $k \geq K$

$$\begin{aligned}&\left|\mathcal{S}_n\left(x_t, \mathcal{C}_k(x_{1:t-1})\right) - S(x_t, x_{x_{1:t-1}})\right| \\ &\quad \leq \left|\mathcal{S}_n\left(x_t, \mathcal{C}_k(x_{1:t-1})\right) - \mathcal{S}_n(x_t, \mathcal{C}_n(x_{1:t-1}))\right| + \\ &\quad\quad \left|\mathcal{S}_n(x_t, \mathcal{C}_n(x_{1:t-1})) - S(x_t, x_{x_{1:t-1}})\right| \\ &\quad < \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon\end{aligned}$$

Having proved the univariate case, we add back the subscript t to denote the index of the function. From the above, we know that given any $\epsilon > 0$ for each t, there exist $N(t)$ and a sequence of univariate functions $\mathcal{S}_{n,t}$ such that for all $n \geq N(t)$, $|\mathcal{S}_{n,t} - S_t| < \epsilon$ for all $x_{1:t}$. Choosing $N = \max_{t \in [1,m]} N(t)$, we have that there exists a sequence of multivariate functions $\mathcal{S}_n$ in the given form such that for all $n \geq N$, $\| \mathcal{S}_n - S \|_\infty < \epsilon$ for all $x_{1:m}$.

□

G. Proof of Universal Approximation of DSF

Lemma 4. *Let* $X \in \mathcal{X}$ *be a random variable, and* $\mathcal{X} \subseteq \mathbb{R}^m$ *and* $\mathcal{Y} \subseteq \mathbb{R}^m$*. Given any function* $J : \mathcal{X} \to \mathcal{Y}$ *and a sequences of functions* J_n *that converges pointwise to* J*, the sequence of random variables induced by the transformations* $Y_n \doteq J_n(X)$ *converges in distribution to* $Y \doteq J(X)$*.*

Proof. (Lemma 4)

Let h be any bounded, continuous function on $\mathbb{R}^m$, so that $h \circ J_n$ converges pointwise to $h \circ J$ by continuity of h. Since h is bounded, then by the **dominated convergence theorem**, $\mathbb{E}[h(Y_n)] = \mathbb{E}[h(J_n(X))]$ converges to $\mathbb{E}[h(J(X)] = \mathbb{E}[h(Y)]$. As this result holds for any bounded continuous function h, by the **Portmanteau's lemma**, we have $Y_n \xrightarrow{d} Y$. □

[11] $S(\cdot) : [r_0, r_1]^m \to [0, 1]^m$ is a multivariate-multivariable function, where $S(\cdot)_t$ is its t-th component, which is a univariate-multivariable function, written as $S_t(\cdot, \cdot) : [r_0, r_1] \times [r_0, r_1]^{t-1} \to [0, 1]$.

[12] Note that $\mathcal{C}_n$ is a chosen function (we are not assuming its parameterization; i.e. not necessarily a neural network) that we seek to approximate using $\mathcal{C}_k$, which is the output of a neural network.

Proof. (Proposition 2)

Given an arbitrary ordering, let F be the CDFs of Y, defined as $F_t(y_t, x_{1:t-1}) = \Pr(Y_t \leq y_t | x_{1:t-1})$. According to Theorem 1 of Hyvärinen & Pajunen (1999), $F(Y)$ is uniformly distributed in the cube $[0,1]^m$. F has an upper triangular Jacobian matrix, whose diagonal entries are conditional densities which are positive by assumption. Let G be a multivariate and multivariable function where G_t is the inverse of the CDF of Y_t: $G_t(F_t(y_t, x_{1:t-1}), x_{1:t-1}) = y_t$.

According to Lemma 3, there exists a sequence of functions in the given form $(\mathcal{S}_n)_{n\geq 1}$ that converge uniformly to $\sigma \circ G$. Since uniform convergence implies pointwise convergence, $G_n = \sigma^{-1} \circ S_n$ converges pointwise to G, by continuity of σ^{-1}. Since G_n converges pointwise to G and $G(X) = Y$, by Lemma 4, we have $Y_n \xrightarrow{d} Y$

□

Proof. (Proposition 3)

Given an arbitrary ordering, let H be the CDFs of X:

$$\begin{aligned} y_1 &\doteq H_1(x_1, \emptyset) = F_1(x_1, \emptyset) = \Pr(X_1 \leq x_1 | \emptyset) \\ y_t &\doteq H_t(x_t, x_{1:t-1}) \\ &= F_t\big(x_t, \{H_{t-t'}(x_{t-t'}, x_{1:t-t'-1})\}_{t'=1}^{t-1}\big) \\ &= \Pr(X_t \leq x_t | y_{1:t-1}) \quad \text{for } 2 \leq t \leq m \end{aligned}$$

Due to Hyvärinen & Pajunen (1999), $y_1, ...y_m$ are independently and uniformly distributed in $(0,1)^m$.

According to Lemma 3, there exists a sequence of functions in the given form $(\mathcal{S}_n)_{n\geq 1}$ that converge uniformly to H. Since $H_n = \mathcal{S}_n$ converges pointwise to H and $H(X) = Y$, by Lemma 4, we have $Y_n \xrightarrow{d} Y$

□

Proof. (Theorem 1)

Given an arbitrary ordering, let H be the CDFs of X defined the same way in the proof for Proposition 3, and let G be the inverse of the CDFs of Y defined the same way in the proof for Proposition 2. Due to Hyvärinen & Pajunen (1999), $H(X)$ is uniformly distributed in $(0,1)^m$, so $G(H(X)) = Y$. Since $H_t(x_t, x_{1:t-1})$ is monotonic wrt x_t given $x_{1:t-1}$, and $G_t(H_t, H_{1:t-1})$ is monotonic wrt H_t given $H_{1:t-1}$, G_t is also monotonic wrt x_t given $x_{1:t-1}$, as

$$\frac{\partial G_t(H_t, H_{1:t-1})}{\partial x_t} = \frac{\partial G_t(H_t, H_{1:t-1})}{\partial H_t} \frac{\partial H_t(x_t, x_{1:t-1})}{\partial x_t}$$

is always positive.

According to Lemma 3, there exists a sequence of functions in the given form $(\mathcal{S}_n)_{n\geq 1}$ that converge uniformly to $\sigma \circ G \circ H$. Since uniform convergence implies pointwise convergence, $K_n = \sigma^{-1} \circ S_n$ converges pointwise to $G \circ H$, by continuity of σ^{-1}. Since K_n converges pointwise to $G \circ H$ and $G(H(X)) = Y$, by Lemma 4, we have $Y_n \xrightarrow{d} Y$

□

H. Experimental Details

For the experiment of amortized variational inference, we implement the Variational Autoencoder (Kingma & Welling, 2013). Specifically, we follow the architecture used in Kingma et al. (2016): the encoder has three layers with $[16, 32, 32]$ feature maps. We use resnet blocks (He et al., 2016) with 3×3 convolution filters and a stride of 2 to downsize the feature maps. The convolution layers are followed by a fully connected layer of size 450 as a context for the flow layers that transform the noise sampled from a standard normal distribution of dimension 32. The decoder is symmetrical with the encoder, with the strided convolution replaced by a combination of bilinear upsampling and regular resnet convolution to double the feature map size. We used the ELUs activation function (Clevert et al., 2015) and weight normalization (Salimans & Kingma, 2016) in the encoder and decoder. In terms of optimization, Adam (Kingma et al., 2015) is used with learning rate fined tuned for each inference setting, and Polyak averaging (Polyak & Juditsky, 1992) was used for evaluation with $\alpha = 0.998$ which stands for the proportion of the past at each time step. We also consider a variant of Adam known as Amsgrad (Reddi et al., 2018) as a hyperparameter. For vanilla VAE, we simply apply a resnet dot product with the context vector to output the mean and the pre-softplus standard deviation, and transform each dimension of the noise vector independently. We call this linear flow. For IAF-affine and IAF-DSF, we employ MADE (Germain et al., 2015) as the conditioner $c(x_{1:t-1})$, and we apply dot product on the context vector to output a scale vector and a bias vector to conditionally rescale and shift the preactivation of each layer of the MADE. Each MADE has one hidden layer with 1920 hidden units. The IAF experiments all start with a linear flow layer followed by IAF-affine or IAF-DSF transformations. For DSF, we choose $d = 16$.

For the experiment of density estimation with MAF, we followed the implementation of Papamakarios et al. (2017). Specifically for each dataset, we experimented with both 5 and 10 flow layers, followed by one linear flow layer. The following table specifies the number of hidden layers and the number of hidden units per hidden layer for MADE:

Table 3. Architecture specification of MADE in the MAF experiment. Number of hidden layers and number of hidden units.

POWER	GAS	HEPMASS	MINIBOONE	BSDS300
2×100	2×100	2×512	1×512	2×1024