Transformer Neural Autoregressive Flows

Massimiliano Patacchiola [1] **Aliaksandra Shysheya** [1] **Katja Hofmann** [2] **Richard E. Turner** [1]

Abstract

Density estimation, a central problem in machine learning, can be performed using Normalizing Flows (NFs). NFs comprise a sequence of invertible transformations, that turn a complex target distribution into a simple one, by exploiting the change of variables theorem. Neural Autoregressive Flows (NAFs) and Block Neural Autoregressive Flows (B-NAFs) are arguably the most perfomant members of the NF family. However, they suffer scalability issues and training instability due to the constraints imposed on the network structure. In this paper, we propose a novel solution to these challenges by exploiting transformers to define a new class of neural flows called Transformer Neural Autoregressive Flows (T-NAFs). T-NAFs treat each dimension of a random variable as a separate input token, using attention masking to enforce an autoregressive constraint. We take an amortization-inspired approach where the transformer outputs the parameters of an invertible transformation. The experimental results demonstrate that T-NAFs consistently match or outperform NAFs and B-NAFs across multiple datasets from the UCI benchmark. Remarkably, T-NAFs achieve these results using an order of magnitude fewer parameters than previous approaches, without composing multiple flows.

1. Introduction

Normalizing Flows (NFs) have emerged as a powerful paradigm in probabilistic modeling and machine learning, offering a versatile framework for mapping between probability density functions through invertible transformations. NFs are suitable for applications which require models of complex probability distributions that support computationally efficient sampling and density evaluation. Consequently, NPs are useful in density modelling problems and also for variational inference in which they are used to specify approximate posterior distributions.

Most of the research related to NFs focuses on designing expressive transformations that conform to practical constraints. NFs specify an bijective mapping from a simple base distribution to a target distribution so that the transformation of densities formula can be used to specify the density over the target variables. The transformation of densities formula requires the computation of the determinant of the Jacobian of this mapping and therefore NF models are structured in such a way that these objects are simple to compute. For example, one notable class of NFs, known as Autoregressive Flows (AFs), decompose a joint distribution into a product of univariate conditionals. These transformations possess a lower triangular Jacobian, making it computationally tractable to compute their determinants, which is an essential requirement for the application of the change of variables theorem. Kingma et al. 2016 introduced the concept of inverse autoregressive flows (IAFs), refining AFs by employing a composition of trivially invertible affine transformations to model each conditional, paving the way for more efficient computation while retaining expressiveness.

Recent advances in the NF landscape include the introduction of Neural Autoregressive Flows (NAFs, Huang et al. 2018) and Block Neural Autoregressive Flows (B-NAFs, De Cao et al. 2020). NAFs and B-NAFs replace the IAF's transformation with a learned bijection implemented via a strictly monotonic neural network. This fundamental paradigm shift makes NAFs and BNAFs more flexible than IAFs, as they are universal approximators of real and continuous distributions. However, despite these advancements, both NAFs and B-NAFs encounter challenges in scalability and efficiency, primarily due to the intricate structures and constraints imposed on their networks. The scalability issue becomes particularly pronounced when dealing with high-dimensional data, where the number of parameters in the network increases significantly. Additionally, the training instability often observed in these models can be attributed to the strict monotonicity and masking constraints, which can limit the expressive power and learning dynamics of the networks. These challenges necessitate a more flexible and scalable approach to enhance the practical applicability of NFs in complex, real-world scenarios.

[1]Department of Engineering, University of Cambridge, Cambridge, UK [2]Microsoft Research, Cambridge, UK. Correspondence to: Massimiliano Patacchiola <mp2008@cam.ac.uk>.

Accepted by the Structured Probabilistic Inference & Generative Modeling workshop of ICML 2024, Vienna, Austria.

In response to these limitations, we introduce Transformer Neural Autoregressive Flows (T-NAFs), a novel approach that leverages the power of transformer architectures within the realm of NFs. T-NAFs treat each dimension of a random variable as a separate input token and utilize attention masking to enforce an autoregressive constraint. This design choice allows T-NAFs to handle high-dimensional data more efficiently, as the transformer architecture inherently scales better with dimensionality and does not require the same level of parameterization as traditional feedforward networks. Moreover, T-NAFs offer enhanced flexibility and stability in training by freeing the majority of the parameters from the constraints typically imposed in NAFs and B-NAFs. By incorporating transformers, T-NAFs not only address the scalability and efficiency issues but also open up new avenues for innovation in the field of NFs, particularly in complex density estimation tasks.

The main contributions of this paper can be summarized as follows:

1. We introduce Transformer Neural Autoregressive Flows (T-NAFs), a novel model that effectively combines transformers with neural autoregressive flows for density estimation, utilizing attention masking to enforce autoregression efficiently.

2. We demonstrate remarkable flexibility in handling various invertible transformations and showcase significant improvements in parameter efficiency and scalability compared to previous models like NAFs and B-NAFs.

3. Through extensive ablation studies and empirical evaluations across multiple benchmarks, we provide key insights into the adaptability, complexity, and effectiveness of the model. The code is released with an open-source license [1].

2. Previous work

In this section we delve into the extensive body of research surrounding NFs, a domain that has witnessed substantial developments in the last years. For a comprehensive and detailed understanding of the progress in this field, we refer the reader to recent review papers of Kobyzev et al. 2019 and Papamakarios et al. 2021.

One broad line of research views NFs as a finite number of tractable simple transformations and hence focuses on developing more expressive transformations that have a tractable inverse and Jacobian determinant. Rezende & Mohamed 2015 were the first to introduce two parametric families of such transformations, i.e. the planar and the radial flows. Another well-known representatives of this research direction are Autoregressive Flows (Section 3.2), which are particularly relevant to our work. Dinh et al. 2016 and Kingma et al. 2016 proposed to build Autoregressive Flows with simple affine transformations. To improve the limited expressivity of the affine autoregressive transformations, several non-affine transformations were proposed: Huang et al. 2018 and De Cao et al. 2020 use a monotonic multi-layered perceptron, while Jaini et al. 2019 and Wehenkel & Louppe 2019 use an integral of some positive function represented as neural network. Although these transformations can be made arbitrary flexible, their major drawback is that they do not have an analytic inverse. To overcome this issue, different spline-based transformations were proposed (Müller et al., 2018; Durkan et al., 2019; Dolatabadi et al., 2020).

Another important design choice for Autoregressive Flows is how to implement the conditioning on observed variables. Even though the only constraint imposed on the conditioning function is the autoregressive one, in practice it is still important to choose a parameter-efficient function that could accommodate inputs of variable size. For instance, Oliva et al. 2018 and Kingma et al. 2016 use recurrent neural network (RNN) as a conditioning function. Even though parameter-efficient, RNNs require sequential computation, which can be prohibitively expensive for longer sequences. As an alternative to RNNs, the idea of masking can be used: usually a standard feedforward network is trained, but some of the connections are zeroed out to preserve the autoregressive structure of the conditioning function. Germain et al. 2015 proposed a masking procedure for fully-connected neural networks. Masking was picked up in many autoregressive flows models (Kingma et al., 2016; De Cao et al., 2020; Huang et al., 2018; Papamakarios et al., 2017).

Transformers have been used for distribution modeling (Fakoor et al., 2020). The method exploits a transformer encoder that takes the input sequence $x_1, x_2, \ldots, x_i$ and predicts the i-th conditional distribution $p(x_i|x_1, x_2, \ldots, x_{i-1})$. The output is a categorical distribution (discrete variables), or a mixture of Gaussians (continuous variables). The main issue with this approach is that it is not possible to get an exact estimate of the likelihood of a sample, as the transformations are not bijective. Moreover, if the input sequence is long the output of the model will be disproportionally large, since each input is mapped to multiple outputs (e.g. categories or parameters of the mixture).

Most recent work has explored the possibility of building an invertible normalizing flow network that unfolds the Wasserstein gradient flow via a neural ODE model (Xu et al., 2024).

[1] https://github.com/mpatacchiola/TNAF

3. Background

3.1. Normalizing Flows

A Normalizing Flow (NF) is an invertible (bijective) funciton $f : \mathcal{X} \to \mathcal{Y}$ between two continuous random variables. Since f is invertible, we can use the change of variables formula to translate between two densities $p_X(\mathbf{x})$ and $p_Y(\mathbf{y})$

$$p_Y(\mathbf{y}) = p_X(\mathbf{x}) \left|\det \mathbf{J}_{f(\mathbf{x})}\right|^{-1} \quad \text{where} \quad \mathbf{J}_{f(\mathbf{x})} = \tfrac{\partial f(\mathbf{x})}{\partial \mathbf{x}}.$$

The determinant of the Jacobian $\mathbf{J}_{f(\mathbf{x})}$ accounts for local expansions/contractions of $\mathcal{X}$ at $\mathbf{x}$. Estimating this determinant is one of the major bottlenecks to overcome, since typically this has complexity $\mathcal{O}(N^3)$ for a Jacobian matrix of size $N \times N$. One way to deal with this issue is to enforce a lower-triangular structure on the Jacobian. The determinant of a lower-triangular matrix can be estimated by multiplying the elements on the main diagonal, which has cost $\mathcal{O}(N)$. The lower-triangular structure of the Jacobian can be enforced through an autoregressive factorization, as detailed in the next subsection.

3.2. Autoregressive Flows

For a multivariate random variable $\mathbf{x} \in \mathbb{R}^K$ sampled from $p_X(\mathbf{x})$ we can estimate the joint distribution with an autoregressive factorization by exploiting the chain rule

$$p_X(\mathbf{x}) = p_{X_1}(x_1) p_{X_2}(x_2|x_1) \dots p_{X_K}(x_K|x_{K-1}, \dots, x_1).$$

Exploiting this structure it is possible to estimate an arbitrary y_i by defining an autoregressive conditioner c and an invertible transformation t, such that

$$y_i = f(x_1, \dots, x_i) = t(c(x_1, \dots, x_{i-1}), x_i). \qquad (1)$$

Previous work has investigated different ways of parametrizing c and t. A line of work focused on defining simple affine transformations such as $t(\mu, \sigma, x_i) = \mu + \sigma x_i$ in RealNVP (Dinh et al., 2016) or $t(\mu, \sigma, x_i) = \sigma x_i + (1 - \sigma)\mu$ in IAF (Kingma et al., 2016). While those transformations are trivial to invert, they suffer from limited flexibility, which makes them inadequate to model complex distributions. In order to overcome this issue another line of work has proposed to use neural networks to represent the invertible transformation and/or the conditioner. In Section 3.3, we describe these models in more detail.

3.3. Neural Autoregressive Flows

Neural Autoregressive Flows (NAFs). In NAFs (Huang et al., 2018), the transformation in (1) is reframed as follows

$$t(c(x_1, \dots, x_{i-1}), x_i) = \\ \text{NN}\big(x_i;\ \boldsymbol{\psi}_i = c(x_1, \dots, x_{i-1};\ \boldsymbol{\theta}_i)\big),$$

where NN is a feed-forward neural network and $\boldsymbol{\psi}_i$ are pseudo-parameters generated by a conditioner network (parametrized by $\boldsymbol{\theta}$). Those pseudo-parameters represent the weights and biases of the neural network. The conditioner network is implemented using a masked feed-forward model as in MADE (Germain et al., 2015), while the invertible transformation is implemented as a deep sigmoidal flow (DSF) or a deep dense sigmoidal flow (DDSF). Both DSF and DDSF exploit sigmoidal inflection points in a fully-connected model to induce multi-modality. A major limitation of NAFs is the use of MADE in the conditioner. With MADE the number of parameters in the network grows with the dimensionality of the random variable, which makes the computational cost prohibitively expensive. Moreover, MADE exploits a zero-masking strategy to impose the autoregressive constraint and to parallelize the computation in the forward pass, which results in a large portion of unused weights that affects the parameter count without having any concrete utility.

Block Neural Autoregressive Flows (B-NAFs). In order to overcome some of the limitations of NAFs, De Cao et al. 2020 proposed B-NAFs. The main idea of B-NAF is to directly parametrize the transformation t without the conditioner c by using a single neural network

$$t(c(x_1, \dots, x_{i-1}), x_i) = \text{NN}\big(x_1, \dots, x_{i-1};\ \boldsymbol{\theta}_i\big).$$

The neural network is a feed-forward model, carefully designed to be both autoregressive and strictly monotone (bijective). The autoregressive constraint is imposed by using masking as in MADE (Germain et al., 2015) and the monotonic constraint is imposed by using strictly positive weights and invertible activation functions. While B-NAFs are more effective than NAFs on a variety of datasets (De Cao et al., 2020), they still have two major limitations. The first limitation is that, similarly to NAFs, B-NAFs rely on masked feed-forward networks (MADE) that scale poorly with the dimensionality of the random variable, which results in unused parameters due to zero-masking. The second limitation is that enforcing monotonicity in a large model leads to training instability.

In the next section, we describe how to overcome the limitations of NAFs and B-NAFs by exploiting transformers as conditioning mechanisms, presenting a new type of neural autoregressive flow.

4. Transformer Neural Autoregressive Flows

In order to overcome the limitations of NAF and B-NAF we propose a new type of flow called Transformer Neural Autoregressive Flow (T-NAF). The main idea of T-NAF is to allocate most of the computational capacity to the conditioner, using a transformer neural network (Vaswani et al., 2017). This design choice has a series of advantages: (i)

we can define an autoregressive constrain by simply using an autoregressive mask in the attention layer, which, contrary to MADE, does not result in unused weights; (ii) the parameters of the transformer are shared across all the dimensions of the random variable, which significantly reduces the number of total parameters in the model; (iii) the conditioner does not need to be monotonic, which makes training significantly more stable; and (iv) we can exploit all the techniques that have been developed in the last few years to boost the performance of vision and language transformers (e.g. FlashAttention Dao et al. 2022, PagedAttention Kwon et al. 2023).

In T-NAFs we define the invertible transformation and conditioner as follows

$$t(c(x_1, \dots, x_{i-1}), x_i) = \\ t\big(x_i;\ \boldsymbol{\psi}_i = \text{TN}(x_1, \dots, x_{i-1}, i;\ \boldsymbol{\theta})\big), \qquad (2)$$

where TN is a Transformer Network parametrized by $\boldsymbol{\theta}$, that takes as input the random variable and the index i (implicitly included via positional encoding), and generates the parameters $\boldsymbol{\psi}_i$ of the transformation t. We do not impose any particular constraint on the transformation t, which can be any type of invertible transformation. In Section 4.2, we describe in more details some of the transformations we have used in our experiments. Empirically we have observed that a compact monotonic neural network (taking the variable x_i as input and producing $y_i \in [0, 1]$ as output) offers the best trade-off between performance and parameter count. This is equivalent to a parametric cumulative distribution function (CDF). Crucially, because of the single input-output structure, this network does not require MADE masking and it is stable to train due to its limited size.

Importantly, in Equation (2) there is a single set of parameters $\boldsymbol{\theta}$ in the conditioner that are shared across all the dimensions of the random variable. In contrast, in both NAFs and B-NAFs there is a separate set of parameters $\boldsymbol{\theta}_i$ for each dimension x_i. This means that T-NAFs require significantly fewer parameters (see Section 5 for more details).

4.1. Overview of the architecture

The overall design of the proposed transformer is similar to the Vision Transformer (ViT, Dosovitskiy et al. 2020) with two key differences: (i) we replace the convolutional projections with simple linear projections, as we are not dealing with image patches; (ii) we replace the classification head (taking as input only the first embedding in the sequence) with a projection head that generates the pseudo-parameters (for all the embeddings in the sequence). Figure 1 showcases a high-level overview of the T-NAF architecture.

In the initial stage of the pipeline, each dimension of the random variable is linearly projected to get input embeddings. Then these are added to learnable positional embeddings and passed to a series of L transformer encoder layers. The first element of the autoregressive factorization is the marginal distribution $p(x_1)$, which does not require any conditioning. To impose this constraint we use a learnable input embedding as the first element of the sequence. This guarantees that the first set of pseudo-parameters $\boldsymbol{\psi}_1$ is generated without having access to any element of the sequence. This is similar in spirit to the Beginning of Sequence (BoS) token or class token used in a variety of transformer models (Devlin et al., 2018; Dosovitskiy et al., 2020).

A detailed representation of the encoder layer is provided on the right-hand side of Figure 1. The encoder layer performs the following operations: normalization via layer-norm, attention estimation via Multi-Head Attention (MHA), normalization via layer-norm, forward pass in a Multi-Layer Perceptron (MLP). Skip connections are used from the input to the output of the MHA, and from the output of the MHA to the output of the encoder. An autoregressive mask is used in the MHA to enforce the autoregressive constraint. The output of the encoder is a series of hidden embedding vectors $\mathbf{h}_i$, which are passed to a projection head to generate the pseudo-parameters $\boldsymbol{\psi}_i$. The pseudo-parameters are used to define the invertible transformation $y_i = t(x_i;\ \boldsymbol{\psi}_i)$. The projection head can be a set of linear layers that takes as input the embeddings $\mathbf{h}_i$ and generates the pseudo-parameters $\boldsymbol{\psi}_i$, or it can simply be an identity function (see Section 4.2 for additional details).

4.2. Transformation

The proposed architecture allows using a large and diversified set of transformations, and below we detail some of them.

Affine transformation. One of the simplest transformations is the affine one. An affine transformation is parameterized by a scale μ_i and shift σ_i parameters, grouped into a set of pseudo-parameters $\boldsymbol{\psi}_i = \{\mu_i, \sigma_i\}$, and is defined as follows

$$t_{\text{affine}}(x_i;\ \boldsymbol{\psi}_i) = \mu_i + \sigma_i\, x_i \qquad (3)$$

with $\mu_i = \text{LIN}_\mu(\mathbf{h}_i; \boldsymbol{\theta}_\mu), \sigma_i = \text{LIN}_\sigma(\mathbf{h}_i; \boldsymbol{\theta}_\sigma)$. where LIN_μ and LIN_σ are linear layers parametrized by $\boldsymbol{\theta}_\mu$ and $\boldsymbol{\theta}_\sigma$ used to generate the scale μ_i and shift σ_i. The linear layers of the projection head are shared across all the output tokens. Note that it is possible to concatenate multiple affine transformations via function composition. In this case a separate set of linear layers can be assigned to each transformation.

CDF transformation. A more sophisticated transformation is given by a parametric cumulative distribution function (CDF) that maps x_i to a uniform distribution. Following previous work, we can define a parametric CDF as a neural network with positive weights (Archer & Wang, 1993; Sill, 1997; Daniels & Velikova, 2010; De Cao et al., 2020). In

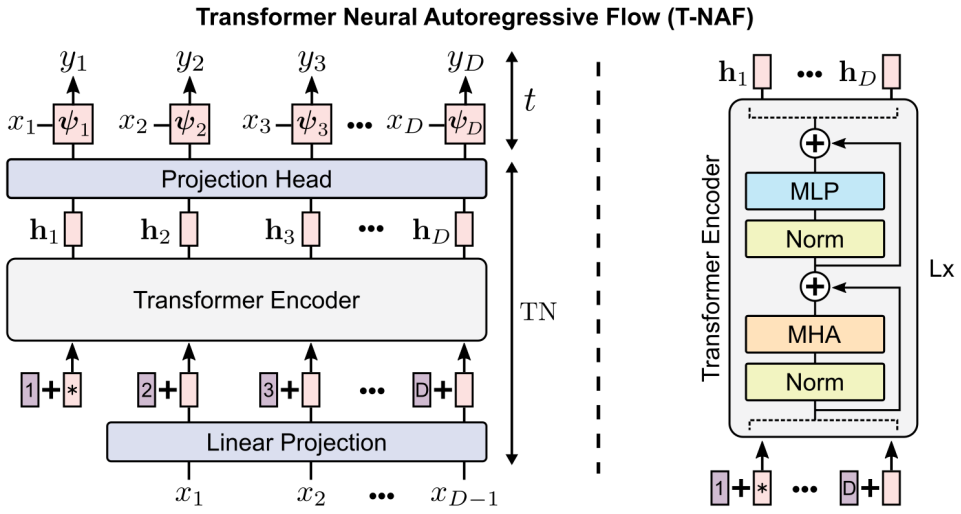

Figure 1. Graphical representation of a T-NAF model. **Left**: the architecture includes a transformer neural network conditioner TN and an invertible transformation t. In $\text{TN}(x_1, \dots, x_D, i; \boldsymbol{\theta})$ each dimension of the random variable is linearly projected to get an embedding which is added to a learnable position vector. A sequence of L transformer layers is used to produce hidden embeddings $\mathbf{h}_1, \dots, \mathbf{h}_D$ that are passed through a projection head to generate pseudo-parameters $\boldsymbol{\psi}_1, \dots, \boldsymbol{\psi}_D$. The pseudo-parameters are used as part of an invertible transformation $t(x_i; \boldsymbol{\psi}_i)$. **Right**: detailed schematic of a transformer encoder layer. Inputs are passed through a normalization layer, a Multy-Head Attention (MHA) layer with autoregressive mask, another normalization layer, and an MLP.

this case the pseudo-parameters are the weights and biases of the neural network $\boldsymbol{\psi}_i = \{\mathbf{w}_i^{(1)}, \mathbf{b}_i^{(1)}, \mathbf{w}_i^{(2)}, b_i^{(2)}\}$, which are used in the following transformation

$$t_{\text{CDF}}(x_i;\ \boldsymbol{\psi}_i) = \\ \text{sig}\left(\tanh\left(\exp \mathbf{w}_i^{(1)}\ x_i + \mathbf{b}_i^{(1)}\right)^T \exp \mathbf{w}_i^{(2)} + b_i^{(2)}\right), \tag{4}$$

where $\text{sig}(\cdot)$ and $\tanh(\cdot)$ are the sigmoid and hyperbolic tangent functions. Similarly to Equation (3), the weights and biases are generated via dedicated linear layers $\mathbf{w}_i^j = \text{LIN}_{\mathbf{w}^j}(\mathbf{h}_i; \boldsymbol{\theta}_{\mathbf{w}^j})$ and $\mathbf{b}_i^j = \text{LIN}_{\mathbf{b}^j}(\mathbf{h}_i; \boldsymbol{\theta}_{\mathbf{b}^j})$ as a part of the projection head. The exponential function used in Equation (4) ensures that the weights are strictly-positive enforcing the monotonic constraint (softplus can also be used). This and the use of invertible activation functions makes the transformation bijective. It is possible to invert t_{CDF} since it has a single input, a single output, and is bijective. The inverse of t_{CDF} is an inverse-CDF, which can be used for inversion sampling following two steps: (i) first sample a value y_i from the uniform distribution, (ii) find the corresponding x_i via a root-finding method, such as the bisection method (Wehenkel & Louppe, 2019).

Shared-CDF transformation. In the CDF transformation defined in the previous paragraph, the pseudo-parameters are generated by using the projection head and the transformer embeddings. An alternative approach is to define the CDF neural network in advance, using a fixed set of weights that are shared across all the outputs. In this case, y_i is obtained via a conditional CDF, that takes as input x_i and the transformer embeddings $\mathbf{h}_i$. Assuming that the conditional CDF is modeled using a neural network with a single hidden layer, we can formalize this particular case defining a set of pseudo-parameters $\boldsymbol{\psi}_i$ and a set of shared-parameters $\boldsymbol{\phi}$:

$$\boldsymbol{\psi}_i = \{\mathbf{h}_i\} \text{ and } \boldsymbol{\phi} = \left\{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{b}^{(1)}, b^{(2)}, \hat{\mathbf{W}}^{(1)}, \hat{\mathbf{w}}^{(2)}\right\}.$$

The pseudo-parameters coincide with the embeddings produced by the transformer conditioner $\mathbf{h}_i = \text{TN}(x_1, \dots, x_{i-1}, i; \boldsymbol{\theta})$. The shared parameters include $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{b}^{(1)}, b^{(2)}$, which are related to the input x_i, and $\hat{\mathbf{W}}^{(1)}, \hat{\mathbf{w}}^{(2)}$, which are related to the conditional input $\mathbf{h}_i$. The shared-CDF transformation is defined as follows

$$t_{\text{S-CDF}}(x_i;\ \boldsymbol{\psi}_i, \boldsymbol{\phi}) = \\ \text{sig}\Bigg(\tanh\left(\exp \mathbf{w}^{(1)}\ x_i + \hat{\mathbf{W}}^{(1)}\ \mathbf{h}_i + \mathbf{b}^{(1)}\right)^T \exp \mathbf{w}^{(2)} \\ + \hat{\mathbf{w}}^{(2)}\mathbf{h}_i + b^{(2)}\Bigg), \tag{5}$$

where we have discarded any reference to the projection head for clarity (this corresponds to the case where identity function is used as the projection head). Note the crucial distinction between positive-constraint weights $\exp \mathbf{w}^{(1)}, \exp \mathbf{w}^{(2)}$ used for the input and the unconstrained

weights $\hat{\mathbf{W}}^{(1)}, \hat{\mathbf{w}}^{(2)}$ used in the conditioning mechanism. The main advantage of the shared-CDF approach is that the transformer needs to generate just the vector of embeddings $\mathbf{h}_i$ for each token, while in the CDF transformation it needs to generate the weights and biases of the entire CDF network. Therefore, it is more convenient to use a shared-CDF transformation when the size of the CDF neural network is large.

Spline transformation. Following Durkan et al. 2019, we also incorporated a spline transformation based on monotonic rational-quadratic splines. Besides being comparable to the CDF transformation in terms of performance and capacity (Durkan et al., 2019), the spline transformation has the advantage of being easily invertible. Each block of this transformation consists of a element-wise rational-quadratic spline

$$t_{\text{spline}}(x_i;\ \boldsymbol{\psi}_i) = g(x_i; \boldsymbol{\psi}_i = \text{TN}\,(x_1, \ldots, x_{i-1}, i; \boldsymbol{\theta})),$$

where $g(\cdot;\ \boldsymbol{\psi})$ is a monotonic rational-quadratic spline function parameterized by a set of pseudo-parameters $\boldsymbol{\psi}$ generated by the transformer conditioner (see Durkan et al., 2019 for the details on the parameterization of the spline). To ensure cross-variable interactions, the element-wise function is commonly followed by an invertible linear transformation across variables

$$t_{\text{linear}}(\mathbf{x};\ \boldsymbol{\phi}) = \mathbf{PLUx},$$

with the set of shared parameters $\boldsymbol{\phi} = \{\mathbf{P}, \mathbf{L}, \mathbf{U}\}$, where $\mathbf{P}$ is a fixed permutation matrix, $\mathbf{L}$ is lower triangular with ones on the diagonal, and $\mathbf{U}$ is upper triangular. We generate a set of $\boldsymbol{\psi}_i^j$ from the initial variables $x_1, \ldots, x_{i-1}$ at each spline block transformation j. However, this means that the suggested $t_{\text{linear}}(\mathbf{x};\ \boldsymbol{\phi})$ breaks the autoregressive constraint and the determinant of the Jacobian cannot be computed in an efficient way. To fix this issue, we simplify the linear transformation to

$$t_{\text{linear}}(\mathbf{x};\ \boldsymbol{\phi}) = \mathbf{Lx},$$

with $\boldsymbol{\phi} = \{\mathbf{L}\}$ for all spline transformation blocks except for the first one. As a result, each block of spline transformation is a composition of t_{spline} and t_{linear}, where t_{spline} is applied element-wise and t_{linear} acts on the whole set of variables.

5. Experiments

5.1. Density Estimation

In this section we evaluate the performance of our method on density estimation with real-world data, using classic benchmarks from the UCI repository. Following standard practice, we evaluate our model on four datasets from UCI (POWER, GAS, HEPMASS, MINIBOONE) and on BSDS300 (Martin et al., 2001), a dataset obtained by extracting random patches from the homonym datasets of natural image. We compare against a variety of methods: RealNVP (Dinh et al., 2016), Glow (Kingma & Dhariwal, 2018), MADE (Germain et al., 2015), MAF (Papamakarios et al., 2017), TAN (Oliva et al., 2018), FFORJD (Grathwohl et al., 2018), SOS (Jaini et al., 2019), RQ-NSF (Durkan et al., 2019). We also include neural autoregressive models such as NAF (Huang et al., 2018) and B-NAF (De Cao et al., 2020). The results for other methods are taken from the corresponding papers. Note that some methods (FFJORD, B-NAF and TAN) perform a hyper-parameter search over each dataset while we used the same hyper-parameters in all experiments. We used the following configuration for T-NAF: 1 flow, 3 or 5 layers in the transformer encoder, embeddings of size 32, 8 attention heads, 1 hidden layer with 64 units in the encoder MLPs, the projection head generates the parameters of a CDF network with 1 hidden layer with 128 units.

Results. The results are reported in Table 1. Overall, T-NAF is able to outperform all the other methods on HEPMASS using the larger model with 5 layers. On BSDS300 and GAS T-NAF obtains the second highest score after TAN and RQ-NSF, but without performing any ad-hoc hyper-parameter search and without using multiple flows. The only dataset where other methods outperform T-NAF is MINIBOONE.

T-NAF is more efficient than other models for two reasons: a reduced number of flows and reduced number of model parameters. In terms of number of flows, most of the other methods concatenate multiple transformations and permute the order of the variable in between. While this attenuates possible issues due to the variable ordering, it comes at the price of an increased overhead and parameter count. T-NAF achieves top performance using only one flow, therefore substantially reducing the complexity of the model.

Parameter efficiency. We compared T-NAF and B-NAF in terms of number of parameters vs. test log-likelihood on the UCI and BSDS300 datasets. By assessing the total parameter count required by each model, we aim to demonstrate how T-NAFs optimizes parameter usage. This evaluation offers insights into the practical implications of employing transformer-based architectures in probabilistic modeling, especially in scenarios where computational resources and model simplicity are important. For B-NAF, we report the parameters and performance of the best model as specified in De Cao et al. 2020, that is: MLP with 2 hidden layers, number of hidden units defined as $D \times 40$, and 5 flows. For our T-NAF model we report the number of parameters and performance for the variant based on a CDF transformation and a transformer with 3 and 5 layers. For a fair comparison we also include the T-NAF pseudo-parameters $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_D$ generated during the inference step. The results are summarized in Figure 2. The figure shows that T-NAF offers a better trade-off in terms of parameter count vs. performance.

Table 1. Log-Likelihood (higher is better) for 4 datasets from UCI and for BSDS300. Average and standard deviation over 3 seeds. We report number of dimensions (D) and dataset size (N, Millions). T-NAF obtains some of the best results by using just one flow and no ad hoc architectures. *Methods using ad hoc architectures on each dataset. Best result in bold, second-best underlined.

Model	Flows	POWER $D=6$ $N=2.0$	GAS $D=8$ $N=1.0$	HEPMASS $D=21$ $N=0.5$	MINIBOONE $D=43$ $N=0.04$	BSDS300 $D=63$ $N=1.3$
RealNVP	5	-0.02±0.01	4.78±1.80	-19.62±0.02	-13.55±0.49	152.97±0.28
RealNVP	10	0.17±0.01	8.33±0.14	-18.71±0.02	-13.84±0.52	153.28±1.78
Glow	n/a	0.17±0.01	8.15±0.40	-18.92±0.08	-11.35±0.07	155.07±0.03
MADE MoG	n/a	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 MoG	5	0.30±0.01	9.59±0.02	-17.39±0.02	-11.68±0.44	156.36±0.28
FFJORD*	n/a	0.46±0.01	8.59±0.12	-14.92±0.08	-10.43±0.04	157.40±0.19
SOS	7	0.60±0.01	11.99±0.41	-15.15±0.10	-8.90±0.11	157.48±0.41
NAF-DDSF	5	0.62±0.01	11.91±0.13	-15.09±0.40	**-8.86**±0.15	157.73±0.04
NAF-DDSF	10	0.60±0.02	11.96±0.33	-15.32±0.23	-9.01±0.01	157.43±0.30
B-NAF*	5	0.61±0.01	12.06±0.02	-14.71±0.38	-8.95±0.07	157.36±0.03
TAN*	5	0.60±0.01	12.06±0.02	-13.78±0.02	-11.01±0.48	**159.80**±0.07
RQ-NSF (AR)	10	**0.66**±0.01	**13.09**±0.02	-14.01±0.03	-9.22±0.48	157.31±0.28
T-NAF/3 (ours)	**1**	0.63±0.01	12.01±0.02	-14.88±0.37	-11.59±0.67	157.83±0.10
T-NAF/5 (ours)	**1**	0.54±0.01	12.27±0.01	**-13.20**±0.26	-10.67±0.06	159.41±0.04

In particular, the gap between the two methods gets larger as the dimensionality of the input increases.

5.2. Ablations

In this section, we provide a detailed ablation study to investigate the impact of different components of T-NAF on performance. A critical aspect of T-NAF is the use of different invertible transformations and their effects on the model's density estimation capabilities. Our experiments primarily focus on contrasting the performance of three distinct types of transformations: CDF, shared-CDF, and Spline transformations. Each of these transformations offers unique characteristics: CDF transformations are expressive and allow mapping inputs to the uniform distribution, potentially capturing complex patterns in the data more effectively; Spline transformations can offer high levels of flexibility and are expected to model intricate data distributions with greater precision. Additionally, the experiments compare the Shared-CDF approach, where a fixed set of weights is shared across outputs, with the standard CDF approach, where the transformer generates the parameters of the neural-CDF separately for each output. This comparison is crucial to evaluate the benefits of parameter sharing versus individual parameterization in the context of density estimation tasks. Furthermore, we explore the impact of varying the number of layers in the transformer encoder on the overall performance of the model. This exploration helps in understanding the trade-off between model complexity (in terms of depth) and performance gains. A deeper model might capture more complex relationships in the data but at the cost of increased computational requirements and potential overfitting.

Results. A summary of the results is presented in Table 2. The standard CDF model with five layers consistently outperforms the remaining models across most datasets. The Spline Flow with five layers is competitive on three datasets (POWER, HEPMASS, and BSDS300), while suffering the most on MINIBOONE. Overall, the increase in transformer layers from 3 to 5 shows a noticeable improvement across the board. This outcome indicates that additional layers might help capture more complex dependencies. Only in one case the use of additional layers seems to reduce the performance (POWER dataset). Using a shared CDF over the standard one does not seem to give particular benefits, there is only a marginal advantage on MINIBOONE and BSDS300 when compared with the CDF with three layers.

6. Conclusions

In this paper we have introduced T-NAFs, a new class of neural flows that exploits transformers as an autoregressive conditioner mechanism that generates the parameters of invertible transformations. We have showcased the performance of the model on a variety of real-world density estimation problems from the classic UCI benchmarks. Overall, the results have demonstrated that T-NAFs offer a powerful combination of efficiency, scalability, and flexibility, outperforming existing models like B-NAFs and NAFs in various scenarios. The innovative use of transformer architectures within T-NAFs not only addresses the scalability and train-

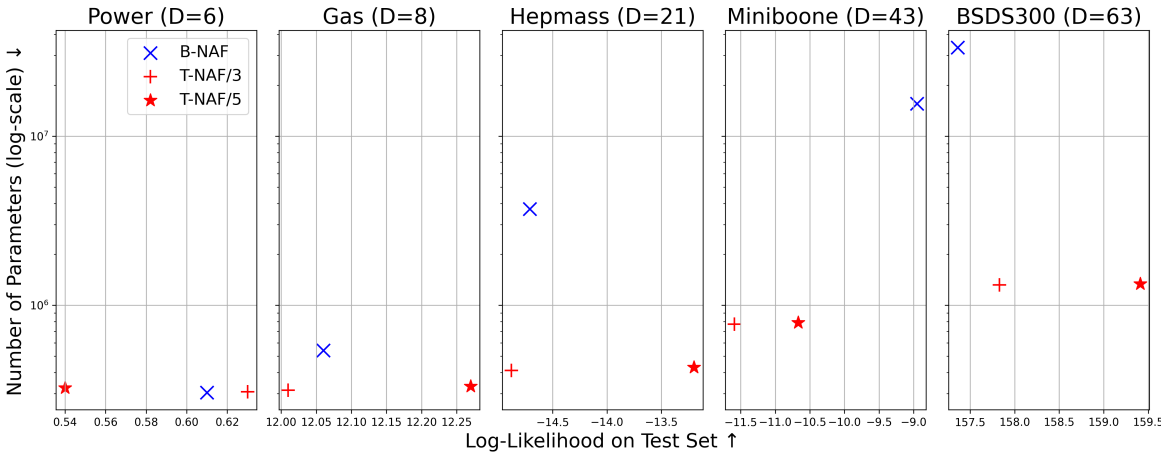

Figure 2. Trade-off between number of parameters and performance. Vertical axis represents the number of parameters in log-scale (lower is better), and horizontal axis represent the log-likelihood on the test set (higher is better). The optimal trade-off is represented by points in the bottom-right corner. Overall T-NAF offers a better trade-off w.r.t. B-NAF; the gap in number of parameters between T-NAF and B-NAF gets larger as the number of input dimensions D increases.

Table 2. Ablation experiments on head type. CDF: the parameters of a CDF are generated by the transformer for each output separately; Shared-CDF: conditional CDF with parameters shared across outputs and conditional input provided by the transformer embeddings; Spline Flow: the parameters of a spline flow are generated by the transformer for each output separately. Log-Likelihood (higher is better) for 4 datasets from UCI and for BSDS300. Average and standard deviation over 3 seeds. We report number of dimensions (D) and dataset size (N, Millions).

Head Type	Layers	POWER $D=6$ $N=2.0$	GAS $D=8$ $N=1.0$	HEPMASS $D=21$ $N=0.5$	MINIBOONE $D=43$ $N=0.04$	BSDS300 $D=63$ $N=1.3$
CDF	3	0.63±0.01	12.01±0.02	-14.88±0.37	-11.59±0.67	157.83±0.10
Shared-CDF	3	0.52±0.02	11.12±0.09	-14.97±0.17	-11.34±0.03	158.10±0.07
Spline Flow	3	**0.64**±0.01	11.99±0.03	-14.05±0.14	-11.08±0.04	159.05±0.07
CDF	5	0.54±0.01	**12.27**±0.01	**-13.20**±0.26	**-10.67**±0.06	**159.41**±0.04
Spline Flow	5	**0.65**±0.01	12.20±0.03	**-13.15**±0.19	-11.09±0.01	**159.47**±0.02

ing stability issues inherent in traditional autoregressive models, but also paves the way for more sophisticated and nuanced modeling of complex data distributions. Future work can explore further optimizations and applications of T-NAFs, potentially expanding their utility in broader machine learning and data science contexts.

Acknowledgements

Funding in direct support of this work: Massimiliano Patacchiola, Aliaksandra Shysheya, and Richard E. Turner are supported by an EPSRC Prosperity Partnership EP/T005386/1 between the EPSRC, Microsoft Research and the University of Cambridge.

References

Archer, N. P. and Wang, S. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, 1993.

Daniels, H. and Velikova, M. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 2010.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, 2022.

De Cao, N., Aziz, W., and Titov, I. Block neural autoregressive flow. In *Uncertainty in Artificial Intelligence*, 2020.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

Dolatabadi, H. M., Erfani, S., and Leckie, C. Invertible generative modeling using linear rational splines. In *International Conference on Artificial Intelligence and Statistics*, 2020.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. In *Advances in Neural Information Processing Systems*, 2019.

Fakoor, R., Chaudhari, P., Mueller, J., and Smola, A. J. Trade: Transformers for density estimation. *arXiv preprint arXiv:2004.02441*, 2020.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, 2015.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

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

Jaini, P., Selby, K. A., and Yu, Y. Sum-of-squares polynomial flow. In *International Conference on Machine Learning*, 2019.

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

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*, 2016.

Kobyzev, I., Prince, S., and Brubaker, M. A. Normalizing flows: Introduction and ideas. *arXiv preprint arXiv:1908.09257*, 2019.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. *arXiv preprint arXiv:2309.06180*, 2023.

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 *International Conference on Computer Vision*, 2001.

Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. Neural importance sampling. *CoRR*, 2018.

Oliva, J., Dubey, A., Zaheer, M., Poczos, B., Salakhutdinov, R., Xing, E., and Schneider, J. Transformation autoregressive networks. In *International Conference on Machine Learning*, 2018.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, 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*, 2021.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International Conference on Machine Learning*, 2015.

Sill, J. Monotonic networks. *Advances in Neural Information Processing Systems*, 1997.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Wehenkel, A. and Louppe, G. Unconstrained monotonic neural networks. In *Advances in Neural Information Processing Systems*, 2019.

Xu, C., Cheng, X., and Xie, Y. Normalizing flow neural networks by jko scheme. *Advances in Neural Information Processing Systems*, 36, 2024.