Graphical Normalizing Flows

Antoine Wehenkel
University of Liège
antoine.wehenkel@uliege.be

Gilles Louppe
University of Liège
g.louppe@uliege.be

Abstract

Normalizing flows model complex probability distributions by combining a base distribution with a series of bijective neural networks. State-of-the-art architectures rely on coupling and autoregressive transformations to lift up invertible functions from scalars to vectors. In this work, we revisit these transformations as probabilistic graphical models, showing that a flow reduces to a Bayesian network with a pre-defined topology and a learnable density at each node. From this new perspective, we propose the graphical normalizing flow, a new invertible transformation with either a prescribed or a learnable graphical structure. This model provides a promising way to inject domain knowledge into normalizing flows while preserving both the interpretability of Bayesian networks and the representation capacity of normalizing flows. We demonstrate experimentally that normalizing flows built on top of graphical conditioners are competitive density estimators. Finally, we illustrate how inductive bias can be embedded into normalizing flows by parameterizing graphical conditioners with convolutional networks.

1 Introduction

Normalizing flows [NFs, Rezende and Mohamed, 2015] have proven to be an effective way to model complex data distributions with neural networks. These models map data points to latent variables through an invertible function while keeping track of the change of density caused by the transformation. In contrast to variational auto-encoders (VAEs) and generative adversarial networks (GANs), NFs provide access to the exact likelihood of the data under the model, hence offering a sound and direct way to optimize the network parameters. Normalizing flows have proven to be of practical interest as demonstrated by Oord et al. [2018] for speech synthesis, by Rezende and Mohamed [2015] for variational inference or by Papamakarios et al. [2019b] for simulation-based inference. Yet, their usage as a base component of the machine learning toolbox is still limited in comparison to GANs or VAEs. Recent efforts have been made by Papamakarios et al. [2019a] to define the fundamental principles of flow design and by Durkan et al. [2019] to provide coding tools for modular implementations. We argue that normalizing flows would gain in popularity by offering stronger inductive bias as well as more interpretability.

Sometimes forgotten in favor of more data oriented methods, probabilistic graphical models (PGMs) have been popular for modeling complex data distributions while being relatively simple to build and read [Koller and Friedman, 2009, Johnson et al., 2016]. Among PGMs, Bayesian networks [BNs, Pearl and Russell, 2011] offer a balance between modeling capacity and simplicity. Most notably, these models have been at the basis of expert systems before the big data era (e.g. Diez et al. [1997], Kahn et al. [1997], Seixas et al. [2014]) and were commonly used to merge qualitative expert knowledge and quantitative information together. On the one hand, experts stated independence assumptions that should be encoded by the structure of the network. On the other hand, data were used to estimate the parameters of the conditional probabilities/densities encoding the quantitative aspects of the data distribution. These models have progressively received less attention from the machine learning community because of the difficulty to scale to high-dimensional datasets.

Preprint. Under review.
Driven by the objective of bringing intuition into normalizing flows and the proven relevance of BNs for combining qualitative and quantitative reasoning, we summarize our contributions as follows:

- From the insight that coupling and autoregressive transformations can be reduced to Bayesian networks with a fixed topology, we introduce the more general graphical conditioner for normalizing flows, featuring either a prescribed or a learnable BN topology.
- We show that graphical normalizing flows perform well in a large variety of low and high-dimensional tasks. They are not only competitive as a black-box normalizing flow, but also provide an operational way to introduce domain knowledge into neural network based density estimation.

2 Background

Normalizing flows A normalizing flow is defined as a sequence of invertible transformation steps \( g_k : \mathbb{R}^d \rightarrow \mathbb{R}^d \) composed together to create an expressive invertible mapping \( g = g_1 \circ \cdots \circ g_K : \mathbb{R}^d \rightarrow \mathbb{R}^d \). This mapping can be used to perform density estimation, using \( g(\cdot; \theta) : \mathbb{R}^d \rightarrow \mathbb{R}^d \) to map a sample \( x \in \mathbb{R}^d \) onto a latent vector \( z \in \mathbb{R}^d \) equipped with a density \( p_z(z) \). The transformation \( g \) implicitly defines a density \( p(x; \theta) \) as given by the change of variables formula,

\[
p(x; \theta) = p_z(g(x; \theta)) \left| \det J_{g(x; \theta)} \right|,
\]

where \( J_{g(x; \theta)} \) is the Jacobian of \( g(x; \theta) \) with respect to \( x \). The resulting model is trained by maximizing the likelihood of the training dataset \( X = \{x^1, ..., x^N\} \). Unless needed, we will not distinguish between \( g \) and \( g_k \) for the rest of our discussion.

In general the steps \( g \) can take any form as long as they define a bijective map. Here, we focus on a sub-class of normalizing flows for which the steps can be mathematically described as follows:

\[
g(x) = \begin{bmatrix} g^1(x; c^1(x)) & \cdots & g^i(x; c^i(x)) & \cdots & g^d(x;d; c^d(x)) \end{bmatrix},
\]

where the \( c^i \) are the conditioners which role is to constrain the structure of the Jacobian of \( g \). The functions \( g^i \), partially parameterized by their conditioner, must be invertible with respect to their input variable \( x_i \). They are often referred to as transformers, however in this work we will use the term normalizers to avoid any confusion with attention-based transformer architectures.

The conditioners examined in this work can be combined with any normalizers. In particular, we consider affine and monotonic normalizers. An affine normalizer \( g : \mathbb{R} \times \mathbb{R}^2 \rightarrow \mathbb{R} \) can be expressed as \( g(x; m, s) = x \exp(s) + m \), where \( m \in \mathbb{R} \) and \( s \in \mathbb{R} \) are computed by the conditioner. There exist multiple methods to parameterize monotonic normalizers, but in this work we rely on Unconstrained Monotonic Neural Networks [UMNNs, Wehenkel and Louppe, 2019] which can be expressed as \( g(x; c) = \int_{0}^{c} f(t, c)dt + \beta(c) \), where \( c \in \mathbb{R}^{|c|} \) is an embedding made by the conditioner and \( f : \mathbb{R}^{|c|+1} \rightarrow \mathbb{R}^{+} \) is a neural network with a strictly positive scalar output.

Bayesian networks A Bayesian network is a directed acyclic graph (DAG) that represents independence assumptions among the components of a random vector. Formally, let \( x = [x_1, ..., x_d]^T \in \mathbb{R}^d \) be a random vector distributed under \( p_x \). A BN associated to \( x \) is a directed acyclic graph made of \( d \) vertices representing the components \( x_i \) of \( x \). In this kind of network, the absence of edges models conditional independence between groups of components through the concept of d-separation [Geiger et al., 1990]. A BN is a valid representation of a random vector \( x \) iff its density can be factorized as:

\[
p_x(x) = \prod_{i=1}^{d} p(x_i|\mathcal{P}_i),
\]

where \( \mathcal{P}_i = \{ j : A_{i,j} = 1 \} \) denotes the set of parents of the vertex \( i \) and \( A \in \{0,1\}^{d \times d} \) is the adjacency matrix of the BN. As an example, Fig. 1a is a valid BN for any distribution over \( x \) because it does not state any independence and leads to a factorization that corresponds to the chain rule. However, in practice we seek for a sparse and valid BN which models most of the independence between the components of \( x \) and leads to an efficient factorization of the modeled probability distribution.
3 Normalizing flows as Bayesian networks

Autoregressive conditioners To be of practical use, NFs must be composed of transformations for which the determinant of the Jacobian can be computed efficiently, otherwise its evaluation would be running in $\mathcal{O}(d^3)$. A common solution is to use autoregressive conditioners, i.e., such that the conditioners

$$c^i(x) = h^i \left( [x_1 \cdots x_{i-1}]^T \right)$$

are functions $h^i$ of the first $i - 1$ components of $x$. This particular form constrains the Jacobian of $g$ to be lower triangular, which makes the computation of its determinant $\mathcal{O}(d)$.

For the multivariate density $p(x; \theta)$ induced by $g(x; \theta)$ and $p_z(z)$, we can use the chain rule to express the joint probability of $x$ as a product of $d$ univariate conditional densities,

$$p(x; \theta) = p(x_1; \theta) \prod_{i=2}^d p(x_i|x_{1:i-1}; \theta). \quad (4)$$

When $p_z(z)$ is a factored distribution $p_z(z) = \prod_{i=1}^d p(z_i)$, we identify that each component $z_i$ coupled with the corresponding function $g^i$ encodes for the conditional $p(x_i|x_{1:i-1}; \theta)$. Therefore, and as illustrated in Fig. 1a, autoregressive transformations can be seen as a way to model the conditional factors of a BN that does not state any independence. This becomes clear if we define $\mathcal{P}_i = \{x_1, \ldots, x_{i-1}\}$ and compare (4) with (3).

The complexity of the conditional factors strongly depends on the ordering of the vector components. While not hurting the universal representation capacity of normalizing flows, the agnostic ordering used in autoregressive transformations leads to poor inductive bias and to factors that are most of the time difficult to learn. In practice, one often alleviates the arbitrariness of the ordering by stacking multiple autoregressive transformations combined with random permutations on top of each other.

Coupling conditioners Coupling layers [Dinh et al., 2017] are another popular type of conditioners used in normalizing flows. The conditioners $c^i$ made from coupling layers are defined as

$$c^i(x) = \begin{cases} h^i & \text{if } i \leq k \\ h^i \left( [x_1 \cdots x_k]^T \right) & \text{if } i > k \end{cases}$$

where the underlined $h^i \in \mathbb{R}^{|\mathcal{C}|}$ denote constant values and $k \in \{1, \ldots, d\}$ is a hyper-parameter usually set to $\lceil \frac{d}{2} \rceil$. As for autoregressive conditioners, the Jacobian of $g$ made of coupling layers is lower triangular. Again, and as shown in Fig. 1b and 1c, these transformations can be seen as a specific class of BN where $\mathcal{P}_i = \{\}$ for $i \leq k$ and $\mathcal{P}_i = \{x_1, \ldots, x_k\}$ for $i > k$. D-separation can be used to read off the independencies stated by this class of BNs such as the conditional independence between each pair in $x_{k+1:d}$ knowing $x_{1:k}$. For this reason, and in contrast to autoregressive transformations, coupling layers are not by themselves universal density approximators even when associated with very expressive normalizers $g^i$. In practice, these structural independencies can be relaxed by stacking multiple layers, although they can also lead to useful inductive bias, such as in the multi-scale architecture with checkerboard masking [Kingma and Dhariwal, 2018].
4 Graphical normalizing flow

4.1 Graphical conditioner

Following up on the previous discussion, we introduce the graphical conditioner architecture. We motivate our approach by observing that the topological ordering (or ancestral ordering) of any BN leads to a lower triangular adjacency matrix whose determinant computation is $O(d)$. Therefore, conditioning factors $c^i(x)$ selected by following a BN adjacency matrix necessarily leads to a transformation $g$ whose Jacobian determinant remains efficient to compute. We provide a formal proof of this result in Appendix B.

Formally, given a BN with adjacency matrix $A \in \{0, 1\}^{d \times d}$, we define the graphical conditioner as being

$$c^i(x) = h^i(x \odot A_{i,:}),$$

(5)

where $x \odot A_{i,:}$ is the element-wise product between the vectors $x$ and the one made of the $i$th row of $A$ – i.e., $A_{i,:}$ is used to mask on $x$. This new conditioner architecture is illustrated in Fig. 2 and leads to NFs that can be inverted by sequentially inverting each component in the topological ordering.

The graphical conditioner architecture can be used to learn the conditional factors in a continuous BN while elegantly setting structural independencies prescribed from domain knowledge. We also now notice how coupling and autoregressive conditioners are special cases in which the adjacency matrix reflects the classes of BNs discussed in Section 3.

4.2 Learning the topology

In many cases, defining the whole structure of a BN is not possible due to a lack of knowledge about the problem at hand. Fortunately, not only the density at each node is learnable, but also the DAG structure itself: defining an arbitrary topology and ordering, as it is implicitly the case for autoregressive and coupling conditioners, is unnecessary.

Building upon the work of Zheng et al. [2018], we convert the combinatorial optimization of score-based learning of a DAG into a continuous optimization by relaxing the domain of $A$ to real numbers instead of boolean values. That is,

$$\min_{A \in \mathbb{R}^{d \times d}} F(A) \quad \iff \quad \min_{A \in \mathbb{R}^{d \times d}} F(A) \quad \text{subject to} \quad w(A) = 0,$$

(6)

where $G(A)$ is the graph with adjacency matrix $A$ and $F : \mathbb{R}^{d \times d} \to \mathbb{R}$ is the likelihood function evaluated through the NF $g$. The constraint $w(A)$ is expressed as [Yu et al., 2019]

$$w(A) = \text{tr} \left( \sum_{i=1}^{d} A^i \right) = \text{tr} \left( (I + A)^d \right) - d = 0.$$

In the case of positively valued $A$, an element $(i, j)$ of $A^k = \underbrace{AA \ldots AA}_{k \text{ terms}}$ is equal to zero if and only if there exists no path going from node $j$ to node $i$ that is made of $k$ edges. Intuitively $w(A)$ expresses to which extent the graph is cyclic. Indeed, the element $(i, j)$ of $A^k$ will be as larger as there are low cost paths (i.e., paths made of edges that corresponds to large values in $A$) of length $k$ from $j$ to $i$.

In comparison to our work, Zheng et al. [2018] use the quadratic loss on the corresponding linear structural equation model (SEM) as the score function $F(A)$. By attaching normalizing flows to topology learning, our method does not make strong assumptions on the form of the conditional factors because it has a continuously adjustable level of complexity as set by the capacity of the normalizers $g^i$ and the functions $h^i$.

In order to learn the BN topology from the data, the adjacency matrix must be relaxed to contain real values instead of binary ones and to be temporarily (during training) not defining a DAG. Directly plugging the real matrix $A$ in (5) would be impractical because the quantity of information going from node $j$ to node $i$ would not continuously relate to the value of $A_{i,j}$. Indeed, the information would either be null if $A_{i,j} = 0$ or pass completely if not. Instead, a natural solution consists in
adding noise that is proportional to the absolute value of $A_{i,j}$. To this end, we create the stochastic pseudo binary valued matrix $A'$ from $A$, defined as:

$$A'_{i,j} = \frac{e^{\log(p(x|A')) + \gamma_1}}{e^{\log(p(x|A')) + \gamma_1} + e^{\log(1-p(x|A')) + \gamma_2}}, \quad \gamma_1, \gamma_2 \sim \text{Gumble},$$

where $\sigma(a) = 2(\text{sigmoid}(2a^2) - \frac{1}{2})$ normalizes the values of $A$ between 0 and 1, being close to 1 for large values and close to zero for values close to 0. The hyper-parameter $T$ controls the sampling temperature and is fixed to 0.5 in all our experiments. If $A_{i,j}$ is large, this transformation maps a real value to a large value with high probability and to a small value with low probability; if $A_{i,j}$ is small, it is the other way around. In contrast to directly using the matrix $A$, this stochastic transformation allows to create a direct and continuous relationship between the weight of the edges and the quantity of information that can transit from node $j$ to node $i$.

### 4.3 Optimization

We rely on the augmented Lagrangian approach to solve the constrained optimization problem (6) as initially proposed by Zheng et al. [2018]. This optimization procedure requires solving iteratively the following sub-problems:

$$\max_{A,\lambda_t, \mu_t} E_{\gamma_1, \gamma_2} [L(A, \lambda_t, \mu_t)] := E_{\gamma_1, \gamma_2} [F(A)] - \lambda_t w(A) - \frac{\mu_t}{2} w(A)^2,$$

where $\lambda_t$ and $\mu_t$ respectively denote the Lagrangian and penalty coefficients of the sub-problem $t$. Plugging in for $F$ the likelihood given by $g$, we arrive to the following sequence of problems:

$$\max_{\theta, A, \lambda_t, \mu_t} E_{\gamma_1, \gamma_2} [\mathcal{L}(\theta, A, \lambda_t, \mu_t)]$$

$$= E_{\gamma_1, \gamma_2} \left[ \sum_{i=1}^{N} \log(p_z(g(x^i; \theta) | \det J_{g(x^i; \theta)}) - \lambda_t w(A) - \frac{\mu_t}{2} w(A)^2 \right]$$

$$= E_{\gamma_1, \gamma_2} \left[ \sum_{i=1}^{N} \left( \log(p_z(g(x^i; \theta)) + \sum_{j=1}^{d} \left[ \frac{\partial g(x^i; c(x \cdot A'_{:j}, \theta))}{\partial x^i_j} \right] \right) \right] - \lambda_t w(A) - \frac{\mu_t}{2} w(A)^2,$$

where $N$ is the number of training samples $x^i$. We solve these optimization problems by mini-batch stochastic gradient ascent. It is worth noting that during training the graph represented by the matrix $A$ can be cyclic making the computation of the log-likelihood as above only approximately correct. However, the method is shown to converge to a solution that satisfies the constraint, making the final optimization steps correct in terms of the likelihood. The exact optimization procedure used for our experiments is provided in Appendix A.

### 5 Experiments

#### 5.1 Conditioners study on tabular datasets

In these experiments, we compare autoregressive, coupling and graphical conditioners on benchmark tabular datasets for density estimation. We evaluate each conditioner in combination with monotonic and affine normalizers. We only compare NFs with a single transformation step because our focus is on the conditioner capacity. To provide a fair comparison we have fixed in advance the neural architectures used to parameterize the normalizers and conditioners as well as the training parameters by taking inspiration from those used by Wehenkel and Louppe [2019] and Papamakarios et al. [2017]. The variable ordering of each dataset is chosen as initially proposed by Papamakarios et al. [2017]. All hyper-parameters are provided in Appendix C and a public implementation will be released on Github.

First, Table 1 presents the test negative log-likelihood obtained by each architecture. These results indicate that graphical conditioners offer the best performance in general. Unsurprisingly, coupling layers show the worse performance, due to the arbitrarily assumed independencies. Autoregressive and graphical conditioners show very similar performance for monotonic normalizers, the latter being slightly better on 4 out of the 5 datasets. To contextualize the discussion the table also provides...
results obtained with 5-step NFs composed of autoregressive conditioners combined with affine [Papamakarios et al., 2017] and monotonic normalizers. Comparing the results together, we see that while additional steps lead to noticeable improvements for affine normalizers, benefits are questionnable for monotonic transformations.

Second, Table 2 presents the number of edges in the BN associated with each flow. For POWER and GAS, the number of edges found by the graphical conditioners is close or equal to the maximum number of edges. Interestingly, graphical conditioners outperform autoregressive conditioners on these two tasks, demonstrating the value of finding an appropriate ordering particularly when using affine normalizers. Moreover, graphical conditioners correspond to BNs whose sparsity is largely greater than for autoregressive conditioners while providing equivalent if not better performance.

### 5.2 Density estimation of images

We now demonstrate how graphical conditioners can be used to fold in domain knowledge into NFs by performing density estimation on MNIST images. The design of the graphical conditioner is adapted to images by parameterizing the functions $h^i$ with convolutional neural networks (CNNs) whose parameters are shared for all $i \in \{1, ..., d\}$ as illustrated in Fig. 2. Inputs to the network $h^i$ are masked images specified by both the adjacency matrix $A$ and the entire input image $x$. Using a CNN together with the graphical conditioner allows for an inductive bias suitably designed for processing images. We consider single step normalizing flows whose conditioners are either coupling, autoregressive or graphical-CNN as described above, each combined with either affine or monotonic normalizers. The graphical conditioners that we use include an additional inductive bias that enforces a sparsity constraint on $A$ and which prevents a pixel’s parents to be too distant from their descendants in the images. Formally, given a pixel located at $(i, j)$, only the pixels $(i \pm l_1, j \pm l_2)$, $l_1, l_2 \in \{1, ..., L\}$ are allowed to be its parents.

Results reported in Table 3 show that graphical conditioners lead to the best performing affine NFs even if they are made of a single step. This performance gain can probably be attributed to the combination of both learning a masking scheme and processing the result with a convolutional network. These results also show that when the capacity of the normalizers is limited, finding a meaningful factorization is very effective to improve performance. The number of edges in the equivalent BN is about two orders of magnitude smaller than for coupling and autoregressive conditioners. This sparsity is beneficial for the inversion since the evaluation of the inverse of the flow requires a number of steps equal to the depth [Bezek, 2016] of the equivalent BN. Indeed, we find that while obtaining density models that are as expressive, the computation complexity to generate samples is approximately divided by $\frac{2 \times 784}{100} \approx 40$ in comparison to the autoregressive flows made of 5 steps and comprising many more parameters.

Fig. 4 shows random samples generated with the architectures we trained on MNIST. These few samples are representative of the beneficial inductive bias produced by the combination of the graphical conditioner with a CNN. While the global structure of the images produced by the different architectures is of similar quality, the images produced by graphical conditioners are more consistent locally. We believe that the possibility granted by graphical conditioners to introduce domain knowl-
Masking operation from the adjacency matrix.

\[ A_{ij} : \]

Figure 2: Illustration of how a graphical conditioner’s output \( c_i(x) \) is computed for images. The sample \( x \), on the left, is an image of a 4. The stripes denote the pixel \( x_i \). The parents of \( x_i \) in the learned DAG are shown as white pixels on the mask \( A_{ij} \), the other pixels are in black. The element-wise product between the image \( x \) and the mask \( A_{ij} \) is processed by a convolutional neural network that produces the embedding vector \( c_i(x) \) conditioning the pixel \( x_i \).

| Model           | Neg. LL | Parameters | Edges   | Depth |
|-----------------|---------|------------|---------|-------|
| (a) G-Affine (1)| 1.84 ± 0.01 | 1 x 10^6  | 5016    | 103   |
| (b) G-Monotonic (1)| 1.17 ± 0.03 | 1 x 10^6  | 2928    | 125   |
| (c) A-Affine (1)| 2.12 ± 0.02 | 3 x 10^6  | 306936  | 783   |
| (d) A-Monotonic (1)| 1.37 ± 0.04 | 3.1 x 10^6 | 306936  | 783   |
| (e) C-Affine (1)| 2.39 ± 0.03 | 3 x 10^6  | 153664  | 1     |
| (f) C-Monotonic (1)| 1.67 ± 0.08 | 3.1 x 10^6 | 153664  | 1     |
| (g) A-Affine (5)| 1.89 ± 0.01 | 6 x 10^6  | 5 x 306936 | 5 x 783 |
| (h) A-Monotonic (5)| 1.13 ± 0.02 | 6.6 x 10^6 | 5 x 306936 | 5 x 783 |

Table 3: Results on MNIST. The negative log-likelihood is reported in bits per pixel on the test set over 3 runs on MNIST, error bars are equal to the standard deviation. The number of edges and the depth of the equivalent Bayesian network is reported. Results are divided into 3 categories: (a) The architectures introduced in this work. (b) Classical single-step architectures. (c) The best performing architectures based on multi-steps autoregressive flows.

6 Discussion

Bayesian network topology learning Formal BN topology learning has extensively been studied for more than 30 years and many strong theoretical results on the computational complexity have been obtained. Most of these results however focus on discrete random variables, and how they generalize in the continuous case is yet to be explained. The topic of BN topology learning for discrete variables has been proven to be NP-hard by Chickering et al. [2004]. However, while some greedy algorithms exist, they do not lead in general to a minimal I-map although allowing for an efficient factorization of random discrete vectors distributions in most of the cases. These algorithms

edge could be beneficial for a large panel of density estimation applications. Fig. 3 shows the in and out degrees of the equivalent BN learned on MNIST by a NF made of graphical conditioners. It can be observed that most of the connections in the equivalent BN come from the center of the images. In a certain sense, this shows that the graphical conditioner has successfully discovered the most informative pixels and made sense of their relationships.

These experiments show that, in addition to be a favorable tool for introducing inductive bias into NFs, graphical conditioners open the possibility to build BNs for large datasets. Unlocking the BN machinery for modern datasets and computing infrastructures.
are usually separated between the constrained-based family such as the PC algorithm [Spirtes et al., 2001] or the incremental association Markov blanket [Koller and Friedman, 2009] and the score-based family as used in the present work. Finding the best BN topology for continuous variables has not been proven to be NP-hard however the results for discrete variables suggest that without strong assumptions on the function class the problem is hard.

The recent progress made in the continuous setting relies on the heuristic used in score-based methods. In particular, Zheng et al. [2018] showed that the acyclicity constraint required in BNs can be expressed as a continuous function of the adjacency matrix, allowing Lagrangian formulation to be used. Yu et al. [2019] proposed DAG-GNN, a follow up work of Zheng et al. [2018] which relies on variational inference and auto-encoders to generalize the method to non-linear structural equation models. Further investigation of continuous DAG learning in the context of causal models was achieved by Lachapelle et al. [2019]. They use the adjacency matrix of the causal network as a mask over neural networks to design a score which is the log-likelihood of a parameterized normal distribution. The requirement to pre-define a parametric distribution before learning restricts the factors to simple conditional distribution. In contrast, our method combines the constraints given by the BN topology with NFs which are free-form universal density estimators. Remarkably, their method leads to an efficient one-pass computation of the joint density, however this neural masking technique could also be implemented for normalizing flow architectures such as already demonstrated by Papamakarios et al. [2017] and De Cao et al. [2019] for autoregressive conditioners.

**Shuffling between transformation steps** As already mentioned, consecutive transformation steps are often combined with randomly fixed permutations in order to mitigate the ordering problem. Linear flow steps [Oliva et al., 2018] generalize these fixed permutations. They are parameterized by a matrix $W = PLU$ where $P$ is the fixed permutation matrix, and $L$ and $U$ are respectively a lower and an upper triangular matrix. Although linear flow improves the simple permutation scheme, they do still rely on an arbitrary permutation. To the best of our knowledge, graphical conditioners are the first attempt to get completely rid of any fixed permutation in NFs.

**Inductive bias** Graphical conditioners can be seen as a way to learn the right mask for each conditional factor of a factored joint distribution. In this way, the conditioners process their input as they would process the full vector, which allows for an intuitive design of the neural network that is in charge of compressing the useful information contained in the conditioning factors into embeddings. We have shown experimentally that this effectively leads to performing and efficient parameterization for applying normalizing flows on images. In addition, we have shown that normalizing flows built from graphical conditioners combined with monotonic transformations leads to very expressive density estimators. In effect, this means that enforcing some a priori known independencies can be performed thanks to graphical normalizing flows without hurting their modeling capacity. We believe such models could be of high practical interest because they cope well with large dataset and complex distributions while preserving some readability through their equivalent BN.

**Causal networks and Structural Equation Models** The research in NFs could directly benefit causality and SEM thanks to graphical conditioners. Indeed, the graphical conditioners introduced in this work clearly state a BN structure while maintaining the great modeling capacity of normalizing flows. The results obtained by graphical conditioners combined with monotonic transformations on density estimation tasks empirically demonstrate that NFs with a direct BN interpretation are very competitive density estimators. This frees the possibility to combine causal networks and normalizing flows with or without a predefined (sub-)structure. Further research should be performed to work with interventional data by finding how do-calculus can be implemented in the training strategy of graphical conditioners, enhancing them with a causal interpretation.

**Conclusion** We have revisited coupling and autoregressive conditioners for normalizing flows as Bayesian networks. From this new perspective, we proposed the more general graphical conditioner architecture for normalizing flows. We have shown that this new architecture compares favorably with autoregressive and coupling conditioners on low and high-dimensional density estimation tasks. In addition, we have illustrated the opportunity offered by this new architecture to introduce domain knowledge into normalizing flows by combining convolutional neural networks with it. Finally, we believe that graphical conditioners could be of very strong interest to develop further research at the intersection of normalizing flows and causal networks.
7 Broader impact

In the context of density estimation and generative models, graphical normalizing flows have the potential to improve the way scientists may enforce domain assumptions through explicit and prescribed independencies. We believe they will also be helpful in uncovering and reasoning about independencies, or the lack thereof, found in data. More broadly, generative models carry a risk of being abused for the generation of fake data. This danger also applies to normalizing flows, including graphical normalizing flows.

Acknowledgments

The authors would like to thank Johann Brehmer and Louis Wehenkel for proofreading the manuscript. Antoine Wehenkel is a research fellow of the F.R.S.-FNRS (Belgium) and acknowledges its financial support. Gilles Louppe is recipient of the ULiège - NRB Chair on Big data and is thankful for the support of NRB.

References

M. Bezek. Characterizing dag-depth of directed graphs. *arXiv preprint arXiv:1612.04980*, 2016.

D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5(Oct):1287–1330, 2004.

N. De Cao, I. Titov, and W. Aziz. Block neural autoregressive flow. *arXiv preprint arXiv:1904.04676*, 2019.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. In *International Conference in Learning Representations*, 2017.

C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, pages 7509–7520, 2019.

F. Díez, J. Mira, E. Iturralde, and S. Zubillaga. DIAVAL, a Bayesian expert system for echocardiography. *Artif. Intell. Med.*, 10(1):59–73, May 1997. URL https://doi.org/10.1016/s0933-3657(97)00384-9.

D. Geiger, T. Verma, and J. Pearl. d-separation: From theorems to algorithms. In *Uncertainty in Artificial Intelligence*, volume 10, pages 139–148. Elsevier, 1990. URL https://doi.org/10.1016/b978-0-444-88738-2.50018-x.

M. J. Johnson, D. K. Duvenaud, A. Wiltschko, R. P. Adams, and S. R. Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.

C. E. Kahn, L. M. Roberts, K. A. Shaffer, and P. Haddawy. Construction of a Bayesian network for mammographic diagnosis of breast cancer. *Comput. Biol. Med.*, 27(1):19–29, Jan. 1997. URL https://doi.org/10.1016/s0010-4825(96)00039-x.

D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018.

D. Koller and N. Friedman. *Probabilistic graphical models: Principles and techniques*. MIT press, 2009.

S. Lachapelle, P. Brouillard, T. Deleu, and S. Lacoste-Julien. Gradient-based neural dag learning. *arXiv preprint arXiv:1906.02226*, 2019.

J. Oliva, A. Dubey, M. Zaheer, B. Poczos, R. Salakhutdinov, E. Xing, and J. Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3895–3904, 2018.
A. Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. Driessche, E. Lockhart, L. Cobo, F. Stimberg, et al. Parallel WaveNet: Fast high-fidelity speech synthesis. In International Conference on Machine Learning, pages 3915–3923, 2018.

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In Advances in Neural Information Processing Systems, pages 2338–2347, 2017.

G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. arXiv preprint arXiv:1912.02762, 2019a.

G. Papamakarios, D. C. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In 22nd International Conference on Artificial Intelligence and Statistics (AISTATS), 2019b.

J. Pearl and S. Russell. Bayesian networks. California Digital Library, 2011.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In International Conference on Machine Learning, pages 1530–1538, 2015.

R. Sedgewick and K. Wayne. Algorithms. Addison-wesley professional, 2011.

F. L. Seixas, B. Zadrozny, J. Laks, A. Conci, and D. C. Muchaluat Saade. A Bayesian network decision model for supporting the diagnosis of dementia, alzheimer’s disease and mild cognitive impairment. Comput. Biol. Med., 51:140–158, Aug. 2014. URL https://doi.org/10.1016/j.compbiomed.2014.04.010.

P. Spirtes, C. Glymour, and R. Scheines. Causation, Prediction, and Search. The MIT Press, 2001. URL https://doi.org/10.7551/mitpress/1754.001.0001.

A. Wehenkel and G. Louppe. Unconstrained monotonic neural networks. In Advances in Neural Information Processing Systems, pages 1543–1553, 2019.

Y. Yu, J. Chen, T. Gao, and M. Yu. DAG-GNN: Dag structure learning with graph neural networks. In International Conference on Machine Learning, pages 7154–7163, 2019.

X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing. DAGs with NO TEARS: Continuous optimization for structure learning. In Advances in Neural Information Processing Systems, pages 9472–9483, 2018.
A Optimization procedure

Algorithm 1 Main Loop

[118x664]epoch ← 0
while !Stopping criterion do
    foreach batch $X \in X$ do
        loss ← $-ll + \text{COMPUTELOSS}(\text{flow}, X)$
        OPTIMIZE($\text{flow}, \text{loss}$)
        epoch ← epoch + 1
        UPDATE COEFFICIENTS($\text{flow}, \text{epoch}$)
        if ISDAGCONSTRAINTNULL($\text{flow}$) then
            POSTPROCESS($\text{flow}$)

The method COMPUTELOSS($flow, X$) is computed as described by Eq. (7). The OPTIMIZE($\text{flow}, \text{loss}$) method performs a backward pass and an optimization step with the chosen optimizer (Adam in our experiments). The post-processing is performed by POSTPROCESS($\text{flow}$) and consists in thresholding the values in $A$ such that the values below a certain threshold are set to 0 and the other values to 1, after post-processing the stochastic door is deactivated. The threshold is the smallest real value that makes the equivalent graph acyclic. In addition, when the value of $d$ is large such as in MNIST experiments, the value of the constraint $w(A)$ is approximated by $\tilde{w}(A) = (I + \alpha A)^{\tilde{d}}$ where $\tilde{d} < d$ is increased to $d$ along the optimization. The combination of $\tilde{d}$ and $\alpha$ helps avoiding the explosion of $w(A)$. The update of the Lagrangian and penalty coefficients is performed at intervals separated by a fixed number of epochs and as described by Yu et al. [2019] in equations (14) to (16).

B Jacobian of graphical conditioners

Proposition B.1. The absolute value of the determinant of the Jacobian of a normalizing flow step based on graphical conditioners is equal to the product of its diagonal terms.

Proof. Proposition B.1 A Bayesian Network is a directed acyclic graph. Sedgewick and Wayne [2011] showed that every directed acyclic graph has a topological ordering, it is to say an ordering of the vertices such that the starting endpoint of every edge occurs earlier in the ordering than the ending endpoint of the edge. Let us suppose that an oracle gives us the permutation matrix $P$ that orders the components of $g$ in the topological defined by $A$. Let us introduce the following new transformation $g_P(x_P) = Pg(P^{-1}(P x))$ on the permuted vector $x_P = Px$. Thus the Jacobian of the transformation $g_P$ (with respect to $x_P$) is lower triangular with diagonal terms given by the derivative of the normalizers with respect to their input component. The determinant of such Jacobian is equal to the product of the diagonal terms. Finally, we have

$$|\det(J_{g_P(x_P)})| = |\det(P)| |\det(J_{g(x)})| |\det(P)|^-1$$

$$= |\det(J_{g(x)})|,$$

because of (1) the chain rule; (2) The determinant of the product is equal to the product of the determinants; (3) The determinant of a permutation matrix is equal to 1 or $-1$. The absolute value of the determinant of the Jacobian of $g$ is equal to the absolute value of the determinant of $g_P$, the latter given by the product of its diagonal terms that are the same as the diagonal terms of $g$. Thus the absolute value of the determinant of the Jacobian of a normalizing flow step based on graphical conditioners is equal to the product of its diagonal terms.

C Tabular density estimation - Training parameters

Table 4 provides the hyper-parameters used to train the normalizing flows for the tabular density estimation tasks. In our experiments we parameterize the functions $h^i$ with a unique neural network that takes a one hot encoded version of $i$ in addition to its expected input $x \odot A_i$. The embedding net
architecture corresponds to the network that computes an embedding of the conditioning variables for the coupling and DAG conditioners, for the autoregressive conditioner it corresponds to the architecture of the masked autoregressive network. The output of this network is equal to $2 (2 \times d$ for the autoregressive conditioner) when combined with an affine normalizer and to a hyper-parameter named embedding size when combined with a UMNN. The number of dual steps corresponds to the number of epoch between two updates of the DAGness constraint (performed as in Yu et al. [2019]).

| Dataset          | POWER | GAS  | HEPMASS | MINIBOONE | BSDS300 |
|------------------|-------|------|---------|-----------|---------|
| Batch size       | 2500  | 10000| 100     | 100       | 100     |
| Integral Net     | $3 \times 100$ | $3 \times 200$ | $3 \times 200$ | $3 \times 40$ | $3 \times 150$ |
| Embedding Net    | $3 \times 60$ | $3 \times 80$ | $3 \times 210$ | $3 \times 430$ | $3 \times 630$ |
| Embedding Size   | 30    | 30   | 30      | 30        | 30      |
| N° steps Dual Update | 30    | 100  | 25      | 200       | 20      |
| Learning Rate    | 0.001 | 0.001| 0.001   | 0.001     | 0.001   |
| Weight Decay     | $10^{-5}$ | $10^{-3}$ | $10^{-4}$ | $10^{-2}$ | $10^{-4}$ |

Table 4: Training configurations for density estimation tasks.

In addition, in all our experiments (tabular and MNIST) the integrand networks used to model the monotonic transformations have their parameters shared and receive an additional input that one hot encodes the index of the transformed variable. The models are trained until no improvement of the average log-likelihood on the validation set is observed for 10 consecutive epochs.

### D MNIST density estimation - Training parameters

For all experiments the batch size was 100, the learning rate $10^{-3}$, the weight decay $10^{-5}$. For the graphical conditioners the number of epochs between two coefficient updates was chosen to 10, the greater this number the better were the performance however the longer is the optimization. The CNN is made of 2 layers of 16 convolutions with $3 \times 3$ kernels followed by an MLP with two hidden layers of size 2304 and 128. The neural network used for the Coupling and the autoregressive conditioner are neural networks with $3 \times 1024$ hidden layers. For all experiments with a monotonic normalizer the size of the embedding was chosen to 30 and the integral net was made of 3 hidden layers of size 50. The models are trained until no improvements of the average log-likelihood on the validation set is observed for 10 consecutive epochs.