

---

# Graphical Normalizing Flows

---

Antoine Wehenkel  
ULiège

Gilles Louppe  
ULiège

## 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 they reduce to Bayesian networks 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 show that graphical conditioners discover relevant graph structure when we cannot hypothesize it. In addition, we analyze the effect of  $\ell_1$ -penalization on the recovered structure and on the quality of the resulting density estimation. Finally, we show that graphical conditioners lead to competitive white box density estimators. Our implementation is available at <https://github.com/AWehenkel/DAG-NF>.

## 1 Introduction

Normalizing flows [NFs, [Rezende and Mohamed, 2015](#), [Tabak et al., 2010](#), [Tabak and Turner, 2013](#), [Rippel and Adams, 2013](#)] 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 model’s parameters, hence offering a sound and direct way to optimize the network parameters. Normalizing flows have proven to be of practical interest as demonstrated by [Van Den Oord et al. \[2018\]](#), [Kim et al. \[2018\]](#) and [Prenger et al. \[2019\]](#) for speech synthesis, by [Rezende and Mohamed \[2015\]](#), [Kingma et al. \[2016\]](#) and [Van Den Berg et al. \[2018\]](#) for variational inference or by [Papamakarios et al. \[2019b\]](#) and [Greenberg et al. \[2019\]](#) 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. \[2019a\]](#) and [Kobyzev et al. \[2020\]](#) 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 an appealing 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. [[Díez 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 in favor of other methods

that scale better with the dimensionality of the data.

Driven by the objective of integrating intuition into normalizing flows and the proven relevance of BNs for combining qualitative and quantitative reasoning, we summarize our contributions as follows: **(i)** 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; **(ii)** We show that using a correct prescribed topology leads to improvements in the modeled density compared to autoregressive methods. When the topology is not known we observe that, with the right amount of  $\ell_1$ -penalization, graphical conditioners discover relevant relationships; **(iii)** In addition, we show that graphical normalizing flows perform well in a large variety of density estimation tasks compared to classical black-box flow architectures.

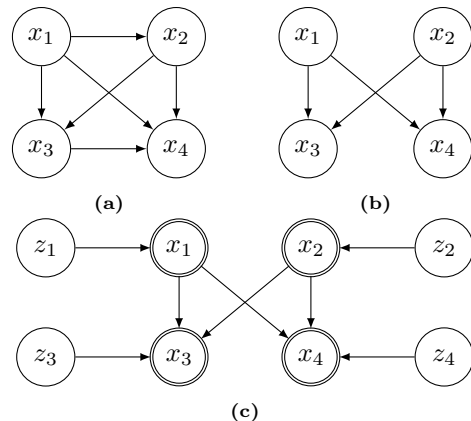
## 2 Background

**Bayesian networks** A Bayesian network is a directed acyclic graph (DAG) that represents independence assumptions between the components of a random vector. Formally, let  $\mathbf{x} = [x_1, \dots, x_d]^T \in \mathbb{R}^d$  be a random vector distributed under  $p_{\mathbf{x}}$ . A BN associated to  $\mathbf{x}$  is a directed acyclic graph made of  $d$  vertices representing the components  $x_i$  of  $\mathbf{x}$ . 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  $\mathbf{x}$  iff its density can be factorized as

$$p_{\mathbf{x}}(\mathbf{x}) = \prod_{i=1}^d p(x_i | \mathcal{P}_i), \quad (1)$$

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  $\mathbf{x}$  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  $\mathbf{x}$ , leading to an efficient factorization of the modeled probability distribution. It is worth noting that making hypotheses on the graph structure is equivalent to assuming certain conditional independence between some of the vector’s components.

**Normalizing flows** A normalizing flow is defined as a sequence of invertible transformation steps  $\mathbf{g}_k : \mathbb{R}^d \rightarrow \mathbb{R}^d$  ( $k = 1, \dots, K$ ) composed together to create



**Figure 1:** Bayesian networks equivalent to normalizing flows made of a single transformation step. (a) Autoregressive conditioner. (b) Coupling conditioner. (c) Coupling conditioner, with latent variables shown explicitly. Double circles stand for deterministic functions of the parents.

an expressive invertible mapping  $\mathbf{g} := \mathbf{g}_1 \circ \dots \circ \mathbf{g}_K : \mathbb{R}^d \rightarrow \mathbb{R}^d$ . This mapping can be used to perform density estimation, using  $\mathbf{g}(\cdot; \boldsymbol{\theta}) : \mathbb{R}^d \rightarrow \mathbb{R}^d$  to map a sample  $\mathbf{x} \in \mathbb{R}^d$  onto a latent vector  $\mathbf{z} \in \mathbb{R}^d$  equipped with a prescribed density  $p_{\mathbf{z}}(\mathbf{z})$  such as an isotropic Normal. The transformation  $\mathbf{g}$  implicitly defines a density  $p(\mathbf{x}; \boldsymbol{\theta})$  as given by the change of variables formula,

$$p(\mathbf{x}; \boldsymbol{\theta}) = p_{\mathbf{z}}(\mathbf{g}(\mathbf{x}; \boldsymbol{\theta})) |\det J_{\mathbf{g}(\mathbf{x}; \boldsymbol{\theta})}|, \quad (2)$$

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

In general the steps  $\mathbf{g}$  can take any form as long as they define a bijective map. Here, we focus on a subclass of normalizing flows for which the steps can be mathematically described as

$$\mathbf{g}(\mathbf{x}) = [g^1(x_1; \mathbf{c}^1(\mathbf{x})) \quad \dots \quad g^d(x_d; \mathbf{c}^d(\mathbf{x}))]^T, \quad (3)$$

where the  $\mathbf{c}^i$  are the **conditioners** which role is to constrain the structure of the Jacobian of  $\mathbf{g}$ . 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 normalizer. 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 [Huang et al., 2018, De Cao et al., 2020, Durkan et al., 2019, Jaini et al., 2019], but in this work we rely on Unconstrained Monotonic Neural Networks [UMNNs, Wehenkel and Louppe, 2019] which can be expressed as  $g(x; \mathbf{c}) = \int_0^x f(t, \mathbf{c}) dt + \beta(\mathbf{c})$ , where  $\mathbf{c} \in \mathbb{R}^{|\mathcal{C}|}$  is an embedding made by the conditioner and  $f : \mathbb{R}^{|\mathcal{C}|+1} \rightarrow \mathbb{R}^+$  and  $\beta : \mathbb{R}^d \rightarrow \mathbb{R}$  are two neural networks respectively with a strictly positive scalar output and a real scalar output. Huang et al. [2018] proved NFs built with autoregressive conditioners and monotonic normalizers are universal density approximators of continuous random variables.

### 3 Normalizing flows as Bayesian networks

**Autoregressive conditioners** Due to computing speed considerations, NFs are usually composed of transformations for which the determinant of the Jacobian can be computed efficiently, as otherwise its evaluation would scale cubically with the input dimension. A common solution is to use autoregressive conditioners, i.e., such that

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

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

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

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

When  $p_{\mathbf{z}}(\mathbf{z})$  is a factored distribution  $p_{\mathbf{z}}(\mathbf{z}) = \prod_{i=1}^d p(z_i)$ , we identify that each component  $z_i$  coupled with the corresponding functions  $g^i$  and embedding vectors  $\mathbf{c}^i$  encode for the conditional  $p(x_i | \mathbf{x}_{1:i-1}; \boldsymbol{\theta})$ . 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 but relies on a predefined node ordering. This becomes clear if we define  $\mathcal{P}_i = \{x_1, \dots, x_{i-1}\}$  and compare (4) with (1).

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 arbitrary 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., 2015] are another popular type of conditioners that lead to a bipartite structure. The conditioners  $\mathbf{c}^i$  made from coupling layers are defined as

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

where the underlined  $\underline{\mathbf{h}}^i \in \mathbb{R}^{|\mathcal{C}|}$  denote constant values and  $k \in \{1, \dots, d\}$  is a hyper-parameter usually set to  $\lfloor \frac{d}{2} \rfloor$ . As for autoregressive conditioners, the Jacobian of  $\mathbf{g}$  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  $\mathcal{P}_i = \{i\}$  for  $i \leq k$  and  $\mathcal{P}_i = \{1, \dots, k\}$  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  $\mathbf{x}_{k+1:d}$  knowing  $\mathbf{x}_{1:k}$ . 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  $g^i$  [Wehenkel and Louppe, 2020]. In practice, these bipartite structural independencies can be relaxed by stacking multiple layers, and may even recover an autoregressive structure. They also lead to useful inductive bias, such as in the multi-scale architecture with checkerboard masking [Dinh et al., 2017, Kingma and Dhariwal, 2018].

---

#### Algorithm 1 Sampling

---

- 1:  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, I)$
  - 2:  $\mathbf{x} \leftarrow \mathbf{z}$
  - 3: **repeat**
  - 4:    $\mathbf{c}^i \leftarrow \mathbf{h}^i(\mathbf{x} \odot A_{i,:}) \quad \forall i \in \{1, \dots, d\}$
  - 5:    $x_i \leftarrow (g^i)^{-1}(z_i; \mathbf{c}^i, \boldsymbol{\theta}) \quad \forall i \in \{1, \dots, d\}$
  - 6: **until**  $\mathbf{x}$  converged
- 

## 4 Graphical normalizing flow

### 4.1 Graphical conditioners

Following up on the previous discussion, we introduce the graphical conditioner architecture. We motivate our approach by observing that the topological ordering (a.k.a. ancestral ordering) of any BN leads to a

lower triangular adjacency matrix whose determinant is equal to the product of its diagonal terms (proof in Appendix B). Therefore, conditioning factors  $\mathbf{c}^i(\mathbf{x})$  selected by following a BN adjacency matrix necessarily lead to a transformation  $\mathbf{g}$  whose Jacobian determinant remains efficient to compute.

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

$$\mathbf{c}^i(\mathbf{x}) = \mathbf{h}^i(\mathbf{x} \odot A_{i,:}), \quad (5)$$

where  $\mathbf{x} \odot A_{i,:}$  is the element-wise product between the vector  $\mathbf{x}$  and the  $i^{\text{th}}$  row of  $A$  – i.e., the binary vector  $A_{i,:}$  is used to mask on  $\mathbf{x}$ . NFs built with this new conditioner architecture can be inverted by sequentially inverting each component in the topological ordering. In our implementation the neural networks modeling the  $\mathbf{h}^i$  functions are shared to save memory and they take an additional input that one-hot encodes the value  $i$ . An alternative approach would be to use a masking scheme similar to what is done by Germain et al. [2015] in MADE as suggested by Lachapelle et al. [2019].

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. In addition, the inverse of NFs built with graphical conditioners is as simple as it is for autoregressive and coupling conditioners, Algorithm 1 describes an inversion procedure. We also now note how these two conditioners are just 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 is the density at each node 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 not necessary.

Building upon *Non-combinatorial Optimization via Trace Exponential and Augmented lagRangian for Structure learning* [NO TEARS, 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 binary values. That is,

$$\begin{aligned} \max_{A \in \mathbb{R}^{d \times d}} F(A) & \iff \max_{A \in \mathbb{R}^{d \times d}} F(A) \\ \text{s.t. } \mathcal{G}(A) \in \text{DAGs} & \quad \text{s.t. } w(A) = 0, \end{aligned} \quad (6)$$

where  $\mathcal{G}(A)$  is the graph induced by the weighted adjacency matrix  $A$  and  $F : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}$  is the log-likelihood of the graphical NF  $\mathbf{g}$  plus a regularization term, i.e.,

$$F(A) = \sum_{j=1}^N \log(p(\mathbf{x}^j; \boldsymbol{\theta})) + \lambda_{\ell_1} \|A\|_1, \quad (7)$$

where  $\lambda_{\ell_1}$  is an  $\ell_1$ -regularization coefficient and  $N$  is the number of training samples  $\mathbf{x}^i$ . The likelihood is computed as

$$p(\mathbf{x}; \boldsymbol{\theta}) = p_{\mathbf{z}}(\mathbf{g}(\mathbf{x}; \boldsymbol{\theta})) \prod_{i=1}^d \left| \frac{\partial g^i(x_i; \mathbf{h}^i(\mathbf{x} \odot A_{i,:}), \boldsymbol{\theta})}{\partial x_i} \right|.$$

The function  $w(A)$  that enforces the acyclicity is expressed as suggested by Yu et al. [2019] as

$$w(A) := \text{tr}((I + \alpha A)^d) - d \alpha \text{tr} \left( \sum_{k=1}^d \alpha^k A^k \right),$$

where  $\alpha \in \mathbb{R}_+$  is a hyper-parameter that avoids exploding values for  $w(A)$ . In the case of positively valued  $A$ , an element  $(i, j)$  of  $A^k = \underbrace{AA \dots A}_{k \text{ terms}}$  is non-null if and only if there exists a path going from node  $j$  to node  $i$  that is made of exactly  $k$  edges. Intuitively  $w(A)$  expresses to which extent the graph is cyclic. Indeed, the diagonal elements  $(i, i)$  of  $A^k$  will be as large as there are many paths made of edges that correspond to large values in  $A$  from a node to itself in  $k$  steps.

In comparison to our work, Zheng et al. [2018] use a 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 has a continuously adjustable level of complexity and does not make any strong assumptions on the form of the conditional factors.

## 4.3 Stochastic adjacency matrix

In order to learn the BN topology from the data, the adjacency matrix must be relaxed to contain reals instead of booleans. It also implies that the graph induced by  $A$  does not formally define a DAG during training. Work using NO TEARS to perform topology learning directly plug the real matrix  $A$  in (6) [Zheng et al., 2018, Yu et al., 2019, Lachapelle et al., 2019, Zheng et al., 2020] however this is inadequate because the quantity of information going from node  $j$  to node  $i$  does not continuously relate to the value of  $A_{i,j}$ . Either the information is null if  $A_{i,j} = 0$  or it passes completely if not. Instead, we propose to build the stochastic pseudo binary valued matrix  $A'$  from  $A$ ,

defined as

$$A'_{i,j} = \frac{e^{\frac{\log(\sigma(A_{i,j}^2)) + \gamma_1}{T}}}{e^{\frac{\log(\sigma(A_{i,j}^2)) + \gamma_1}{T}} + e^{\frac{\log(1 - \sigma(A_{i,j}^2)) + \gamma_2}{T}}},$$

where  $\gamma_1, \gamma_2 \sim \text{Gumbel}(0, 1)$  and  $\sigma(a) = 2(\text{sigmoid}(2a^2) - \frac{1}{2})$  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 hyperparameter  $T$  controls the sampling temperature and is fixed to 0.5 in all our experiments. In contrast to directly using the matrix  $A$ , this stochastic transformation referred to as the Gumbel-Softmax trick in the literature [Maddison et al., 2016, Jang et al., 2016] allows to create a direct and continuously differentiable relationship between the weights of the edges and the quantity of information that can transit between two nodes. Indeed, the probability mass of the random variables  $A'_{i,j}$  is mainly located around 0 and 1, and its expected value converges to 1 when  $A_{i,j}$  increases.

#### 4.4 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:

$$\max_A \mathbb{E}_{\gamma_1, \gamma_2} [F(A)] - \lambda_t w(A) - \frac{\mu_t}{2} w(A)^2, \quad (8)$$

where  $\lambda_t$  and  $\mu_t$  respectively denote the Lagrangian multiplier and penalty coefficients of the sub-problem  $t$ .

We solve these optimization problems with mini-batch stochastic gradient ascent. We update the values of  $\gamma_t$  and  $\mu_t$  as suggested by Yu et al. [2019] when the validation loss does not improve for 10 consecutive epochs. Once  $w(A)$  equals 0, the adjacency matrix is acyclic up to numerical errors. We recover an exact DAG by thresholding the elements of  $A$  while checking for acyclicity with a path finding algorithm. We provide additional details about the optimization procedure used in our experiments in Appendix A.

## 5 Experiments

In this section, we demonstrate some applications of graphical NFs in addition to unifying NFs and BN under a common framework. We first demonstrate how pre-loading a known or hypothesized DAG structure can help finding an accurate distribution of the data. Then, we show that learning the graph topology leads to relevant BNs that support generalization

**Table 1:** Datasets description.  $d$ =Dimension of the data.  $V$ =Number of edges in the ground truth Bayesian Network.

Dataset	$d$	$V$	Train	Test
<b>Arithmetic Circuit</b>	8	8	10,000	5,000
<b>8 Pairs</b>	16	8	10,000	5,000
<b>Tree</b>	7	8	10,000	5,000
<b>Protein</b>	11	20	6,000	1,466
<b>POWER</b>	6	$\leq 15$	1,659,917	204,928
<b>GAS</b>	8	$\leq 28$	852,174	105,206
<b>HEPMASS</b>	21	$\leq 210$	315,123	174,987
<b>MINIBOONE</b>	43	$\leq 903$	29,556	3,648
<b>BSDS300</b>	63	$\leq 1,953$	1,000,000	250,000

**Table 2:** Graphical vs autoregressive conditioners combined with monotonic normalizers. Average log-likelihood on test data over 5 runs, under-scripted error bars are equal to the standard deviation. Results are reported in nats; higher is better. The best performing architecture for each dataset is written in bold. *Graphical conditioners clearly lead to improved density estimation when given a relevant prescribed topology in 3 out of the 4 datasets.*

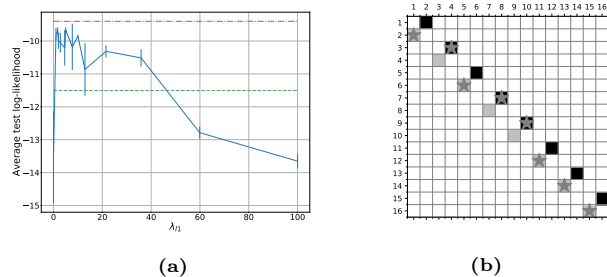
Conditioner	Graphical	Autoreg.
<b>Arithmetic Circuit</b>	<b>3.99<sub>±.16</sub></b>	3.06 <sub>±.38</sub>
<b>8 Pairs</b>	<b>-9.40<sub>±.06</sub></b>	-11.50 <sub>±.27</sub>
<b>Tree</b>	<b>-6.85<sub>±.02</sub></b>	-6.96 <sub>±.05</sub>
<b>Protein</b>	6.46 <sub>±.08</sub>	<b>7.52<sub>±.10</sub></b>

well when combined with  $\ell_1$ -penalization. Finally, we demonstrate that mono-step normalizing flows made of graphical conditioners are competitive density estimators.

### 5.1 On the importance of graph topology

The following experiments are performed on four distinct datasets, three of which are synthetic, such that we can define a ground-truth minimal Bayesian network, and the fourth is a causal protein-signaling network derived from single-cell data [Sachs et al., 2005]. Additional information about the datasets are provided in Table 1 and in Appendix C. For each experimental run we first randomly permute the features before training the model in order to compare autoregressive and graphical conditioners fairly.

**Prescribed topology** Rarely do real data come with their associated Bayesian network however oftentimes experts want to hypothesize a network topology and to rely on it for the downstream tasks. Sometimes the topology is known a priori, as an example the sequence of instructions in stochastic simulators can usually be translated into a graph topology (e.g. in probabilistic programming [van de Meent et al., 2018, Weibach et al., 2020]). In both cases, graphical conditioners allow to explicitly take advantage of this to



**Figure 2:** (a): Test log-likelihood as a function of  $\ell_1$ -penalization on **8 pairs** dataset. The upper bound is the average result when given a prescribed topology, the lower bound is the result with an autoregressive conditioner. *Learning the right topology leads to better results than autoregressive conditioners.* (b): The blacked cells corresponds to one correct topology of the 8 pairs dataset and the grey cells to the transposed adjacency matrix. The stars denote the edges discovered by the graphical conditioner when trained with  $\lambda_{\ell_1} = 4$ . *The optimization discovers a relevant BN (equivalent to the ground truth).*

build density estimators while keeping the assumptions made about the network topology valid.

Table 2 presents the test likelihood of autoregressive and graphical normalizing flows on the four datasets. The flows are made of a single step and use monotonic normalizers. The neural network architectures are all identical. Further details on the experimental settings as well as additional results for affine normalizers are respectively provided in Appendix C.2 and C.3. *We observe how using correct BN structures lead to good test performance in Table 2.* Surprisingly, the performance on the protein dataset are not improved when using the ground truth graph. We observed during our experiments (see Appendix C.3) that learning the topology from this dataset sometimes led to improved density estimation performance with respect to the ground truth graph. The limited dataset size does not allow us to answer if this comes from the limited capacity of the flow and/or from the erroneous assumptions in the ground truth graph. However, we stress out that the graphical flow respects the assumed causal structure in opposition to the autoregressive flow.

**Learning the topology** The  $\lambda_{\ell_1}$  coefficient introduced in (7) controls the sparsity of the optimized BN, allowing to avoid some spurious connections. We now analyze the effect of sparsity on the density estimation performance. Fig. 2a shows the test log-likelihood as a function of the  $\ell_1$ -penalization on the 8 pairs dataset. We observe that the worst results are obtained when there is no penalization. Indeed, in this case the algorithm finds multiple spurious connections between independent vector’s components and then overfits on

**Figure 3:** The adjacency matrix of the protein interaction network. The blacked cells are the directed connections proposed by domain experts and the grey is the transposed adjacency matrix. The stars denote the edges discovered by the graphical conditioner when trained with  $\lambda_{\ell_1} = 12$ . *In a realistic setting, the optimization leads to a graph that shares a lot with the one designed by experts.*

these incorrect relationships. Another extreme case shows up when the effect of penalization is too strong, in this case, the learning procedure underfits the data because it ignores too many relevant connections. It can also be concluded from the plot that the optimal  $\ell_1$ -penalization performs on par with the ground truth topology, and certainly better than the autoregressive conditioner. Additional results on the other datasets provided in Appendix C lead to similar conclusions.

**Protein network** The adjacency matrix discovered by optimizing (8) (with  $\lambda_{\ell_1} = 12$ ) on the protein network dataset is shown in Fig. 3. All the connections discovered by the method correspond to ground truth connections. Unsurprisingly, their orientation do not always match with what is expected by the experts. Overall we can see that the optimization procedure is able to find relevant connections between variables and avoids spurious ones when the  $\ell_1$ -penalization is optimized. Previous work on topology learning such as Zheng et al. [2018], Yu et al. [2019], Lachapelle et al. [2019] compare their method to others by looking at the topology discovered on the same dataset. Here we do not claim that learning the topology with a graphical conditioner improves over previous methods. Indeed, we believe that the difference between the methods mainly relies on the hypothesis and/or inductive biased made on the conditional densities. Which method would perform better is dependent on the application. Moreover, the Structural Hamiltonian Distance (SHD) and the Structural Inference Distance (SID), often used to compare BNs, are not well motivated in general. On the one hand the SHD does not take into account the independence relationships modeled with Bayesian networks, as an example the SHD between the graph found on 8 pairs dataset of Fig. 2b and one possible ground truth is non zero whereas the two BNs are equivalent. On the other hand the SID aims to compare causal graphs whereas the methods

**Table 3:** Average log-likelihood on test data over 3 runs, under-scripted error bars are equal to the standard deviation. Results are reported in nats; higher is better. The best performing architecture per category for each dataset is written in bold. (a) 1-step affine normalizers (b) 1-step monotonic normalizers. *Graphical normalizing flows outperform coupling and autoregressive architectures on most of the datasets.*

Dataset	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Coup.	-5.60 $\pm$ .00	-3.05 $\pm$ .01	-25.74 $\pm$ .01	-38.34 $\pm$ .02	57.33 $\pm$ .00
(a)Auto.	-3.55 $\pm$ .00	-0.34 $\pm$ .01	-21.66 $\pm$ .01	<b>-16.70<math>\pm</math>.05</b>	<b>63.74<math>\pm</math>.00</b>
Graph.	<b>-2.80<math>\pm</math>.01</b>	<b>1.99<math>\pm</math>.02</b>	<b>-21.18<math>\pm</math>.07</b>	-19.67 $\pm$ .06	62.85 $\pm$ .07
Coup.	0.25 $\pm$ .00	5.12 $\pm$ .03	-20.55 $\pm$ .04	-32.04 $\pm$ .12	107.17 $\pm$ .46
(b)Auto.	0.58 $\pm$ .00	9.79 $\pm$ .04	-14.52 $\pm$ .16	<b>-11.66<math>\pm</math>.02</b>	151.29 $\pm$ .31
Graph.	<b>0.62<math>\pm</math>.04</b>	<b>10.15<math>\pm</math>.15</b>	<b>-14.17<math>\pm</math>.13</b>	-16.23 $\pm$ .52	<b>155.22<math>\pm</math>.11</b>

discussed here do not learn causal relationships. In general BN topology identification is an ill posed problem and thus we believe using these metrics to compare different methods without additional downstream context is irrelevant. However, we conclude from Fig. 2b and Fig. 3 that the scores computed with graphical NFs can be used to learn relevant BN structures.

## 5.2 Density estimation benchmark

In these experiments, we compare autoregressive, coupling and graphical conditioners with no  $\ell_1$ -penalization on benchmark tabular datasets as introduced by Papamakarios et al. [2017] for density estimation. See Table 1 for a description. 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. We observed during preliminary experiments that stacking multiple conditioners improves the performance slightly, however the gain is marginal compared to the loss of interpretability. 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 randomly permuted at each run. All hyper-parameters are provided in Appendix D and a public implementation will be released on Github.

First, Table 3 presents the test log-likelihood obtained by each architecture. These results indicate that graphical conditioners offer the best performance in general. Unsurprisingly, coupling layers show the worst performance, due to the arbitrarily assumed interdependencies. Autoregressive and graphical conditioners show very similar performance for monotonic nor-

**Table 4:** Average log-likelihood on test data over 3 runs, under-scripted error bars are equal to the standard deviation. Results are reported in nats, higher is better. The results followed by a star are copied from the literature and the number of steps in the flow is indicated in parenthesis for each architecture. *Graphical normalizing flows reach density estimation performance on par with the most popular flow architectures whereas it is only made of 1 transformation step.*

Dataset	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Graph-UMNN (1)	0.62 $\pm$ .04	10.15 $\pm$ .15	-14.17 $\pm$ .13	-16.23 $\pm$ .52	155.22 $\pm$ .11
MAF (5)	0.14 $\pm$ .01	9.07 $\pm$ .01	-17.70 $\pm$ .01	-11.75 $\pm$ .22	155.69 $\pm$ .14
Glow* (10)	0.42 $\pm$ .01	12.24 $\pm$ .03	-16.99 $\pm$ .02	-10.55 $\pm$ .45	156.95 $\pm$ .28
UMNN-MAF* (5)	0.63 $\pm$ .01	10.89 $\pm$ .70	-13.99 $\pm$ .21	-9.67 $\pm$ .13	157.98 $\pm$ .01
Q-NSF* (10)	0.66 $\pm$ .01	12.91 $\pm$ .01	-14.67 $\pm$ .02	-9.72 $\pm$ .24	157.42 $\pm$ .14
FFJORD* (5-5-10-1-2)	0.46 $\pm$ .01	8.59 $\pm$ .12	-14.92 $\pm$ .08	-10.43 $\pm$ .04	157.40 $\pm$ .19

malizers, the latter being slightly better on 4 out of the 5 datasets. Table 4 contextualizes the performance of graphical normalizing flows with respect to the most popular normalizing flow architectures. Comparing the results together, we see that while additional steps lead to noticeable improvements for affine normalizers (MAF), benefits are questionable for monotonic transformations. Overall, graphical normalizing flows made of a single transformation step are competitive with the best flow architectures with the added value that they can directly be translated into their equivalent BNs. *From these results we stress out that single step graphical NFs are able to model complex densities on par with SOTA while they offer new ways of introducing domain knowledge.*

Second, Table 5 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. The depth [Bezek, 2016] of the equivalent BN directly limits the number of steps required to inverse the flow. Thus sparser graphs that are inevitably shallower correspond to NFs for which sampling, i.e. computing their inverse, is faster.

## 6 Discussion

**Cost of learning the graph structure** The attentive reader will notice that learning the topology does not come for free. Indeed, the Lagrangian for-

**Table 5:** Rounded average number of edges (over 3 runs) in the equivalent Bayesian network. *The graphical conditioners lead to sparser BNs compared to autoregressive conditioners.*

Dataset	P	G	H	M	B
Graph.-Aff.	15	26	152	277	471
Graph.-Mon.	15	27	159	265	1594
Coupling	9	16	110	462	992
Autoreg.	15	28	210	903	1953

mulation requires solving a sequence of optimization problems which increases the number of epochs before convergence. In our experiments we observed different overheads depending on the problems, however in general the training time is at least doubled. This does not impede the practical interest of using graphical normalizing flows. The computation overhead is more striking for non-affine normalizers (e.g. UMNNs) that are computationally heavy. However, we observed that most of the time the topology recovered by affine graphical NFs is relevant. It can thus be used as a prescribed topology for normalizers that are heavier to run, hence alleviating the computation overhead. Moreover, one can always hypothesize on the graph topology but more importantly the graph learned is usually sparser than an autoregressive one while achieving similar if not better results. The sparsity is interesting for two reasons: it can be exploited for speeding up the forward density evaluation; but more importantly it usually corresponds to shallower BNs that can be inverted faster than autoregressive structures.

**Bayesian network topology learning** Formal BN topology learning has extensively been studied for more than 30 years now 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 are usually separated between the constrained-based family such as the *PC algorithm* [Spirites 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 with NO TEARS, as a continuous function of the adjacency matrix, allowing the 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 carried out 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 distributions. 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. This neural masking scheme can also be implemented for NF 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] and 1x1 invertible convolutions [Kingma and Dhariwal, 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 eventually lead to binary masks that model the conditioning components of a factored joint distribution. In this way, the conditioners process their input as they would process the full vector. We show experimentally in Appendix E that this effectively leads to good inductive bias for processing images with NFs. In addition, we have shown that normalizing flows built from graphical conditioners combined with monotonic transformations are expressive density estimators. In effect, this means that enforcing some a priori known independencies can be performed thanks to graphical normal-



izing flows without hurting their modeling capacity. We believe such models could be of high practical interest because they cope well with large datasets and complex distributions while preserving some readability through their equivalent BN.

Close to our work, [Weilbach et al. \[2020\]](#) improve amortized inference by prescribing a BN structure between the latent and observed variables into a FFJORD NF, once again showing the interest of using the potential BN knowledge. Similar to our work, [Khemakhem et al. \[2020\]](#) see causal autoregressive flows as structural equation modelling. They show bivariate autoregressive affine flows can be used to identify the causal direction under mild conditions. Under similar mild conditions, discovering causal relationships with graphical normalizing flows could well be an exciting research direction.

**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 illustrated the importance of assuming or learning a relevant Bayesian network topology for density estimation. In addition, we have shown that this new architecture compares favorably with autoregressive and coupling conditioners and on par to the most common flow architectures on standard density estimation tasks even without any hypothesized topology. One interesting and straightforward extension of our work would be to combine it with normalizing flows for discrete variables. We also believe that graphical conditioners could be used when the equivalent Bayesian network is required for downstream tasks such as in causal reasoning.

## Acknowledgments

We thank Vân Anh Huynh-Thu, Johann Brehmer, and Louis Wehenkel for proofreading this manuscript or an earlier version of it. We are also thankful to the reviewers for helpful comments. 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

- Matúš Bezek. Characterizing dag-depth of directed graphs. *arXiv preprint arXiv:1612.04980*, 2016.
- David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5(Oct):1287–1330, 2004.
- Nicola De Cao, Ivan Titov, and Wilker Aziz. Block neural autoregressive flow. *arXiv preprint arXiv:1904.04676*, 2019.
- Nicola De Cao, Wilker Aziz, and Ivan Titov. Block neural autoregressive flow. In *Uncertainty in Artificial Intelligence*, pages 1263–1273. PMLR, 2020.
- Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. In *International Conference in Learning Representations workshop track*, 2015.
- Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference in Learning Representations*, 2017.
- Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, pages 7509–7520, 2019.
- F.J. Díez, J. Mira, E. Iturralde, and S. Zubillaga. DI-AVAL, 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](https://doi.org/10.1016/s0933-3657(97)00384-9).
- Dan Geiger, Thomas Verma, and Judea 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>.
- Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.
- Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Machine Learning*, 2018.
- David S Greenberg, Marcel Nonnenmacher, and Jakob H Macke. Automatic posterior transformation for likelihood-free inference. *arXiv preprint arXiv:1905.07488*, 2019.
- Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779*, 2018.
- Priyank Jaini, Kira A Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. *arXiv preprint arXiv:1905.02325*, 2019.
- Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- Matthew J Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep 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.
- Charles E. Kahn, Linda M. Roberts, Katherine A. Shaffer, and Peter Haddawy. Construction of a Bayesian network for mammographic diagnosis of breast cancer. *Comput. Biol. Med.*, 27(1):19–29, January 1997. URL [https://doi.org/10.1016/S0010-4825\(96\)00039-X](https://doi.org/10.1016/S0010-4825(96)00039-X).
- Ilyes Khemakhem, Ricardo Pio Monti, Robert Leech, and Aapo Hyvärinen. Causal autoregressive flows. *arXiv preprint arXiv:2011.02268*, 2020.
- Sungwon Kim, Sang-gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. Flowavenet: A generative flow for raw audio. *arXiv preprint arXiv:1811.02155*, 2018.
- Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018.
- Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016.
- Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.
- Daphne Koller and Nir Friedman. *Probabilistic graphical models: Principles and techniques*. MIT press, 2009.
- Sébastien Lachapelle, Philippe Brouillard, Tristan Deleu, and Simon Lacoste-Julien. Gradient-based neural dag learning. *arXiv preprint arXiv:1906.02226*, 2019.
- Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.
- Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3895–3904, 2018.
- George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.
- George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019a.
- George Papamakarios, David C Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019b.
- Judea Pearl and Stuart Russell. *Bayesian networks*. California Digital Library, 2011.
- Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019.
- Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015.
- Oren Rippel and Ryan Prescott Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.
- K. Sachs, O. Perez, D. Pe’er, D. A. Lauffenburger, and G. P. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308:523–529, 2005.
- Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-wesley professional, 2011.
- Flávio Luiz Seixas, Bianca Zadrozny, Jerson Laks, Aura Conci, and Débora Christina 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, August 2014. URL <https://doi.org/10.1016/j.compbiomed.2014.04.010>.
- Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. The MIT Press, 2001. URL <https://doi.org/10.7551/mitpress/1754.001.0001>.
- Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.
- Esteban G Tabak, Eric Vanden-Eijnden, et al. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.
- Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction

to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

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

Aäron Van Den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. Parallel WaveNet: Fast high-fidelity speech synthesis. In *International Conference on Machine Learning*, pages 3915–3923, 2018.

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

Antoine Wehenkel and Gilles Louppe. You say normalizing flows i see bayesian networks. *arXiv preprint arXiv:2006.00866*, 2020.

Christian Weillbach, Boyan Beronov, Frank Wood, and William Harvey. Structured conditional continuous normalizing flows for efficient amortized inference in graphical models. In *International Conference on Artificial Intelligence and Statistics*, pages 4441–4451. PMLR, 2020.

Yue Yu, Jie Chen, Tian Gao, and Mo Yu. DAG-GNN: Dag structure learning with graph neural networks. In *International Conference on Machine Learning*, pages 7154–7163, 2019.

Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. DAGs with NO TEARS: Continuous optimization for structure learning. In *Advances in Neural Information Processing Systems*, pages 9472–9483, 2018.

Xun Zheng, Chen Dan, Bryon Aragam, Pradeep Ravikumar, and Eric Xing. Learning sparse non-parametric dags. In *International Conference on Artificial Intelligence and Statistics*, pages 3414–3425. PMLR, 2020.

## A Optimization procedure

---

### Algorithm 2 Main Loop

---

```

epoch ← 0
while !Stopping criterion do
  foreach batch  $\mathbf{X} \in \mathbf{X}_{train}$  do
    loss ← COMPUTELOSS(flow,  $\mathbf{X}$ )
    OPTIMIZE(flow, loss)

  lossvalid ← COMPUTELOSS(flow,  $\mathbf{X}_{test}$ )
  epoch ← epoch + 1
  UPDATECOEFFICIENTS(flow, epoch, lossvalid)
  if ISDAGCONSTRAINTNULL(flow) then
    POSTPROCESS(flow)

```

---

The method COMPUTELOSS(flow,  $\mathbf{X}$ ) is computed as described by equation (8). The OPTIMIZE(flow, 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(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. The method UPDATECOEFFICIENTS() updates the Lagrangian coefficients as described in section 4.4.

## 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  $\mathbf{g}$  in the topological defined by  $A$ . Let us introduce the following new transformation  $\mathbf{g}_P(\mathbf{x}_P) = P\mathbf{g}(P^{-1}(P\mathbf{x}))$  on the permuted vector  $\mathbf{x}_P = P\mathbf{x}$ . The Jacobian of the transformation  $\mathbf{g}_P$  (with respect to  $\mathbf{x}_P$ ) 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

$$\begin{aligned}
 |\det(J_{\mathbf{g}_P(\mathbf{x}_P)})| &= |\det(P)| |\det(J_{\mathbf{g}(\mathbf{x})})| \frac{|\det(P)|}{|\det(P)|} \\
 &= |\det(J_{\mathbf{g}(\mathbf{x})})|,
 \end{aligned}$$

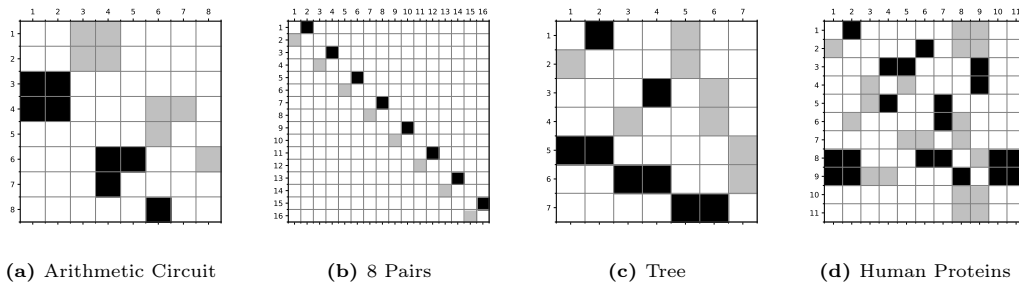
because of (1) the chain rule; (2) The determinant of the product is equal to the product of the determinants; (3) The determinant of a permutation matrix is equal to 1 or  $-1$ . The absolute value of the determinant of the Jacobian of  $\mathbf{g}$  is equal to the absolute value of the determinant of  $\mathbf{g}_P$ , the latter given by the product of its diagonal terms that are the same as the diagonal terms of  $\mathbf{g}$ . 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.  $\square$

## C Experiments on topology learning

### C.1 Neural networks architecture

We use the same neural network architectures for all the experiments on the topology. The conditioner functions  $h_i$  are modeled by shared neural networks made of 3 layers of 100 neurons. When using UMNNS for the normalizer we use an embedding size equal to 30 and a 3 layers of 50 neurons MLP for the integrand network.

**Figure 4:** Ground truth adjacency matrices. Black squares denote direct connections and in light grey is their transposed.



## C.2 Dataset description

**Arithmetic Circuit** The arithmetic circuit reproduced the generative model described by [Weilbach et al. \[2020\]](#). It is composed of heavy tailed and conditional normal distributions, the dependencies are non-linear. We found that some of the relationships are rarely found by during topology learning, we guess that this is due to the non-linearity of the relationships which can quickly saturates and thus almost appears as constant.

**8 pairs** This is an artificial dataset made by us which is a concatenation of 8 2D toy problems borrowed from [Grathwohl et al. \[2018\]](#) implementation. These 2D variables are multi-modal and/or discontinuous. We found that learning the independence between the pairs of variables is most of the time successful even when using affine normalizers.

**Tree** This problem is also made on top of 2D toy problems proposed by [Grathwohl et al. \[2018\]](#), in particular a sample  $X = [X_1, \dots, X_7]^T$  is generated as follows:

1. The pairs variables  $(X_1, X_2)$  and  $(X_3, X_4)$  are respectively drawn from *Circles* and *8-Gaussians*;
2.  $X_5 \sim \mathcal{N}(\max(X_1, X_2), 1)$ ;
3.  $X_6 \sim \mathcal{N}(\min(X_3, X_4), 1)$ ;
4.  $X_7 \sim 0.5\mathcal{N}(\sin(X_5 + X_6), 1) + 0.5\mathcal{N}(\cos(X_5 + X_6), 1)$ .

**Human Proteins** A causal protein-signaling networks derived from single-cell data. Experts have annotated 20 ground truth edges between the 11 nodes. The dataset is made of 7466 entries which we kept 5,000 for training and 1,466 for testing.

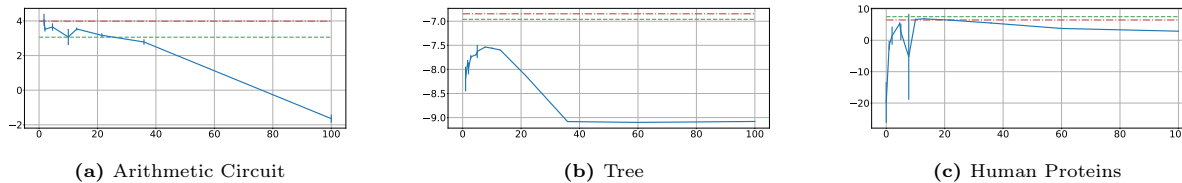
## C.3 Additional experiments

Fig. 5 and Fig. 6 present the test log likelihood as a function of the  $\ell_1$ -penalization on the four datasets for monotonic and affine normalizers respectively. It can be observed that graphical conditioners perform better than autoregressive ones for certain values of regularization and when given a prescribed topology in many cases. It is interesting to observe that autoregressive architectures perform better than a prescribed topology when an affine normalizer is used. We believe this is due to the non-universality of mono-step affine normalizers which leads to different modeling trade-offs. In opposition, learning the topology improves the results in comparison to autoregressive architectures.

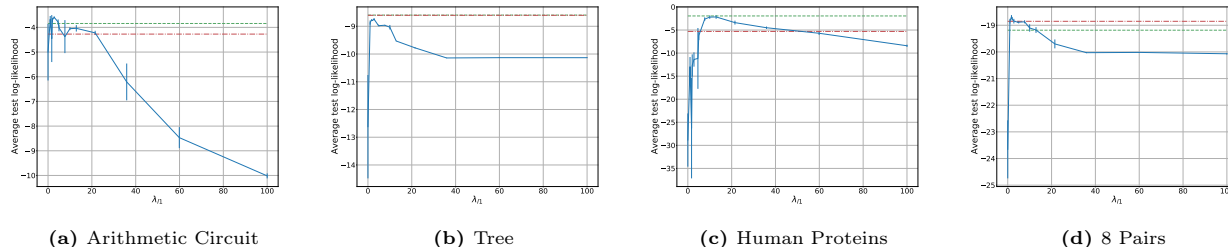
## D Tabular density estimation - Training parameters

Table 6 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  $x \odot A_{i,\cdot}$ . The embedding net architecture corresponds to the network that computes an embedding of the conditioning variables for the coupling and DAG conditioners,

**Figure 5:** Test log-likelihood as a function of  $\ell_1$ -penalization for monotonic normalizers. The red horizontal line is the average result when given a prescribed topology, the green horizontal line is the result with an autoregressive conditioner.



**Figure 6:** Test log-likelihood as a function of  $\ell_1$ -penalization for affine normalizers. The red horizontal line is the average result when given a prescribed topology, the green horizontal line is the result with an autoregressive conditioner.



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 an hyper-parameter named *embedding size* when combined with a UMNN. The number of dual steps corresponds to the number of epochs between two updates of the DAGness constraint (performed as in Yu et al. [2019]).

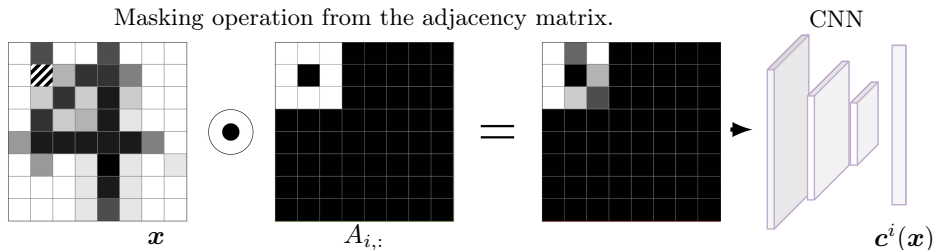
Dataset	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Batch size	2500	10000	100	100	100
Integ. Net	$3 \times 100$	$3 \times 200$	$3 \times 200$	$3 \times 40$	$3 \times 150$
Embedd. Net	$3 \times 60$	$3 \times 80$	$3 \times 210$	$3 \times 430$	$3 \times 630$
Embed. Size	30	30	30	30	30
Learning Rate	0.001	0.001	0.001	0.001	0.001
Weight Decay	$10^{-5}$	$10^{-3}$	$10^{-4}$	$10^{-2}$	$10^{-4}$
$\lambda_{\ell_1}$	0	0	0	0	0

**Table 6:** 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.

## E Density estimation of images

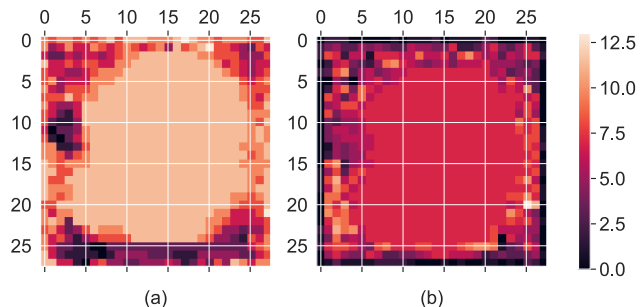
We now demonstrate how graphical conditioners can be used to fold in domain knowledge into NFs by performing density estimation on MNIST images. The design of the graphical conditioner is adapted to images by parameterizing the functions  $h^i$  with convolutional neural networks (CNNs) whose parameters are shared for all  $i \in \{1, \dots, d\}$  as illustrated in Fig. 7. Inputs to the network  $h^i$  are masked images specified by both the adjacency matrix  $A$  and the entire input image  $\mathbf{x}$ . 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



**Figure 7:** Illustration of how a graphical conditioner’s output  $c^i(\mathbf{x})$  is computed for images. The sample  $\mathbf{x}$ , 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  $A_{i,:}$ , the other pixels are in black. The element-wise product between the image  $\mathbf{x}$  and the mask  $A_{i,:}$  is processed by a convolutional neural network that produces the embedding vector  $c^i(\mathbf{x})$  conditioning the pixel  $x_i$ .

Model	Neg. LL.	Parameters	Edges	Depth
G-Affine (1)	$1.81 \pm .01$	$1 \times 10^6$	5016	103
(a) G-Monotonic (1)	$1.17 \pm .03$	$1 \times 10^6$	2928	125
A-Affine (1)	$2.12 \pm .02$	$3 \times 10^6$	306936	783
(b) A-Monotonic (1)	$1.37 \pm .04$	$3.1 \times 10^6$	306936	783
C-Affine (1)	$2.39 \pm .03$	$3 \times 10^6$	153664	1
C-Monotonic (1)	$1.67 \pm .08$	$3.1 \times 10^6$	153664	1
(c) A-Affine (5)	$1.89 \pm .01$	$6 \times 10^6$	$5 \times 306936$	$5 \times 783$
A-Monotonic (5)	$1.13 \pm .02$	$6.6 \times 10^6$	$5 \times 306936$	$5 \times 783$

**Table 7:** 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.



**Figure 8:** The in (a) and out (b) degrees of the nodes in the equivalent BN learned in the MNIST experiments.

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  $(i \pm l_1, j \pm l_2)$ ,  $l_1, l_2 \in \{1, \dots, L\}$  are allowed to be its parents. In early experiments we also tried not constraining the parents and observed slower but successful training leading to a relevant structure.

Results reported in Table 7 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  $\frac{5 \times 784}{100} \approx 40$  in comparison to the autoregressive flows made of 5 steps and comprising many more parameters.

These experiments show that, in addition to being 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.

## F 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.