A Optimization procedure

Algorithm 2 Main Loop

```
epoch ← 0
while !Stopping criterion do
    foreach batch X ∈ X_train do
        loss ← computeLoss(flow, X)
        optimize(flow, loss)
    loss_valid ← computeLoss(flow, X_test)
    epoch ← epoch + 1
    updateCoefficients(flow, epoch, loss_valid)
    if isDagConstraintNull(flow) then
        PostProcess(flow)
```

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

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

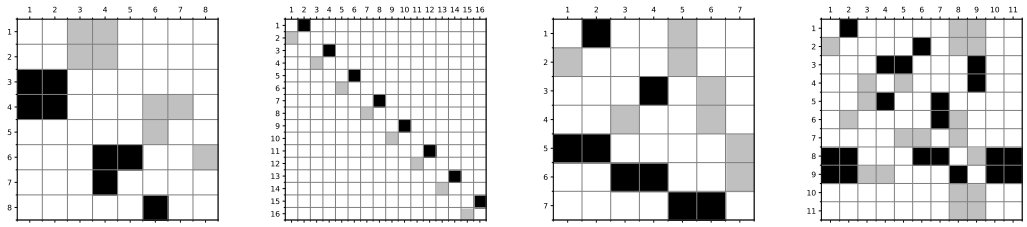
because of (1) the chain rule; (2) The determinant of the product is equal to the product of the determinants; (3) The determinant of a permutation matrix is equal to 1 or -1. The absolute value of the determinant of the Jacobian of $\boldsymbol{g}$ is equal to the absolute value of the determinant of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



(a) Arithmetic Circuit **(b)** 8 Pairs **(c)** Tree **(d)** Human Proteins

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 annoted 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.

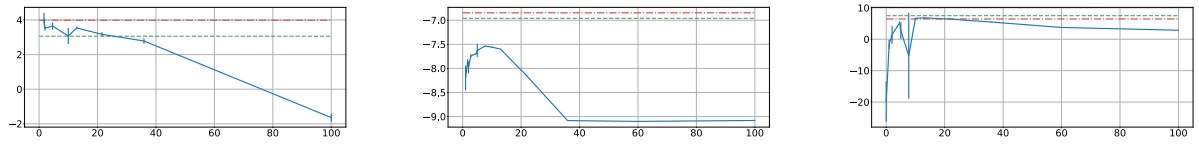
C.3 Additional experiments

Fig. 5 and Fig. 6 present the test log likelihood as a function of the ℓ_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

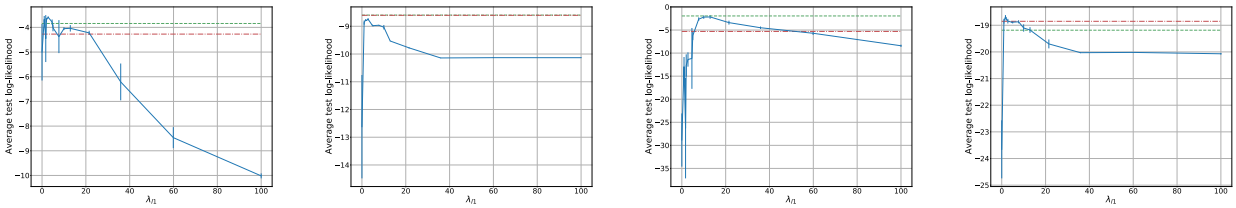
Table 6 provides the hyper-parameters used to train the normalizing flows for the tabular density estimation tasks. In our experiments we parameterize the functions $\boldsymbol{h}^i$ with a unique neural network that takes a one hot encoded version of i in addition to its expected input $\boldsymbol{x} \odot A_{i,:}$. The embedding net architecture corresponds to the network that computes an embedding of the conditioning variables for the coupling and DAG conditioners,

Figure 5: Test log-likelihood as a function of ℓ_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.



(a) Arithmetic Circuit **(b)** Tree **(c)** Human Proteins

Figure 6: Test log-likelihood as a function of ℓ_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.



(a) Arithmetic Circuit **(b)** Tree **(c)** Human Proteins **(d)** 8 Pairs

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	3×100	3×200	3×200	3×40	3×150
Embedd. Net	3×60	3×80	3×210	3×430	3×630
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}
λ_{ℓ_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 $\boldsymbol{h}^i$ with convolutional neural networks (CNNs) whose parameters are shared for all $i \in \{1, ..., d\}$ as illustrated in Fig. 7. Inputs to the network $\boldsymbol{h}^i$ are masked images specified by both the adjacency matrix A and the entire input image $\boldsymbol{x}$. 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

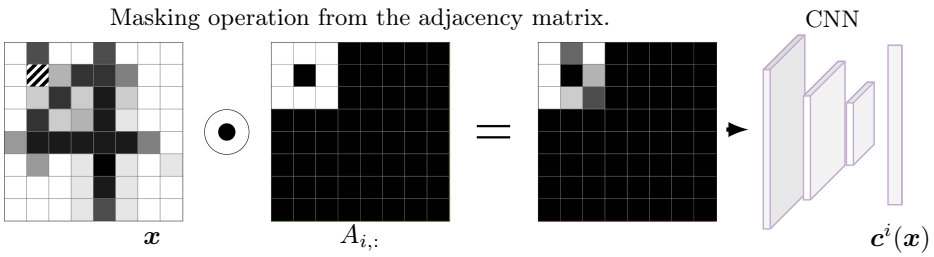


Figure 7: Illustration of how a graphical conditioner's output $\boldsymbol{c}^i(\boldsymbol{x})$ is computed for images. The sample $\boldsymbol{x}$, on the left, is an image of a 4. The stripes denote the pixel x_i. The parents of x_i in the learned DAG are shown as white pixels on the mask $A_{i,:}$, the other pixels are in black. The element-wise product between the image $\boldsymbol{x}$ and the mask $A_{i,:}$ is processed by a convolutional neural network that produces the embedding vector $\boldsymbol{c}^i(\boldsymbol{x})$ conditioning the pixel x_i.

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

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.

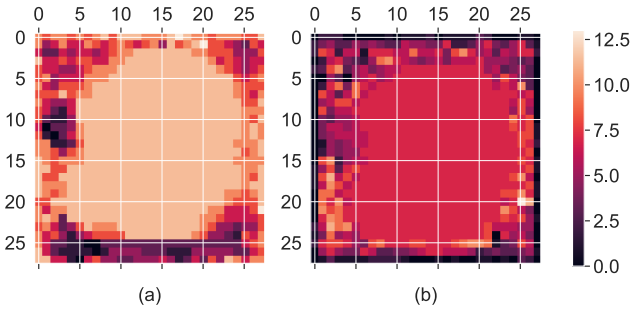


Figure 8: The in (a) and out (b) degrees of the nodes in the equivalent BN learned in the MNIST experiments.

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 $(i \pm l_1, j \pm l_2), l_1, l_2 \in \{1, ..., L\}$ 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 $\frac{5\times784}{100} \approx 40$ 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 3×3 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 3×1024 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.