Normalizing Flows and Bayesian Networks

HES Geneva, February 2022

Antoine Wehenkel

Our lab in Liège

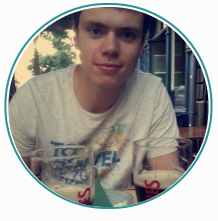
Research Topics:

- Methodological development in SBI;
- Application in particle physics, astrophysics, astronomy, robotics, ...;
- Algorithmic development in deep learning.

Gilles Louppe (Professor)

Joeri Hermans

Antoine Wehenkel



Norman Marlier

Maxime Quesnel

Malavika Vasist

Arnaud Delaunoy

François Rozet

Omer Rochman

Today's menu

Entree

Introduction to normalizing flows

Plat

Normalizing flows as Bayesian networks

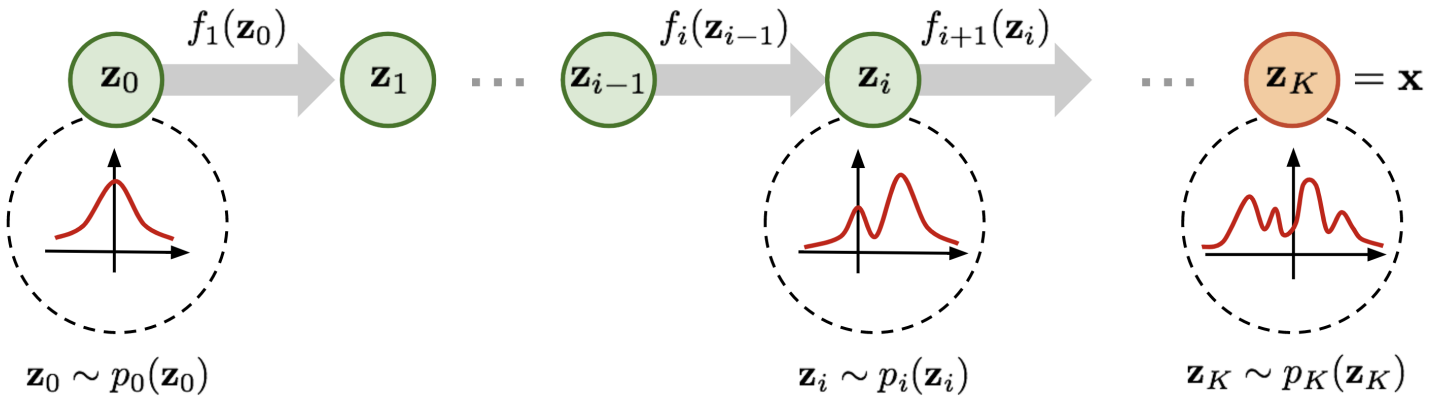
Dessert

Graphical normalizing flows

Introduction to normalizing flows

Normalizing flows

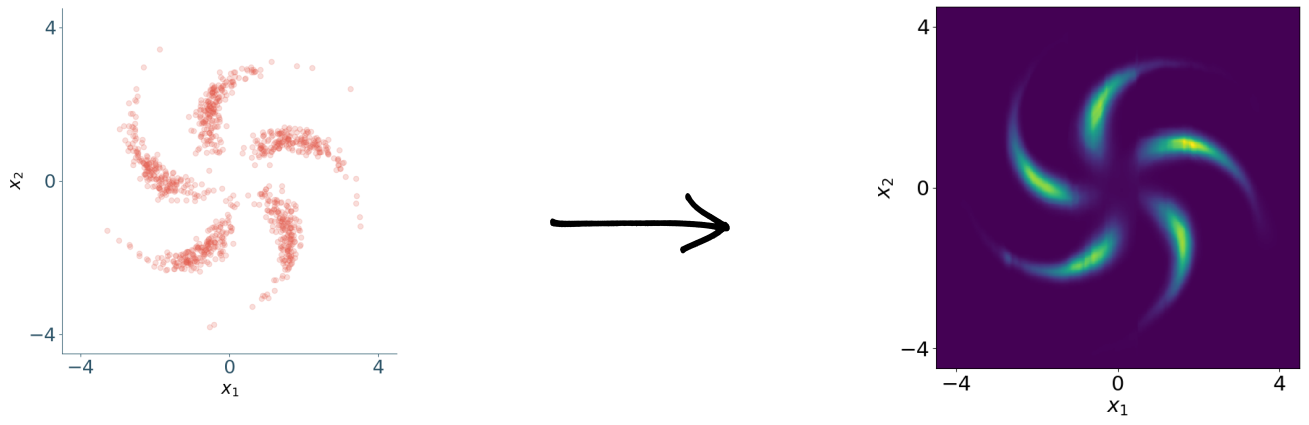
A normalizing flow is a bijective function used to model a probability distribution via a change of variables.



———

https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Density Estimation 👇



> Density estimation aims at estimating the pdf of underlying data from an iid dataset.

Applications:

- Model uncertainty: e.g., reinforcement learning, approximate inference in SBI, ...;
- Out Of Distribution detection;
- Sampling.

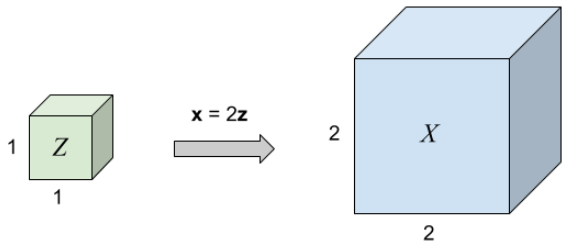
Change of Variables Theorem (1)

> Given a random variable $\mathcal{Z}$ and a differentiable bijective function f, what is the density of $\mathcal{X} = f(\mathcal{Z})$?

Change of Variables Theorem (1)

> Given a random variable $\mathcal{Z}$ and a differentiable bijective function f, what is the density of $\mathcal{X} = f(\mathcal{Z})$?

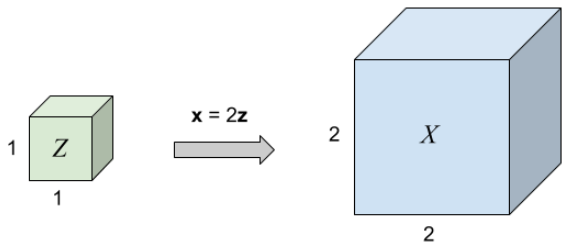
Assume $p(z)$ is a uniformely distributed unit cube in $\mathbb{R}^3$, and $\mathbf{x} = f(\mathbf{z}) = 2\mathbf{z}$.



Change of Variables Theorem (1)

> Given a random variable $\mathcal{Z}$ and a differentiable bijective function f, what is the density of $\mathcal{X} = f(\mathcal{Z})$?

Assume $p(z)$ is a uniformely distributed unit cube in $\mathbb{R}^3$, and $\mathbf{x} = f(\mathbf{z}) = 2\mathbf{z}$.



The total probability mass must be conserved, therefore

$p(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{z})\frac{V_{\mathbf{z}}}{V_{\mathbf{x}}} = p_{\mathbf{z}}(\mathbf{z})\frac{1}{8}$, where $\frac{1}{8} = \left|\det \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}\right|^{-1}$ is the determinant

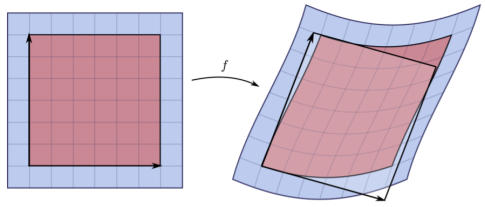
of the linear transformation f.

———

Change of Variables Theorem (2)

What if the transformation is non linear?

- The Jacobian $J_f(\mathbf{z})$ of $f(\mathbf{z}) = \mathbf{x}$ represents the infinitesimal linear transformation in the neighbourhood of $\mathbf{z}$.



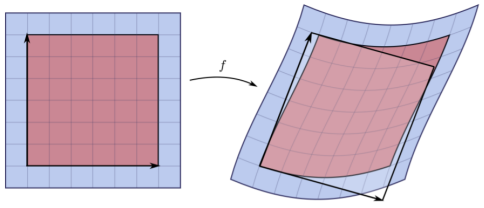
———

https://www.lesswrong.com/posts/KKwvgkcQz2gvqPLAD/a-primer-on-matrix-calculus-part-2-jacobians-and-other-fun

Change of Variables Theorem (2)

What if the transformation is non linear?

- The Jacobian $J_f(\mathbf{z})$ of $f(\mathbf{z}) = \mathbf{x}$ represents the infinitesimal linear transformation in the neighbourhood of $\mathbf{z}$.



- If the function is a differentiable (and so continuous) bijective map then the mass must be conserved locally.

Therefore, we can compute the local change of density as

$$p(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{z}) \left|\det J_f(\mathbf{z})\right|^{-1} .$$

———

https://www.lesswrong.com/posts/KKwvgkcQz2gvqPLAD/a-primer-on-matrix-calculus-part-2-jacobians-and-other-fun

Change of Variables Theorem (3)

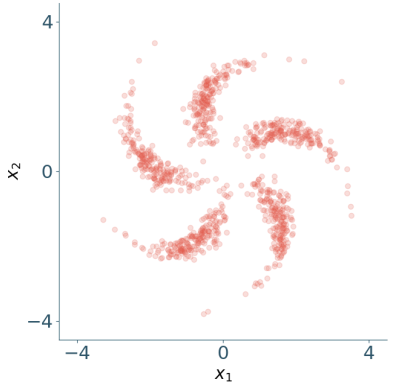
The right bijective map (with any base distribution) allows to represent any continuous random variable.

Change of Variables Theorem (3)

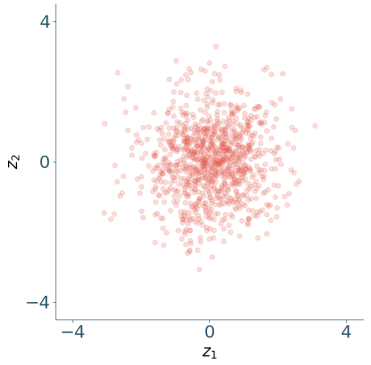
The right bijective map (with any base distribution) allows to represent any continuous random variable.

$$p(\mathbf{x};\theta) = p_{\mathbf{z}}(\mathbf{g}(\mathbf{x};\theta))\left|\det J_g(\mathbf{x};\theta)\right|, \quad \mathbf{g}(.;\theta) \text{ a neural network.}$$

- The bijective function takes in samples and maps them to noise.
- This process is refered as normalization if the noise distribution is normal.

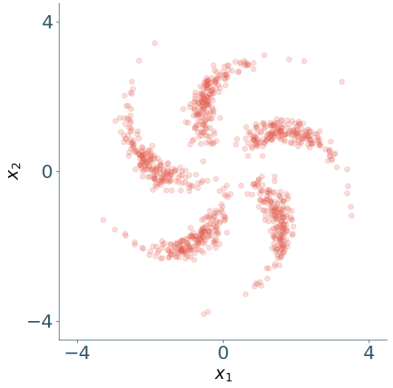


Density Estimation

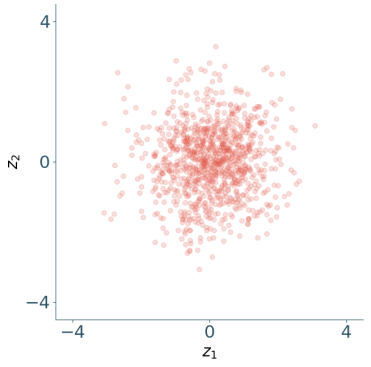


Change of Variables Theorem (3)

The right bijective map (with any base distribution) allows to represent any continuous random variable.



Sampling



Once learned, the function can be inverted in order to generate samples.

Bijectivity with Neural Nets? 🤔

- $\begin{bmatrix} z_1 & \dots & z_d \end{bmatrix} = \mathbf{g}(\begin{bmatrix} x_1 & \dots & x_d \end{bmatrix})$, $\mathbf{g}$ can be a NN.
- g is autoregressive if it can be decomposed as: $z_i = g_i(\begin{bmatrix} x_1 & \dots & x_i \end{bmatrix})$
- If the g_i are invertible with respect to $x_i \forall i$, $\mathbf{g}$ is bijective.

Bijectivity with Neural Nets? 🤔

- $\begin{bmatrix} z_1 & \dots & z_d \end{bmatrix} = \mathbf{g}(\begin{bmatrix} x_1 & \dots & x_d \end{bmatrix})$, $\mathbf{g}$ can be a NN.
- g is autoregressive if it can be decomposed as: $z_i = g_i(\begin{bmatrix} x_1 & \dots & x_i \end{bmatrix})$
- If the g_i are invertible with respect to $x_i \forall i$, $\mathbf{g}$ is bijective.

The determinant of the Jacobian can be efficiently computed.

The Jacobian of an autoregressive tranformation has the following form:

$$J_F(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \frac{\partial g_1}{\partial x_3} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \frac{\partial g_2}{\partial x_3} \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & 0 & 0 \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & 0 \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix}.$$

Bijectivity with Neural Nets? 🤔

- $\begin{bmatrix} z_1 & \dots & z_d \end{bmatrix} = \mathbf{g}(\begin{bmatrix} x_1 & \dots & x_d \end{bmatrix})$, $\mathbf{g}$ can be a NN.
- g is autoregressive if it can be decomposed as: $z_i = g_i(\begin{bmatrix} x_1 & \dots & x_i \end{bmatrix})$
- If the g_i are invertible with respect to $x_i \forall i$, $\mathbf{g}$ is bijective.

The determinant of the Jacobian can be efficiently computed.

The Jacobian of an autoregressive tranformation has the following form:

$$J_F(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \frac{\partial g_1}{\partial x_3} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \frac{\partial g_2}{\partial x_3} \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & 0 & 0 \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & 0 \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix}.$$

Chain Rule

An autoregressive density estimator learns the chain rule's factors:
$p(\mathbf{x}) = p(x_1)\Pi_{i=2}^{d} p(x_i|x_1, ..., x_{i-1}).$

Example: Masked Autoregressive Networks

> Idea: Autoregessive Networks combined with linear transformations.

- $z_1 = \sigma_1 \times x_1 + \mu_1$
- $z_i = \sigma_i(x_1, ..., x_{i-1}) \times x_i + \mu_i(x_1, ..., x_{i-1})$

Example: Masked Autoregressive Networks

> Idea: Autoregessive Networks combined with linear transformations.

- $z_1 = \sigma_1 \times x_1 + \mu_1$
- $z_i = \sigma_i(x_1, ..., x_{i-1}) \times x_i + \mu_i(x_1, ..., x_{i-1})$

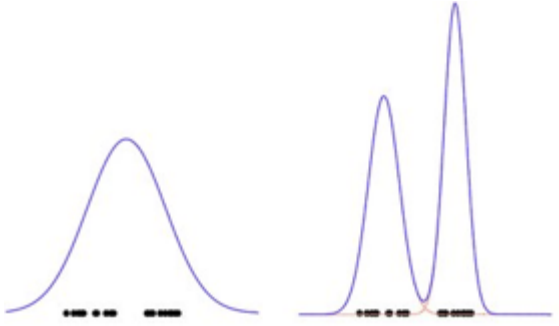
Invertible?

- $x_1 = g_1^{-1}(\begin{bmatrix} z_1 & ... & z_d \end{bmatrix}) = g_1^{-1}(z_1) = \frac{(z_1 - \mu_1)}{\sigma_1}$
- $x_i = \frac{z_i - \mu_i(\begin{bmatrix} x_1 & ... & x_{i-1} \end{bmatrix})}{\sigma_i(\begin{bmatrix} x_1 & ... & x_{i-1} \end{bmatrix})}$

Example: Masked Autoregressive Networks

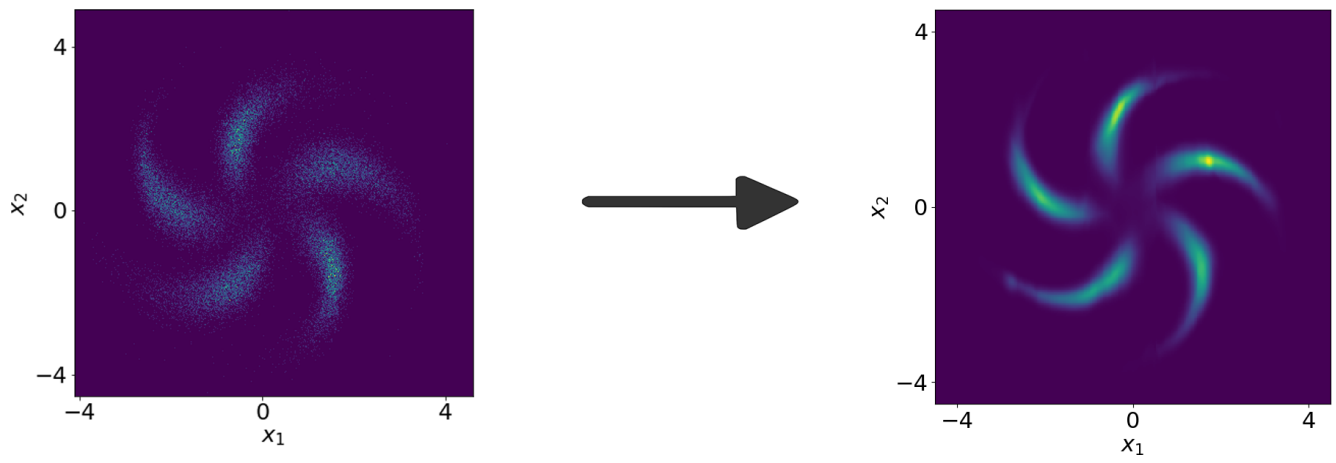
> Idea: Autoregessive Networks combined with linear transformations.

But linear transformations are not very expressive:



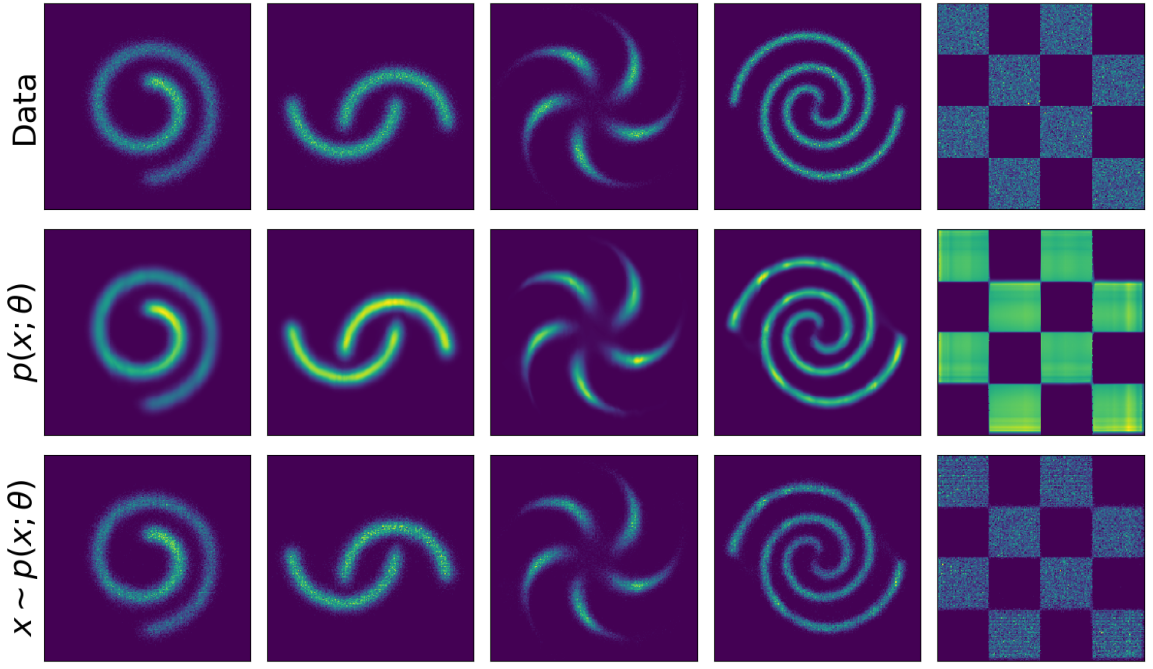
NFs pros 💪

- Access to the model's likelihood



NFs pros 💪

- Access to the model's likelihood
- Universal density estimators



NFs pros 💪

- Access to the model's likelihood
- Universal density estimators
- Good results for high dimensional data

———

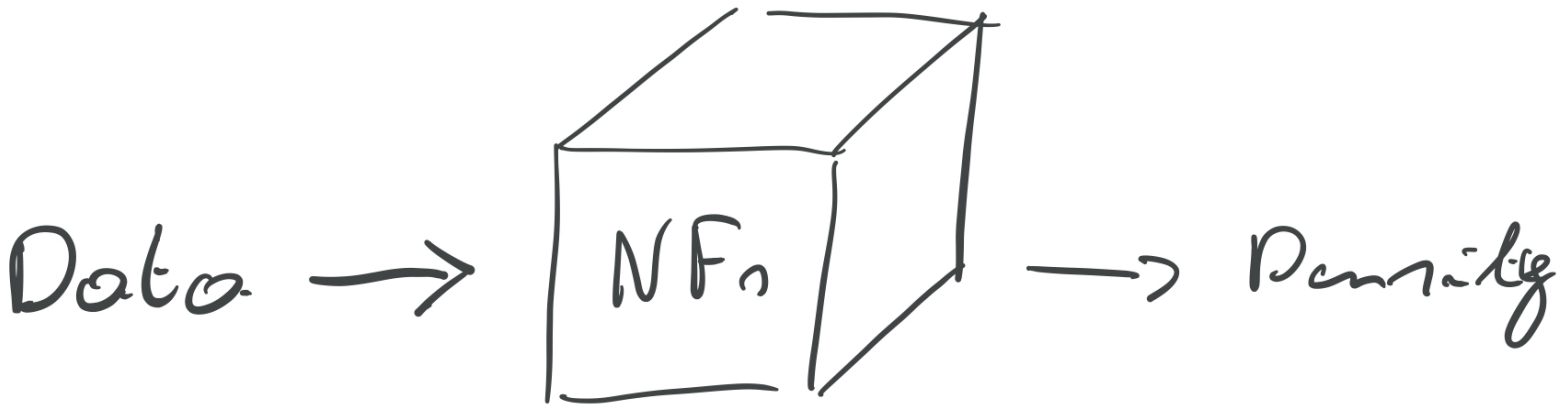
https://openai.com/blog/glow/

NFs cons 😈

- Arbitrary architectural choices

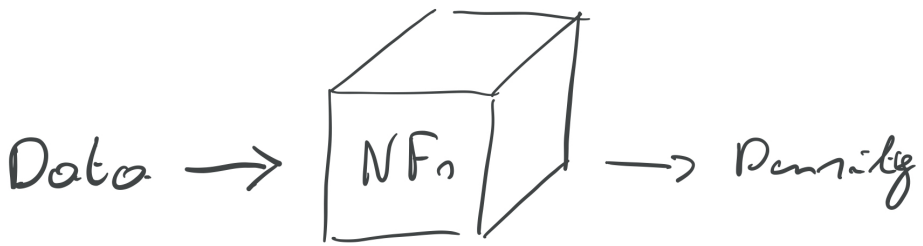
NFs cons 😈

- Arbitrary architectural choices
- Hard to interpret



NFs cons 😈

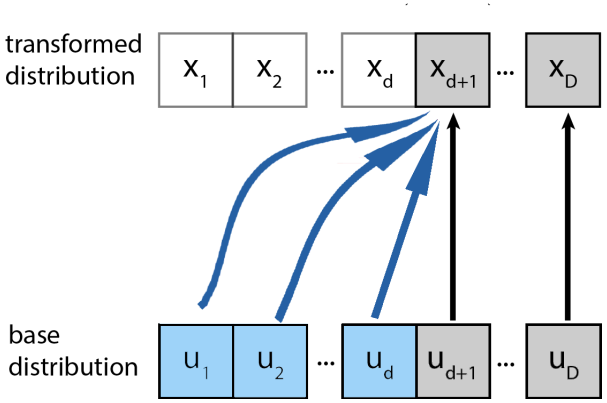
- Arbitrary architectural choices
- Hard to interpret
- Poor inductive bias



Inductive bias in NFs

How is it tackled now?

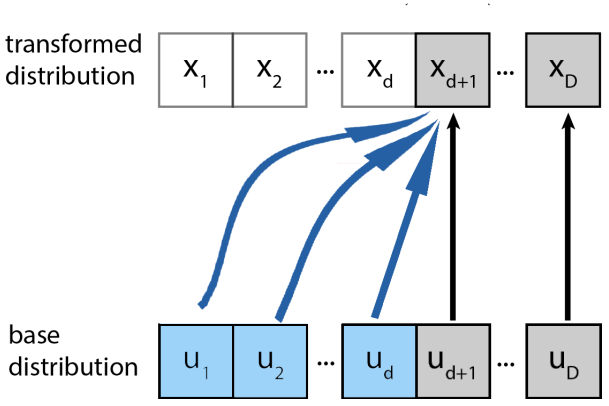
- For images:
  - Coupling layers

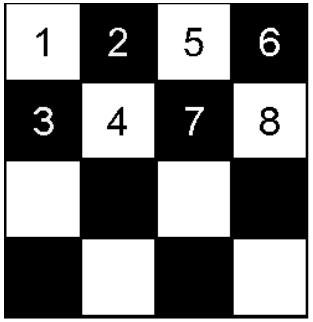


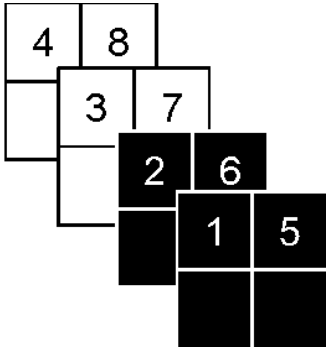
Inductive bias in NFs

How is it tackled now?

- For images:
  - Coupling layers
  - Multi-scale architectures





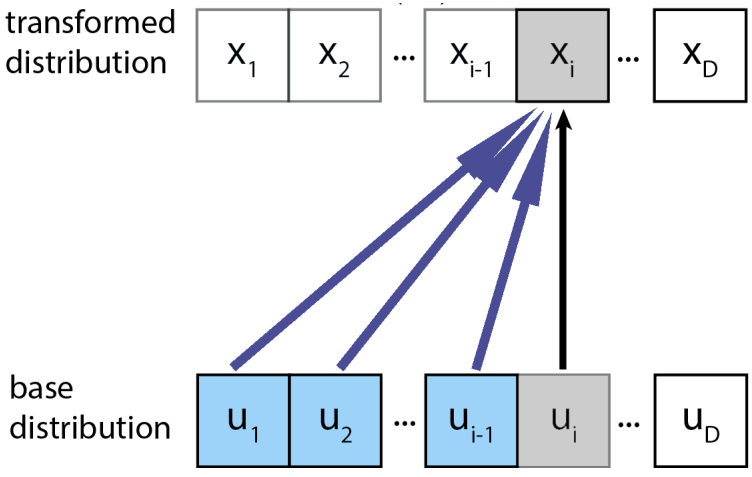


Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using real nvp.

Inductive bias in NFs

How is it tackled now?

- For images:
  - Coupling layers
  - Multi-scale architectures
- For time series:
  - Autoregressive architectures

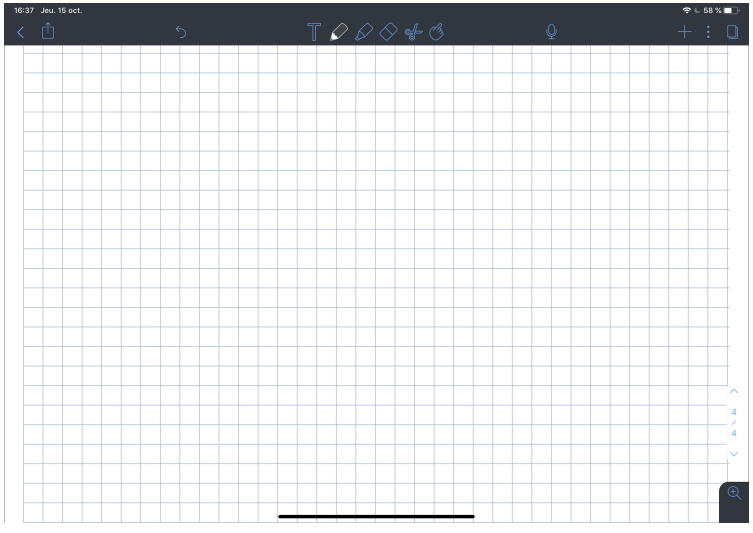


https://blog.evjang.com/2018/01/nf2.html

Inductive bias in NFs

How is it tackled now?

- For images:
  - Coupling layers
  - Multi-scale architectures
- For time series:
  - Autoregressive architectures
- What about tabular data or mixed data?



It is not easy to design the architecture and to understand the modeling assumptions!

Normalizing flows as Bayesian networks

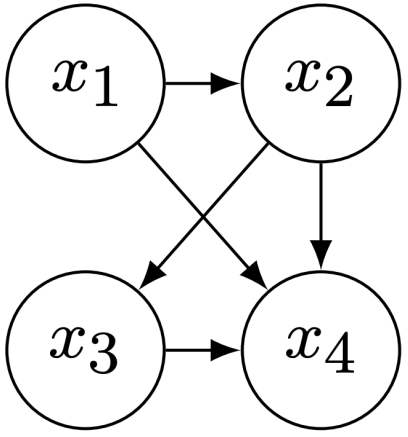
Bayesian Networks

- Probabilistic graphical models formally introduced by Judea Pearl in the 80's
- A Bayesian network is a directed acyclic graph that factorizes the model distribution as

$$p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i|\mathcal{P}_i).$$

- e.g when $d = 4$:

  $p(\mathbf{x}) = p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_1, x_2, x_3)$



BNs: pros 💪 and cons 😈

- Good for modeling independencies and check their global impact on the modeled density 💪

BNs: pros 💪 and cons 😈

- Good for modeling independencies and check their global impact on the modeled density 💪
- Applications across science and technology 💪

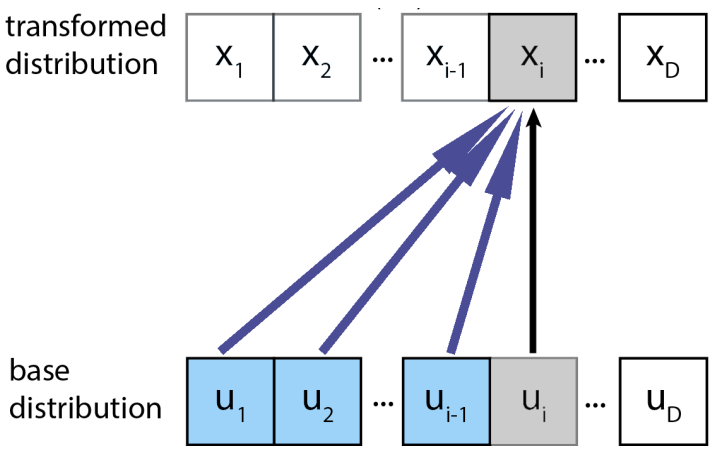
BNs: pros 💪 and cons 😈

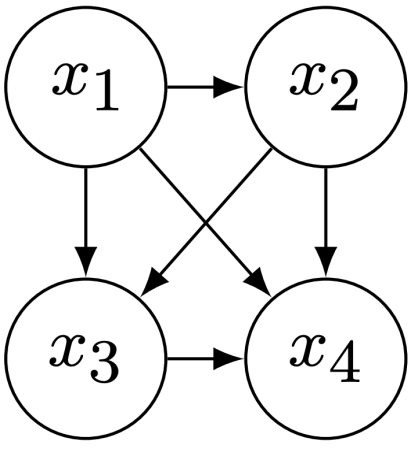
- Good for modeling independencies and check their global impact on the modeled density 💪
- Applications across science and technology 💪
- Often used with discrete or discretized data 😈

BNs: pros 💪 and cons 😈

- Good for modeling independencies and check their global impact on the modeled density 💪
- Applications across science and technology 💪
- Often used with discrete or discretized data 😈
- Do not handle large datasets naturally. 😈

Autoregressive NFs are BNs



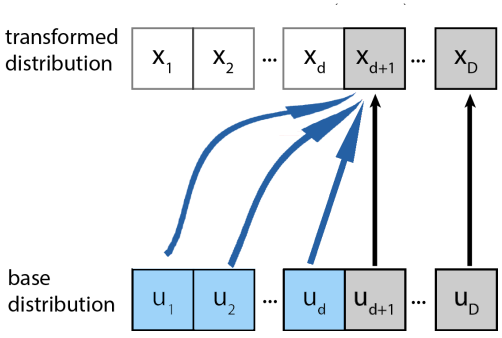


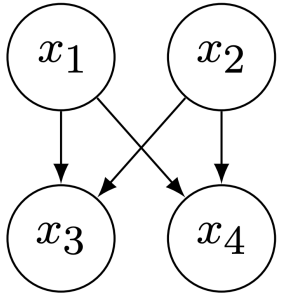
The flows can be expressed as: $x_i = f(u_i; \mathbf{c}^i(\mathbf{u}))$,

where the autoregressive conditioner is $\mathbf{c}^i(\mathbf{u}) = \mathbf{h}^i\left(\begin{bmatrix} u_1 & \dots & u_{i-1} \end{bmatrix}^T\right)$.

An autoregressive density estimator learns the chain rule's factors:
$p(\mathbf{x}) = p(x_1)\Pi_{i=2}^{D} p(x_i|x_1, ..., x_{i-1})$.

Coupling NFs are BNs





The coupling conditioner can be defined as $\mathbf{c}^i(\mathbf{u}) =$

- $\underline{\mathbf{h}}^i \quad \text{if} \quad i \leq d$ (a constant);
- $\mathbf{h}^i\left(\begin{bmatrix} u_1 & \dots & u_d \end{bmatrix}^T\right)$ if $i > d$.

Coupling learns the factors of the following factorization:
$p(\mathbf{x}) = \Pi_{i=1}^{d} p(x_i) \Pi_{j=k+1}^{D} p(x_j | x_1, ..., x_d).$

———
https://blog.evjang.com/2018/01/nf2.html

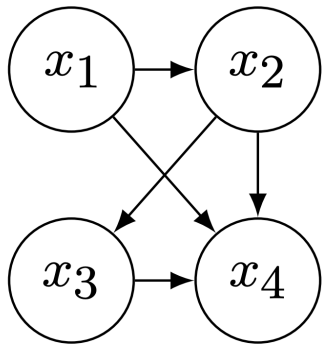
Graphical normalizing flows

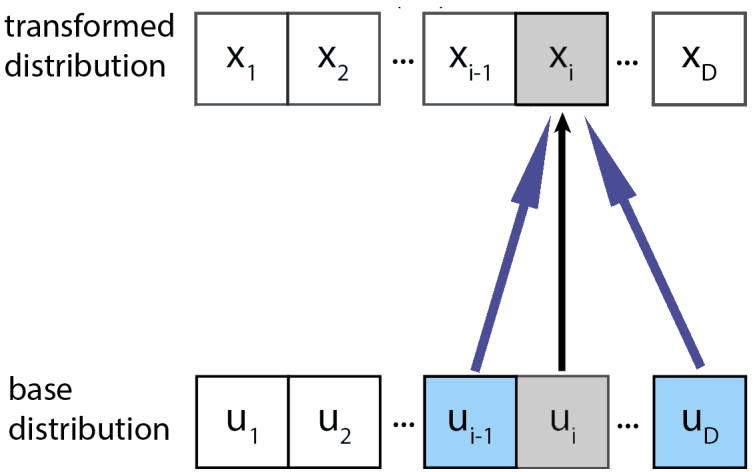
Is the opposite true? 💡

Is the opposite true? 💡

The graphical conditioner

Let $A \in \{0, 1\}^D$ be the adjacency matrix of a given Bayesian network for a random vector $\mathbf{x} \in \mathbb{R}^d$. We define the graphical conditioner as: $\mathbf{c}^i(\mathbf{u}) = \mathbf{h}^i(\mathbf{u} \odot A_{i,:})$.



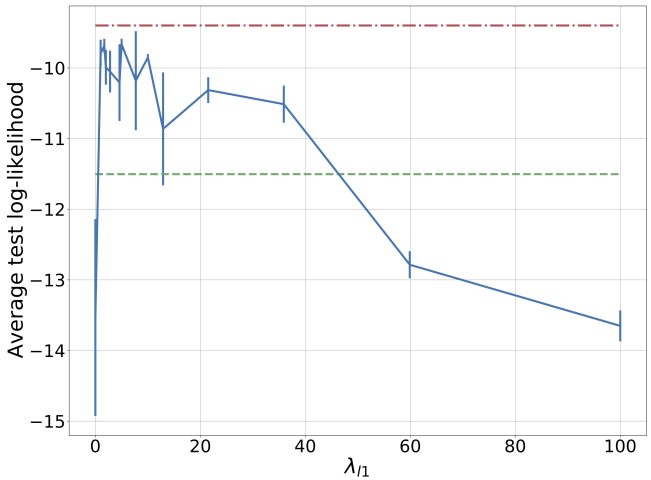


Useful in practice?

- It can be critical or convenient to ensure some independencies.
    - E.g. assuming independencies between gender and salary.

Useful in practice?

- It can be critical or convenient to ensure some independencies.
  - E.g. assuming independencies between gender and salary.
- Knowing the topology helps learning good densities.



Why not learning the topology?

- Any BN corresponds to a DAG, but any DAG can be seen as the topology of a BN as well.

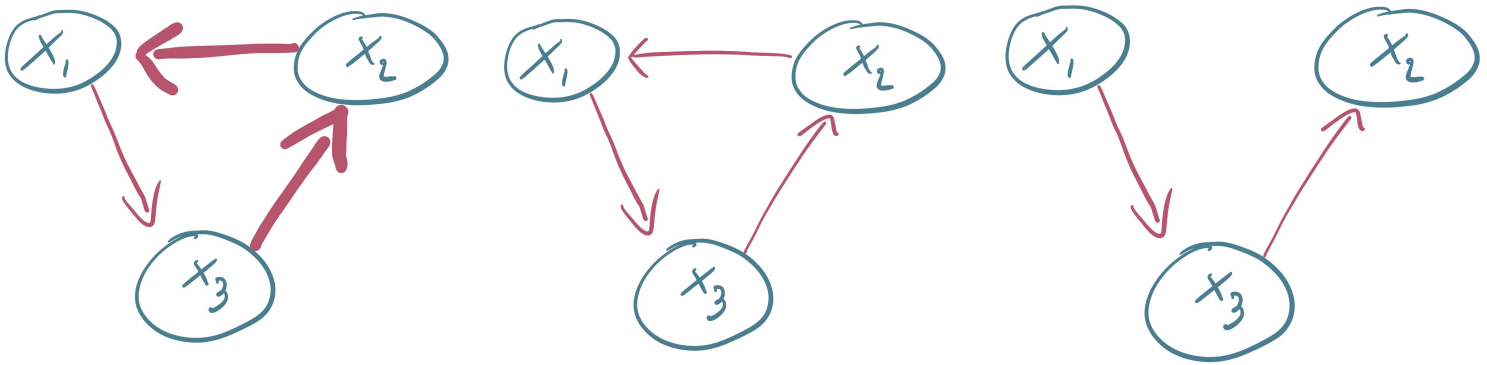
Why not learning the topology?

- Any BN corresponds to a DAG, but any DAG can be seen as the topology of a BN as well.
- We look for the DAG that maximizes the model's likelihood: $\max_{A \in \mathbb{R}^{d \times d}} F(A)$ s.t. $\mathcal{G}(A) \in$ **DAGs**.

Why not learning the topology?

- Any BN corresponds to a DAG, but any DAG can be seen as the topology of a BN as well.
- We look for the DAG that maximizes the model's likelihood: $\max_{A \in \mathbb{R}^{d \times d}} F(A)$ s.t. $\mathcal{G}(A) \in$ **DAGs**.
- We can formulate it as a continuous constraint:

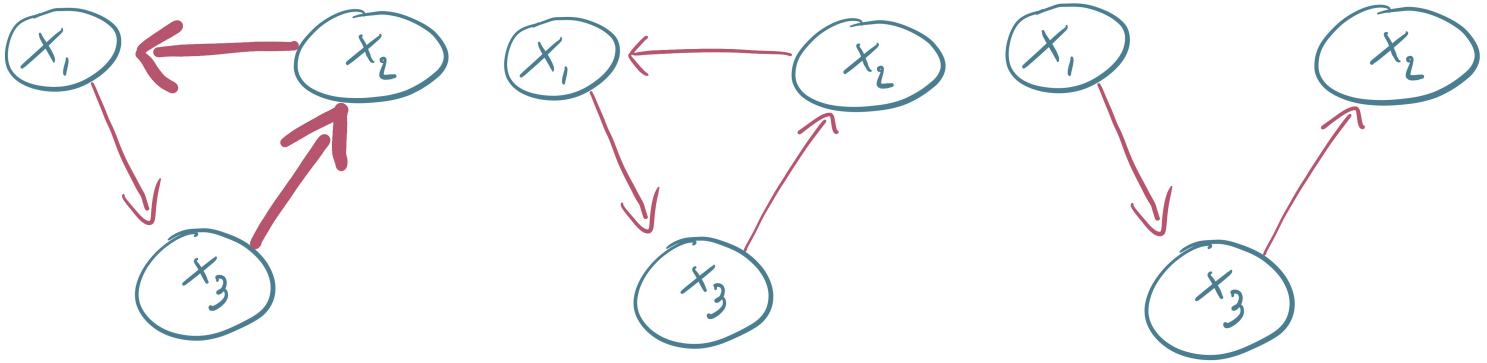
  $\max_{A \in \mathbb{R}^{d \times d}} F(A)$ s.t. $w(A) = 0$ where $w(A) := \mathrm{Trace}\left(\sum_{i=1}^{D} A^i\right)$.



Why not learning the topology?

- Any BN corresponds to a DAG, but any DAG can be seen as the topology of a BN as well.
- We look for the DAG that maximizes the model's likelihood: $\max_{A \in \mathbb{R}^{d \times d}} F(A)$ s.t. $\mathcal{G}(A) \in$ **DAGs**.
- We can formulate it as a continuous constraint:

  $\max_{A \in \mathbb{R}^{d \times d}} F(A)$ s.t. $w(A) = 0$ where $w(A) := \mathrm{Trace}\left(\sum_{i=1}^{D} A^i\right)$.



- We can solve the continuously constrained problem with a Lagrangian formulation!

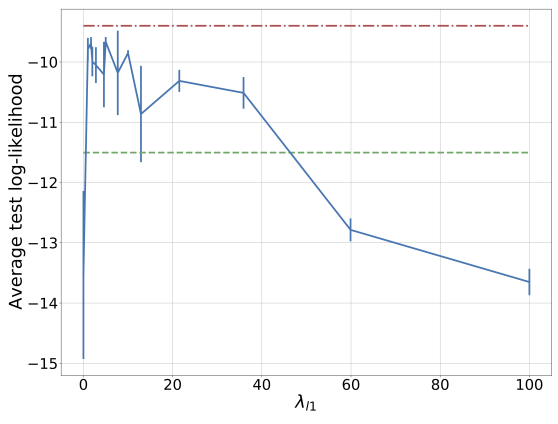
Computational cost

- Solving the sub-problems to optimality increases computational cost 😈
- As fast as autoregressive or coupling layers at inference time 💪
- The inversion of the flow will be often faster than autoregressive architectures 💪

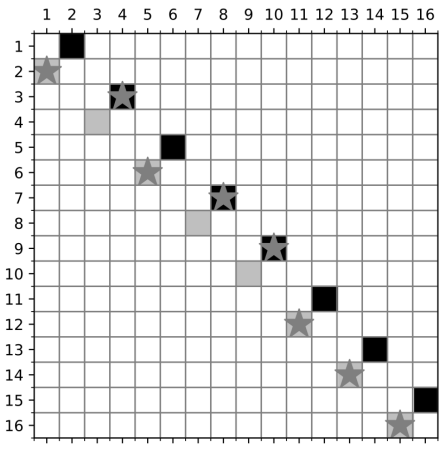
Results

Known vs Unknown Topology (Monotonic transformer)

Effect of sparsity



Topology recovered



Learning a good topology helps for density estimation.

Results

Density estimation benchmark

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)	$0.46_{\pm.01}$	$8.59_{\pm.12}$	$-14.92_{\pm.08}$	$-10.43_{\pm.04}$	$157.40_{\pm.19}$

We may obtain density estimation results on par with the best NF architectures.

Conclusion

- Standard Normalizing Flows are Bayesian Networks.
- Any Bayesian network can be easily turned into a normalizing flow.
- This allows introducing independency assumptions into the modeled density.
- This architecture can be used to do graphical model discovery.

References:

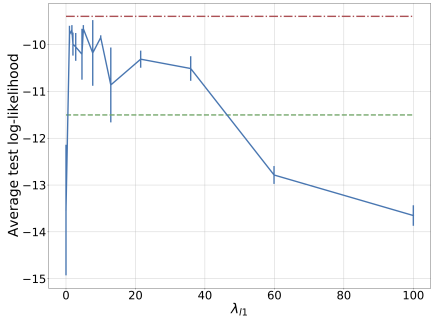
- Graphical Normalizing Flows, A. Wehenkel and G. Louppe, October 2020 - https://arxiv.org/abs/2006.02548
- You say Normalizing Flows I see Bayesian Networks, A. Wehenkel and G. Louppe, June 2020 - https://arxiv.org/abs/2006.00866

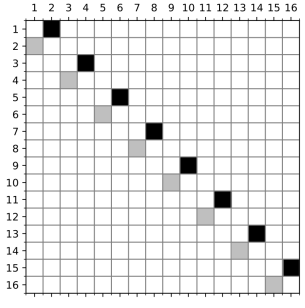
Thanks for listening

Results

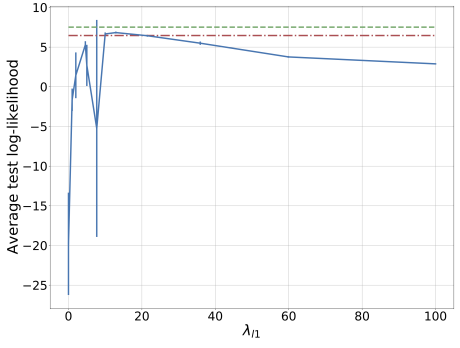
Known vs Unknown Topology (Monotonic transformer)

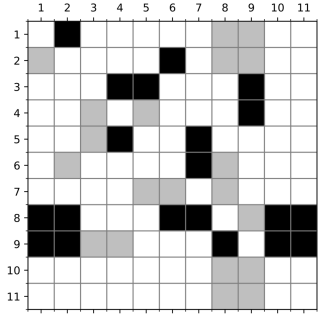
8 pairs of independent variables





Human protein dataset



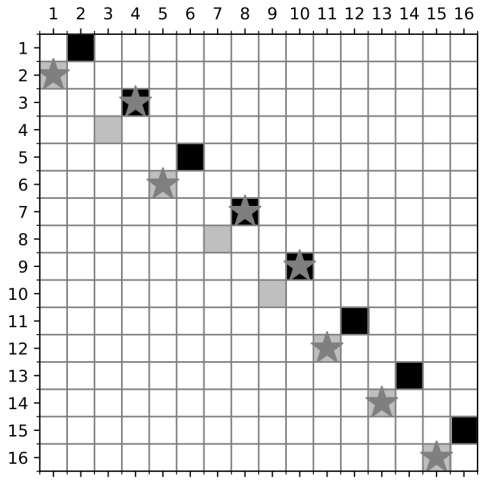


Learning a good topology helps for density estimation.

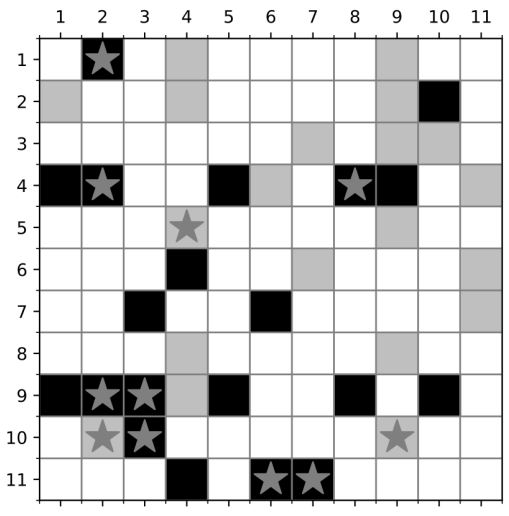
Results

Relevance of the discovered topology (Monotonic transformer)

8 pairs of independent variables



Human protein dataset



The optimization is able to remove spurious dependencies and keeps the correct ones.