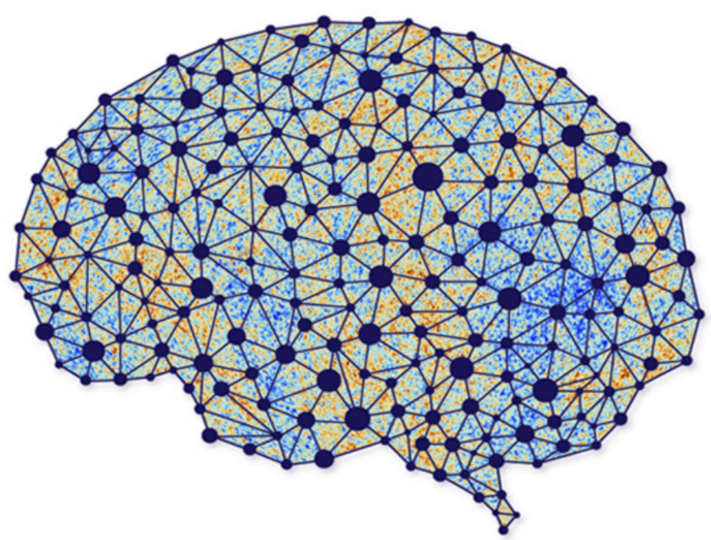
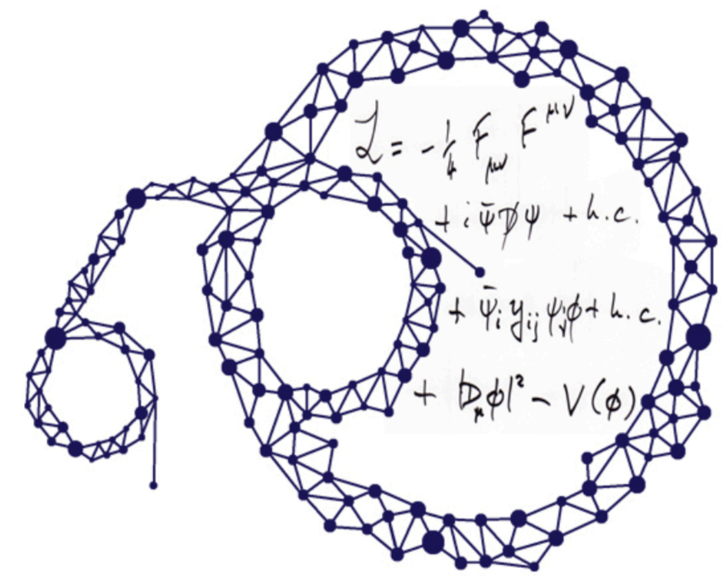
Physics 361 - Machine Learning in Physics

Lecture 24 – Normalizing flows

April 18th 2024



AI
∩
Universe



Moritz Münchmeyer

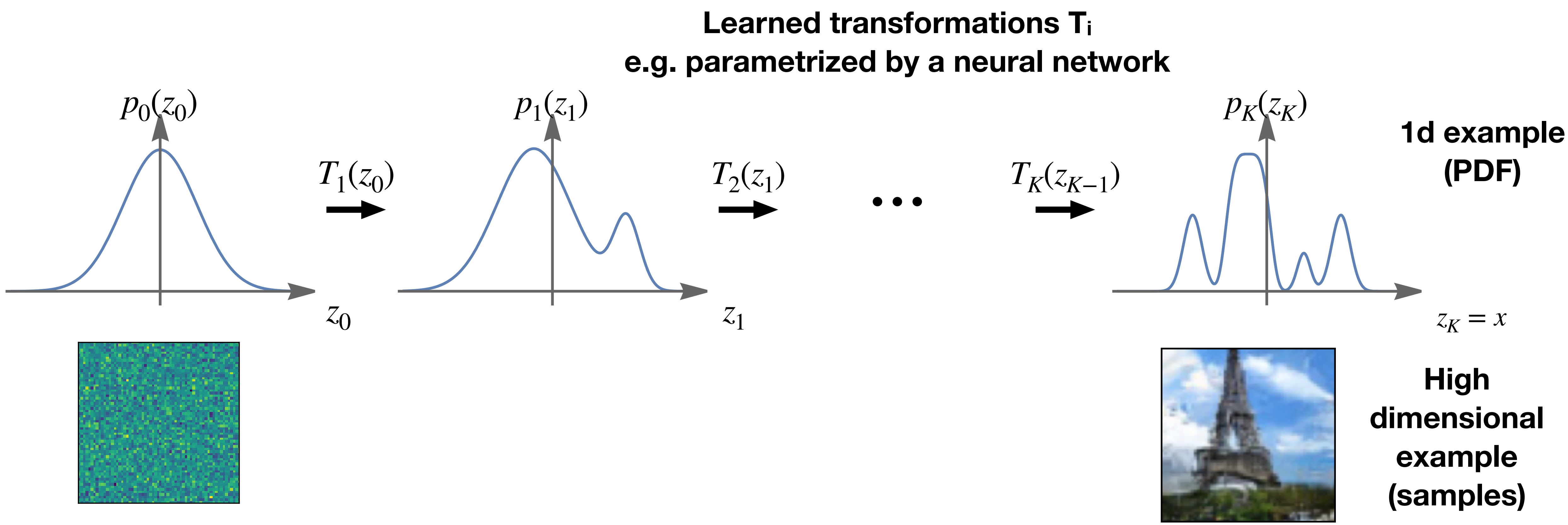
Normalizing Flows

Intro to Normalizing flows

Basics of Normalizing Flows

Normalizing flows

- Normalizing flow: Series of learned transformations that **deform a simple base distribution into a complicated target distribution**.



- Difference with most other ML methods: We learn a probability distribution, rather than an arbitrary input->output mapping.

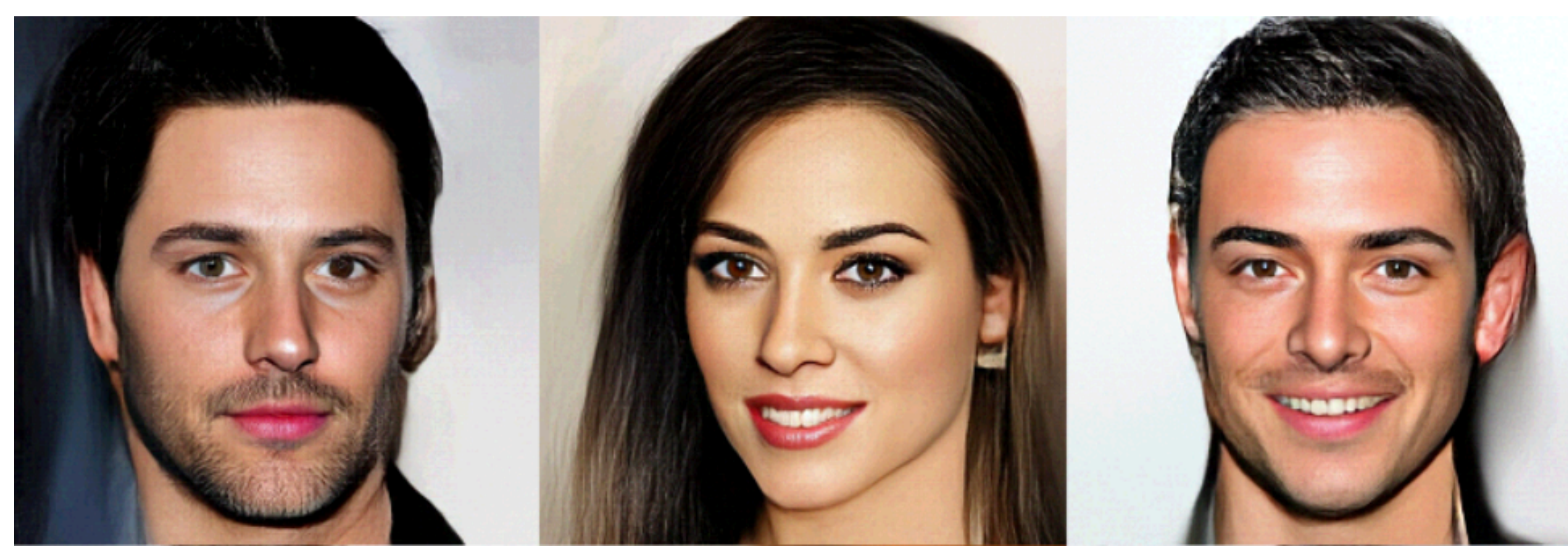
- Review: https://arxiv.org/abs/1912.02762. Widely used in physics e.g. in QFT, likelihood-free inference and cosmology

Normalizing flows are generative models

- Like GANs and diffusion models, normalizing flows are generative models.
- They can be used to generate images too. However they are not currently as good at that as these other models.
- But they can do something other models cannot: give a normalized probability density for the sample. The are real PDFs.

Real-NVP flow

Glow flow



Normalizing flows

- Transformation T (the "flow")

$$\mathbf{x} = T(\mathbf{u})$$

Change of variables of PDF

$$p_x(\mathbf{x}) = p_{\mathrm{u}}(\mathbf{u}) \left|\det J_T(\mathbf{u})\right|^{-1}$$

- Chain many "simple" transformations together to make a complicated distribution:

$$T = T_K \circ \cdots \circ T_1.$$

$$\mathbf{z}_k = T_k(\mathbf{z}_{k-1})$$

- After training **two basic operations can be performed**:
  - **Exact density evaluation** (backward mode)

    Sample x → **p(x)**

  - **Sampling from the distribution** (forward mode)

    Base distribution sample u → **Target sample x**

Basics and definitions

- **data distribution:** $\vec{x}$: D-dimensional real vector

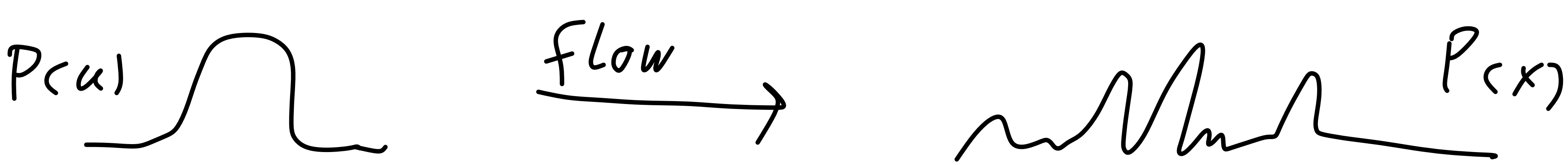
  $\vec{x} \sim P(\vec{x})$

  E.g.: $\vec{x} = \begin{pmatrix} \text{height} \\ \text{age} \\ \text{weight} \\ \text{strength} \end{pmatrix}$ of people

- Main idea of **normalizing flows**:
  Express $\vec{x}$ as a **transformation** T of a real vector $\vec{u}$ sampled from a simple **base distribution**.

$$\vec{x} = T(\vec{u}) \quad \text{where} \quad \vec{u} \sim P(\vec{u})$$

E.g.: $\vec{u}$ is a Gaussian random variable



Properties of the transformation

- To specify $p(x)$ we need to specify
  - The transformation $T(x;\theta)$ with „learned" parameters θ
  - The base distribution $p_u(u;\phi)$ with „learned" parameters ϕ } often kept static

- Defining properties of transformation T:
  - T must be invertible
  - Both T and T^{-1} must be differentiable

  Thus T is a diffeomorphism.

- $\Longrightarrow$ $u = T^{-1}(x)$ must also be D-dimensional.

Properties of the transformation

- The flow transformation is a change of variables.

$$p_x(x) = p_u(u)\,\left|\det J_T(u)\right|^{-1}$$

where $u = T^{-1}(x)$

with Jacobian

$$J_T(u) = \begin{bmatrix} \frac{\partial T_1}{\partial u_1} & \cdots & \frac{\partial T_1}{\partial u_D} \\ \vdots & \ddots & \\ \frac{\partial T_D}{\partial u_1} & & \frac{\partial T_D}{\partial u_D} \end{bmatrix}$$

$\left|\det J_T(u)\right|$ quantifies the local change of volume that „molds" $p(u)$ into $p(x)$.

Composition of transformations

- Invertible and differentiable transformations are composable. In practice we stack many simple base transformations:

$$T = T_K \circ T_{K-1} \circ \dots \circ T_1$$

$$\det J_{T_2 \circ T_1}(u) = \det J_{T_2}(T_1(u)) \det J_{T_1}(u) \quad \text{etc.}$$

- Example: Flow from Gaussian to cross shape with 4 transf.

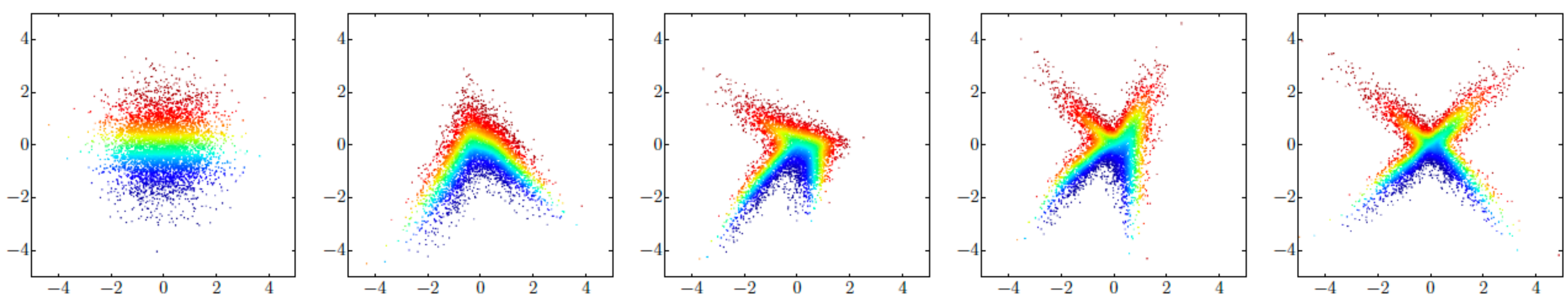


Figure 1: Example of a 4-step flow transforming samples from a standard-normal base density to a cross-shaped target density.

Forward and backward use of the flow

- The flow (after training) has two fundamental operations:
- **Sampling**: Draw samples from $p(u)$ and transform
$$x = T(u)$$
to get samples from x. Other generative models (GANs, VAE) can also sample.
- **Density evaluation**: From a sample x, we can calculate $p(x)$ as
$$p_x(x) = p_u(u)\,|\det J(u)|^{-1}$$

Computational tradeoffs

- Sampling and density evaluation have different computational requirements.

- Density evaluation requires calculating T^{-1} and $\det J_{T^{-1}}$.

- For large D, these calculations can easily be forbiddingly expensive.

$\implies$ Need flow architectures that are both flexible (expressive) and fast.

- T must be easily invertible. Most functions are not.

- Some flows are universal approximators, some aren't, and sometimes it is unknown.

Training the flow

- Goal: Fit a flow $p_x(x;\theta)$ to target distribution $p_x^*(x)$

  $\uparrow$ parameters of transformation $T(x;\theta)$ and sometimes prior $p(u;\theta)$

- Usually we have a collection of samples from $p_x^*(x)$, the training data.

- There are different measures of similarities between p_x and $p^*(x)$.

- The most popular choice: Kullback-Leibler (KL) divergence

Training the flow: KL divergence

- The loss is given by

(see lecture 4)

$$\mathcal{L}(\theta) = D_{KL}\left[p_x^* \,||\, p_x(x;\theta) \right]$$

$$= -\mathbb{E}_{p_x^*}\left[\log p_x(x;\theta) \right] + \text{const}$$

make samples from training data likely under flow pdf

- using $p_x(x) = p_u(T^{-1}(x)) \, |\det J_{T^{-1}}(x)|$

$$\Rightarrow \quad \mathcal{L}(\theta) = -\frac{1}{N}\sum_{n=1}^{N} \log p_u(T^{-1}(x_n,\theta)) + \log|\det J_{T^{-1}}(x_n;\theta)|$$

- Training: SGD for $\nabla_\theta \mathcal{L}$

Designing normalizing flows

Designing efficient flows

- We stack many simple building blocks:

$$T = T_K \circ \cdots \circ T_1$$

$$z_K = T_K(z_{K-1})$$

$$\log |J_T(z)| = \sum_{K=1}^{K} \log |J_{T_K}(z_{K-1})|$$

more depth $\rightarrow$ $O(K)$ growth in cost :)

- Note: making T_K invertible in theory and actually inverting it are very different requirements
$\Rightarrow$ want easily invertible functions

Designing efficient flows

- We want a tractable Jacobian determinant.

  For a general map D inputs $\to$ D outputs calculating $\det J$ has cost $\mathcal{O}(D^3)$, often intractable for large D.

  Most flows are using forms for which $\det J$ is $\mathcal{O}(D)$.

- We now discuss building blocks T_K that have this feature.

$$\vec{z}\,' = f(\vec{z})$$

Planar flows

- Building block:

elementwise non-linearity

1505.05770

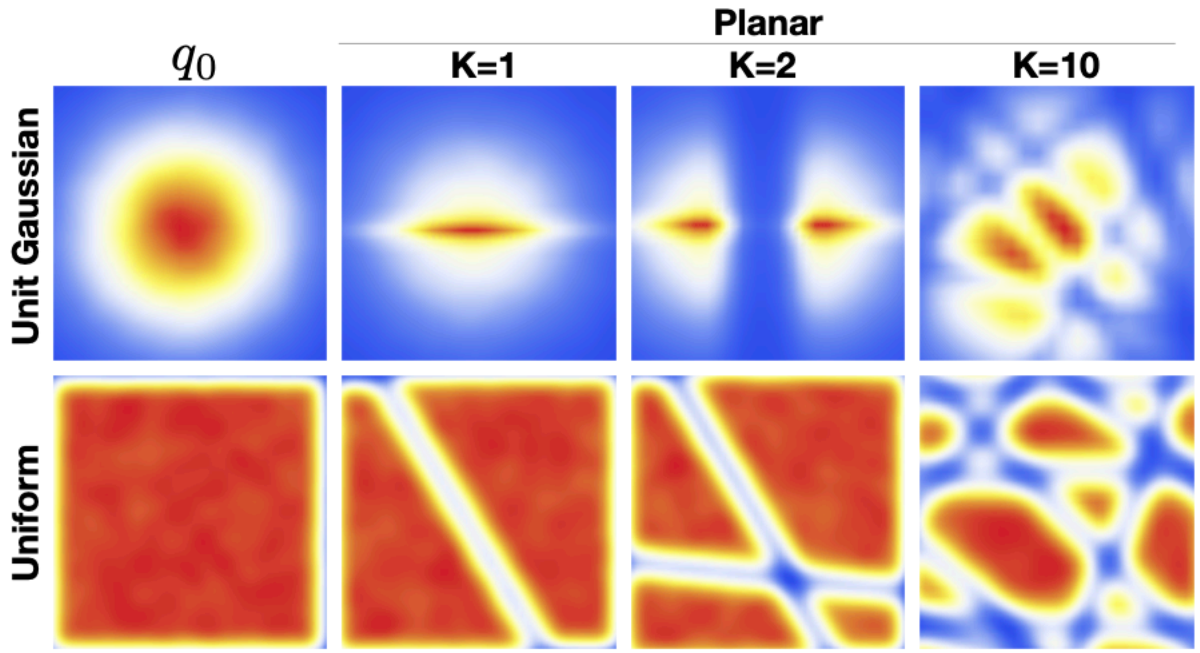
$$f(\vec{z}) = \vec{z} + \vec{u}\, h(\vec{w}^T \vec{z} + b)$$

vectors dim D

- can calculate det J in $\mathcal{O}(D)$ time.
- stack many of these transformations

- Example

z: 2dim.



Not suitable for large dimensions, since transformations are too local (i.e. affect a small volume)

Autoregressive flows

- Autoregressive property: $\vec{z} = (z_1, z_2, z_3 \dots z_i \dots z_D)$

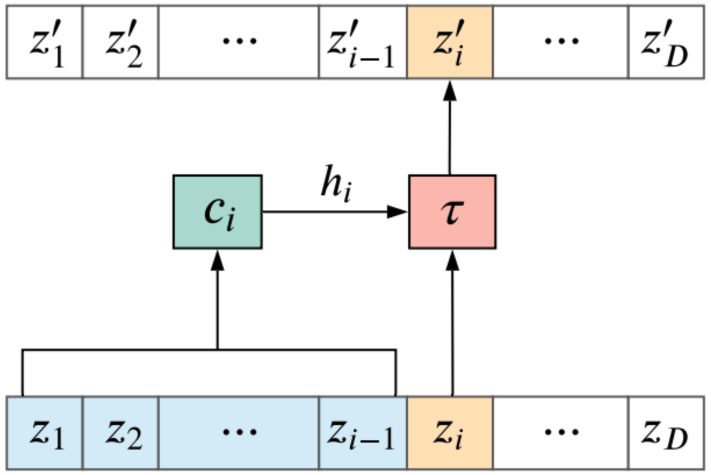
  variable z_i depends on $z_{1:i}$ only

$$z_i' = \tau(z_i, \vec{h}_i)$$

„transformer": must be invertible, i.e. monotonic in z_i

$$\vec{h}_i = c_i(\vec{z}_{<i})$$

„conditioner": modulates the transformer



- This leads to a triangular Jacobian:

$$J_T(z) = \begin{bmatrix} \ddots & 0 \\ \sim & \ddots \end{bmatrix} \Rightarrow \det J = \prod_i J_{ii} \qquad \mathcal{O}(d)$$

Affine Autoregressive Flows

- Particalarly simple linear transformer

$$\tau(z_i) = \alpha_i z_i + \beta_i$$

shift and scale z_i

invertible for $\alpha \neq 0$

"affine transformation"

or: $\exp^{\alpha_i} z_i + \beta_i$

- The conditioner here is:

α, β depend on $\{z_{<i}\}$ by some neural network parametrisation

$$\{z_{<i}\} \longrightarrow \boxed{NN_\Theta} \longrightarrow \alpha_i,\ \beta_i$$

NN: e.g. MLP or CNN

So the conditioner has the learned parameters Θ.

Different autoregressive flows

- Again, autoregressive flows are

$$z_i' = \tau(z_i, \vec{h}_i)$$

„transformer". must be invertible.
i.e. monotonic in z_i

$$\vec{h}_i = c_i(\vec{z}_{<i})$$

„conditioner". modulates the transformer. not a bijection.

- Many different transformers and conditioners have been proposed. Popular:
  - Masked autoregressive flows (MAF)
  - Inverse autoregressive flows (IAF)

They have different cost and expressivity.

Illustration of the MAF flow

Source: 2310.03741

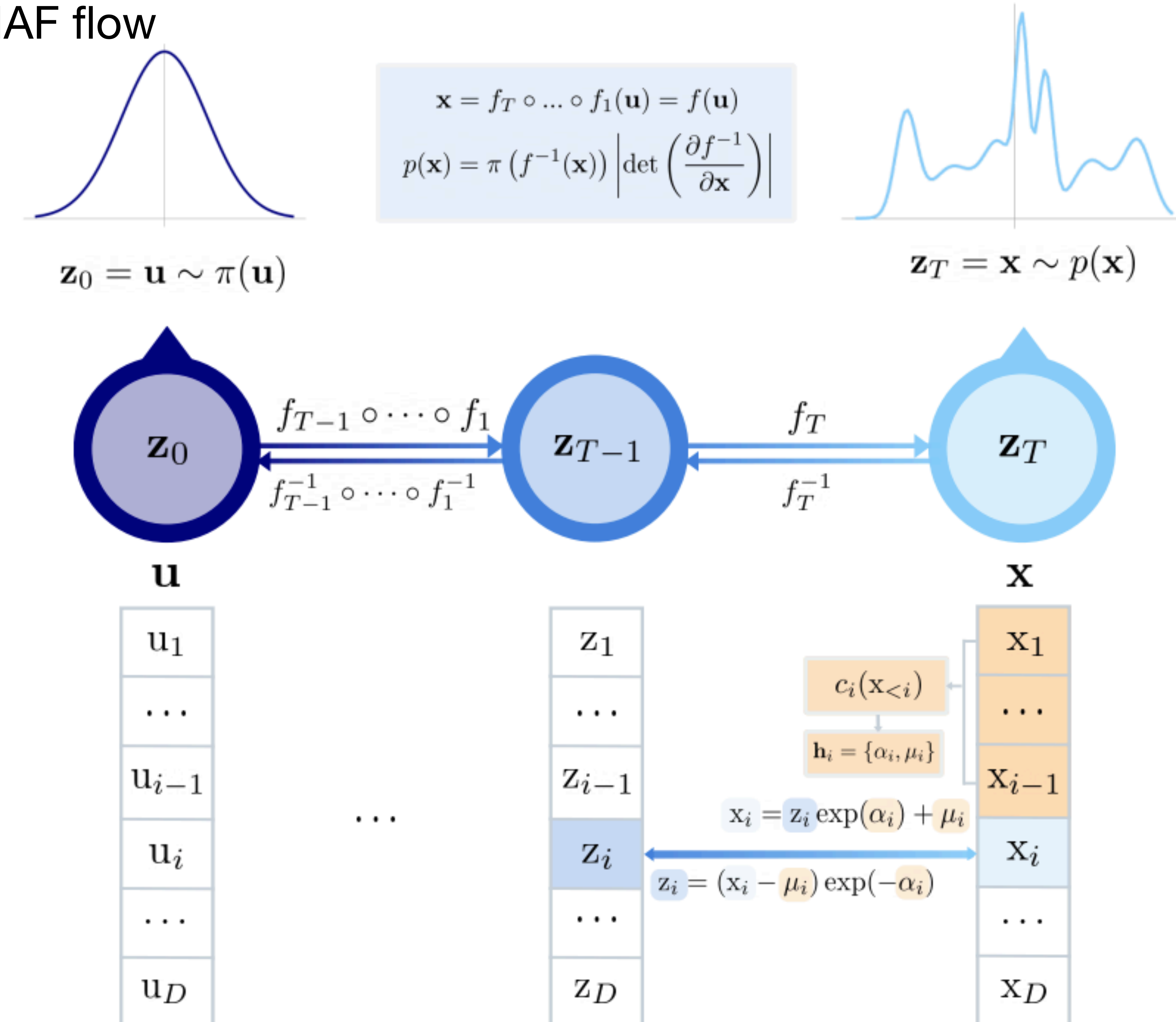


Figure 2. Diagram of how normalizing flows work, with the specific example of Masked Autoregressive Flows. The samples from the vector $\mathbf{z}_0 = \mathbf{u}$, sampled from the simple distribution $\pi(\mathbf{u})$, are deformed through the sequence of transformations $f = f_T \circ \cdots \circ f_1$ into those of $\mathbf{z}_T = \mathbf{x}$, which follow a more complex distribution $p(\mathbf{x})$. In the lower panel, we illustrate the conditioner that "masks out" the connections between z_i and $\mathbf{h}_{<i}$, as well as the affine functions applied to the vector components.

Real-NVP

2d spatial array

- Baseline flow for many „image" applications
  1605.08803

- Special case of affine autoregressive flow:
  Split variables into two halfs:

$$z'_{1:k} = z_{1:k}$$

$$z'_{k+1:D} = z_{k+1:d}\, \alpha(z_{1:k}) + \beta(z_{1:k})$$

$\Rightarrow$ Jacobian $\left(\begin{array}{c|c} \begin{matrix} 1 & & \\ & 1 & \\ & & 1 \end{matrix} & 0 \\ \hline \begin{matrix} \cdots & \cdots \\ \cdots & \cdots \\ \cdots & \cdots \end{matrix} & \begin{matrix} \ddots & 0 \\ 0 & \ddots \end{matrix} \end{array}\right)$

Allows parrallel computation of z' since all inputs are available.

- stack many such layers with different orderings of the variables.

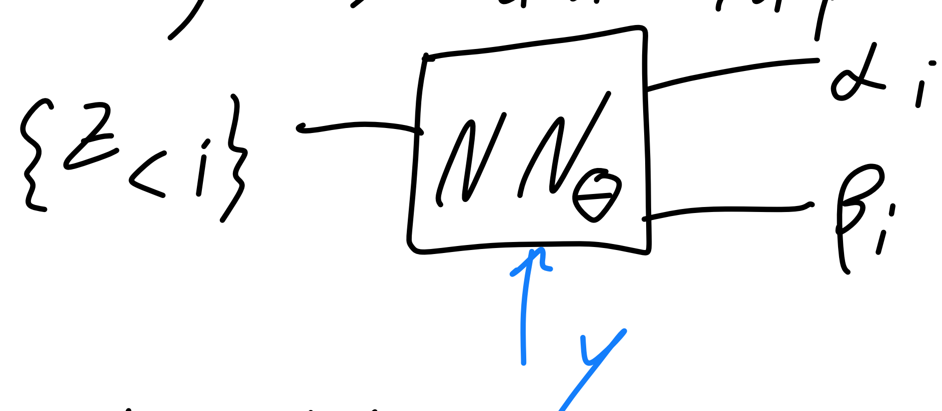
Use example: https://arxiv.org/abs/2101.08176 Normalizing Flows for Lattice Field Theory

Conditional flows

- We want to learn not just PDFs $p(x)$ but also **conditional PDFs**

$$p(x \mid y)$$

- This is achieved by making the flow transformations T dependent on the condition y.
- For example, in a autoregressive flow, the neural network which parametrizes z now also gets an input of y:



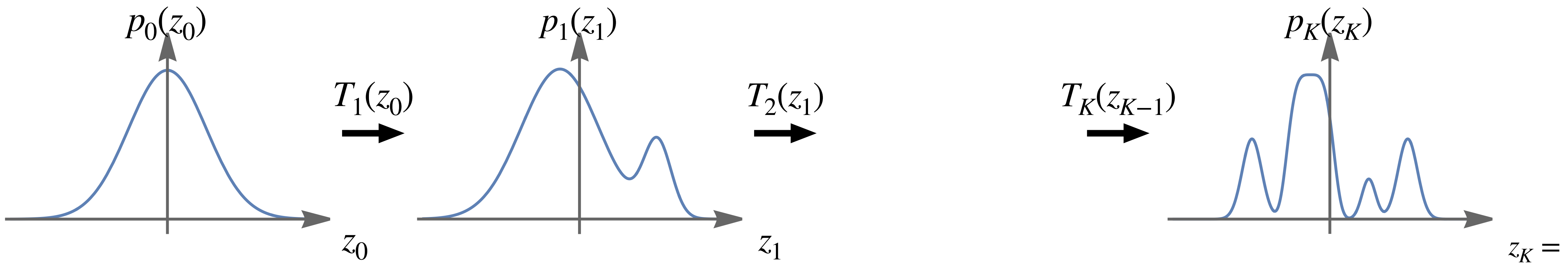
- Training works the same as before (KL-div.).

Research Example: Normalizing flows to model the matter distribution in cosmology

(work from my group)

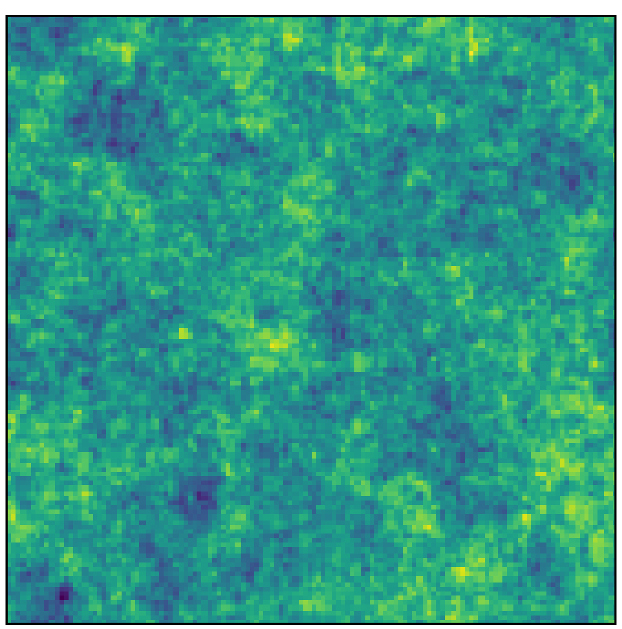
NFs vs structure formation

Gaussian initial conditions PDF morphs into complicated late-time matter distribution.

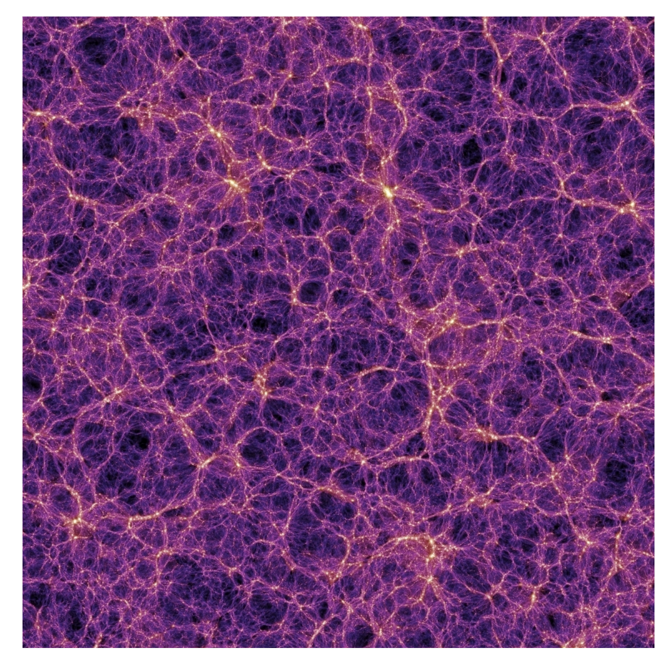


Gaussian primordial matter perturbations

Non-gaussian matter/galaxy distribution today

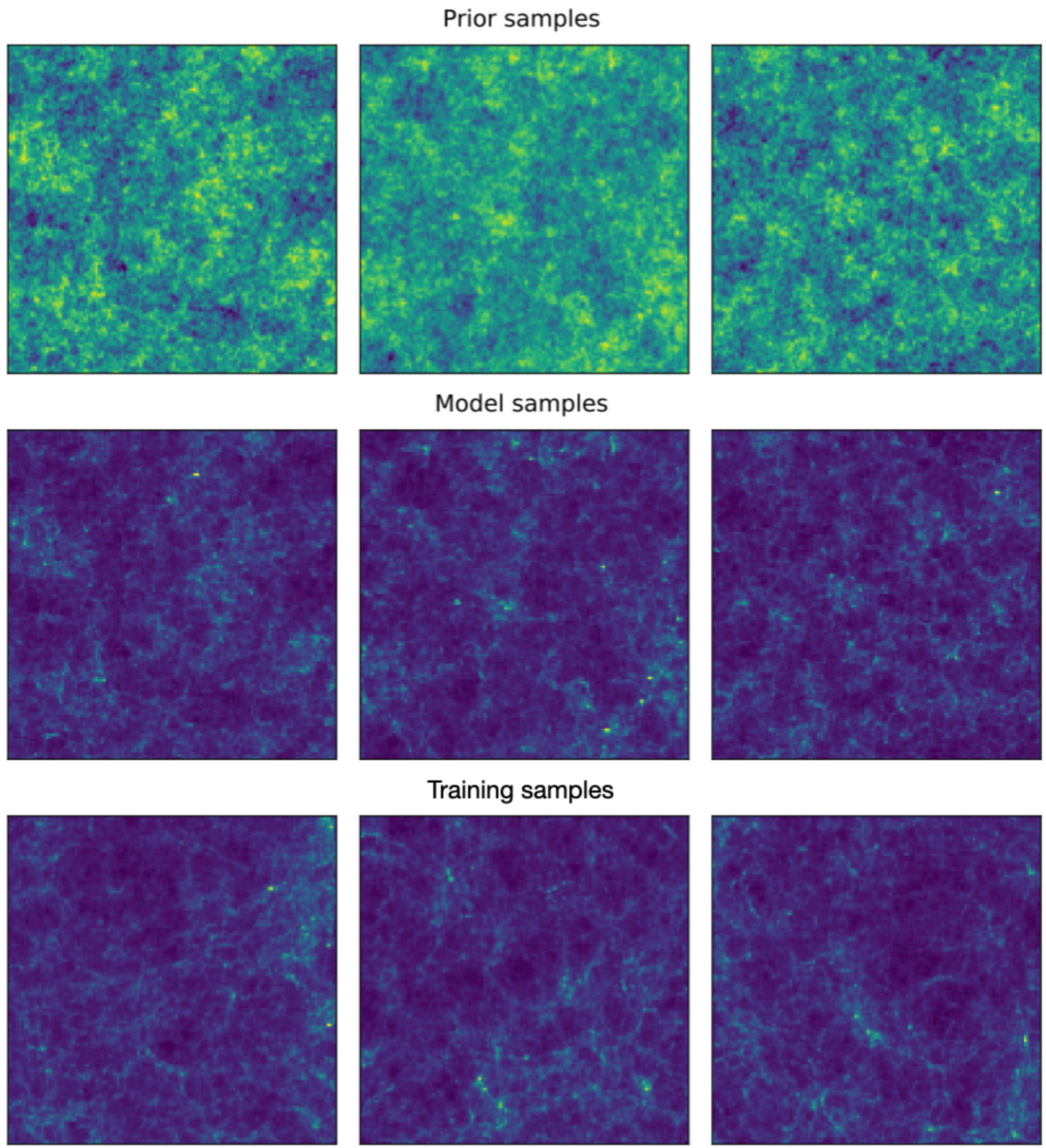


Cosmological time evolution



Rouhiainen, MM: arXiv:2105.12024 Normalizing flows for random fields in cosmology

Flowing from a correlated Gaussian to todays matter distribution



Flow:
RealNVP

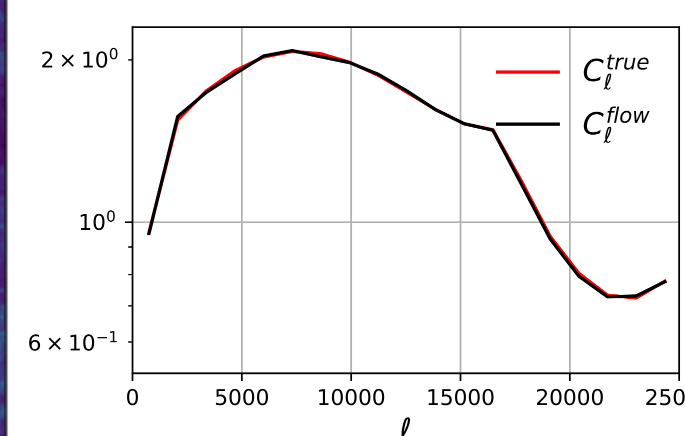
Density peaks match, as in physical structure formation.

The flow learned to deform a Gaussian PDF into a highly non-Gaussian PDF

$125\ h^{-1}$ Mpc

About 128^3 particles per field

Power spectrum matches



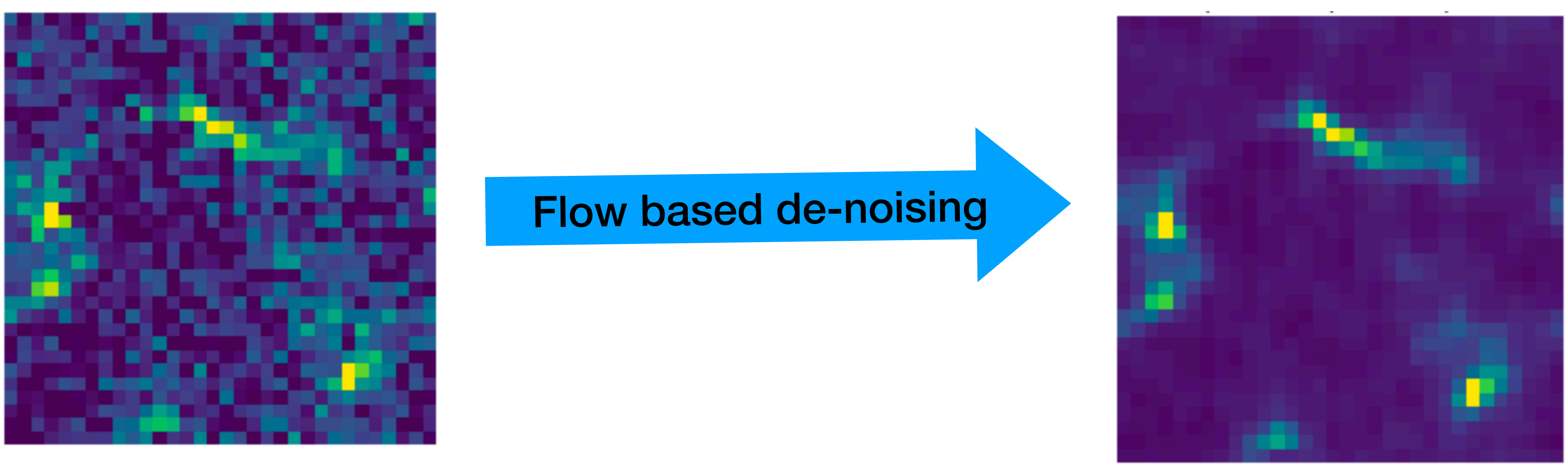
De-noising with a Generative Prior

In data analysis in cosmology we often make use of **Gaussian priors (Wiener Filter)**. This is no longer justified for very high resolution observations. Using the trained normalizing flow we can **include non-Gaussian priors:**

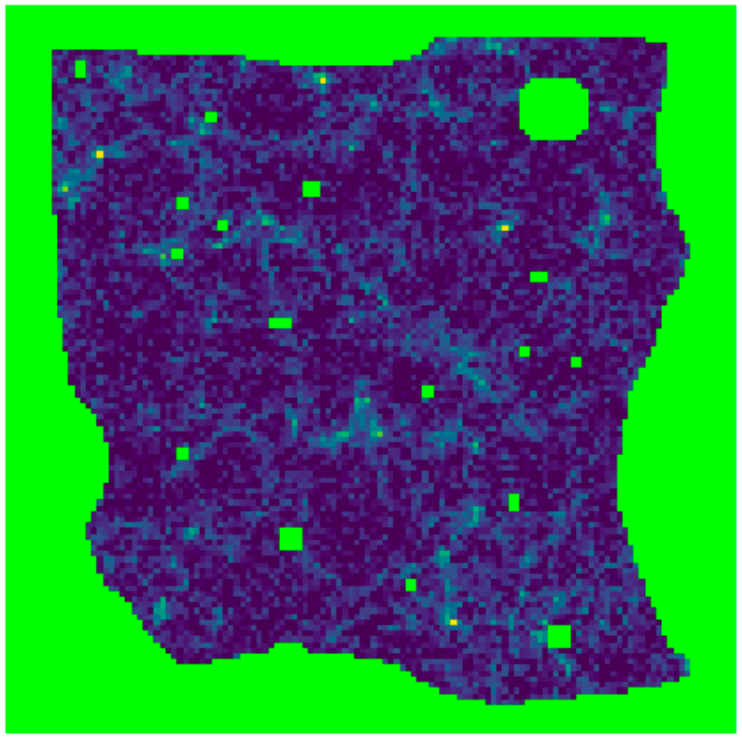
$$\ln p(y\,|\,d) = -\frac{1}{2}(y-d)^{\mathrm{T}} N^{-1}(y-d) - \ln p_{\mathrm{flow}}(y)$$

True matter field   Noisy observation

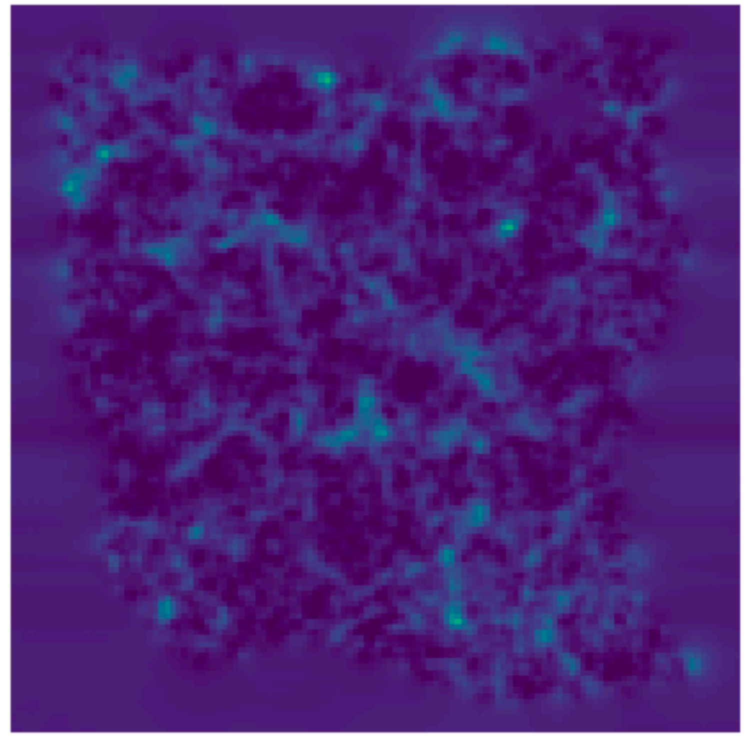
We use a flow trained on simulations of the matter distribution. Then we use this knowledge of the matter PDF to **de-noise an observation of the matter field by maximizing the posterior**.



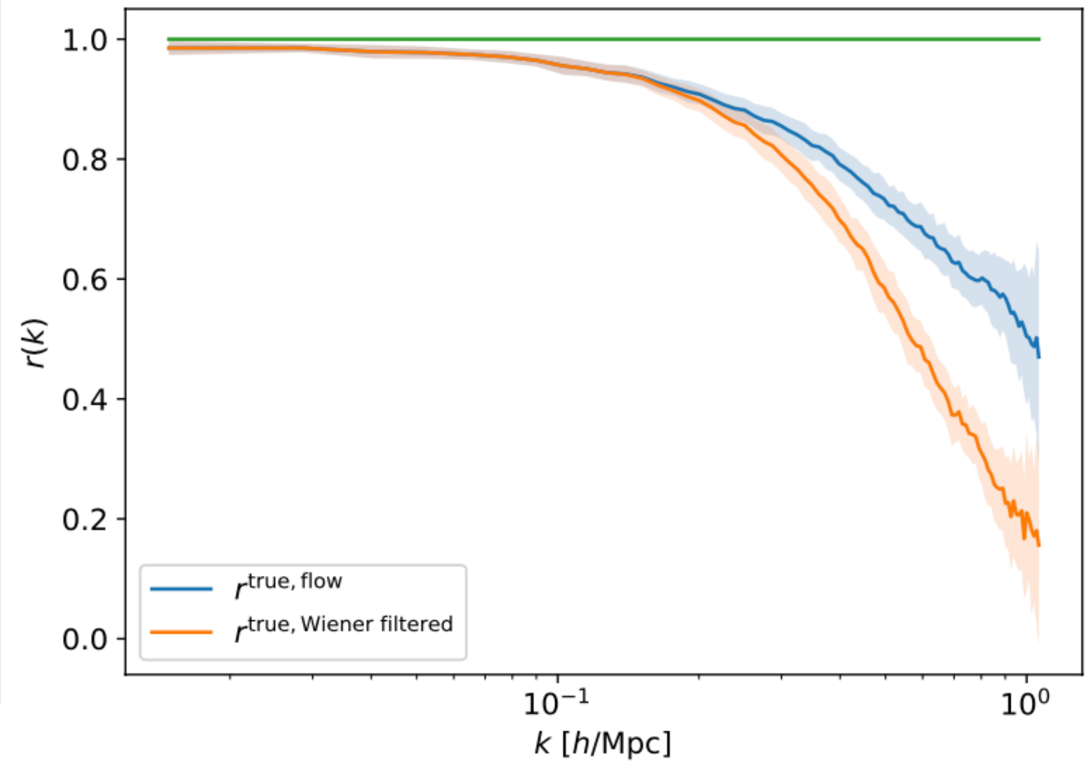
De-noising the observed matter field

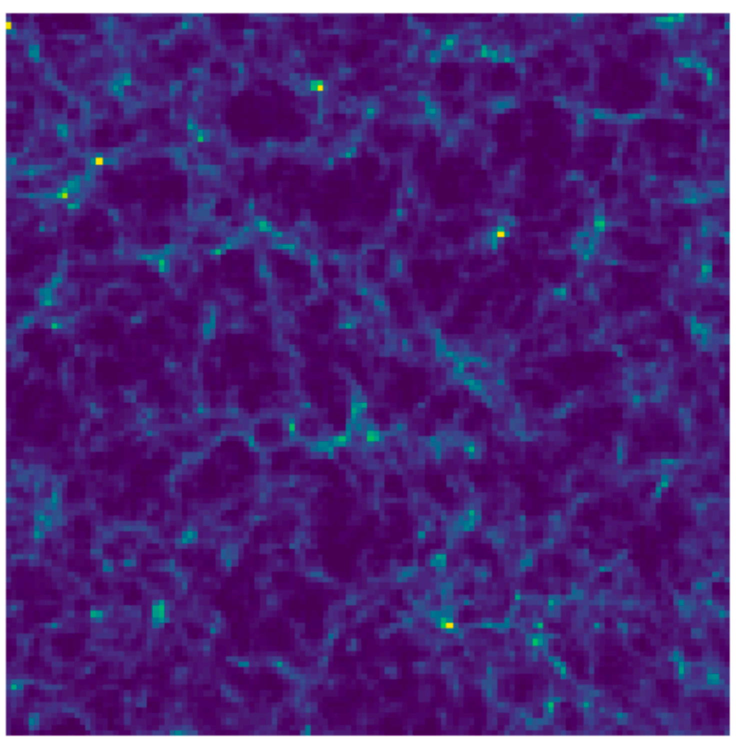


Observed (noisy, masked)

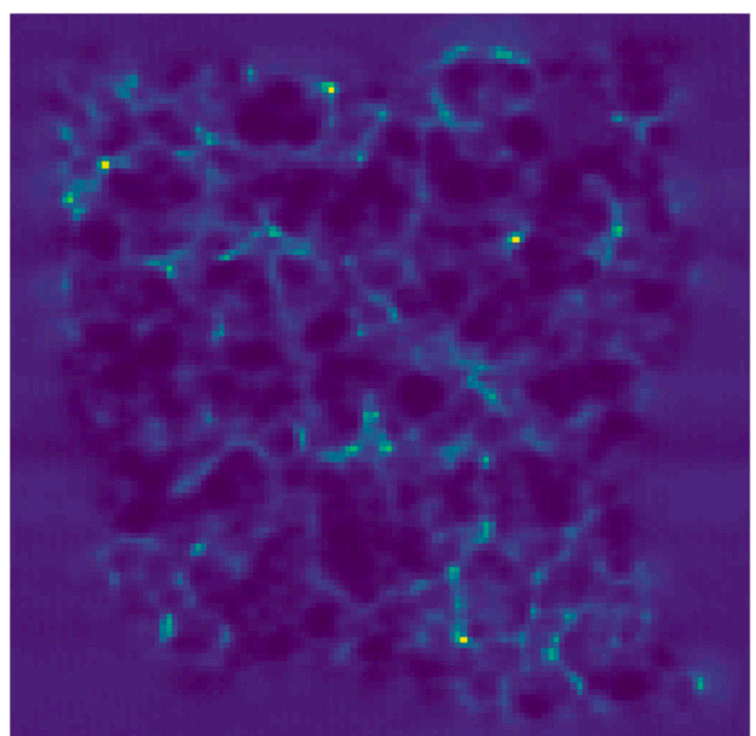


Wiener filtered





Truth



Flow **MAP**

As expected, the NF lowers the reconstruction noise on non-linear scales compared to the Wiener filter.

Generative de-noising is useful in many other domains.

Rouhiainen, MM: arXiv:2211.15161 De-noising non-Gaussian fields in cosmology with normalizing flows