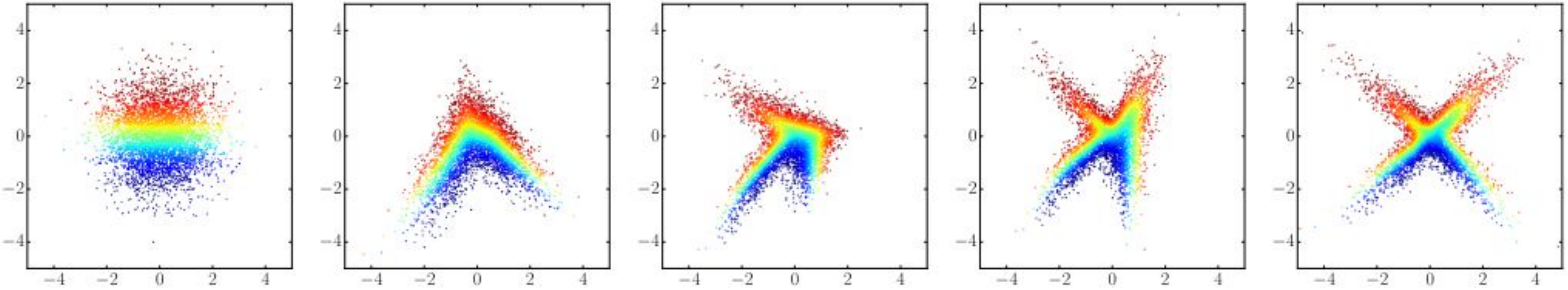
ML Tools for Precision Physics in HEP

Dr. Davide Valsecchi (ETH Zurich)

11/10/2024

Normalizing Flows

Normalizing direction → density estimation



Sampling direction

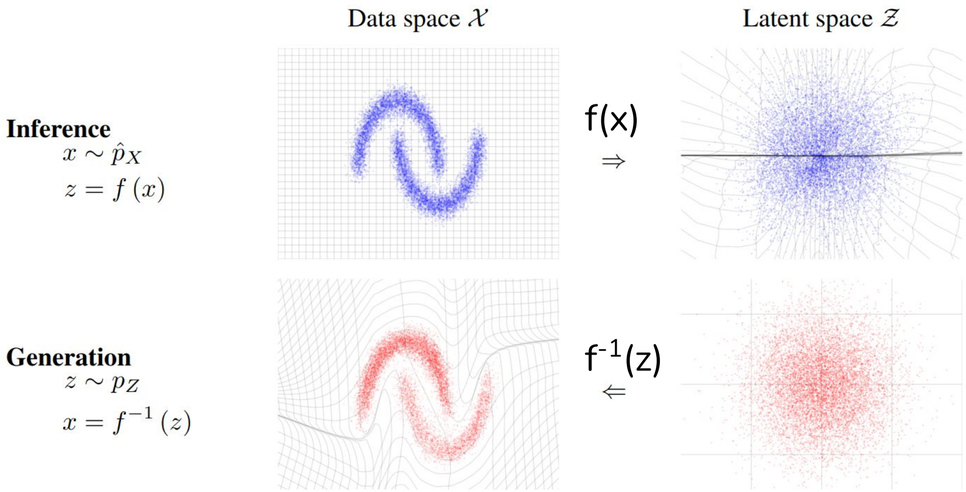
Normalizing flows

From the rules of change of integration variables

$$p_X(x) = p_Z(f(x)) \left| \det \left(\frac{\partial f(x)}{\partial x^T} \right) \right|$$

$$\log(p_X(x)) = \log\Big(p_Z(f(x))\Big) + \log\left(\left|\det\left(\frac{\partial f(x)}{\partial x^T}\right)\right|\right),$$

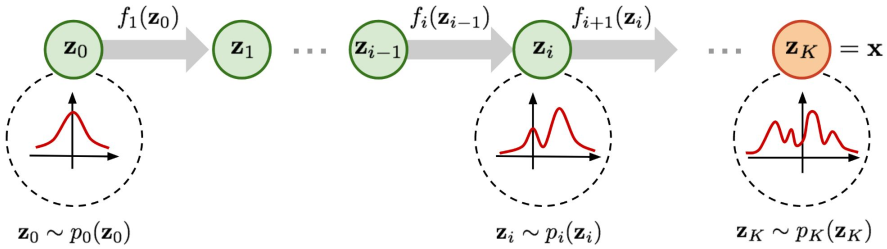
where f(x) goes in the “normalizing” direction to the z latent space.



We can both **sample** and evaluate the **density**

- If the p.d.f in the l**atent space is tractable** (multidim gaussian, uniform)
- if the transformation is **invertible**

Requirement: the jacobian of the transformation must be computed in an efficient way
→ this defines the possible implementation of the flows



Expressiveness: transformations are composable!

$$(T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1}$$
$$\det J_{T_2 \circ T_1}(\mathbf{u}) = \det J_{T_2}(T_1(\mathbf{u})) \cdot \det J_{T_1}(\mathbf{u}).$$

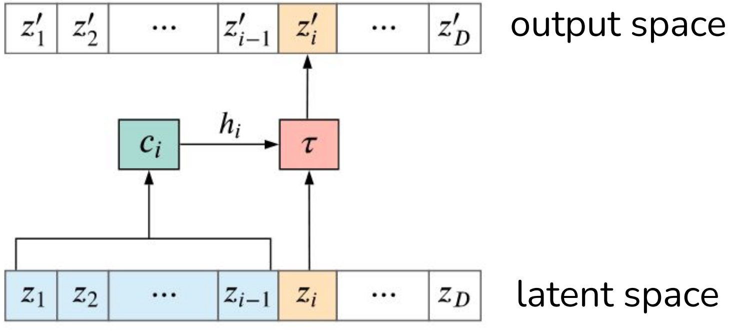
How to build a flow

We need to model **non-factorizable p.d.f:**
dimensions depend non linearly on each other

DNNs are not invertible: use DNN as **conditioners** c_i

which parametrize invertible **transformations** τ
for which we have analytical inversion

Choose a structure with an efficient jacobian.



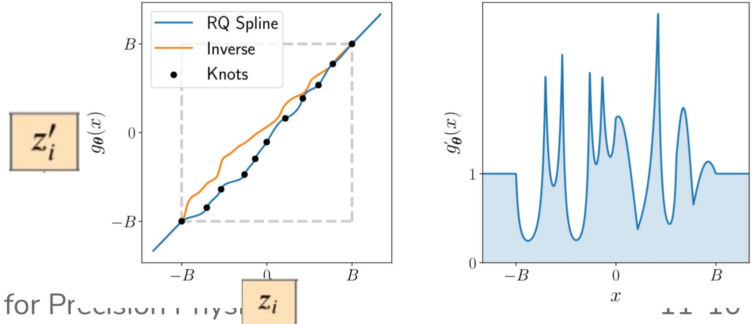
(a) Forward

The **transformation** τ can be:

affine: *μ, α parameters* from the DNN conditioner

$$z'_i = (z_i - \mu_i)\exp(-\alpha_i)$$

or **spline** based: model N knots with the DNN conditioner, which creates a spline to transform differently each dimension → very expressive



arxiv1906.04032
arxiv1912.02762
arxiv1705.07057

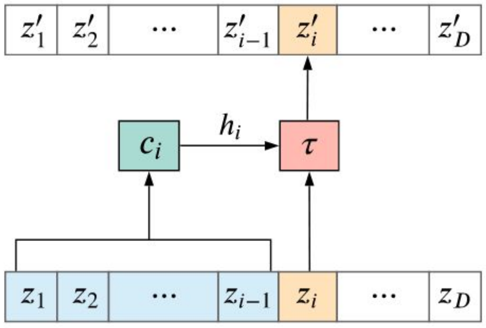
Conditioners

arxiv1912.02762

To model complex relations in the p.d.f. phase-space, the dimensions must *interact* between each other.

Two strategies to build **easily computable Jacobians**

- **coupling** transformations: split the space in two and make one group depends on the other (then rotate)
- **autoregressive** transformation: dimension X_j is conditioned only by $X_{0<i<j}$



(a) Forward

In both cases you get a lower-triangular Jacobian

$$J_{f_\phi}(\mathbf{z}) = \begin{bmatrix} \frac{\partial \tau}{\partial \mathrm{z}_1}(\mathrm{z}_1; \boldsymbol{h}_1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{L}(\mathbf{z}) & & \frac{\partial \tau}{\partial \mathrm{z}_D}(\mathrm{z}_D; \boldsymbol{h}_D) \end{bmatrix}.$$

The *logdet* is just the sum of the diagonal terms

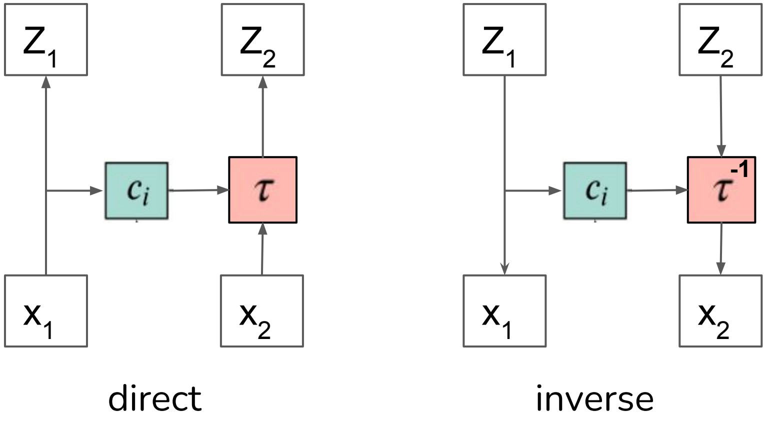
$$\log \left|\det J_{f_\phi}(\mathbf{z})\right| = \log \left| \prod_{i=1}^{D} \frac{\partial \tau}{\partial \mathrm{z}_i}(\mathrm{z}_i; \boldsymbol{h}_i) \right| = \sum_{i=1}^{D} \log \left| \frac{\partial \tau}{\partial \mathrm{z}_i}(\mathrm{z}_i; \boldsymbol{h}_i) \right|.$$

arxiv1705.07057

Coupling structure

- Split the **input space in half** and make one group depends on the other
- Shuffle the grouping (permute or rotate)
- Stack many layers to model all the correlations

Coupling layers



Pros:

- Fully parallelizable over dimensions **in both directions**: 1 pass computation, super fast on GPU
- Fast to use in both sampling and density estimations

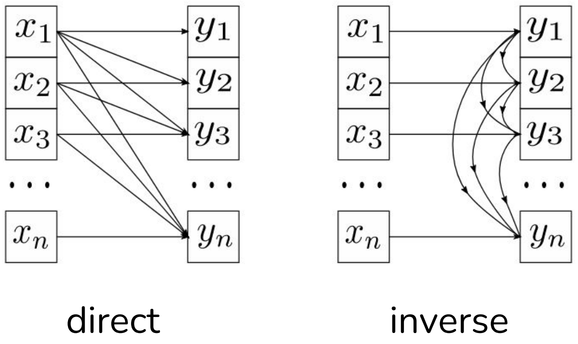
Cons:

- Many layers are needed to fully model the correlations in the input space D dimensions. (at least D layers usually)

Autoregressive structure

- dimension X_k is conditioned only by $X_{0<i<k}$
- Implemented with Masked Autoencoders (MADE):
    - Fully connected neural networks with masked applied at each layer to create the autoregressive structure

Autoregressive



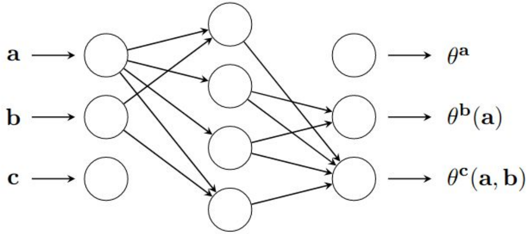
Pro:

- More powerful than coupling strategy:
    - using few stacked layers all the dimensions talks to each other

Cons:

- Parallel in one direction, D steps in the version (D = dimension of the input space)
- Need to choose the direction of the implementation if we need faster sampling (IAF arxiv1606.04934) or faster density estimation (MAF arxiv1705.07057)

How to create an autoregressive function with a feed-forward neural network (MADE)



How to train a flow

It depends if you **the target p.d.f** is:

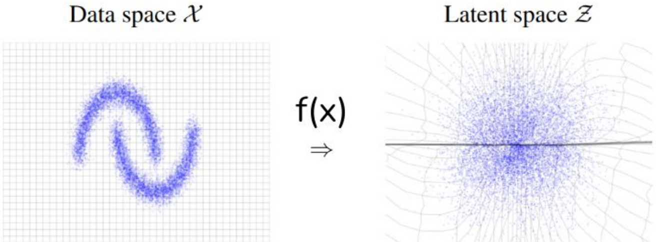
1. easy to **sample,** difficult to **evaluate**: p.d.f. of MC or Data in a control region → we have events
2. difficult to **sample**, easy to **evaluate**: multidimensional integrand → we have the function

1. Training by **maximum likelihood**

Take samples X, get their density from the flow, maximize the total likelihood, optimize flow parameters by gradient descent

$$\begin{aligned} D_{\mathrm{KL}}\left[p^*_{\mathrm{x}}(\mathbf{x}) \,\|\, p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta})\right] &= -\mathbb{E}_{p^*_{\mathrm{x}}(\mathbf{x})}\left[\log p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta})\right] + \text{const} \\ &\approx -\frac{1}{N}\sum_{n=1}^{N} \log p_{\mathrm{x}}(\mathbf{x}_n;\boldsymbol{\theta}) + \text{const} \\ &= -\frac{1}{N}\sum_{n=1}^{N} \log p_{\mathrm{u}}(T^{-1}(\mathbf{x}_n;\boldsymbol{\phi});\boldsymbol{\psi}) + \log\left|J_{T^{-1}}(\mathbf{x}_n;\boldsymbol{\phi})\right| + \text{const.} \end{aligned}$$

KL divergence

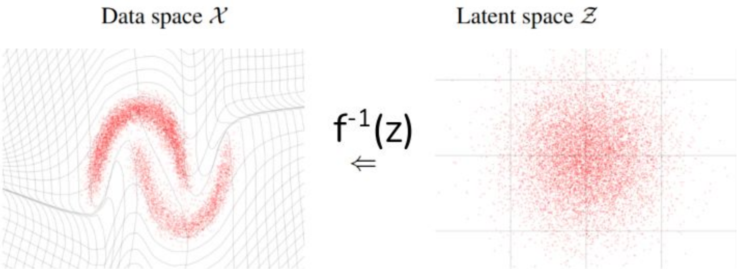


2. Training by **sampling**

Sample Z samples from the latent space,
Pass through the flow to get X samples and their density p(x)
Evaluate the function → compute a divergence between p(x) and target p*(x) → optimize flow parameters by gradient descent

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= D_{\mathrm{KL}}\left[p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta}) \,\|\, p^*_{\mathrm{x}}(\mathbf{x})\right] \\ &= \mathbb{E}_{p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta})}\left[\log p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta}) - \log p^*_{\mathrm{x}}(\mathbf{x})\right] \\ &= \mathbb{E}_{p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})}\left[\log p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi}) - \log\left|\det J_T(\mathbf{u};\boldsymbol{\phi})\right| - \log p^*_{\mathrm{x}}(T(\mathbf{u};\boldsymbol{\phi}))\right]. \end{aligned}$$

reverse KL divergence

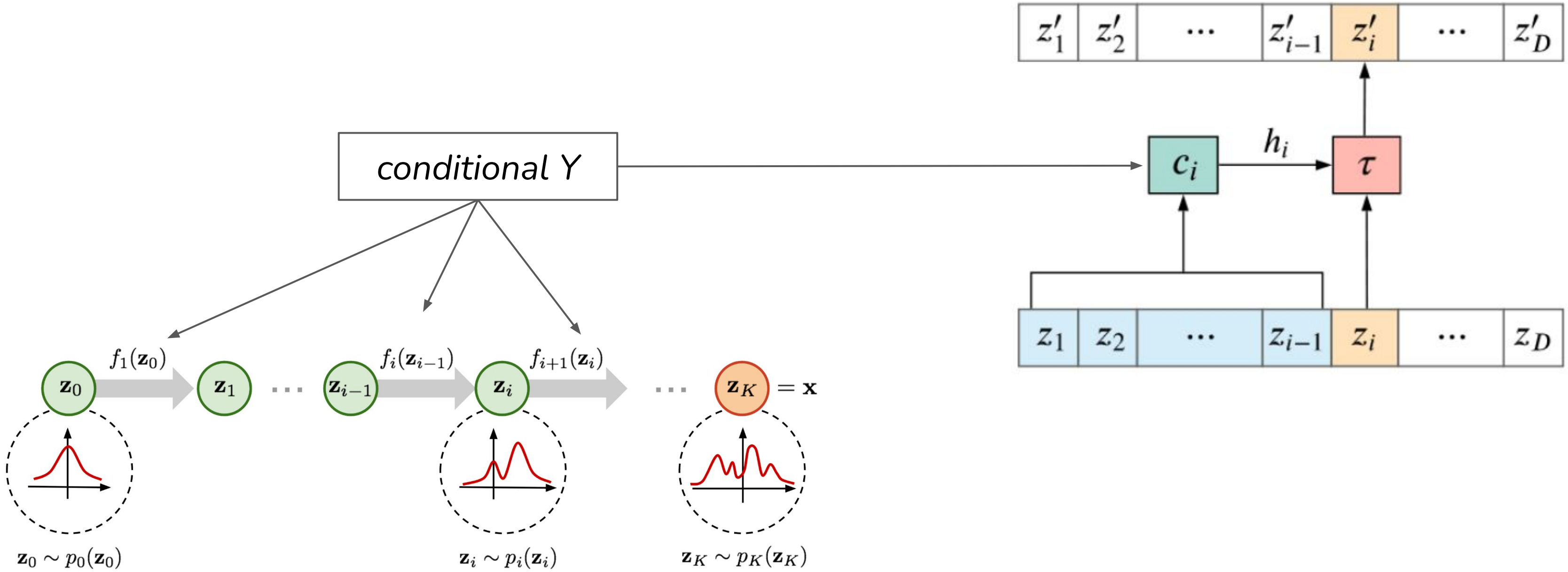


Flows conditioning

A flow can be conditioned by external information to model p(x | y):

- include the dependence in the conditioner DNNs

- N.B. the conditioning dimensions ***y* are not part** of the flow

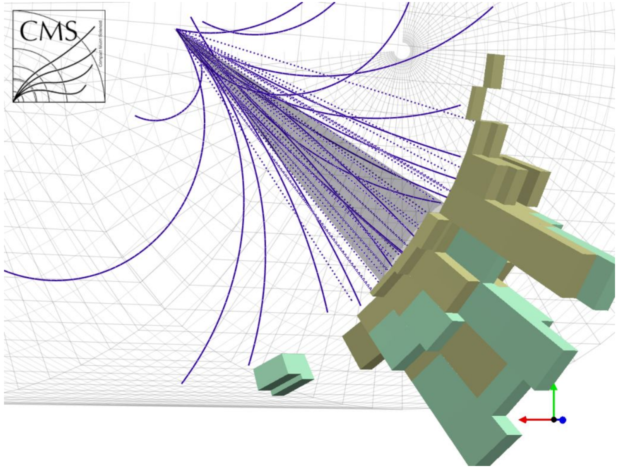


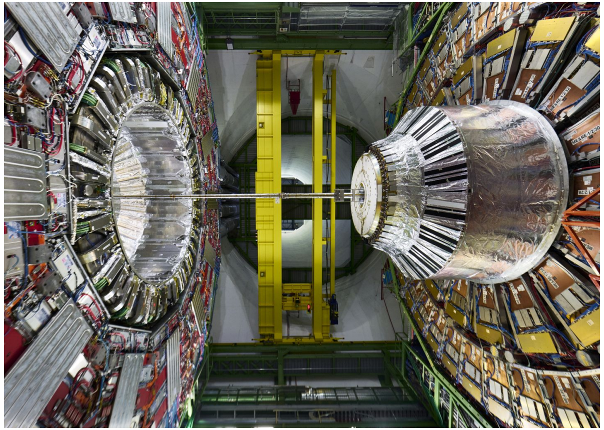
Applications in HEP

- Simple example: initial gluon momenta from reconstructed objects
- Importance sampling (MC integration, MCMC processes)
- Conditional unfolding
- Matrix Element methods
- Data/MC morphing/reweighting:

One flow to correct them all: improving simulations in high-energy physics with a single normalising flow and a switch (C. C. Daumann, M. Donega, J. Erdmann, M. Galli, J.L.Spah, D. Valsecchi)

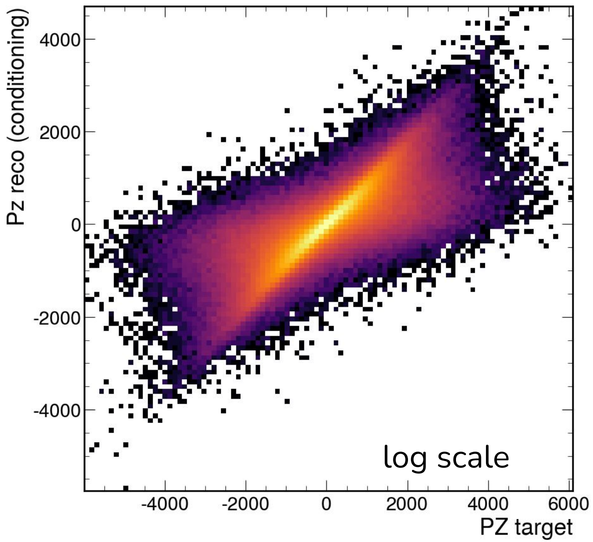
https://arxiv.org/abs/2403.18582



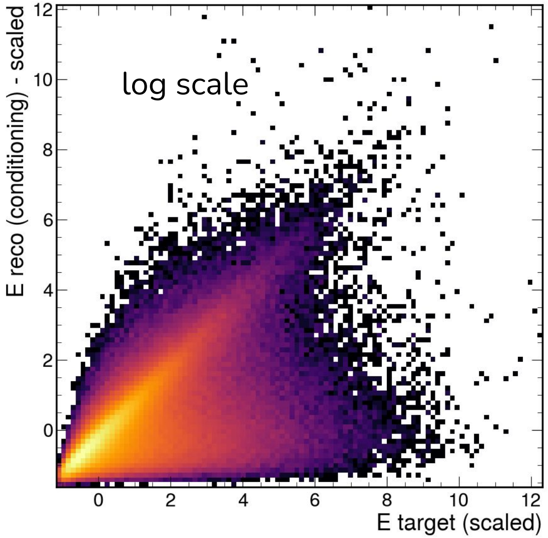


Example: gluon momenta from reco-level boost

- Get the **initial parton fractions** from the **final state total boost**
- Easy task given the good pileup rejection of the CMS reconstruction
    - Strong correlation between the conditioning variables (reconstruction level boost) and the target variables (incoming gluon momenta)



gen-level

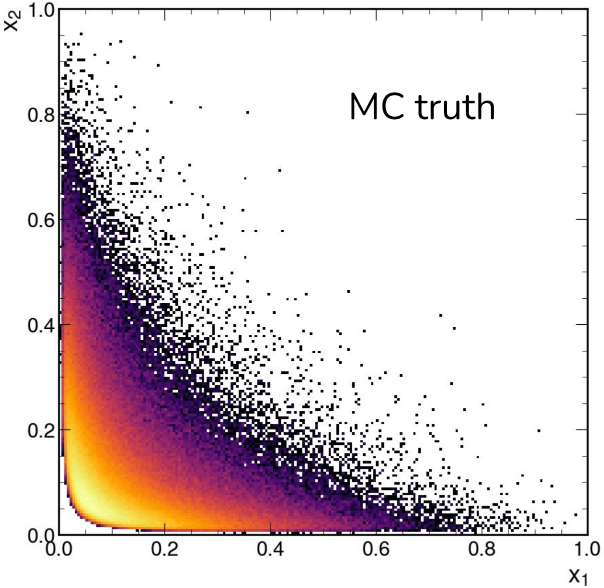


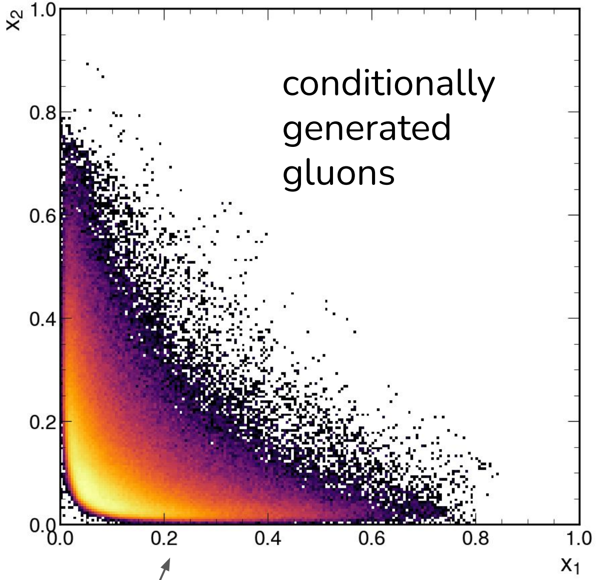
gen-level

Example: gluon momenta from reco-level boost

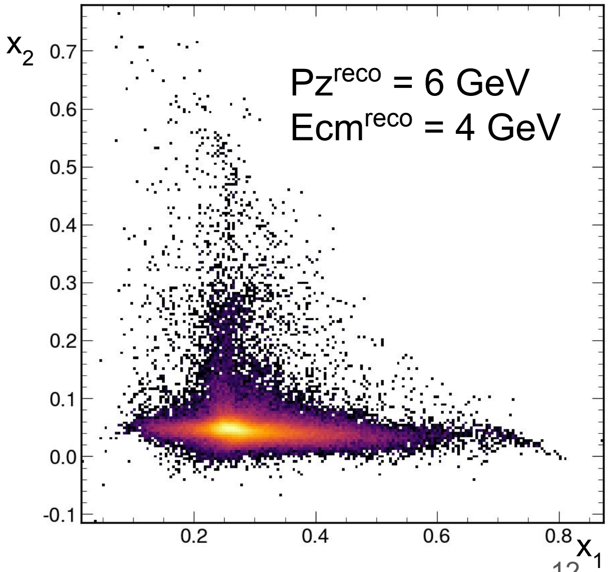
- Built a simple autoregressive spline-based conditional flow:
    - modelling ***p(gluon | reco boost)***
    - 2D conditional space (pz, E), 2D feature space (pz, E)

- Train it using the gluon and reco level boost from MC by maximum likelihood





we get conditional generation!



generated one point for each MC events

Practice: Generation of neutrinos in WW events

We can use a normalizing flow to generate
the distribution of possible neutrinos quadrimomenta in
WW VBS events.

We will build a **conditional p.d.f (neutrinos | events).**

The conditioning vector needs to be fixed dimensional
... we need an encoder or the event information
→ transformer encoder + accumulation

6_NeutrinoFlow notebook