Probabilistic models in ML for HEP

Dr. Davide Valsecchi
(Prof. Wallny - CMS group)

1st IPA ML Workshop

21/03/2023

Outline

Part 1: Normalizing Flows

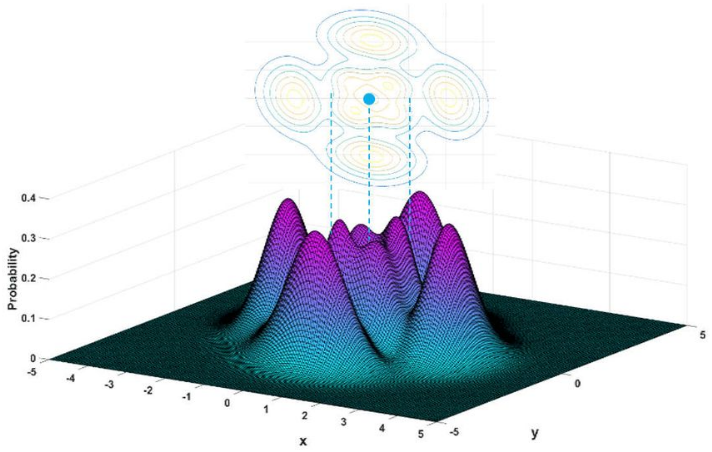
- when you need one
- how they work
- how to train them

Part 2: Examples of applications for HEP

- flows for importance sampling → fast MC integration
- Unfolding detector level information
- MEM method computation with flows

Example

Your data is described by a complex multidimensional p.d.f



- very often multimodal
    - not factorizable as product of gaussians for each dimension
- We need both **sampling** and **density estimation**
- almost always **conditional** p.d.f. p(x|y)

How to work with ML and probability densities?

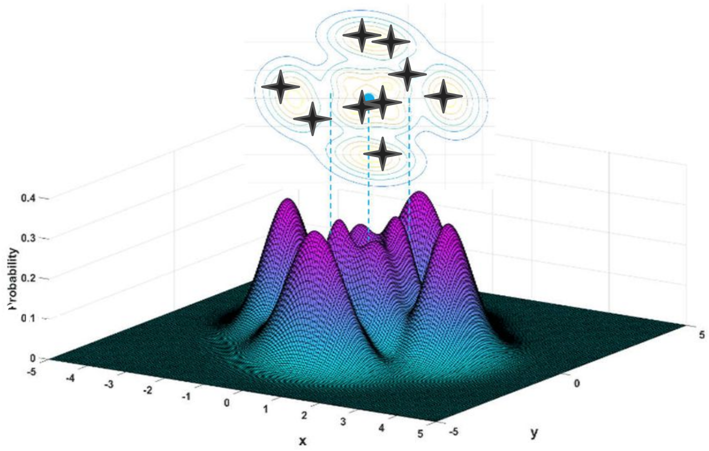
Normalizing flows

arxiv1908.09257
arxiv1912.02762

Goal: model a complex high-dimensional p.d.f

Requirements: we want to be able to:

- draw **samples** from the p.d.f



Strategy:

- Model the p.d.f as a series of bijective transformation from a base distribution
- Expressiveness: parametrize the transformations with neural networks

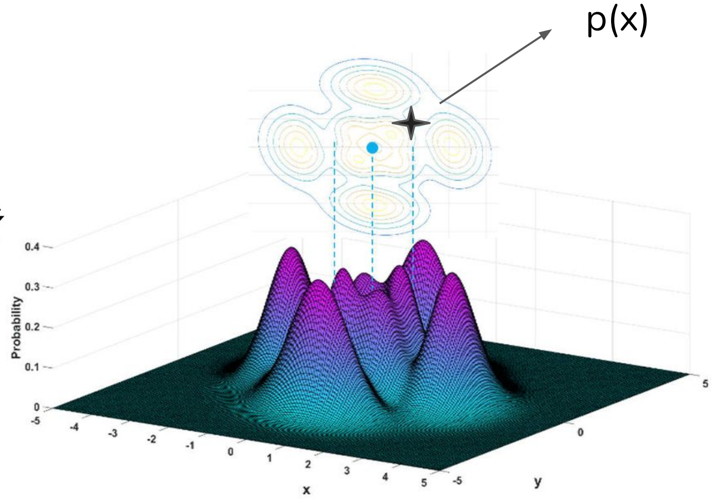
Normalizing flows

arxiv1908.09257
arxiv1912.02762

Goal: model a complex high-dimensional p.d.f

Requirements: we want to be able to:

- draw **samples** from the p.d.f
- get the **probability density** at a particular point $\vec{X}$



Strategy:

- Model the p.d.f as a series of bijective transformation from a base distribution
- Expressiveness: parametrize the transformations with neural networks

arxiv1908.09257
arxiv1912.02762

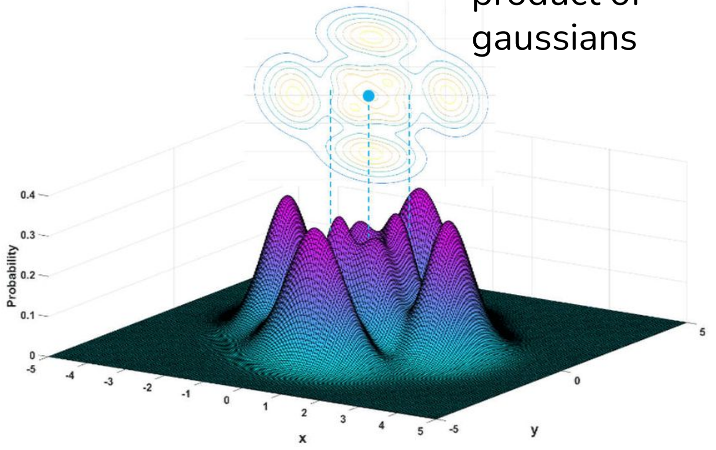
Normalizing flows

Goal: model a complex high-dimensional p.d.f

Requirements: we want to be able to:

- draw **samples** from the p.d.f
- get the **probability density** at a particular point $\vec{X}$
- model highly non-gaussian, multi-modal distributions

not factorizable as product of gaussians



Strategy:

- Model the p.d.f as a series of bijective transformation from a base distribution
- Expressiveness: parametrize the transformations with neural networks

Normalizing flows

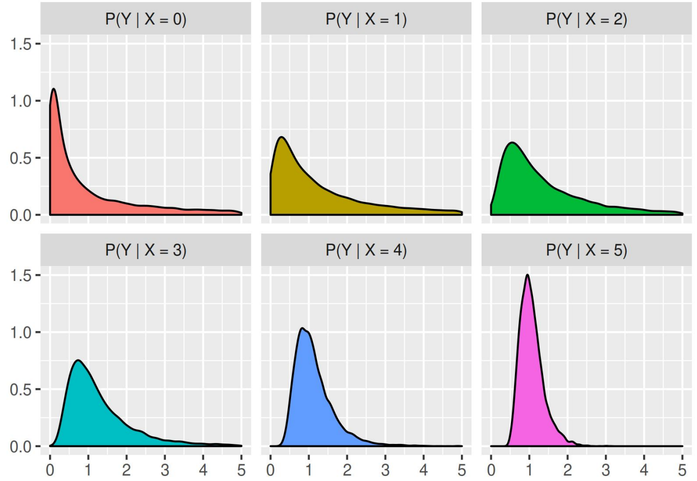
arxiv1908.09257
arxiv1912.02762

Goal: model a complex high-dimensional p.d.f

Requirements: we want to be able to:

- draw **samples** from the p.d.f
- get the **probability density** at a particular point $\vec{X}$
- model highly non-gaussian, multi-modal distributions
- create conditional p.d.f *p(x | y)*

family of (marginal) p.d.f. depending on a condition



Strategy:

- Model the p.d.f as a series of bijective transformation from a base distribution
- Expressiveness: parametrize the transformations with neural networks

Normalizing flows

Goal: model a complex high-dimensional p.d.f

Requirements: we want to be able to:

- draw **samples** from the p.d.f
- get the **probability density** at a particular point $\vec{X}$
- model highly non-gaussian, multi-modal distributions
- create conditional p.d.f *p(x | y)*
- computationally efficient

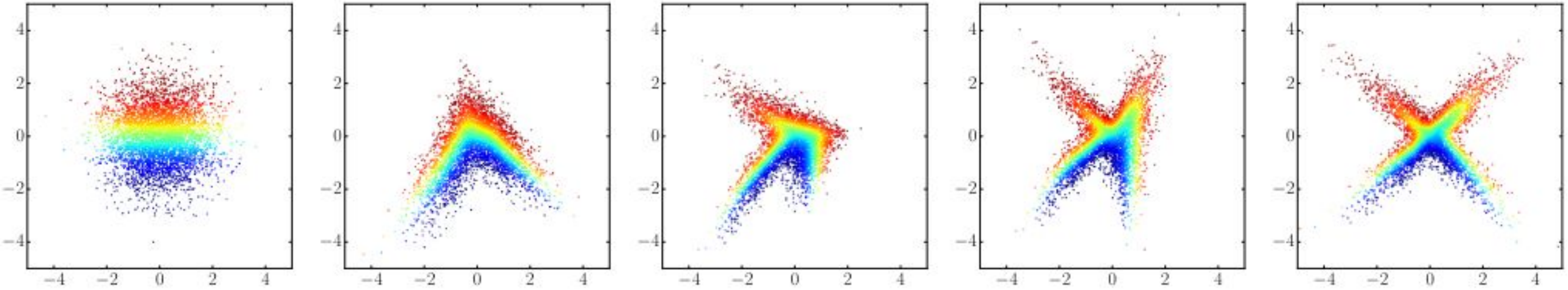
Accelerate it on GPU!
~10k samples in ~ms

Strategy:

- Model the p.d.f as a series of bijective transformation from a base distribution
- Expressiveness: parametrize the transformations with neural networks

Normalizing Flow

Normalizing direction → density estimation



Sampling direction

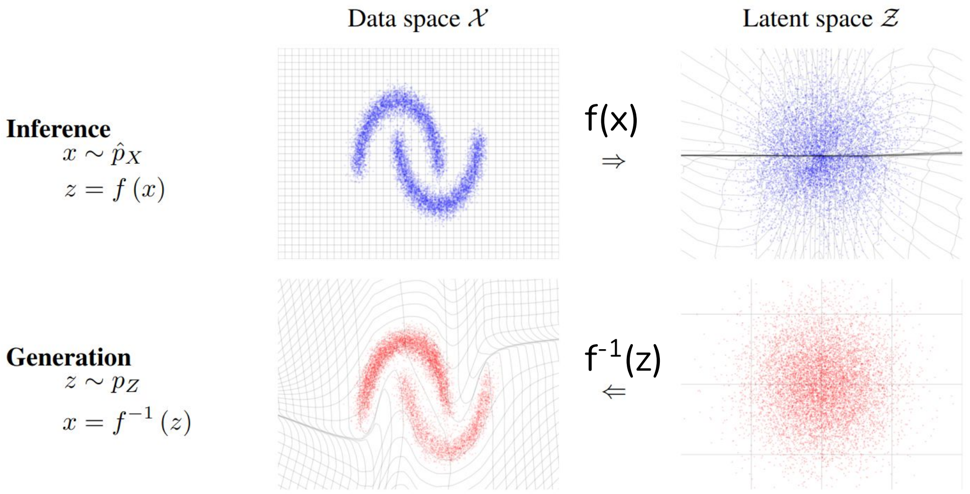
Normalizing flows : More formally

From the rules of change of integration variables

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

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

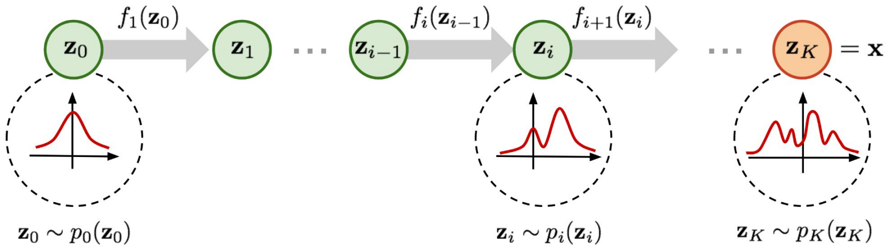
where f(x) goes in the “normalizing” direction to the z latent space.



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

- If the p.d.f in the **latent 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!

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

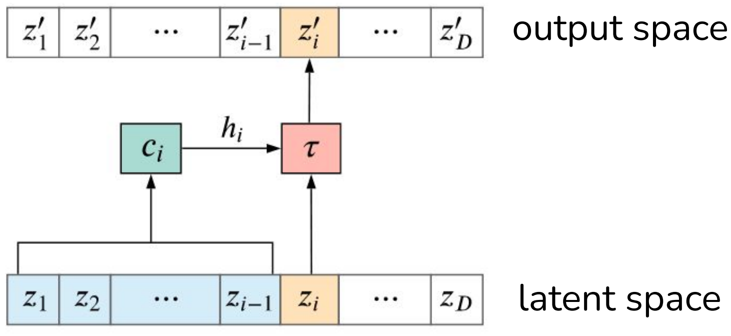
How to build a flow

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

DNNs are not invertible: use DNN as **conditioners** $\boxed{c_i}$

which parametrize invertible **transformations** $\boxed{\tau}$
for which we have analytical inversion

Choose a structure with an efficient jacobian.



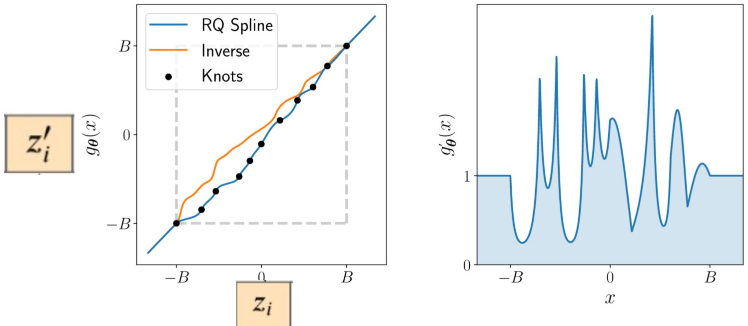
(a) Forward

The **transformation** $\boxed{\tau}$ can be:

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

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

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



[arxiv1906.04032](https://arxiv.org/abs/1906.04032)
[arxiv1912.02762](https://arxiv.org/abs/1912.02762)
[arxiv1705.07057](https://arxiv.org/abs/1705.07057)

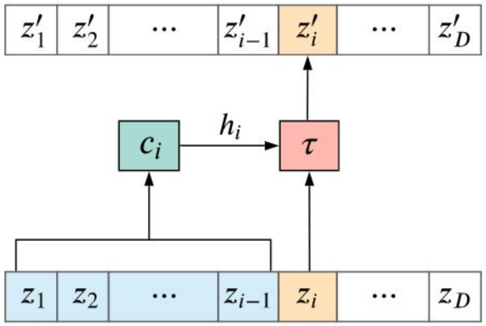
arxiv1912.02762

Conditioners

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

$$J_{f_\phi}(\mathbf{z}) = \begin{bmatrix} \frac{\partial \tau}{\partial z_1}(z_1; \boldsymbol{h}_1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{L}(\mathbf{z}) & & \frac{\partial \tau}{\partial z_D}(z_D; \boldsymbol{h}_D) \end{bmatrix}.$$

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

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

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

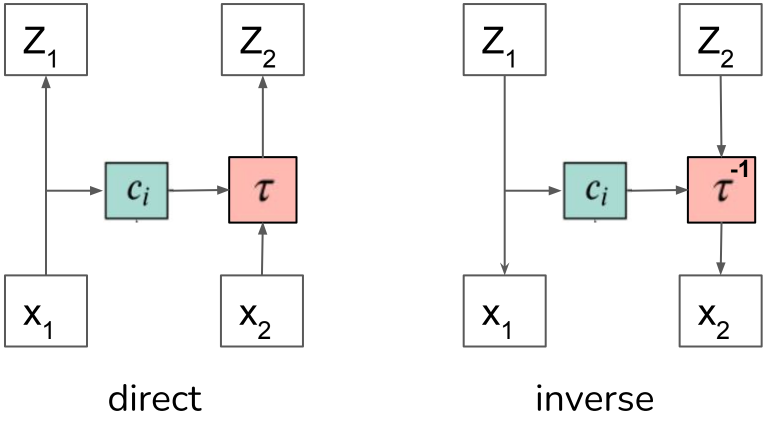
Pros:

- 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)

Coupling layers



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

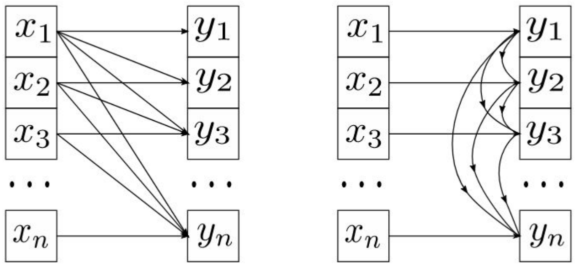
Pro:

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

Cons:

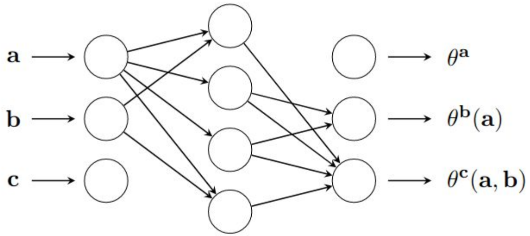
- Parallel in one direction, D steps in the version (D = dimension of the input space)
- Need to choose the direction of the implementation if we need faster sampling (IAF arxiv1606.04934) or faster density estimation (MAF arxiv1705.07057)

Autoregressive



direct inverse

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:

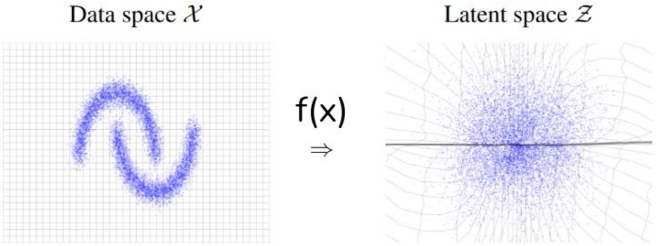
1. easy to **sample,** difficult to **evaluate**: p.d.f. of MC or Data in a control region → we have events
2. difficult to **sample**, easy to **evaluate**: multidimensional integrand → we have the function

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

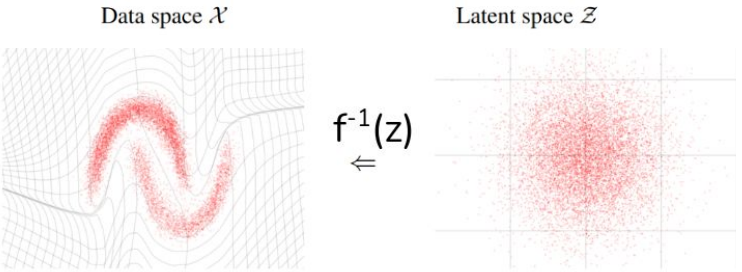


2. Training by **sampling**

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



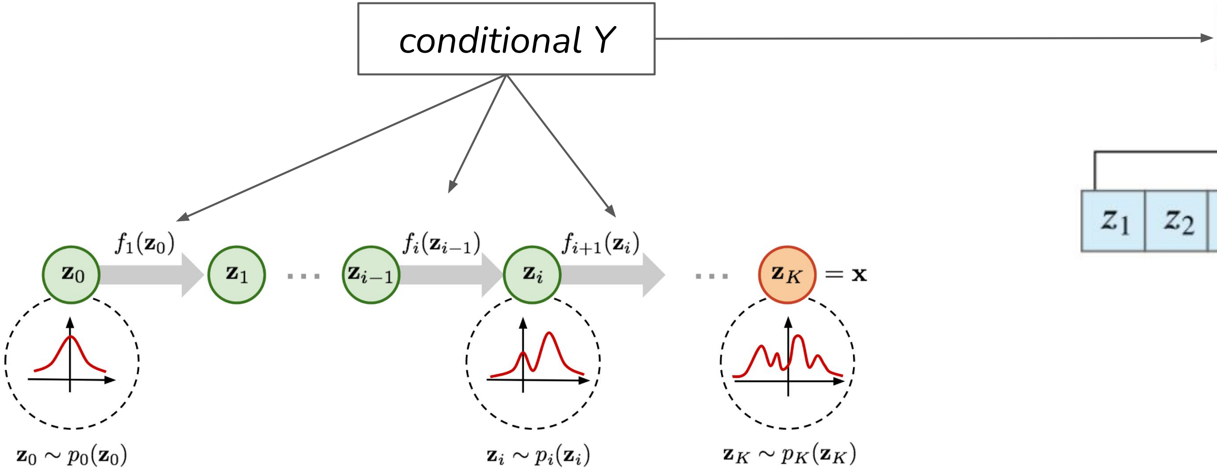
Flows conditioning

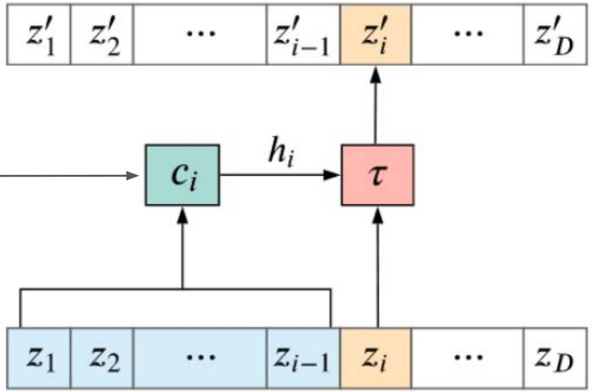
A flow can be conditioned by external information to model p(x | y):

- include the dependence in the conditioner DNNs
- N.B. the conditioning dimensions ***y* are not part** of the flow

Example:

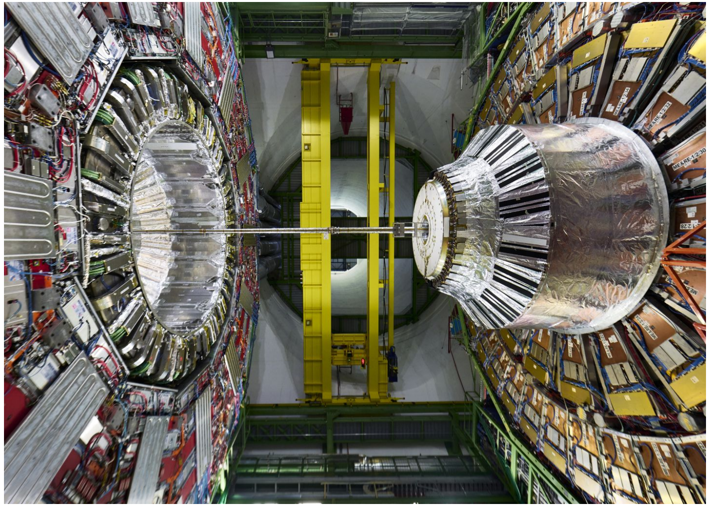
- Model parton distribution given jet observable

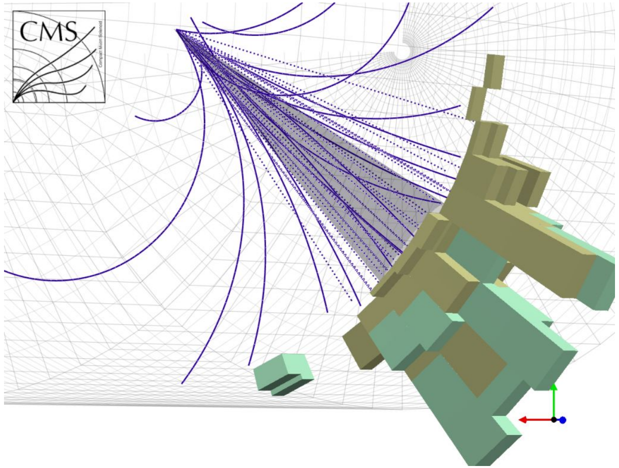




Applications in HEP

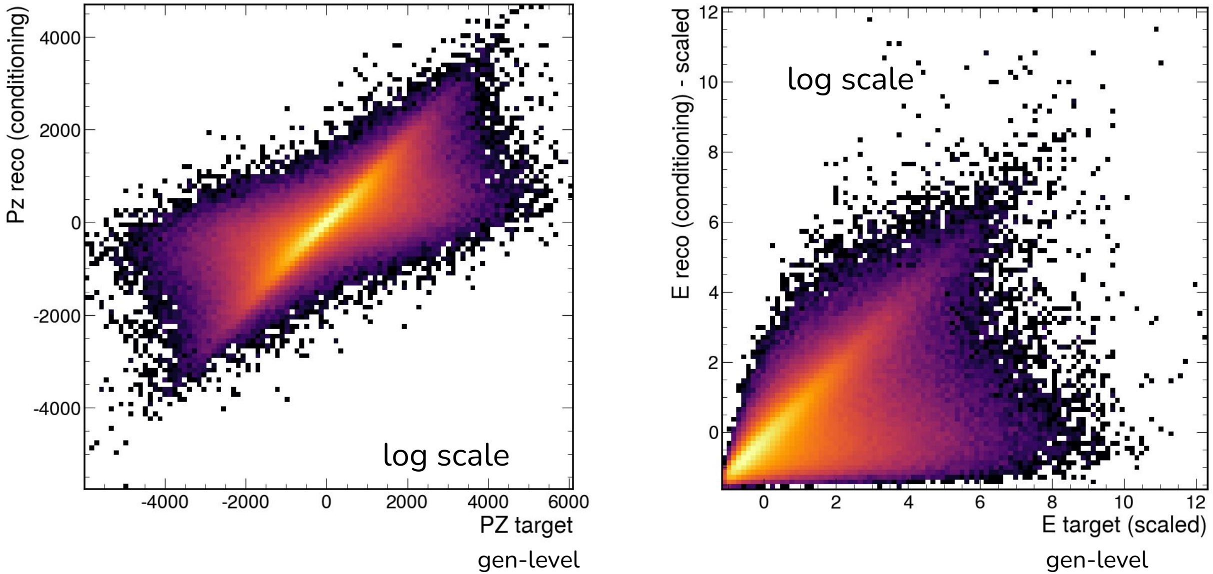
- Simple example: initial gluon momenta from reconstructed objects
- Importance sampling (MC integration, MCMC processes,
- Conditional unfolding
- Matrix Element methods
- Data/MC morphing/reweighting → see first talk tomorrow from Massimiliano





Example: gluon momenta from reco-level boost

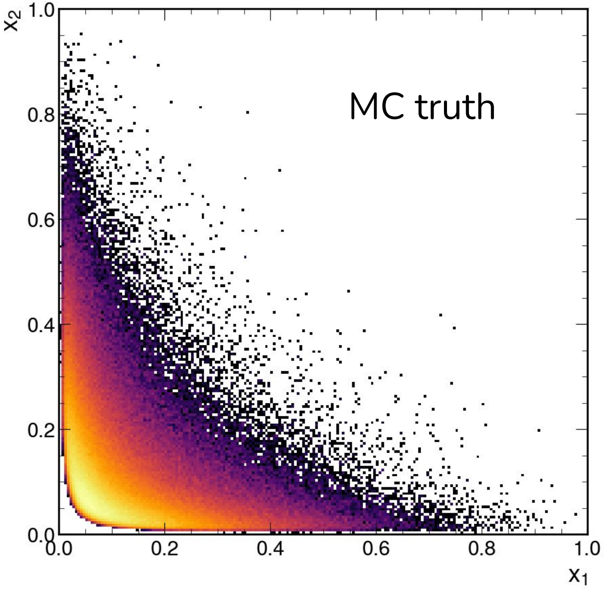
- Get the **initial gluons** from the **final state total boost**
- Easy task given the good pileup rejection of the CMS reconstruction
  - Strong correlation between the conditioning variables (reconstruction level boost) and the target variables (incoming gluon momenta)

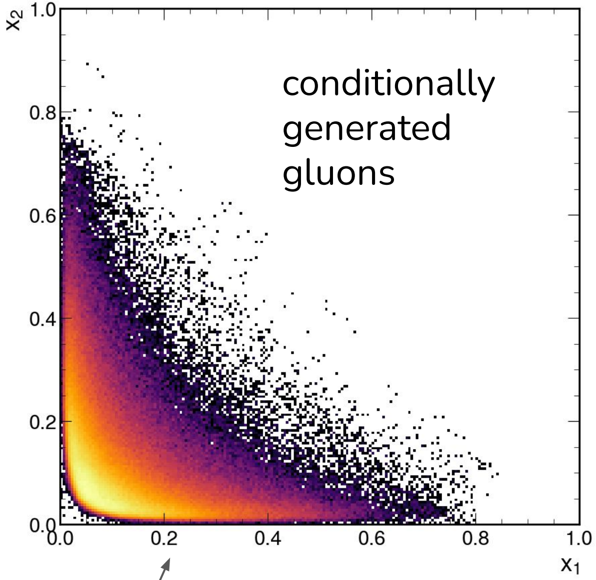


Example: gluon momenta from reco-level boost

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

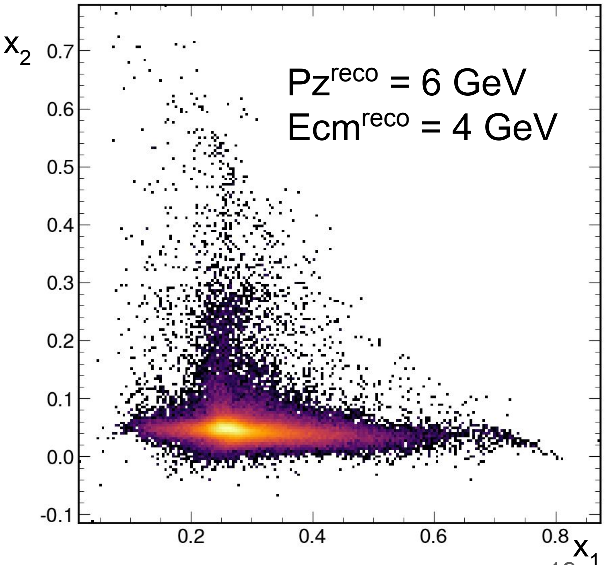
- Train it using the gluon and reco level boost from MC by maximum likelihood





generated one point for each MC events

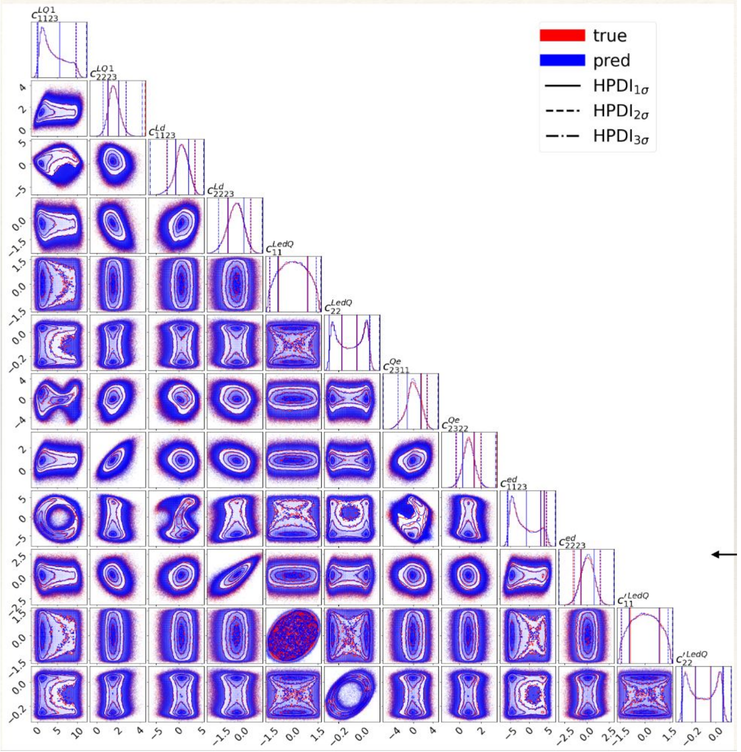
we get conditional generation!



Publish the LHC likelihood function

Talk at IML

EFT parameters likelihood from arxiv.1903.09632



- Goal: publishing likelihoods as full functions efficiently.
- Models POIs and nuisance parameters

N_{train}	Flow	N bij	N knots	Range	Hidden layers	L1 factor	N epochs	N iters.
$\mathbf{10^5}$ $\mathbf{5 \cdot 10^5}$ $\mathbf{10^6}$	A-RQS	2	16	[-6,6]	1024×3	10^{-4}	1200	12

Theorist can use the flow to generate toys with complete uncertainty description

Importance Sampling

arxiv2001.05486

arxiv2011.13445

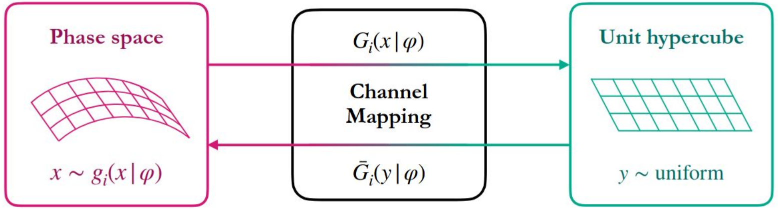
Flows for integration by importance sampling are gaining a lot of momentum in the theory community:

- general algorithm described as *i-flow* arxiv2001.05486

Large interest to **optimize the phase-sampling for cross-section** calculations

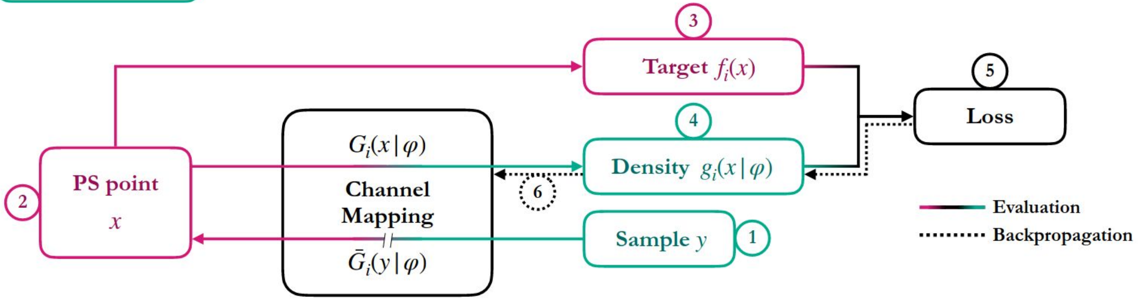
Very recent nice paper about multi-channel integration via normalizing flows to be integrated with MadGraph:

- MadNIS – Neural Multi-Channel Importance Sampling **arxiv2212.06172**



The integrand function is approximated by normalizing flows, (for different integration channels like in Madgraph)

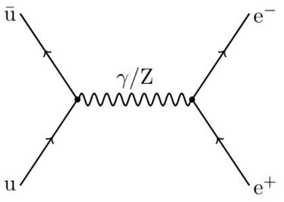
The flow is optimized by sampling and evaluating the target density (hard-scattering probability)

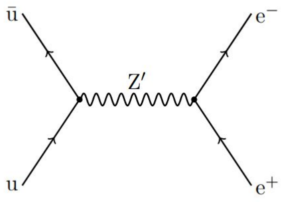


MadNis

Going to be integrated in Madgraph Generator

Parameter	Value	Parameter	Value
Loss function	variance	Coupling blocks	rational-quadratic splines
Learning rate	0.001	Permutations	exchange
LR schedule	inverse time decay	Blocks	6
Decay rate	0.01	Subnet hidden nodes	16
Batch size	10000	Subnet layers	2
Epochs	60	CWnet layers	2
Batches per epoch	50	CWnet hidden nodes	16
		Activation function	leaky ReLU





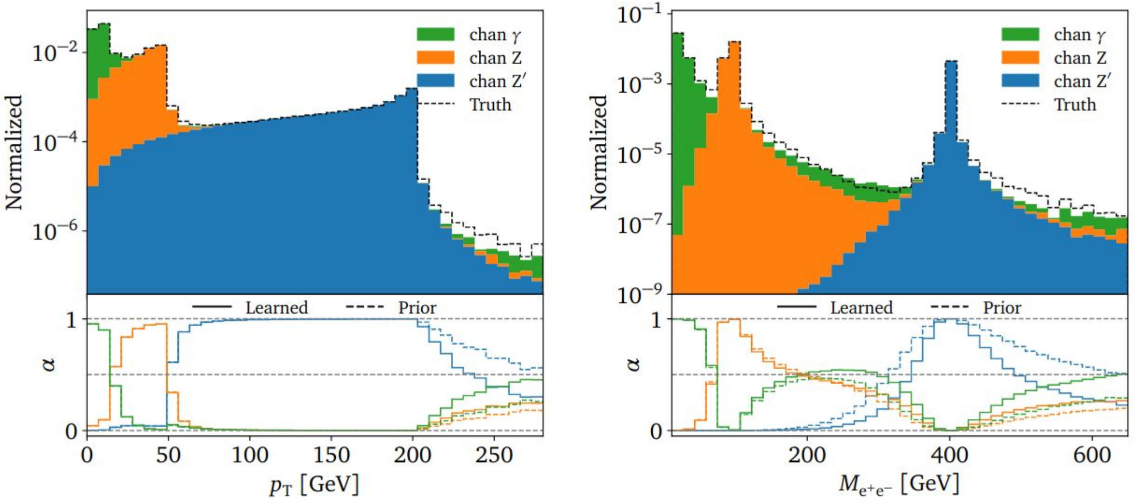


Figure 12: Learned p_T and $M_{e^+e^-}$ distributions for the Z′-extended Drell-Yan process. In the lower panels we show the learned channel weights.

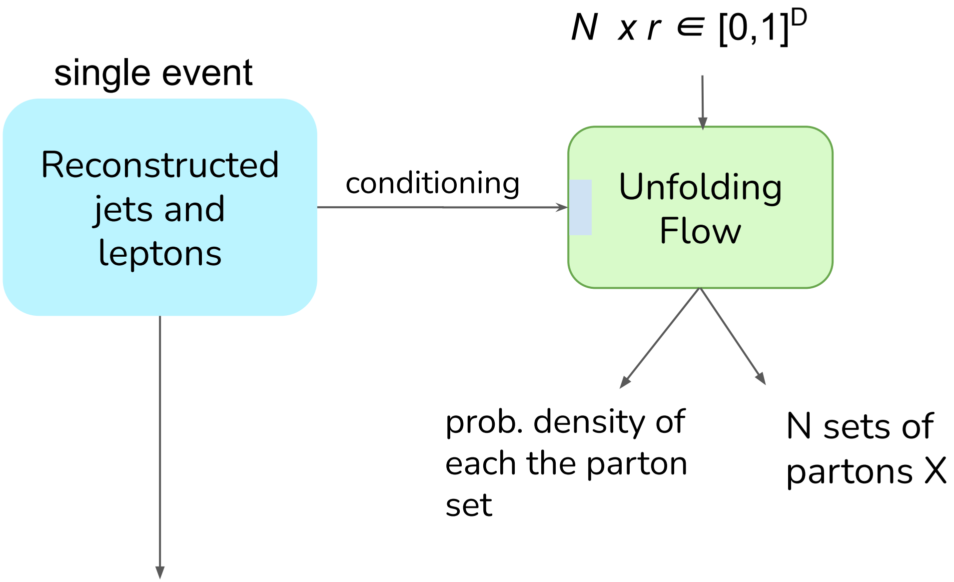
$$I[f] = \sum_i \int_{U_i} \mathrm{d}^d y \; \alpha_i(x) \, \frac{f(x)}{g_i(x|\varphi)} \bigg|_{x=\overline{G}_i(y|\varphi)}$$

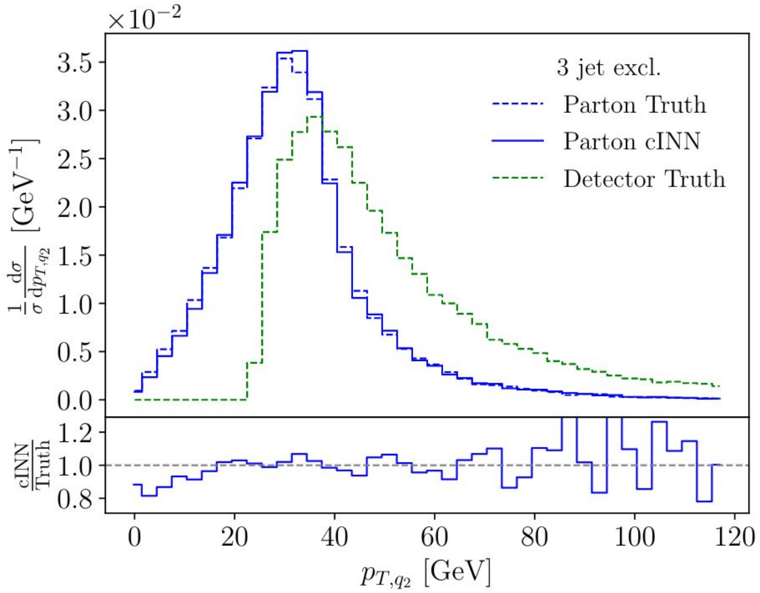
Learned a sampling distribution for each channel and also channel mapping weights (starting from Madgraph prior)

Unfolding

cINN for unfolding
arxiv2006.06685

Given an reconstructed event in the detector → distribution of possible parton-level particles

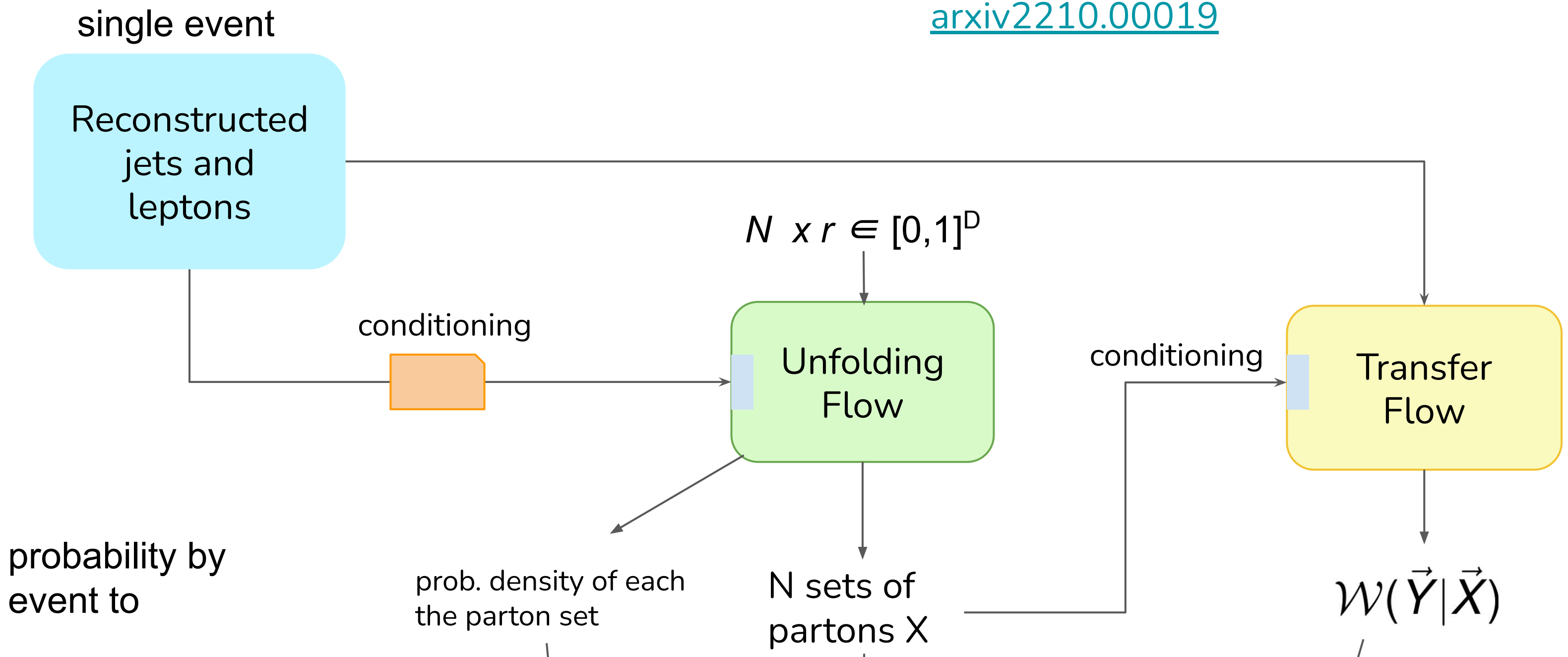




the conditioning network can be not trivial, and needs to be trained along the flow

Matrix Element Method

Compose unfolding and density estimation to compute the Matrix Element Method integral [arxiv2210.00019](https://arxiv.org/abs/2210.00019)



$$\mathcal{P}(\vec{Y}|\vec{\theta}) = \int_{\phi} d\vec{X} \cdot |\mathcal{M}(\vec{X}|\vec{\theta})|^2 \cdot Pdf \cdot \mathcal{W}(\vec{Y}|\vec{X})$$

Conclusions

- Described new ML techniques to model probability densities and their sampling: **normalizing flows**
  - how they work
  - how to train them
  - why they can be useful for HEP
- Many common points with other Particle Physics applications: let's discuss tomorrow possible contact points

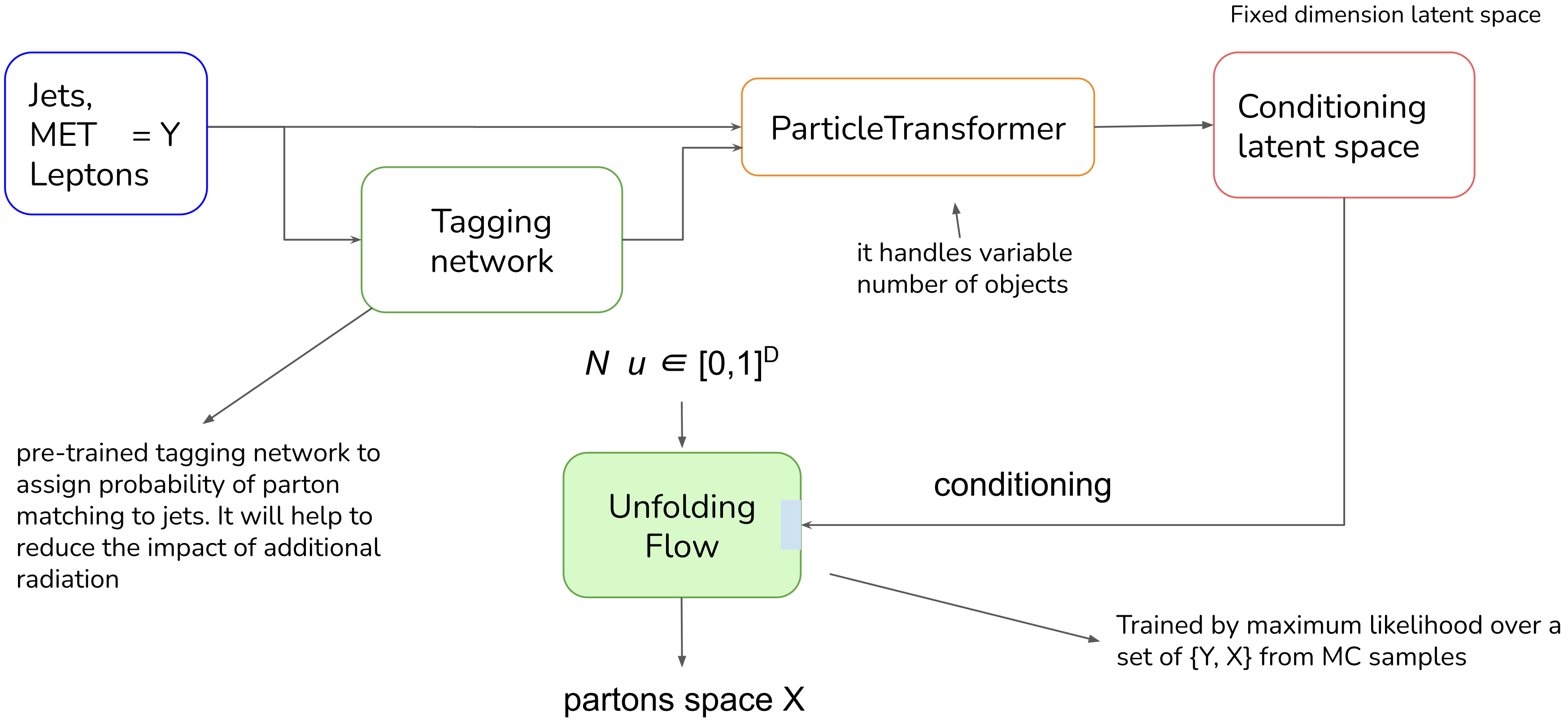
Bibliography

- Normalizing Flows for Probabilistic Modeling and Inference 1912.02762
- Normalizing Flows: An Introduction and Review of Current Methods 1908.09257
- Phase Space Sampling and Inference from Weighted Events with Autoregressive Flows 2011.13445
- i-flow: High-dimensional Integration and Sampling with Normalizing Flows 2001.05486
- MadNIS – Neural Multi-Channel Importance Sampling 2212.06172
- Matrix Element Method in HEP: Transfer Functions, Efficiencies, and Likelihood Normalization 1101.2259
- Normalizing Flows for LHC Theory link
- Masked Autoregressive Flow for Density Estimation 1705.07057
- Invertible Networks or Partons to Detector and Back Again 2006.06685
- Two Invertible Networks for the Matrix Element Method 2210.00019

Backup

Conditioning on reco events

The unfolding flow must be properly conditioned on the reconstructed event:
the conditioning network is trained alongside the unfolding flow



Density estimation

arxiv2210.00019
arxiv2006.06685

- A similar architecture can be used to model the transfer function.
- We need a different flow for **each jet multiplicity**.
- It can be modeled with Bayesian network to return a **probability + uncertainty**
    - It would be an additional nuisance for the analysis
- Again trained from MC samples of partons and reconstructed objects