Normalizing Flows

— Active Training Course "Advanced Deep Learning" —

Claudius Krause

Institute for Theoretical Physics, University of Heidelberg

November 30, 2022

Universität Heidelberg
Zukunft. Seit 1386.

Claudius.Krause@thphys.uni-heidelberg.de

Further Ressources

If you have questions, please interrupt me and ask!

This lecture is based on:

⇒ "Modern Machine Learning for LHC Physicists",
SS2022 lecture notes of Heidelberg University, arXiv: 2211.01421

⇒ "Physics 694, Advanced Topics in HEP",
Spring 2021 lecture notes of Rutgers University,
`https://www.physics.rutgers.edu/~dshih/694/`

Further Reading:

- "HEPML - Living Review"
`https://iml-wg.github.io/HEPML-LivingReview/`
- "Normalizing Flows for Probabilistic Modeling and Inference"
arXiv: 1912.02762

Motivation: Density Estimation

Problem:

Learn the underlying pdf from which a set of iid samples was drawn.

given: $\{x_i\}$ want: $p(x)$

Motivation: Density Estimation

probability density function.
$p(x) \geq 0, \ \int dx \, p(x) = 1$

Problem:

Learn the underlying pdf from which a set of iid samples was drawn.

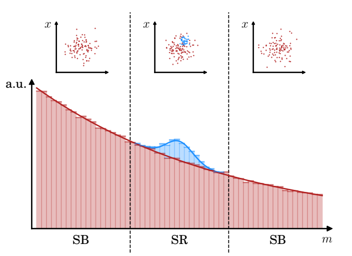
independent, identically distributed

Motivation: Density Estimation

Problem:

Learn the underlying pdf from which a set of iid samples was drawn.

given: $\{x_i\}$ want: $p(x)$



- Important for statistical data analysis (likelihoods, expectation values, ...).
- histograms, kernel density estimation, Gaussian mixture models, etc. suffer from the Curse of Dimensionality!

Figure from Hallin et al. [arXiv:2109.00546, PRD]

Motivation: Generative Models

Problem:

We have a distribution $p(x)$ and want to sample ("generate") new elements that follow it.

given: $\{x_i\}$ — want: $x \sim p(x)$

- or -

given: $f(x)$ — want: $x \sim f(x)/\int f(x)$

Motivation: Generative Models

Problem:

We have a distribution $p(x)$ and want to sample ("generate") new elements that follow it.

given: $\{x_i\}$ want: $x \sim p(x)$

- or -

given: $f(x)$ want: $x \sim f(x)/\int f(x)$

- Generation is an important aspect of simulation.
- GANs, VAEs, Normalizing Flows, Diffusion Models, and their derivates have different advantages and disadvantages.

https://thispersondoesnotexist.com/, based on T. Karras et al. [1912.04958]

Normalizing Flows in a Nutshell

Normalizing Flows learn a coordinate transformation.

Function / Distribution in $z \in \mathcal{R}^d$ $\Leftrightarrow$ Normalizing Flow $\Leftrightarrow$ Function / Distribution in $x \in \mathcal{R}^d$

$$\xrightarrow{f(z)=x}$$

$$\xleftarrow{f^{-1}(x)=z}$$

Mathematically speaking, this is a bijective function.

Normalizing Flows in a Nutshell

Normalizing Flows learn a coordinate transformation.

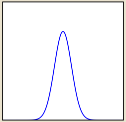
Function / Distribution in $z \in \mathcal{R}^d$ $\Leftrightarrow$ Normalizing Flow $\Leftrightarrow$ Function / Distribution in $x \in \mathcal{R}^d$

$$\xrightarrow{f(z)=x}$$

$$\xleftarrow{f^{-1}(x)=z}$$

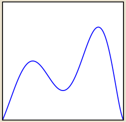
Mathematically speaking, this is a bijective function.

"easy" base distribution



$\Leftrightarrow$ bijective transformation $\Leftrightarrow$

"target" distribution



$$p(x) = \pi(f^{-1}(x)) \left| \det \frac{\partial f^{-1}(x)}{\partial x} \right|$$

$\Leftarrow$ density estimation, $p(x)$ $\Rightarrow$ sample generation

$\Leftrightarrow$ invertible function

Training Normalizing Flows

Maximum Likelihood Estimation gives the best loss functions:

- Regression: Mean Squared Error Loss
- Binary classification: Binary Cross Entropy Loss
- ...

Normalizing Flows give us the log-likelihood (LL) explicitly!

$\Rightarrow$ Maximize $\log q$ (the LL) over the given samples.
$\mathcal{L} = -\sum_i \log q(x_i)$

$\Rightarrow$ If we don't have samples, but a target $f(x)$, we can use the KL-divergence.
$\mathcal{L} = D_{KL}[f, q] = \int dx\, f(x)\, \log \frac{f(x)}{q(x)} = \left\langle \frac{f(x)}{q(x)} \log \frac{f(x)}{q(x)} \right\rangle_{x \sim q(x)}$

Normalizing Flows are great!

Pros and Cons of Normalizing Flows:

+ LL optimaztion is more stable than saddlepoint optimization of GANs.
+ Do not suffer from mode-collapse.
+ Model selection is straightforward with LL(val-set).
+ Flows are versatile (train for one thing, use for another).
+ Empirically: better at learning distributions to the %-level

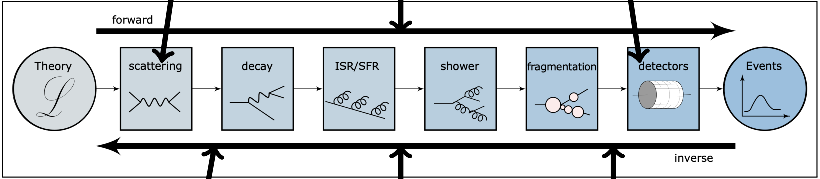
– They scale bad with the dimensionality of the problem.
– Some architectures might be slow.
– There are topological constraints.
– Sparse data is hard to learn.

Applications of Normalizing Flows: Overview

Phase Space Sampling
$\Rightarrow$ sample according to $d\sigma$

Detector Simulation
$\Rightarrow$ sample from $p(\text{showers}|E)$

End-to-End Simulation
$\Rightarrow$ sample from $p(\text{events})$



Bump-Hunt Searches
$\Rightarrow$ use $p(\text{data})$ as bg estimate

Inference
$\Rightarrow$ learn $p(\text{parameters}|\text{data})$

Unfolding
$\Rightarrow$ learn $p(\text{parton}|\mathit{event})$

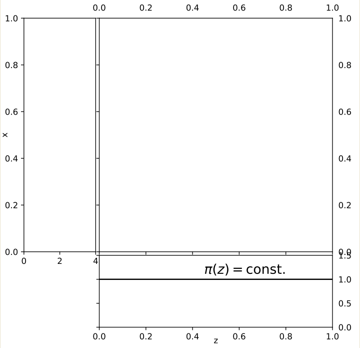
Lattice QCD
$\Rightarrow$ improve MCMC proposals

Astrophysics
$\Rightarrow$ stellar densities

At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

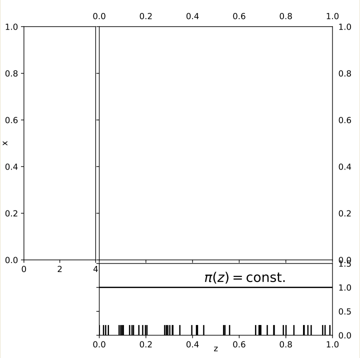
$$\bar{\pi}(\vec{x}) = \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} \;=\; f(\vec{z})$ changes the distribution according to

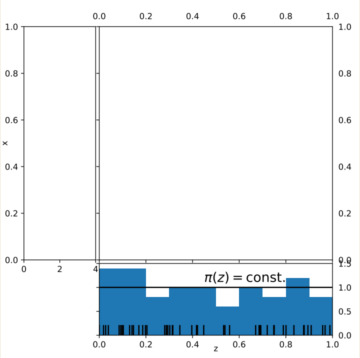
$$\bar{\pi}(\vec{x}) \;=\; \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

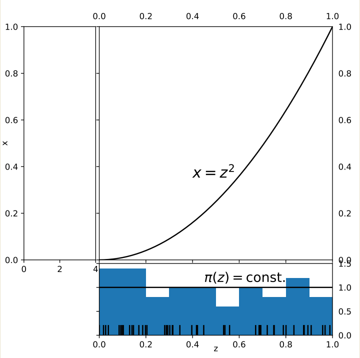
$$\bar{\pi}(\vec{x}) = \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

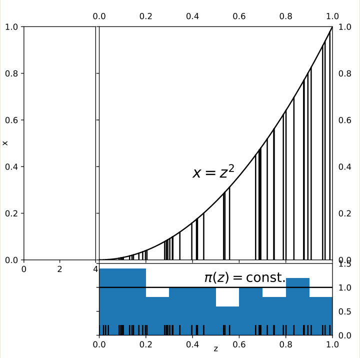
$$\bar{\pi}(\vec{x}) = \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

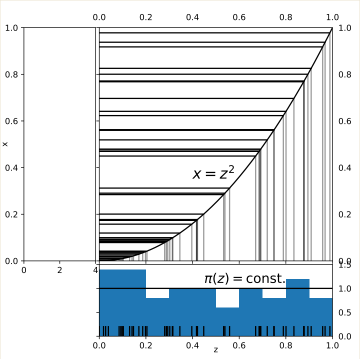
$$\bar{\pi}(\vec{x}) = \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

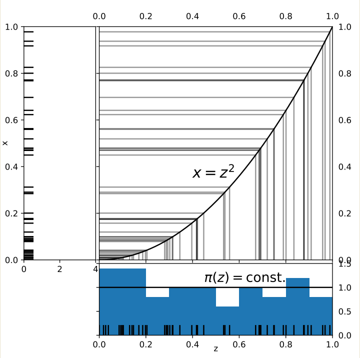
$$\bar{\pi}(\vec{x}) = \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

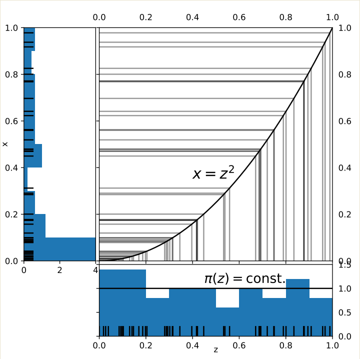
$$\bar{\pi}(\vec{x}) = \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

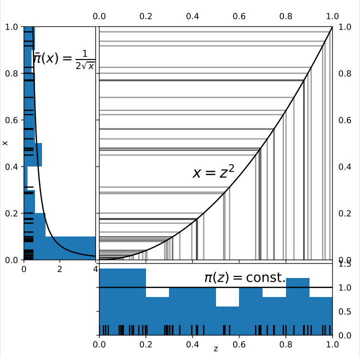
$$\bar{\pi}(\vec{x}) = \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



At the Core: Change of Coordinates Formula

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} \;=\; f(\vec{z})$ changes the distribution according to

$$\bar{\pi}(\vec{x}) \;=\; \pi(\vec{z}) \left|\det \frac{\partial f(\vec{z})}{\partial \vec{z}}\right|^{-1} = \pi(f^{-1}(\vec{x})) \left|\det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}}\right|$$



Base distributions

$$\bar{\pi}(\vec{x}) \quad = \quad \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} \quad = \quad \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$

- Can be any distribution with only 2 requirements:
  - We can easily sample from it
  - We have access to $\pi(x)$
- Sets the initial domain of the coordinates.
- Most common choices:
  - uniform distribution (compact in $[a, b]$)
  - Gaussian distribution (in $\mathbb{R}$)
- Topology should match the topology of the target space.

We need a trackable Jacobian and Inverse.

$$\bar{\pi}(\vec{x}) \quad = \quad \pi(\vec{z}) \left|\det \frac{\partial f(\vec{z})}{\partial \vec{z}}\right|^{-1} = \pi(f^{-1}(\vec{x})) \left|\det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}}\right|$$

- First idea: making f a NN.
  - × inverse does not always exist
  - × Jacobian slow via autograd
  - × $\left|\det \frac{\partial f}{\partial z}\right| \propto \mathcal{O}(n_{dim}^3)$

Dinh et al. [arXiv:1410.8516], Rezende/Mohamed [arXiv:1505.05770]

We need a trackable Jacobian and Inverse.

$$\bar{\pi}(\vec{x}) \quad = \quad \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$

- First idea: making f a NN.
  - × inverse does not always exist
  - × Jacobian slow via autograd
  - × $\left|\det \frac{\partial f}{\partial z}\right| \propto \mathcal{O}(n_{dim}^3)$

⇒ Let a NN learn parameters θ of a pre-defined transformation!

- Each transformation is 1d & has an analytic Jacobian and inverse.
  $\Rightarrow \vec{f}(\vec{x}; \vec{\theta}) = (C_1(x_1; \theta_1), C_2(x_2; \theta_2), \ldots, C_n(x_n; \theta_n))^T$

Dinh et al. [arXiv:1410.8516], Rezende/Mohamed [arXiv:1505.05770]

We need a trackable Jacobian and Inverse.

$$\bar{\pi}(\vec{x}) \quad = \quad \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$

- First idea: making f a NN.
  - × inverse does not always exist
  - × Jacobian slow via autograd
  - × $\left|\det \frac{\partial f}{\partial z}\right| \propto \mathcal{O}(n_{dim}^3)$

⇒ Let a NN learn parameters θ of a pre-defined transformation!

- Each transformation is 1d & has an analytic Jacobian and inverse.
  $\Rightarrow \vec{f}(\vec{x};\vec{\theta}) = (C_1(x_1;\theta_1), C_2(x_2;\theta_2), \dots, C_n(x_n;\theta_n))^T$

- Require a triangular Jacobian for faster evaluation.
  ⇒ The parameters θ depend only on a subset of all other coordinates.

Dinh et al. [arXiv:1410.8516], Rezende/Mohamed [arXiv:1505.05770]

A chain of bijectors is also a bijector

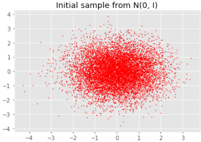
The full transformation is a chain of these bijectors.

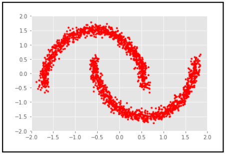
$$\xrightarrow{\pi_0(z_0)} \boxed{\begin{array}{c} z_0 = \\ f_0(z_1) \end{array}} \xrightarrow{\pi_1(z_1)} \boxed{\begin{array}{c} z_1 = \\ f_1(z_2) \end{array}} \xrightarrow{\dots} \boxed{\begin{array}{c} z_i = \\ f_i(z_{i+1}) \end{array}} \xrightarrow{\dots} \boxed{\begin{array}{c} z_k = \\ f_{k-1}(z_k) \end{array}} \xrightarrow{\pi_k(z_k)}$$

A chain of bijectors is also a bijector

The full transformation is a chain of these bijectors.

$$\xrightarrow{\pi_0(z_0)} \boxed{\begin{matrix} z_0 = \\ f_0(z_1) \end{matrix}} \xrightarrow{\pi_1(z_1)} \boxed{\begin{matrix} z_1 = \\ f_1(z_2) \end{matrix}} \xrightarrow{\dots} \boxed{\begin{matrix} z_i = \\ f_i(z_{i+1}) \end{matrix}} \xrightarrow{\dots} \boxed{\begin{matrix} z_k = \\ f_{k-1}(z_k) \end{matrix}} \xrightarrow{\pi_k(z_k)}$$



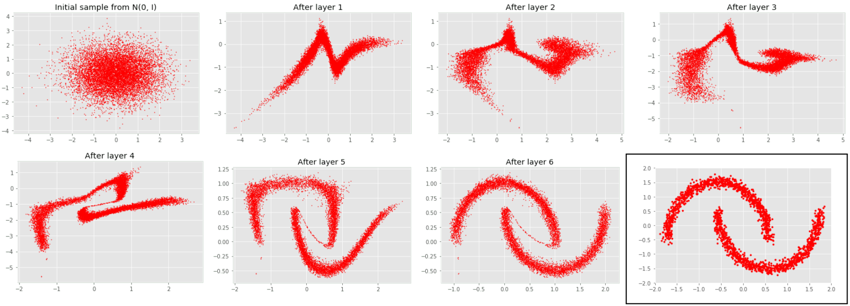


https://engineering.papercup.com/posts/normalizing-flows-part-2/

A chain of bijectors is also a bijector

The full transformation is a chain of these bijectors.

$$\xrightarrow{\pi_0(z_0)} \boxed{z_0 = f_0(z_1)} \xrightarrow{\pi_1(z_1)} \boxed{z_1 = f_1(z_2)} \longrightarrow \dots \longrightarrow \boxed{z_i = f_i(z_{i+1})} \longrightarrow \dots \longrightarrow \boxed{z_k = f_{k-1}(z_k)} \xrightarrow{\pi_k(z_k)}$$



https://engineering.papercup.com/posts/normalizing-flows-part-2/

Affine Transformations

The coupling function (transformation)

- must be invertible and expressive
- is chosen to factorize:
  $\vec{f}(\vec{x};\vec{\theta}) = (C_1(x_1;\theta_1), C_2(x_2;\theta_2), \ldots, C_n(x_n;\theta_n))^T$,
  where $\vec{x}$ are the coordinates to be transformed and $\vec{\theta}$ the parameters of the transformation.

historically first: the affine coupling function

$$C(x; s, t) = \exp(s)\, x + t$$

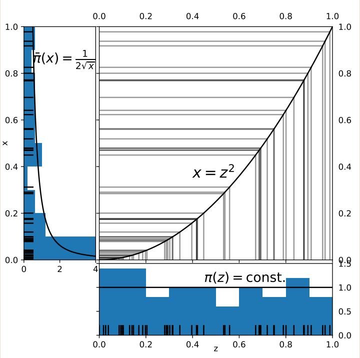
where s and t are predicted by a NN.

- It requires $x \in \mathbb{R}$.
- Inverse and Jacobian are trivial.
- Its transformation powers are limited.

Any monotonic function can be used.

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} \;=\; f(\vec{z})$ changes the distribution according to

$$\bar{\pi}(\vec{x}) \;=\; \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$

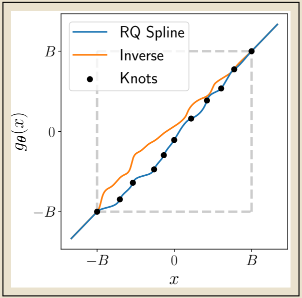


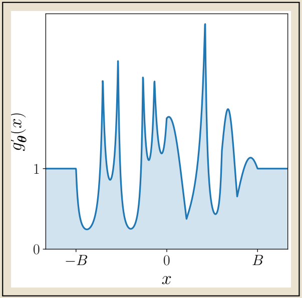
Any monotonic function can be used.

Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

$$\bar{\pi}(\vec{x}) = \pi(\vec{z}) \left| \det \frac{\partial f(\vec{z})}{\partial \vec{z}} \right|^{-1} = \pi(f^{-1}(\vec{x})) \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$

A more complicated transformation then leads to a more complicated transformed distribution. Splines act in a finite domain.

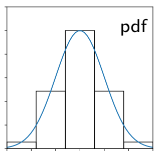


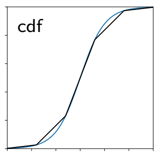


figures taken from Durkan et al. [arXiv:1906.04032]

Piecewise Transformations (Splines)

piecewise linear coupling function: Müller et al. [arXiv:1808.03856]



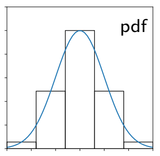


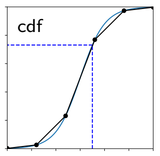
The NN predicts the pdf bin heights Q_i.

Piecewise Transformations (Splines)

piecewise linear coupling function:

Müller et al. [arXiv:1808.03856]





$$C = \sum_{k=1}^{b-1} Q_k + \alpha Q_b$$

$$\alpha = \frac{x-(b-1)w}{w}$$

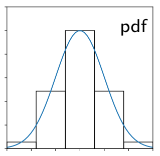
$$\left|\frac{\partial C}{\partial x_B}\right| = \prod_i \frac{Q_{b_i}}{w}$$

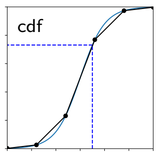
The NN predicts the pdf bin heights Q_i.

Piecewise Transformations (Splines)

piecewise linear coupling function:

Müller et al. [arXiv:1808.03856]





$$C = \sum_{k=1}^{b-1} Q_k + \alpha Q_b$$

$$\alpha = \frac{x-(b-1)w}{w}$$

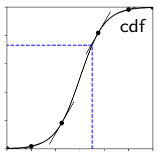
$$\left|\frac{\partial C}{\partial x_B}\right| = \prod_i \frac{Q_{b_i}}{w}$$

The NN predicts the pdf bin heights Q_i.

rational quadratic spline coupling function:

Durkan et al. [arXiv:1906.04032]

Gregory/Delbourgo [IMA Journal of Numerical Analysis, '82]



$$C = \frac{a_2\alpha^2 + a_1\alpha + a_0}{b_2\alpha^2 + b_1\alpha + b_0}$$

- still rather easy
- more flexible

The NN predicts the cdf bin widths, heights, and derivatives that go in $a_i \& b_i$.

Taming Jacobians 1: Autoregressive Models

Remember: To tame the determinants, the parameters θ must depend only on a subset of all other coordinates.

Autoregressive models solve this by $\quad \vec{\theta}_i = \vec{\theta}_i(x_{j<i})$

$$\begin{array}{c|} \vec{\theta}_1 = \text{const.} \\ \downarrow \\ p(x_1) \end{array}$$

Taming Jacobians 1: Autoregressive Models

Remember: To tame the determinants, the parameters θ must depend only on a subset of all other coordinates.

Autoregressive models solve this by $\quad \vec{\theta}_i = \vec{\theta}_i(x_{j<i})$

$$\begin{array}{c|c|}
\vec{\theta}_1 = \text{const.} & \vec{\theta}_2 = \vec{\theta}_2(z_1) \\
\downarrow & \downarrow \\
p(x_1) & p(x_2|x_1)
\end{array}$$

Taming Jacobians 1: Autoregressive Models

Remember: To tame the determinants, the parameters θ must depend only on a subset of all other coordinates.

Autoregressive models solve this by $\quad \vec{\theta}_i = \vec{\theta}_i(x_{j<i})$

$\vec{\theta}_1 = \text{const.}$	$\vec{\theta}_2 = \vec{\theta}_2(z_1)$	$\vec{\theta}_3 = \vec{\theta}_3(z_1, z_2)$	
$\downarrow$	$\downarrow$	$\downarrow$	
$p(x_1)$	$p(x_2 \vert x_1)$	$p(x_3 \vert x_1, x_2)$	

Taming Jacobians 1: Autoregressive Models

Remember: To tame the determinants, the parameters θ must depend only on a subset of all other coordinates.

Autoregressive models solve this by $\quad \vec{\theta}_i = \vec{\theta}_i(x_{j<i})$

$$
\begin{array}{c|c|c|c|c}
\vec{\theta}_1 = \text{const.} & \vec{\theta}_2 = \vec{\theta}_2(z_1) & \vec{\theta}_3 = \vec{\theta}_3(z_1, z_2) & & \vec{\theta}_i = \vec{\theta}_i(z_1, \dots, z_{i-1}) \\
\downarrow & \downarrow & \downarrow & \dots & \downarrow \\
p(x_1) & p(x_2|x_1) & p(x_3|x_1, x_2) & & p(x_i|x_1, \dots, x_{i-1})
\end{array}
$$

Taming Jacobians 1: Autoregressive Models

Remember: To tame the determinants, the parameters θ must depend only on a subset of all other coordinates.

Autoregressive models solve this by $\quad \vec{\theta}_i = \vec{\theta}_i(x_{j<i})$

$$\begin{array}{c|c|c|c|c}
\vec{\theta}_1 = \text{const.} & \vec{\theta}_2 = \vec{\theta}_2(z_1) & \vec{\theta}_3 = \vec{\theta}_3(z_1, z_2) & & \vec{\theta}_i = \vec{\theta}_i(z_1, \dots, z_{i-1}) \\
\downarrow & \downarrow & \downarrow & \dots & \downarrow \\
p(x_1) & p(x_2|x_1) & p(x_3|x_1, x_2) & & p(x_i|x_1, \dots, x_{i-1})
\end{array}$$

$$\text{Jacobian :} \quad \underbrace{\left| \begin{pmatrix} \diagdown & 0 \\ & \end{pmatrix} \right|}_{\mathcal{O}(d)}$$

Taming Jacobians 1: Autoregressive Models

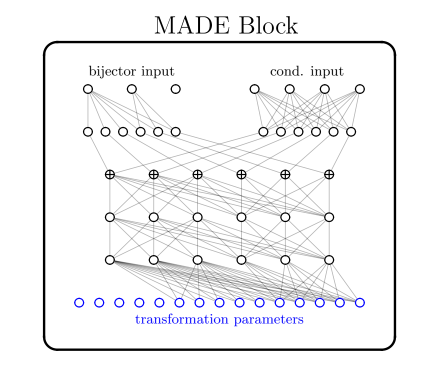
Remember: To tame the determinants, the parameters θ must depend only on a subset of all other coordinates.

Autoregressive models solve this by $\quad \vec{\theta}_i = \vec{\theta}_i(x_{j<i})$

$$
\begin{array}{c|c|c|c|c}
\vec{\theta}_1 = \text{const.} & \vec{\theta}_2 = \vec{\theta}_2(z_1) & \vec{\theta}_3 = \vec{\theta}_3(z_1, z_2) & & \vec{\theta}_i = \vec{\theta}_i(z_1, \dots, z_{i-1}) \\
\downarrow & \downarrow & \downarrow & \dots & \downarrow \\
p(x_1) & p(x_2|x_1) & p(x_3|x_1, x_2) & & p(x_i|x_1, \dots, x_{i-1})
\end{array}
$$

$$
\text{Jacobian}: \quad \underbrace{\left| \begin{pmatrix} \diagdown & 0 \end{pmatrix} \right|}_{\mathcal{O}(d)} = \prod_{i=1}^{d} p(x_i|x_1, \dots, x_{i-1}) = p(\vec{x})
$$

Autoregressive NNs: MADE Blocks



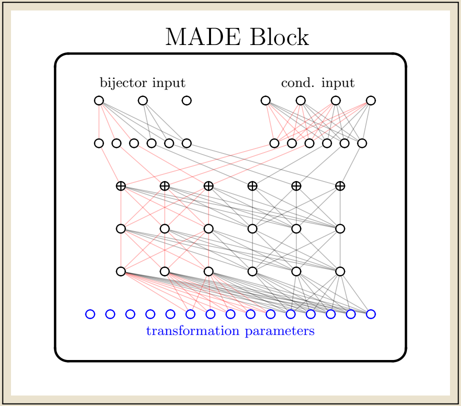
$$\boxed{\vec{\theta}_i = \vec{\theta}_i(x_1, x_2, \ldots, x_{j<i})}$$

Implementation via masking:

- a single "forward" pass gives all $\vec{\theta}_i(x_1, \ldots, x_{i-1})$.
  $\Rightarrow$ very fast
- its "inverse" needs to loop through all dimensions.
  $\Rightarrow$ very slow

Germain et al. [arXiv:1502.03509]

Autoregressive NNs: MADE Blocks



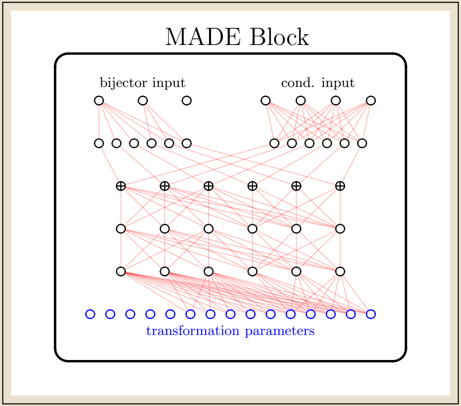
$$\boxed{\vec{\theta}_i = \vec{\theta}_i(x_1, x_2, \ldots, x_{j<i})}$$

Implementation via masking:

- a single "forward" pass gives all $\vec{\theta}_i(x_1, \ldots, x_{i-1})$.
  $\Rightarrow$ very fast
- its "inverse" needs to loop through all dimensions.
  $\Rightarrow$ very slow

Germain et al. [arXiv:1502.03509]

Autoregressive NNs: MADE Blocks



$$\boxed{\vec{\theta}_i = \vec{\theta}_i(x_1, x_2, \ldots, x_{j<i})}$$

Implementation via masking:

- a single "forward" pass gives all $\vec{\theta}_i(x_1, \ldots, x_{i-1})$.
  $\Rightarrow$ very fast
- its "inverse" needs to loop through all dimensions.
  $\Rightarrow$ very slow

Germain et al. [arXiv:1502.03509]

Autoregressive Normalizing Flows allow for 2 different realizations: MAF / IAF

Masked Autoregressive Flow (MAF)

$\Rightarrow$ slow in sampling and fast in density estimation.

- Can be trained via the log-likelihood.

  Papamakarios et al. [arXiv:1705.07057]

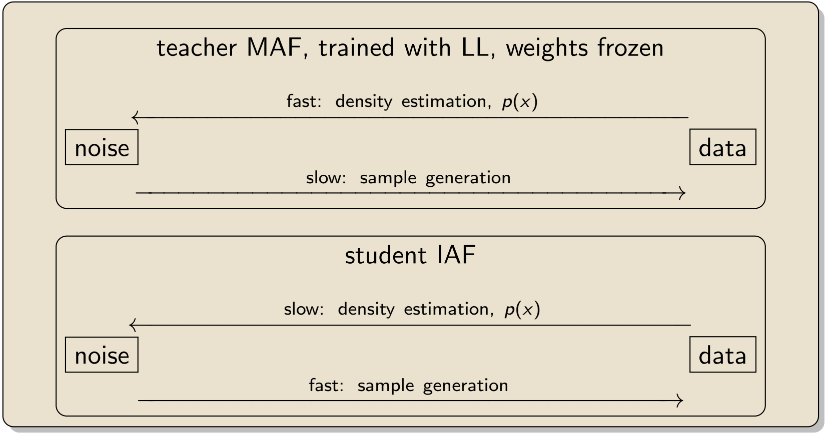
Inverse Autoregressive Flow (IAF)

$\Rightarrow$ fast in sampling and slow in density estimation.

- Log-likelihood training is usually prohibitive in memory and time.
- Instead, we can train an IAF with "Probability Density Distillation" or "teacher-student training".

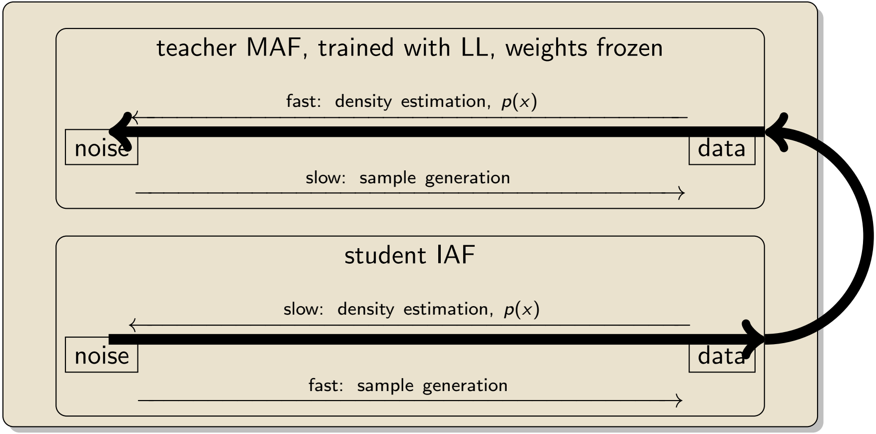
  Kingma et al. [arXiv:1606.04934]

Probability Density Distillation passes the information from the teacher to the student



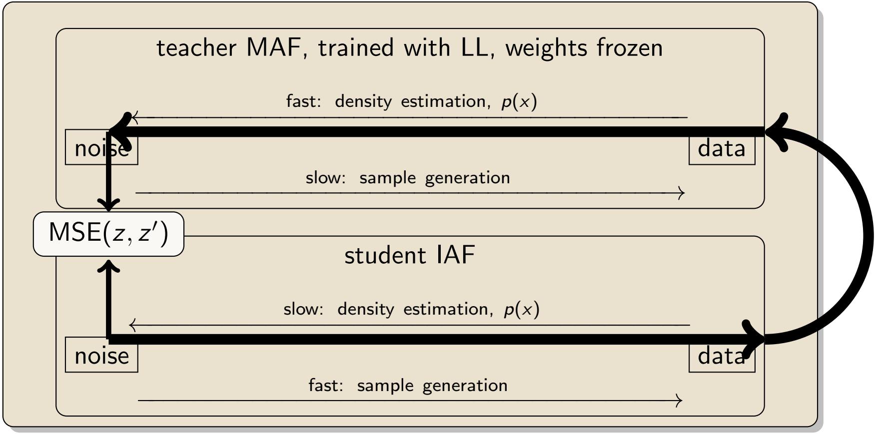
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(\theta_z, \theta_z') + \text{MSE}(\theta_x, \theta_x')$$

Probability Density Distillation passes the information from the teacher to the student



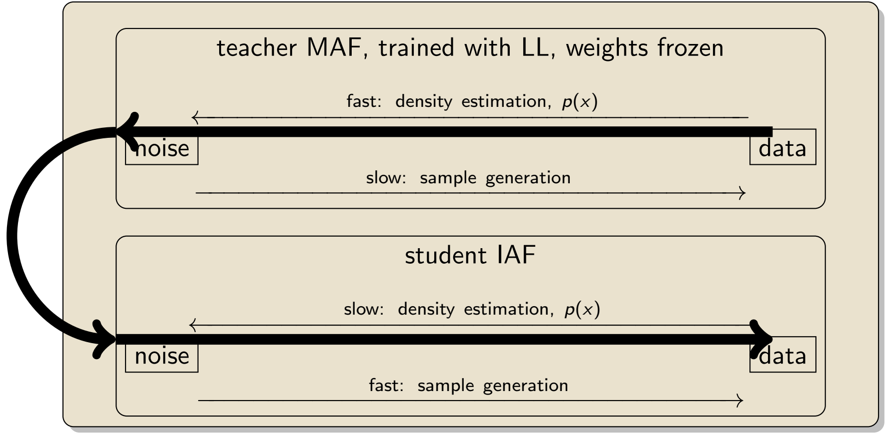
$$\text{Loss} = \color{red}{\text{MSE}(z, z')} + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(\theta_z, \theta_z') + \text{MSE}(\theta_x, \theta_x')$$

Probability Density Distillation passes the information from the teacher to the student



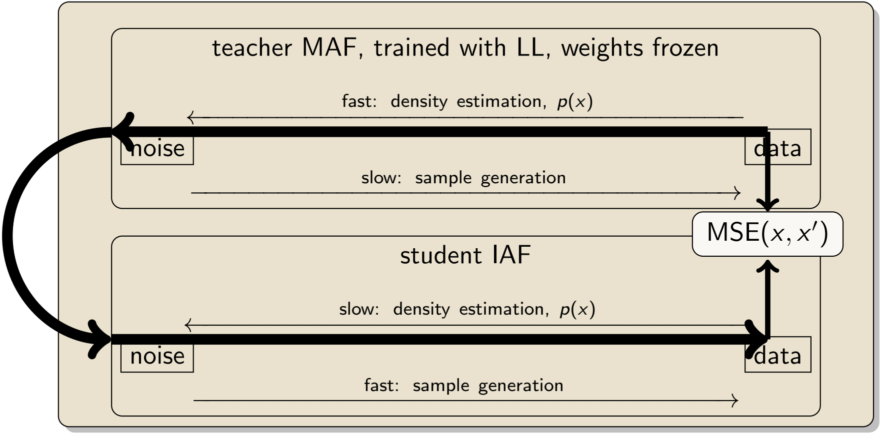
$$\text{Loss} = {\color{red}\text{MSE}(z, z')} + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(\theta_z, \theta_z') + \text{MSE}(\theta_x, \theta_x')$$

Probability Density Distillation passes the information from the teacher to the student



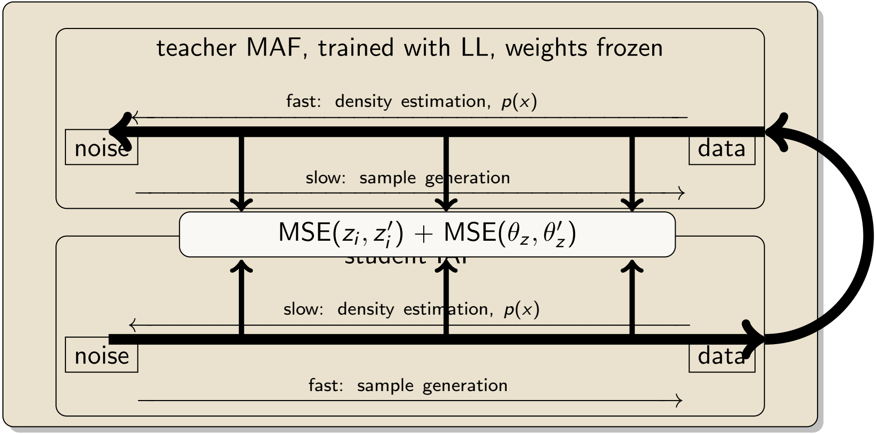
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(\theta_z, \theta_z') + \text{MSE}(\theta_x, \theta_x')$$

Probability Density Distillation passes the information from the teacher to the student



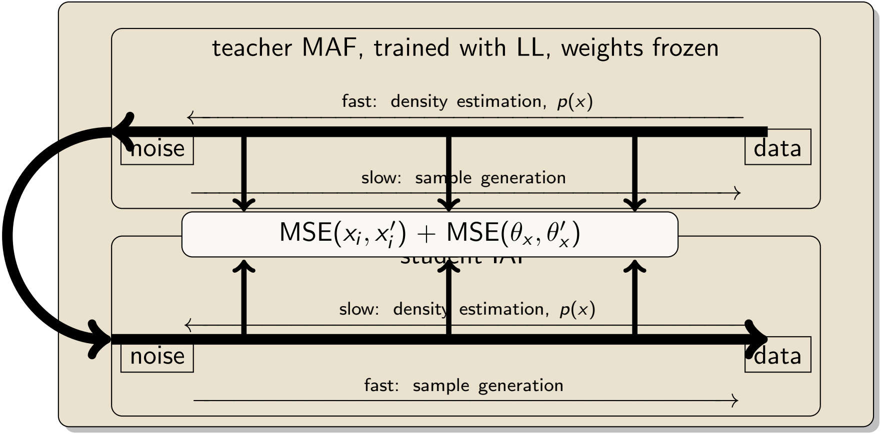
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(\theta_z, \theta_z') + \text{MSE}(\theta_x, \theta_x')$$

Probability Density Distillation passes the information from the teacher to the student



$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(\theta_z, \theta_z') + \text{MSE}(\theta_x, \theta_x')$$

Probability Density Distillation passes the information from the teacher to the student



$$\text{Loss} = \mathrm{MSE}(z, z') + \mathrm{MSE}(x, x') + \mathrm{MSE}(z_i, z_i') + \mathrm{MSE}(x_i, x_i') + \mathrm{MSE}(\theta_z, \theta_z') + \mathrm{MSE}(\theta_x, \theta_x')$$

Taming Jacobians 2: Bipartite Flows ("INNs")

$$\boxed{\theta_{x\in A}(x \in B) \qquad \& \qquad \theta_{x\in B}(x \in A)}$$

$\Rightarrow$ Coordinates are split in 2 sets, transforming each other.

forward:

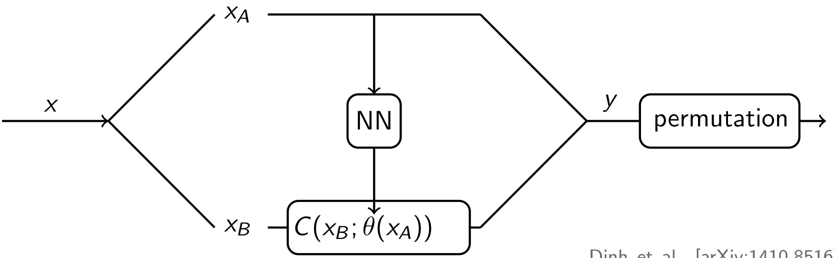
$$y_A = x_A$$
$$y_{B,i} = C(x_{B,i};\theta(x_A))$$

inverse:

$$x_A = y_A$$
$$x_{B,i} = C^{-1}(y_{B,i};\theta(x_A))$$

Jacobian:

$$\begin{vmatrix} 1 & \frac{\partial C}{\partial x_A} \\ 0 & \frac{\partial C}{\partial x_B} \end{vmatrix} = \prod_i \frac{\partial C(x_{B,i};\theta(x_A))}{\partial x_{B,i}}$$



Dinh et al. [arXiv:1410.8516]

Further improvements

Incorporating Symmetries:

- Symmetric base distribution
- Equivariant transformation: $f(g \cdot x) = g \cdot f(x)$

Kanwar et al. [arXiv:2003.06413]; Köhler et al. [arXiv:2006.02425]

Further improvements

Incorporating Symmetries:

- Symmetric base distribution
- Equivariant transformation: $f(g \cdot x) = g \cdot f(x)$

Kanwar et al. [arXiv:2003.06413]; Köhler et al. [arXiv:2006.02425]

More expressive transformations:

- Make C a monotonic NN, with θ given by another NN.

  Huang et al. [arXiv:1804.00779]

- Make C the solution of an ODE, with C' given by the NN.

  Grathwohl et al. [arXiv:1810.01367]

Further improvements

Incorporating Symmetries:

- Symmetric base distribution
- Equivariant transformation: $f(g \cdot x) = g \cdot f(x)$

Kanwar et al. [arXiv:2003.06413]; Köhler et al. [arXiv:2006.02425]

More expressive transformations:

- Make C a monotonic NN, with θ given by another NN.

  Huang et al. [arXiv:1804.00779]

- Make C the solution of an ODE, with C' given by the NN.

  Grathwohl et al. [arXiv:1810.01367]

Dimensional reduction:

- Project data to submanifold and learn on this space.

Esser et al. [arXiv:2004.13166], Brehmer/Cranmer [arXiv:2003.13913]

Further improvements II

Improving precision of sampled distributions by using classifiers:

- Train a classifier on samples vs truth.
- By the Neyman-Pearson Lemma, the output of the classifier is related to the LL ratio. $NN(x) = \frac{p_{\text{truth}}(x)}{1 - p_{\text{truth}}(x)} = \frac{p_{\text{truth}}(x)}{p_{\text{generated}}(x)} \equiv w$

Further improvements II

Improving precision of sampled distributions by using classifiers:

- Train a classifier on samples vs truth.
- By the Neyman-Pearson Lemma, the output of the classifier is related to the LL ratio. $NN(x) = \frac{p_{\text{truth}}(x)}{1-p_{\text{truth}}(x)} = \frac{p_{\text{truth}}(x)}{p_{\text{generated}}(x)} \equiv w$

1 instead of the plain samples x, we can now consider them weighted by $w(x)$ DCTRGAN: Diefenbacher et al. [arXiv:2009.03796]

$\Rightarrow$ corrects $p_{\text{generated}}(x)$ to $p_{\text{truth}}(x)$

Further improvements II

Improving precision of sampled distributions by using classifiers:

- Train a classifier on samples vs truth.
- By the Neyman-Pearson Lemma, the output of the classifier is related to the LL ratio. $NN(x) = \frac{p_{\text{truth}}(x)}{1-p_{\text{truth}}(x)} = \frac{p_{\text{truth}}(x)}{p_{\text{generated}}(x)} \equiv w$

1 instead of the plain samples x, we can now consider them weighted by $w(x)$

DCTRGAN: Diefenbacher et al. [arXiv:2009.03796]

$\Rightarrow$ corrects $p_{\text{generated}}(x)$ to $p_{\text{truth}}(x)$

2 Modify loss to $\mathcal{L} = -\sum_i \frac{1}{w(x_i)} \log q(x_i)$

$\Rightarrow$ "bad" points are more important for optimization.

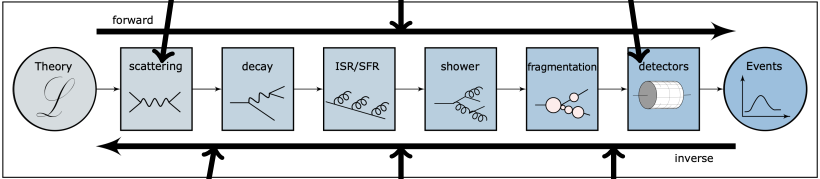
DiscFlow: Butter et al. [arXiv:2110.13632]

Applications of Normalizing Flows: Overview

Phase Space Sampling
$\Rightarrow$ sample according to $d\sigma$

Detector Simulation
$\Rightarrow$ sample from $p(\text{showers}|E)$

End-to-End Simulation
$\Rightarrow$ sample from $p(\text{events})$



Bump-Hunt Searches
$\Rightarrow$ use $p(\text{data})$ as bg estimate

Inference
$\Rightarrow$ learn $p(\text{parameters}|\text{data})$

Unfolding
$\Rightarrow$ learn $p(\text{parton}|\mathit{event})$

Lattice QCD
$\Rightarrow$ improve MCMC proposals

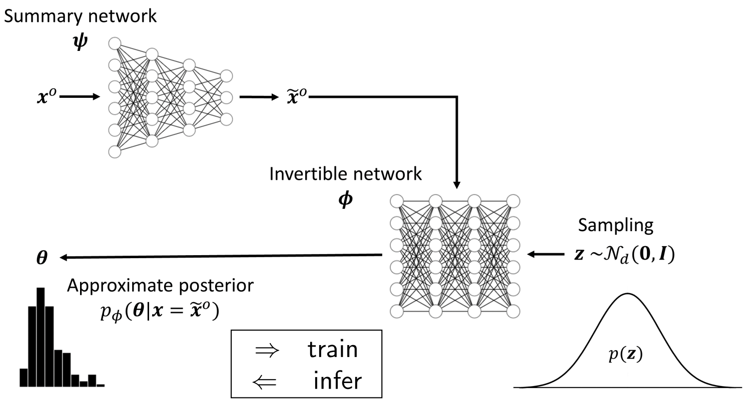
Astrophysics
$\Rightarrow$ stellar densities

Applications: Learning the true Posterior Distribution

Normalizing Flows can learn conditional probabilities.

$\Rightarrow$ use them to learn the posterior $p(\text{parameters}|\text{data})$

BayesFlow/cINN: Radev et al. [arXiv:2003.06281]



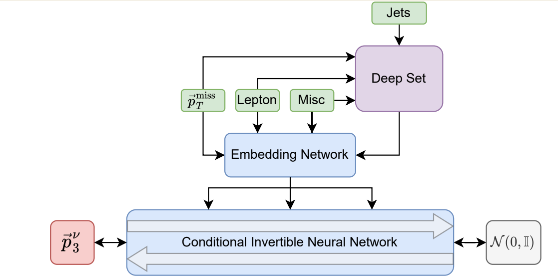
Applications: Learning the true Posterior Distribution

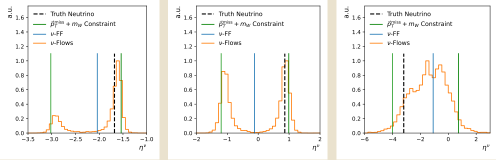
Normalizing Flows can learn conditional probabilities.

$\Rightarrow$ use them to learn the posterior $p(\text{parameters}|\text{data})$

BayesFlow/cINN: Radev et al. [arXiv:2003.06281]

ν-Flows: infering the ν-momentum in semileptonic $\bar{t}t$-events.





ν-Flows: Leigh et al. [arXiv:2207.00664]

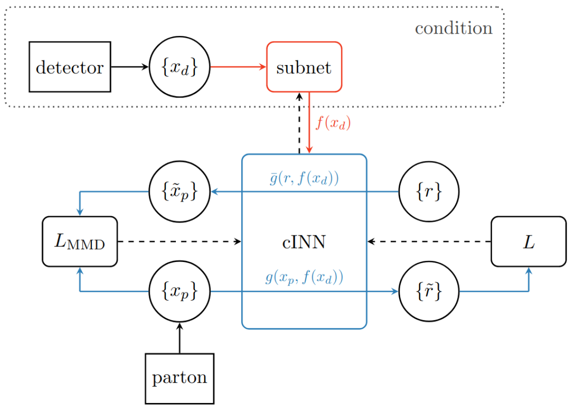
Applications: Learning the true Posterior Distribution

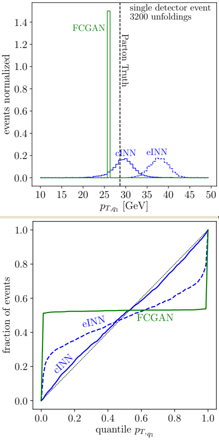
Normalizing Flows can learn conditional probabilities.

$\Rightarrow$ use them to learn the posterior $p(\text{parameters}|\text{data})$

BayesFlow/cINN: Radev et al. [arXiv:2003.06281]

Unfolding detector effects:

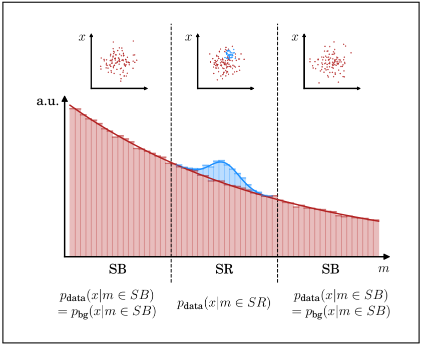




Butter et al. [arXiv:2006.06685]

Applications: Anomaly Detection (Bump Hunts)

Introducing Bump Hunts: Searches with few model assumptions



Assumptions

- signal is localized in m
- background in m is smooth
- $\exists$ additional discriminating features x

Select events with

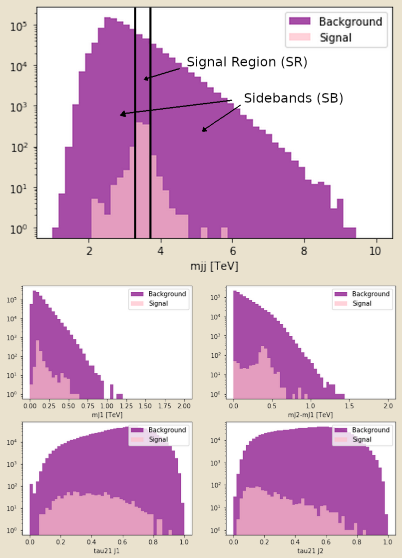
$$\Rightarrow \frac{p_{\text{data}}}{p_{\text{background}}} \sim \frac{p_{\text{signal}}}{p_{\text{background}}}$$

Applications: Anomaly Detection (Bump Hunts)

LHC Olympics R&D dataset:

- 1,000,000 QCD dijet events
- 1,000 signal events
  $W' \to X(\to qq)Y(\to qq)$
- $m_{W'} = 3.5\text{TeV}$,
  $m_X = 500\text{GeV}, \quad m_Y = 100\text{GeV}$
- In SR, $3.3\text{TeV} < m_{JJ} < 3.7\text{TeV}$:
  - 121,352 bg events
  - 772 sg events
- $S/\sqrt{B} = 2.2$

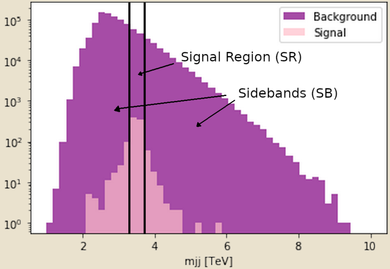
LHCO: G. Kasieczka et al. [2101.08320]



Applications: Anomaly Detection (ANODE)

Anomaly Detection with Density Estimation (ANODE):

- train "outer" density estimator $p_{\text{data}}(x|m_{JJ} \in SB)$
- train "inner" density estimator $p_{\text{data}}(x|m_{JJ} \in SR)$
- compute $\frac{p_{\text{inner}}(x|m_{JJ})}{p_{\text{outer}}(x|m_{JJ})}$ for $m_{JJ} \in SR$

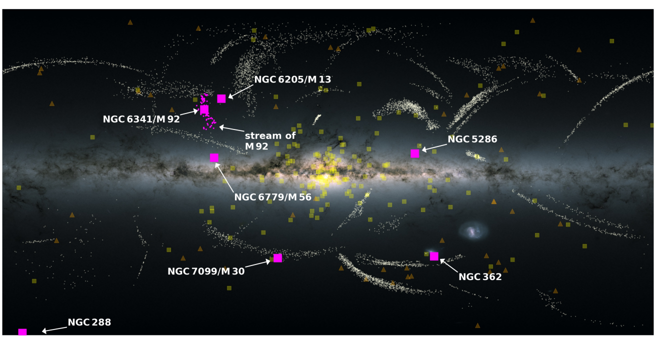


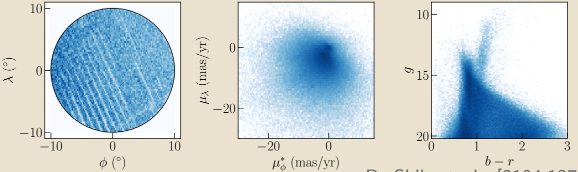
B. Nachman, D. Shih, [2001.04990, PRD]

Applications: Anomaly Detection (ANODE) on Gaia data

Looking for stellar streams:

www.esa.int



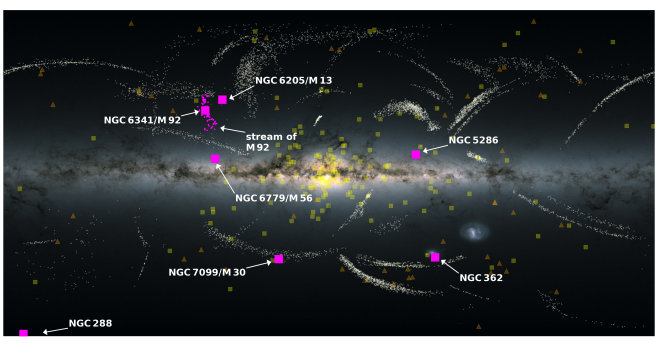


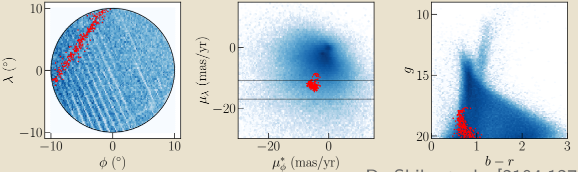
D. Shih et al. [2104.12789, MNRAS]

Applications: Anomaly Detection (ANODE) on Gaia data

Looking for stellar streams:

www.esa.int



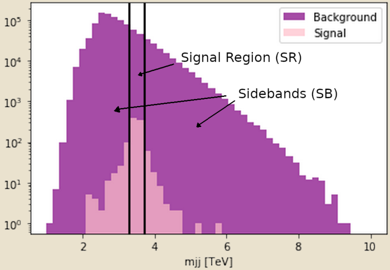


D. Shih et al. [2104.12789, MNRAS]

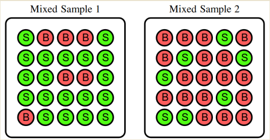
Applications: Anomaly Detection (CATHODE)

Classifying Anomalies THrough Outer Density Estimation (CATHODE):

- train "outer" density estimator $p_{\text{data}}(x|m_{JJ} \in SB)$
- sample "artificial" events from $p_{\text{outer}}(x|m_{JJ} \in SR)$
- can also oversample
- train a classifier on these samples vs data



A. Hallin, J. Isaacson, G. Kasieczka, **CK**, B. Nachman, T. Quadfasel, M. Schlaffer, D. Shih, M. Sommerhalder [2109.00546, PRD]



- Classification without Labels (CWoLa) learns from mixed samples.
- An optimal classifier is also optimal for distinguishing S from B.

E.M. Metodiev, B. Nachman, J. Thaler, [1708.02949 JHEP]

Applications: Numerical Integration with Importance Sampling

$$I = \int_0^1 f(\vec{x})\, d\vec{x} \quad \xrightarrow{\text{MC}} \quad \frac{1}{N}\sum_i f(\vec{x}_i) \qquad \vec{x}_i \ldots \text{uniform}, \quad \sigma_{\text{MC}}(I) \sim \frac{1}{\sqrt{N}}$$

$$= \int_0^1 \frac{f(\vec{x})}{q(\vec{x})}\, q(\vec{x}) d\vec{x} \quad \xrightarrow[\text{importance sampling}]{\text{MC}} \quad \frac{1}{N}\sum_i \frac{f(\vec{x}_i)}{q(\vec{x}_i)} \qquad \vec{x}_i \ldots q(\vec{x}),$$

In the limit $q(\vec{x}) \propto f(\vec{x})$, we get $\sigma_{\text{IS}}(I) = 0$

We therefore have to find a $q(\vec{x})$ that approximates the shape of $f(\vec{x})$.

$\Rightarrow$ Once found, we can use it for event generation,
i.e. sampling p_i, ϑ_i, and φ_i according to $d\sigma(p_i, \vartheta_i, \varphi_i)$

Applications: Numerical Integration with Importance Sampling

$$I = \int_0^1 f(\vec{x})\, d\vec{x} \quad \xrightarrow{\text{MC}} \quad \frac{1}{N}\sum_i f(\vec{x}_i) \qquad \vec{x}_i \ldots \text{uniform}, \quad \sigma_{\text{MC}}(I) \sim \frac{1}{\sqrt{N}}$$

$$= \int_0^1 \frac{f(\vec{x})}{q(\vec{x})}\, q(\vec{x}) d\vec{x} \quad \xrightarrow[\text{importance sampling}]{\text{MC}} \quad \frac{1}{N}\sum_i \frac{f(\vec{x}_i)}{q(\vec{x}_i)} \qquad \vec{x}_i \ldots q(\vec{x}),$$

In the limit $q(\vec{x}) \propto f(\vec{x})$, we get $\sigma_{\text{IS}}(I) = 0$

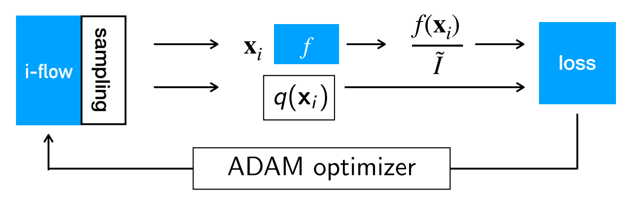
We therefore have to find a $q(\vec{x})$ that approximates the shape of $f(\vec{x})$.

$\Rightarrow$ Once found, we can use it for event generation,
i.e. sampling p_i, ϑ_i, and φ_i according to $d\sigma(p_i, \vartheta_i, \varphi_i)$

We need both samples x and their probability $q(x)$.
$\Rightarrow$ We use a bipartite, coupling-layer-based Flow.

Applications: Numerical Integration with Importance Sampling

How it works:

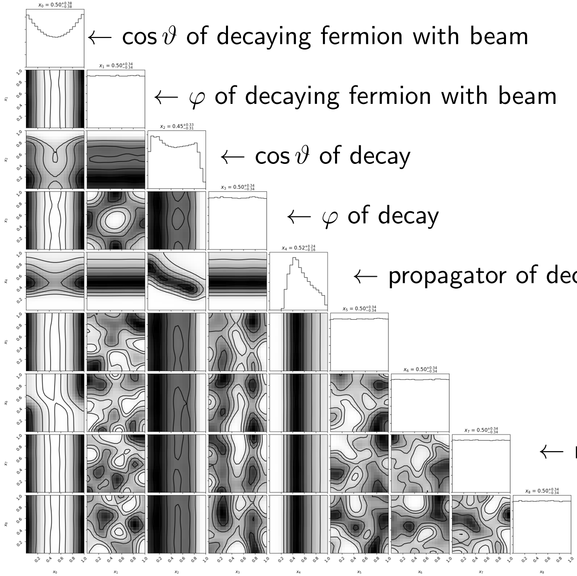


`i-flow`: C. Gao, J. Isaacson, **CK** [arXiv:2001.05486, ML:ST]
`gitlab.com/i-flow/i-flow`

Statistical Divergences are used as loss functions:

- Kullback-Leibler (KL) divergence:
  $D_{KL} = \int p(x) \log \frac{p(x)}{q(x)} dx \quad \approx \quad \frac{1}{N} \sum \frac{p(x_i)}{q(x_i)} \log \frac{p(x_i)}{q(x_i)}, \quad x_i \ldots q(x)$

Applications: $e^+e^- \to 3j$.



$\leftarrow \cos\vartheta$ of decaying fermion with beam

$\leftarrow \varphi$ of decaying fermion with beam

Target distribution

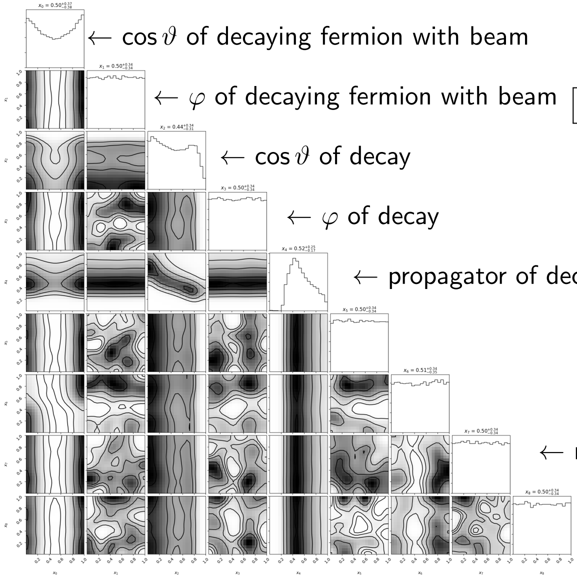
$\leftarrow \cos\vartheta$ of decay

$\leftarrow \varphi$ of decay

$\leftarrow$ propagator of decaying fermion

$\leftarrow$ multichannel

Applications: $e^+e^- \to 3j$.



$\leftarrow \cos\vartheta$ of decaying fermion with beam

$\leftarrow \varphi$ of decaying fermion with beam

Learned distribution

$\leftarrow \cos\vartheta$ of decay

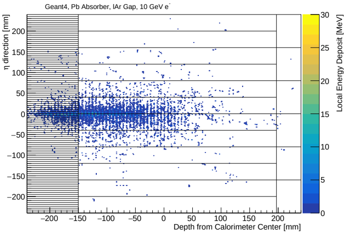
$\leftarrow \varphi$ of decay

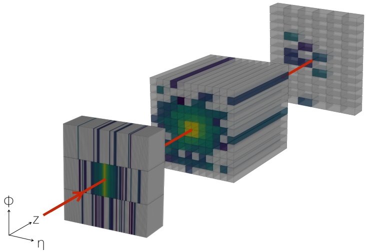
$\leftarrow$ propagator of decaying fermion

$\leftarrow$ multichannel

Applications: Calorimeter Shower Generation

- We consider a toy calorimeter inspired by the ATLAS ECal: flat alternating layers of lead and LAr
- They form three instrumented layers of dimension 3×96, 12×12, and 12×6
- Showers of e^+, γ, and π^+ (100k each)
- All are centered and perpendicular
- E_{inc} is uniform in $[1, 100]$ GeV





CaloGAN: Paganini, de Oliveira, Nachman [1705.02355, PRL; 1712.10321, PRD]

Calorimeter Shower Generation in 2 steps: learn $p(\vec{\mathcal{I}}|E_{\rm inc})$

Flow I

- learns $p_1(E_0, E_1, E_2|E_{\rm inc})$
- is optimized using the log-likelihood.

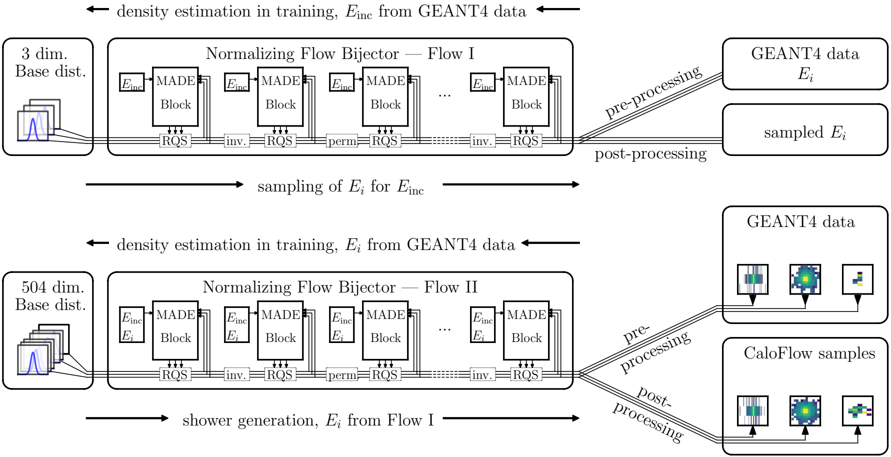
Flow II

- learns $p_2(\hat{\vec{\mathcal{I}}}|E_0, E_1, E_2, E_{\rm inc})$ of normalized showers
- in CaloFlow v1 (2106.05285 — called "teacher"):
  - Masked Autoregressive Flow trained with log-likelihood
  - Slow in sampling ($\approx 500\times$ slower than CaloGAN)
- in CaloFlow v2 (2110.11377 — called "student"):
  - Inverse Autoregressive Flow trained with Probability Density Distillation from teacher (log-likelihood prohibitive)

    van den Oord et al. [1711.10433]

    i.e. matching IAF parameters to frozen MAF
  - Fast in sampling ($\approx 500\times$ faster than CaloFlow v1)

Calorimeter Shower Generation in 2 steps: learn $p(\vec{\mathcal{I}}|E_{\text{inc}})$



Data processing Flow I

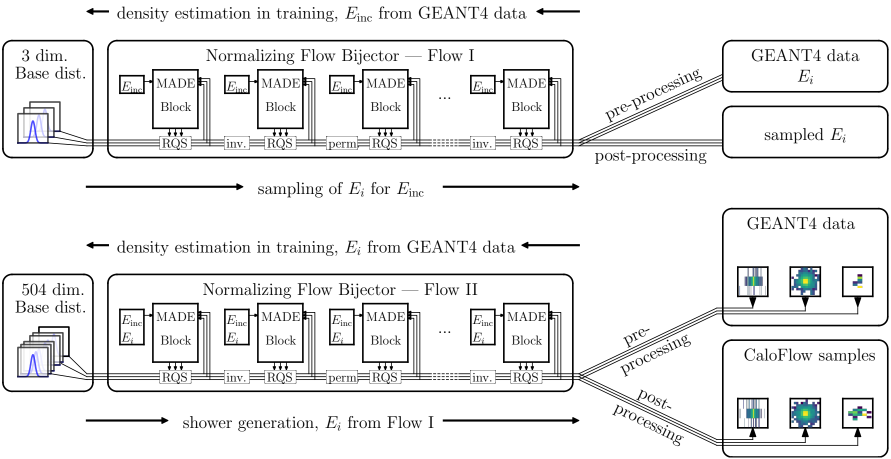
"$\leftarrow$" map E_i to $[0, 1]$

"$\leftarrow$" work in logit space

"$\rightarrow$" invert logit

"$\rightarrow$" map back to E_i

Calorimeter Shower Generation in 2 steps: learn $p(\vec{\mathcal{I}}|E_{\rm inc})$



Data processing Flow II

"←" add noise	"→" invert logit
"←" normalize layers to 1	"→" renormalize to E_i
"←" work in logit space	"→" apply threshold

Applications: Calorimeter Shower Generation

A Classifier provides the "ultimate metric".

According to the Neyman-Pearson Lemma we have:

- The likelihood ratio is the most powerful test statistic to distinguish the two samples.
- A powerful classifier trained to distinguish the samples should therefore learn (something monotonically related to) this.
- If this classifier is confused, we conclude $p_{\textsc{Geant4}}(x) = p_{\text{generated}}(x)$

⇒ This captures the full 504-dim. space.

? But why wasn't this used before?

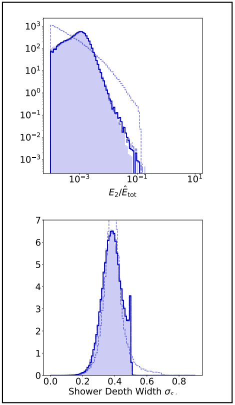
⇒ Previous deep generative models were separable to almost 100%!

DCTRGAN: Diefenbacher et al. [2009.03796, JINST]

Applications: Calorimeter Shower Generation

CK, D. Shih [2106.05285, 2110.11377]

- First generative model to fool a classifier.
- Does not scale well to higher dimensions.
- Good generation times with teacher-student-training or CL-based flow.



AUC	DNN based classifier			
	GEANT4 vs. CALOGAN	GEANT4 vs. (teacher) CALOFLOW v1	GEANT4 vs. (student) CALOFLOW v2	GEANT4 vs. CL-based flow
e^+	1.000(0)	0.859(10)	0.786(7)	0.638
γ	1.000(0)	0.756(48)	0.758(14)	0.631
π^+	1.000(0)	0.649(3)	0.729(2)	0.705

Work in progress with F. Ernst, L. Favaro, T. Plehn, D. Shih