Normalizing Flows

— COMETA WG2 Meeting —

Claudius Krause

Institute of High Energy Physics (HEPHY), Austrian Academy of Sciences (OeAW)

March 28, 2024

Normalizing Flows learn a coordinate transformation.

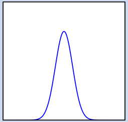
Function / Distribution in $z \in \mathcal{R}^d$ $\Leftrightarrow$ Normalizing Flow $\Leftrightarrow$ Function / Distribution in $x \in \mathcal{R}^d$

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

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

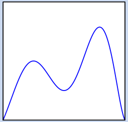
Mathematically speaking, this is a bijective function.

"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

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!

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

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

Base distributions

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

- 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 $\mathbb{R}$)

We need a trackable Jacobian and Inverse.

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

- 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.

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

- 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.

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

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

$\Rightarrow$ Let a NN learn parameters θ of a pre-defined transformation!

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

  $\Rightarrow$ 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

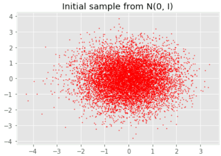
The full transformation is a chain of these bijectors.

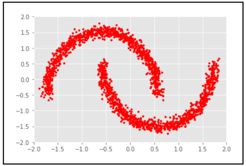
$$\xrightarrow{\pi_0(z_0)} \boxed{z_0 = f_0(z_1)} \xrightarrow{\pi_1(z_1)} \boxed{z_1 = f_1(z_2)} \xrightarrow{\dots} \boxed{z_i = f_i(z_{i+1})} \xrightarrow{\dots} \boxed{z_k = f_{k-1}(z_k)} \xrightarrow{\pi_k(z_k)}$$

A chain of bijectors is also a bijector

The full transformation is a chain of these bijectors.

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



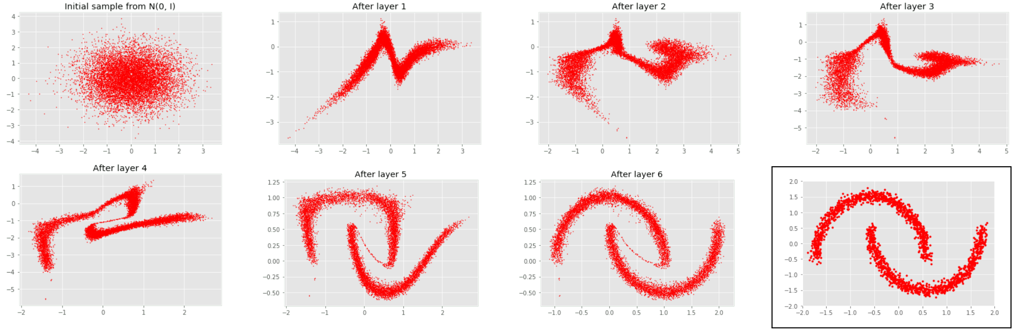


https://engineering.papercup.com/posts/normalizing-flows-part-2/

A chain of bijectors is also a bijector

The full transformation is a chain of these bijectors.

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



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

Affine Transformations

The coupling function (transformation)

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

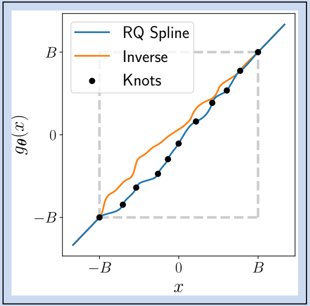
- It requires $x \in \mathbb{R}$.
- Inverse and Jacobian are trivial.
- Its transformation powers are limited.

Any monotonic function can be used.

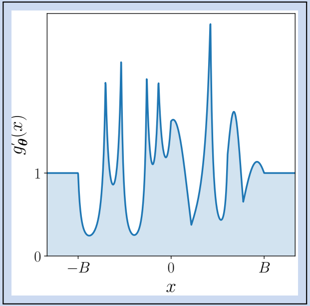
Changing coordinates from $\vec{z}$ to $\vec{x}$ with a map $\vec{x} = f(\vec{z})$ changes the distribution according to

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

A more complicated transformation then leads to a more complicated transformed distribution.



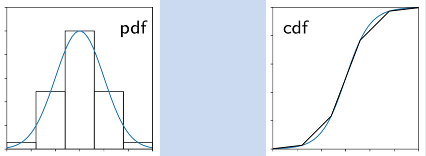
figures taken from Durkan et al. [arXiv:1906.04032]



Piecewise Transformations (Splines) in a finite domain

piecewise linear coupling function:

Müller et al. [arXiv:1808.03856]

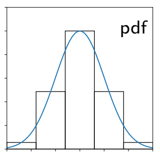


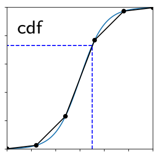
The NN predicts the pdf bin heights Q_i.

Piecewise Transformations (Splines) in a finite domain

piecewise linear coupling function:

Müller et al. [arXiv:1808.03856]





$$C = \sum_{k=1}^{b-1} Q_k + \alpha Q_b \qquad \alpha = \frac{x-(b-1)w}{w}$$

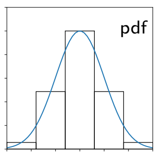
$$\left|\frac{\partial C}{\partial x_B}\right| = \prod_i \frac{Q_{b_i}}{w}$$

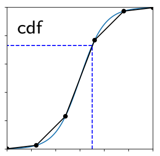
The NN predicts the pdf bin heights Q_i.

Piecewise Transformations (Splines) in a finite domain

piecewise linear coupling function:

Müller et al. [arXiv:1808.03856]





$$C = \sum_{k=1}^{b-1} Q_k + \alpha Q_b \qquad \alpha = \frac{x-(b-1)w}{w}$$

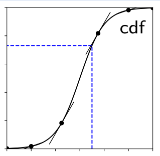
$$\left|\frac{\partial C}{\partial x_B}\right| = \prod_i \frac{Q_{b_i}}{w}$$

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

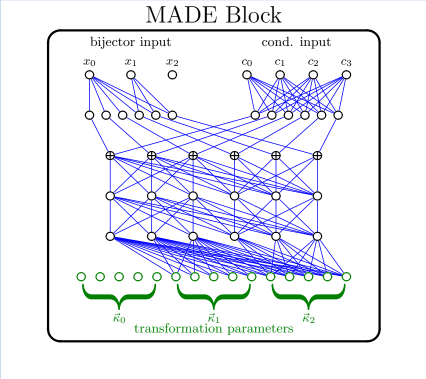
Remember: To tame the determinants, the parameters θ must depend only on a subset of all other coordinates.

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

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

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

Autoregressive NNs: MADE Blocks



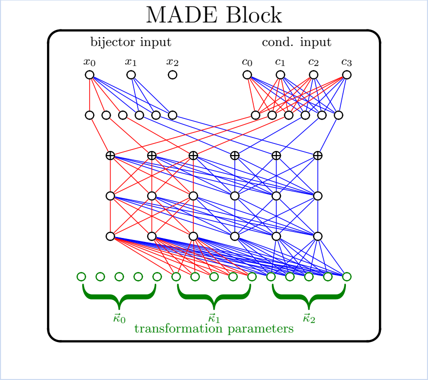
$$\boxed{\vec{\theta}_i = \vec{\theta}_i(x_1, x_2, \dots, x_{j<i})}$$

Implementation via masking:

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

Germain et al. [arXiv:1502.03509]

Autoregressive NNs: MADE Blocks



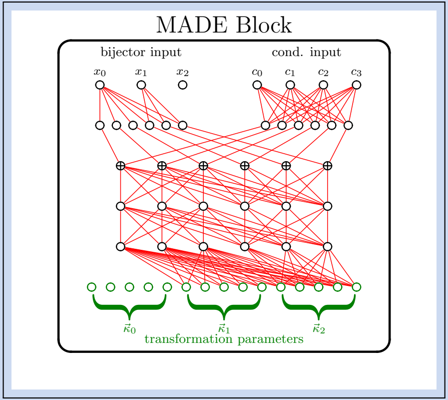
$$\boxed{\vec{\theta}_i = \vec{\theta}_i(x_1, x_2, \dots, x_{j<i})}$$

Implementation via masking:

- a single "forward" pass gives all $\vec{\theta}_i(x_1, \dots, x_{i-1})$.
  $\Rightarrow$ 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]

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:

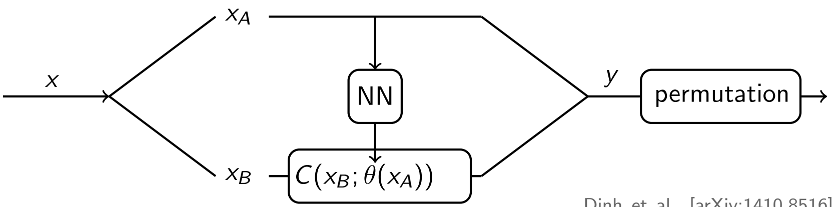
$$y_A = x_A$$
$$y_{B,i} = C(x_{B,i}; \theta(x_A))$$

inverse:

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

Jacobian:

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



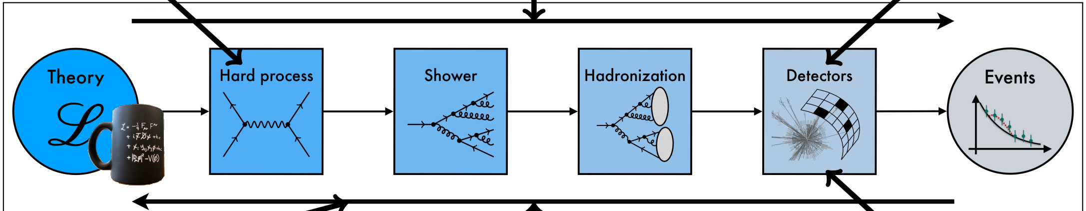
Dinh et al. [arXiv:1410.8516]

Applications of Normalizing Flows: An Outline

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

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

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



Machine Learning and LHC Event Generation, A. Butter et al. [2203.07460], R. Winterhalder

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

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

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

Lattice QCD
$\Rightarrow$ improve MCMC proposals

Astrophysics
$\Rightarrow$ stellar densities

Applications of Normalizing Flows: An Outline

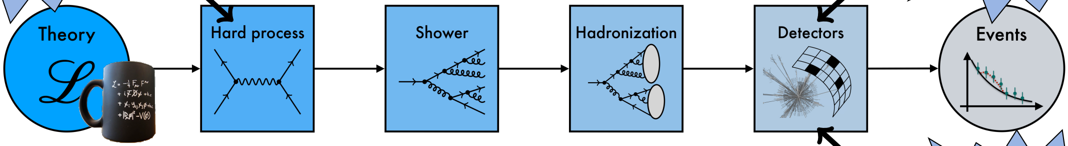
Phase Space Sampling
$\Rightarrow$ sample according to $d\sigma$

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

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

by Ramon

by Claudius



Machine Learning and LHC Event Generation, A. Butter et al. [2203.07460], R. Winterhalder

by Sofia

Inference
$\Rightarrow$ learn p(paramete

by Kim

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

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

Lattice QCD
$\Rightarrow$ improve MCMC proposals

Astrophysics
$\Rightarrow$ stellar densities