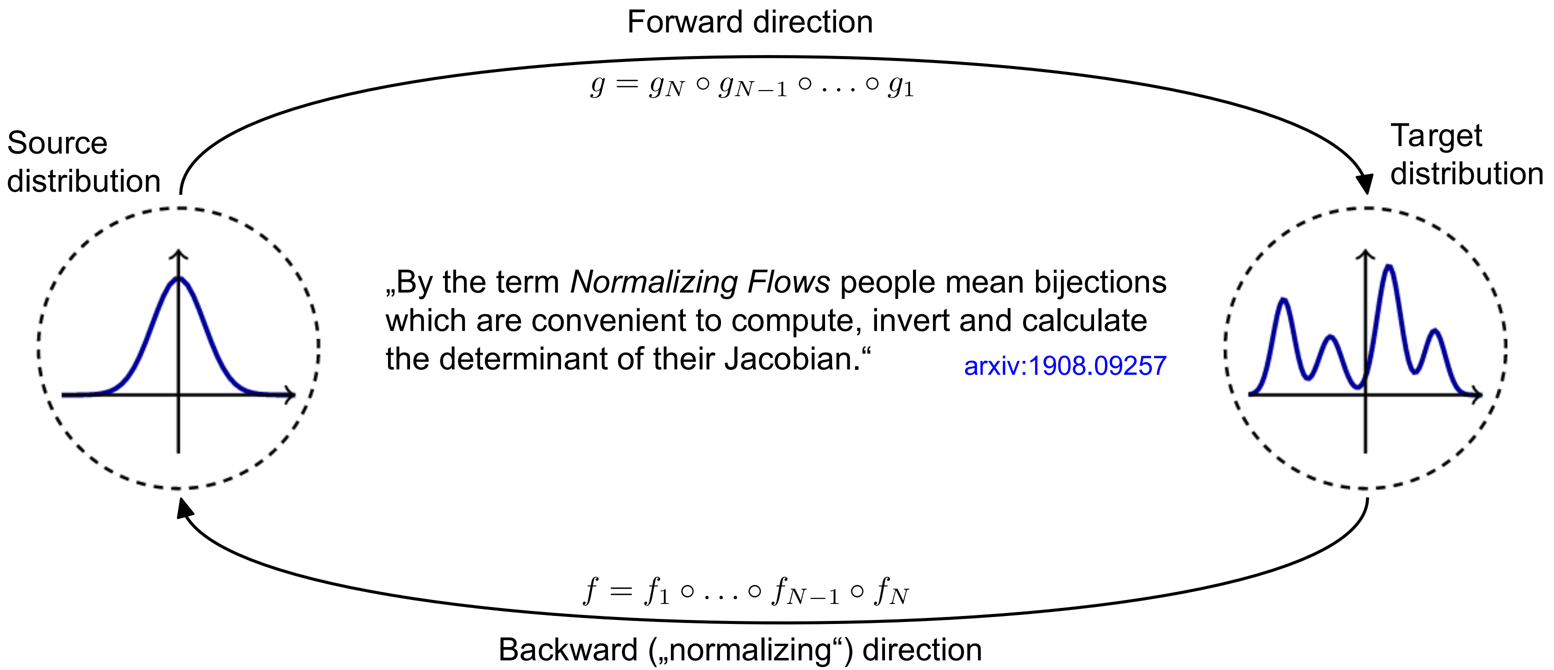
HighRR Lecture Week

11–15 September 2023
Ruprecht-Karls University, Heidelberg

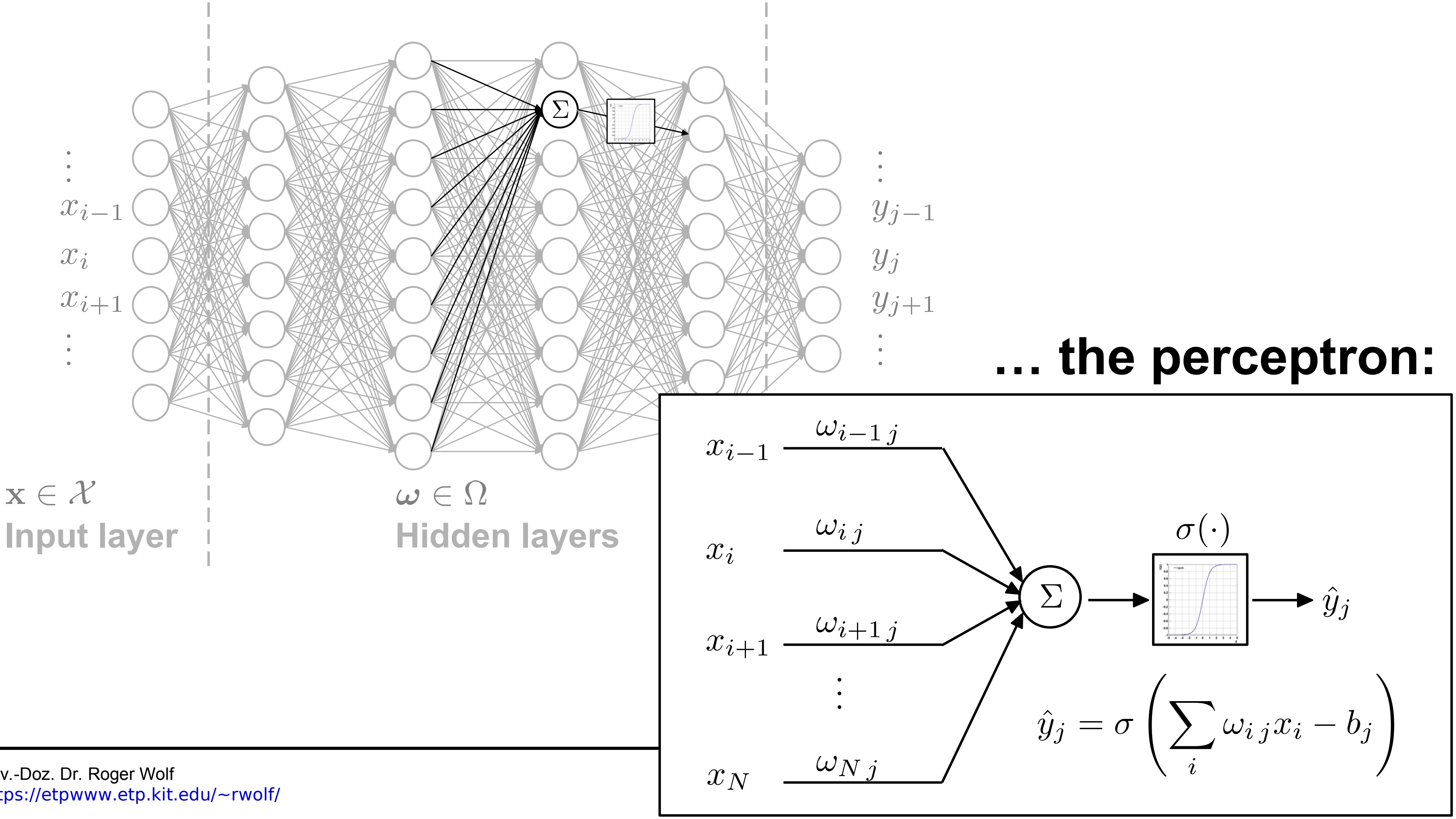
Normalizing Flows

Roger Wolf (roger.wolf@kit.edu)


Forward direction
$g = g_N \circ g_{N-1} \circ \ldots \circ g_1$
Source distribution
Target distribution
$f = f_1 \circ \ldots \circ f_{N-1} \circ f_N$
Backward („normalizing“) direction

„By the term *Normalizing Flows* people mean bijections which are convenient to compute, invert and calculate the determinant of their Jacobian.“ arxiv:1908.09257

The main building block of NNs …



… the perceptron:

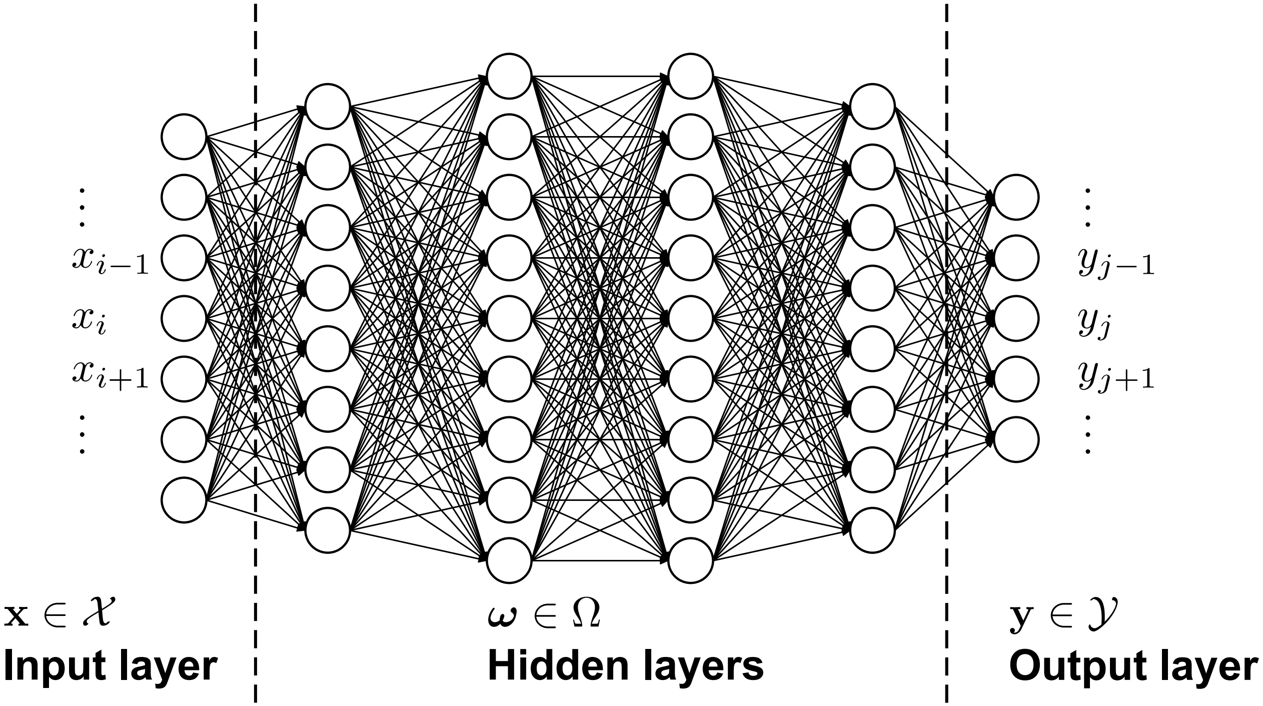
$$\hat{y}_j = \sigma\left(\sum_i \omega_{i\,j} x_i - b_j\right)$$

Priv.-Doz. Dr. Roger Wolf
https://etpwww.etp.kit.edu/~rwolf/

Fully connected feed-forward NN

- All nodes of consecutive layers are *connected* with each other.

- Inputs are propagated only in *forward* direction.



$\mathbf{x} \in \mathcal{X}$
Input layer

$\boldsymbol{\omega} \in \Omega$
Hidden layers

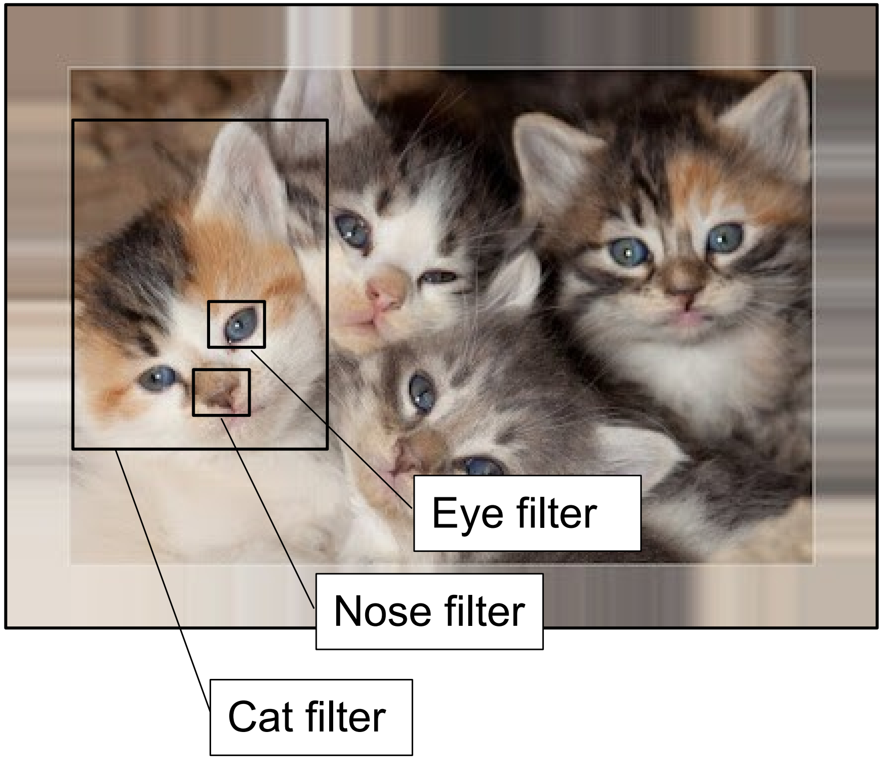
$\mathbf{y} \in \mathcal{Y}$
Output layer

- An NN is called **deep** if it has ≥2 hidden layers.

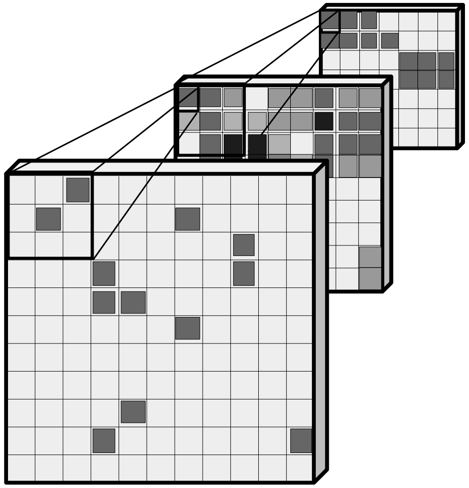
Convolutional NN (CNN)

- Inspired by 2D **image processing**.

- Reduce complexity by convolutional layers and *filters* (→ subnets scanning full images).



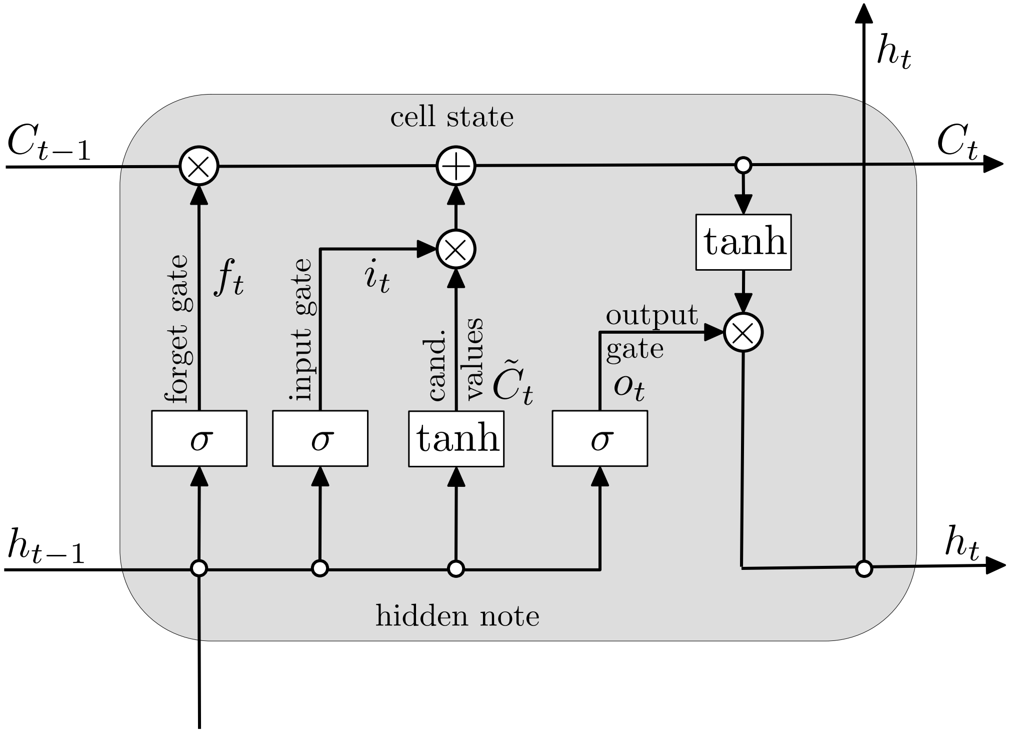
Example: 3-fold 3x3 convolution by summing



- Supports *2D translation invariance* of specific features (e.g. cats, eyes, noses) in images.

Recurrent NN (RNN)

- Inspired by **language processing** (→ sequential problem).
- Allow backward propagation and loops in the NN architecture (→ identify recurring features in sequences).



$$
\begin{aligned}
f_t &= \sigma\left(\omega_{hh}^{(f)}\mathbf{h}_{t-1} + \omega_{hx}^{(f)}\mathbf{x}_t + \mathbf{b}_f\right)\\
i_t &= \sigma\left(\omega_{hh}^{(i)}\mathbf{h}_{t-1} + \omega_{hx}^{(i)}\mathbf{x}_t + \mathbf{b}_i\right)\\
\tilde{C}_t &= \tanh\left(\omega_{hh}^{(C)}\mathbf{h}_{t-1} + \omega_{hx}^{(C)}\mathbf{x}_t + \mathbf{b}_C\right)\\
C_t &= f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t\\
o_t &= \sigma\left(\omega_{hh}^{(o)}\mathbf{h}_{t-1} + \omega_{hx}^{(o)}\mathbf{x}_t + \mathbf{b}_o\right)\\
h_t &= o_t \cdot \tanh\left(C_t\right)
\end{aligned}
$$

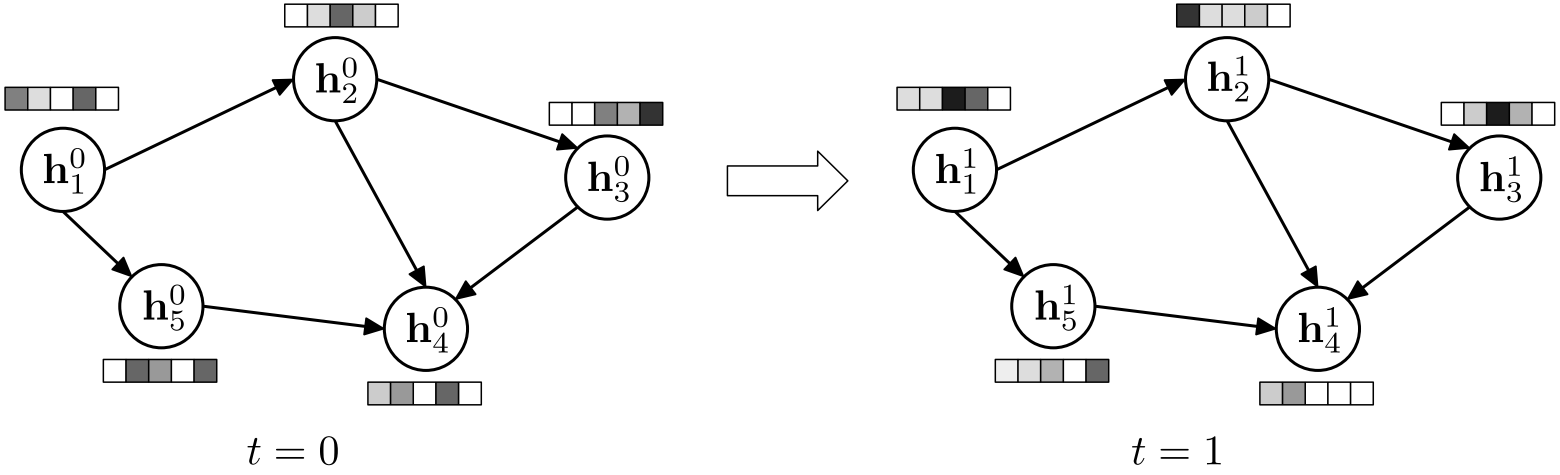
From „Understanding LSTM Networks“ (visited 30.05.22)

- Supports *translation invariance* of specific features (e.g. words) in sequences.

Graph NN (graphNN)

- Inspired by **unordered graph-like structures** with arbitrary number of nodes (→ particle clusters, traffic networks, molecules, …). Allows node, edge, and graph classification.

Node features in embedding space



Message passing/neighor aggregation:

$$\mathbf{h}_i^{t+1} = \sigma\left(\frac{1}{|N_i|}\mathbf{W}_t\mathbf{h}_i^t + \sum_{j\in N_i}\mathbf{W}_t\mathbf{h}_j^t\right)\,, \quad N_i : \text{Neighborhood of } i.$$

- Supports *permutation invariance* and versatility of the data.

Probabilistic generative NNs (PGNs)

- Applications
- GAN, VAE, **normalizing flow**
- Normalizing flow – in a nutshell

Cool applications …

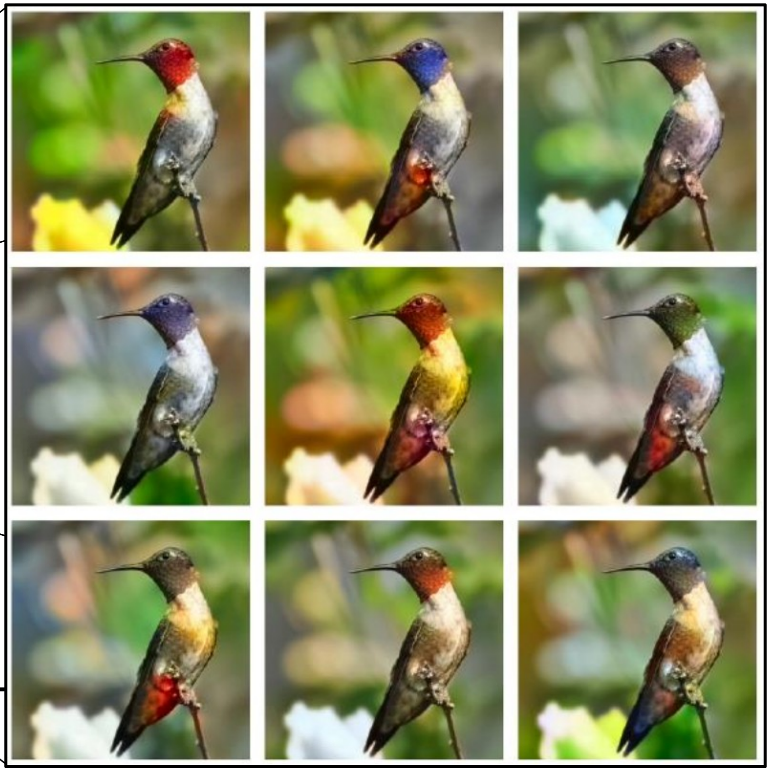
- **Create new examples** based on (implicit) rules, learned from (unlabeled) training data (→ prime example of *unsupervised learning*).

Example-1: Creation of non-existing faces

D. Kingma & P. Dariwahl *Glow: Generative Flow with invertible 1x1 convolutions*, NIPS 2018.

Example-2: Coloring of b/w pictures

arxiv:1907.02392 Guess which picture is the original one

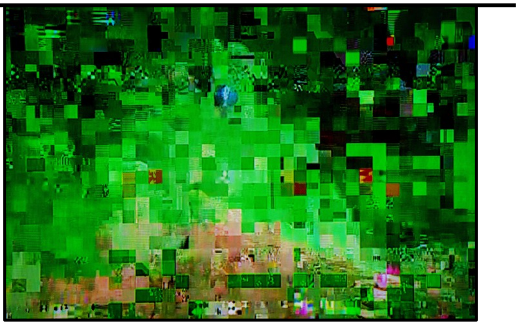


More useful applications …

Example-3: (Lossless [1]) compression of data

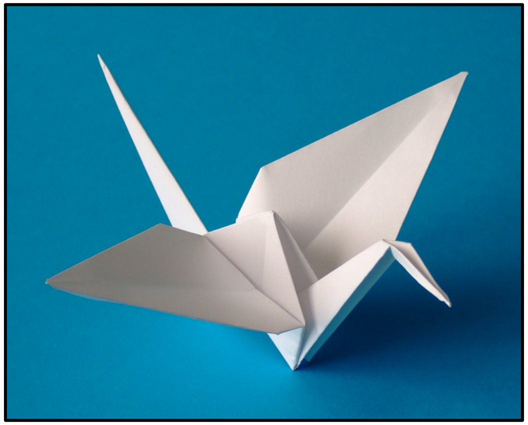
Example-4: (Fast) simulation (→ sampling of likelihoods).

Examples-5: Error correction.



Examples-6: Approximation of untractable likelihoods [2].

Example-7: Regularized unfolding [1].

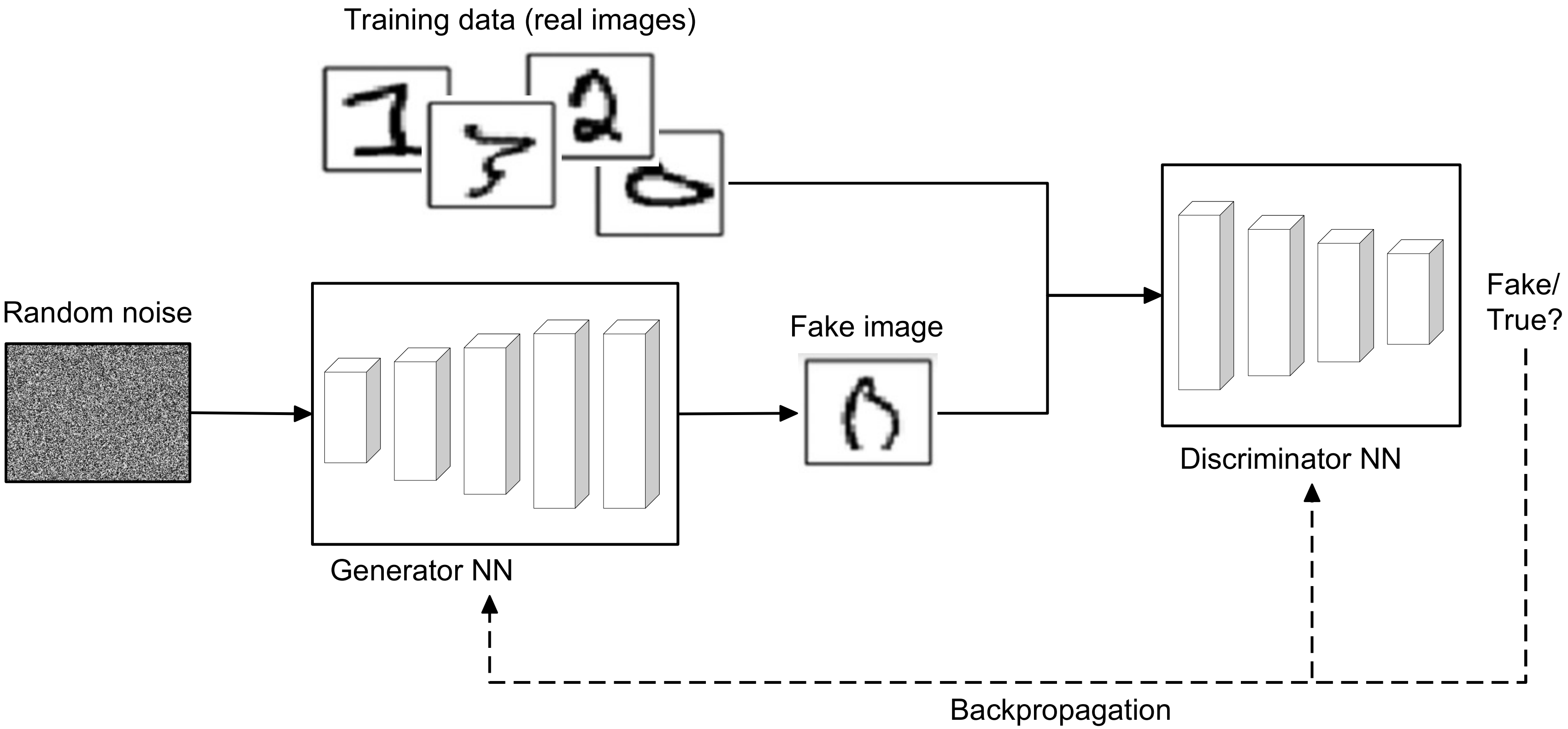


[1] Properties which are exclusive for normalizing flows.

[2] Either not analytically calculable or calculation generally unfeasible.

Generative adversarial NN (GAN)

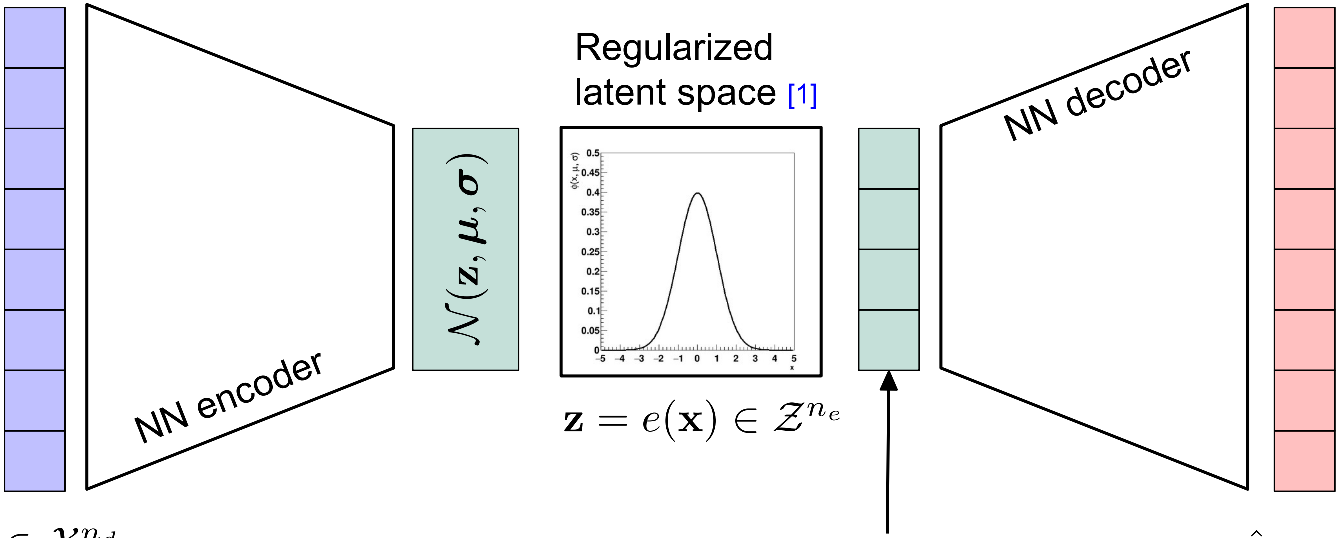
- *Generator* NN **competing** with (adversarial) *discriminator* NN ($\mathcal{D}$). Successful training, if $\mathcal{D}$ cannot distinguish between „Fake“ and „True“ outputs.



- MINIMAX problem → convergence not quaranteed.

Variational Auto Encoder (VAE)

- Map samples of the input space ($\mathcal{X}$) into a (high-dimensional) *latent space* ($\mathcal{Z}$, Encoder) and back (Decoder).
- After training, the Decoder can be used to **create new samples from** $\mathcal{Z}$.



Regularized latent space [1]

$\mathbf{z} = e(\mathbf{x}) \in \mathcal{Z}^{n_e}$

$\mathbf{x} \in \mathcal{X}^{n_d}$

Sampling from latent space

$$\begin{aligned}\hat{\mathbf{x}} &= d(\mathbf{z}) \\ &= d(e(\mathbf{x}))\end{aligned}$$

Loss function:

$$L = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 + \underbrace{\mathrm{KL}\left[\mathcal{N}(\mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\sigma}), \mathcal{N}(\mathbf{z}, 0, 1)\right]}_{\text{Kullback-Leibler divergence}}$$

[1] $\mathcal{N}(\mathbf{z}, 0, 1) = \frac{1}{\sqrt{2\pi}^{n_e}} \exp\left(-\frac{z^2}{2}\right)$

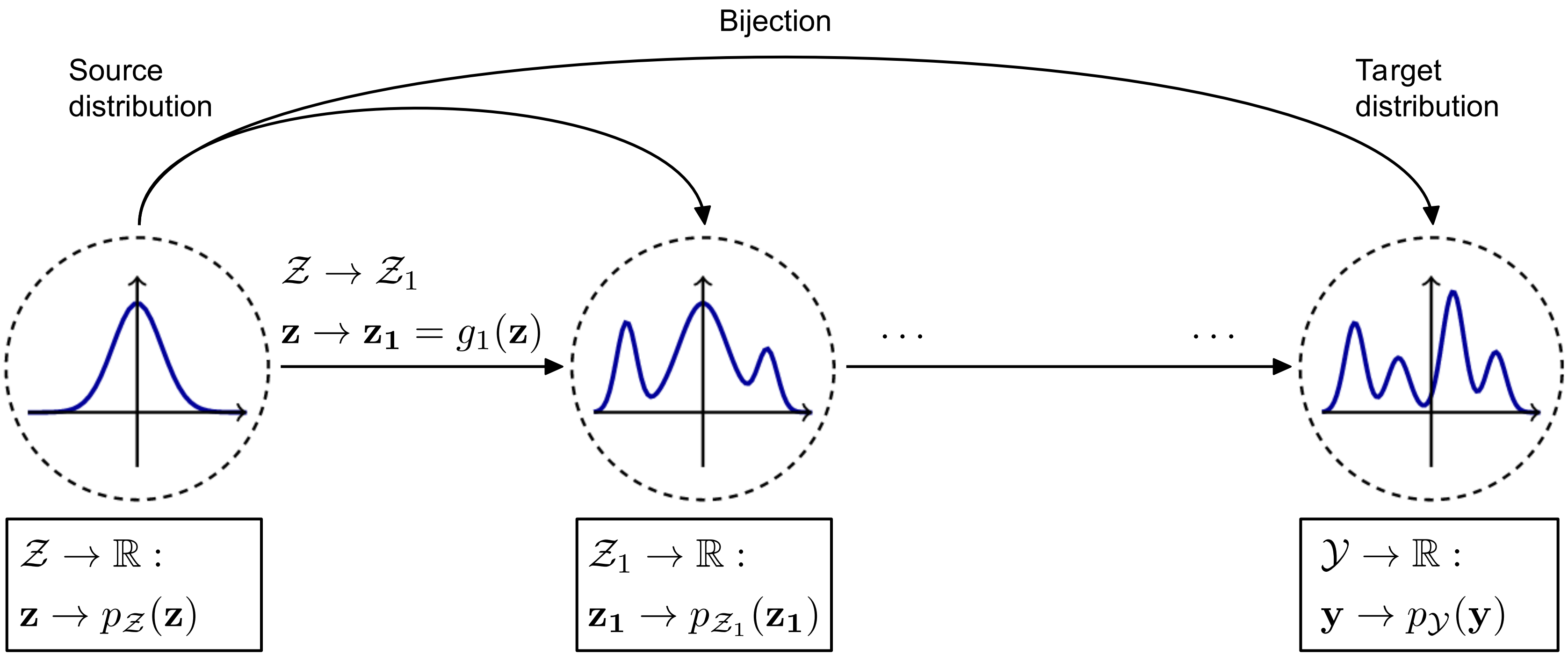
From „Understanding variational autoencoders (VAEs)“ (visited 31.05.22)

Normalizing flow

- Transform a (presumably simple) **source distribution** $p_{\mathcal{Z}}(\mathbf{z})$[1] **into any arbitrary target distribution** $p_{\mathcal{Y}}(\mathbf{y})$ by (repeated,) cleverly chosen *bijective variable transformation(s)* $\{g_i\}$.

$\mathcal{Z} \rightarrow \mathcal{Y}$ [2]

$$\mathbf{z} \rightarrow \mathbf{y} = \underbrace{g_N \circ g_{N-1} \circ \ldots \circ g_1}_{\equiv g}(\mathbf{z})$$



Bijection

Source distribution

Target distribution

$\mathcal{Z} \rightarrow \mathcal{Z}_1$

$\mathbf{z} \rightarrow \mathbf{z_1} = g_1(\mathbf{z})$

$\ldots$ $\ldots$

$\mathcal{Z} \rightarrow \mathbb{R}:$

$\mathbf{z} \rightarrow p_{\mathcal{Z}}(\mathbf{z})$

$\mathcal{Z}_1 \rightarrow \mathbb{R}:$

$\mathbf{z_1} \rightarrow p_{\mathcal{Z}_1}(\mathbf{z_1})$

$\mathcal{Y} \rightarrow \mathbb{R}:$

$\mathbf{y} \rightarrow p_{\mathcal{Y}}(\mathbf{y})$

[2] Originally $\mathcal{Z}$ and $\mathcal{Y}$ need to have same dimensionality (check also 1908.01686).

[1] This discussion is always with probability densities in mind.

Properties of g_i

cleverly chosen

$$\mathcal{Z} \to \mathcal{Y}$$

$$\mathbf{z} \to \mathbf{y} = \underbrace{g_N \circ g_{N-1} \circ \ldots \circ g_1}_{\equiv g}(\mathbf{z})$$

Properties/constraints of g_i :

- g_i must be a bijection, thus invertible (with $g_i^{-1} = f_i$);
- f_i should have an analytically closed form;
- both, g_i and f_i should be differentiable (→ backpropagation);
- The Jacobian determinant $\det\left(J_{g_i}(\mathbf{z}_i)\right)$ should be *easily calculable*;

Properties of g_i

$$\mathcal{Z}\to\mathcal{Y}$$

$$\mathbf{z}\to\mathbf{y}=\underbrace{g_N\circ g_{N-1}\circ\ldots\circ g_1}_{\equiv g}(\mathbf{z})$$

cleverly chosen

- g_i is referred to as (forward) flow.
- f_i is referred to as (backward) **normalizing flow**.

Properties/constraints of g_i :

- g_i must be a bijection, thus invertible (with $g_i^{-1}=f_i$);
- f_i should have an analytically closed form;
- both, g_i and f_i should be differentiable (→ backpropagation);
- The Jacobian determinant $\det\left(J_{g_i}(\mathbf{z}_i)\right)$ should be *easily calculable*;

Properties of g_i

$$\mathcal{Z} \to \mathcal{Y}$$

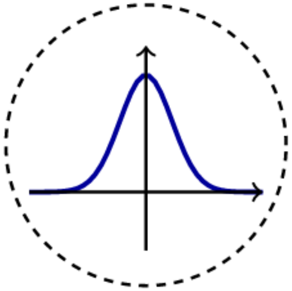
$$\mathbf{z} \to \mathbf{y} = \underbrace{g_N \circ g_{N-1} \circ \ldots \circ g_1}_{\equiv g}(\mathbf{z})$$

cleverly chosen

- g_i is referred to as (forward) flow.
- f_i is referred to as (backward) **normalizing flow**.

Properties/constraints of g_i :

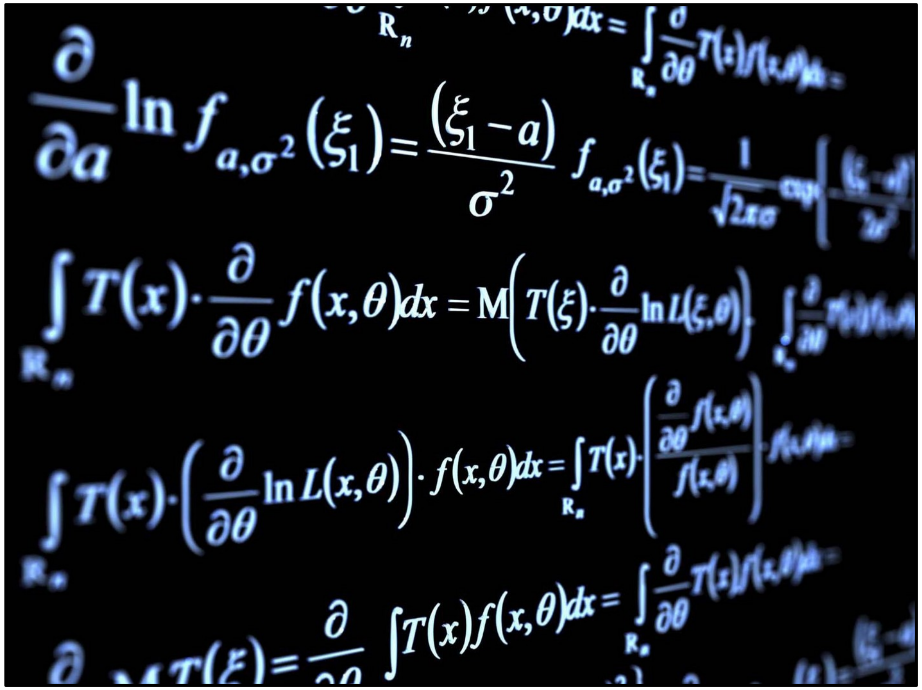
- g_i must be a bijection, thus invertible (with $g_i^{-1} = f_i$);
- f_i should have an analytically closed form;
- both, g_i and f_i should be differentiable (→ backpropagation);
- The Jacobian determinant $\det\left(J_{g_i}(\mathbf{z}_i)\right)$ should be *easily calculable*;



For reasons that will become clear soon people usually choose a standard Normal $\mathcal{N}(\mathbf{z}, 0, 1)$ [1] as source distribution.

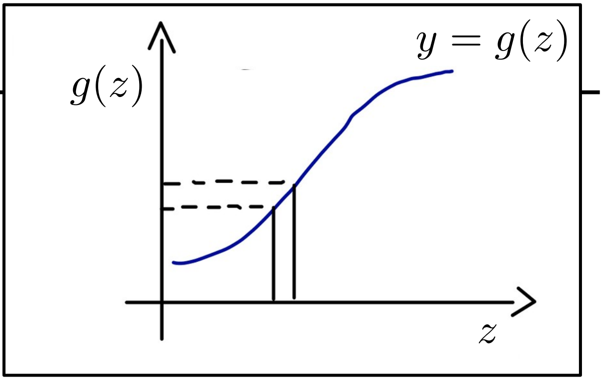
[1] $\mathcal{N}(\mathbf{z}, 0, 1) = \frac{1}{\sqrt{2\pi}^D} \exp\left(-\frac{\mathbf{z}^2}{2}\right)$

Math prerequisites


$\frac{\partial}{\partial a} \ln f_{a,\sigma^2}(\xi_1) = \frac{(\xi_1 - a)}{\sigma^2}$
$\int_{R_n} T(x) \cdot \frac{\partial}{\partial \theta} f(x,\theta) dx = M\left(T(\xi) \cdot \frac{\partial}{\partial \theta} \ln L(\xi,\theta) \right)$
$\int_{R_n} T(x) \cdot \left(\frac{\partial}{\partial \theta} \ln L(x,\theta) \right) \cdot f(x,\theta) dx = \int_{R_n} T(x) \cdot \left(\frac{\frac{\partial}{\partial \theta} f(x,\theta)}{f(x,\theta)} \right)$
$\int T(x) f(x,\theta) dx = \int_{R_n} \frac{\partial}{\partial \theta}$

- Change of variables and conservation of probability
- Composition of bijections
- Normalizing flow model & training strategy
- Overview of concrete implementations

Change of variables



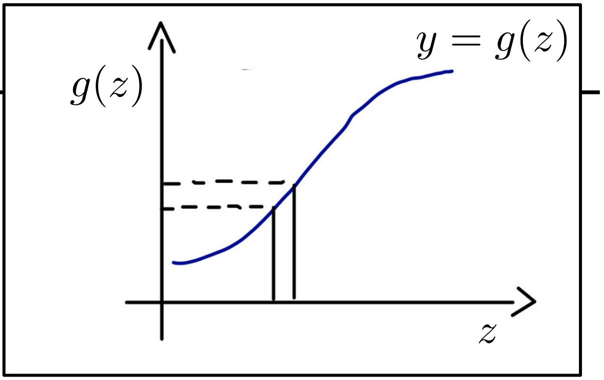
- $p_{\mathcal{Y}}(y)$ can be obtained from $p_{\mathcal{Z}}(z)$ via **conservation of probability**:

$$P(A) = \int\limits_A p_{\mathcal{Y}}(y)\,\mathrm{d}y = \int\limits_A p_{\mathcal{Z}}(z)\,\mathrm{d}z$$

$$p_{\mathcal{Y}}(y)\,\mathrm{d}y = p_{\mathcal{Z}}(z)\,\mathrm{d}z$$

$$p_{\mathcal{Y}}(y) = p_{\mathcal{Z}}(z)\left|\frac{\mathrm{d}z}{\mathrm{d}y}\right| = p_{\mathcal{Z}}(z)\left|\frac{\mathrm{d}f(y)}{\mathrm{d}y}\right| = p_{\mathcal{Z}}(z)\left|\frac{\mathrm{d}y}{\mathrm{d}z}\right|^{-1} = p_{\mathcal{Z}}(z)\left|\frac{\mathrm{d}g(z)}{\mathrm{d}z}\right|^{-1}$$

Change of variables



- $p_{\mathcal{Y}}(y)$ can be obtained from $p_{\mathcal{Z}}(z)$ via **conservation of probability**:

$$P(A) = \int\limits_A p_{\mathcal{Y}}(y)\,\mathrm{d}y = \int\limits_A p_{\mathcal{Z}}(z)\,\mathrm{d}z$$

$$p_{\mathcal{Y}}(y)\,\mathrm{d}y = p_{\mathcal{Z}}(z)\,\mathrm{d}z$$

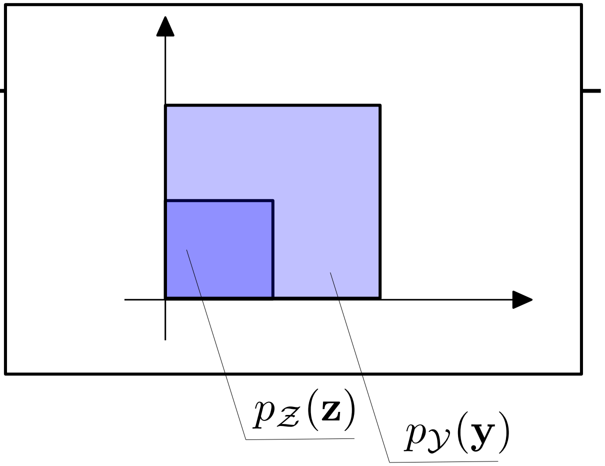
$$p_{\mathcal{Y}}(y) = p_{\mathcal{Z}}(z)\left|\frac{\mathrm{d}z}{\mathrm{d}y}\right| = p_{\mathcal{Z}}(z)\left|\frac{\mathrm{d}f(y)}{\mathrm{d}y}\right| = p_{\mathcal{Z}}(z)\left|\frac{\mathrm{d}y}{\mathrm{d}z}\right|^{-1} = p_{\mathcal{Z}}(z)\boxed{\left|\frac{\mathrm{d}g(z)}{\mathrm{d}z}\right|}^{-1}$$

For $\mathcal{Z},\mathcal{Y}\in\mathbb{R}^D$

$$\boxed{|\det(J_g)| = \begin{vmatrix} \frac{\partial g_1}{\partial z_1} & \frac{\partial g_1}{\partial z_2} & \dots & \frac{\partial g_1}{\partial z_n} \\ \frac{\partial g_2}{\partial z_1} & \frac{\partial g_2}{\partial z_2} & \dots & \frac{\partial g_2}{\partial z_n} \\ \vdots & & & \vdots \\ & \dots & & \frac{\partial g_n}{\partial z_n} \end{vmatrix}}$$

(Jacobian determinant)

Example-1



Bijection (forward flow):

$$g:\mathbb{R}^2\to\mathbb{R}^2;\quad \mathbf{z}\to g(\mathbf{z})=2\mathbf{z};\quad \text{with: } \mathbf{z}\equiv\left(\begin{array}{c} z_1\\ z_2\end{array}\right).$$

Inverse (backward flow):

$$f:\mathbb{R}^2\to\mathbb{R}^2;\quad \mathbf{y}\to f(\mathbf{y})=\frac{\mathbf{y}}{2};\quad \text{with: } \mathbf{y}\equiv\left(\begin{array}{c} y_1\\ y_2\end{array}\right).$$

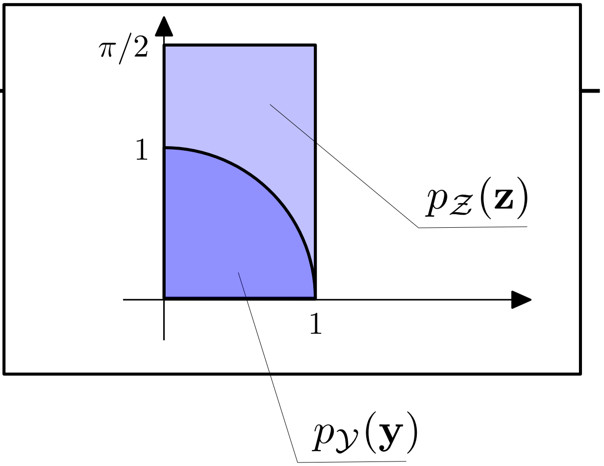
Transfomation of $p(\cdot)$:

$$p_{\mathcal{Z}}(\mathbf{z})=\left\{\begin{array}{ll} 1 & 0\leq x_{1,2}\leq a\\ 0 & \text{else}\end{array}\right. ;\qquad \det\left(J_g\right)=\left|\begin{array}{cc} 2 & 0\\ 0 & 2\end{array}\right|=4;\qquad p_{\mathcal{Y}}(\mathbf{y})=p_{\mathcal{Z}}(f(\mathbf{y}))\,\underbrace{\left|J_g\right|^{-1}}_{\equiv\frac{1}{4}}$$

$$p_{\mathcal{Y}}(\mathbf{y})=\left\{\begin{array}{ll} \frac{1}{4} & 0\leq y_{1,2}\leq 2\,a\\ 0 & \text{else}\end{array}\right. .$$

Here $p_{\mathcal{Z}}(\mathbf{z})$ is „streched“ over a 4 times larger volume in variable space.

Example-2



Bijection (forward flow):

$$g:\mathbb{R}^2\to\mathbb{R}^2;\quad \mathbf{z}\to g(\mathbf{z})=\begin{pmatrix} z_1\sin z_2 \\ z_1\cos z_2\end{pmatrix};\quad \text{with: } \mathbf{z}\equiv\begin{pmatrix} r \\ \varphi\end{pmatrix}.$$

Inverse (backward flow):

$$f:\mathbb{R}^2\to\mathbb{R}^2;\quad \mathbf{y}\to f(\mathbf{y})=\begin{pmatrix} \sqrt{y_1^2+y_2^2} \\ \arctan\left(y_2/y_1\right)\end{pmatrix};\quad \text{with: } \mathbf{y}\equiv\begin{pmatrix} x \\ y\end{pmatrix}.$$

Transfomation of $p(\cdot)$:

$$p_{\mathcal{Z}}(\mathbf{z})=\begin{cases} 1 & 0\leq x_{1,2}\leq a \\ 0 & \text{else}\end{cases};\qquad \det\left(J_g\right)=\begin{vmatrix} \sin z_2 & z_1\cos z_2 \\ \cos z_2 & -z_1\sin z_2\end{vmatrix}=z_1;$$

$$p_{\mathcal{Y}}(\mathbf{y})=p_{\mathcal{Z}}(f(\mathbf{y}))\,\underbrace{\left|J_g\right|^{-1}}$$

$$\equiv\frac{1}{\sqrt{y_1^2+y_2^2}}$$

Q: Is this variable transform volume preserving/compressing/expanding?

Composition of bijections

- A composition of bijections

$$\mathcal{Z} \to \mathcal{Y}$$
$$z \to y = \underbrace{g_N \circ g_{N-1} \circ \ldots \circ g_1}_{\equiv g}(z)$$

is a bijection in itself, with the inverse $f = f_1 \circ \ldots \circ f_{N-1} \circ f_N(y)$ and the transformation formulas

$$p_{\mathcal{Y}}(g(z)) = p_{\mathcal{Z}}(z) \prod_{i=1}^{N} \left|\frac{\mathrm{d}g_i(z_i)}{\mathrm{d}z_i}\right|^{-1}$$
$$p_{\mathcal{Y}}(y) = p_{\mathcal{Z}}(f(y)) \prod_{i=1}^{N} \left|\frac{\mathrm{d}f_i(y_i)}{\mathrm{d}y_i}\right|$$

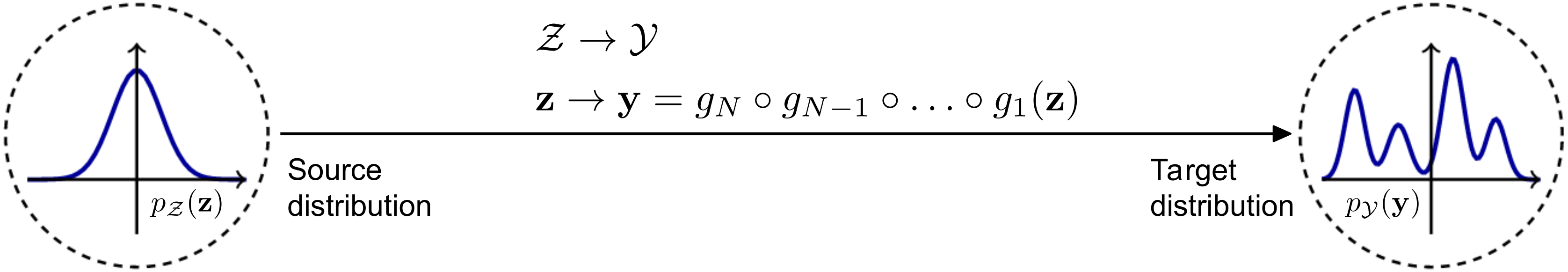
NB: Simple application of the *chain rule*.
NNB: One can omit the i in the derivatives.

Normalizing flow model

- A simple source distribution (e.g. $p_{\mathcal{Z}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, 0, 1)$) can be transformed into any arbitrary (potentially unknown) target distribution $p_{\mathcal{Y}}(\mathbf{y})$.

- The $\{g_i\}$ to do so, are a priori *unknown*, but they can be *approximated by any sufficiently expressive basic NN* ($p_{\mathcal{Y}}(\mathbf{y}) \rightarrow \hat{p}_{\mathcal{Y}}(\mathbf{y}, \boldsymbol{\omega})$.

- The objects to be learned are the bijections $\{g_i\}$ (resp. $\{f_i\}$). Knowing one implies knowledge of the other one.



$\mathcal{Z} \rightarrow \mathcal{Y}$

$\mathbf{z} \rightarrow \mathbf{y} = g_N \circ g_{N-1} \circ \ldots \circ g_1(\mathbf{z})$

Source distribution

Target distribution

Training objective

$\hat{p}_{\mathcal{Y}}(\mathbf{y},\boldsymbol{\omega})$ should match $p_{\mathcal{Y}}(\mathbf{y})$ as close as possible.

- Quantified by the **Kullback-Leibler** divergence $\mathrm{KL}[\,\cdot\,]$:

$$\mathrm{KL}[p_{\mathcal{Y}}(\boldsymbol{y}),\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})] = \int p_{\mathcal{Y}}(\boldsymbol{y})\ln\left(\frac{p_{\mathcal{Y}}(\boldsymbol{y})}{\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})}\right)\mathrm{d}\boldsymbol{y} = \mathrm{const.} - \underbrace{\int p_{\mathcal{Y}}(\boldsymbol{y})\ln\left(\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})\right)\mathrm{d}\boldsymbol{y}}_{\equiv E\left[\ln\left(\hat{p}_{\mathcal{Y}}(\boldsymbol{\omega})\right)\right]\text{[1]}}$$

$$= \mathrm{const.} - E\left[\ln\left(p_{\mathcal{Z}}(\mathbf{z})\prod_{i=1}^{N}\left|\frac{\partial g_i(\mathbf{z}_i,\boldsymbol{\omega})}{\partial \mathbf{z}_i}\right|^{-1}\right)\right]$$

$$= \mathrm{const.} - E\left[\ln\left(p_{\mathcal{Z}}(f(\mathbf{y}))\right)\right] - E\left[\sum_{i=1}^{N}\ln\left(\left|\frac{\partial f_i(\mathbf{y}_i,\boldsymbol{\omega})}{\partial \mathbf{y}_i}\right|\right)\right]$$

(Expected loss or risk)

Defining the log-likelihood ratio of the two distributions as loss.

[1] $E[\,\cdot\,]$ (Expectation value)

Training objective

$\hat{p}_{\mathcal{Y}}(\mathbf{y},\boldsymbol{\omega})$ should match $p_{\mathcal{Y}}(\mathbf{y})$ as close as possible.

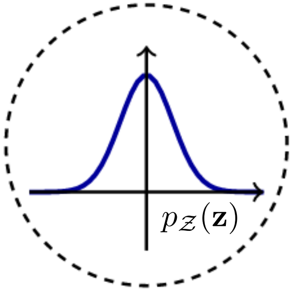
- Quantified by the **Kullback-Leibler** divergence $\mathrm{KL}[\,\cdot\,]$:

$$\mathrm{KL}[p_{\mathcal{Y}}(\boldsymbol{y}),\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})] = \int p_{\mathcal{Y}}(\boldsymbol{y})\ln\left(\frac{p_{\mathcal{Y}}(\boldsymbol{y})}{\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})}\right)\mathrm{d}\boldsymbol{y} = \mathrm{const.} - \underbrace{\int p_{\mathcal{Y}}(\boldsymbol{y})\ln\left(\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})\right)\mathrm{d}\boldsymbol{y}}_{\equiv E\left[\ln\left(\hat{p}_{\mathcal{Y}}(\boldsymbol{\omega})\right)\right]\text{[1]}}$$

$$= \mathrm{const.} - E\left[\ln\left(p_{\mathcal{Z}}(\mathbf{z})\prod_{i=1}^{N}\left|\frac{\partial g_i(\mathbf{z}_i,\boldsymbol{\omega})}{\partial\mathbf{z}_i}\right|^{-1}\right)\right]$$

$$= \mathrm{const.} - E\underbrace{\Big[\ln\left(p_{\mathcal{Z}}(f(\mathbf{y}))\right)\Big]}_{\propto E\left[\|f(\mathbf{y})\|_2^2\right] = 0} - E\left[\sum_{i=1}^{N}\ln\left(\left|\frac{\partial f_i(\mathbf{y}_i,\boldsymbol{\omega})}{\partial\mathbf{y}_i}\right|\right)\right]$$

(Expected loss or risk)



with: $\quad p_{\mathcal{Z}}(\mathbf{z}) = \mathcal{N}(\mathbf{z},0,1)$

Defining the log-likelihood ratio of the two distributions as loss.

[1] $E[\,\cdot\,]$ (Expectation value)

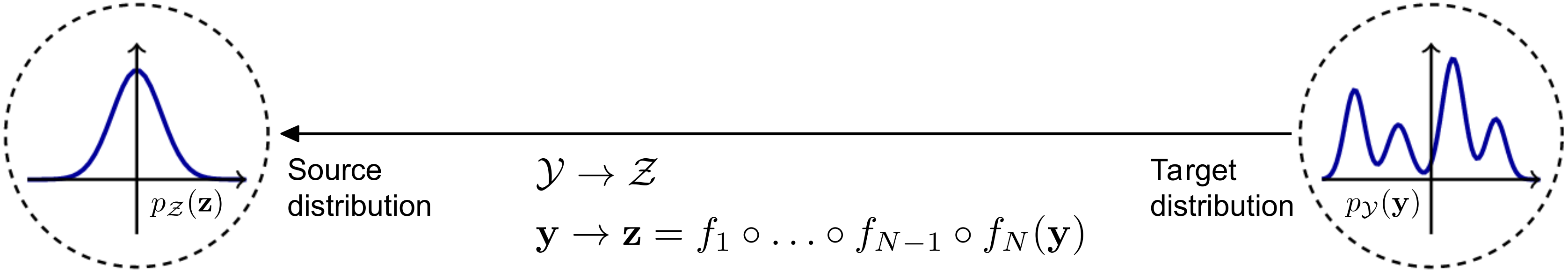
Training strategy

- Assume that we don‘t know $p_{\mathcal{Y}}(\mathbf{y})$, *but we can sample from it*, e.g., via the Monte Carlo method (ignoring the $\mathrm{const.}$).

$$L = E\left[\|f(\mathbf{y})\|_2^2\right] - E\left[\sum_{i=1}^{N}\ln\left(\left|\frac{\partial f_i(\mathbf{y}_i,\boldsymbol{\omega})}{\partial\mathbf{y}_i}\right|\right)\right]$$ (Risk functional)

$$R = \frac{1}{2}\mathrm{MSE}[\mathbf{y}] - \frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{N}\ln\left(\left|\frac{\partial f_i(\mathbf{y}_i,\boldsymbol{\omega})}{\partial\mathbf{y}_i}\right|\right)$$ (Empirical risk functional)

- Train f_i in **reverse order**, in (mini-)batches of m simulated events, mapping $\mathbf{y}$ to the trivially known source distribution $\mathcal{N}(\mathbf{z},0,1)$.



Source distribution

$\mathcal{Y}\to\mathcal{Z}$

$\mathbf{y}\to\mathbf{z} = f_1\circ\ldots\circ f_{N-1}\circ f_N(\mathbf{y})$

Target distribution

Training strategy

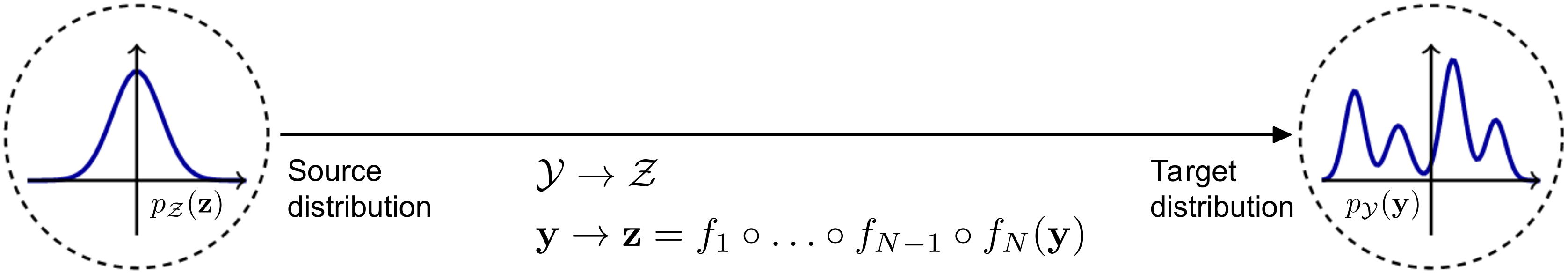
- Assume that we don‘t know $p_{\mathcal{Y}}(\mathbf{y})$, *but we can sample from it*, e.g., via the Monte Carlo method (ignoring the $\mathrm{const.}$).

$$L = E\left[\|f(\mathbf{y})\|_2^2\right] - E\left[\sum_{i=1}^{N}\ln\left(\left|\frac{\partial f_i(\mathbf{y}_i,\boldsymbol{\omega})}{\partial\mathbf{y}_i}\right|\right)\right] \quad \text{(Risk functional)}$$

$$R = \frac{1}{2}\mathrm{MSE}[\mathbf{y}] - \frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{N}\ln\left(\left|\frac{\partial f_i(\mathbf{y}_i,\boldsymbol{\omega})}{\partial\mathbf{y}_i}\right|\right) \quad \text{(Empirical risk functional)}$$

- Train f_i in **reverse order**, in (mini-)batches of m simulated events, mapping $\mathbf{y}$ to the trivially known source distribution $\mathcal{N}(\mathbf{z},0,1)$.

- The evaluation happens in **forward direction** sampling from $\mathcal{N}(\mathbf{z},0,1)$.



Source distribution

$\mathcal{Y} \to \mathcal{Z}$

$\mathbf{y} \to \mathbf{z} = f_1 \circ \ldots \circ f_{N-1} \circ f_N(\mathbf{y})$

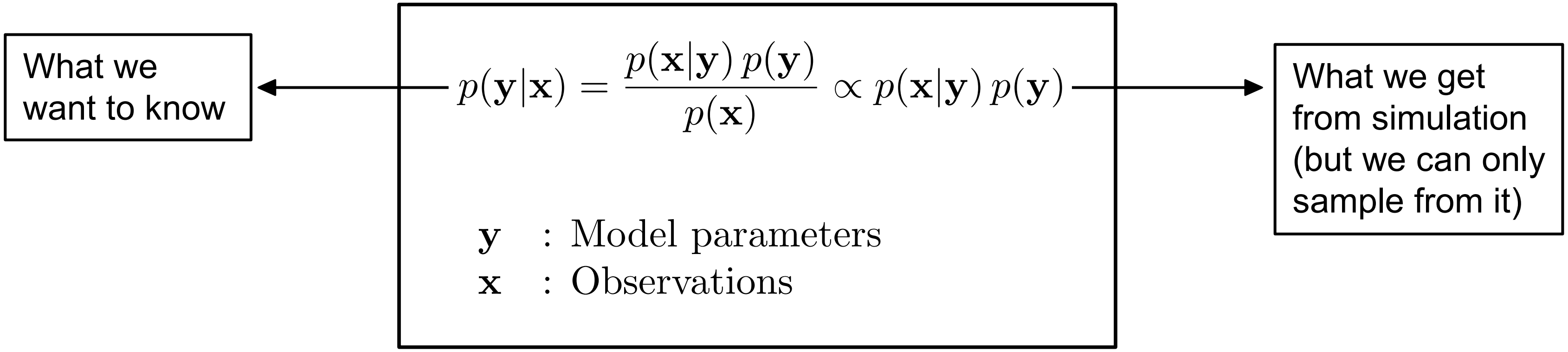
Target distribution

Inverse problem

- We use complex Monte Carlo simulations to obtain the **likelihood** $p(\mathbf{x}|\mathbf{y})$ to observe $\mathbf{x}$ given the model parameters $\mathbf{y}$.

- $p(\mathbf{x}|\mathbf{y})$ is *untractable*; we can only sample from it.

- For measurements we are interested in the **posterior** $p(\mathbf{y}|\mathbf{x})$ that can be obtained from Bayes theorem:



What we want to know

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{y})\,p(\mathbf{y})}{p(\mathbf{x})} \propto p(\mathbf{x}|\mathbf{y})\,p(\mathbf{y})$$

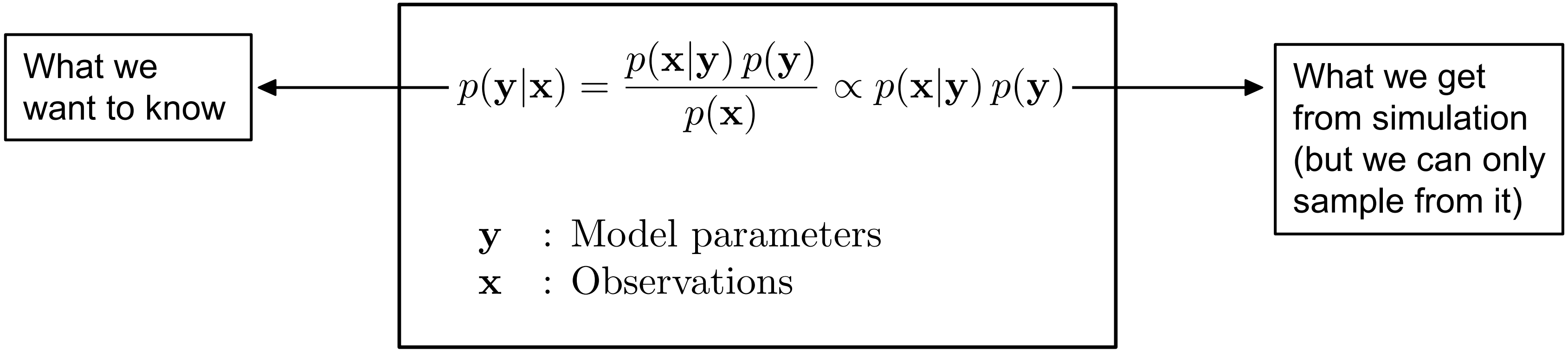
$\mathbf{y}$: Model parameters

$\mathbf{x}$: Observations

What we get from simulation (but we can only sample from it)

Inverse problem ↔ normalizing flow

- The space of $\mathcal{Y}$ can be high-dimensional and sampling from $\mathcal{Y}$ tedious.
- The normalizing flow can be used to map $p_{\mathcal{Y}}(\mathbf{y}|\mathbf{x})$ to $p_{\mathcal{Z}}(\mathbf{z})$ (during training). **NB**: This can still be tedious.
- In the forward pass (after training) $p_{\mathcal{Z}}(\mathbf{z})$ can be sampled with **significantly reduced effort**.
- Since the likelihood is never explicitly used, this procedure is referred to as „*likelihood-free inference*“.



What we want to know

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{y})\,p(\mathbf{y})}{p(\mathbf{x})} \propto p(\mathbf{x}|\mathbf{y})\,p(\mathbf{y})$$

$\mathbf{y}$: Model parameters

$\mathbf{x}$: Observations

What we get from simulation (but we can only sample from it)

Concrete implementations

- Subject of research of normalizing flows: construct g such that the flow is expressive and f and $\det(J_{g_i})$ can be obtained at low computatonial cost.

Implementation	Characteristic	Comments
Elementswise	Non-linear elementwise transform	No mixing of variables
Linear	Affine combination of variables	Limited represenational power
Planar and radial flows	Non-linear transformations	Hard to compute inverse
Coupling flows	Architectures that allow invertible non-linear transformations	Several couplings
Autoregressive flows		
Residual flows	Invertible residual flows	
Infinitesimal flows	Continuous flows based ODEs or SDEs	

Taken from arxiv:1908.09257

- We will focus on **planar** and **coupling flows** (viz. the RealNVP and cINN).

The planar flow

- Planar flow definition
- Jacobian determinant
- Backward flow

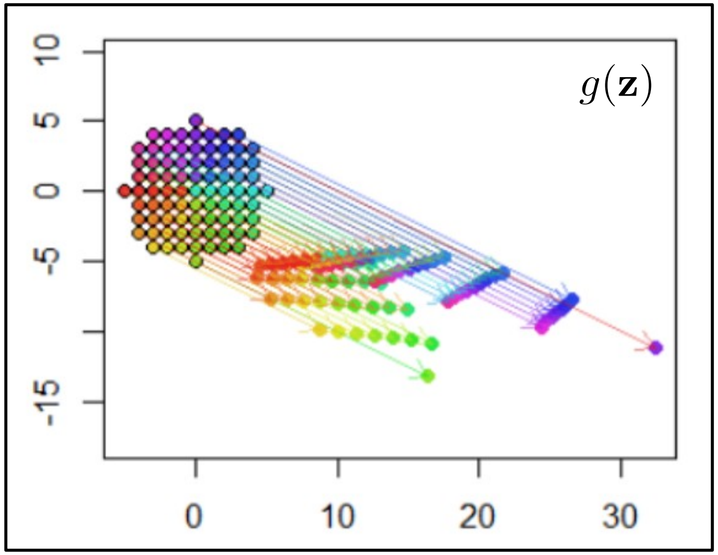
Forward flow $g(\mathbf{z})$

- One of the simplest transformations one could think of is of the form:

$$g(\mathbf{z}) = \mathbf{z} + \mathbf{u}\, h(\mathbf{w}^{\mathsf{T}}\mathbf{z} + b)$$

$$\text{with: } \mathbf{u}, \mathbf{w} \in \mathbb{R}^{D}, b \in \mathbb{R}$$

$$h(\cdot) \quad : \text{ non-linearity, e.g. } \tanh(\cdot).$$



Taken from stackexchange (visited 04.06.22)

- $g(\mathbf{z})$ shifts every point $\mathbf{z} \in \mathbb{R}^{D}$ parallel to $\mathbf{u}$.

- The argument $\mathbf{w}^{\mathsf{T}}\mathbf{z} - b = 0$ of $h(\cdot)$ defines a hyperplane in $\mathbb{R}^{D}$ perpendicular to $\mathbf{w}$. The function $h(\cdot)$ scales the shift along $\mathbf{u}$ depending on the distance of $\mathbf{z}$ from this hyperplane (→ **planar flow**).

- **NB**: If $\mathbf{z}$ is stretched depending on the distance from a fixed point this defines a **radial flow**.

Jacobian determinant $\det\left(J_g\right)$

- The Jacobian determinant can be easily obtained (with complexity $\mathcal{O}(D)$) from the matrix determinant lemma (MDL):

$$g(\mathbf{z}) = \mathbf{z} + \mathbf{u}\,h(\mathbf{w}^{\intercal}\mathbf{z} + b)$$

$$\text{with: }\; \mathbf{u}, \mathbf{w} \in \mathbb{R}^{D}, b \in \mathbb{R}$$

MDL

$$\boxed{\det\left(\mathbf{A} + \mathbf{u}\mathbf{w}^{\intercal}\right) = \left(1 + \mathbf{w}^{\intercal}\mathbf{A}^{-1}\mathbf{u}\right)\det\left(\mathbf{A}\right)}$$

$$\text{with: }\; \mathbf{A} \equiv \mathbb{I}_{D}; \quad \mathbf{z}' = \mathbf{w}^{\intercal}\mathbf{z} + b; \quad \frac{\partial}{\partial\mathbf{z}}h(\mathbf{z}') = \frac{\partial}{\partial\mathbf{z}'}h(\mathbf{z}')\mathbf{w}^{\intercal} \equiv h'(\mathbf{z}')\mathbf{w}^{\intercal}$$

$$\det\left(J_g\right) = \det\left(\underbrace{\mathbb{I}_{D} + \mathbf{u}\,h'(\mathbf{z}')\mathbf{w}^{\intercal}}_{=\frac{\partial}{\partial\mathbf{z}}g(\mathbf{z})}\right) = \left(1 + h'(\mathbf{z}')\mathbf{w}^{\intercal}\mathbf{u}\right)$$

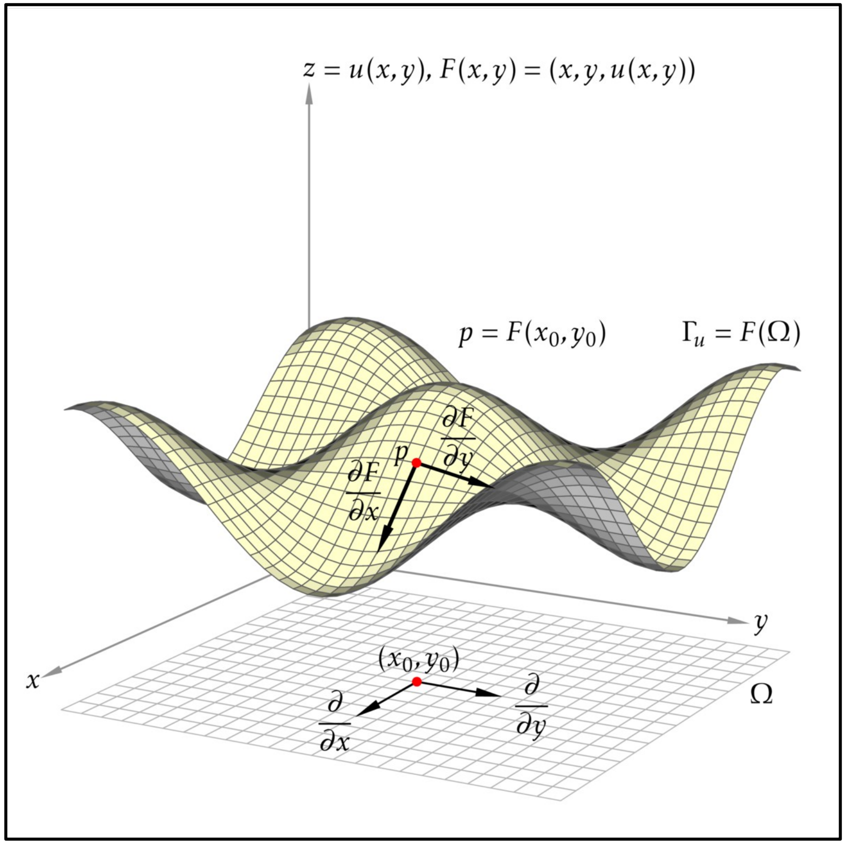
Backward flow $f(\mathbf{y})$

- A peculiarity of the planar flow is that the existence of $f(\mathbf{y})$ depends on the choice of $h(\cdot)$ and the parameters $\mathbf{u}$, $\mathbf{w}$.

- For $h(\cdot) = \tanh(\cdot)$ the condition $\mathbf{w}^{\mathsf{T}}\mathbf{u} \geq -1$ is sufficient for $f(\mathbf{y})$ to exist, as shown in 1505.05770 (Appendix A.1).

arxiv:1605.08803

The RealNVP


$z = u(x,y)$, $F(x,y) = (x,y,u(x,y))$
$p = F(x_0,y_0)$
$\Gamma_u = F(\Omega)$
p
$\frac{\partial F}{\partial x}$
$\frac{\partial F}{\partial y}$
y
x
(x_0,y_0)
$\frac{\partial}{\partial x}$
$\frac{\partial}{\partial y}$
Ω

RealNVP = real-valued non-volume preserving

- Coupling layer definition
- Backward flow
- Jacobian determinant
- Permutation layer
- Conditional invertible NN (cINN)

Forward flow $g(\mathbf{z})$

- The main component of the RealNVP is the **coupling layer**:

- We assume the input to the coupling layer to be split in $\mathbf{z} = [\mathbf{z_a}, \mathbf{z_b}]$ and apply the following transformation:

$$g : \mathbb{R}^D \to \mathbb{R}^D \quad \begin{pmatrix} \mathbf{z_a} \\ \mathbf{z_b} \end{pmatrix} \to \begin{pmatrix} \mathbf{y_a} \\ \mathbf{y_b} \end{pmatrix} = \begin{pmatrix} \mathbf{z_a} \\ \exp(s(\mathbf{z_a})) \odot \mathbf{z_b} + t(\mathbf{z_a}) \end{pmatrix},$$

  where $\odot$ refers to an elementwise product, and $s(\cdot)$ and $t(\cdot)$ are abitrary neural NNs, called *scaling* and *transition* NNs.

- We assume the splitting of $[\mathbf{z_a}, \mathbf{z_b}]$ to be arranged in the following way: $\mathbf{z_a} : z_{1:d},\ \mathbf{z_b} : z_{d+1:D}$ (in *python* slicing notation).

Backward flow $f(\mathbf{y})$

- The **inverse** of $g(\cdot)$ in this case can be easily obtained:

$$g : \mathbb{R}^D \to \mathbb{R}^D \quad \begin{pmatrix} \mathbf{z_a} \\ \mathbf{z_b} \end{pmatrix} \to \begin{pmatrix} \mathbf{y_a} \\ \mathbf{y_b} \end{pmatrix} = \begin{pmatrix} \mathbf{z_a} \\ \exp(s(\mathbf{z_a})) \odot \mathbf{z_b} + t(\mathbf{z_a}) \end{pmatrix},$$

$$f : \mathbb{R}^D \to \mathbb{R}^D \quad \begin{pmatrix} \mathbf{y_a} \\ \mathbf{y_b} \end{pmatrix} \to \begin{pmatrix} \mathbf{z_a} \\ \mathbf{z_b} \end{pmatrix} = \begin{pmatrix} \mathbf{y_a} \\ (\mathbf{y_b} - t(\mathbf{z_a})) \odot \exp(-s(\mathbf{z_a})) \end{pmatrix},$$

- $g(\mathbf{z_a}) = \mathbf{z_a}$ is just the identity.
- $g(\mathbf{z_b})$ is just an affine function that can be easily inverted.
- The use of $\exp(\cdot)$ prevents division by 0.

Jacobian determinant $\det\left(J_g\right)$

- J_g is a *triangular matrix* of which the determinant again is easy to calculate (with complexity $\mathcal{O}(D)$) as the product of the diagonal elements:

$$|\det(J_g)| = \left| \begin{array}{ccc|ccc} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & & 1 & 0 & & 0 \\ \hline \frac{\partial y_{d+1}}{\partial z_1} & \cdots & \frac{\partial y_{d+1}}{\partial z_d} & \exp(s(\mathbf{z_a})) & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & \ddots & 0 \\ \frac{\partial y_D}{\partial z_1} & \cdots & \frac{\partial y_D}{\partial z_d} & 0 & 0 & \exp(s(\mathbf{z_a})) \end{array} \right|$$

$$= \prod_{j=d+1}^{D} \exp(s(\mathbf{z_a}))_j = \exp\left(\sum_{j=d+1}^{D} s(\mathbf{z_a})_j \right)$$

Training objective – revisited –

$\hat{p}_{\mathcal{Y}}(\mathbf{y},\boldsymbol{\omega})$ should match $p_{\mathcal{Y}}(\mathbf{y})$ as close as possible.

- Quantified by the **Kullback-Leibler** divergence $\mathrm{KL}[\,\cdot\,]$:

$$\mathrm{KL}[p_{\mathcal{Y}}(\boldsymbol{y}),\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})] = \int p_{\mathcal{Y}}(\boldsymbol{y})\ln\left(\frac{p_{\mathcal{Y}}(\boldsymbol{y})}{\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})}\right)\mathrm{d}\boldsymbol{y} = \mathrm{const.} - \underbrace{\int p_{\mathcal{Y}}(\boldsymbol{y})\ln\left(\hat{p}_{\mathcal{Y}}(\boldsymbol{y},\boldsymbol{\omega})\right)\mathrm{d}\boldsymbol{y}}_{\equiv E\left[\ln\left(\hat{p}_{\mathcal{Y}}(\boldsymbol{\omega})\right)\right]}$$

$$= \mathrm{const.} - E\left[\ln\left(p_{\mathcal{Z}}(\mathbf{z})\prod_{i=1}^{N}\left|\frac{\partial g_i(\mathbf{z}_i,\boldsymbol{\omega})}{\partial\mathbf{z}_i}\right|^{-1}\right)\right]$$

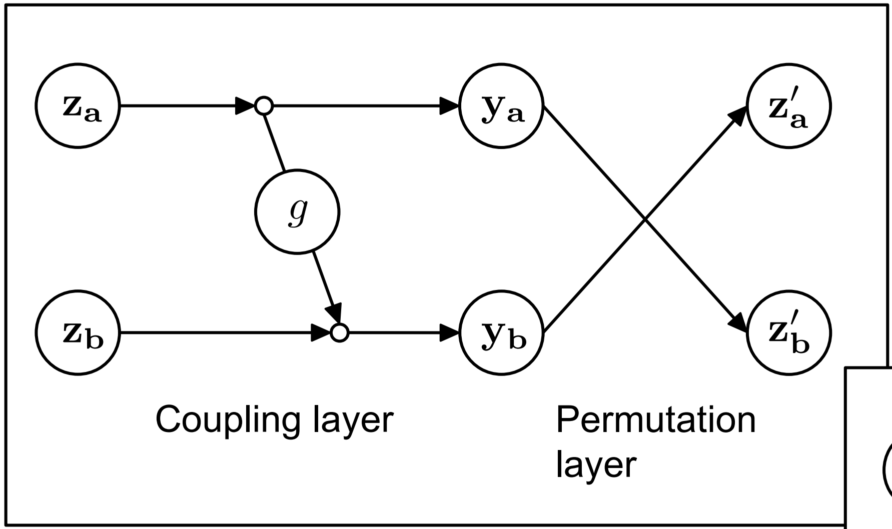
$$= \mathrm{const.} - E\underbrace{\left[\ln\left(p_{\mathcal{Z}}(f(\mathbf{y}))\right)\right]}_{\propto E\left[\|f(\mathbf{y})\|_2^2\right]} - E\underbrace{\left[\sum_{i=1}^{N}\ln\left(\left|\frac{\partial f_i(\mathbf{y}_i,\boldsymbol{\omega})}{\partial\mathbf{y}_i}\right|\right)\right]}_{E\left[\sum\limits_{i=1}^{N}\sum\limits_{j=d+1}^{D}s(\mathbf{z_a})_j\right]}$$

Second reason to choose $\exp(\cdot)$ for the scale in the affine transformation.

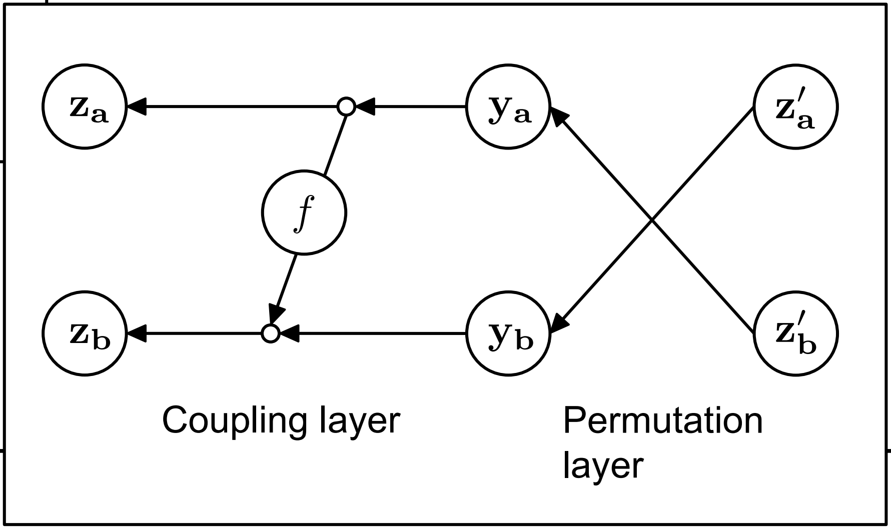
Permutation layer

- The coupling layer transforms only $\mathbf{z_b}$ and leaves $\mathbf{z_a}$ untouched.
- This issue can be easily addressed by a subsequent **permutation** layer.
- Since permuations are volume preserving their Jacobian determinant is $\equiv 1$.

Forward direction:



Normalizing direction:



arxiv:1808.04730 arxiv:1907.02392 arxiv:2003.06281

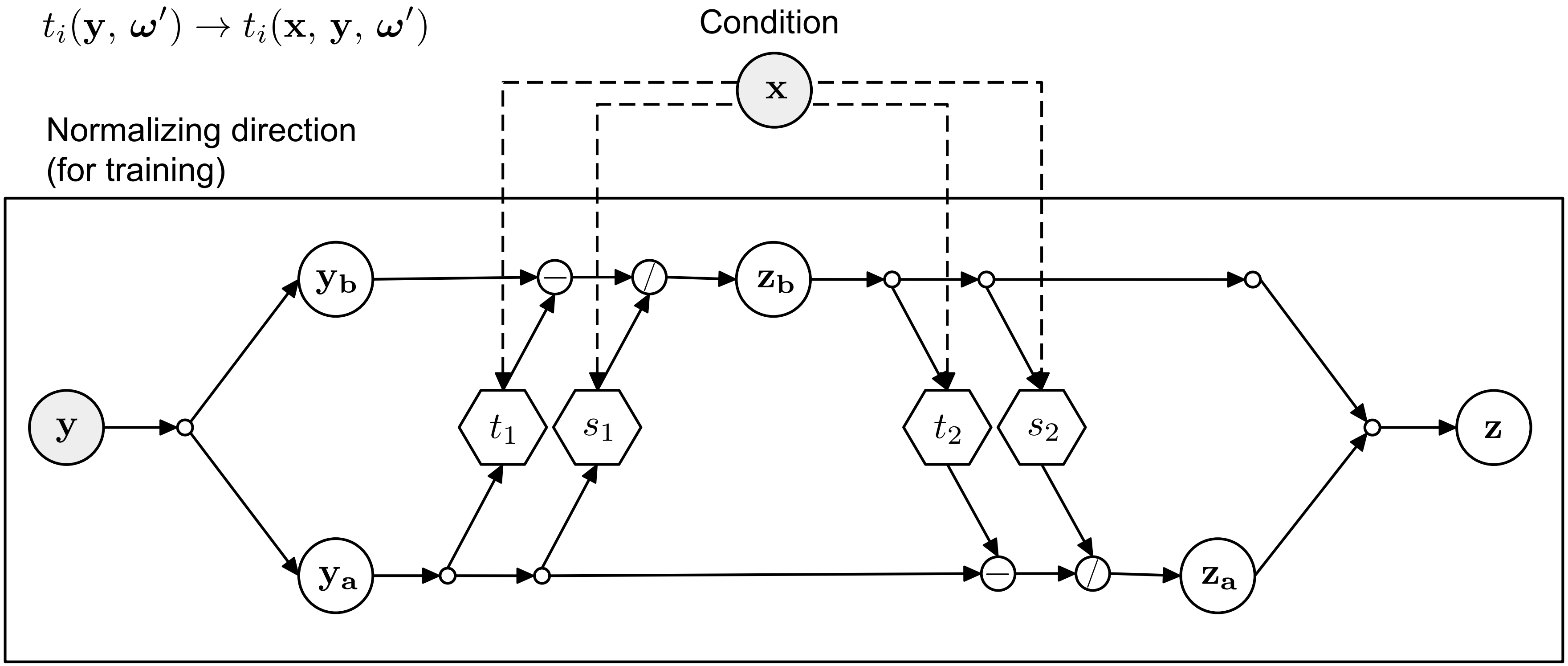
Conditional invertable NN (cINN)

- Assume $(\mathbf{y},\,\mathbf{x})$ to be a pair of true (→$\mathbf{y}$) and observable (→$\mathbf{x}$) parameters from **simulation**.

Sample $(\mathbf{y},\,\mathbf{x})$ from simulation and augment $\mathbf{y}$ with $\mathbf{x}$.

$$s_i(\mathbf{y},\,\boldsymbol{\omega}\,) \rightarrow s_i(\mathbf{x},\,\mathbf{y},\,\boldsymbol{\omega}\,)$$
$$t_i(\mathbf{y},\,\boldsymbol{\omega}') \rightarrow t_i(\mathbf{x},\,\mathbf{y},\,\boldsymbol{\omega}')$$

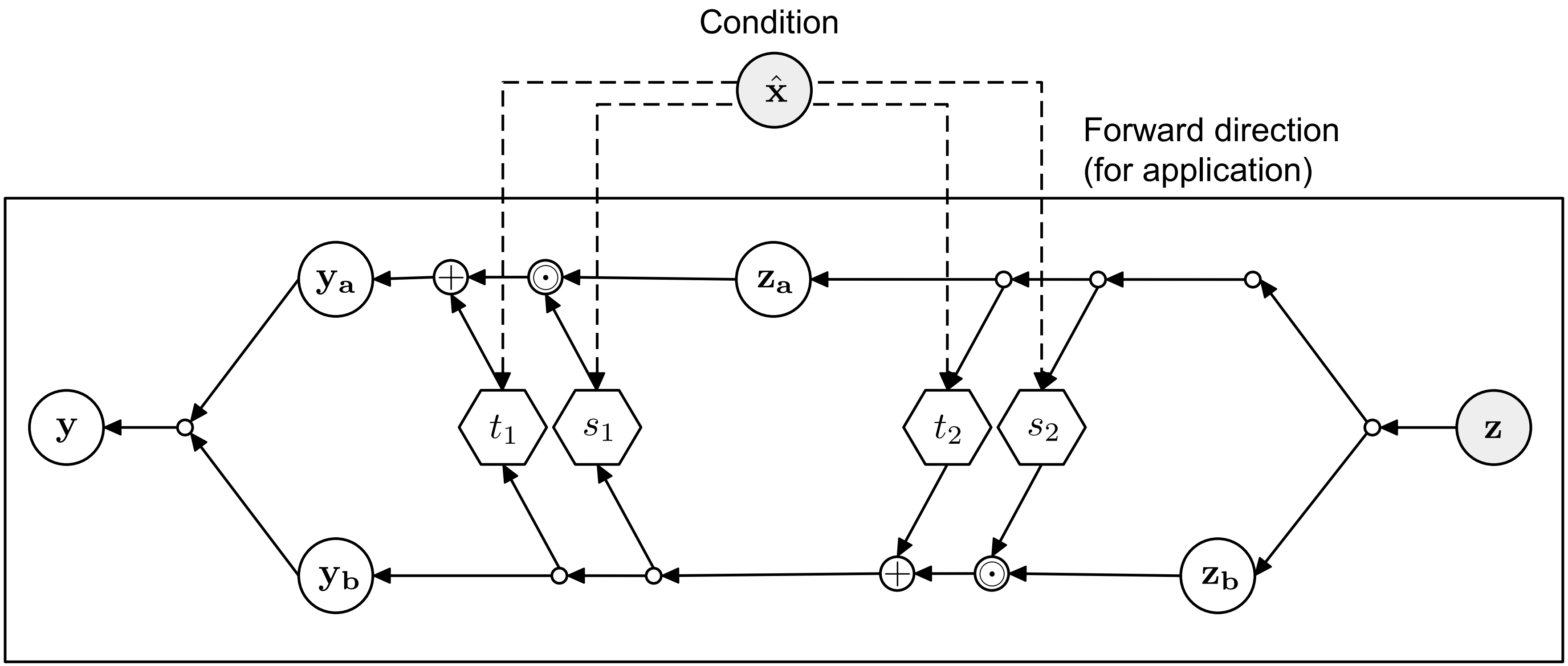
Normalizing direction
(for training)



Conditional invertable NN (cINN)

- Assume $(\mathbf{y},\,\mathbf{x})$ to be a pair of true (→$\mathbf{y}$) and observable (→$\mathbf{x}$) parameters from **simulation**.

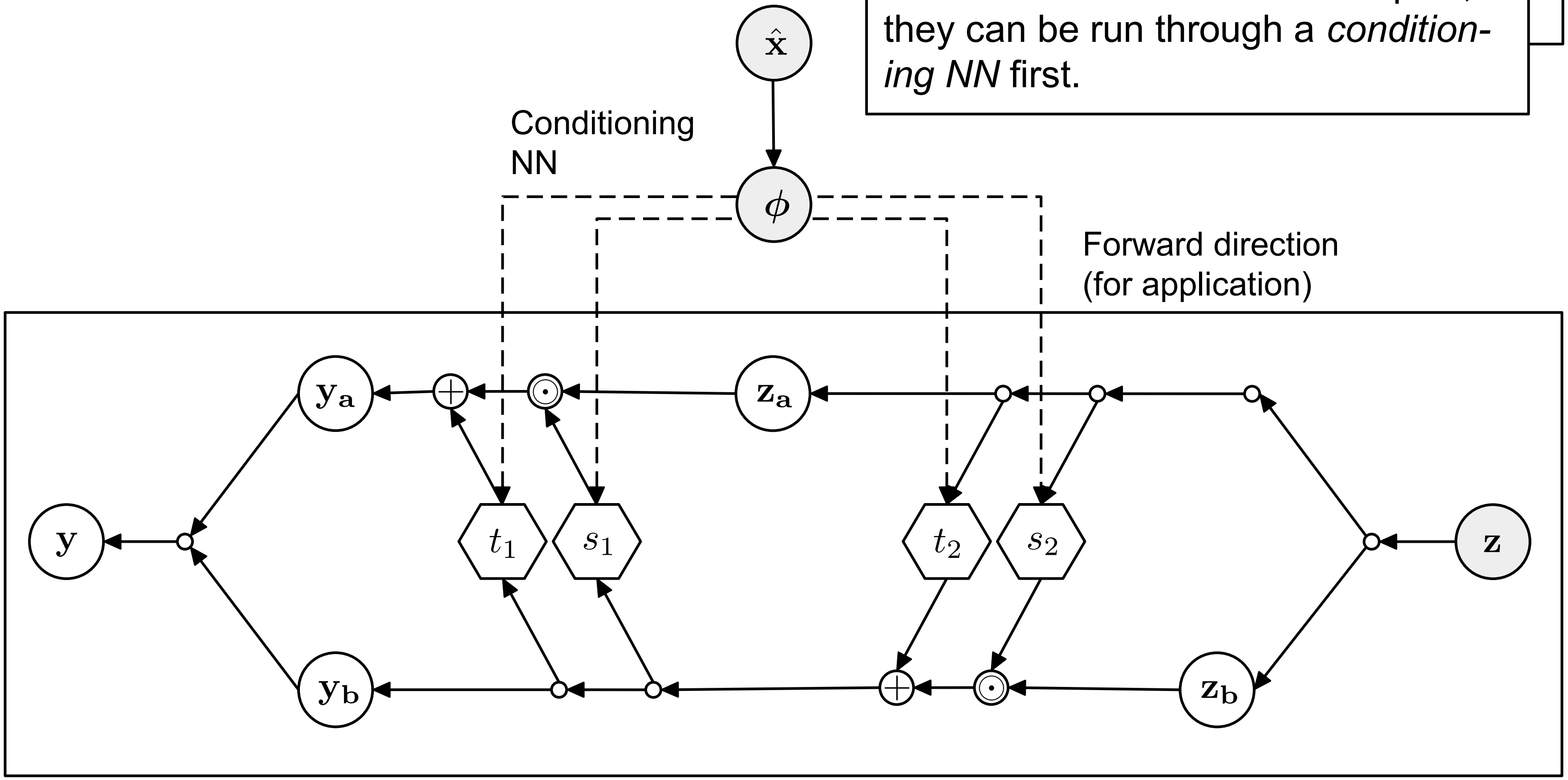
Sample $\mathbf{z}$ and augment with *measured* observables $\hat{\mathbf{x}}$.

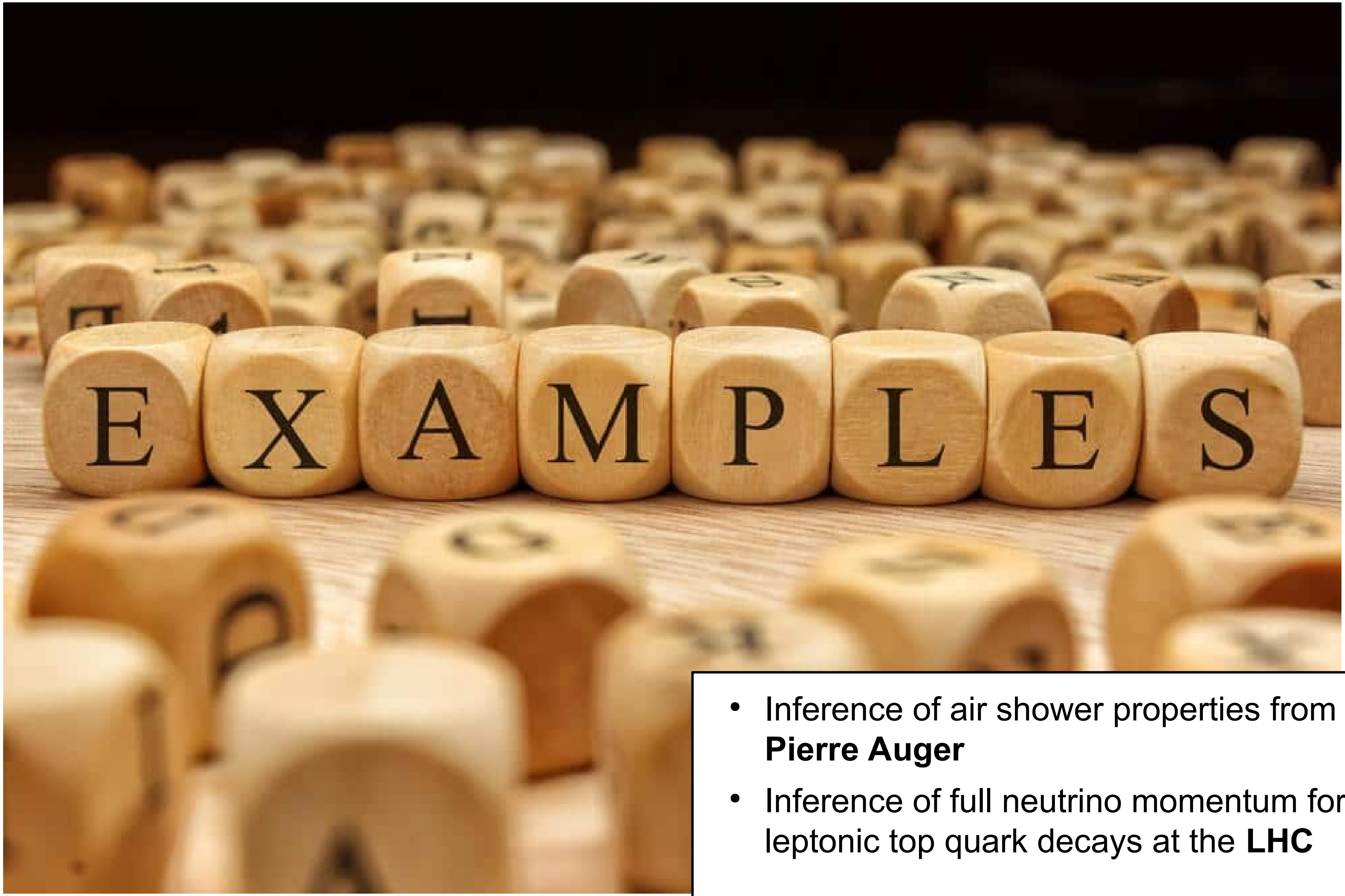


Conditional invertable NN (cINN)

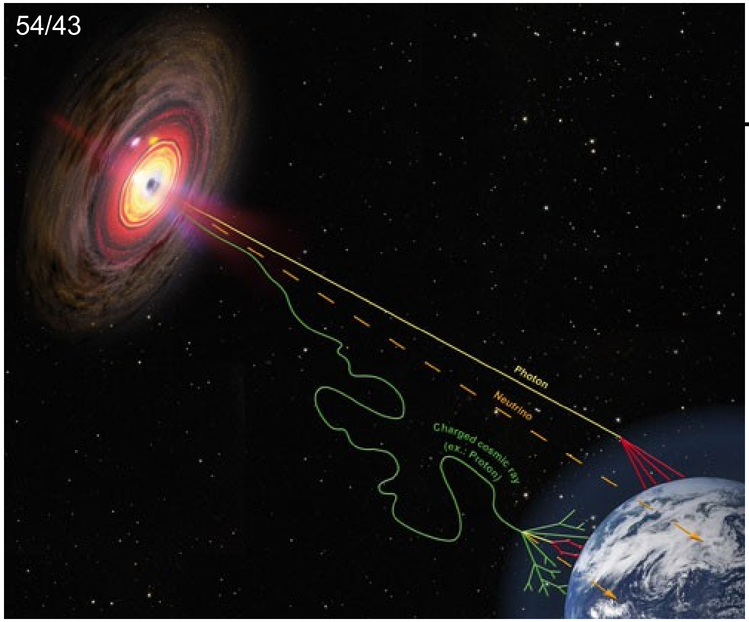
- Assume $(\mathbf{y},\, \mathbf{x})$ to be a pair of true (→$\mathbf{y}$) and observable (→$\mathbf{x}$) parameters from **simulation**.

If the conditions are more complex, they can be run through a *conditioning NN* first.





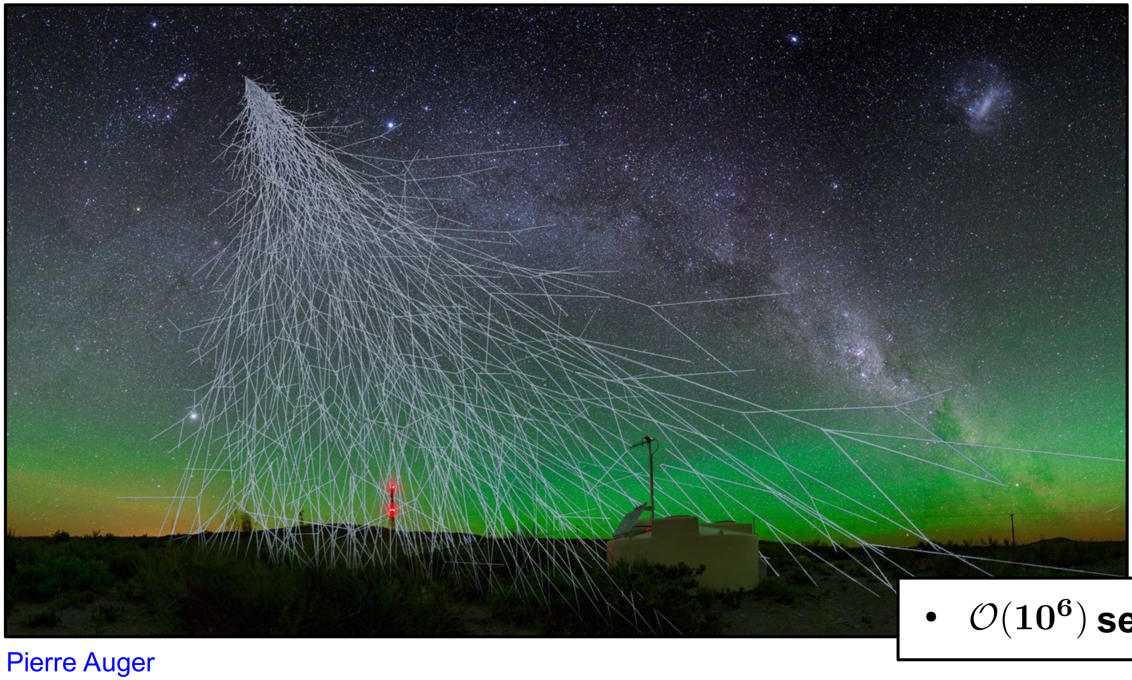
- Inference of air shower properties from **Pierre Auger**
- Inference of full neutrino momentum for leptonic top quark decays at the **LHC**



Inference with air showers

- When detected on Earth *charged* cosmic rays carry a rich convolution of information:
  - **original source**;
  - path through the universe;
  - detection environment on Earth.

Inference task



Pierre Auger

- $\mathcal{O}(10^6)$ **secondary particles**;

arxiv:2110.09493

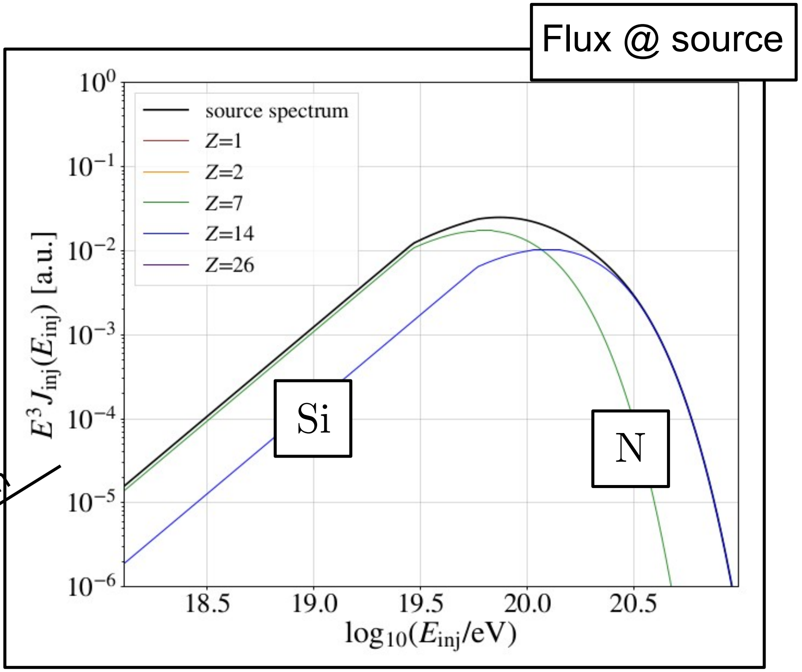
Data model

- Assumed flux and spectrum of **primary cosmic ray particles** at their cosmic source:

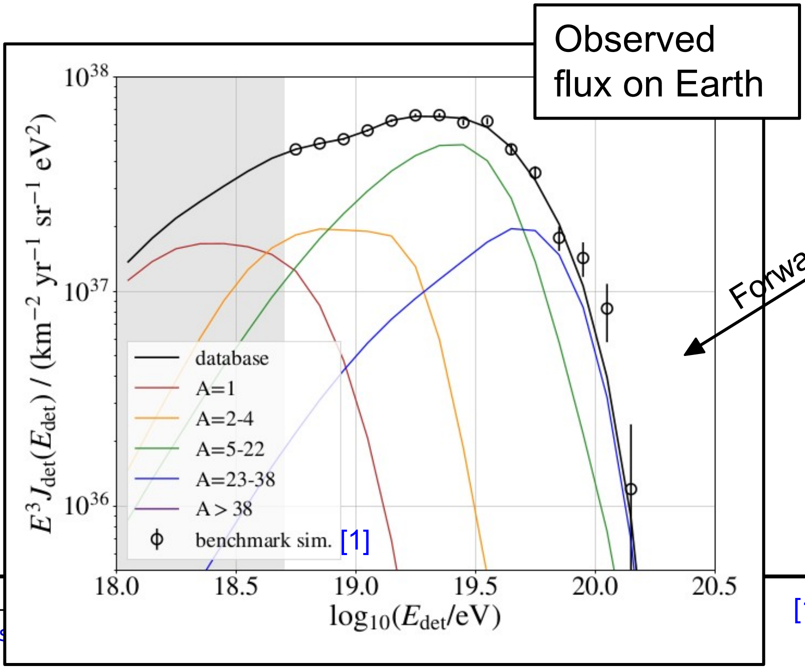
$$J_0(E_i, A_i) = J_0\, a(A_i) \left(\frac{E_i}{10^{18}\,\mathrm{eV}}\right)^{-\gamma} \begin{cases} 1 & Z_i\, R_{\mathrm{cut}} < E_i \\ \exp\left(1 - \frac{E_i}{Z_i\, R_{\mathrm{cut}}}\right) 1 & Z_i\, R_{\mathrm{cut}} \geq E_i \end{cases}$$

$$\begin{array}{lcl} \gamma & : & \text{Spectral index} \\ R_{\mathrm{cut}} & : & \text{Cutoff} \end{array}$$

$$\left.\begin{array}{lcl} E_i & : & \text{Energy} \\ A_i & : & \text{Mass number} \\ Z_i & : & \text{Charge number} \\ a(\cdot) & : & \text{Rel. abundancy} \end{array}\right\} \text{@ cosmic source}$$


Flux @ source

Forward simulation


Observed flux on Earth

[1] Pseudo data with statistical power as expected by Pierre Auger.

Inference task

Task: infer $\mathbf{y} = \{\gamma,\, R_{\mathrm{cut}},\, a(\mathrm{H}),\, a(\mathrm{He}),\, a(\mathrm{N}),\, a(\mathrm{Si}),\, a(\mathrm{Fe})\}$ from the observable air shower properties $\mathbf{x}$ on Earth (→ 7D value space).

- Propagation DB from forward simulation with varied assumptions for $\mathbf{y}$.
- Vary $\mathbf{y}$ until $p(\mathbf{x}|\mathbf{y})\,p(\mathbf{y})$ matches the observation $\hat{\mathbf{x}}$ (of the pseudo data).
- **Traditional approach**:
  - Estimate $p(\mathbf{y}|\hat{\mathbf{x}})$ with the help of Markov Chain Monte Carlo (MCMC).
  - 4–6 h per Markov chain.

Inference task

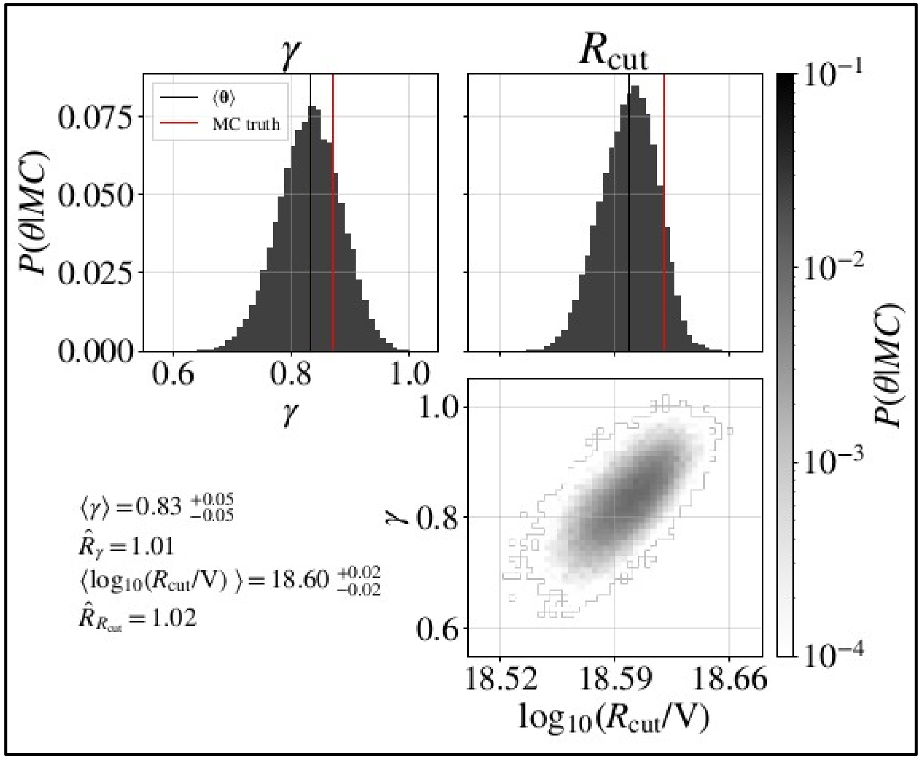
Task: infer $\mathbf{y} = \{\gamma,\, R_{\mathrm{cut}},\, a(\mathrm{H}),\, a(\mathrm{He}),\, a(\mathrm{N}),\, a(\mathrm{Si}),\, a(\mathrm{Fe})\}$ from the observable air shower properties $\mathbf{x}$ on Earth (→ 7D value space).

- Propagation DB from forward simulation with varied assumptions for $\mathbf{y}$.
- Vary $\mathbf{y}$ until $p(\mathbf{x}|\mathbf{y})\,p(\mathbf{y})$ matches the observation $\hat{\mathbf{x}}$ (of the pseudo data).
- **cINN approach**:
  - Using 6 cINN layers connected via the GLOW approach[1];
  - $s_i(\cdot)$ and $t_i(\cdot)$ chosen as NNs with 3 layers of width 256 and ReLU activation each;
  - Augmented with 420 observables $\mathbf{x}$ (counts in bins of shower energy and shower maxima);
  - Training (on 1M samples) 30h (on single GPU), evaluation O(sec).

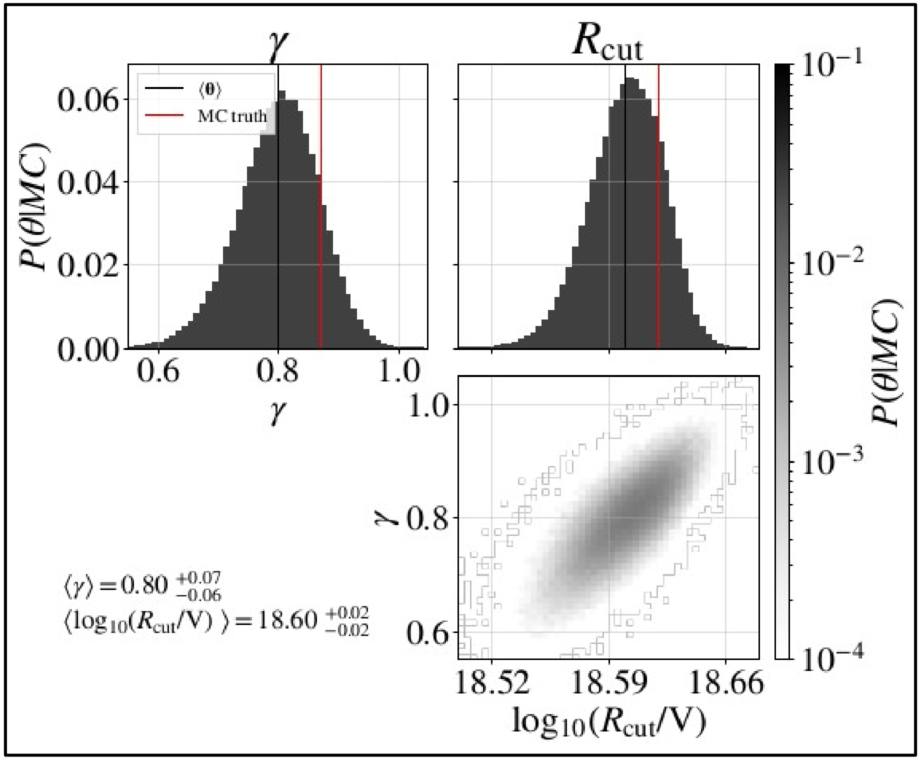
[1] arxiv:1807.03039

MCMC vs. cINN

MCMC

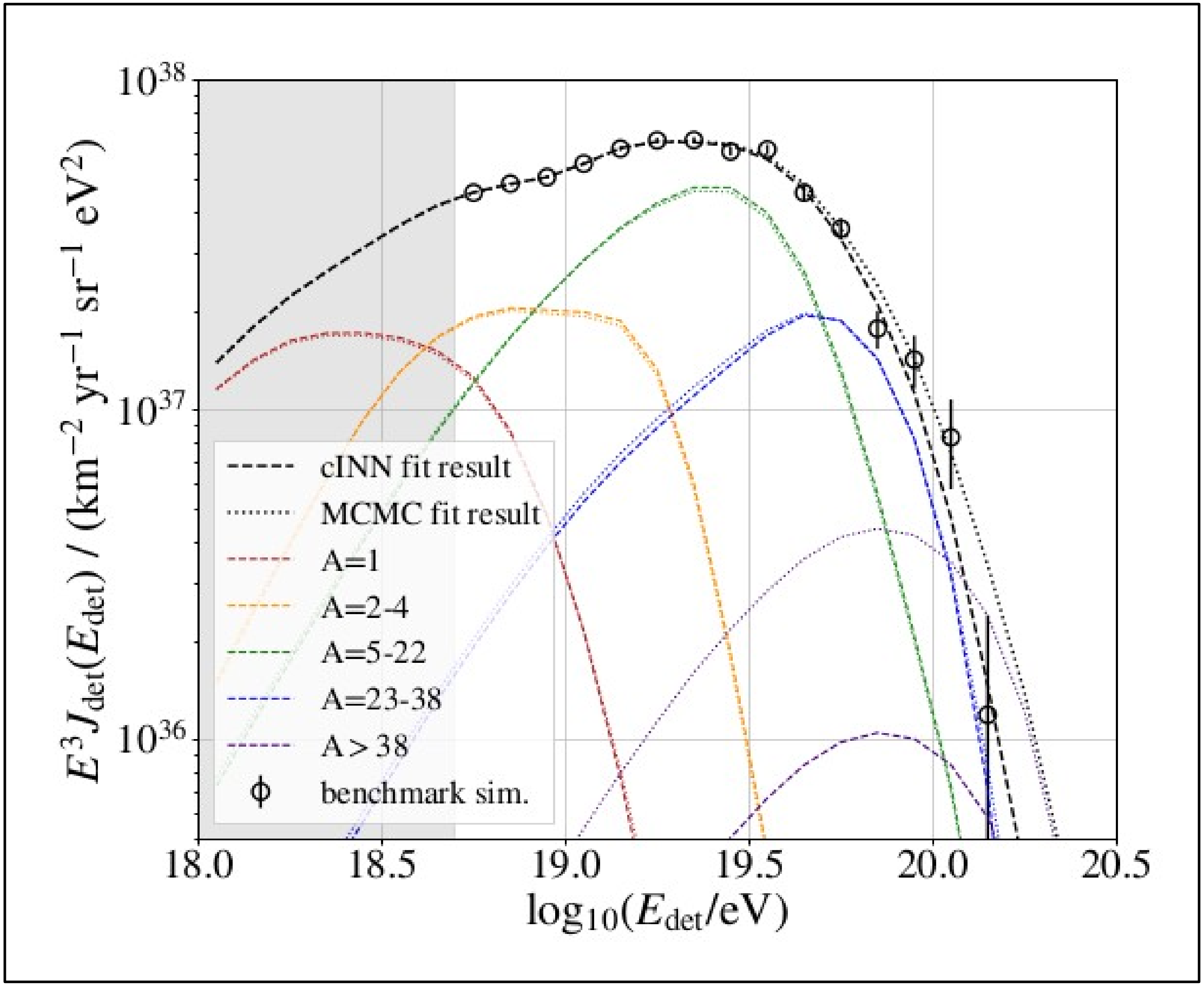


cINN



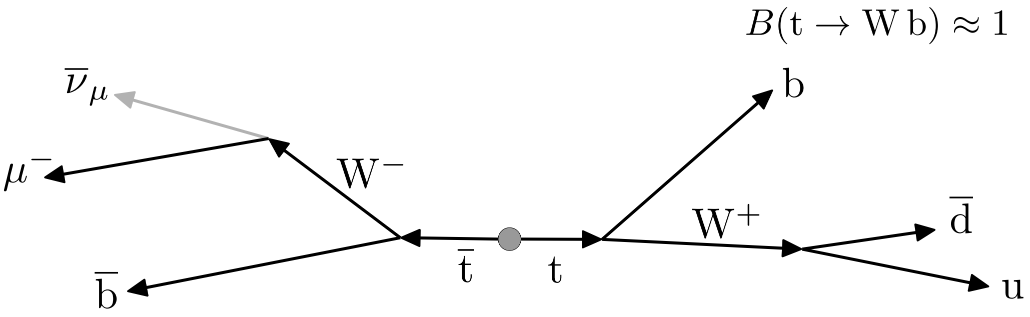
Red line is the truth of the pseudo data.

MCMC vs. cINN


10^{38}
10^{37}
10^{36}
$E^3 J_{\mathrm{det}}(E_{\mathrm{det}})$ / (km^{-2} yr^{-1} sr^{-1} eV^2)
cINN fit result
MCMC fit result
A=1
A=2-4
A=5-22
A=23-38
A > 38
benchmark sim.
18.0
18.5
19.0
19.5
20.0
20.5
$\log_{10}(E_{\mathrm{det}}/\mathrm{eV})$

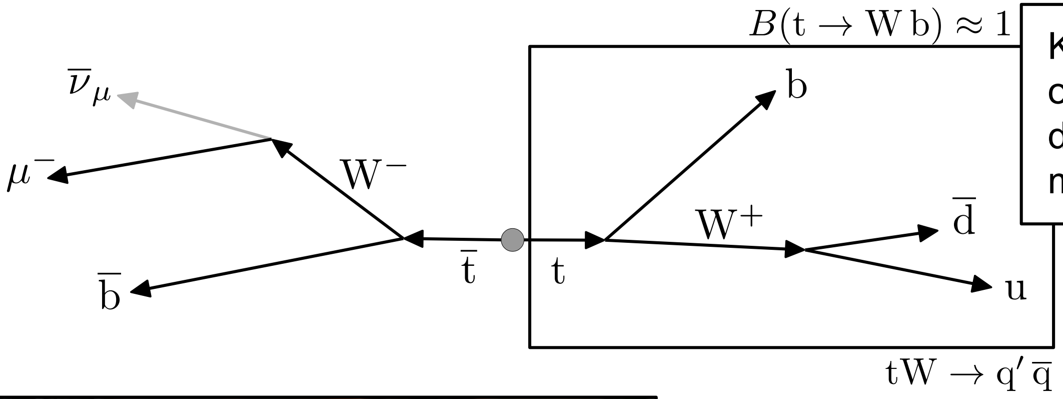
Top quark pair production at the LHC

- The CERN LHC is a top quark pair factory.



Top quark pair production at the LHC

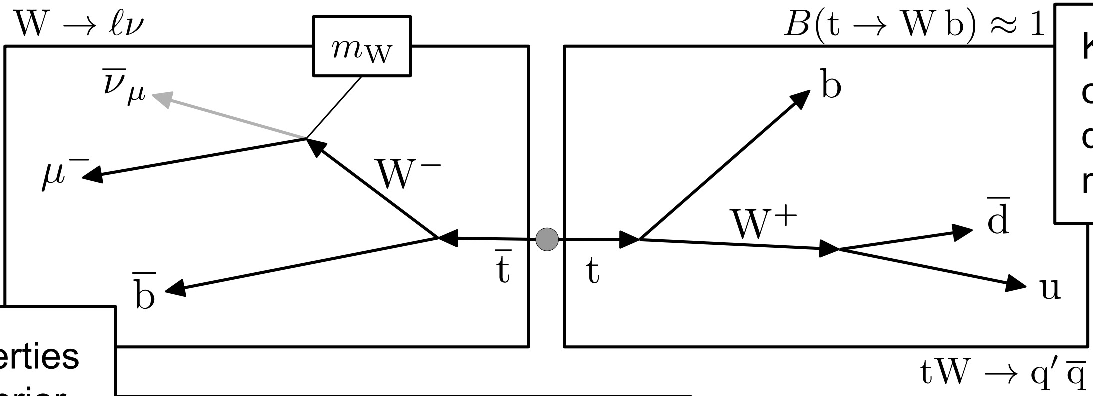
- The CERN LHC is a top quark pair factory.



Kinematic properties of the decay fully determined by measurement.

Top quark pair production at the LHC

- The CERN LHC is a top quark pair factory.

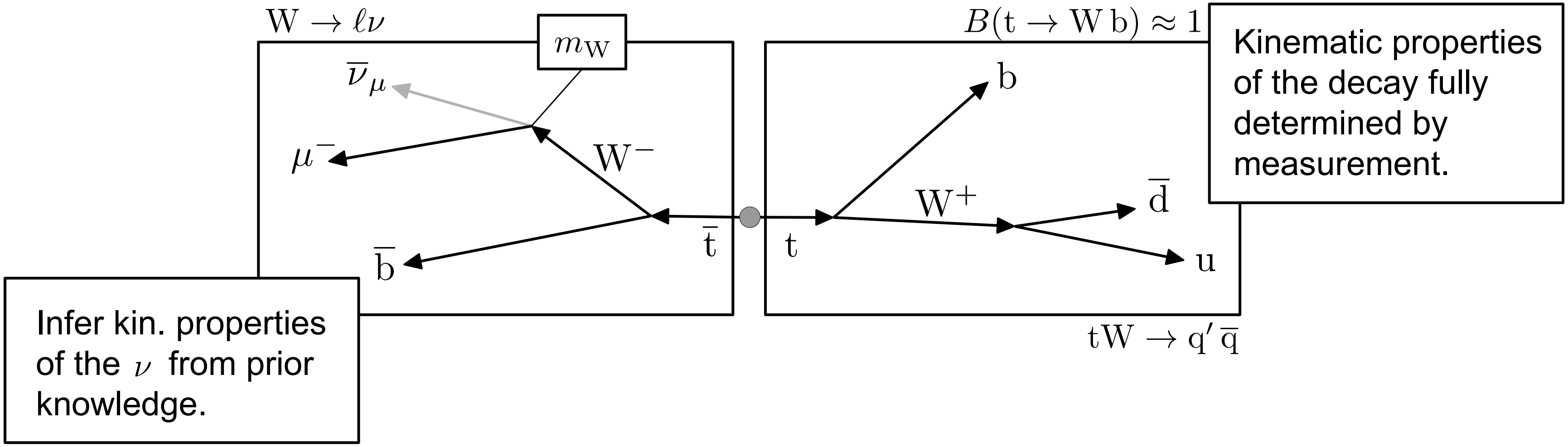


Kinematic properties of the decay fully determined by measurement.

Infer kin. properties of the ν from prior knowledge.

Top quark pair production at the LHC

- The CERN LHC is a top quark pair factory.



Kinematic properties of the decay fully determined by measurement.

Infer kin. properties of the ν from prior knowledge.

$$p_z^\nu = \frac{-b \pm \sqrt{b^2 - 4\,a\,c}}{2\,a}$$

with:

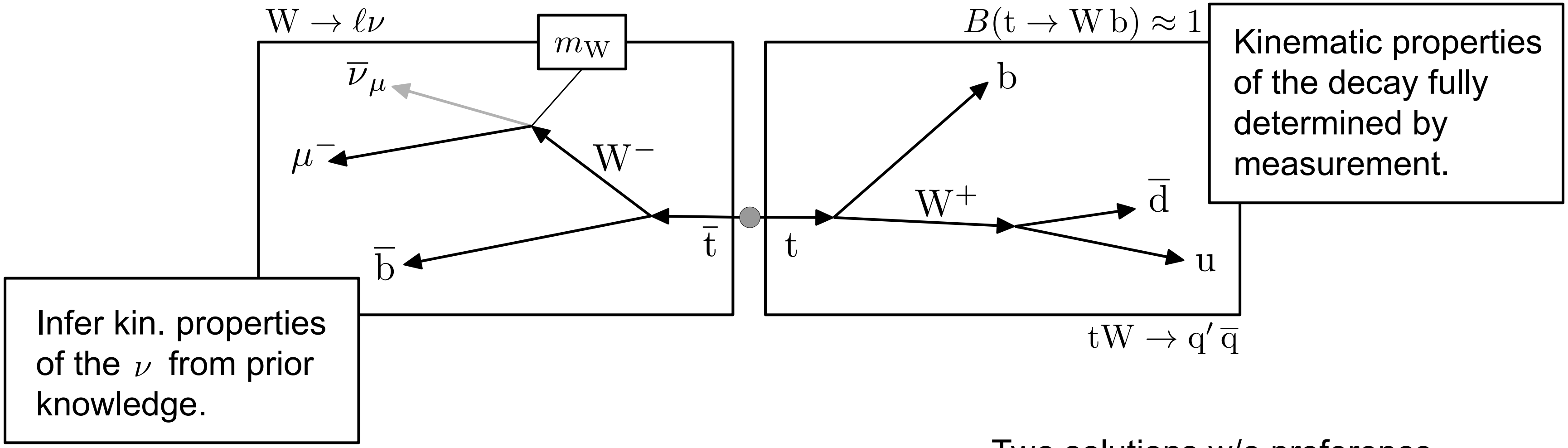
$$a = p_z^{\ell\,2} - E^{\ell\,2} \qquad \vec{p}_\mathrm{T}^{\,\nu} = -\vec{p}_\mathrm{T}^{\,\mathrm{miss}}$$

$$b = \kappa\,p_z^\ell \qquad m_\mathrm{W} = 80.38\,\mathrm{GeV}$$

$$c = \left(\frac{\kappa}{2}\right)^2 - E^{\ell\,2} p_\mathrm{T}^{\nu\,2} \qquad \kappa = m_\mathrm{W}^2 - m_\ell^2 - 2\vec{p}_\mathrm{T}^{\,\ell}\vec{p}_\mathrm{T}^{\,\nu}$$

Top quark pair production at the LHC

- The CERN LHC is a top quark pair factory.



Kinematic properties of the decay fully determined by measurement.

Infer kin. properties of the ν from prior knowledge.

$$\boxed{p_{z}^{\nu} = \frac{-b \pm \sqrt{b^{2} - 4\,a\,c}}{2\,a}}$$

Two solutions w/o preference. Depending on exp. resolution cases with no real solution.

with:

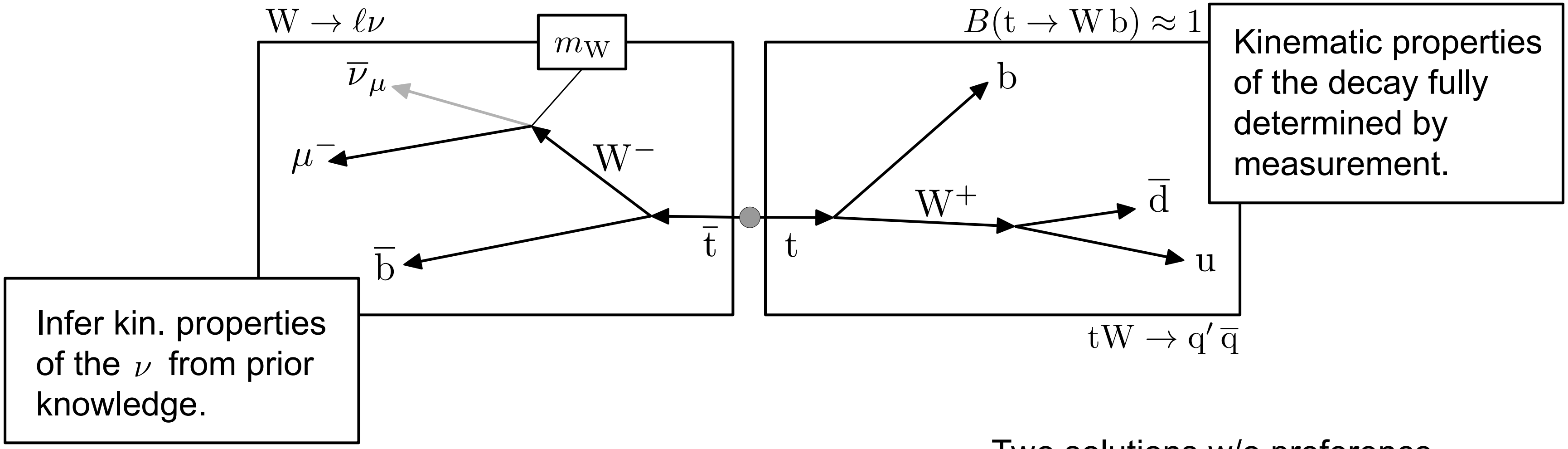
$$a = p_{z}^{\ell\,2} - E^{\ell\,2} \qquad \vec{p}_{\mathrm{T}}^{\,\nu} = -\vec{p}_{\mathrm{T}}^{\,\mathrm{miss}}$$

$$b = \kappa\, p_{z}^{\ell} \qquad m_{\mathrm{W}} = 80.38\,\mathrm{GeV}$$

$$c = \left(\frac{\kappa}{2}\right)^{2} - E^{\ell\,2} p_{\mathrm{T}}^{\nu\,2} \qquad \kappa = m_{\mathrm{W}}^{2} - m_{\ell}^{2} - 2\vec{p}_{\mathrm{T}}^{\,\ell}\vec{p}_{\mathrm{T}}^{\,\nu}$$

Top quark pair production at the LHC

- The CERN LHC is a top quark pair factory.



Kinematic properties of the decay fully determined by measurement.

Infer kin. properties of the ν from prior knowledge.

$$\boxed{p_z^\nu = \frac{-b \pm \sqrt{b^2 - 4\,a\,c}}{2\,a}}$$

Two solutions w/o preference. Depending on exp. resolution cases with no real solution.

with:

$$a = p_z^{\ell\,2} - E^{\ell\,2} \qquad \vec{p}_\mathrm{T}^{\,\nu} = -\vec{p}_\mathrm{T}^{\,\mathrm{miss}}$$

$$b = \kappa\, p_z^\ell \qquad \boxed{m_\mathrm{W} = 80.38\,\mathrm{GeV}}$$

Bias in view of experimental resolution.

$$c = \left(\frac{\kappa}{2}\right)^2 - E^{\ell\,2} p_\mathrm{T}^{\nu\,2} \qquad \kappa = m_\mathrm{W}^2 - m_\ell^2 - 2\vec{p}_\mathrm{T}^{\,\ell}\vec{p}_\mathrm{T}^{\,\nu}$$

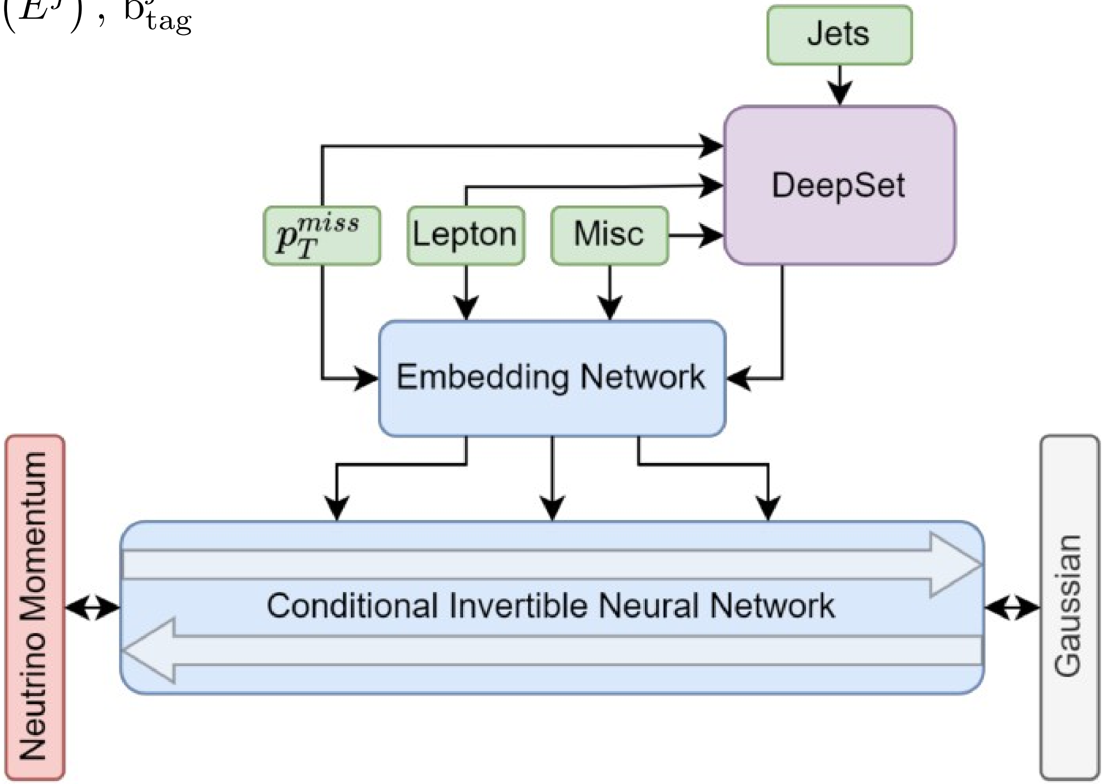
cINN approach

- Conditioning observables ($\mathbf{x}$).

$p_{\mathrm{T}}^{\mathrm{miss}}$	: p_x^{miss}, p_y^{miss}
Lepton (ℓ)	: p_x^{ℓ}, p_y^{ℓ}, η^{ℓ}, $\log\left(E^{\ell}\right)$, ID [2]
Misc	: N_{Jet}, $N_{\mathrm{b\ Jet}}$, C_1, C_2 [1]
Uo to 10 jets	: p_x^{j}, p_y^{j}, η^{j}, $\log\left(E^{j}\right)$, $\mathrm{b}_{\mathrm{tag}}^{j}$

- Targets ($\mathbf{y}$).

p_x^{ν}, p_y^{ν}, η^{ν}

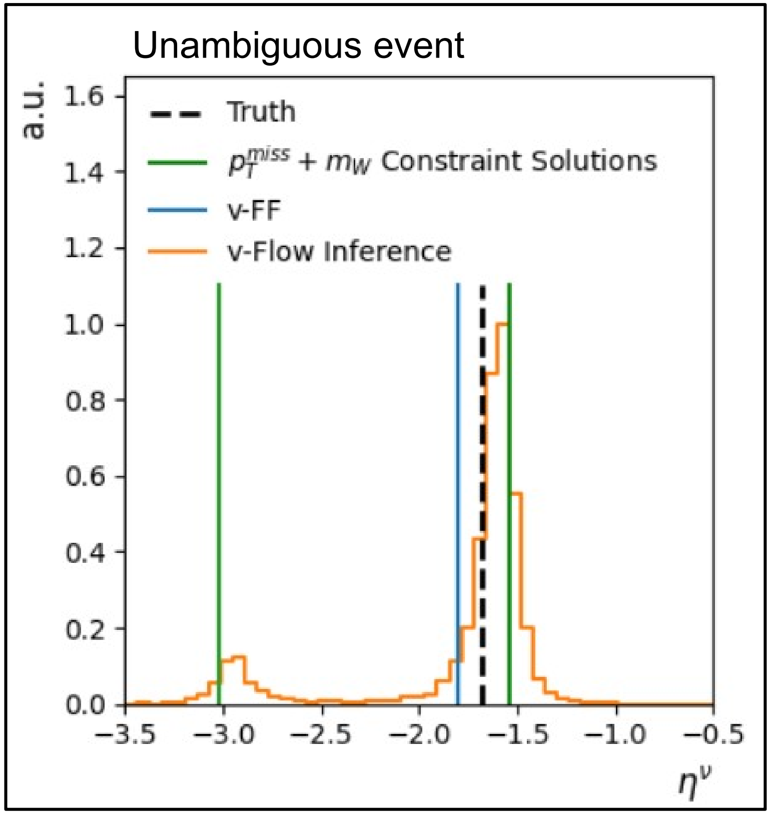


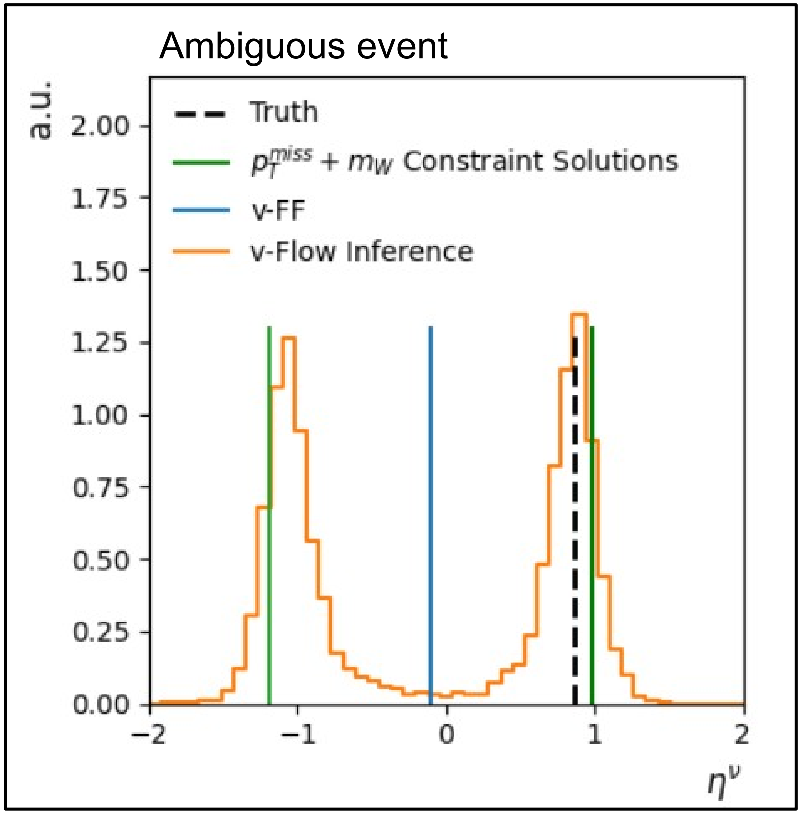
[2] η: Pseudorapidity

[1] $C_1 = \frac{-b}{2\,a}$; $C_2 = \frac{b^2 - 4\,a\,c}{2\,a}$ from previous slide

Comparison of inference methods

- Individual case studies (▬ ν-FF: feed-forward NN , ▬ ν-Flow: norm.-flow model):


Unambiguous event

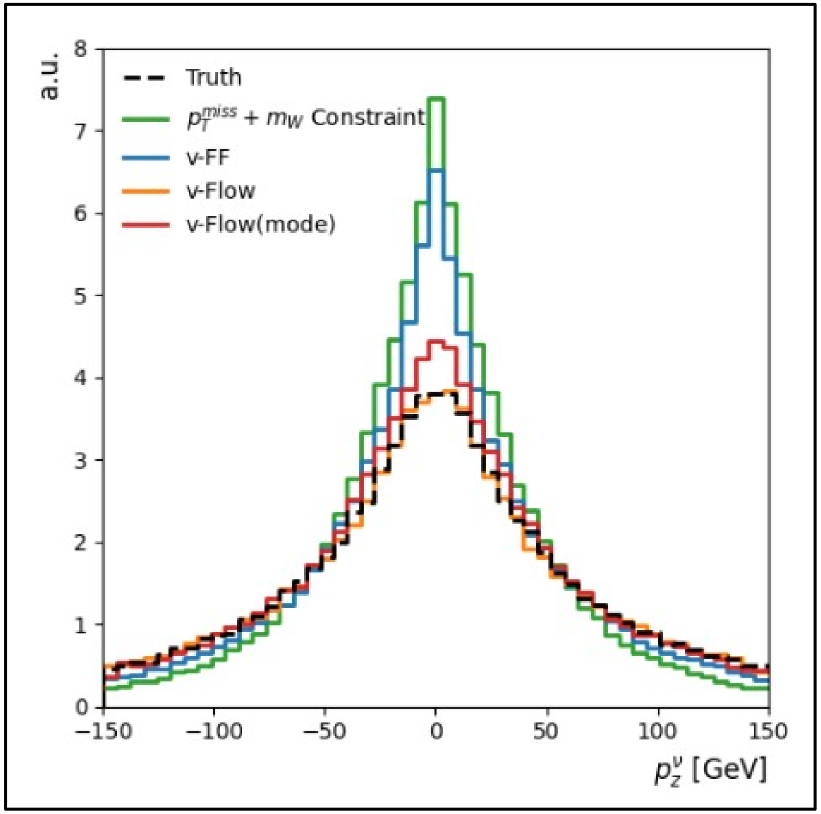

Ambiguous event

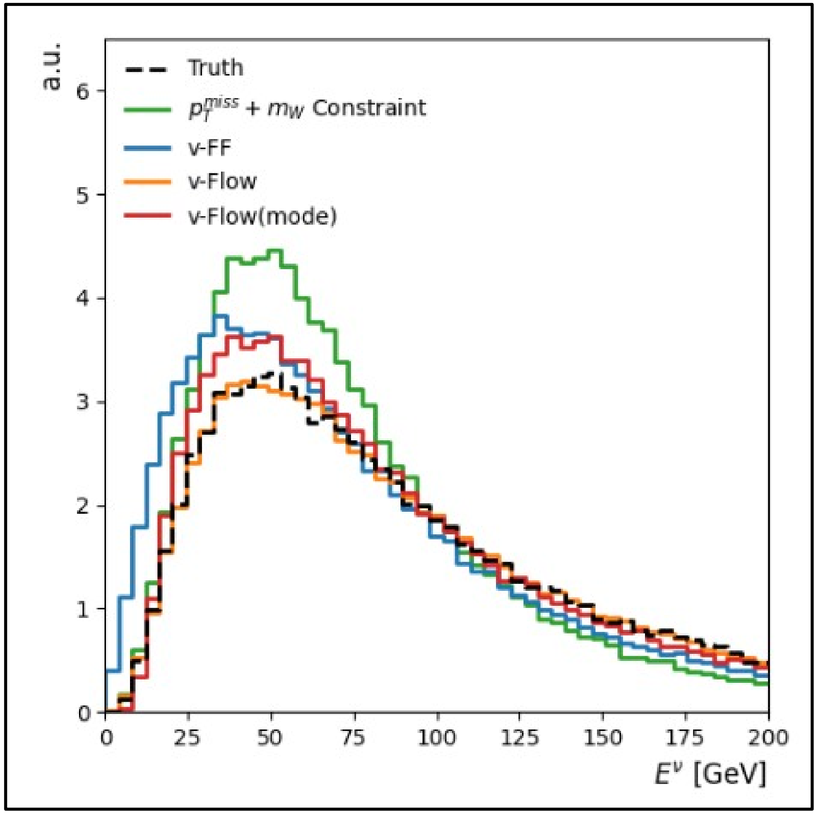
NN-based inference models are able to identify the correct solution (w/ high probability).

Flow-based inference model provides equal spread of probability where feed-forward NN „fails“.

Comparison of inference methods

- Ensemble study (🟦 ν-FF: feed-forward NN, 🟧 ν-Flow: norm.-flow model, ignore the red):





- Best reproduction of kinematic ν-properties by flow-based model.

Summary

- Normalizing-flow models are very interesting and promising for our field.

- They are mathematically clear, with many good properties in turn, and easy to understand.

- Most prominent features:
  - Conservation of probability;
  - Lossless compression;
  - Applicability for unfolding.

- Most obvious and useful applications (presented here with two very good examples from Pierre Auger and LHC), both based on classical Monte Carlo techniques for training and exploiting cINNs:
  - Sampling from untractable likelihoods/posteriors;
  - Regularized unfolding („likelihood-free inference“).

Literature

- **Literature you can use to get an overiew of the matter:**
  - J. M. Tomczak *Deep generative modeling* (Springer 2022).
  - I. Kobyev et al, *Normalizing flows: An introduction and review of current methods* (arxiv:1908.09257).
  - U. Koethe, *Solving inverse problems with invertable neural networks*, (4th IML Workshop, CERN 2020).
  - Literature referred to on the slides.

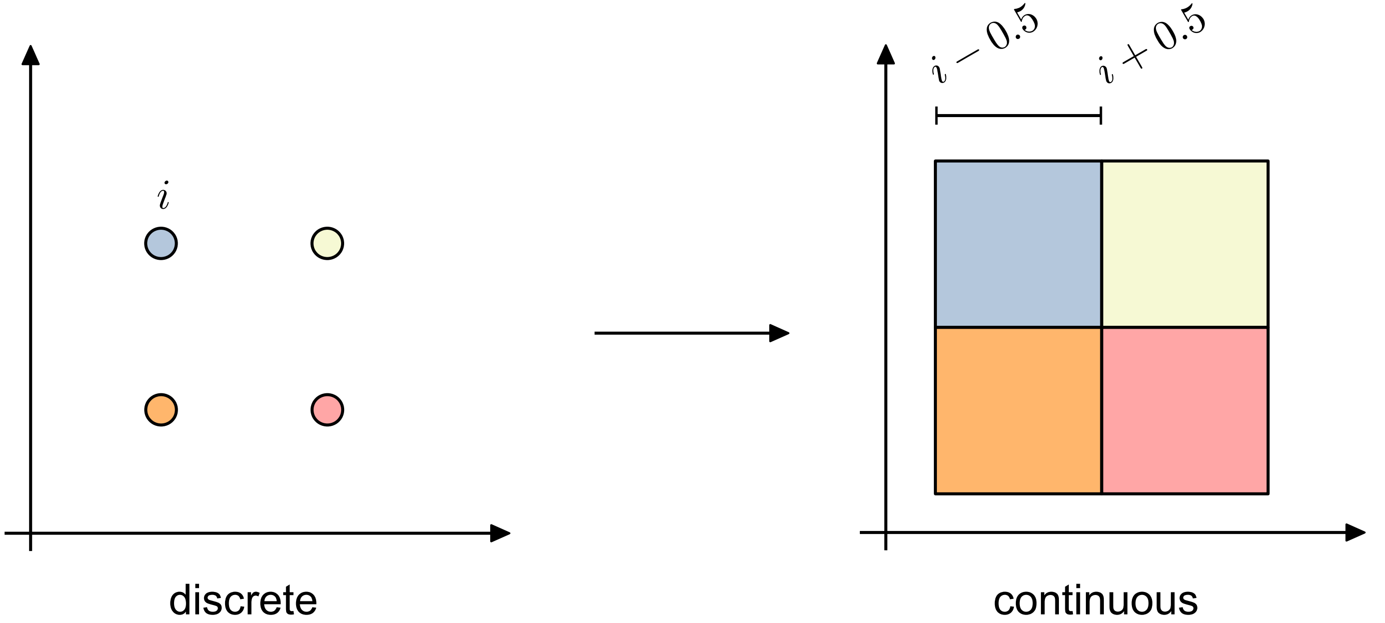
Backup

Discrete inputs

- What has been discussed so far, has been with **real-valued inputs** in mind.
- *Discrete* can be transformed into *real-valued inputs* by adding uniform random noise.
- The following example is given for integer-valued inputs:

For $x_i \in [1,2,3,\ldots,I]$ and $u \in [-0.5;0.5]$ apply:

$$[1,2,3,\ldots,I] \rightarrow [0.5; I+0.5]: \quad x_i \rightarrow x_i' = x_i + u$$


i
discrete
$i-0.5$
$i+0.5$
continuous