Free-form flows for physics-informed generative modeling

Ullrich Köthe
Computer Vision and Learning Lab, Heidelberg University

October 2024

with: Lynton Ardizzone, Stefan Radev, Peter Sorrenson, Felix Draxler,
Armand Rousselot, Carsten Rother, Sander Hummerich and many others

The Zoo of Deep Generative Models

Goal: learn complex probability densities

$$p(X)$$

or conditional densities

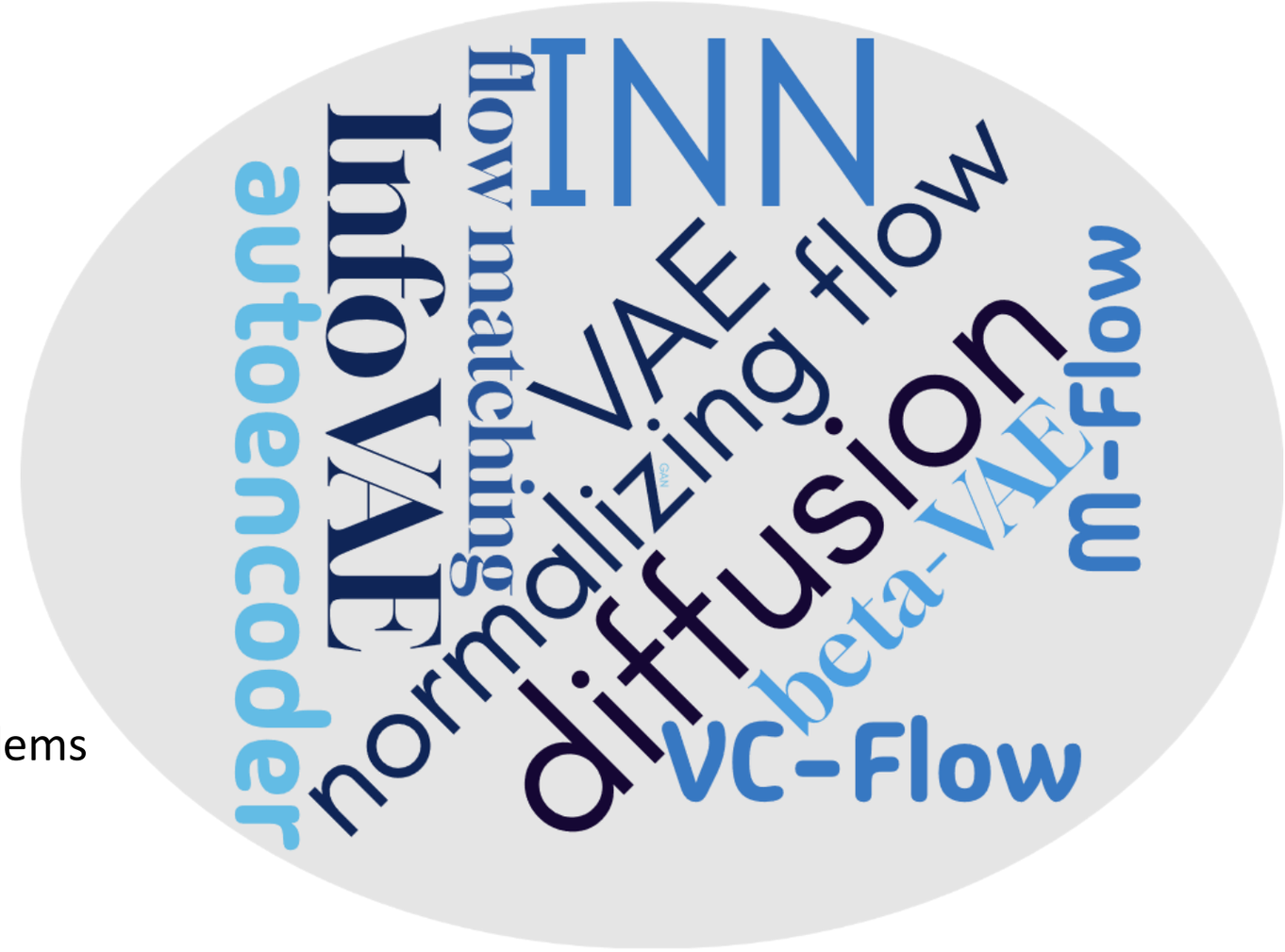
$$p(X \mid Y)$$

from training sets

$$\{x_i\}_{i=1}^N \quad \text{or} \quad \{(x_i, y_i)\}_{i=1}^N$$

- use trained model to
  - sample synthetic data $x \sim \hat{p}(X)$
  - calculate probabilities $\hat{p}(X = x)$
  - utilize these in downstream tasks, e.g. Bayesian inference, inverse problems

- many competing models have been proposed

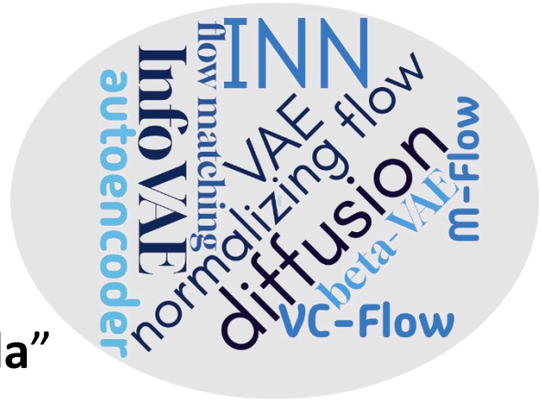


The Zoo of Deep Generative Models

How to tame the mess?

- All use some form of inverse transform sampling:
  $x \sim \hat{p}(X) \quad \Leftrightarrow \quad z \sim q(Z)\ [= \mathcal{N}(Z \mid 0, \mathbb{I})], \quad x = \hat{g}(z)$
  sample form known latent prior, transform through learned decoder

⇨ **Categorize methods by** how target $\hat{p}(X)$ is expressed – "**change-of-variables formula**"

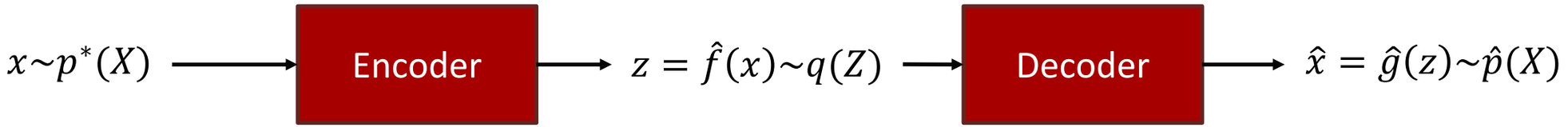


Normalizing flows

What is a „change-of-variables formula“?

Normalizing flows are the canonical realization:

- encoder-decoder architecture



- $\dim Z = \dim X$, $g(z)$ is bijective with inverse $\hat{f}(x) = \hat{g}^{-1}(x)$ (encoder)
- inverse realized by the same network running backwards
- change-of-variables formula:

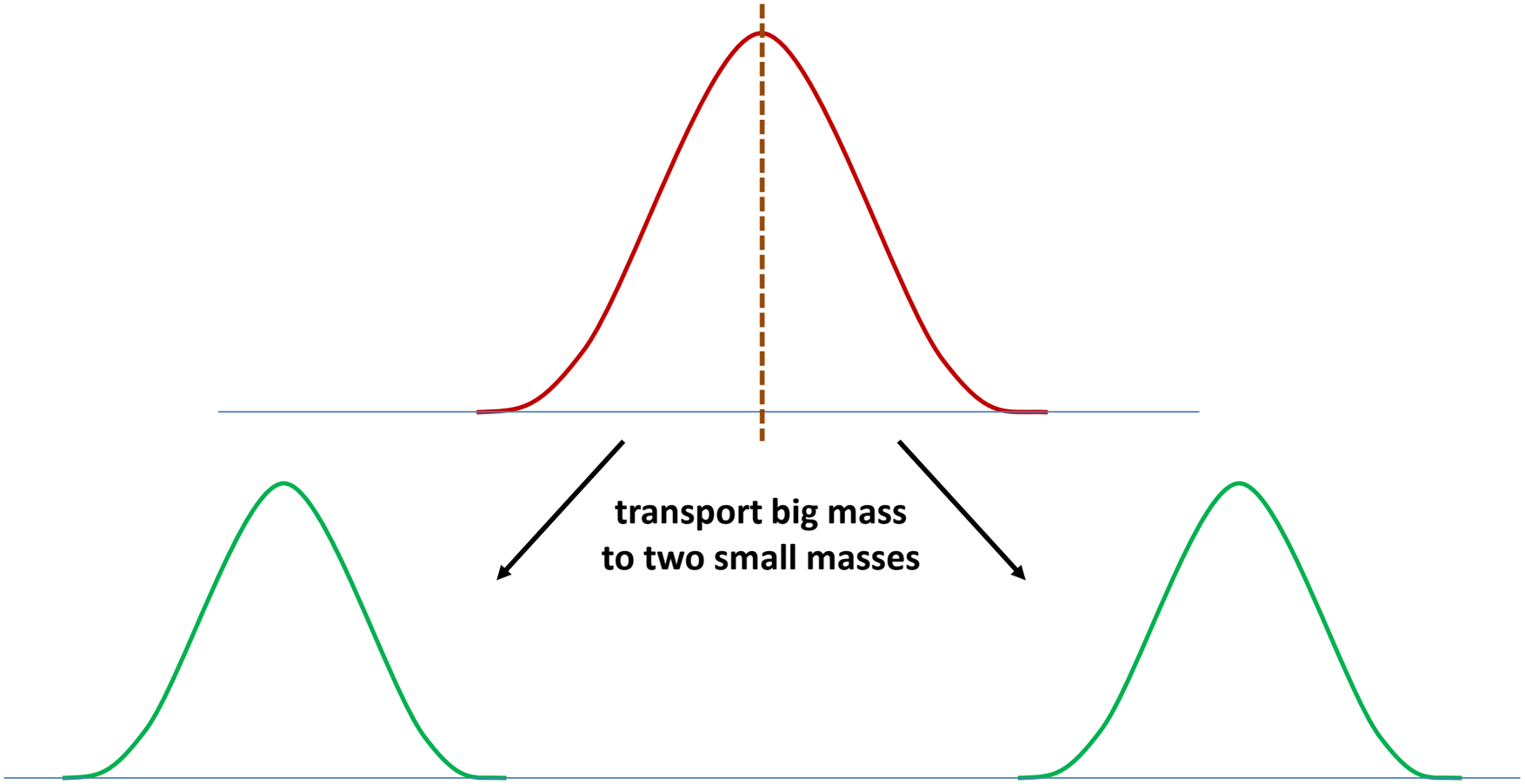
$$\hat{p}\big(X = \hat{g}(z)\big) = q(Z = z) \cdot \left|\det \frac{\partial \hat{g}(z)}{\partial z}\right|^{-1}$$

- example: inverse transformation sampling of the 1-D standard normal:
  latent prior: $q(Z) = \text{uniform}(0,1)$ decoder: $g(z) = \sqrt{2}\,\text{erf}^{-1}(2z - 1)$

Normalizing flows

Model complicated probabilities as bijective mappings of simple ones

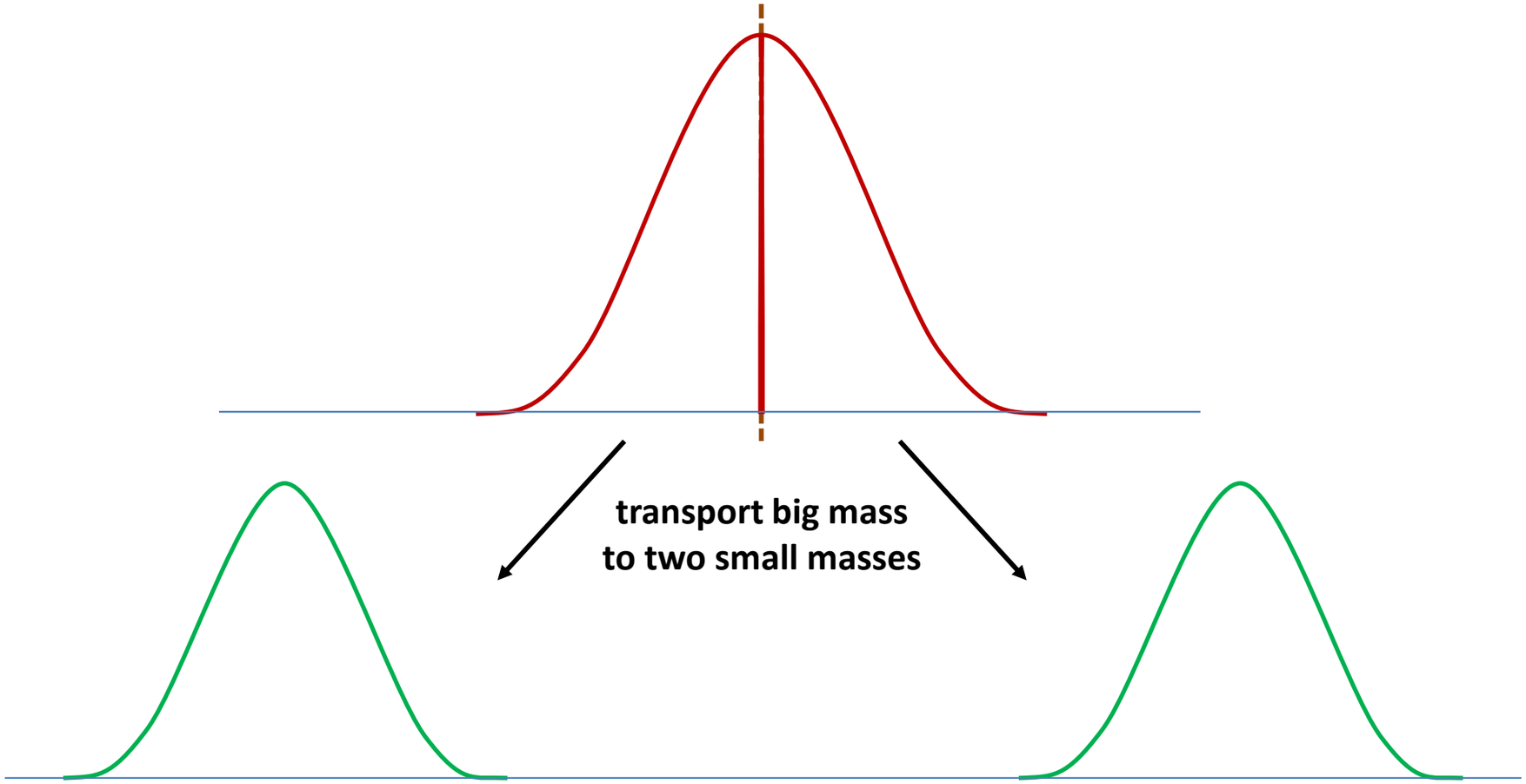
- Intuition: transport (“flow”) from simple “sand pile” to target „sand piles“



Normalizing flows

Model complicated probabilities as bijective mappings of simple ones

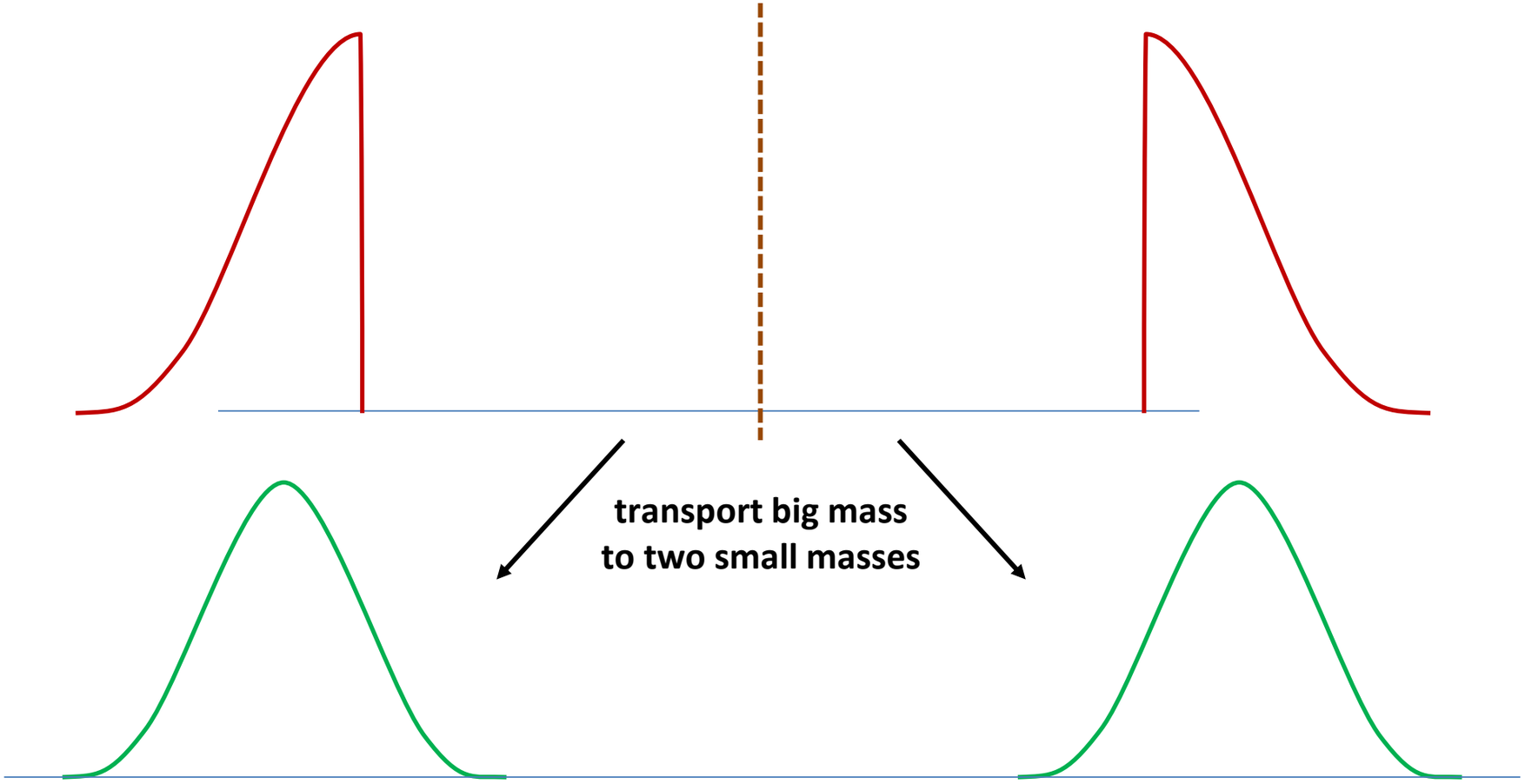
- Intuition: transport (“flow”) from simple “sand pile” to target „sand piles“



Normalizing flows

Model complicated probabilities as bijective mappings of simple ones

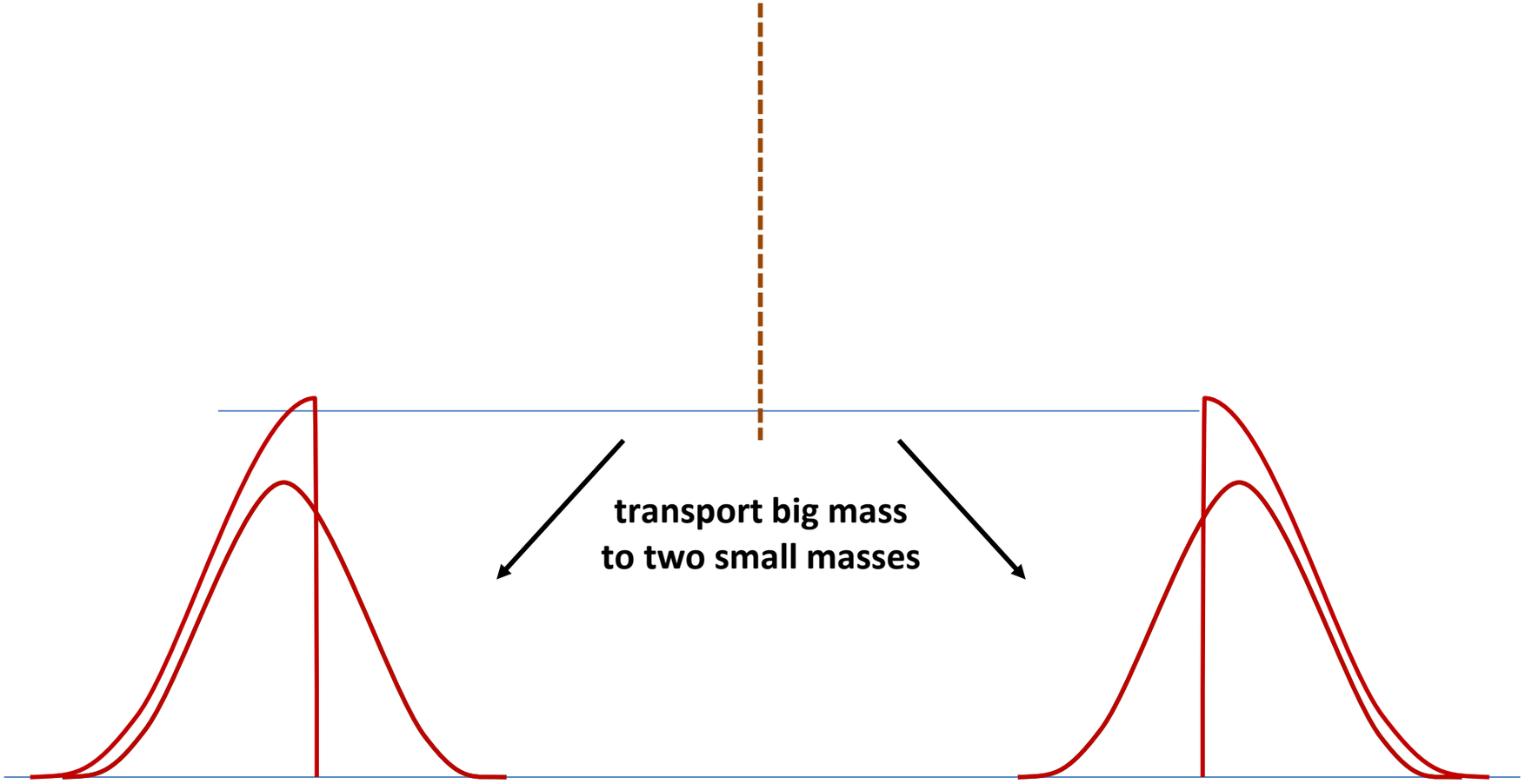
- Intuition: transport (“flow”) from simple “sand pile” to target „sand piles“



Normalizing flows

Model complicated probabilities as bijective mappings of simple ones

- Intuition: transport (“flow”) from simple “sand pile” to target „sand piles“

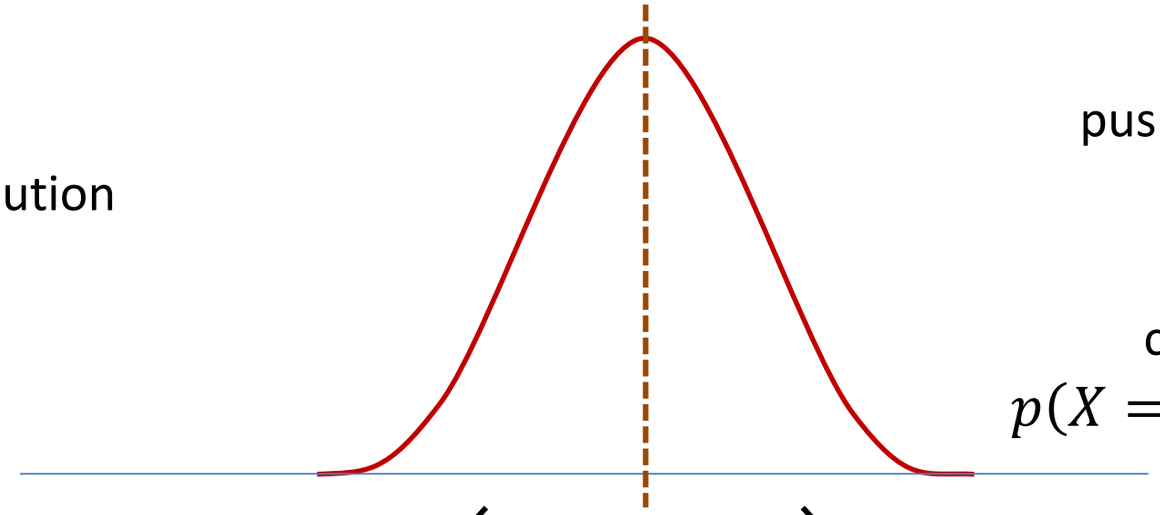


Normalizing flows

Model complicated probabilities as bijective mappings of simple ones

- Mathematically: target distribution is a push-forward of reference distribution

latent distribution
$q(Z)$

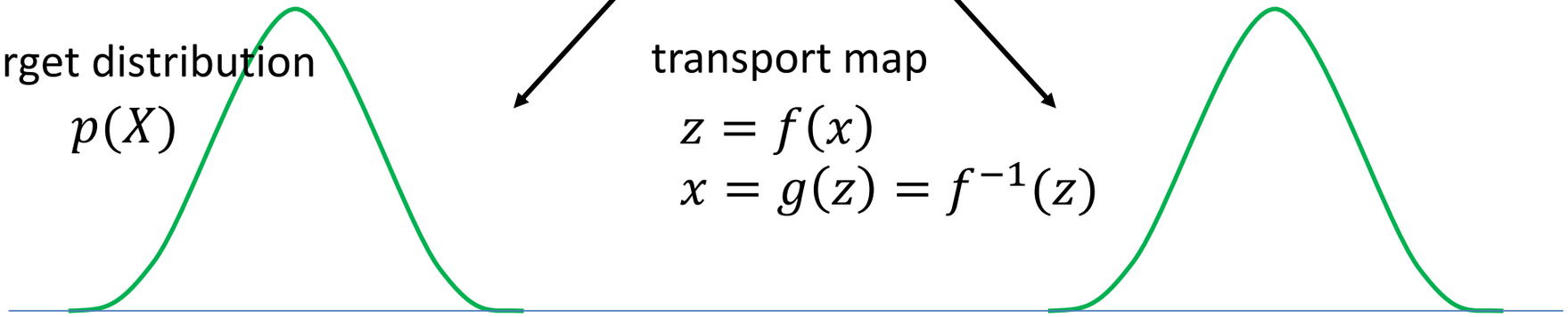


push-forward of latent distribution
$p(X) = g_{\#}q(Z)$

change of variables formula
$p(X = g(z)) = q(Z = z) \cdot |\det \nabla g|^{-1}$

target distribution
$p(X)$

transport map
$z = f(x)$
$x = g(z) = f^{-1}(z)$



Maximum-likelihood training of normalizing flows

Goal: choose $g(x)$ such that the likelihood of the training data $\{x_i\}_{i=1}^N$ is maximized

$$\hat{p}(X) = \text{argmax}_p \, \mathbb{E}_{x\sim p^*(X)}[p(X=x)] = \text{argmin}_p \, \mathbb{E}_{x\sim p^*(X)}[-\log p(X=x)]$$

$$\hat{g}(x) = \text{argmin}_g \, \mathbb{E}_{x\sim p^*(X)}\left[-\log q\left(Z = g^{-1}(x)\right) + \log\left|\det \frac{\partial g(z')}{\partial z'}\right|_{z'=g^{-1}(x)}\right]$$

problem: how to calculate $\log\left|\det \frac{\partial g(z)}{\partial z}\right|$ efficiently if $\mathrm{D} = \dim(X)$ is high and $g(x)$ is complicated?

- simple if the Jacobian $\boldsymbol{J} = \frac{\partial g(z)}{\partial z}$ is triangular: $\det \boldsymbol{J} = \prod_{j=1}^{D} \boldsymbol{J}_{jj}$

⇨ current standard solution: implement $gx)$ by a neural network with "triangular layers" $z^{(l)} = g^{(l)}(z^{(l-1)})$

$$\boldsymbol{J}^{(l)} = \frac{\partial g^{(l)}(z)}{\partial z} \text{ is triangular } \Rightarrow \det \boldsymbol{J}^{(l)} = \prod_{j=1}^{D} \boldsymbol{J}_{jj}^{(l)} \text{ and } \det \boldsymbol{J} = \prod_{l=1}^{L} \boldsymbol{J}^{(l)}$$

⇨ invertible neural networks (INNs) with a coupling layer architecture

Invertible Neural Networks (INNs) with Coupling Layers

Powerful generative models: RealNVP („non-volume preserving“) [Dinh et al. 2017]

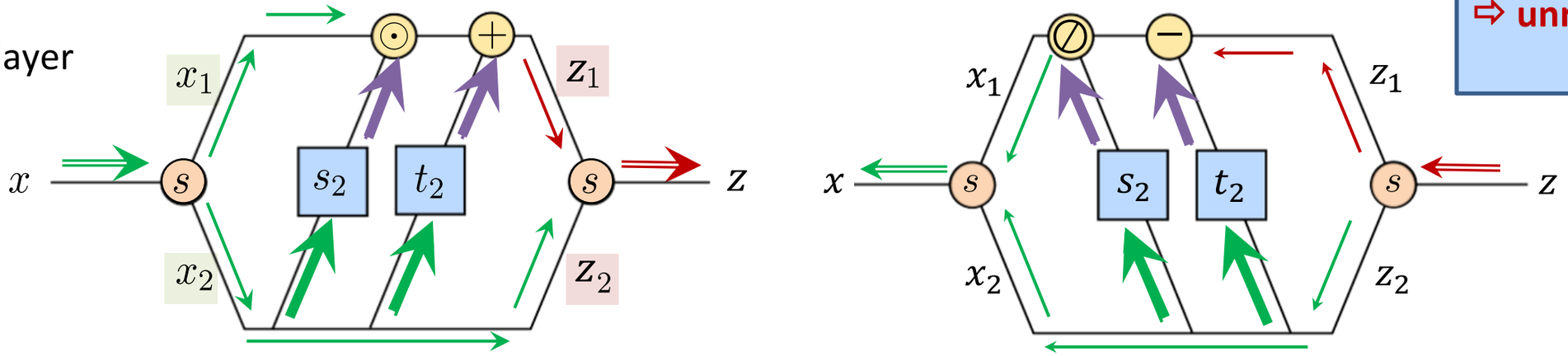
- Network is a sequence of *affine coupling layers*
- Each coupling layer splits its input $x \in \mathbb{R}^D$ into two halves $x_1, x_2 \in \mathbb{R}^{D/2}$
- Upper half is subjected to an affine transformation ⇨ outputs $z_1, z_2 \in \mathbb{R}^{D/2}$
- Affine coefficients are computed by standard fully connected or convolutional networks $s_2 \in \mathbb{R}_+^{D/2}$ and $t_2 \in \mathbb{R}^{D/2}$ from the lower half's data

Forward computation: $z_1 = x_1 \odot s_2(x_2) + t_2(x_2), \quad z_2 = x_2$

Inverse computation: $x_1 = (z_1 - t_2(z_2)) \oslash s_2(z_2), \quad x_2 = z_2$

nested functions s_2 and t_2 are always executed in the same direction ⇨ unrestricted neural networks

Coupling layer



Simulation-based Inference with INNs

amortized simulation-based inference (SBI) with hidden parameters x and observables y

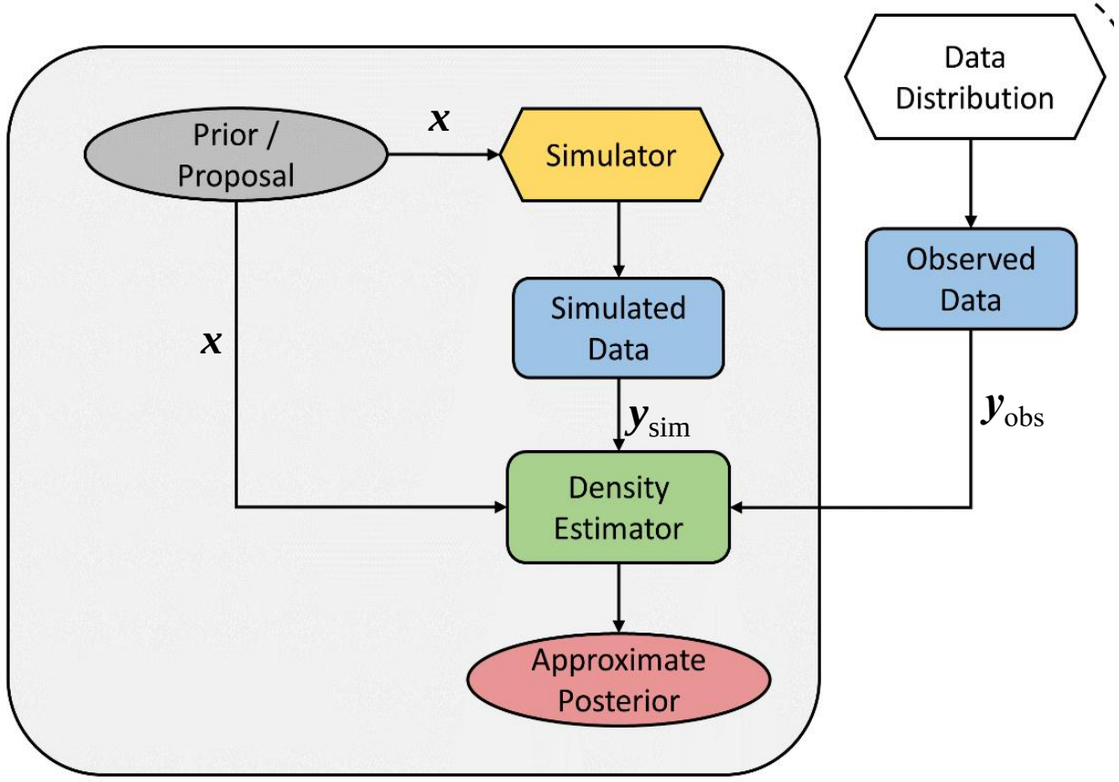
- use an existing simulator to generate synthetic training data: $\{(x_i \sim p_0(X),\ y_i = \mathrm{sim}(x_i))\}_{i=1}^{N}$
- train a conditional INN for the posterior (gray box)

$$p(X = x_i \mid Y = y_i)$$

- apply the trained model to real data

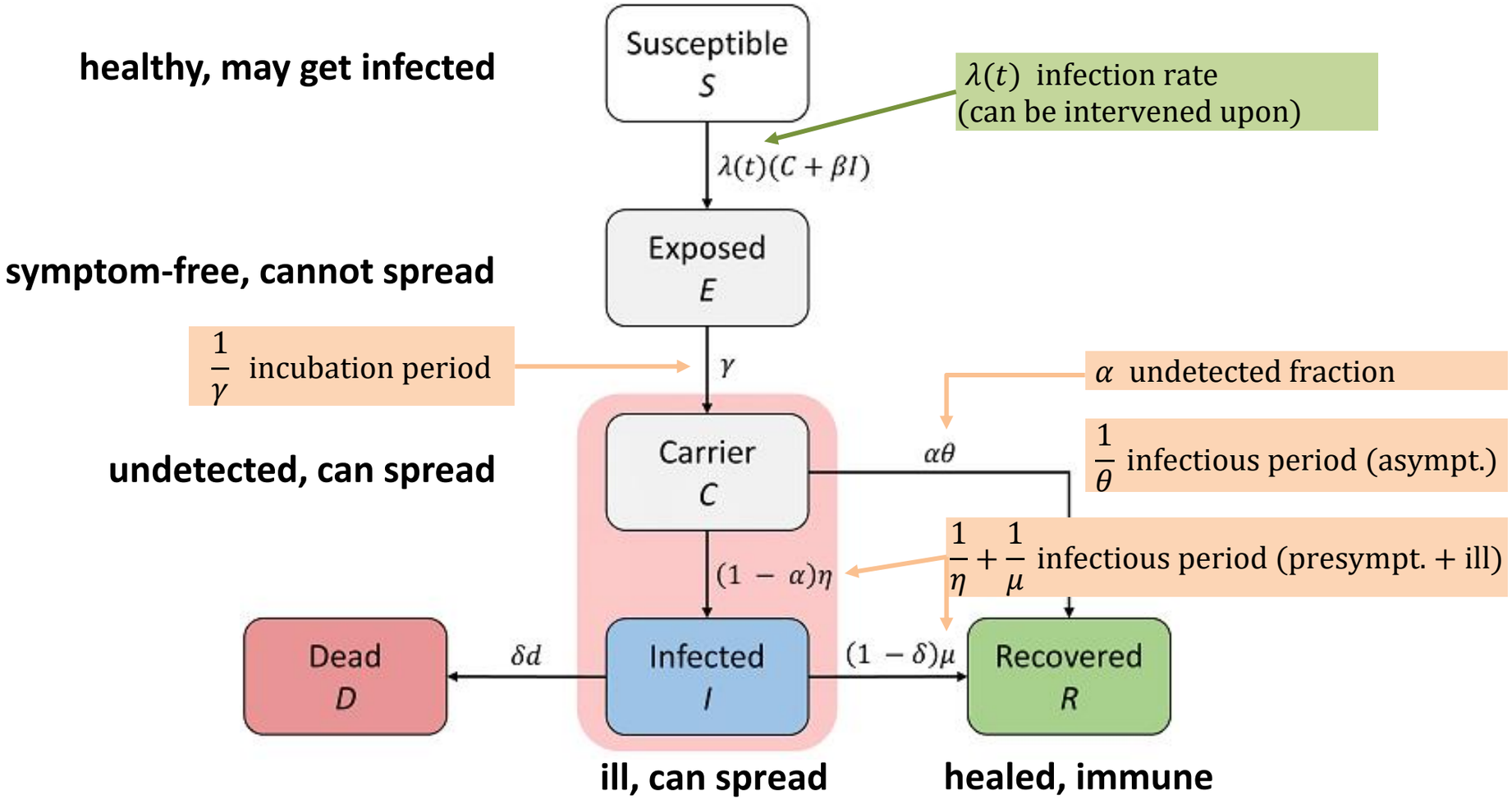
$$p(X = x \mid Y = y_{\mathrm{obs}})$$

- very good results for hard inverse problems
  - image analysis
  - astrophysics, particle physics
  - cognitive science
  - mechanical engineering
  - medicine, epidemiology (e.g. Covid-19 models)



SBI for Covid-19 epidemiology: model setup

- Forward process: SIR compartmental model (Lotka-Volterra type ODE system):



Dehning et al. **"Inferring change points in the spread of COVID-19 reveals the effectiveness of interventions"**, Science (2020).

SBI for Covid-19 epidemiology: model setup

- Forward process: SIR compartmental model (Lotka-Volterra type ODE system):

healthy, may get infected

$$\frac{dS}{dt} = -\lambda(t)\left(\frac{C+\beta I}{N}\right) S$$

symptom-free, cannot spread

$$\frac{dE}{dt} = \lambda(t)\left(\frac{C+\beta I}{N}\right) S - \gamma E$$

undetected, can spread

$$\frac{dC}{dt} = \gamma E - (1-\alpha)\eta C - \alpha\theta C$$

ill, can spread

$$\frac{dI}{dt} = (1-\alpha)\eta C - (1-\delta)\mu I - \delta d I$$

healed, immune

$$\frac{dR}{dt} = \alpha\theta C + (1-\delta)\mu I$$

dead

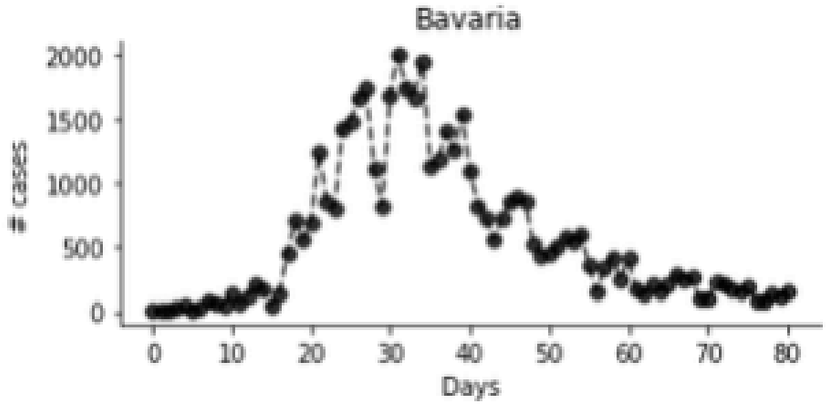
$$\frac{dD}{dt} = \delta d I$$

Dehning et al. **"Inferring change points in the spread of COVID-19 reveals the effectiveness of interventions"**, Science (2020).

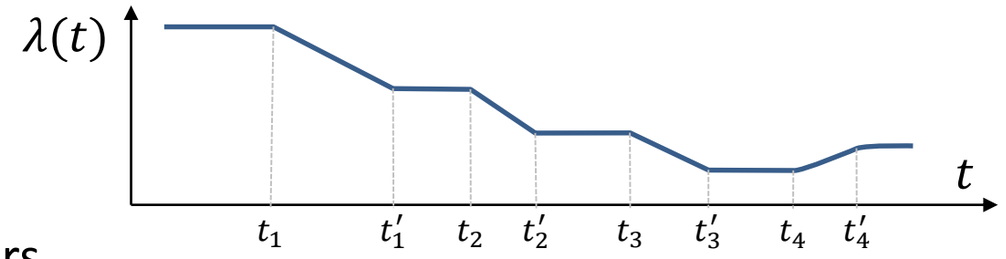
SBI for Covid-19 epidemiology: model setup

- Enhance realism
  - Observation model: reporting delay, noise, weekly modulation

$$I_t^{(obs)} = I_{t-1}^{(obs)} + (1 - f_I(t))\,(1-\alpha)\,\eta\,C_{t-D_I} + \sqrt{I_{t-1}^{(obs)}}\,\sigma_I\,\xi_t$$
$$R_t^{(obs)} = R_{t-1}^{(obs)} + (1 - f_R(t))\,(1-\delta)\,\mu\,I_{t-D_R} + \sqrt{R_{t-1}^{(obs)}}\,\sigma_R\,\xi_t$$
$$D_t^{(obs)} = D_{t-1}^{(obs)} + (1 - f_D(t))\,\delta\,d\,I_{t-D_D} + \sqrt{D_{t-1}^{(obs)}}\,\sigma_D\,\xi_t$$



  - Intervention model:
    four intervals where countermeasures were implemented or relaxed



  - Total: 34 parameters with uninformative or very wide priors
  - Forward simulation is easy: sample from prior, solve ODEs by Runge-Kutta (~50000 simulations)
  - **Inverse: find parameter posteriors from observed time series of detected, dead, recovered cases.**

Dehning et al. **"Inferring change points in the spread of COVID-19 reveals the effectiveness of interventions"**, Science (2020).
Radev, Graw, Köthe et al. **"OutbreakFlow"**, PLOS Computational Biology, 2021.

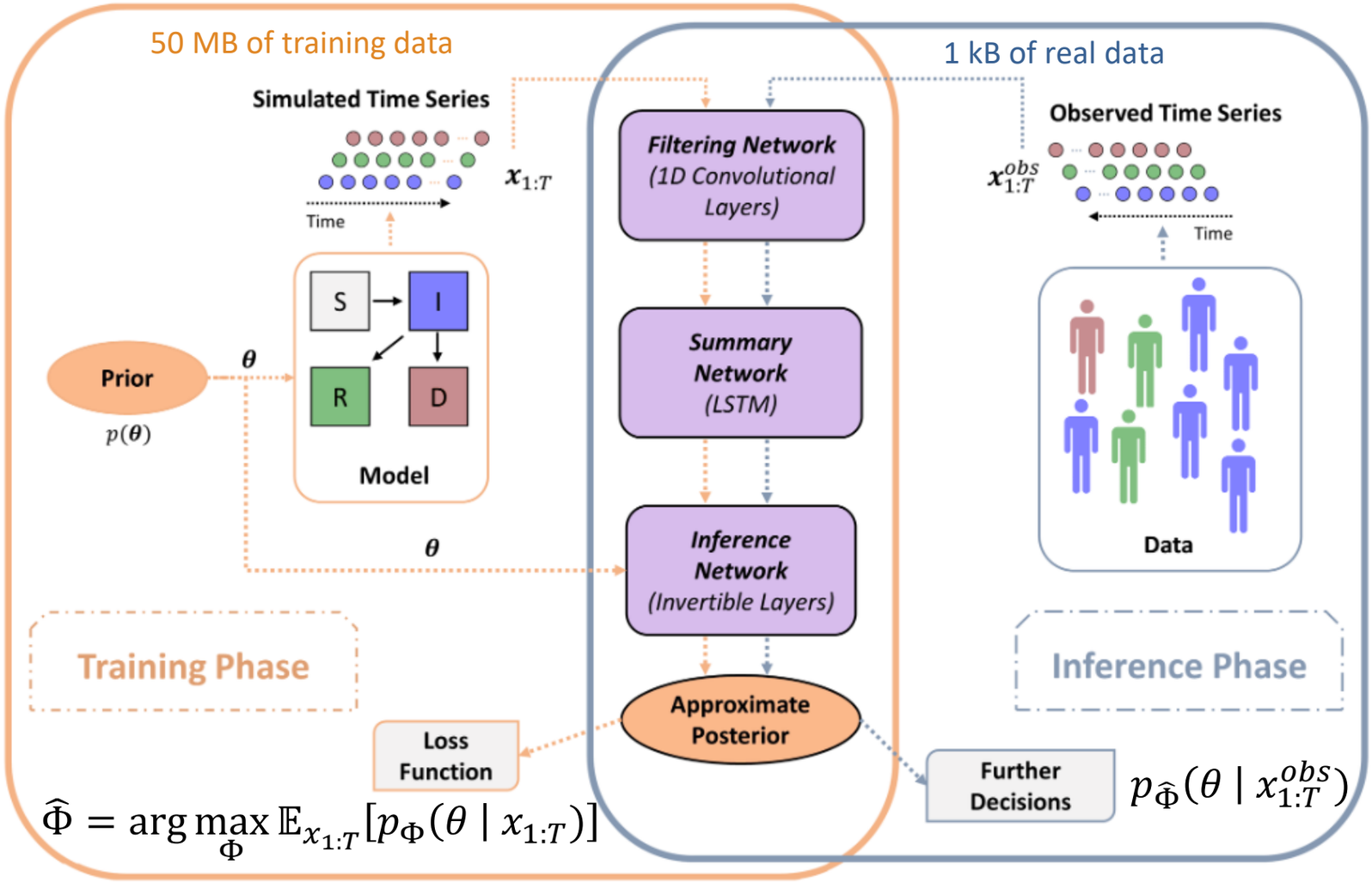
SBI for Covid-19 epidemiology: training

Covid-19 ODEs:

6 compartments: Susceptible, Exposed Carrier, Infected, Deceased, Recovered

$$\frac{dS}{dt} = -\lambda(t)\left(\frac{C+\beta I}{N}\right)S$$
$$\frac{dE}{dt} = \lambda(t)\left(\frac{C+\beta I}{N}\right)S - \gamma E$$
$$\frac{dC}{dt} = \gamma E - (1-\alpha)\eta C - \alpha\theta C$$
$$\frac{dI}{dt} = (1-\alpha)\eta C - (1-\delta)\mu I - \delta d I$$
$$\frac{dR}{dt} = \alpha\theta C + (1-\delta)\mu I$$
$$\frac{dD}{dt} = \delta d I$$

plus model extensions for reporting errors and interventions



Radev, Graw, Köthe et al. **"OutbreakFlow"**, PLOS Computational Biology, 2021.

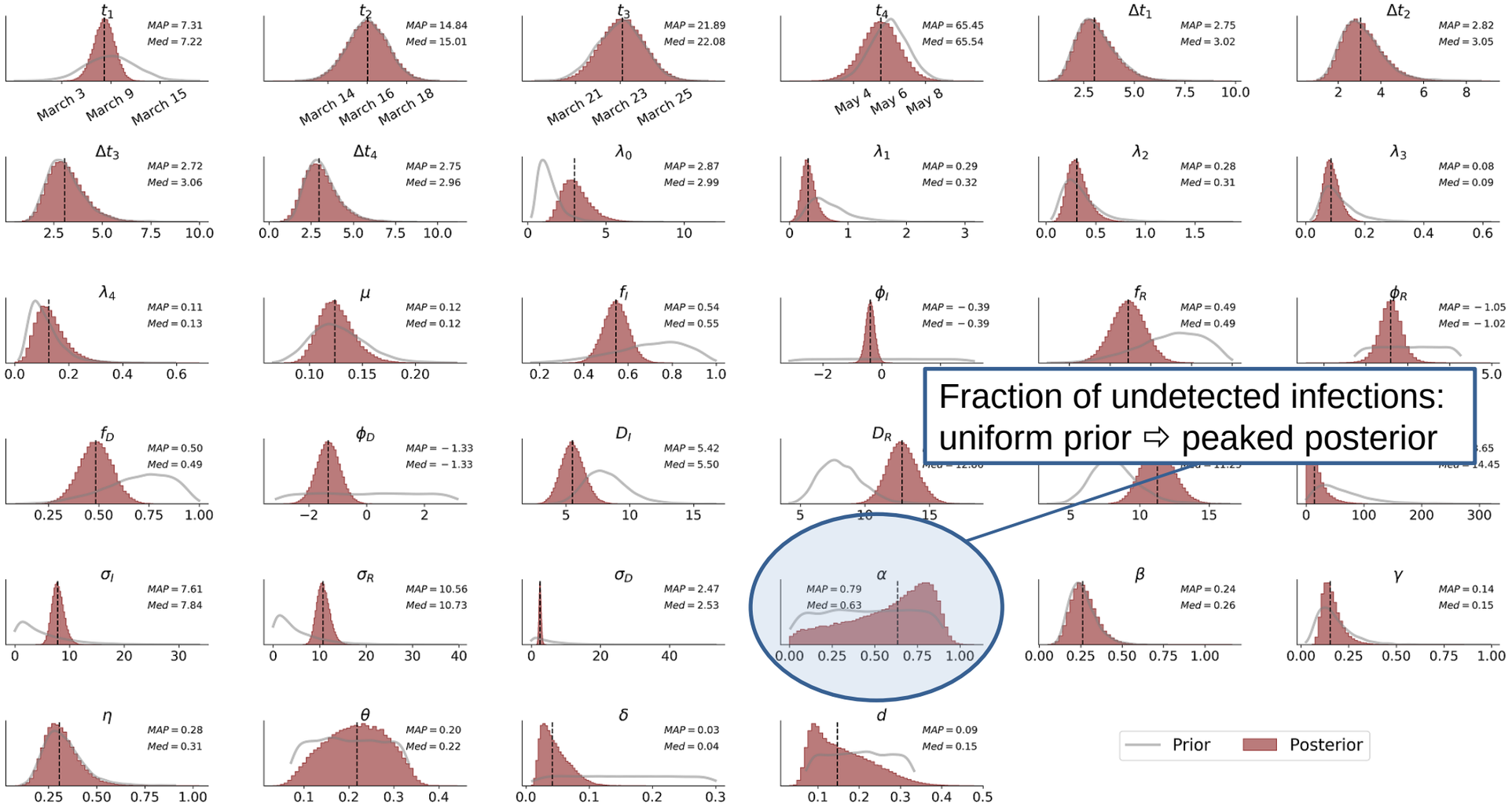
BayesFlow for Epidemiology: Covid-19 Marginal Posteriors

Results: marginal posteriors for first wave in Germany (March – June 2020, 81 time steps)

stable results for 34 parameters

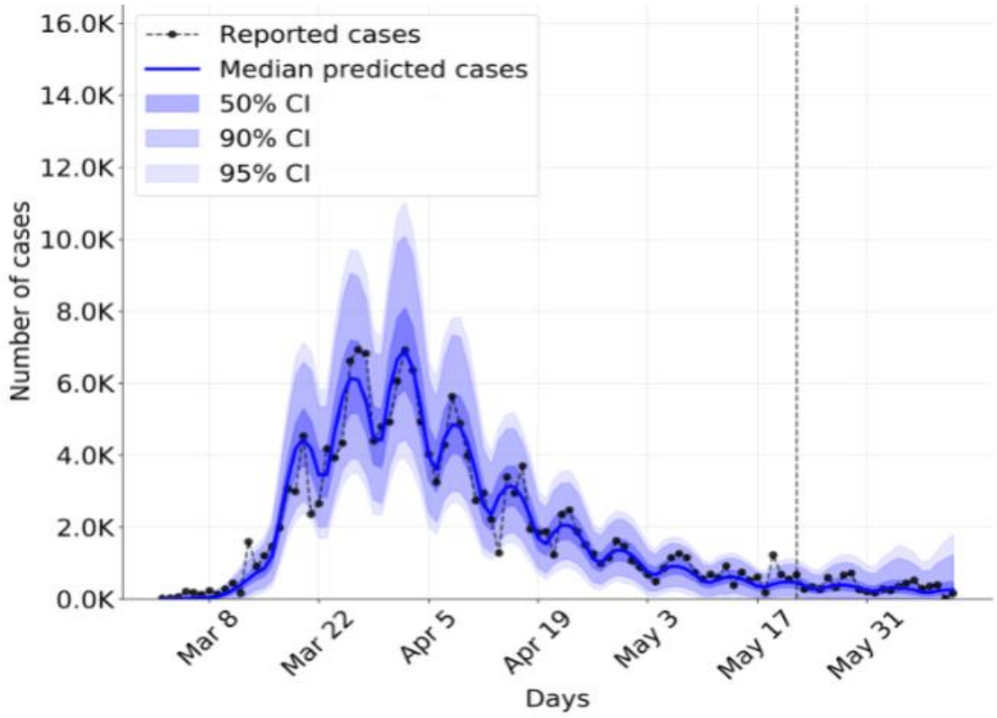
some parameters unidentifiable (posterior = prior)

if identifiable: good correspondence to clinical findings

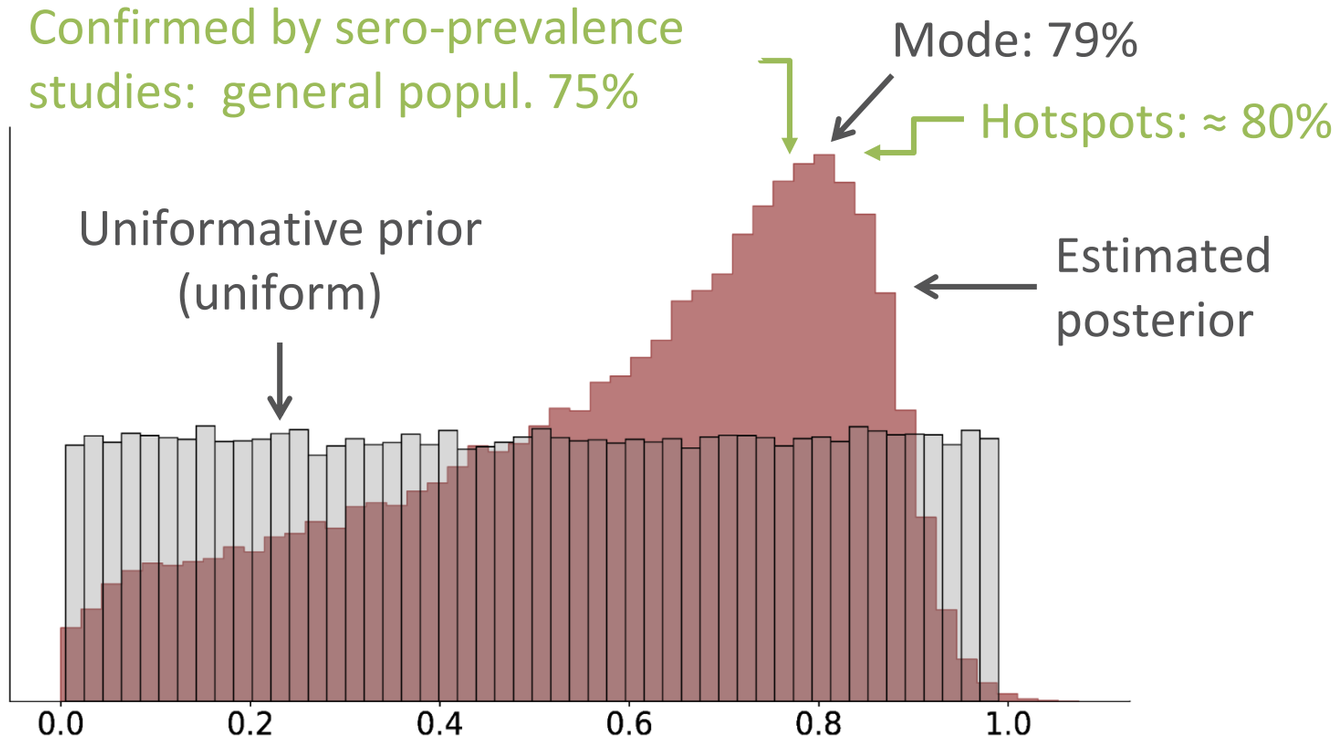


SBI for Covid-19 epidemiology: inference results

Prediction and uncertainty quantification of dynamics
example: **newly diagnosed cases**



Parameter posterior inference
example: **fraction of undetected infections**



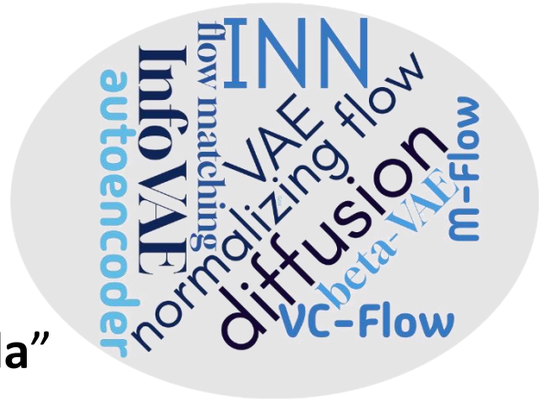
Radev, Graw, Köthe et al. **"OutbreakFlow"**, PLOS Computational Biology, 2021.

The Zoo of Deep Generative Models

How to tame the mess?

- All use some form of inverse transform sampling:
  $x \sim \hat{p}(X)$ $\Leftrightarrow$ $z \sim q(Z)\ [= \mathcal{N}(Z \mid 0, \mathbb{I})]$, $x = \hat{g}(z)$
  sample form known latent prior, transform through learned decoder

⇨ **Categorize methods by** how target $\hat{p}(X)$ is expressed – "**change-of-variables formula**"



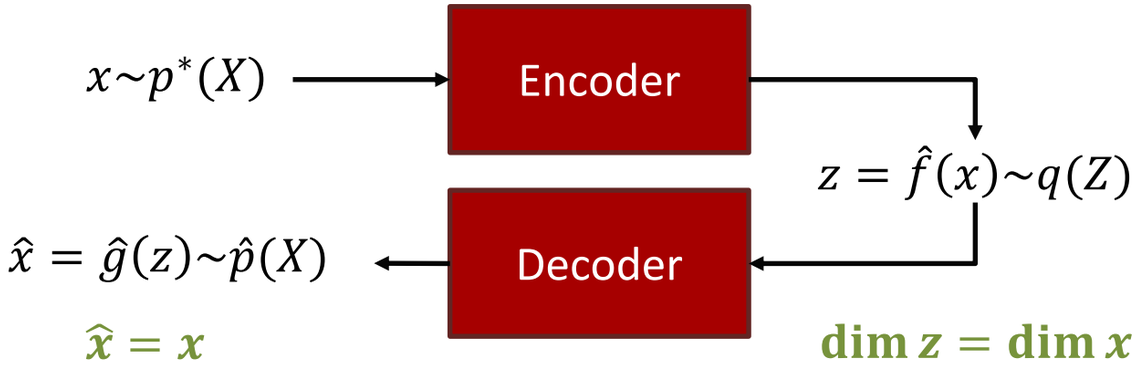
Q1: is the model **bijective** or **injective**?

normalizing flow: lossless encoding **autoencoder: lossy encoding**

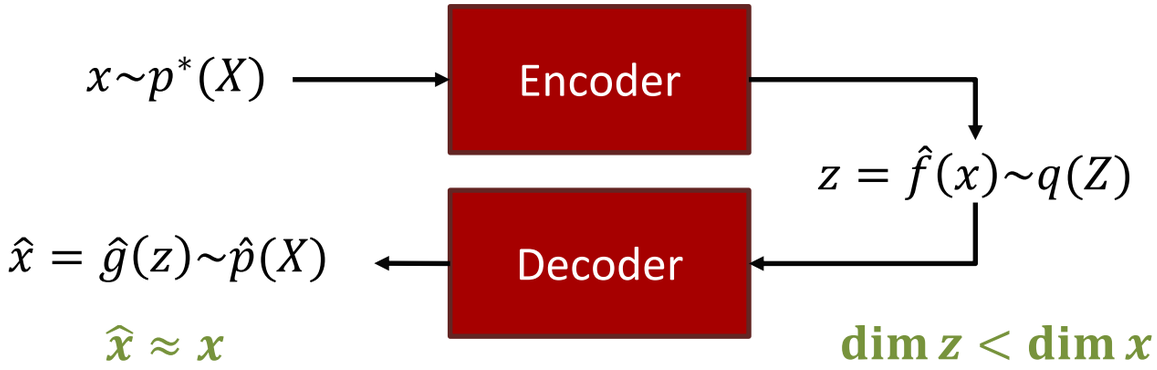
Injective flows

Q1: is the model **bijective** or **injective**?

normalizing flow: lossless encoding



autoencoder: lossy encoding

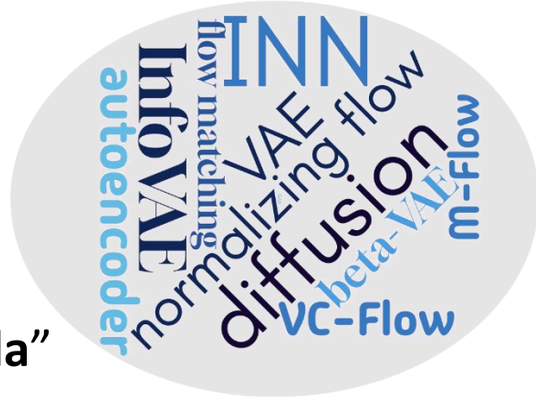


change-of-variable formula:

$$\hat{p}\big(X = \hat{g}(z)\big) = q(Z = z) \cdot \left|\det \frac{\partial \hat{g}(z)}{\partial z}\right|^{-1}$$

$$\hat{p}\big(X = \hat{g}(z)\big) = q(Z = z) \cdot \left|\det \left(\frac{\partial \hat{g}(z)}{\partial z}\right)^T \left(\frac{\partial \hat{g}(z)}{\partial z}\right)\right|^{-1/2}$$

The Zoo of Deep Generative Models



How to tame the mess?

- All use some form of inverse transform sampling:
  $x \sim \hat{p}(X) \quad \Leftrightarrow \quad z \sim q(Z)\ [= \mathcal{N}(Z \mid 0, \mathbb{I})], \quad x = \hat{g}(z)$
  sample form known latent prior, transform through learned decoder

⇨ **Categorize methods by** how target $\hat{p}(X)$ is expressed – "**change-of-variables formula**"

Q1: is the model **bijective** or **injective**?

normalizing flow: lossless encoding **autoencoder: lossy encoding**

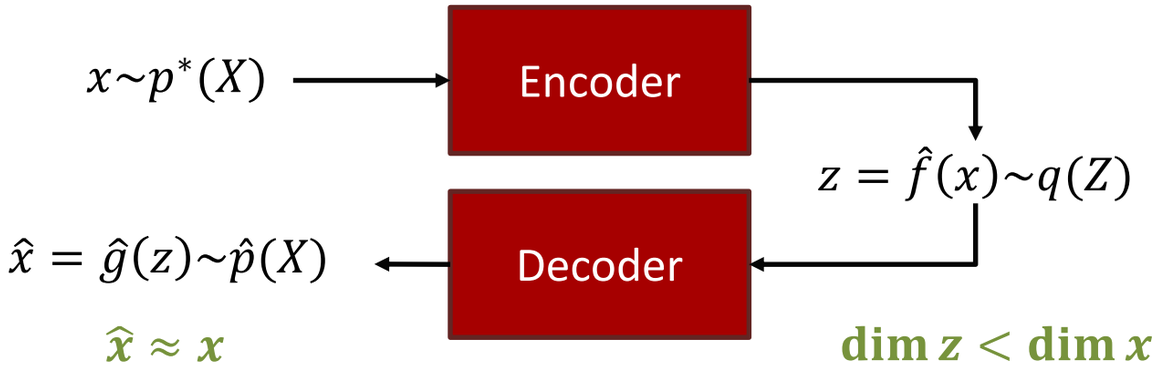
Q2: is the model **deterministic** or **stochastic**?

autoencoder: unique sample per code **variational autoencoder: set of samples per code**

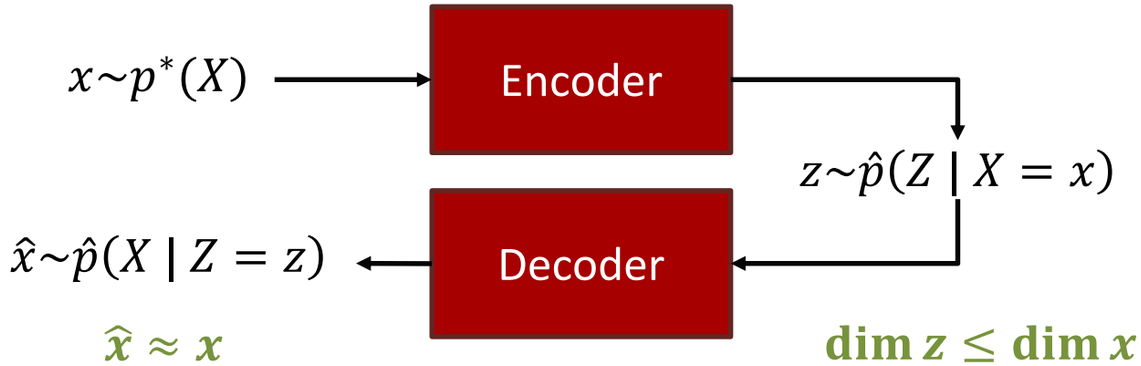
Stochastic flows

Q2: is the model **deterministic** or **stochastic**?

autoencoder: unique sample per code



variational autoencoder: set of samples per code



change-of-variable formula:

$$\hat{p}\big(X = \hat{g}(z)\big) = q(Z = z) \cdot \left| \det \left(\frac{\partial \hat{g}(z)}{\partial z} \right)^T \cdot \left(\frac{\partial \hat{g}(z)}{\partial z} \right) \right|^{-1/2}$$

$$\hat{p}(X = x) = q(Z = z) \cdot \frac{\hat{p}(X = x \mid Z = z)}{\hat{p}(Z = z \mid X = x)}$$

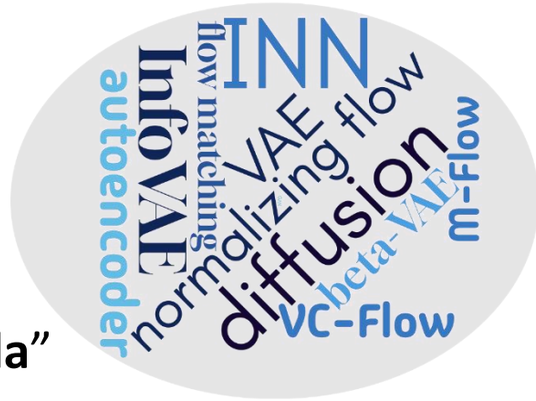
for all $z \in \text{domain}\big(\hat{p}(Z = z \mid X = x)\big)$
(often violated be actual VAEs)

The Zoo of Deep Generative Models

How to tame the mess?

- All use some form of inverse transform sampling:
  $x \sim \hat{p}(X) \quad \Leftrightarrow \quad z \sim q(Z)\ [= \mathcal{N}(Z \mid 0, \mathbb{I})], \quad x = \hat{g}(z)$
  sample form known latent prior, transform through learned decoder

⇨ **Categorize methods by** how target $\hat{p}(X)$ is expressed – "**change-of-variables formula**"



Q1: is the model **bijective** or **injective**?
normalizing flow: lossless encoding **autoencoder: lossy encoding**

Q2: is the model **deterministic** or **stochastic**?
autoencoder: unique sample per code **variational autoencoder: set of samples per code**

Q3: is the model executed in **discrete** or **continuous** time steps?
multi-layer network **diffusion model**

Diffusion models

Q3: is the model executed in **discrete** **multi-layer network** or **continuous** time steps? **diffusion model**

$\hat{x} = \hat{g}(z) \sim \hat{p}(X)$ ← Decoder ← $z \sim q(Z)$

$$\hat{g}(z) = (\hat{g}_L \circ \hat{g}_{L-1} \circ \cdots \circ \hat{g}_2 \circ \hat{g}_1)\,(z)$$

change-of-variable formula:

$$\hat{p}\big(X = \hat{g}(z)\big) = q(Z = z) \cdot \prod_{l=1}^{L} \left| \det \frac{\partial \hat{g}_l(z_{l-1})}{\partial z_{l-1}} \right|^{-1}$$

$\hat{x} = \hat{g}(z) \sim \hat{p}(X)$ ← Decoder ← $z \sim q(Z)$

$$\hat{g}(z) = z(1) \text{ with } \frac{dz(t)}{dt} = \hat{G}(z(t), t) \text{ and } z(0) = z,\quad t = 0 \to 1$$

$$\hat{p}(X = x) = q(Z = z) \cdot \exp\left(- \int_0^1 \operatorname{tr}\left(\frac{\partial}{\partial z} \hat{G}(z(t), t) \right) dt \right)$$

„instantaneous change–of–variables formula"

check out my paper: U. Köthe (2023) "A review of change of variable formulas for generative modeling", arXiv:2308.02652

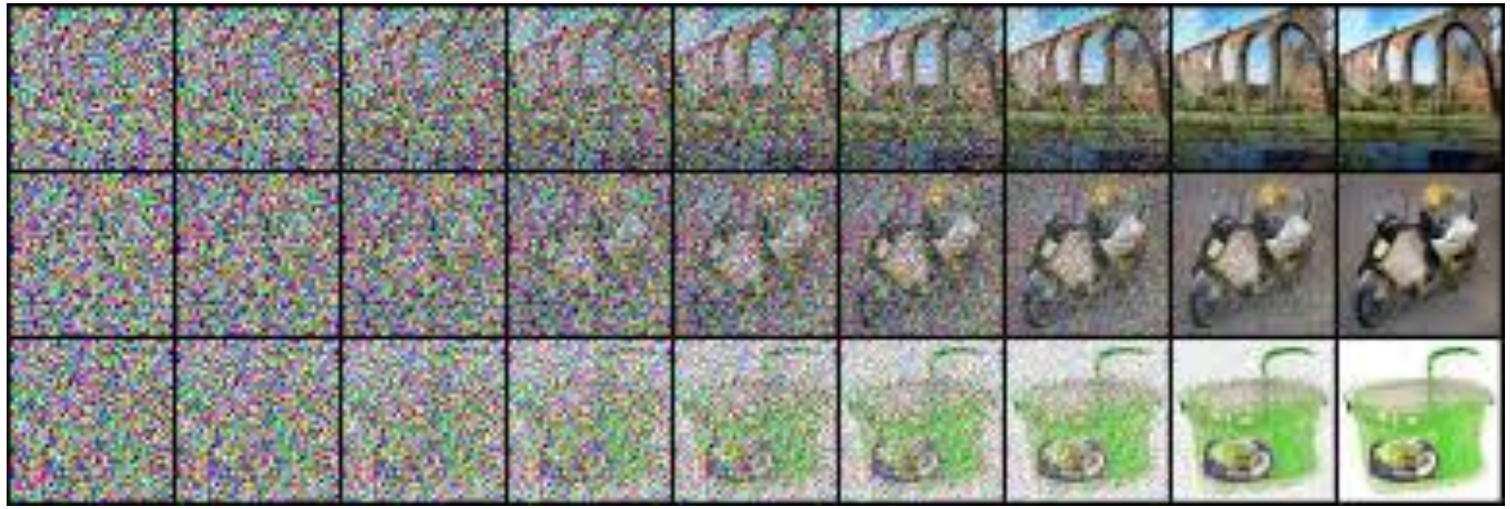
Towards Free-form Flows

Normalizing flows are our workhorse, but coupling layers

- may limit expressivity
- prevent easy incorporation of application prior knowledge

Diffusion models have currently the best generative accuracy, but

- are expensive
- are not interpretable



Lipman et al. ICLR 2023

Q: Can we have the best of both worlds?

- Must evaluate the Jacobian determinant without the coupling layer trick

$$p(X = \hat{g}(z)) = q(Z = z) \cdot \left|\det \frac{\partial\, \hat{g}(\boldsymbol{z})}{\partial\, \boldsymbol{z}}\right|^{-1}$$

Free-form Flows

Q: Can we have the best of both worlds?

- Must evaluate the Jacobian determinant without the coupling layer trick

$$p(X = \hat{g}(z)) = q(Z = z) \cdot \left|\det \frac{\partial\, \hat{g}(\boldsymbol{z})}{\partial\, \boldsymbol{z}}\right|^{-1}$$

- Jacobian $\widehat{\boldsymbol{g}}' = \frac{\partial\, \hat{g}(\boldsymbol{z})}{\partial\, \boldsymbol{z}}$ can be constructed using autodiff:

$$\widehat{\boldsymbol{g}}'_k = \mathrm{jvp}(\hat{g}, \boldsymbol{z}, e_k)$$

  gives column $\widehat{\boldsymbol{g}}'_k$, with Jacobian-vector product library function $\mathrm{jvp}(.)$ and k-th unit vector e_k

  – conventional wisdom: this is too expensive
  – but `tensorflow` and `pytorch` on current GPUs can handle this reasonably for $D \in \mathcal{O}(10^3)$

  ⇨ architecture tricks like coupling layers are not needed at inference time

- however, it is still too slow for millions of training iterations ⇨ need another solution for training

Basic Ideas behind Free-form Flows

$$\hat{\theta} = \text{argmin}_\theta \; \mathbb{E}_x[-\log p(X = x)] = \text{argmin}_\theta \; \mathbb{E}_x\left[-\log q(Z = \boldsymbol{z}) + \log\left|\det \frac{\partial\, \hat{g}(\boldsymbol{z})}{\partial\, \boldsymbol{z}}\right|\right]$$

- Crucial observation for tractable training
  - to minimize NLL via gradient descend, we need $\nabla_\theta \log\, |\det \hat{\boldsymbol{g}}'|$
  - can be calculated *without constructing* $\hat{\boldsymbol{g}}'$ *explicitly*

1. trace formula: for any matrix $\boldsymbol{A}(\theta)$: $\nabla_\theta \log\, |\det \boldsymbol{A}(\theta)| = \text{tr}(\boldsymbol{A}(\theta)^{-1}\, \nabla_\theta \boldsymbol{A}(\theta))$
2. Hutchinson trace estimator: let $v \sim p_v(V)$ with $\mathbb{E}_v[v\, v^T] = \mathbb{I}_D$, e.g. $v \sim \mathcal{N}(0, \mathbb{I}_D)$

$$\begin{aligned}
\text{tr}(\boldsymbol{A}(\theta)^{-1}\, \nabla_\theta \boldsymbol{A}(\theta)) &= \text{tr}(\boldsymbol{A}(\theta)^{-1}\, \nabla_\theta \boldsymbol{A}(\theta)\, \mathbb{E}_v[v\, v^T]) \\
&= \mathbb{E}_v[\text{tr}(\boldsymbol{A}(\theta)^{-1}\, \nabla_\theta \boldsymbol{A}(\theta)\, v\, v^T)] \\
&= \mathbb{E}_v[\text{tr}(v^T \boldsymbol{A}(\theta)^{-1}\, \nabla_\theta \boldsymbol{A}(\theta)\, v\,)] \\
&= \mathbb{E}_v[v^T \boldsymbol{A}(\theta)^{-1}\, \nabla_\theta \boldsymbol{A}(\theta)\, v]
\end{aligned}$$

$$\nabla_\theta \log\, |\det \hat{\boldsymbol{g}}'| = \mathbb{E}_v\left[v^T \hat{\boldsymbol{g}}'^{-1}\, \nabla_\theta\, \hat{\boldsymbol{g}}'\, v\right] \approx \frac{1}{M} \sum_{m=1}^{M} v_m^T\, \hat{\boldsymbol{g}}'^{-1}\, \nabla_\theta\, \hat{\boldsymbol{g}}'\, v_m \quad (M = 1 \text{ sufficient for SGD})$$

Basic Ideas behind Free-form Flows

$$\hat{\theta} = \text{argmin}_\theta \; \mathbb{E}_x[-\log p(X = x)] = \text{argmin}_\theta \; \mathbb{E}_x\left[-\log q(Z = \boldsymbol{z}) + \log\left|\det \frac{\partial\, \hat{g}(\boldsymbol{z})}{\partial\, \boldsymbol{z}}\right|\right]$$

- Crucial observation for tractable training
  - to minimize NLL via gradient descend, we need $\nabla_\theta \log\, |\det \hat{\boldsymbol{g}}'|$
  - can be calculated *without constructing* $\hat{\boldsymbol{g}}'$ *explicitly*

1. trace formula: for any matrix $\boldsymbol{A}(\theta)$: $\nabla_\theta \log\, |\det \boldsymbol{A}(\theta)| = \text{tr}(\boldsymbol{A}(\theta)^{-1}\, \nabla_\theta \boldsymbol{A}(\theta))$
2. Hutchinson trace estimator: let $v \sim p_v(V)$ with $\mathbb{E}_v[v\, v^T] = \mathbb{I}_D$, e.g. $v \sim \mathcal{N}(0, \mathbb{I}_D)$

$$\nabla_\theta \log\, |\det \hat{\boldsymbol{g}}'| = \mathbb{E}_v\left[v^T \hat{\boldsymbol{g}}'^{-1}\, \nabla_\theta\, \hat{\boldsymbol{g}}' v\right] \approx \frac{1}{M} \sum_{m=1}^{M} v_m^T\, \hat{\boldsymbol{g}}'^{-1}\, \nabla_\theta\, \hat{\boldsymbol{g}}'\, v_m \quad (M = 1 \text{ sufficient for SGD})$$

3. pseudo-inverse pairs: train separate encoder $f_\theta(x)$ and decoder $g_\theta(z)$ networks s.t. $g_\theta(z) \approx f_\theta^{-1}(z)$

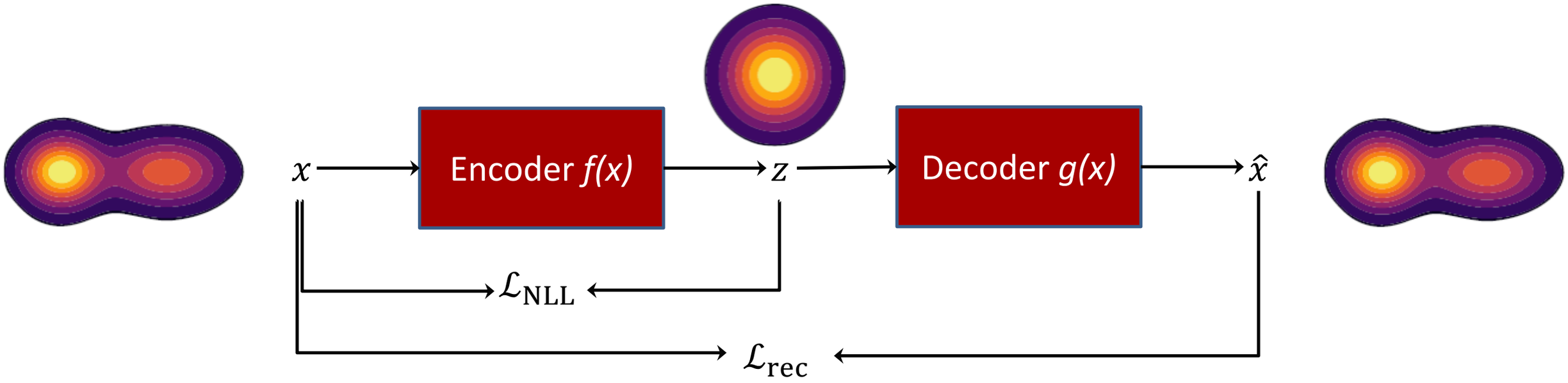
$$\nabla_\theta \mathcal{L}_{\text{rec}} = \mathbb{E}_x[\nabla_\theta \|x - g_\theta(f_\theta(x))\|_2^2]$$

⇨ substitute $\hat{\boldsymbol{g}}'^{-1} \approx \hat{\boldsymbol{f}}' = \frac{\partial\, f_\theta(x)}{\partial x}$ and use Jacobian-vector and vector-Jacobian products

$$\nabla_\theta \mathcal{L}_{\text{NLL}} = \mathbb{E}_{x,v}\left[-\nabla_\theta \log q\big(Z = f_\theta(x)\big) - \text{vjp}(g_\theta,\, f_\theta(x), v^T)\, \nabla_\theta\, \text{jvp}(f_\theta\,, x, v)\right]$$

Basic Ideas behind Free-form Flows

Free-form flow looks like an autoencoder, but $D = \dim(X) = \dim(Z)$



train with a weighted sum of the two losses

$$\mathcal{L}_{\mathrm{tot}} = \mathcal{L}_{\mathrm{NLL}} + \beta\, \mathcal{L}_{\mathrm{rec}}$$

with β big enough

advantage: no restrictions on architecture of encoder and decoder
⇨ can use networks incorporating application-defined constraints and symmetries

Free-form flow results

Application to molecular modeling

- energy of molecules (in vacuum) is independent of location and rotation

⇨ model the Gibbs distribution

$$p(X = x) \propto \exp\left(-\frac{E(x)}{k\,T}\right)$$

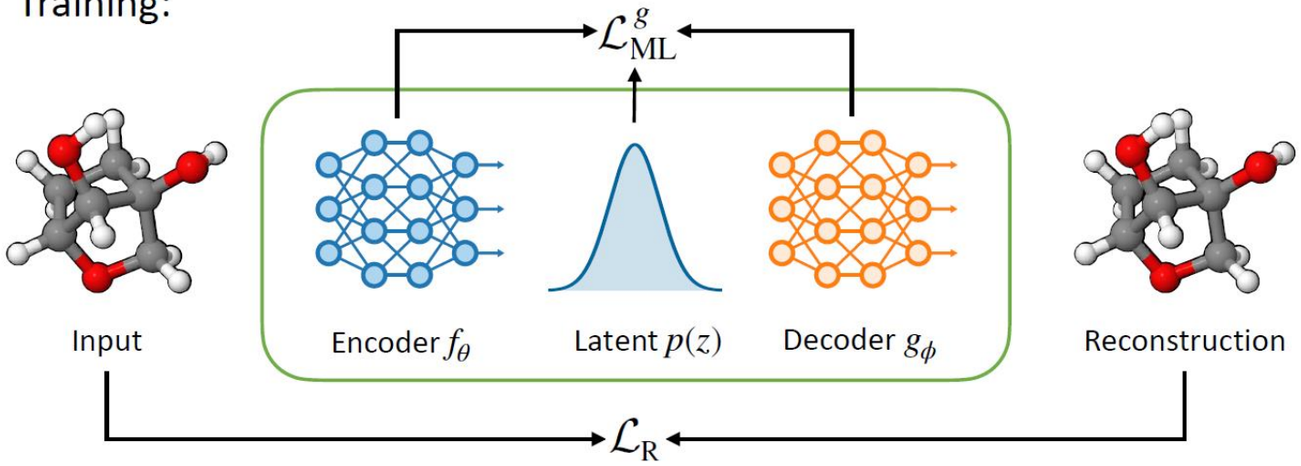
with $\boldsymbol{SE(3)}$-equivariant graph neural networks

[Sartorras et al. 2021], [Klein et al. 2023]

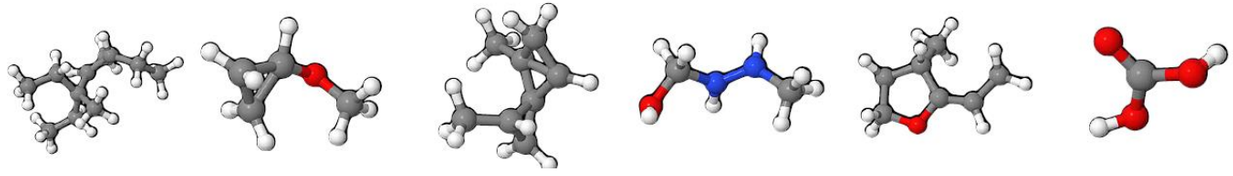
- free-form flows allow NLL training and beat original E-GNNs on three benchmark datasets: (negative log-likelihood, lower is better)

	DW4	LJ13	LJ55
$E(n)$-NF	1.72 ± 0.01	-16.28 ± 0.04	n/a
OT-FM	1.70 ± 0.02	-16.54 ± 0.03	-94.43 ± 0.22
E-OT-FM	1.68 ± 0.01	-16.70 ± 0.12	-97.93 ± 0.52
$E(n)$-FFF	1.68 ± 0.01	**-17.09 ± 0.16**	**-144.86 ± 1.42**

Training:



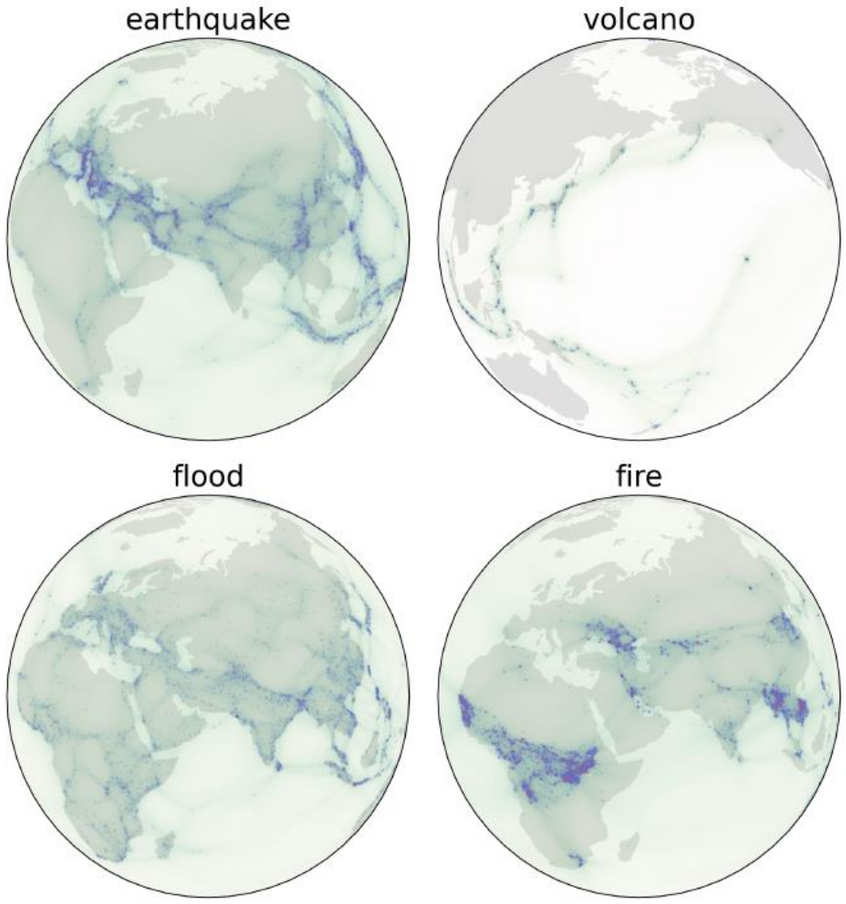
Samples:



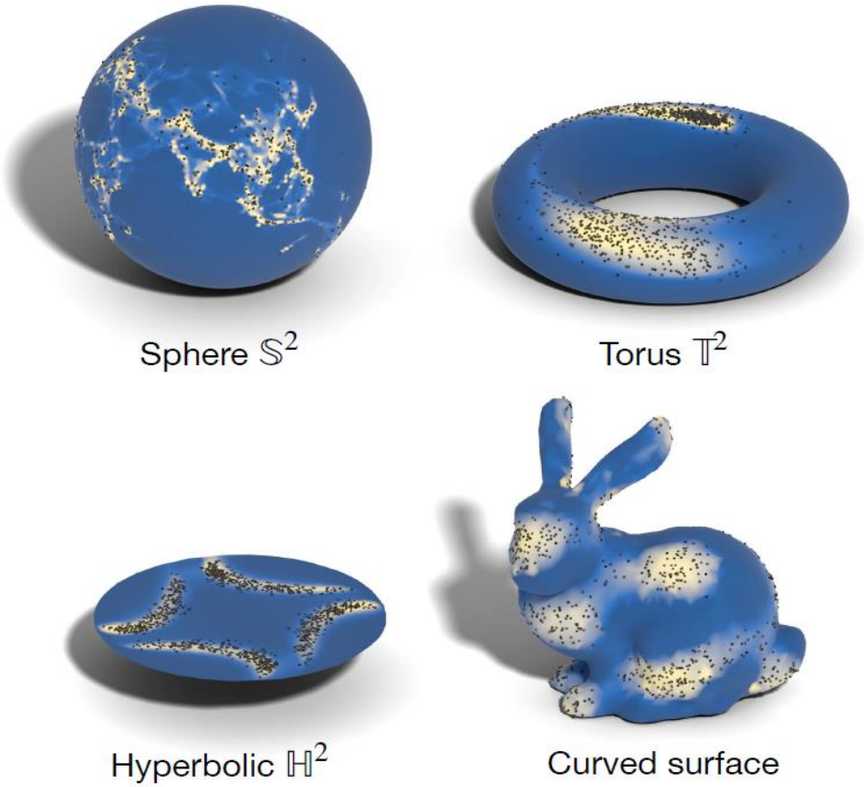
Manifold free-form flows

Free-form flows are easy to adapt to distributions on manifolds $\mathcal{M}$, e.g. the surface of the earth

learned distributions of natural disasters

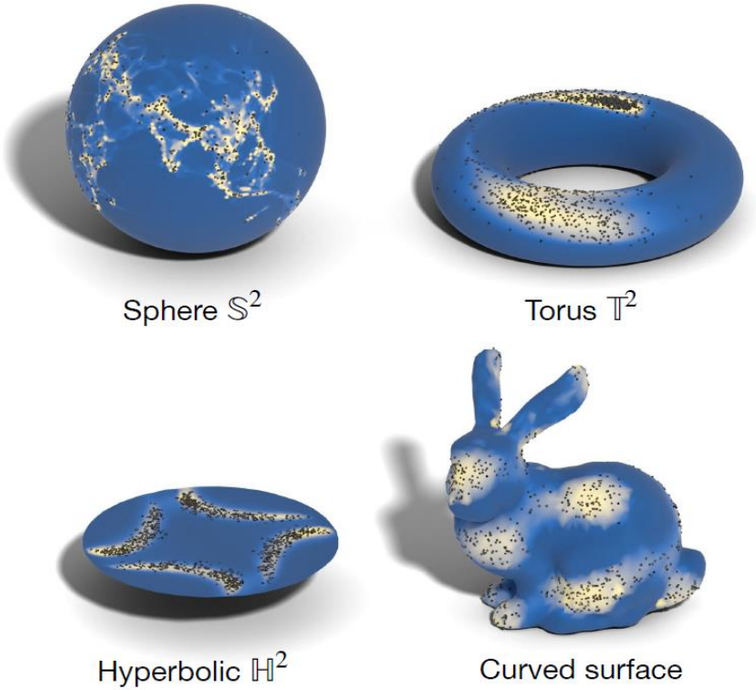


distributions on many different manifolds



Manifold free-form flows

Free-form flows are easy to adapt to distributions on manifolds $\mathcal{M}$, e.g. the surface of the earth



change-of-variables formula:

$$p(X = \hat{g}(z)) = q(Z = z) \cdot \left| \det \left(Q^T \cdot \frac{\partial\, \hat{g}(\boldsymbol{z})}{\partial\, \boldsymbol{z}} \cdot R \right) \right|^{-1}$$

with

Q and R basis matrices in the tangent spaces $\mathcal{T}_x\mathcal{M}$ and $\mathcal{T}_z\mathcal{M}$

and gradient trick as before

$$\nabla_\theta \mathcal{L}_{\text{NLL}} = \mathbb{E}_{x,v}\left[-\nabla_\theta \log q\big(Z = f_\theta(x)\big) - \text{vjp}(g_\theta, f_\theta(x), v^T)\, \nabla_\theta\, \text{jvp}(f_\theta, x, v) \right]$$

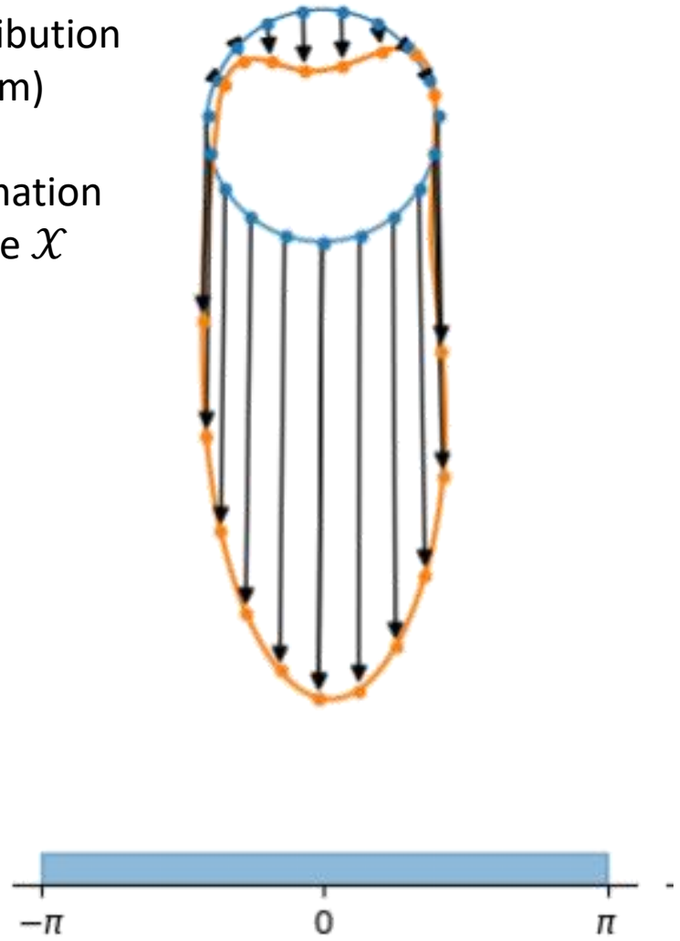
but v sampled from tangent space $\mathcal{T}_z\mathcal{M}$

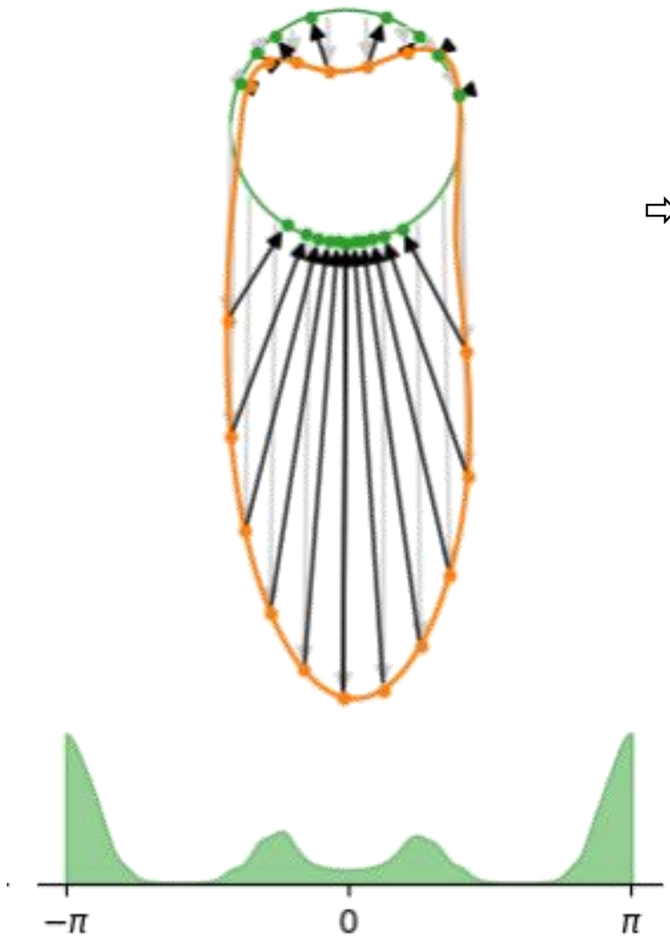
Manifold free-form flows

Free-form flows are easy to adapt to distributions on manifolds $\mathcal{M}$, e.g. the surface of the earth

start with simple distribution on $\mathcal{M}$ (e.g. uniform)

and learn a transformation to embedding space $\mathcal{X}$



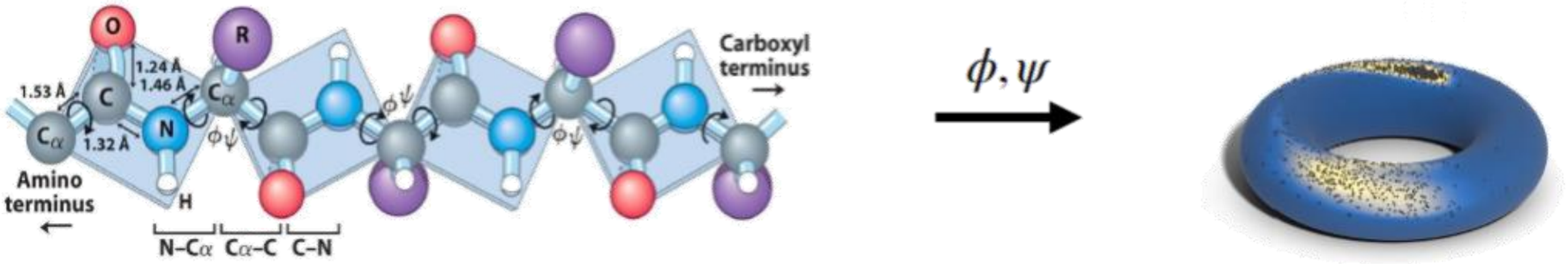


use analytical formula to project onto $\mathcal{M}$

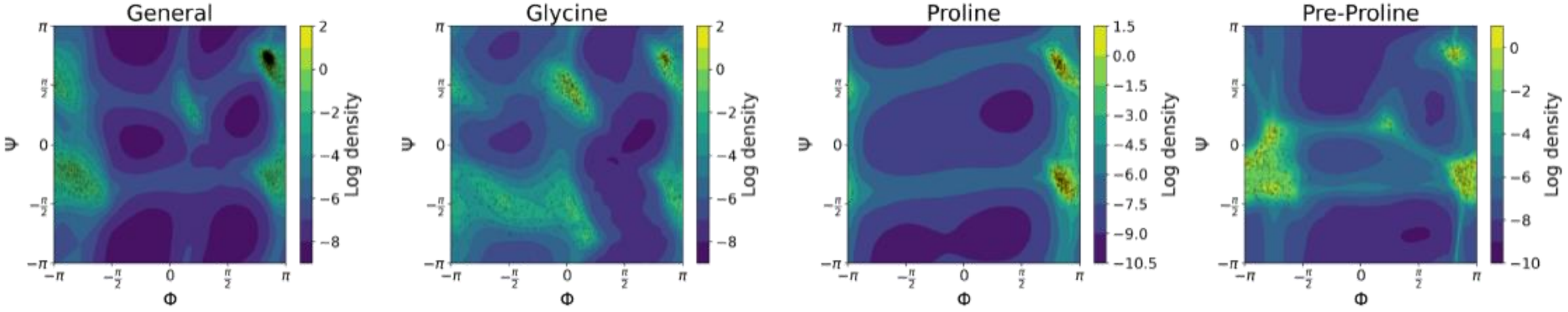
⇨ projected mass forms the target distribution

Manifold free-form flows: molecule generation results

- Example: sample protein backbone dihedrals on the 2-dimensional torus



comparison of learned densities in the (periodic) ϕ, ψ-plane with samples from true Boltzmann distribution (black dots) for different amino acids



Summary

Free-form flows are a very interesting alternative to established coupling flows

- can incorporate constraints, invariants, conservation laws from the application
- support bottleneck architectures

- many interesting applications and open problems to be explored

check out our new papers:

- Free-form Flows: Make Any Architecture a Normalizing Flow, AISTAT ‘24, arXiv:2310.16624
- Lifting Architectural Constraints of Injective Flows, ICLR ‘24, arXiv:2306.01843
- Learning Distributions on Manifolds with Free-form Flows, to appear: NeurIPS ‘24, arXiv:2312.09852

and the **BayesFlow framework** for simulation based inference:

- https://bayesflow.org/
- https://github.com/bayesflow-org/bayesflow