MAX PLANCK INSTITUTE
FOR INTELLIGENT SYSTEMS

Normalizing Flows

Lecture @ GenAI Workshop

Annalena Kofler, Munich, 23.09.2024

Short Introduction: Who am I?

- PhD student at the Max-Planck-Institute for Intelligent Systems in Tübingen supervised by Prof. Bernhard Schölkopf
- Working on ML for Physics
    - Posterior Estimation for Gravitational Wave Inference
    - Deep Generative Modeling & Simulation-based inference
    - Differentiable Physics Simulators
- Previously: Master's student at TUM working with Prof. Lukas Heinrich

Annalena
(On holiday)

Short Introduction: Who are you?

- Questions:
  - Are you a Bachelor student/Master student/PhD student/PostDoc?
  - What is your background? (Physics, computer science, maths, …)
  - Do you know what a normalizing flow is?

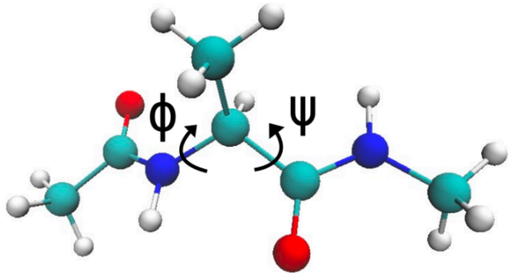
Why care about probability distributions?

- Data samples usually originate from some distribution
- Examples in physics?

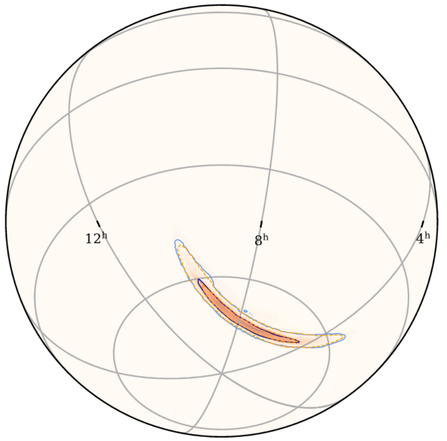
Which examples of probability distributions in physics come to your mind?

Why care about probability distributions?

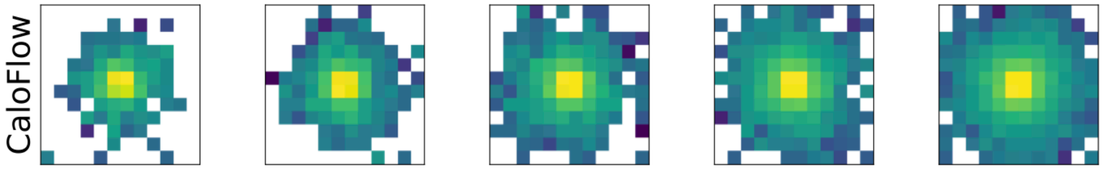
- Data samples usually originate from some distribution
- Examples in physics:
  - Positions of atoms in molecules
  - Parameters of gravitational waves or exoplanet atmospheres
  - Hits in detector from particle showers
  - …
- But: Distributions might be unknown, e.g. distribution over pixels in image



Midgley, Stimper, et al., Flow Annealed Importance Sampling Bootstrap, ICLR 2023



Dax, et al.,Real-time gravitational-wave science with neural posterior estimation, PRL2021



Krause and Shih, CaloFlow: Fast and Accurate Generation of Calorimeter Showers with Normalizing Flows, PRD2021

How can we learn a distribution over data?

- Train model to get model distribution $q_{\text{model}} \approx$ target distribution p_{target}

- What can we do with such models?
  - **Sampling** new data points $x \sim q_{\text{model}}(x)$

$$\sim q_{\text{dog-model}}(x)$$

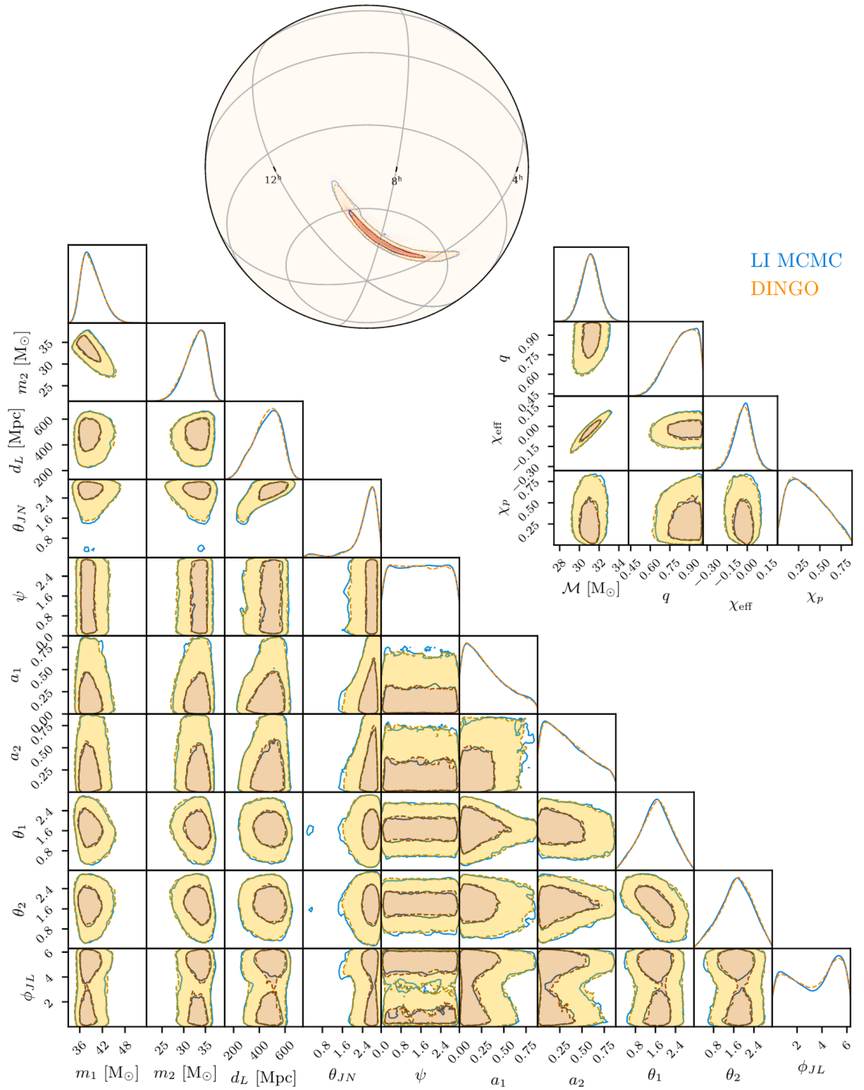
  - **Density evaluation** of the model distribution $q_{\text{model}}(x_{\text{sample}})$ for a given data point x_{sample}

$$q_{\text{dog-model}}\left(\text{  }\right) = 0.87$$

Challenge: Complexity of Data Distribution

- Data distributions are usually extremely complex
  - Multi-modal
  - Unknown structure
  - High dimensional, e.g. pixels in image
- Not enough to learn mean and variance of Gaussian distribution!

→ powerful model required



Dax, et al.,Real-time gravitational-wave science with neural posterior estimation, PRL2021

How to build a powerful model?

- Use a Deep Neural Network (DNN) f_θ

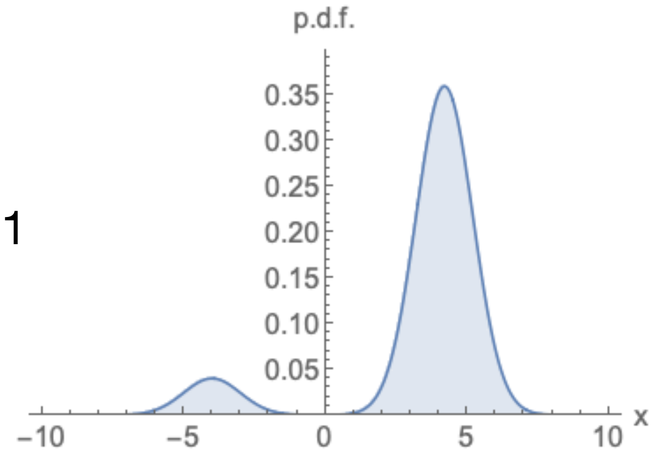
https://pollev.com/nicolehartman968

Is the output $f_\theta(x)$ of the DNN for a sample x already a probability?

→ No, it does not fulfill the properties of a pdf.

1. Positive: $p(x) \geq 0 \forall x$
2. Normalized: $\int p(x)\ \mathrm{d}x = 1$

Area = 1



Animations from blog post by Yang Song

How to build a powerful model?

How can we adapt $f_\theta(x)$ such that it is a valid probability density function?

Form groups and discuss!

1. Positive $\rightarrow \exp(f_\theta(x))$
2. Normalized $\rightarrow \quad q_\theta(x) = \dfrac{\exp(f_\theta(x))}{Z_\theta}$

   with normalization constant $\quad Z_\theta = \int \exp(f_\theta(x))\mathrm{d}x$

- Problem: calculating the normalization constant is intractable for DNN

How to normalize? - Model Families

1. Approximate Z_θ
   - Energy-based models

X Inaccurate probability

2. Use restricted NN architectures
   - Variational autoencoders
   - Autoregressive models
   - Discrete normalizing flows
   - Continuous normalizing flows

X Limited flexibility of the model

3. Focus only on the generative process
   - GANs
   - Flow matching
   - Diffusion models

X No/slower evaluation of the probability

How to normalize? - Model Families

1. Approximate Z_θ
   - Energy-based models
2. Use restricted NN architectures
   - Variational autoencoders
   - Autoregressive models
   - **Discrete normalizing flows** Day 1
   - **Continuous normalizing flows** Day 2a
3. Focus only on the generative process
   - GANs
   - **Flow matching** Day 2b
   - **Diffusion models** Day 3

Goal of GenAI Workshop

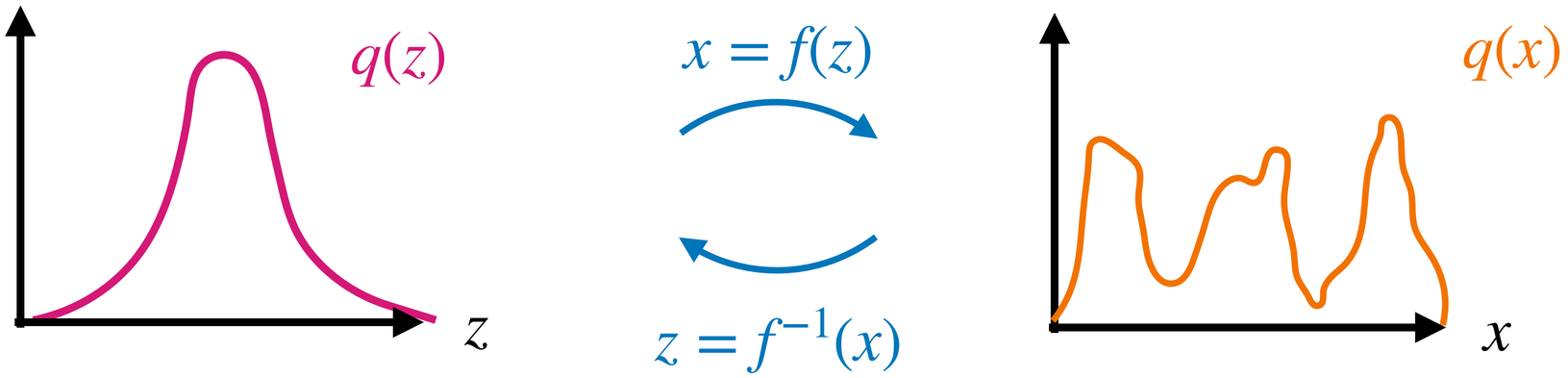
- How to train machine learning models to approximate complex distributions?
- Ideal model:
    - Flexible and expressive → High-quality samples
    - Easy to train and evaluate
    - Fast sampling and density evaluation

Normalizing Flows

Concept

- Simple distribution $q(z)$, e.g. Gaussian, Uniform, …
  → We know how to sample and evaluate the density!

- Transform z with invertible transformation $x = f(z)$
  → $q(x)$ is more complex density than $q(z)$

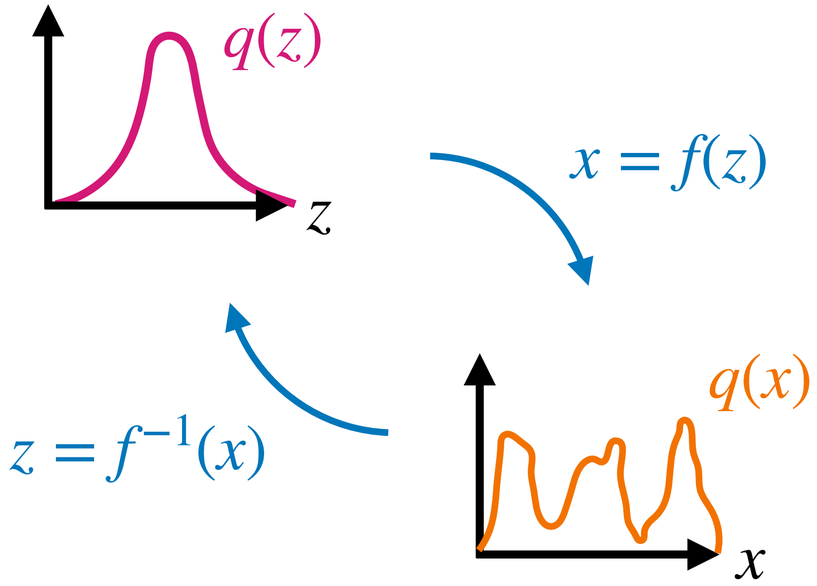


Change of Variables

- Change of variables formula:

$$q_X(x)\,\mathrm{d}x = q_Z(z)\,\mathrm{d}z$$

$$q_X(x) = q_Z(z)\,\frac{\mathrm{d}z}{\mathrm{d}x} = q_Z\big(f^{-1}(x)\big)\frac{\mathrm{d}f^{-1}(x)}{\mathrm{d}x}$$



- For arbitrary $\vec{x}, \vec{z} \in \mathbb{R}^D$:

$$q_X(\vec{x}) = q_Z\big(f^{-1}(\vec{x})\big) \cdot \det\left(\frac{\mathrm{d}f^{-1}(\vec{x})}{\mathrm{d}\vec{x}}\right) = q_Z\big(f^{-1}(\vec{x})\big) \cdot \det J_{f^{-1}}$$

$$= q_Z\big(f^{-1}(\vec{x})\big) \cdot \det J_f^{-1}$$

Change of Variables - Example

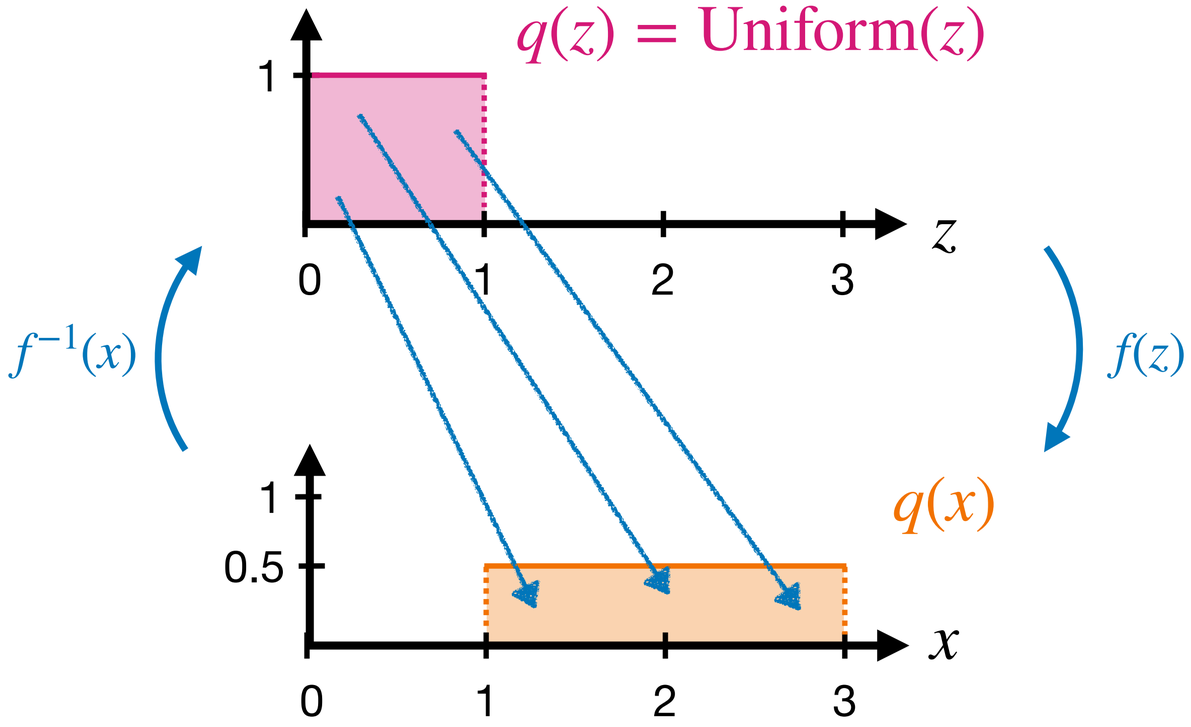
- Change of variables formula:

$$q_X(x) = q_Z(z)\,\frac{\mathrm{d}z}{\mathrm{d}x} = q_Z\big(f^{-1}(x)\big)\frac{\mathrm{d}f^{-1}(x)}{\mathrm{d}x}$$

- Example: $f(z) = 2z + 1$

$$f^{-1}(x) = \frac{1}{2}(x - 1)$$

$$\frac{\mathrm{d}f^{-1}(x)}{\mathrm{d}x} = \frac{1}{2}$$



Problem of Jacobian Determinant

- Expensive to compute $\det J_{f^{-1}}$ for large D
  → complexity for $D \times D$ Jacobian determinant is $\mathcal{O}(D^3)$

- How to reduce cost?
  → triangular Jacobian:

$$\det J_{f^{-1}} = \det \begin{pmatrix} \frac{\partial f^{-1}(x_1)}{\partial x_1} & 0 & \dots & 0 \\ \frac{\partial f^{-1}(x_2)}{\partial x_1} & \frac{\partial f^{-1}(x_2)}{\partial x_2} & & \vdots \\ \vdots & & \ddots & 0 \\ \frac{\partial f^{-1}(x_D)}{\partial x_1} & \dots & & \frac{\partial f^{-1}(x_D)}{\partial x_D} \end{pmatrix} = \prod_{d=1}^{D} \frac{\partial f^{-1}(x_d)}{\partial x_d}$$

  → $\mathcal{O}(D)$

Triangular Jacobian

- What does a triangular Jacobian mean for f?

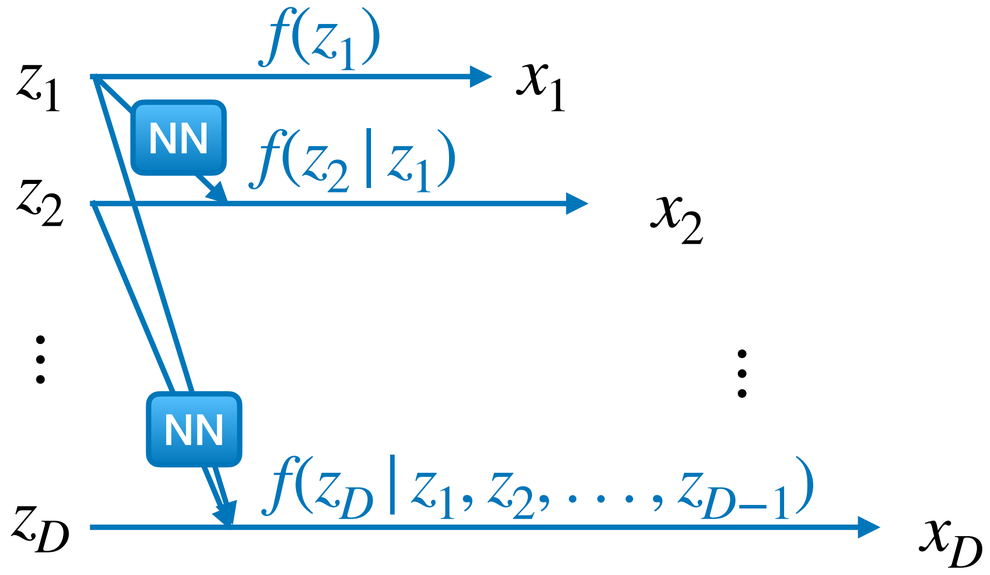
$$\det J_{f^{-1}} = \det \begin{pmatrix} \frac{\partial f^{-1}(x_1)}{\partial x_1} & 0 & \dots & 0 \\ \frac{\partial f^{-1}(x_2)}{\partial x_1} & \frac{\partial f^{-1}(x_2)}{\partial x_2} & & \vdots \\ \vdots & & \ddots & 0 \\ \frac{\partial f^{-1}(x_D)}{\partial x_1} & \dots & & \frac{\partial f^{-1}(x_D)}{\partial x_D} \end{pmatrix}$$

$f^{-1}(x_1)$ does not depend on $x_2, \dots, x_D$

$f^{-1}(x_D)$ depends on $x_1, x_2, \dots, x_{D-1}$

Autoregressive Approach: Sampling

- Model iterative dependency on previous dimensions:



For example: $z_i \sim \mathcal{N}(0,1)$

$$x_i = f(z_i) = \mu_i + \exp(\alpha_i) \cdot z_i$$

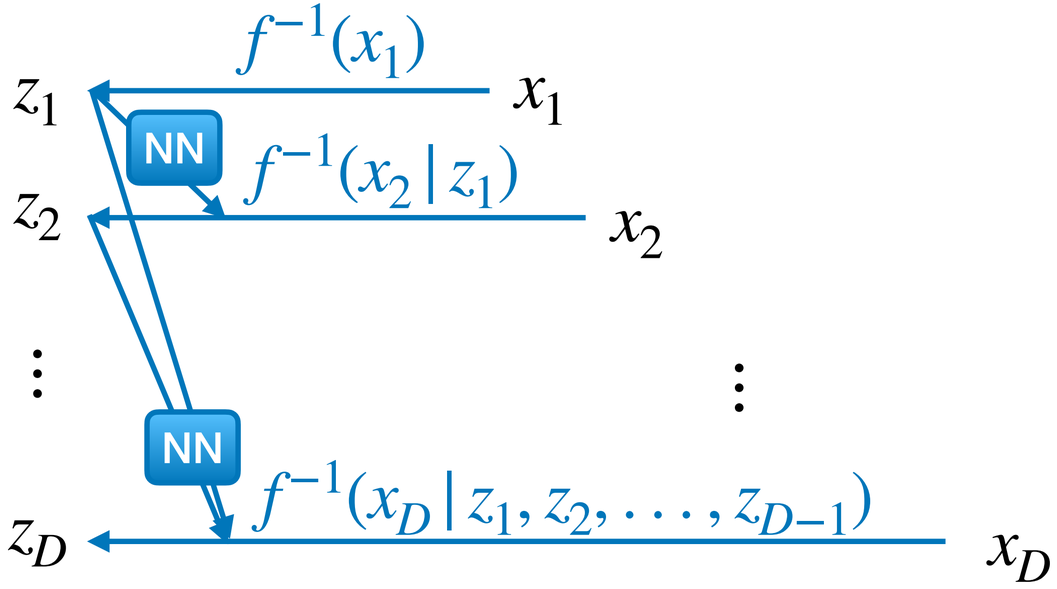
$$[\mu_i, \alpha_i] = \mathrm{NN}(z_{1:i-1})$$

- Fast sampling: Evaluating f can be parallelized $\rightarrow \mathcal{O}(1)$

Kingma et al.,Improved Variational Inference with Inverse Autoregressive Flow, NeurIPS 2016

Autoregressive Approach: Density evaluation

- Given sample $\vec{x} \in \mathbb{R}^D$



→ Iterative unwrapping of dimensions to obtain $\vec{z}$

→ Requires D evaluations of f^{-1}

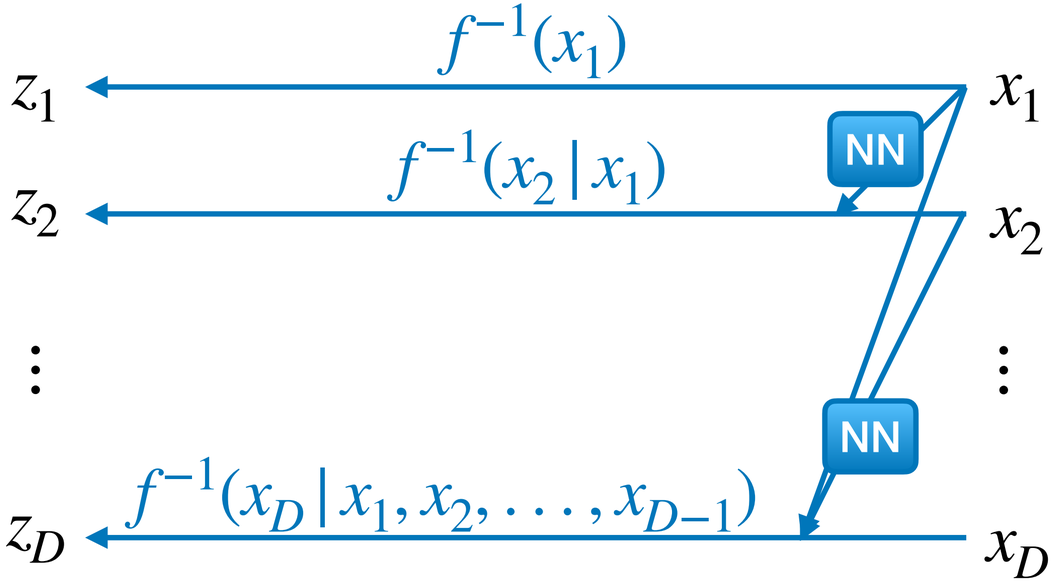
→ Complexity $\mathcal{O}(D)$

→ “Inverse Autoregressive Flow”

Kingma et al.,Improved Variational Inference with Inverse Autoregressive Flow, NeurIPS 2016

Autoregressive Approach: Density Evaluation

- Can we also build a flow with fast density evaluation?
  → Yes, “Masked Autoregressive Flow”



For example:

$$z_i \sim \mathcal{N}(0,1)$$

$$x_i = f(z_i) = \mu_i + \exp(\alpha_i) \cdot z_i$$

$$[\mu_i, \alpha_i] = \mathrm{NN}(x_{1:i-1})$$

$$p(x_1, \ldots, x_D) = p(x_1) \cdot p(x_2 \mid x_1) \cdot \ldots \cdot p(x_D \mid x_1, \ldots, x_{D-1})$$

Papamakarios, Pavlakou, and Murray, Masked Autoregressive Flow, NeurIPS 2017

Autoregressive Normalizing Flows

- Types:
  - Inverse Autoregressive Flow (IAF)
  - Masked Autoregressive Flow (MAF)

Complexity	Sampling	Density evaluation
Inverse Autoregressive Flow (IAF)	$\mathcal{O}(1)$	$\mathcal{O}(D)$
Masked Autoregressive Flow (MAF)	$\mathcal{O}(D)$	$\mathcal{O}(1)$

Kingma et al.,Improved Variational Inference with Inverse Autoregressive Flow, NeurIPS 2016

Papamakarios, Pavlakou, and Murray, Masked Autoregressive Flow, NeurIPS 2017

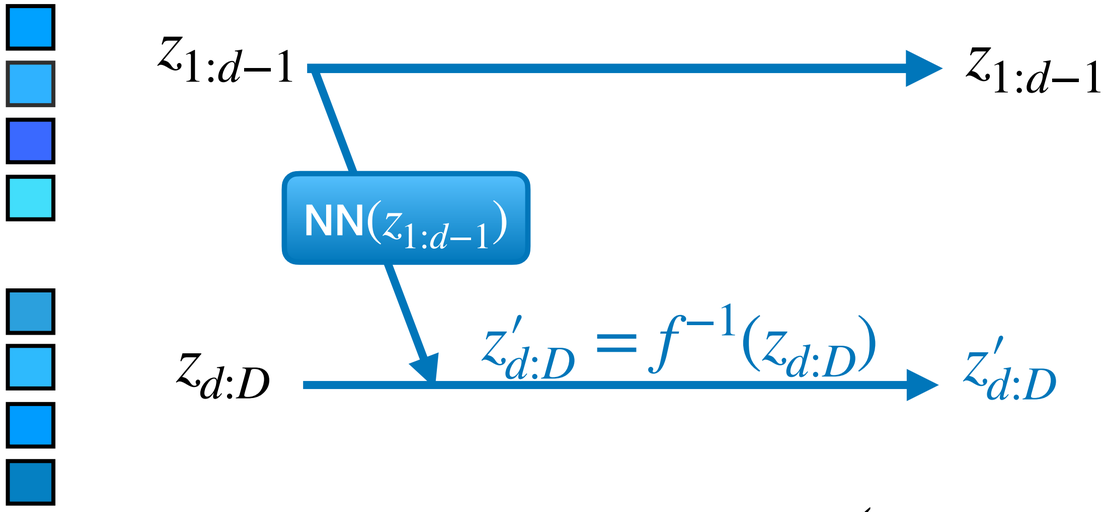
Goal: Fast sampling and density evaluation

- But we want $\mathcal{O}(1)$ sampling and $\mathcal{O}(1)$ density evaluation!

How to get fast sampling and density evaluation?
Any ideas?

Coupling Layers

- Further simplification of Jacobian
- Coupling layers: split $\vec{z} = (z_{1:d-1}, z_{d:D}) \in \mathbb{R}^D$

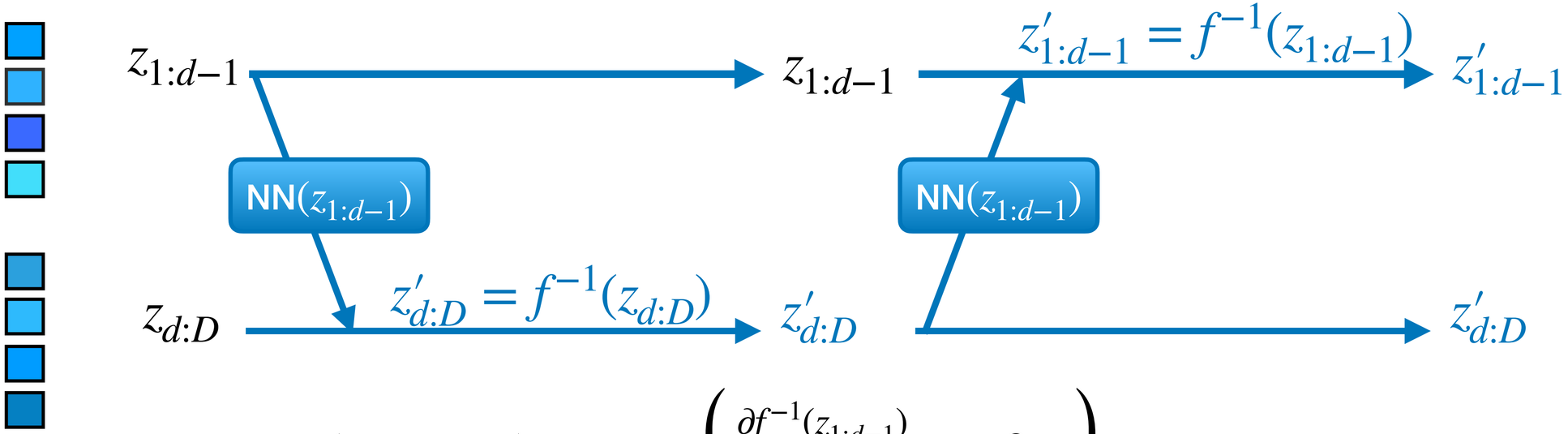


$$\det\left(\frac{\partial f^{-1}(\vec{z})}{\partial \vec{z}}\right) = \det\begin{pmatrix} \mathbb{I} & 0 \\ \frac{\partial f^{-1}(z_{d:D})}{\partial z_{1:d-1}} & \frac{\partial f^{-1}(z_{d:D})}{\partial z_{d:D}} \end{pmatrix} = \prod_{\tilde{d}=d}^{D} \frac{\partial f^{-1}(z_{\tilde{d}})}{\partial z_{\tilde{d}}}$$

Dinh, Krueger, and Bengio, NICE: Non-linear Independent Components Estimation, CoRR 2014

Coupling Layers

- Further simplification of Jacobian
- Coupling layers: split $\vec{z} = (z_{1:d-1}, z_{d:D}) \in \mathbb{R}^D$

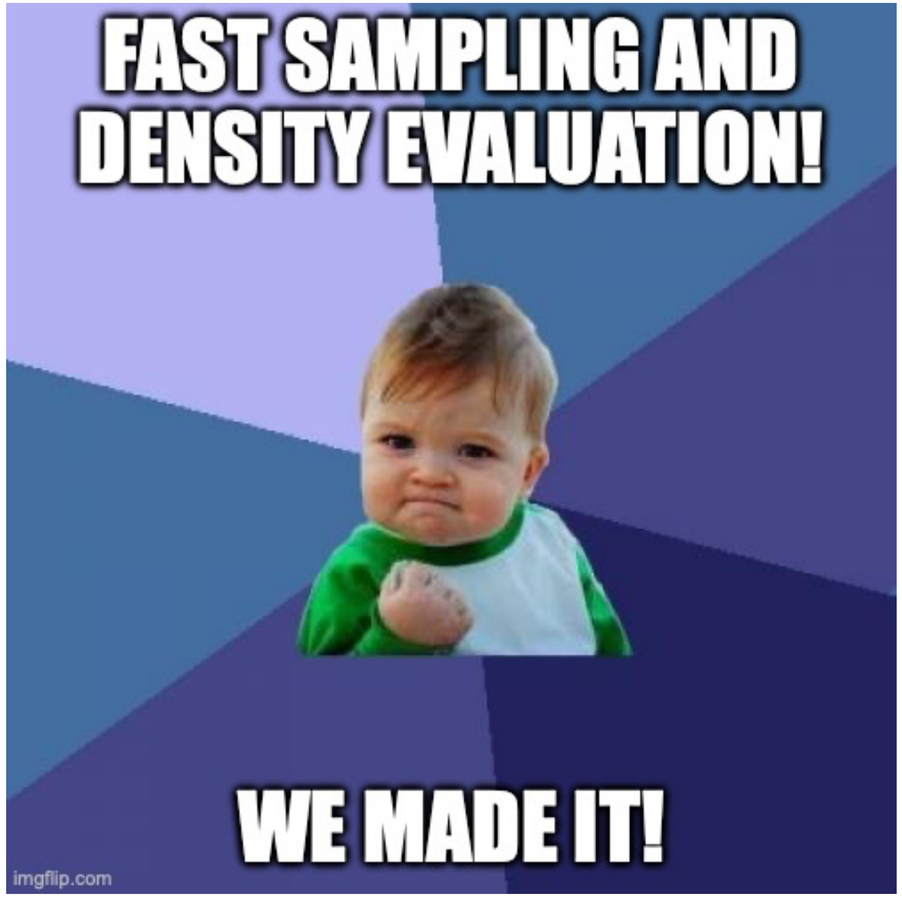


$$\det\left(\frac{\partial f^{-1}(\vec{z})}{\partial \vec{z}}\right) = \det\begin{pmatrix} \frac{\partial f^{-1}(z_{1:d-1})}{\partial z_{1:d-1}} & 0 \\ \frac{\partial f^{-1}(z_{d:D})}{\partial z_{1:d-1}} & \mathbb{I} \end{pmatrix} = \prod_{\tilde{d}=1}^{d-1} \frac{\partial f^{-1}(z_{\tilde{d}})}{\partial z_{\tilde{d}}}$$

Dinh, Krueger, and Bengio, NICE: Non-linear Independent Components Estimation, CoRR 2014

Coupling Layers

- Always apply multiple subsequent transformations
- Permute split after each transformation
  → All dimensions should influence each other
- Sampling and density evaluation is $\mathcal{O}(1)$!



Types of Normalizing Flows

Complexity	Sampling	Density evaluation
Inverse Autoregressive Flow (IAF)	$\mathcal{O}(1)$	$\mathcal{O}(D)$
Masked Autoregressive Flow (MAF)	$\mathcal{O}(D)$	$\mathcal{O}(1)$
Coupling Flow	$\mathcal{O}(1)$	$\mathcal{O}(1)$

How to design flow transformations?

- **Desired properties**:
  - Easy to invert and differentiable
  - Complex dependency on other (non-transformed) dimensions

- Example from autoregressive flow: $x_i = f(z_i) = \mu_i + \exp(\alpha_i) \cdot z_i$

$$[\mu_i, \alpha_i] = \mathrm{NN}(x_{1:i-1})$$

- Form of **affine/additive** transformation: $z'_{d:D} = \alpha \cdot z_{d:D} + \beta$

$$[\alpha, \beta] = \mathrm{NN}(z_{1:d-1})$$

How to design flow transformations?

- Transformations designed for coupling layers:
  - **affine/additive** Dinh, Krueger, and Bengio, NICE: Non-linear Independent Components Estimation, CoRR 2014
  - **RealNVP** Dinh, Sohl-Dickstein, Bengio, Density estimation using Real NVP, ICLR 2017
  - Quadratic, cubic, **rational-quadratic splines** Durkan et al., Neural Spline Flows, NeurIPS 2019
  - …
- Specific manifolds:
  - Planar or radial flows Rezende and Mohamed, Variational Inference with Normalizing Flows, PMLR 2015
  - Smooth normalizing flows Köhler, Krämer and Noe, Smooth Normalizing Flows, NeurIPS 2021
  - …

RealNVP

- realNVP = real-valued Non-Volume Preserving flow transformation
- Affine-like transformation:

$$z'_{d:D} = \exp\left(\underbrace{s_\theta(z_{1:d-1})}_{\text{Scale}}\right) \cdot z_{d:D} + \underbrace{t_\theta(z_{1:d-1})}_{\text{Translation}}$$

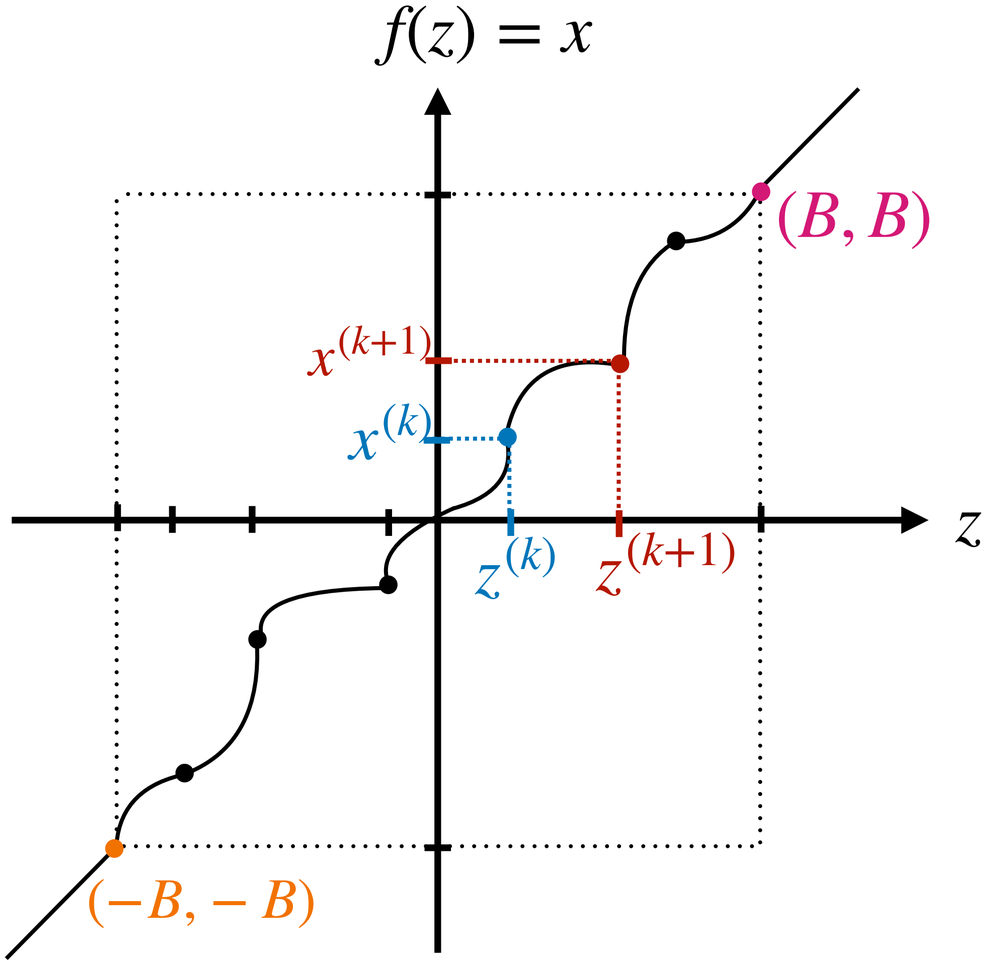
- Predict scale and translation with NN: $[s_\theta(z_{1:d-1}), t_\theta(z_{1:d-1})] = \mathrm{NN}_\theta(z_{1:d-1})$
- Jacobian determinant:

$$\det J_{f^{-1}} = \prod_{\tilde{d}=d}^{D} \frac{\partial f^{-1}(z_{\tilde{d}})}{\partial z_{\tilde{d}}} = \prod_{\tilde{d}=d}^{D} \exp\left(s_\theta(z_{1:d-1})\right)$$

Dinh, Sohl-Dickstein, Bengio, Density estimation using Real NVP, ICLR 2017

Rational-Quadratic Splines

- Define transform on limited interval $[-B, B]$
- Partition interval into K bins
- Define monotonically increasing rational-quadratic transformation in each bin



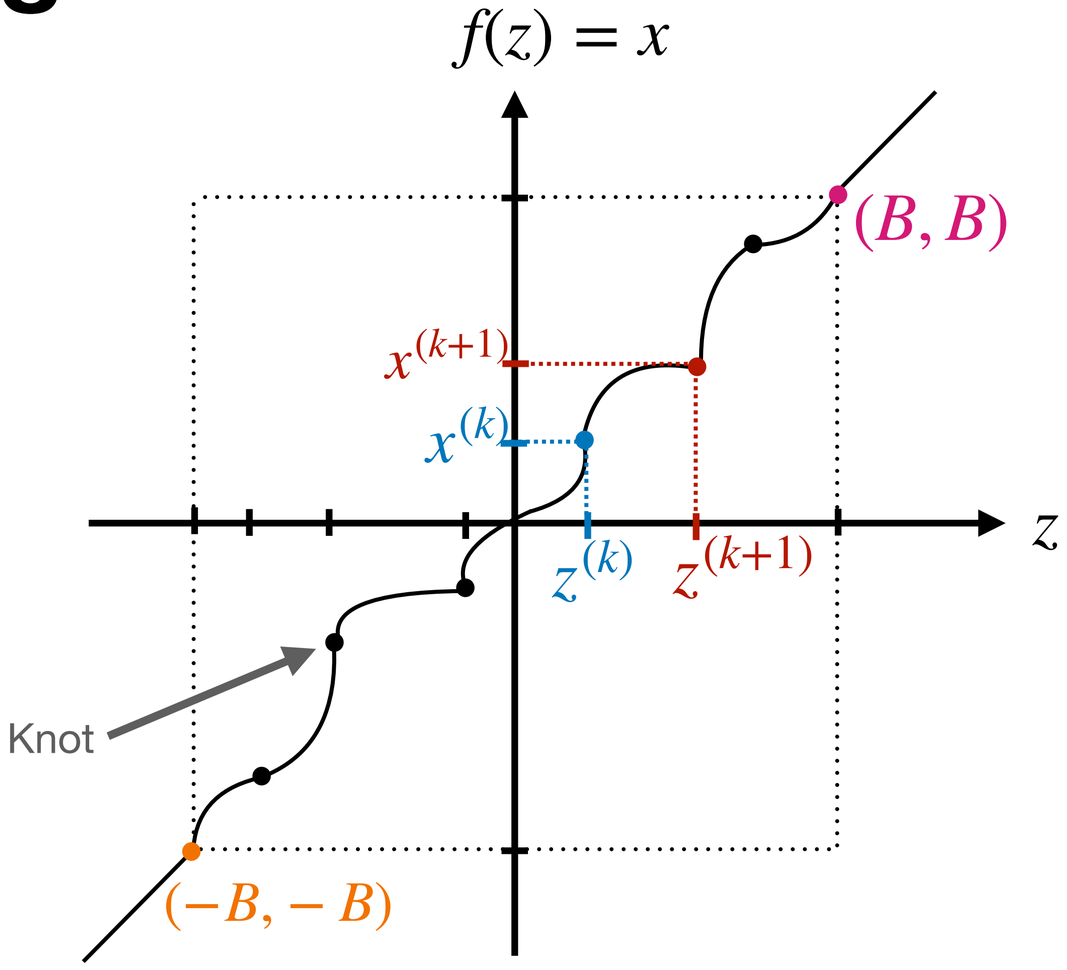
Durkan et al., Neural Spline Flows, NeurIPS 2019

Rational-Quadratic Splines

- NN parametrizes:
  - $\vec{b}_w \in \mathbb{R}^K$ bin widths
  - $\vec{b}_h \in \mathbb{R}^K$ bin heights
  - $\vec{b}_\delta \in \mathbb{R}^{K-1}$ derivatives at internal knot points
- Normalize NN output:

$$\vec{b}'_{w,h} = 2B \cdot \text{softmax}(\vec{b}_{w,h})$$

$$\vec{b}'_\delta = \text{softplus}(\vec{b}_\delta)$$



Durkan et al., Neural Spline Flows, NeurIPS 2019

Rational-Quadratic Splines

- Complicated formula for spline in each bin (not relevant here, see paper)
- Relevant properties:
  - Invertible
  - Differentiable
- Piece-wise approach results in expressive flows!

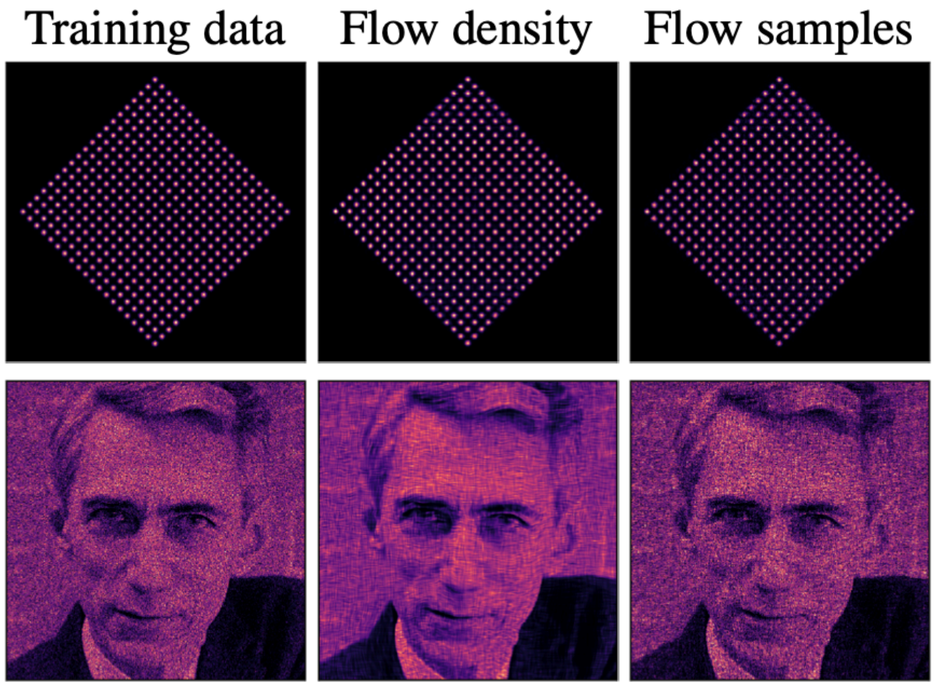


Figure 2: Qualitative results for two-dimensional synthetic datasets using RQ-NSF with two coupling layers.

Durkan et al., Neural Spline Flows, NeurIPS 2019

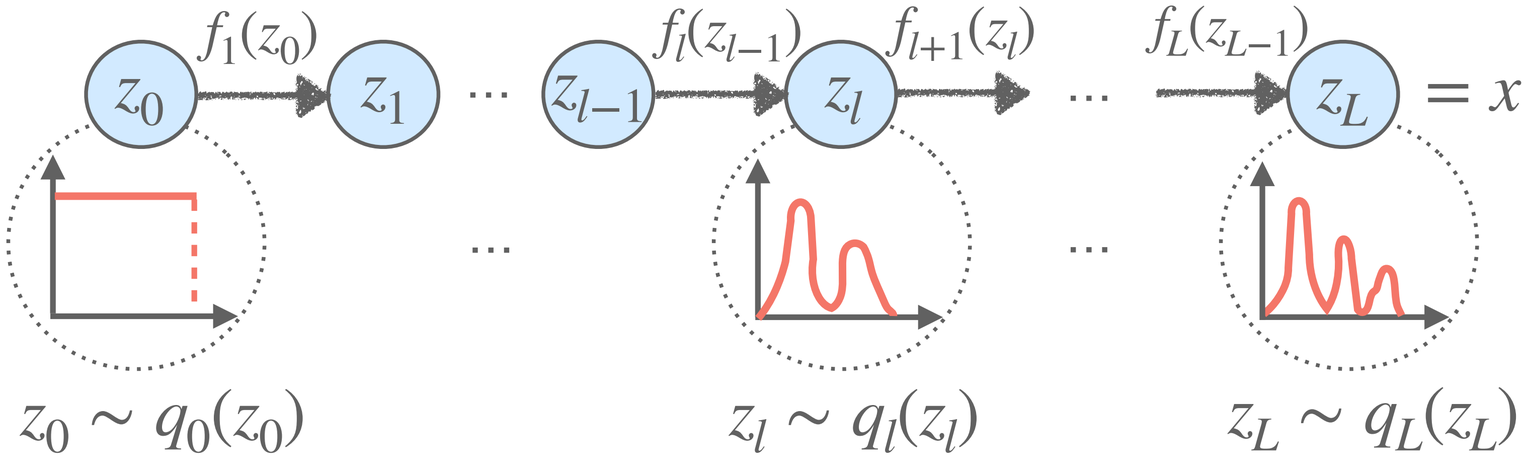
Chain of Transformations

Base Distribution
$q_0(z_0)$

Learned Flow Distribution
$q_\theta(x) = q_L(z_L)$

Chain of Transformations

Base Distribution
$q_0(z_0)$

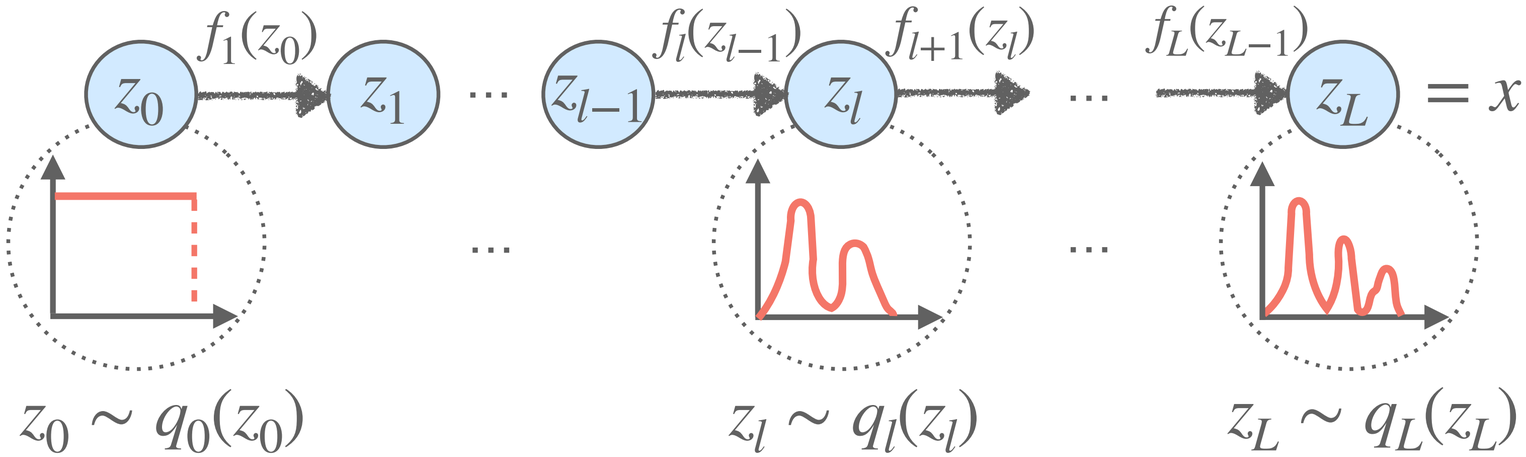


Learned Flow Distribution
$q_\theta(x) = q_L(z_L)$

Sampling: $z_0 \sim q_0(z_0)$ → $x = z_L = f_L \circ \ldots \circ f_1(z_0)$

Chain of Transformations

Base Distribution $q_0(z_0)$



Learned Flow Distribution $q_\theta(x) = q_L(z_L)$

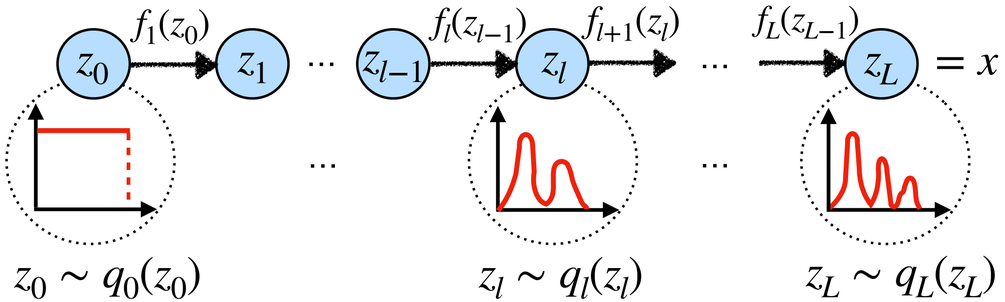
Evaluating the density: $z_0 = f_1^{-1} \circ \ldots \circ f_L^{-1}(x)$ ⟵ $x = z_L = f_L(z_{L-1})$

$$q(x) = q_0\big(f^{-1}(x)\big) \prod_{i=1}^{L} \left| \det \left(\frac{\partial f_i^{-1}(x)}{\partial x} \right) \right| = q_0(f^{-1}(x)) \prod_{i=1}^{L} \left| \det J_{f_i^{-1}} \right|$$

Example: Gaussian with two modes

Base Distribution

$q_0(z_0)$

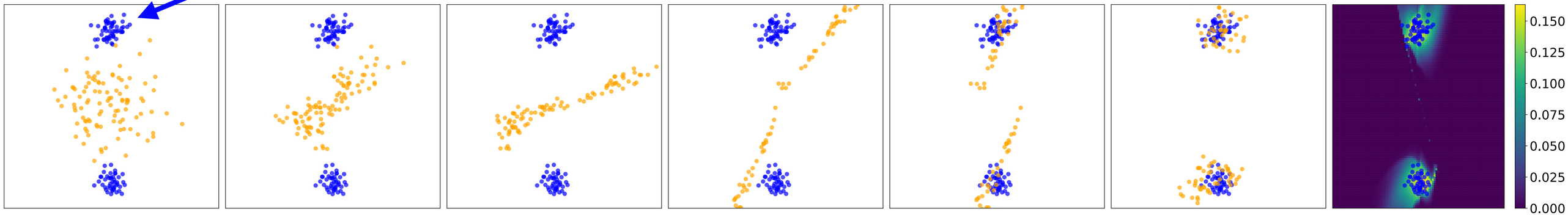


Learned Flow Distribution

$q_\theta(x) = q_L(z_L)$

Example:

Samples from target $p(x)$

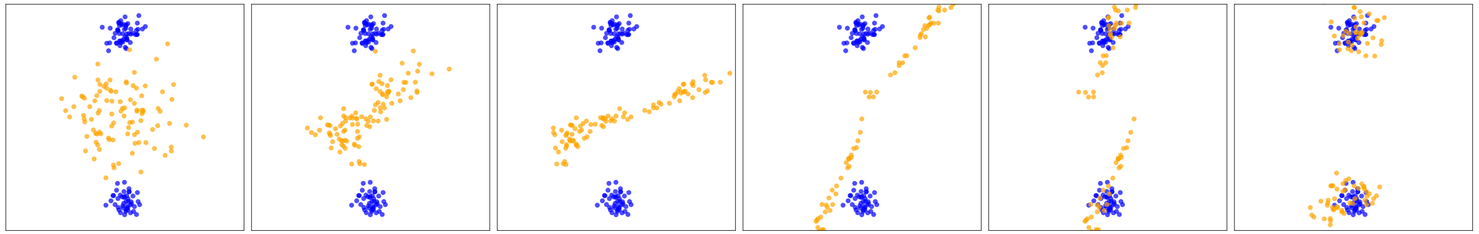


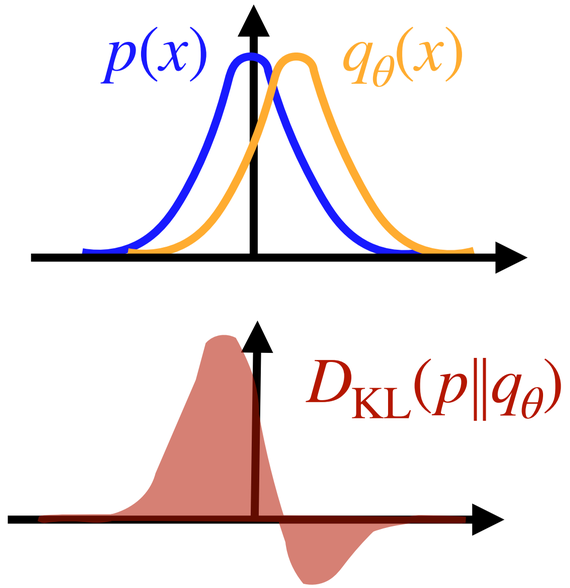
$z_0 \sim q_0(z_0)$

$z_5 \sim q_5(z_5)$ $q(x) = q_5(z_5)$

How to train a normalizing flow?

General question in generative modeling:
How can we compare two distributions?





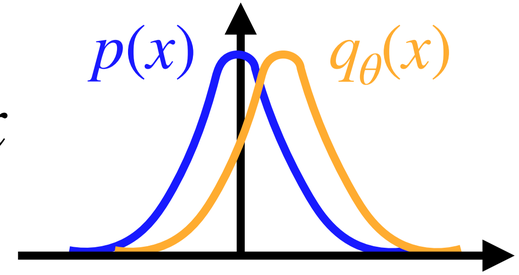
→ divergences from probability theory

- Divergence = difference measure in probability space
- Quantifies how dis-similar two distributions are → smaller is better

Standard choice: Kullback-Leibler divergence $D_{\mathrm{KL}}(p\,\|q_\theta) = \int p(x) \log \frac{p(x)}{q_\theta(x)}\,\mathrm{d}x$

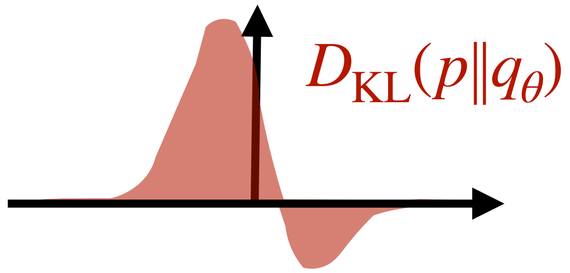
How to train a normalizing flow?

Kullback-Leibler divergence: $D_{\mathrm{KL}}(p \,\|\, q_\theta) = \int p(x) \log \frac{p(x)}{q_\theta(x)} \, \mathrm{d}x$



$$\mathscr{L} = D_{\mathrm{KL}}(p \,\|\, q_\theta) = \mathbb{E}_{x\sim p(x)}\left[\log \frac{p(x)}{q_\theta(x)}\right]$$

$$= \mathbb{E}_{x\sim p(x)}[\log p(x)] - \mathbb{E}_{x\sim p(x)}[\log q_\theta(x)]$$



$$\nabla_\theta \mathscr{L} = \nabla_\theta D_{\mathrm{KL}}(p \,\|\, q_\theta) = -\nabla_\theta \mathbb{E}_{x\sim p(x)}[\log q_\theta(x)]$$

$$= -\sum_{i=1}^{N} \nabla_\theta \log q_\theta(x_i) + \text{const}. \qquad \text{with } x_i \sim p(x_i)$$

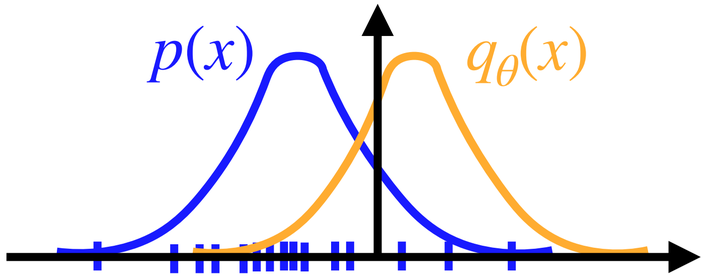
→ Loss is negative log-likelihood: $\mathscr{L} = -\sum_{i=1}^{N} \log q_\theta(x_i)$

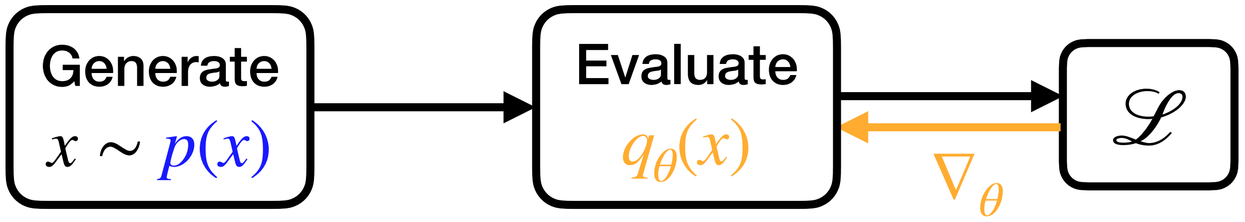
How to train a normalizing flow?

Loss is negative log-likelihood: $\mathscr{L} = -\sum_{i=1}^{N} \log q_\theta(x_i)$ with $x_i \sim p(x_i)$

Standard setting:

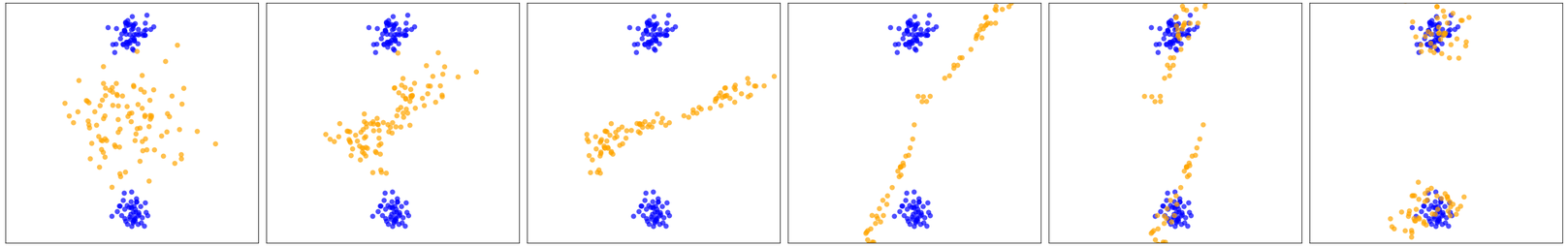
- (Simulated) data set available $\rightarrow x_i \sim p(x_i)$
- But $p(x)$ unknown/intractable





Normalizing flows: 3 Take-Aways

1. Normalizing flow = sequence of invertible, differentiable transformations



2. Transformations constructed to have tractable (i.e. triangular) Jacobian
   → Autoregressive vs. coupling layers
   → Different transformations (realNVP, RQS)
   → Exact density evaluation, but limited architecture

3. Trained by evaluating and minimizing the negative log-likelihood of training data samples

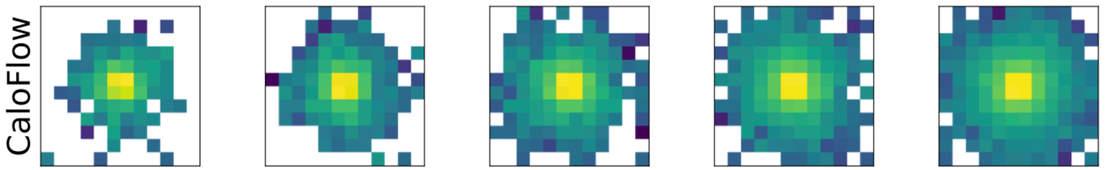
Normalizing flows in Physics

https://pollev.com/nicolehartman968

- What is x in your (or an arbitrary) physics example? What is its dimensionality?
- Can you generate a training dataset $x \sim p(x)$?
- Do you want to evaluate the density $q_\theta(x)$ in your example?

Example: CaloFlow

- x = hits in calorimeter detector
- Yes, we can run shower and detector simulations
- No → used MAF with RQS transformations



Krause and Shih, CaloFlow: Fast and Accurate Generation of Calorimeter Showers with Normalizing Flows, PRD2021

Conditional Normalizing Flows

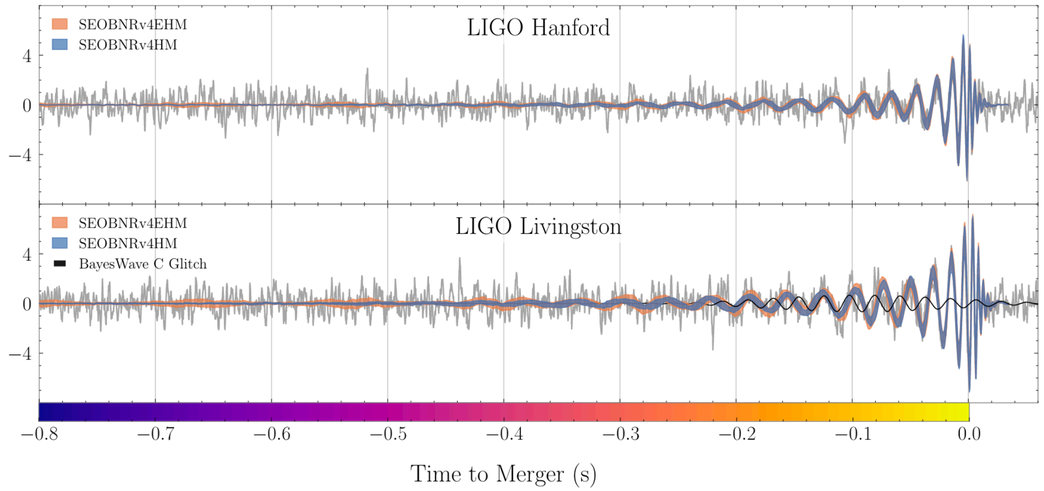
- So far: unconditional density estimation
- What about a posterior distribution?

$$p(\theta \mid d) = \frac{p(d \mid \theta)}{p(d)}\ p(\theta)$$

θ: physics parameters

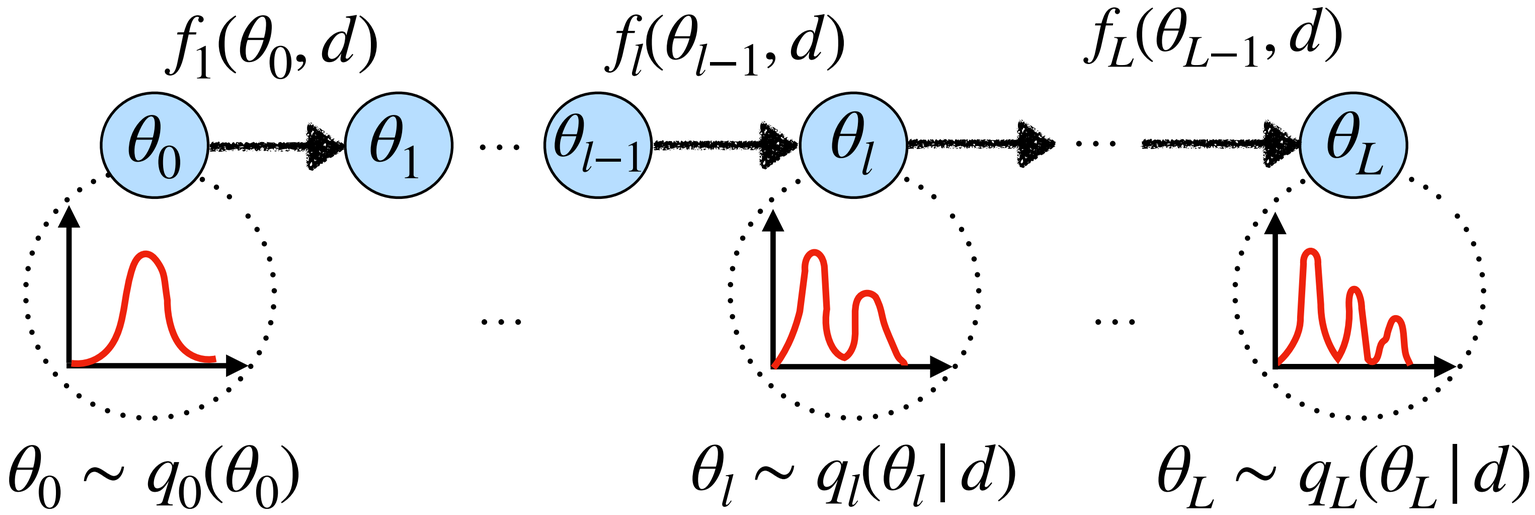
d: data

- Distribution over parameters θ conditional on data d
- Example: gravitational waves
  - θ: masses, spins, sky position
  - d: measured signal



Conditional Normalizing Flow

Prior
$q_0(\theta_0)$



Learned Posterior
$q(\theta \mid d)$

- How to include data in flow transformation f ?

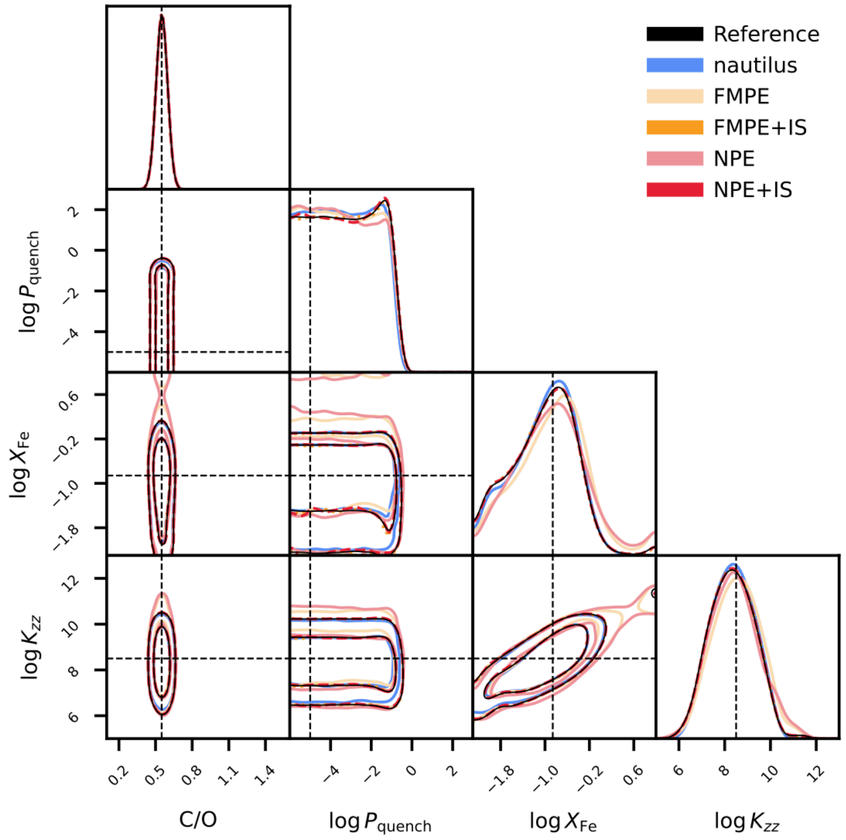
Conditional Normalizing Flow

- How to include data in flow transformation f ?
  - High-dimensional data (>100) → compress
  - Provide d to every flow transformation by
    - Concatenation to NN input
    - Scale to same size as NN input with linear layer and add with gated-linear unit (GLU)

Conditional Normalizing Flows in Physics

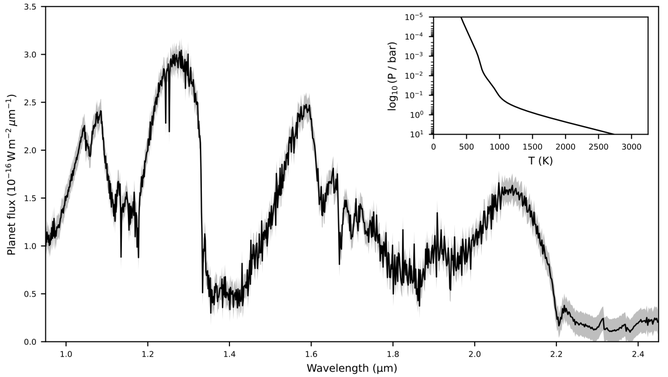
- What are parameters θ and data d in your (or an arbitrary) physics example? What are their dimensionalities?
- Can you generate a training dataset $\theta \sim p(\theta)\ d \sim p(d\,|\,\theta)$?
- Do you want to evaluate the density $q(\theta\,|\,d)$ in your example?

Conditional Normalizing Flows in Physics



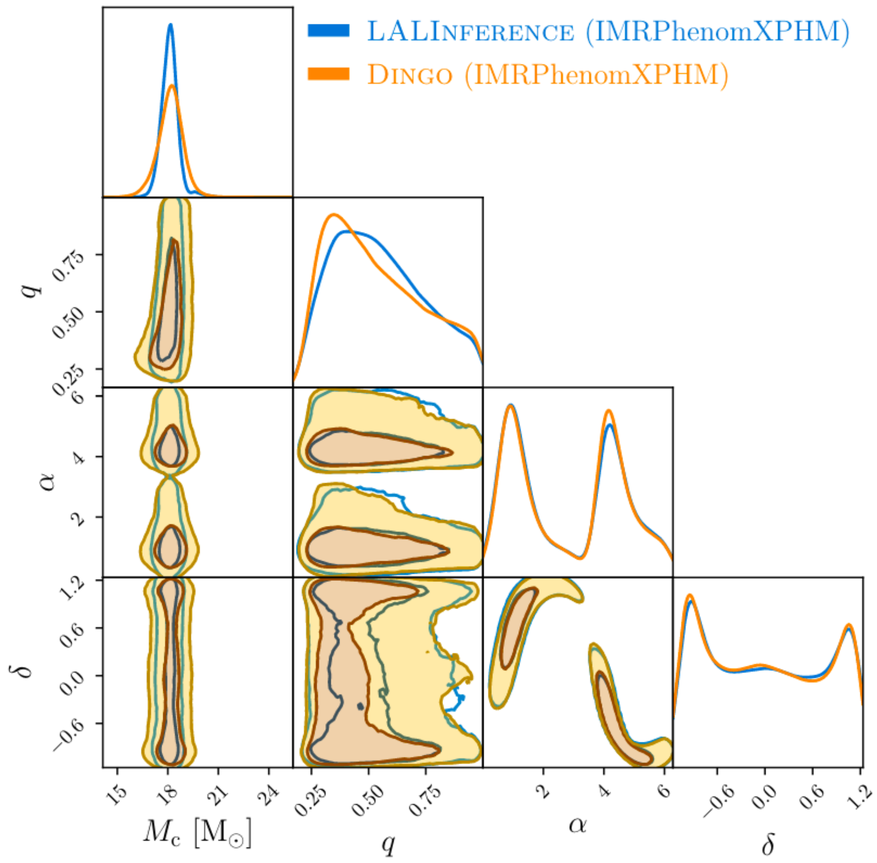
Example: Atmospheric properties of exoplanets

- θ = C/O ratio, metallicity, vertical diffusion coefficient, …
- d = emission spectrum
- Yes, we can run simulations
- Yes, because we want to run importance sampling



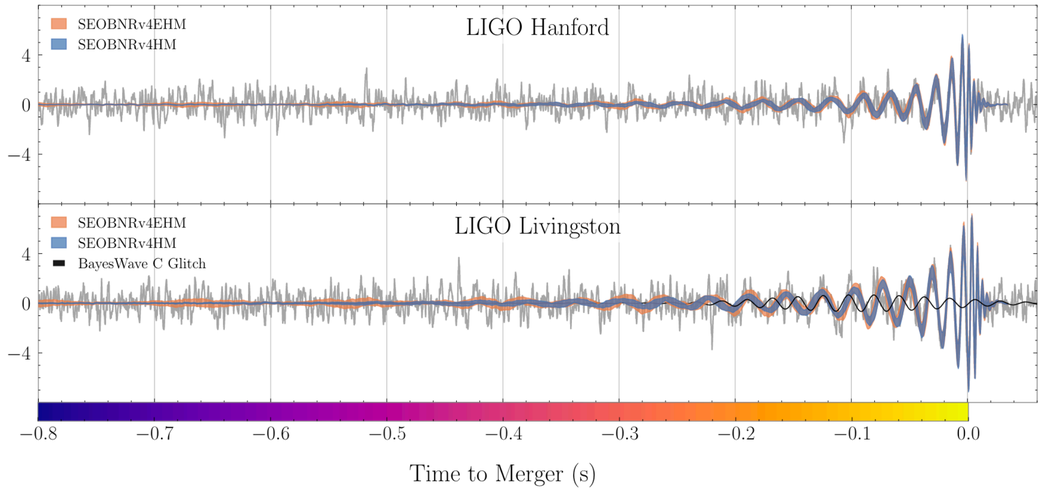
Gebhard et al., Inferring Atmospheric Properties of Exoplanets with Flow Matching and Neural Importance Sampling, AI2ASE 2024

Conditional Normalizing Flows in Physics



Example: Parameters of gravitational waves

- θ = masses, spins of black holes, sky position, …
- d = gravitational wave signal
- Yes, we can run simulations
- Yes, because we want to run importance sampling

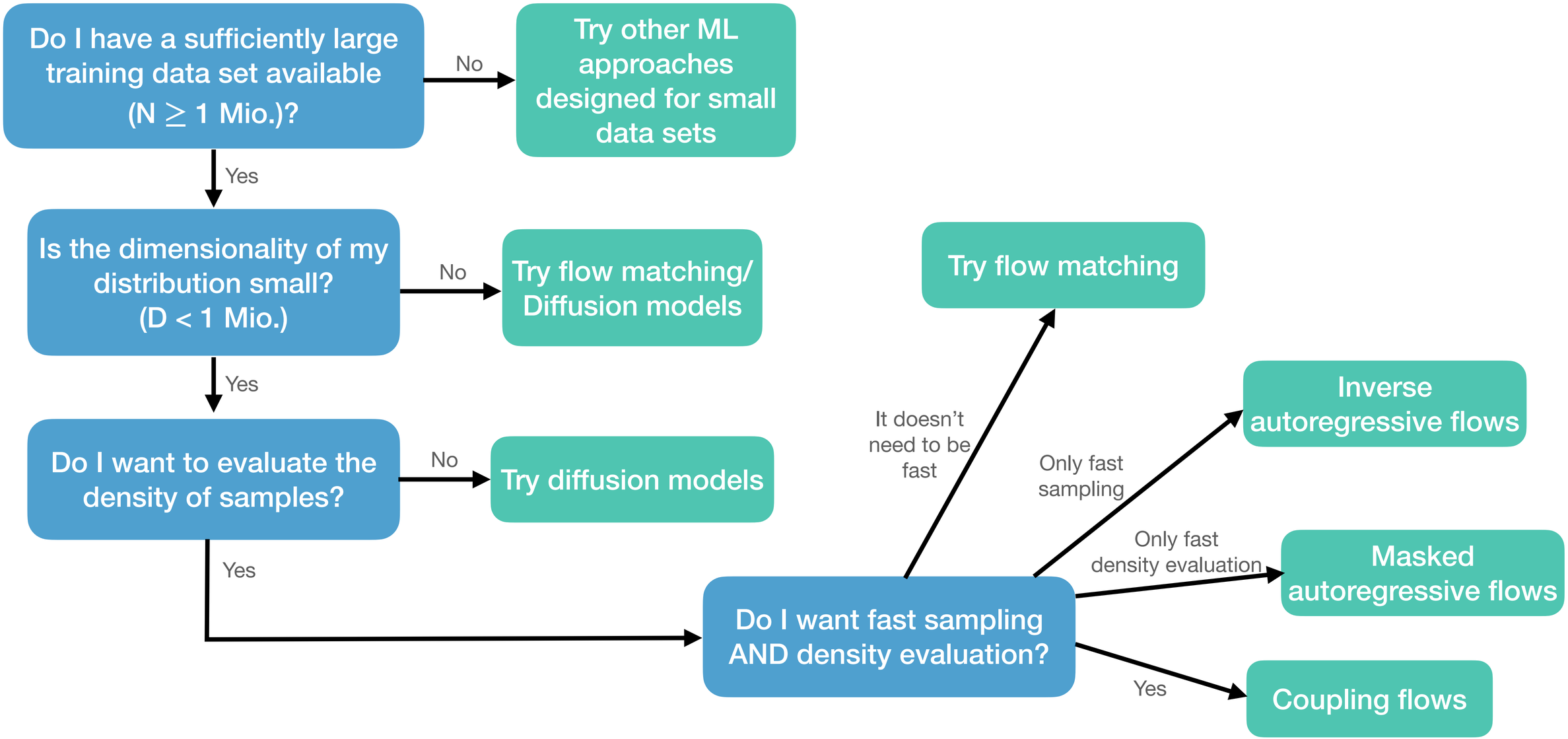


Dax et al., Neural Importance Sampling for Rapid and Reliable Gravitational-Wave Inference, PRL2023

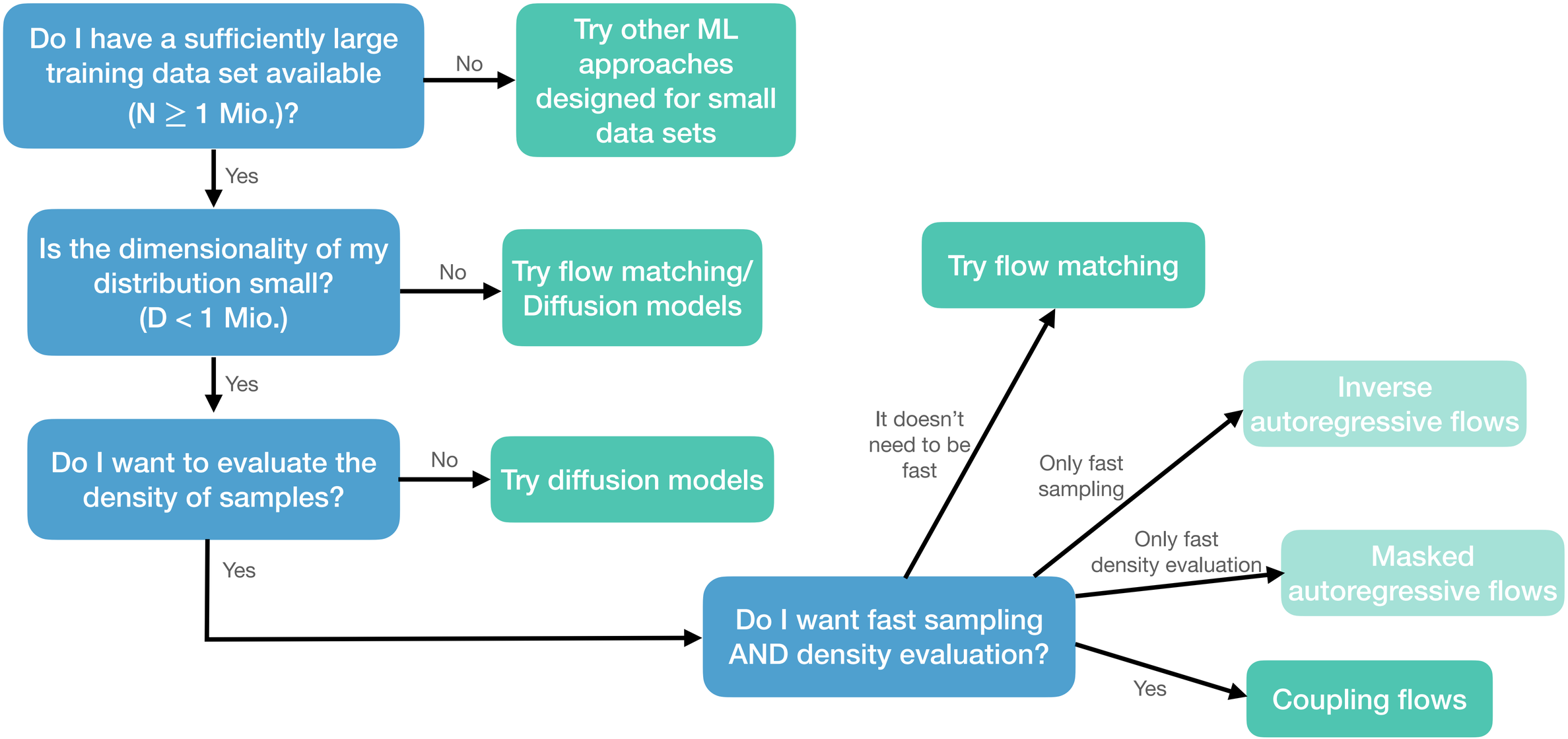
Tutorial: Conditional Normalizing Flow

- Goal: How to setup normalizing flow training?
- Realistic setting → easily transfer to your application
- Based on normflows/glasflows package
- Predict posterior distribution over the component masses of a gravitational wave
- $\theta = [m_1, m_2]$
- $d =$ gravitational wave strain

“Flow” chart: Should I use a normalizing flow?



“Flow” chart: Should I use a normalizing flow?



Thank you!

Do you have any questions?

Appendix

Recommended References for the Interested

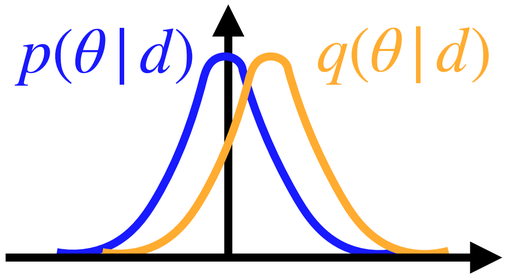
- Rezende and Mohamed, “Variational Inference with Normalizing Flows”, 2016
- Kobyzev, et al., “Normalizing Flows: An Introduction and Review of Current Methods”, 2021
-

Importance Sampling

Bayes theorem: $p(d\,|\,\theta)p(\theta) = p(\theta\,|\,d)p(d)$

- Evidence estimation:

$$p(d) = \int p(d\,|\,\theta)p(\theta)\ \mathrm{d}\theta = \int q(\theta\,|\,d)\frac{p(d\,|\,\theta)p(\theta)}{q(\theta\,|\,d)}\ \mathrm{d}\theta$$



- Importance weight:

$$w_i \propto \frac{p(d\,|\,\theta_i)p(\theta_i)}{q(\theta_i\,|\,d)}$$

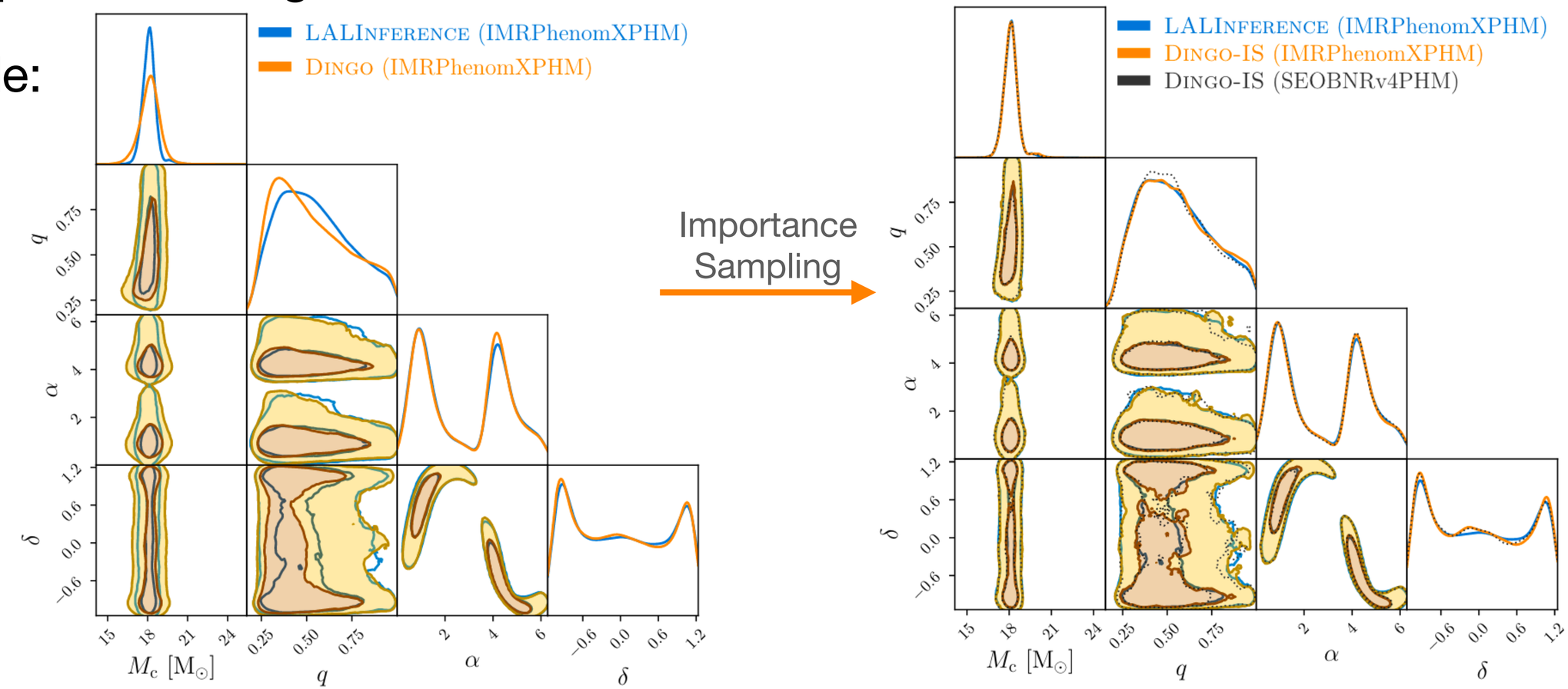
Likelihood · Prior

Proposal (Flow)

- Reweight posterior samples $x \sim q(\theta\,|\,d)$ towards true posterior with importance weights

Importance Sampling

- Reweight posterior samples $x \sim q(\theta \mid d)$ towards true posterior with importance weights
- Example:



Dax et al., Neural Importance Sampling for Rapid and Reliable Gravitational-Wave Inference, PRL2023