Normalising Flows

Gavin Kerrigan
2025 November 25

gavin.kerrigan@stats.ox.ac.uk
GavinKerrigan.github.io/teaching

Slides available here!

Motivation: Generative Models

Unconditional generative models learn $p(x)$ from data.

Motivation: Generative Models

Unconditional generative models learn $p(x)$ from data.

Common use cases:

1. Sampling

Given a collection of data, how can I learn to produce new data with similar properties?

Example: generating molecules

[Zhai 2025]

Motivation: Generative Models

Unconditional generative models learn $p(x)$ from data.

Common use cases:

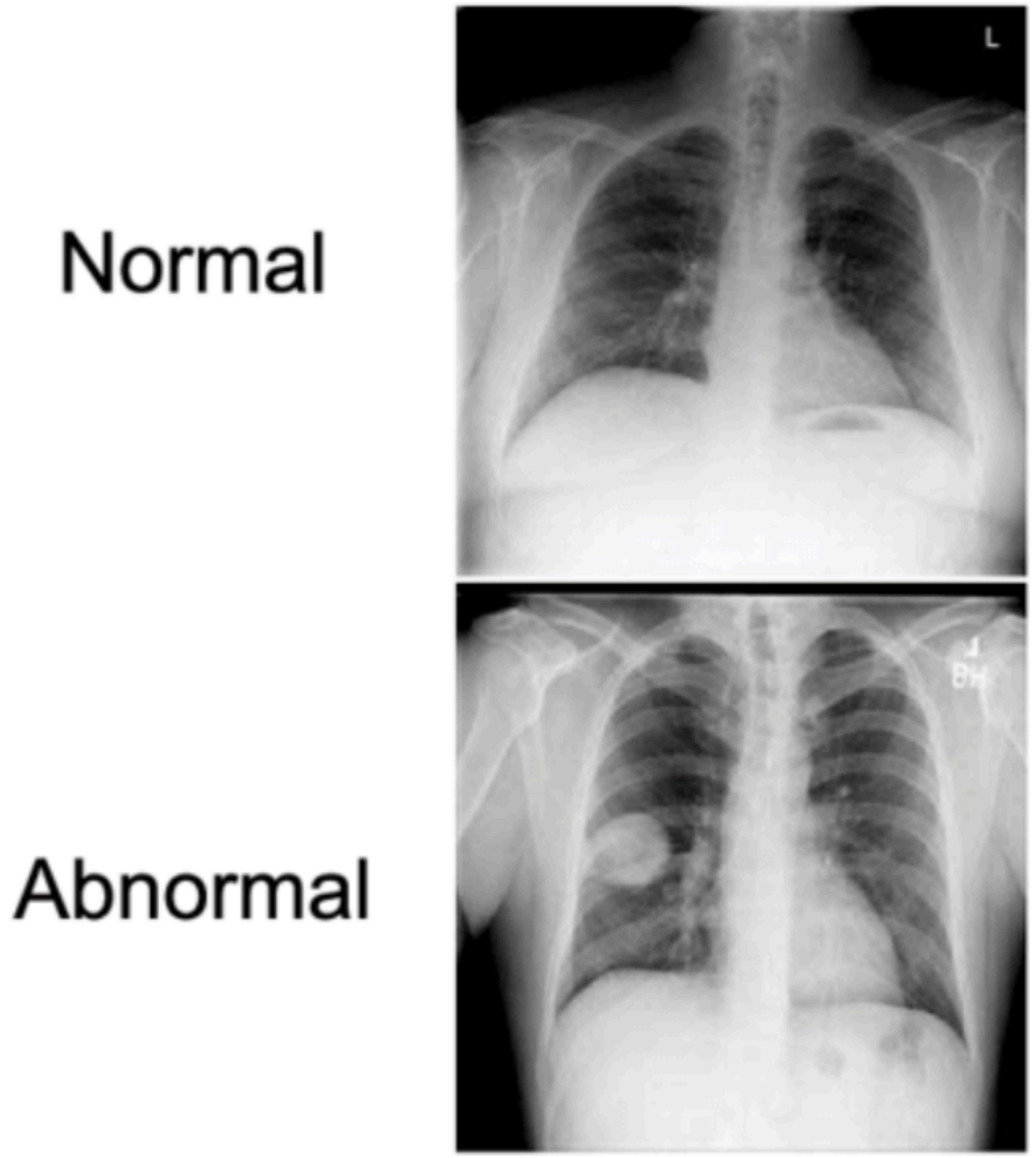
1. Sampling

2. Anomaly detection

Given a collection of "normal" data, how can I detect if a new datapoint is "abnormal"?

Example: learn a density $p(x)$ for normal images, and evaluate $p(x')$ on a test image.

[Nakao 2021]



Motivation: Generative Models

Unconditional generative models learn $p(x)$ from data.

Common use cases:

1. Sampling

2. Anomaly detection

3. Inverse problems

Can I recover an underlying signal from a partial measurement?

Example: super-resolution, inpainting, …

[Saharia 2023]

y x $\hat{x}$

Flows in 2025?

Normalising flows are a class of generative models that learn $p(x)$ **explicitly**.

… why are we still talking about them when we have diffusion?

1. Directly related to **density estimation**

   Example uses: Anomaly detection, Bayesian inference, compression, …

2. Relatively efficient

   Can sample with a single forward pass

   Simple models can be effective for low-dimensional problems

3. Diffusion is secretly a high-tech normalising flow

   Important for understanding and historical context

Flows in 2025?

Folklore: “Normalising flows are good at density estimation but bad at sampling.”

SOTA Flow in 2017 (RealNVP)

SOTA Flow in 2024 (TarFlow)

Chapter 1
Discrete-Time Flows

The Basic Idea

Setting the stage:

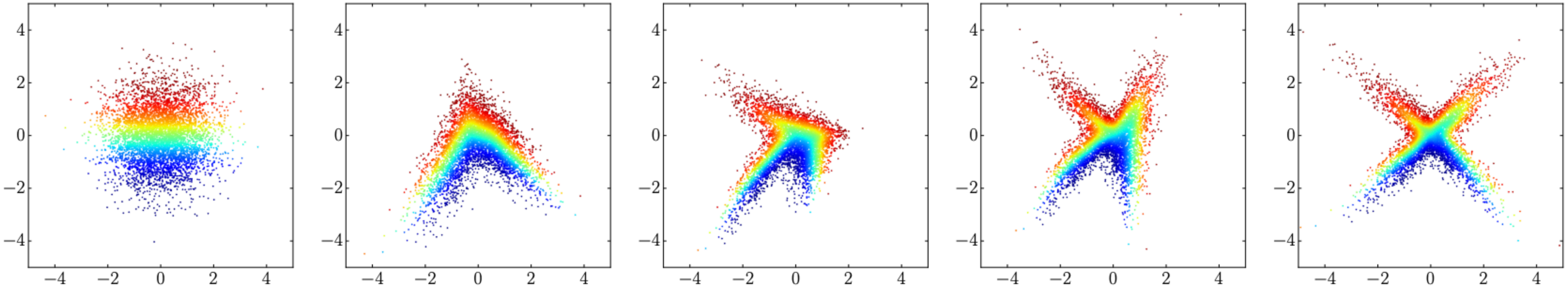
- All random variables are assumed to be continuous vectors in $\mathbb{R}^d$.

Question: how can we get a highly expressive $p(x)$?

Simple parametric distributions are not enough.

Key idea: transform samples from a simple distribution!

[Papamakarios 2021]



The Basic Idea

Choose a **base distribution** $p_0(x_0)$

Requirements:

1. Easy to sample $x_0 \sim p_0$
2. Easy to evaluate $p_0(x)$

Most common choice: $p_0(x_0) = \mathcal{N}\left(x_0 \mid 0, I\right)$

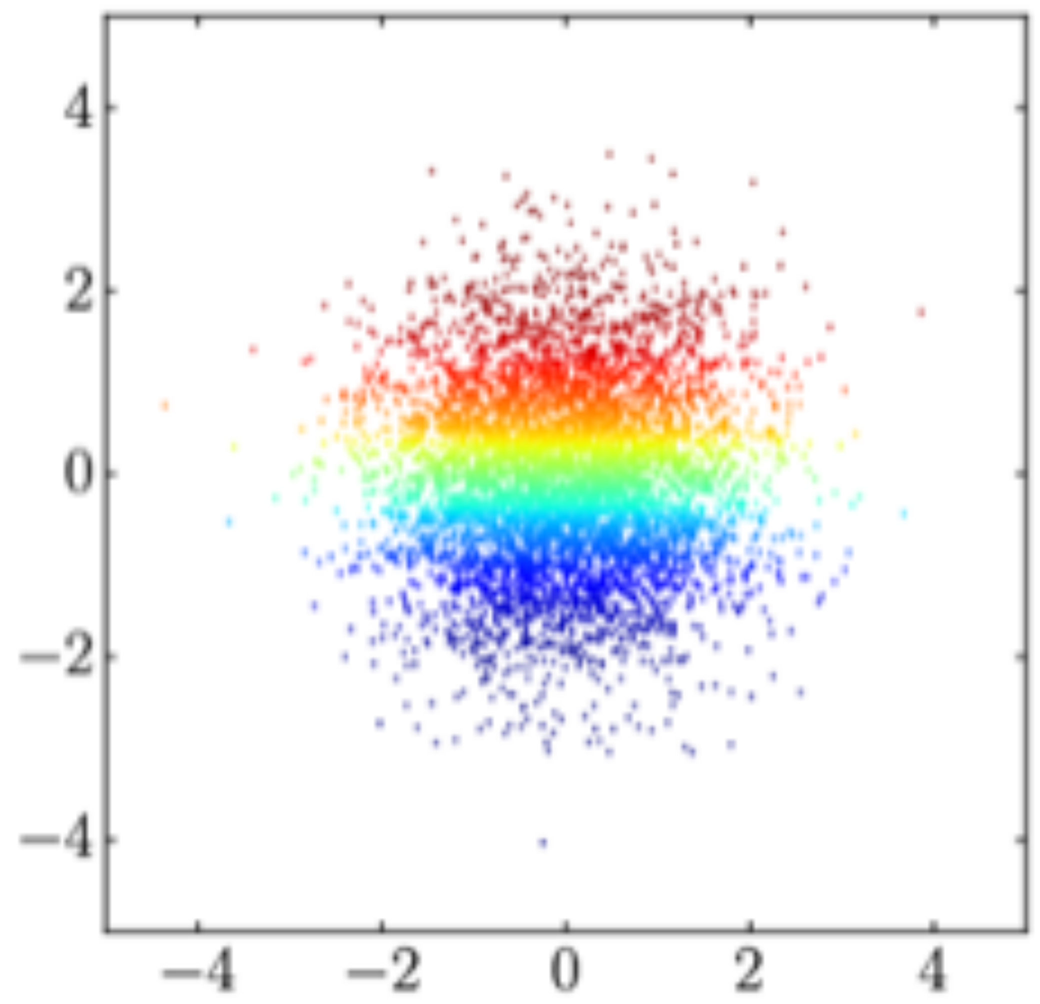
Choose a transformation $T : \mathbb{R}^d \to \mathbb{R}^d$

Given $x_0 \sim p_0$, we obtain a transformed version $x_1 = T(x_0)$

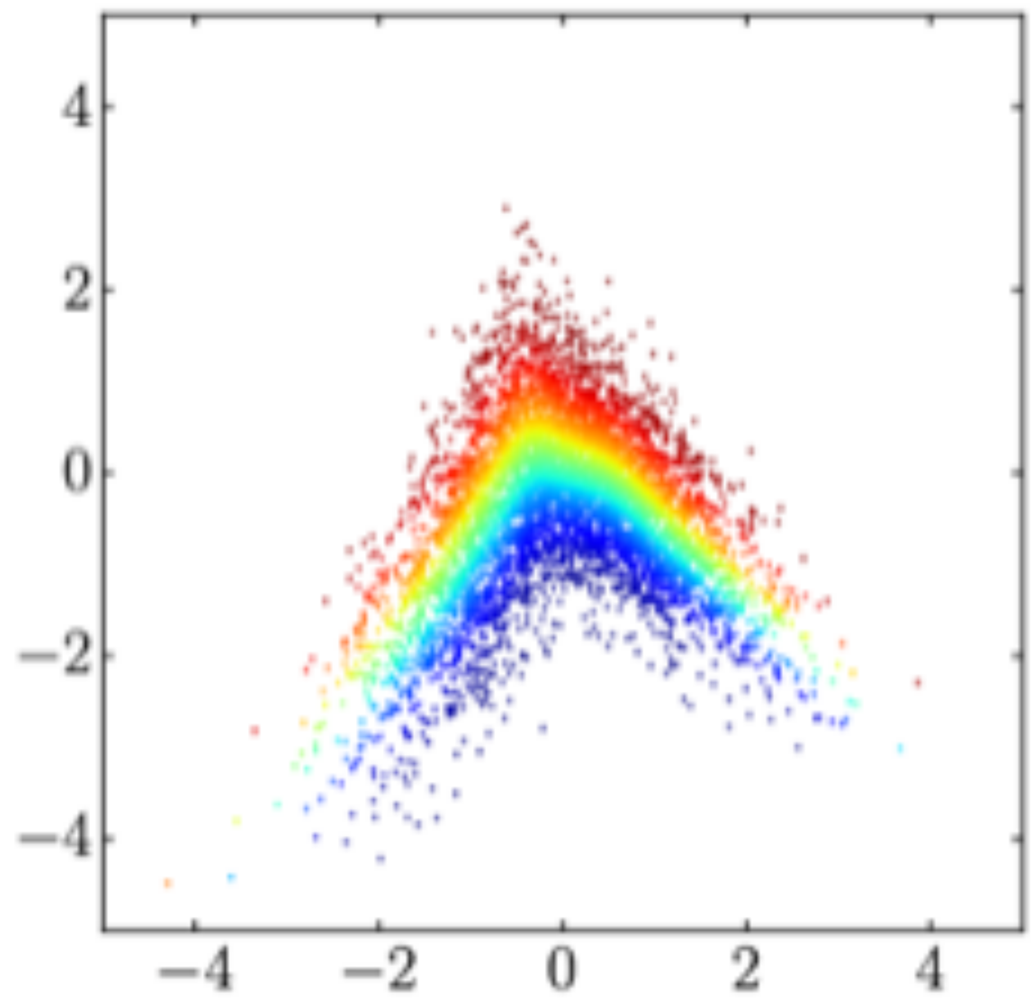
If I sample many x_0's, I get a **new distribution** $p_1(x_1)$

Question: what is the density $p_1(x_1)$?

$x_0 \sim p_0$



$x_1 \sim p_1$



Pushforwards

$x_0 \sim p_0 \qquad x_1 = T(x_0)$

$$J_T(x_0) = \begin{bmatrix} \frac{\partial T^{(1)}}{\partial x_0^{(1)}} & \cdots & \frac{\partial T^{(1)}}{\partial x_0^{(d)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial T^{(d)}}{\partial x_0^{(1)}} & \cdots & \frac{\partial T^{(d)}}{\partial x_0^{(d)}} \end{bmatrix}$$

Question: what is the density $p_1(x_1)$?

Called the **pushforward** of p_0 along T, denoted $p_1 = T_\# p_0$

Proposition [Change-of-Variables].

If T is invertible and T, T^{-1} are differentiable, then

$$p_1(x_1) = p_0(x_0)|\det J_T(x_0)|^{-1} \quad \text{where} \quad x_0 = T^{-1}(x_1)$$

$$= p_0\left(T^{-1}(x_1)\right)|\det J_{T^{-1}}(x_1)|$$

Intuition: normalise how likely the "source point" x_0 is by how much T stretches out the space

Diffeomorphisms

An invertible transformation T with T, T^{-1} differentiable is called a **diffeomorphism**

Diffeomorphisms are nice.

Sidebar: they set of diffeomorphisms on a space forms a group.

Given diffeomorphisms T_1, T_2, their **composition** $T_2 \circ T_1$ is a diffeomorphism

Inverse: $$(T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1}$$

Jacobian: $$\det J_{T_2 \circ T_1}(x) = \det J_{T_2}(T_1(x)) \cdot \det J_{T_1}(x)$$

$$\begin{tikzcd} \bullet \arrow[r, "T_1"] \arrow[rr, bend left, "T_2 \circ T_1"] & \bullet \arrow[r, "T_2"] & \bullet \end{tikzcd}$$

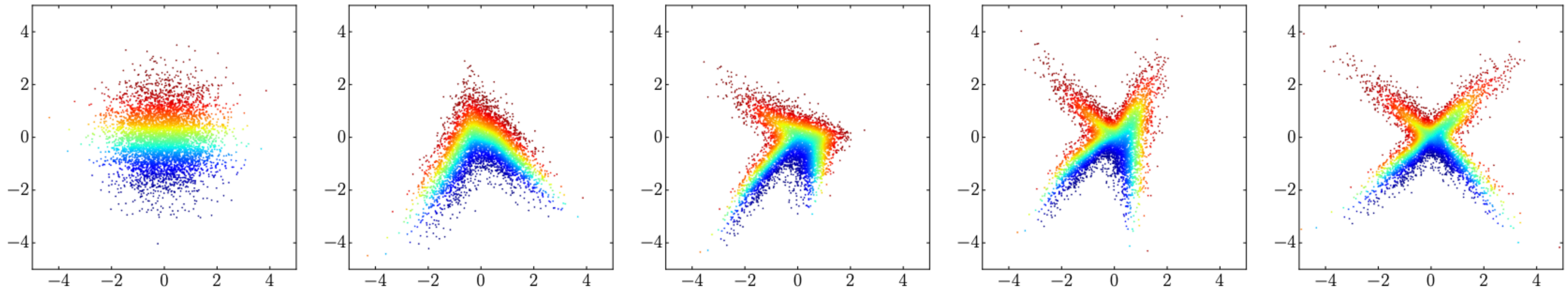
Questions?

Diffeomorphisms

We can stack together many small transformations to get a big transformation.

If each step is a diffeomorphism, we can compute the resulting density.

$x_0 \sim p_0$ $\quad$ $x_1 \sim p_1 = T_{1\#}p_0$ $\quad$ $x_2 \sim p_2 = T_{2\#}p_1 = (T_2 \circ T_1)_\# p_0$ $\quad$ • • • $\quad$ $x_L \sim (T_L \circ \ldots \circ T_1)_\# p_0$



[Papamakarios 2021]

Discrete-Time Normalising Flows

Have **data samples** $x^{(i)} \sim q(x)$ for $i = 1,2,\ldots,n$

Choose a **base distribution** $p_0(x_0)$

Implement some **diffeomorphisms** $T_{k,\theta} : \mathbb{R}^d \to \mathbb{R}^d$

Typically: a specific kind layer (or block) of a neural network

Layer index k

θ represents *all* parameters (which differ across layers)

Overall transformation: full neural network, stacking each block

$$T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$$

Defines a model distribution $p_\theta = T_{\theta\#} p_0$

Discrete-Time Normalising Flows

Have:

data samples $x^{(i)} \sim q(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#} p_0$

Goal: $p_\theta(x) \approx q(x)$

Let's minimise a **discrepancy** between $p_\theta(x)$ and $q(x)$

Almost always it will be the KL:

$$\mathscr{L}(\theta) = \mathrm{KL}\left[\, q(x) \,||\, p_\theta(x) \right]$$

Discrete-Time Normalising Flows

Have:

data samples $x^{(i)} \sim q(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#} p_0$

Goal: $p_\theta(x) \approx q(x)$

Let's minimise a **discrepancy** between $p_\theta(x)$ and $q(x)$

Almost always it will be the KL:

$$\mathscr{L}(\theta) = \mathrm{KL}\left[\, q(x) \,||\, p_\theta(x) \,\right] \qquad =_{+C} -\mathbb{E}_{q(x)}\left[\log p_\theta(x)\right]$$

Discrete-Time Normalising Flows

Have:

data samples $x^{(i)} \sim q(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#} p_0$

Goal: $p_\theta(x) \approx q(x)$

Let's minimise a **discrepancy** between $p_\theta(x)$ and $q(x)$

Almost always it will be the KL:

$$\mathscr{L}(\theta) = \text{KL}\left[q(x)\,||\,p_\theta(x)\right] \quad =_{+C} -\mathbb{E}_{q(x)}\left[\log p_\theta(x)\right]$$

Change-of-Variables:

$$p_\theta(x) = p_0(T_\theta^{-1}(x))|\det J_{T_\theta}^{-1}(x)|^{-1}$$

$$= -\mathbb{E}_{q(x)}\left[\log p_0\left(T_\theta^{-1}(x)\right) + \log|\det J_{T_\theta^{-1}}(x)|\right]$$

Discrete-Time Normalising Flows

Have:

data samples $x^{(i)} \sim q(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#} p_0$

Goal: $p_\theta(x) \approx q(x)$

Let's minimise a **discrepancy** between $p_\theta(x)$ and $q(x)$

Almost always it will be the KL:

$$\mathscr{L}(\theta) = \mathrm{KL}\left[q(x) \,||\, p_\theta(x) \right] \quad =_{+C} -\mathbb{E}_{q(x)}\left[\log p_\theta(x) \right]$$

Change-of-Variables:

$$p_\theta\left(x\right) = p_0(T_\theta^{-1}(x))|\det J_{T_\theta}^{-1}(x)|^{-1}$$

$$= -\mathbb{E}_{q(x)}\left[\log p_0\left(T_\theta^{-1}(x)\right) + \log|\det J_{T_\theta^{-1}}(x)| \right]$$

Monte Carlo estimate

$$\approx -\frac{1}{n}\sum_{i=1}^{n} \log p_0\left(T_\theta^{-1}(x^{(i)})\right) + \log|\det J_{T_\theta^{-1}}(x^{(i)})|$$

Discrete-Time Normalising Flows

Have:

data samples $x^{(i)} \sim q(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#} p_0$

Goal: Minimize

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \log p_0 \left(T_\theta^{-1}(x^{(i)}) \right) + \log |\det J_{T_\theta^{-1}}(x^{(i)})|$$

To evaluate this:

1. Need samples $x^{(i)} \sim q(x)$, but **not the value** of $q(x^{(i)})$
2. T_θ must be a diffeomorphism (equivalently, each $T_{k,\theta}$)

Note: you don't need to be able to evaluate T_θ to train

.... only to sample!

Practically:

3. Evaluate $T_\theta^{-1}(x)$ and $\det J_{T^{-1}}(x)$ efficiently

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

base distribution $p_0(x_0)$

model distribution $p_\theta = T_{\theta\#} p_0$

Goal: draw samples $x \sim q(x)$

Example: Bayesian inference

Posterior $q(\theta \mid z) = \frac{q(\theta)q(z \mid \theta)}{Z}$ is known up to the normalising constant Z

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#}p_0$

We can use the **Reverse KL** to derive a loss:

$$\mathcal{J}(\theta) = \mathrm{KL}\left[\, p_\theta(x) \,||\, q(x) \,\right]$$

$$= \int \left[\, \log p_\theta(x) - \log q(x) \,\right] p_\theta(x) dx$$

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#} p_0$

We can use the **Reverse KL** to derive a loss:

$$\mathcal{J}(\theta) = \mathrm{KL}\left[p_\theta(x) \,||\, q(x)\right]$$

$$= \int \left[\log p_\theta(x) - \log q(x)\right] p_\theta(x) \boxed{dx}$$

Substitution:

$x = T(x_0) \quad dx = |\det J_T(x_0)| dx_0$

$$= \int \left[\log p_\theta\left(T(x_0)\right) - \log q\left(T(x_0)\right)\right] p_\theta\left(T(x_0)\right) |\det J_T(x_0)| \boxed{dx_0}$$

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#}p_0$

We can use the **Reverse KL** to derive a loss:

$$\mathcal{J}(\theta) = \mathrm{KL}\left[p_\theta(x) \,||\, q(x)\right]$$

$$= \int \left[\log p_\theta(x) - \log q(x)\right] p_\theta(x) dx$$

Substitution:

$x = T(x_0) \quad dx = |\det J_T(x_0)| dx_0$

$$= \int \left[\log p_\theta\left(T(x_0)\right) - \log q\left(T(x_0)\right)\right] \boxed{p_\theta\left(T(x_0)\right)} |\det J_T(x_0)| dx_0$$

Change-of-Variables:

$p_\theta\left(T(x_0)\right) = p_0(x_0)|\det J_T(x_0)|^{-1}$

$$= \int \left[\log p_0\left(x_0\right) - \log|\det J_T(x_0)| - \log q\left(T(x_0)\right)\right] \boxed{p_0(x_0)|\det J_T(x_0)|^{-1}} |\det J_T(x_0)| dx_0$$

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

base distribution $p_0(x_0)$

model distribution $p_\theta = T_{\theta\#}p_0$

We can use the **Reverse KL** to derive a loss:

$$\mathcal{J}(\theta) = \mathrm{KL}\left[p_\theta(x) \,||\, q(x)\right]$$

$$= \int \left[\log p_\theta(x) - \log q(x)\right] p_\theta(x)dx$$

Substitution:

$x = T(x_0) \quad dx = |\det J_T(x_0)|dx_0$

$$= \int \left[\log p_\theta\left(T(x_0)\right) - \log q\left(T(x_0)\right)\right] \boxed{p_\theta\left(T(x_0)\right)} |\det J_T(x_0)|dx_0$$

Change-of-Variables:

$p_\theta\left(T(x_0)\right) = p_0(x_0)|\det J_T(x_0)|^{-1}$

$$= \int \left[\log p_0\left(x_0\right) - \log|\det J_T(x_0)| - \log q\left(T(x_0)\right)\right] \boxed{p_0(x_0)|\det J_T(x_0)|^{-1}} |\det J_T(x_0)|dx_0$$

$$= \mathbb{E}_{p_0(x_0)}\left[\log p_0(x_0) - \log\left|\det J_T(x_0)\right| - \log q\left(T(x_0)\right)\right]$$

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#} p_0$

We can use the **Reverse KL** to derive a loss:

$$\begin{aligned}\mathcal{J}(\theta) &= \mathrm{KL}\left[p_\theta(x) \,||\, q(x)\right] \\ &= \mathbb{E}_{p_0(x_0)}\left[\log p_0(x_0) - \log\left|\det J_T(x_0)\right| - \log q\left(T(x_0)\right)\right]\end{aligned}$$

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#}p_0$

We can use the **Reverse KL** to derive a loss:

$$\mathcal{J}(\theta) = \text{KL}\left[p_\theta(x) \,||\, q(x)\right]$$

$$= \mathbb{E}_{p_0(x_0)}\left[\boxed{\log p_0(x_0)} - \log\left|\det J_T(x_0)\right| - \boxed{\log q\left(T(x_0)\right)}\right]$$

$$=_{+C} -\mathbb{E}_{p_0(x_0)}\left[\log|\det J_T(x_0)| + \boxed{\log \tilde{q}\left(T(x_0)\right)}\right]$$

Since $\log q(x) = \log \tilde{q}(x) - Z$

and $p_0(x_0)$ does not depend on θ

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

base distribution $p_0(x_0)$

model distribution $p_\theta = T_{\theta\#}p_0$

We can use the **Reverse KL** to derive a loss:

$$\mathcal{J}(\theta) = \mathrm{KL}\left[p_\theta(x) \,||\, q(x)\right]$$

$$= \mathbb{E}_{p_0(x_0)}\left[\log p_0(x_0) - \log\left|\det J_T(x_0)\right| - \log q\left(T(x_0)\right)\right]$$

$$=_{+C} -\mathbb{E}_{p_0(x_0)}\left[\log|\det J_T(x_0)| + \log\tilde{q}\left(T(x_0)\right)\right]$$

Since $\log q(x) = \log\tilde{q}(x) - Z$

and $p_0(x_0)$ does not depend on θ

$$\approx -\frac{1}{n}\sum_{i=1}^{n}\left[\log|\det J_T(x_0^{(i)})| + \log\tilde{q}\left(T(x_0^{(i)})\right)\right]$$

Estimate from samples $x_0^{(i)} \sim p_0$

Reverse KL Training

Have:

A density $q(x) = \frac{1}{Z}\tilde{q}(x)$

base distribution $p_0(x_0)$

diffeomorphism $T_\theta = T_{K,\theta} \circ \ldots \circ T_{2,\theta} \circ T_{1,\theta}$

model distribution $p_\theta = T_{\theta\#}p_0$

Goal: Minimize

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[\log|\det J_T(x_0^{(i)})| + \log \tilde{q}\left(T(x_0^{(i)})\right)\right]$$

To evaluate this:

1. Need **the value** $\tilde{q}(x)$, but **not samples**
2. T_θ must be a diffeomorphism (equivalently, each $T_{k,\theta}$)

Note: you don't need to be able to evaluate T_θ^{-1} to train

… but you do need it if you want to evaluate the model density.

Practically:

3. Evaluate $T_\theta(x)$ and $\det J_T(x)$ efficiently

Summary

Objective	Sample $q(x)$?
Forward KL $\mathscr{L}(\theta) = \mathrm{KL}\left[\, q(x) \,||\, p_\theta(x) \,\right]$	✅
Reverse KL $\mathscr{J}(\theta) = \mathrm{KL}\left[\, p_\theta(x) \,||\, q(x) \,\right]$	❌

Summary

Objective	Sample $q(x)$?	Evaluate $\tilde{q}(x)$?
Forward KL $\mathscr{L}(\theta) = \mathrm{KL}\left[\, q(x) \,||\, p_\theta(x) \,\right]$	✅	❌
Reverse KL $\mathscr{J}(\theta) = \mathrm{KL}\left[\, p_\theta(x) \,||\, q(x) \,\right]$	❌	✅

Summary

Objective	Sample $q(x)$?	Evaluate $\tilde{q}(x)$?	Model Param.
Forward KL $\mathscr{L}(\theta) = \mathrm{KL}\left[\, q(x) \,||\, p_\theta(x) \,\right]$	✅	❌	T_θ^{-1}
Reverse KL $\mathscr{J}(\theta) = \mathrm{KL}\left[\, p_\theta(x) \,||\, q(x) \,\right]$	❌	✅	T_θ

Summary

Objective	Sample $q(x)$?	Evaluate $\tilde{q}(x)$?	Model Param.	Use Case (Without inverting model)
Forward KL $\mathscr{L}(\theta) = \mathrm{KL}\left[q(x) \,\|\, p_\theta(x)\right]$	✅	❌	T_θ^{-1}	Density Estimator
Reverse KL $\mathscr{J}(\theta) = \mathrm{KL}\left[p_\theta(x) \,\|\, q(x)\right]$	❌	✅	T_θ	Sampler

Questions?

Practicalities

We will focus on implementing T_θ — everything holds for the other case.

Goal: Minimize $\quad J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[\log|\det J_T(x_0^{(i)})| + \log \tilde{q}\left(T(x_0^{(i)})\right)\right]$

Requirements on $T_{k,\theta}$:

1. $T_{k,\theta}$ is a diffeomorphism
2. Efficiently Trainable: $\det J_{T_{k,\theta}}$ can be evaluated efficiently
3. (Optional) Easily Invertible: $T_{k,\theta}^{-1}$ can be evaluated efficiently
   - *The inverses must exist, but maybe aren't easy to compute.*

Tension between expressivity and tractability

Hot topic ~2015-2020: inventing new flow architectures

NICE, RealNVP, Glow, Masked Autoregressive Flows, Inverse Autoregressive Flows, Neural Spline Flows, …

Practicalities

Desired:

1. Efficiently Trainable: $\det J_{T_{k,\theta}}$ can be evaluated efficiently
2. (Optional) Invertible: $T_{k,\theta}^{-1}$ can be evaluated efficiently

What do we mean by “tractable” Jacobians?

For any differentiable $f : \mathbb{R}^d \to \mathbb{R}^d$, you can compute $J_f(x) \in \mathbb{R}^{d \times d}$ via automatic differentiation

- Recall: autodiff computes $v^T J_f(x)$ (VJP) or $J_f(x)v$ (JVP)
- Requires d autodiff calls (one per row/column; take v to be one-hot)
- Expensive if f is a neural network block or d is large

Then, explicitly compute $\det J_f(x)$

- This is $O(d^3)$ — expensive if d is even moderately large

We need to design $T_{k,\theta}$ such that $\det J_{T_{k,\theta}}(x)$ can be computed quickly.

Autoregressive Flows

We need to design $T_{k,\theta}$ such that $\det J_{T_{k,\theta}}(x)$ can be computed quickly.

Special case:
J_T is **triangular**

$$J_T(x_0) = \begin{bmatrix} \frac{\partial T^{(1)}}{\partial x_0^{(1)}} & \cdots & \frac{\partial T^{(1)}}{\partial x_0^{(d)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial T^{(d)}}{\partial x_0^{(1)}} & \cdots & \frac{\partial T^{(d)}}{\partial x_0^{(d)}} \end{bmatrix}$$

$$J_T = \begin{bmatrix} J_{11} & 0 & 0 & \dots & 0 \\ J_{21} & J_{22} & 0 & \dots & 0 \\ J_{31} & J_{32} & J_{33} & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ J_{d1} & J_{d2} & J_{d3} & \dots & J_{dd} \end{bmatrix}$$

$$\implies \det J_T = \prod_{i=1}^{d} J_{T,ii}$$

Autoregressive Flows

We need to design $T_{k,\theta}$ such that $\det J_{T_{k,\theta}}(x)$ can be computed quickly.

If $J_{T_{k,\theta}}$ is **triangular**, we can compute $\det J_{T_{k,\theta}}(x)$ in $O(d)$ time

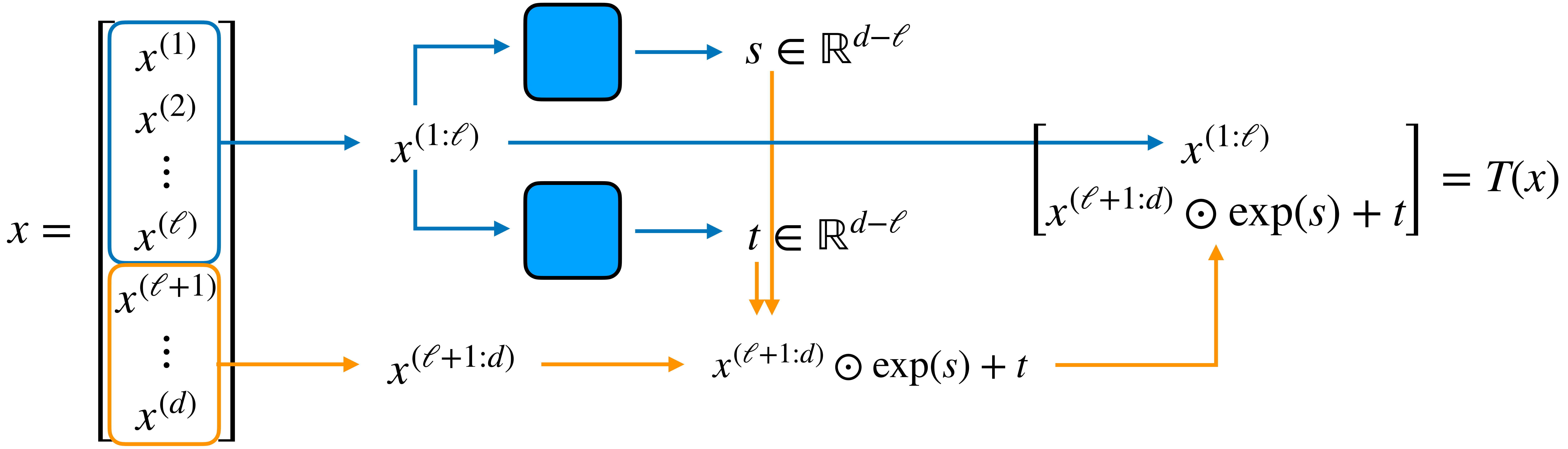
$$J_T(x_0) = \begin{bmatrix} \frac{\partial T^{(1)}}{\partial x_0^{(1)}} & \cdots & \frac{\partial T^{(1)}}{\partial x_0^{(d)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial T^{(d)}}{\partial x_0^{(1)}} & \cdots & \frac{\partial T^{(d)}}{\partial x_0^{(d)}} \end{bmatrix} \quad J_T = \begin{bmatrix} J_{11} & 0 & 0 & \dots & 0 \\ J_{21} & J_{22} & 0 & \dots & 0 \\ J_{31} & J_{32} & J_{33} & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ J_{d1} & J_{d2} & J_{d3} & \dots & J_{dd} \end{bmatrix}$$

The ith output coordinate only depends on the first i input coordinates

$$T(x) = \begin{bmatrix} f_1(x^{(1)}) \\ f_2(x^{(1:2)}) \\ \vdots \\ f_i(x^{(1:i)}) \\ \vdots \\ f_d(x^{(1:d)}) \end{bmatrix}$$

$\Longrightarrow J_T$ is triangular.

Example: RealNVP [Dinh 2016]



This mapping is triangular.

Is it easily invertible? **Yes**!

$$T^{-1}(x) = \begin{bmatrix} x^{(1:\ell)} \\ (x^{(\ell+1:d)} - t) \odot \exp(-s) \end{bmatrix}$$

Example: RealNVP [Dinh 2016]

Samples on CelebA at 64x64 resolution

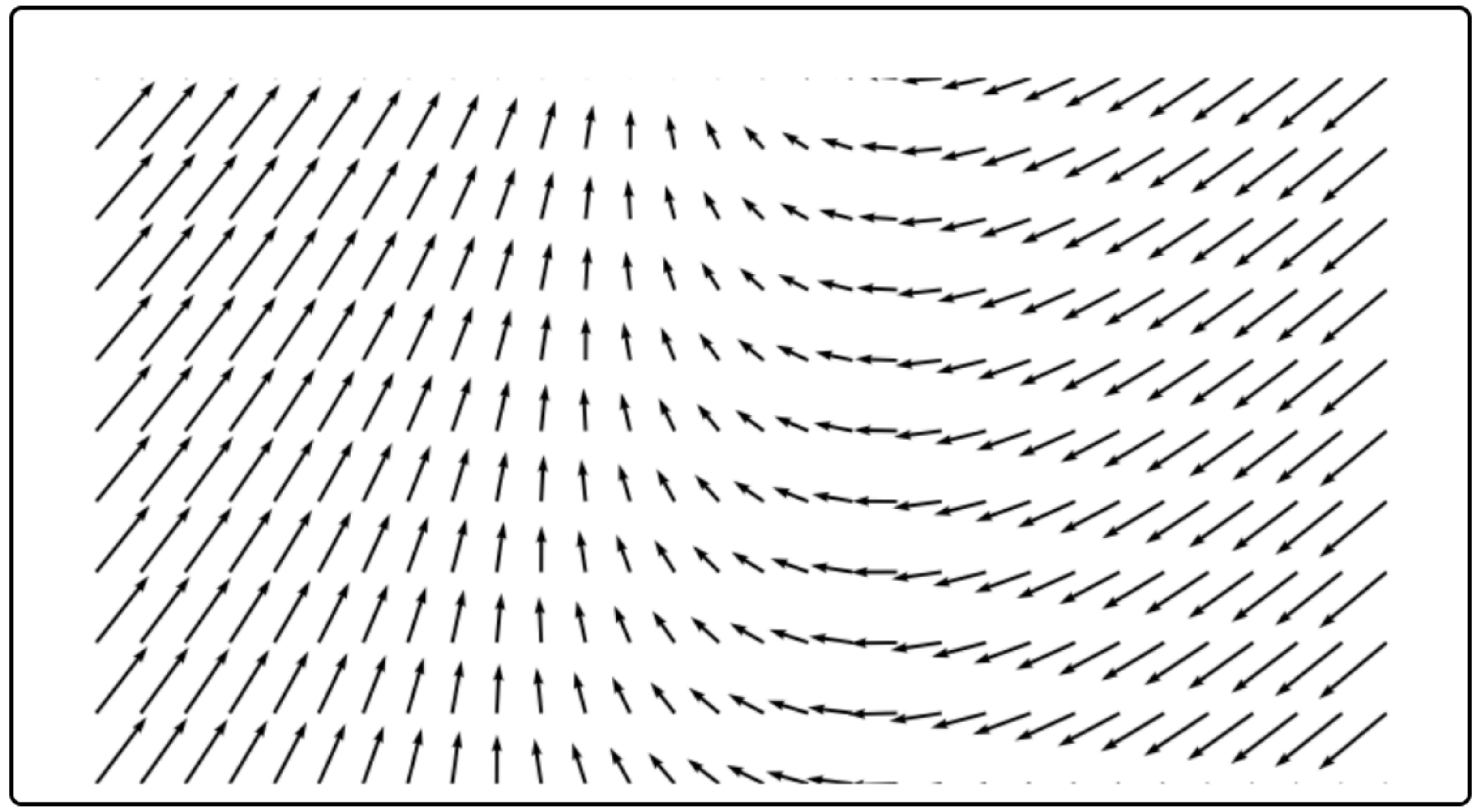
Good enough for an ICLR paper in 2017 :)

Questions?

Chapter 2
Continuous-Time Flows

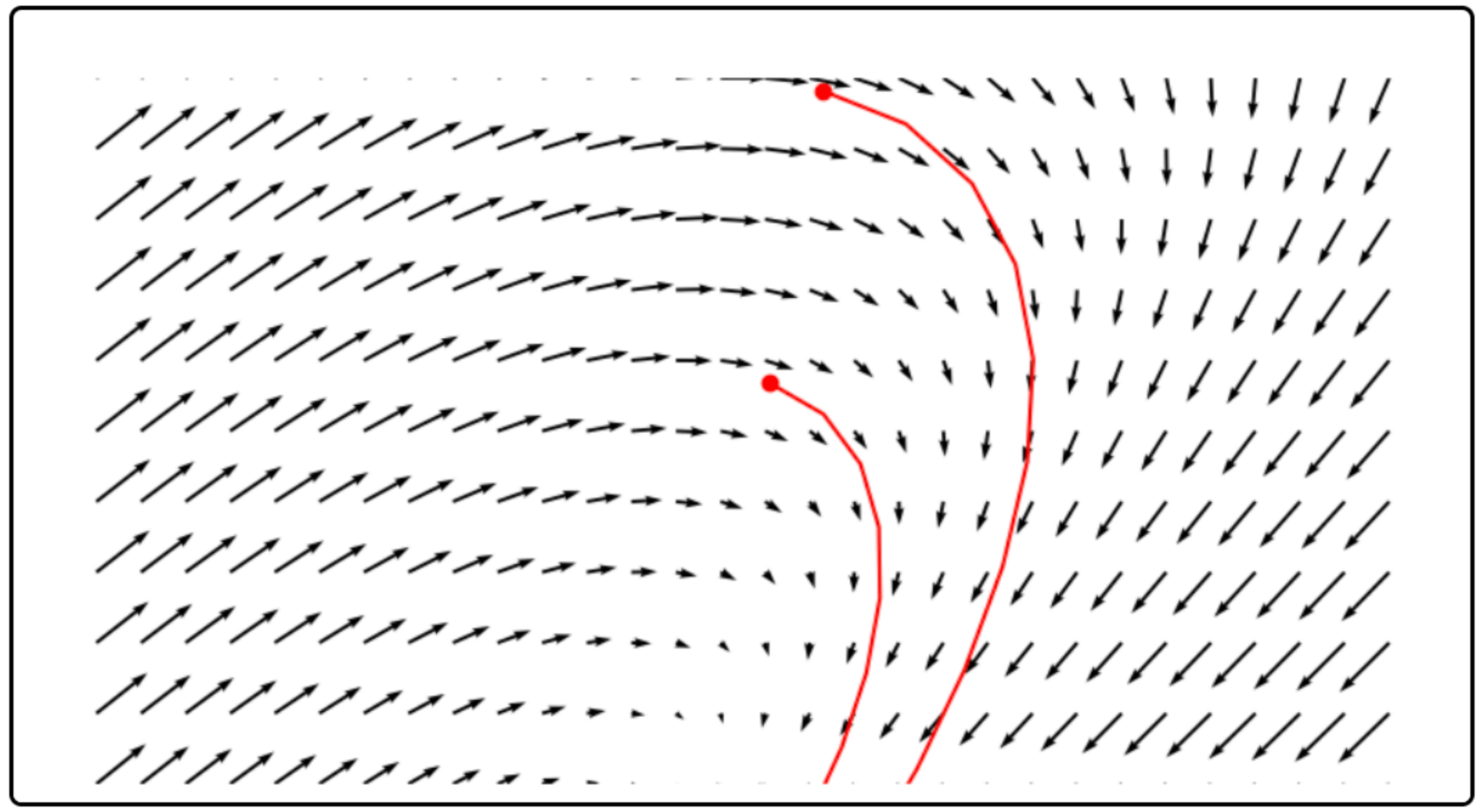
Crash Course on ODEs

Vector fields $v(t, x) : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ tell you **velocity** of a particle located at x at time t



An ODE $\quad dx_t = v(t, x_t)dt \quad\quad x_0 = x$

tells us how a particle moves starting from x_0



Crash Course on ODEs

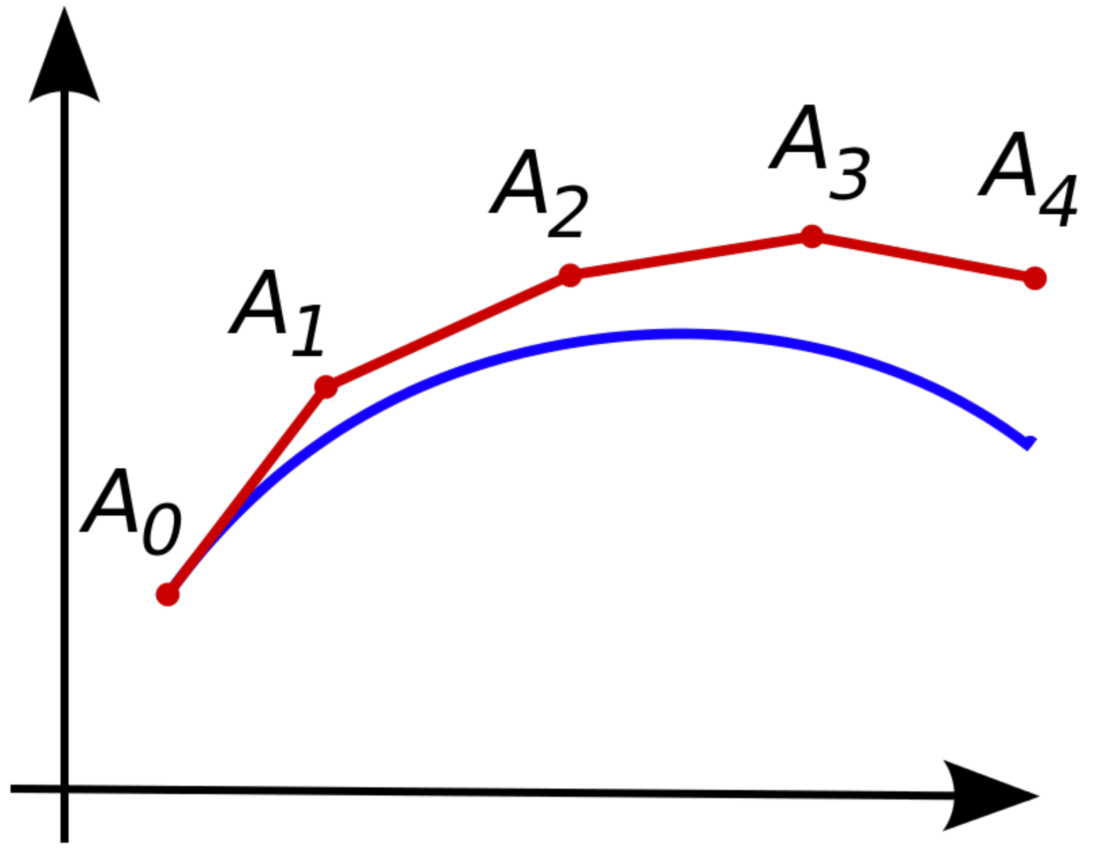
Leonhard Euler
1707-83

Euler's Method: simplest scheme for numerically solving an ODE $dx_t = v(t, x_t)dt \quad x_0 = x$

Discretize time: $$0 = t_0 < t_1 < \cdots < t_N = T$$

$$t_{n+1} = t_n + h$$

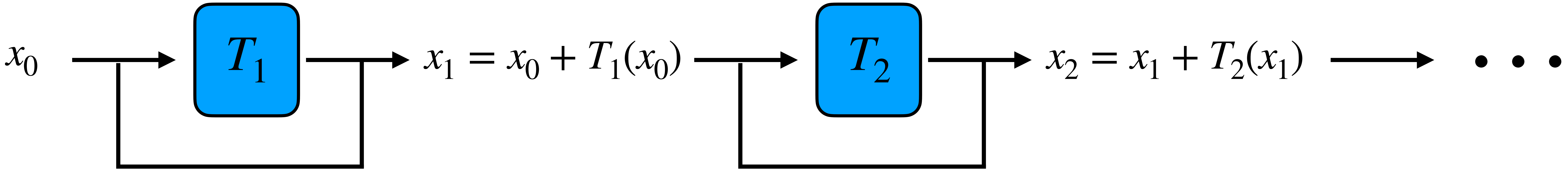
"Step size"



Move particle: $$x_{t+h} = x_t + hv(t, x_t)$$

From Discrete to Continuous Dynamics

Normalising flows **iteratively transform** a source sample:



Instead of thinking about layers, think of ***time***.

$$x_{k+1} = x_k + T(x_k, k)$$

Each layer tells us where x_t moves after Δt seconds:

$$x_{t+\Delta t} = x_t + \Delta t \cdot T(x_t, t)$$

$$\frac{x_{t+\Delta t} - x_t}{\Delta t} = T(x_t, t)$$

Making $\Delta t \to dt$ small, we get an **ODE**:

$$\frac{d}{dt} x_t = T(x_t, t)$$

Continuous-Time Flows

Critical idea:

Instead of directly implementing a transformation $T_\theta(x)$, implement a **velocity** $v_\theta(x, t)$

This induces a transformation: $$T_\theta(x_0) = x_1 = x_0 + \int_0^1 v_\theta(x_t, t)dt$$

Why would we do this?

1. Flexibility

   Very mild conditions on $v \implies T_\theta$ is a diffeomorphism!

   (existence and uniqueness of ODE solution)

   $$T_\theta^{-1}(x_1) = x_1 - \int_0^1 v_\theta(x_t, t)dt$$

   No need to worry about tractable Jacobians, inverting your network, etc.

2. Expressivity

   Like having an infinitely deep network

3. Leads to Diffusions and Flow Matching

   This reformulation is a major conceptual leap

Training CNFs

Have **data samples** $x^{(i)} \sim q(x)$ for $i = 1,2,\ldots,n$

Choose a **base distribution** $p_0(x_0)$

Implement **velocity** $v_\theta(t, x)$

Induces model distribution $p_1 = (T_\theta)_\# p_0$ $\quad T_\theta(x_0) = x_1 = x_0 + \int_0^1 v_\theta(x_t, t)dt$

i.e., draw initial particles $x_0 \sim p_0$ and solve $dx_t = v_\theta(x_t, t)dt$ on $t \in [0,1]$

To evaluate a KL divergence, need $\log p_1(x)$

$$\mathscr{L}(\theta) = \mathrm{KL}\left[q(x) \,||\, p_1(x)\right] \quad =_{+C} -\mathbb{E}_{q(x)}\left[\log p_1(x)\right]$$

We want a continuous-time change of variables formula

Continuity PDE

$$\text{div}(v) = \sum_{i=1}^{d} \frac{\partial}{\partial x^{(i)}} v(t, x^{(i)})$$
$$= \nabla \cdot v(t, x_t)$$

Proposition [Continuity Equation].

If the particles X_t follow the dynamics $\quad dX_t = v(X_t, t)dt \qquad X_0 \sim p_0$

Then the density $p_t(x)$ at any time $t > 0$ solves the continuity PDE

$$\partial_t p_t(x) + \text{div}(p_t(x)v(x, t)) = 0$$

Intuition:

Continuity PDE tells us how the density at every fixed location x changes over time

Two equivalent descriptions of the same process:

("Local") Evolution of individual particles		("Global") Evolution of distribution of particles
$\frac{d}{dt} x_t = v(x_t, t)$	$\longleftrightarrow$	$\partial_t p_t = -\text{div}(v_t p_t)$

Change-of-Variables 2.0

Corollary.

$$\frac{d}{dt}\log p_t(x_t) = -\,\mathrm{div}(v_t(x_t)) = -\,\mathrm{tr}\left(J_v(t, x_t)\right)$$

Intuition:

This tells us how the density of a **particular particle** changes over time.

Proof:

Usual chain rule: $$\partial_t \log p_t(x) = \frac{1}{p_t(x)}\partial_t p_t(x)$$

Continuity PDE: $$= -\frac{1}{p_t(x)}\mathrm{div}(v_t(x)p_t(x))$$

Chain rule for div: $$= -\frac{1}{p_t(x)}\left(p_t(x)\mathrm{div}(v_t(x)) + \langle v_t(x), \nabla_x p_t(x)\rangle\right)$$

$$= -\,\mathrm{div}(v_t(x)) - \langle v_t(x), \nabla_x \log p_t(x)\rangle$$

Change-of-Variables 2.0

Corollary.

$$\frac{d}{dt}\log p_t(x_t) = -\,\mathrm{div}(v_t(x_t)) = -\,\mathrm{tr}\left(J_v(t, x_t)\right)$$

Intuition:

This tells us how the density of a **particular particle** changes over time.

Proof (continued): $\partial_t \log p_t(x) = -\,\mathrm{div}(v_t(x)) - \langle v_t(x), \nabla_x \log p_t(x)\rangle$

Compute the total derivative via the chain rule:

$$\frac{d}{dt}\log p_t(x_t) = \partial_t \log p_t(x_t) + \left\langle \frac{d}{dt}x_t, \nabla_x \log p_t(x_t)\right\rangle$$

$$= \partial_t \log p_t(x_t) + \left\langle v_t(x_t)\,\nabla_x \log p_t(x_t)\right\rangle$$

$$= -\,\mathrm{div}(v_t(x_t)) \qquad \blacksquare$$

Change-of-Variables 2.0

Corollary.

$$\frac{d}{dt}\log p_t(x_t) = -\,\mathrm{div}(v_t(x_t)) = -\,\mathrm{tr}\left(J_v(t, x_t)\right)$$

Takeaway: $$\log p_1(X_1) = \log p_0(X_0) - \int_0^1 \mathrm{div}\left(v(t, X_t)\right) dt$$

Given $x_0 \sim p_0$, can get x_1 and its density $\log p_1(x_1)$ simultaneously:

$$\begin{bmatrix} X_1 \\ \log p_1(X_1) \end{bmatrix} = \begin{bmatrix} X_0 \\ \log p_0(X_0) \end{bmatrix} + \int_0^1 \begin{bmatrix} v_\theta(t, X_t) \\ -\mathrm{div}\left(v_\theta(t, X_t)\right) \end{bmatrix} dt$$

- The same idea works “in reverse”.

Training CNFs

Algorithm:

1. Draw samples $x_1^{(i)} \sim q(x)$ for $i = 1,2,\ldots,n$

2. Compute $\log p_1(x)$ via $\log p_1(X_1) = \log p_0(X_0) - \int_0^1 \text{div}\left(v_\theta(t, X_t)\right) dt$

   - *In practice, plug into your favourite numerical ODE solver*

3. Do gradient descent to minimize $\mathscr{L}(\theta) = \text{KL}\left[\, q(x) \,||\, p_1(x) \right] =_{+C} - \mathbb{E}_{q(x)}\left[\log p_{1(x)}\right]$

Training CNFs

Remarks:

1. The neural network $v_\theta(t, x)$ no longer is required to be a diffeomorphism!

2. Training requires numerically solving an ODE ("simulating")

- Requires $\mathrm{tr}\left(J_{v_\theta}(t, x)\right)$ at every ODE step

  … cheaper than a determinant, but still needs $O(d)$ backward passes

- A common trick: Hutchinson's Trace Estimator

$$\mathrm{tr}\left(J_{v_\theta}(t, x)\right) \approx w^T J_{v_\theta}(t, x) w \qquad w \sim N(0, I)$$

  Intuition: project your Jacobian onto a random direction w

  Only needs one backwards pass

  Example: FFJORD [Grathwohl 2018]

3. Training requires backprop through ODE solver

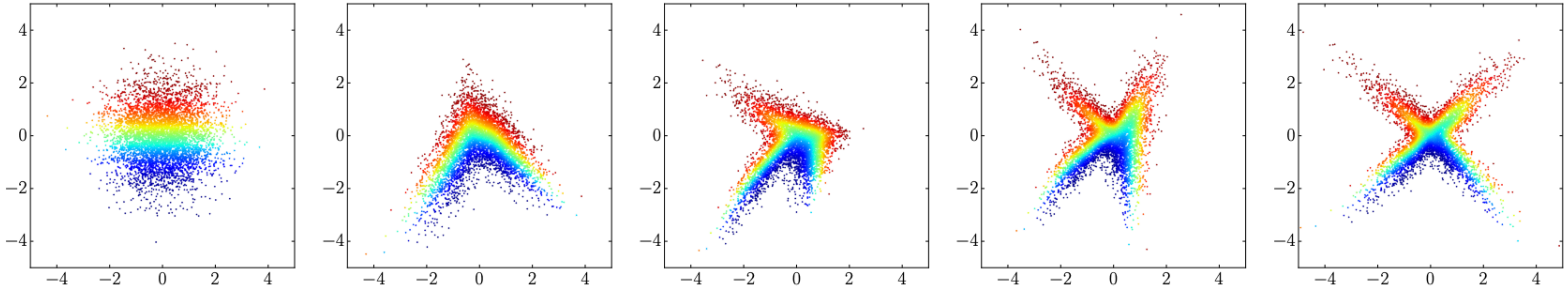
- Huge memory costs if done naively
- "Adjoint method" solves this

Questions?

Summary

Discrete-Time Flows iteratively transform samples from a simple distribution

[Papamakarios 2021]



- Useful for both **density estimation** and **sampling**
- Transformations must be invertible and T, T^{-1} differentiable (***diffeomorphism***)

- Trained by **divergence minimisation** (~maximum likelihood)
- Transformed density can be computed via **change-of-variables**
    - Require **specialised architectures for** tractable Jacobians

Summary

Discrete-Time Flows transform samples through an ODE

("Local")
Evolution of individual particles

$$\frac{d}{dt}x_t = v(x_t, t)$$

$\longleftrightarrow$

("Global")
Evolution of distribution of particles

$$\partial_t p_t = -\,\mathrm{div}(v_t p_t)$$

- Neural network parameterises vector field
    - No need for **specialised architectures**
- Trained by **divergence minimisation** (~maximum likelihood)
    - Continuous-time change of variables derived via continuity PDE