CVPR 2021 Tutorial:
Normalizing Flows and Invertible Neural Networks in Computer Vision

Marcus A. Brubaker and Ullrich Köthe

CVPR 2021 Tutorial:
Normalizing Flows and Invertible Neural Networks in Computer Vision

Introduction to Normalizing Flows

Marcus A. Brubaker

VECTOR INSTITUTE

LASSONDE SCHOOL OF ENGINEERING

YORK UNIVERSITÉ UNIVERSITY

Generative Models

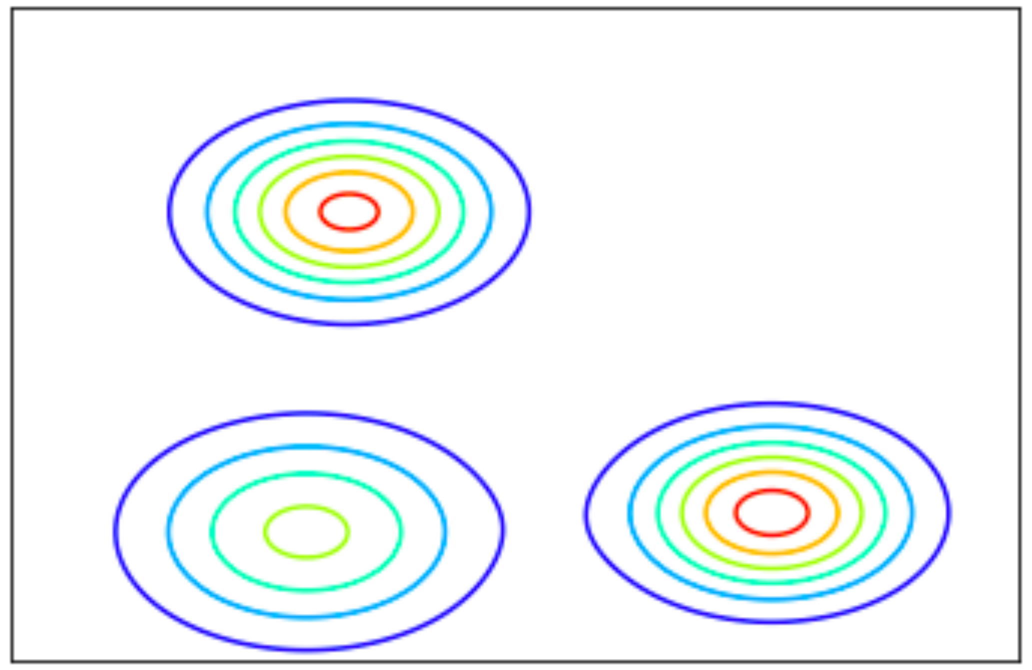
A **generative model** is a probability distribution over a random variable $\mathbf{X}$ which we attempt to learn from a set of observed data $\{\mathbf{x}_i\}_{i=1}^N$ with some probability density $p_{\mathbf{X}}(\mathbf{x})$ parameterized by θ

Given a GM we may want to generate samples, evaluate new data points, etc

Different distributions and different learning objectives and approaches lead to different GMs, e.g., GANs, VAEs, NFs etc

GMs: Mixture Models

(Gaussian) Mixture Model



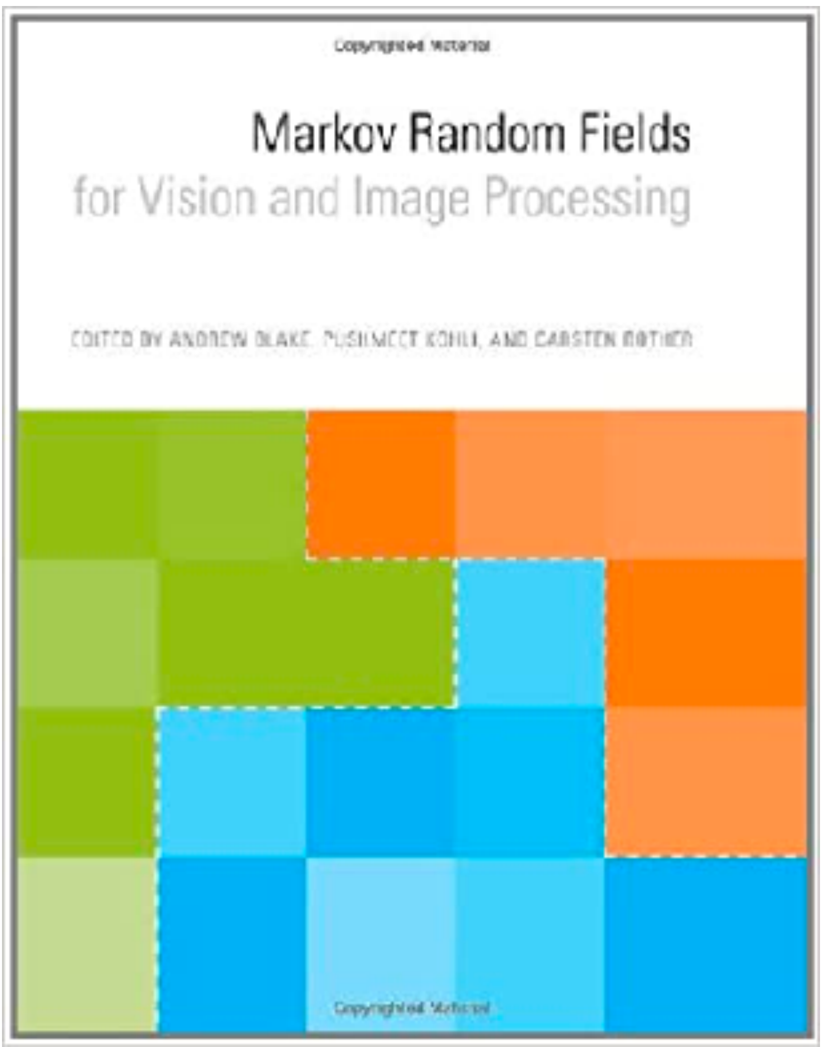
- a classical example of a GM which has been studied extensively
- trained either via ML or a variational bound on likelihood
- sampling and evaluating $p_{\mathbf{X}}(\mathbf{x})$ is straightforward
- performance scales poorly with dimensionality and added expressiveness

[Richardson and Weiss, NeurIPS 2018]

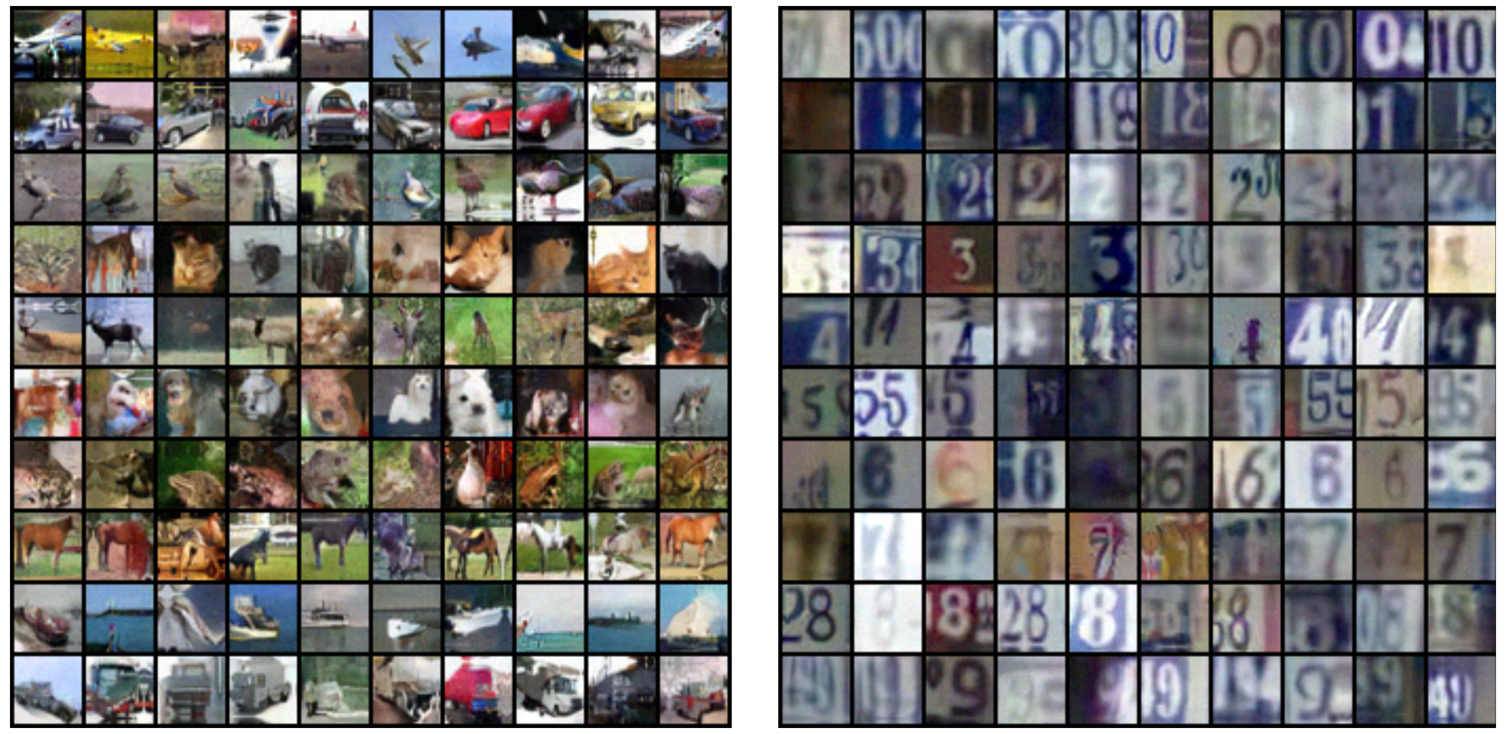
GMs: Energy-based Models

Energy-Based Models

- $p_{\mathbf{X}}(\mathbf{x})$ is unnormalized
- familiar in classical computer vision
- some recent successes
- training and sampling from $p_{\mathbf{X}}(\mathbf{x})$ is complex, typically requiring MCMC



[Blake, Kohli and Rother eds, 2011]



[Grathwohl et al, ICLR 2020]

[Song and Kingma, 2021]

GMs: Generative Adversarial Networks

Generative Adversarial Networks

- impressive results
- trained through an adversarial process which (roughly) minimizes a divergence or integral probability metric
- sampling from $p_{\mathbf{X}}(\mathbf{x})$ is straightforward
- evaluating $p_{\mathbf{X}}(\mathbf{x})$ is generally not possible

[Karras et al, StyleGAN2 2019]

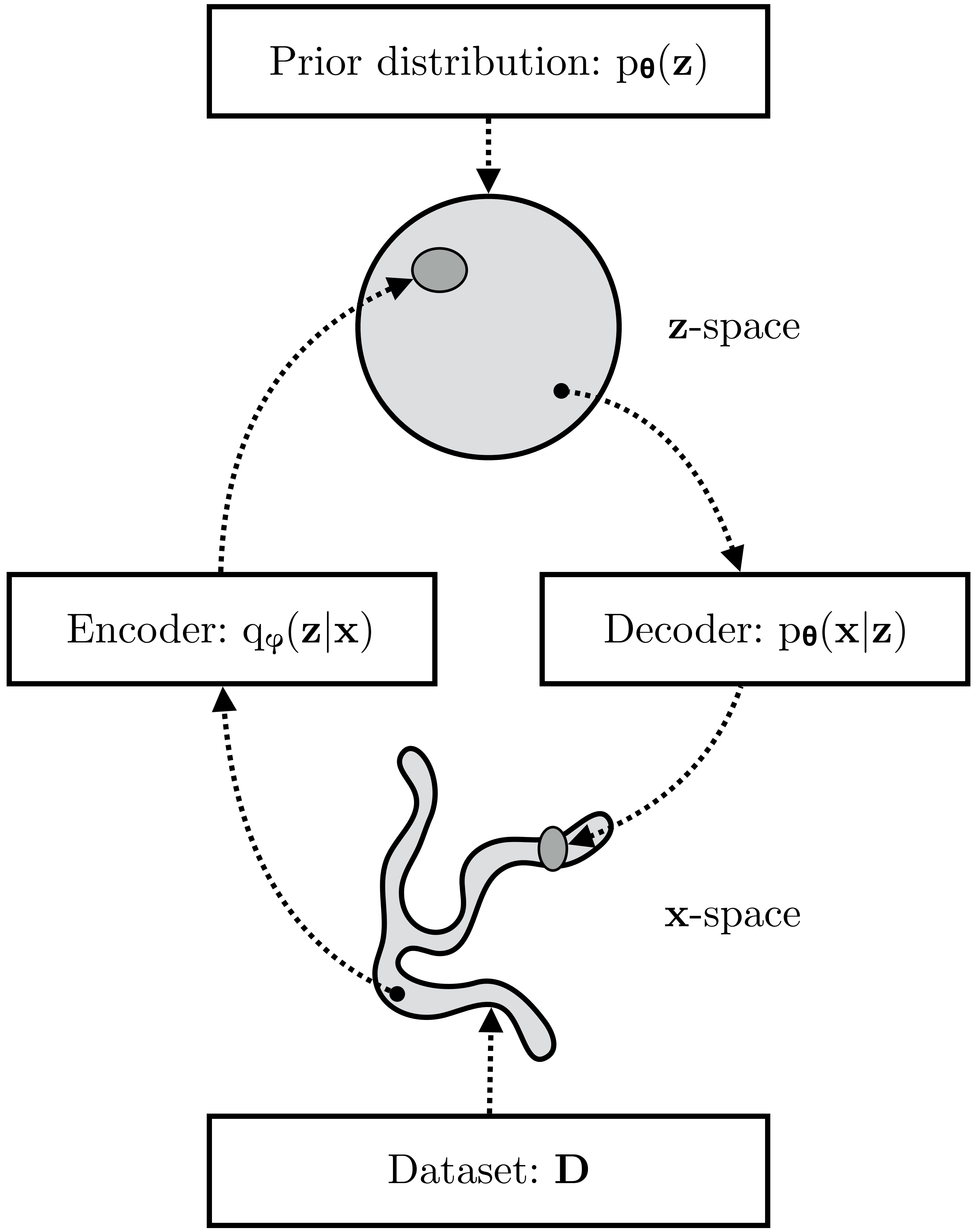
Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie,[*] Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,[†] Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

GMs: Variational Autoencoders

Variational Auto-encoders

- probabilistic latent variables models
- successful in learning useful low-dimensional representations
- trained with bound on marginal likelihood
- sampling from $p_{\mathbf{X}}(\mathbf{x})$ is straightforward
- approximate evaluation of $p_{\mathbf{X}}(\mathbf{x})$ is possible



[Kingma and Welling, 2019]

What are Normalizing Flows?

Normalizing Flows are a GM built on invertible transformations

They are generally:

- Efficient to sample from $p_{\mathbf{X}}(\mathbf{x})$
- Efficient to evaluate $p_{\mathbf{X}}(\mathbf{x})$
- Highly expressive
- Useful latent representation
- Straightforward to train

History of Normalizing Flows

A family of non-parametric density estimation algorithms

E. G. TABAK
Courant Institute of Mathematical Sciences
AND
CRISTINA V. TURNER
FaMAF, Universidad Nacional de Córdoba

[Tabak and Turner, CPAM 2013]

NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

[Dinh et al, ICLR 2015]

2010 — 2013 — 2014 — 2015

High-Dimensional Probability Estimation with Deep Density Models

Oren Rippel*
Massachusetts Institute of Technology,
Harvard University
rippel@math.mit.edu

Ryan Prescott Adams†
Harvard University
rpa@seas.harvard.edu

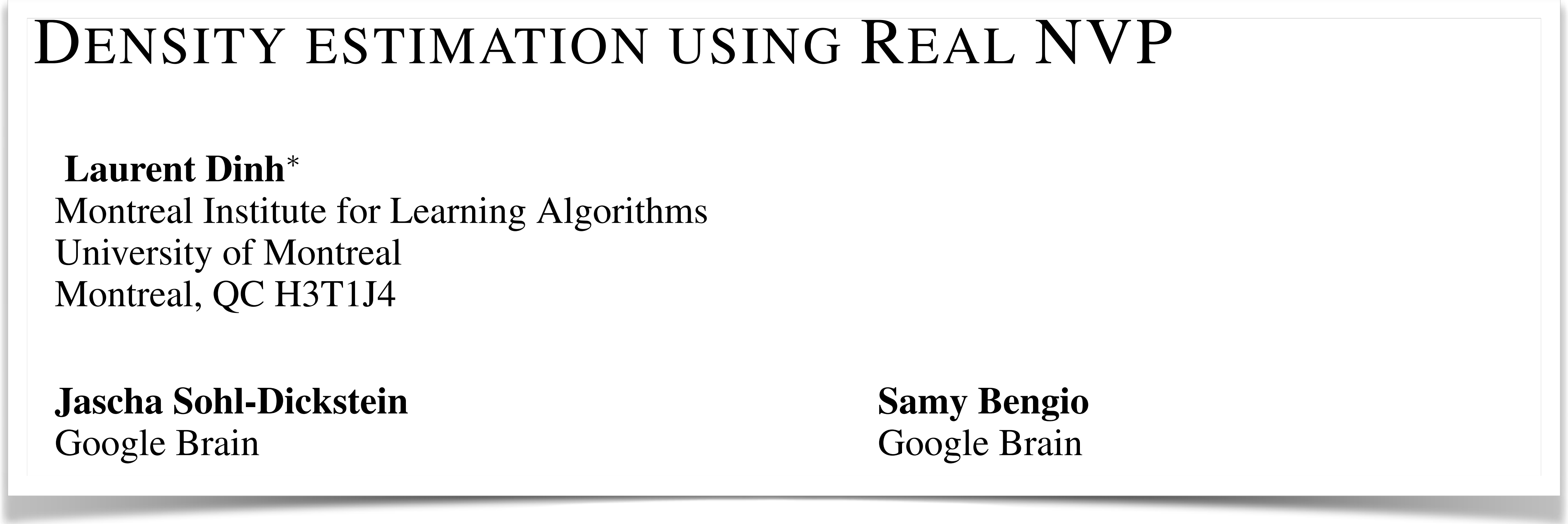
[Rippel and Adams, arXiv 2013]

Variational Inference with Normalizing Flows

Danilo Jimenez Rezende DANILOR@GOOGLE.COM
Shakir Mohamed SHAKIR@GOOGLE.COM
Google DeepMind, London

[Rezende and Mohamed, ICML 2015]

History of Normalizing Flows



DENSITY ESTIMATION USING REAL NVP

Laurent Dinh*
Montreal Institute for Learning Algorithms
University of Montreal
Montreal, QC H3T1J4

Jascha Sohl-Dickstein
Google Brain

Samy Bengio
Google Brain

[Dinh et al, ICLR 2017]

2016

History of Normalizing Flows

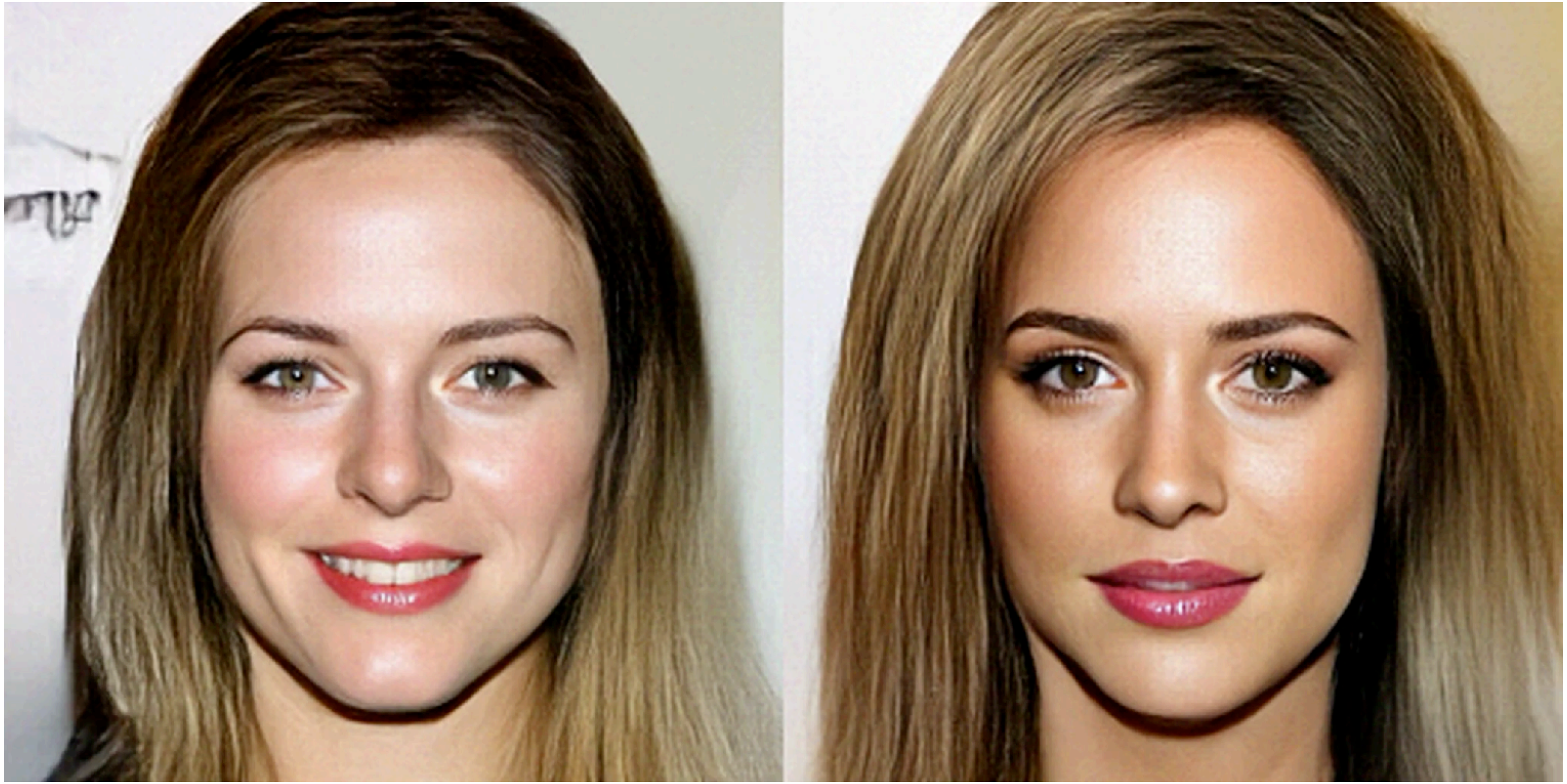
Glow: Generative Flow with Invertible 1×1 Convolutions

Diederik P. Kingma[*], Prafulla Dhariwal[*]
OpenAI, San Francisco

[Kingma and Dhariwal, NeurIPS 2018]

2018

History of Normalizing Flows



[Kingma and Dhariwal, NeurIPS 2018]

History of Normalizing Flows

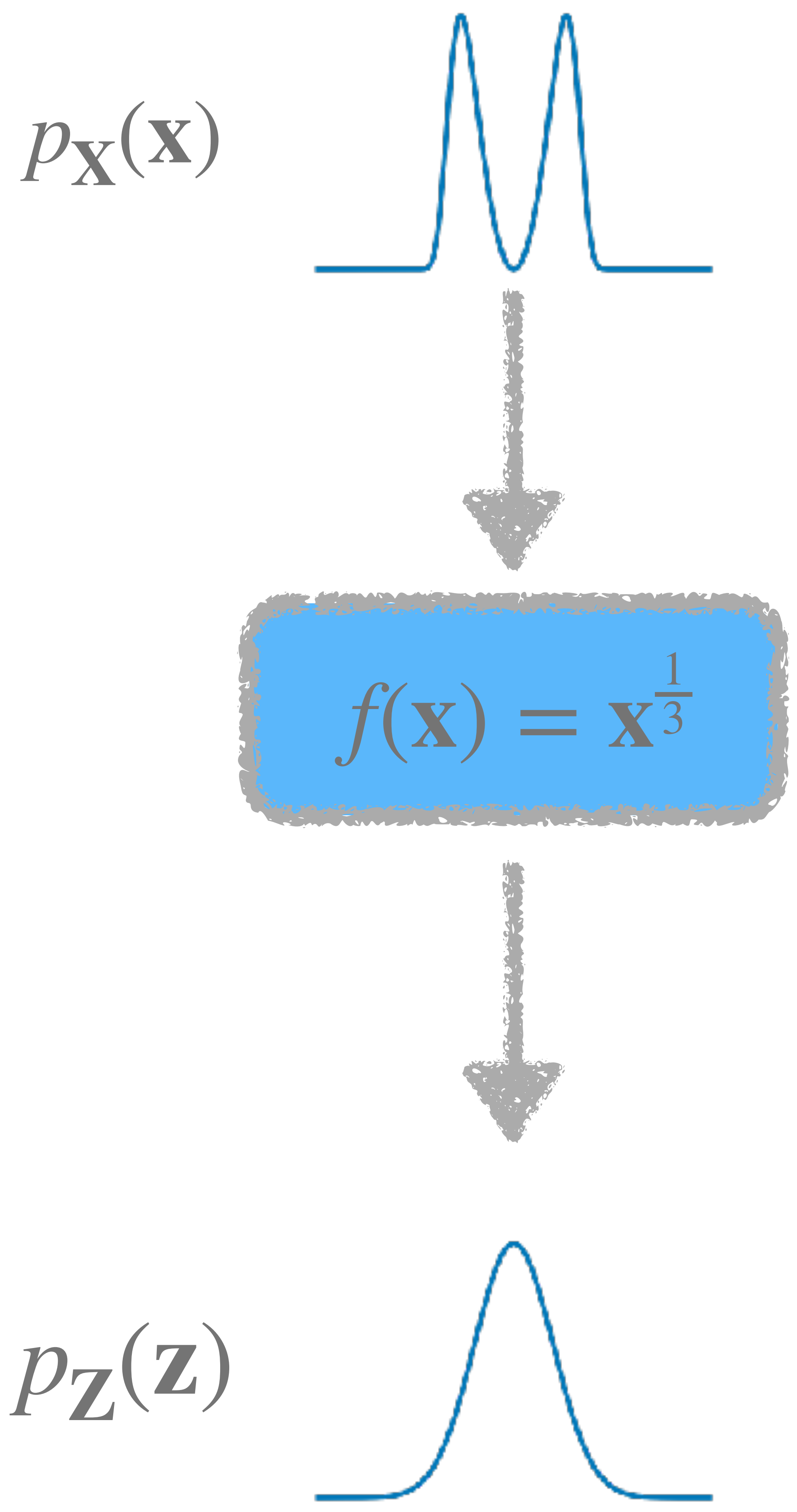
[Kingma and Dhariwal, NeurIPS 2018]

Normalizing Flows

Change of variables

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(\underbrace{f(\mathbf{x})}_{\text{Invertible Transform}})\overbrace{\left|\det Df(\mathbf{x})\right|}^{\text{Volume Correction}}$$

where $\mathbf{Z} = f(\mathbf{X})$ is an invertible, differentiable function
and $Df(\mathbf{x})$ is the Jacobian of $f(\mathbf{x})$



Normalizing Flows

Can represent a given $p_{\mathbf{X}}(\mathbf{x})$ in terms of $p_{\mathbf{Z}}(\mathbf{z})$ and $f(\mathbf{x})$

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(f(\mathbf{x}))\left|\det Df(\mathbf{x})\right|$$

Normalizing Flows

Learn $f(\mathbf{x})$ to transform data distribution $p_{\mathbf{X}}(\mathbf{x})$ into $p_{\mathbf{Z}}(\mathbf{z})$

Two pieces

- **Base Measure:** $p_{\mathbf{Z}}(\mathbf{z})$ - Typically selected as $\mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$
- **Flow:** $f(\mathbf{x})$ - Must be invertible and differentiable

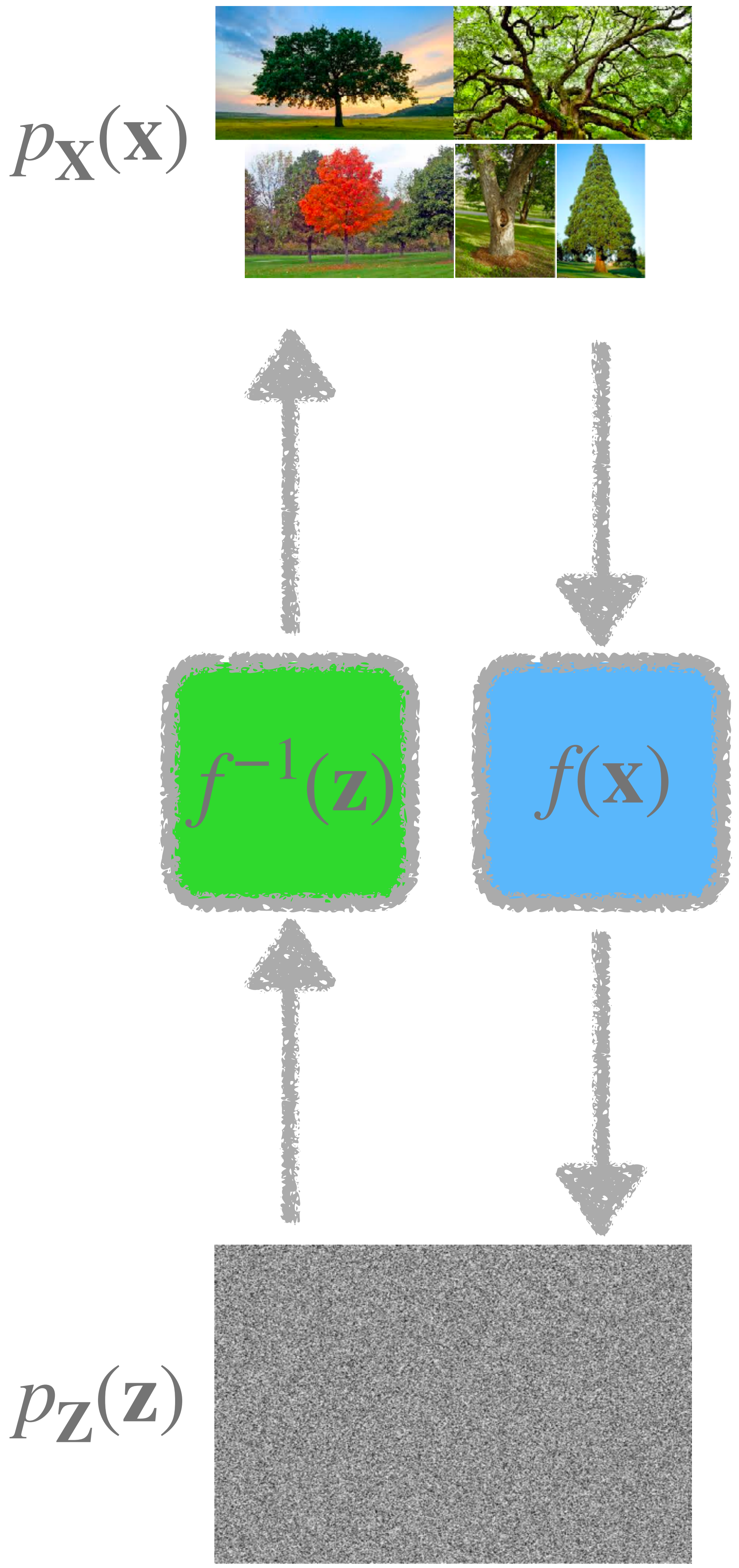
Normalizing Flows

Density evaluation:

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(f(\mathbf{x}))\left|\det Df(\mathbf{x})\right|$$

Sampling:

- Sample $\mathbf{z} \sim p_{\mathbf{Z}}(\,\cdot\,)$
- Compute $\mathbf{x} = f^{-1}(\mathbf{z})$



Normalizing Flows

Training can be done with maximum (log-)likelihood

$$\max_{\theta} \sum_{i=1}^{N} \log p_{\mathbf{Z}}(f(\mathbf{x}_i \,|\, \theta)) + \log |\det Df(\mathbf{x}_i \,|\, \theta)|$$

where θ are the parameters of the flow $f(\mathbf{x} \,|\, \theta)$

Flows

A **flow** is a parametric function $f(\mathbf{x})$ which:

- is invertible
- is differentiable
- has an efficiently computable inverse and Jacobian determinant $|\det Df(\mathbf{x})|$

Also sometimes called a **flow layer**, **bijection**, etc.

Designing and understanding flows is the core technical challenge with NFs

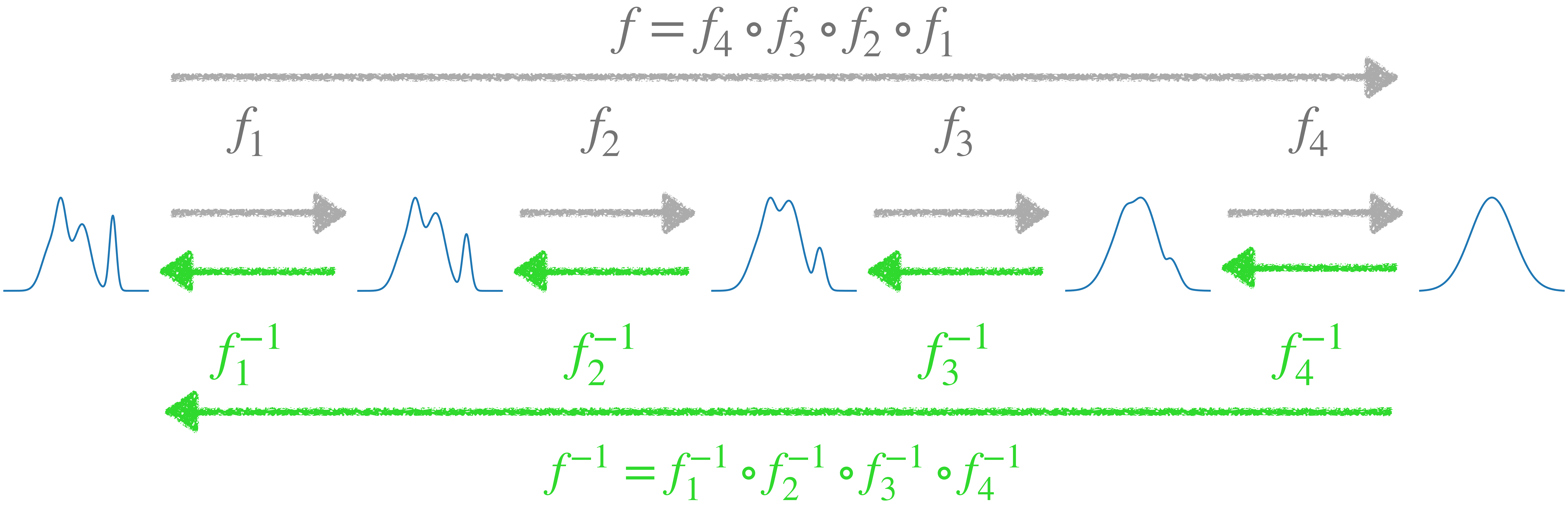
Composition of Flows

Invertible, differentiable functions are closed under composition

$$f = f_K \circ f_{K-1} \circ \cdots \circ f_2 \circ f_1$$

Build up a complex flow from composition of simpler flows

Composition of Flows



Composition of Flows

Determinant:

$$\det Df = \det \prod_{k=1}^{K} Df_k = \prod_{k=1}^{K} \det Df_k$$

Likelihood:

$$\max_{\theta} \sum_{i=1}^{N} \log p_{\mathbf{Z}}(f(\mathbf{x}_i \,|\, \theta)) + \sum_{k=1}^{K} \log |\det Df_k(\mathbf{x}_i \,|\, \theta)\,|$$

Linear Flows

A linear transformation can be a flow if the matrix is invertible

$$f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

Inverse: $f^{-1}(\mathbf{z}) = \mathbf{A}^{-1}(\mathbf{z} - \mathbf{b})$

Determinant: $\det Df(\mathbf{x}) = \det \mathbf{A}$

Problem:

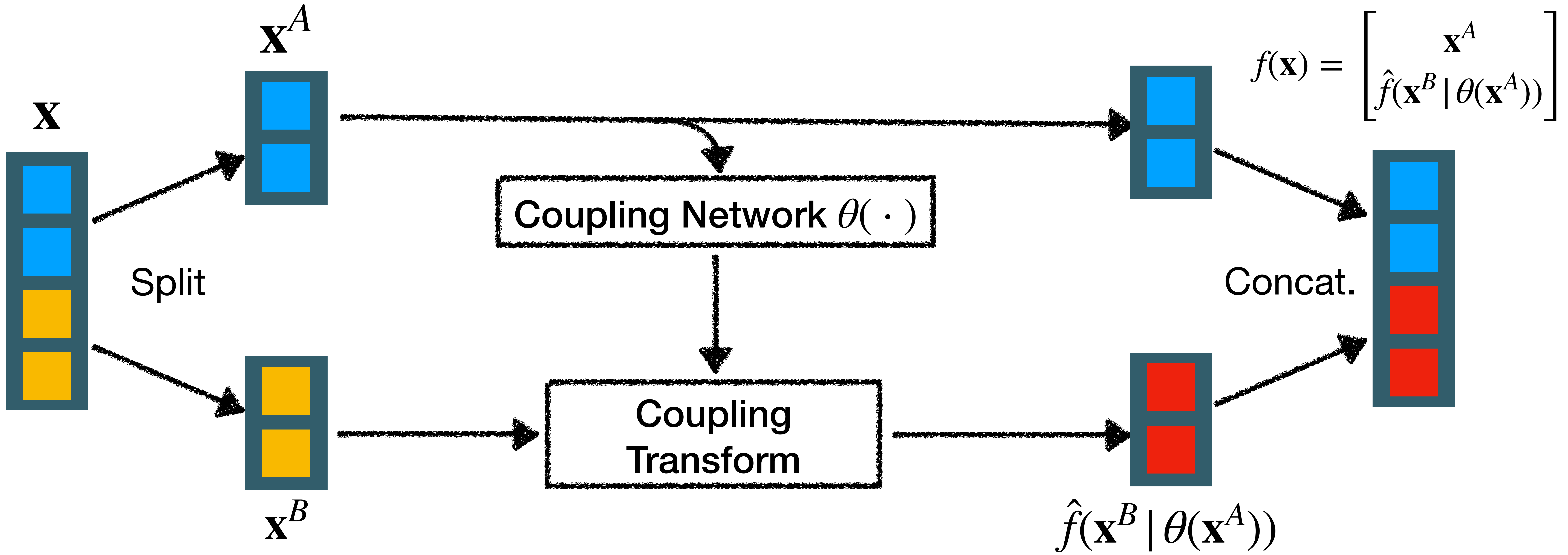
- Inexpressive (linear functions are closed under composition)
- Determinant/inverse could be $O(d^3)$

Linear Flows

Restricting the form of the matrix can reduce the determinant/inverse costs

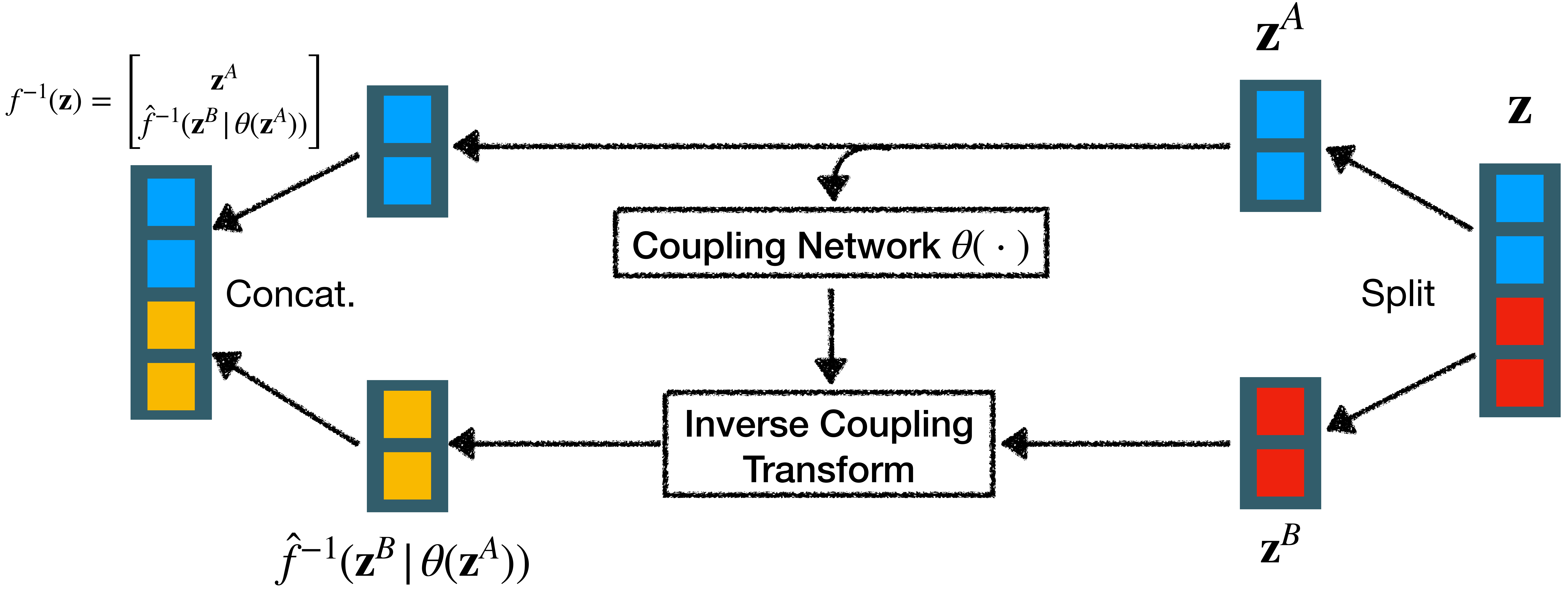
	Inverse	Determinant
Full	$O(d^3)$	$O(d^3)$
Diagonal	$O(d)$	$O(d)$
Triangular	$O(d^2)$	$O(d)$
Block Diagonal	$O(c^3 d)$	$O(c^3 d)$
LU Factorized *[Kingma and Dhariwal 2018]*	$O(d^2)$	$O(d)$
Spatial Convolution *[Hoogeboom et al 2019; Karami et al., 2019]*	$O(d \log d)$	$O(d)$
1x1 Convolution *[Kingma and Dhariwal 2018]*	$O(c^3 + c^2 d)$	$O(c^3)$

Coupling Flows



[Figure adapted from Jason Yu]

Coupling Flows: Inverse



[Figure adapted from Jason Yu]

Coupling Flows

Jacobian:

$$Df(\mathbf{x}) = \begin{bmatrix} \mathbf{I} & 0 \\ \frac{\partial}{\partial \mathbf{x}^A} \hat{f}(\mathbf{x}^B \,|\, \theta(\mathbf{x}^A)) & D\hat{f}(\mathbf{x}^B \,|\, \theta(\mathbf{x}^A)) \end{bmatrix}$$

Determinant:

$$\det Df(\mathbf{x}) = \det D\hat{f}(\mathbf{x}^B \,|\, \theta(\mathbf{x}^A))$$

[Dinh et al 2014 & Dinh et al 2016]

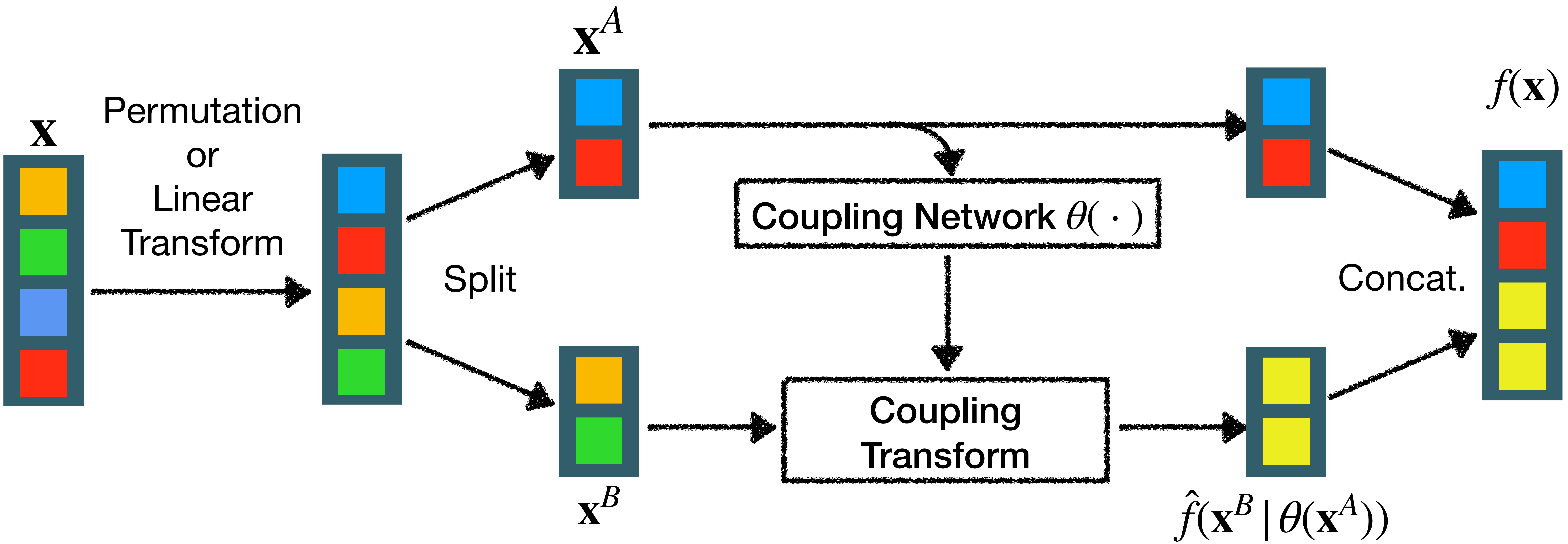
Coupling Flows

Can make $\theta(\mathbf{x}^A)$ arbitrarily complex, e.g., MLP, CNN, etc

Important to change the splits to ensure full expressiveness, but how?

[Dinh et al 2014 & Dinh et al 2016]

Coupling Flows



[Figure adapted from Jason Yu]

Coupling Flows

Coupling Transforms

- Additive [NICE, Dinh et al 2014]

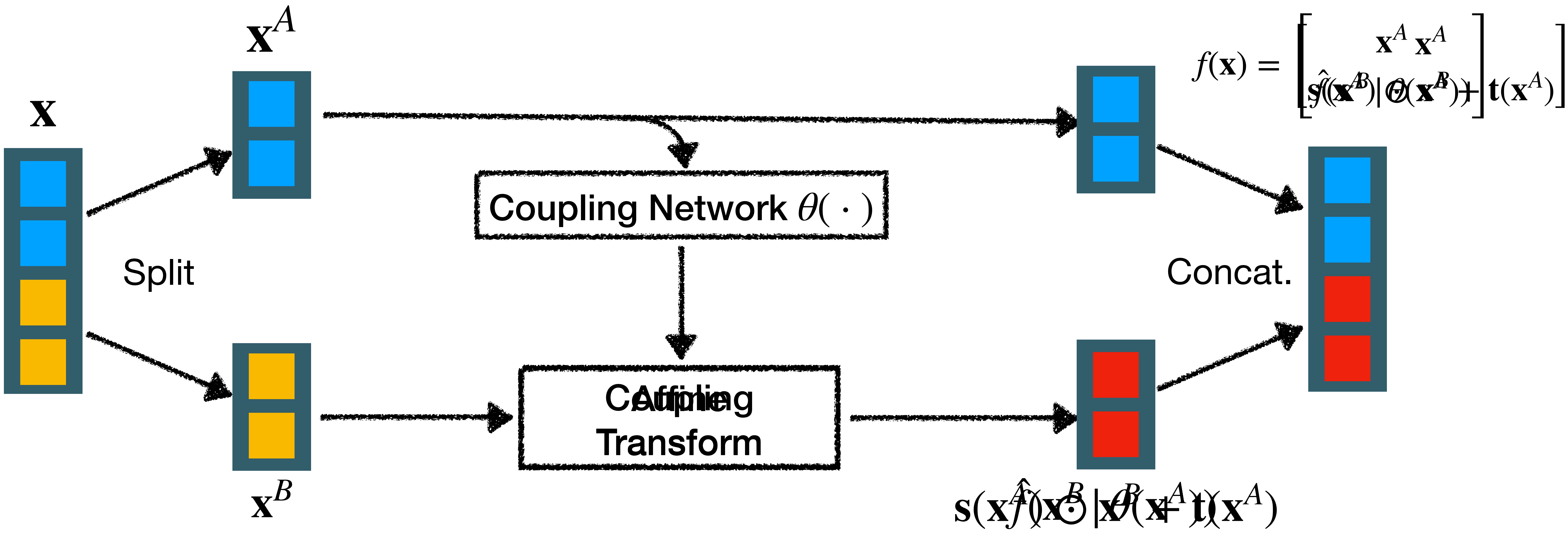
$$\hat{f}(\mathbf{x} \mid \mathbf{t}) = \mathbf{x} + \mathbf{t}$$

- Affine [RealNVP, Dinh et al 2016]

$$\hat{f}(\mathbf{x} \mid \mathbf{s}, \mathbf{t}) = \mathbf{s} \odot \mathbf{x} + \mathbf{t}$$

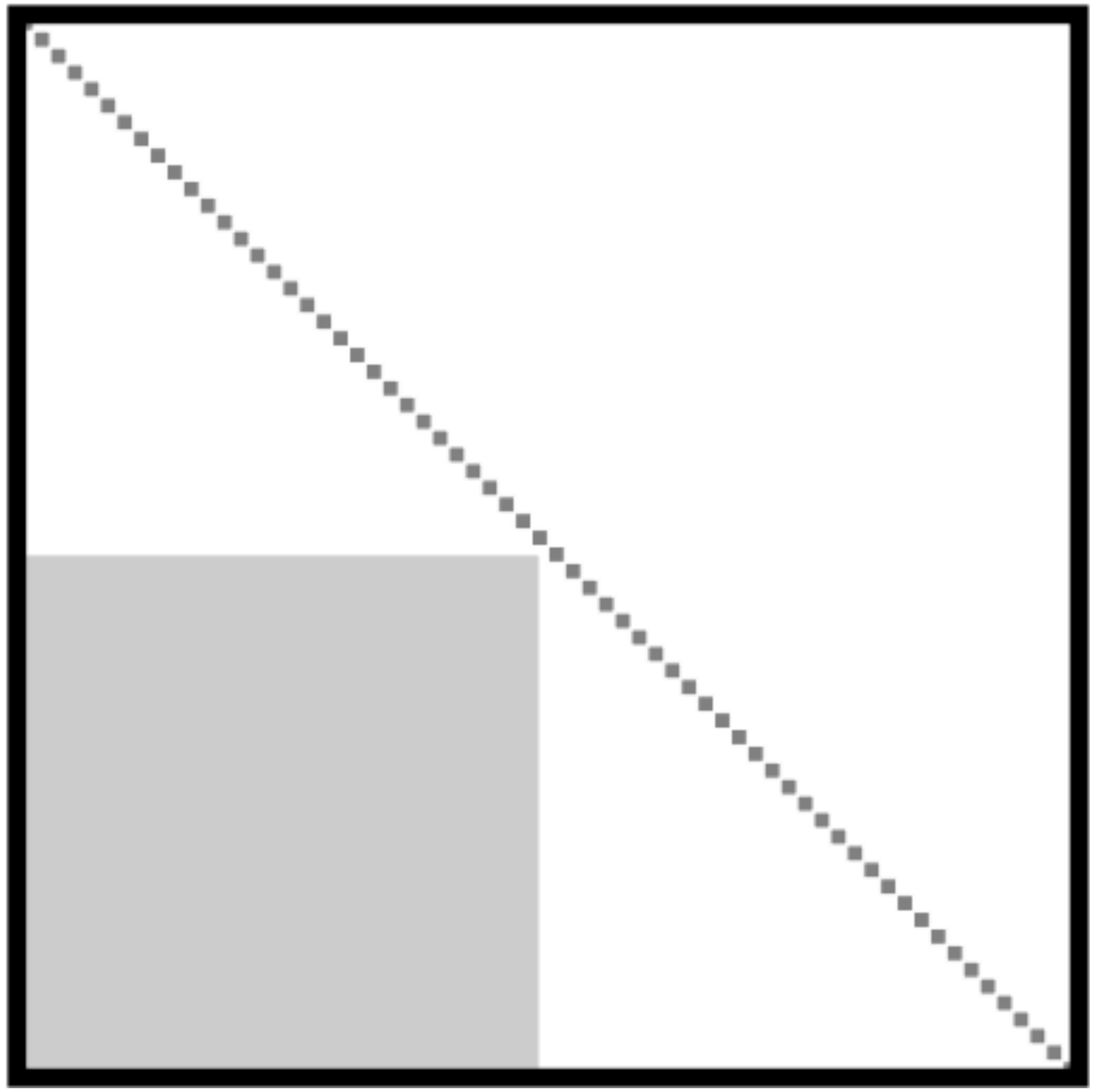
- MLPs [NAF, Huang et al, 2018], MixLogCDF [Flow++, Ho et al, 2019], Splines [Spline Flow, Durkan et al, 2019], etc…

Affine Coupling Flows



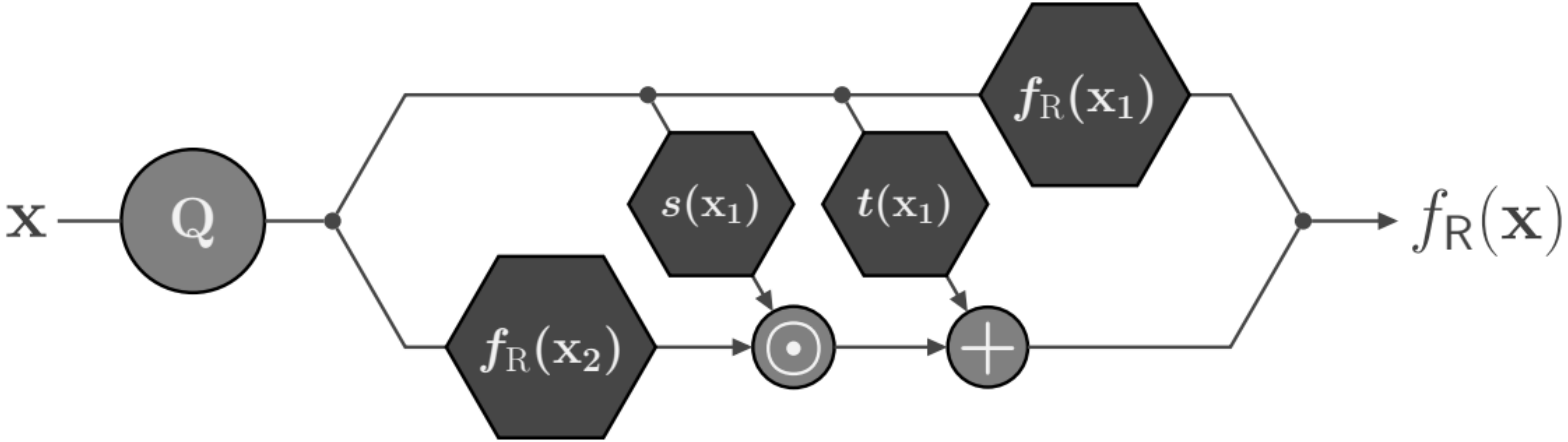
[Figure adapted from Jason Yu]

Recursive Coupling Flows: HINT



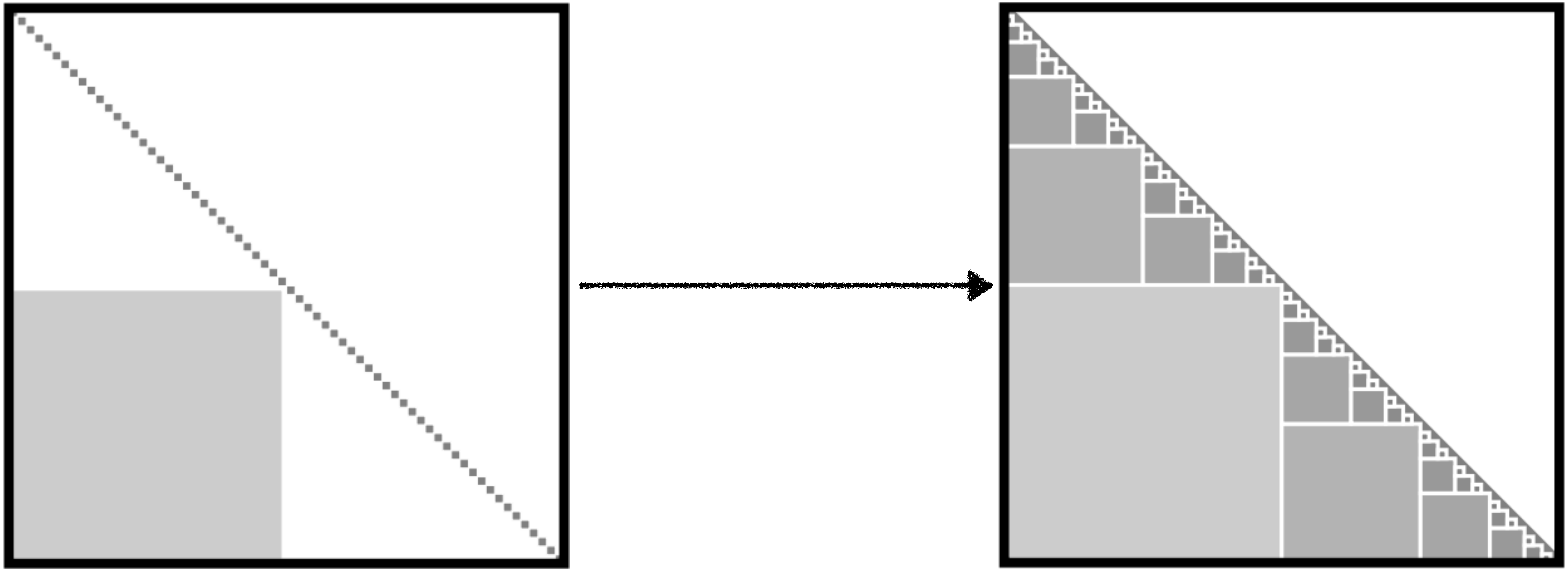
[Kruse & Detommaso et al. “HINT: Hierarchical Invertible Neural Transport for Density Estimation and Bayesian Inference”. AAAI 2021.]

Recursive Coupling Flows: HINT



[Kruse & Detommaso et al. "HINT: Hierarchical Invertible Neural Transport for Density Estimation and Bayesian Inference". AAAI 2021.]

Recursive Coupling Flows: HINT



[Kruse & Detommaso et al. “HINT: Hierarchical Invertible Neural Transport
for Density Estimation and Bayesian Inference”. AAAI 2021.]

Autoregressive Models as Flows

Autoregressive models are a form of normalizing flow

$$p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i \,|\, \mathbf{x}_{<i})$$

Autoregressive Models as Flows

Gaussian marginals

$$p(x_i \mid \mathbf{x}_{<i}) = \mathcal{N}\left(x_i \mid \mu(\mathbf{x}_{<i}), \sigma^2(\mathbf{x}_{<i})\right)$$

Reparameterization trick:

$$x_i = \mu(\mathbf{x}_{<i}) + \sigma(\mathbf{x}_{<i}) z_i \quad \text{where} \quad z_i \sim \mathcal{N}(0,1)$$

[Kingma et al NeurIPS 2016; Papamakarios et al NeurIPS 2017]

Autoregressive Models as Flows

(Affine) Autoregressive Flow:

$$f_i^{-1}(\mathbf{z}) = \mu(f_{<i}^{-1}(\mathbf{z}_{<i})) + \sigma(f_{<i}^{-1}(\mathbf{z}_{<i}))z_i$$

$$f_i(\mathbf{x}) = \frac{x_i - \mu(\mathbf{x}_{<i})}{\sigma(\mathbf{x}_{<i})}$$

Determinant:

$$\det Df(\mathbf{x}) = \prod_i \sigma^{-1}(\mathbf{x}_{<i})$$

[Kingma et al NeurIPS 2016;
Papamakarios et al NeurIPS 2017]

Autoregressive Models as Flows

Sampling is sequential and slow

Density evaluation, ie, computing $f(\mathbf{x})$, can be done in parallel

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

[Kingma et al NeurIPS 2016]

Autoregressive Models as Flows

(Affine) Inverse Autoregressive Flow:

$$f_i(\mathbf{x}) = \mu(f_{<i}(\mathbf{x}_{<i})) + \sigma(f_{<i}(\mathbf{x}_{<i}))x_i$$

$$f_i^{-1}(\mathbf{z}) = \frac{z_i - \mu(\mathbf{z}_{<i})}{\sigma(\mathbf{z}_{<i})}$$

Determinant:

$$\det Df(\mathbf{x}) = \prod_i \sigma(f_{<i}(\mathbf{x}_{<i}))$$

[Kingma et al NeurIPS 2016; Papamakarios et al NeurIPS 2017]

Multi-Scale Flows

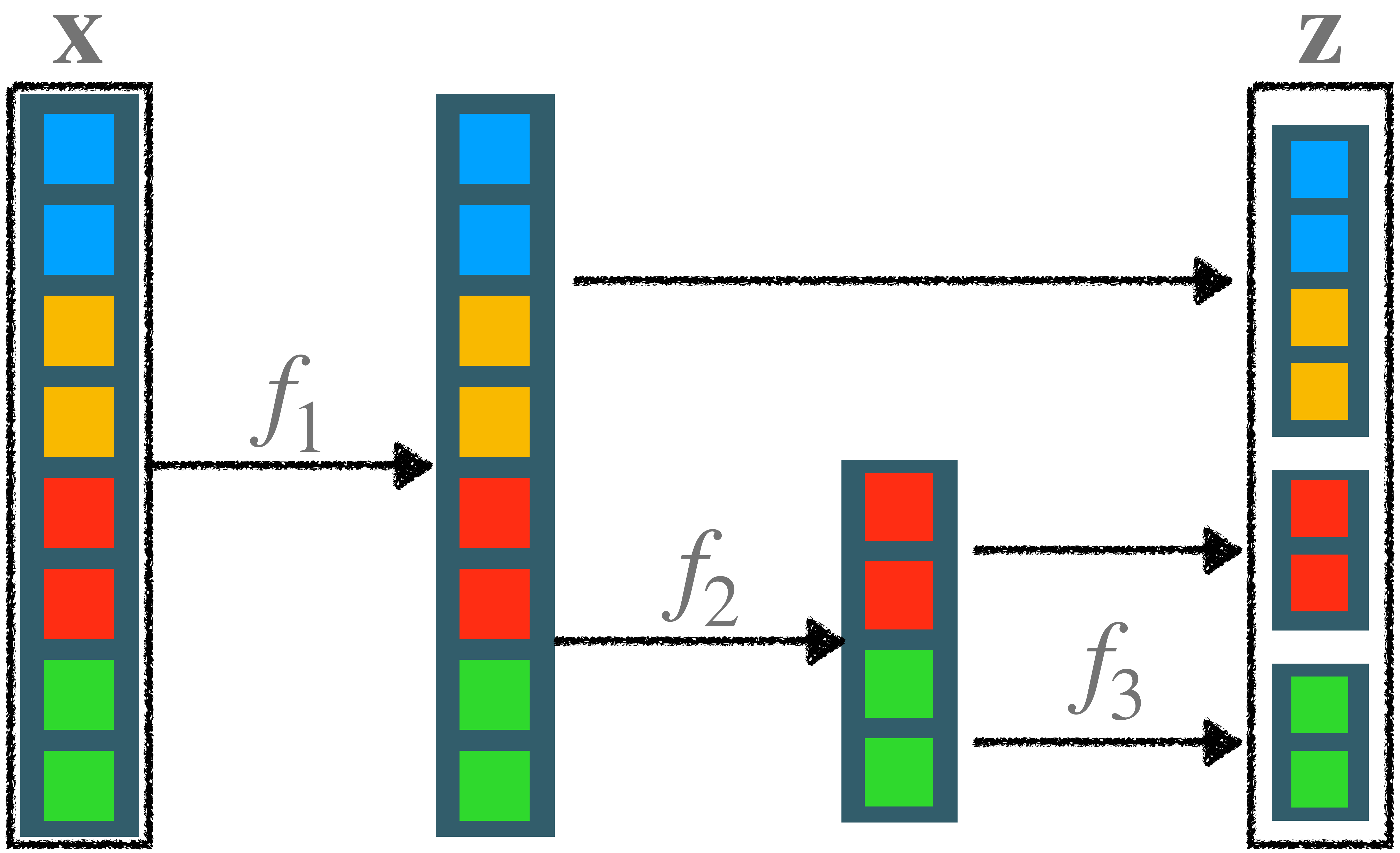
A flow preserves dimensionality, but this is expensive in high dimensions

Just stop using subsets of dimensions

Practically, acts like dropping dimensions

[Dinh et al 2016]

Multi-Scale Flows



[Dinh et al 2016]

Multi-Scale Flows

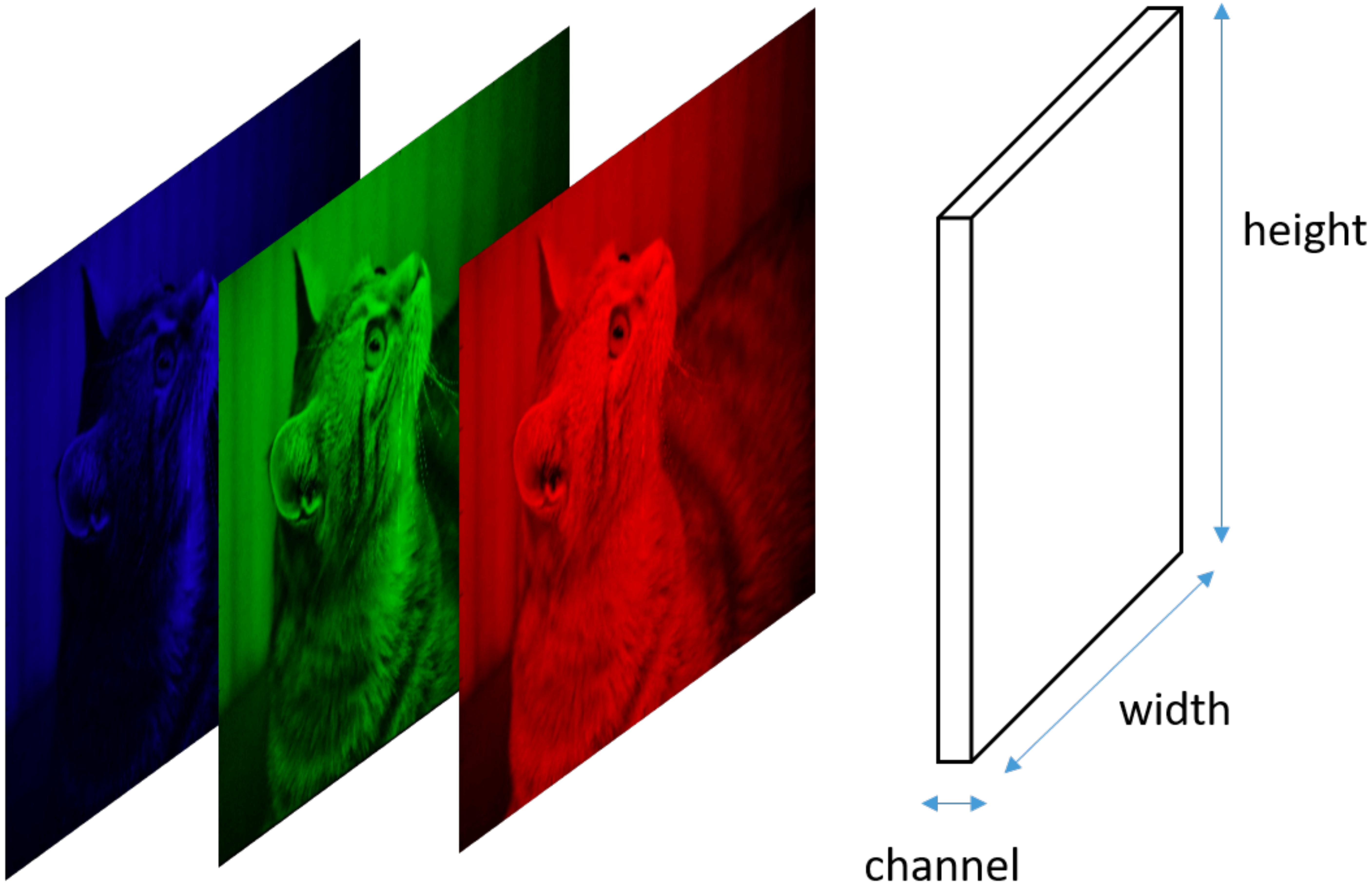
Multi-scale flows are just a special coupling flow

$$f(\mathbf{x}) = (\mathbf{x}^A, \hat{f}(\mathbf{x}^B \,|\, \theta))$$

- Important: must track "dropped" dimensions to preserve invertibility

Multi-Scale Flows

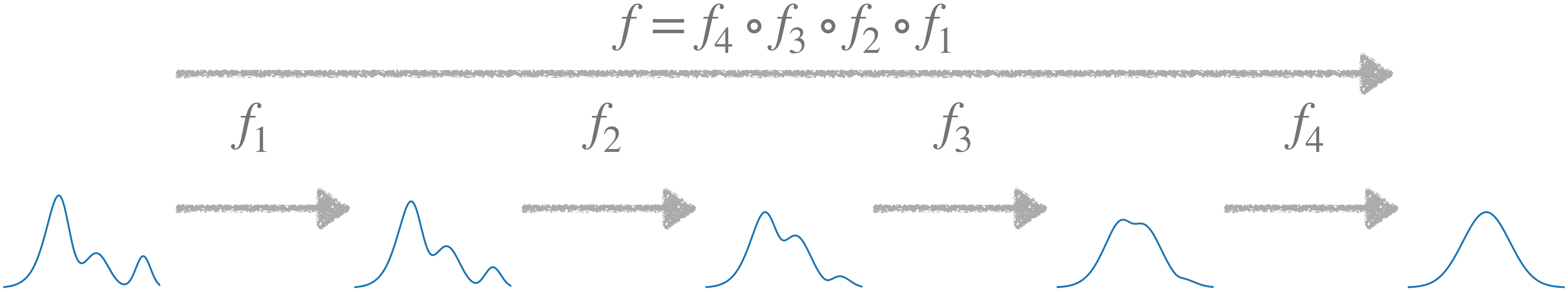
How do we split the dimensions for images?



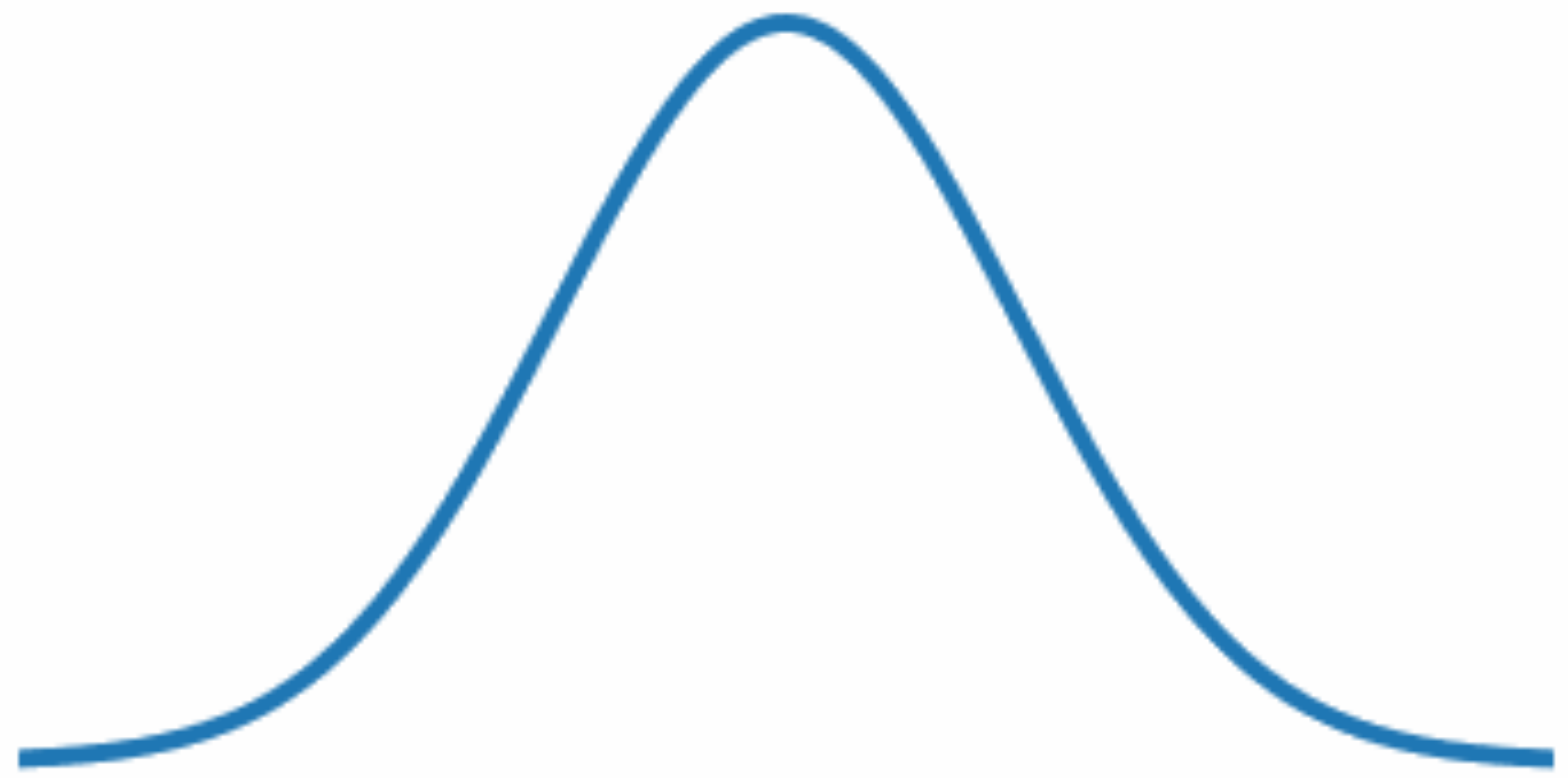
Multi-Scale Flows

“Squeeze” the spatial arrangement to get more channels

Discrete-time Normalizing Flows



Continuous-time Normalizing Flows



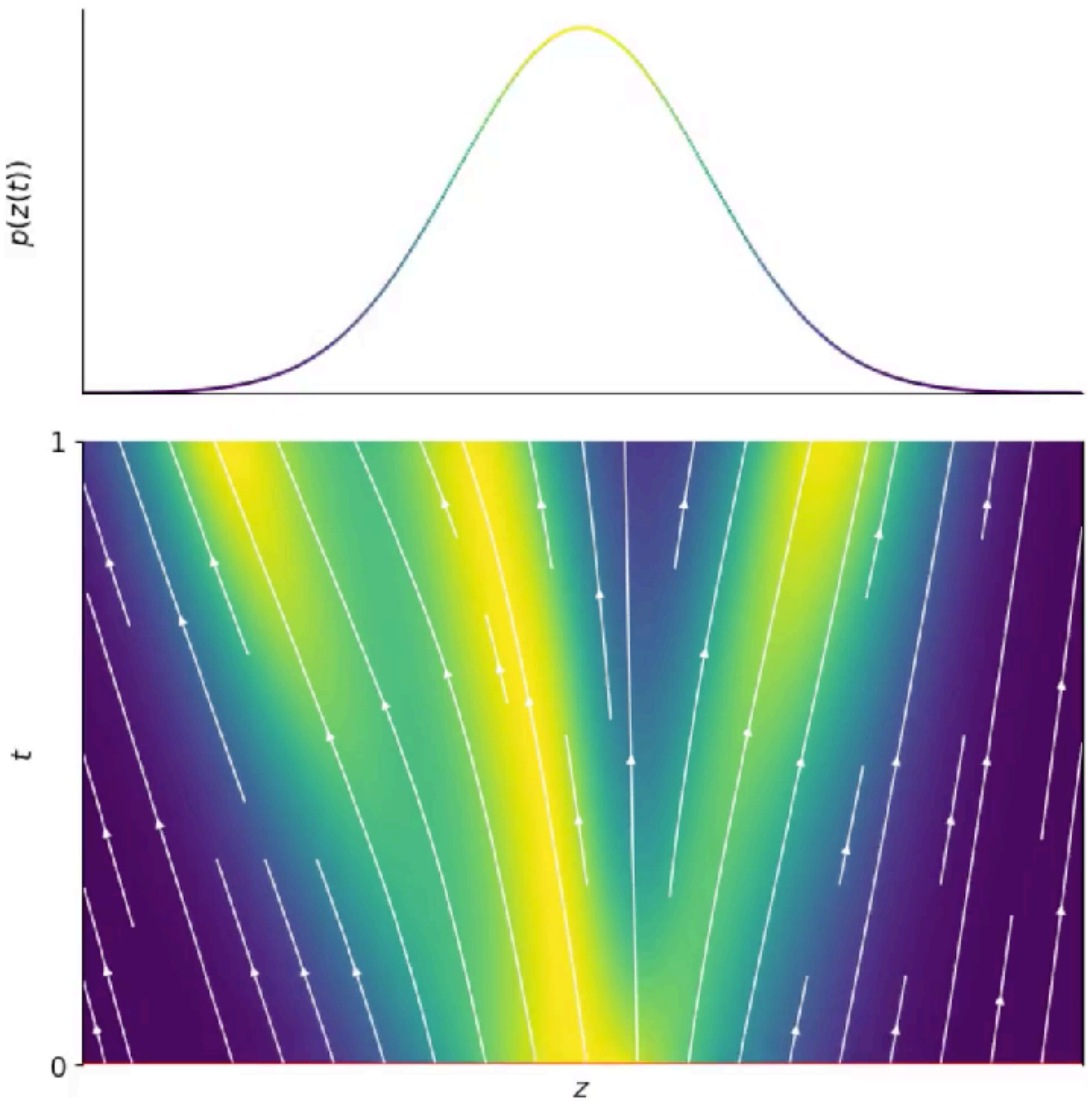
FFJORD

ODEs as a flow

$$f(\mathbf{x}) = \mathbf{y}_0 + \int_0^1 h(t, \mathbf{y}_t)dt \text{ with } \mathbf{y}_0 = \mathbf{x}$$

Inverse:

$$f^{-1}(\mathbf{z}) = \mathbf{y}_1 + \int_1^0 h(t, \mathbf{y}_t)dt \text{ with } \mathbf{y}_1 = \mathbf{z}$$



[Grathwohl and Chen et al 2019]

FFJORD

Continuous change of variable

$$\log p_{\mathbf{X}}(\mathbf{x}) = \log p_{\mathbf{Z}}(f(\mathbf{x})) + \int_0^1 Tr\left(\frac{\partial h}{\partial \mathbf{y}}(t, \mathbf{y}_t)\right) dt$$

[Grathwohl and Chen et al 2019]

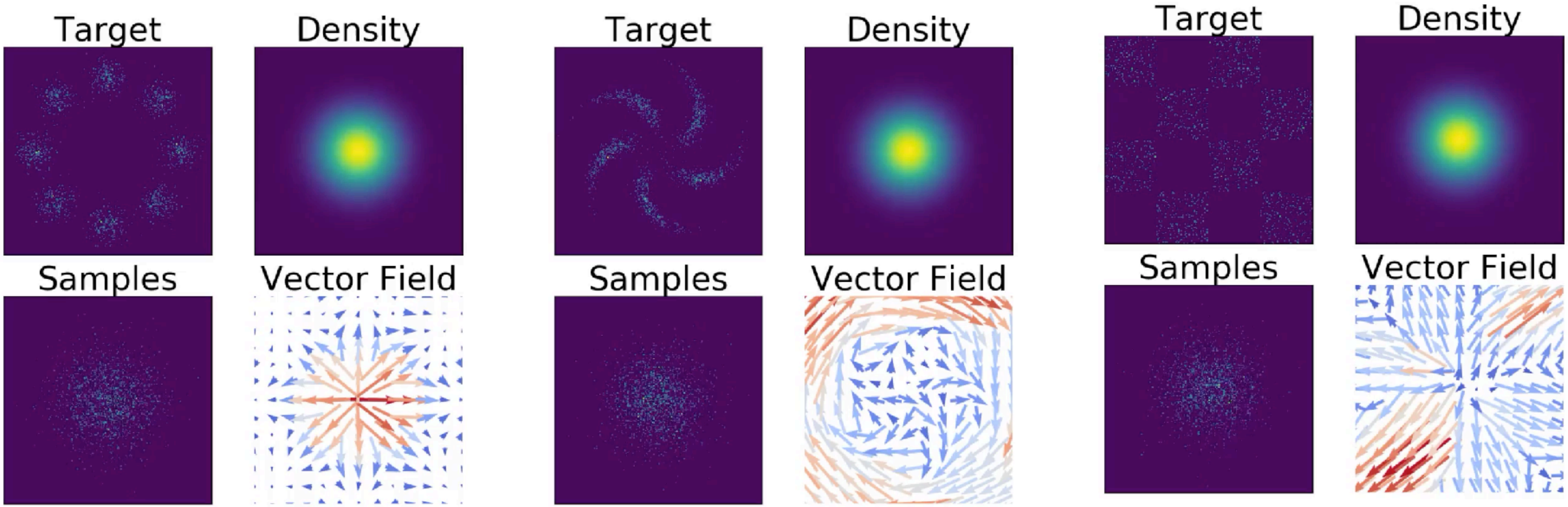
FFJORD

Hutchinson Trace Estimator

$$\int_0^1 Tr\left(\frac{\partial h}{\partial \mathbf{y}}(t, \mathbf{y}_t)\right) dt = \mathbb{E}_{\epsilon \sim p(\epsilon)}\left[\int_0^1 \epsilon^T \frac{\partial h}{\partial \mathbf{y}}(t, \mathbf{y}_t)\epsilon dt\right]$$

[Grathwohl and Chen et al 2019]

FFJORD



[Grathwohl and Chen et al 2019]

Training PGMs with Maximum Likelihood

Normalizing Flows are a model of continuous data

Pixel intensities are typically discrete or **quantized**

Training PGMs with Maximum Likelihood

ML learning of continuous models w/ discrete data can cause singularities

Really want to optimize

Probability Density of
Continuous Values

$$P_{\mathbf{Y}}(\mathbf{y}) = \int_{[0,1]^D} p_{\mathbf{X}}(\mathbf{y} + \mathbf{u}) p_{\mathbf{U}}(\mathbf{u}) d\mathbf{u}$$

Probability of
Discrete Values

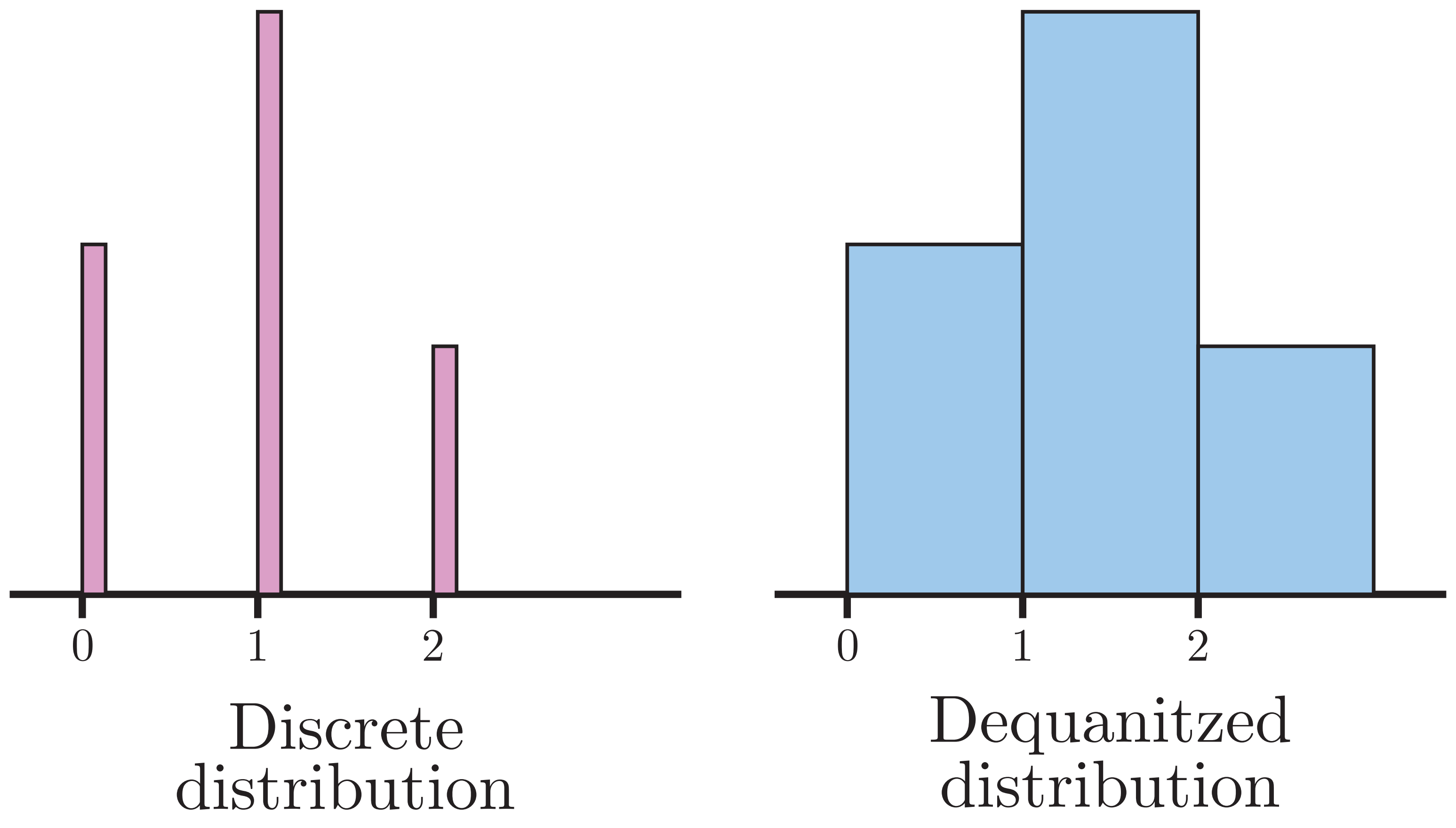
Probability Density of
Quantization Noise

Uniform Dequantization

During training, **dequantize** the data (i.e., add noise)

$$P_{\mathbf{Y}}(\mathbf{y}) = \int_{[0,1]^D} p_{\mathbf{X}}(\mathbf{y}+\mathbf{u})p_{\mathbf{U}}(\mathbf{u})d\mathbf{u}$$

$$\approx \frac{1}{K}\sum_{k=1}^{K} p_{\mathbf{X}}(\mathbf{y}+\mathbf{u}_k)$$



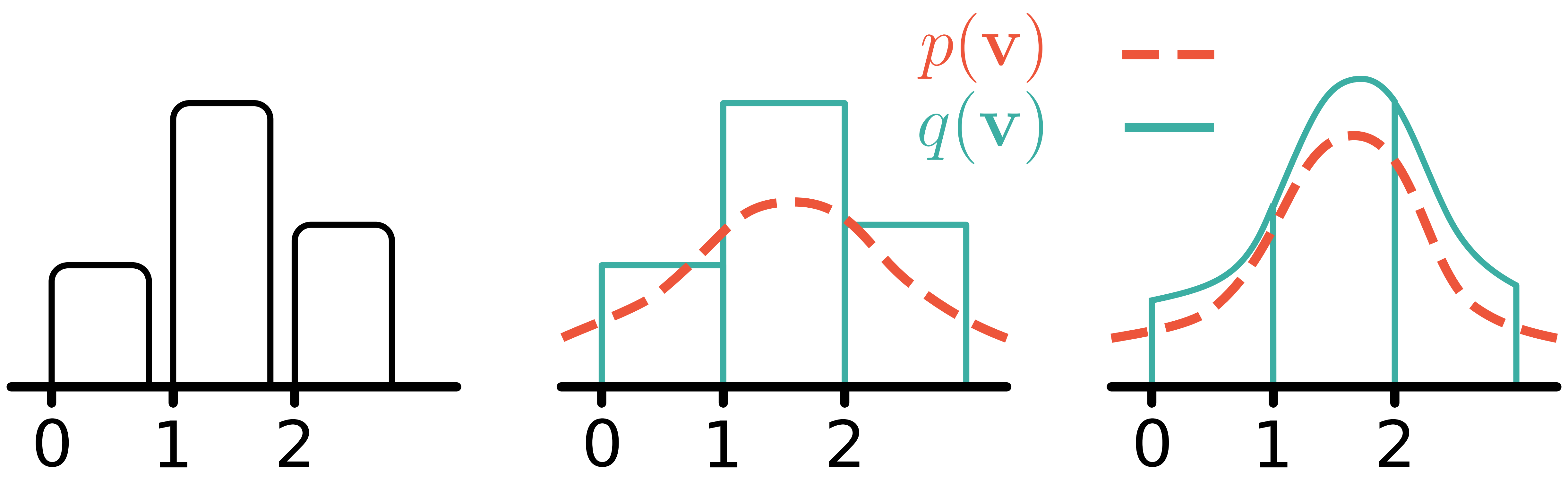
Simplest choice of $p_{\mathbf{U}}$ is uniform

Variational Dequantization

View $p_{\mathbf{U}}$ as a variational distribution and learn it

$$\log P_{\mathbf{Y}}(\mathbf{y}) \geq \int_{[0,1]^D} \log \frac{p_{\mathbf{X}}(\mathbf{y}+\mathbf{u})}{p_{\mathbf{U}}(\mathbf{u}\,|\,\mathbf{y})} d\mathbf{u}$$

$$\approx \frac{1}{K}\sum_{k=1}^{K} \log \frac{p_{\mathbf{X}}(\mathbf{y}+\mathbf{u}_k)}{p_{\mathbf{U}}(\mathbf{u}_k\,|\,\mathbf{y})}$$



[Hoogeboom et al 2020]

[Ho et al, 2019]

Common Flow Architectures for Images

	Transformations	Dequantization	Multi-Scale
NICE [Dinh et al, 2014]	Additive Coupling + Diagonal Linear	Uniform	No
RealNVP [Dinh et al, 2016]	Affine Coupling + Channelwise Permutation	Uniform	Yes
Glow [Kingma and Dhariwal, 2018]	Affine Coupling + Channelwise Linear	Uniform	Yes
Flow++ [Ho et al, 2019]	MixLogCDF Coupling + Channelwise Linear	Variational	Yes

Conclusions

References

Dequantization

- Uria, B., Murray, I., & Larochelle, H. (2013). RNADE: The real-valued neural autoregressive density-estimator. In Advances in Neural Information Processing Systems (pp. 2175-2183).
- Theis, L., Oord, A. V. D., & Bethge, M. (2015). A note on the evaluation of generative models. arXiv preprint arXiv:1511.01844.
- Hoogeboom, E., Cohen, T. S., & Tomczak, J. M. (2020). Learning Discrete Distributions by Dequantization. arXiv preprint arXiv:2001.11235.

Architectures

- Dinh, L., Krueger, D., & Bengio, Y. (2014). Nice: Non-linear independent components estimation. arXiv preprint arXiv:1410.8516.
- Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. arXiv preprint arXiv:1605.08803.
- Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In Advances in neural information processing systems (pp. 10215-10224).
- Ho, J., Chen, X., Srinivas, A., Duan, Y., & Abbeel, P. (2019). Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design. In International Conference on Machine Learning (pp. 2722-2730).

Review Articles

- Kobyzev, I., Prince, S., & Brubaker, M. (2020). Normalizing flows: An introduction and review of current methods. IEEE Transactions on Pattern Analysis and Machine Intelligence.
- Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2019). Normalizing flows for probabilistic modeling and inference. arXiv preprint arXiv:1912.02762.