Introduction to Normalizing Flows

2021-05-21

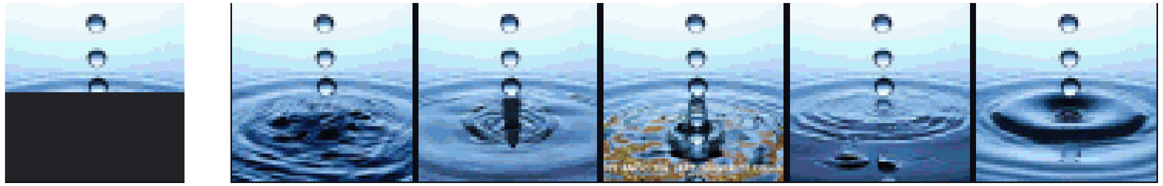
손형욱

hyounguk.shon@kaist.ac.kr

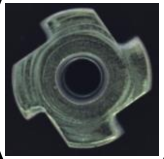
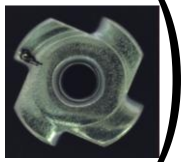
What are generative models?

- Generative models approximate some observed data distribution $p_{\mathbf{X}}(\mathbf{x})$.
- Generative modelling is a typical unsupervised learning problem.

Generate new samples / Conditionally complete



Evaluate the density

$p\Bigg($  $\Bigg)$ vs. $p\Bigg($  $\Bigg)$

Learn latent structure

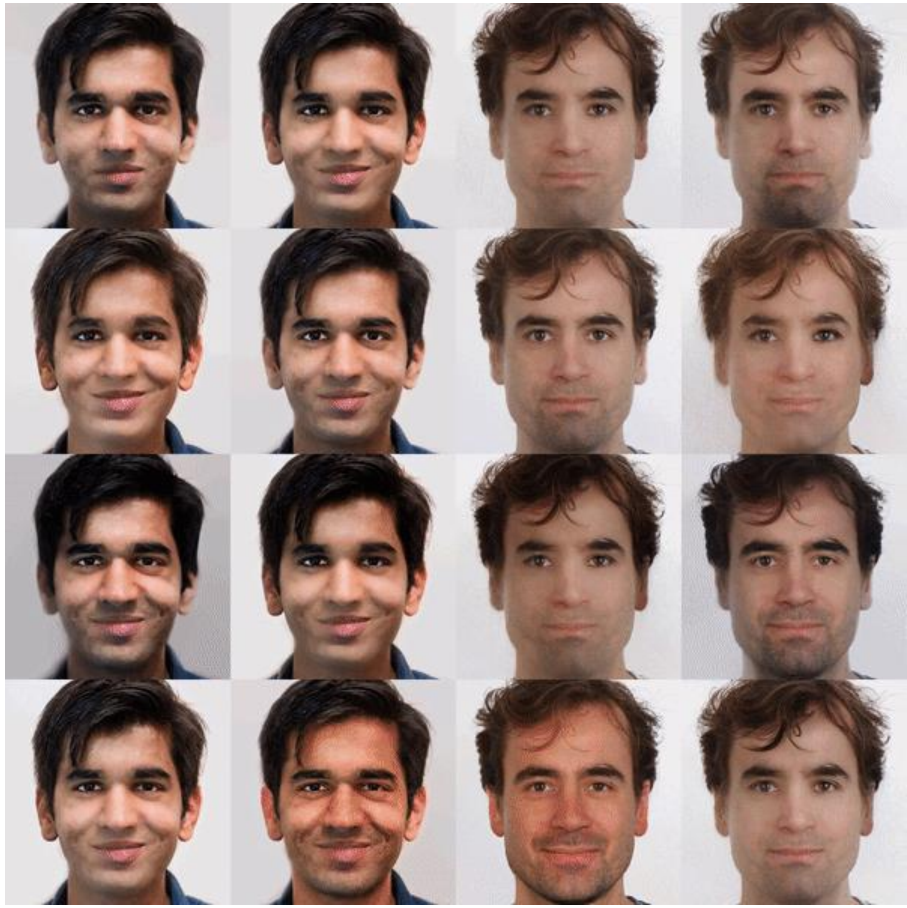
$\mathbf{z}$ $\Rightarrow$

Smile Gray Hair Mustache

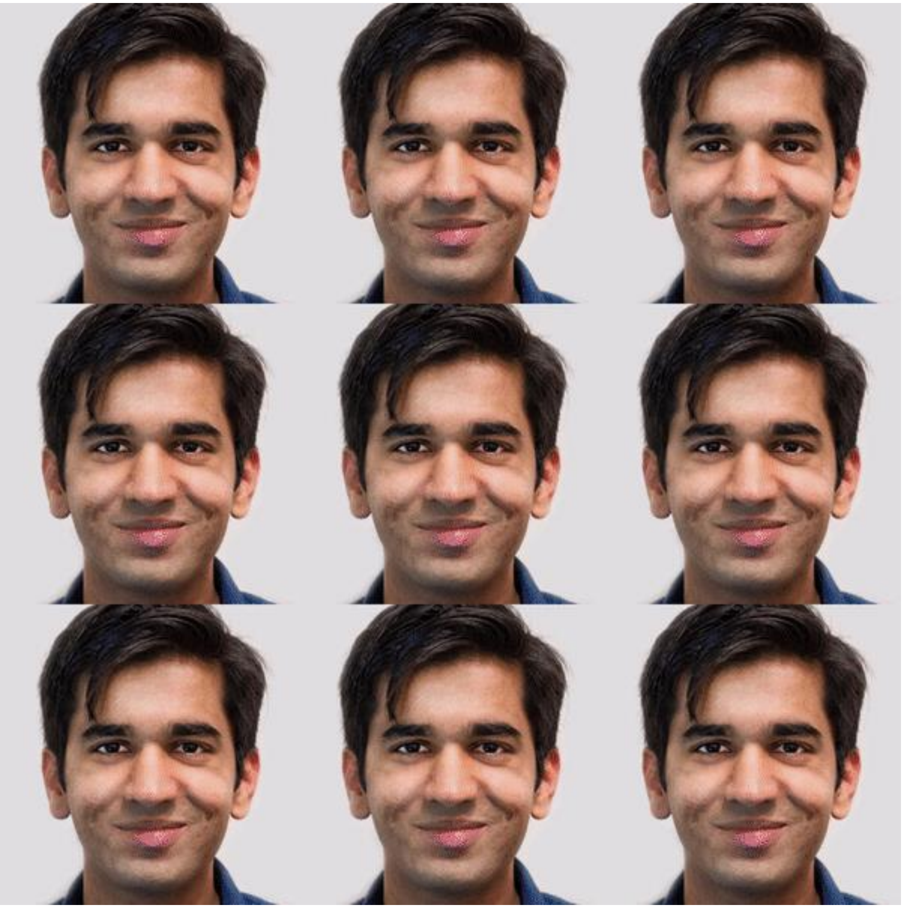
What are Normalizing Flows?

- Normalizing Flows are a type of PGM built on invertible transformations.
- They are generally:
    - Efficient to evaluate / sample from $p_{\mathbf{X}}(\mathbf{x})$.
    - Flexible, useful latent representation, maximum likelihood training.
- Firstly introduced in
- Tabak and Venden-Eijnden (2010), Tabak and Turner (2013), Agnelli eet al. (2010), …
- Popularized by: Rezende and Mohamed (2015): variational inference, Dinh et al. (2015): density estimation
- Invertible Neural Networks

Glow (2018 OpenAI)



Manipulating attributes
(no labels given at training)



Interpolating between points

Comparing against other generative models

Normalizing Flows	Other generative models
Exact evaluation of $p(\mathbf{x})$ Exact evaluation of $\mathbf{z}$	• GAN Avoids evaluating $p(\mathbf{x})$ altogether. No guarantee of full support over $p(\mathbf{x})$ • VAE Only can evaluate the lower bound of $p(\mathbf{x})$ Only approximate distribution $\mathrm{q}(\mathbf{z}\|\mathbf{x})$ • EBM Only can evaluate unnormalized $p(\mathbf{x})$

The idea behind Normalizing Flows

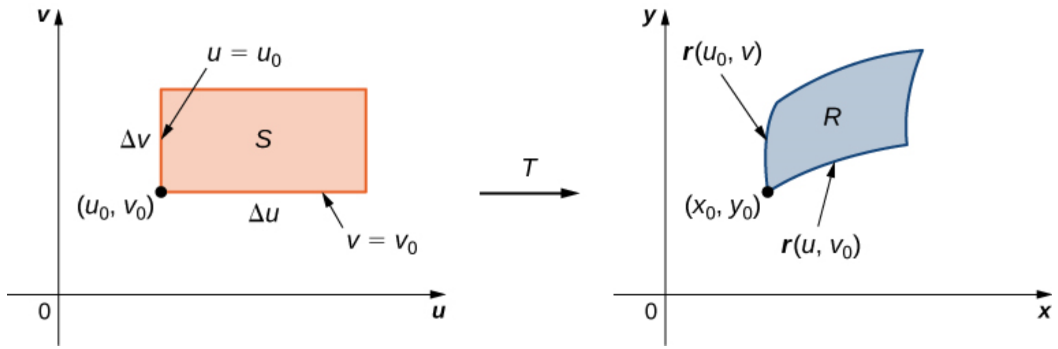
- Normalizing Flow:
  - transformation of a simple probability distribution (e.g., a normal distribution) into a more complex distribution by a sequence of invertible and differentiable mappings.
- The probability density of a sample can be evaluated by transforming it back to the original simple distribution and then computing the product of i) the density of the inverse-transformed sample under this distribution and ii) the associated change in volume induced by the sequence of inverse transformations.

Basic principles

- For an invertible & differentiable function $f(\cdot)\colon \mathrm{x} \longmapsto \mathrm{z}$

$$p_{\mathrm{X}}(\mathrm{x}) \cdot d\mathrm{x} = p_{\mathrm{Z}}(\mathrm{z}) \cdot d\mathrm{z}$$

mass = density × volume



- This leads to the change of variables formula:

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(\mathbf{z}) \cdot \left|\det\left(\frac{d\mathbf{z}}{d\mathbf{x}}\right)\right|$$

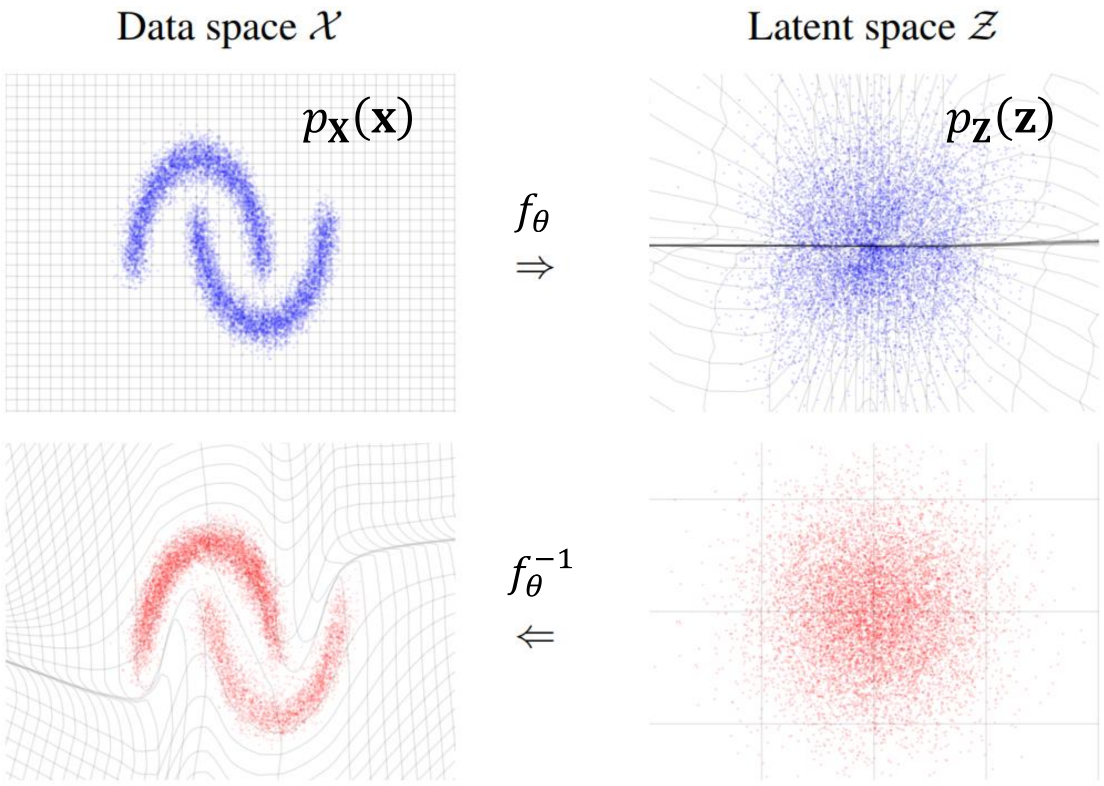
volume correction term

Basic principles

- Base Measure: the choice of $p_{\mathbf{Z}}(\mathbf{z})$. Typically selected as $\mathcal{N}(\mathbf{0}, \mathbf{I})$.
- Flow: $f(\mathbf{x})$ - invertible and differentiable function.

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

- Density Evaluation:
  - Compute $\mathbf{z} = f(\mathbf{x})$
  - $p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(\mathbf{z}) \cdot \left|\frac{d\mathbf{z}}{d\mathbf{x}}\right|$
- Sampling:
  - Sample a random vector $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
  - Compute the inverse transform $f^{-1}(\mathbf{z})$.



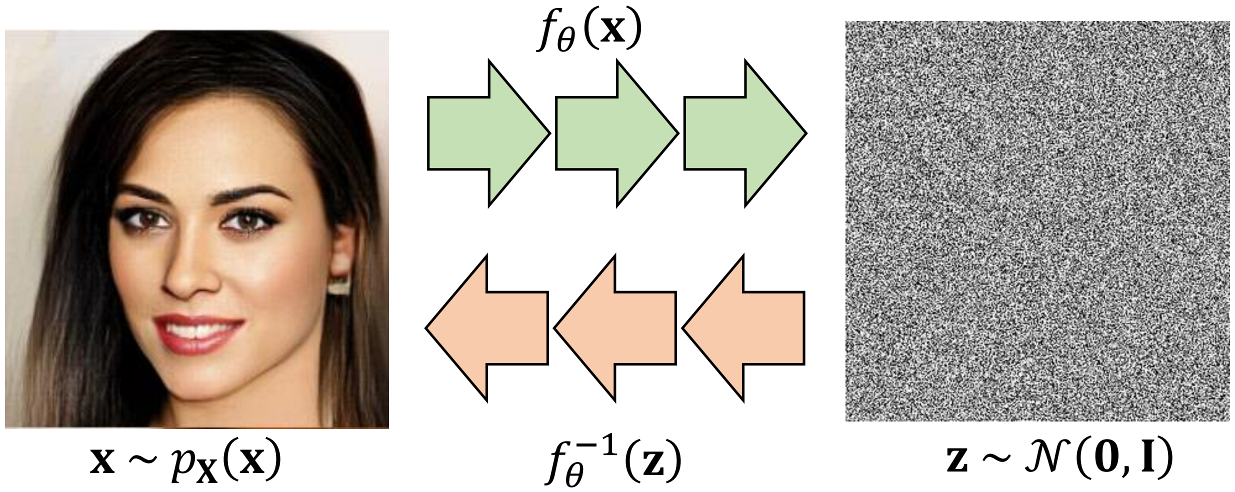
Density Estimation using Real NVP. Dinh, Sohl-Dickstein, Bengio (2017)

Basic principles

- Base Measure: the choice of $p_{\mathbf{Z}}(\mathbf{z})$. Typically selected as $\mathcal{N}(\mathbf{0}, \mathbf{I})$.
- Flow: $f(\mathbf{x})$ - invertible and differentiable function.

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

- Density Evaluation:
  - Compute $\mathbf{z} = f(\mathbf{x})$
  - $p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(\mathbf{z}) \cdot \left|\frac{d\mathbf{z}}{d\mathbf{x}}\right|$
- Sampling:
  - Sample a random vector $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
  - Compute the inverse transform $f^{-1}(\mathbf{z})$.

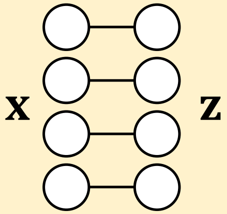


Flow Layers

- Flow is a parametric function with special properties:
  - invertible & differentiable.
  - Inverse transform $f^{-1}(\mathbf{z})$ must be efficiently computable.
  - Jacobian determinant must be efficiently computable.
- (Example) Linear Flow
  - $f(\mathbf{x}) = \mathbf{Ax} + \mathbf{b}$ with invertible $\mathbf{A}$
  - Inverse: $f^{-1}(\mathbf{z}) = \mathbf{A}^{-1}(\mathbf{z} - \mathbf{b})$
  - Determinant: $\det Df(\mathbf{x}) = \det(\mathbf{A})$
- Problem:
  - Inverse/determinant of a dense matrix is expensive - $\mathcal{O}(dim^3)$
  - To reduce cost, we enforce structures.
  - For example, we can restrict $\mathbf{A}$ to be a diagonal matrix. - $\mathcal{O}(dim)$
- Designing flow architecture is the core research problem.

A simple diagonal linear flow.

$$\mathbf{A} = \begin{bmatrix} a_1 & 0 & 0 & 0 \\ 0 & a_2 & 0 & 0 \\ 0 & 0 & a_3 & 0 \\ 0 & 0 & 0 & a_4 \end{bmatrix}$$



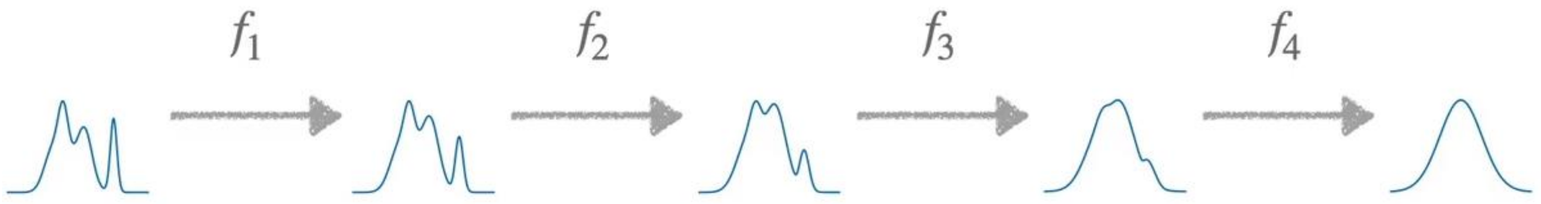
$$\mathbf{A}^{-1} = \begin{bmatrix} a_1^{-1} & 0 & 0 & 0 \\ 0 & a_2^{-1} & 0 & 0 \\ 0 & 0 & a_3^{-1} & 0 \\ 0 & 0 & 0 & a_4^{-1} \end{bmatrix}$$

$$\det(Df) = \mathrm{a}_1\mathrm{a}_2\mathrm{a}_3\mathrm{a}_4$$

Composition of Flows

- We can fit an arbitrarily complex distribution by stacking flows.

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



Normalizing direction

- Jacobian determinant

$$\det(Df) = \det(Df_K) \cdot \det(Df_{K-1}) \cdots \det(Df_1)$$

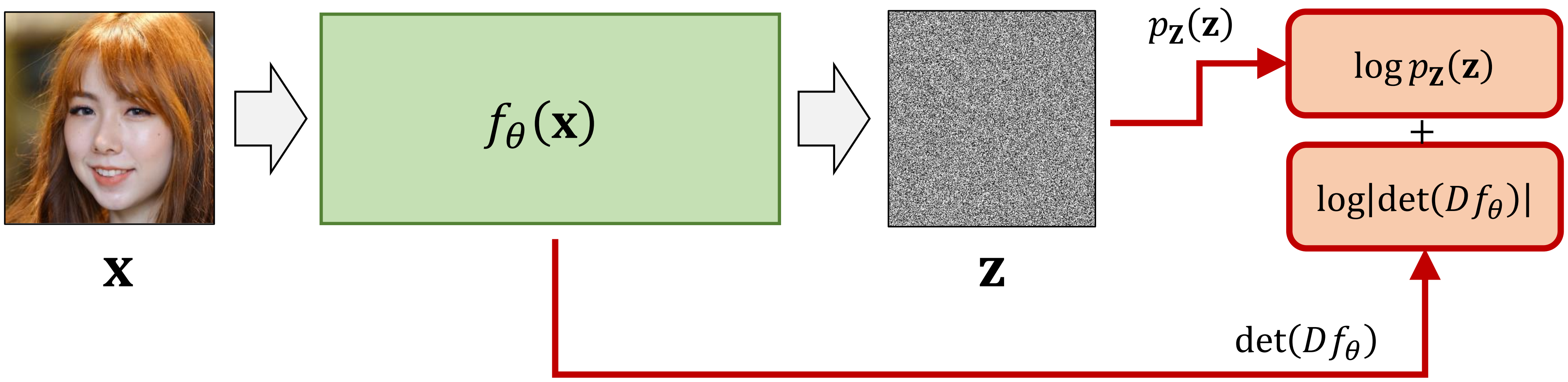
- Likelihood evaluation

$$\log p_{\mathbf{X}}(\mathbf{x}) = \log p_{\mathbf{Z}}(\mathbf{z}) + \log|\det(Df_K) \cdot \det(Df_{K-1}) \cdots \det(Df_1)|$$

Training Flows

- Unlike GANs (mini-max) or VAEs (variational inference)
- Flows simply train by maximum-likelihood:

$$\max_{\theta} \log p_{\mathbf{X}}(\mathbf{x}) = \max_{\theta} \left\{ \log p_{\mathbf{Z}}\big(f_\theta(\mathbf{x})\big) + \log\left|\det(D f_\theta)\right| \right\}$$

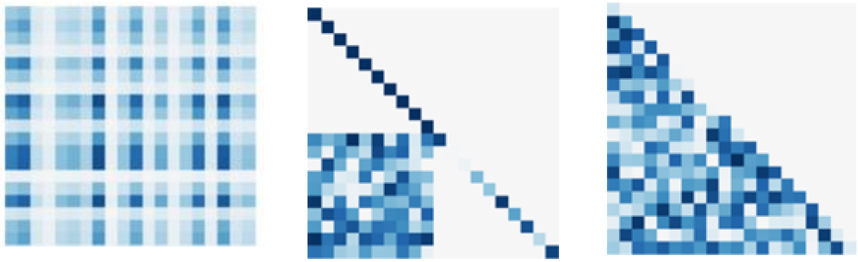


Computing inverse and determinants

- Restricting the form of the matrix can reduce 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]*	$O(c^3 + c^2 d)$	$O(c^3)$

Structured Jacobian matrices



Low-rank Sparse Triangular

Moving beyond linear flows:

- Coupling Flows ✓
- Autoregressive Flows ✓
- Inverse Autoregressive Flows ✓
- Multi-scale Flows ✓
- Sylvester Flows
- Neural Spline Flows
- Fourier Convolution
- Continuous Flows ✓
- Residual Flows
- ....

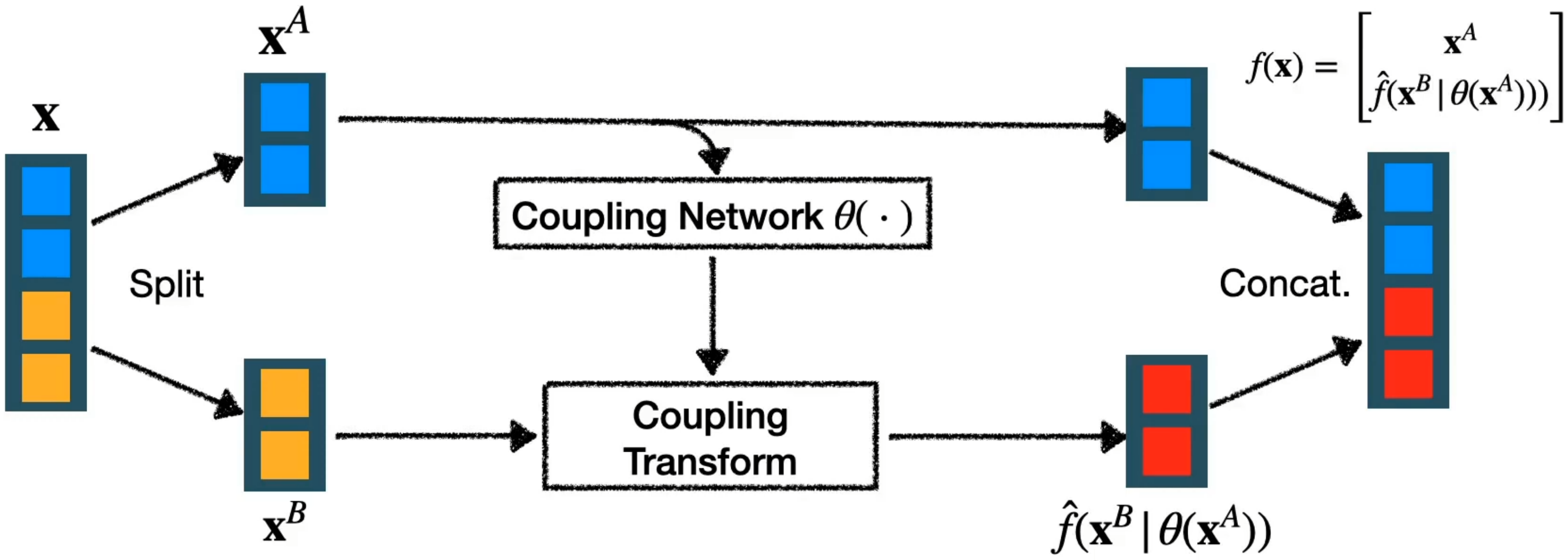
We shall take a look at a few popular flow operations.

Coupling Flows

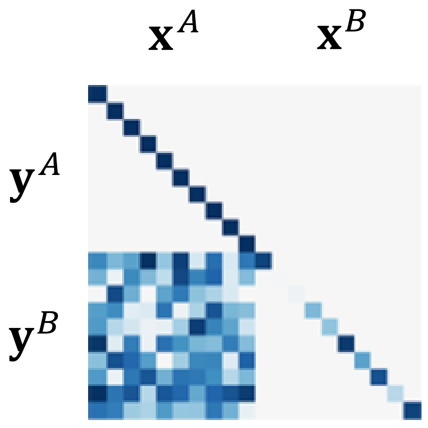
- A general strategy to construct non-linear flows.
- Splits the input $\mathbf{x}$ into $[\mathbf{x}^A \quad \mathbf{x}^B]$, and $f(\mathbf{x}) = [\mathbf{x}^A \quad \hat{f}(\mathbf{x}^B | \theta(\mathbf{x}^A))]$
- (Example) Affine coupling flows:

$$\mathbf{y} = \begin{cases} \mathbf{y}^A = \mathbf{x}^A \\ \mathbf{y}^B = \mathbf{s}(\mathbf{x}^A) \odot \mathbf{x}^B + \mathbf{t}(\mathbf{x}^A) \end{cases}$$

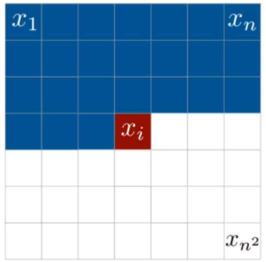
Inverse: straightforward since $\mathbf{x}^A = \mathbf{y}^A$
Determinant: product of Jacobian diagonals



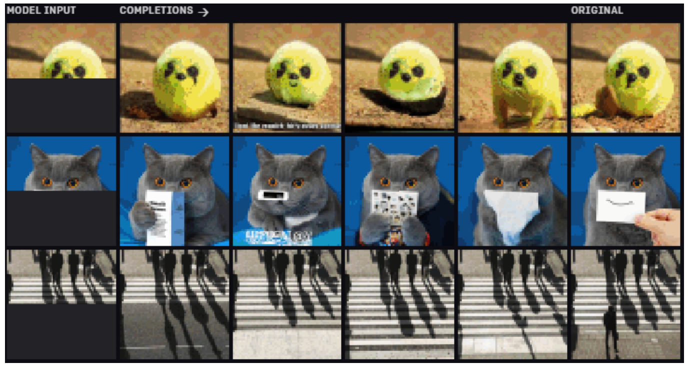
Jacobian matrix



Autoregressive models as Flows



PixelRNN

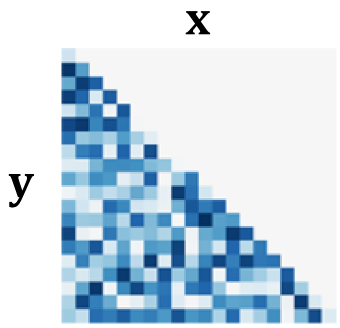


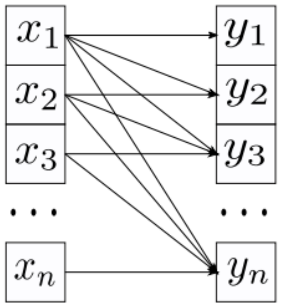
- Autoregressive models:

$$p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i|\mathbf{x}_{<i}) = p(x_1)p(x_2|x_1) \dots p(x_D|x_{D-1} \dots x_1)$$

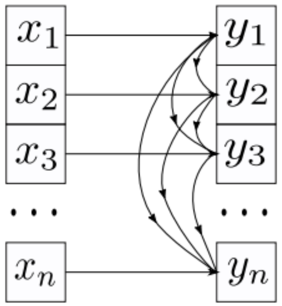
- Autoregressive models are a special case of normalizing flows.
  - Sampling must be computed sequentially. (slow)
  - Density can be computed in parallel. (fast)
- Inverse Autoregressive Flows
  - Sampling can be computed in parallel. (fast)
  - Density must be computed sequentially. (slow)
  - Ideal for variational inference.

Jacobian matrix





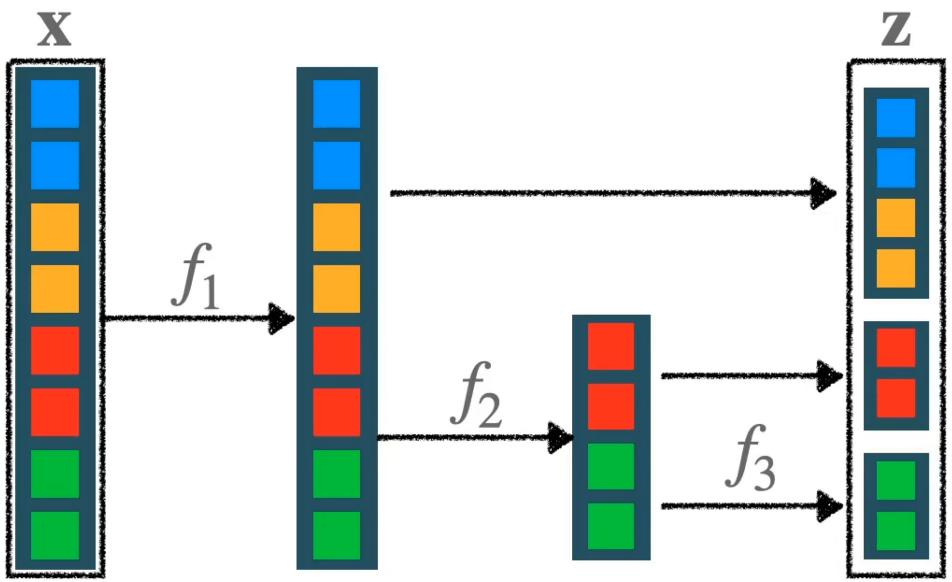
AF



IAF

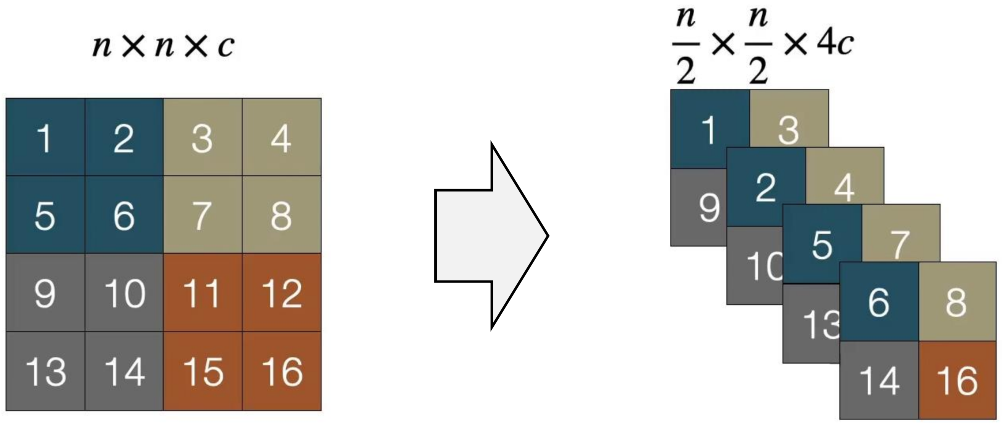
Multi-scale Flows

- Flows must preserve dimensionality.
- But, this is expensive in computation and memory.
- Just stop using subsets of dimensions. Practically, acts like dropping dimensions.



Multi-scale Flows

- For images, we only have three dimensions (R,G,B) to start with.
- How do we split the channel dimension?
- Squeeze spatial size, increase channel size.

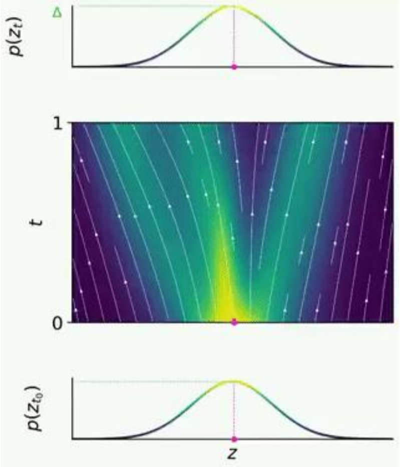


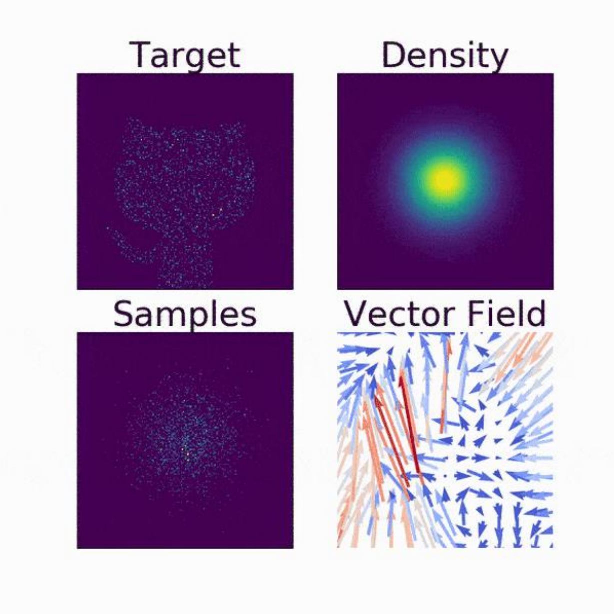
Continuous Flows

Grathwohl et al., FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. ICLR 2019

- Neural Ordinary Differential Equations
  - $\partial \mathbf{z}(t) = f_\theta(\mathbf{z}, t)\partial t$
- Instantaneous change of variables
  - $\frac{\partial}{\partial t} \log p(\mathbf{z}) = -\mathrm{tr}\left(\frac{\partial f(\mathbf{z},t)}{\partial \mathbf{z}}\right)$
  - $\log p(\mathbf{z}) = \log p(\mathbf{z}_0) - \int_0^T \mathrm{tr}\left(\frac{\partial f(\mathbf{z},t)}{\partial \mathbf{z}}\right) dt$
- Advantages of CNF
  - Inverting is easy. (solving reversed ODE)
  - Trace costs $\mathcal{O}(dim)$ for any matrix.
  - Free-form Jacobian
- Hutchinson trace estimator
  - $\mathrm{tr}(A) = \mathbb{E}_{v\sim\mathcal{N}(0,1)}[v^\top A v]$
  - $\int_0^T \mathrm{tr}\left(\frac{\partial f_\theta(z,t)}{\partial z}\right) dt = \mathbb{E}_{v\sim\mathcal{N}(0,1)}\left[\int_0^T v^\top \frac{\partial f(z,t)}{\partial z} v \, dt\right]$

VJP is cheap to compute.





Discussion

- Can deep generative models beat contrastive representation learning?
- Normalizing flows for self-supervised representation learning?
- What are the state of GANs in generative modelling?
- Deep generative models for anomaly detection?
- Self-organized semantic representation in deep generative models

References

- Normalizing Flows: an introduction and review of current methods
- https://www.youtube.com/watch?v=i7LjDvsLWCg&t=5s
- https://www.youtube.com/watch?v=u3vVyFVU_lI&t=17s
- https://www.youtube.com/watch?v=P4Ta-TZPVi0