CS 480/680

Lecture 23: July 24, 2019

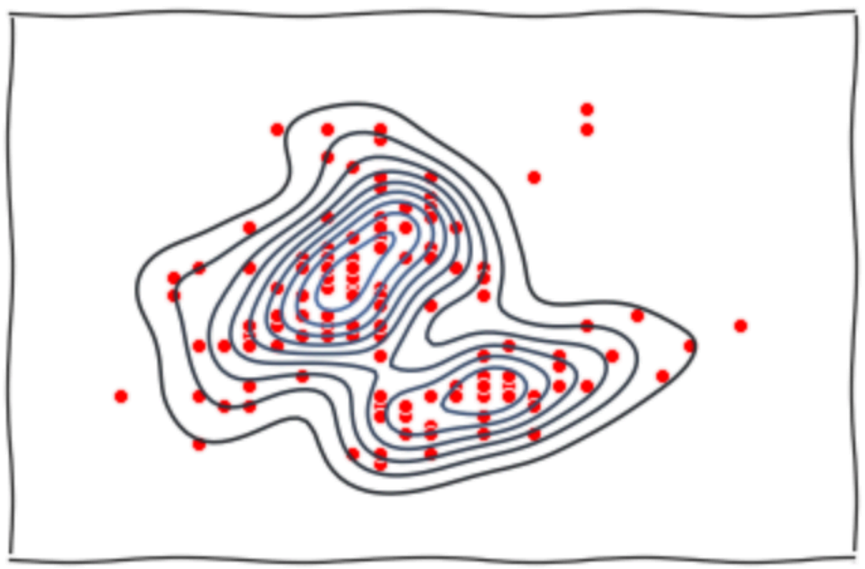
Normalizing Flows

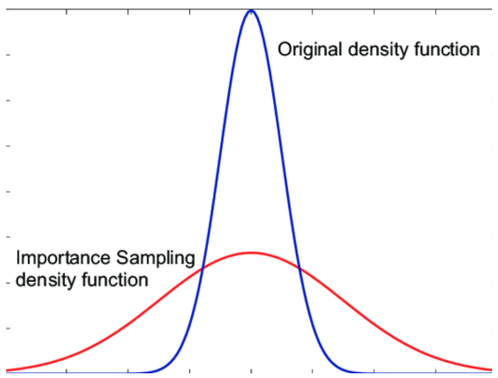
[GBC] Sec: 20.10.7

Complimentary Reading:

- Sum-of-Squares Polynomial Flows, ICML 2019
- Tutorial on Normalizing Flows, Eric Jang

density estimation





data $= \{x_1, x_2, \ldots, x_n\}$

estimate $q(x)$

importance sampling

bayesian inference

image & audio synthesis

many applications…

van den oord et. al. Pixel Recurrent Neural Networks, ICML 2016

Kingma et. al. Glow:Generative Flow with Invertible 1x1 Convolutions, NeurIPS 2018

van den oord et. al. WaveNet: A Generative Model for Raw Audio, SSW 2016

Jaini et. al. Online Bayesian Transfer Learning for Sequential Data Modeling, ICLR 2017

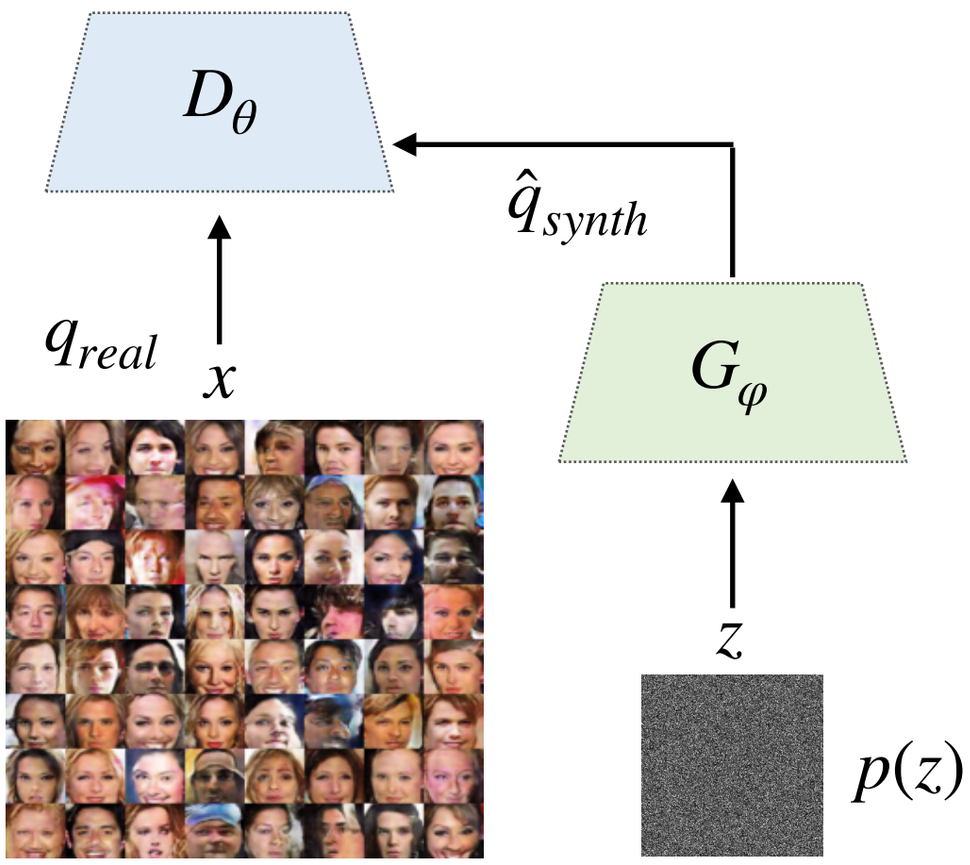
Moselhy, T., Marzouk, Y. et.al. Bayesian Inference with Optimal Maps, JCP 2012

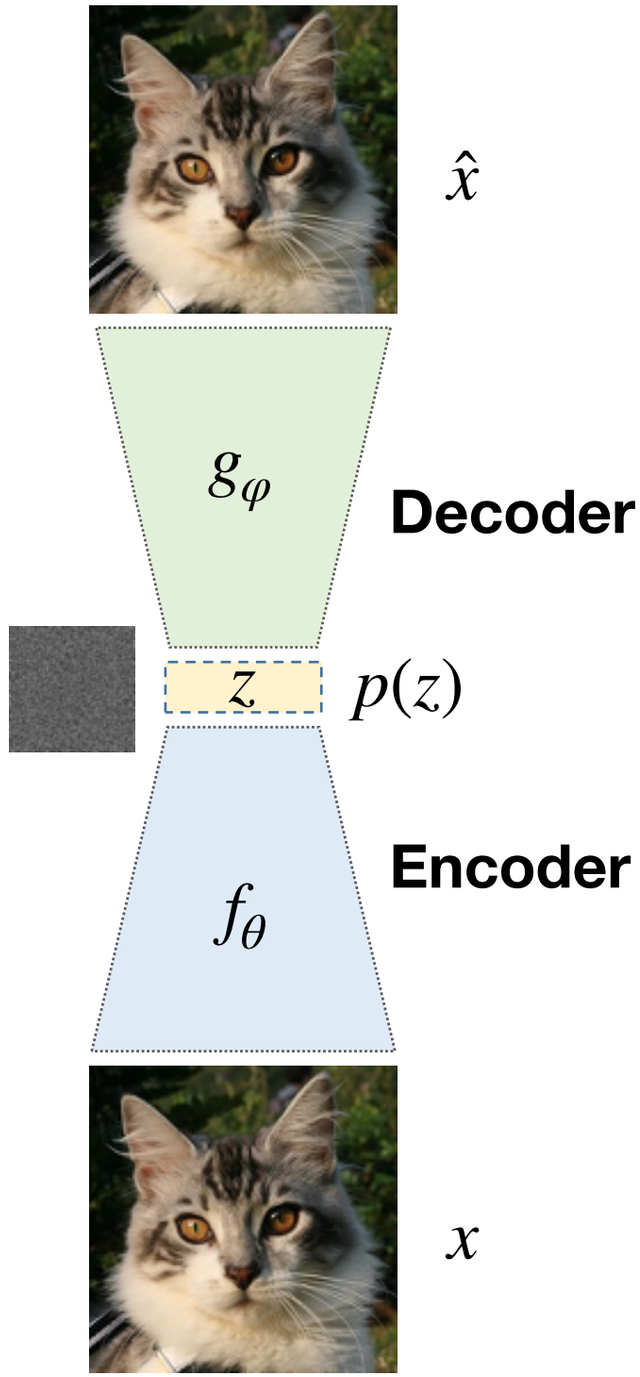
conditional generative models

courtesy: wimbledon

courtesy: 9GAG

GANs and VAE's

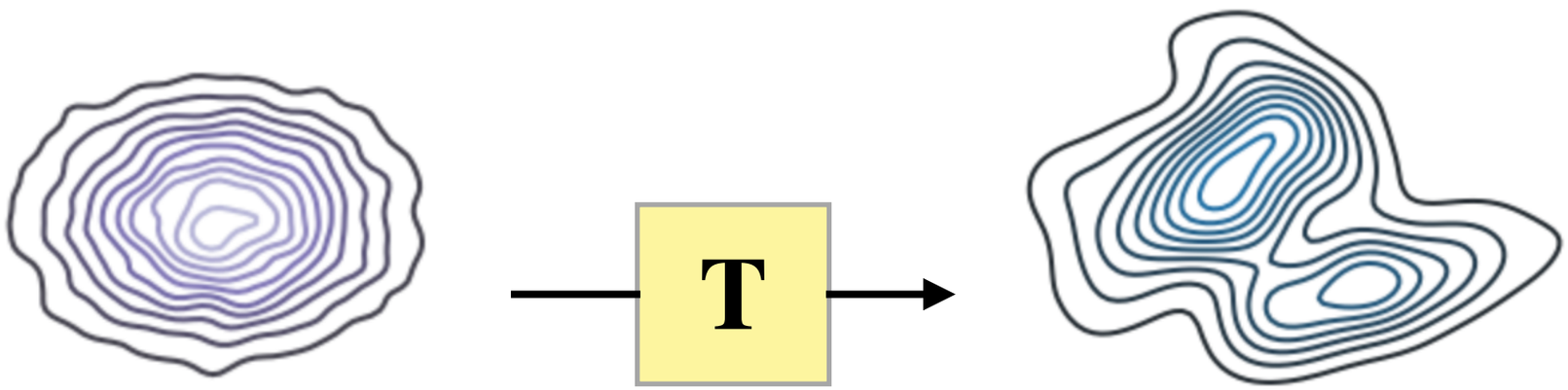




implicit representation of density functions

agenda

explicit representation of density functions



source distribution

$p(z)$

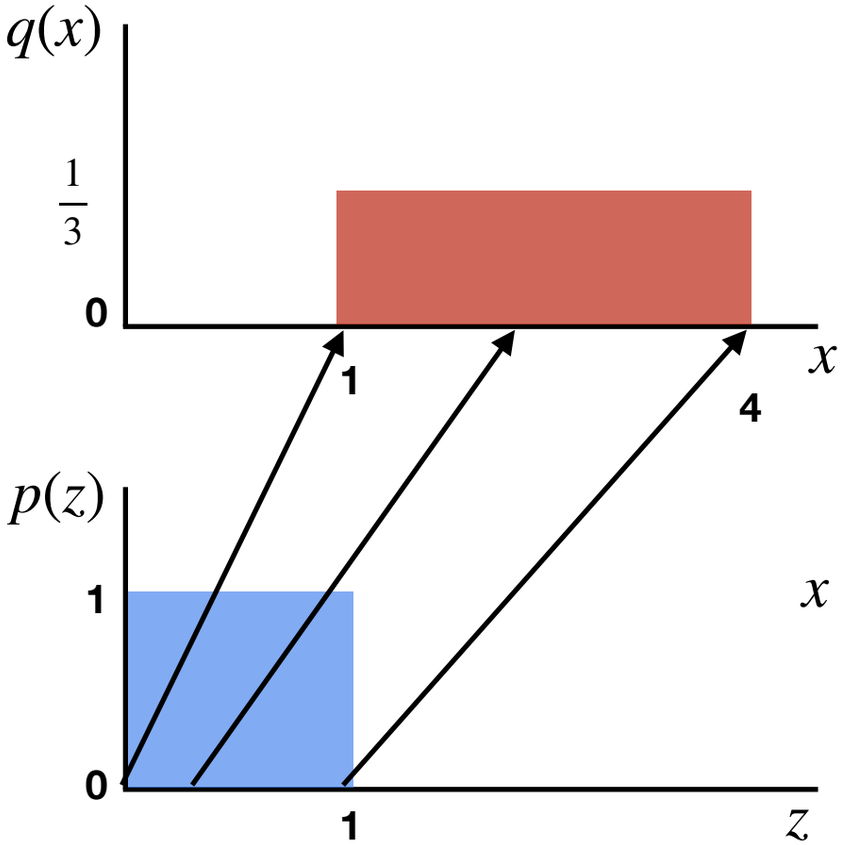
target distribution

$q(x)$

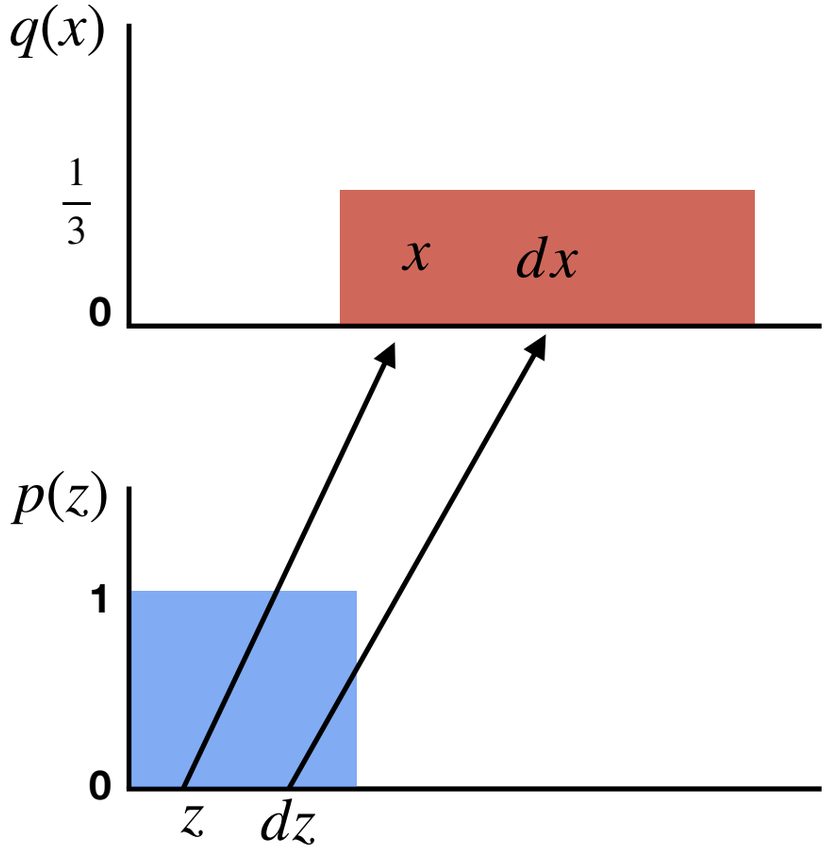
find *deterministic* maps from source to target density

conversation of probability mass

Transforming a uniform random variable into another uniform variable



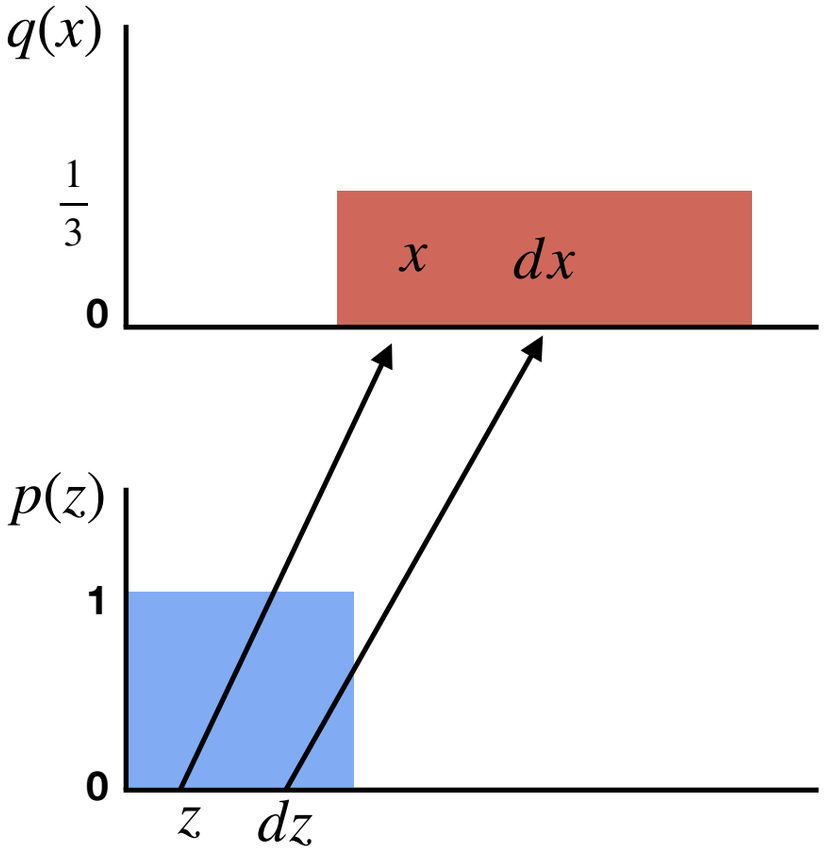
$$x := T(z) = 3z + 1$$



$$p(z)dz = q(x)dx$$

$$q(x) = p(z)\left|\frac{dz}{dx}\right|$$

change of variables



$$p(z)dz = q(x)dx$$

$$q(x) = p(z)\left|\frac{dz}{dx}\right|$$

univariate

Z : **source random variable and density** $p(z)$

X : **source random variable and density** $q(x)$

$$T : \mathsf{Z} \rightarrow \mathsf{X}$$

$$q(x) = p(z)\left|\frac{\partial T(z)}{\partial z}\right|^{-1}$$

multivariate

$$\mathbf{T} : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

$$q(\mathbf{x}) = p(\mathbf{z})\left|det\left(\nabla_{\mathbf{z}}\mathbf{T}(\mathbf{z})\right)\right|^{-1}$$

recipe for learning

Given: dataset $\mathscr{D} := \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n\} \sim q(\mathbf{x})$

learn the density $q(\mathbf{x})$

choose a simple source density $p(\mathbf{z})$

use maximum likelihood

$$\prod_{i=1}^{n} q(\mathbf{x}_i) = \prod_{i=1}^{n} p(\mathbf{z}_i) \left| det\left(\nabla_{\mathbf{z}} \mathbf{T}(\mathbf{z}_i) \right) \right|^{-1}$$

$$\hat{\mathbf{T}} := \arg\max_{\mathbf{T}} \prod_{i=1}^{n} p(\mathbf{z}_i) \left| det\left(\nabla_{\mathbf{z}} \mathbf{T}(\mathbf{z}_i) \right) \right|^{-1}$$

$$\hat{\mathbf{T}} := \arg\max_{\mathbf{T}} \sum_{i=1}^{n} \log p(\mathbf{z}_i) - \log \left| det\left(\nabla_{\mathbf{z}} \mathbf{T}(\mathbf{z}_i) \right) \right|$$

But.....

what are $\mathbf{z}_i$ **?**

$$\mathbf{z}_i = \mathbf{T}^{-1}(\mathbf{x}_i)$$

so we need the inverse $\mathbf{T}^{-1}$

computing determinant is computationally expensive

→ triangular maps

computation of inverse and Jacobian must be *cheap*

increasing triangular maps

$$\mathbf{T} : \mathbb{R}^d \to \mathbb{R}^d$$

$$
\begin{aligned}
x_1 &= T_1(z_1) \\
x_2 &= T_2(z_1,\, z_2) \\
x_3 &= T_3(z_1,\, z_2,\, z_3) \\
&\vdots \\
x_d &= T_d(z_1,\, z_2,\, z_3,\, \ldots,\, z_d)
\end{aligned}
$$

$$
\nabla_{\mathbf{z}}\mathbf{T} = \begin{bmatrix}
\frac{\partial T_1}{\partial z_1} & 0 & \cdots & 0 \\
\frac{\partial T_2}{\partial z_1} & \frac{\partial T_2}{\partial z_2} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial T_d}{\partial z_1} & \frac{\partial T_d}{\partial z_2} & \cdots & \frac{\partial T_d}{\partial z_d}
\end{bmatrix}
$$

triangular : T_j **is a function of** $z_1, z_2, \ldots, z_j$

increasing : T_j **is increasing w.r.t** z_j

$$\frac{\partial T_j}{\partial z_j} > 0$$

triangular maps

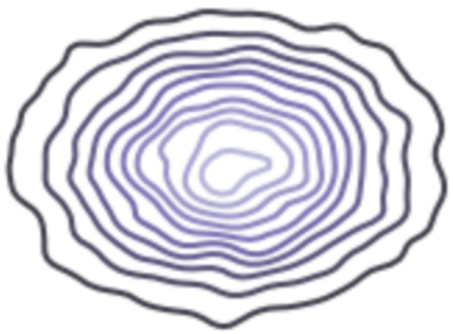
inverse and Jacobian are *easy* to compute

Theorem (paraphrase) : there always exists a unique* increasing triangular map that transforms a source density to a target density

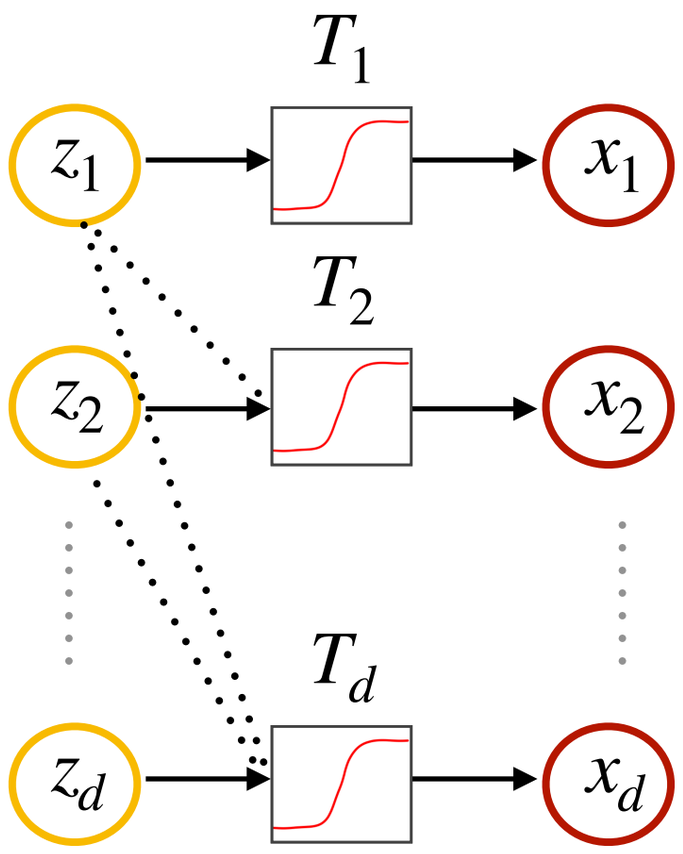
Bogachev, V. et. al. Triangular Transformation of Measures, Sbornik: Mathematics, 2005

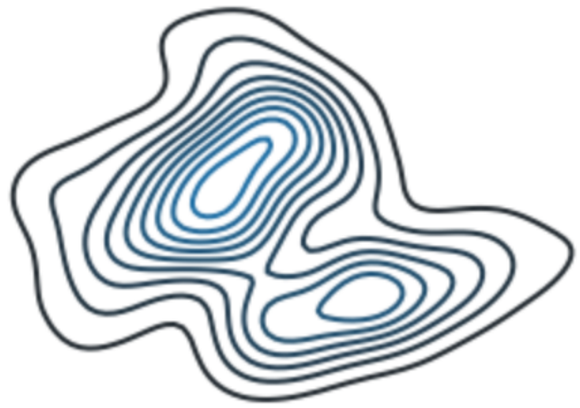
**** for a fixed ordering***

framework



$z \sim p(z)$





$x \sim q(x)$

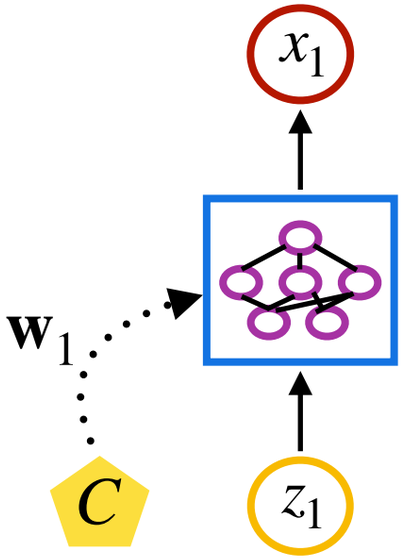
learn T by maximizing likelihood

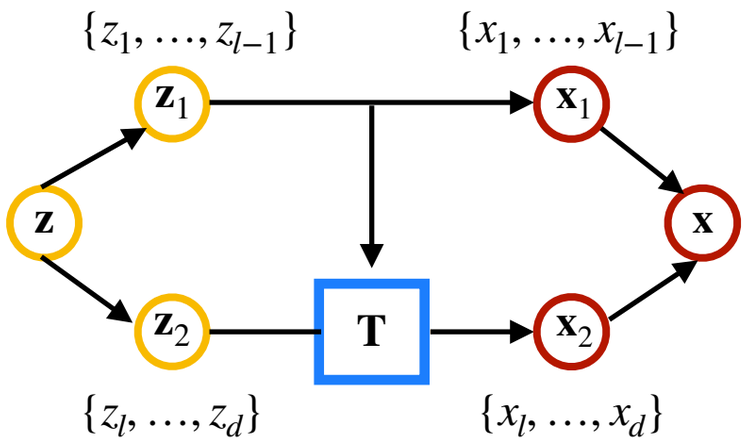
$$\min_{T} \sum_{i=1}^{n} \left[-\log p\left(\mathbf{T}^{-1}(\mathbf{x}_i)\right) + \sum_{j} \log \partial_j T_j\left(\mathbf{T}^{-1}(\mathbf{x}_i)\right) \right]$$

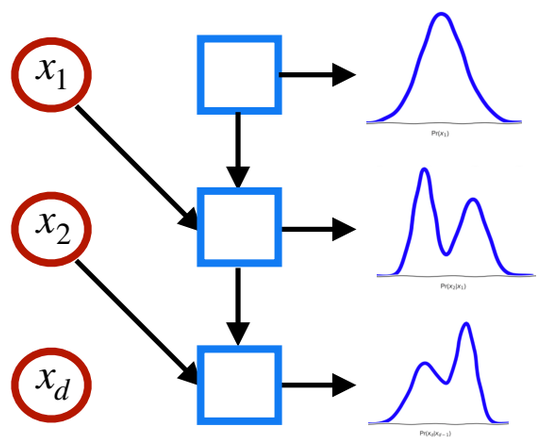
Marzouk, Y. et.al. Sampling via Measure Transport: An Introduction, Springer, 2016

flow models as triangular maps

study commonalities & differences of flow based methods

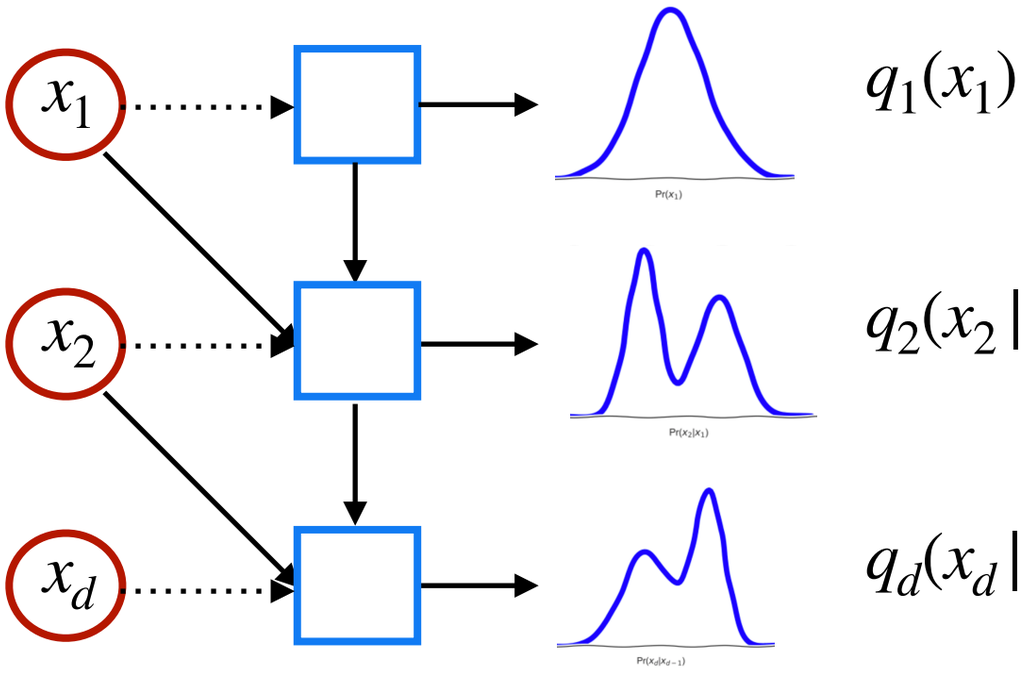


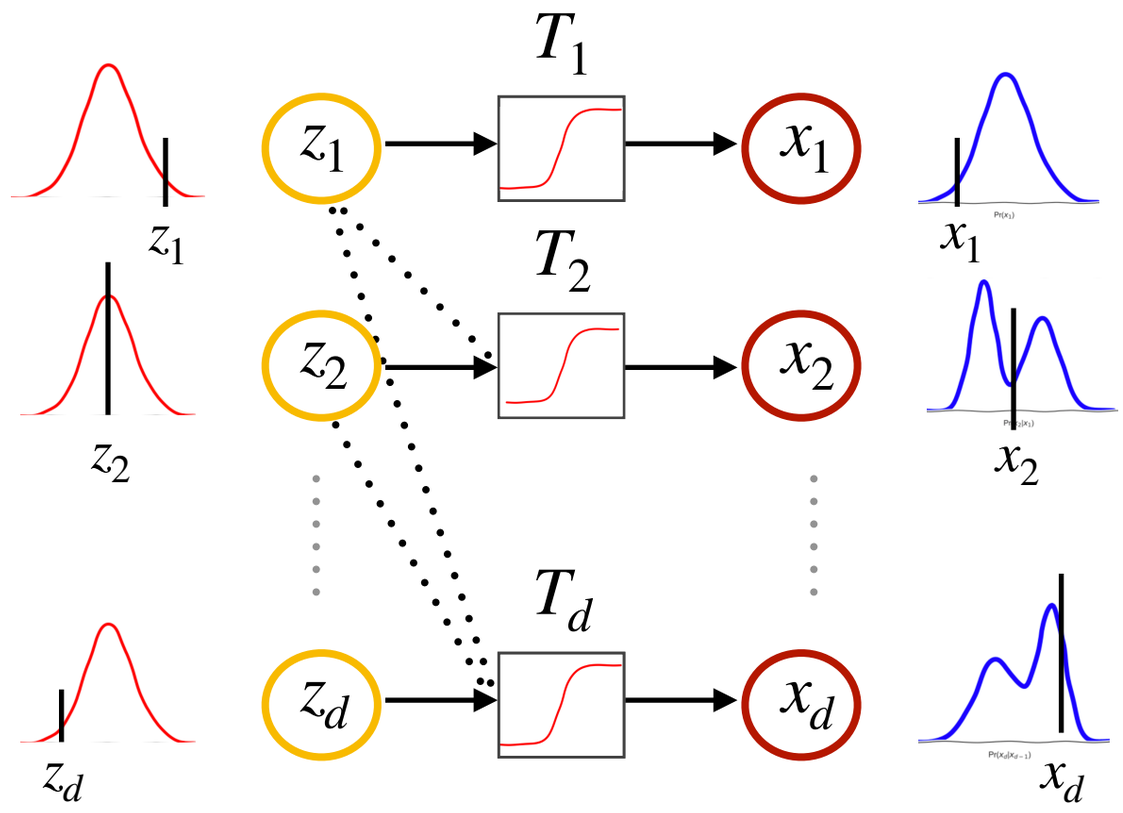




autoregressive models

$$q(x) = q_1(x_1) \cdot q_2(x_2 \,|\, x_1) \cdot \ldots \cdot q_d(x_d \,|\, x_{<d})$$





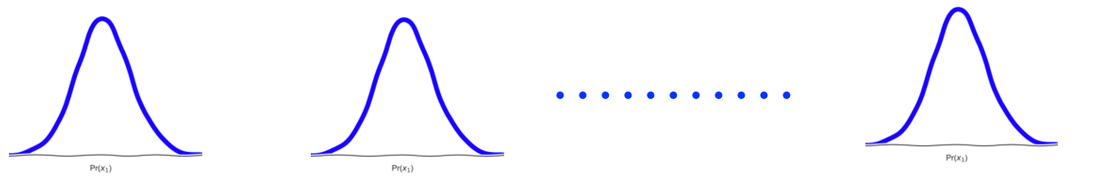
choosing a conditional implicitly fixes a family of triangular maps

$$x_j = T_j\left(z_j;\ \theta_j(z_{<j})\right)$$

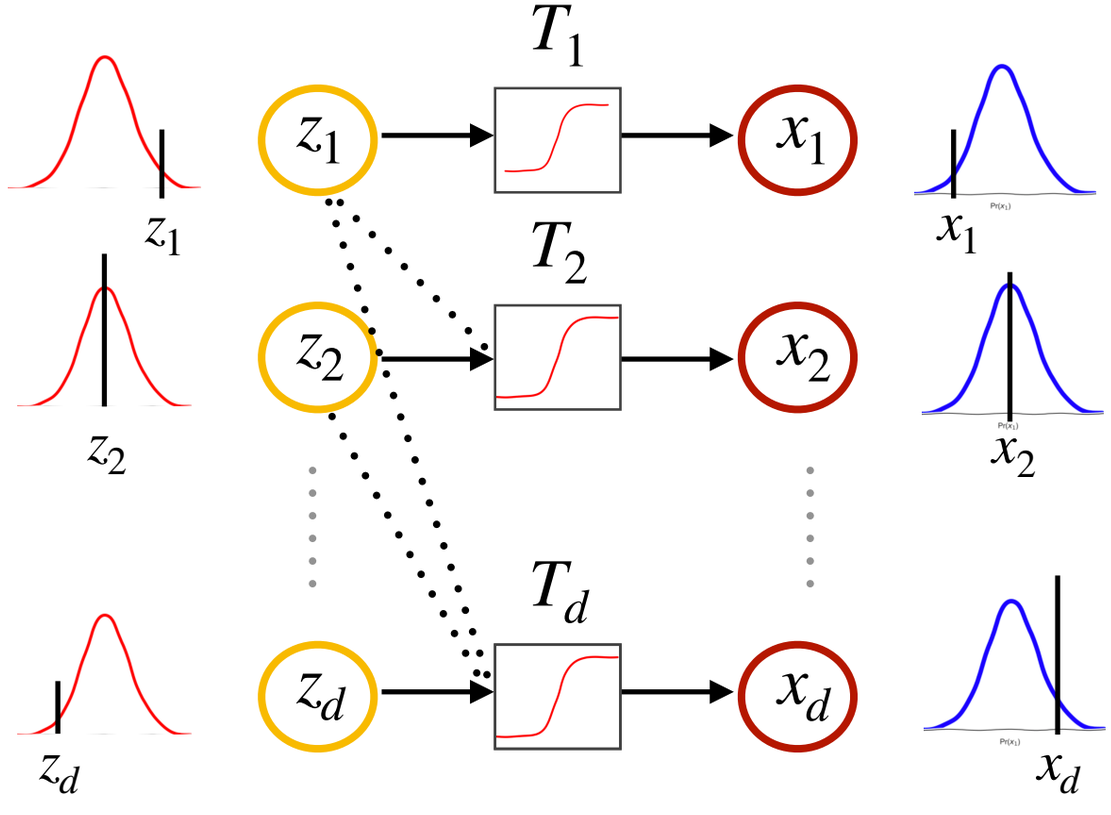
Larochelle, H and Murray, I. The Neural Autoregressive Distribution Estimator, AISTATS, 2011
Uria, et.al. Neural Autoregressive Distribution Estimation, JMLR, 2016

AR with Gaussian conditionals

$$q(x) = q_1(x_1) \cdot q_2(x_2 \,|\, x_1) \cdot \ldots \cdot q_d(x_d \,|\, x_{<d})$$



$\mathcal{N}(\mu_1, \sigma_1^2)$ $\mathcal{N}(\mu_2, \sigma_2^2)$ $\mathcal{N}(\mu_d, \sigma_d^2)$



$$x_1 = \sigma_1 \cdot z_1 + \mu_1 =: T_1(z_1)$$

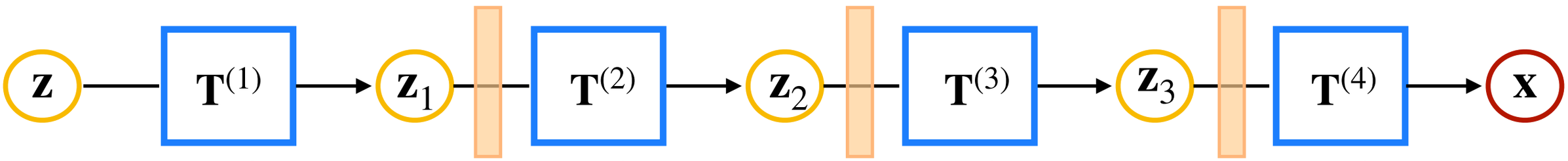
$$x_2 = \sigma_2(z_1) \cdot z_2 + \mu_2(z_1) := T_2(z_2 \,; z_1)$$

$$x_d = \sigma_d(z_{<d}) \cdot z_d + \mu_d(z_{<d}) := T_d(z_d \,; z_{<d})$$

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

masked autoregressive flows (MAFs)

deep autoregressive flows with Gaussian conditionals*



$$q(x) = p(z) \cdot \left| \nabla \mathbf{T}^{(1)} \right|^{-1}_{\cdot} \left| \nabla \mathbf{T}^{(2)} \right|^{-1}_{\cdot} \left| \nabla \mathbf{T}^{(3)} \right|^{-1}_{\cdot} \left| \nabla \mathbf{T}^{(4)} \right|^{-1}$$

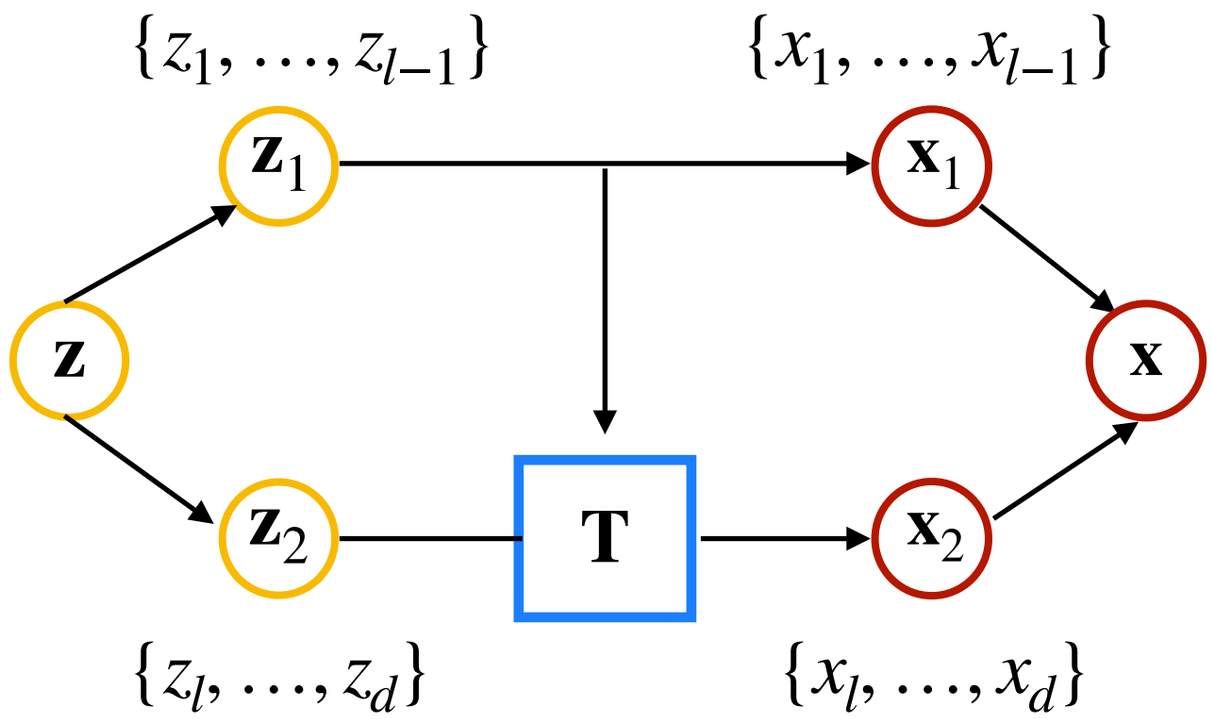
$$x_j = z_j \cdot \exp\left(\alpha_j(z_{<j})\right) + \mu_j(z_{<j}) =: T_j(z_j\ ; z_{<j})$$

triangular maps are fundamental blocks for complex models

Papamakarios, et.al. Masked Autoregressive Flow for Density Estimation, NeurIPS, 2017

**** can consider mixture of gaussians***

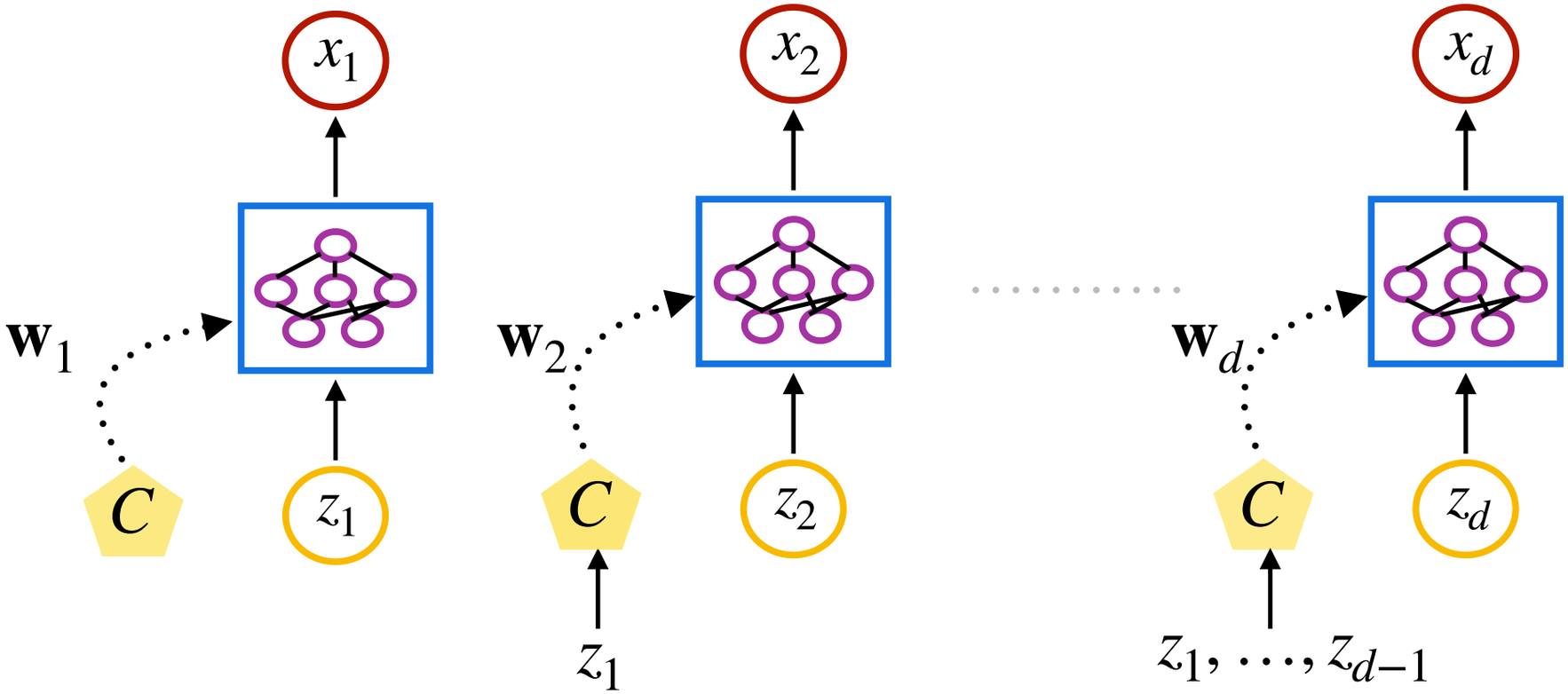
real-NVP



$$T_j(z_j \; ; \; z_{<l}) = \exp\Big(\alpha_j(z_{<l}) \cdot \mathbf{1}_{j \notin [l-1]}\Big) \cdot z_j + \mu_j(z_{<l}) \cdot \mathbf{1}_{j \notin [l-1]}$$

Dinh, et.al. Density estimation using Real NVP, ICLR, 2017

neural autoregressive flows (NAFs)



$$x_j = \textbf{DNN}\left(z_j \; ; \; \mathbf{w}_j(z_{<j})\right) =: T_j(z_j \; ; z_{<j})$$

Strictly positive weights & strictly monotonic activation function ensure that the map is increasing

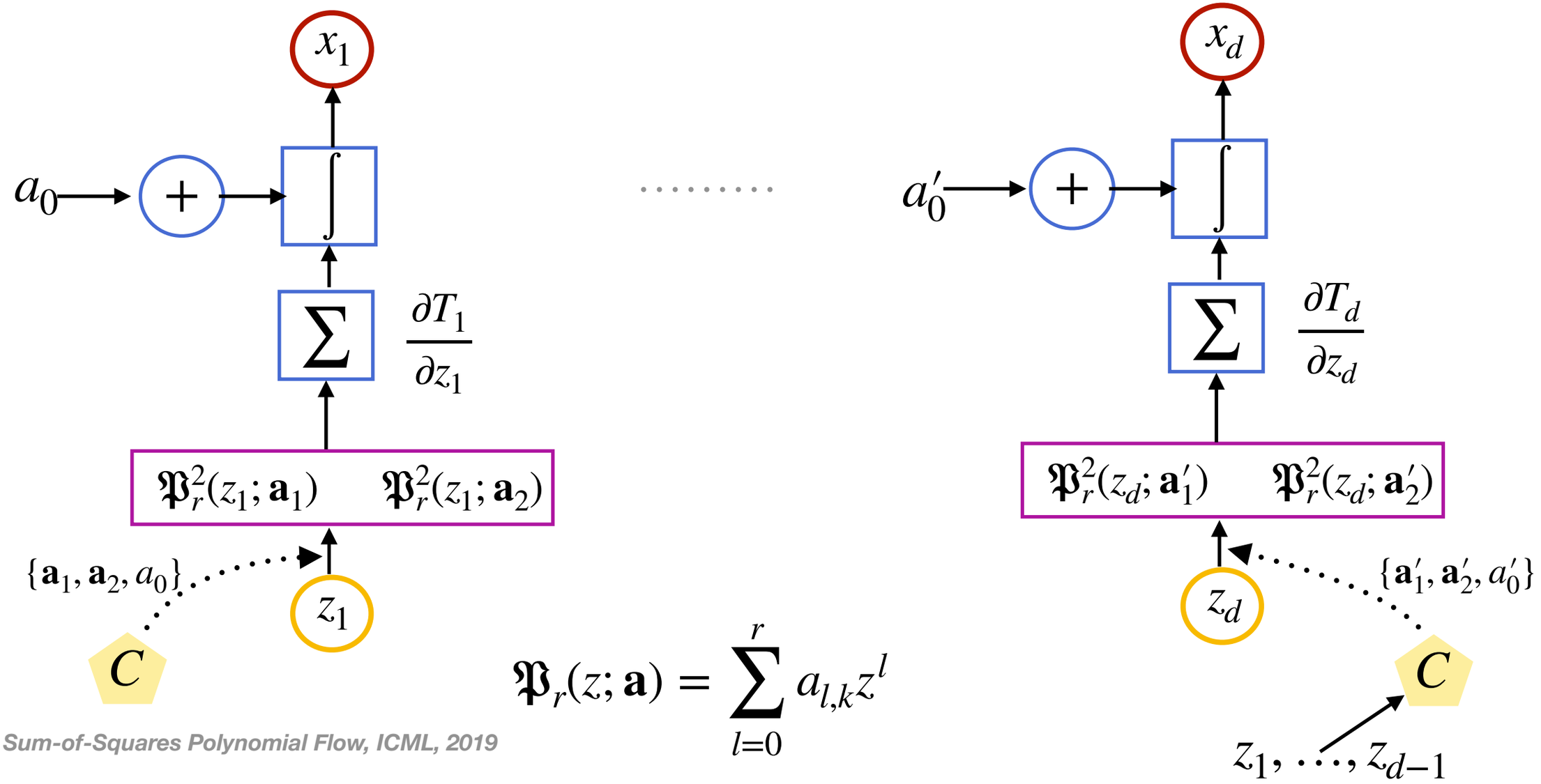
Universal

Huang, et.al. Neural Autoregressive Flows, ICML, 2018

sum-of-squares polynomial flows

goal : learn a universal increasing triangular function

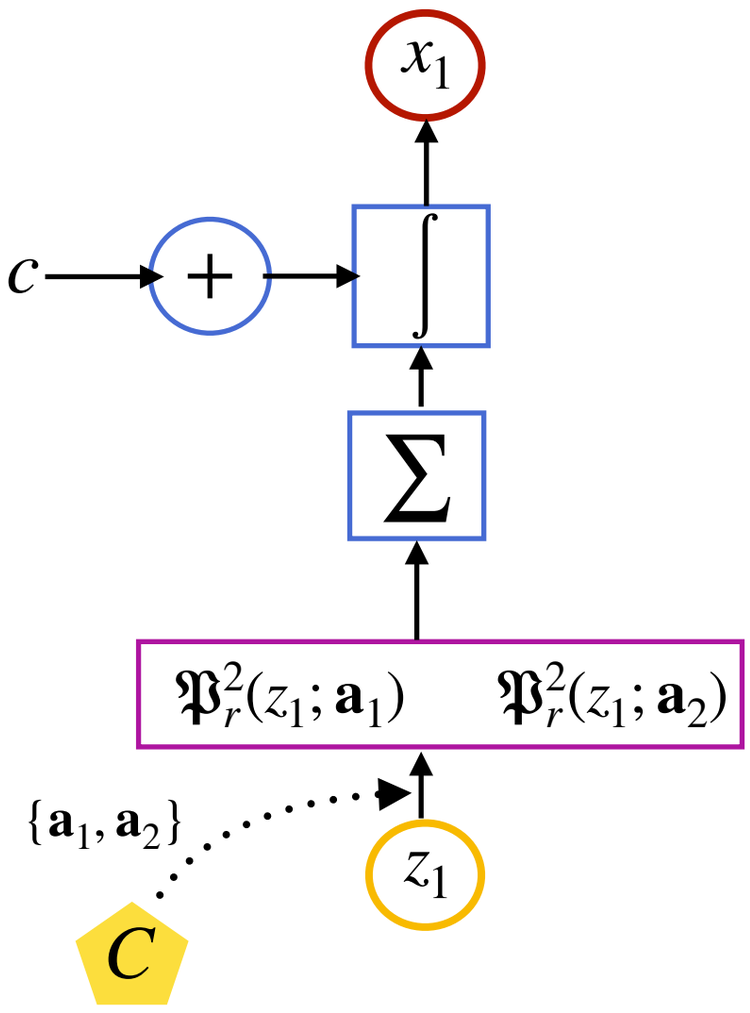
increasing functions $\approx$ primitive of non-negative polynomials $=$ primitive of sum-of-squares of polynomials



$$\mathfrak{P}_r(z; \mathbf{a}) = \sum_{l=0}^{r} a_{l,k} z^l$$

Jaini, et.al. Sum-of-Squares Polynomial Flow, ICML, 2019

SOS flows



Generalization of other flows

(e.g. IAF, MAF, Real-NVP etc)

Interpretable

coefficients control higher order moments

Easier to train

no constraints on coefficients

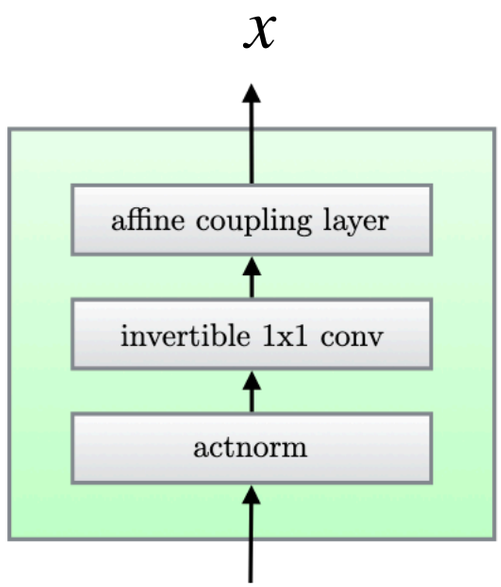
Universal

approximate any density given enough capacity

Application to stochastic simulation

Jaini, et.al. Sum-of-Squares Polynomial Flow, ICML, 2019

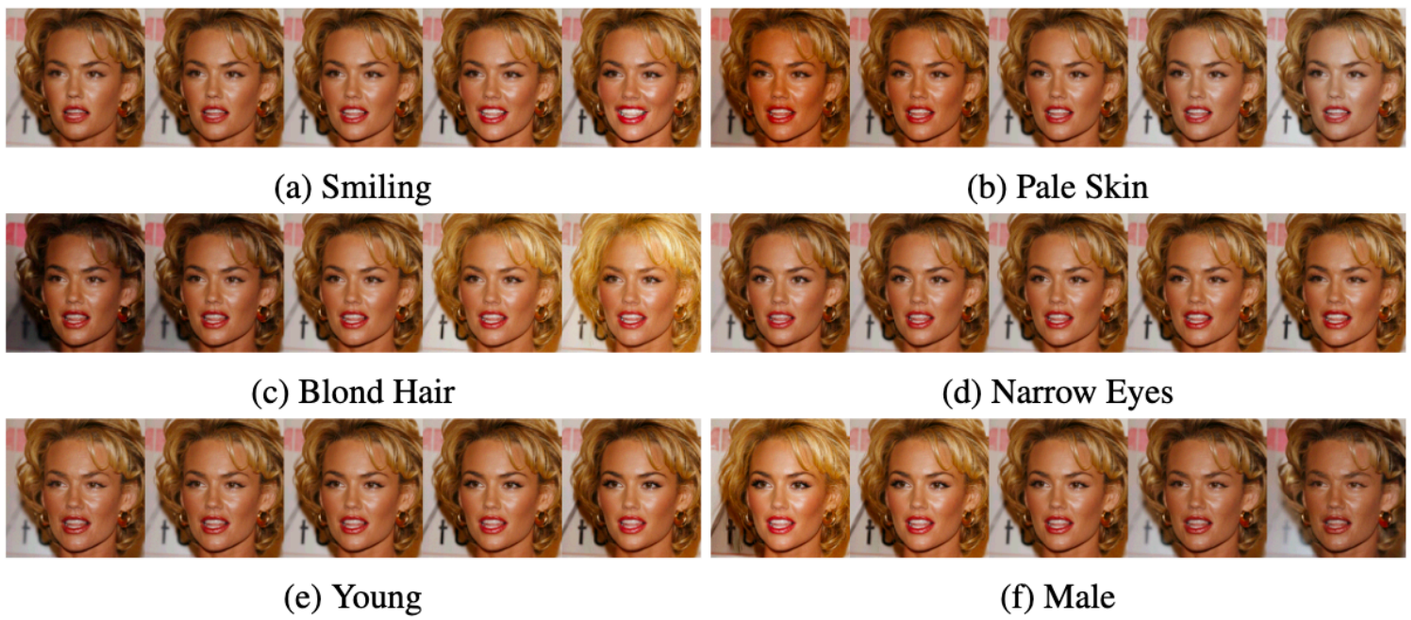
Glow : invertible 1x1 convolutions



random rotation matrix

data dependent

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert \det(\mathbf{W}) \rvert$ or $h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$



(a) Smiling (b) Pale Skin

(c) Blond Hair (d) Narrow Eyes

(e) Young (f) Male

Kingma, D.P. and Dhariwal, P, et.al. Glow: Generative Flow with Invertible 1x1 Convolutions, NeurIPS, 2018