Lecture 20: Normalizing Flows
CS480/680 Intro to Machine Learning

2023-3-23

Pascal Poupart
David R. Cheriton School of Computer Science

UNIVERSITY OF
WATERLOO

Normalizing Flows

- Neural networks to estimate probability density functions
  - Can estimate the likelihood of a data point
  - Generative models (i.e., generate data point from random embedding)
  - Invertible models (i.e., generate embedding corresponding to a data point)

Preview: GLOW (Generative fLOW)

Kingma, Dhariwal et al. (2018)

Figure 4: Random samples from the model, with temperature 0.7

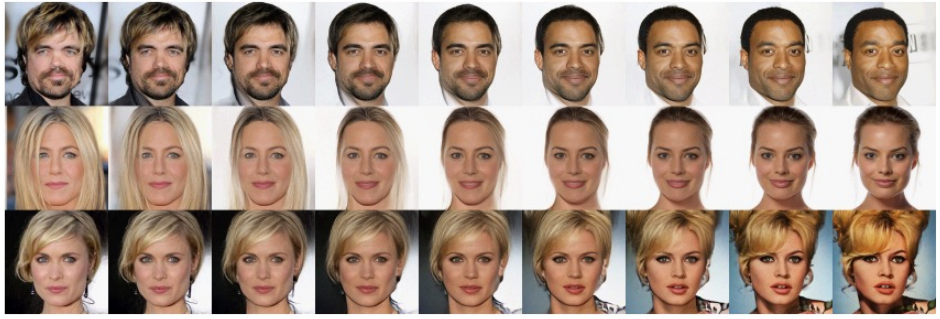
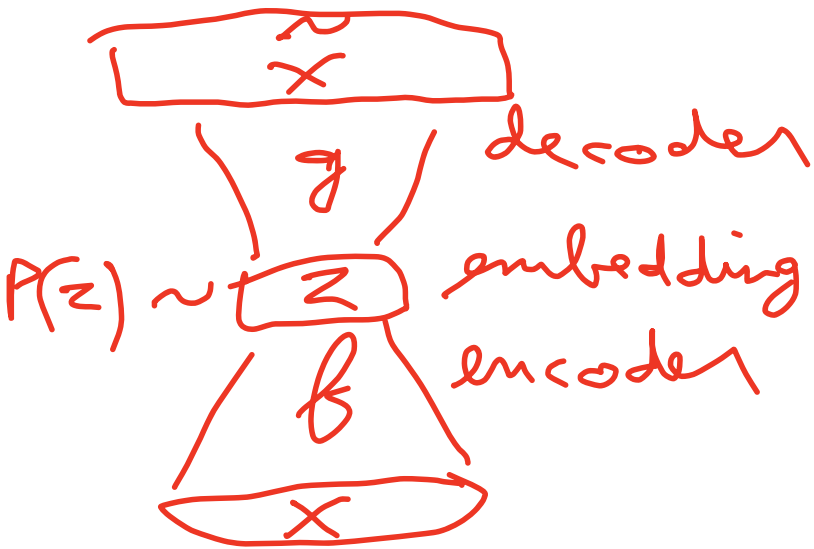


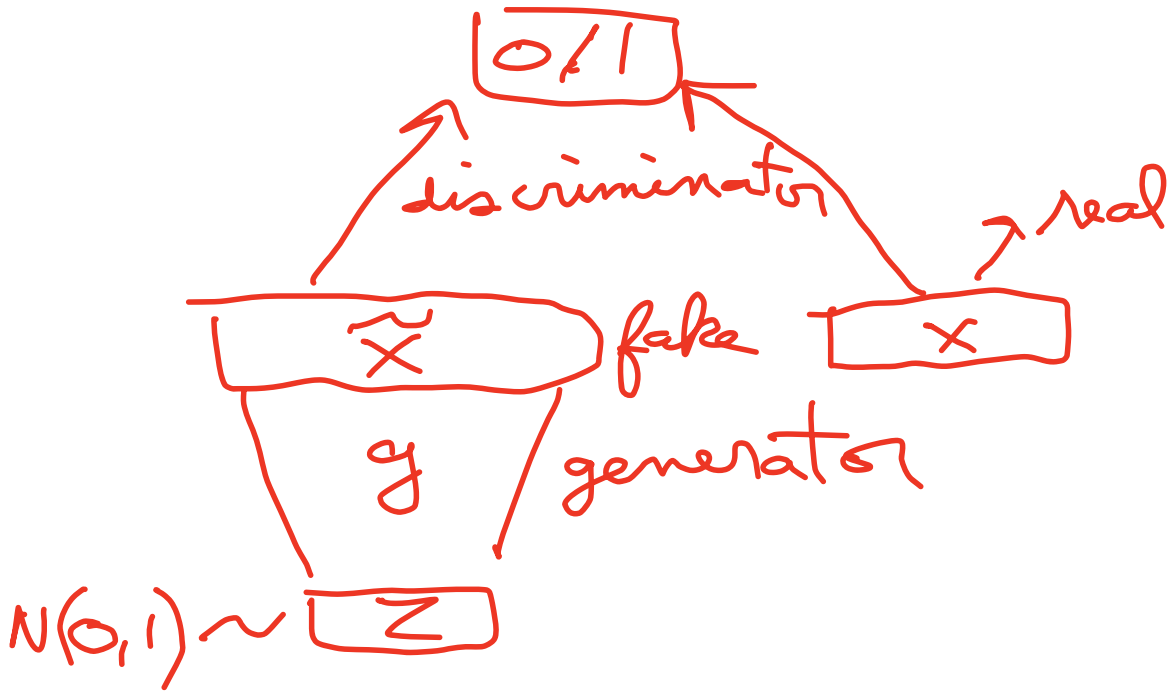
Figure 5: Linear interpolation in latent space between real images

Recap

Variational Autoencoder

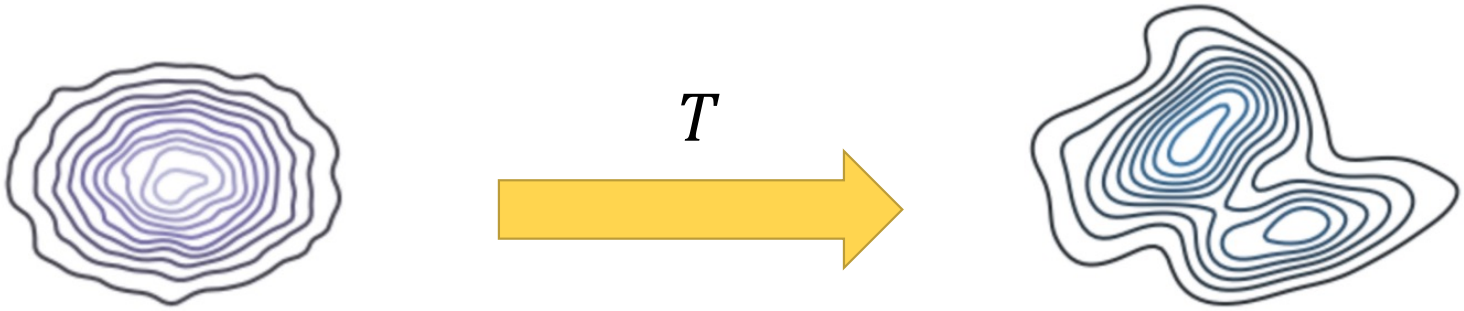


Generative Adversarial Network



Explicit density function

From Priyank Jaini



Source distribution
$\boldsymbol{z} \sim p(\boldsymbol{z})$

Target distribution
$\boldsymbol{x} \sim q(\boldsymbol{x})$

- Goal: learn a deterministic function T that transforms p into q

Conservation of Mass

- When we transform random variable Z into random variable $X = T(Z)$, the probability mass must be conserved

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

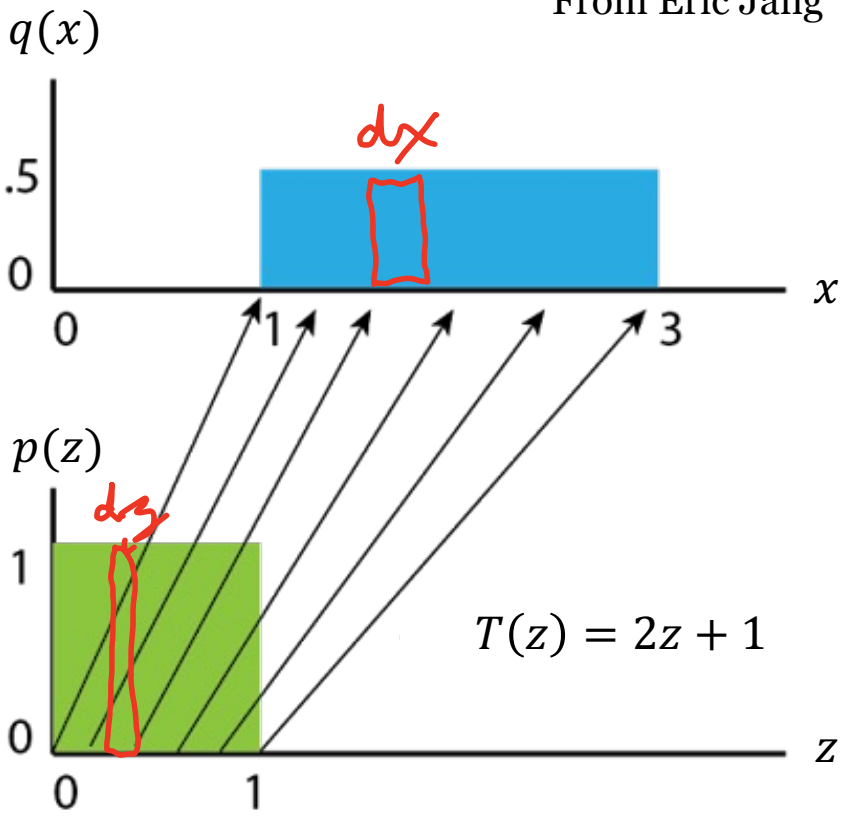
- Hence

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

$$= p(z)\left|\frac{dx}{dz}\right|^{-1}$$

$$= p(z)\left|\frac{dT(z)}{dz}\right|^{-1}$$

From Eric Jang



Change of Variable

- Multivariable distributions $p(\boldsymbol{X})$ and $q(\boldsymbol{Z})$
  - $\boldsymbol{X}, \boldsymbol{Z}$: vectors of random variables
  - $T: \boldsymbol{Z} \rightarrow \boldsymbol{X}$ where $T_1(\boldsymbol{z}) = x_1, T_2(\boldsymbol{z}) = x_2, T_3(\boldsymbol{z}) = x_3$, etc.
- Change of variable formula:

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

where T is bijective and increasing

Jacobian

determinant

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

Learning

- Data
  - Let $\boldsymbol{Z} \sim \mathrm{N}(\mathbf{0}, \mathbf{I})$ be random vectors
  - Let $\boldsymbol{X}$ be data (i.e., images in a dataset)
- Find $T: Z \rightarrow X$ by maximum likelihood

$$\begin{aligned}\operatorname{argmax}_T \prod_n P\left(\boldsymbol{z}^{(n)}\right) &= argmax_T \sum_n \log P\left(\boldsymbol{z}^{(n)}\right) \\ &= argmax_T \sum_n \log \left[P\left(\boldsymbol{x}^{(n)}\right)\left|\det\left(\nabla_{\boldsymbol{x}} T^{-1}(\boldsymbol{x})\right)\right|^{-1}\right] \\ &= argmax_T \sum_n \log P\left(\boldsymbol{x}^{(n)}\right) - \log \left|\det\left(\nabla_{\boldsymbol{x}} T^{-1}(\boldsymbol{x})\right)\right|\end{aligned}$$

- NB: need T^{-1} and efficient way to compute $\det\left(\nabla_{\boldsymbol{x}} T^{-1}(\boldsymbol{x})\right)$

Triangular Maps

- Let T be such that $T_i(\boldsymbol{z}_{\leq i}) = x_i$
  - $x_1 = T_1(z_1)$
  - $x_2 = T_2(z_1, z_2)$
  - ...
  - $x_m = T_m(z_1, z_2, \ldots, z_m)$

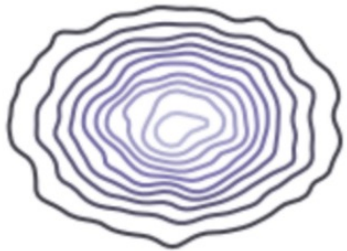
- $\det(\nabla T^{-1})$ is easy to compute
  - T lower triangular → T^{-1} lower triangular
  - $\det(lower\ triangular\ M) = \prod_i M_{ii}$

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

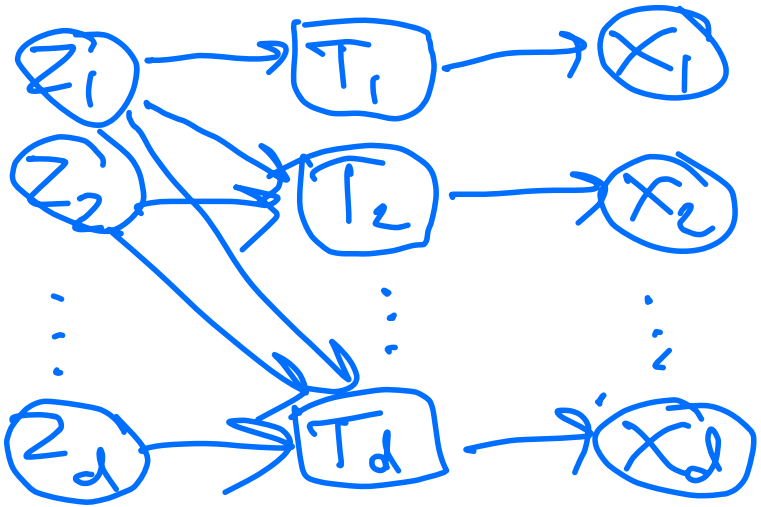
- Theorem: There always exists a **unique** increasing triangular map that transforms a source density into a target density

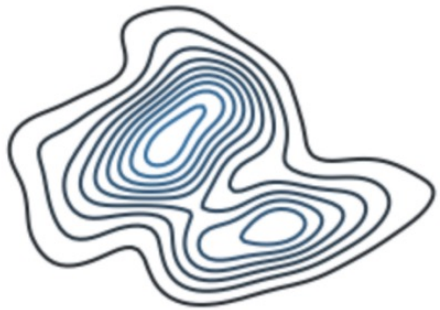
$$\frac{\partial T_i}{\partial z_i} > 0$$

Framework



$z \sim p(z)$





$x \sim q(x)$

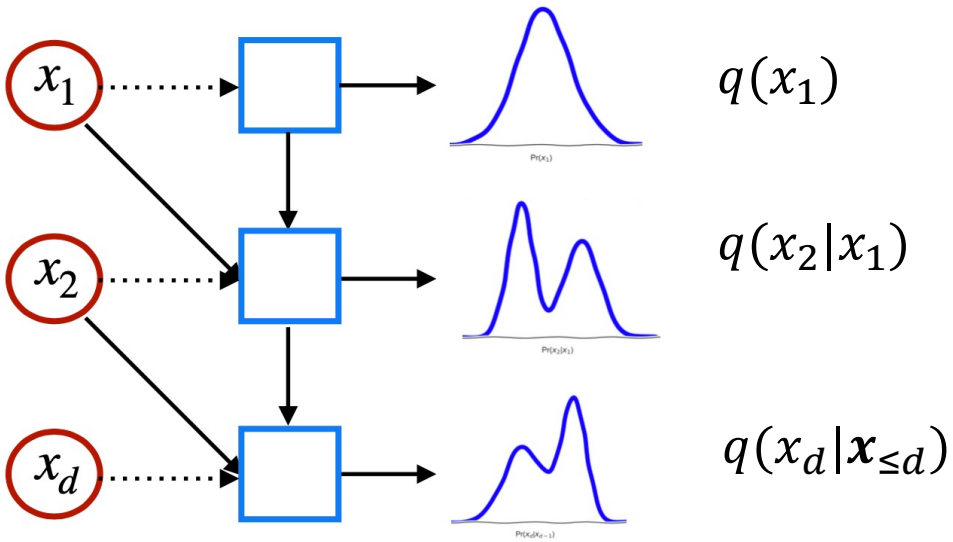
Find T by maximizing log likelihood

Set $P(\boldsymbol{z}) = N(0, I)$

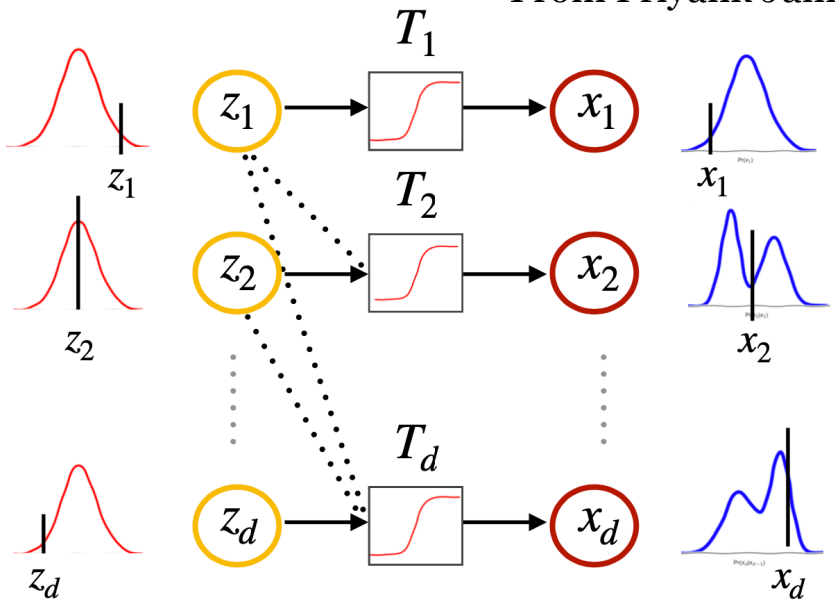
$$\underset{T}{\operatorname{argmax}} \sum_n \log q(\boldsymbol{x}^{(n)}) = \underset{T}{\operatorname{argmax}} \sum_n \log p\left(T^{-1}\left(\boldsymbol{x}^{(n)}\right)\right) - \sum_i \log \frac{\partial T_i^{-1}\left(\boldsymbol{x}^{(n)}\right)}{\partial x_i}$$

Autoregressive Models

- Recall the chain rule: $q(\boldsymbol{x}) = q(x_1)q(x_2|x_1) \dots q(x_d|\boldsymbol{x}_{<d})$



From Priyank Jaini



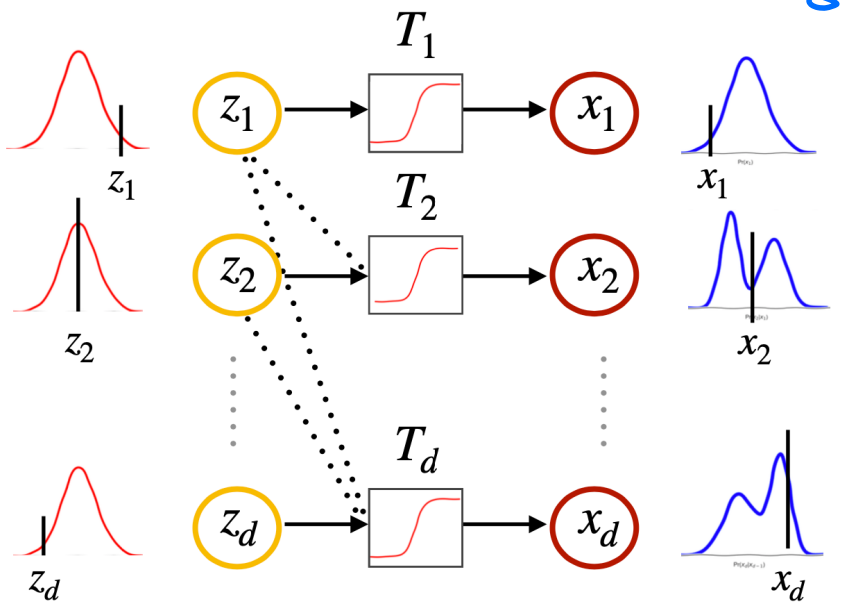
- Each mapping $x_i = T_i(\boldsymbol{z}_{\leq i})$ implicitly determines a corresponding $q(x_i|\boldsymbol{x}_{<i})$

Autoregressive Models with Gaussian Conditionals

- Recall the reparameterization trick

$N(0,1) \sim z \rightarrow \sigma, \mu \rightarrow x \sim N(\mu, \sigma)$

From Priyank Jaini



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

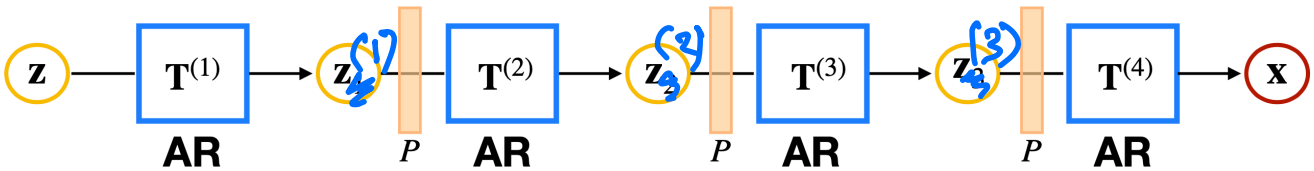
$$x_2 = T_2(z_1, z_2) = \sigma_2(z_1) z_2 + \mu_2(z_1)$$

$$x_d = T_d(z_1, \ldots, z_d) = \sigma_d(\mathbf{z}_{<d}) z_d + \mu_d(\mathbf{z}_{<d})$$

Masked Autoregressive Flows (MAFs)

Papamakarios et al., 2017

- Deep autoregressive flows with Gaussian conditionals

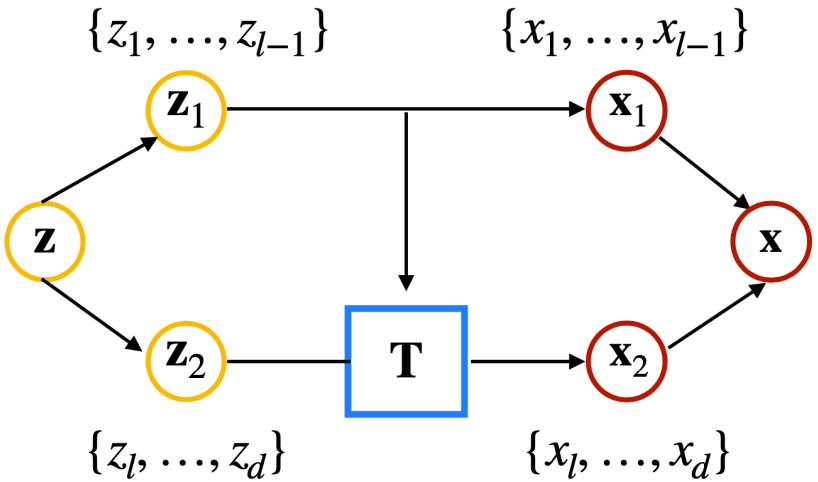


$$q(\boldsymbol{x}) = p(\boldsymbol{z})\left|\det(\nabla T^{(1)})\right|^{-1}\left|\det(\nabla T^{(2)})\right|^{-1}\left|\det(\nabla T^{(3)})\right|^{-1}\left|\det(\nabla T^{(4)})\right|^{-1}$$

$$x_i = T_i(\boldsymbol{z}_{\leq i}) = \exp\left(\alpha_i(\boldsymbol{z}_{<i})\right) z_i + \mu_i(\boldsymbol{z}_{<i})$$

Real NVP

Dinh et al., 2017



From Priyank Jaini

$$x_i = T_i(z_i, \mathbf{z}_{<i}) = \exp(\alpha_i(\mathbf{z}_{<i})\delta(i \geq l))\, z_i + \mu_i(\mathbf{z}_{<i})\delta(i \geq l)$$

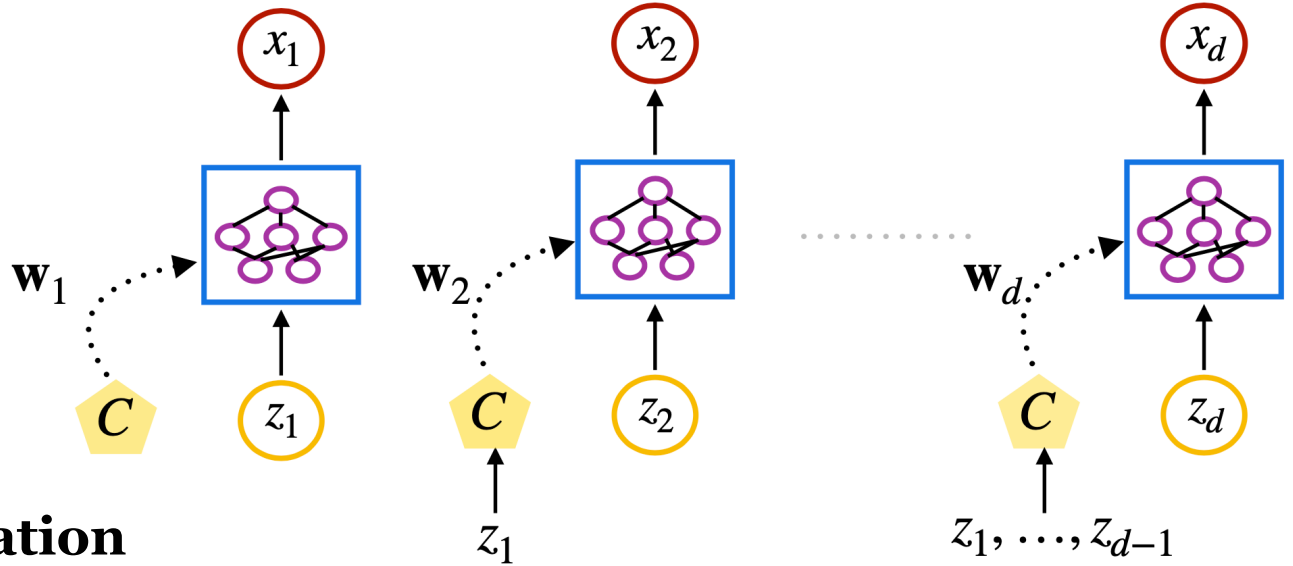
↳ $\delta(x) = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{otherwise} \end{cases}$

Neural Autoregressive Flows (NAFs)

- Strictly positive weights
- Strictly monotonic activation functions

- Hence: increasing map

- **Universal approximation**



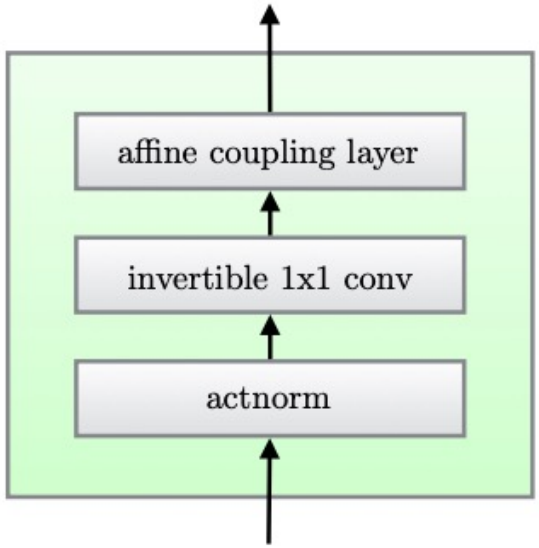
From Priyank Jaini

$$x_i = T_i(z_i, \mathbf{z}_{<i}) = DNN(z_i; w_i(z_{<i}))$$

↳ Deep neural net

GLOW (Generative fLOW)

Kingma, Dhariwal et al. (2018)



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

GLOW (Generative fLOW)

Kingma, Dhariwal et al. (2018)

Figure 4: Random samples from the model, with temperature 0.7

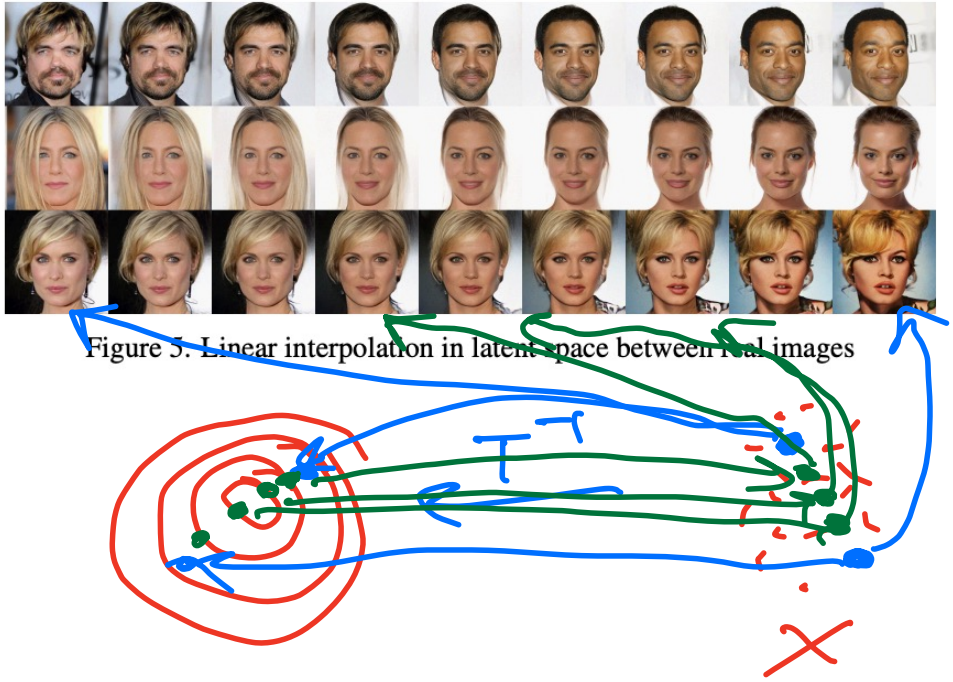


Figure 5. Linear interpolation in latent space between real images