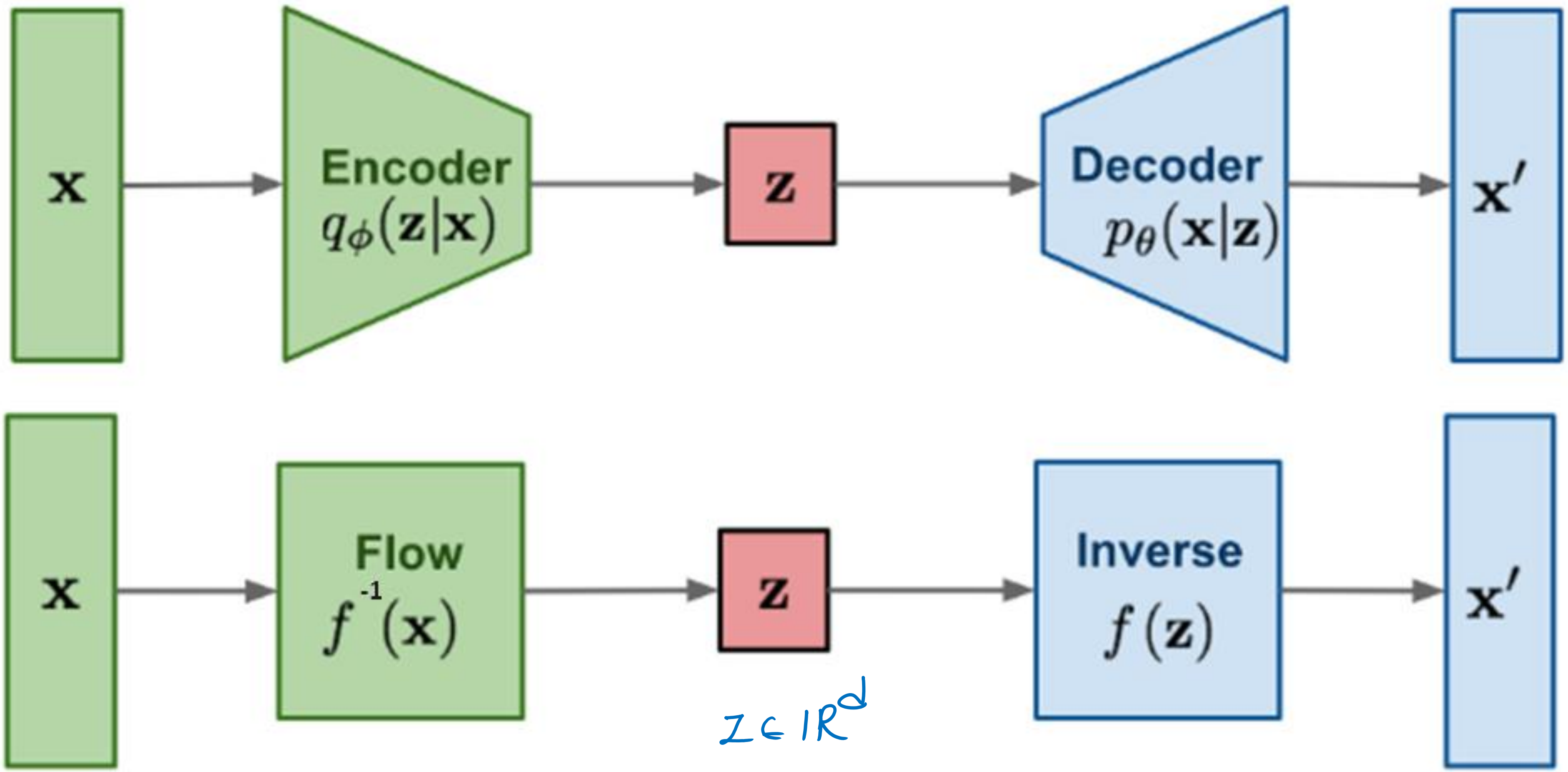
Normalizing Flows

Mostafa Tavassolipour

Fall 2024

Normalizing Flows vs. VAE

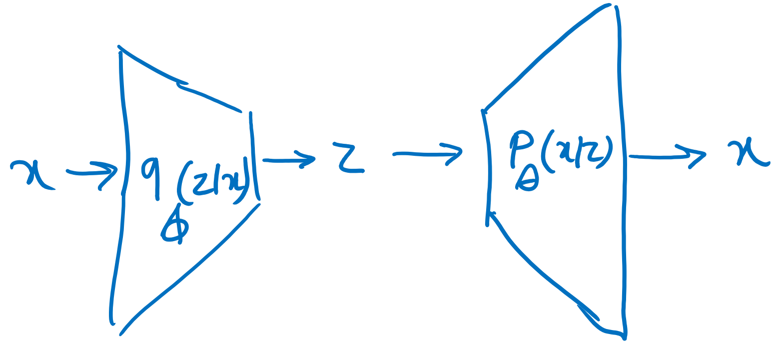
$x \in \mathbb{R}^n$



$z \in \mathbb{R}^d$

Learning

VAE



$$\hat{\theta}_{ML} = \max_{\theta} \sum_{i=1}^{n} \log p(x_i) \quad \Longrightarrow \quad \hat{\theta} = \arg\max_{\theta} \overbrace{-KL\left(q_\phi(z|x) \,\|\, p_\theta(x,z)\right)}^{ELBO}$$

$$\phi = \arg\min_{\phi} KL\left(q_\phi(z|x) \,\|\, p_\theta(x,z)\right)$$

Normalizing Flow:

$$z \rightarrow \boxed{f_\theta} \rightarrow x$$

$$\hat{\theta}_{ML} = \arg\max_{\theta} \sum_{i=1}^{n} \log P(x_i)$$

$$z \sim N(0, I)$$

$$P_X(x) = P_Z(f_\theta^{-1}(x)) \underbrace{\left|\det(J)\right|}$$

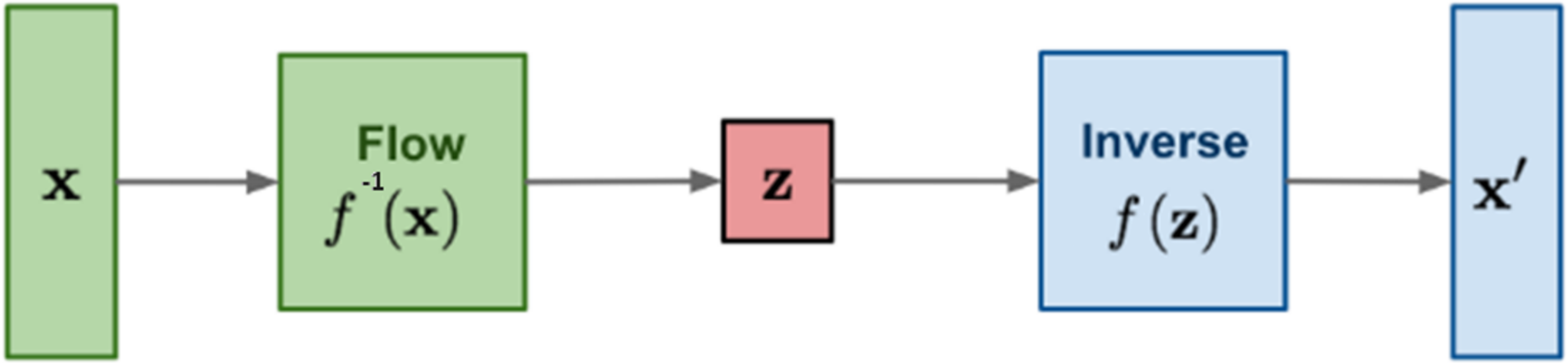
$$J = \frac{d f_\theta^{-1}}{d x}$$

$J_{n \times n}$

$x \in \mathbb{R}^n \qquad z \in \mathbb{R}^n$

$O(n^3)$

Change of Variable Formula: 1-Dimensional



$$P_X(x) = P_Z\left(f^{-1}(x)\right) \left|\frac{\partial f^{-1}(x)}{\partial x}\right|$$

Normalizing: Change of variables gives a normalized density after applying an invertible transformation
Flow: Invertible transformations can be composed with each other

$$\mathbf{z}_m = \mathbf{f}_\theta^m \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^m(\mathbf{f}_\theta^{m-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0)$$

- Start with a simple distribution for $\mathbf{z}_0$ (e.g., Gaussian)
- Apply a sequence of M invertible transformations to finally obtain $\mathbf{x} = \mathbf{z}_M$
- By change of variables

$$\underline{p_X(\mathbf{x};\theta)} = \underline{p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right)} \overbrace{\prod_{m=1}^{M} \left| \det \left(\frac{\partial (\mathbf{f}_\theta^m)^{-1}(\mathbf{z}_m)}{\partial \mathbf{z}_m} \right) \right|}$$

(Note: determininant of product equals product of determinants)

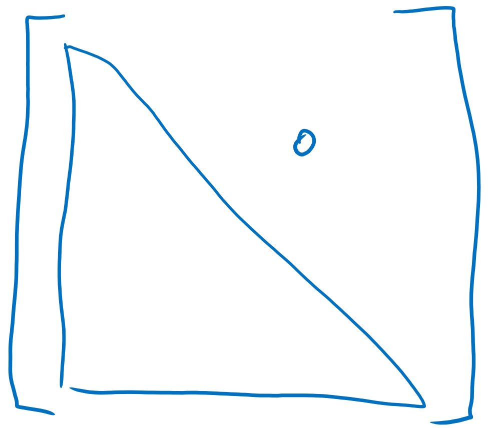
Diagonal Jacobian

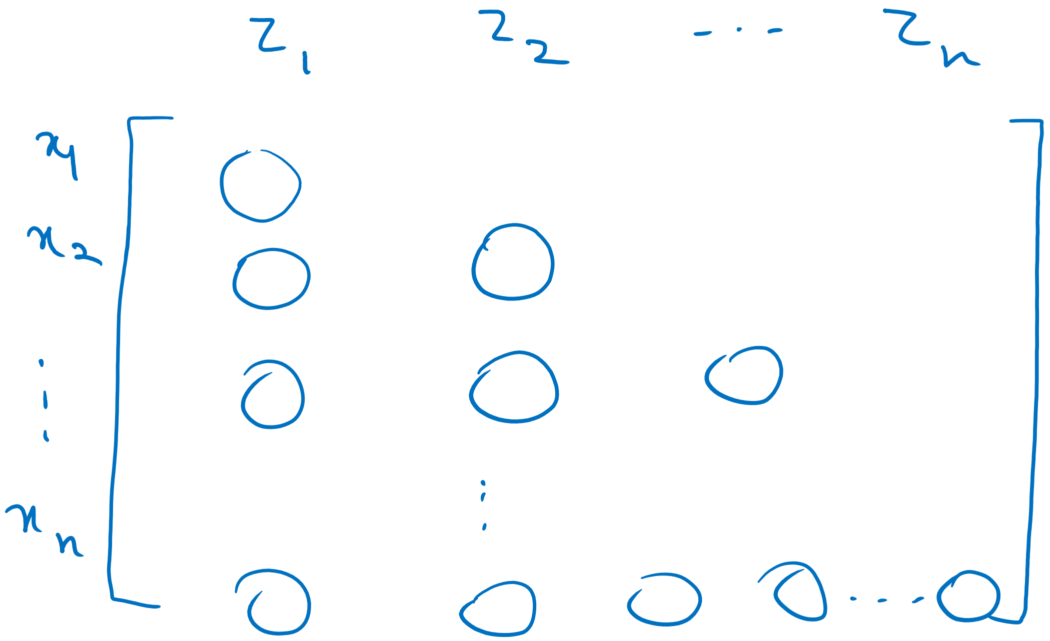
$$\begin{bmatrix} O & & & 0 \\ & O & & \\ & & \ddots & \\ 0 & & & O \end{bmatrix}_{n \times n}$$

$$\begin{array}{c|c} & \begin{array}{cccc} z_1 & z_2 & \cdots & z_n \end{array} \\ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} & \begin{bmatrix} \frac{dx_1}{dz_1} & \frac{dx_1}{dz_2} & & \frac{dx_1}{dz_n} \\ \frac{dx_2}{dz_1} & \frac{dx_2}{dz_2} & & \\ & & \ddots & \\ \frac{dx_n}{dz_1} & & & \frac{dx_n}{dz_n} \end{bmatrix} \end{array}$$

Triangular Jacobian

$J =$





$$x_i = f_{z_{j \le i}}$$

Models with Normalizing Flows

- **NICE** (Non-linear Independent Component Estimation, 2015)
- **RealNVP** (Real-valued Non-Volume Preserving, 2017)
- **Inverse Autoregressive Flow** (Kingma et al., 2016)
- **Masked Autoregressive Flow** (Papamakarios et al., 2017)

- I-resnet (Behrmann et al, 2018)
- Glow (Kingma et al, 2018)
- MintNet (Song et al., 2019)

- And many more

NICE

- Unit Jacobian determinant: $\boxed{|J| = 1}$

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \end{bmatrix}$$

$$\boldsymbol{z_1} = \boldsymbol{x_1}$$

$$\boldsymbol{z_2} = \boldsymbol{x_2} + \boldsymbol{m}(\boldsymbol{x_1})$$

$$\begin{array}{c|c} & \begin{array}{cc} z_1 & z_2 \end{array} \\ \begin{array}{c} x_1 \\ x_2 \end{array} & \begin{bmatrix} I_{d \times d} & 0 \\ -m'(z_1) & I \end{bmatrix} \end{array}$$

Inverse:

$$\boldsymbol{x_1} = \boldsymbol{z_1}$$

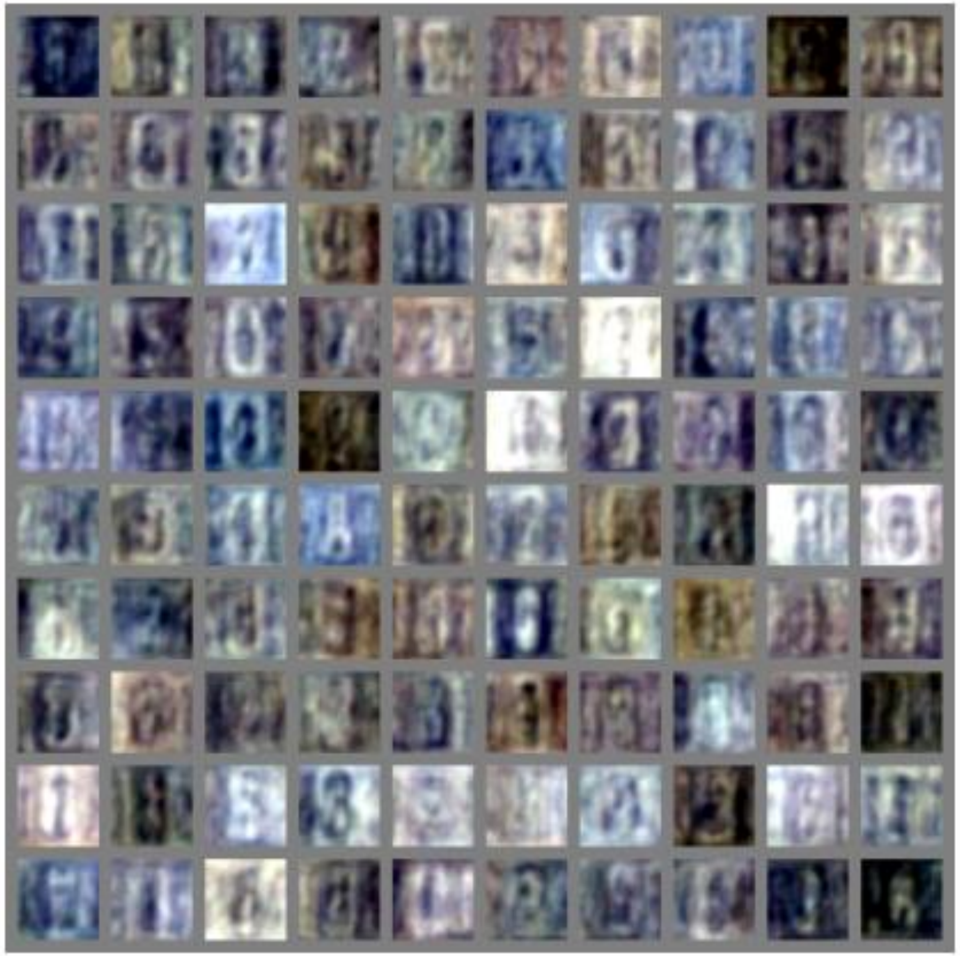
$$\boldsymbol{x_2} = \boldsymbol{z_2} - \underbrace{\boldsymbol{m}(\boldsymbol{z_1})}_{\text{شبکه عصبی}}$$

NICE: Results

(a) Model trained on MNIST

(b) Model trained on TFD

NICE: Results on Complex Data



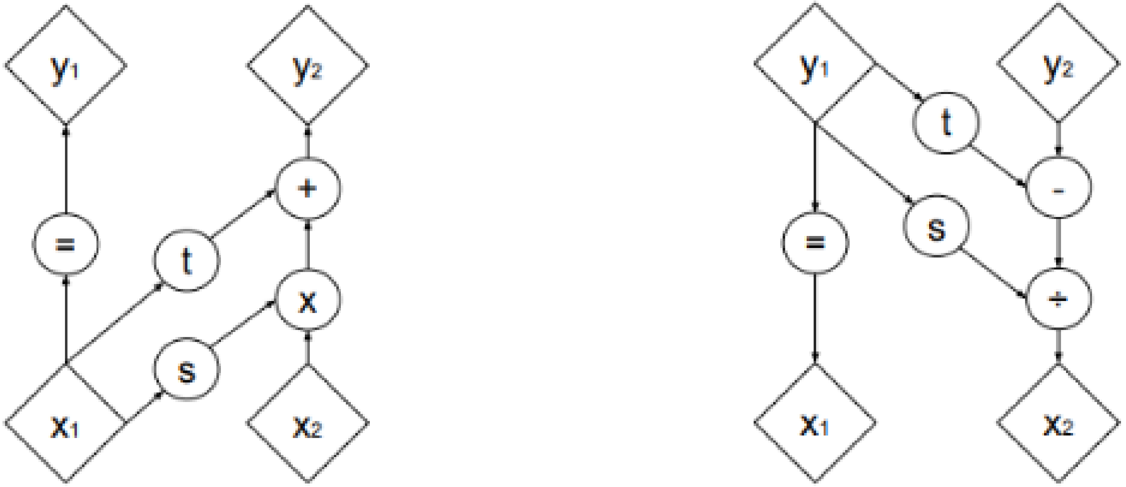
(c) Model trained on SVHN

(d) Model trained on CIFAR-10

RealNVP: non-Volume preserving extension of Nice

$$\boxed{z_0 \to \boxed{f_1} \to z_1} \to \boxed{f_2} \to z_2 \to \cdots \to \boxed{f_k} \to z_k = x$$

$$\sum_{i=1}^{n} \underbrace{\log P_X (x_i)}$$



(a) Forward propagation (b) Inverse propagation

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \end{bmatrix}$$

$$\boldsymbol{y_1} = \boldsymbol{x_1}$$
$$\boldsymbol{y_2} = s(\boldsymbol{x_1})\boldsymbol{x_2} + \boldsymbol{t}(\boldsymbol{x_1})$$

$$\to \boldsymbol{x_1} = \boldsymbol{y_1}$$
$$\boldsymbol{x_2} = \frac{\boldsymbol{y_2} - \boldsymbol{t}(\boldsymbol{y_1})}{s(\boldsymbol{y_1})}$$

$$\begin{array}{c|cc} & y_1 & y_2 \\ \hline x_1 & \multicolumn{2}{c}{\begin{bmatrix} I & 0 \\ ? & \frac{1}{s(y_1)} I_{d \times d} \end{bmatrix}} \\ x_2 & & \end{array}$$

$$|J| = \left(\frac{1}{s(y_1)}\right)^d$$

Samples generated via Real-NVP

RealNVP: More Results

CIFAR-10

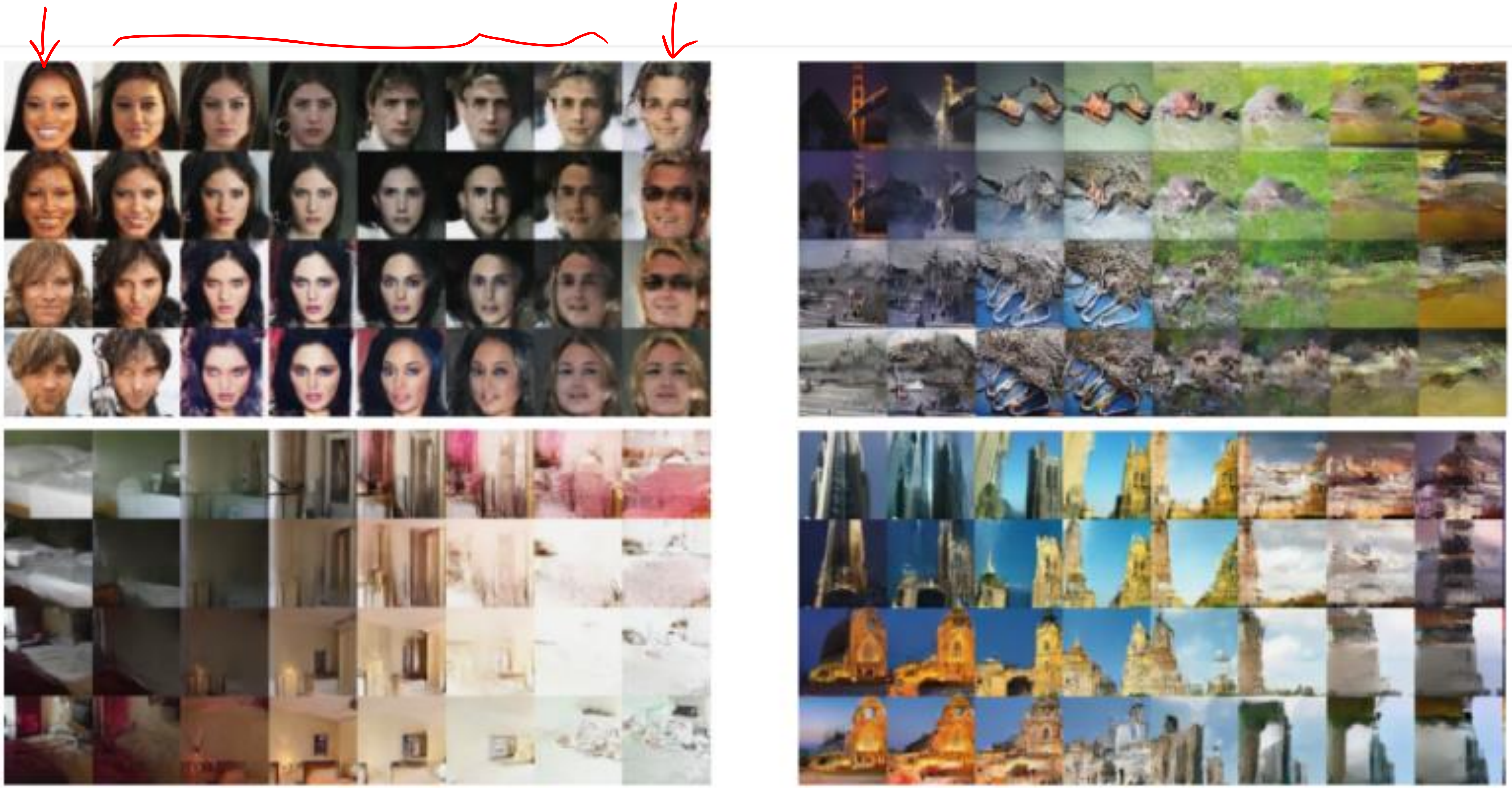
Imagenet (32x32)

Imagenet (64x64)

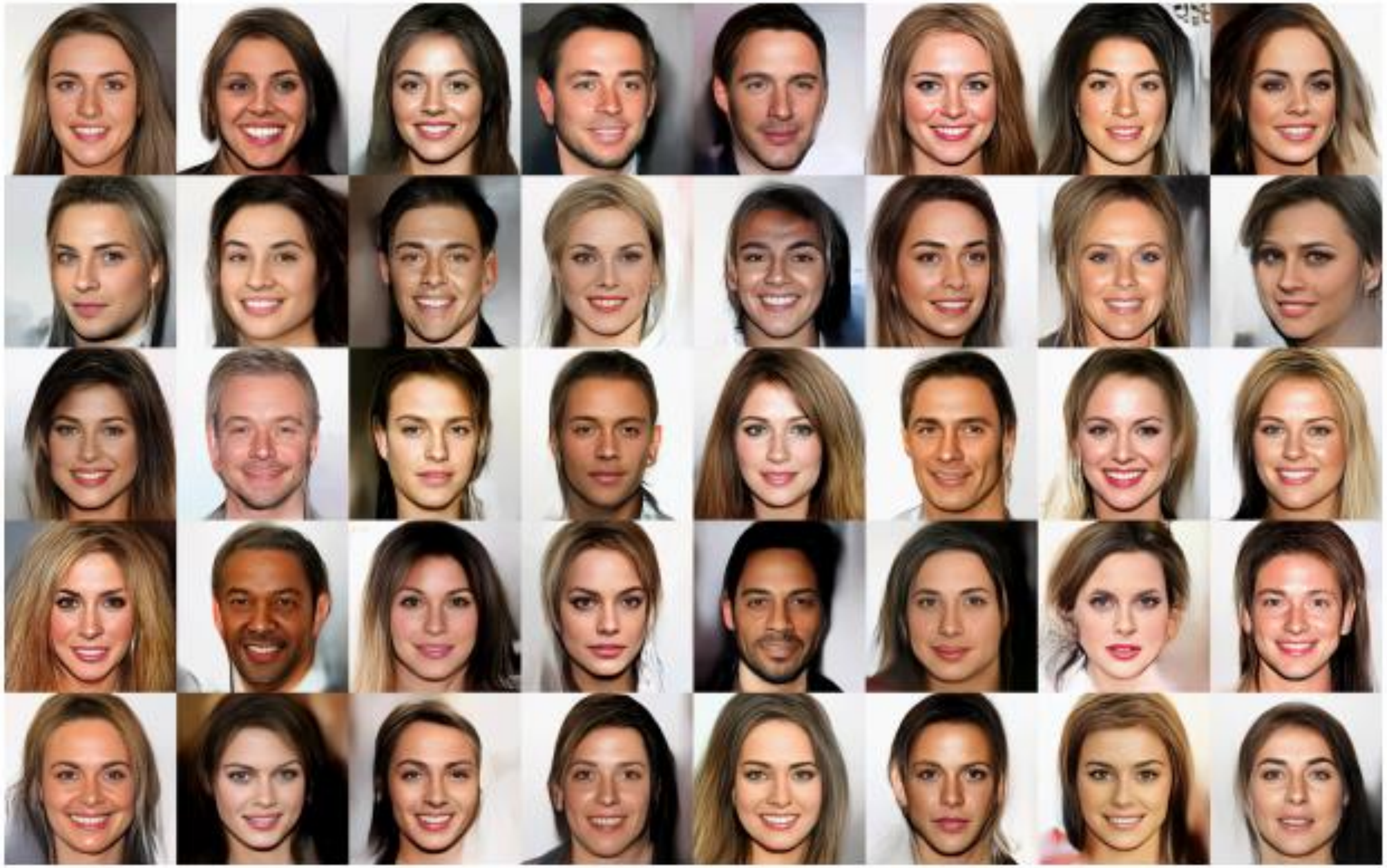
CelebA

LSUN

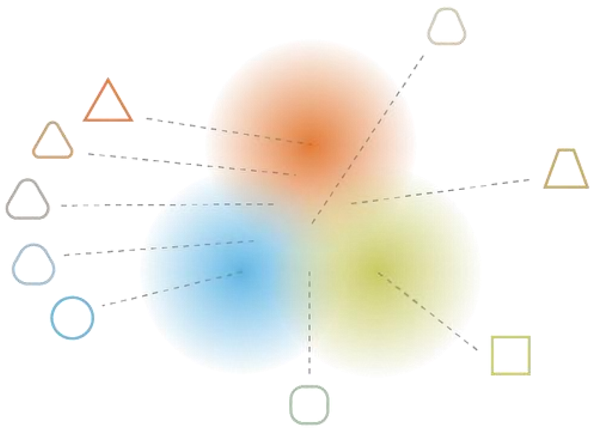
Real-NVP: latent space interpolation



GLOW: Result on CelebA

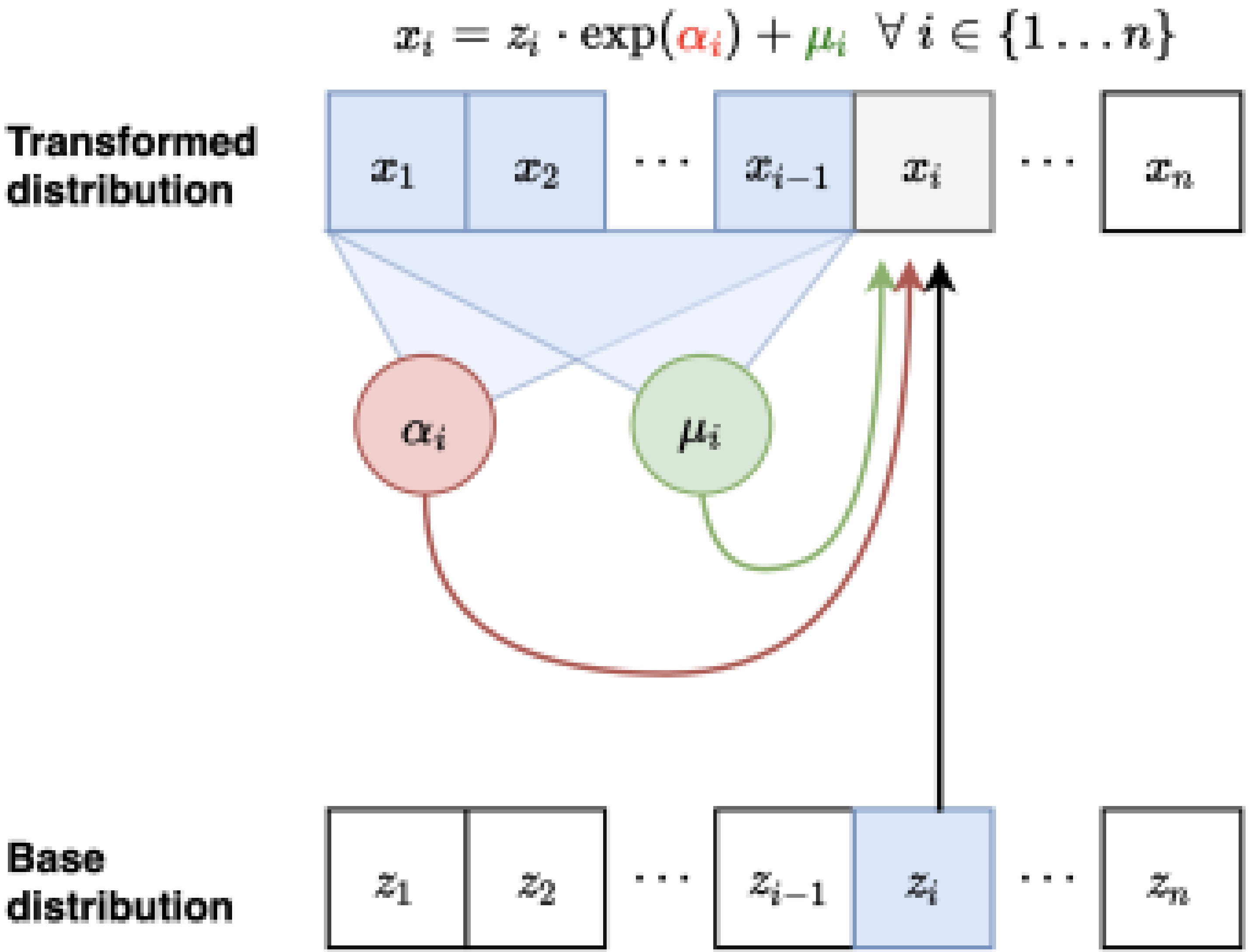


GLOW: Interpolation

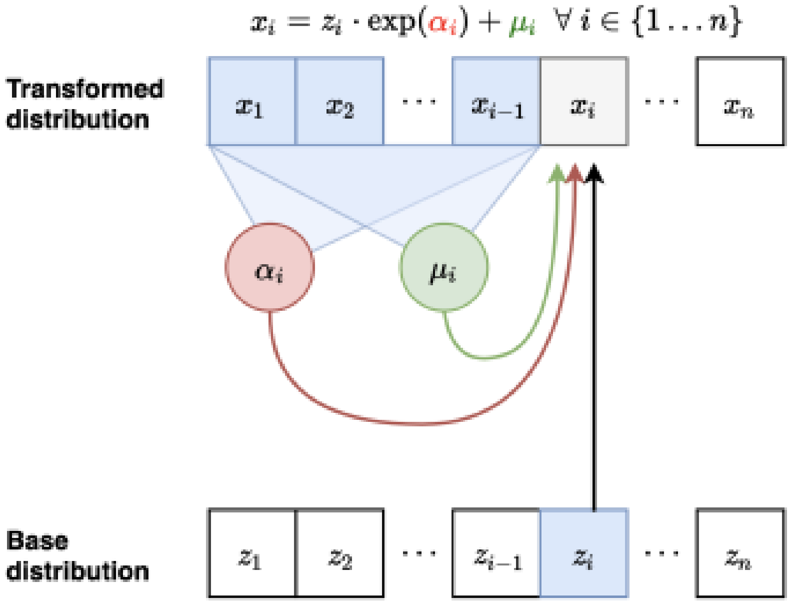


GLOW: Sample Generated Face

Masked Autoregressive Flow (MAF)

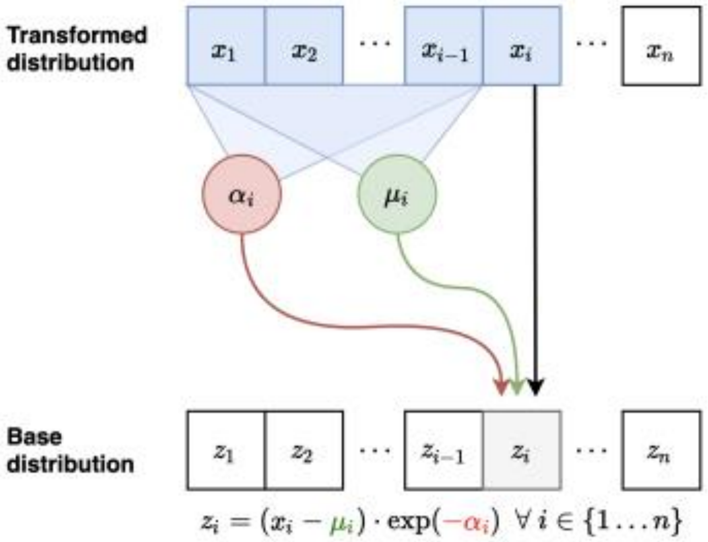


MAF



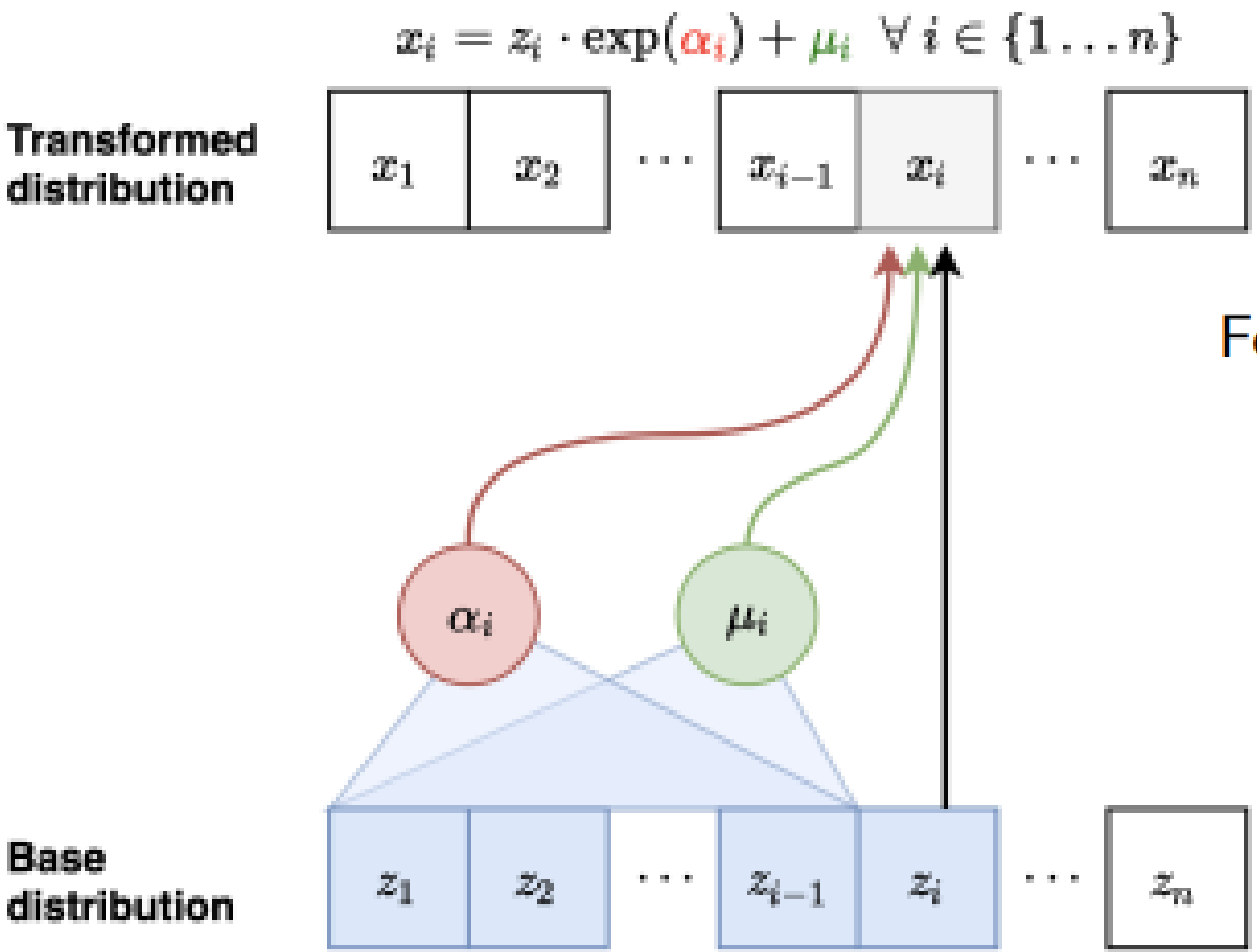
- Forward mapping from $\mathbf{z} \mapsto \mathbf{x}$:
  - Let $x_1 = \exp(\alpha_1)z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$
  - Let $x_2 = \exp(\alpha_2)z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
- Sampling is sequential and slow (like autoregressive): $O(n)$ time

Masked Autoregressive Flow (MAF)



- Inverse mapping from $\mathbf{x} \mapsto \mathbf{z}$:
  - Compute **all** μ_i, α_i (can be done in parallel using e.g., MADE)
  - Let $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$ (scale and shift)
  - Let $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$
  - Let $z_3 = (x_3 - \mu_3)/\exp(\alpha_3)$ ...
- Jacobian is lower diagonal, hence efficient determinant computation
- Likelihood evaluation is easy and parallelizable (like MADE)
- Layers with different variable orderings can be stacked

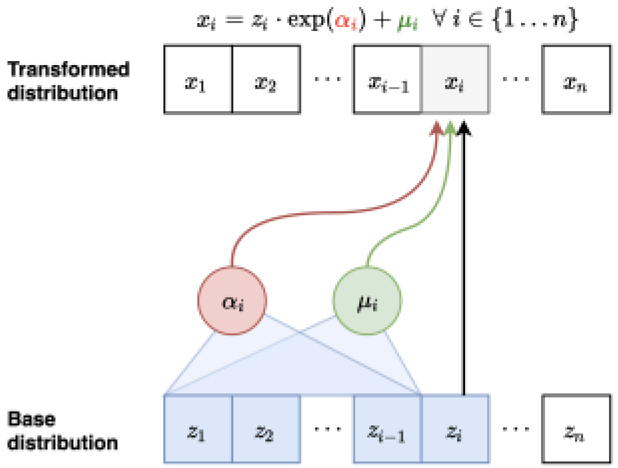
Inverse Autoregressive Flow (IAF)



Forward mapping from $\mathbf{z} \mapsto \mathbf{x}$ (parallel):

- Sample $z_i \sim \mathcal{N}(0,1)$ for $i = 1, \cdots, n$
- Compute all μ_i, α_i (can be done in parallel)
- Let $x_1 = \exp(\alpha_1) z_1 + \mu_1$
- Let $x_2 = \exp(\alpha_2) z_2 + \mu_2$ ...

IAF



- Forward mapping from $\mathbf{z} \mapsto \mathbf{x}$ (parallel):
  - Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \cdots, n$
  - Compute all μ_i, α_i (can be done in parallel)
  - Let $x_1 = \exp(\alpha_1) z_1 + \mu_1$
  - Let $x_2 = \exp(\alpha_2) z_2 + \mu_2 \ldots$
- Inverse mapping from $\mathbf{x} \mapsto \mathbf{z}$ (sequential):
  - Let $z_1 = (x_1 - \mu_1)/\exp(\alpha_1)$. Compute $\mu_2(z_1), \alpha_2(z_1)$
  - Let $z_2 = (x_2 - \mu_2)/\exp(\alpha_2)$. Compute $\mu_3(z_1, z_2), \alpha_3(z_1, z_2)$
- Fast to sample from, slow to evaluate likelihoods of data points (train)
- Note: Fast to evaluate likelihoods of a generated point (cache $z_1, z_2, \ldots, z_n$)

Figure adapted from Eric Jang's blog

IAF vs. MAF

$p(x_1, x_2, \ldots, x_n)$

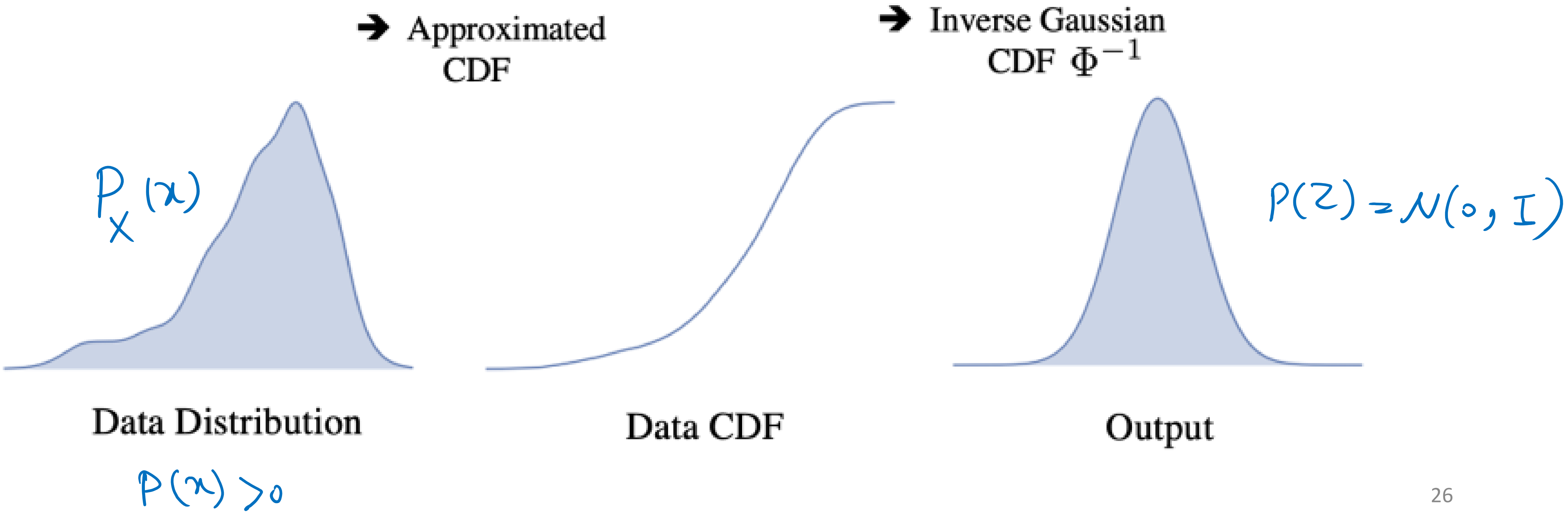
$= p(x_1)\, p(x_2|x_1)\, p(x_3|x_1, x_2) \cdots p(x_n|x_1, \ldots, x_{n-1})$

- Computational tradeoffs
  - MAF: Fast likelihood evaluation, slow sampling
  - IAF: Fast sampling, slow likelihood evaluation
- MAF more suited for training based on MLE, density estimation
- IAF more suited for real-time generation
- Can we get the best of both worlds?

Gaussianization Flows (Meng et al., 2020)

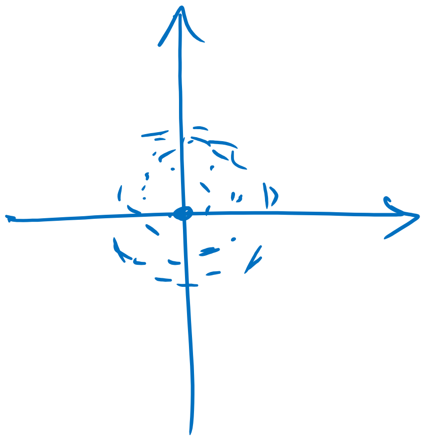
- Inverse CDF trick:

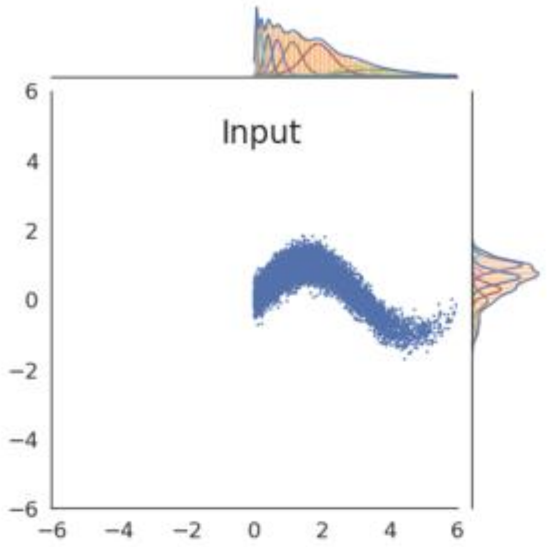
$$Z = \Phi^{-1}\big(\underbrace{F_X(X)}_{U}\big)$$



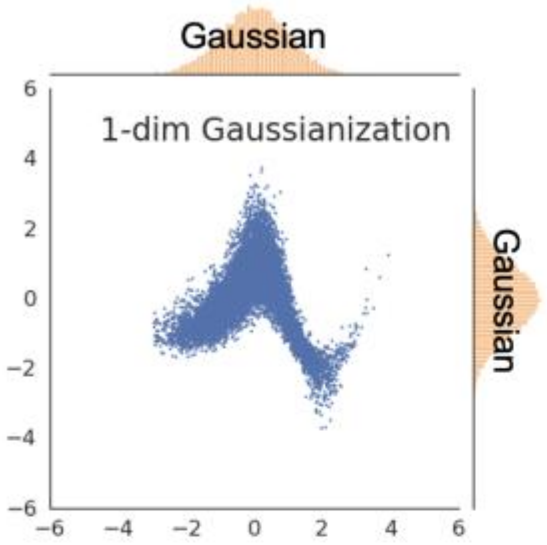
Gaussianization Flows (Meng et al., 2020)

- Step 1: Dimension-wise Gaussianization (Jacobian is a diagonal matrix and is tractable)





Input data

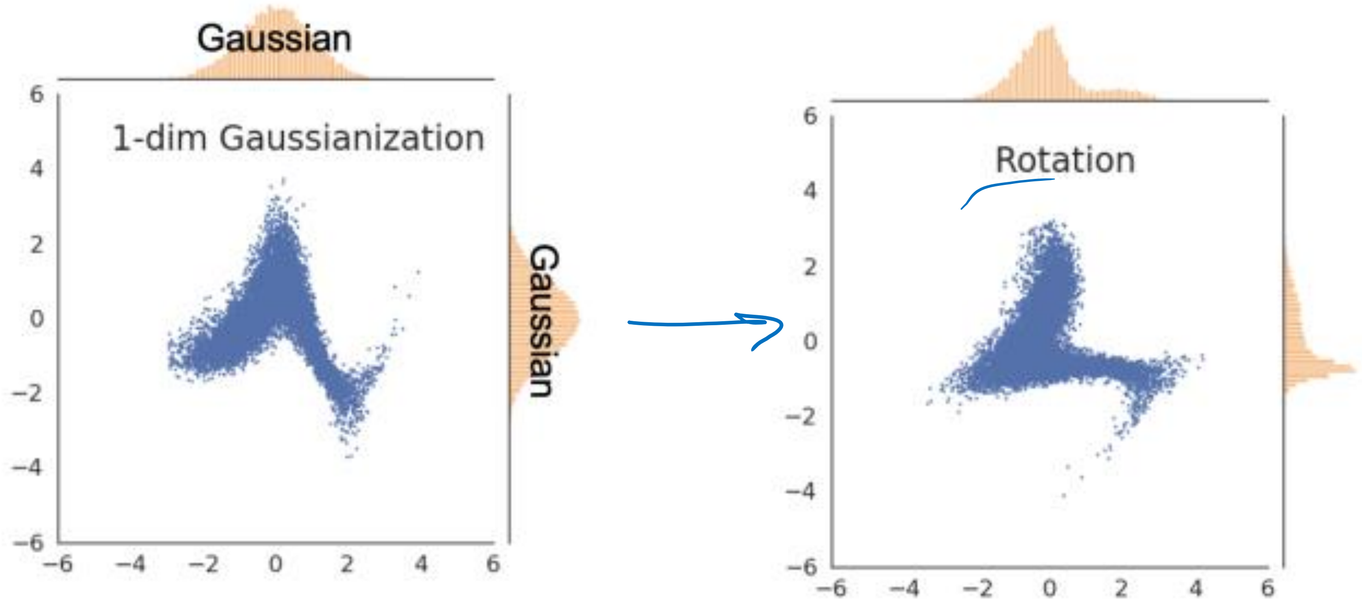


Dimension-wise Gaussianization

Note: Even though each dimension is marginally Gaussian, they are **not** jointly Gaussian. Aside: Approximating this with a Gaussian prior is a shallow flow model known as a copula model (Sklar, 1959).

Gaussianization Flows (Meng et al., 2020)

- Step 2: apply a rotation matrix to the transformed data (Jacobian is an orthogonal matrix and is tractable)

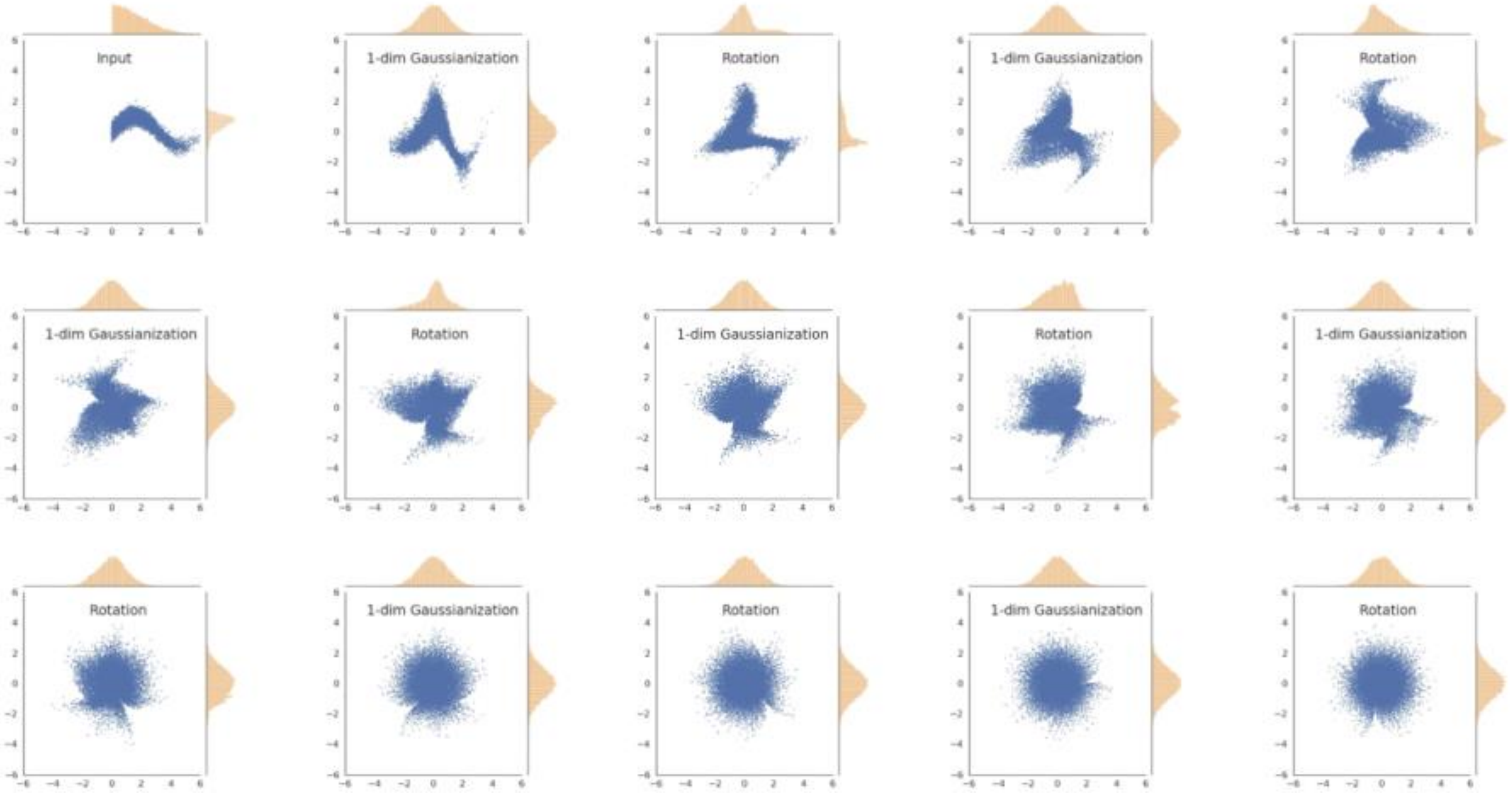


Input　　　　After rotation

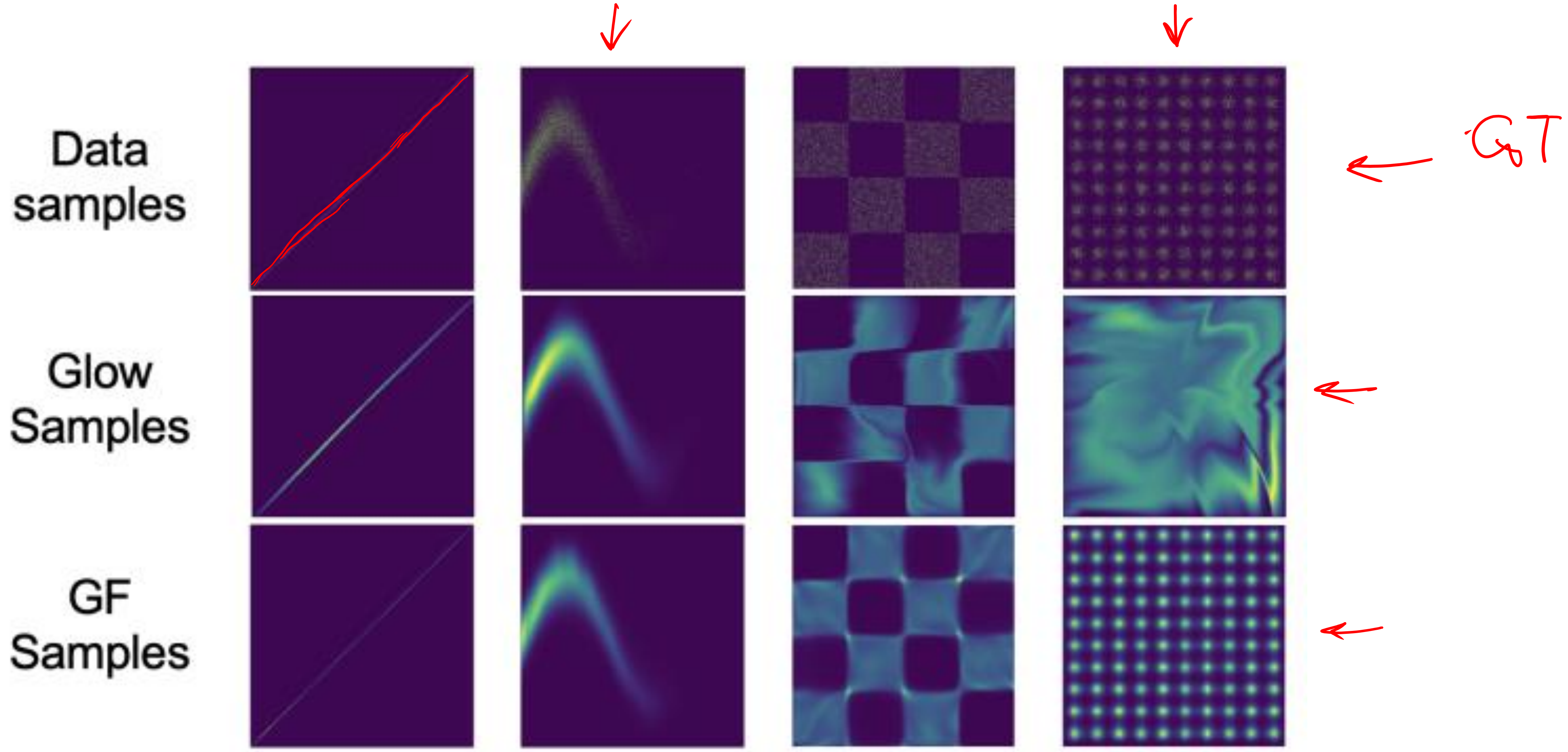
- Note: $\mathcal{N}(\mathbf{0}, \mathbf{I})$ is rotationally invariant

Gaussianization Flows (Meng et al., 2020)

- Gaussianization flow: repeat Step 1 and Step 2 (stacking learnable Gaussian copula). Transform data into a normal distribution.



Experiment: 2-D density estimation



Summary

- Transform simple distributions into more complex distributions via change of variables

- Jacobian of transformations should have tractable determinant for efficient learning and density estimation

- Computational tradeoffs in evaluating forward and inverse transformations