Normalizing flow

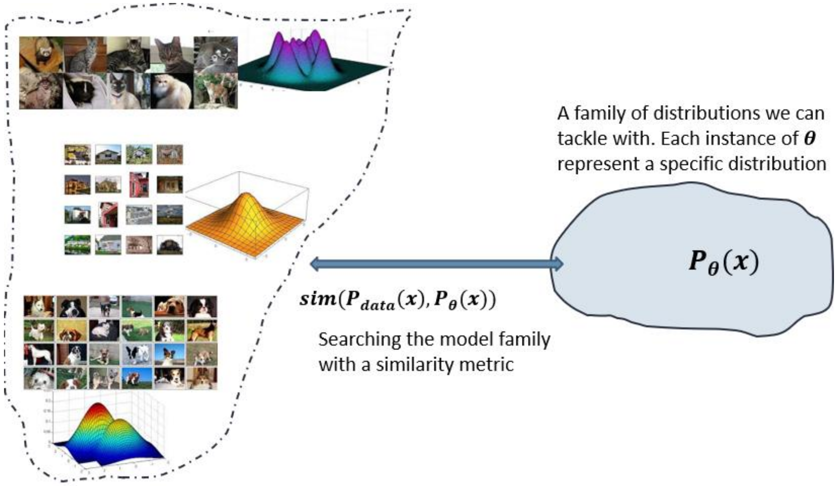
22-808: Generative models

Sharif University of Technology

Fall 2025

Fatemeh Seyyedsalehi

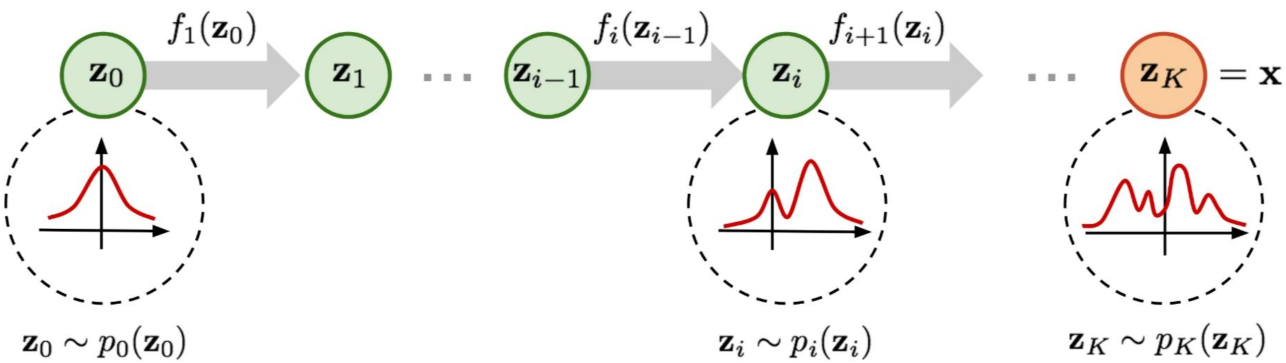
Recap

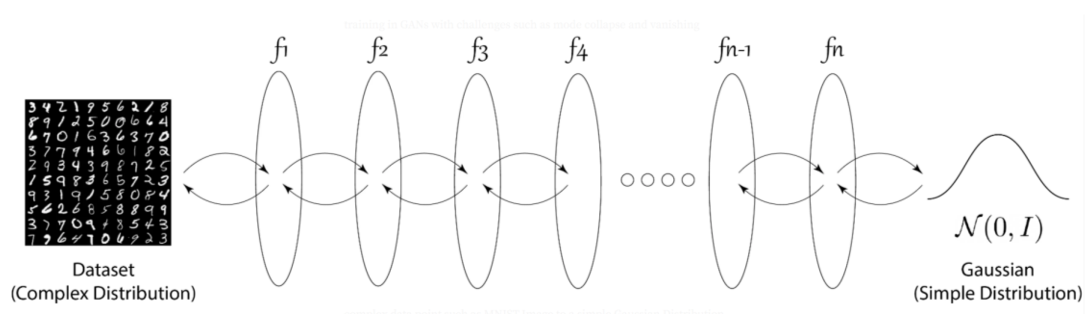


- We need a framework to interact with distributions for statistical generative models.
  - Probabilistic generative models
  - Deep generative models
    - Autoregressive models
    - Variational Autoencoders
    - Generative adversarial networks
    - **Normalizing Flow -> a latent variable model with tractable likelihood**

Motivation

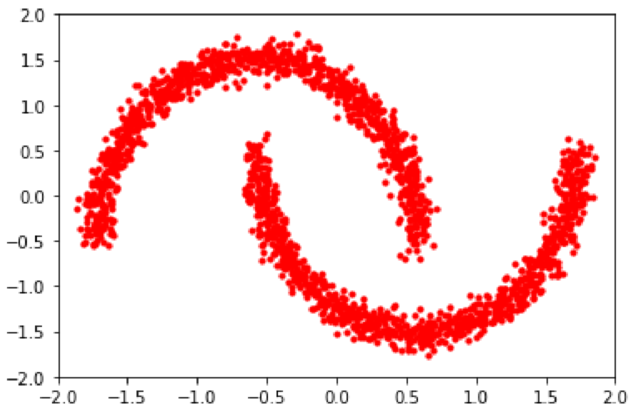
- Any two distributions can be converted to each other ☺
  - Start from a simple distribution and convert it to reach a sufficiently complex one to describe the data distribution
    - Find conversions with neural networks

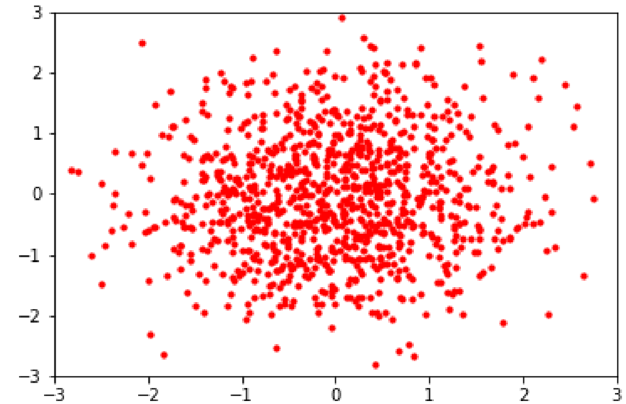


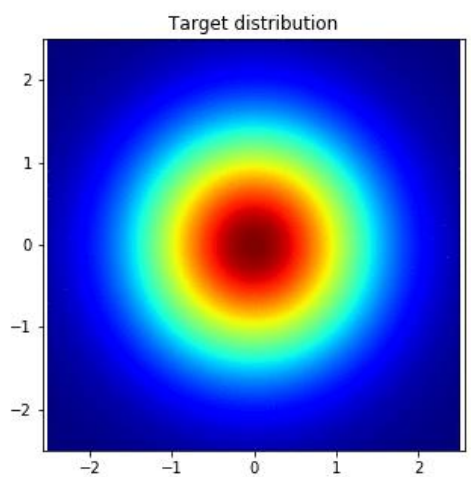


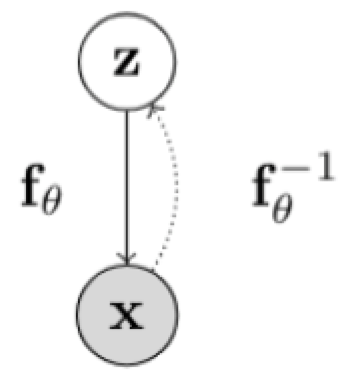
Motivation

- Any two distributions can be converted to each other ☺









Change of variable formula

- Imagine uniform random variable z on the green area
- We obtain another random variable $x = Az$

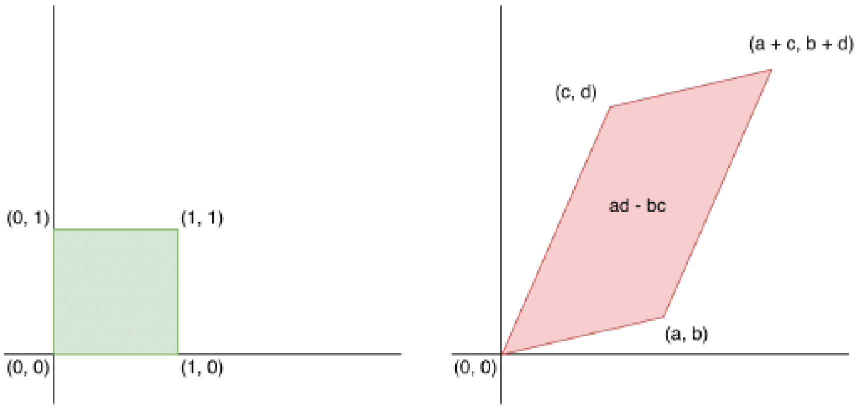
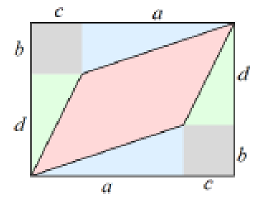


Figure: The matrix $A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ maps a unit square to a parallelogram

Change of variable formula

- Imagine uniform random variable z on the green area
- We obtain another random variable $x = Az$
- The volume of the parallelotope is:

$$\det(A) = \det\begin{pmatrix} a & c \\ b & d \end{pmatrix} = ad - bc$$



$(a+c)(b+d)-ab-2bc-cd=ad-bc$

- As z uniformly distributed over the square, x is also uniformly distributed in this volume, therefore:

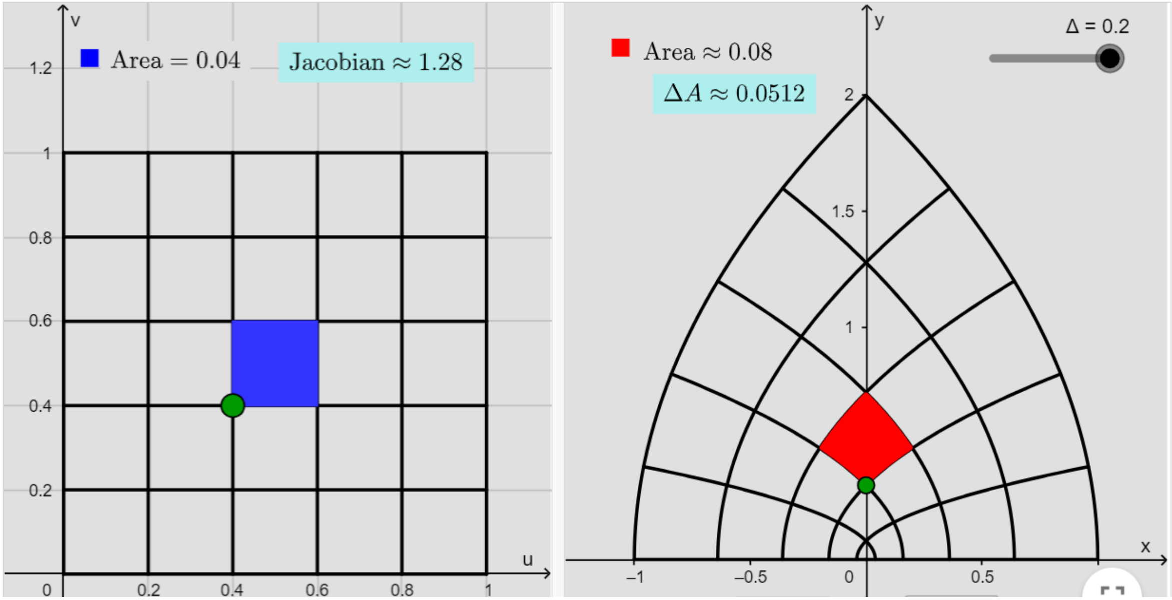
$$p_X(\mathbf{x}) = p_Z(W\mathbf{x}) \,/\, |\det(A)| = p_Z(W\mathbf{x})\,|\det(W)|$$

$$W = A^{-1},\ \det(W) = \frac{1}{\det(A)}$$

Generalized change of variable

- For an arbitrarily non-linear transformation f:

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



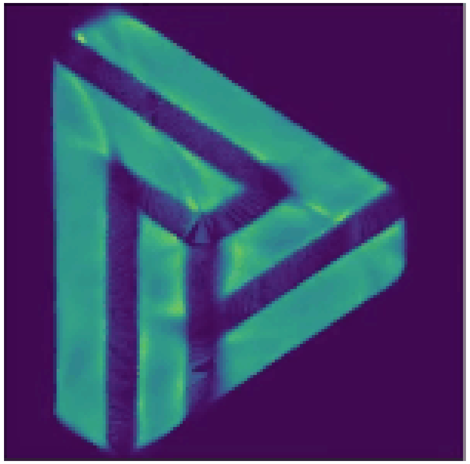
Jacobian matrix

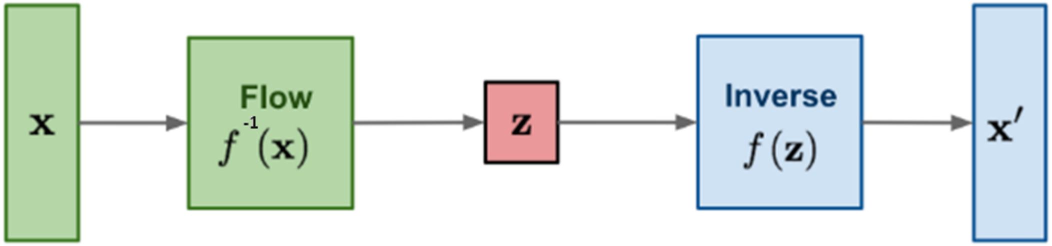
$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \xrightarrow{f} \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$\frac{\partial(y_1, \ldots, y_n)}{\partial(x_1, \ldots, x_n)} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_n} \end{bmatrix}$$

Normalizing flow

- In contrast to VAEs the variable the variable z has the same dimension of x.
- The function f should be **deterministic and invertible**





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

Learning and inference

- Learning with maximum likelihood

$$\max_{\theta} \log p_X(\mathcal{D};\theta) = \sum_{x\in\mathcal{D}} \log p_Z\left(f_\theta^{-1}(x)\right) + \log\left|\det\left(\frac{\partial f_\theta^{-1}(x)}{\partial x}\right)\right|$$

- Sampling

$$z \sim p_Z(z) \quad x = f_\theta(z)$$

- Latent representation

$$z = f_\theta^{-1}(x)$$

Learning and inference

- Computing the determinant for an $n \times n$ matrix is $\boldsymbol{O(n^3)}$: prohibitively expensive within a learning loop!

- **Key idea**: Choose transformations so that the resulting Jacobian matrix has **special structure**. For example, the determinant of a **triangular** matrix is the product of the diagonal entries, i.e., an $\boldsymbol{O(n)}$ operation.

- Therefore, we have to only consider spatial family of models which limits the ability of this approach.

NICE - Additive coupling layers

Partition the variables $\mathbf{z}$ into two disjoint subsets, say $\mathbf{z}_{1:d}$ and $\mathbf{z}_{d+1:n}$ for any $1 \leq d < n$

- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d})$ ($m_\theta(\cdot)$ is a neural network with parameters θ, d input units, and $n-d$ output units)
- Inverse mapping $\mathbf{x} \mapsto \mathbf{z}$:
  - $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$ (identity transformation)
  - $\mathbf{z}_{d+1:n} = \mathbf{x}_{d+1:n} - m_\theta(\mathbf{x}_{1:d})$
- Jacobian of forward mapping:

$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & I_{n-d} \end{pmatrix}$$

$$\det(J) = 1$$

- **Volume preserving transformation** since determinant is 1.

NICE - Additive coupling layers

(a) Model trained on MNIST

(b) Model trained on TFD

NICE - Additive coupling layers

(c) Model trained on SVHN

(d) Model trained on CIFAR-10

Real-NVP: Non-volume preserving extension of NICE

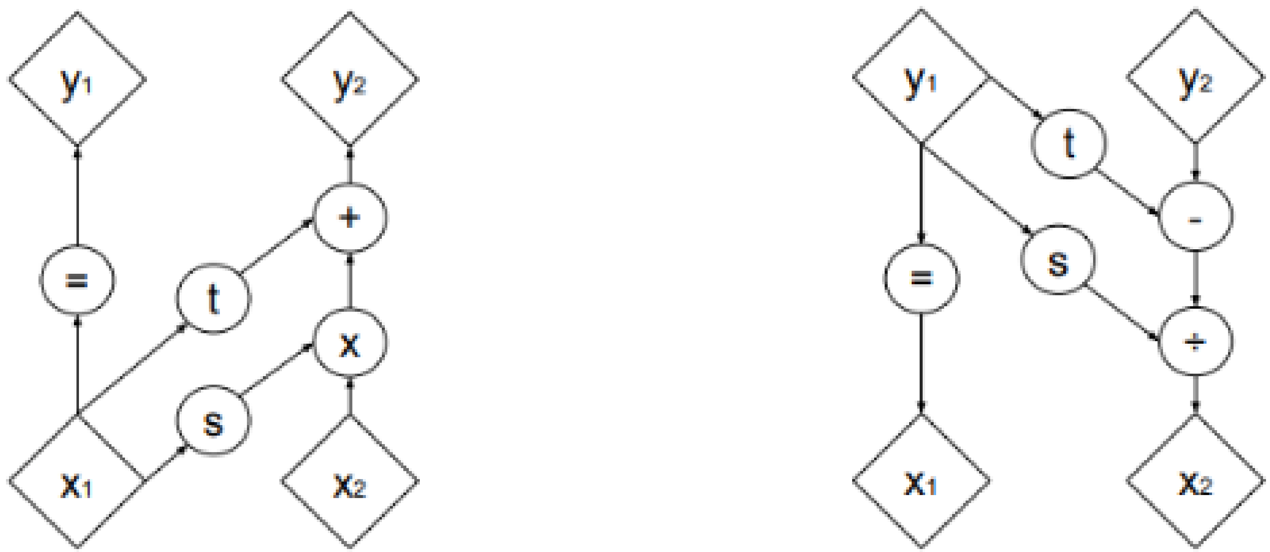
- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
  - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
  - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\theta(\mathbf{z}_{1:d})) + \mu_\theta(\mathbf{z}_{1:d})$
  - $\mu_\theta(\cdot)$ and $\alpha_\theta(\cdot)$ are both neural networks with parameters θ, d input units, and $n - d$ output units [$\odot$ denotes elementwise product]
- Inverse mapping $\mathbf{x} \mapsto \mathbf{z}$:
  - $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$ (identity transformation)
  - $\mathbf{z}_{d+1:n} = (\mathbf{x}_{d+1:n} - \mu_\theta(\mathbf{x}_{1:d})) \odot (\exp(-\alpha_\theta(\mathbf{x}_{1:d})))$
- Jacobian of forward mapping:

$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \operatorname{diag}(\exp(\alpha_\theta(\mathbf{z}_{1:d}))) \end{pmatrix}$$

$$\det(J) = \prod_{i=d+1}^{n} \exp(\alpha_\theta(\mathbf{z}_{1:d})_i) = \exp\left(\sum_{i=d+1}^{n} \alpha_\theta(\mathbf{z}_{1:d})_i\right)$$

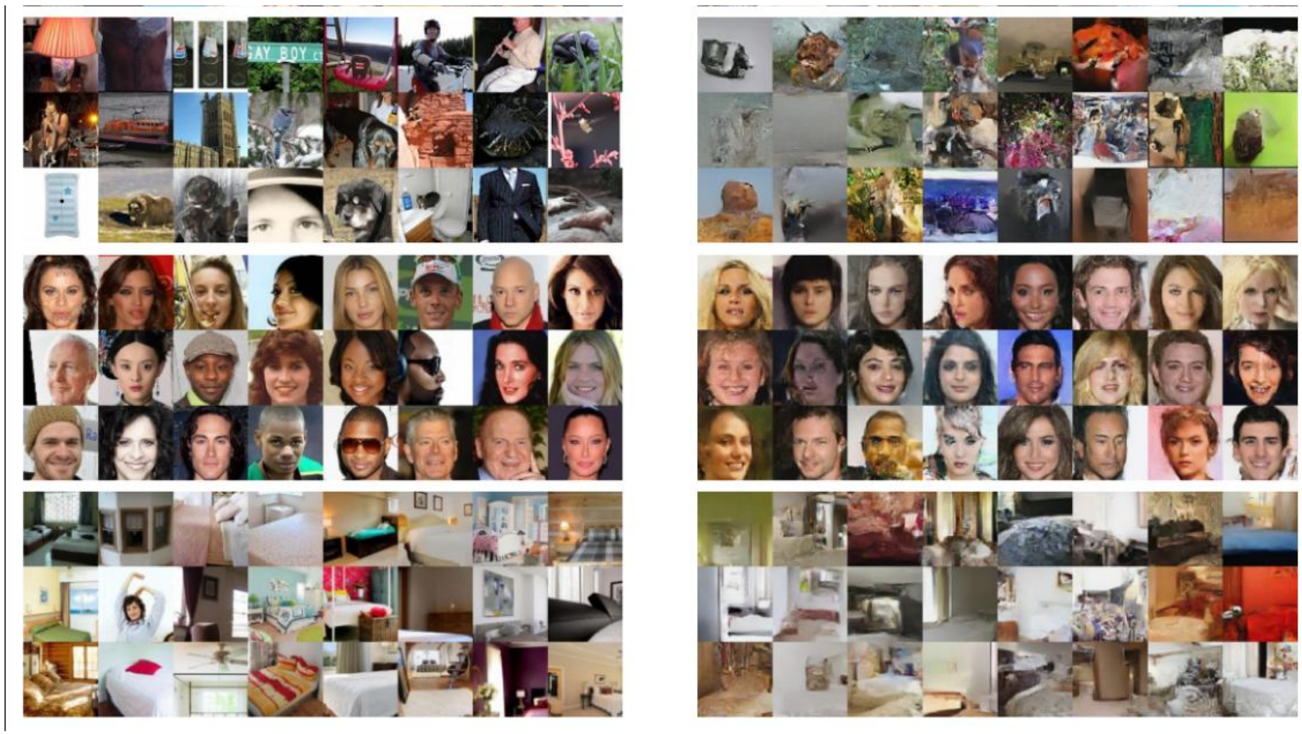
- **Non-volume preserving transformation** in general since determinant can be less than or greater than 1

Real-NVP: Non-volume preserving extension of NICE



(a) Forward propagation (b) Inverse propagation

Real-NVP: Non-volume preserving extension of NICE



Continuous Autoregressive models as flow models

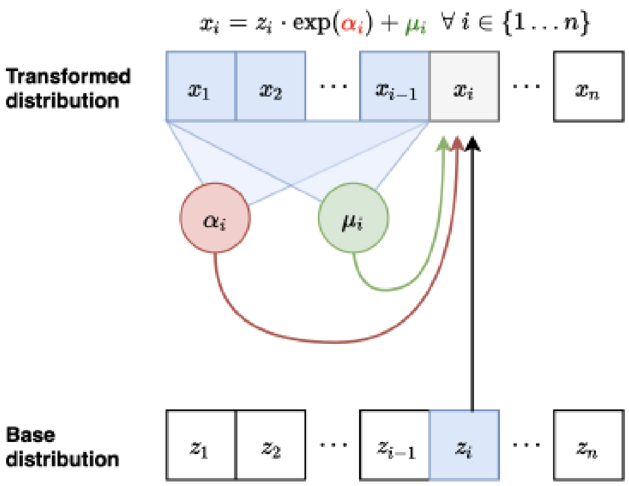
- Consider a Gaussian autoregressive model:

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

such that $p(x_i \mid \mathbf{x}_{<i}) = \mathcal{N}(\mu_i(x_1, \cdots, x_{i-1}), \exp(\alpha_i(x_1, \cdots, x_{i-1}))^2)$. Here, $\mu_i(\cdot)$ and $\alpha_i(\cdot)$ are neural networks for $i > 1$ and constants for $i = 1$.

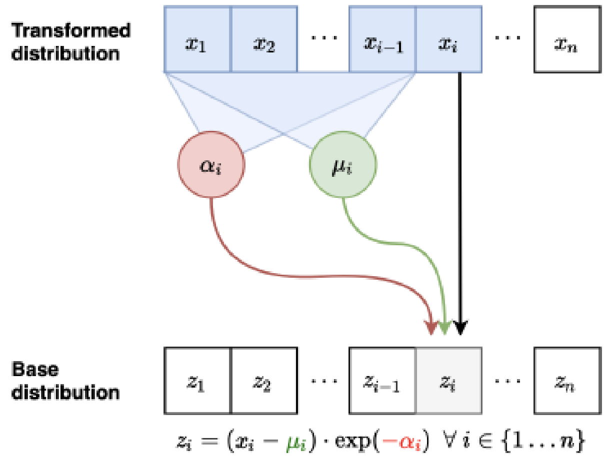
- Sampler for this model:
  - Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \cdots, n$
  - Let $x_1 = \exp(\alpha_1)z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$
  - Let $x_2 = \exp(\alpha_2)z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
  - Let $x_3 = \exp(\alpha_3)z_3 + \mu_3$. ...
- **Flow interpretation:** transforms samples from the standard Gaussian $(z_1, z_2, \dots, z_n)$ to those generated from the model $(x_1, x_2, \dots, x_n)$ via invertible transformations (parameterized by $\mu_i(\cdot), \alpha_i(\cdot)$)

Masked Autoregressive Flow (MAF)



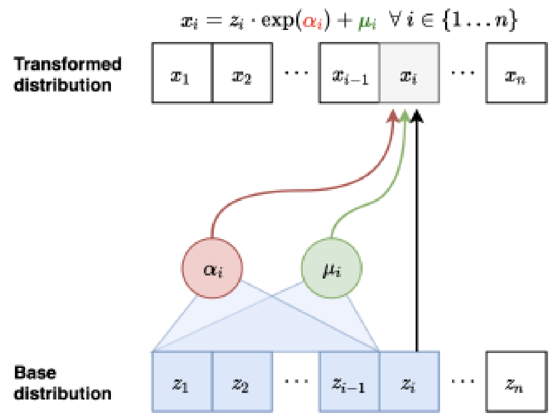
- Forward mapping from $\mathbf{z} \mapsto \mathbf{x}$:
  - Let $x_1 = \exp(\alpha_1) z_1 + \mu_1$. Compute $\mu_2(x_1), \alpha_2(x_1)$
  - Let $x_2 = \exp(\alpha_2) z_2 + \mu_2$. Compute $\mu_3(x_1, x_2), \alpha_3(x_1, x_2)$
- Sampling is sequential and slow (like autoregressive): $O(n)$ time

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 $x_2 = \exp(\alpha_2) z_2 + \mu_2$ ...
- 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 $z_1, z_2, \ldots, z_n$)

IAF vs. MAF

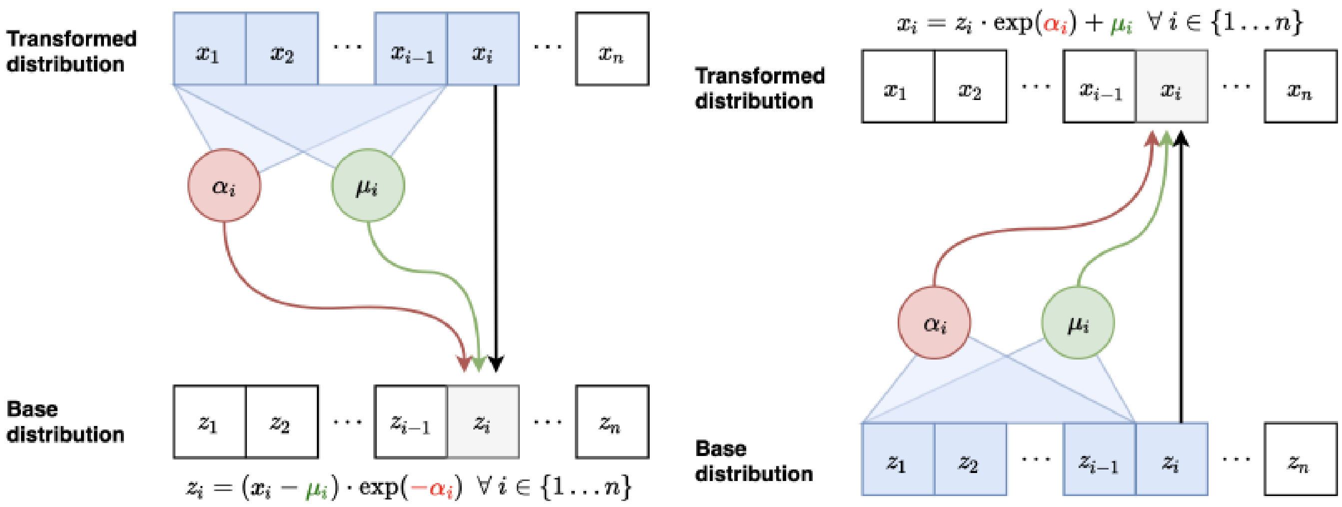


Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

- Interchanging **z** and **x** in the inverse transformation of MAF gives the forward transformation of IAF
- Similarly, forward transformation of MAF is inverse transformation of IAF

Activat

IAF vs. MAF

- 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?