Normalizing Flow

INTELLIGENT SYSTEMS FOR PATTERN RECOGNITION (ISPR)

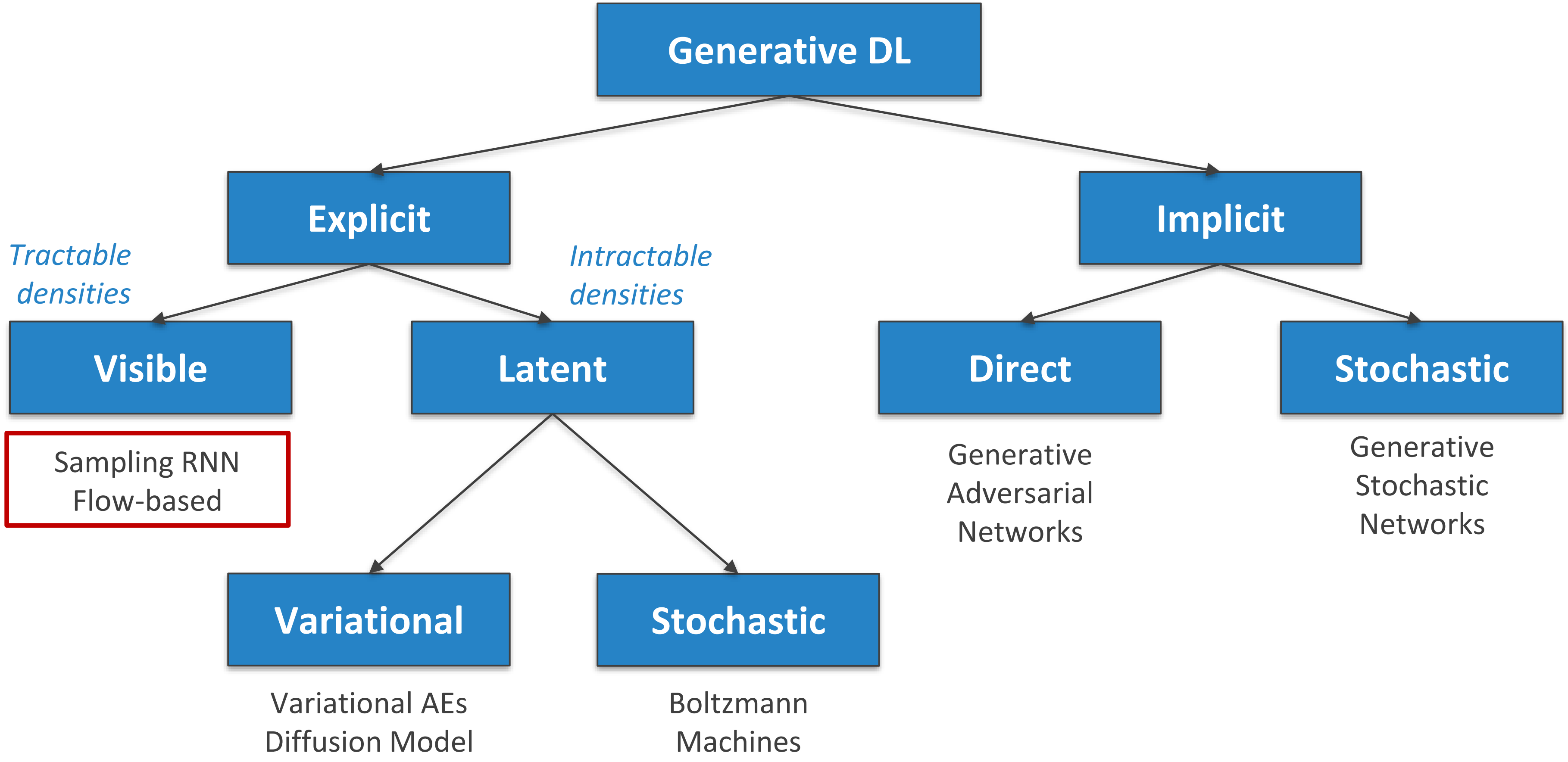
DAVIDE BACCIU – DIPARTIMENTO DI INFORMATICA - UNIVERSITA' DI PISA

DAVIDE.BACCIU@UNIPI.IT

Lecture Outline

- Introduction
  - Change of variable
  - Flows fundamentals
  - From 1D to multi-dimensional flows
- Neural flow layers
  - Coupling flows
  - Masking & squeezing
  - Invertible convolutions
  - Autoregressive flows
- Normalizing flows and deep generative models wrap-up

A Taxonomy

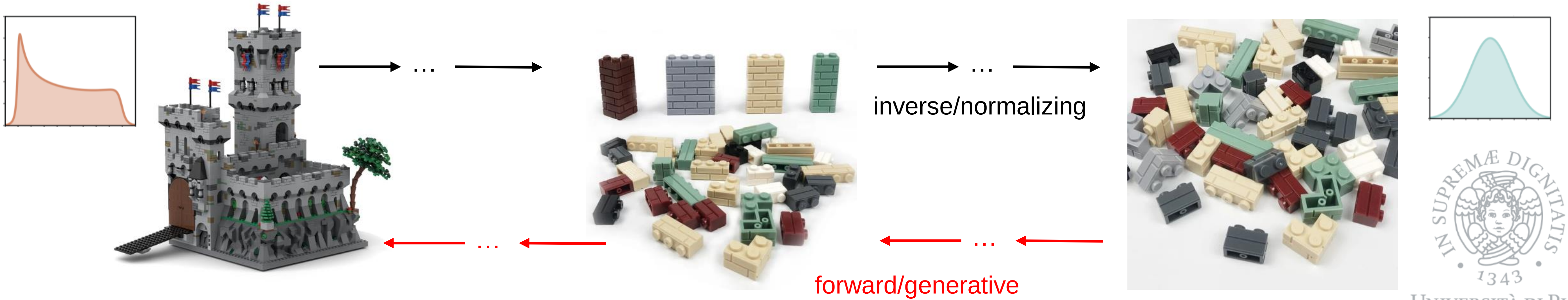


Adapted from I. Goodfellow, Tutorial on Generative Adversarial Networks, 2017

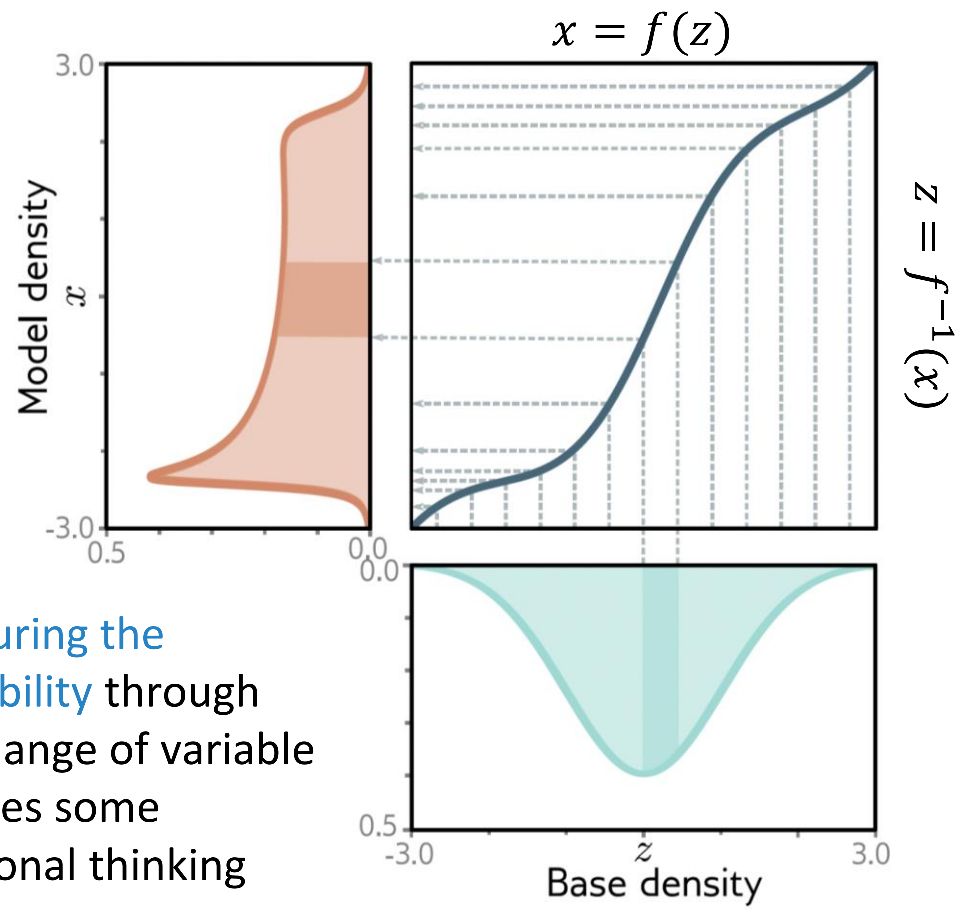
Normalizing Flow Fundamentals

Normalizing Flow (NF) – The Intuition

- Learn a probabilistic model by transforming a simple distribution into the complex data generating distribution using a deep network
  - Easy to sample and evaluate the probability
  - Requires a specialized architecture where each layer must be invertible



Probabilistic Change of Variable



Measuring the probability through the change of variable requires some additional thinking

- Take a tractable base distribution $P(z)$ over latent variable z and a model density $P(x)$ over data x
- Apply a change of variable function (possibly learned with parameters θ)

$$x = f(z; \theta)$$

- In addition, we are going to require that f is invertible

$$z = f^{-1}(z; \theta)$$

Linear 1D Change of Variable

NF define complex densities by transforming a base one by invertible mappings (bijections)

- Simplest case in 1D is a univariate Gaussian base density

$$z \sim \mathcal{N}(0,1)$$

- Simplest change of variable (forward) by linear transformation

$$x = f(z; \mu, \sigma) = \mu + z\sigma$$

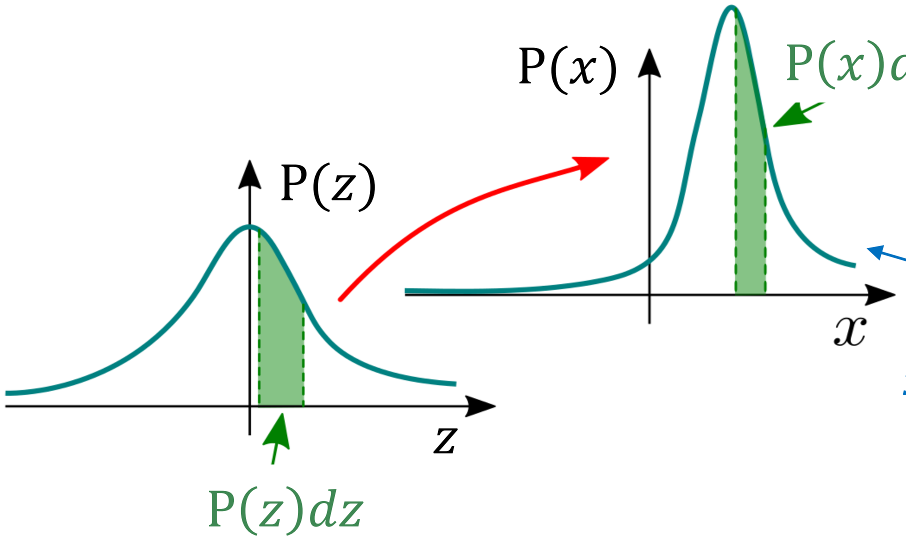
- Inverse then (under $\sigma \neq 0$)

$$z = f^{-1}(x; \mu, \sigma) = (x - \mu)/\sigma$$

- With $\mathrm{P}(z)$ known we want to find $\mathrm{P}(\mathrm{x})$

Linear 1D – Mass conservation

The volume may change but the density must be preserved



The necessary condition for this is

$$\mathrm{P}(z)dz = P(x)dx$$

The probability of data x under the transformed distribution is

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

$x \sim \mathcal{N}(\mu, \sigma^2)$

Change of volume

$x = \mu + \sigma z$

$\frac{x-\mu}{\sigma}$

UNIVERSITÀ DI PISA

Linear 1D – Iterated forward pass

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

Forward transformation equation

- Sample x through 2 mappings (transformations)

$$z_0 \sim P(z) \qquad z_1 = f_1(z_0) \qquad x = f_2(z_1)$$

- Density obtained by composing forward transformations

$$P(x) = P(z_0)\left|\frac{dz^1}{dz^0}\right|^{-1}\left|\frac{dx}{dz^1}\right|^{-1}$$

Linear 1D – Inverse Flow

- We may be interested in estimating the density of a given input sample x
- Requires building the inverse flow ($g = f^{-1}$)

$$z_1 = f_2^{-1}(x) = g_2(x) \qquad z_0 = f_1^{-1}(z_1) = g_1(z_1)$$

- And computing the density accordingly

$$P(x) = \left|\frac{dz^1}{dx}\right| \left|\frac{dz^0}{dz^1}\right| P(z_0)$$

Multidimensional flow

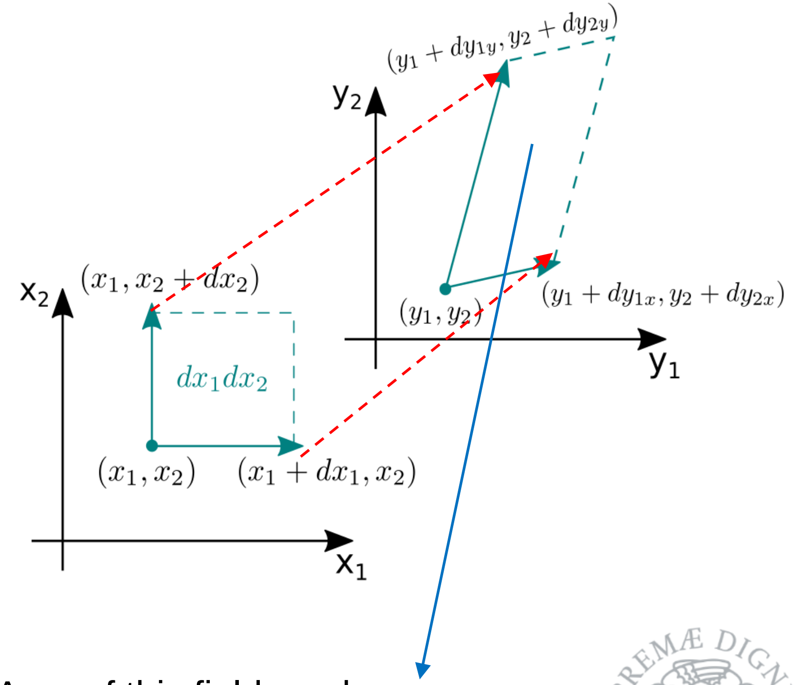
- Extend the approach to multi-dimensional case
  - $\boldsymbol{x}, \boldsymbol{z}$ vectorial RVs with density $\mathrm{P}(\boldsymbol{z})$ and $\mathrm{P}(\boldsymbol{x})$
  - Flow $\mathrm{f}(\boldsymbol{z})$ invertible and differentiable (closed under composition)
- Transformation $\mathbf{x} = \mathrm{f}(\boldsymbol{z})$ leads to the probability change

$$P(\boldsymbol{x}) = P(\boldsymbol{z}) \left| det \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}} \right|^{-1}$$

Determinant

Jacobian

Provides information on the rate of change of the volume affected by the f transformation



Area of this field can be computed with vector calculus and turns out to be the Jacobian determinant

General Multistep Case

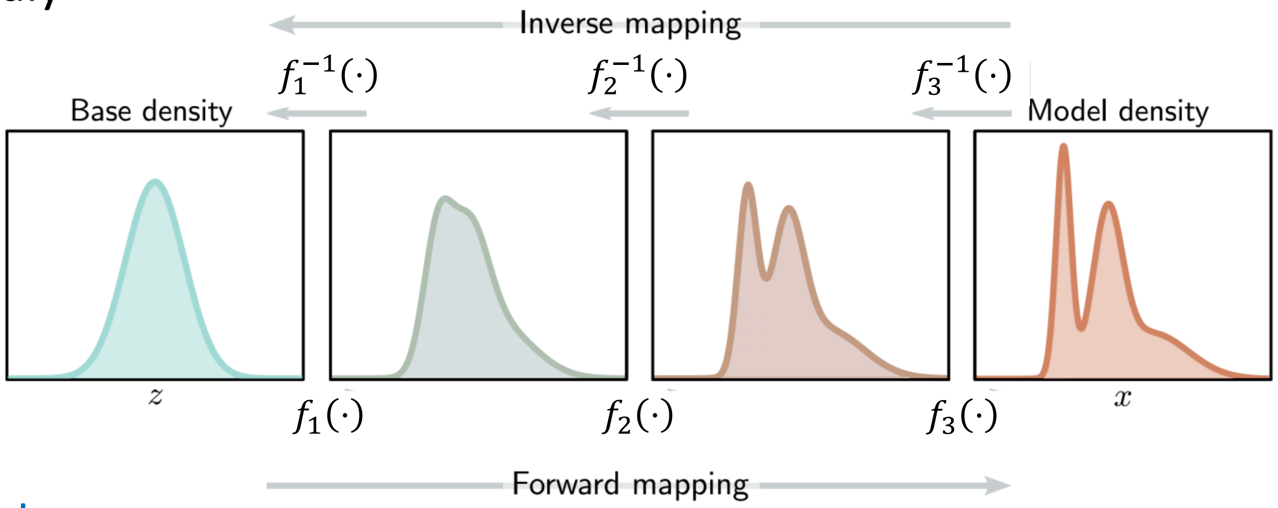
Density of the input

Used to "know" the likelihood (e.g. learning, anomaly detection)

$\boldsymbol{z}_0 = f_1^{-1}(\boldsymbol{z}_1)$

$$P(\boldsymbol{x}) = P(\boldsymbol{z}_0)\prod_{i=1}^{N}\left|det\frac{\partial f_i^{-1}(\boldsymbol{z}_i)}{\partial \boldsymbol{z}_i}\right| = P(\boldsymbol{z}_0)\prod_{i=1}^{N}\left|\det J_{\boldsymbol{z}_i}(f_i^{-1})\right|$$

$\boldsymbol{z}_N = \boldsymbol{x}$
$\boldsymbol{z}_0 = \boldsymbol{z}$



Density of the sample

Used for sampling

$\boldsymbol{z}_0 \sim P(\boldsymbol{z}_0)$

$$P(\boldsymbol{x}) = P(\boldsymbol{z}_0)\prod_{i=1}^{N}\left|det\frac{\partial f_i(\boldsymbol{z}_{i-1})}{\partial \boldsymbol{z}_{i-1}}\right|^{-1} P(\boldsymbol{z}_0)\prod_{i=1}^{N}\left|\det J_{\boldsymbol{z}_{i-1}}(f_i)\right|^{-1}$$

$\boldsymbol{z}_N = \boldsymbol{x}$
$\boldsymbol{z}_0 = \boldsymbol{z}$

Some considerations & desiderata

- Can use log densities for stability and learning

$$\log P(\boldsymbol{x}) = \log P(\boldsymbol{z}_0) + \sum_{i=1}^{N} \log\left|\det J_{\boldsymbol{z}_{i-1}}\left(f_i^{-1}\right)\right| = \log P(\boldsymbol{z}_0) - \sum_{i=1}^{N} \log\left|\det J_{\boldsymbol{z}_i}(f_i)\right|$$

- Can optimize the parameters θ of the $f_i(\cdot\,;\theta)$ by gradient based optimization of the log-likelihood above
  - f_i needs to be invertible and differentiable (and remain so throughout learning)
  - f_i composition needs to be expressive enough to map Normal into arbitrary distributions
  - Need to compute determinant easily (e.g. Jacobian diagonal or triangular matrix)
  - Computation of f_i needs to be efficient for sampling
  - Computation of f_i^{-1} needs to be efficient for learning
  - Computation of f_i needs to be stable numerically

Neural Flow Layers

Flows as invertible neural layers

- Affine flows (not sufficiently expressive)

$$\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{z}$$

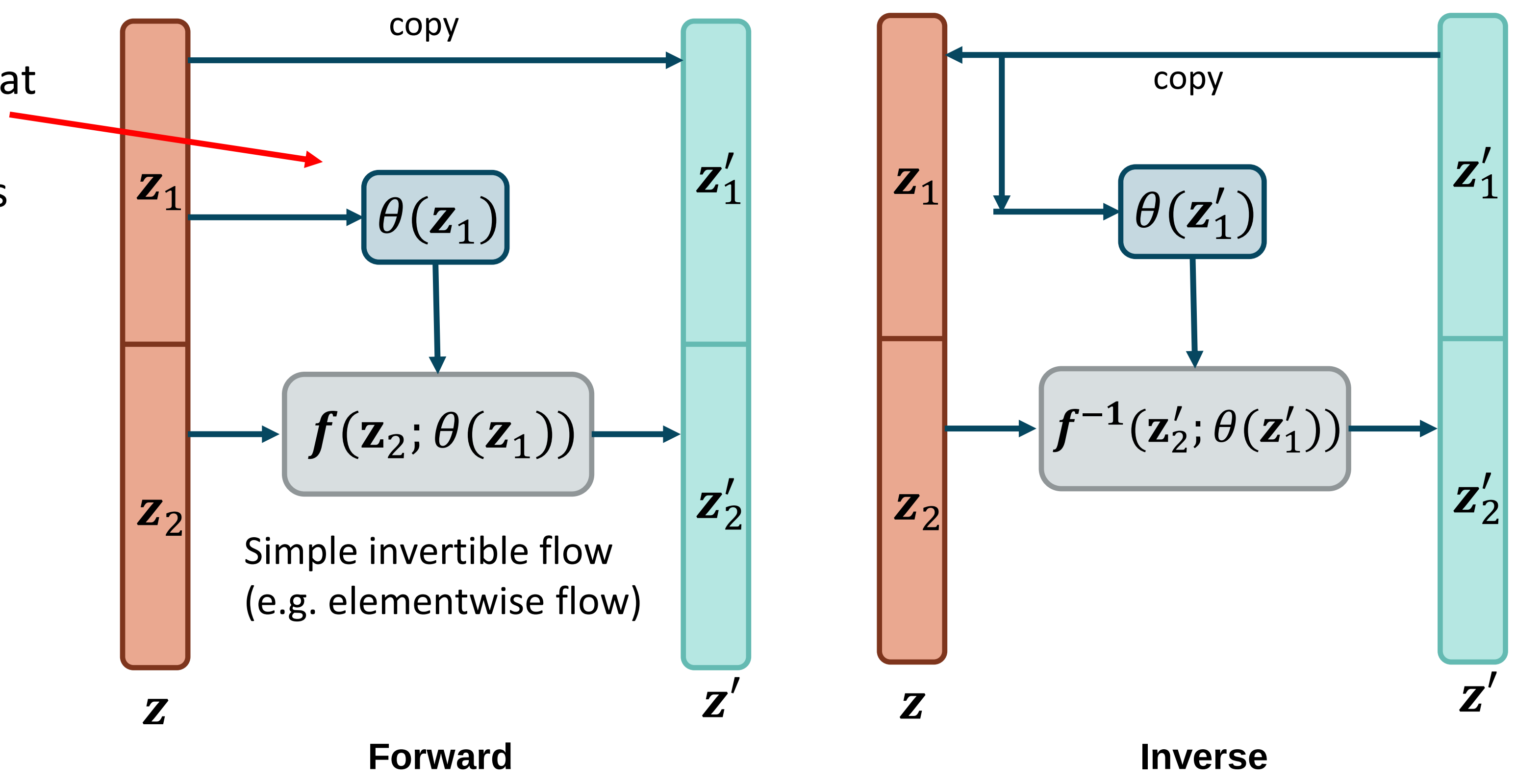
- Pointwise nonlinear where f are piecewise linear or smooth splines (nonlinear and easy to compute)

$$\boldsymbol{f}(\boldsymbol{z}) = \left[f\left(z^{(1)}, \theta\right), f\left(z^{(2)}, \theta\right), \ldots, f\left(z^{(D)}, \theta\right)\right]$$

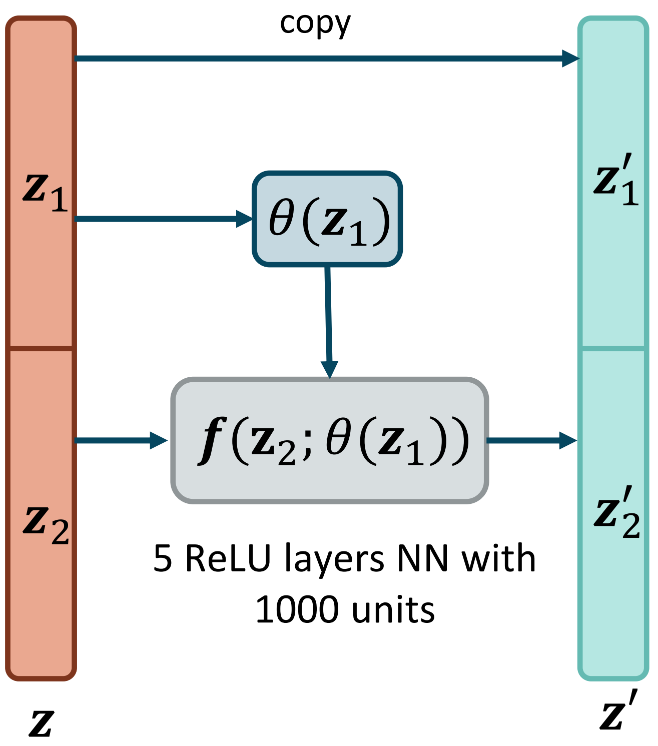
- Pointwise does not allow capturing correlations between dimensions
- Coupling flows: arguably most popular neural layer design

Coupling Flows

Neural network that generates parameters θ for the invertible function



Non-linear Independent Components Estimation (NICE)



Copy Shift

$$\mathbf{z}'_2 = \mathbf{z}_2 + \boldsymbol{\theta}(\mathbf{z}_1)$$

Inverse

$$\mathbf{z}_2 = \mathbf{z}'_2 - \boldsymbol{\theta}(\mathbf{y}_1)$$

Jacobian

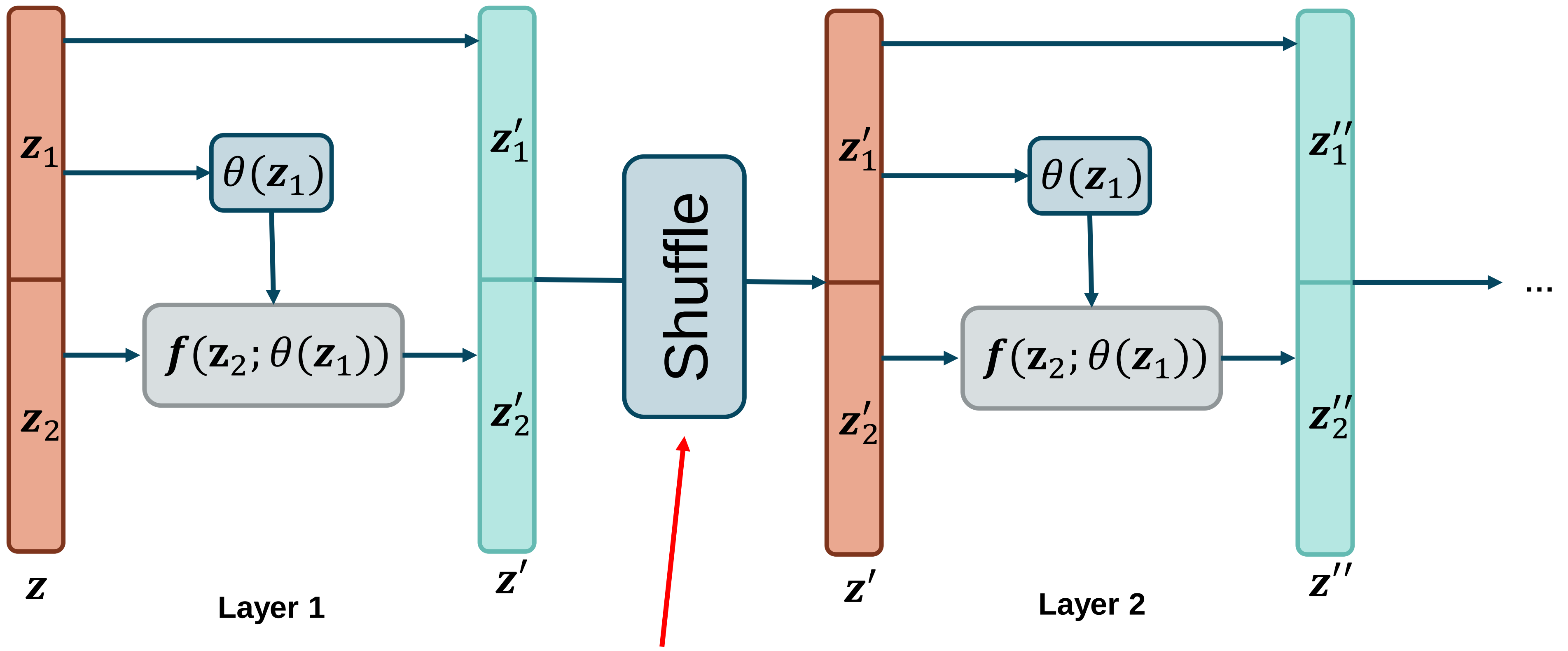
$$\begin{pmatrix} \boldsymbol{I} & \mathbf{0} \\ \frac{\partial \boldsymbol{\theta}(\mathbf{z}_1)}{\partial \mathbf{z}_1} & \boldsymbol{I} \end{pmatrix}$$

First example of coupling layer

- Simple affine transformation

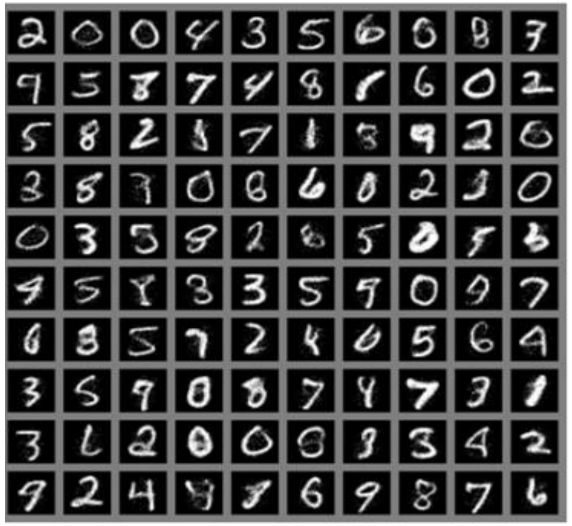
L Dinh et al, Non-linear Independent Components Estimation (NICE), ICLR-WS 2014

NICE – Stacked Coupling Flows



Random shuffle allows more general transformations than between only elements in 1st and 2nd half

(Not so) NICE Results


(a) Model trained on MNIST

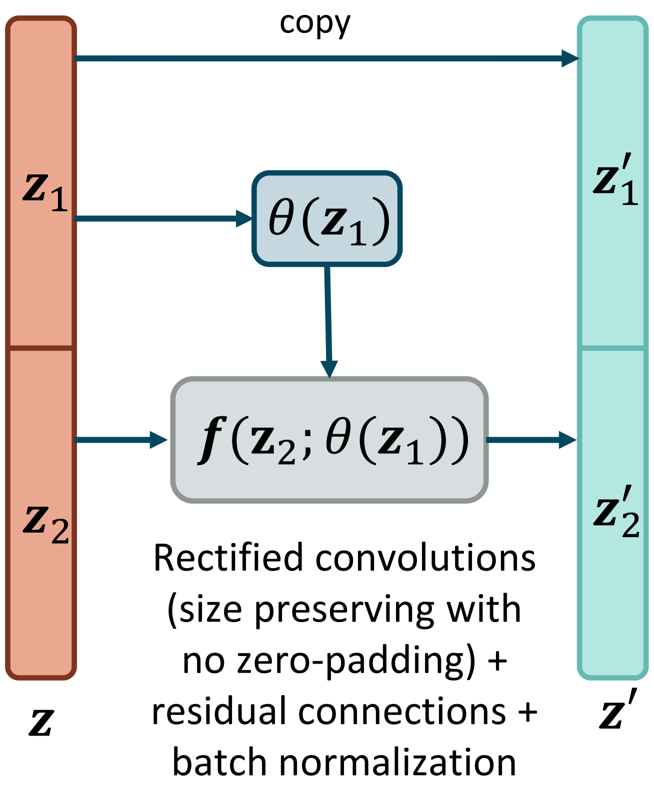

(b) Model trained on TFD


(c) Model trained on SVHN


(d) Model trained on CIFAR-10

L Dinh et al, Non-linear Independent Components Estimation (NICE), ICLR-WS 2014

RealNVP – Multiscale Nonlinear Flow



Scale, Shift

$$\mathbf{z}'_2 = \underbrace{\exp(\boldsymbol{\theta}_A(\boldsymbol{z}_1)) \odot \boldsymbol{z}_2}_{\text{Scale}} + \underbrace{\boldsymbol{\theta}_B(\boldsymbol{z}_1)}_{\text{Shift}}$$

(Forward)

Jacobian

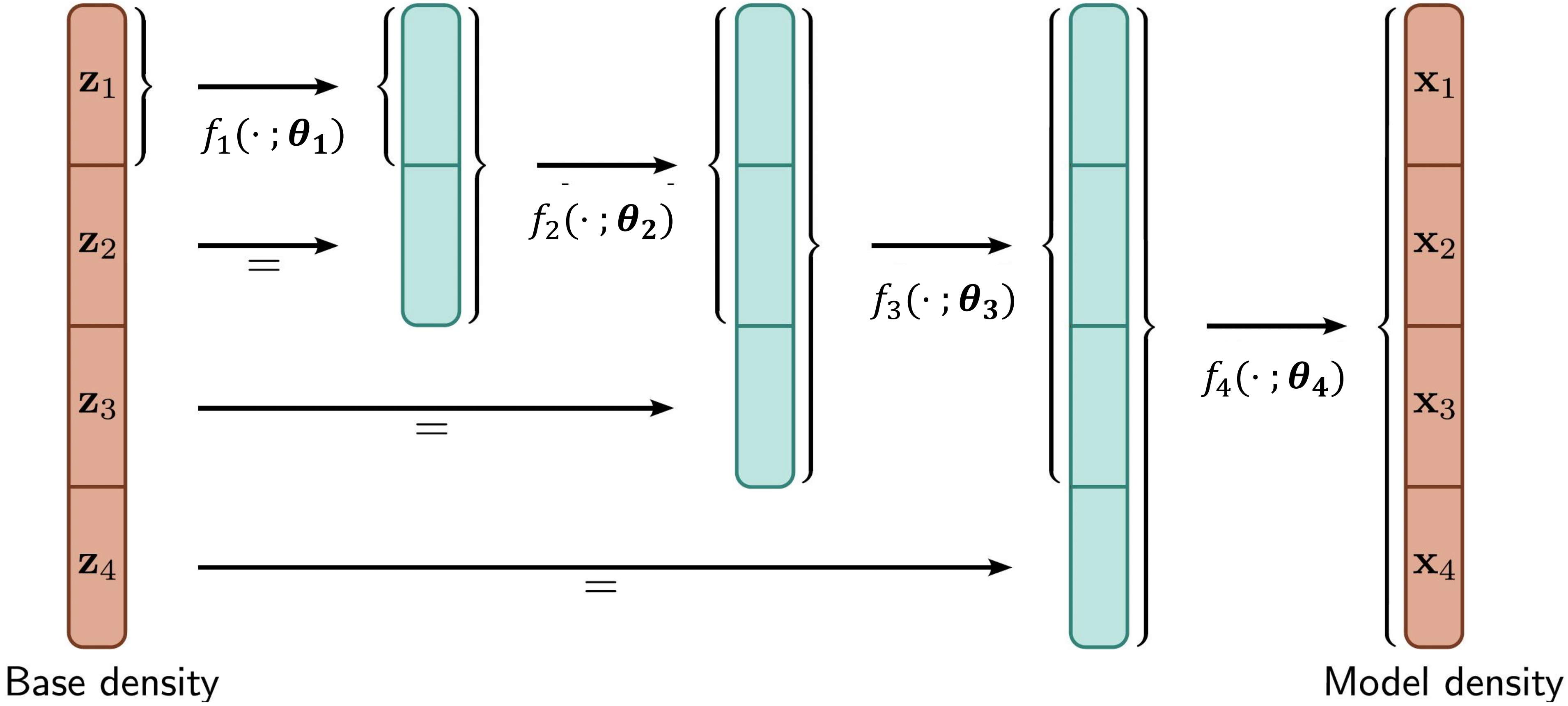
$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \frac{\partial \boldsymbol{\theta}_b(\boldsymbol{z}_1)}{\partial \boldsymbol{z}_1} & diag \exp(\boldsymbol{\theta}_A(\boldsymbol{z}_1)) \end{pmatrix}$$

$$\mathbf{z}_2 = \frac{\mathbf{z}'_2 - \boldsymbol{\theta}_B(\boldsymbol{z}'_1)}{\exp(\boldsymbol{\theta}_A(\boldsymbol{z}'_1))}$$

(Inverse)

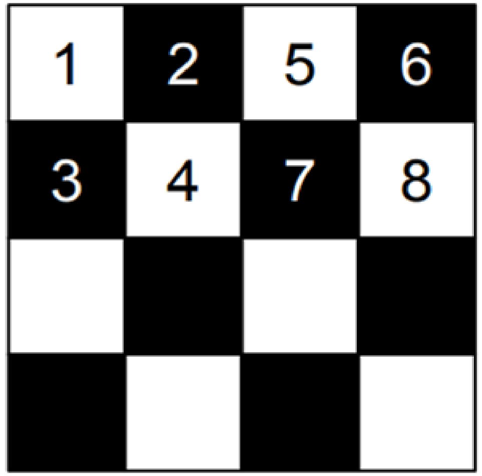
L Dinh et al, Density Estimation using real NVP, ICLR 2017

Multiscale Flows

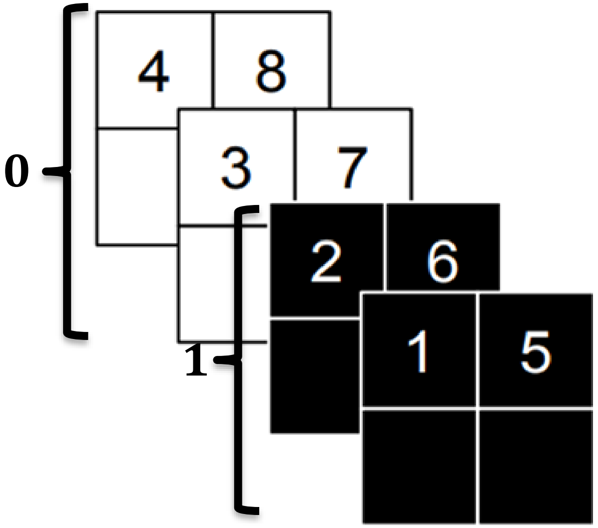


RealNVP – Masking and Squeezing

$$\boldsymbol{z}' = \boldsymbol{b} \odot \boldsymbol{z} + (1 - \boldsymbol{b}) \odot \{\exp(\boldsymbol{\theta}_A(\boldsymbol{b} \odot \boldsymbol{z})) \odot \boldsymbol{z} + \boldsymbol{\theta}_B(\boldsymbol{b} \odot \boldsymbol{z})\}$$



Partitioning
with checkerboard pattern



Squeezing
followed by channel-wise masking

$s \times s \times c$ → $\frac{s}{2} \times \frac{s}{2} \times 4c$

Squeezing

Multiscale flow implemented by alternating binary masking ($b_i \in \{0,1\}$) and squeezing

- Pixel masking before squeeze
- Channel masking after squeeze

L Dinh et al, Density Estimation using real NVP, ICLR 2017

RealNVP Results

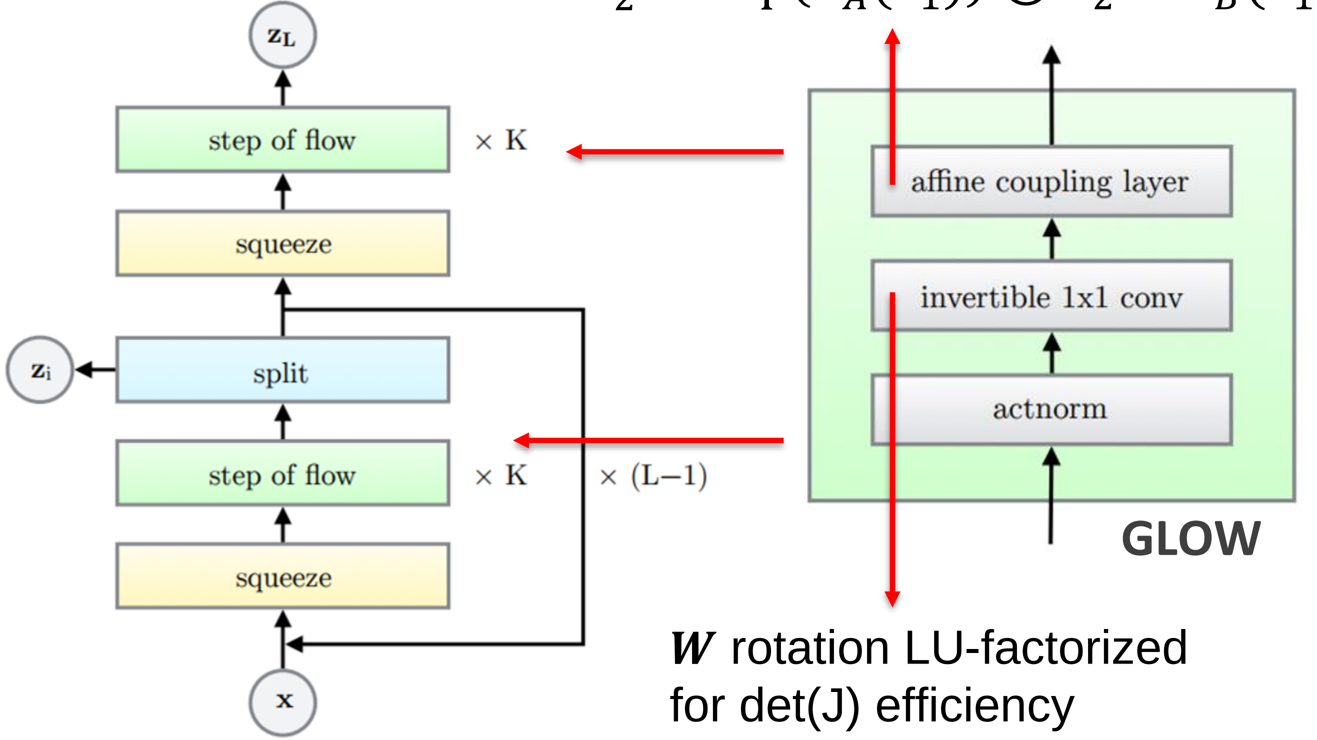
L Dinh et al, Density Estimation using real NVP, ICLR 2017

dataset

sampled

GLOW – Multiscale Coupling Flow with Invertible 1x1 Convolutions

$$\mathbf{z}_2' = \exp(\boldsymbol{\theta}_A(\boldsymbol{z}_1)) \odot \boldsymbol{z}_2 + \boldsymbol{\theta}_B(\boldsymbol{z}_1)$$

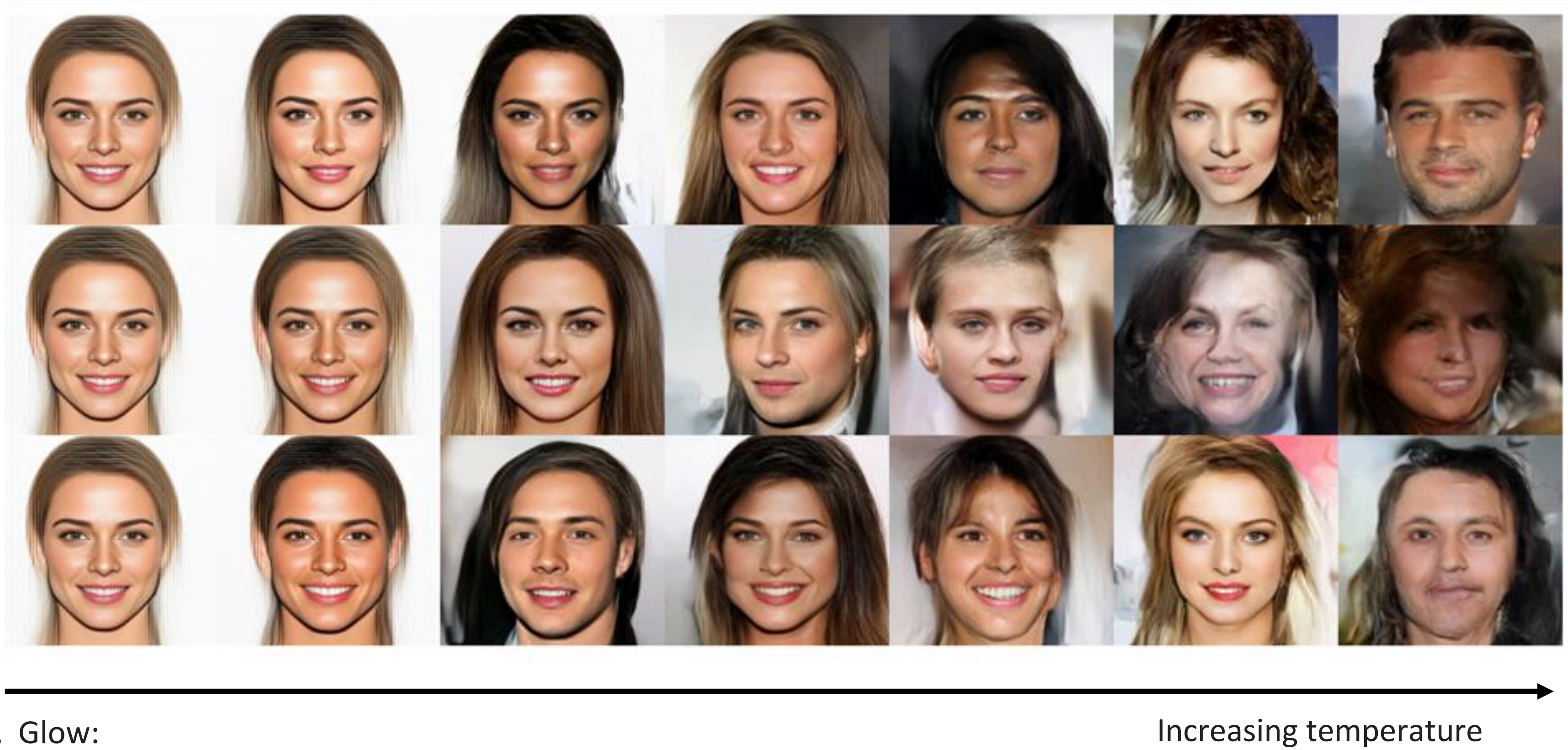


W rotation LU-factorized for det(J) efficiency

- Start with RGB tensor
- Split channels in 2 halves
- Run **1x1 convolutions parameterized with an LU decomposition** (channel mixing/permutation)
- Affine transform each spatial position in second half
- Multiscale & periodic squeeze

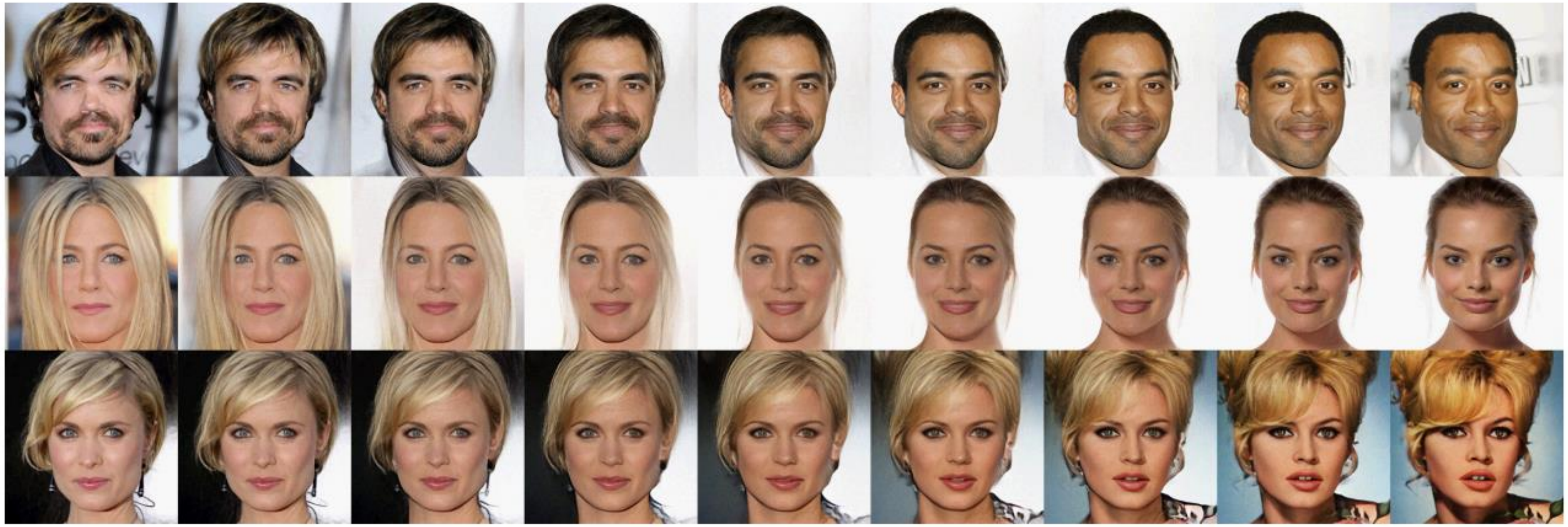
Kingma & Dhariwal, P, Glow: Generative flow with invertible 1x1 convolutions, NeurIPS 2018

GLOW Results - Sampling



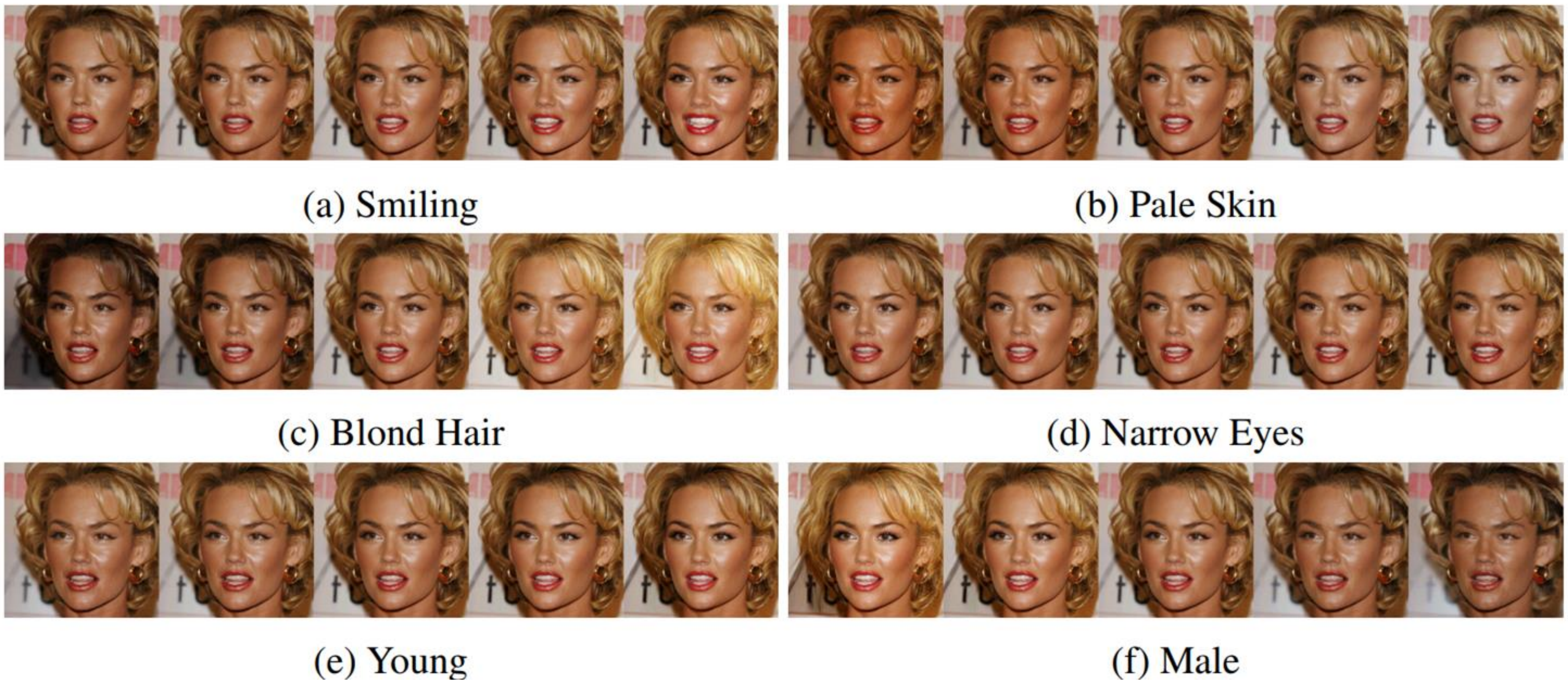
Kingma & Dhariwal, P, Glow: Generative flow with invertible 1x1 convolutions, NeurIPS 2018

GLOW Results - Interpolation



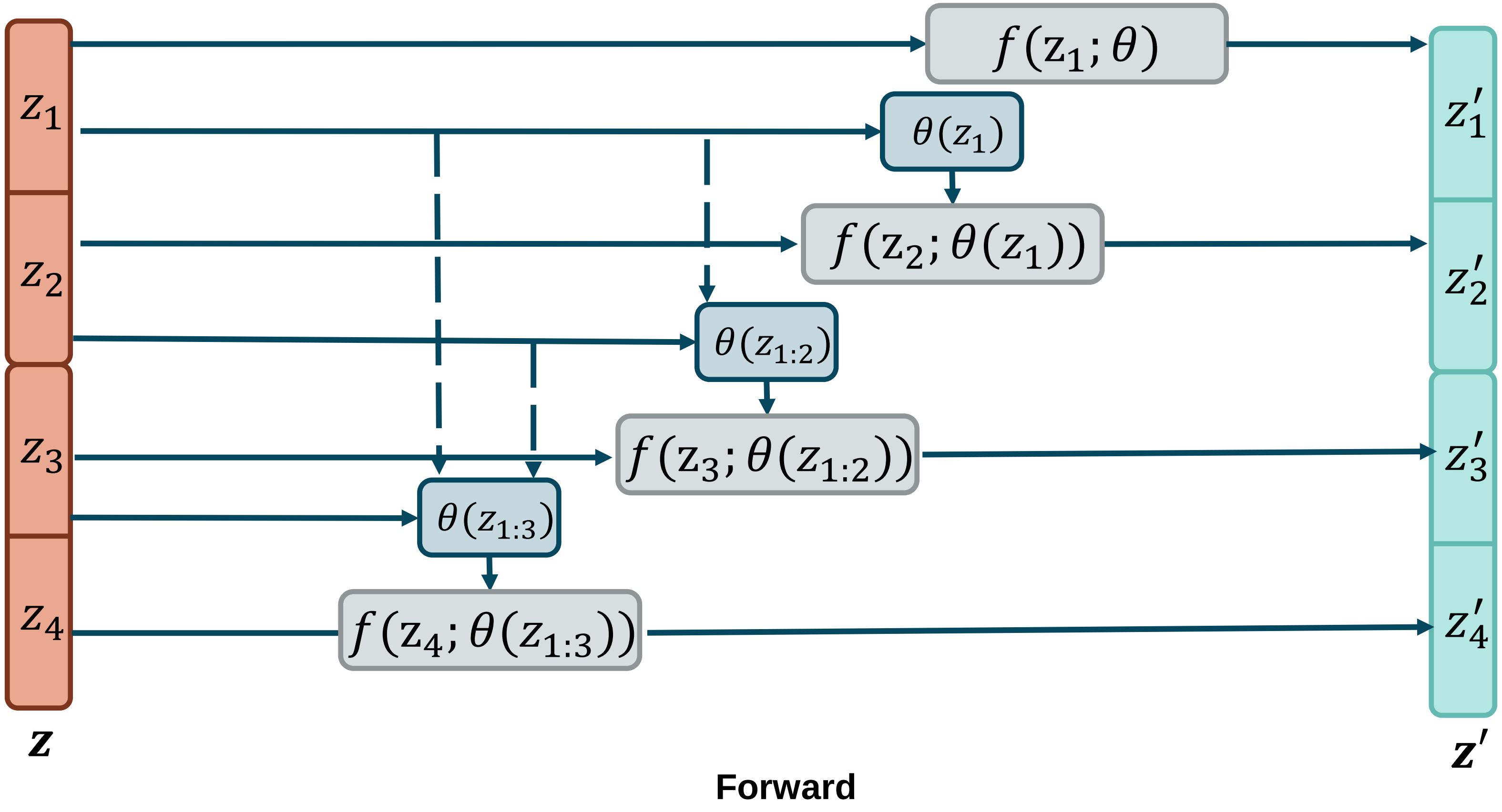
Kingma & Dhariwal, P, Glow: Generative flow with invertible 1x1 convolutions, NeurIPS 2018

GLOW Results - Manipulation



Kingma & Dhariwal, P, Glow: Generative flow with invertible 1x1 convolutions, NeurIPS 2018

Autoregressive Flows

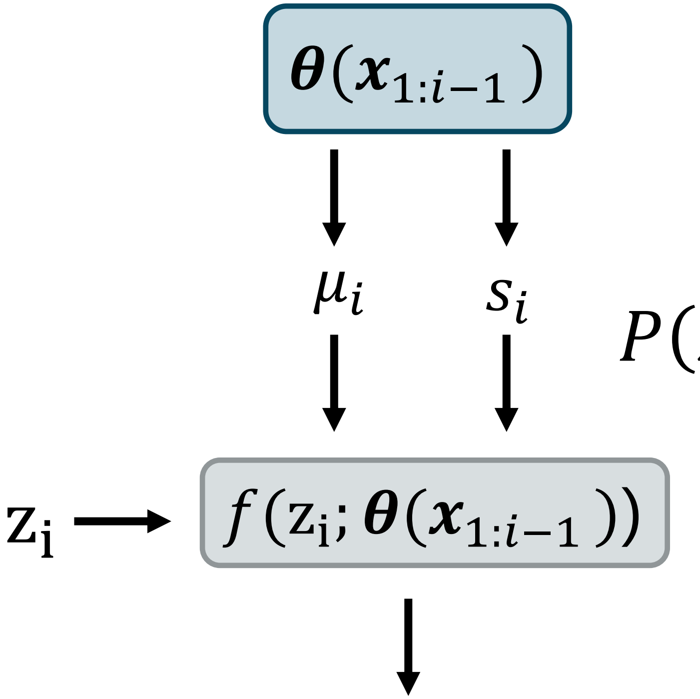


Generalization of coupling flows that treats each input dimension as a separate block

- Forward and inverse directions have different costs (parallel/sequential)

Masked Autoregressive Flow

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



Autoregressive model as a transformation $\boldsymbol{f}$ from the space of random vectors $\boldsymbol{\epsilon}$ to space of data $\boldsymbol{x}$

$$P(z_i' | \boldsymbol{x}_{1:i-1}) = \mathcal{N}(\mu_i, (\exp(s_i))^2)$$

$\boldsymbol{f}$ is easily invertible, Jacobian is triangular and easily computable determinant

$$\epsilon_i = f^{-1}(z_i') = (\mathrm{z}_\mathrm{i}' - \mu_i) \exp(-s_i)$$

$$\mathrm{z}_\mathrm{i}' = \mu_i + \epsilon_i \exp(s_i)$$

$$\epsilon_i \sim \mathcal{N}(0,1)$$

$$\left| \det\left(\frac{\partial f^{-1}}{\partial z} \right) \right| = \exp - \sum_i s_i$$

MADE Masking

M. Germain et al, MADE: Masked Autoencoder for Distribution Estimation, ICML 2015

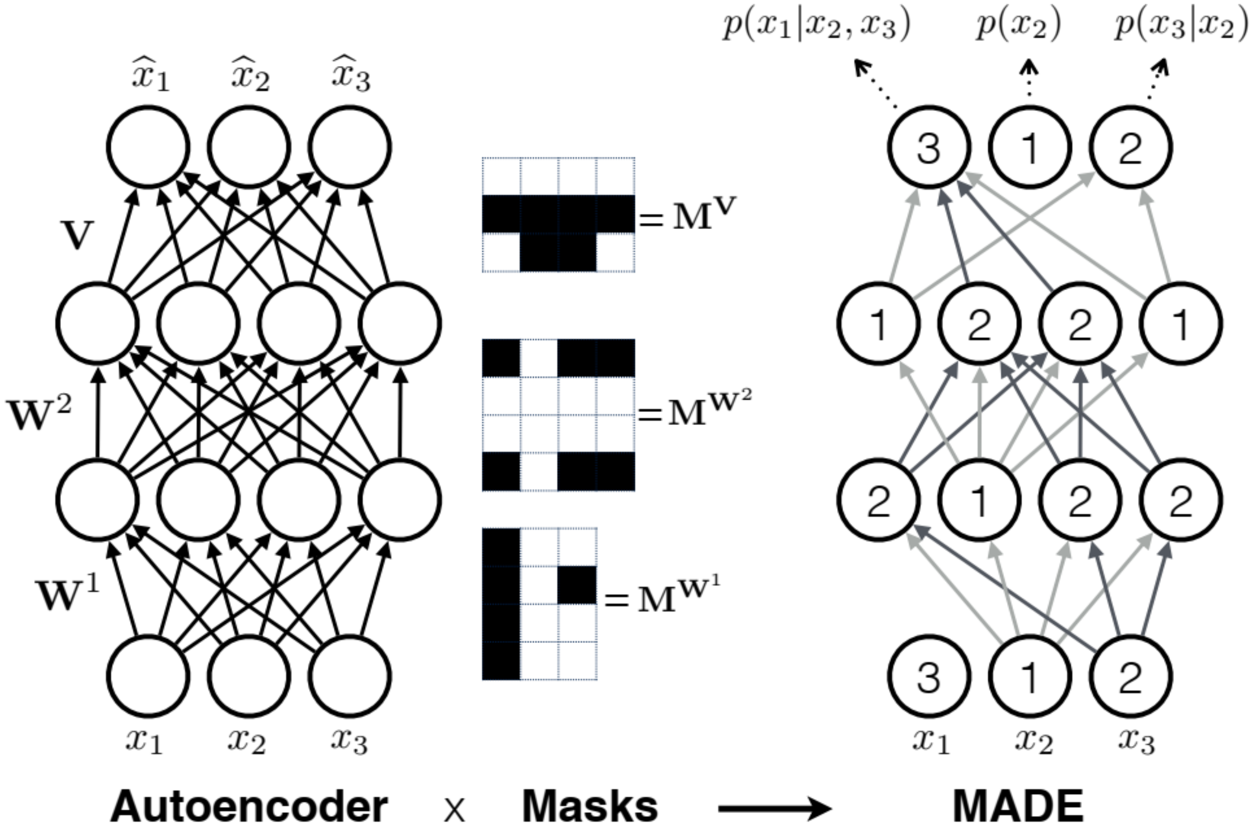
$\boldsymbol{\theta}(\boldsymbol{z}_{1:i-1})$

Efficient implementation of $\boldsymbol{\theta}(\cdot)$ using a feedforward neural network that computes all ϵ_i, s_i terms in a single forward pass (instead of recursively)

$$h(\boldsymbol{x}) = g(\boldsymbol{b} + (\boldsymbol{W} \odot \boldsymbol{M}^W)\boldsymbol{x})$$

$$M_{k,d}^W = \mathbb{I}(m(k) \geq d)$$

Integer unique identifier of a hidden neuron



Wrap-Up

Implementations & Libraries

- Normalizing flows are natively supported by Tensorflow (through the TF Probability module)
  - tf.probability.distribution (for base distributions)
  - tf.probability.bijector (for predefined layers, e.g. masked autoregressive)
  - Chain() (to chain bijectors and compose complex modules)
  - You can of course define you own bijectors according to a template
- Normflows - PyTorch package for Normalizing Flows
- Flowtorch – PyRo based Pytorch library for Normalizing Flows

Take Home Messages

- Normalizing flows as an effective and tractable way to generate new samples (efficient) and to evaluate the likelihood of samples (not so efficient)
- Universality property - The flow can learn any target density to any required precision given sufficient capacity and data
  - Flow can be used to generate samples that approximate a density easy to evaluate but difficult to sample
- Normalizing flow design needs to take care of
  - Keeping flow invertible and efficient
  - Making the determinant of the Jacobian easy to compute
- Normalizing flows can be made continuous using a neural ODE scheme

Generative DL Summary (I) - Sampling

Generative adversarial networks

- Adversarial learning as a general and effective principle
- Effective and efficient in generating high quality samples
- Do not learn sample likelihood
- GANs generally more unstable than other deep generative models

Variational Autoencoders

- ELBO trained and imposing standard normal structure
- Encoder-Decoder scheme with latent variables of any dimension
- Can be integrated with adversarial, diffusion and flow approaches
- Useful to study representation learning aspects but bad at sampling

Diffusion models

- Generate data from noise through a learned incremental denoising with fixed steps
- A hierarchical VAE with fixed encoder and no explicit density
- Easy to train, scalable to parallel hardware and generate high quality samples (though can be slow)

Generative DL Summary (II) - Density

Autoregressive

- Generate data by sampling based on the chain rule factorization (e.g. PixelRNN)
- Effective density estimators, but sampling is very costly and impractical for high dimensional data

Normalizing flows

- Can learn arbitrary distributions for high-dimensional data in a tractable way using change of variable
- Can handle efficient sampling and interpolation
- Generalize and make tractable autoregressive modeling
- Require bijective transformations and “well-behaved” Jacobians

Energy-based models

- Neural networks trained in a generative fashion as Markov Random Fields
- Does not require that all components are distributions
- Need to be trained by MCMC (due to the usual partition function term)

Coming-up next

- Thursday 02/05 – Fundamentals of deep learning for graphs
  - Processing graph structured data in neural network
  - Learning tasks on graphs
  - Foundational neural models for graphs
- (Will have a follow-up on advanced models)