Normalizing Flow Models

2022
Multicampus

Il Gu Yi
ModuLabs, Research Scientist
Soochul Park
Vivestudios, Research Scientist

V.1.2

Taxonomy of Generative Models

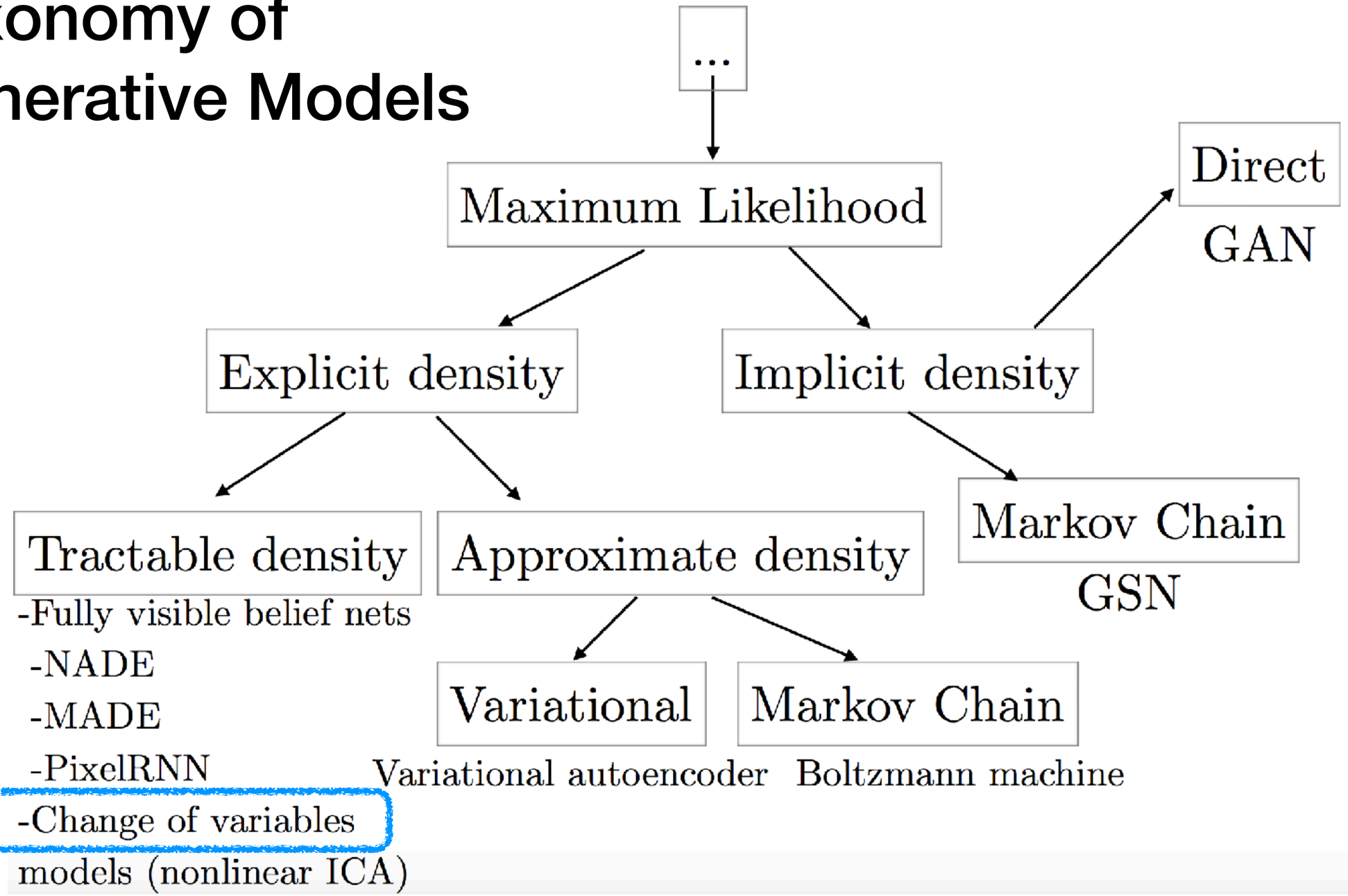


Figure credit: Ian Goodfellow, NIPS 2016 Tutorial: Generative Adversarial Networks

Recap of AutoRegressive and Likelihood based Models

- Model families:
  - Autoregressive Models: $p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i|\mathbf{x}_{<i})$
  - Variational Autoencoders: $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z}$
- Autoregressive models provide tractable likelihoods but no direct mechanism for learning features
- Variational autoencoders can learn feature representations (via latent variables z) but have intractable marginal likelihoods
- **Key question**: Can we design a latent variable model with tractable likelihoods? Yes!

Simple Prior to Complex Data Distributions

- Desirable properties of any model distribution:
  - Analytic density
  - Easy-to-sample
- Many simple distributions satisfy the above properties e.g., Gaussian, uniform distributions
- Unfortunately, data distributions could be much more complex (multi-modal)
- **Key idea**: Map simple distributions (easy to sample and evaluate densities) to complex distributions (learned via data) using change of variables

Change of Variables

- Let Z be a uniform random variable $\mathcal{U}[0, 2]$ with density p_Z
  - What is $p_Z(1)$?: 1/2
- Let $X = 4Z$, and let p_X be its density
  - What is $p_X(4)$?
- $p_X(4) = p(X = 4) = p(4Z = 4) = p(Z = 1) = p_Z(1) = 1/2$ **No**
- Clearly, X is uniform in [0, 8], so $p_X(4) = 1/8$

Change of Variables (Cont.)

- **Change of variables (1D case)**: If $X = f(Z)$ and $f(\cdot)$ is monotone with inverse $Z = f^{-1}(X) = h(X)$, then:

$$p_X(x) = p_Z(h(x))|h'(x)|$$

- Previous example: If $X = 4Z$ and $Z \sim \mathcal{U}[0, 2]$, what is $p_X(4)$?
- Note that $h(X) = X/4$
- $p_X(4) = p_Z(1)h'(4) = 1/2 \times 1/4 = 1/8$

Geometry: Determinants and volumes

- Let Z be a uniform random vector in $[0,1]^n$
- Let $X = AZ$ for a square invertible matrix A, with inverse $W = A^{-1}$. How is X distributed?
- Geometrically, the matrix A maps the unit hypercube $[0,1]^n$ to a parallelotope
- Hypercube and parallelotope are generalizations of square/cube and parallelogram/parallelopiped to higher dimensions

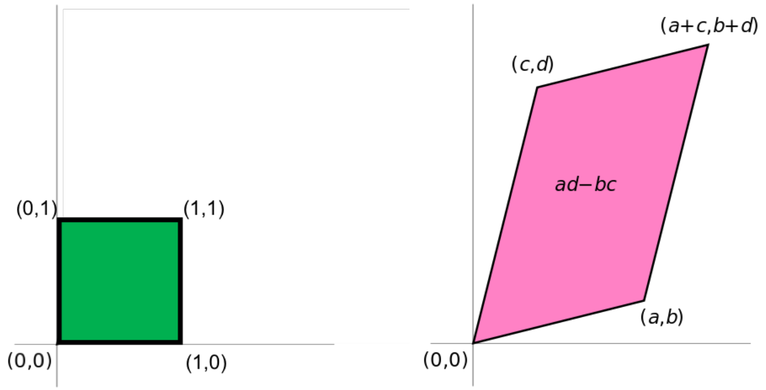


Figure: The matrix $A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ maps a unit square to a parallelogram

Geometry: Determinants and Volumes

- The volume of the parallelotope is equal to the determinant of the transformation A

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

- X is uniformly distributed over the parallelotope. Hence, we have

$$\begin{aligned} p_X(\mathbf{x}) &= p_Z(W\mathbf{x})|\det(W)| \\ &= p_Z(W\mathbf{x})/|\det(A)| \end{aligned}$$

Generalized change of variables

- For linear transformations specified via A, change in volume is given by the determinant of A
- For non-linear transformations $\mathbf{f}(\cdot)$, the *linearized* change in volume is given by the determinant of the Jacobian of $\mathbf{f}(\cdot)$.
- **Change of variables (General case)**: The mapping between Z and X, given by $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$, is invertible such that $X = \mathbf{f}(Z)$ and $Z = \mathbf{f}^{-1}(X)$.

$$p_X(\mathbf{x}) = p_Z\left(\mathbf{f}^{-1}(\mathbf{x})\right) \left| \det \left(\frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

- Note 1: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension. For example, if $\mathbf{x} \in \mathbb{R}^n$ then $\mathbf{z} \in \mathbb{R}^n$
- Note 2: For any invertible matrix A, $det(A^{-1}) = det(A)^{-1}$

$$p_X(\mathbf{x}) = p_Z\left(\mathbf{z}\right) \left| \det \left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1}$$

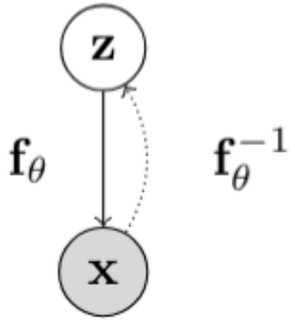
Two Dimensional Example

- Let Z_1 and Z_2 be continuous random variables with joint density p_{Z_1,Z_2}.
- Let $u = (u_1, u_2)$ be a transformation
- Let $v = (v_1, v_2)$ be the inverse transformation
- Let $X_1 = u_1(Z_1, Z_2)$ and $X_2 = u_2(Z_1, Z_2)$ Then, $Z_1 = v_1(X_1, X_2)$ and $Z_2 = v_2(X_1, X_2)$

$$
\begin{aligned}
& p_{X_1,X_2}(x_1, x_2) \\
&= p_{Z_1,Z_2}(v_1(x_1, x_2), v_2(x_1, x_2)) \left| \det \begin{pmatrix} \frac{\partial v_1(x_1,x_2)}{\partial x_1} & \frac{\partial v_1(x_1,x_2)}{\partial x_2} \\ \frac{\partial v_2(x_1,x_2)}{\partial x_1} & \frac{\partial v_2(x_1,x_2)}{\partial x_2} \end{pmatrix} \right| \text{(inverse)} \\
&= p_{Z_1,Z_2}(z_1, z_2) \left| \det \begin{pmatrix} \frac{\partial u_1(z_1,z_2)}{\partial z_1} & \frac{\partial u_1(z_1,z_2)}{\partial z_2} \\ \frac{\partial u_2(z_1,z_2)}{\partial z_1} & \frac{\partial u_2(z_1,z_2)}{\partial z_2} \end{pmatrix} \right|^{-1} \text{(forward)}
\end{aligned}
$$

Normalizing flow models

- Consider a directed, latent-variable model over observed variables X and latent variables Z
- In a **normalizing flow model**, the mapping between Z and X, given by $\mathbf{f}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$, is deterministic and invertible such that $X = \mathbf{f}_\theta(Z)$ and $Z = \mathbf{f}_\theta^{-1}(X)$



- Using change of variables, the marginal likelihood $p(\mathbf{x})$ is given by

$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

- Note: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension.

A Flow of Transformations

Normalizing: Change of variables gives a normalized density after applying an invertible transformation

Flow: Invertible transformations can be composed with each other

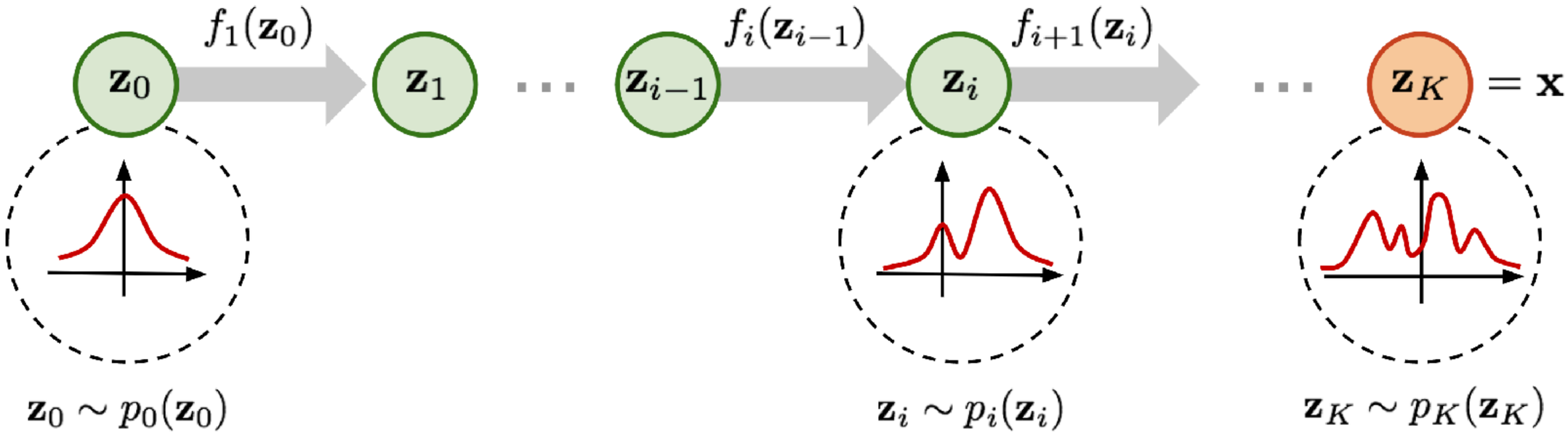
$$\mathbf{x} \triangleq \mathbf{z}_M = \mathbf{f}_\theta^M \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^M(\mathbf{f}_\theta^{M-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0)$$

- Start with a simple distribution for $\mathbf{z}_0$ (e.g., Gaussian)
- Apply a sequence of M invertible transformations

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

(determininant of product equals product of determinants)

Normalizing Flow



Transforms a simple distribution into a complex one by applying a sequence of invertible transformation functions

Figure credit: https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Learning and Inference

- Learning via **maximum likelihood** over the dataset $\mathcal{D}$

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

- **Exact likelihood evaluation** via inverse tranformation $\mathbf{x} \mapsto \mathbf{z}$ and change of variables formula
- **Sampling** via forward transformation $\mathbf{z} \mapsto \mathbf{x}$

$$\mathbf{z} \sim p_Z(\mathbf{z}) \quad \mathbf{x} = \mathbf{f}_\theta(\mathbf{z})$$

- **Latent representations** inferred via inverse transformation (no inference network required!)

$$\mathbf{z} = \mathbf{f}_\theta^{-1}(\mathbf{x})$$

Triangular Jacobian

$$\mathbf{x} = (x_1, \cdots, x_n) = \mathbf{f}(\mathbf{z}) = (f_1(\mathbf{z}), \cdots, f_n(\mathbf{z}))$$

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial z_1} & \cdots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$$

Suppose $x_i = f_i(\mathbf{z})$ only depends on $\mathbf{z}_{\leq i}$. Then

$$J = \frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & 0 \\ \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial z_1} & \cdots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$$

has lower triangular structure. Determinant can be computed in **linear time**. Similarly, the Jacobian is upper triangular if x_i only depends on $\mathbf{z}_{\geq i}$

Designing Invertible Transformations

- NICE or Nonlinear Independent Components Estimation (Dinh et al., 2014) composes two kinds of invertible transformations: additive coupling layers and rescaling layers
- Real-NVP (Dinh et al., 2017)
- Inverse Autoregressive Flow (Kingma et al., 2016)
- Masked Autoregressive Flow (Papamakarios et al., 2017)
- Glow (Kingma et al., 2018)

Non-linear Independent Components Estimation (NICE)

- Maximum likelihood

$$\log(p_X(x)) = \log(p_H(f(x))) + \log\left(\left|\det\left(\frac{\partial f(x)}{\partial x}\right)\right|\right)$$

- If the prior distribution is factorial (i.e. with independent dimensions), then we obtain the following formula

$$\log(p_X(x)) = \sum_{d=1}^{D} \log(p_{H_d}(f_d(x))) + \log\left(\left|\det\left(\frac{\partial f(x)}{\partial x}\right)\right|\right)$$

NICE - General Coupling Layer

- **Bijective transformation** with triangular Jacobian (therefore tractable Jacobian determinant)

- Let $x \in \mathcal{X}, I_1, I_2$ a partition of [1, D]

  - such that $d = |I_1|$ and m is a function defined on $\mathbb{R}^d$

  - define $y = (y_{I_1}, y_{I_2})$ $$\begin{aligned} y_{I_1} &= x_{I_1} \\ y_{I_2} &= g(x_{I_2}; m(x_{I_1})) \end{aligned}$$

- where $g : \mathbb{R}^{D-d} \times m(\mathbb{R}^d) \to \mathbb{R}^{D-d}$ is the coupling law

NICE - Invertible map

- Jacobinan:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_{I_2}}{\partial x_{I_1}} & \frac{\partial y_{I_2}}{\partial x_{I_2}} \end{bmatrix}$$

  - where I_d is the identity matrix of size d
  - That means that $$\det \frac{\partial y}{\partial x} = \det \frac{\partial y_{I_2}}{\partial x_{I_2}}$$

- Inverse mapping

$$\begin{aligned} x_{I_1} &= y_{I_1} \\ x_{I_2} &= g^{-1}(y_{I_2}; m(y_{I_1})) \end{aligned}$$

- Call such a transformation a coupling layer with coupling function m

NICE - Additive Coupling Layer

- Additive coupling law: $g(a; b) = a + b$
  - where $a = x_{I_2}$ and $b = m(x_{I_1})$
- Forward

$$\begin{aligned} y_{I_1} &= x_{I_1} \\ y_{I_2} &= x_{I_2} + m(x_{I_1}) \end{aligned}$$

Inverse

$$\begin{aligned} x_{I_1} &= y_{I_1} \\ x_{I_2} &= y_{I_2} - m(y_{I_1}) \end{aligned}$$

- m can be a neural network with d input units and $D - d$ output units

NICE - Jacobian of Additive Coupling Layer

- Additive coupling law: $g(a; b) = a + b$
  - where $a = x_{I_2}$ and $b = m(x_{I_1})$
- Forward

$$\begin{aligned} y_{I_1} &= x_{I_1} \\ y_{I_2} &= x_{I_2} + m(x_{I_1}) \end{aligned}$$

Inverse

$$\begin{aligned} x_{I_1} &= y_{I_1} \\ x_{I_2} &= y_{I_2} - m(y_{I_1}) \end{aligned}$$

- Jacobian (**Volume preserving transformation**)

$$\det \frac{\partial y_{I_2}}{\partial x_{I_2}} = 1 \qquad \therefore \det \frac{\partial y}{\partial x} = \det \begin{bmatrix} I_d & 0 \\ \frac{\partial y_{I_2}}{\partial x_{I_1}} & \frac{\partial y_{I_2}}{\partial x_{I_2}} \end{bmatrix} = 1$$

NICE - Rescaling Layers

- As each additive coupling layers has unit Jacobian determinant (i.e. is volume preserving), their composition will necessarily have unit Jacobian determinant too
  - Scaling matrix S as the top layer $S_{ii} : x_i \rightarrow S_{ii} x_i$ where $S_{ii} > 0$
- This allows the learner to give more weight (i.e. model more variation) on some dimensions and less in others
- NICE criterion

$$\log(p_X(x)) = \sum_{i=1}^{D} \log(p_{H_i}(f_i(x))) + \log\left(|S_{ii}|\right)$$

NICE - Prior Distribution

- Prior distribution to be factorial

$$p_H(h) = \prod_{d=1}^{D} p_{H_d}(h_d)$$

- Gaussian distribution

$$\log(p_H) = -\frac{1}{2}(h_d^2 + \log(2\pi))$$

- Logistic distribution

$$\log(p_H) = -\log(1 + \exp(h_d)) - \log(1 + \exp(-h_d))$$

Results via NICE

Figure credit: L. Dinh, et. al., NICE: Non-Linear Independent Components Estimation

Real NVP - Coupling Layer

- Forward

$$
\begin{aligned}
y_{I_1} &= x_{I_1} \\
y_{I_2} &= x_{I_2} \odot \exp\left(s(x_{I_1})\right) + t(x_{I_1})
\end{aligned}
$$

- Backward

$$
\begin{aligned}
x_{I_1} &= y_{I_1} \\
x_{I_2} &= (y_{I_2} - t(y_{I_1})) \odot \exp(-s(y_{I_1}))
\end{aligned}
$$

- t and s are functions: $\mathbb{R}^d \rightarrow \mathbb{R}^{D-d}$

- cf) NICE: coupling layer

$$
\begin{aligned}
y_{I_1} &= x_{I_1} \\
y_{I_2} &= x_{I_2} + m(x_{I_1})
\end{aligned}
$$

Real NVP - Jacobian of Additive Coupling Layer

- Jacobian

$$J = \frac{\partial y}{\partial x} = \begin{pmatrix} I_d & 0 \\ \frac{\partial y_{I_2}}{\partial x_{I_1}} & \text{diag}\ (\exp[s(x_{I_1})]) \end{pmatrix}$$

- Determinant

$$\det(J) = \prod_{i=d+1}^{D} \exp[s(x_{I_1})_i] = \exp\left(\sum_{i=d+1}^{D} s(x_{I_1})_i\right)$$

- **Non-volume preserving(NVP)** transformation
  - $\det(J) \neq 1$

Advantages of Real NVP

- Jacobian

$$J = \frac{\partial y}{\partial x} = \begin{pmatrix} I_d & 0 \\ \frac{\partial y_{I_2}}{\partial x_{I_1}} & \text{diag}\ (\exp[s(x_{I_1})]) \end{pmatrix}$$

- Determinant

$$\det(J) = \prod_{i=d+1}^{D} \exp[s(x_{I_1})_i] = \exp\left(\sum_{i=d+1}^{D} s(x_{I_1})_i\right)$$

- Advantages
  - **backward** operation does not require the inverse s and t
  - **calculating** det(J) does not involve the Jacobian s and t

Results via RealNVP

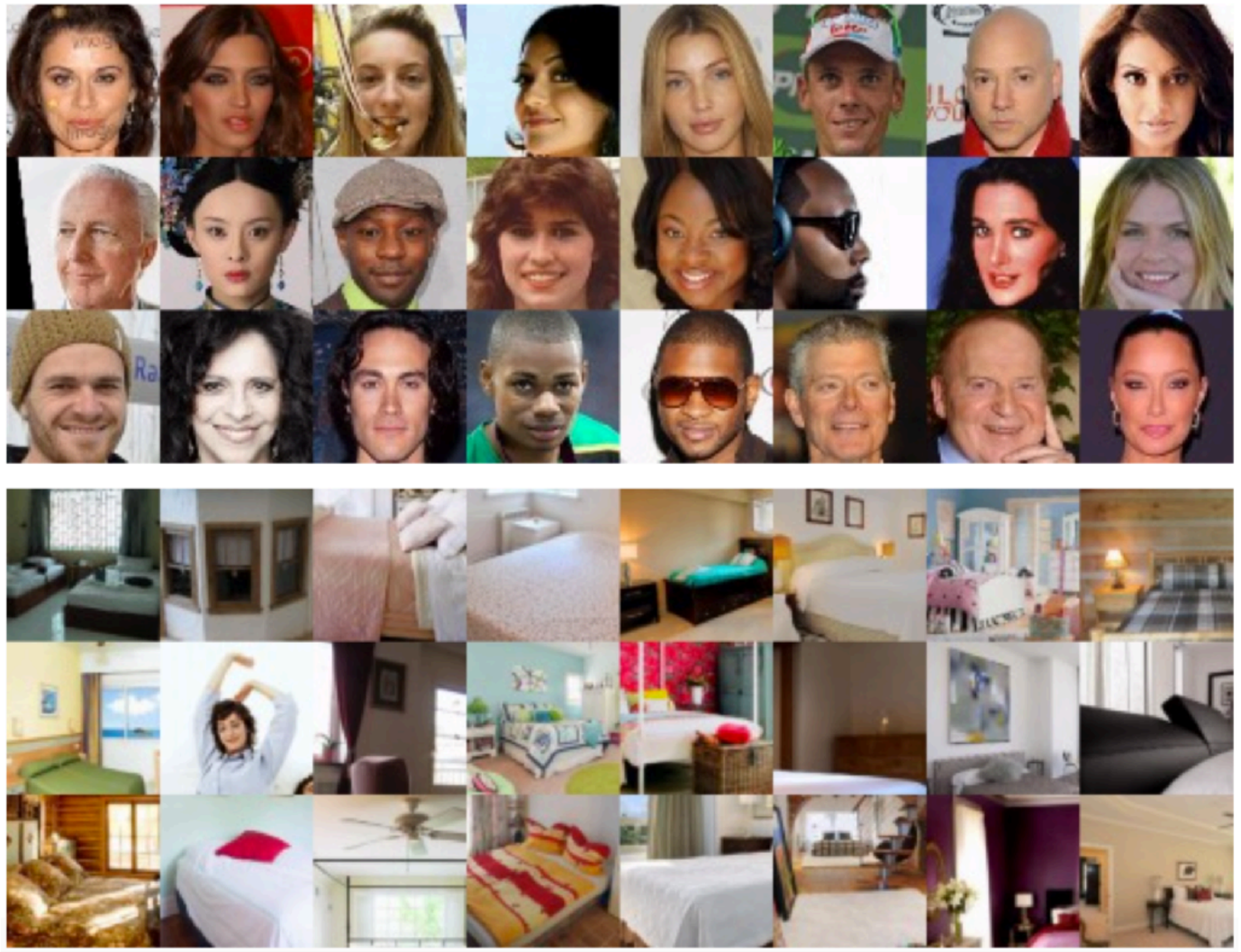


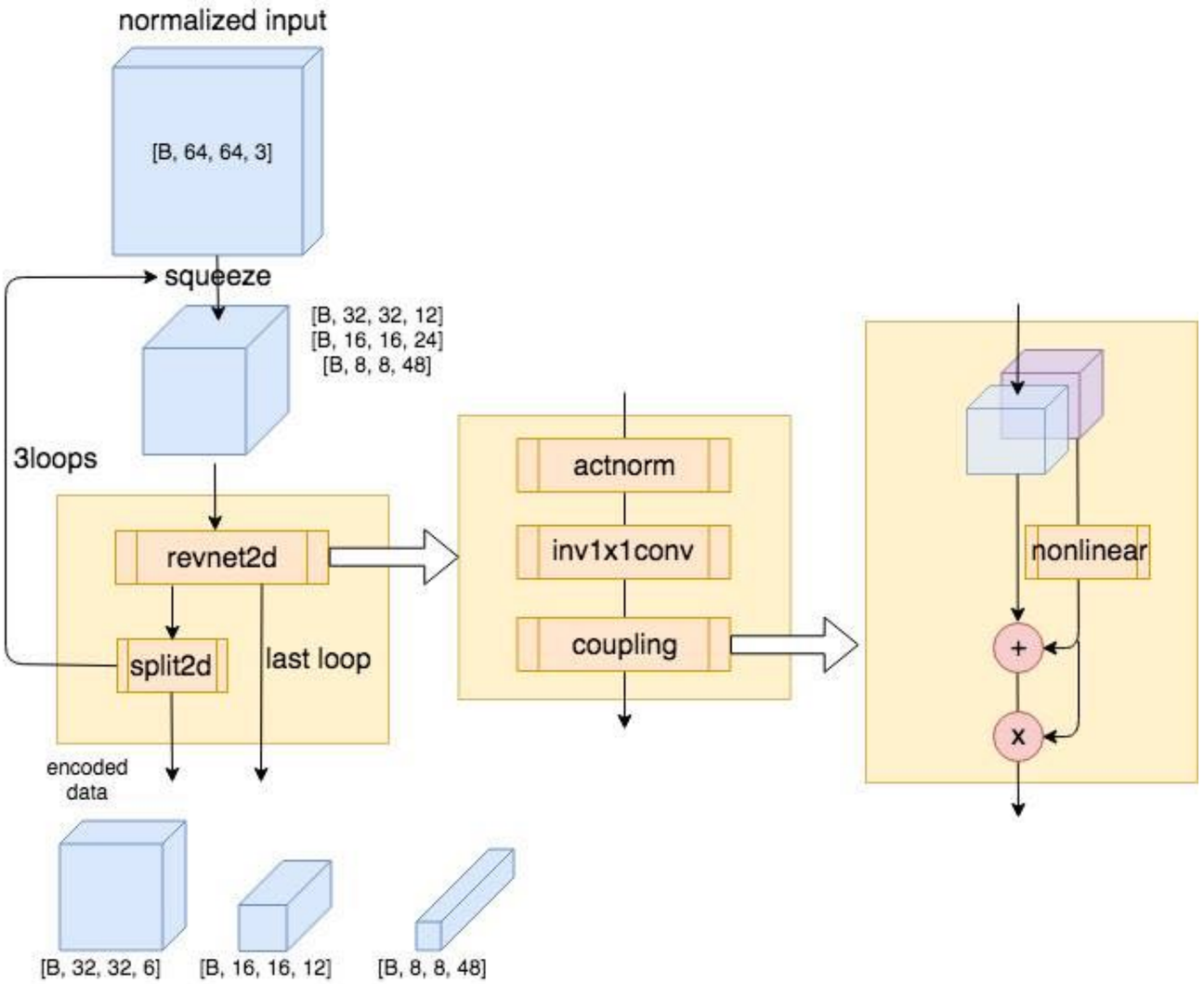
Figure credit: L. Dinh, et. al., Density estimation using Real NVP

Interpolations via RealNVP

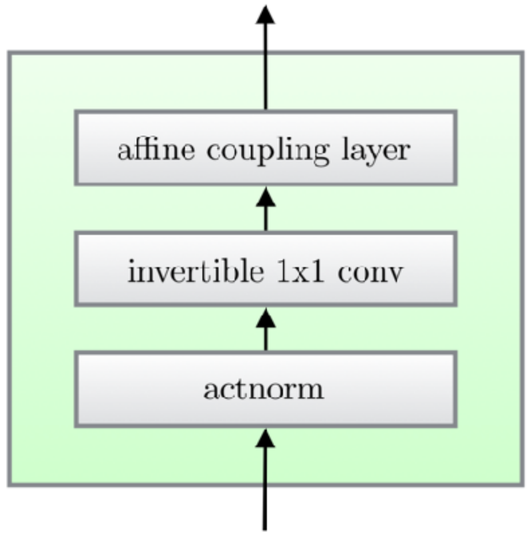
Figure credit: L. Dinh, et. al., Density estimation using Real NVP

GLOW

GLOW networks



GLOW modules



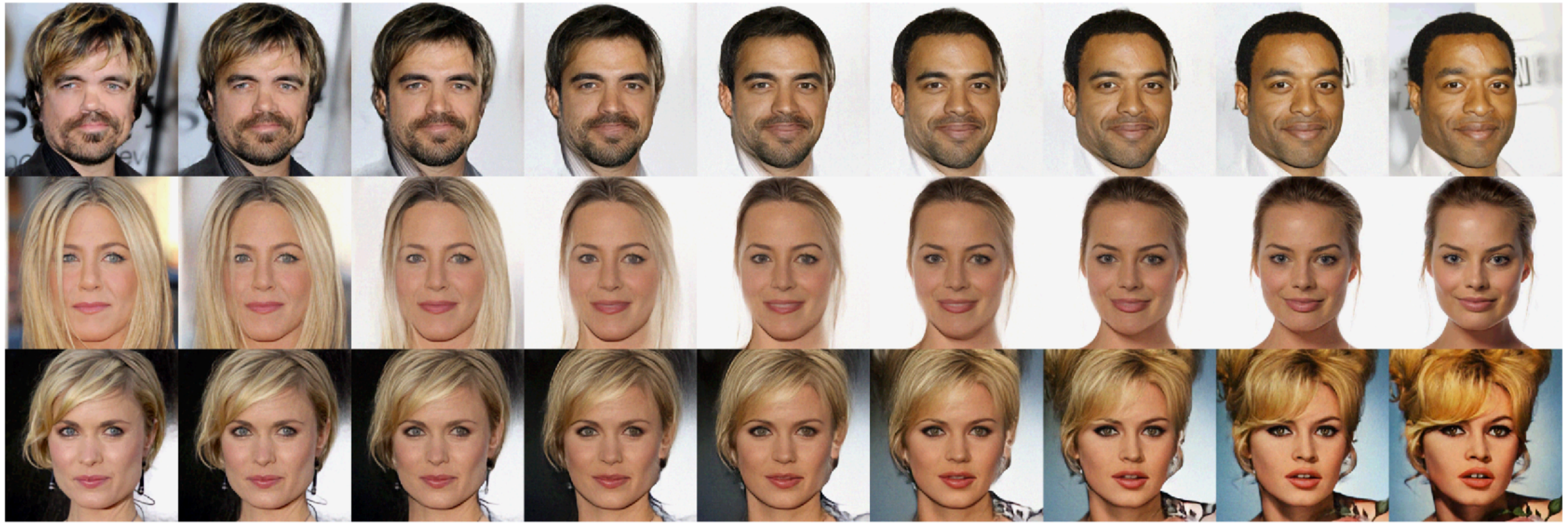
Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 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)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

Figure credit: D. Kingma, et. al., Glow: Generative Flow with Invertible 1x1 Convolutions

Results via Glow

Figure credit: D. Kingma, et. al., Glow: Generative Flow with Invertible 1x1 Convolutions

Interpolations via Glow



Demo: https://openai.com/blog/glow/
Figure credit: D. Kingma, et. al., Glow: Generative Flow with Invertible 1x1 Convolutions

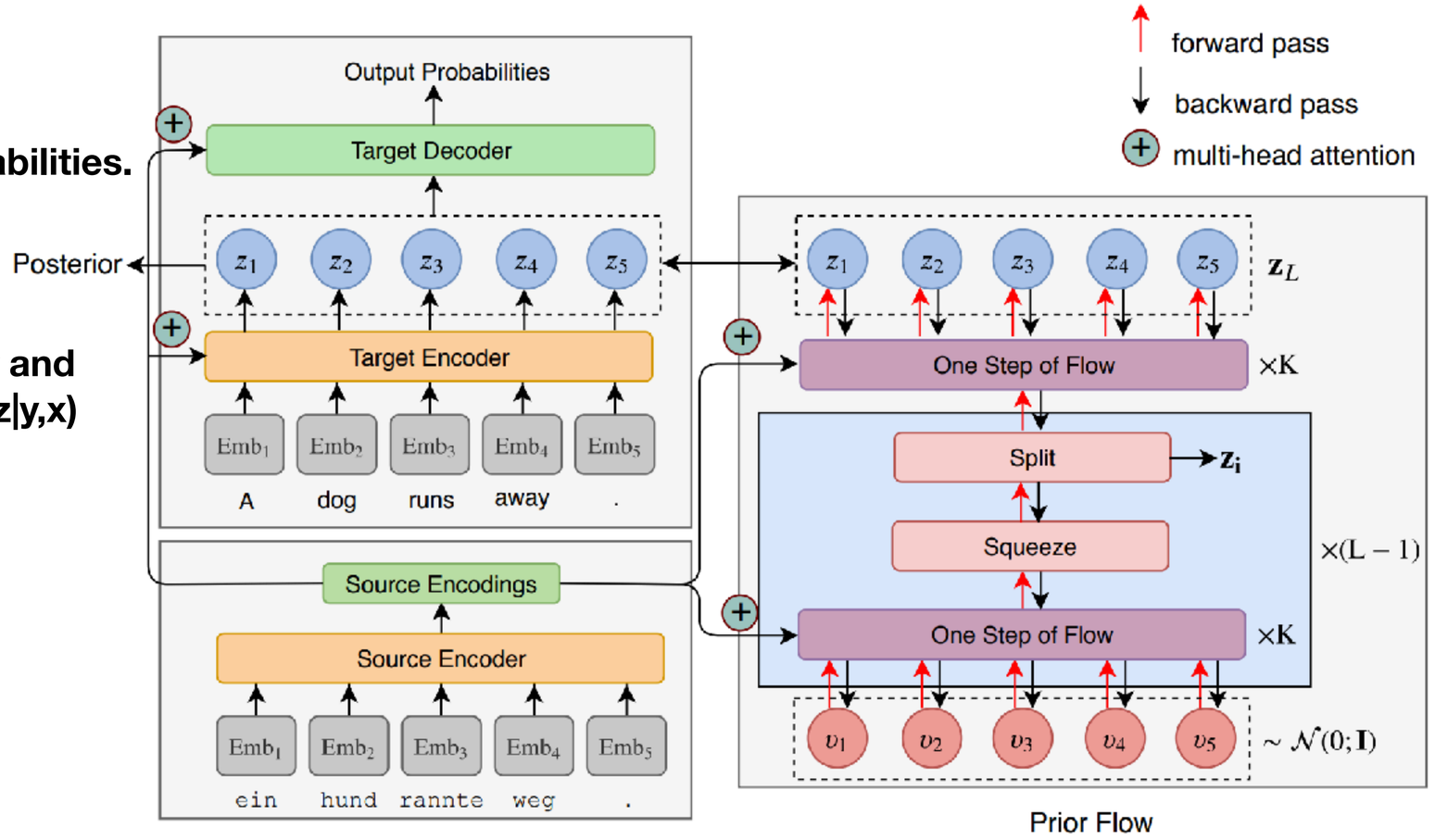
FlowSeq

FlowSeq - training time

3. Decode z and get the output probabilities.

2. Sample z from the posterior

1. Encode the target y and get the posterior q(z|y,x)



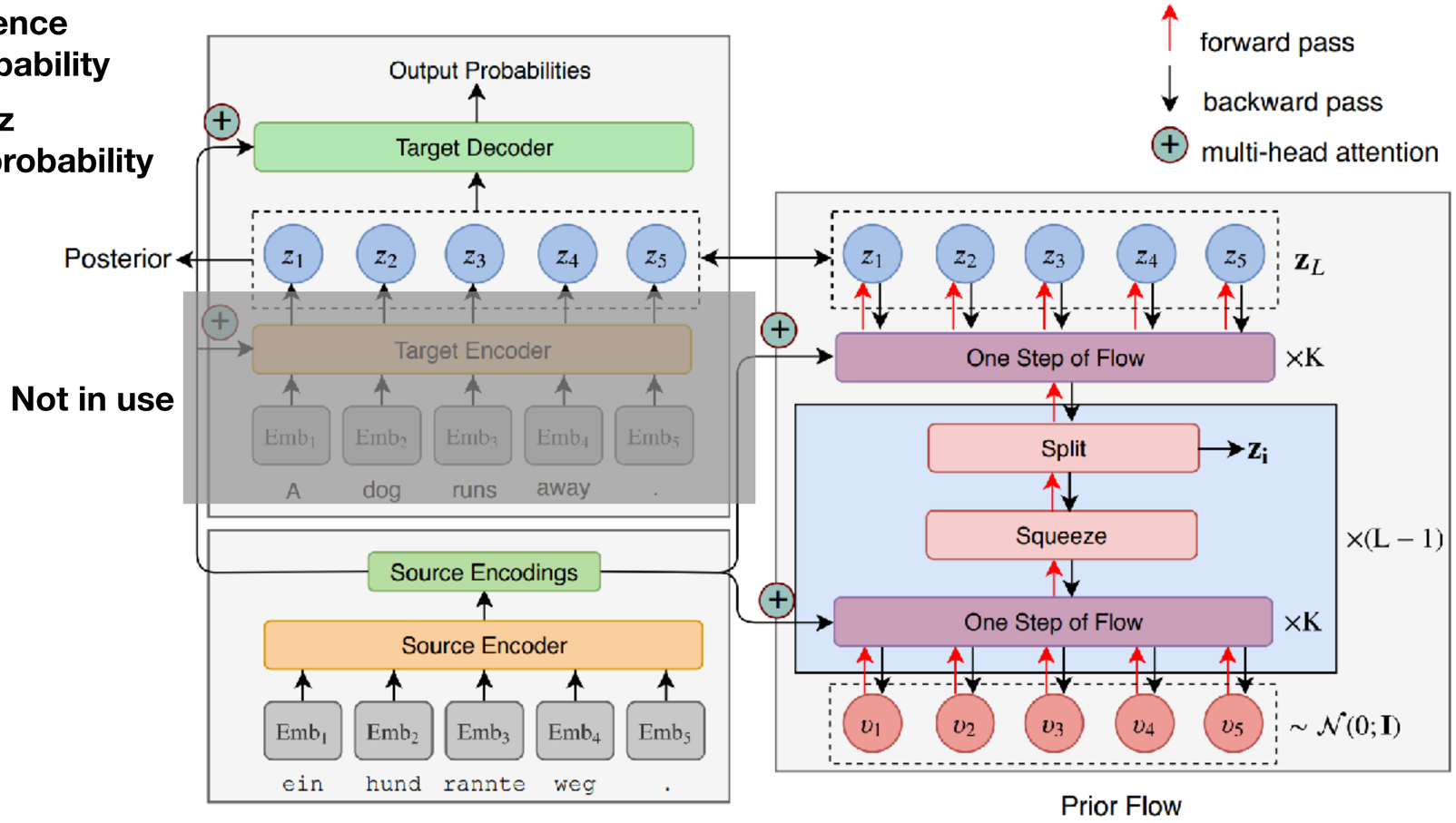
4. Normalize the sample z through the flow steps

Figure credit: Xuezhe Ma et. al., FlowSeq: Non-Autoregressive Conditional Sequence Generation with Generative Flow

FlowSeq - test time

4. Get the target sequence from the output probability

3. Decode the sample z and get the output probability



2. Get the posterior sample z through the flow steps

1. Sample v from prior

Figure credit: Xuezhe Ma et. al., FlowSeq: Non-Autoregressive Conditional Sequence Generation with Generative Flow

FlowSeq - samples

Source	Grundnahrungsmittel gibt es schlielich berall und jeder Supermarkt hat mittlerweile Sojamilch und andere Produkte.
Ground Truth	There are basic foodstuffs available everywhere , and every supermarket now has soya milk and other products.
Sample 1	After all, there are basic foods everywhere and every supermarket now has soya amch and other products.
Sample 2	After all, the food are available everywhere everywhere and every supermarket has soya milk and other products.
Sample 3	After all, basic foods exist everywhere and every supermarket has now had soy milk and other products.
Source	Es kann nicht erklären, weshalb die National Security Agency Daten ber das Privatleben von Amerikanern sammelt und warum Whistleblower bestraft werden, die staatliches Fehlverhalten offenlegen.
Ground Truth	And, most recently, it cannot excuse the failure to design a simple website more than three years since the Affordable Care Act was signed into law.
Sample 1	And recently, it cannot apologise for the inability to design a simple website in the more than three years since the adoption of Affordable Care Act.
Sample 2	And recently, it cannot excuse the inability to design a simple website in more than three years since the adoption of Affordable Care Act.
Sample 3	Recently, it cannot excuse the inability to design a simple website in more than three years since the Affordable Care Act has passed.

Figure credit: Xuezhe Ma et. al., FlowSeq: Non-Autoregressive Conditional Sequence Generation with Generative Flow

BeautyGLOW

BeautyGLOW

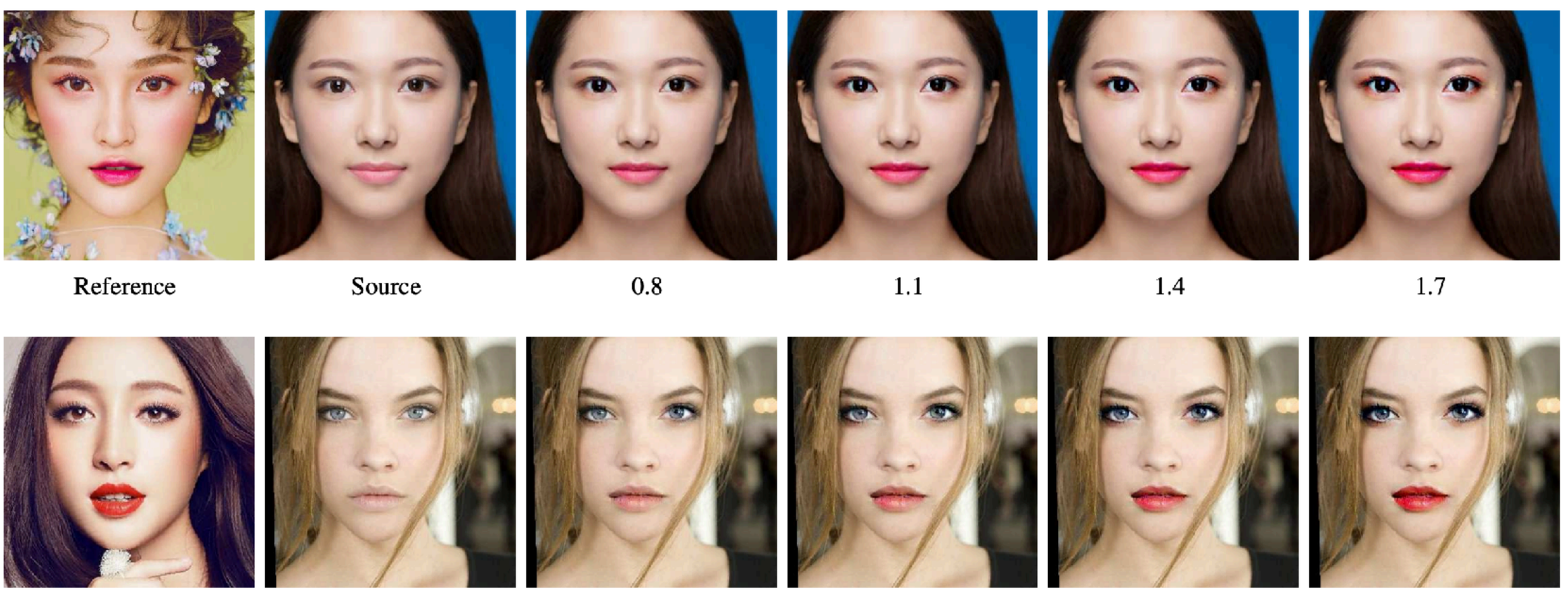


Figure credit: Hung-Jen Chen et. al., BeautyGlow: On-Demand Makeup Transfer Framework with Reversible Generative Network

BeautyGLOW

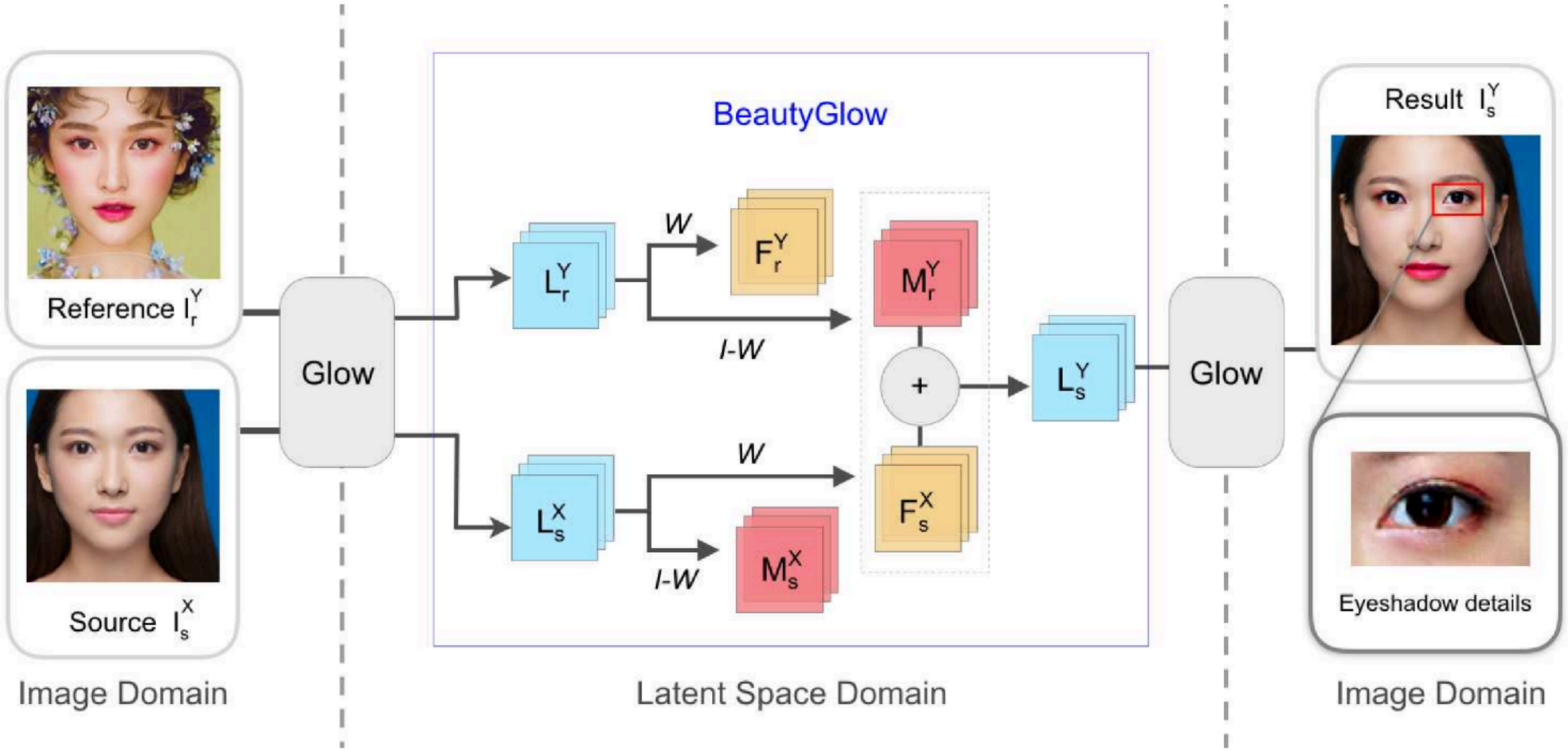


Figure credit: Hung-Jen Chen et. al., BeautyGlow: On-Demand Makeup Transfer Framework with Reversible Generative Network