Flow模型

Generative model

- Generative adversarial networks (GANs)
- Likelihood-based methods

1) Autoregressive models

这些方法的优点是简单，缺点是合成的并行性有限，因为合成的计算长度与数据的维数成正比;对于较大的图像或视频，这尤其麻烦。

2) Variational autoencoders (VAEs)

VAE采用的是优化数据对数似然下界的方式，优点是能够并行，但是在优化上存在困难，只能求得近似值。

3) Flow-based generative models

Flow-based generative models

➢1.精确的隐变量推断和对数似然估计

在VAEs中，只能得到数据点对应的潜在变量的估计值。GAN根本没有编码器来推断隐变量。而在可逆生成模型中，完全可以得到潜变量的精确值。不仅如此，模型能够优化数据的精确对数似然，而不是VAE那样仅仅优化它的下界。

➢2.隐空间能够用来做一些下游任务

自回归模型的隐含层具有未知的边缘分布，使得对于数据的操作变得十分困难。而GANs没有编码器，因而数据没法在隐空间中表示。但是具有隐空间的可逆生成模型和VAE等他们能够对数据进行各种操作，比如数据点之间的插值和对现有数据点的有意义的修改。

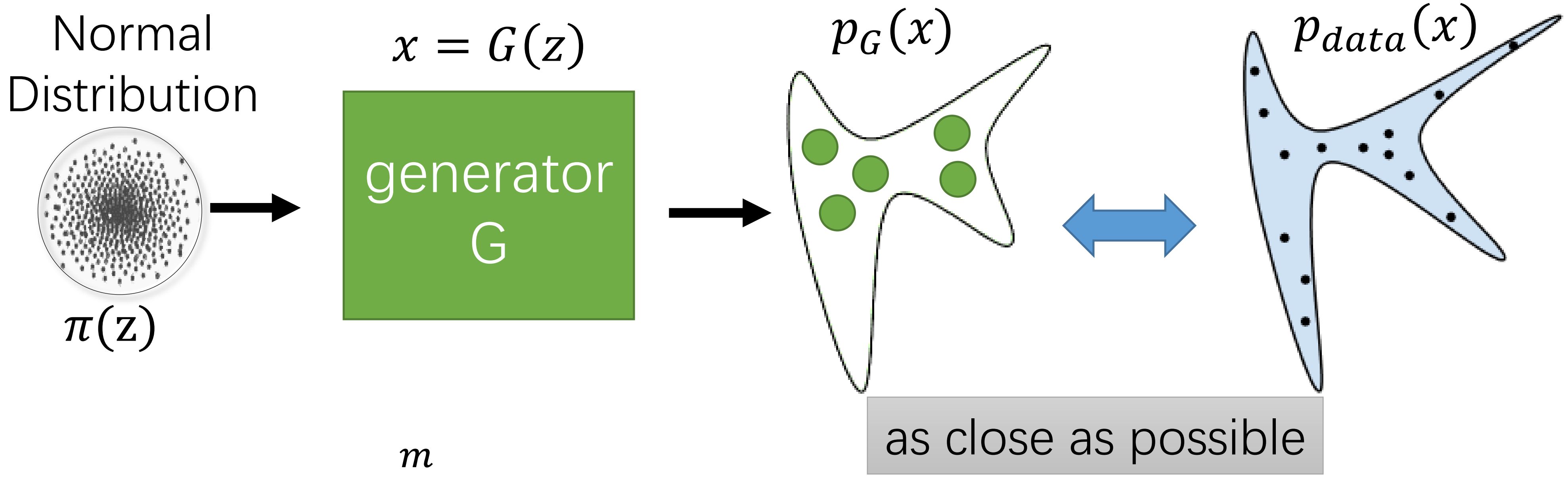
➢ 3.推理和合成的效率都很高，能够并行

Autoregressive models 像Pixel-CNN也能够可逆，但是却存在并行性的困难。而Flow不管是在inference还是在synthesis的时候能够高效的并行.

➢4.节省内存资源

Generative Model的基本思想

- A generator G is a network. The network defines a probability distribution p_G



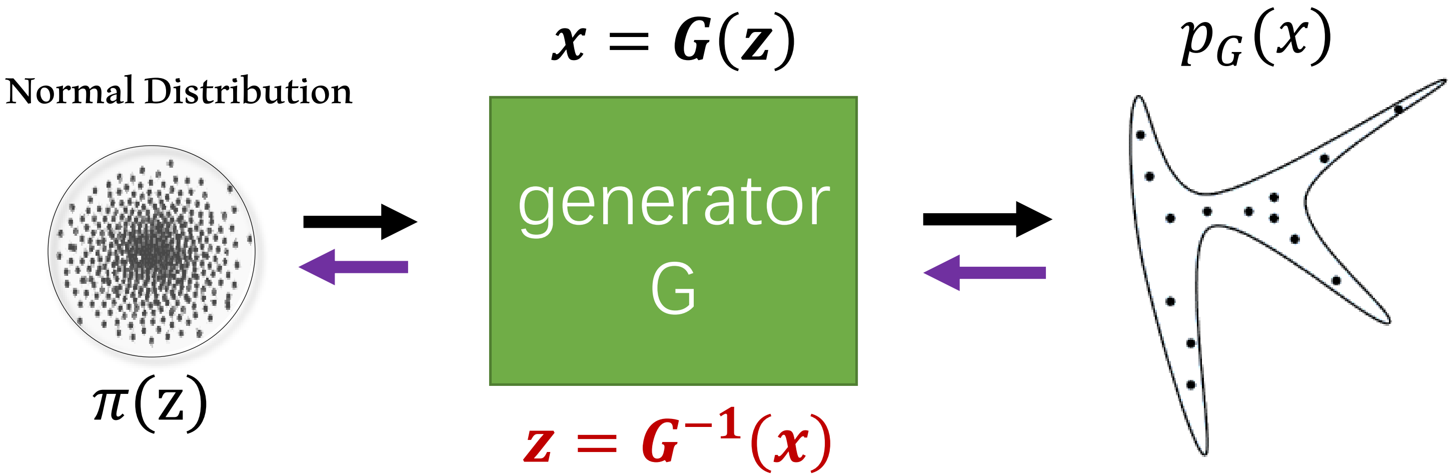
$$G^* = arg \max_G \sum_{i=1}^{m} log P_G(x^i)$$

$$\approx arg \min_G KL(P_{data}||P_G)$$

$\{x^1, x^2, \dots, x^m\}$ from $P_{data}(x)$

Flow-based模型的思想

- A generator G is a network. The network defines a probability distribution p_G



$$G^* = arg\max_G \sum_{i=1}^{m} logP_G(x^i)$$

Flow：

直接优化 objective function G^*

同时还要求 G^*是可逆的

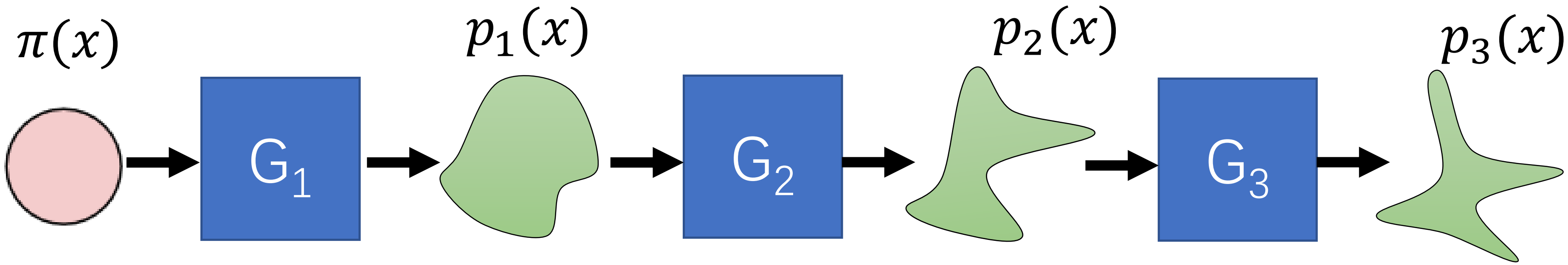
$$p(x) = \pi(z)\left|det\left(\frac{\partial z}{\partial x}\right)\right| \Rightarrow p(x) = \pi\left(G^{-1}(x)\right)\left|det\left(\frac{\partial G^{-1}(x)}{\partial x}\right)\right|$$

$$\boldsymbol{logp_G(x^i) = log\pi\left(G^{-1}(x^i)\right) + log\left|det(J_{G^{-1}})\right|}$$

Volume preserving

Norm distribution ~ $(0, \mathbf{1})$ $\Leftarrow$ $f_{\mu,\Sigma}(x) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$

Flow的由来



We focus on functions where $\mathbf{f}$ (and, likewise, $\mathbf{g}$) is composed of a sequence of transformations: $\mathbf{f} = \mathbf{f}_1 \circ \mathbf{f}_2 \circ \cdots \circ \mathbf{f}_K$, such that the relationship between $\mathbf{x}$ and $\mathbf{z}$ can be written as:

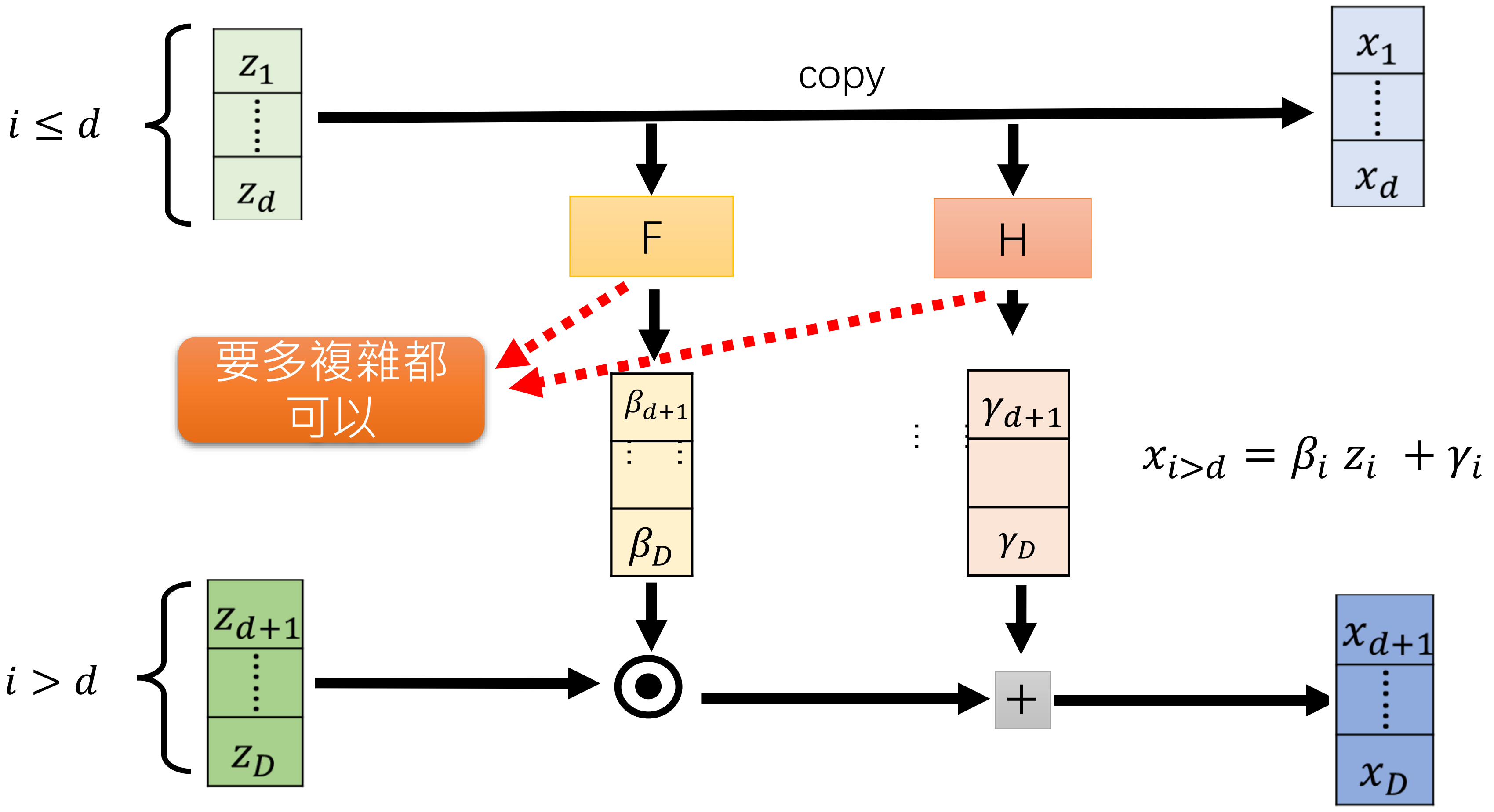
$$\mathbf{x} \overset{\mathbf{f}_1}{\longleftrightarrow} \mathbf{h}_1 \overset{\mathbf{f}_2}{\longleftrightarrow} \mathbf{h}_2 \cdots \overset{\mathbf{f}_K}{\longleftrightarrow} \mathbf{z} \tag{5}$$

Such a sequence of invertible transformations is also called a (normalizing) *flow* (Rezende and Mohamed, 2015). Under the *change of variables* of eq. (4), the probability density function (pdf) of the model given a datapoint can be written as:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{z}) + \log |\det(d\mathbf{z}/d\mathbf{x})| \tag{6}$$

$$= \log p_{\boldsymbol{\theta}}(\mathbf{z}) + \sum_{i=1}^{K} \log |\det(d\mathbf{h}_i/d\mathbf{h}_{i-1})| \tag{7}$$

Coupling Layer

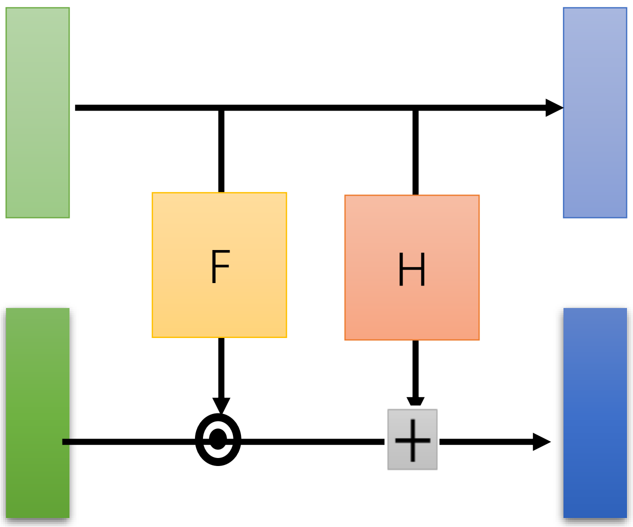


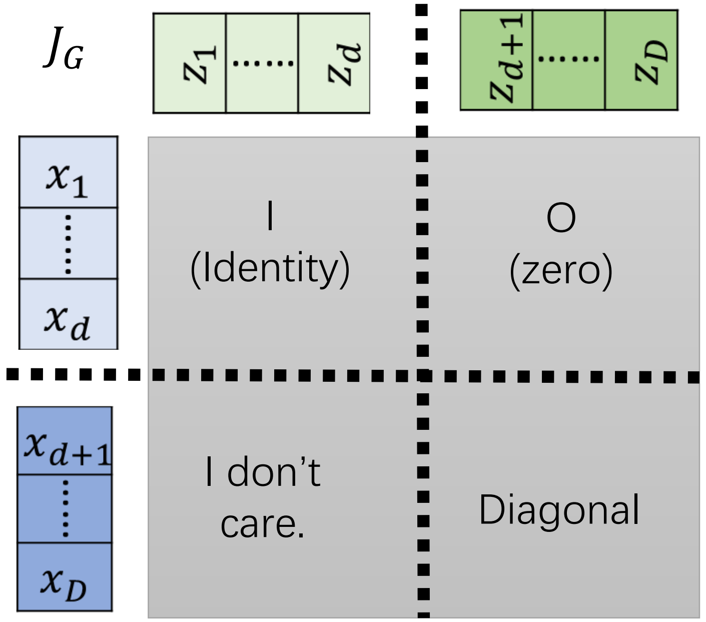
$$\begin{cases} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp\big(s(x_{1:d})\big) + t(x_{1:d}) \end{cases} \tag{7}$$

$$\Leftrightarrow \begin{cases} x_{1:d} &= y_{1:d} \\ x_{d+1:D} &= \big(y_{d+1:D} - t(y_{1:d})\big) \odot \exp\big(-s(y_{1:d})\big), \end{cases} \tag{8}$$

meaning that sampling is as efficient as inference for this model. Note again that computing the inverse of the coupling layer does not require computing the inverse of s or t, so these functions can be arbitrarily complex and difficult to invert.

Coupling Layer





$$det(J_G)$$

$$= \frac{\partial x_{d+1}}{\partial z_{d+1}} \frac{\partial x_{d+2}}{\partial z_{d+2}} \cdots \frac{\partial x_{\mathrm{D}}}{\partial z_{\mathrm{D}}}$$

$$= \beta_{d+1}\beta_{d+2} \cdots \beta_D$$

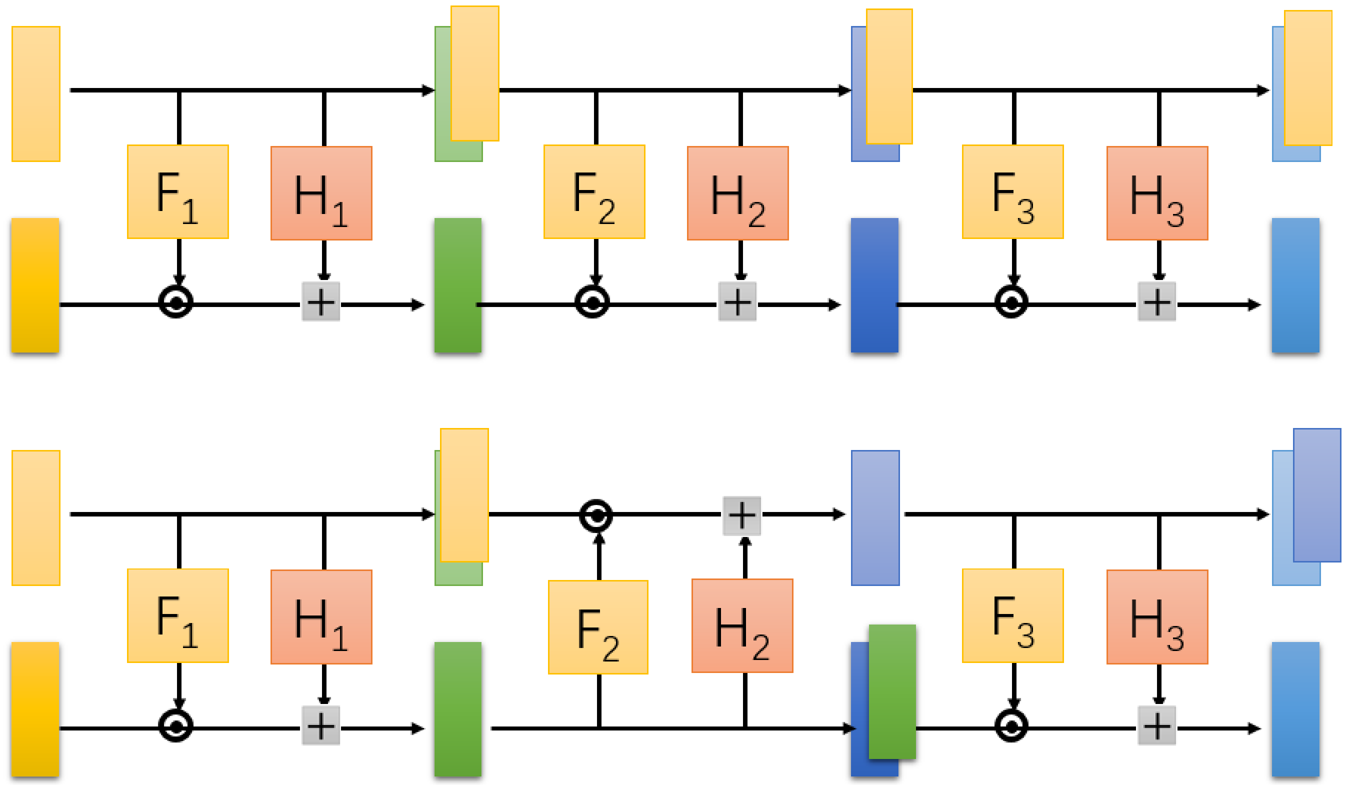
$$x_{i>d} = \beta_i \, z_i \; + \gamma_i$$

The Jacobian of this transformation is

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \operatorname{diag}\left(\exp\left[s\left(x_{1:d}\right)\right]\right) \end{bmatrix}, \tag{6}$$

where $\operatorname{diag}\left(\exp\left[s\left(x_{1:d}\right)\right]\right)$ is the diagonal matrix whose diagonal elements correspond to the vector $\exp\left[s\left(x_{1:d}\right)\right]$. Given the observation that this Jacobian is triangular, we can efficiently compute its determinant as $\exp\left[\sum_j s\left(x_{1:d}\right)_j\right]$. Since computing the Jacobian determinant of the coupling

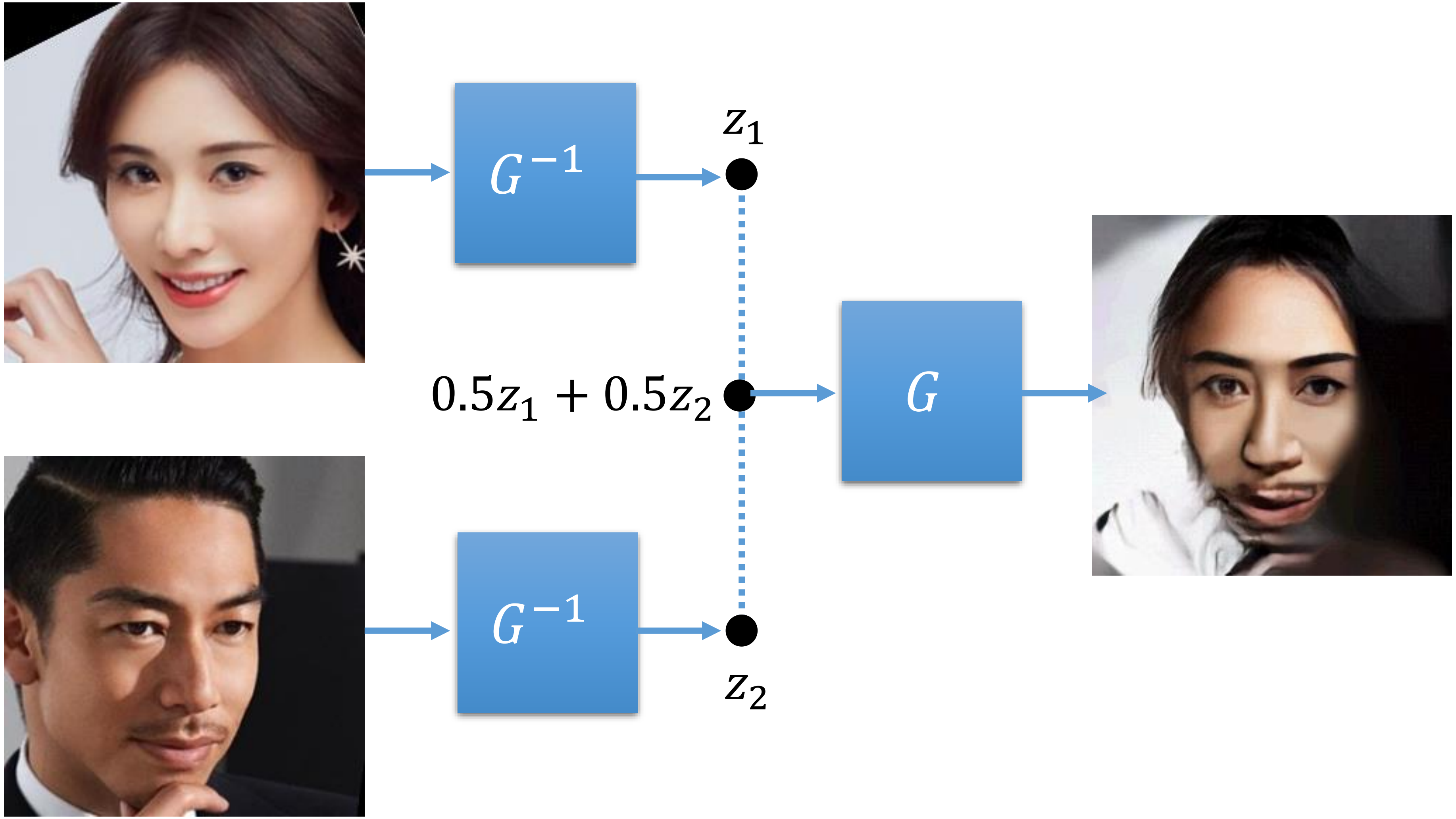
Coupling Layer - Stacking



Description	Function	Reverse Function	Log-determinant
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathtt{sum}(\log(\lvert\mathbf{s}\rvert))$

Source of image:
https://hd.stheadline.com/life/ent/realtime/1517562/

Demo of OpenAI



如何讓人笑起來

Demo of OpenAI