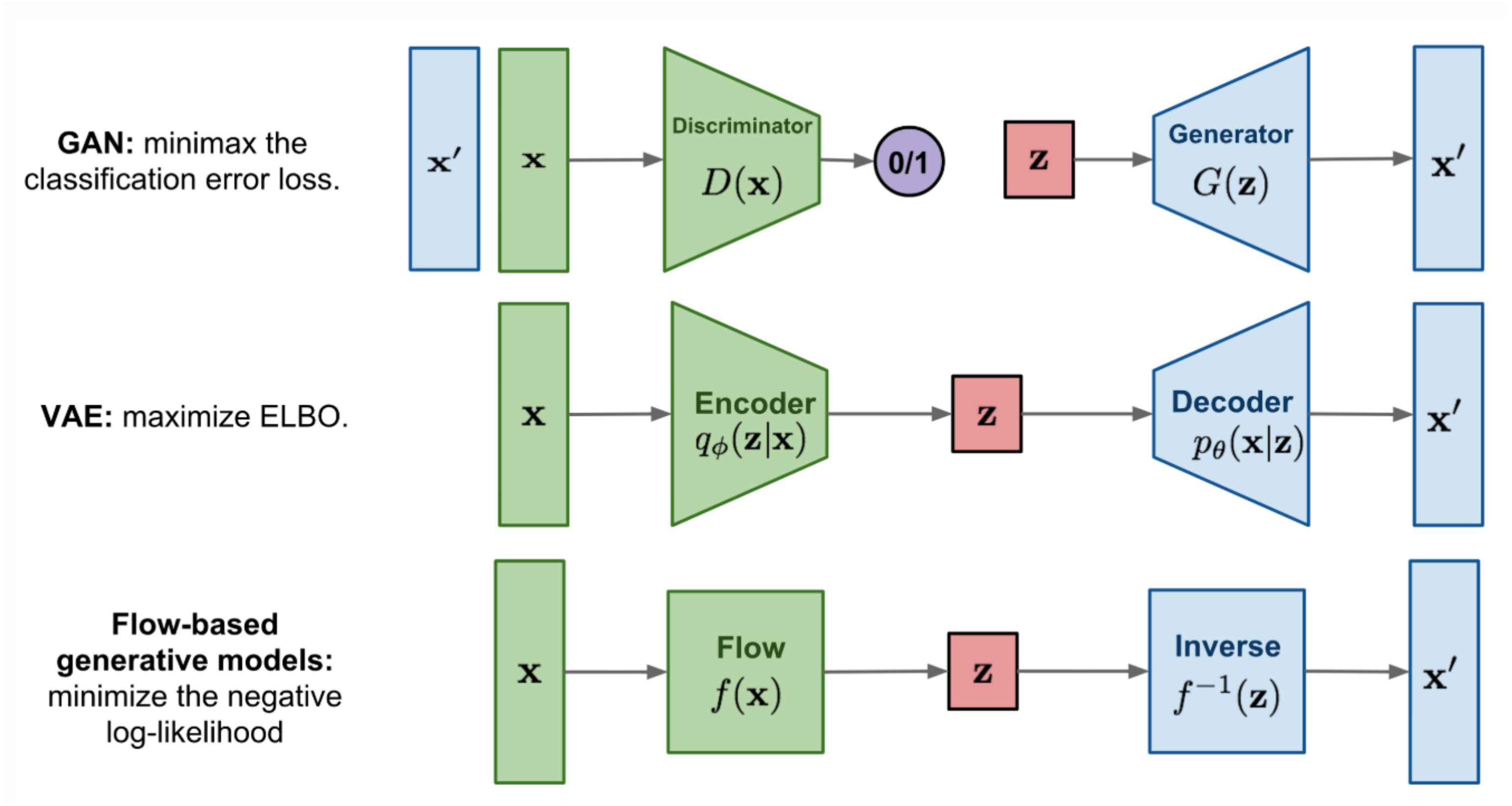
Glow: Generative Flow with Invertible 1x1 Convolutions
(Flow-based generative models)

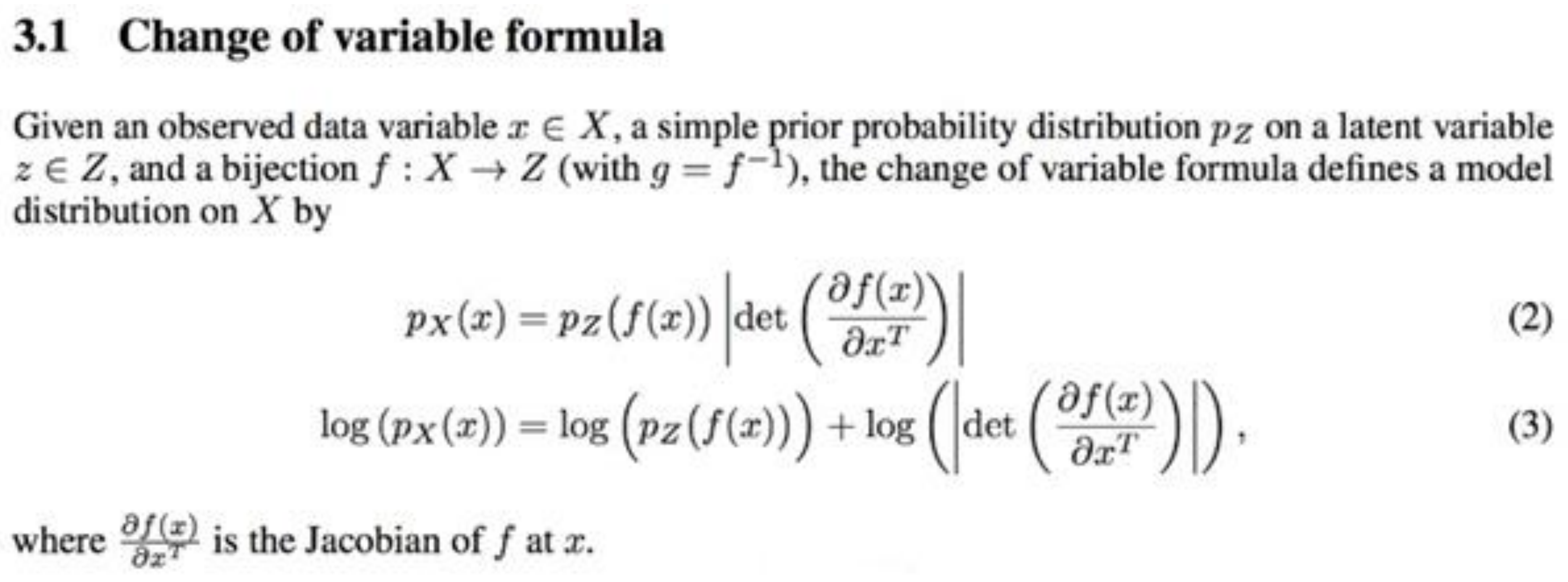
NeurIPS 2018

Generative model

1. Generative adversarial networks
2. Variational autoencoder
3. Flow-based generative models


GAN: minimax the classification error loss.
$\mathbf{x}'$
$\mathbf{x}$
Discriminator $D(\mathbf{x})$
0/1
$\mathbf{z}$
Generator $G(\mathbf{z})$
$\mathbf{x}'$
VAE: maximize ELBO.
$\mathbf{x}$
Encoder $q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{z}$
Decoder $p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
Flow-based generative models: minimize the negative log-likelihood
$\mathbf{x}$
Flow $f(\mathbf{x})$
$\mathbf{z}$
Inverse $f^{-1}(\mathbf{z})$
$\mathbf{x}'$

Change of Variable Theorem



3.1 Change of variable formula

Given an observed data variable $x \in X$, a simple prior probability distribution p_Z on a latent variable $z \in Z$, and a bijection $f : X \rightarrow Z$ (with $g = f^{-1}$), the change of variable formula defines a model distribution on X by

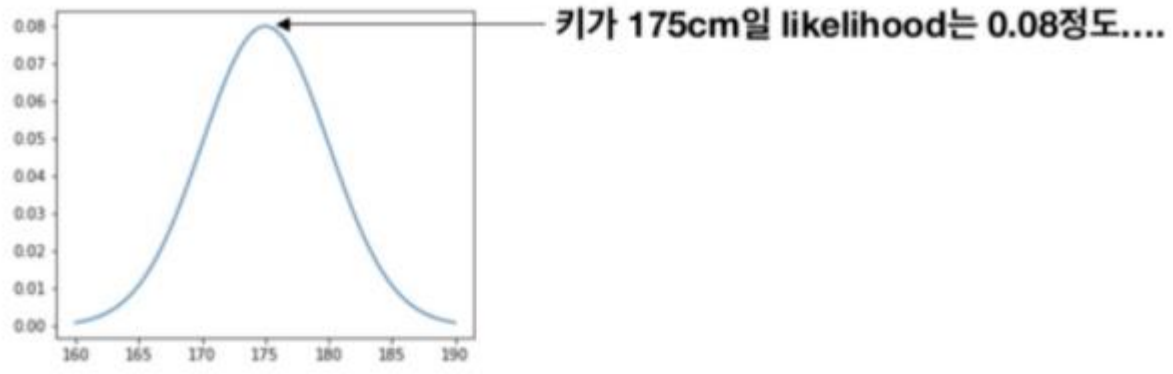
$$p_X(x) = p_Z\big(f(x)\big) \left|\det\left(\frac{\partial f(x)}{\partial x^T}\right)\right| \tag{2}$$

$$\log\left(p_X(x)\right) = \log\Big(p_Z\big(f(x)\big)\Big) + \log\left(\left|\det\left(\frac{\partial f(x)}{\partial x^T}\right)\right|\right), \tag{3}$$

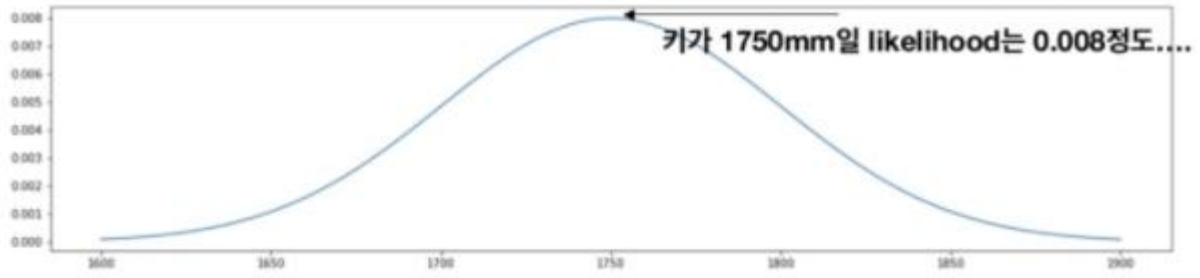
where $\frac{\partial f(x)}{\partial x^T}$ is the Jacobian of f at x.

Change of Variable Theorem

어떤 반 학생들의 키의 평균이 175cm, 표준편차 5cm라고 하고, normal distribution을 따른다고 하자.

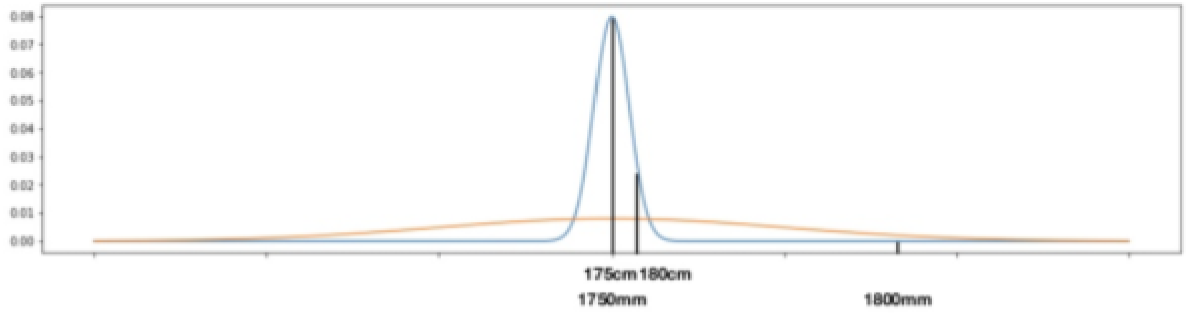


여기서 단위만 mm로 바꾼다면 아래와 같이 표현할 수 있을 것이다.



Change of Variable Theorem

단위만 바꿨지 같은 말인데 likelihood가 10배가 차이나는 것을 알 수 있다. 왜 이런 차이가 나는 것일까?

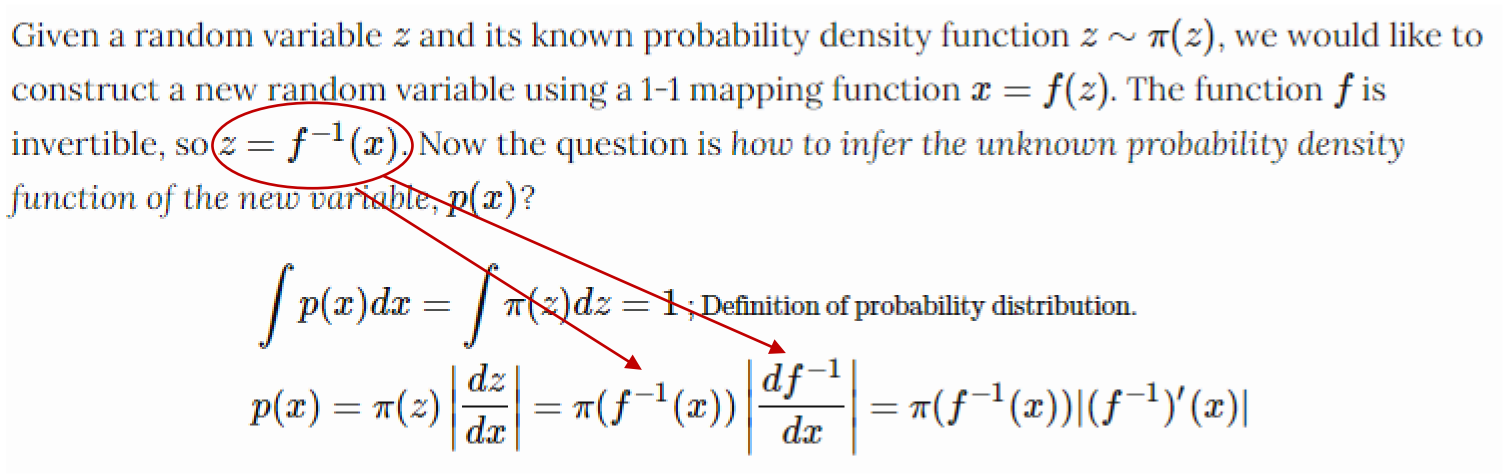


Continuous distribution의 likelihood는 해당 값이 일어날 확률을 말해주지 않는다!! pdf영역의 크기가 해당 구간이 일어날 확률을 말해준다. 아까 봤던 변수변환 식에 이 예시를 대입해보면,

$$y = 10x, \frac{\delta y}{\delta x} = 10$$
$$p_X(x) = p_Y(y) \cdot 10$$

변환된 변수와 원래 변수사이의 likelihood변화를 확인할 수 있다.

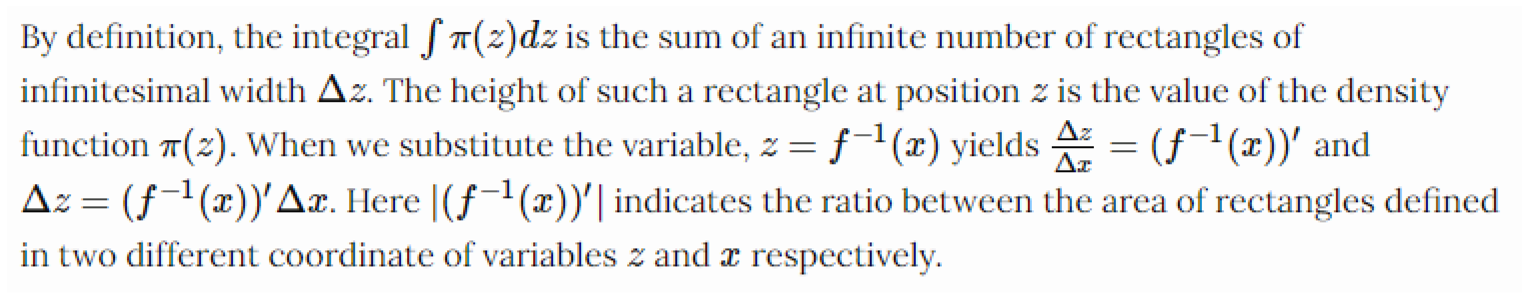
Change of Variable Theorem



Given a random variable z and its known probability density function $z \sim \pi(z)$, we would like to construct a new random variable using a 1-1 mapping function $x = f(z)$. The function f is invertible, so $z = f^{-1}(x)$. Now the question is *how to infer the unknown probability density function of the new variable*, $p(x)$?

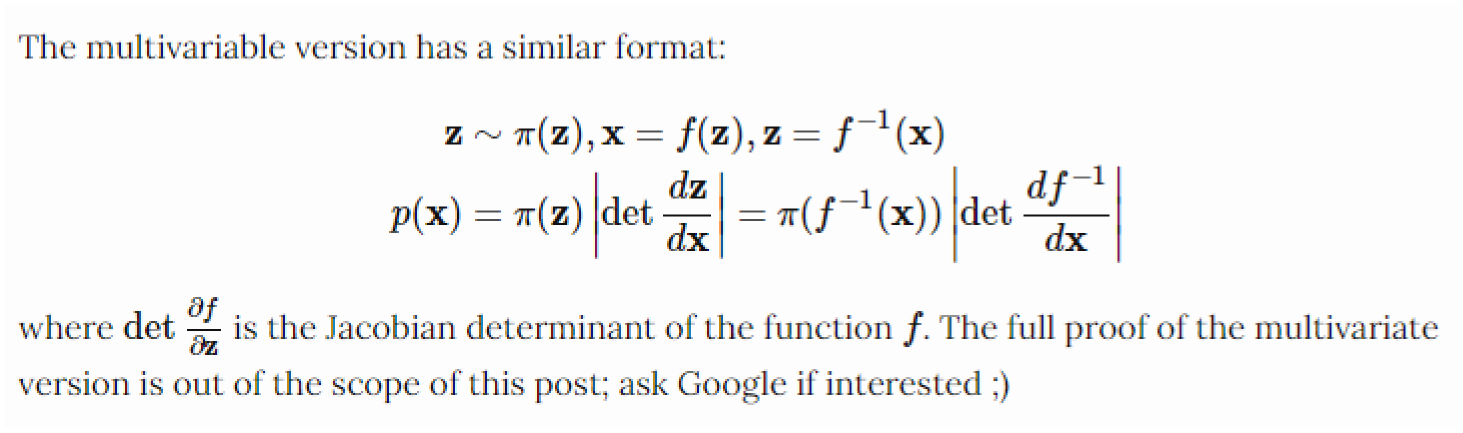
$$\int p(x)dx = \int \pi(z)dz = 1 \text{ ; Definition of probability distribution.}$$

$$p(x) = \pi(z)\left|\frac{dz}{dx}\right| = \pi(f^{-1}(x))\left|\frac{df^{-1}}{dx}\right| = \pi(f^{-1}(x))|(f^{-1})'(x)|$$



By definition, the integral $\int \pi(z)dz$ is the sum of an infinite number of rectangles of infinitesimal width Δz. The height of such a rectangle at position z is the value of the density function $\pi(z)$. When we substitute the variable, $z = f^{-1}(x)$ yields $\frac{\Delta z}{\Delta x} = (f^{-1}(x))'$ and $\Delta z = (f^{-1}(x))'\Delta x$. Here $|(f^{-1}(x))'|$ indicates the ratio between the area of rectangles defined in two different coordinate of variables z and x respectively.

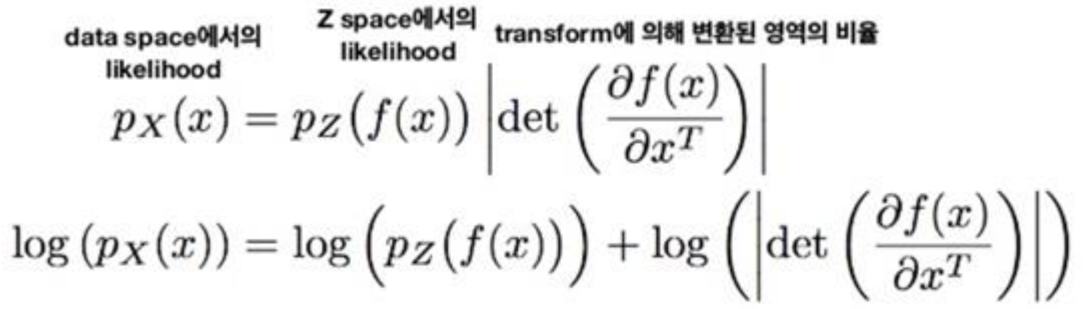
Change of Variable Theorem



The multivariable version has a similar format:

$$\mathbf{z} \sim \pi(\mathbf{z}), \mathbf{x} = f(\mathbf{z}), \mathbf{z} = f^{-1}(\mathbf{x})$$
$$p(\mathbf{x}) = \pi(\mathbf{z}) \left| \det \frac{d\mathbf{z}}{d\mathbf{x}} \right| = \pi(f^{-1}(\mathbf{x})) \left| \det \frac{df^{-1}}{d\mathbf{x}} \right|$$

where $\det \frac{\partial f}{\partial \mathbf{z}}$ is the Jacobian determinant of the function f. The full proof of the multivariate version is out of the scope of this post; ask Google if interested ;)



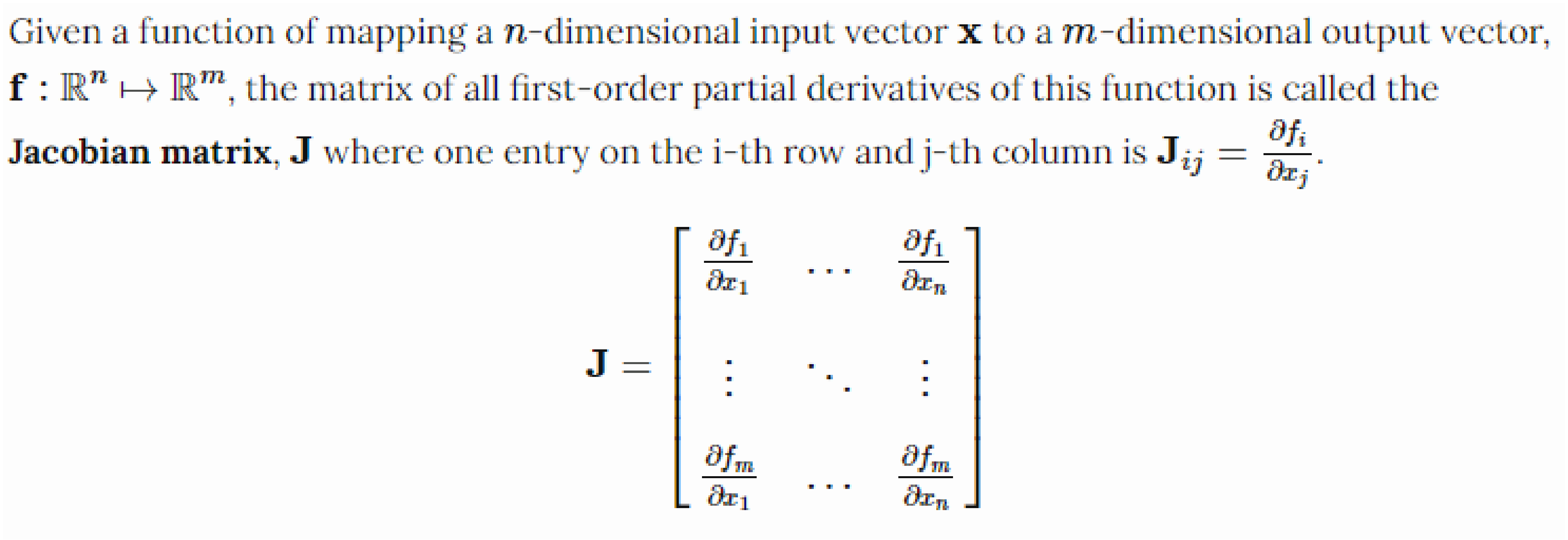
data space에서의 likelihood

Z space에서의 likelihood

transform에 의해 변환된 영역의 비율

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

Jacobian Matrix and Determinant



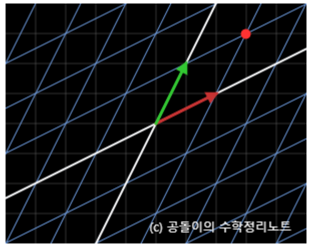
Given a function of mapping a n-dimensional input vector $\mathbf{x}$ to a m-dimensional output vector, $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$, the matrix of all first-order partial derivatives of this function is called the **Jacobian matrix**, $\mathbf{J}$ where one entry on the i-th row and j-th column is $\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Jacobian Matrix and Determinant

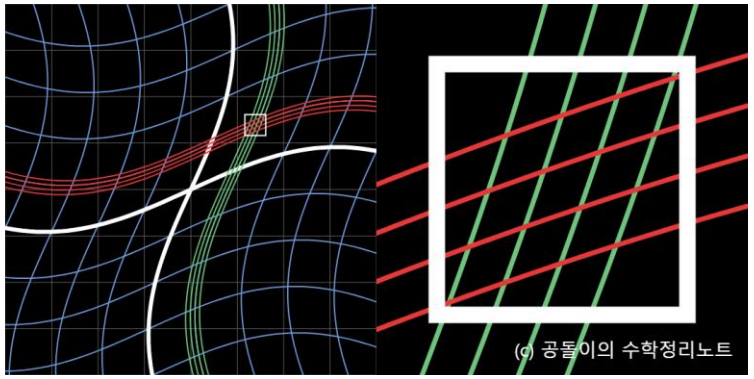
행렬 : 선형변환
Jacobian Matrix : 비선형 변환을 선형 변환으로 근사시킨것

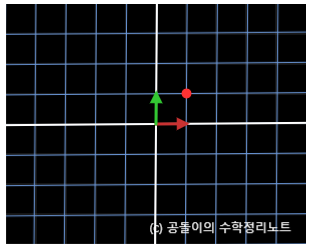
$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$



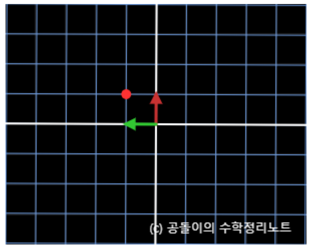
shearing

$$f(x,y) = \begin{bmatrix} x + \sin(y/2) \\ y + \sin(x/2) \end{bmatrix}$$



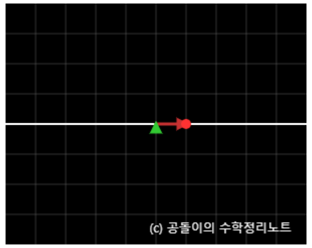


$$\begin{bmatrix} \cos(\pi/2) & -\sin(\pi/2) \\ \sin(\pi/2) & \cos(\pi/2) \end{bmatrix}$$



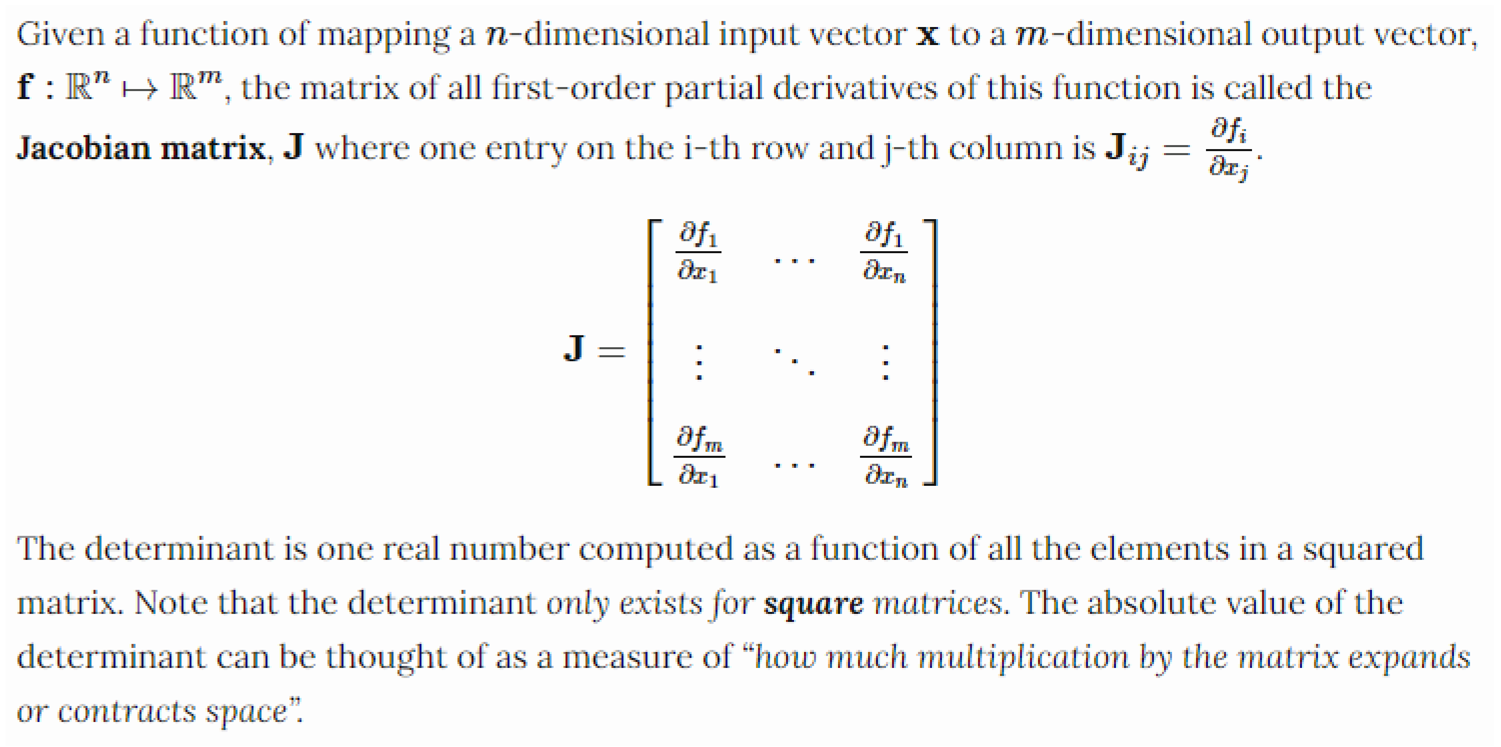
rotation

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$



projection on x-axis

Jacobian Matrix and Determinant



Given a function of mapping a n-dimensional input vector $\mathbf{x}$ to a m-dimensional output vector, $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$, the matrix of all first-order partial derivatives of this function is called the **Jacobian matrix**, $\mathbf{J}$ where one entry on the i-th row and j-th column is $\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

The determinant is one real number computed as a function of all the elements in a squared matrix. Note that the determinant *only exists for* ***square*** *matrices*. The absolute value of the determinant can be thought of as a measure of *"how much multiplication by the matrix expands or contracts space"*.

Normalizing Flow

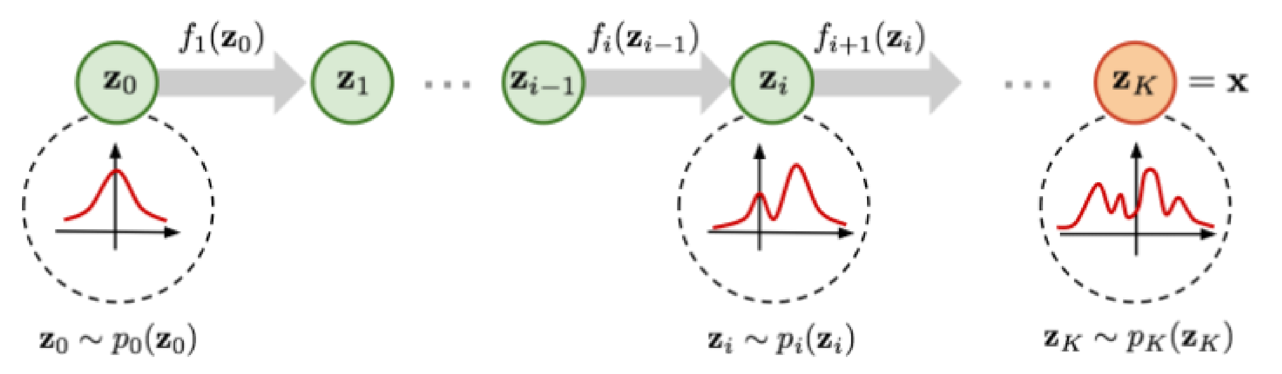
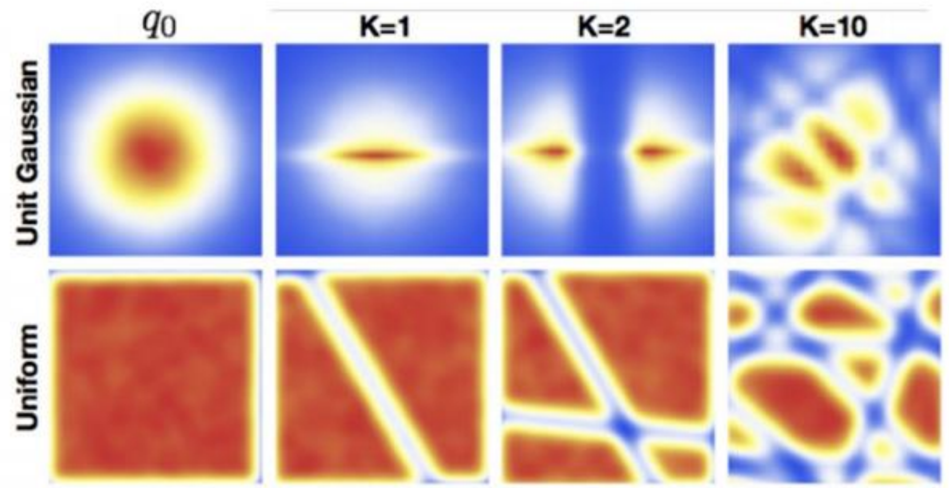
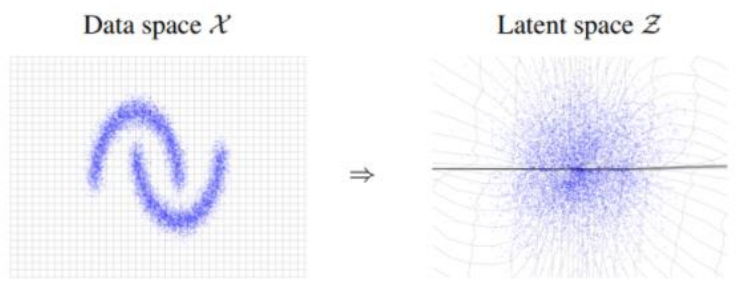

$f_1(\mathbf{z}_0)$
$\mathbf{z}_0$
$\mathbf{z}_1$
$\mathbf{z}_{i-1}$
$f_i(\mathbf{z}_{i-1})$
$\mathbf{z}_i$
$f_{i+1}(\mathbf{z}_i)$
$\mathbf{z}_K = \mathbf{x}$
$\mathbf{z}_0 \sim p_0(\mathbf{z}_0)$
$\mathbf{z}_i \sim p_i(\mathbf{z}_i)$
$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$

Fig. 2. Illustration of a normalizing flow model, transforming a simple distribution $p_0(\mathbf{z}_0)$ *to a complex one* $p_K(\mathbf{z}_K)$ *step by step.*

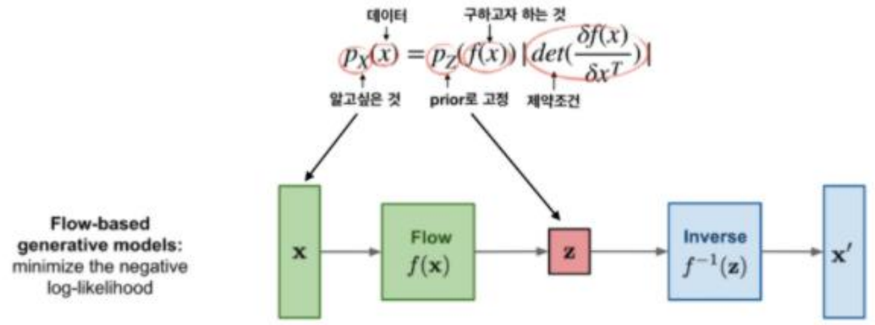
unit gaussian/uniform분포를 가정하고 flow를 10번 거치니 복잡한 분포도 표현할 수 있음을 예시로 확인가능



Normalizing Flow



총 정리를 해보면,
구하고자 하는 것: f(x)
제약조건: 자코비안 행렬의 determinant



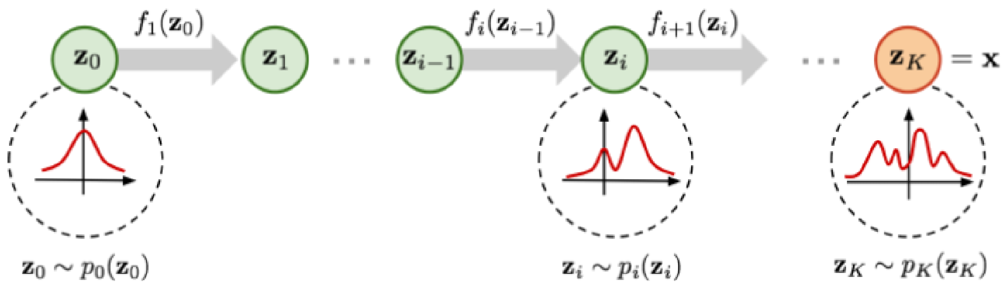
데이터
구하고자 하는 것
$p_X(x) = p_Z(f(x)) |det(\frac{\delta f(x)}{\delta x^T})|$
알고싶은 것
prior로 고정
제약조건
Flow-based generative models: minimize the negative log-likelihood
x
Flow $f(\mathbf{x})$
z
Inverse $f^{-1}(\mathbf{z})$
$\mathbf{x}'$

이렇게 pz(z)를 간단하게 정하고,

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

위의 식을 만족시키는 z=f(x)에서의 복잡한 f(x)를 찾는 문제로 변환할 수 있다.

Normalizing Flow


$\mathbf{z}_0$
$f_1(\mathbf{z}_0)$
$\mathbf{z}_1$
$\mathbf{z}_{i-1}$
$f_i(\mathbf{z}_{i-1})$
$\mathbf{z}_i$
$f_{i+1}(\mathbf{z}_i)$
$\mathbf{z}_K = \mathbf{x}$
$\mathbf{z}_0 \sim p_0(\mathbf{z}_0)$
$\mathbf{z}_i \sim p_i(\mathbf{z}_i)$
$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$

$$\mathbf{z} \sim \pi(\mathbf{z}), \mathbf{x} = f(\mathbf{z}), \mathbf{z} = f^{-1}(\mathbf{x})$$
$$p(\mathbf{x}) = \pi(\mathbf{z}) \left|\det \frac{d\mathbf{z}}{d\mathbf{x}}\right| = \pi(f^{-1}(\mathbf{x})) \left|\det \frac{df^{-1}}{d\mathbf{x}}\right|$$

$$\Rightarrow$$

$$\mathbf{z}_{i-1} \sim p_{i-1}(\mathbf{z}_{i-1})$$
$$\mathbf{z}_i = f_i(\mathbf{z}_{i-1}), \text{ thus } \mathbf{z}_{i-1} = f_i^{-1}(\mathbf{z}_i)$$
$$p_i(\mathbf{z}_i) = p_{i-1}(f_i^{-1}(\mathbf{z}_i)) \left|\det \frac{df_i^{-1}}{d\mathbf{z}_i}\right|$$

Normalizing Flow

$$\mathbf{z}_{i-1} \sim p_{i-1}(\mathbf{z}_{i-1})$$

$$\mathbf{z}_i = f_i(\mathbf{z}_{i-1}), \text{ thus } \mathbf{z}_{i-1} = f_i^{-1}(\mathbf{z}_i)$$

$$p_i(\mathbf{z}_i) = p_{i-1}(f_i^{-1}(\mathbf{z}_i)) \left|\det \frac{df_i^{-1}}{d\mathbf{z}_i}\right|$$

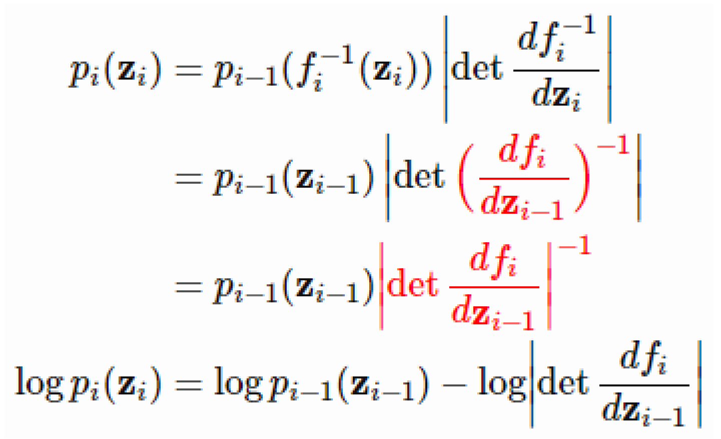
(*) A note on the "*inverse function theorem*": If $y = f(x)$ and $x = f^{-1}(y)$, we have:

$$\frac{df^{-1}(y)}{dy} = \frac{dx}{dy} = \left(\frac{dy}{dx}\right)^{-1} = \left(\frac{df(x)}{dx}\right)^{-1}$$

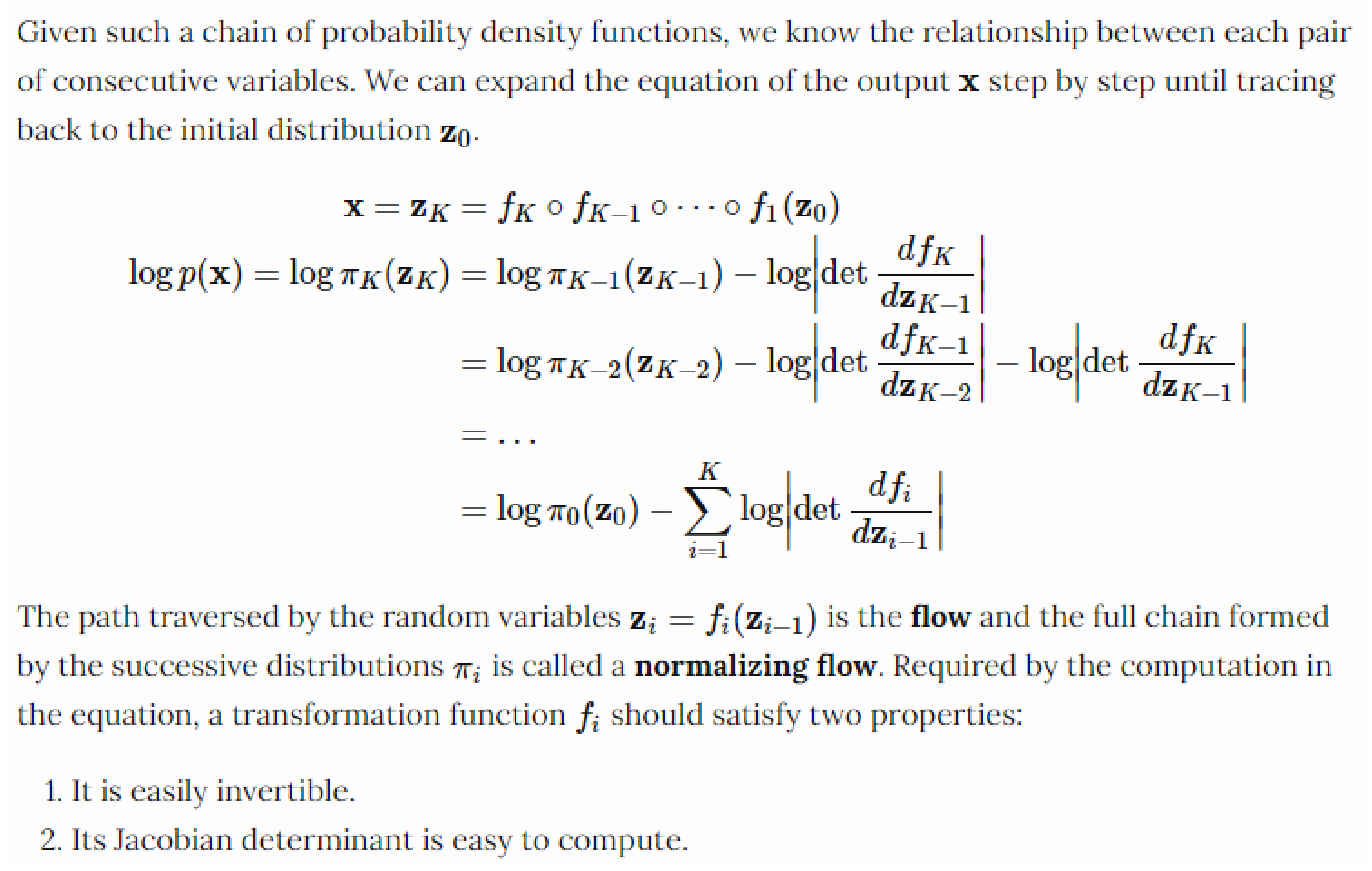
(*) A note on "*Jacobians of invertible function*": The determinant of the inverse of an invertible matrix is the inverse of the determinant: $\det(M^{-1}) = (\det(M))^{-1}$, because $\det(M)\det(M^{-1}) = \det(M \cdot M^{-1}) = \det(I) = 1$.

$$\begin{aligned}
p_i(\mathbf{z}_i) &= p_{i-1}(f_i^{-1}(\mathbf{z}_i)) \left|\det \frac{df_i^{-1}}{d\mathbf{z}_i}\right| \\
&= p_{i-1}(\mathbf{z}_{i-1}) \left|\det \left(\frac{df_i}{d\mathbf{z}_{i-1}}\right)^{-1}\right| && \text{; According to the inverse func theorem.} \\
&= p_{i-1}(\mathbf{z}_{i-1}) \left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right|^{-1} && \text{; According to a property of Jacobians of invertible func.} \\
\log p_i(\mathbf{z}_i) &= \log p_{i-1}(\mathbf{z}_{i-1}) - \log\left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right|
\end{aligned}$$

Normalizing Flow



$$\begin{aligned} p_i(\mathbf{z}_i) &= p_{i-1}(f_i^{-1}(\mathbf{z}_i)) \left|\det \frac{df_i^{-1}}{d\mathbf{z}_i}\right| \\ &= p_{i-1}(\mathbf{z}_{i-1}) \left|\det \left(\frac{df_i}{d\mathbf{z}_{i-1}}\right)^{-1}\right| \\ &= p_{i-1}(\mathbf{z}_{i-1}) \left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right|^{-1} \\ \log p_i(\mathbf{z}_i) &= \log p_{i-1}(\mathbf{z}_{i-1}) - \log\left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right| \end{aligned}$$



Given such a chain of probability density functions, we know the relationship between each pair of consecutive variables. We can expand the equation of the output $\mathbf{x}$ step by step until tracing back to the initial distribution $\mathbf{z}_0$.

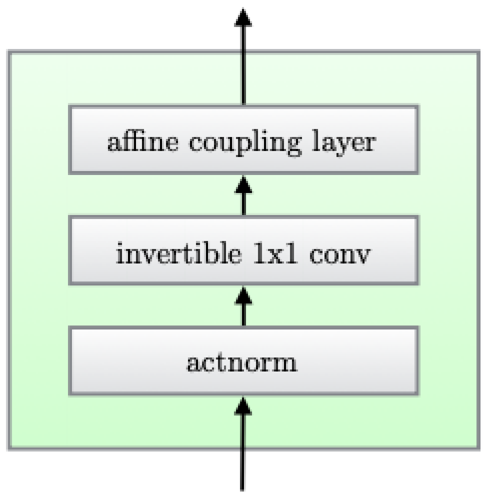
$$\mathbf{x} = \mathbf{z}_K = f_K \circ f_{K-1} \circ \dots \circ f_1(\mathbf{z}_0)$$

$$\begin{aligned} \log p(\mathbf{x}) = \log \pi_K(\mathbf{z}_K) &= \log \pi_{K-1}(\mathbf{z}_{K-1}) - \log\left|\det \frac{df_K}{d\mathbf{z}_{K-1}}\right| \\ &= \log \pi_{K-2}(\mathbf{z}_{K-2}) - \log\left|\det \frac{df_{K-1}}{d\mathbf{z}_{K-2}}\right| - \log\left|\det \frac{df_K}{d\mathbf{z}_{K-1}}\right| \\ &= \dots \\ &= \log \pi_0(\mathbf{z}_0) - \sum_{i=1}^{K} \log\left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right| \end{aligned}$$

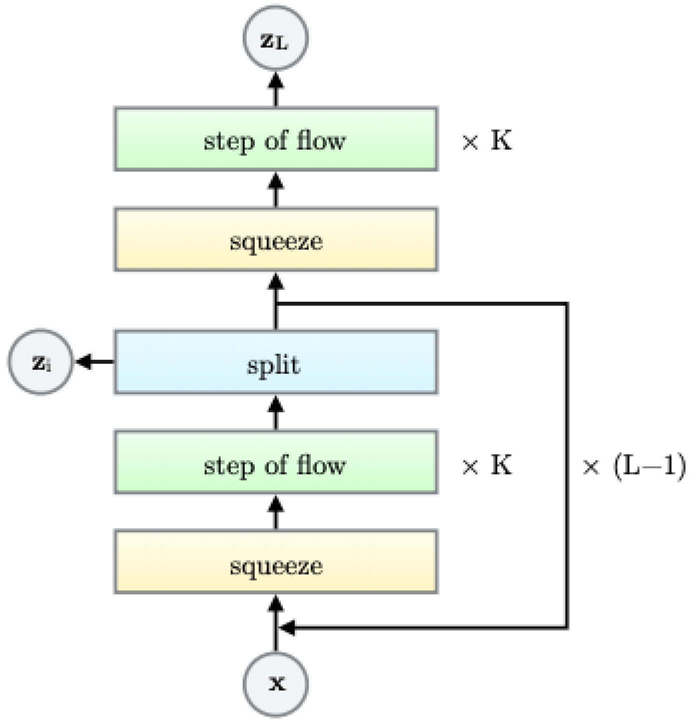
The path traversed by the random variables $\mathbf{z}_i = f_i(\mathbf{z}_{i-1})$ is the **flow** and the full chain formed by the successive distributions π_i is called a **normalizing flow**. Required by the computation in the equation, a transformation function f_i should satisfy two properties:

1. It is easily invertible.
2. Its Jacobian determinant is easy to compute.

Glow



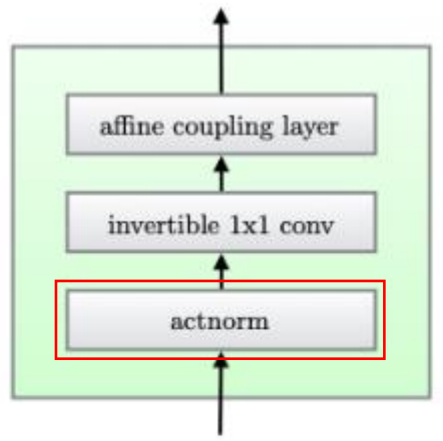
(a) One step of our flow.



(b) Multi-scale architecture (Dinh et al., 2016).

Glow

- Actnorm



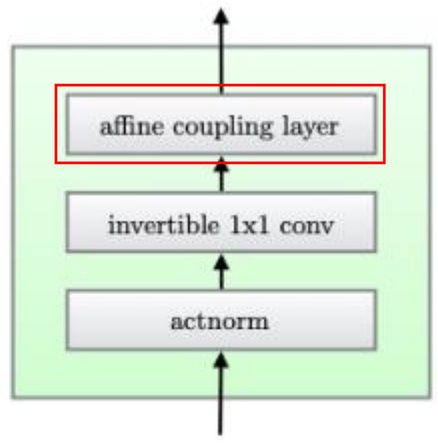
(a) One step of our flow.

- Scale과 bias parameter를 이용해서 affine transform
- Batch normalization은 minibatch 사이즈가 작을수록 성능이 줄어듬
- 이미지가 크면 minibatch size를 1로 두고 하는데 actnorm을 쓰는 것이 효과적임
- Mean = 0, standard deviation = 1
- 처음 initialize할때는 데이터에 분포에 의존하지만 이후로는 trainable parameters

$$\forall i, j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$$

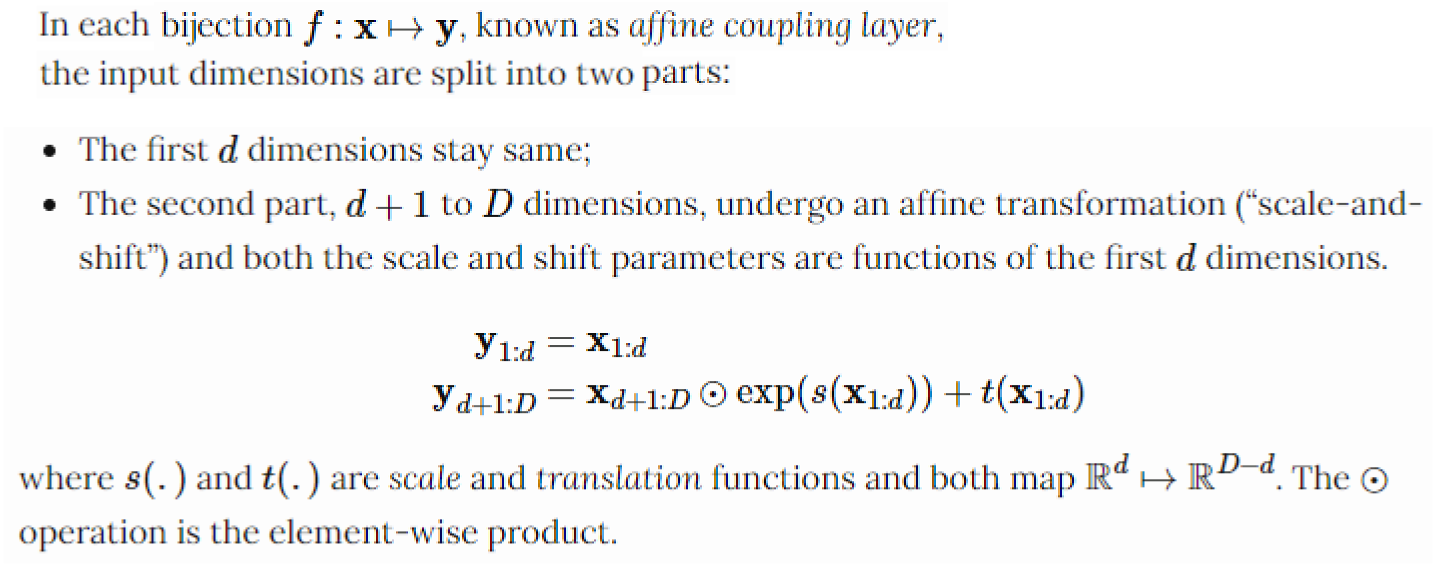
Glow

- Affine coupling layer



(a) One step of our flow.

RealNVP



In each bijection $f : \mathbf{x} \mapsto \mathbf{y}$, known as *affine coupling layer*, the input dimensions are split into two parts:

- The first d dimensions stay same;
- The second part, $d+1$ to D dimensions, undergo an affine transformation ("scale-and-shift") and both the scale and shift parameters are functions of the first d dimensions.

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d}) \end{aligned}$$

where $s(.)$ and $t(.)$ are *scale* and *translation* functions and both map $\mathbb{R}^d \mapsto \mathbb{R}^{D-d}$. The $\odot$ operation is the element-wise product.

Glow

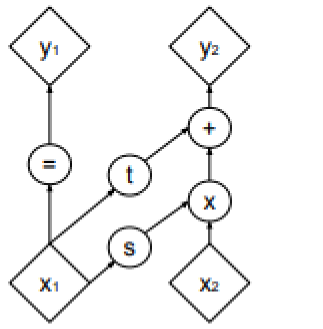
- The first d dimensions stay same;
- The second part, $d+1$ to D dimensions, undergo an affine transformation ("scale-and-shift") and both the scale and shift parameters are functions of the first d dimensions.

$$\begin{aligned}\mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d})\end{aligned}$$

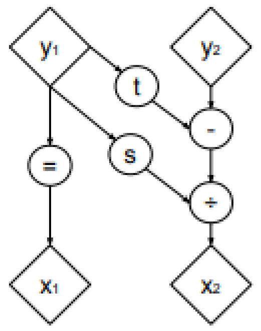
Condition 1: "It is easily invertible."

Yes and it is fairly straightforward.

$$\begin{cases}\mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d})\end{cases} \Leftrightarrow \begin{cases}\mathbf{x}_{1:d} &= \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} &= (\mathbf{y}_{d+1:D} - t(\mathbf{y}_{1:d})) \odot \exp(-s(\mathbf{y}_{1:d}))\end{cases}$$

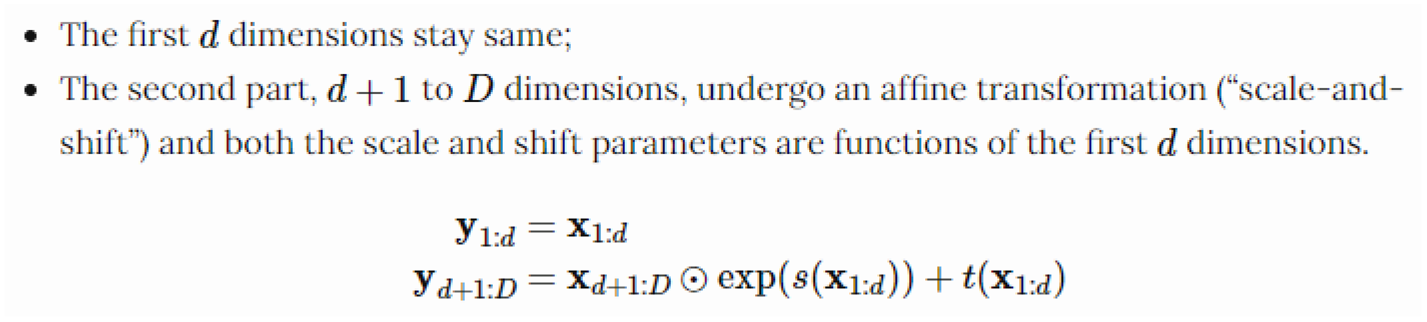


(a) Forward propagation



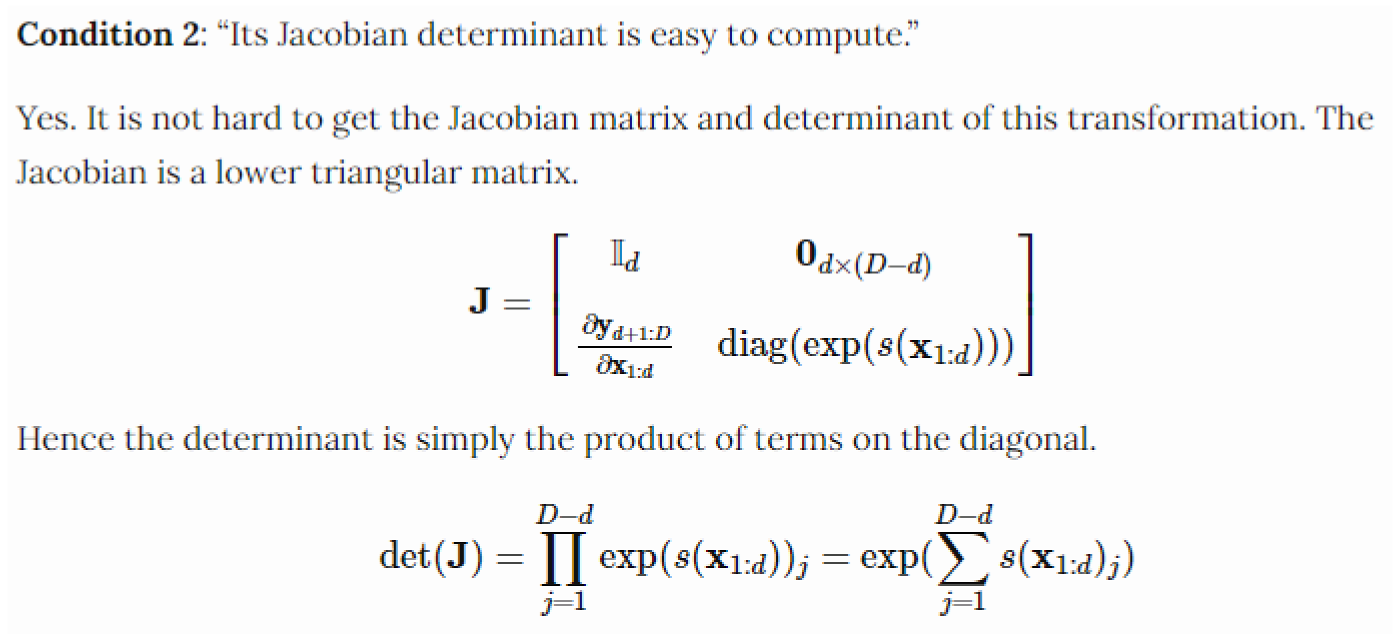
(b) Inverse propagation

Glow



- The first d dimensions stay same;
- The second part, $d+1$ to D dimensions, undergo an affine transformation ("scale-and-shift") and both the scale and shift parameters are functions of the first d dimensions.

$$\begin{aligned}\mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d})\end{aligned}$$



Condition 2: "Its Jacobian determinant is easy to compute."

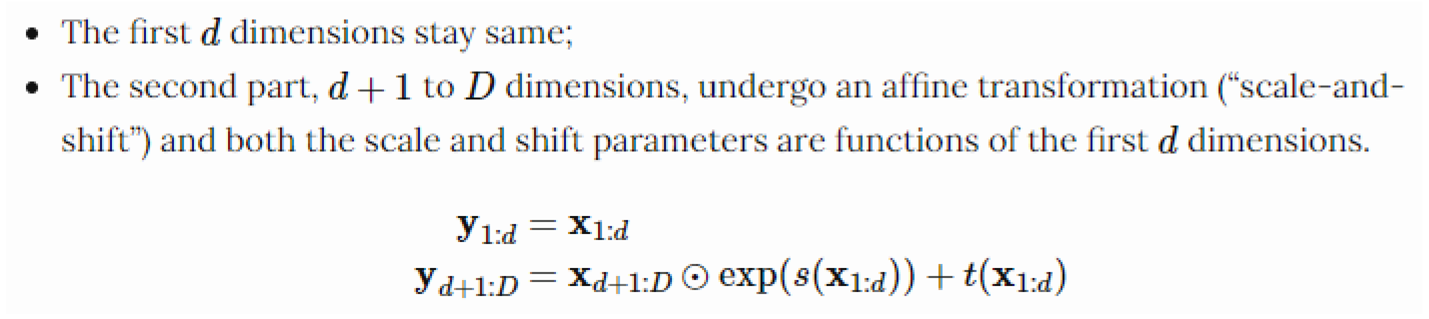
Yes. It is not hard to get the Jacobian matrix and determinant of this transformation. The Jacobian is a lower triangular matrix.

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \text{diag}(\exp(s(\mathbf{x}_{1:d}))) \end{bmatrix}$$

Hence the determinant is simply the product of terms on the diagonal.

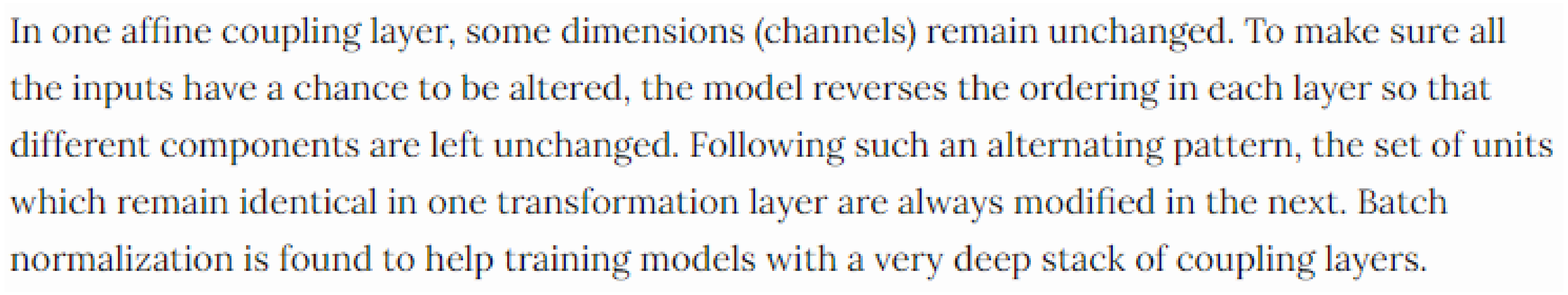
$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp(s(\mathbf{x}_{1:d}))_j = \exp(\sum_{j=1}^{D-d} s(\mathbf{x}_{1:d})_j)$$

Glow



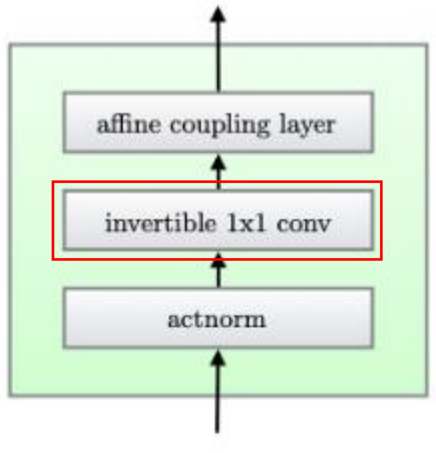
- The first d dimensions stay same;
- The second part, $d+1$ to D dimensions, undergo an affine transformation (“scale-and-shift”) and both the scale and shift parameters are functions of the first d dimensions.

$$\begin{aligned}\mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d})\end{aligned}$$



In one affine coupling layer, some dimensions (channels) remain unchanged. To make sure all the inputs have a chance to be altered, the model reverses the ordering in each layer so that different components are left unchanged. Following such an alternating pattern, the set of units which remain identical in one transformation layer are always modified in the next. Batch normalization is found to help training models with a very deep stack of coupling layers.

Glow



(a) One step of our flow.

1x1 convolution with equal number of input and output channels is a generalization of any permutation of the channel ordering.

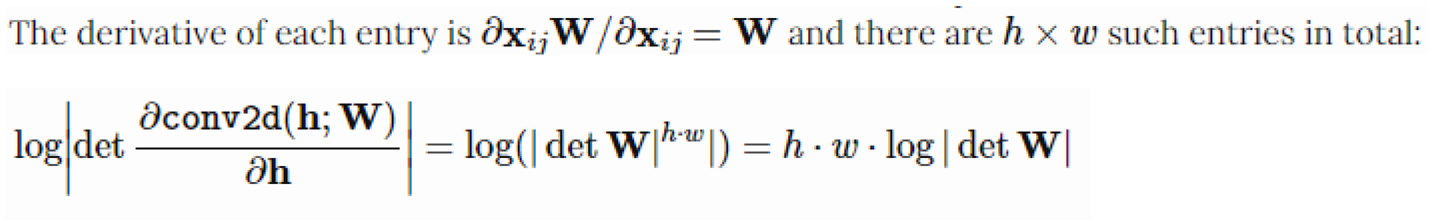
h x w x c tensor h를 weight matrix W of size c x c로 1x1 convolution
Output : h x w x c tensor, f = conv2d(h; W)

$$\log\left|\det \frac{\partial \text{conv2d}(\mathbf{h};\mathbf{W})}{\partial \mathbf{h}}\right| = \log(|\det \mathbf{W}|^{h \cdot w}|) = h \cdot w \cdot \log|\det \mathbf{W}|$$

conv2D(h; W)의 시간복잡도는 O(hwc^2)
det(W)의 시간복잡도는 O(c^3)

LU Decomposition을 통해 O(c)로 줄일수 있음

Glow



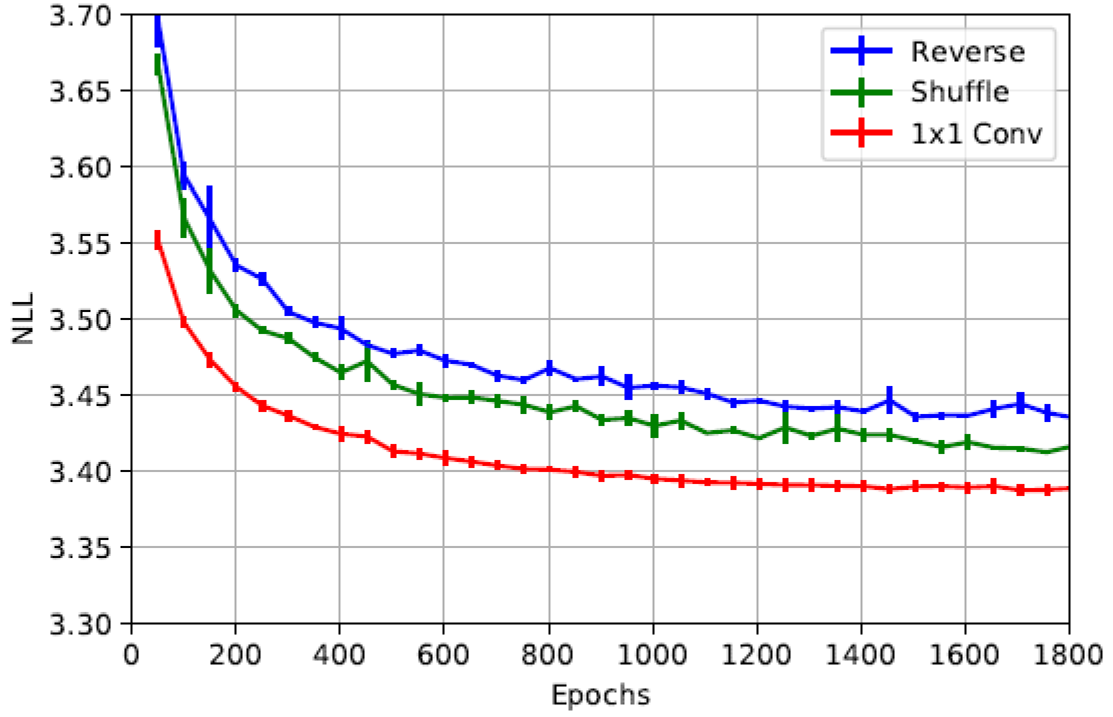
The derivative of each entry is $\partial \mathbf{x}_{ij}\mathbf{W}/\partial \mathbf{x}_{ij} = \mathbf{W}$ and there are $h \times w$ such entries in total:

$$\log\left|\det \frac{\partial \mathbf{conv2d}(\mathbf{h};\mathbf{W})}{\partial \mathbf{h}}\right| = \log(|\det \mathbf{W}|^{h\cdot w}) = h \cdot w \cdot \log|\det \mathbf{W}|$$

log-determinant가 0에서 시작하기위해 W는 random rotation matrix로 설정

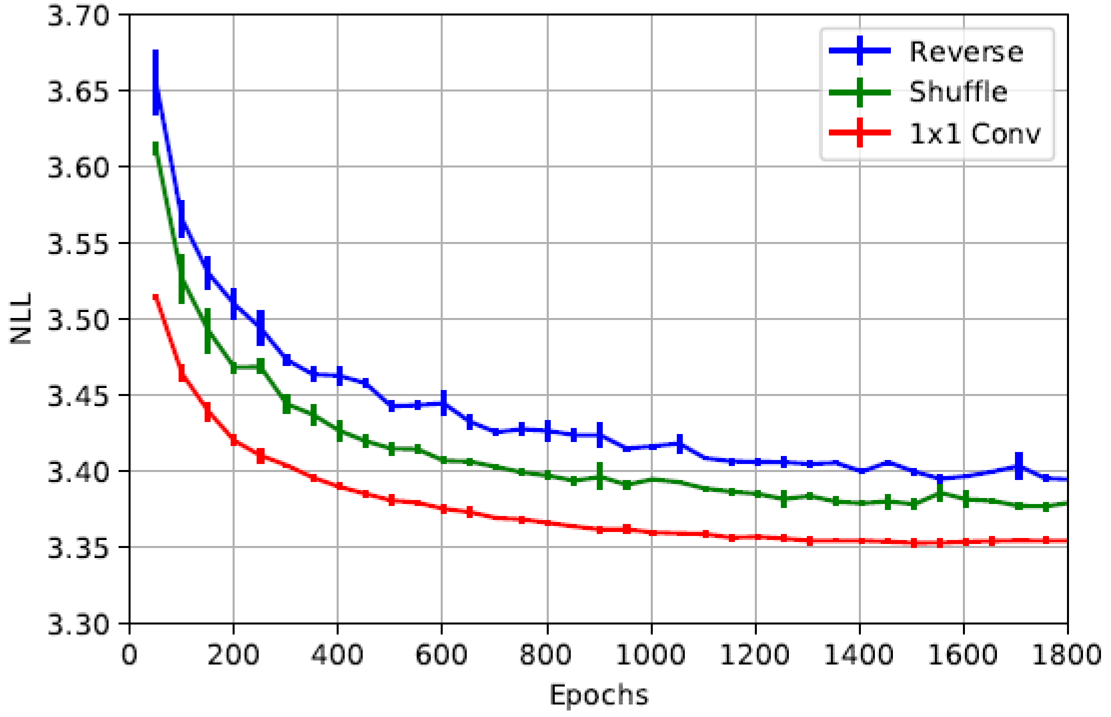
Random rotation matrix : random matrix를 만들어서 qr분해를 한 결과에서 q가 random rotation matrix

Glow



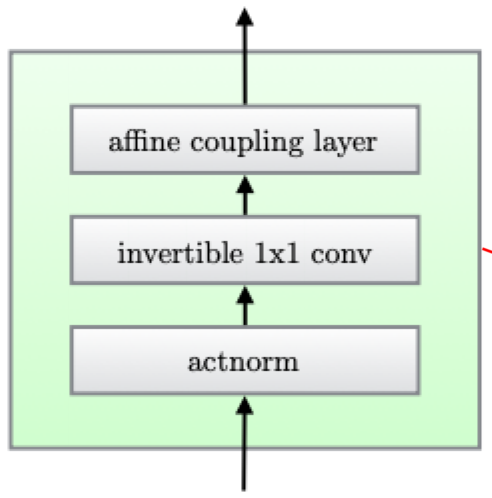
(a) Additive coupling.

S = 1, log-determinant of 0

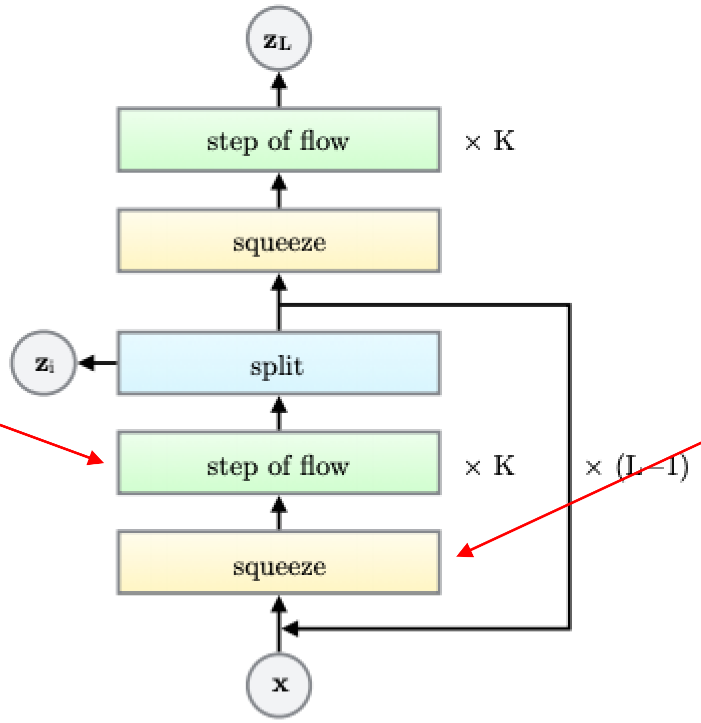


(b) Affine coupling.

Glow

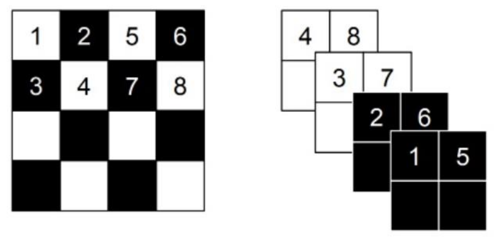


(a) One step of our flow.

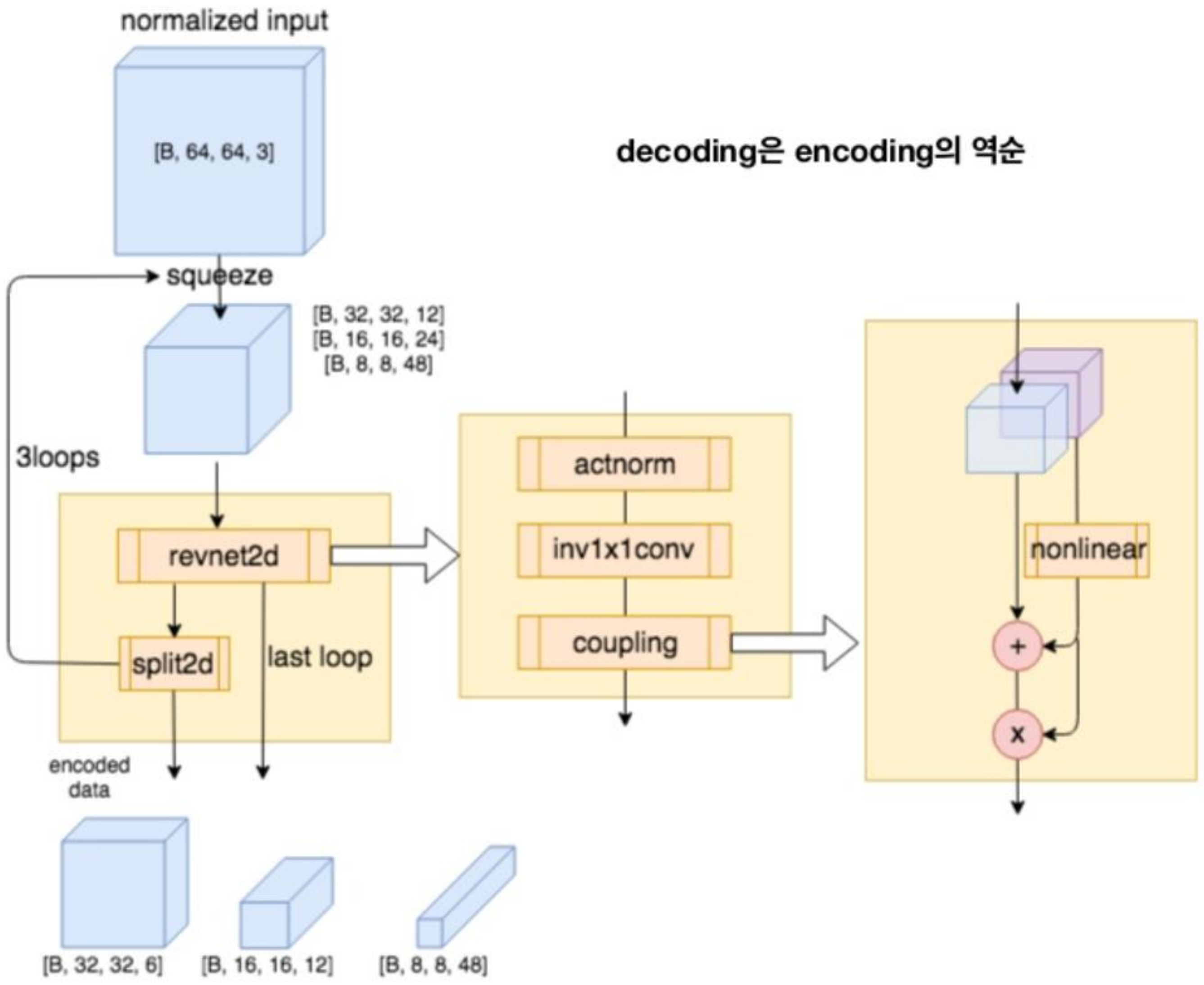


(b) Multi-scale architecture (Dinh et al., 2016).

Masking



Glow


normalized input
[B, 64, 64, 3]
squeeze
[B, 32, 32, 12]
[B, 16, 16, 24]
[B, 8, 8, 48]
3loops
revnet2d
split2d
last loop
encoded data
[B, 32, 32, 6]
[B, 16, 16, 12]
[B, 8, 8, 48]
decoding은 encoding의 역순
actnorm
inv1x1conv
coupling
nonlinear
+
x

Result

Table 2: Best results in bits per dimension of our model compared to RealNVP.

Model	CIFAR-10	ImageNet 32x32	ImageNet 64x64	LSUN (bedroom)	LSUN (tower)	LSUN (church outdoor)
RealNVP	3.49	4.28	3.98	2.72	2.81	3.08
Glow	**3.35**	**4.09**	**3.81**	**2.38**	**2.46**	**2.67**

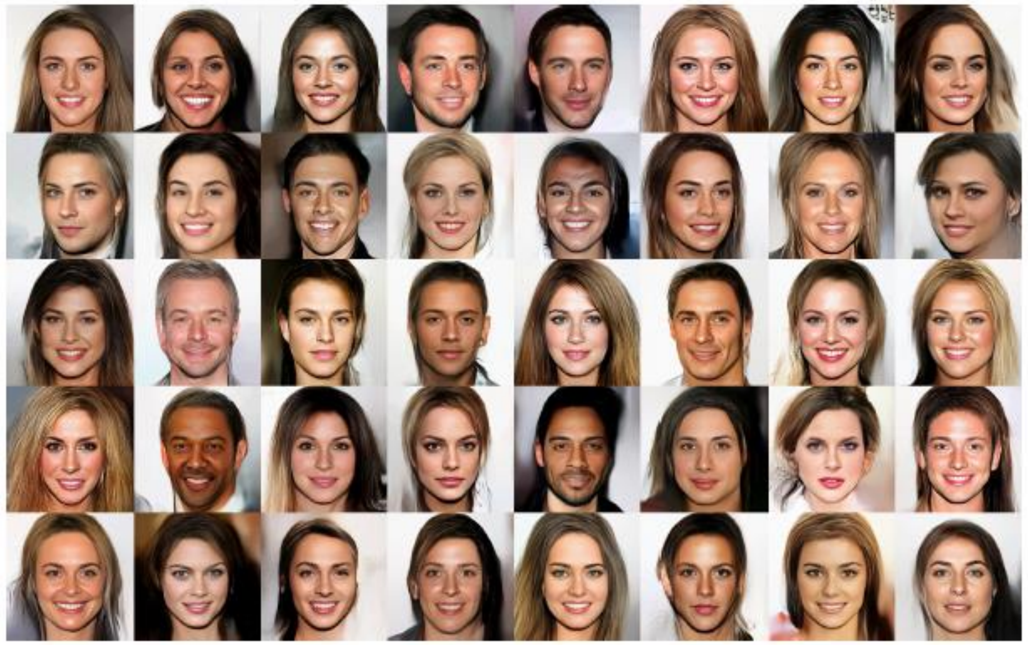


Figure 4: Random samples from the model, with temperature 0.7

Result

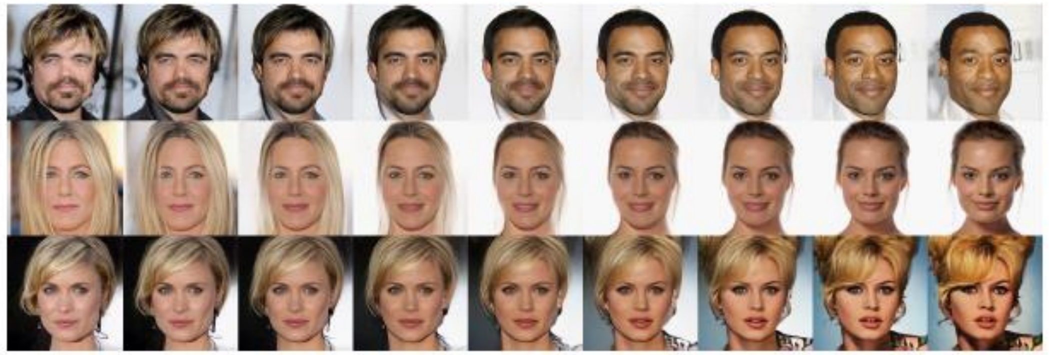
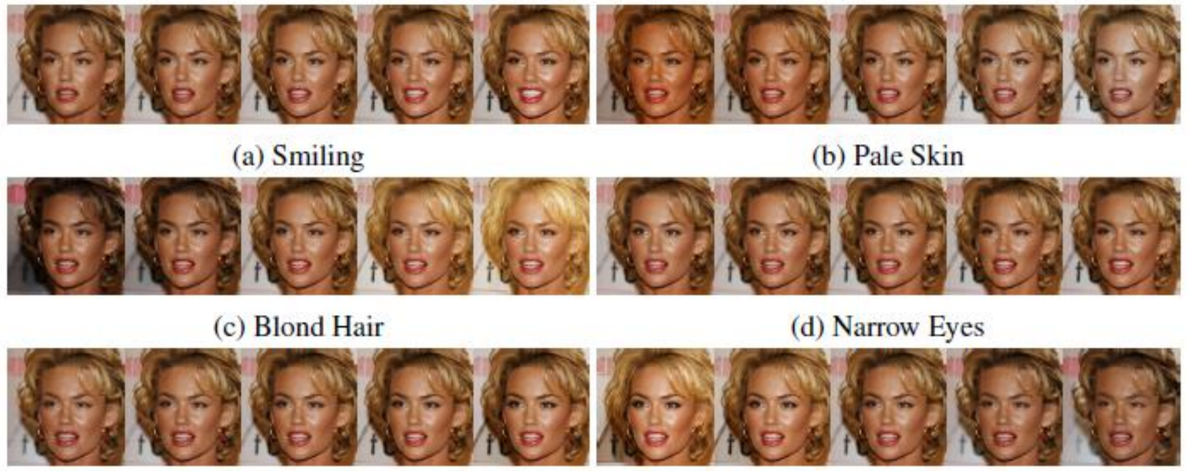


Figure 5: Linear interpolation in latent space between real images



(a) Smiling

(b) Pale Skin

(c) Blond Hair

(d) Narrow Eyes

(e) Young

(f) Male

Figure 6: Manipulation of attributes of a face. Each row is made by interpolating the latent code of an image along a vector corresponding to the attribute, with the middle image being the original image

참고자료

- https://sanghyu.tistory.com/18
- https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html#what-is-normalizing-flows
- https://angeloyeo.github.io/2020/07/24/Jacobian.html