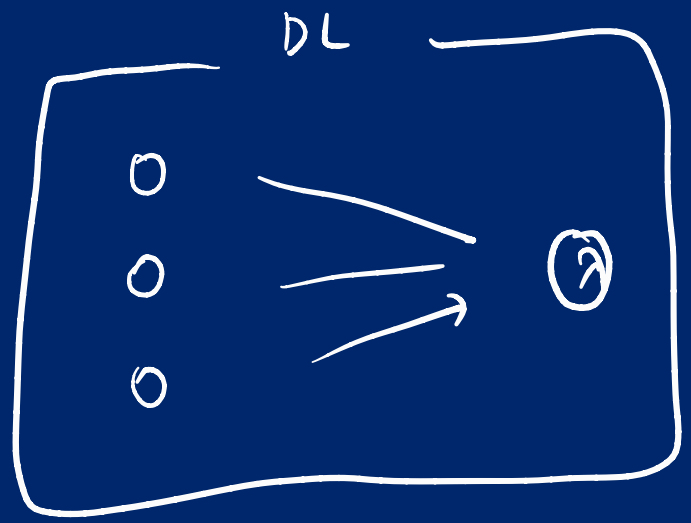
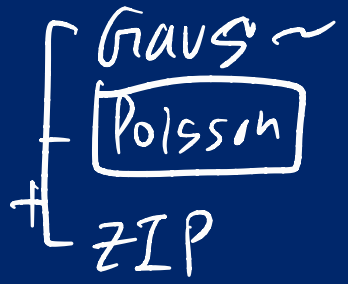
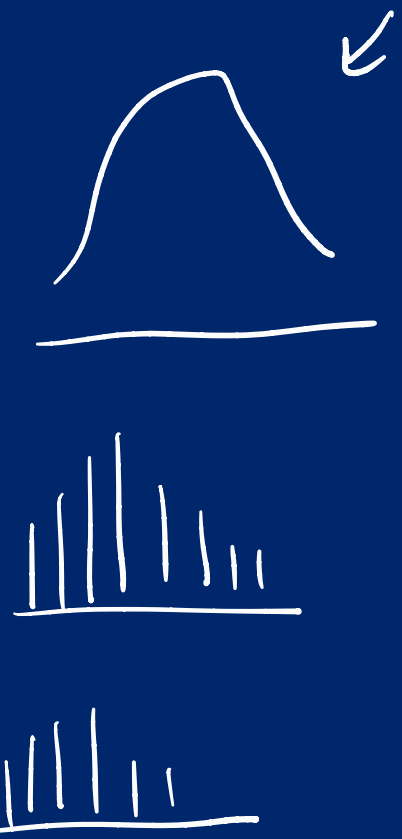
3. Probabilistic DL Models in the Wild

Part 2: Real world distributions





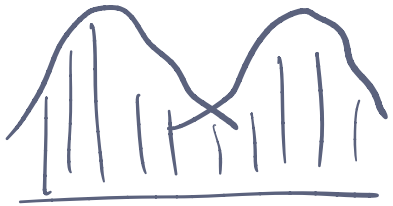


Contents

Complex – Discrete

1. Multinomial Distribution — MNIST 0~9 10 class, $P_0 \sim P_9$, 10^4 θ
2. Discretized Logistic Mixture ← Simple Mix
3. Normalizing Flow

Real-word Data [Sound / Image]



1
Multinomial Distribution

Very flexible CPD for categorical data

Multinomial Distribution

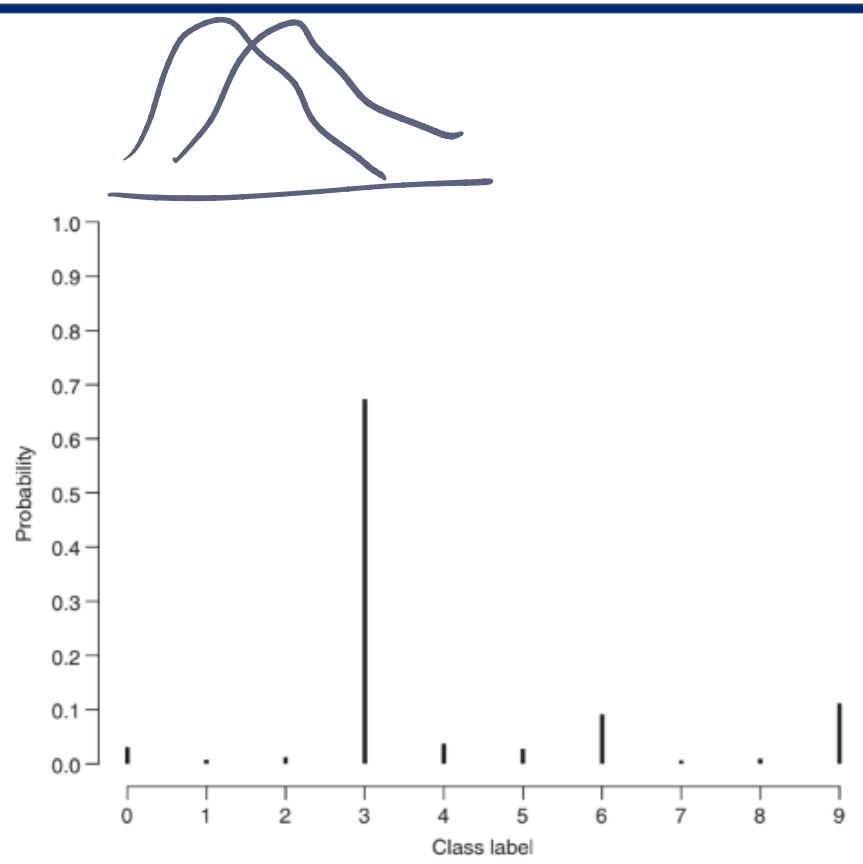


Figure 6.1 Multinomial distribution with ten classes: $MN(p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9)$

According to this criterion, the Multinomial distribution is especially flexible because it has as many parameters as possible values.

e.g. MNIST example - 9 params.

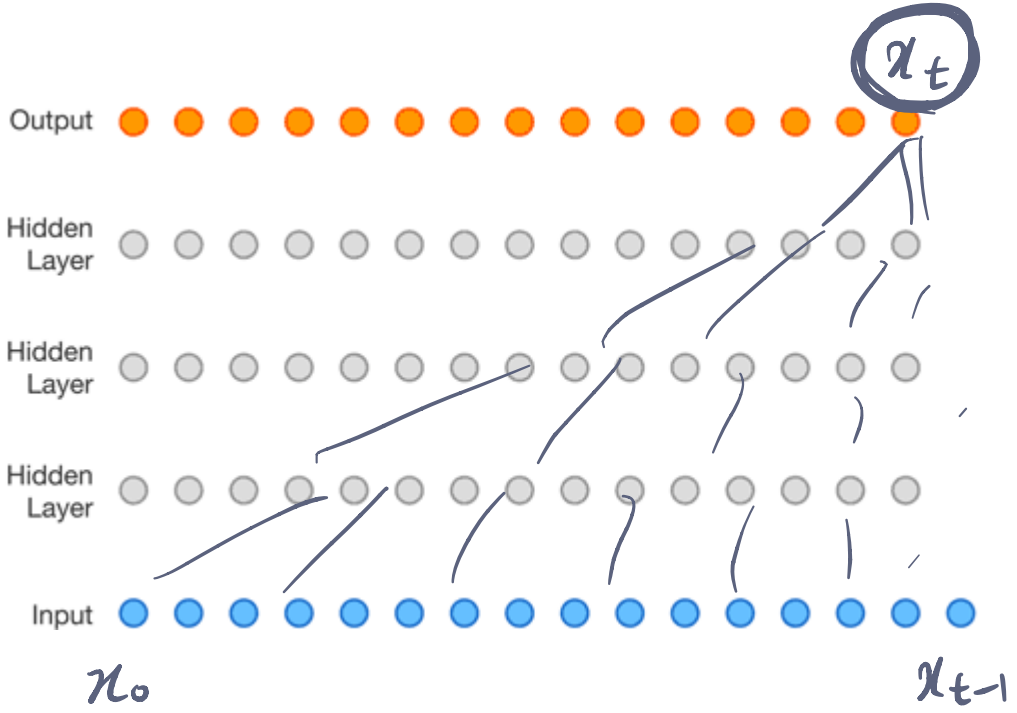
$$Y|X \sim Multinomial(p_1, p_2, \ldots, p_9)$$

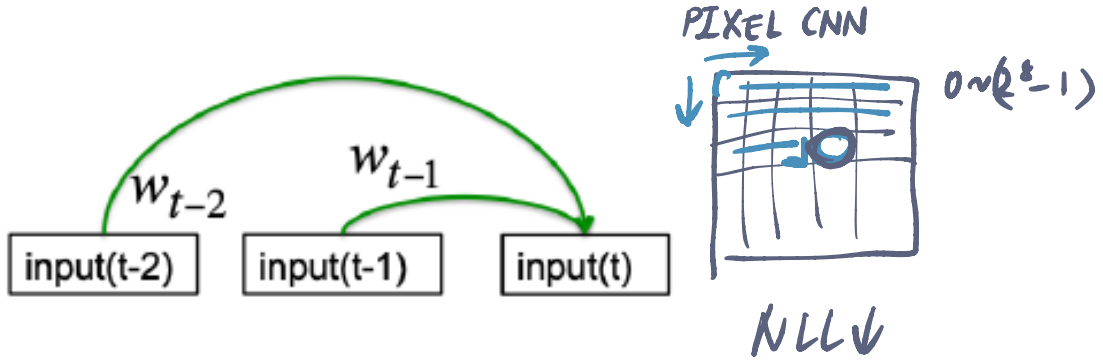
Google - WaveNet (Text → Speech)

1초 16000 sample → 16-bit discretize. $x_t = 0 \sim (2^{16}-1)$ (정수) : 65535

Autoregressive Model – Multinomial as CPD

$$P(x_t) = P(x_t | x_{t-1}, x_{t-2}, \ldots x_0)$$

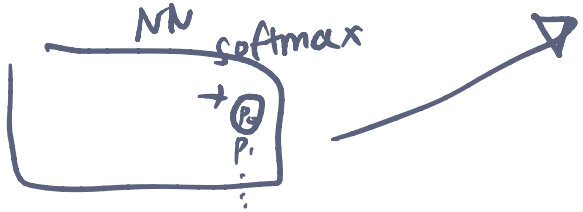




Autoregressive Models

- Predict the next term in a sequence from a fixed number of previous terms using “delay taps”.
- Kind of memoryless models, unlike RNN(Recursive Neural Net).
- Can be generative model also.

nonlinear transf → $0 \sim (2^8-1)$ → Multinomial CPD



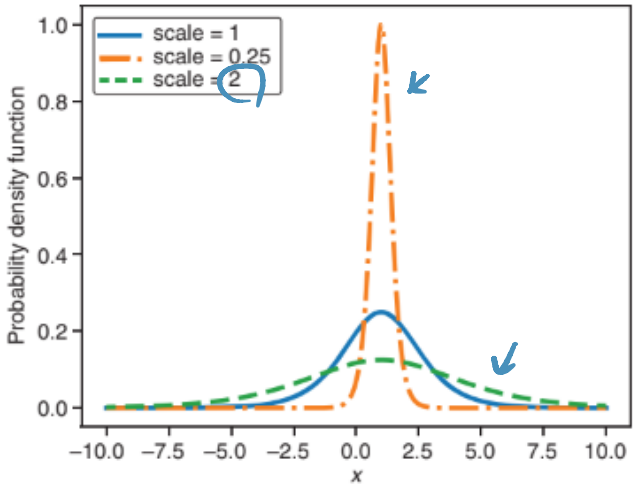
→ Parallel WaveNET

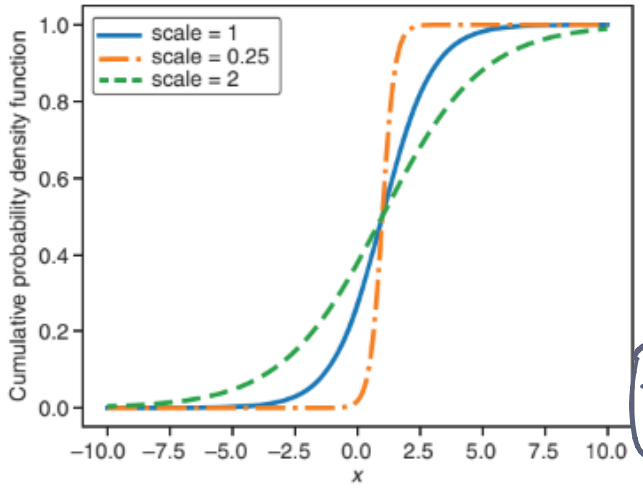
2

Discretized Logistic Mixture Distribution

Less parameters than multinomial CPD, but still efficient

Logistic Distribution





$X \sim Logistic(\mu, s)$

-pdf: $f(x;\mu,s) = \dfrac{e^{-(x-\mu)/s}}{s\left(1+e^{-(x-\mu)/s}\right)^2}$

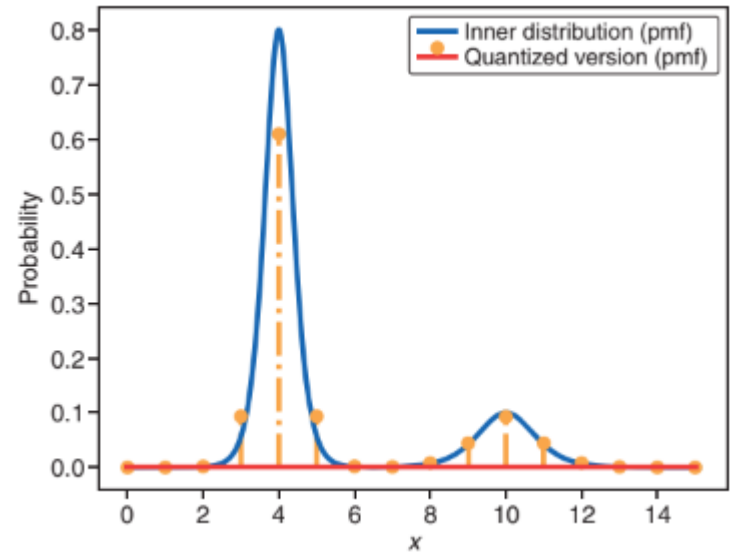
-cdf: $F(x;\mu,s) = \dfrac{1}{1+e^{-(x-\mu)/s}}$ = sigmoid.

$\dfrac{x-\mu}{s}$

<Code Implement>

tfd. Logistic(loc= μ, scale = s)

Logistic Mixture Distribution



$X_1 \sim Logistic(\mu_1, s_1)$

$X_2 \sim Logistic(\mu_2, s_2)$

...

$X_n \sim Logistic(\mu_n, s_n)$

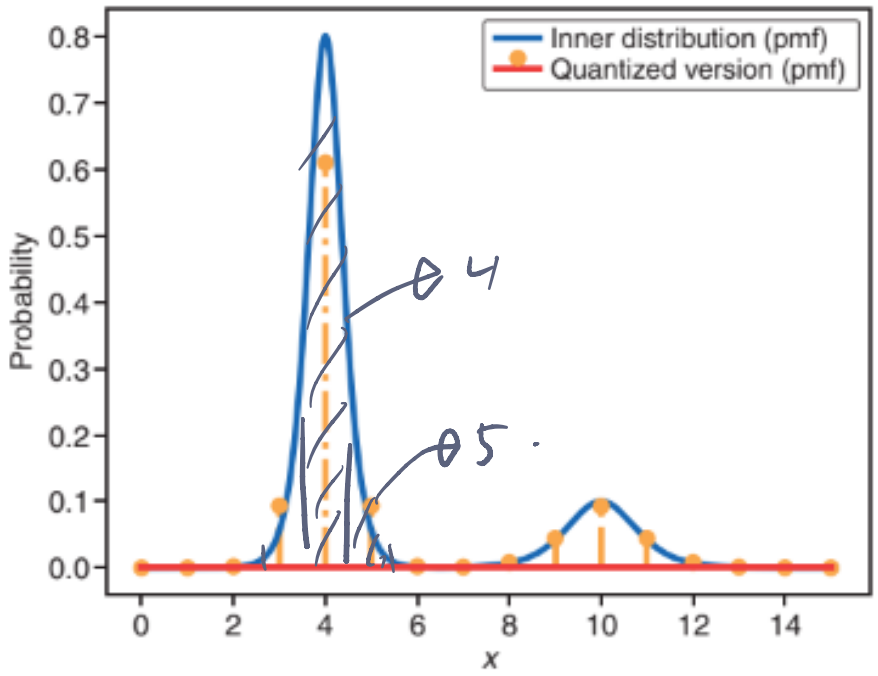
$$X = \sum p_i X_i$$

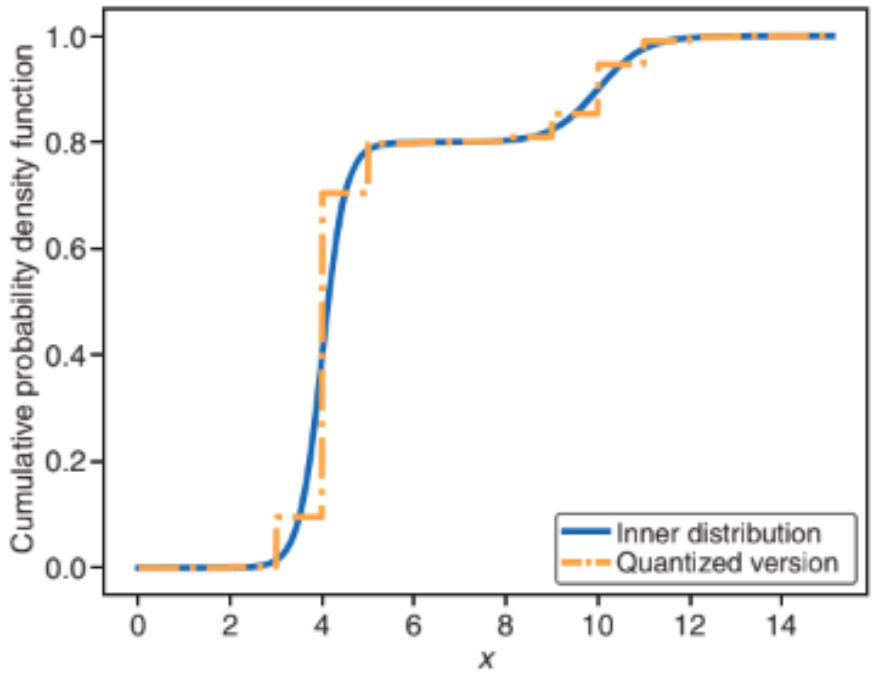
To specify r.v, X, $3n$ parameters required.

By setting mixture distribution,

we can get simple yet flexible distribution.

Logistic Mixture Distribution



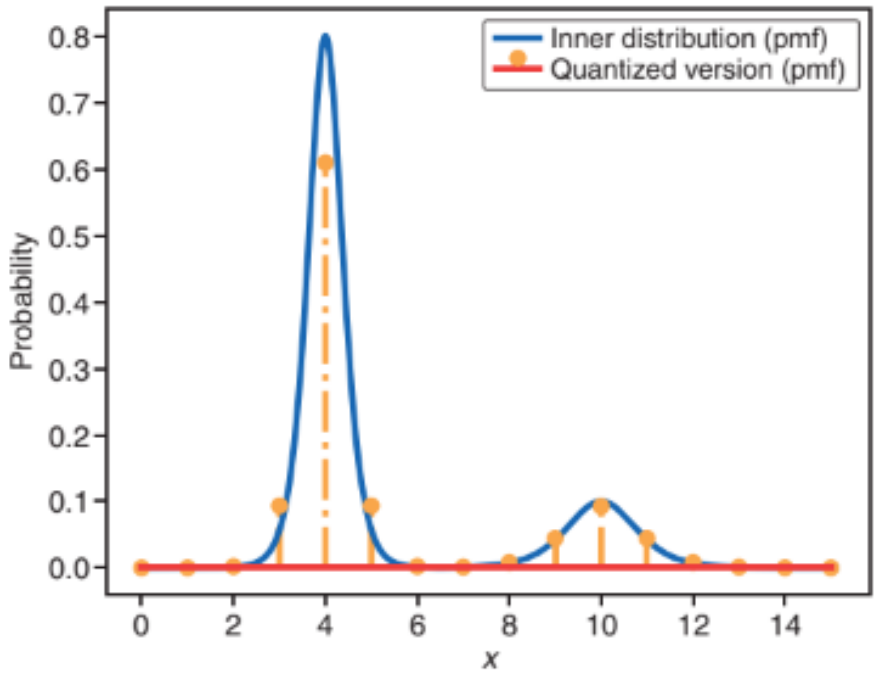


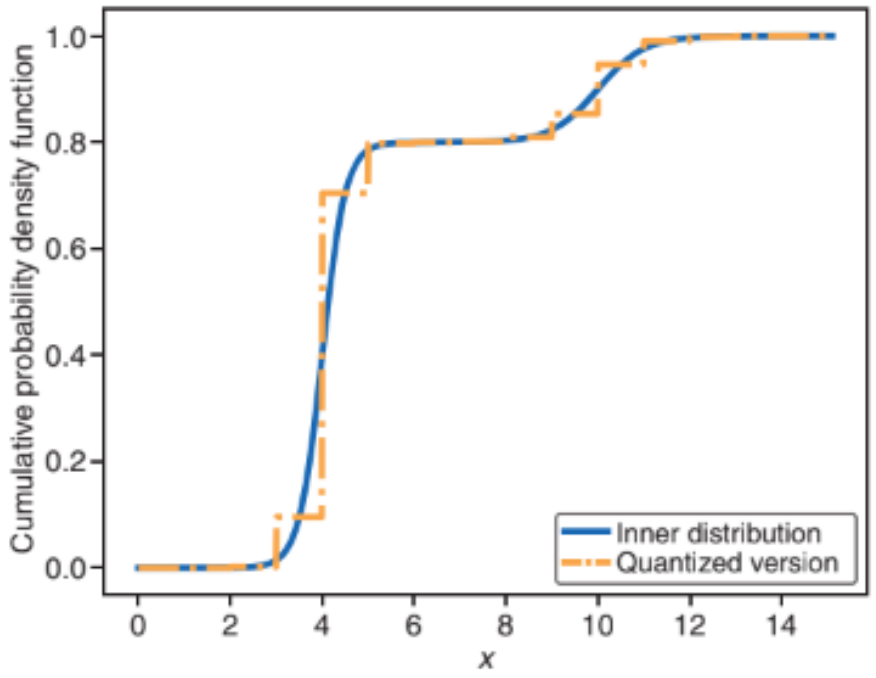
As output need to be discrete, we can discretize it!

(i.e. converting pdf to pmf)

Use NN to find parameters(MaxLike approach)

Why Not Gaussian Mixture?





$3.5 \sim 4.5 \rightarrow p(x=4)$

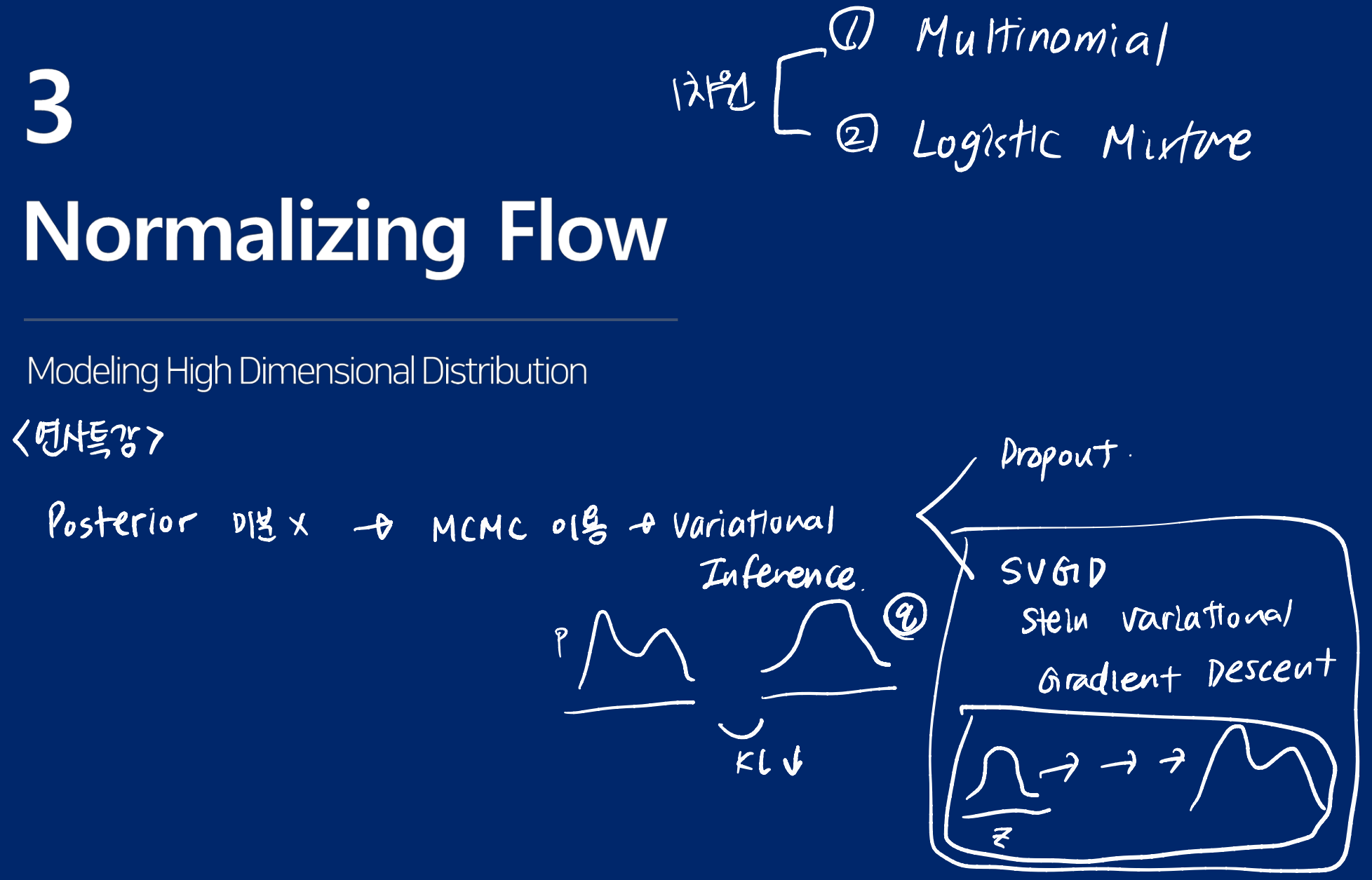
$$\frac{1}{1+e^{-x}} \Big|_{3.5}^{4.5}$$

Need CDF to discretize pdf

: Logistic is easier than Gaussian to use CDF

3 Normalizing Flow

Modeling High Dimensional Distribution

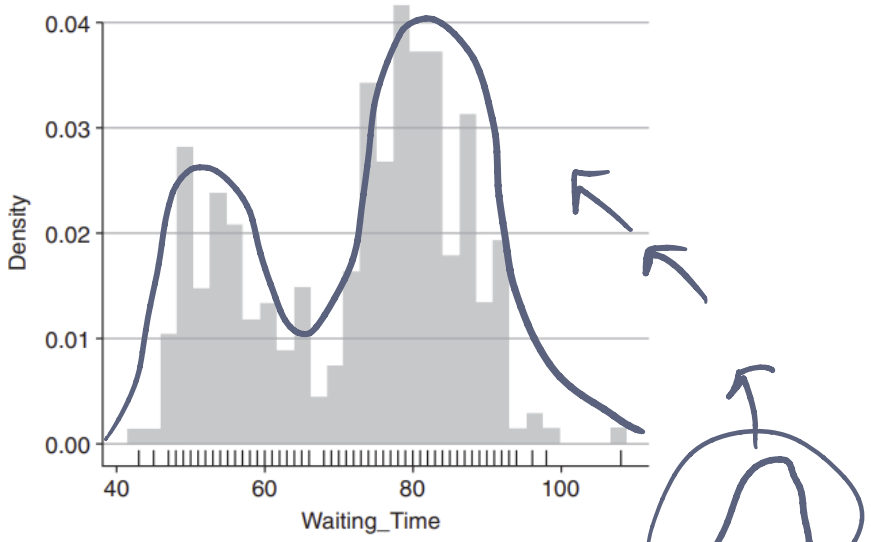


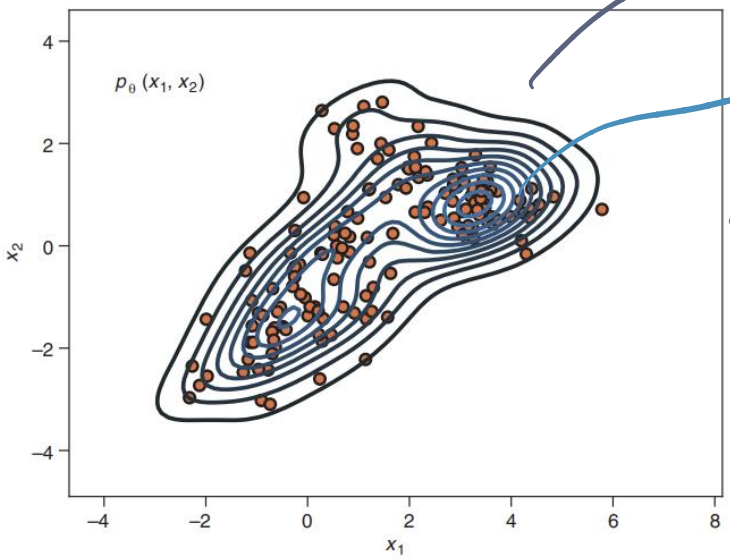
Normalizing Flow

NF의 목적은 복잡한 데이터의 분포를 간단한 분포로부터 변수 변환을 통해 얻어내는 것

1차원의 복잡한 분포는 Logistic Mixture 모델을 통해 구할 수 있었다.

하지만 Logistic Mixture 모델로는 고차원의 복잡한 분포를 설명하기 힘들다 → 따라서 나온 것이 NF!



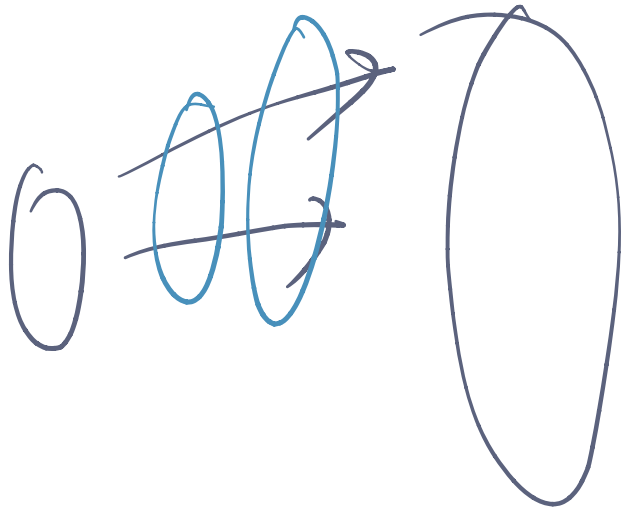


Normalizing Flow – Main Topic

1) Variable Transformation : Indirect computation of likelihood & generating samples

2) Find Bijective Function : Based on MaxLike approach

3) Chain Flow : Stacking more bijective functions



Variable Transformation

Theorem 1.7.1. *Let X be a continuous random variable with pdf $f_X(x)$ and support $\mathcal{S}_X$. Let $Y = g(X)$, where $g(x)$ is a one-to-one differentiable function, on the support of X, $\mathcal{S}_X$. Denote the inverse of g by $x = g^{-1}(y)$ and let $dx/dy = d[g^{-1}(y)]/dy$. Then the pdf of Y is given by*

$$f_Y(y) = f_X(g^{-1}(y)) \left| \frac{dx}{dy} \right|, \quad \text{for } y \in \mathcal{S}_Y, \tag{1.7.11}$$

where the support of Y is the set $\mathcal{S}_Y = \{y = g(x) : x \in \mathcal{S}_X\}$.

$Y = g(X)$

$f_Y(y) = f_X(x) \left| \frac{dx}{dy} \right|$

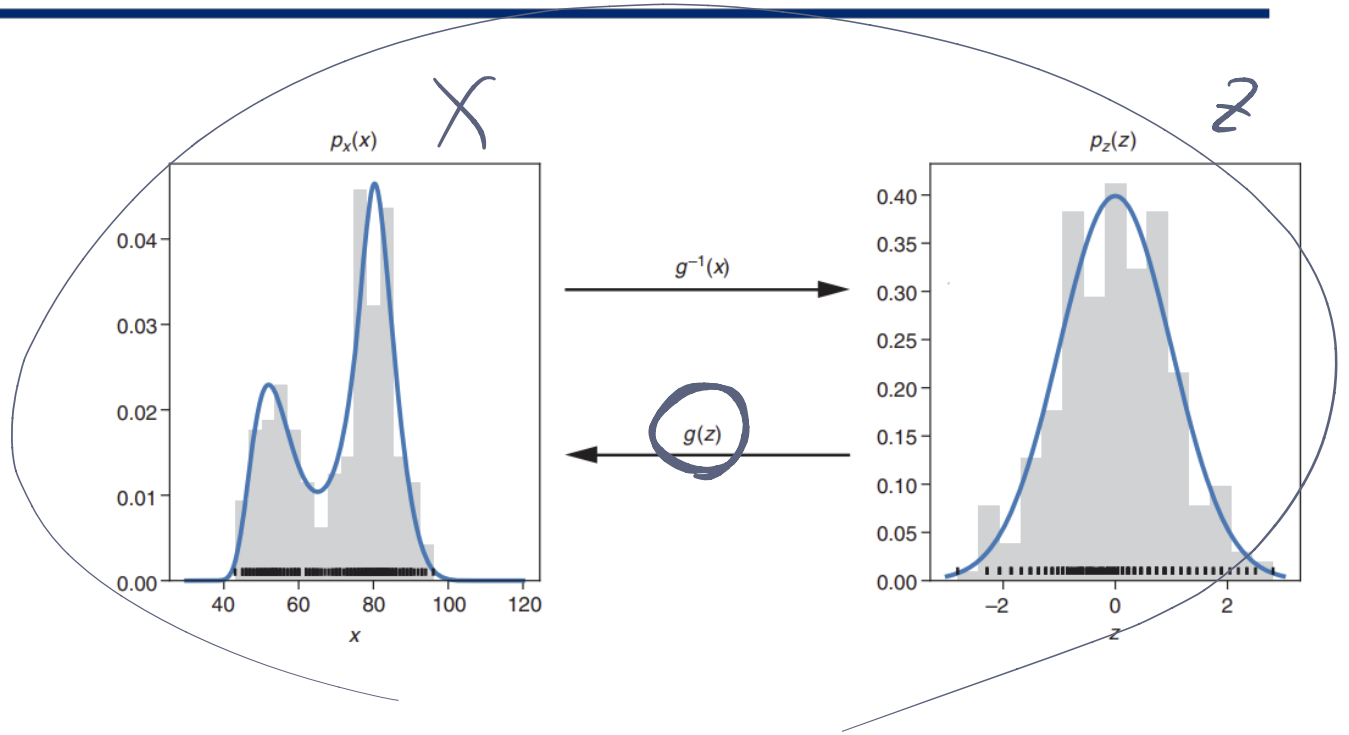
$x \rightarrow g^{-1}(y)$

Variable Transformation을 할 때, Jacobian이 추가로 곱해진다. 이 부분 덕분에 변환된 확률변수의 전체 넓이가 1이 될 수 있고, 이러한 특징 덕분에 Normalizing Flow라고 부른다.

One Dimensional NF

x : 주어진 데이터, **z** : 표준정규분포를 따르는 확률변수

x = g(z), z = g^{-1}(x)



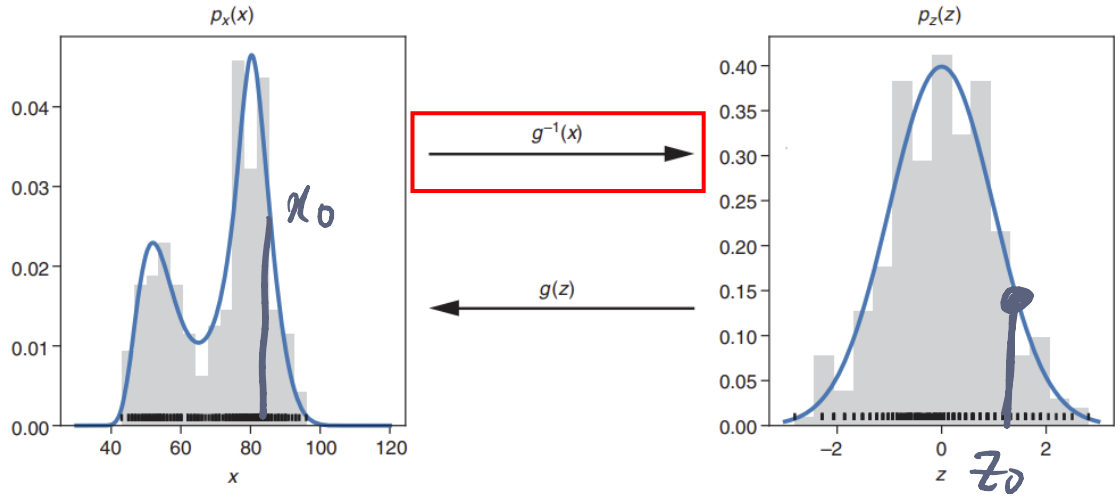
g를 찾아서 좋은걸까?

만약 x와 z의 관계를 잘 설명하는 bijective function g를 찾았다고 가정했을 때:

- 새로운 데이터 x_0의 likelihood를 알 수 있다.
- 실제론 존재하지 않는 데이터를 생성할 수 있다.

Why find g?

1. Likelihood 계산 가능

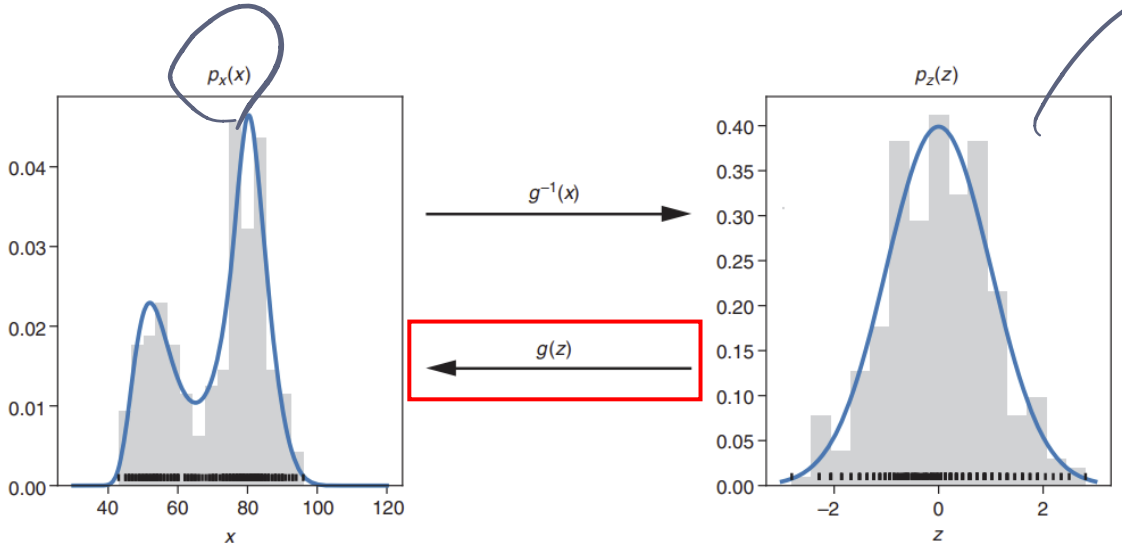


Given $\boldsymbol{x_0}$, we can't compute $\boldsymbol{p_x(x_0)}$ directly

$$\boldsymbol{g^{-1}(x_0) = z_0 \rightarrow p_z(z_0)|J| = p_x(x_0)}$$

Why find g?

2. Sampling 가능 (Data Generation)



$g(z_0) = x_0$

You can't sample from $\boldsymbol{p_x(x)}$ directly

Sample from $\boldsymbol{p_z(z)}$ and than transform!

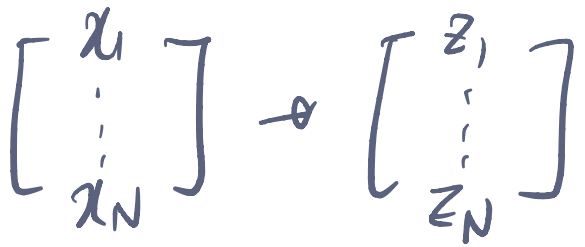
Bijective Function 찾기 – Data Fitting

Variable Transformation 기법을 쓰기 위해선 일단 Bijective function(일대일함수)이라는 조건이 전제

만약 하나의 함수만을 사용한다면, g를 어떠한 파라미터 $\boldsymbol{\theta}$에 의해 매우 flexible하게 변하는 함수로 설정을 하고, MaxLike approach를 통해 θ를 결정지을 수 있다.

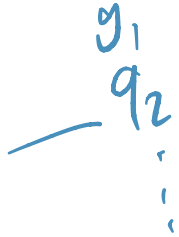
만약 함수 g가 z와 x간의 관계를 잘 설명한다면, 주어진 데이터 $(\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N})'$를 $(\boldsymbol{z_1}, \boldsymbol{z_2}, \ldots, \boldsymbol{z_N})'$로 변환한 후에 계산한 joint probability도 높은 값을 가질 것이다. 따라서 우리는 MaxLike approach를 통해 g를 구할 수 있게 된다.

→ $\boldsymbol{argmax_\theta \prod p_z(g^{-1}(x_i))}$



물론 하나의 함수만을 사용한다면, 아무리 그 함수가 flexible하다고 한들 한계가 있을 것이다.

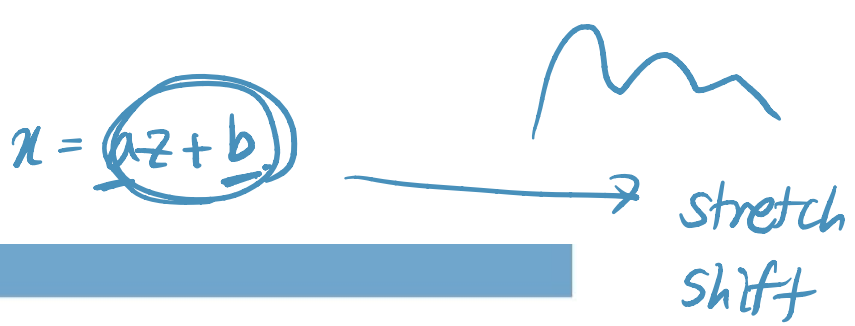
→ Chaining Flows 도입



Bijective Function 찾기 – Data Fitting

<Code Implementation>

What if we want to find g which z ~ N(0, 1) & x ~ N(5, 0.2)



Listing 6.5 A simple example in TFP

```
a = tf.Variable(1.0)
b = tf.Variable(0.0)
bijector = tfb.AffineScalar(shift=b, scale=a)
dist = tfd.TransformedDistribution(distribution=
        tfd.Normal(loc=0,scale=1),bijector=bijector)

optimizer = tf.keras.optimizers.Adam(learning_rate=0.1)

for i in range(1000):
    with tf.GradientTape() as tape:
        loss = -tf.reduce_mean(dist.log_prob(X))
    gradients = tape.gradient(loss,
                dist.trainable_variables)
    optimizer.apply_gradients(
        zip(gradients, dist.trainable_variables))
```

Defines the variables

Sets up the flow using an affine transformation defined by two variables

The NLL of the data

Calculates the gradients for the trainable variables

Applies the gradients to update the variables

a = 0.2006...
b = 0.4997...

Chaining Flows – One Dimensional

$$p_X(x) = p_{z_0}(z_0) \left| \frac{dz_0}{dx} \right|$$

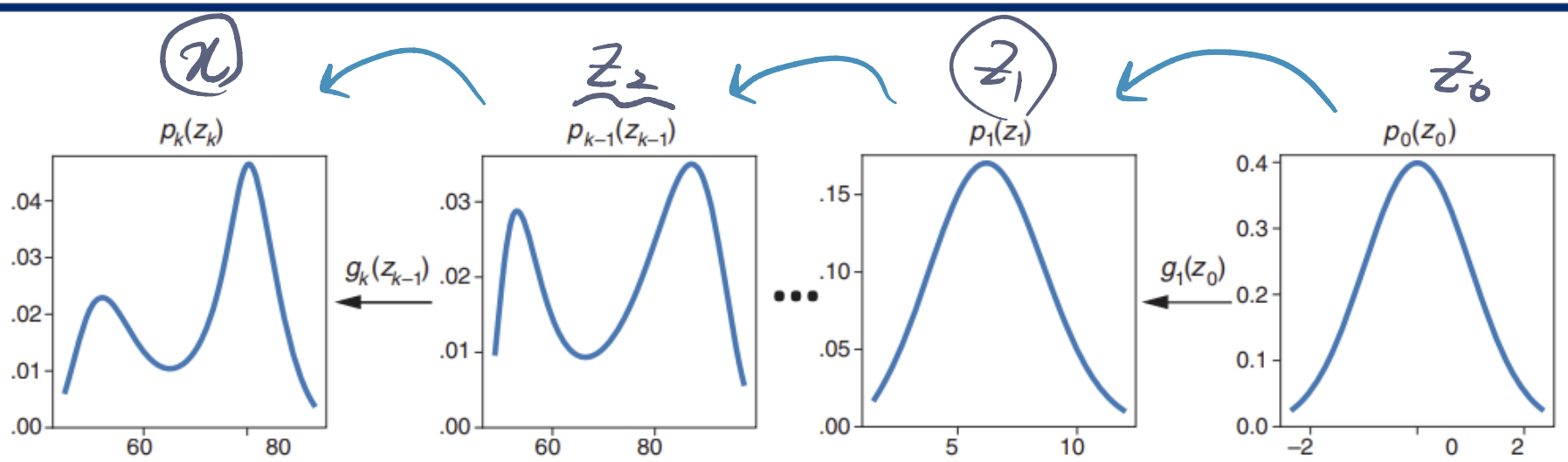


Figure 6.12 A chain of simple transformations makes it possible to create complex transformations needed to model complex distributions. From right to left, starting from a standard Gaussian distribution $z_0 \sim N(0,1)$ changes via successive transformations to a complex distribution with a bimodal shape (on the left).

$$p_X(x) = p_{z_2}(z_2) \left| \frac{dz_2}{dx} \right|$$

$$= p_{z_1}(z_1) \left| \frac{dz_2}{dx} \right| \left| \frac{dz_1}{dz_2} \right|$$

변환을 여러 번 거치는 것을 Chaining flow라고 한다.

: 일반적으로 Linear Transformation 사이에 Non-linear Transformation을 끼워 넣는 식으로 Chain을 구축

$$= p_{z_0}(z_0) \left| \frac{dz_2}{dx} \right| \left| \frac{dz_1}{dz_2} \right| \left| \frac{dz_0}{dz_1} \right|$$

Chaining Flows – One Dimensional

Linear Transformation: g(z) = az + b

- 분포의 모양을 바꾸진 않음.

(즉, z가 가우시안이면, g(z)도 여전히 가우시안을 따름)

따라서 우리는 복잡한 분포를 나타내기 위해

non-linear bijector function인 SinhArcsinh function을 이용

parameter x → Softplus.

skewness μ.

tailweight s.

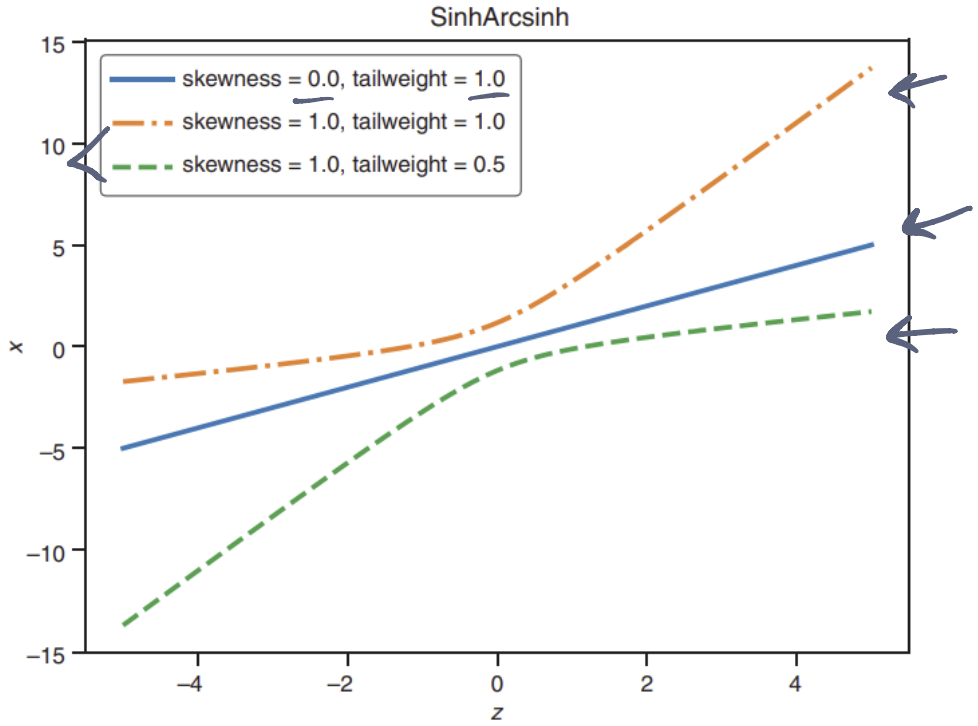
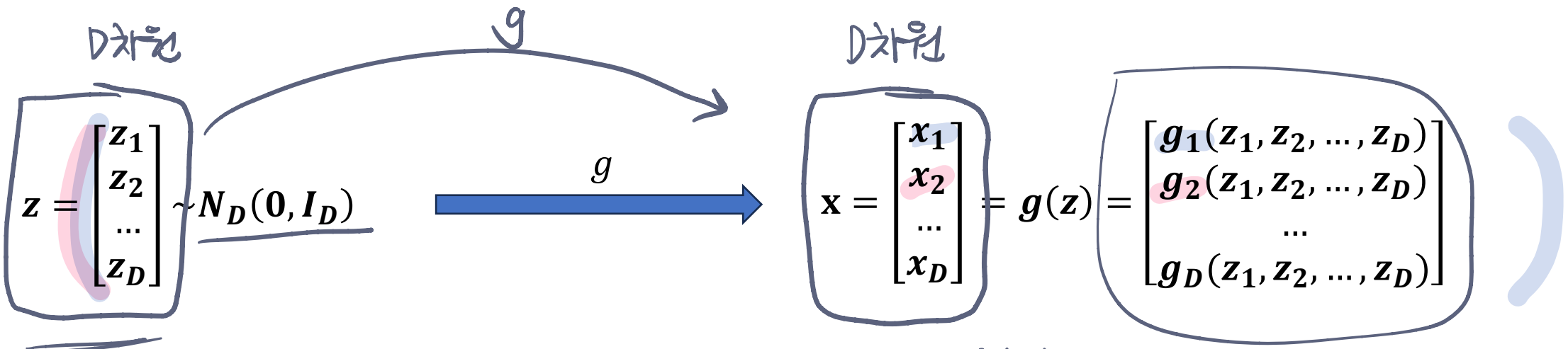


Figure 6.13 The bijector `SinhArcsinh` for different parameter values

Transformation Between High Dimensional Distributions



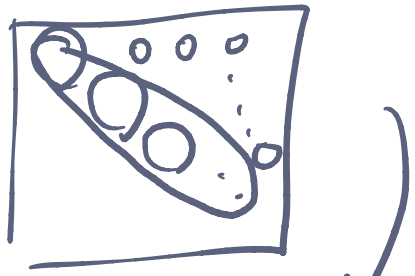
Transformation $p_x(x) = p_z(z) \cdot \left| \det \left(\frac{\partial g(z)}{\partial z} \right) \right|^{-1}$

Jacobian Matrix

$$\frac{\partial g(z)}{\partial z} = \begin{bmatrix} \frac{\partial g_1}{\partial z_1} & \frac{\partial g_1}{\partial z_2} & \cdots & \frac{\partial g_1}{\partial z_D} \\ \frac{\partial g_2}{\partial z_1} & & & \\ \vdots & & & \\ \frac{\partial g_D}{\partial z_1} & \frac{\partial g_D}{\partial z_2} & \cdots & \frac{\partial g_D}{\partial z_D} \end{bmatrix}$$

How to find function g in high dimensions

1. Fully Triangular Flow 삼각형꼴



2. Real Non-Volume Preserving Flow (Real NVP)

Fully Triangular Flow

차원이 높아질수록 Jacobi Matrix의 determinant 계산이 복잡해진다.

→ 따라서 Jacobi Matrix가 삼각행렬이 될 수 있도록 함수 g를 설정해줄 것이다.

$$\frac{\partial g(z)}{\partial z} = \begin{pmatrix} \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_3} \\ \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_3} \\ \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_3} \end{pmatrix} \Rightarrow \frac{\partial g(z)}{\partial z} = \begin{pmatrix} \frac{\partial g_1(z)}{\partial z_1} & \frac{\partial g_1(z)}{\partial z_2} & \frac{\partial g_1(z)}{\partial z_3} \\ \frac{\partial g_2(z)}{\partial z_1} & \frac{\partial g_2(z)}{\partial z_2} & 0 \\ \frac{\partial g_3(z)}{\partial z_1} & \frac{\partial g_3(z)}{\partial z_2} & \frac{\partial g_3(z)}{\partial z_3} \end{pmatrix}$$

z_2, z_3 0 0

$x_1 = g_1(z_1)$ → z_1

$x_2 = g_2(z_1, z_2)$

$x_3 = g_3(z_1, z_2, z_3)$

삼각행렬을 만들기 위해서는 $\frac{\partial g_1(z_1,z_2,z_3)}{\partial z_2} = 0, \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_3} = 0, \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_2} = 0$

이는 i번째 함수 g_i는 z_j $(j > i)$를 포함하지 않아야 한다는 것을 의미한다.

Fully Triangular Flow

$$\frac{\partial g(z)}{\partial z} = \begin{pmatrix} \frac{\partial g_1(z)}{\partial z_1} & 0 & 0 \\ \frac{\partial g_2(z)}{\partial z_1} & \frac{\partial g_2(z)}{\partial z_2} & 0 \\ \frac{\partial g_3(z)}{\partial z_1} & \frac{\partial g_3(z)}{\partial z_2} & \frac{\partial g_3(z)}{\partial z_3} \end{pmatrix}$$

$\det|J|$ g_3

$\prod \frac{\partial g_i(z)}{\partial z_i}$

$z_0\ z_1\ z_2\ z_3$

$g(z) = az + b$

$g_i(z_i) = a_i z_i + b_i$

$= a_i(z_0, \cdots, z_{i-1}) \times z_i + b_i(z_0, \cdots, z_{i-1})$

$= \exp(\alpha_i(z_0, \cdots, z_{i-1}) \times z_i + b_i(z_0, \cdots, z_{i-1})$

삼각행렬의 Determinant : 대각 원소들의 곱

→ g를 앞서 보았던 기준에 따라 설정한다면, determinant 계산을 위해 3번만 편미분을 진행하면 된다(D차원 분포라면 D번 편미분)

→ 조금 더 자세히 보면, i번째 함수 g_i가 z_i에 의해 편미분되는 것만 고려하면 된다

이때, g_i가 z_i에 Linear한 함수라면(i.e., $g_i(z) = az_i + b$), 편미분이 간편해지기 때문에 이 점을 반영하여 함수를 설정할 것이다.

따라서 만약 D차원 분포를 다루는 상황이라면, g_i를 다음과 같이 나타낼 수 있다

→ $g_i(z_1, z_2, \dots, z_D) = g_i(z_1, z_2, \dots, z_i) = b_i(z_1, z_2, .., z_{i-1}) + exp\big(\alpha_i(z_1, z_2, .., z_{i-1})\big) * z_i$

이렇게 설정하면, 비록 z_i에 대해서는 linear transformation일지 몰라도, slope와 bias를 non-linear한 방식으로 계산할 수 있기에 매우 flexible하게 함수를 fitting할 수 있다.

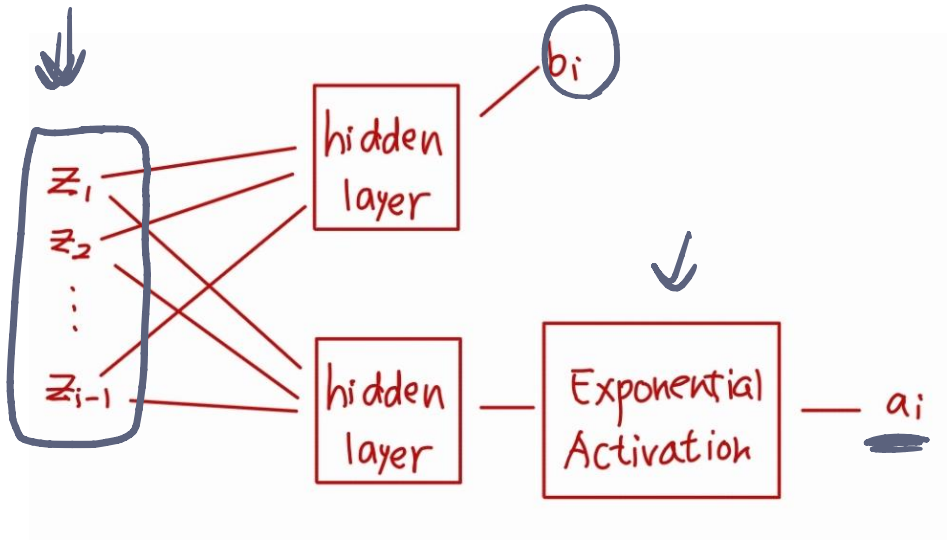
Fully Triangular Flow

$$g_i(z_1, z_2, \dots, z_D) = g_i(z_1, z_2, \dots, z_i) = b_i(z_1, z_2, .., z_{i-1}) + exp\left(\alpha_i(z_1, z_2, .., z_{i-1})\right) * z_i$$

i 번째 slope: $exp\left(\alpha_i(z_1, z_2, .., z_{i-1})\right)$

i 번째 bias: $b_i(z_1, z_2, .., z_{i-1})$

Use Neural Network to get slope and bias!



Why Exponential Activation?

- If slope = 0, then determinant is zero

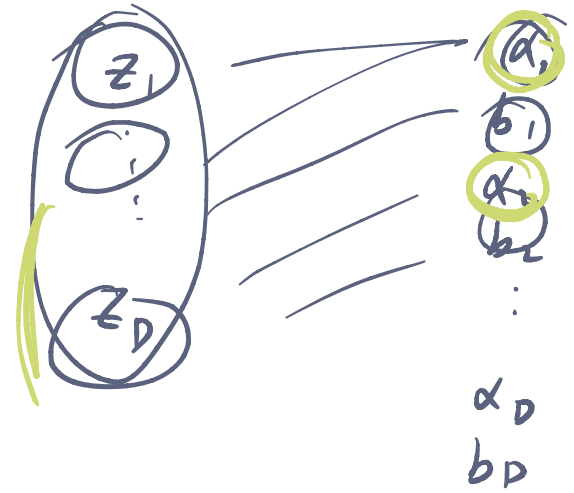
Fully Triangular Flow - Data Fitting

$$p_x(x) = p_z\left(g^{-1}(x)\right)\left|det\left(\frac{\partial g(z)}{\partial z}\right)\right|^{-1}$$

1차원 분포에서와 마찬가지로 MaxLike 방법을 사용해서 g의 파라미터들을 찾을 것이다.

위의 식의 우변 부분을 Maximize하기 위해서는 $z = g^{-1}(x)$를 알아야 한다.

z를 한번에 다 계산할 수는 없으므로 아래와 같이 sequential한 방법으로 계산해야 한다.



$$z_1 = x_1$$
$$z_2 = \frac{x_2 - b_2(z_1)}{\exp(\alpha_2(z_1))}$$
$$z_3 = \frac{x_3 - b_3(z_1, z_2)}{\exp(\alpha_3(z_1, z_2))}$$
$$z_4 = \frac{x_4 - b_4(z_1, z_2, z_3)}{\exp(\alpha_4(z_1, z_2, z_3))}$$

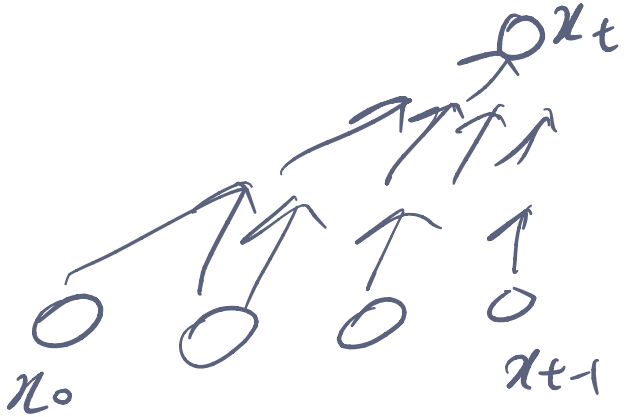
$$x_i = \exp(\alpha_i(z_1 \cdots)) \cdot z_i + b_i(z_1 \cdots)$$

$$x_1 = z_1$$

$$x_2 = \exp(\alpha_2(z_1)) \times z_2 + b_2(z_1)$$

Autoregressive.

$x_t \to x_0 \sim x_{t-1}$.

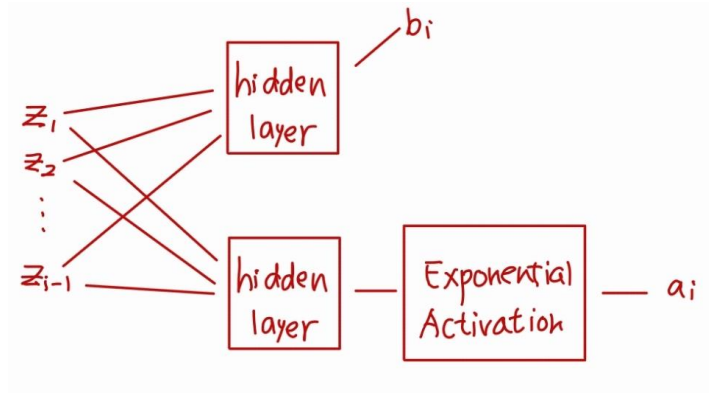


Real Non-volume preserving flow (Real NVP)

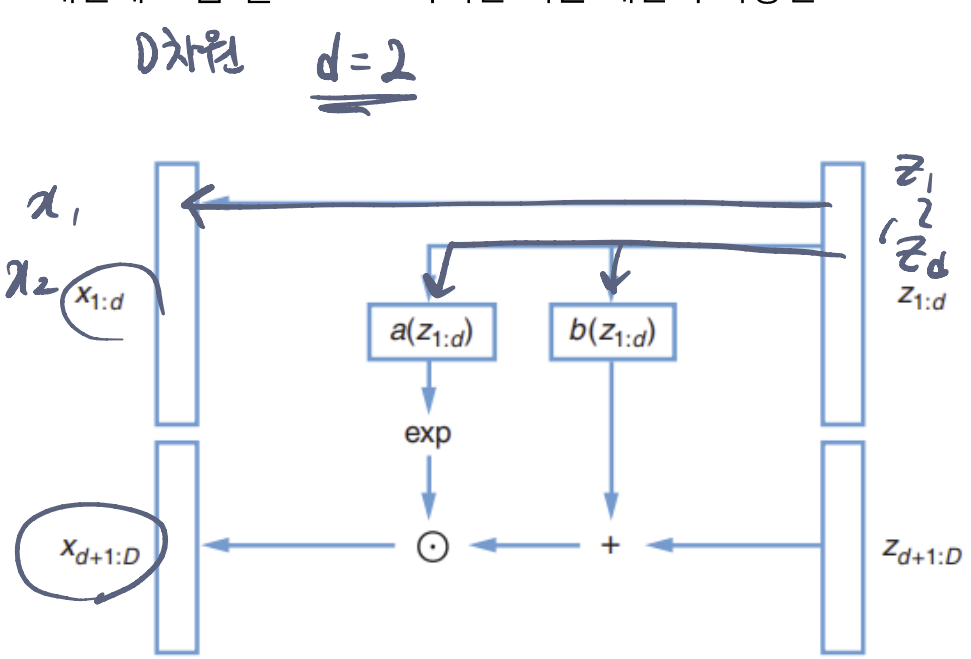
오른쪽 NN은 하나의 함수 g_i를 구하기 위한 NN이다.

따라서 전체 함수들을 다 구해주기 위해선 D개의 NN이 필요

➔ 모두 fitting시키는 건 많은 시간이 소요된다.



대신에 조금 덜 flexible하지만 빠른 계산이 가능한 **Real NVP**라는 것을 사용하기도 한다.



$$x_1 = g_1(z_1) = z_1$$
$$x_2 = g_2(z_2) = z_2$$
$$x_3 = g_3(z_1, z_2, z_3) = b_3(z_1, z_2) + \exp(\alpha_3(z_1, z_2)) \cdot z_3$$
$$x_4 = g_4(z_1, z_2, z_4) = b_4(z_1, z_2) + \exp(\alpha_4(z_1, z_2)) \cdot z_4$$
$$x_5 = g_5(z_1, z_2, z_5) = b_5(z_1, z_2) + \exp(\alpha_5(z_1, z_2)) \cdot z_5$$

$$\frac{\partial g}{\partial z} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \frac{\partial g_3}{\partial z_1} & \frac{\partial g_3}{\partial z_2} & e^{\alpha_3} & 0 & 0 \\ \frac{\partial g_4}{\partial z_1} & \frac{\partial g_4}{\partial z_2} & 0 & e^{\alpha_4} & 0 \\ \frac{\partial g_5}{\partial z_1} & \frac{\partial g_5}{\partial z_2} & 0 & 0 & e^{\alpha_5} \end{pmatrix}$$

$$z_1 = x_1$$
$$z_2 = x_2$$
$$z_3 = \frac{x_3 - \mu_1(z_1, z_2)}{\exp(\alpha_3(z1, z2))}$$
$$z_4 = \frac{x_4 - \mu_2(z1, z2)}{\exp(\alpha_4(z1, z2))}$$
$$z_5 = \frac{x_5 - \mu_3(z1, z2)}{\exp(\alpha_5(z1, z2))}$$

Stack More Layers

1차원 분포에서 봤듯이 여러 번 변환을 거칠 수도 있다!

Listing 6.7 The simple example of a Real NVP TFP

```
bijectors=[]
num_blocks = 5
h = 32
for i in range(num_blocks):
    net = tfb.real_nvp_default_template(
                [h, h])
    bijectors.append(
        tfb.RealNVP(shift_and_log_scale_fn=net,
                    num_masked=num_masked))
    bijectors.append(tfb.Permute([1,0]))
    self.nets.append(net)
bijector = tfb.Chain(list(reversed(bijectors[:-1])))

self.flow = tfd.TransformedDistribution(
    distribution=tfd.MultivariateNormalDiag(loc=[0., 0.]),
    bijector=bijector)
```

Number of hidden layers in the NF model

Size of the hidden layers

Defines the network

Distribution of z with two independent Gaussians

Adds num_blocks of coupling permutations to the list of bijectors

A shift and flow with parameters from the network

Permutation of coordinates