# Introduction of Variational Autoencoder

Ken'ichi Matsui

Today's main topic is

# Variational Autoencoder

which is basic model of
Generative Model
with Deep Learning.

# This presentation is a review of this paper.



## Auto-Encoding Variational Bayes

**Diederik P. Kingma**
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

**Max Welling**
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

### Abstract

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions is two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

### 1 Introduction

How can we perform efficient approximate inference and learning with directed probabilistic models whose continuous latent variables and/or parameters have intractable posterior distributions? The variational Bayesian (VB) approach involves the optimization of an approximation to the intractable

https://arxiv.org/pdf/1312.6114.pdf

# What is Generative Model?

OpenAI

JUNE 16, 2016

# Generative Models

One of our core aspirations at OpenAI is to develop algorithms and techniques that endow computers with an understanding of our world.

It's easy to forget just how much you know about the world: you understand that it is made up of 3D environments, objects that move, collide, interact; people who walk, talk, and think; animals who graze, fly, run, or bark; monitors that display information encoded in language about the weather, who won a basketball game, or what happened in 1970.

This tremendous amount of information is out there and to a large extent easily accessible — either in the physical world of atoms or the digital world of bits. The only tricky part is to develop models and algorithms that can analyze and understand this treasure trove of data.

**Generative models are one of the most promising approaches towards this goal**. To train a generative model we first collect a large amount of data in some domain (e.g., think millions of images, sentences, or sounds, etc.) and then train a model to generate data like it. The intuition behind this approach follows a famous quote from Richard Feynman:

Andrej Karpathy, Pieter Abbeel, Greg Brockman, Peter Chen, Vick Cheung, Rocky Duan, Ian Goodfellow, Durk Kingma, Jonathan Ho, Rein Houthooft, Tim Salimans, John Schulman, Ilya Sutskever, and Wojciech Zaremba.

One of our core aspirations at OpenAI is to develop algorithms and techniques that endow computers with an understanding of our world.

Feynman's message !

*"What I cannot create, I do not understand."*

—Richard Feynman

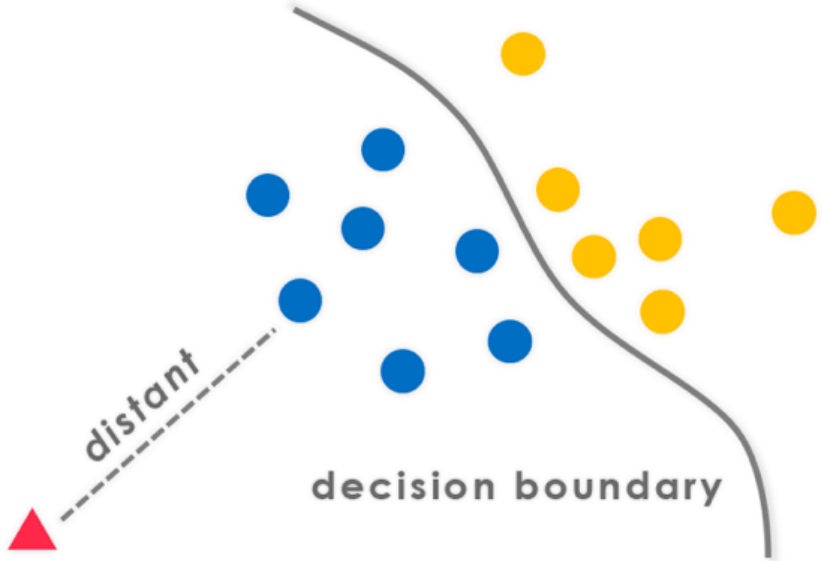
The trick is that the neural networks we use as generative models have a number of parameters significantly smaller than the amount of data we train them on, so the models are forced to discover and efficiently internalize the essence of the data in order to generate it.

Generative models have many short-term applications. But in the long run, they hold the potential to automatically learn the natural features of a dataset, whether categories or dimensions or something else entirely.

https://blog.openai.com/generative-models/

# What is Generative Model

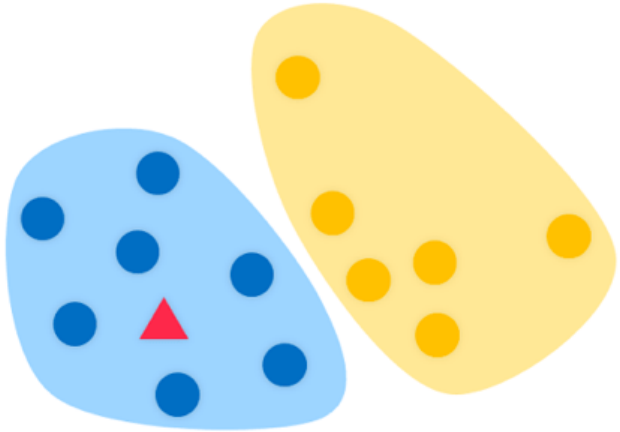
## Discriminative Model




Modeling conditional probability of Class C when Data X was given.

$$p(C_k|X)$$

## Generative Model



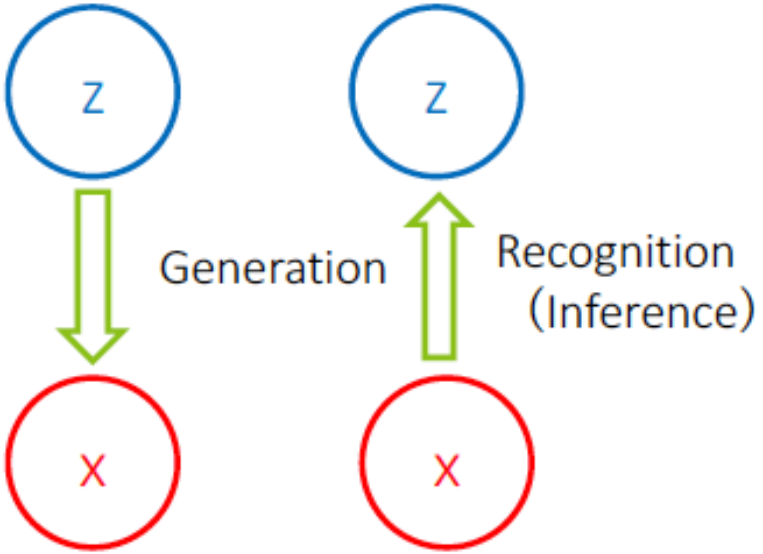
Modeling simultaneous probability of Data X and Class C, understanding the distribution of data X. Data sampling is also enable from the distribution.

$$p(X, C_k)$$

Ref： https://duphan.wordpress.com/tag/generative-model/

# Generation and recognition (1/2)

- Data x is generated from unknown factors z
- Generation and recognition are inverse operations




Inference: Infer the posterior
P(z|x)

E.g. Image generation, recognition

z : Object, Position of camera, Lighting conditio
(Dragon, [10, 2, -4], white)

Generation

Recognition

x : Image

Ref: https://www.slideshare.net/pfi/deep-learning-practice-and-theory

# Generative Models

Given training data, generate new samples from same distribution

Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Approx. with $p_{model}$ (this time, NN is used)

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

True distribution (Unknown)

Addresses density estimation, a core problem in unsupervised learning

**Several flavors:**

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it



Ref： http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

# Taxonomy of Generative Models

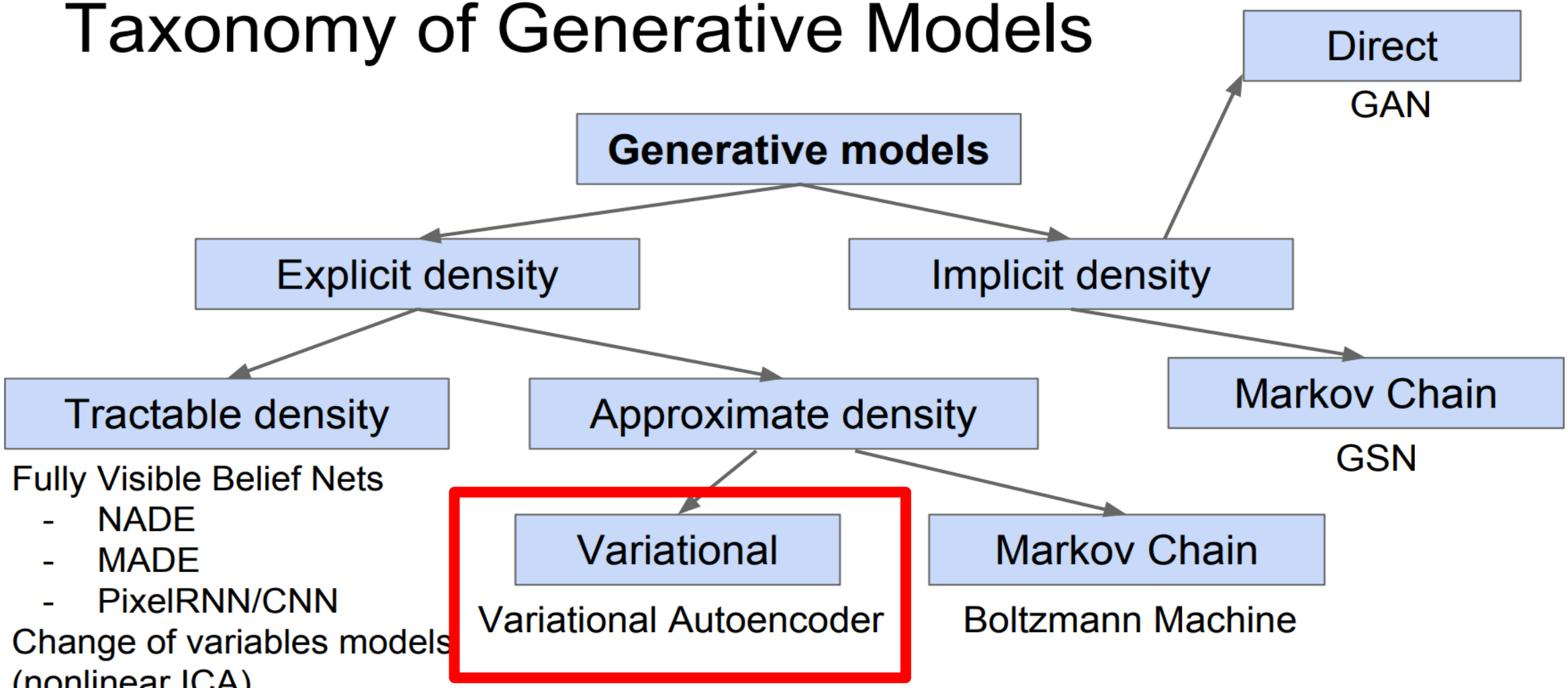



Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

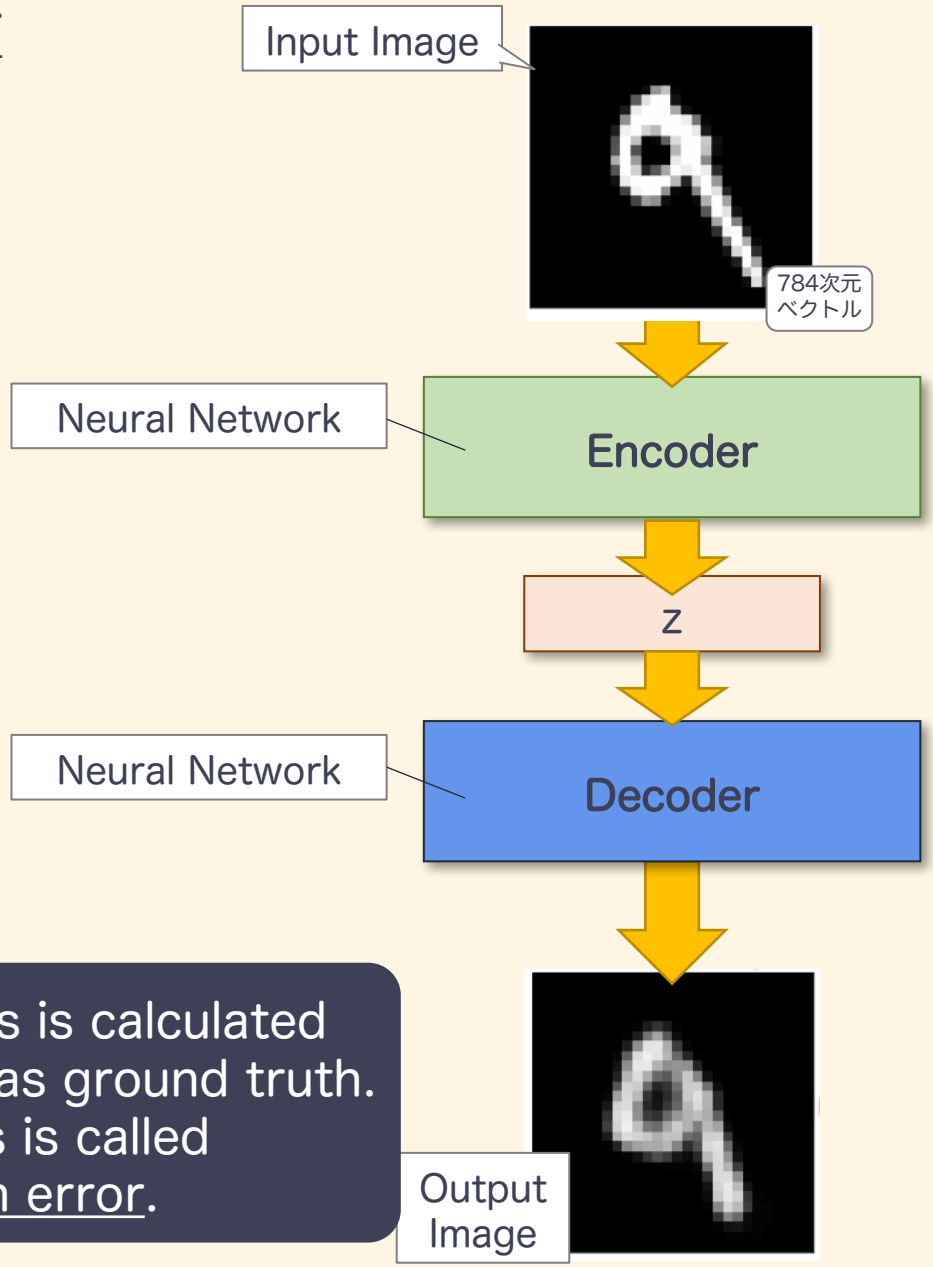


Ref： http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

# Let’s deep dive into Variational Autoencoder !

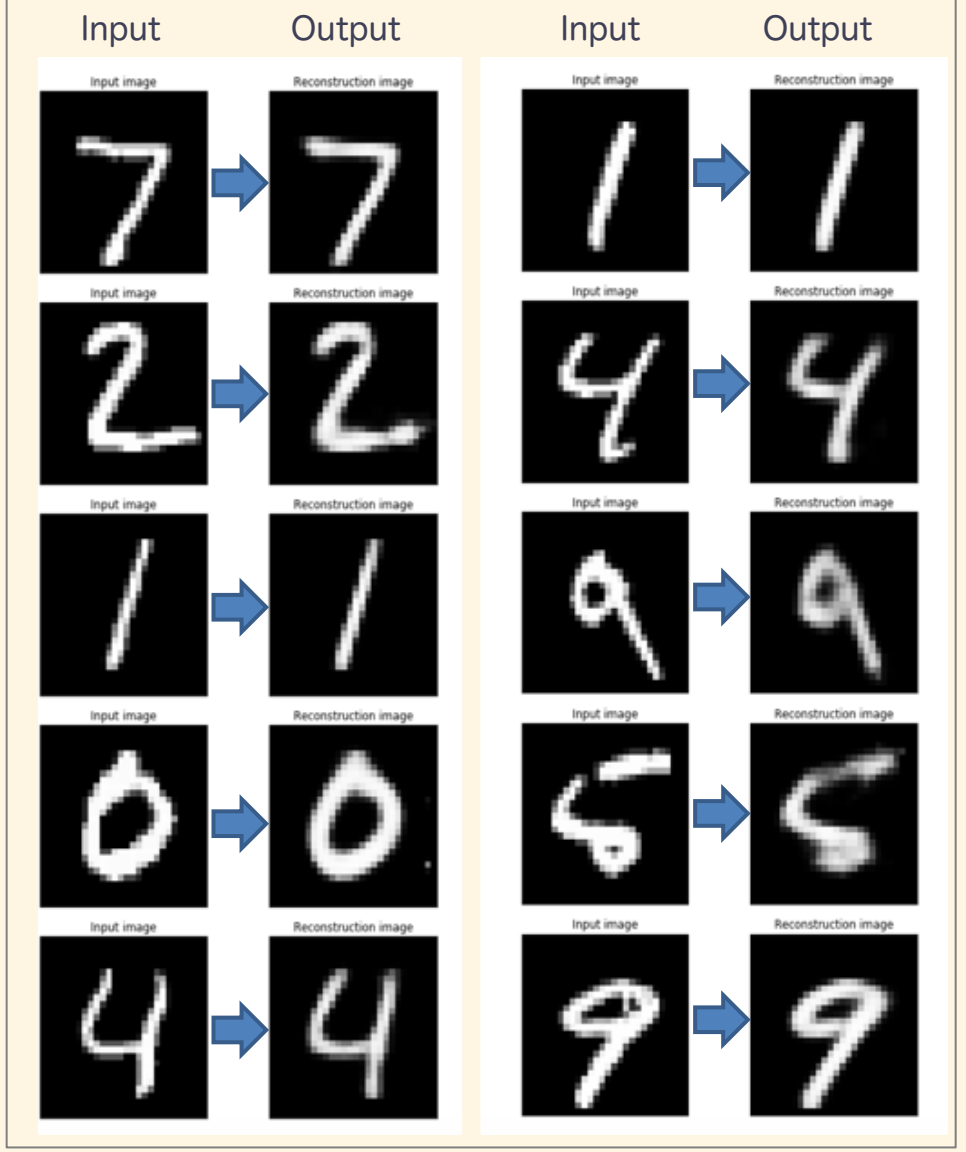
# Normal Autoencoder

Example of MNIST



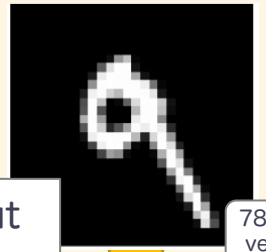

In learning phase, loss is calculated with input image itself as ground truth. This type of loss is called reconstruction error.

## Example of Output



# Variational Autoencoder

## Example of MNIST



Input Image

784dim vector

Neural Network

**Encoder**

$q_\phi(z|X)$

Parameter of Gaussian Distribution

$\mu(X)$ $\sigma(X)$

$z \sim N(\mu(X), \sigma(X))$

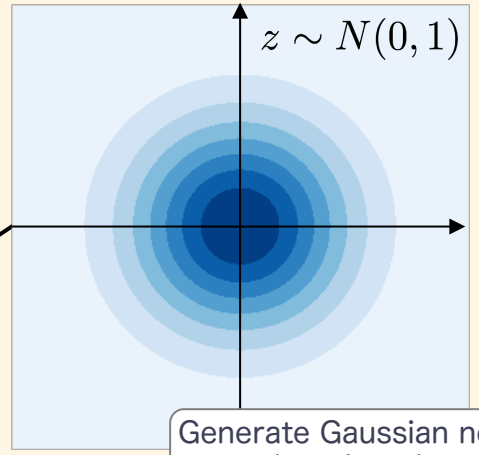
Probability distribution is embedded as latent variable z.

Neural Network

**Decoder**

$p_\theta(X|z)$

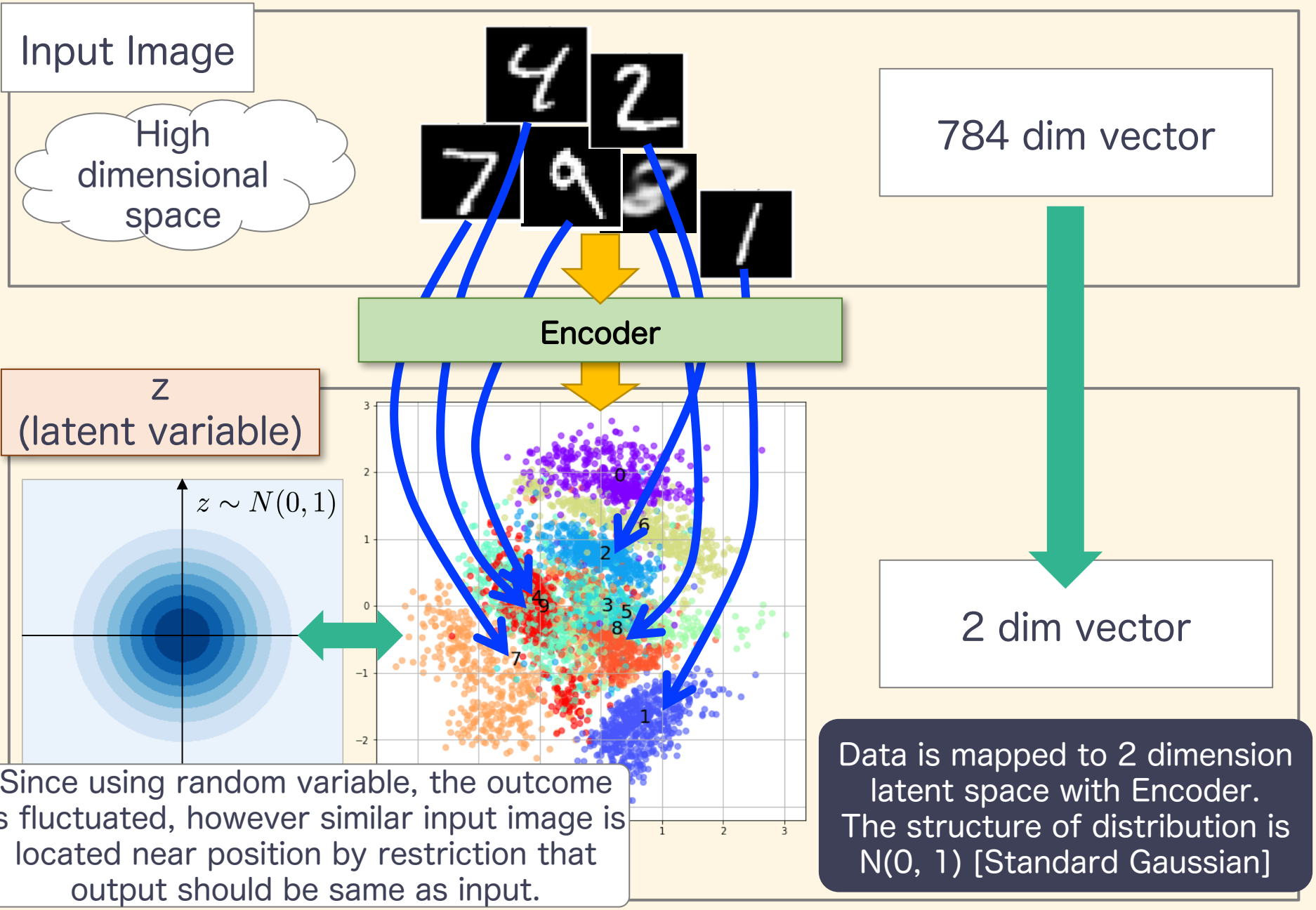
Output Image



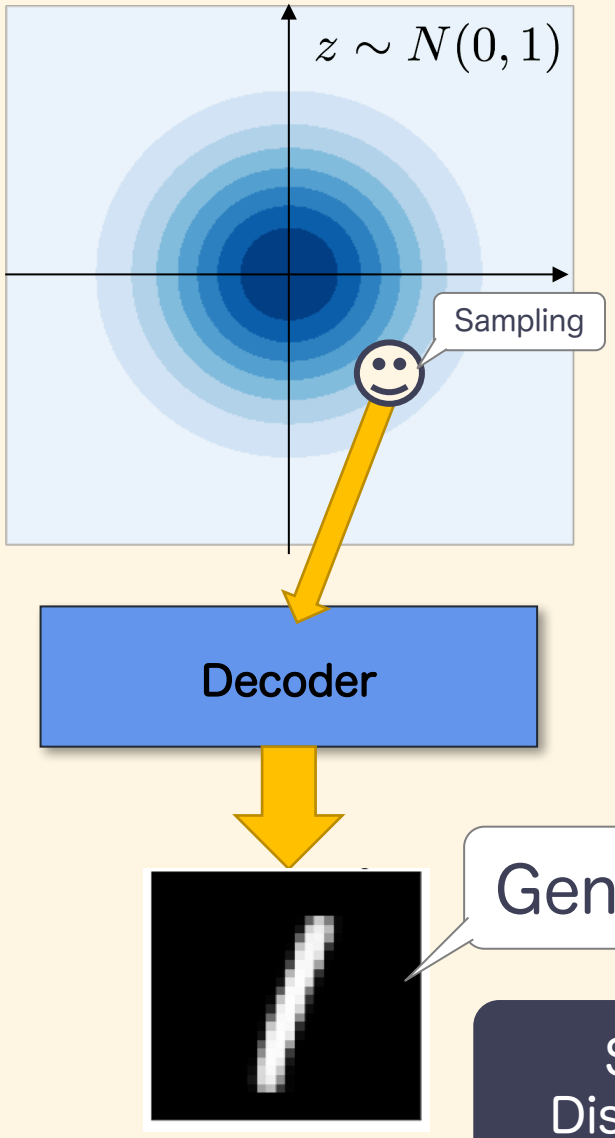

Generate Gaussian noise on learning phase

Neural Network's parameter $\phi, \theta$ is optimized with Back Propagation.

# Encoder


Input Image
High dimensional space
784 dim vector
Encoder
z
(latent variable)
$z \sim N(0, 1)$
2 dim vector
Since using random variable, the outcome is fluctuated, however similar input image is located near position by restriction that output should be same as input.
Data is mapped to 2 dimension latent space with Encoder. The structure of distribution is N(0, 1) [Standard Gaussian]

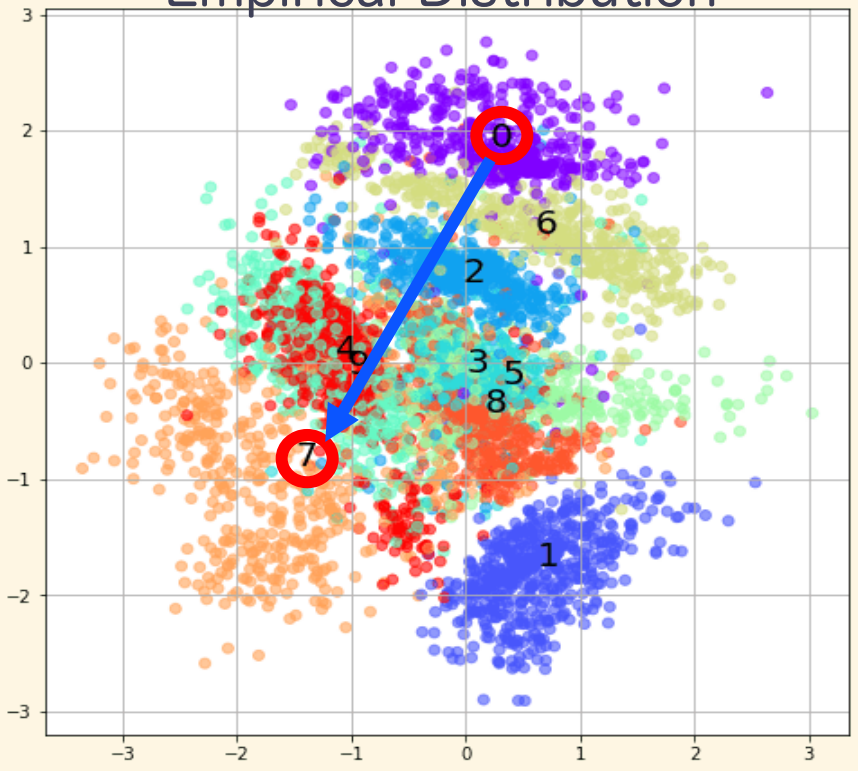
# Decoder



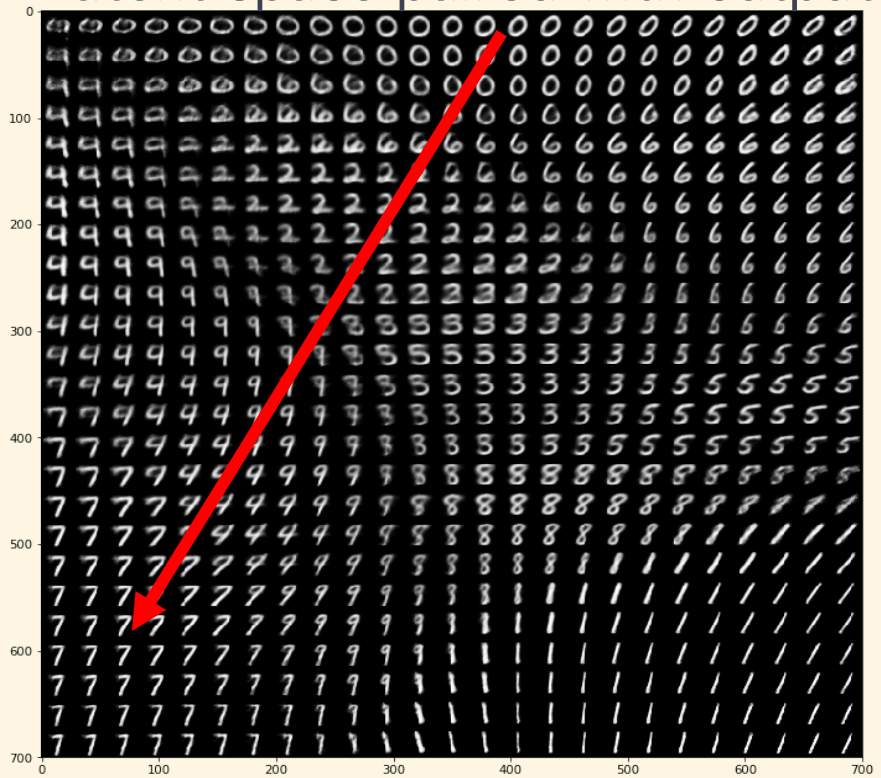

Sampling from Gaussian Distribution N(0, 1), 784 dim vector data can be generated.

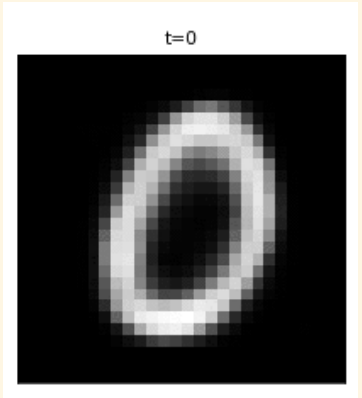
# Image Generation with Decoder

Empirical Distribution




Latent space paired with output



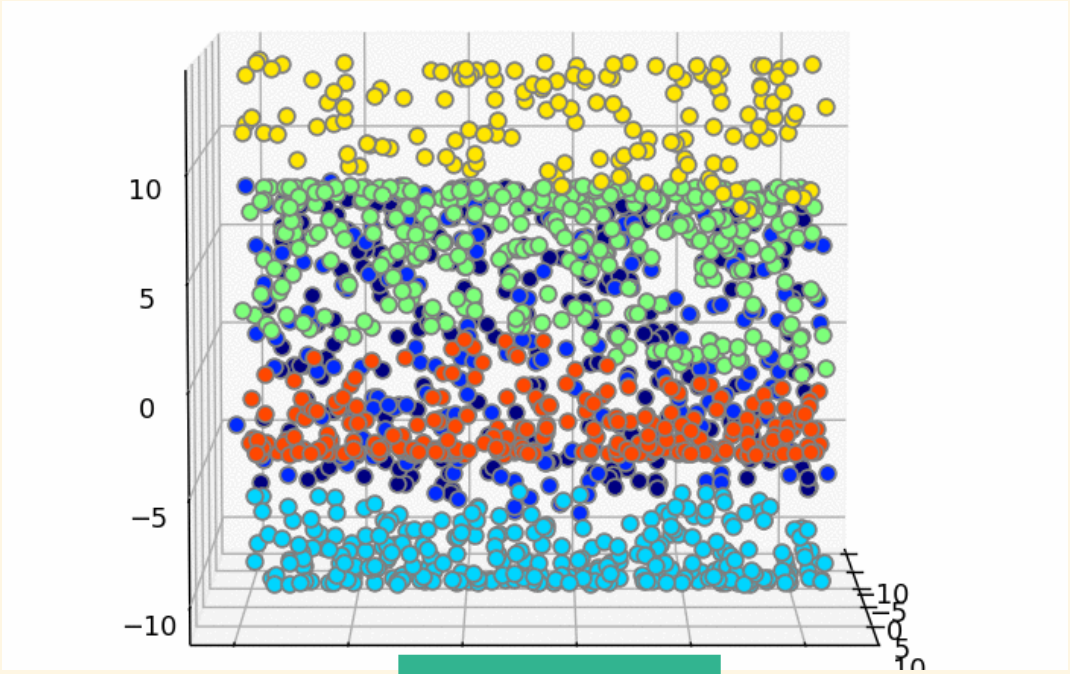



Animation:
https://github.com/matsuken92/Qiita_Contents/blob/master/chainer-vae/vae_0to9.gif

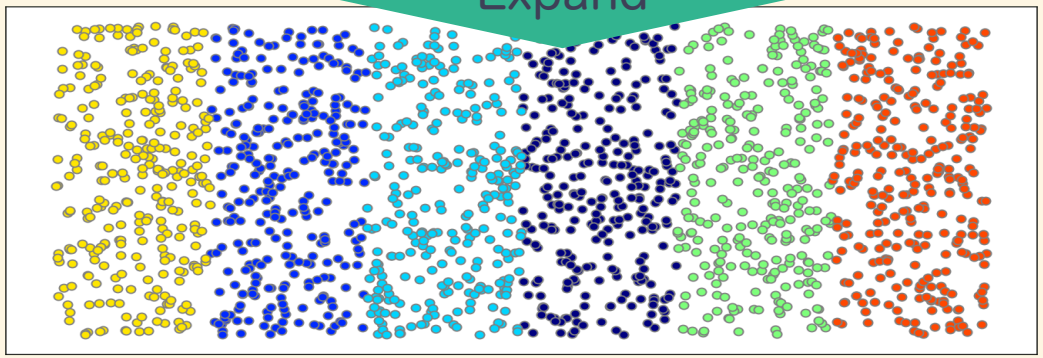
# Demonstration


Animation:
https://github.com/matsuken92/Qiita_Contents/blob/master/chainer-vae/vae_all_9to9.gif

# Manifold Hypothesis （ex: Swiss Roll)




Expand



Data exist high dimensional space, however it is exist only limited space.

This “Swiss Roll” exists 3 dim space, but almost data is represented in 2dim space

The Hypothesis that neural network is extract efficient, well condition manifold from given data.

Animation: https://github.com/matsuken92/Qiita_Contents/blob/master/chainer-vae/swiss_roll2.gif

# Theoretical perspective of Variational Autoencoder

# What is Generative Model (Rethink)

## Objective

Estimating distribution $p(X)$ which generates training data. (existing data at hand is observed data which is generated from this distribution.)

## Example of application

- ✓ Image generation as an art
- ✓ Sentence generation of natural language
- ✓ Semi-supervised learning (Generate labeled data from small amount of labeled supervised data and large amount of un-labeled data)

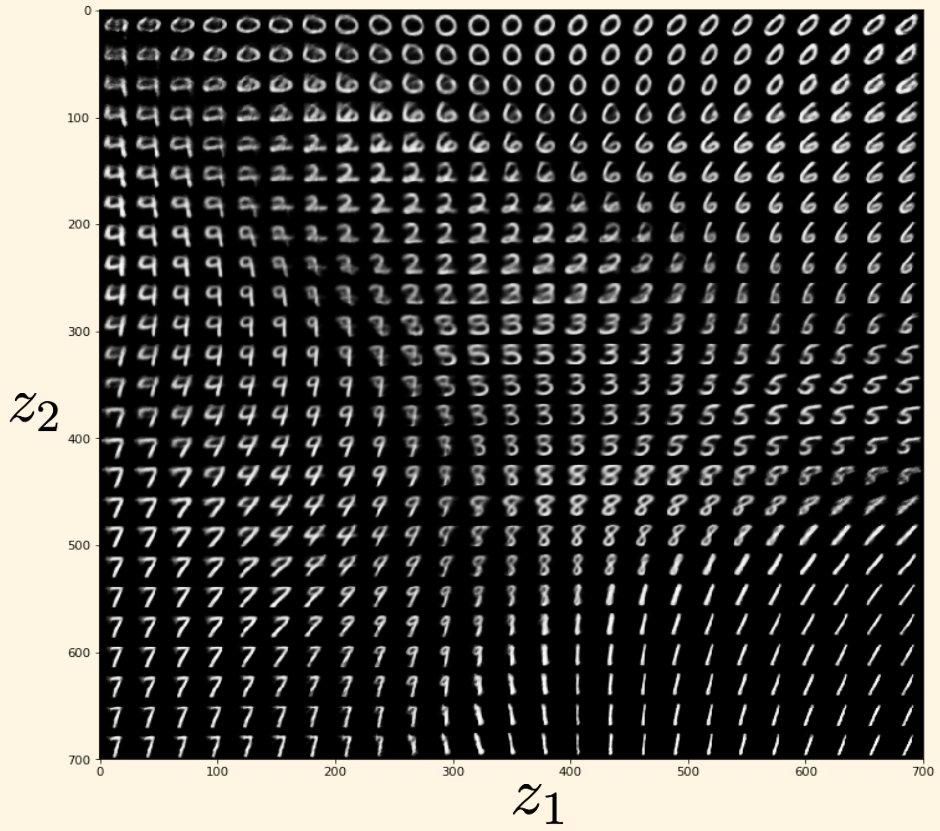
# What is Generative Model (Rethink)

Once $p(X)$ is identified, data can be generated.

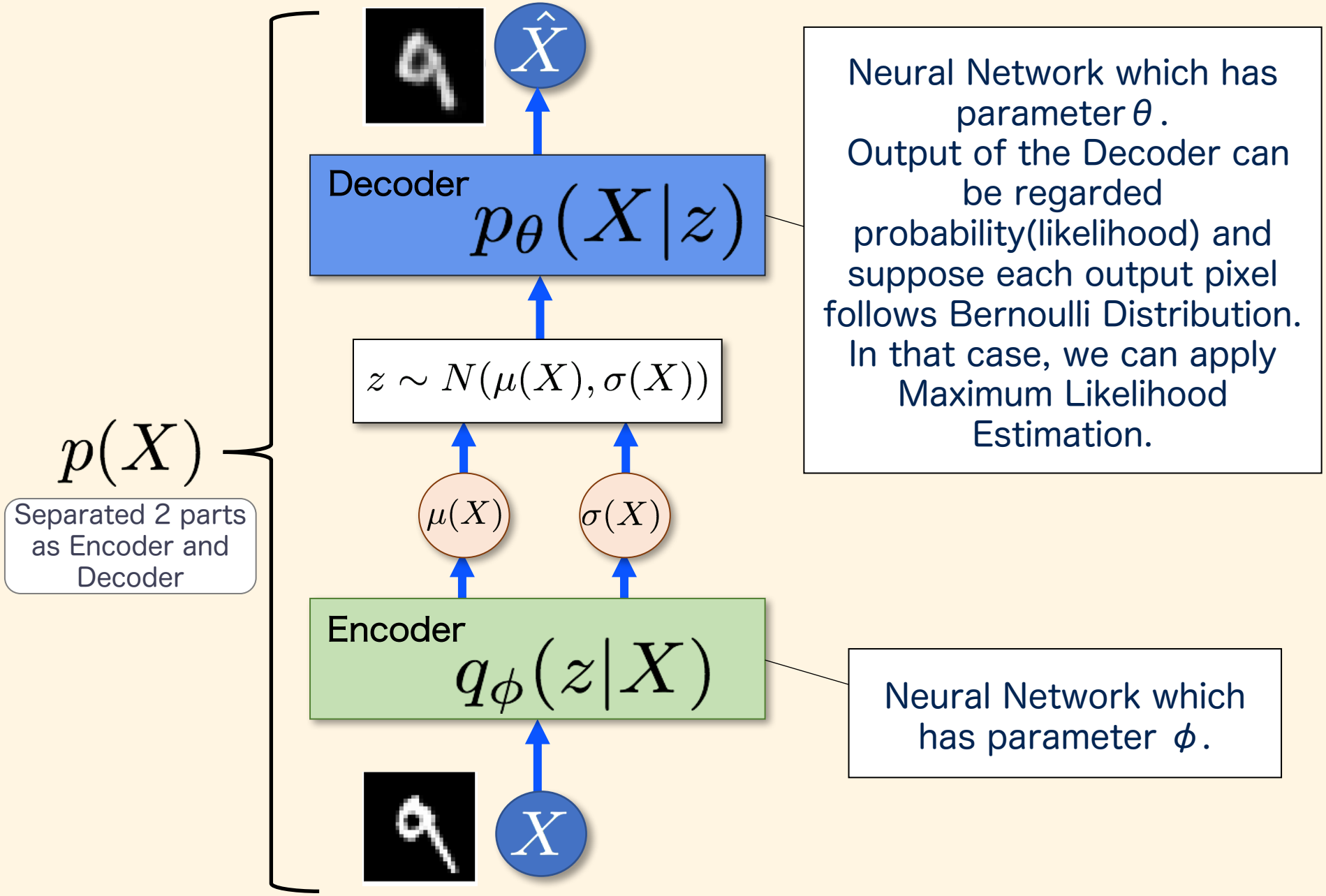
In example of MNIST, a image which is recognized as hand written digits is assigned high probability.

On the other hand, a image which is not recognized as handwritten digits, its probability is low.

Latent space paired with output.

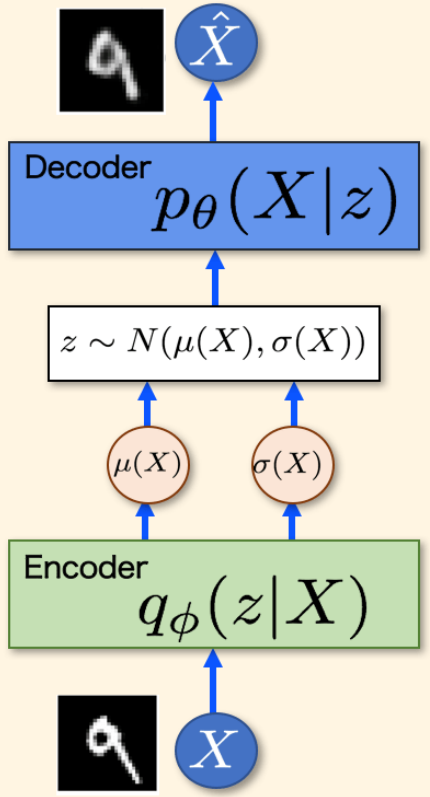


# Structure of VAE



# Step for determining optimal parameters

1. Find the parameter of neural network $\theta, \phi$ maximizing likelihood p(X) with Maximum Likelihood method.
2. For easy to handle, set the target to log likelihood log p(X)
3. log p(X) is intractable, so we introduce **Variational Lower Bound** $\mathcal{L}(X, z)$ , and since p(X) ≧ $\mathcal{L}(X, z)$ , if Variational Lower Bound is maximized, then p(X) is also maximized approximately.




* Detail of Variational

# Variational lower-bound

$p(X)$is intractable, so we need some technique for calculation. and easy to calculate, taking log of p(X).

Marginalization

$$\log p(X) = \log \int p(X, z) dz$$

Multiplying $\frac{q(z|X)}{q(z|X)} = 1$

$$= \log \int q(z|X) \frac{p(X, z)}{q(z|X)} dz$$

Jensen inequation

$$\log E[X] \geq E[\log X]$$

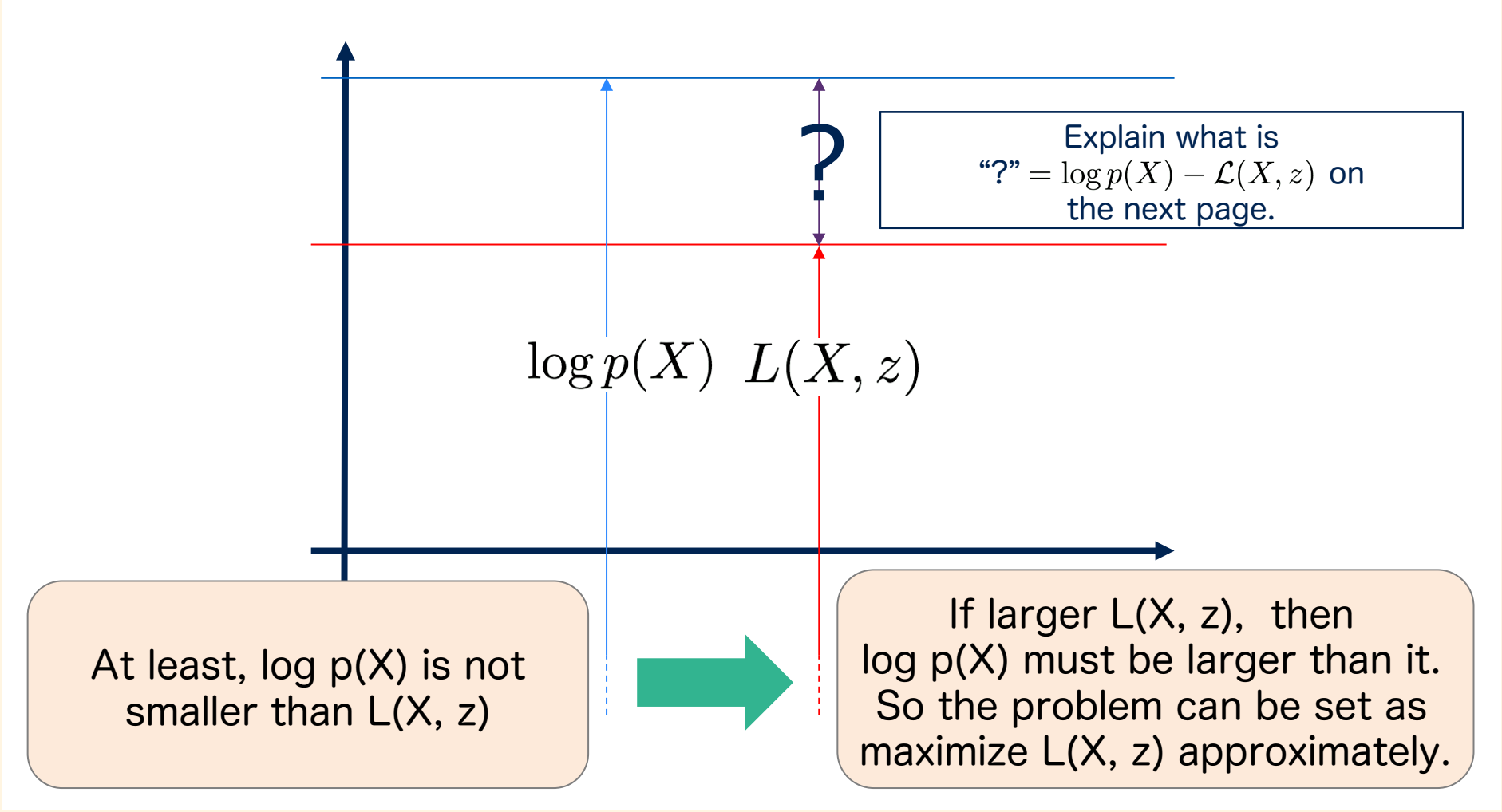
$$\geq \int q(z|X) \log \frac{p(X, z)}{q(z|X)} dz$$

$$= \underline{L(X, z)}$$

→ Variational Lower Bound

# Variational lower-bound

$\log p(X) \geq \mathcal{L}(X, z)$ means …



# Meaning of $\log p(X) - \mathcal{L}(X, z)$

$$\log p(X) - L(X, z)$$

$$= \log p(X) - \int q(z|X) \log \frac{p(X, z)}{q(z|X)} dz$$

$$= \log p(X) \underbrace{\int q(z|X) dz}_{=1} - \int q(z|X) \log \frac{p(X, z)}{q(z|X)} dz$$

multiplicative theorem

$$p(X, z) = p(z|X)p(X)$$

$$= \int q(z|X) \log p(X) dz - \int q(z|X) \log \frac{p(X, z)}{q(z|X)} dz$$

$$= \int q(z|X) \log p(X) dz - \int q(z|X) \log \frac{p(z|X)p(X)}{q(z|X)} dz$$

Decompose

combine

$$= \int q(z|X) \log p(X) dz - \int q(z|X) [\log p(z|X) + \log p(X) - \log q(z|X)] dz$$

$$= \int q(z|X) [-\log p(z|X) + \log q(z|X)] dz$$

Combine

$$= \int q(z|X) \log \frac{q(z|X)}{p(z|X)} dz$$

$$= D_{KL}[q(z|X) \| p(z|X)] \geq 0$$

The value of KL is more than or equal 0

# Kullback–Leibler divergence

Kullback-Leibler divergence is metrics for measuring between 2 probability distribution, represented as the following equation

$$
\begin{aligned}
& D_{KL}[q(z)||p(z)] \\
&= \mathbf{E}_{q(z)}[\log q(z) - \log p(z)] \\
&= \int q(z) \log \frac{q(z)}{p(z)} dz
\end{aligned}
$$

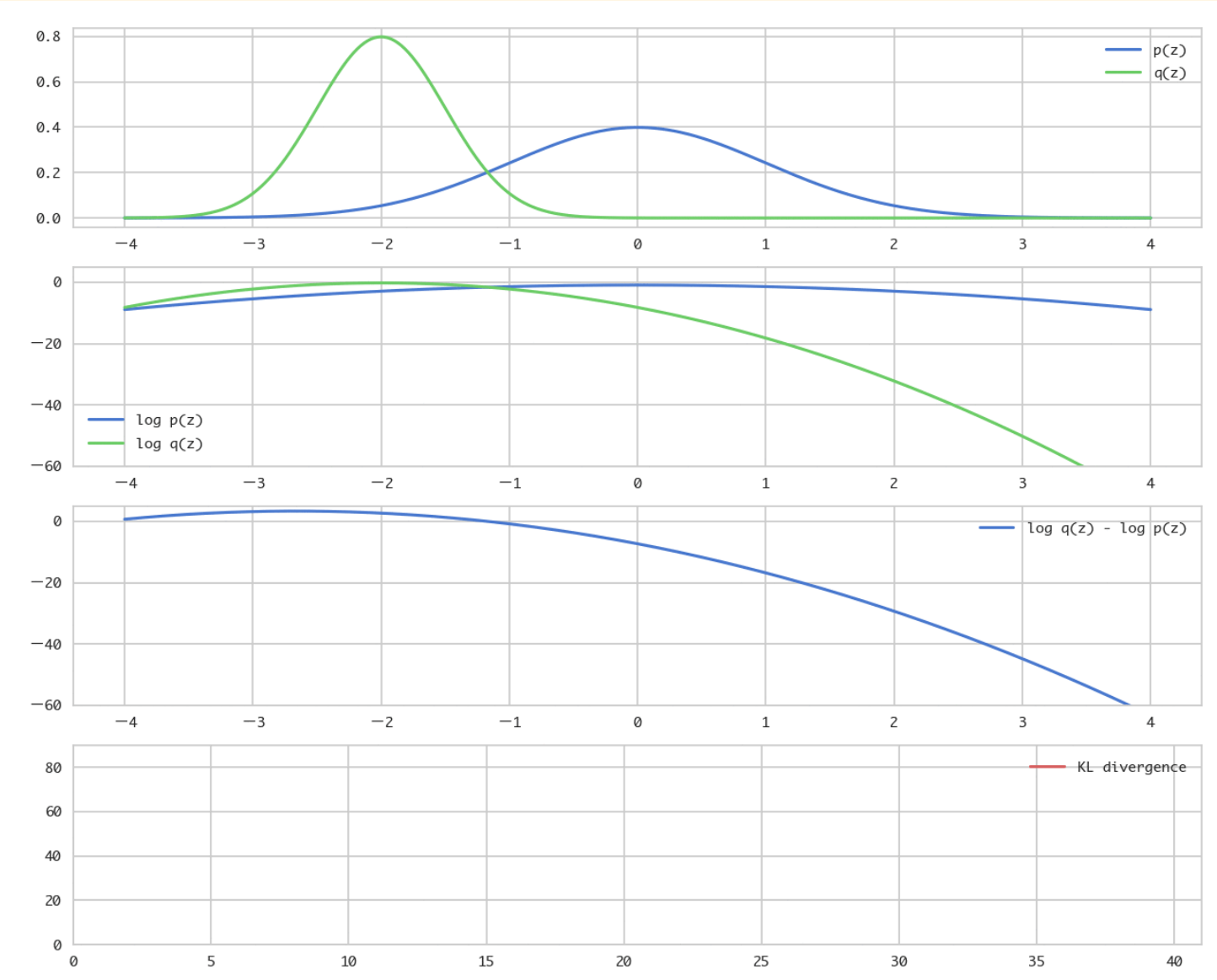
Suppose p(z) is true distribution, and q(z) is a approximate model of the true distribution. KL can be used for indicator of goodness of approximation, and it is better the value is closer to 0 .

KL is similar to “Distance” since it is indicate closeness between 2 distribution, however KL has property as the following equation

$$D_{KL}[q(z)||p(z)] \neq D_{KL}[p(z)||q(z)]$$

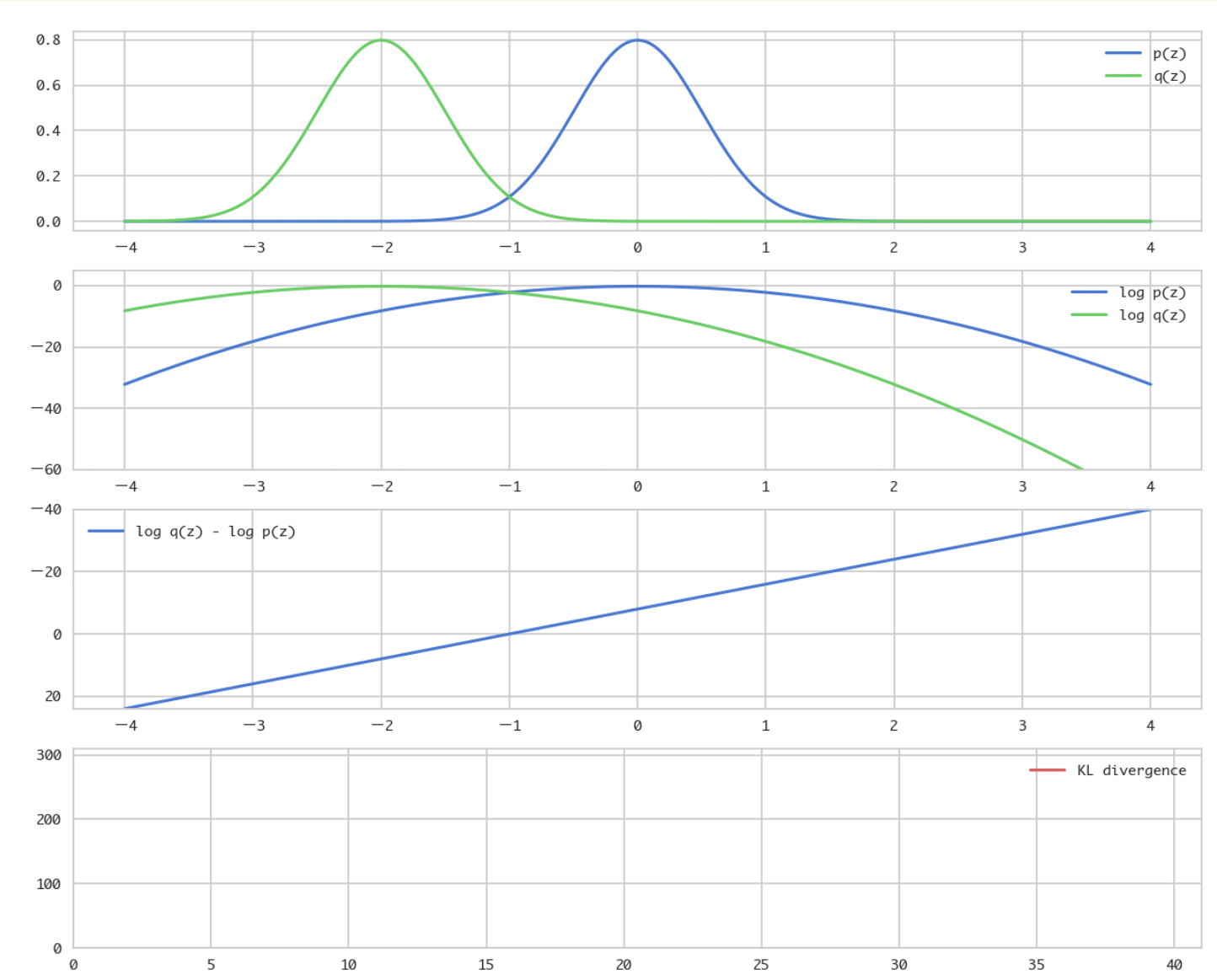
so, it does not meet axiom of distance, so KL is not “Distance”.

# Visualizing Kullback–Leibler divergence 1




アニメーションGIF：
https://github.com/matsuken92/Qiita_Contents/blob/master/chainer-vae/kl_visualize.gif

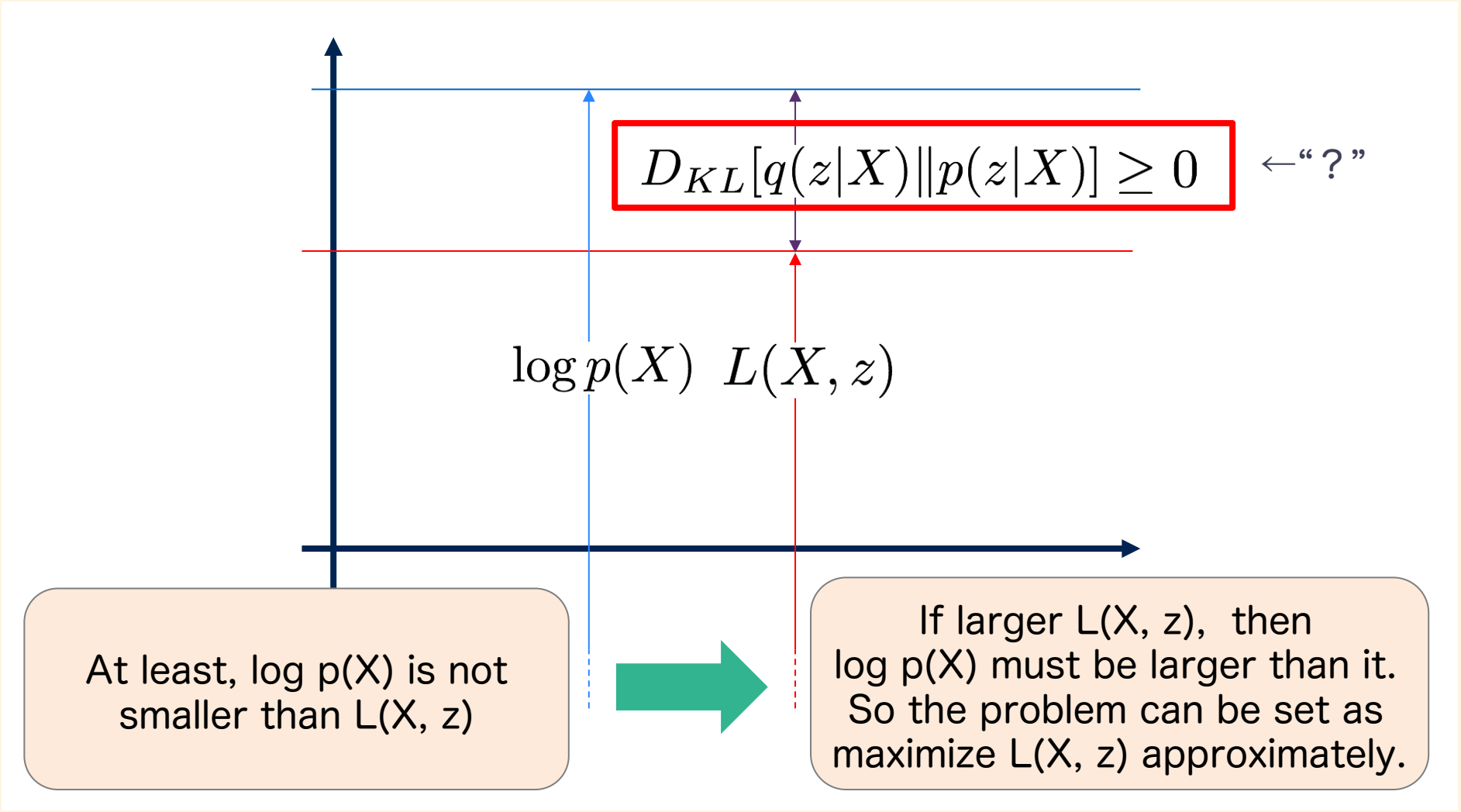
# Visualizing Kullback–Leibler divergence 2




アニメーションGIF：
https://github.com/matsuken92/Qiita_Contents/blob/master/chainer-vae/kl_visualize2.gif

# Variational lower-bound

Now “？” is clear.



# Variational lower-bound

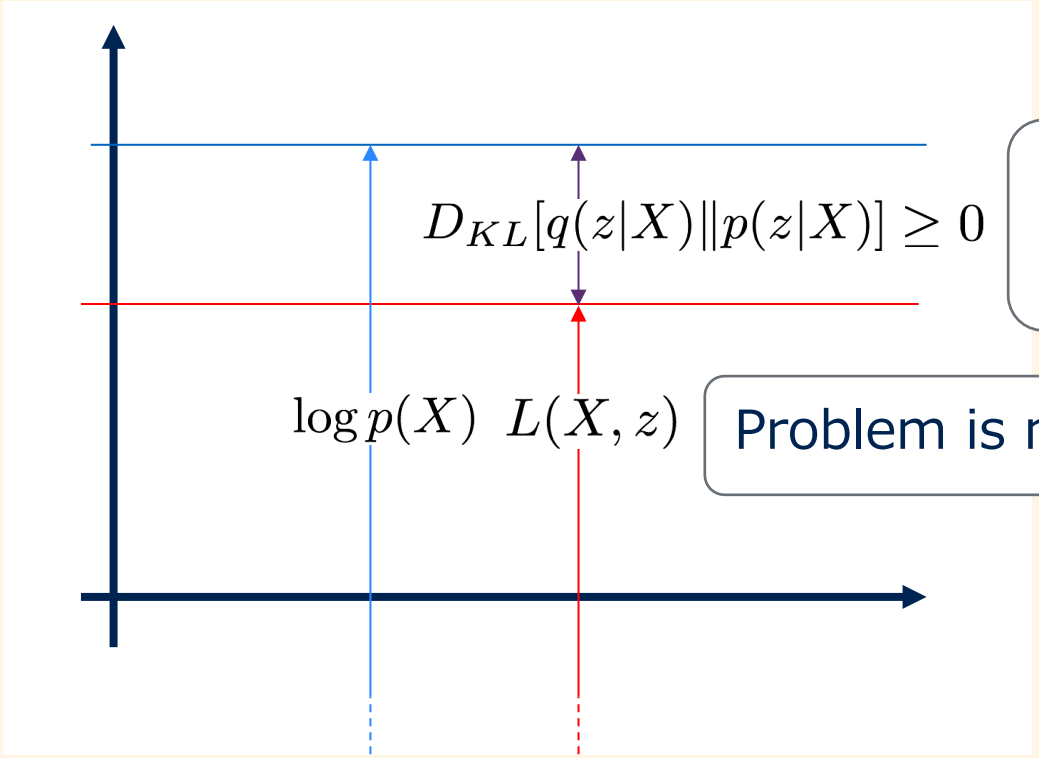
$$L(X,z) = \log p(X) - D_{KL}[q(z|X)\|p(z|X)]$$

$$= \log p(X) - D_{KL}[q_\phi(z|X)\|p_\theta(z|X)]$$

clearly indicate parameters

Constant

Varying depend on $\phi, \theta$ .




changing to minimizing a term of Kullback-Leibler Divergence

Problem is not maximizing $L(X,z)$ ,

# Further decomposition of term of KL

$$
\begin{aligned}
& D_{KL}[q_\phi(z|X)\|p_\theta(z|X)] \\
&= \mathbf{E}_{q_\phi(z|X)}[\log q_\phi(z|X) - \log p_\theta(z|X)] \\
&= \mathbf{E}_{q_\phi(z|X)}[\log q_\phi(z|X) - \log p_\theta(X|z) - \log p(z) + \log p(X)] \\
&= \mathbf{E}_{q_\phi(z|X)}[\log q_\phi(z|X) - \log p_\theta(X|z) - \log p(z)] + \log p(X) \\
&= \mathbf{E}_{q_\phi(z|X)}[\log q_\phi(z|X) - \log p(z)] - \mathbf{E}_{q_\phi(z|X)}[\log p_\theta(X|z)] + \log p(X) \\
&= D_{KL}[q_\phi(z|X)\|p(z)] - \mathbf{E}_{q_\phi(z|X)}[\log p_\theta(X|z)] + \log p(X)
\end{aligned}
$$

**Bayes's Theorem**

$$p(z|X) = \frac{p(X|z)p(z)}{p(X)}$$

putting out it, because it does not depend on $q_\phi(z|X)$

# Rethink Variational Lower Bound

$$L(X,z) = \log p(X) - D_{KL}[q(z|X)\|p(z|X)]$$

Refer prev. page

$$= \log p(X) - D_{KL}[q(z|X)\|p(z)] + \mathbf{E}_{q(z|X)}[\log p(X|z)] - \log p(X)$$

$$= -D_{KL}[q(z|X)\|p(z)] + \mathbf{E}_{q(z|X)}[\log p(X|z)]$$

→ ① Smaller → ② Larger

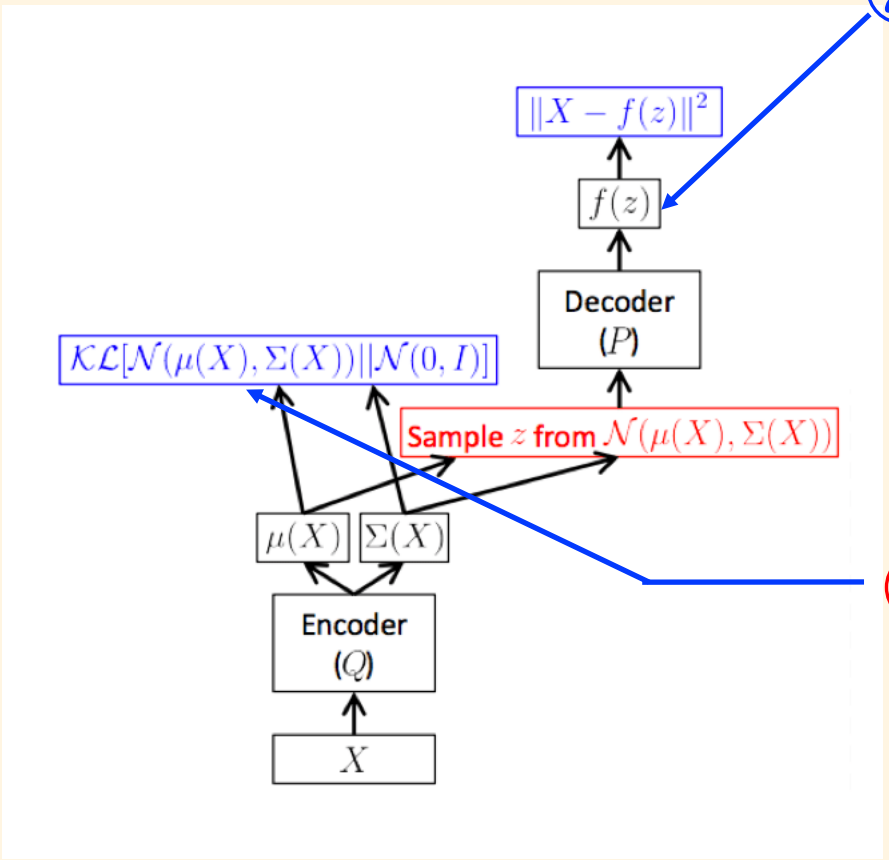
① $D_{KL}[q(z|X)||p(z)]$

⇒ The closeness between true distribution of z and approximate model(Encoder, constructed with NN) is represented with KL divergence.
The smaller is the better.

② $\mathbf{E}_{q(z|X)}[\log p(X|z)]$

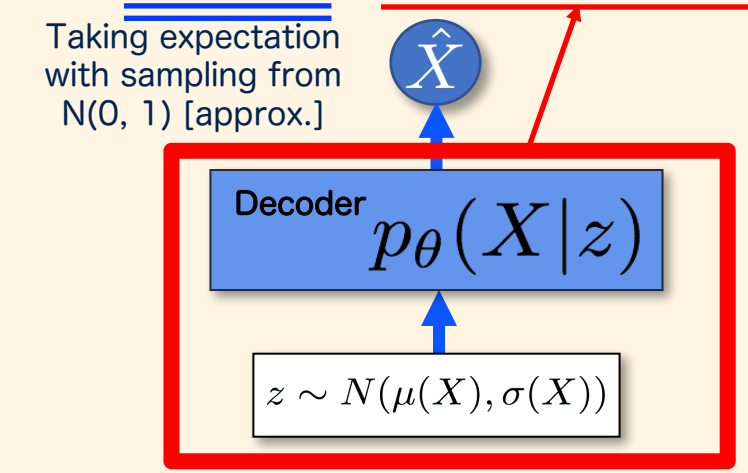
⇒ Taking expectation to Log likelihood with respect to q(z|X).
This is a objective of log likelihood maximization.

# Rethink Variational Lower Bound

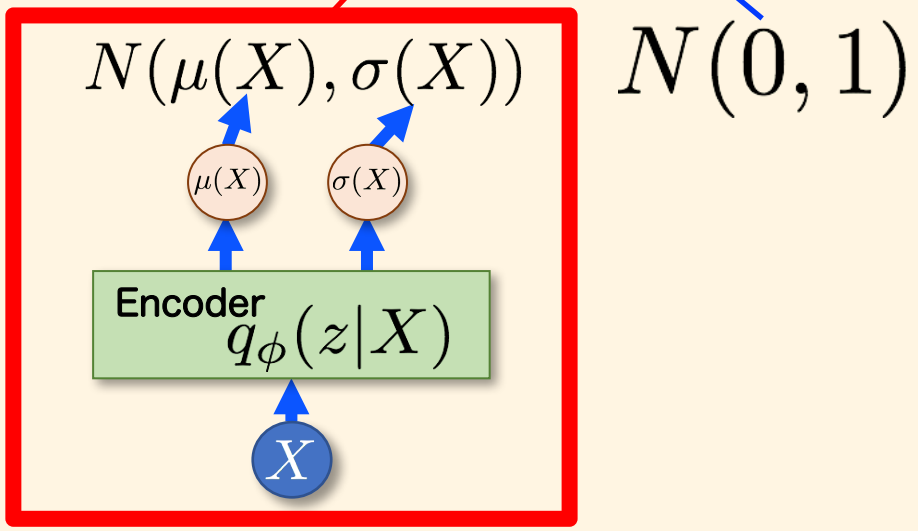



② $\mathbf{E}_{q(z|X)}[\log p(X|z)]$

Taking expectation with sampling from N(0, 1) [approx.]




① $D_{KL}[q(z|X)||p(z)]$




Ref： https://arxiv.org/abs/1606.05908

# Consideration of loss

① $D_{KL}[q(z|X)||p(z)]$

$q(x) \sim N(\boldsymbol{\mu}_q, \Sigma_q),$

$p(x) \sim N(\boldsymbol{\mu}_p, \Sigma_p)$

$$D_{KL}(q(x)\|p(x)) =$$

$$\frac{1}{2}\left[\log \frac{|\Sigma_p|}{|\Sigma_q|} - d + \mathrm{Tr}(\Sigma_p^{-1}\Sigma_q) + (\boldsymbol{\mu}_p - \boldsymbol{\mu}_q)^T\Sigma_p^{-1}(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q)\right]$$

$|\Sigma_p| = 1$, $\mathrm{Tr}(\Sigma_p^{-1}\Sigma_q) = 1$, $(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q) = 0$, $\Sigma_p^{-1} = 1$, $(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q) = 0$

KL divergence of Gaussian

$$\frac{1}{2}\sum_{j=1}^{J}(-\log((\sigma_j)^2) - 1 + (\mu_j)^2 + (\sigma_j)^2)$$

$p(x) \sim N(0,1)$
i.e. $\boldsymbol{\mu}_p = 0, \Sigma_p = 1$ and $d = 1$

J is dim of z

## Chaner code

```
var = exponential.exp(ln_var)
mean_square = mean * mean
loss = (mean_square + var - ln_var - 1) * 0.5
```

Ref: http://qiita.com/kenmatsu4/items/c107bd51503462fb677f

# Consideration of loss

$$②\mathbf{E}_{q(z|X)}[\log p(X|z)]$$

Approximate empirical distribution with sampling

$$\simeq \frac{1}{L}\sum_{\ell=1}^{L}\log p(X|z^{(\ell)}) \qquad z^{(\ell)} \sim N(\mu(X), \Sigma(X)) \quad (\ell = 1, 2, \cdots, L)$$

Suppose X follows Bernoulli distribution, then log likelihood is the following.

$$\log p(X|z) = \sum_{i=1}^{D} x_i \log y_i + (1 - x_i)\log(1 - y_i)$$

D is dim of X

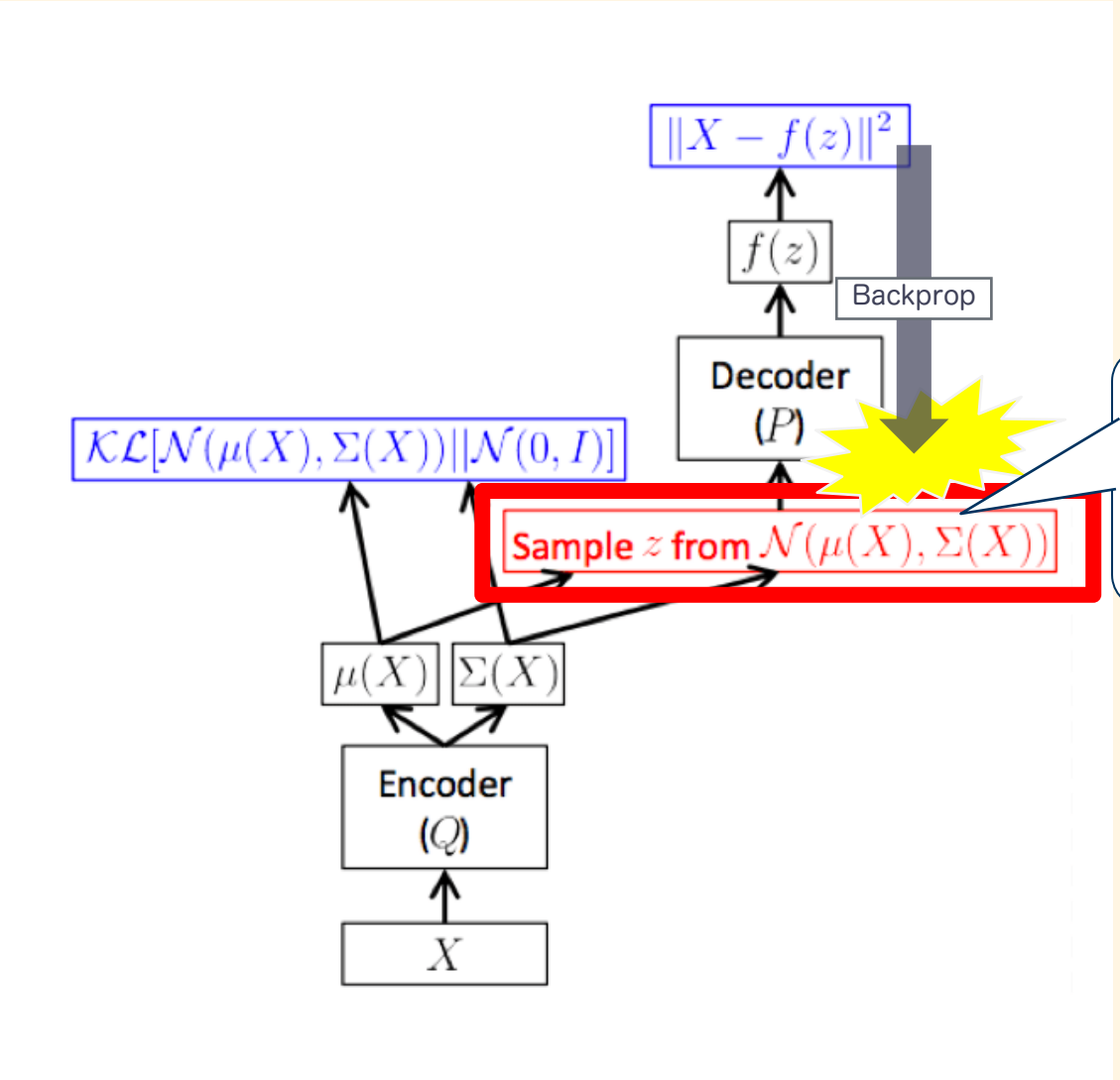
$y_i$ is output of neural network.

## Chanerコード

```
mu, ln_var = self.encode(x)
batchsize = len(mu.data)
rec_loss = 0   # reconstruction error
for l in six.moves.range(k):
    z = F.gaussian(mu, ln_var)
    z.name = "z”
    rec_loss += F.bernoulli_nll(x, self.decode(z, sigmoid=False)) / (k * batchsize)
```

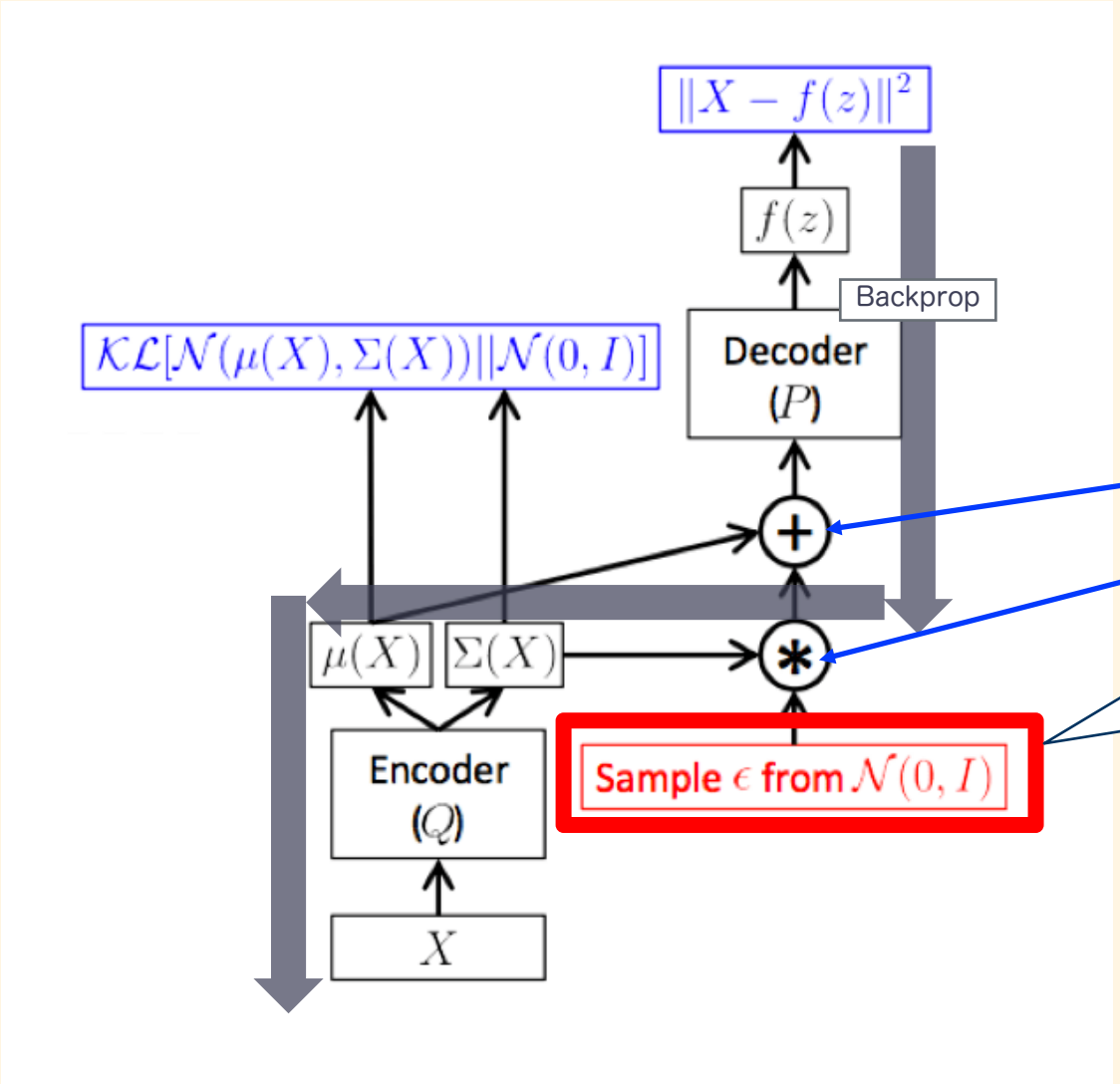
Ref: bernoulli_nill of chainer：
https://docs.chainer.org/en/stable/reference/generated/chainer.functions.bernoulli_nll.html#chainer.functions.bernoulli_nll

# Reparametrization Trick




Variable z here is random variable, so can not conduct back prop.
(Can not calculate gradient)

# Reparametrization Trick




$$z \sim \mu(X) + \Sigma^{1/2} * \varepsilon$$

Random variable generation can be put out of this flow. So enable back prop.

# Reparametrization Trick

② $\mathbf{E}_{q(z|X)}[\log p(X|z)]$

Rewrite equation in order to follow Reparametrization Trick.

$$\mathbf{E}_{\varepsilon \sim N(0,I)}[\log p(X|z = \mu(X) + \Sigma^{1/2} * \varepsilon)]$$

# Chainer code (excerpt)

```
class VAE(chainer.Chain):
    """Variational AutoEncoder"""
    def __init__(self, n_in, n_latent, n_h, act_func=F.tanh):
        super(VAE, self).__init__()
        self.act_func = act_func
        <略>
    def __call__(self, x, sigmoid=True):
        """ AutoEncoder """
        return self.decode(self.encode(x)[0], sigmoid)

    def encode(self, x):
        if type(x) != chainer.variable.Variable:
            x = chainer.Variable(x)
        x.name = "x"
        h1 = self.act_func(self.le1(x))
        h1.name = "enc_h1"
        h2 = self.act_func(self.le2(h1))
        h2.name = "enc_h2"
        mu = self.le3_mu(h2)
        mu.name = "z_mu"
        ln_var = self.le3_ln_var(h2)  # ln_var = log(sigma**2)
        ln_var.name = "z_ln_var"
        return mu, ln_var

    def decode(self, z, sigmoid=True):
        h1 = self.act_func(self.ld1(z))
        h1.name = "dec_h1"
        h2 = self.act_func(self.ld2(h1))
        h2.name = "dec_h2"
        h3 = self.ld3(h2)
        h3.name = "dec_h3"
        if sigmoid:
            return F.sigmoid(h3)
        else:
            return h3

    def get_loss_func(self, C=1.0, k=1):
        def lf(x):
            mu, ln_var = self.encode(x)
            batchsize = len(mu.data)
            # reconstruction loss
            rec_loss = 0
            for l in six.moves.range(k):
                z = F.gaussian(mu, ln_var)
                z.name = "z"
                rec_loss += F.bernoulli_nll(x, self.decode(z, sigmoid=False)) / (k * batchsize)
            self.rec_loss = rec_loss
            self.rec_loss.name = "reconstruction error"
            self.latent_loss = C * gaussian_kl_divergence(mu, ln_var) / batchsize
            self.latent_loss.name = "latent loss"
            self.loss = self.rec_loss + self.latent_loss
            self.loss.name = "loss"
            return self.loss
        return lf
```

https://github.com/matsuken92/Qiita_Contents/blob/master/chainer-vae/chainer-vae.ipynb

# Chainer code (excerpt)

```
# Learning loop
for epoch in six.moves.range(1, n_epoch + 1):
    print('epoch', epoch)

    # training
    perm = np.random.permutation(N)
    sum_loss = 0       # total loss
    sum_rec_loss = 0   # reconstruction loss
    for i in six.moves.range(0, N, batchsize):
        x = chainer.Variable(xp.asarray(x_train[perm[i:i + batchsize]]))
        optimizer.update(model.get_loss_func(), x)
        if epoch == 1 and i == 0:
            with open('graph.dot', 'w') as o:
                g = computational_graph.build_computational_graph(
                    (model.loss, ))
                o.write(g.dump())
            print('graph generated')

        sum_loss += float(model.loss.data) * len(x.data)
        sum_rec_loss += float(model.rec_loss.data) * len(x.data)

    print('train mean loss={}, mean reconstruction loss={}'
          .format(sum_loss / N, sum_rec_loss / N))

    # evaluation
    sum_loss = 0
    sum_rec_loss = 0
    with chainer.no_backprop_mode():
        for i in six.moves.range(0, N_test, batchsize):
            x = chainer.Variable(xp.asarray(x_test[i:i + batchsize]))
            loss_func = model.get_loss_func(k=10)
            loss_func(x)
            sum_loss += float(model.loss.data) * len(x.data)
            sum_rec_loss += float(model.rec_loss.data) * len(x.data)
            del model.loss
    print('test  mean loss={}, mean reconstruction loss={}'
          .format(sum_loss / N_test, sum_rec_loss / N_test))
```

https://github.com/matsuken92/Qiita_Contents/blob/master/chainer-vae/chainer-vae.ipynb

# Application of VAE for Semi-supervised Learning

[Brief introduction]

# Semi-supervised Learning

Comparing classification accuracy with only 100～300 pics of labeled MNIST image.
M1+M2 (explained the following page) can reduce error rate to 3.33% !

Table 1: Benchmark results of semi-supervised classification on MNIST with few labels.

| *N* | NN | CNN | TSVM | CAE | MTC | AtlasRBF | M1+TSVM | M2 | M1+M2 |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 25.81 | 22.98 | 16.81 | 13.47 | 12.03 | 8.10 (± 0.95) | 11.82 (± 0.25) | 11.97 (± 1.71) | **3.33** (± 0.14) |
| 600 | 11.44 | 7.68 | 6.16 | 6.3 | 5.13 | – | 5.72 (± 0.049) | 4.94 (± 0.13) | **2.59** (± 0.05) |
| 1000 | 10.7 | 6.45 | 5.38 | 4.77 | 3.64 | 3.68 (± 0.12) | 4.24 (± 0.07) | 3.60 (± 0.56) | **2.40** (± 0.02) |
| 3000 | 6.04 | 3.35 | 3.45 | 3.22 | 2.57 | – | 3.49 (± 0.04) | 3.92 (± 0.63) | **2.18** (± 0.04) |

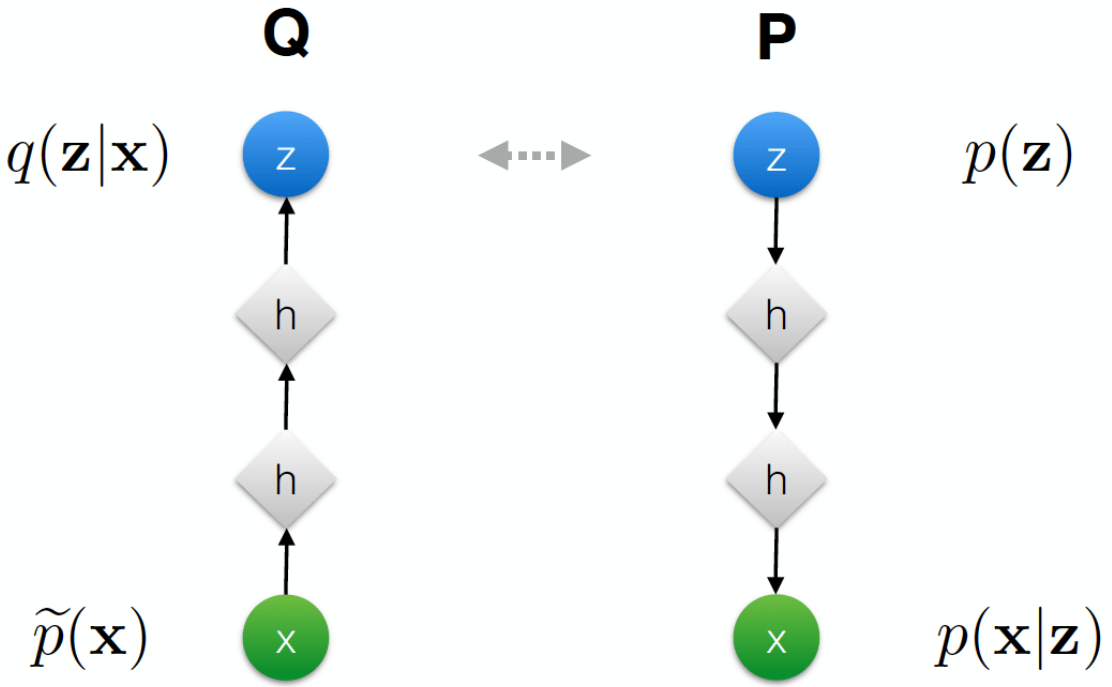
Ref : Semi-supervised Learning with Deep Generative Models, D.P.Kingma et.al
https://arxiv.org/pdf/1406.5298.pdf

# Semi-supervised Learning

**M1 model**

The model explained on first part of this slide.

DLGM / VAE as feature extractor for semi-supervised classifier



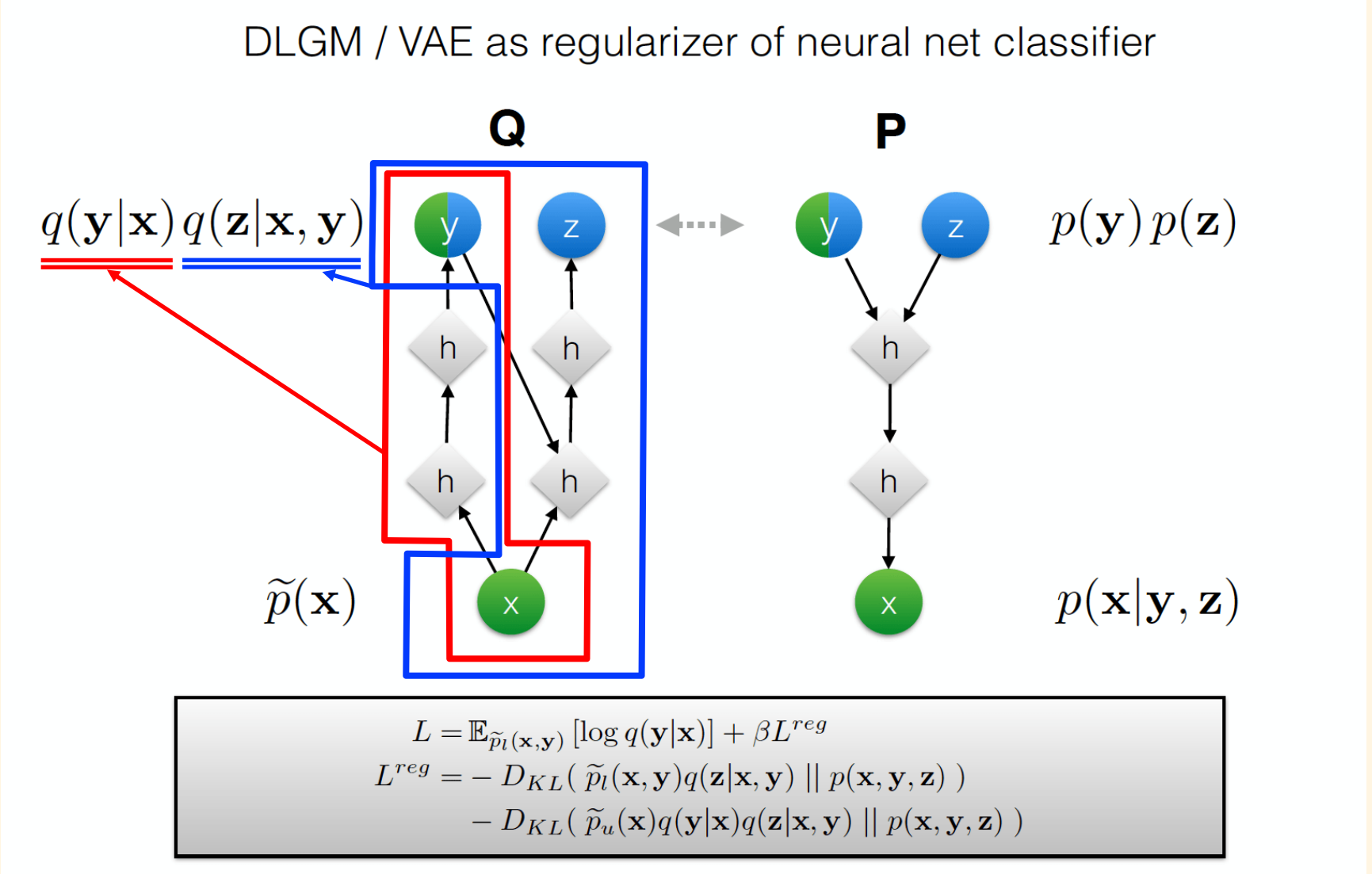

$$L = -D_{KL}(\widetilde{p}(\mathbf{x})q(\mathbf{z}|\mathbf{x})||p(\mathbf{x},\mathbf{z}))$$

Ref：
http://dpkingma.com/wordpress/wp-content/uploads/2014/10/2014-09_deep_prob_models_workshop_nomovies.pdf

# Semi-supervised Learning

M2 model

Enable to generate images with label data y.

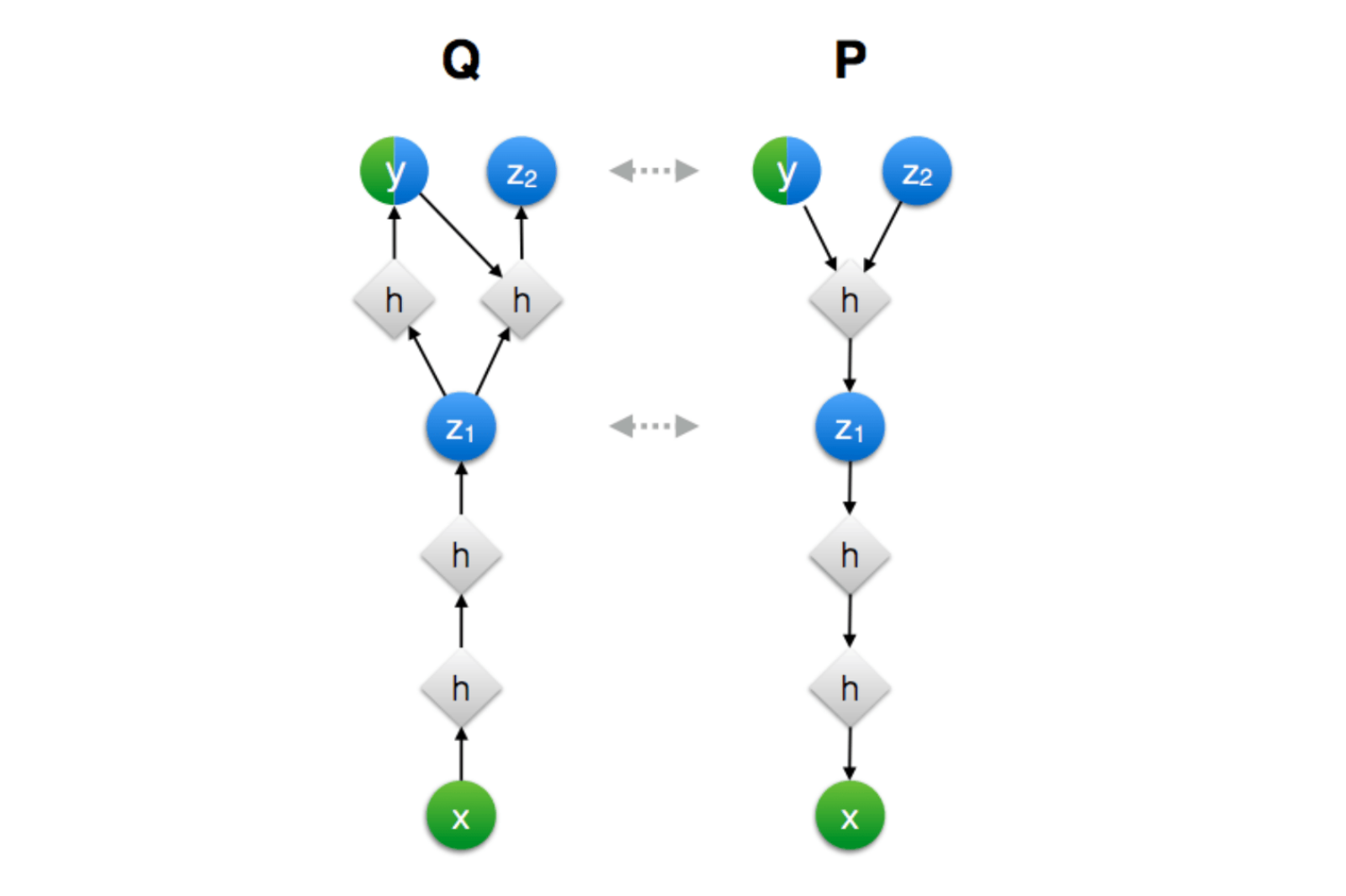



出典：
http://dpkingma.com/wordpress/wp-content/uploads/2014/10/2014-09_deep_prob_models_workshop_nomovies.pdf

# Semi-supervised Learning

M1+M2 model

Combination of M1 & M2. Generating latent variable z with M1model, and this z is used on M2 model as input data.




出典：
http://dpkingma.com/wordpress/wp-content/uploads/2014/10/2014-09_deep_prob_models_workshop_nomovies.pdf

## Reference


[1] Tutorial on Variational Autoencoders
https://arxiv.org/abs/1606.05908
[2] Auto-Encoding Variational Bayes(Diederik P Kingma, Max Welling)
https://arxiv.org/abs/1312.6114
[3] Deep Learning Practice and Theory
https://www.slideshare.net/pfi/deep-learning-practice-and-theory
[4] Lecture 13: Generative Models (Stanford CS231n: Convolutional Neural Networks for Visual Recognition)
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf
[5] Deep Learning Chapter 17 The Manifold Perspective on Representation Learning
http://www.deeplearningbook.org/version-2015-10-03/contents/manifolds.html