Variational AutoEncoders

Dong-Ju Kim
School of Computer Engineering
Kwangwoon University

Variational Autoencoder

➢Variational Autoencoder(VAE)

- Kingma et al., "Auto-Encoding Variational Bayes," 2013
- Generative Model + Stacked Autoencoder
  - ✓Based on Variational approximation

Variational approximations Variational methods define a lower bound

$$\mathcal{L}(\boldsymbol{x};\boldsymbol{\theta}) \leq \log p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta}). \tag{7}$$

Unsupervised Learning

➢Unsupervised Learning

- No label and thus self learning
- More challenging than supervised learning

➢Unsupervised Neural Network Models

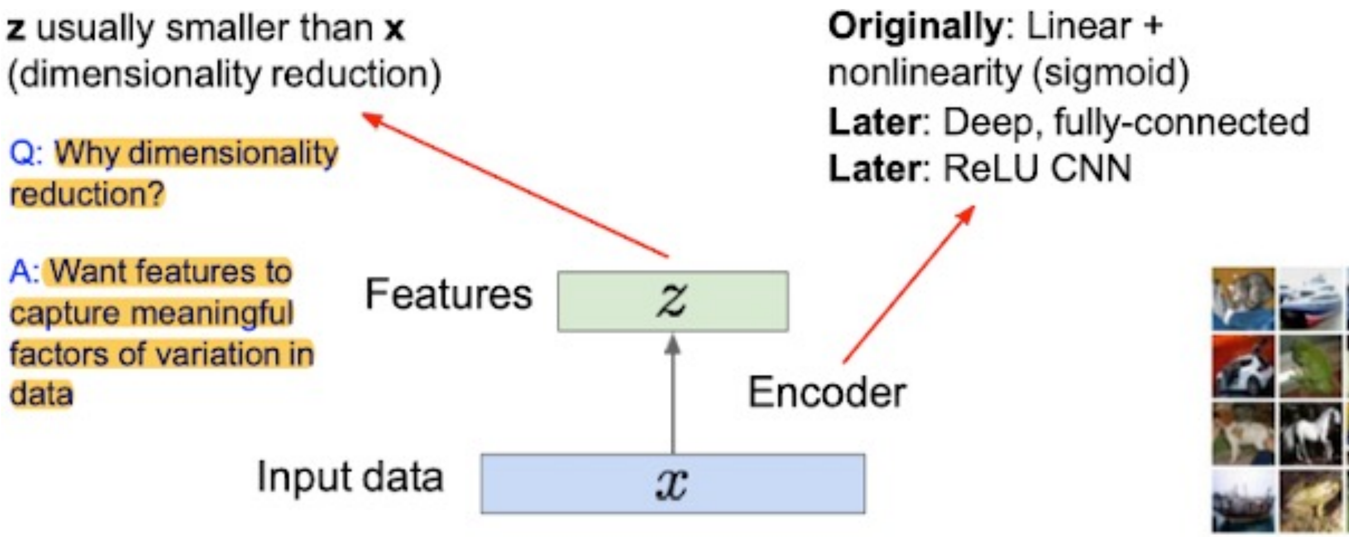
- Boltzmann machine
- Auto-encoder or variational inference
- Generative adversarial network

Contents

Background : Autoencoder

- Purpose of Autoencoder : Encoder to make dimension lower from unlabeled training data

Unlabeled Input → Encode Input → Reconstruct Input

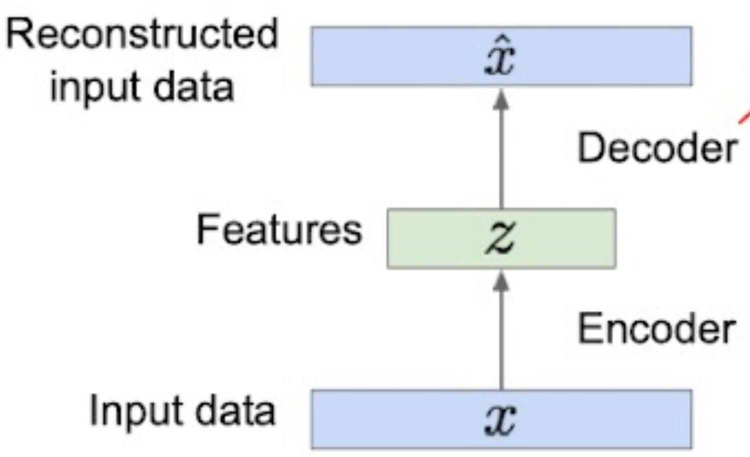


Background : Autoencoder

- Purpose of Autoencoder : Encoder to make dimension lower from unlabeled training data

Unlabeled Input → Encode Input → Reconstruct Input

How to learn this feature representation?
Train such that features can be used to reconstruct original data
"Autoencoding" - encoding itself

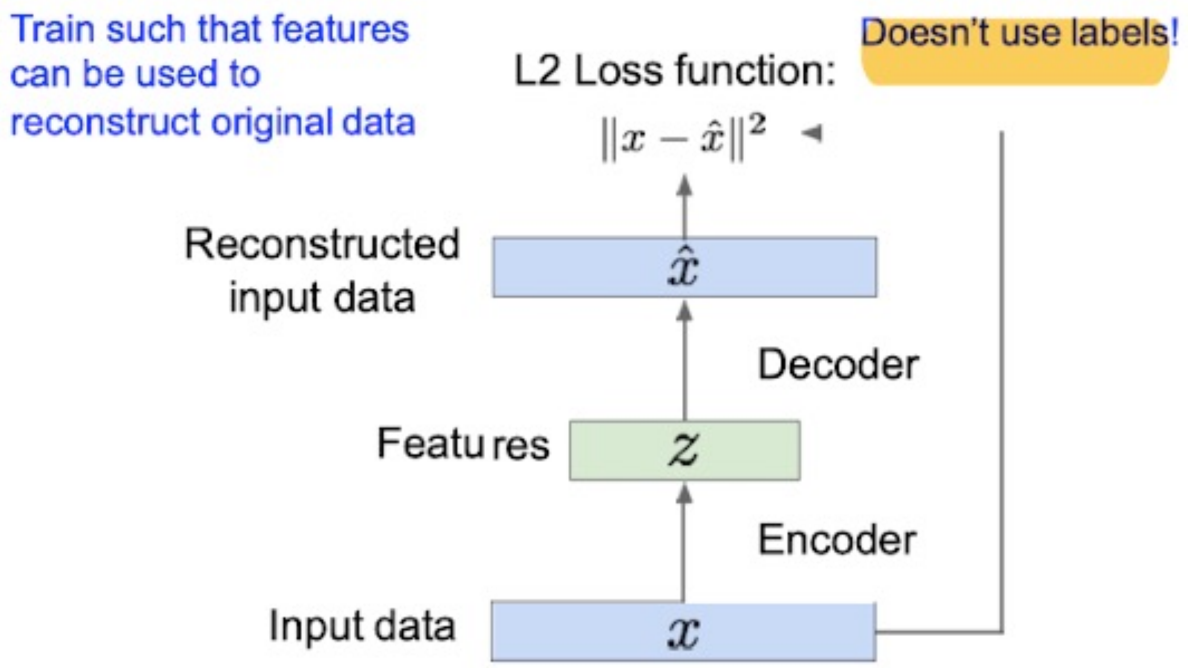


Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)

Background : Autoencoder

- Purpose of Autoencoder : Encoder to make dimension lower from unlabeled training data

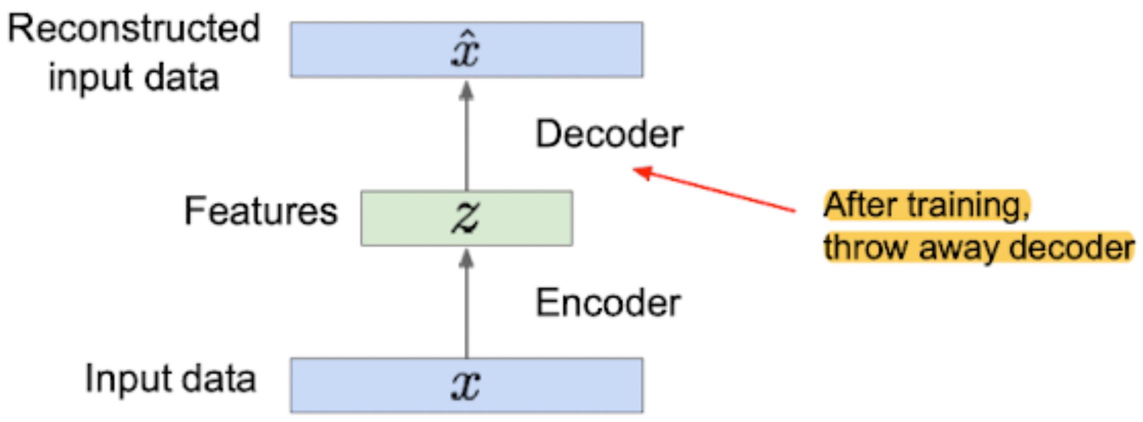
Unlabeled Input → Encode Input → Reconstruct Input



Background : Autoencoder

- Purpose of Autoencoder : Encoder to make dimension lower from unlabeled training data

Unlabeled Input → Encode Input → Reconstruct Input



Background : Autoencoder

- Purpose of Autoencoder : Encoder to make dimension lower from unlabeled training data

Unlabeled Input

Encode Input

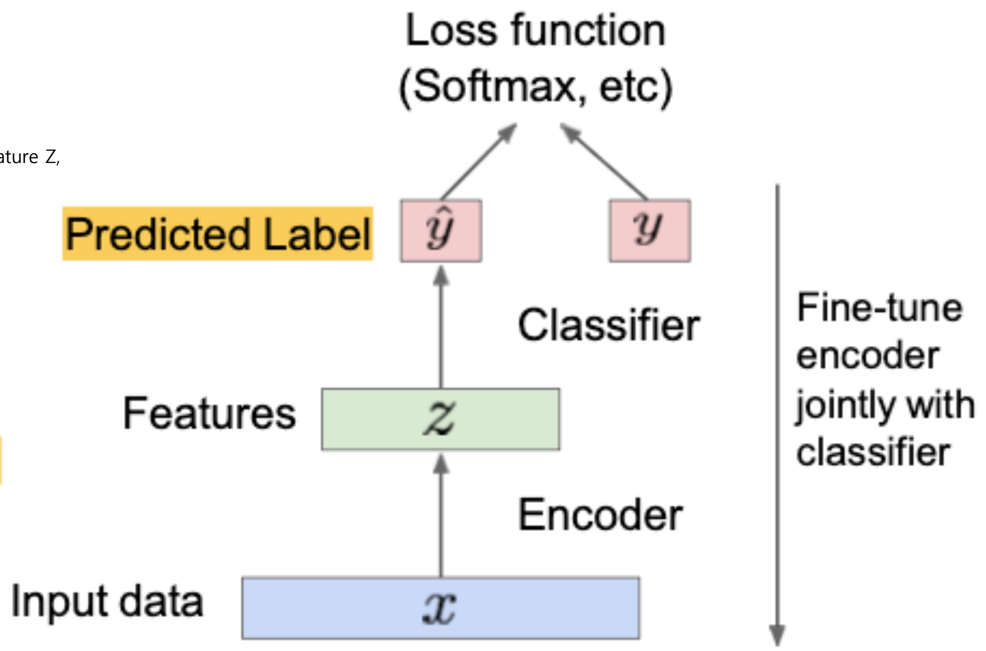
Reconstruct Input

The autoencoder excludes the decoder-part after training is completed,
and just uses the encoder part as the right fig.

Reason: Let, there are few labeled data, and there are many unlabeled data.
No matter how much you pass this data through CNN, you cannot find the main features well,
for the reason that there are few training datasets.
So, with a lot of data that is not labeled in advance, if you let the autoencoder extract the main feature Z,
and then put in a small amount of labeled data,
you can learn with a smaller amount of data much faster than you learn with random initialization.

- Feature extraction
- Data Compression
- Dimensionality reduction
- Learning generative mode

Encoder can be used to initialize a **supervised** model



Autoencoders can reconstruct data,
and can learn features to initialize a
supervised model

Features capture factors of variation in training data.
Can we generate new images from an autoencoder?

Variational Autoencoders : intuition

- Probabilistic spin on autoencoders : Let us sample from the model to generate data

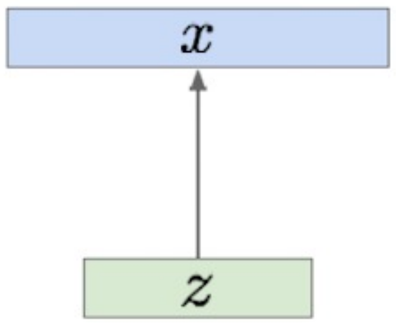
- Let's try to imagine an animal
- Then you might think about a creature having 4 legs, one tail, feather, and so on
- However, you do not come up an idea about imaginable creatures such as Haetae, Unicorn, Dragon, etc.
- Our imagination could be a latent variable to create a real object or sample
- VAE uses latent variables, so called an expressive model

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation **z**

Intuition (remember from autoencoders!): **x** is an image, **z** is latent factors used to generate **x**: attributes, orientation, etc.

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior $p_{\theta^*}(z)$

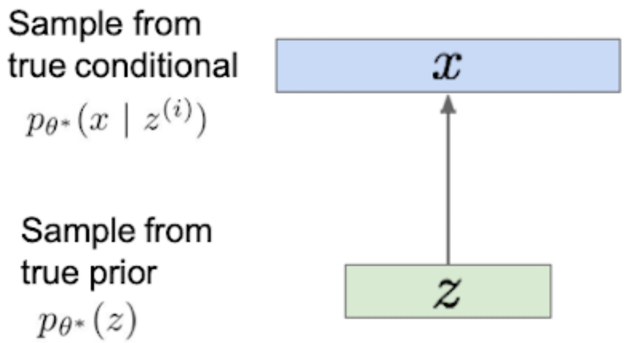


Variational Autoencoders

➢ Notions to define VAE

- X : data we want to model
- z : latent variable, ex)imagination
- P(X) : prob distribution of the data, ex) animal kingdom
- P(z) : prob distribution of latent variable, ex) brain, a source of imagination
- P(X|z) : distribution of generating data given latent variable, ex) turning imagination into real animal

We want to estimate the true parameters θ^* of this generative model.

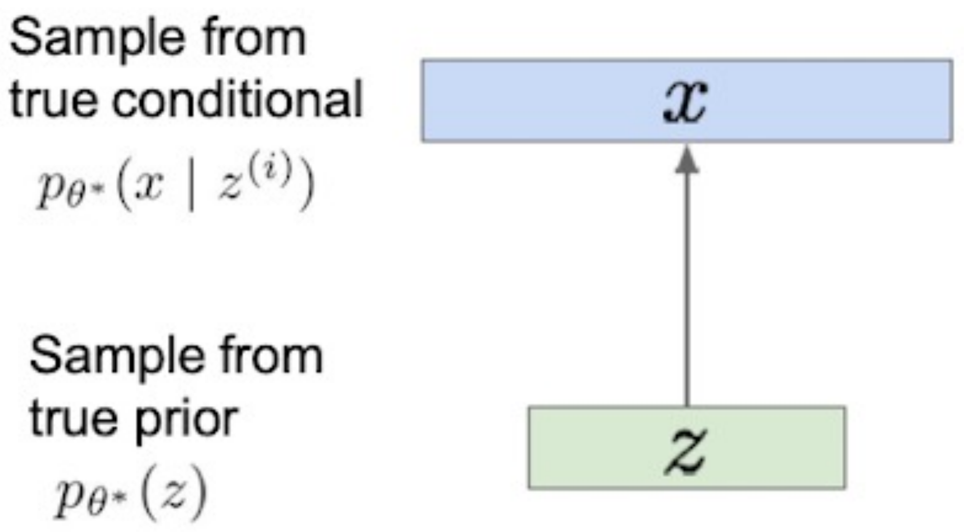


don't know P(x) and p(z), but the goal is to get theta star that expresses it well

➢ Objective: model the date, finding P(X)

$$P(X) = \int P(X|z)P(z)dz$$

Variational Autoencoders

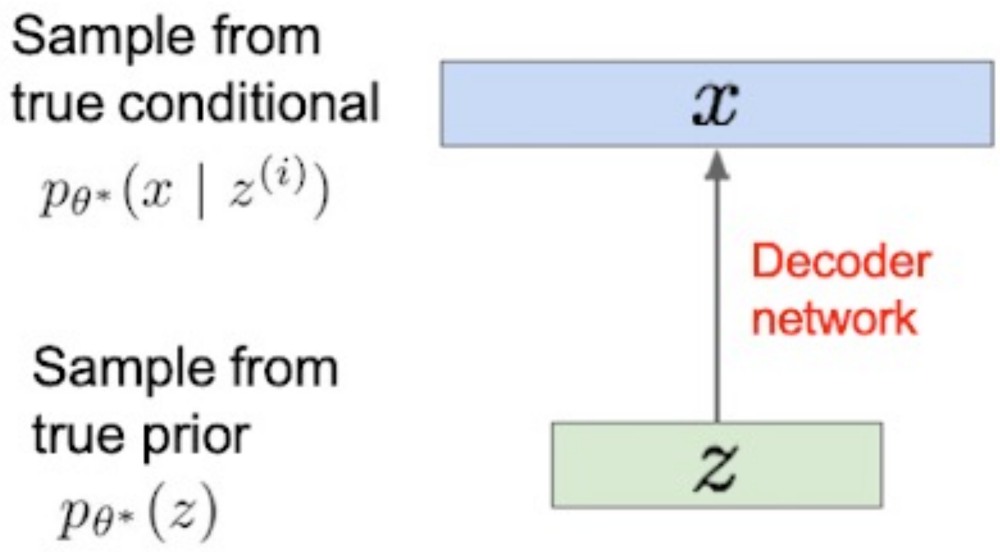


We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Assume that distribution is gaussian



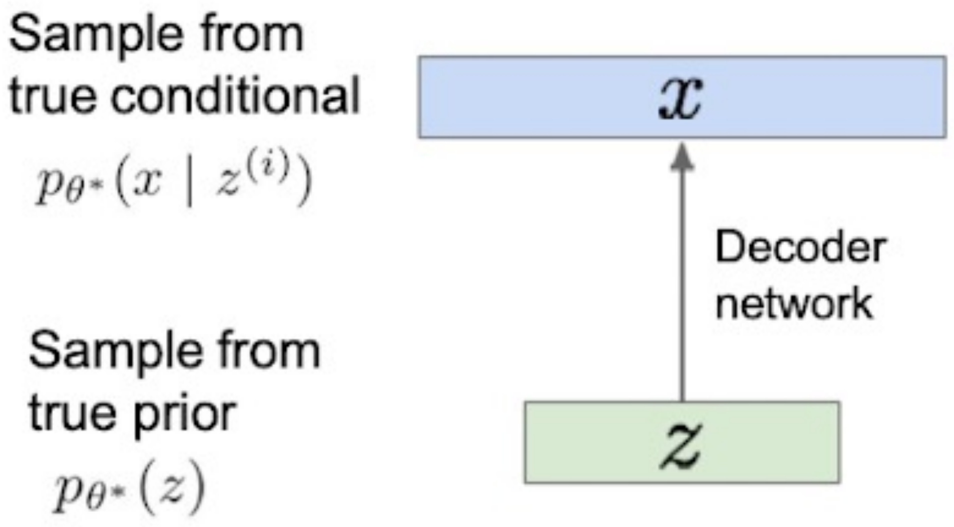
We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian.

Conditional p(x|z) is complex (generates image) => represent with neural network

Variational Autoencoders



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

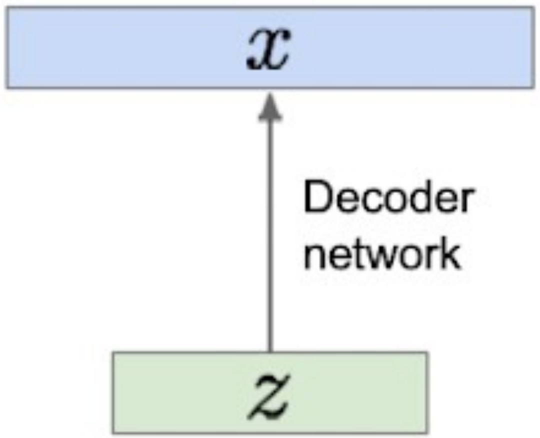
$$P(X) = \int P(X|z)P(z)dz$$

Variational Autoencoders

We want to estimate the true parameters θ^* of this generative model.

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$



Sample from true prior

$p_{\theta^*}(z)$

How to train the model?

Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

$$P(X) = \int P(X|z)P(z)dz$$

$$\frac{P(X,z)}{P(z)} = P(X|z)$$

$$\rightarrow P(X,z) = P(X|z) * P(z)$$

$$\Rightarrow P(X,z) \mathit{means}\ X\ \mathit{and}\ z\ \mathit{joint\ probability}$$

$$\mathit{integrating\ every}\ P(X,z)\ \mathit{for}\ z\ \mathit{space}\ \ \mathit{is}\ P(X)$$

Variational Autoencoders

- Data likelihood : $P(x) = \int p(z)p(x|z)dz$ Intractable to compute $p(x|z)$ for every z

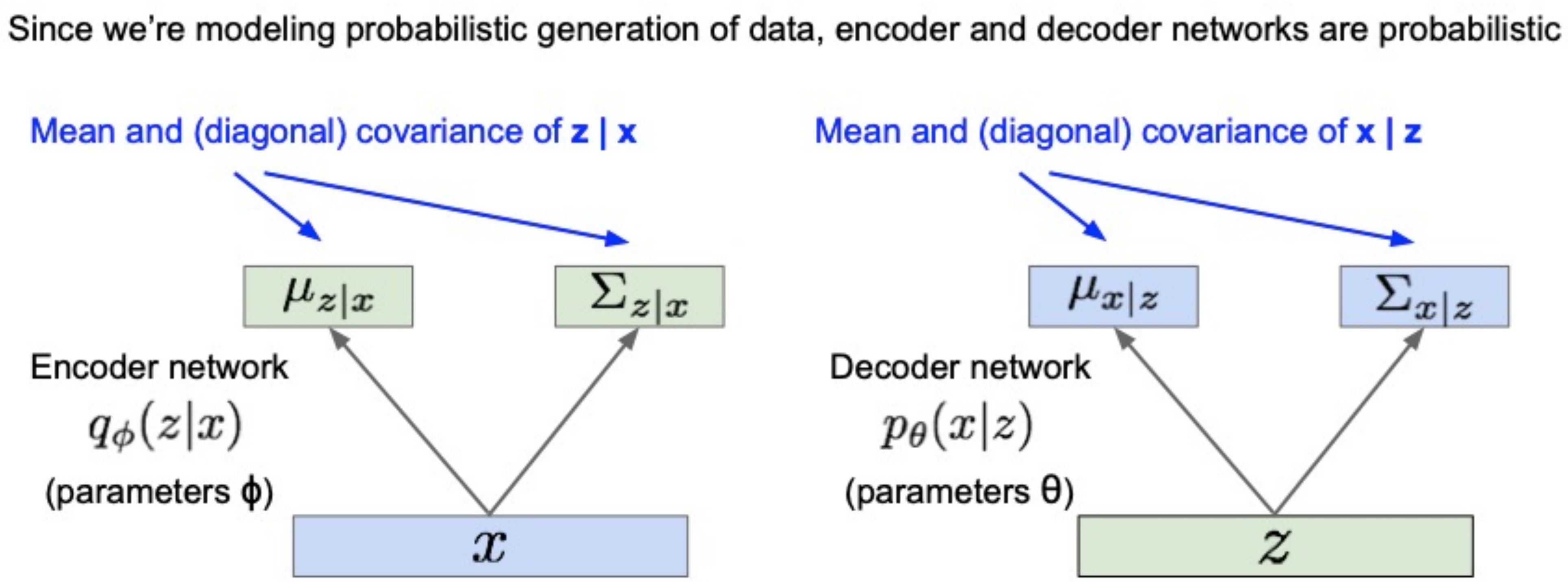
  Simple gaussian prior

  Decoder NN

- Posterior density also intractable : $P(z|x) = p(x|z)p(z)/p(x)$
  - We can't calculate p(x) so we also cant' calculate P(z|x)

- Solution for this prob.
  - Decoder network modelling P(x|z), define additional encoder network q(z|x) that approximates p(z|x)
  - Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which can optimize

Variational Autoencoders



- To make easy to read make q_pi p_theta to Q and P from now

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z \sim Q(z|X)}[log P(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

Encoder network

$$E_{z \sim Q(z|X)}\left[log \frac{Q(z|X)}{P(z)}\right] = \int_z \log \frac{Q(z|X)}{P(z)} Q(z|X) dz$$

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z\sim Q(z|X)}[\log P(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[\log \frac{P(X|z)P(z)}{P(z|X)}\right] \quad (Bayes'Rule)$$

$(Bayes'Rule)$

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

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

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z \sim Q(z|X)}[\log P(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[\log \frac{P(X|z)P(z)}{P(z|X)}\right] \ (Bayes' Rule)$$

$$= E_z\left[\log \frac{P(X|z)P(z)}{P(z|X)} * \frac{Q(z|X)}{Q(z|X)}\right] \ (Multiply\ by\ constant)$$

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z\sim Q(z|X)}[logP(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)}\right] \ (Bayes'Rule)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)} * \frac{Q(z|X)}{Q(z|X)}\right] \ (Multiply\ by\ constant)$$

$$= E_z[logP(X|z)\ ] - E_z\left[log\frac{Q(z|X)}{P(z)}\right] + E_z[\log\frac{Q(z|X)}{p(z|X)}] \ (Logarithms)$$

KL Divergence

When there are any two pdf, a method that is often used to compare mathematically how similar these two pdf are

The KL Divergence can take on values in [0,∞]

$$D_{KL}(p||q) = E[\log p(x) - \log q(x)]$$

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z\sim Q(z|X)}[logP(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)}\right]\ (Bayes'Rule)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)} * \frac{Q(z|X)}{Q(z|X)}\right]\ (Multiply\ by\ constant)$$

$$= E_z[logP(X|z)] - E_z\left[log\frac{Q(z|X)}{P(z)}\right] + E_z[\log\frac{Q(z|X)}{p(z|X)}]\ (Logarithms)$$

$$= E_z[logP(X|z)] - D_{KL}(Q(z|X)||P(z)) + D_{KL}(Q(z|X)||P(z|X))$$

KL Divergence

When there are any two pdf, a method that is often used to compare mathematically how similar these two pdf are

The KL Divergence can take on values in [0,∞]

$$D_{KL}(p||q) = E[\log p(x) - \log q(x)]$$

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z\sim Q(z|X)}[logP(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)}\right] \ (Bayes'Rule)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)} * \frac{Q(z|X)}{Q(z|X)}\right] \ (Multiply\ by\ constant)$$

$$= E_z[logP(X|z)\,] - E_z\left[log\frac{Q(z|X)}{P(z)}\right] + E_z[\log\frac{Q(z|X)}{p(z|X)}] \ \ (Logarithms)$$

$$= E_z[logP(X|z)\,] - \underline{D_{KL}(Q(z|X)||P(z))} + D_{KL}(Q(z|X)||P(z|X))$$

Decoder Network gives P(X|z), can compute estimate of this term through sampling. (sampling differentiable through reparam. Trick.)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution

P(z|X) intractable, can't compute this KL term: but, KL divergence always >= 0

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z\sim Q(z|X)}[logP(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)}\right]\ (Bayes'Rule)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)} * \frac{Q(z|X)}{Q(z|X)}\right]\ (Multiply\ by\ constant)$$

$$= E_z[logP(X|z)] - E_z\left[log\frac{Q(z|X)}{P(z)}\right] + E_z[\log\frac{Q(z|X)}{p(z|X)}]\ (Logarithms)$$

$$= E_z[logP(X|z)] - D_{KL}(Q(z|X)||P(z)) + \underbrace{D_{KL}(Q(z|X)||P(z|X))}_{\geq 0}$$

Since dKL is larger when it is negative and different, maximizing logP(X) means minimizing this equation, and minimizing it means that Q(z|X) and P(z) become similar.
This will cause the encoder to make approximate posterior distribution close to prior

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z\sim Q(z|X)}[logP(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)}\right] \ (Bayes'Rule)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)} * \frac{Q(z|X)}{Q(z|X)}\right] \ (Multiply\ by\ constant)$$

$$= E_z[logP(X|z)] - E_z\left[log\frac{Q(z|X)}{P(z)}\right] + E_z[\log\frac{Q(z|X)}{p(z|X)}] \ (Logarithms)$$

$$= E_z[logP(X|z)] - D_{KL}(Q(z|X)||P(z)) + \underbrace{D_{KL}(Q(z|X)||P(z|X))}_{\geq 0}$$

this term means, Reconstruct the input data, and it shows how likely it is to be from z

Variational Autoencoders : derivation

- (Log) Data likelihood :

$$\log P(X) = E_{z\sim Q(z|X)}[logP(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)}\right] \ (Bayes'Rule)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)} * \frac{Q(z|X)}{Q(z|X)}\right] \ (Multiply\ by\ constant)$$

$$= E_z[logP(X|z)] - E_z\left[log\frac{Q(z|X)}{P(z)}\right] + E_z[\log\frac{Q(z|X)}{p(z|X)}] \ (Logarithms)$$

$$= \boxed{E_z[logP(X|z)] - D_{KL}(Q(z|X)||P(z))} + D_{KL}(Q(z|X)||P(z|X))$$

Tractable Lower Bound
Which can take gradient of and optimize

Variational Autoencoders : derivation

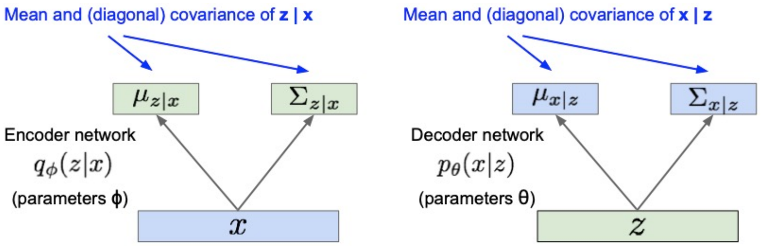
- (Log) Data likelihood :

$$\log P(X) = E_{z\sim Q(z|X)}[logP(X)] \quad (P(X)\ Does\ not\ depend\ on\ z)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)}\right] \ (Bayes'Rule)$$

$$= E_z\left[log\frac{P(X|z)P(z)}{P(z|X)} * \frac{Q(z|X)}{Q(z|X)}\right] \ (Multiply\ by\ constant)$$

$$= E_z[logP(X|z)\,] - E_z\left[log\frac{Q(z|X)}{P(z)}\right] + E_z[\log\frac{Q(z|X)}{p(z|X)}] \ (Logarithms)$$



To explain Variational approximations *bring* $\mathrm{P} = \mathrm{p}_\theta \ \ Q = q_\phi$ back to life.

$$=\underbrace{E_z[logp_\theta(X|z)\,] - D_{KL}(q_\phi(z|X)||P_\theta(z))}_{\mathcal{L}(X,\theta,\phi)} + D_{KL}(q_\phi(z|X)||p_\theta(z|X))$$

$$logp_\theta(X) \geq \mathcal{L}(X,\theta,\phi)$$

X = x^i

$$\theta^*,\phi^* = \arg\max_{\theta,\phi}\sum_{i=1}^{N}\mathcal{L}(x^i,\theta,\phi)$$

Training : maximize lower bound

Variational approximations Variational methods define a lower bound

$$\mathcal{L}(\boldsymbol{x};\boldsymbol{\theta}) \leq \log p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta}). \qquad (7)$$

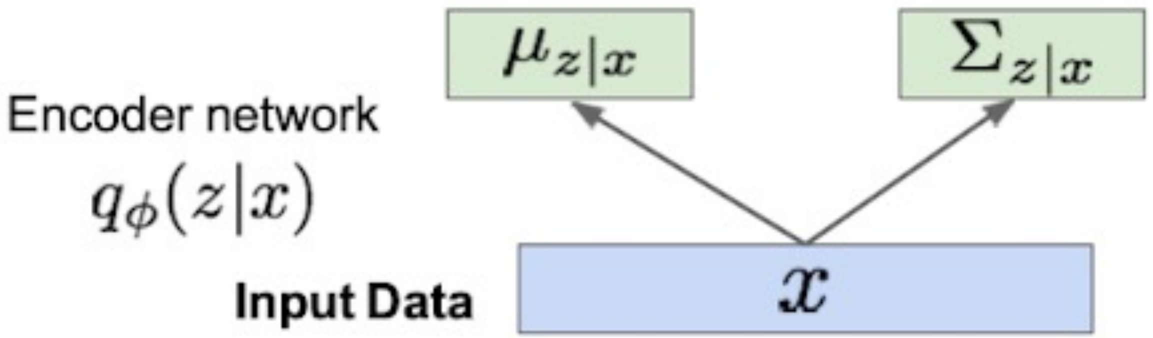
Variational Autoencoders : training process

$$\underbrace{E_z[\log p_\theta(X|z)] - D_{KL}(q_\phi(z|X)||P_\theta(z))}_{\mathcal{L}(X,\theta,\phi)}$$

$$+ D_{KL}(q_\phi(z|X)||p_\theta(z|X))$$

- Maximizing the lower bound.

We can't calculate this part now so just ignore them.

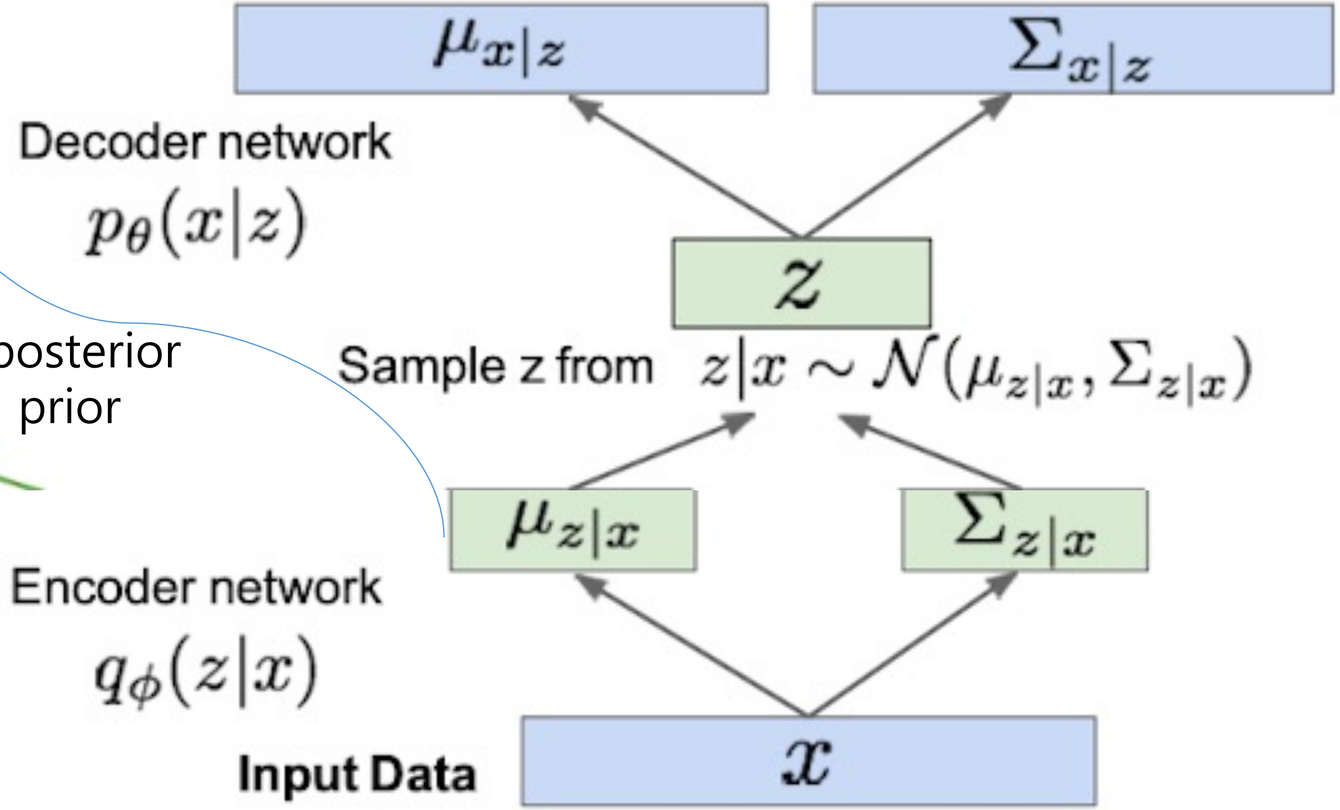


Variational Autoencoders : training process

$$\underbrace{E_z[\log p_\theta(X|z)] - D_{KL}(q_\phi(z|X)||P_\theta(z))}_{\mathcal{L}(X,\theta,\phi)}$$

By maximizing this term
maximizing log likelihood lower bound

Make approximate posterior
distribution close to prior



Variational Autoencoders : training process

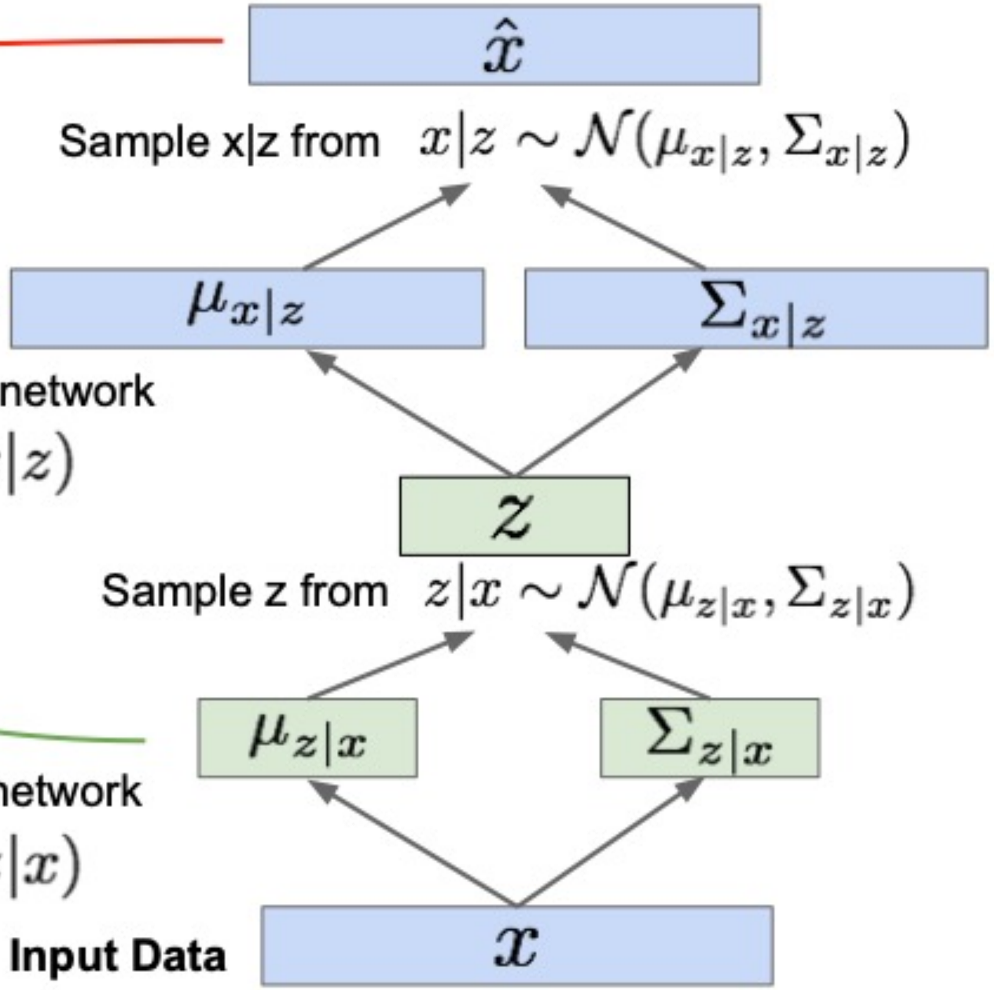
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Maximize likelihood of original input being reconstructed

Make approximate posterior distribution close to prior



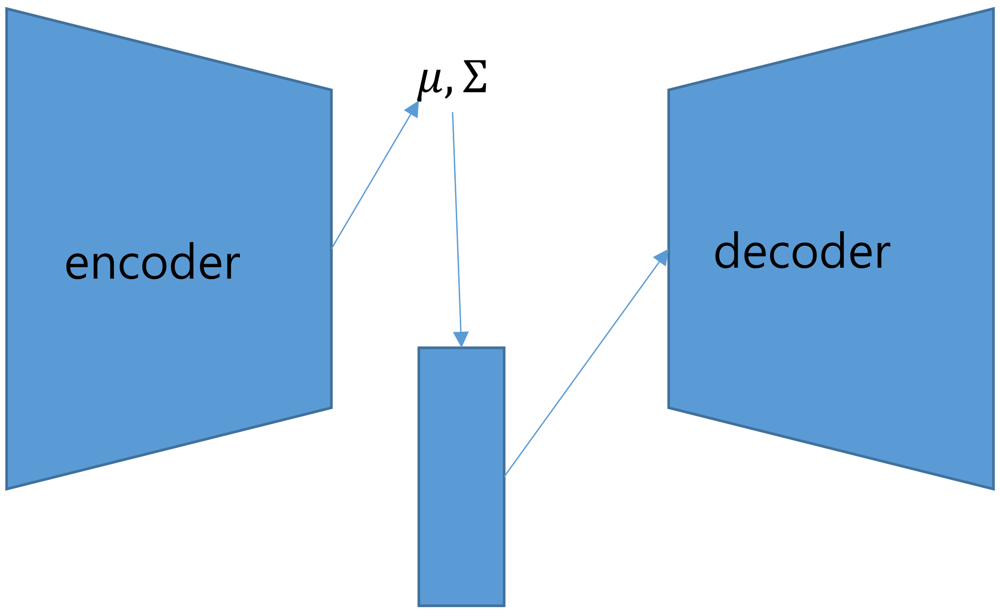
For every minibatch of input data: compute this forward pass, and then backprop!

VAE : Dissecting the objective

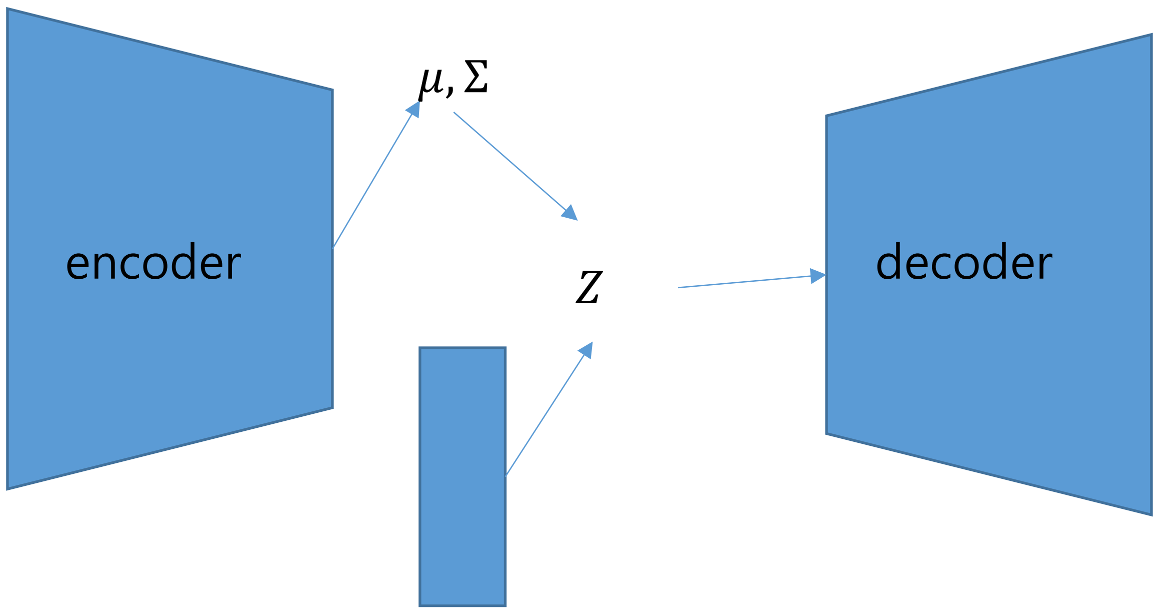
$$logP(X) = = E_z[logP(X|z)] - D_{KL}(Q(z|X)||P(z)) + D_{KL}(Q(z|X)||P(z|X))$$

- VAE try to find the lower bound of $logP(X)$, model of our data, under some error of $D_{KL}[Q(z|X)||P(z|X)]$
- Then, this model could be found by maximizing $E[logP(X|z)]$, and minimizing $D_{KL}[Q(z|X)||P(z)]$
- Maximization of $E[logP(X|z)]$ could be obtained by a maximum likelihood estimation using log loss or regression loss
- For $D_{KL}[Q(z|X)||P(z)]$, let's sample $P(z)$ later, and thus choose the easiest choice, $N(0,1)$, for $P(z)$. Therefore, make $Q(z|X)$ as close as possible to $N(0,1)$

Reparameterization trick



$Z : N(\mu, \Sigma)$



$Z : N(0, 1)$

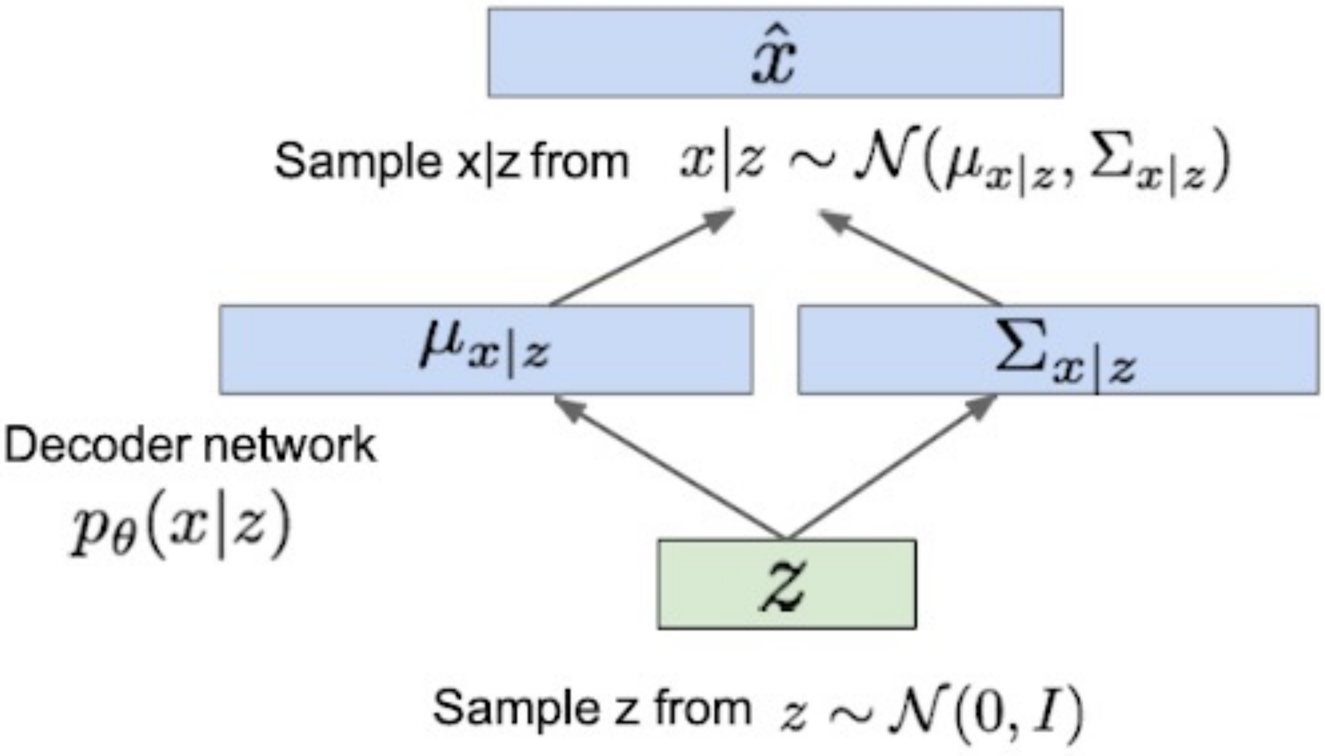
Sampling Process

$$Z \sim N(\mu, \Sigma)$$

$$Z = \mu + \Sigma^{\frac{1}{2}} * \varepsilon$$
$$\varepsilon \sim N(0,1)$$

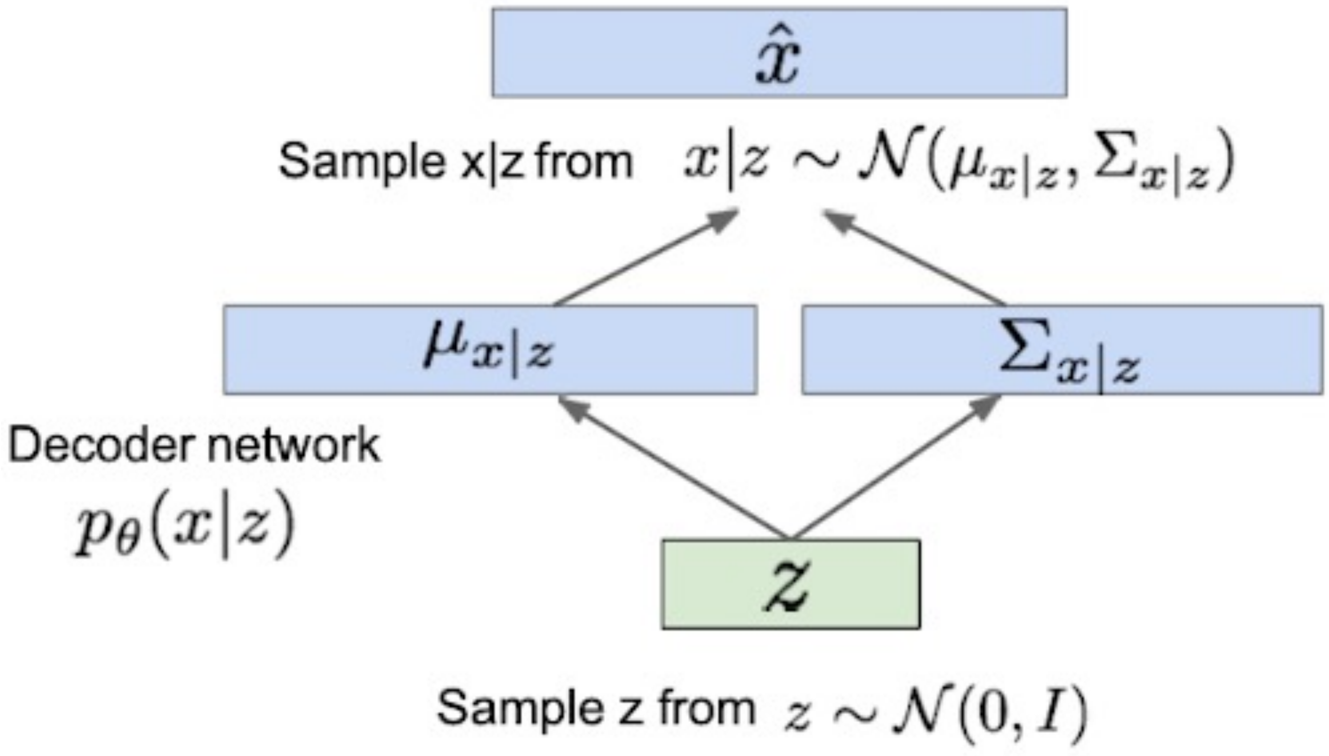
VAE : Generating Data

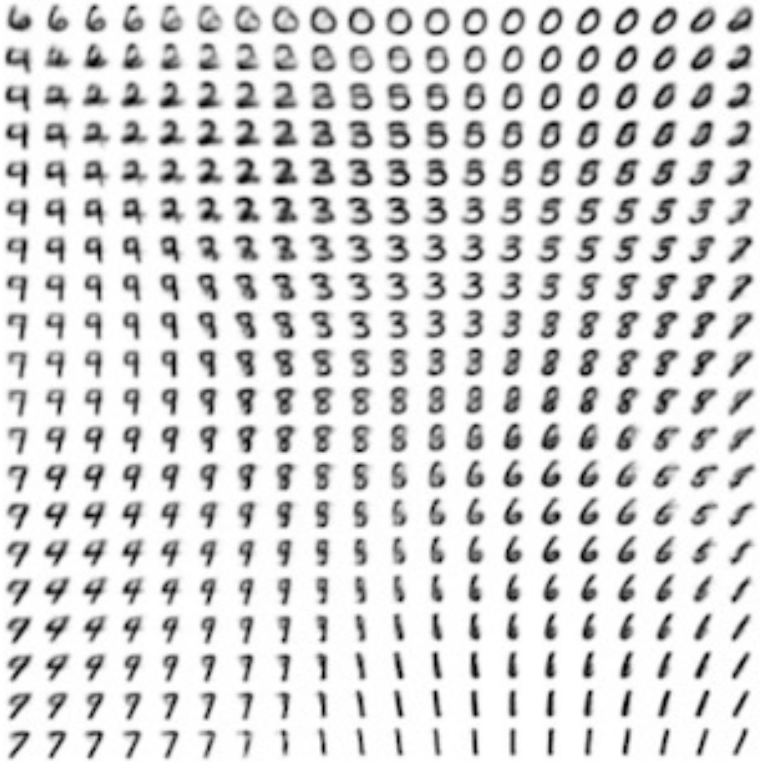
Use decoder network. Now sample z from prior!



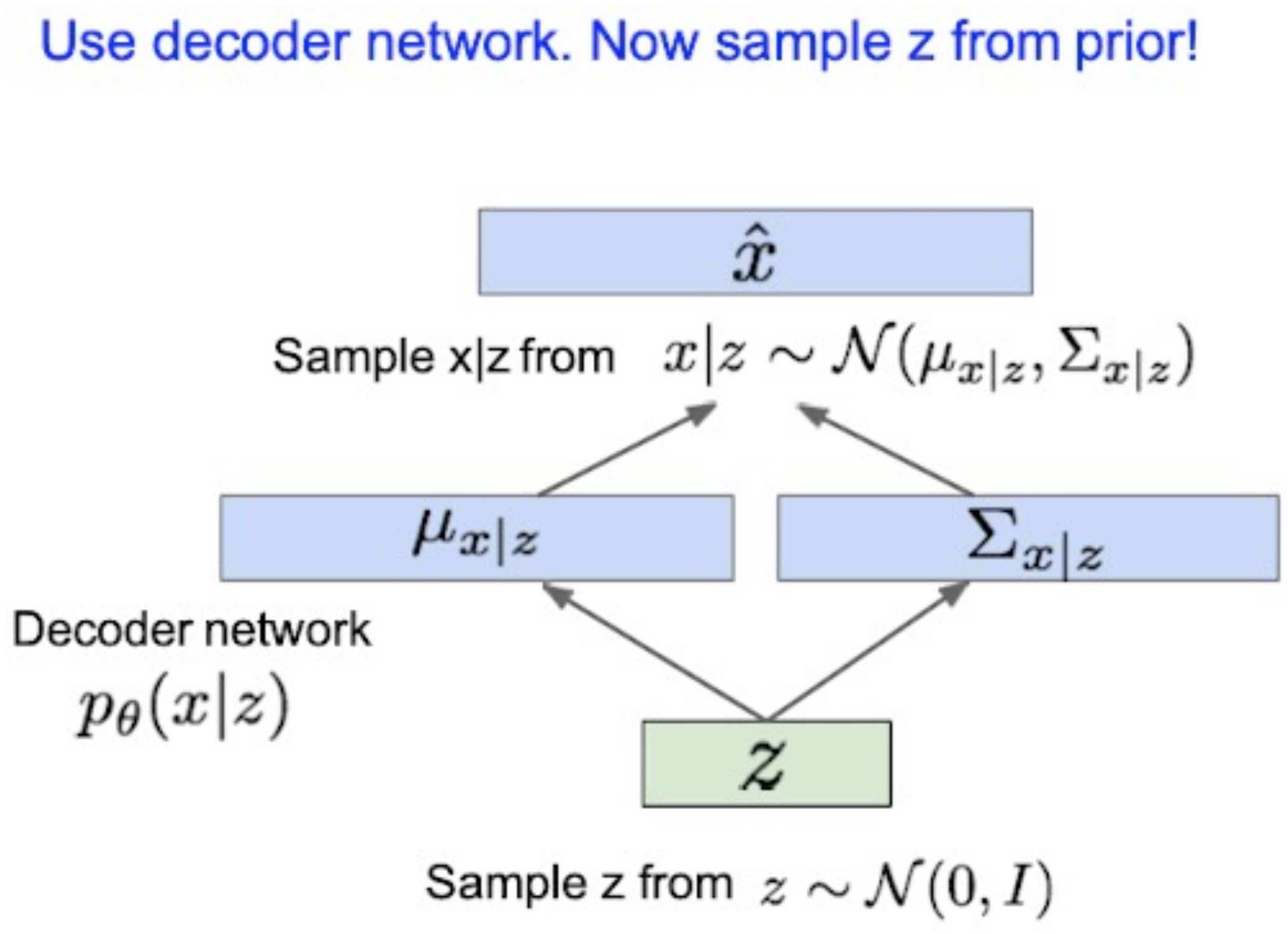
VAE : Generating Data

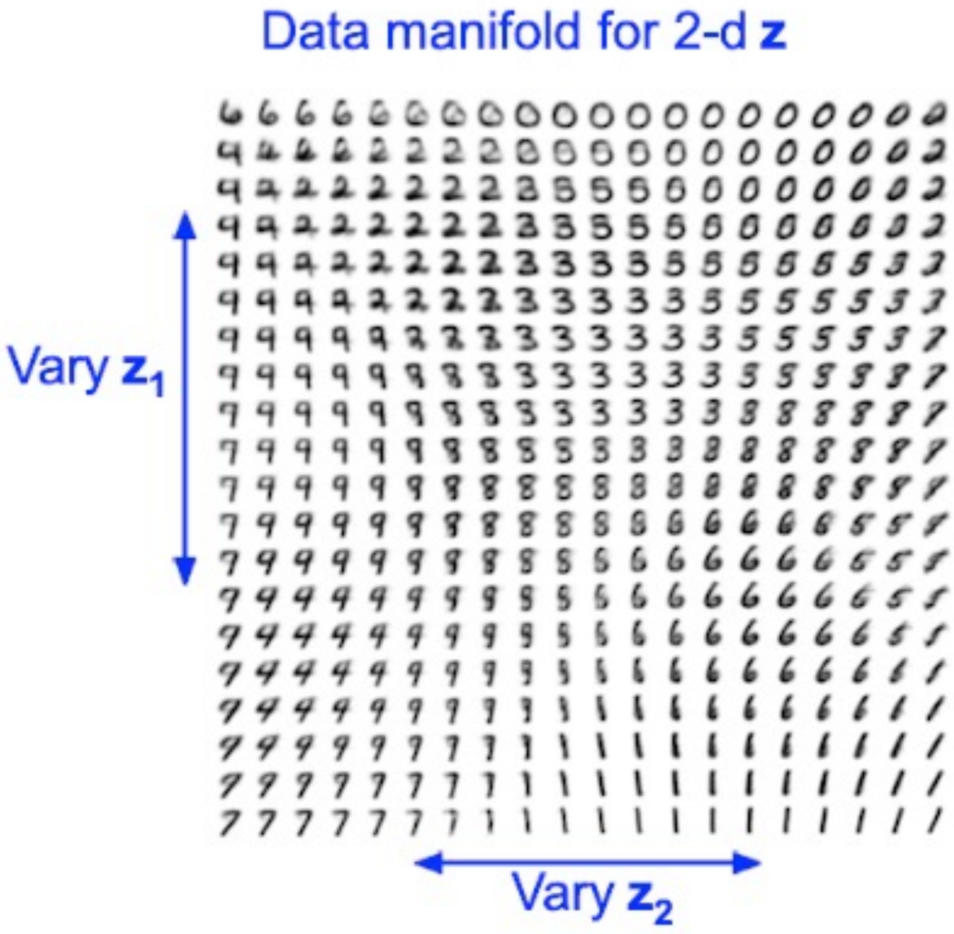
Use decoder network. Now sample z from prior!



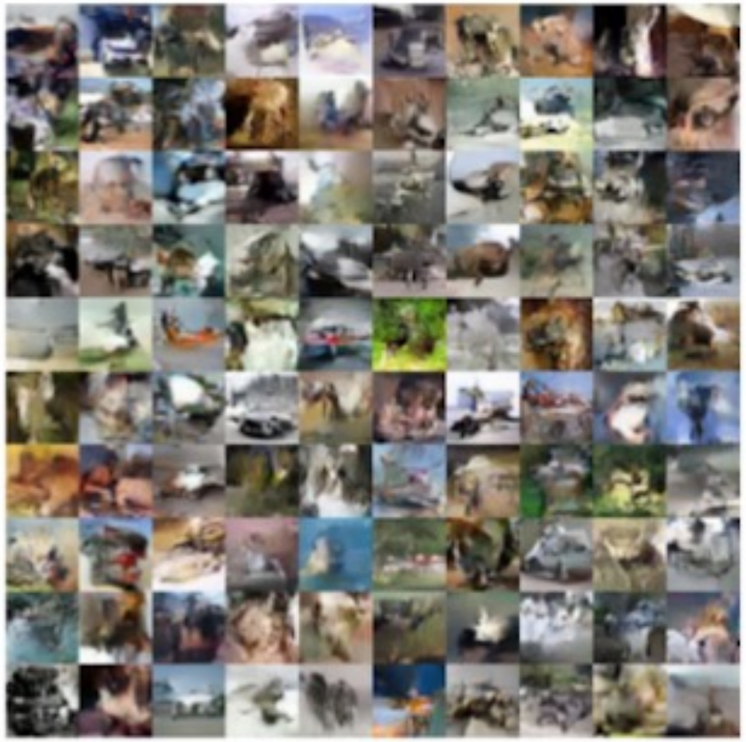


VAE : Generating Data





Variational Autoencoders: Generating Data!



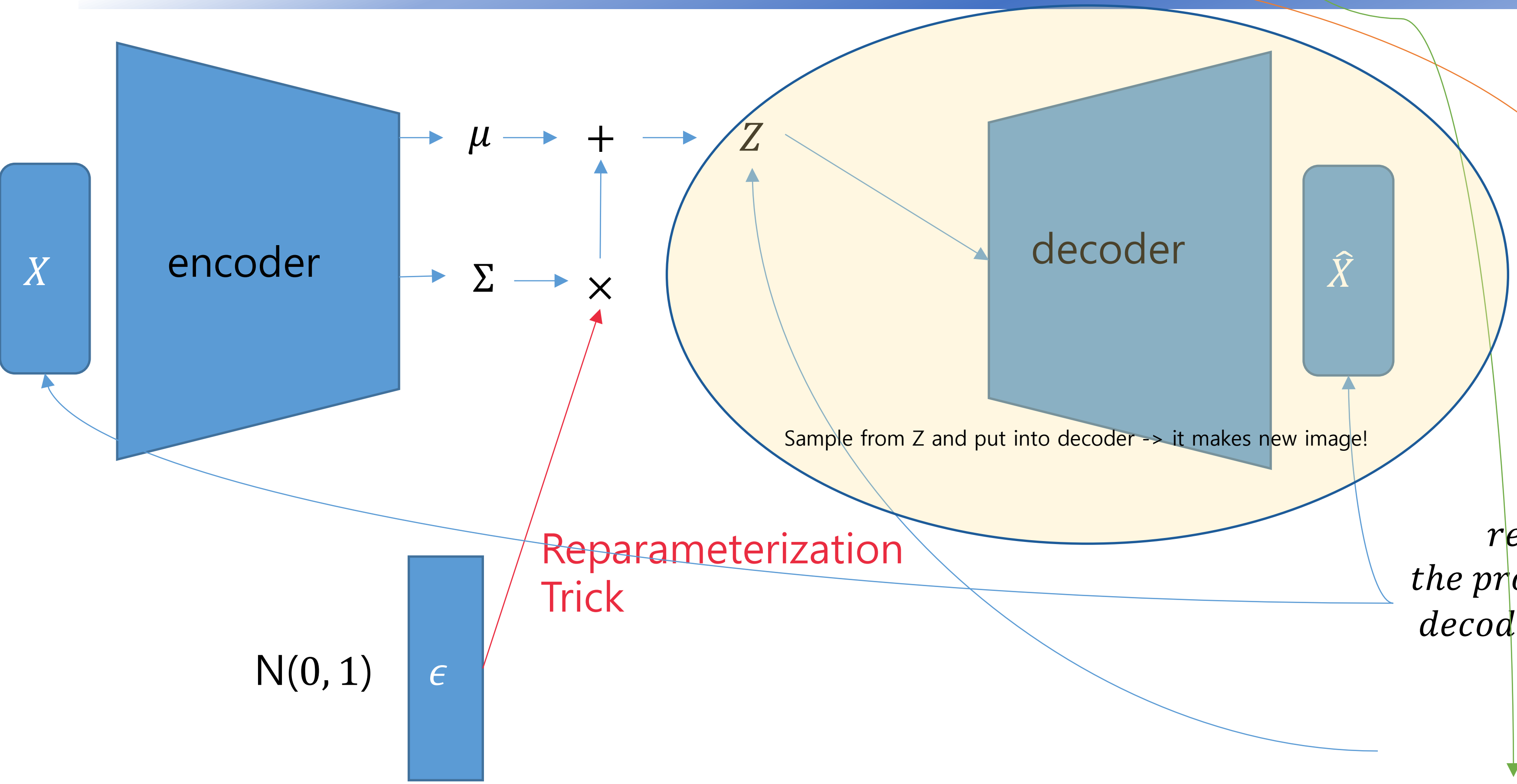
32x32 CIFAR-10

Labeled Faces in the Wild

Review

$$logP(X) = E_z[logp_\theta(X|z)] - D_{KL}(q_\phi(z|X)||P_\theta(z)) + D_{KL}(q_\phi(z|X)||p_\theta(z|X))$$

$can't\ calculate\ this$ ⬆️ $term$



Encoder

Input : data

output : mu and sigma

Z

training : needed to make as similar as N(0,1)

Gen : from Z we do sampling to generate

Decoder

Input : sampled result from Z

Output : new data

$reconstruction\ error\ term:\ E_z[\log P(X|z)]$

$the\ probability\ of\ X\ is\ reconstructed\ through\ decoder$

$Dissecting\ the\ Objective\ d_{KL}[Q(z|X)||P(z)]\ make\ Q(z|X)\ follow\ P(z)$

$let\ P(z) is\ N(0,1)$

$result: Q(z|X) follow\ N(0,1)$

Variational Autoencoders

- Probabilistic spin to traditional autoencoders -> allows generating data
- Defines an intractable density -> derive and optimize a (variational) lower bound
- Pros:
  - Principled approach to generative models
  - Allows inference of $q(z|x)$, can be useful feature representation for other tasks
- Cons:
  - Maximizes lower bound of likelihood : okay, but not as good evaluation as PixelRNN/PixelCNN
  - Samples blurrier and lower quality compared to state-of-the-art (GANs)

references

- https://www.youtube.com/watch?v=5WoItGTWV54&t=2175s
  - http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture11.pdf
- https://youtu.be/GbCAwVVKaHY : reparam trick
- https://www.oliviergibaru.org/courses/ML_VAE.html#VAE2 :Dissecting the objective
- C.Doersch, “Tutorial on Variational Autoencoder,” arXiv:1606.05908v2 [stat.ML], 13 August 2016
- https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

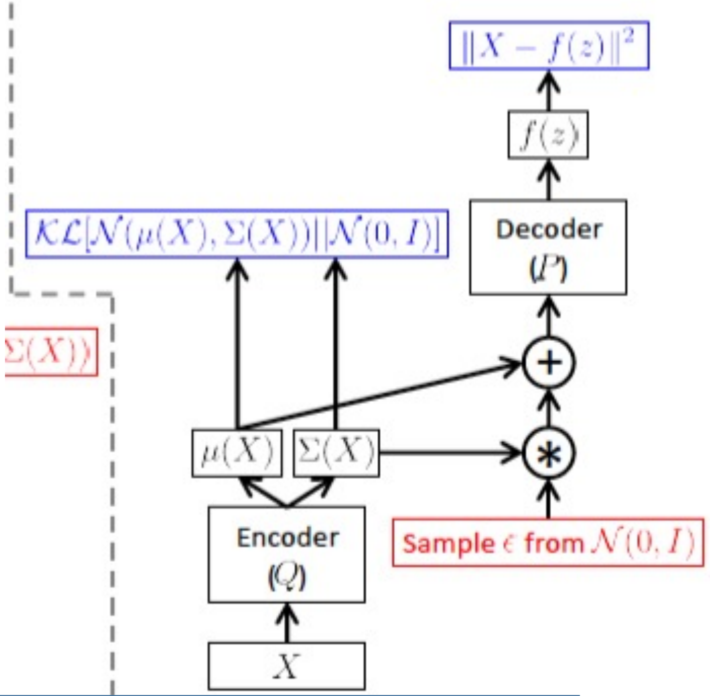
감사합니다.

appendix

- How to make VAE using KERAS

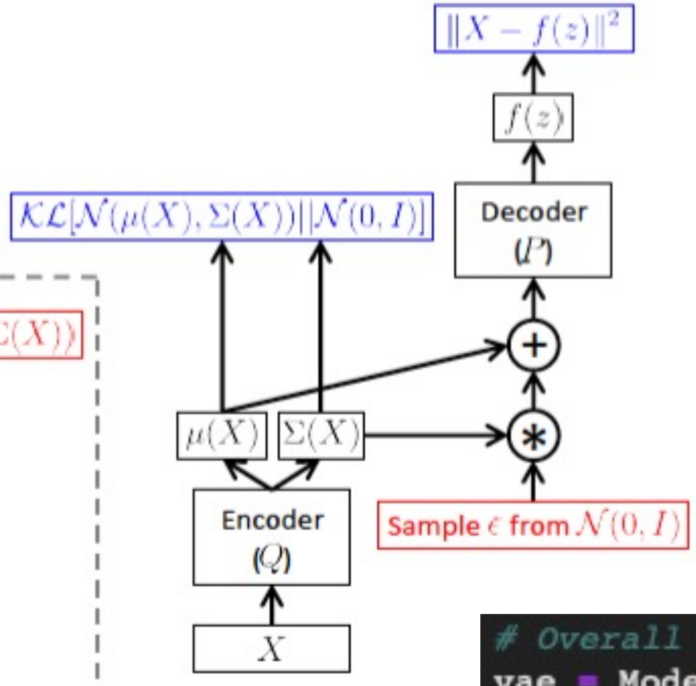
Encoder Net : Q(z|X)

```python
from tensorflow.examples.tutorials.mnist import input_data
from keras.layers import Input, Dense, Lambda
from keras.models import Model
from keras.objectives import binary_crossentropy
from keras.callbacks import LearningRateScheduler

import numpy as np
import matplotlib.pyplot as plt
import keras.backend as K
import tensorflow as tf


m = 50
n_z = 2
n_epoch = 10


# Q(z|X) -- encoder
inputs = Input(shape=(784,))
h_q = Dense(512, activation='relu')(inputs)
mu = Dense(n_z, activation='linear')(h_q)
log_sigma = Dense(n_z, activation='linear')(h_q)
```




```python
def sample_z(args):
    mu, log_sigma = args
    eps = K.random_normal(shape=(m, n_z), mean=0., std=1.)
    return mu + K.exp(log_sigma / 2) * eps


# Sample z ~ Q(z|X)
z = Lambda(sample_z)([mu, log_sigma])
```

Decoder Net : P(X|z)




```python
# P(X|z) -- decoder
decoder_hidden = Dense(512, activation='relu')
decoder_out = Dense(784, activation='sigmoid')

h_p = decoder_hidden(z)
outputs = decoder_out(h_p)
```

Final Keras Models
•reconstruct inputs,
•encode inputs into latent variables,
•generate data from latent variable

```python
# Overall VAE model, for reconstruction and training
vae = Model(inputs, outputs)

# Encoder model, to encode input into latent variable
# We use the mean as the output as it is the center point, the representative of the gaussian
encoder = Model(inputs, mu)

# Generator model, generate new data given latent variable z
d_in = Input(shape=(n_z,))
d_h = decoder_hidden(d_in)
d_out = decoder_out(d_h)
decoder = Model(d_in, d_out)
```

Loss and Training

Then, we need to translate our loss into Keras code:

```
def vae_loss(y_true, y_pred):
    """ Calculate loss = reconstruction loss + KL loss for each data in minibatch """
    # E[log P(X|z)]
    recon = K.sum(K.binary_crossentropy(y_pred, y_true), axis=1)
    # D_KL(Q(z|X) || P(z|X)); calculate in closed form as both dist. are Gaussian
    kl = 0.5 * K.sum(K.exp(log_sigma) + K.square(mu) - 1. - log_sigma, axis=1)

    return recon + kl
```

and then train it:

```
vae.compile(optimizer='adam', loss=vae_loss)
vae.fit(X_train, X_train, batch_size=m, nb_epoch=n_epoch)
```

And that's it, the implementation of VAE in Keras!