Variational Auto Encoder (VAE)

Taewook Nam
MLAI, KAIST

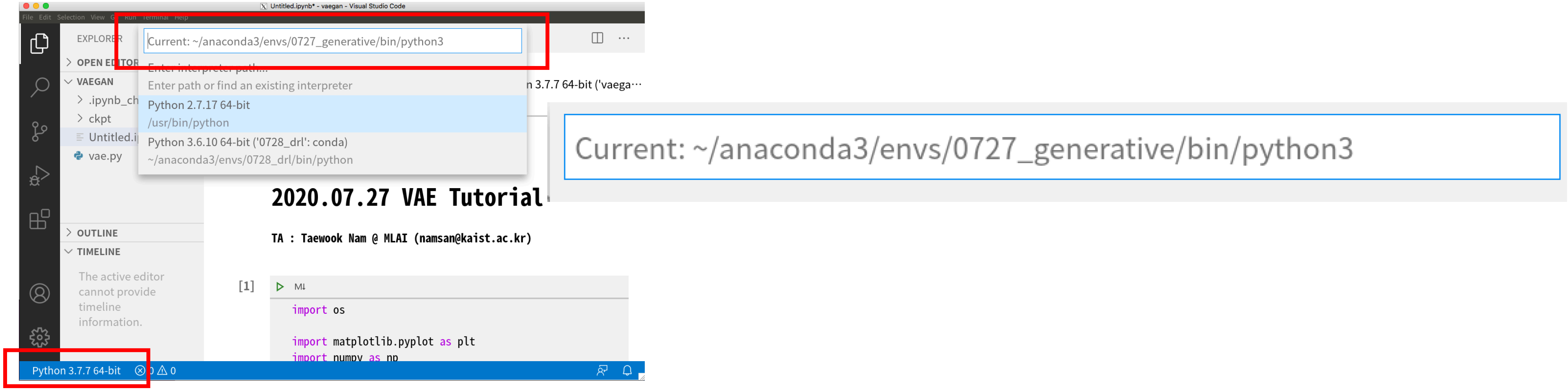
Environment

- Jupyter notebook in vscode

```
$ code
```

- conda path

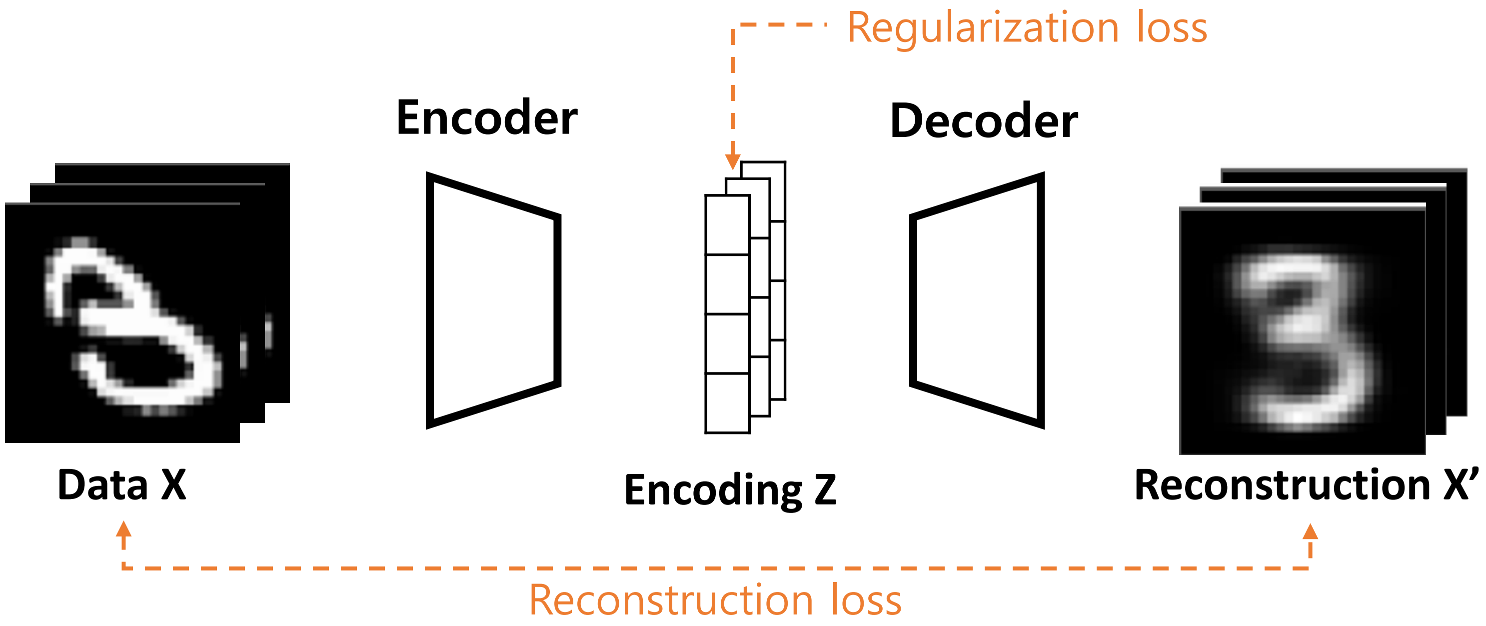
```
~/anaconda3/envs/0727_generative/bin/python
```



VAE Review

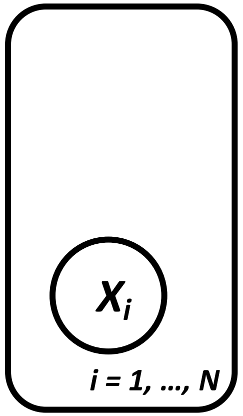
Intuitive View of VAE

- What we have?
  - Data **X** = {$\boldsymbol{x}_1$, … , $\boldsymbol{x}_N$}
- What we want?
  - Latent variable **Z** = {$\boldsymbol{z}_1$, …, $\boldsymbol{z}_N$}, ***automatically***
- Why?
  - **Z** is useful! Generator (***z*** -> ***x***) is useful!
- ***How?***



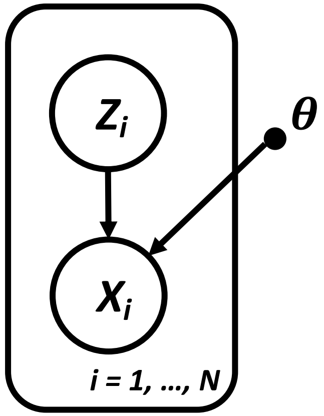
Mathematical View of VAE

- What we have?
  - Data $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$



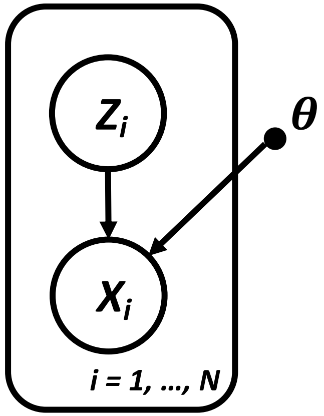
Mathematical View of VAE

- What we have?
  - Data $\mathbf{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$
- What we want?
  - Latent variable $\mathbf{Z} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_N\}$, ***automatically***



Mathematical View of VAE

- What we have?
  - Data **X** = {$\boldsymbol{x}_1$, … , $\boldsymbol{x}_N$}
- What we want?
  - Latent variable **Z** = {$\boldsymbol{z}_1$, …, $\boldsymbol{z}_N$}, ***automatically***
- ***How?***

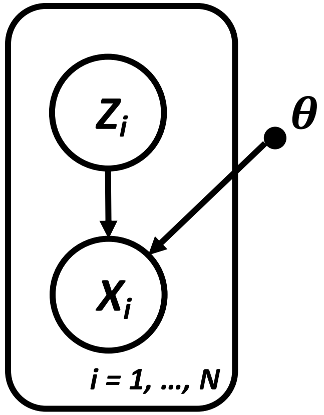


1. Directly maximizing evidence $p(\mathbf{X}|\theta)$

$$\max_{\theta} \prod_{i=1}^{N} p(\boldsymbol{x}_i|\theta) = \max_{\theta} \prod_{i=1}^{N} \int p(\boldsymbol{x}_i|\boldsymbol{z},\theta)p(\boldsymbol{z})\, d\boldsymbol{z}$$

Mathematical View of VAE

- What we have?
  - Data **X** = {$\boldsymbol{x}_1$, … , $\boldsymbol{x}_N$}
- What we want?
  - Latent variable **Z** = {$\boldsymbol{z}_1$, …, $\boldsymbol{z}_N$}, ***automatically***
- ***How?***



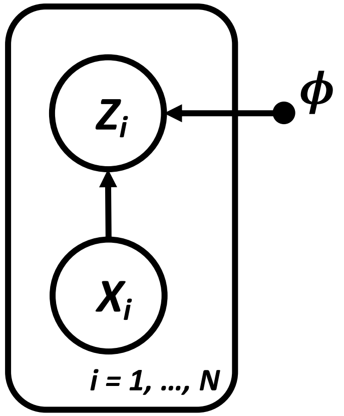
2. If we know true posterior?

$\boldsymbol{z_i} \sim p(\boldsymbol{z}|\boldsymbol{x},\theta)$ is just possible.
But this is ***intractable*** to compute from $p(\boldsymbol{x}|\boldsymbol{z},\theta)$

$$p(\boldsymbol{z}|\boldsymbol{x},\theta) = \frac{p(\boldsymbol{x}|\boldsymbol{z},\theta)p(\boldsymbol{z}|\theta)}{p(\boldsymbol{x}|\theta)} = \frac{p(\boldsymbol{x}|\boldsymbol{z},\theta)p(\boldsymbol{z}|\theta)}{\int p(\boldsymbol{x}|\boldsymbol{z},\theta)p(\boldsymbol{z}|\theta)\, d\boldsymbol{z}}$$

Mathematical View of VAE

- What we have?
  - Data **X** = {$\boldsymbol{x}_1$, … , $\boldsymbol{x}_N$}
- What we want?
  - Latent variable **Z** = {$\boldsymbol{z}_1$, …, $\boldsymbol{z}_N$}, ***automatically***
- ***How?***

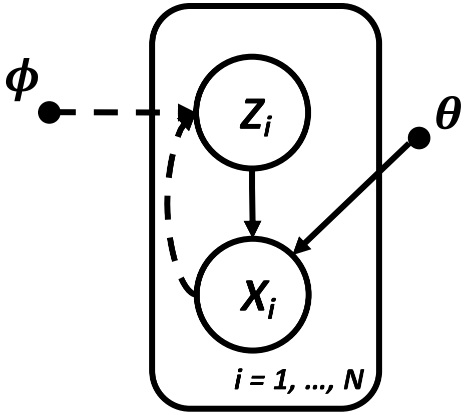


3. Model posterior?

$$\mathrm{p}(\boldsymbol{x}|\phi) = \int p(\boldsymbol{x}|\boldsymbol{z},\phi)p(\boldsymbol{z}|\phi)\, d\boldsymbol{z}$$

Mathematical View of VAE

- What we have?
  - Data **X** = {$\boldsymbol{x}_1$, … , $\boldsymbol{x}_N$}
- What we want?
  - Latent variable **Z** = {$\boldsymbol{z}_1$, …, $\boldsymbol{z}_N$}, ***automatically***
- ***How?***



4. "Variational Inference" does both : maximize evidence & inference posterior

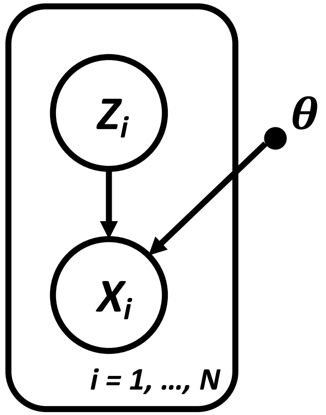
$$\log p(\boldsymbol{x}|\theta) = \log \int p(\boldsymbol{x}|\boldsymbol{z},\theta)p(\boldsymbol{z})\, d\boldsymbol{z}$$

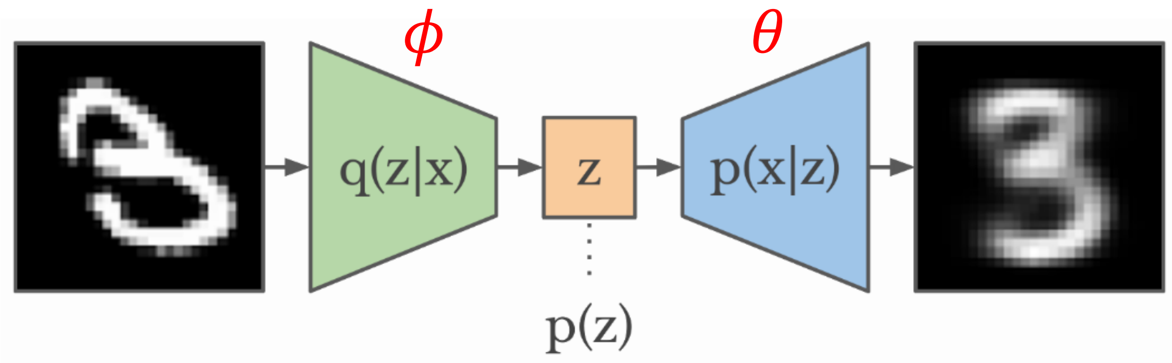
$$= \log \int q(\boldsymbol{z}|\boldsymbol{x},\phi)\ \frac{p(\boldsymbol{x}|\boldsymbol{z},\theta)p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x},\phi)}\, d\boldsymbol{z}$$

$$\geq \int q(\boldsymbol{z}|\boldsymbol{x},\phi)\log \frac{p(\boldsymbol{x}|\boldsymbol{z},\theta)p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x},\phi)}\, d\boldsymbol{z}$$

$$= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}|\boldsymbol{z},\theta)] - KL[q(\boldsymbol{z}|\boldsymbol{x},\phi)||p(\boldsymbol{z})]$$

VAE





$$\max_{\theta,\phi} \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}|\boldsymbol{z},\theta)] - KL[q(\boldsymbol{z}|\boldsymbol{x},\phi)||p(\boldsymbol{z})]$$

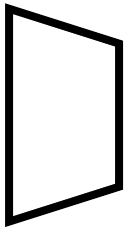
Today’s Goal!

Today's Goal

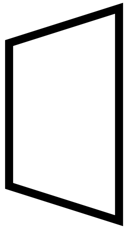
- Tensorflow implementation of VAE
- Visualization of learned encoding
- Visualization of learned decoding
- Visualization of learned latent space

```python
from layers import *

def encoder(x, zdim, name='encoder', reuse=None):
    x = dense(x, 500, activation=relu, name=name+'/dense1', reuse=reuse)
    x = dense(x, 500, activation=relu, name=name+'/dense2', reuse=reuse)
    mu = dense(x, zdim, name=name+'/mu', reuse=reuse)
    sigma = dense(x, zdim, activation=softplus, name=name+'/sigma',
            reuse=reuse)
    return mu, sigma

def decoder(x, name='decoder', reuse=None):
    x = dense(x, 500, activation=relu, name=name+'/dense1', reuse=reuse)
    x = dense(x, 500, activation=relu, name=name+'/dense2', reuse=reuse)
    x = dense(x, 784, activation=sigmoid, name=name+'/output',
            reuse=reuse)
    return x

def autoencoder(x, zdim, training, name='autoencoder', reuse=None):
    mu, sigma = encoder(x, zdim, reuse=reuse)
    z = Normal(mu, sigma).sample() if training else mu
    x_hat = decoder(z, reuse=reuse)

    log_likelihood = tf.reduce_sum(x*log(x_hat) + (1-x)*log(1-x_hat), 1)
    kl = 0.5 * tf.reduce_sum(mu**2 + sigma**2 - log(sigma**2) - 1, 1)
    elbo = tf.reduce_mean(log_likelihood - kl)

    net = {}
    net['elbo'] = elbo
    net['weights'] = tf.trainable_variables()
    return net
```

Encoder

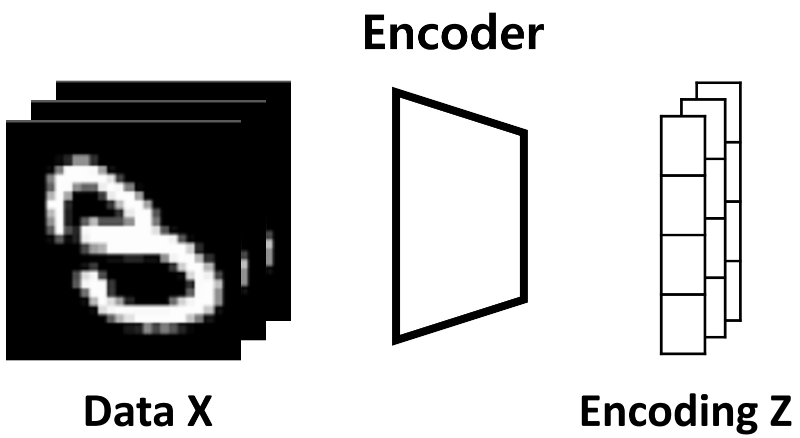


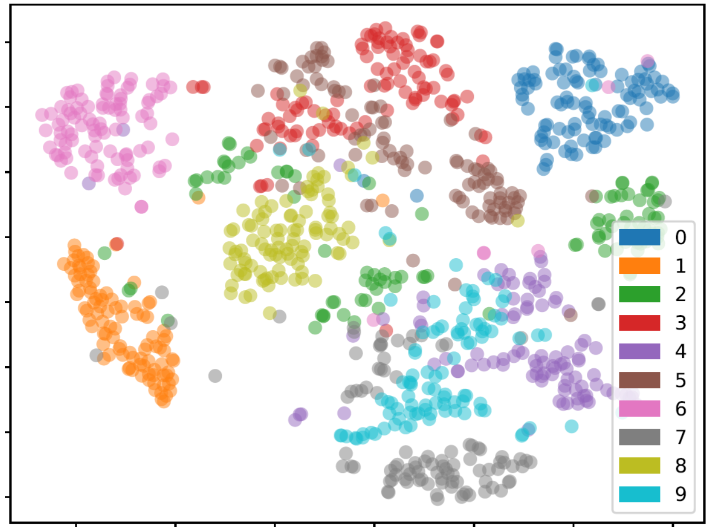
Decoder



Today's Goal

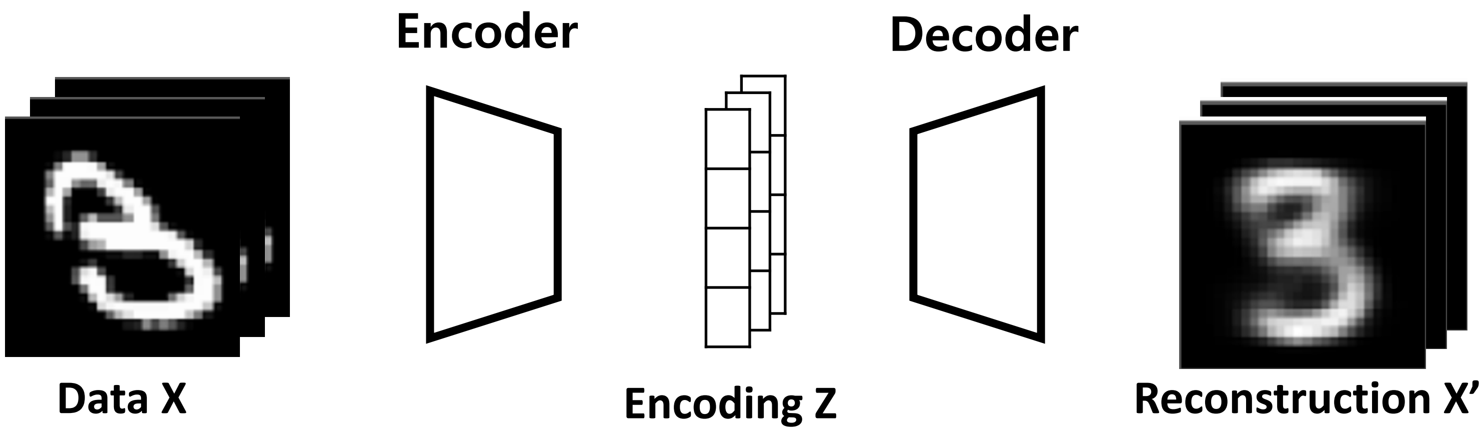
- Tensorflow implementation of VAE
- Visualization of learned encoding
- Visualization of learned decoding
- Visualization of learned latent space

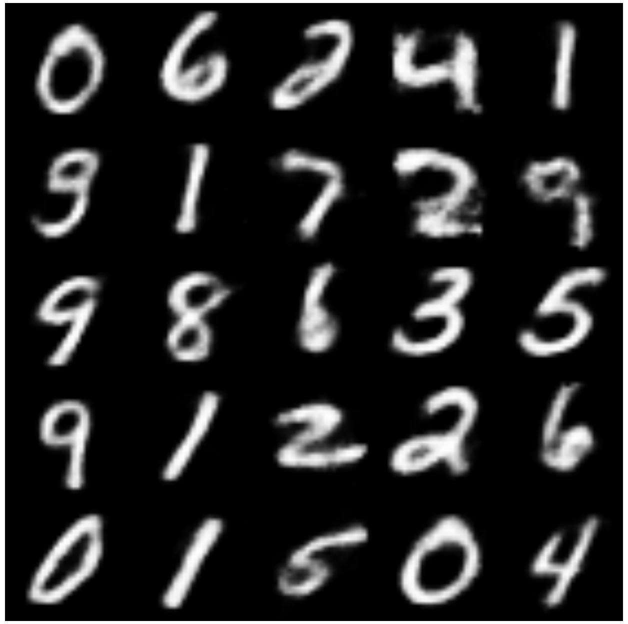




Today’s Goal

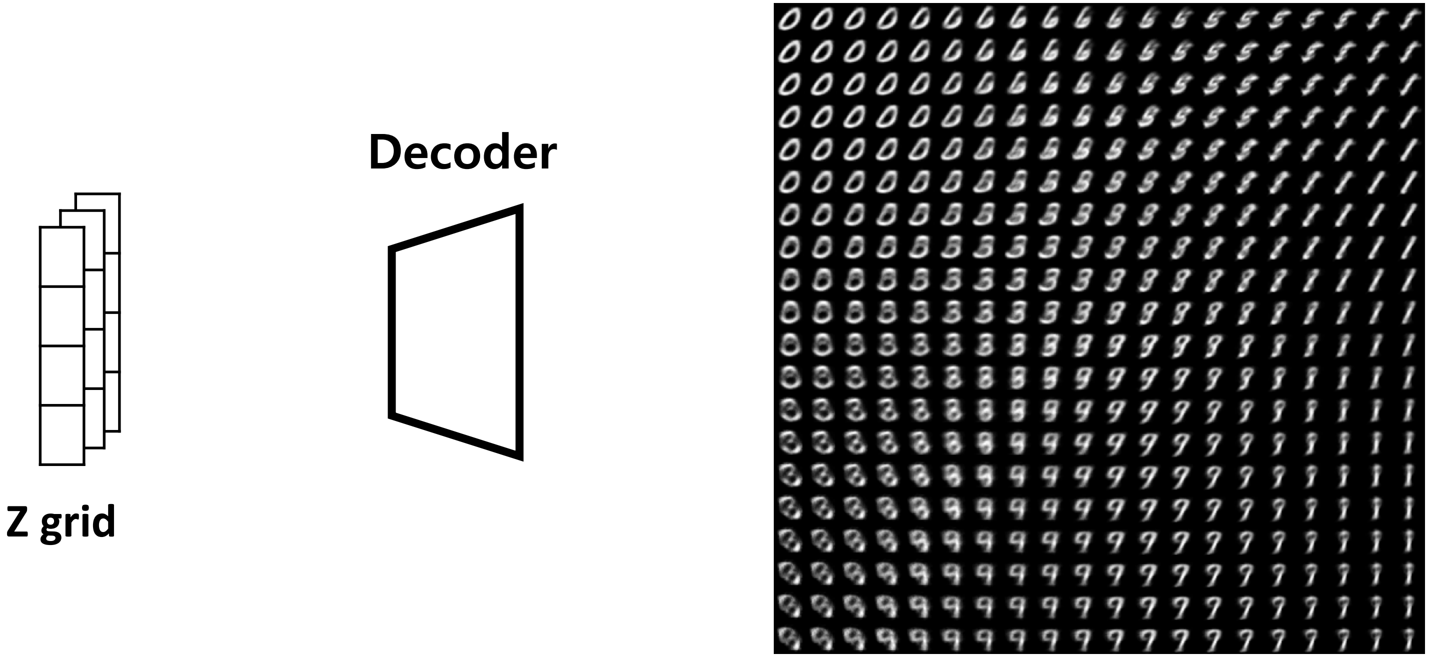
- Tensorflow implementation of VAE
- Visualization of learned encoding
- Visualization of learned decoding
- Visualization of learned latent space





Today's Goal

- Tensorflow implementation of VAE
- Visualization of learned encoding
- Visualization of learned decoding
- Visualization of learned latent space



Today's Goal!

Task 1 : Tensorflow implementation of VAE

Task 2 : Visualization of learned encoding

Task 3 : Visualization of learned decoding

Task 4 : Visualization of learned latent space

Task 1 : TF implementation

- Network

```
class VAE(Model):
  h_dim = 500

  def __init__(self, x_shape, z_dim):
    super().__init__()

    x_dim = np.prod(x_shape)
    self.z_dim = z_dim

    self.encoder = Sequential([
        Flatten(),
        Dense(self.h_dim, activation='relu'),
        Dense(self.h_dim, activation='relu'),
        Dense(2*z_dim, activation='relu')
    ])
    self.decoder = Sequential([
        Dense(self.h_dim, activation='relu'),
        Dense(self.h_dim, activation='relu'),
        Dense(x_dim, activation='sigmoid'),
        Reshape(x_shape)
    ])
```

```
def encode(self, x):
  z_param = self.encoder(x)
  z_mu, z_rho = z_param[:, :self.z_dim], z_param[:, self.z_dim:]
  return z_mu, tf.nn.softplus(z_rho)
```

```
def decode(self, z):
  return self.decoder(z)
```

Task 1 : TF implementation

$$\max_{\theta,\phi} \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}|\boldsymbol{z},\theta)] - KL[q(\boldsymbol{z}|\boldsymbol{x},\phi)||p(\boldsymbol{z})]$$

- MC Sampling

$$\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}|\boldsymbol{z},\theta)] \approx \frac{1}{S}\sum_{s=1}^{S} \log p\left(\boldsymbol{x}\middle|\boldsymbol{z}^{(s)},\theta\right)$$

$$where\ \boldsymbol{z}^{(s)} \sim q(\boldsymbol{z}|\boldsymbol{x},\phi)$$

```
z_mu, z_sigma = vae.encode(x)
z_dist = tfp.distributions.Normal(z_mu, z_sigma)
z = z_dist.sample()
x_reconst = vae.decode(z)
```

- Reparameterization

$$p\left(\boldsymbol{x}\middle|\boldsymbol{z}^{(s)},\theta\right), \qquad \boldsymbol{z}^{(s)} \sim q(\boldsymbol{z}|\boldsymbol{x},\phi)$$

$$\rightarrow \quad p(\boldsymbol{x}|\boldsymbol{\mu}_{\boldsymbol{z}} + \boldsymbol{\epsilon} \otimes \boldsymbol{\sigma}_z), \qquad \boldsymbol{\epsilon} \sim \boldsymbol{N}(\boldsymbol{0},\boldsymbol{I})$$

Task 1 : TF implementation

$$\max_{\theta,\phi} \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}|\boldsymbol{z},\theta)] - KL[q(\boldsymbol{z}|\boldsymbol{x},\phi)||p(\boldsymbol{z})]$$

- Likelihood

$$\log p(\boldsymbol{x}|\mathbf{z},\theta) = \log \prod_{d=1}^{D_x} Ber(x_d|x'_d)$$

$$= \sum_{d=1}^{D} \log(x'_d)^{x_d}(1-x'_d)^{x'_d}$$

$$= \sum_{d=1}^{D} x_d \log x'_d + (1-x_d)\log(1-x'_d)$$

- KL divergence

$$KL[q(\mathbf{z}|\boldsymbol{x},\phi)||p(\mathbf{z})] = \sum_{d=1}^{D_z} KL[N(z_d|\mu_d,\sigma_d^2)||N(z_d|0,1)]$$

$$= \sum_{d=1}^{D_z} \mu_d^2 + \sigma_d^2 - \log\sigma_d^2 - 1$$

```python
def compute_elbo(x, x_reconst, z_mu, z_sigma):
    log_likelihood = tf.reduce_sum(
      x*tf.math.log(x_reconst+1e-6) + (1-x)*tf.math.log(1-x_reconst+1e-6),
      axis=(1,2)
    )
    kl = 0.5 * tf.reduce_sum(
      z_mu**2 + z_sigma**2 - tf.math.log(z_sigma**2) - 1,
      axis=1
    )
    elbo = tf.reduce_mean(log_likelihood - kl)
    return elbo
```

Task 1 : TF implementation

- Training

```
for epoch_i in trange(100):
  for x in train_ds:
    with tf.GradientTape() as tape:
      z_mu, z_sigma = vae.encode(x)
      z_dist = tfp.distributions.Normal(z_mu, z_sigma)
      z = z_dist.sample()
      x_reconst = vae.decode(z)

      elbo = compute_elbo(x, x_reconst, z_mu, z_sigma)
      loss = -elbo
    gradients = tape.gradient(loss, vae.trainable_variables)
    optimizer.apply_gradients(zip(gradients, vae.trainable_variables))

  if (epoch_i+1) % 5 == 0:
    vae.save_weights(f'ckpt/{epoch_i + 1}')
```

- Load

```
ckpt_i = 100
vae.load_weights(f'ckpt/{ckpt_i}')
```

Today’s Goal!

Task 1 : Tensorflow implementation of VAE

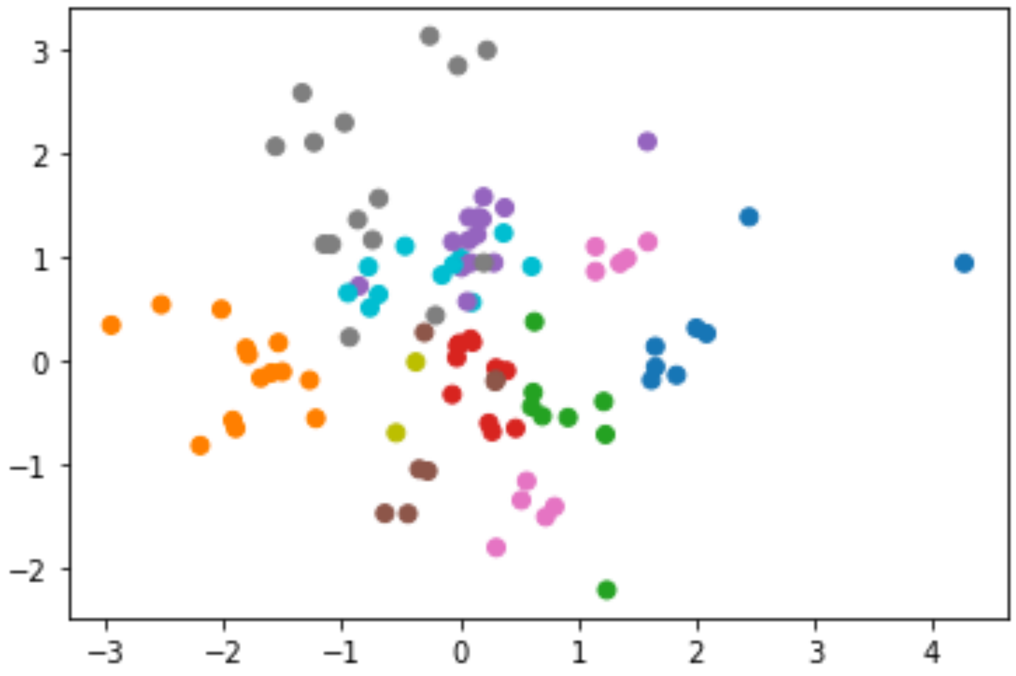
Task 2 : Visualization of learned encoding

Task 3 : Visualization of learned decoding

Task 4 : Visualization of learned latent space

Task 2 : Learned Encoding

```
n_points = 100
z_mu, _ = vae.encode(x_test[:n_points])

fig, ax = plt.subplots()
scatter = plt.scatter(
    z_mu[:, 0], z_mu[:, 1],
    c=[f'C{y_test[i]}' for i in range(n_points)],
    label=[y_test[i] for i in range(n_points)]
)
```



Today's Goal!

Task 1 : Tensorflow implementation of VAE

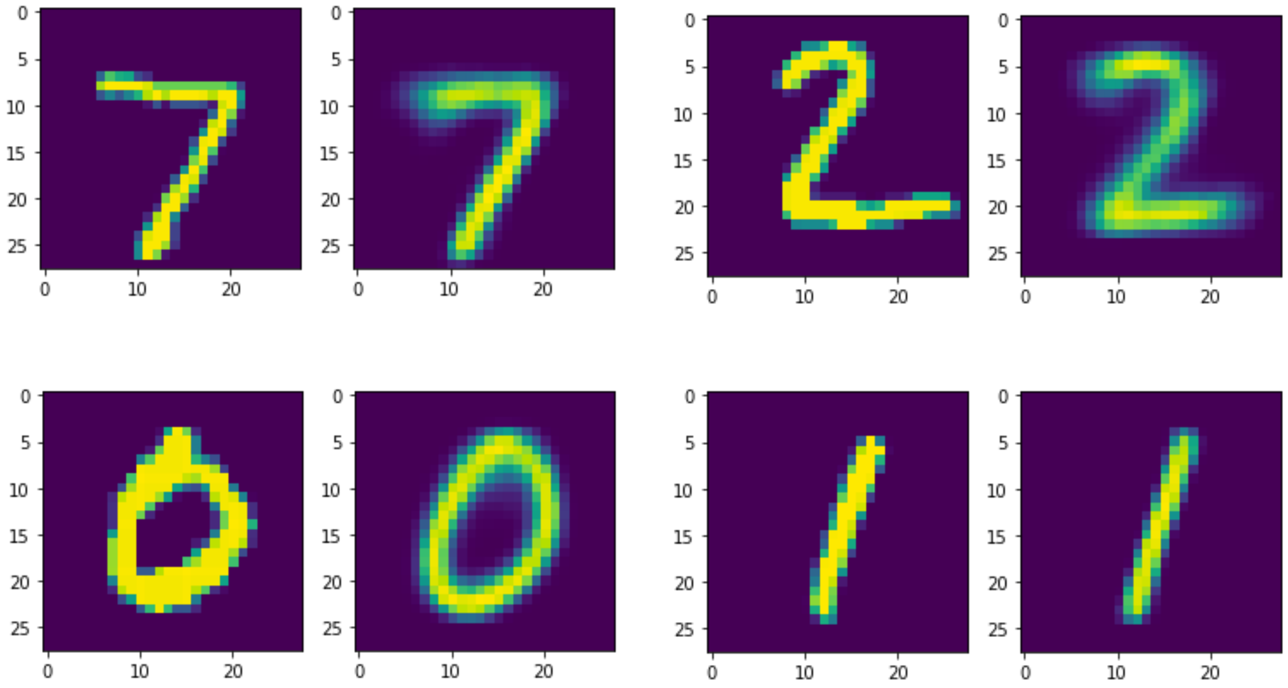
Task 2 : Visualization of learned encoding

Task 3 : Visualization of learned decoding

Task 4 : Visualization of learned latent space

Task 3 : Learned Decoding

```
idx = 10
z_mu, _ = vae.encode(x_test)
x_reconst = vae.decode(z_mu)

fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(x_test[idx])
ax2.imshow(x_reconst[idx])
```



Today's Goal!

Task 1 : Tensorflow implementation of VAE

Task 2 : Visualization of learned encoding

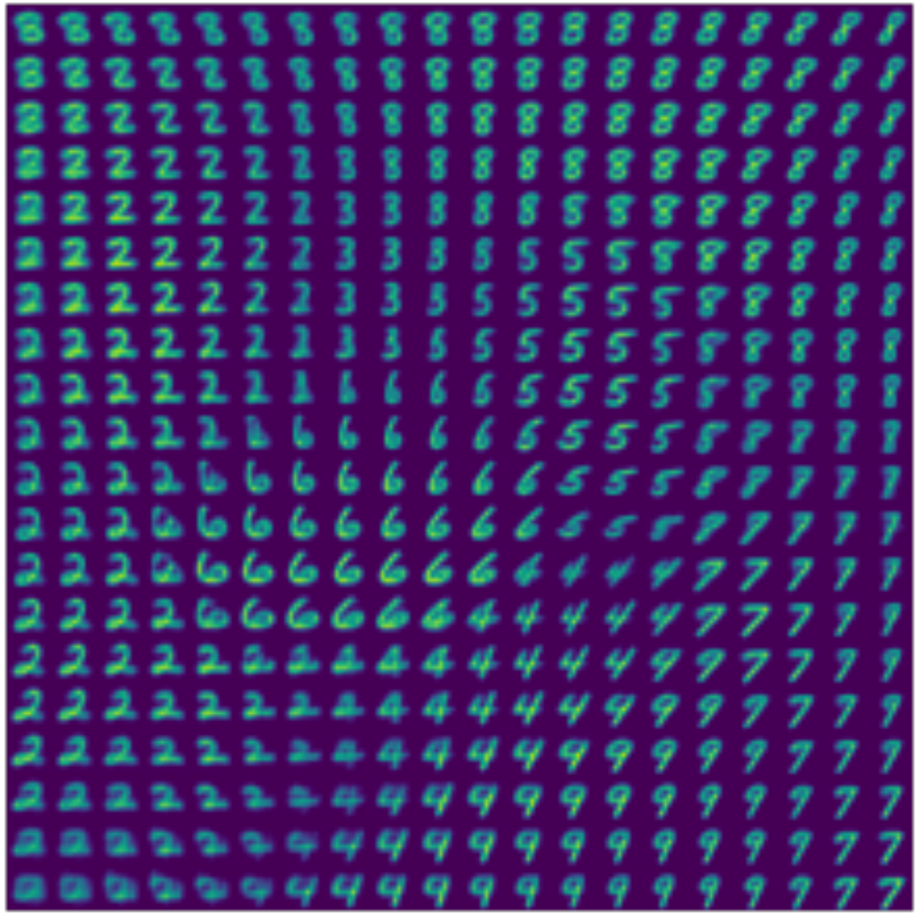
Task 3 : Visualization of learned decoding

Task 4 : Visualization of learned latent space

Task 4 : Learned Latent Space

```
z_range = np.arange(-1, 1, 0.1)
z_grid = np.stack(np.meshgrid(z_range, z_range), axis=-1)

z_grid_flat = z_grid.reshape(400, 2)
x_reconst_flat = vae.decode(tf.convert_to_tensor(z_grid_flat))
x_reconst = x_reconst_flat.numpy().reshape(20, 20, 28, 28)
```

```
x_merged = x_reconst.swapaxes(1, 2).reshape(20*28, 20*28)
plt.imshow(x_merged)
plt.axis('off')
```



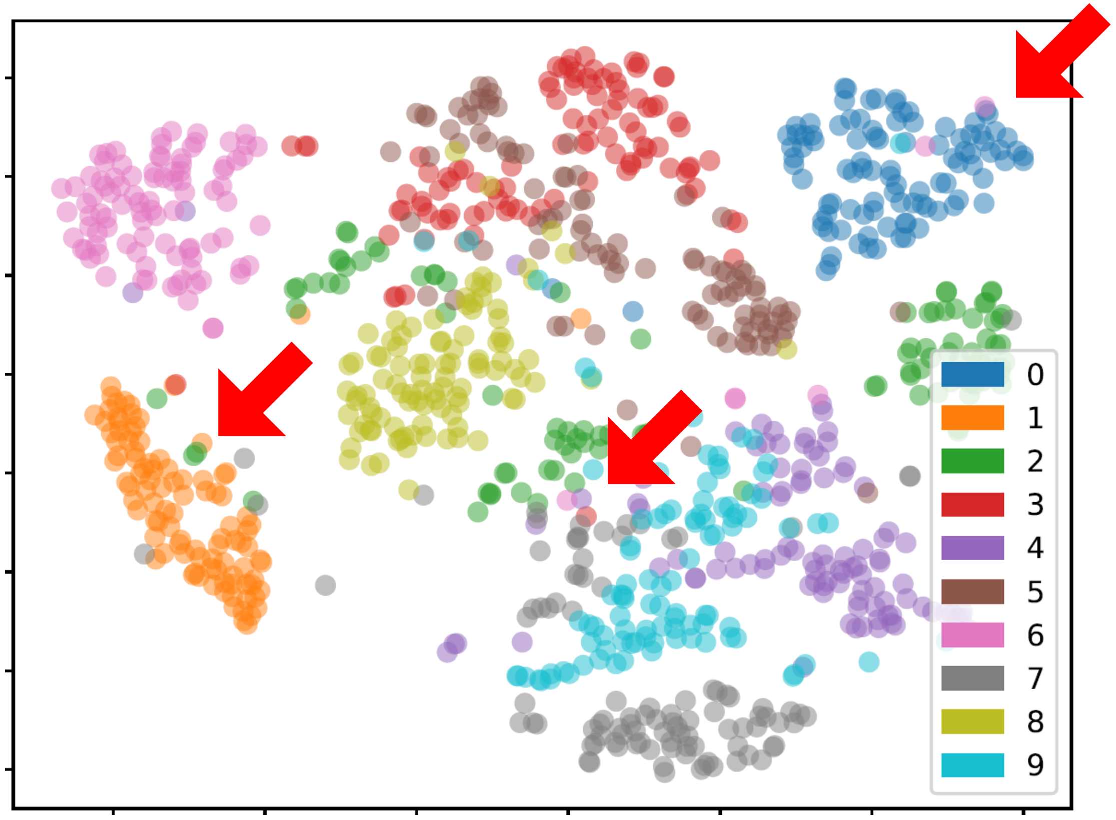
Task 4 : Learned Latent Space

- Plot

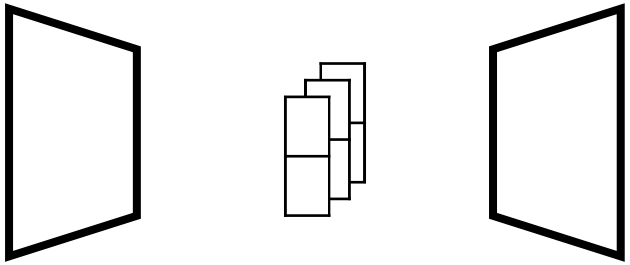
Research Questions

Research Questions

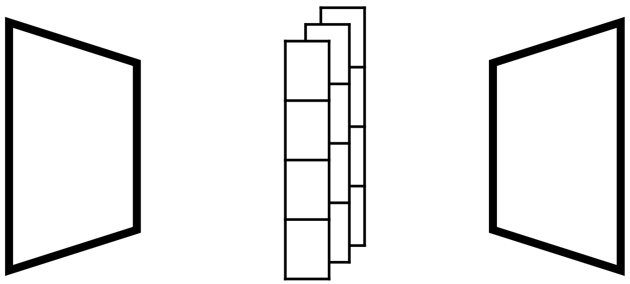
Who are they?



Difference?



Low-dimensional z



High-dimensional z

Research Questions

What happen if we omit this?

And your own ☺!!

$$\log p(X;\theta)$$
$$\geq \sum_{x \in X} E_{q(z|x;\phi)}[\log p(x|z;\theta)]$$
$$\boxed{- KL[q(z|x;\phi) \| p(z)]}$$

감사합니다