Variational AutoEncoder (VAE)
&
Its Application to Missing Data Analysis

Contents

1

Gaussian VAE

Variational AutoEncoder

$$\log p(x) = \log p(x) \int q_\phi(z|x)dz = \int q_\phi(z|x) \log p(x) dz$$

(로그우도를 최대화, pdf 적분하면 1)

$$= \int q_\phi(z|x) \log \frac{p(x|z)p(z)}{p(z|x)} dz$$

(베이즈 정리)

$$= \int q_\phi(z|x) \log \left[\frac{p(x|z)p(z)}{p(z|x)} \cdot \frac{q_\phi(z|x)}{q_\phi(z|x)} \right] dz$$

(분모, 분자에 같은 식 곱하기)

$$= \int q_\phi(z|x) \log p(x|z) dz - \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z)} dz + \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z|x)} dz$$

(로그 성질 이용)

$$= \underbrace{\mathbb{E}_{q_\phi(z|x)} \log p(x|z) - D_{KL}\left(q_\phi(z|x) || p(z)\right)}_{\text{Variational Lower Bound (ELBO)}} + \underbrace{D_{KL}\left(q_\phi(z|x) || p(z|x)\right)}_{\text{Intractable BUT} \geq 0}$$

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

Variational AutoEncoder

$$\log p(x) \geq \underbrace{\mathbb{E}_{q_\phi(z|x)} \log p(x \mid z) - D_{KL}(q_\phi(z \mid x) \parallel p(z))}_{\text{Variational Lower Bound (ELBO)}}$$

$$\begin{aligned}\max \ \text{ELBO} &= \max \left(\mathbb{E}_{q_\phi(z|x)} \log p(x \mid z) - D_{KL}(q_\phi(z \mid x) \parallel p(z))\right) \\ &= \min \underbrace{\left(-\mathbb{E}_{q_\phi(z|x)} \log p(x \mid z)\right)}_{\text{Reconstruction Error}} + \min \underbrace{\left(D_{KL}(q_\phi(z \mid x) \parallel p(z))\right)}_{\text{Regularization}}\end{aligned}$$

차원을 줄였다가 복원한 데이터가
원래 데이터와 비슷해지도록

인코더가 표현하는 근사분포가
사후분포(표준정규분포)에 가까워지도록

Variational AutoEncoder

$$
\begin{aligned}
\mathbb{E}_{q_\phi(z|x_i)}\left[\log p_\theta(x_i|z)\right] &= \int \log\left(p_\theta(x_i|z)\right) q_\phi(z|x_i)dz \\
&\approx \frac{1}{L}\sum_{z^{i,l}} \log\left(p_\theta(x_i|z^{i,l})\right) && \text{(Monte Carlo Integral)} \\
&\approx \log\left(p_\theta(x_i|z^i)\right) && \text{(Set L = 1)} \\
&= \log \prod_{j=1}^{D} p_\theta(x_{i,j}|z^i) \\
&= \sum_{j=1}^{D} \log p_\theta(x_{i,j}|z^i) \\
&= \sum_{j=1}^{D} \log\left(p_{i,j}^{x_{i,j}}(1-p_{i,j})^{1-x_{i,j}}\right) && \text{(Bernoulli Assumption)} \\
&= \sum_{j=1}^{D} \left(x_{i,j}\log p_{i,j} + (1-x_{i,j})\log(1-p_{i,j})\right)
\end{aligned}
$$

Variational AutoEncoder

$$D_{KL}(q_\phi(z|x_i)||p(z)) = \frac{1}{2}\left(\mathrm{tr}(\sigma_i^2 I) + \mu_i^T\mu_i - J + \ln\frac{1}{\prod_{j=1}^{J}\sigma_{i,j}^2}\right)$$

$$= \frac{1}{2}\left(\sum_{j=1}^{J}\sigma_{i,j}^2 + \sum_{j=1}^{J}\mu_{i,j}^2 - J - \sum_{j=1}^{J}\ln(\sigma_{i,j}^2)\right)$$

$$= \frac{1}{2}\sum_{j=1}^{J}\left(\mu_{i,j}^2 + \sigma_{i,j}^2 - \ln(\sigma_{i,j}^2) - 1\right)$$

KLD for multivariate normal distributions:

$$D_{KL}(\mathcal{N}_0||\mathcal{N}_1) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1-\mu_0)^T\Sigma_1^{-1}(\mu_1-\mu_0) - k + \ln\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right)\right)$$

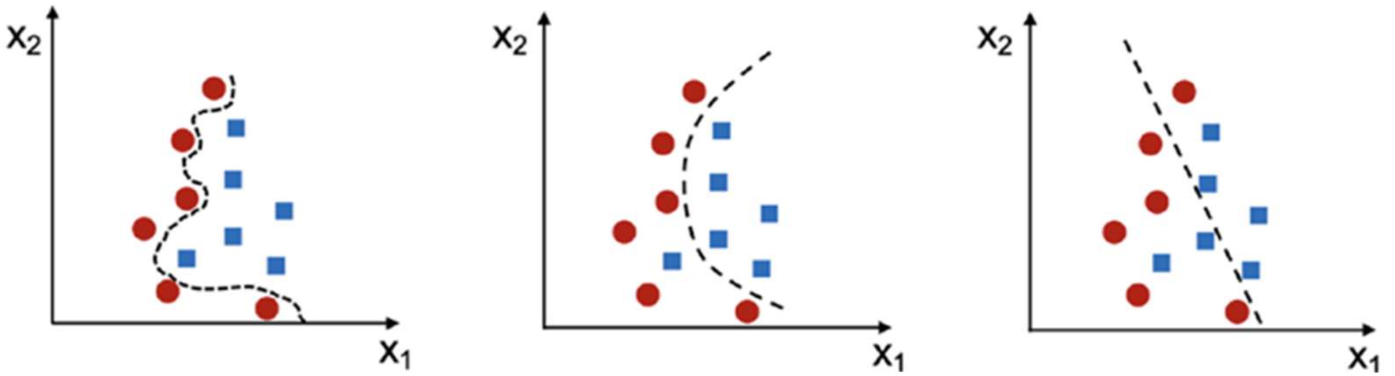
2

t-prior VAE

Mitigating the Overregularization Issue in VAEs

overregularization of VAE

- Overregularization



Caused by the strong influence of the KL divergence term between the gaussian prior and the gaussian encoder's distribution

- Recap variational inference:

$$log(p(\mathbf{x})) = E_{q_\phi(\mathbf{z}|\mathbf{x})}[log(p_\theta(\mathbf{x}|\mathbf{z}))] - KL(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) + KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$$

Define loss function as:

$$Loss = -E_{q_\phi(\mathbf{z}|\mathbf{x})}[log(p_\theta(\mathbf{x}|\mathbf{z}))] + KL(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) := Loss_{RE} + Loss_{KL}$$

t-distribution

- Note that t-distribution belongs to the location-scale family: $Y = \mu + \sigma X,\ f_Y(y) = \frac{1}{\sigma} f_X(\frac{x-\mu}{\sigma})$

$$pdf_{t(\mu,\sigma^2,\nu)}(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sigma\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}(1+\frac{1}{\nu}(\frac{x-\mu}{\sigma})^2)^{-\frac{\nu+1}{2}} \quad \textbf{VS} \quad pdf_{N(\mu,\sigma^2)}(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$$

- The tail decay of the t-distribution is significantly slower compared to the Gaussian distribution.

- KL divergence between a t-distribution is lower then that of a gaussian distribution

$$KL(q(x)||p(x)) = \int q(x) log \frac{q(x)}{p(x)}$$

t-distribution

- Here's why: $$KL(q(x)||p(x)) = \int q(x) log \frac{q(x)}{p(x)}$$

In distributions with lighter tails, such as the Gaussian, tail differences significantly influence the KL divergence. The rapid exponential decay of these tails magnifies the relative values of the probability density function

Therefore, the KL divergence between the t-distribution is relatively smaller compared to the Gaussian distribution

- Based on the above reasoning, we expect that a VAE with t-distributed latent space would improve generalization performance, particularly in identifying outlier patterns

derivation for t-prior VAE

- Gaussian VAE:

$$p_{\mathbf{Z}}(\mathbf{z}) \sim N_m(\mathbf{0},\mathbf{I}),\ \ q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \sim N_m(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}),\boldsymbol{\Sigma}_{\boldsymbol{\phi}}(\mathbf{x})),\ \ p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \sim Multivariate\ bernoulli$$

- To mitigate the aforementioned problem, we propose a new model, the t-prior VAE, defined as follows:

$$p_{\mathbf{Z}}(\mathbf{z}) \sim t_m(\mathbf{0},\mathbf{I},\nu),\ \ q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \sim t_m(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}),\boldsymbol{\Sigma}_{\boldsymbol{\phi}}(\mathbf{x}),\nu),\ \ p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \sim Multivariate\ bernoulli$$

- Loss function derivation

Given that the decoder assumption remains unchanged, $Loss_{RE} \cong -\frac{1}{L}\sum_{i=1}^{L} log(p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}))$ by Monte Carlo approximation.

we now derive the KL divergence between the t-distributed encoder and the prior.

$$Loss_{KL} = KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) log\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} = E_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[log(\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})})]$$

$$\text{for univariate case: } p(z) \sim t(0,1,\nu),\ \ q(z|\mathbf{x}) \sim t(\mu_q,\sigma_q,\nu)$$

$$E_{q(z|x)}[log(\frac{q(z|x)}{p(z)})] = -log(\sigma_q) + E_{q(z|x)}[-\frac{\nu+1}{2}log(1+\frac{1}{\nu}(\frac{z-\mu_q}{\sigma_q})^2) + \frac{\nu+1}{2}log(1+\frac{1}{\nu}z^2)]$$

derivation for t-prior VAE

Applying the Monte Carlo approximation,

$$E_{q(z|x)}[log(\frac{q(z|x)}{p(z)})] \quad \cong -log(\sigma_q) + \frac{1}{L}\sum_{i=1}^{L}[-\frac{\nu+1}{2}log(1+\frac{1}{\nu}(\frac{z^{(i)}-\mu_q}{\sigma_q})^2) + \frac{\nu+1}{2}log(1+\frac{1}{\nu}z^{(i)^2})]$$

Thus, the loss function for the univariate distribution becomes,

$$Loss = Loss_{RE} + Loss_{KL} \cong -\frac{1}{L}\sum_{i=1}^{L}log(p(x^{(i)}|z^{(i)})) \; -log(\sigma_q) + \frac{1}{L}\sum_{i=1}^{L}[-\frac{\nu+1}{2}log(1+\frac{1}{\nu}(\frac{z^{(i)}-\mu_q}{\sigma_q})^2) + \frac{\nu+1}{2}log(1+\frac{1}{\nu}z^{(i)^2})]$$

Setting $L = 1$,

$$Loss = \frac{1}{N}\sum_{i=1}^{N}[\sum_{j=1}^{d}\{x_{i,j}log(p_{i,j}) + (1-x_{i,j})log(1-p_{i,j})\} + \sum_{j=1}^{m}\{-log(\sigma_{i,j}) - \frac{\nu+1}{2}log(1+\frac{1}{\nu}(\frac{z_{i,j}-\mu_{i,j}}{\sigma_{i,j}})^2) + \frac{\nu+1}{2}log(1+\frac{1}{\nu}z_{i,j}^2)]\}$$

derivation for t-prior VAE

- Backpropagation derivation

$$Loss = \frac{1}{N}\sum_{i=1}^{N}[\sum_{j=1}^{d}\{x_{i,j}log(p_{i,j}) + (1-x_{i,j})log(1-p_{i,j})\} + \sum_{j=1}^{m}\{-log(\sigma_{i,j}) - \frac{\nu+1}{2}log(1+\frac{1}{\nu}(\frac{z_{i,j}-\mu_{i,j}}{\sigma_{i,j}})^2) + \frac{\nu+1}{2}log(1+\frac{1}{\nu}z_{i,j}^2)]\}$$

Decoder : only by L_{RE}

note that $L_{RE} = -\log P(x|z)$ is NLL of multivariate bernoulli, which is equivalent to cross entropy.

$$\frac{\partial L}{\partial z^{[4]}} = \frac{\partial L_{RE}}{\partial z^{[4]}} = (\hat{x} - x)/n$$

decoder backprop (same as original NN).

encoder : $\frac{\partial L}{\partial \mu_q} = \frac{\partial L_{RE}}{\partial \mu_q} + \frac{\partial L_{KL}}{\partial \mu_q}$, $\frac{\partial L}{\partial \log\sigma_q^2} = \frac{\partial L}{\partial \sigma_q}\cdot\frac{\partial \sigma_q}{\partial \log\sigma_q^2} = \left(\frac{\partial L_{RE}}{\partial \sigma_q} + \frac{\partial L_{KL}}{\partial \sigma_q}\right)\cdot\frac{\partial \sigma_q}{\partial \log\sigma_q^2} = \left(\frac{\partial L_{RE}}{\partial \sigma_q} + \frac{\partial L_{KL}}{\partial \sigma_q}\right)\cdot\frac{\sigma_q}{2}$

note) $z = \mu_q + \sigma_q \cdot \epsilon$, where $\epsilon \sim t_m(0, I, \nu)$

$$L_{KL} = -\log\sigma_q - \frac{\nu+1}{2}\log\left(1+\frac{1}{\nu}\left(\frac{z-\mu_q}{\sigma_q}\right)^2\right) + \frac{\nu+1}{2}\log\left(1+\frac{1}{\nu}z^2\right)$$

$\frac{\partial L_{KL}}{\partial \mu_q} = \frac{\partial L_{KL}}{\partial z}\cdot\frac{\partial z}{\partial \mu_q} = \frac{\nu+1}{2}\cdot\frac{\frac{2}{\nu}\cdot z}{1+\frac{1}{\nu}z^2} = (\nu+1)\cdot\frac{z}{\nu+z^2}$, $\frac{\partial L_{KL}}{\partial \sigma_q} = -\frac{1}{\sigma_q} + \frac{\nu+1}{2}\cdot\frac{\frac{2}{\nu}\cdot z}{1+\frac{1}{\nu}z^2}\cdot\epsilon = -\frac{1}{\sigma_q} + (\nu+1)\cdot\frac{z}{\nu+z^2}\cdot\epsilon$

$$\frac{\partial L}{\partial z^{[2]}} = \left[\frac{\partial L}{\partial \mu_q}, \frac{\partial L}{\partial \log\sigma_q^2}\right]/n$$

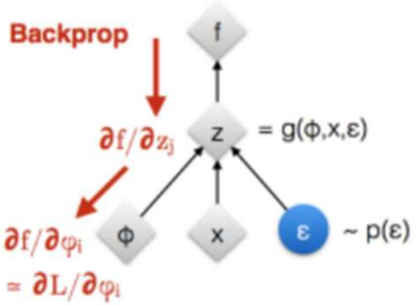
encoder backprop (same as original NN).

done!

implementation issues

- Reparameterization Trick for a t-Distributed Latent Space

- Similar to the Gaussian VAE, this model leverages the location-scale family property



- Here, ν is treated as a hyperparameter. If ν is too large, the model behaves like a Gaussian VAE, potentially leading to overregularization. Conversely, if ν is too small, the model may suffer from poor learning due to sampling from a heavy-tailed distribution.

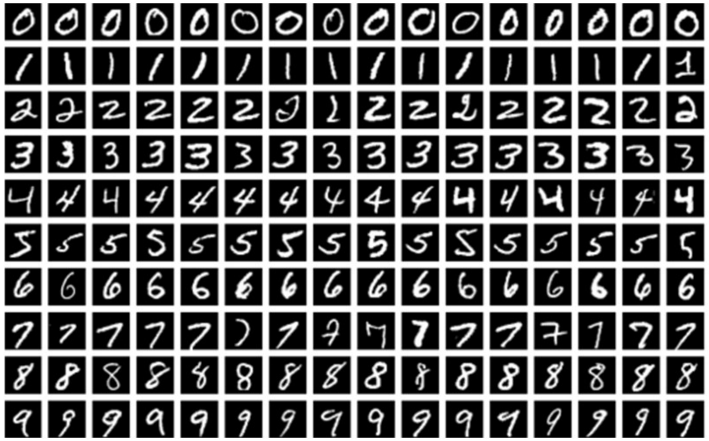
3
Code implementation

Dataset

```
import numpy as np
from sklearn.datasets import fetch_openml

X, y = fetch_openml('mnist_784', version=1, return_X_y=True)#, parser='auto')
X = X.values

X = X / 255. #normalization to [0,1]

y = X

print(X.shape)
print(y.shape)
```

```
(70000, 784)
(70000, 784)
```

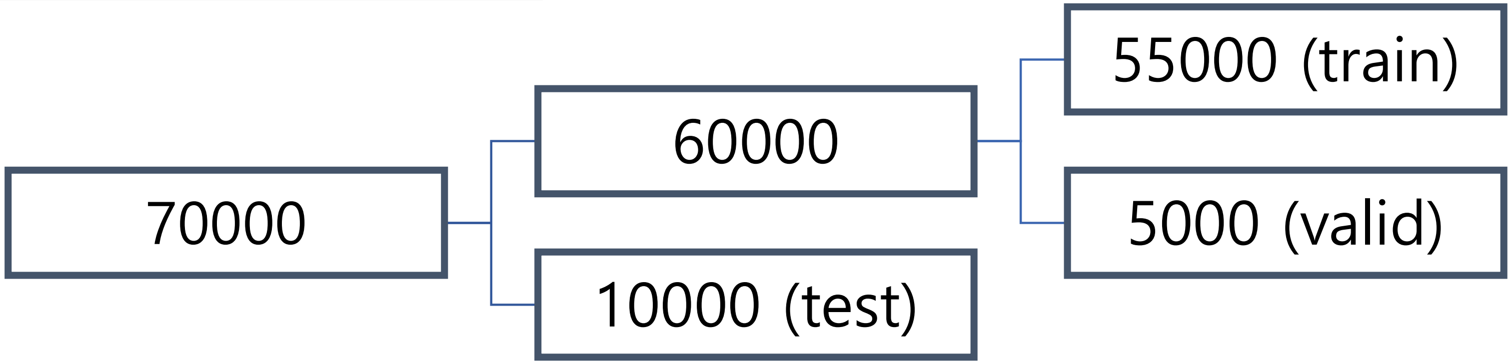
<MNIST Dataset>



- 총 70000개의 이미지 데이터
- 하나의 이미지는 28 x 28 = 784개 픽셀로 구성
  = 784차원 벡터
- 각 픽셀은 0 ~ 255의 정수 → 0~1 사이로 정규화

Dataset

```python
from sklearn.model_selection import train_test_split


X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=10000, random_state=123)

X_train, X_valid, y_train, y_valid = train_test_split(
    X_temp, y_temp, test_size=5000, random_state=123)

del X_temp, y_temp, X, y
```



VAE (Gaussian)

```python
def encoder(self, x):
    z_e1 = x @ self.W_e1.T + self.b_e1
    a_e1 = relu(z_e1) # activation

    z_e2 = a_e1 @ self.W_e2.T + self.b_e2
    a_e2 = relu(z_e2) # activation

    mu_logvar = a_e2 @ self.W_e3.T + self.b_e3 # linear
    mu, logvar = np.split(mu_logvar, 2, axis=1)

    return z_e1, a_e1, z_e2, a_e2, mu, logvar
```

앞서 살펴보았듯, 인코더는 입력 데이터가 들어오면
정규분포에 대한 평균과 분산(근데 이제 로그를 곁들인)을 반환

cf)

```python
def relu(z): #relu
    return np.maximum(0, z)
```

```python
def reparameterize(self, mu, logvar):
    std = np.exp(0.5 * logvar)
    eps = np.random.normal(size=mu.shape)

    latent = mu + eps * std

    return latent, eps
```

$$\sigma^2 = e^{\text{logvar}} \implies \sigma = (\sigma^2)^{0.5} = e^{0.5\cdot\text{logvar}}$$

$$Z \sim \mathcal{N}_d(0, I) \implies \mu + \Sigma^{0.5} Z \sim \mathcal{N}_d(\mu, \Sigma)$$

랜덤이 아닌 부분을 뽑아내어
미분을 통한 오차 역전파 가능하게 함

VAE (Gaussian)

```python
def decoder(self, latent):
    z_d1 = latent @ self.W_d1.T + self.b_d1
    a_d1 = relu(z_d1) # activation

    z_d2 = a_d1 @ self.W_d2.T + self.b_d2
    a_d2 = relu(z_d2) # activation

    z_d3 = a_d2 @ self.W_d3.T + self.b_d3
    recon_x = sigmoid(z_d3) # sigmoid

    return z_d1, a_d1, z_d2, a_d2, recon_x
```

저차원의 잠재공간 벡터를 받아서
고차원의 데이터(이미지)로 재구성

cf)
```python
def sigmoid(z):
    return 1. / (1. + np.exp(-z))
```

```python
def forward(self, x):
    z_e1, a_e1, z_e2, a_e2, mu, logvar = self.encoder(x)
    latent, eps = self.reparameterize(mu, logvar)
    z_d1, a_d1, z_d2, a_d2, recon_x = self.decoder(latent)

    return z_e1, a_e1, z_e2, a_e2, mu, logvar, latent, z_d1, a_d1, z_d2, a_d2, recon_x, eps
```

순전파
인코더 → Reparameterization → 디코더

VAE (Gaussian)

역전파 : 디코더 → Reparameterization → 인코더

① 디코더에서의 역전파

② Reparameterization에서의 역전파

③ 인코더에서의 역전파

```
def backward(self, x, z_e1, a_e1, z_e2, a_e2, mu, logvar, latent, z_d1, a_d1, z_d2, a_d2, recon_x, y, eps):

    # decoder backprop
    dL__dz_d3 = (recon_x - y) / y.shape[0] # only by MSE

    dL__dw_d3 = dL__dz_d3.T @ a_d2
    dL__db_d3 = np.sum(dL__dz_d3, axis=0)


    dL__da_d2 = dL__dz_d3 @ self.W_d3
    dL__dz_d2 = dL__da_d2 * relu_deriv(z_d2) # activation derivative

    dL__dw_d2 = dL__dz_d2.T @ a_d1
    dL__db_d2 = np.sum(dL__dz_d2, axis=0)


    dL__da_d1 = dL__dz_d2 @ self.W_d2
    dL__dz_d1 = dL__da_d1 * relu_deriv(z_d1) # activation derivative

    dL__dw_d1 = dL__dz_d1.T @ latent
    dL__db_d1 = np.sum(dL__dz_d1, axis=0)
```

```
# reparameterize backprop
dL__dlatent = dL__dz_d1 @ self.W_d1

dL__dmu = dL__dlatent * 1 # gradient by CE term
dL__dstd = dL__dlatent * eps
std = np.exp(0.5 * logvar)
dL__dlogvar = dL__dstd * 0.5 * std # gradient by CE term

dL__dmu += mu  / y.shape[0]  # gradient by KL term
dL__dlogvar += 0.5 * (np.exp(logvar) - 1) / y.shape[0] # gradient by KL term
```

```
# encoder backprop
dL__dz_e3 = np.hstack((dL__dmu, dL__dlogvar))

dL__dw_e3 = dL__dz_e3.T @ a_e2
dL__db_e3 = np.sum(dL__dz_e3, axis=0)


dL__da_e2 = dL__dz_e3 @ self.W_e3
dL__dz_e2 = dL__da_e2 * relu_deriv(z_e2) # activation derivative

dL__dw_e2 = dL__dz_e2.T @ a_e1
dL__db_e2 = np.sum(dL__dz_e2, axis=0)


dL__da_e1 = dL__dz_e2 @ self.W_e2
dL__dz_e1 = dL__da_e1 * relu_deriv(z_e1) # activation derivative

dL__dw_e1 = dL__dz_e1.T @ x
dL__db_e1 = np.sum(dL__dz_e1, axis=0)

return dL__dw_d3, dL__db_d3, dL__dw_d2, dL__db_d2, dL__dw_d1, dL__db_d1, ₩
       dL__dw_e3, dL__db_e3, dL__dw_e2, dL__db_e2, dL__dw_e1, dL__db_e1
```

Comparison of Prior Distribution

```python
def reparameterize(self, mu, logvar):
    std = np.exp(0.5 * logvar)
    eps = np.random.normal(size=mu.shape)

    latent = mu + eps * std

    return latent, eps
```

Gaussian VAE

$\epsilon \sim \mathcal{N}(0,1)$: 가우시안 분포를 사용해 샘플링

단순 정규분포를 따르기 때문에, latent space가 제한적 & outlier에 민감.

표준 정규분포에서 sampling하여 latent variable 생성.

$z = \mu + \sigma \cdot \epsilon,\ \epsilon \sim \mathcal{N}(0,1).$

```python
def reparameterize(self, mu, logvar):
    std = np.exp(0.5 * logvar)
    eps = t.rvs(df=self.df, size=mu.shape) # student's t iid matrix with degree of freedom v+n

    latent = mu + eps * std

    return latent, eps
```

t-prior

$\epsilon \sim t(\nu)$:t분포를 사용해 sampling 하여 latent variable 생성.

T분포의 heavy tail 덕분에 outlier에 더 robust,

다양한 데이터에 대한 학습 가능.

자유도를 hyperparameter로 설정해 꼬리의 두께 조정.

$z = \mu + \sigma \cdot \epsilon,\ \epsilon \sim t(\nu).$

Comparison of Forward & Backward Propagation (KL Divergence)

```python
def forward(self, x):
    z_e1, a_e1, z_e2, a_e2, mu, logvar = self.encoder(x)
    latent, eps = self.reparameterize(mu, logvar)
    z_d1, a_d1, z_d2, a_d2, recon_x = self.decoder(latent)

    return z_e1, a_e1, z_e2, a_e2, mu, logvar, latent, z_d1, a_d1, z_d2, a_d2, recon_x, eps
```

Same

Forward 자체는 동일한 순서로 진행

다만, reparameterization trick에서 어떤 distribution을

통해 sampling을 진행할 것인지에 대한 차이만 존재.

```python
# reparameterize backprop
dL__dlatent = dL__dz_d1 @ self.W_d1

dL__dmu = dL__dlatent * 1 # gradient by CE term
dL__dstd = dL__dlatent * eps
std = np.exp(0.5 * logvar)
dL__dlogvar = dL__dstd * 0.5 * std # gradient by CE term

dL__dmu += mu  / y.shape[0] # gradient by KL term
dL__dlogvar += 0.5 * (np.exp(logvar) - 1) / y.shape[0] # gradient by KL term
```

Gaussian VAE

$$\frac{\partial KL}{\partial \mu} = \mu$$

$$\frac{\partial KL}{\partial \log \sigma^2} = \frac{1}{2}(\exp(\log \sigma^2) - 1)$$

정규분포의 단순성으로 linear한 관계, latent space의 tail부분

에 대한 추가적인 고려 X.

```python
# reparameterize backprop
dL__dlatent = dL__dz_d1 @ self.W_d1

dL__dmu = dL__dlatent * 1 # gradient by RCE term
dL__dstd = dL__dlatent * eps # gradient by RCE term
std = np.exp(0.5 * logvar)

dL__dmu += ((self.df + 1) * z_ / (self.df + np.power(z_, 2)) * 1) / y.shape[0] # gradient by KL term
dL__dstd += (-1 / std + (self.df + 1) * z_ / (self.df + np.power(z_, 2)) * eps_ ) / y.shape[0] # gradient by KL term

dL__dlogvar = dL__dstd * 0.5 * std
```

t-prior

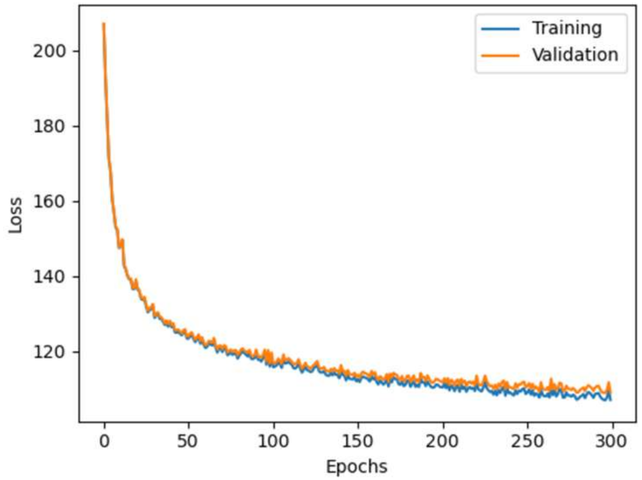
$$\frac{\partial KL}{\partial \mu} = \frac{\nu+1}{\nu+z^2} z$$

$$\frac{\partial KL}{\partial \sigma} = -\frac{1}{\sigma} + \frac{\nu+1}{\nu+z^2}\epsilon$$

Monte Carlo 샘플링을 기반으로, z^2&ϵ term이 추가되어 tail의 두께 반영.

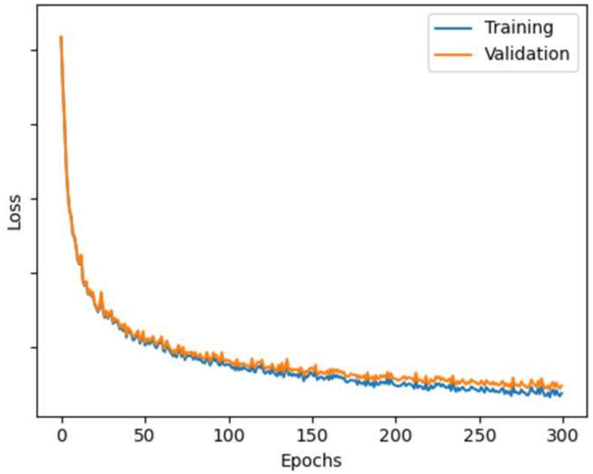
자유도 ν가 줄어들수록 더 heavy tail -> ouliter에 더 robust.

Training




```
epochs=300
batch_size=500
lr=0.005
```

```
model=VAE(28*28, 400, 128, 10)
```




```
epochs=300
batch_size=500
lr=0.005
df=21
```

```
model=t_prior_VAE(28*28, 400, 128, 10, df=df)
```

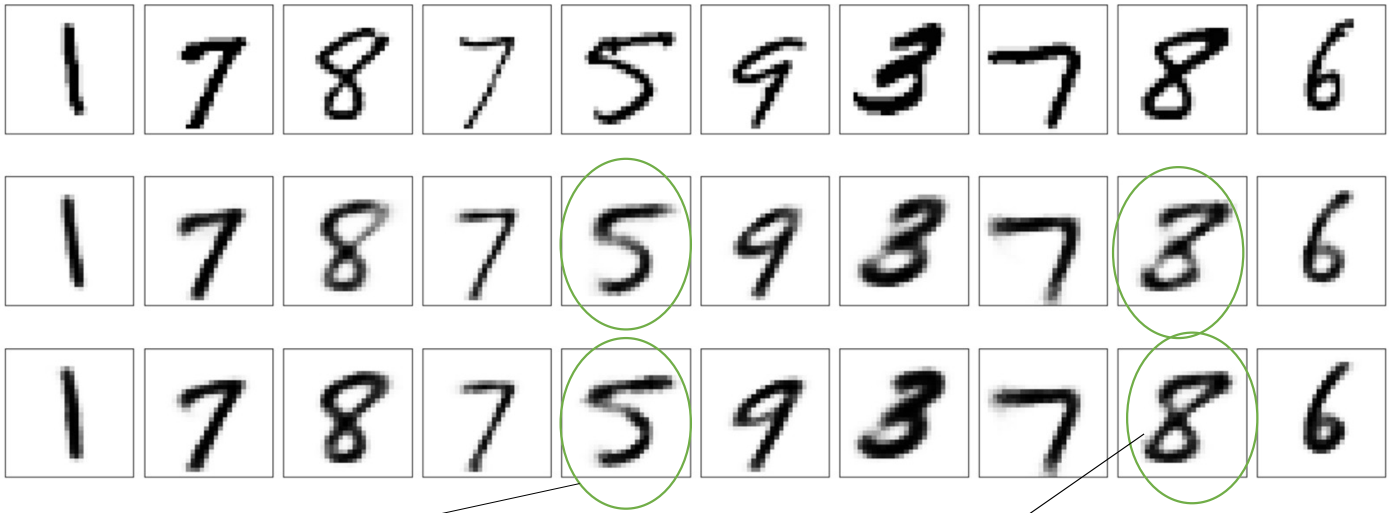
Data Reconstruction

```python
_, _, _, _, _, _, _, _, _, _, _, pred1, _ = model1.forward(X_test) # model1
_, _, _, _, _, _, _, _, _, _, _, pred2, _ = model2.forward(X_test) # model2

fig, ax = plt.subplots(figsize=(20,8), nrows=3, ncols=10, sharex=True, sharey=True)
ax = ax.flatten()

for i, j in enumerate(samples):
    img = X_test[j,:].reshape(28, 28)
    ax[i].imshow(img, cmap='Greys')

for i, j in enumerate(samples):
    img = pred1[j,:].reshape(28, 28)
    ax[i+10].imshow(img, cmap='Greys')

for i, j in enumerate(samples):
    img = pred2[j,:].reshape(28, 28)
    ax[i+20].imshow(img, cmap='Greys')

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
plt.show()
```

test

Gaussian VAE

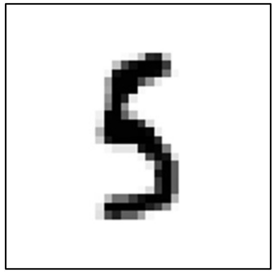
t-prior



T-prior better captures the distinctive features of the original data.

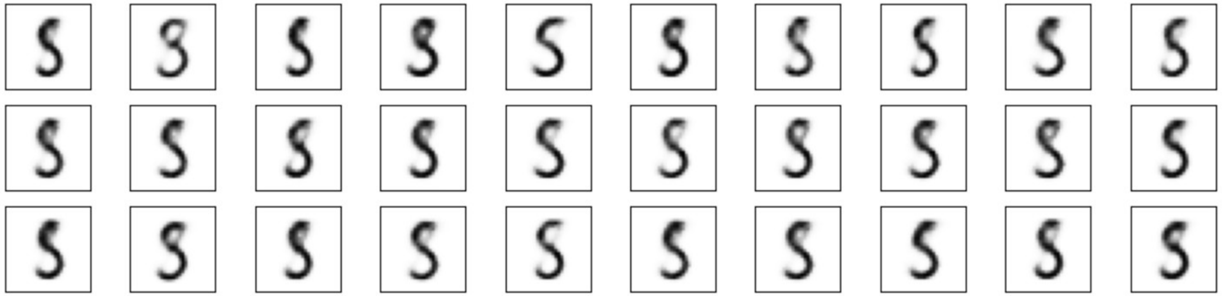
T-prior reconstructs the digit 8 more clearly, as the heavy-tailed nature of the t-distribution makes it more suitable for capturing outliers and complex structures.

Data sampling

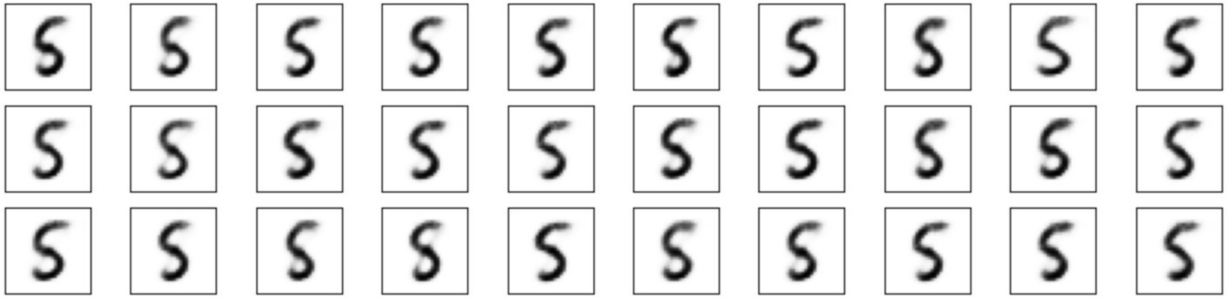


<Original image>

<image reconstructed by VAE>



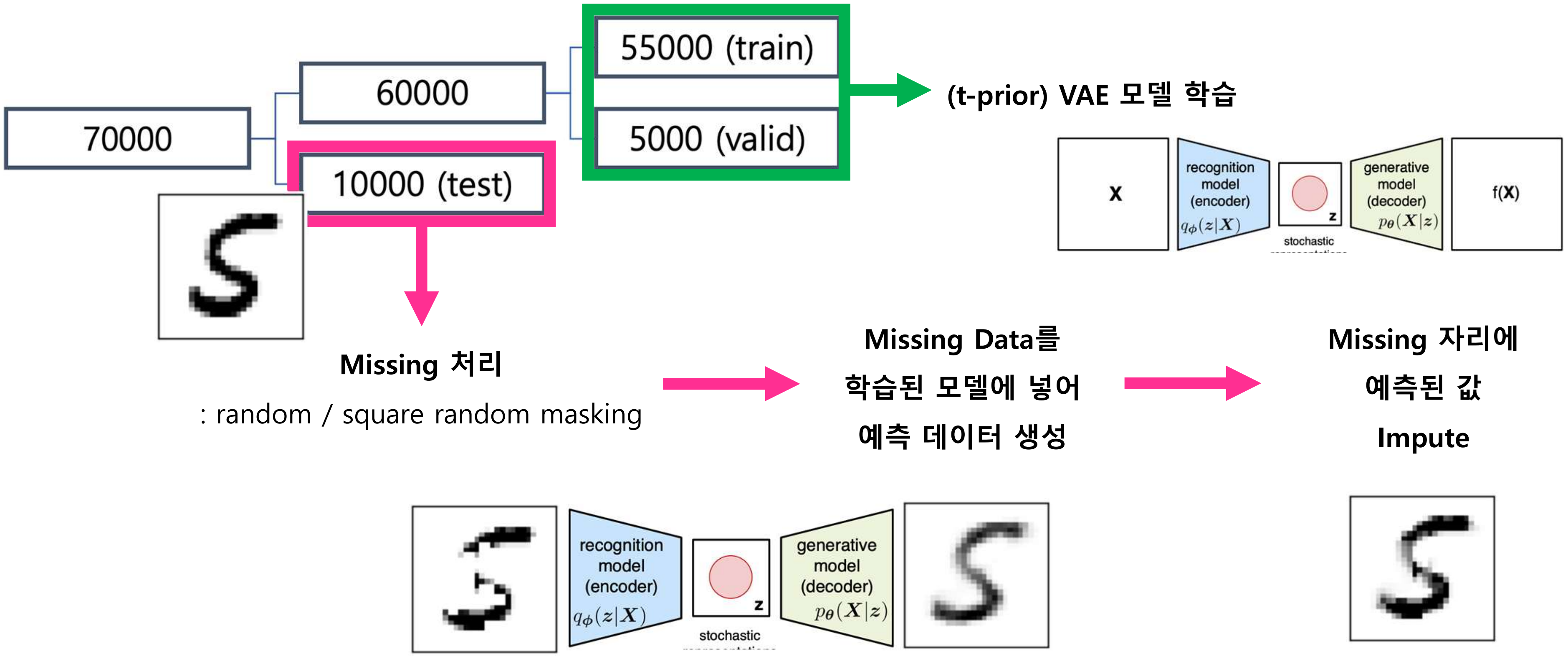
<image reconstructed by t-prior VAE>



4

Missing Analysis with VAE

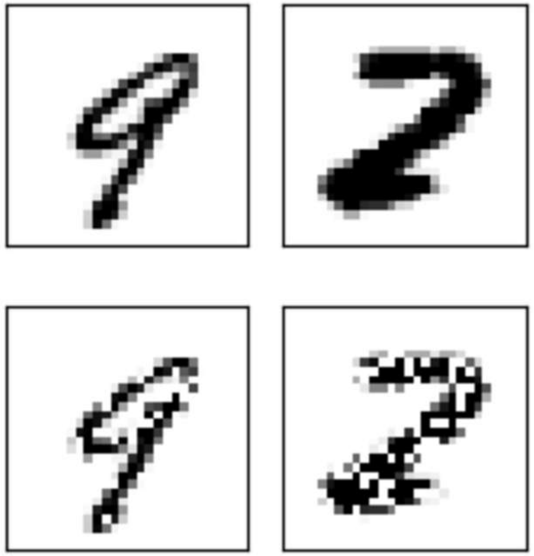
Workflow


55000 (train)
60000
5000 (valid)
70000
10000 (test)
(t-prior) VAE 모델 학습
X
recognition model (encoder) $q_\phi(z|X)$
z
stochastic
generative model (decoder) $p_\theta(X|z)$
f(X)
Missing 처리
: random / square random masking
Missing Data를
학습된 모델에 넣어
예측 데이터 생성
Missing 자리에
예측된 값
Impute
recognition model (encoder) $q_\phi(z|X)$
z
stochastic
generative model (decoder) $p_\theta(X|z)$

Missing Algorithms

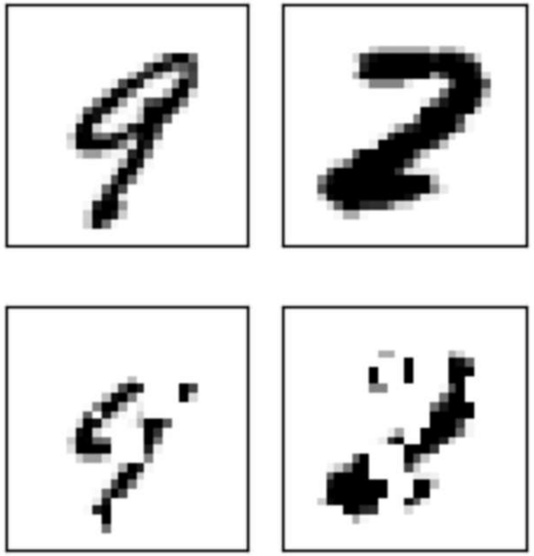
Random Masking

: 랜덤한 픽셀의 값을 0으로 missing 처리

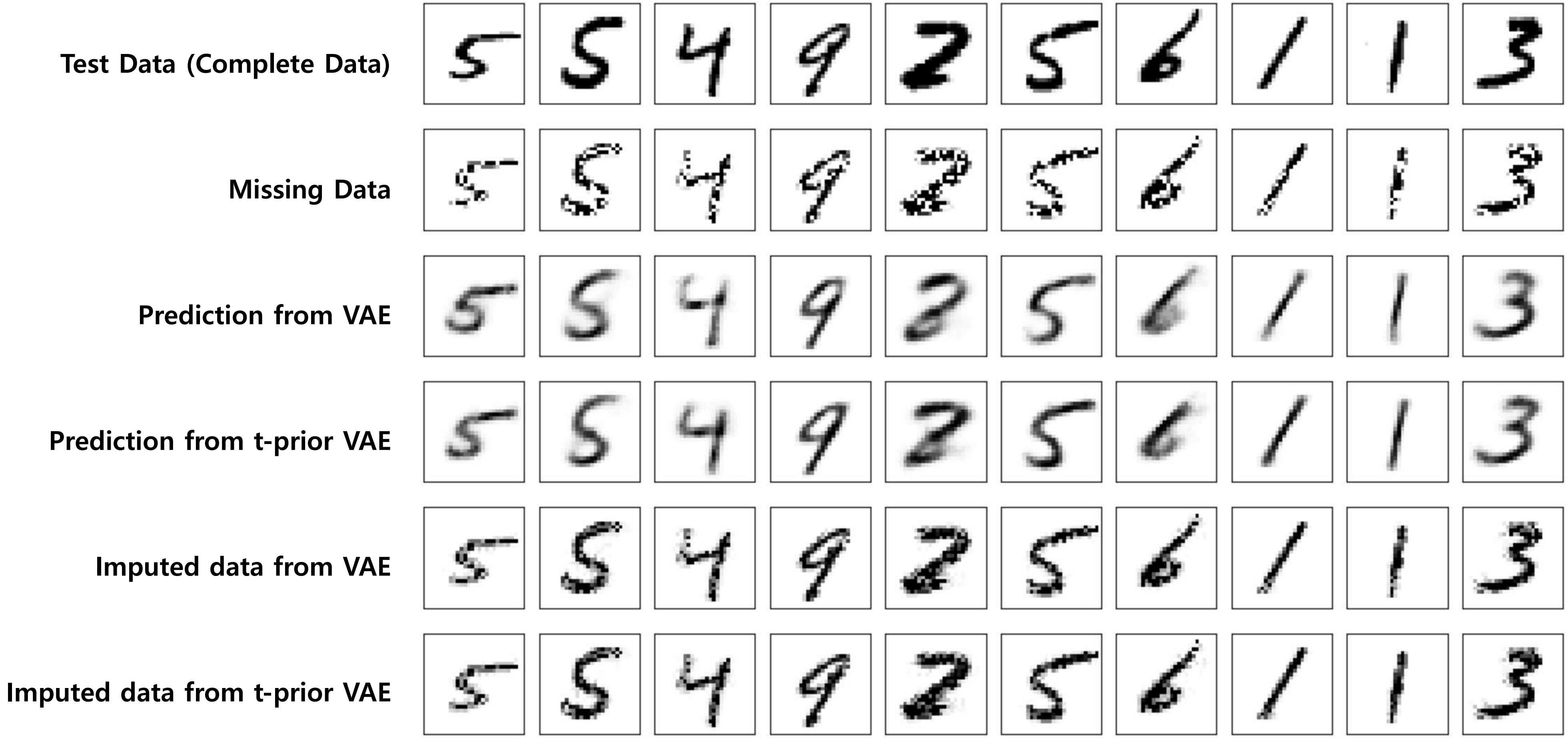


Square Random Masking

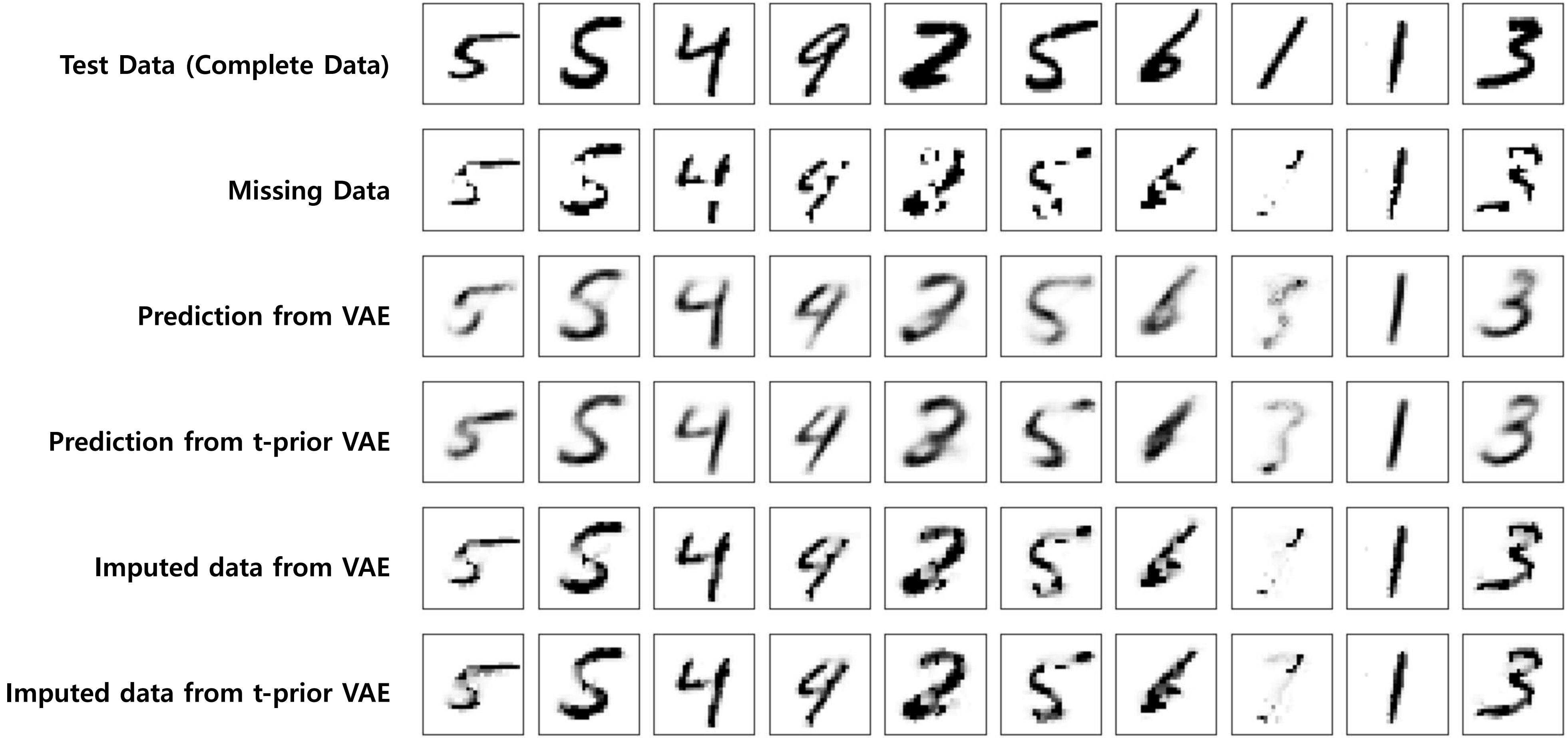
: 랜덤한 정사각형 영역의 픽셀의 값을 0으로 missing 처리



Results – Random missing



Results – Square random missing



Results – VAE vs. t-prior VAE

		Random missing		Square random missing	
		VAE	t-prior VAE	VAE	t-prior VAE
MSE	Prediction	0.0280	**0.0247**	0.0452	**0.0405**
	Imputation	0.0091	**0.0081**	0.0244	**0.0222**
SSIM	Prediction	0.7676	**0.8150**	0.5746	**0.6506**
	Imputation	0.9443	**0.9523**	0.8247	**0.8469**

➡ (Gaussian) VAE 보다 t-prior VAE의 성능이 더 뛰어나다!

감사합니다