Auto-Encoding Variational Bayes

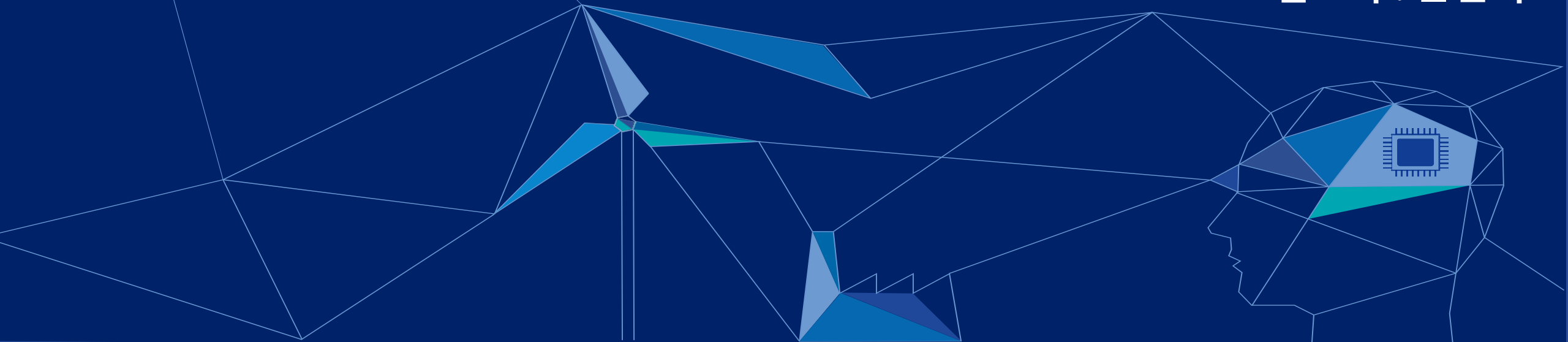
Kingma, D. P. & Welling, M. (2014)

Variational Graph Auto-Encoders

Kipf, T., & Welling, M. (2016)

24.01.31 (수)

발표자 : 윤민석

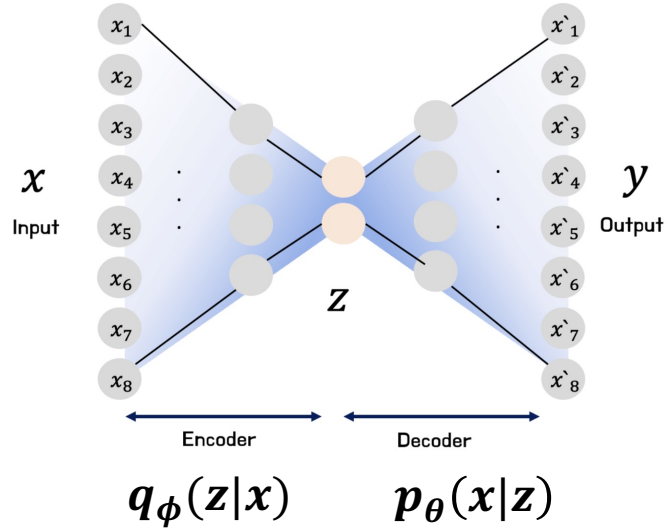


Contents

- ✓ Background
- ✓ Paper #1 (Auto-Encoding Variational Bayes)
  - ✓ Methodology
  - ✓ Code
- ✓ Paper #2 (Variational Graph Auto-Encoders)
  - ✓ Methodology
  - ✓ Code
- ✓ Takeaway

Background

- Autoencoder
  - Unsupervised Leanring
  - Encoder(Recognition network) : Input data를 Latent vector로 변환
  - Decoder(Generative network) : Latent vector를 output data로 변환



Background

- MLE & MAP
  - Maximum Likelihood Estimation (MLE)

$$\theta_{MLE} = \arg\max_{\theta} \log P(X|\theta)$$

$$= \arg\max_{\theta} \log \prod_{i} P(x_i|\theta)$$

$$= \arg\max_{\theta} \sum_{i} \log P(x_i|\theta)$$

  - Maximum A posterior Probability (MAP)

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

$$\theta_{MAP} = \arg\max_{\theta} P(X|\theta)P(\theta)$$

$$= \arg\max_{\theta} \log P(X|\theta) + \log P(\theta)$$

$$= \arg\max_{\theta} \log \prod_{i} P(x_i|\theta) + \log P(\theta)$$

$$= \arg\max_{\theta} \sum_{i} \log P(x_i|\theta) + \log P(\theta)$$

Background

- MLE
  - Negative Log-likelihood

$$\theta_{MLE} = \arg\max_{\theta} \log P(X|\theta)$$

$$= \arg\max_{\theta} \log \prod_{i} P(x_i|\theta)$$

$$= \arg\max_{\theta} \sum_{i} \log P(x_i|\theta)$$

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{N} -\log P(y_i|x_i;\theta)$$

If Gaussian

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \Sigma) = -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\Sigma| - \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}_n - \boldsymbol{\mu})$$

If Bernoulli

$$-\sum_{i=1}^{N} \log P(y_i|x_i;\theta) = -\sum_{i=1}^{N} y_i^T \cdot \log \hat{y}_i$$

Background

- KL Divergence
  - 두 확률분포의 차이를 계산하는 함수

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

$$KL(p,q) = -\int p(x)\log q(x)dx + \int p(x)\log p(x)dx$$

$$= \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}(1 + \log 2\pi\sigma_1^2)$$

$$= \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

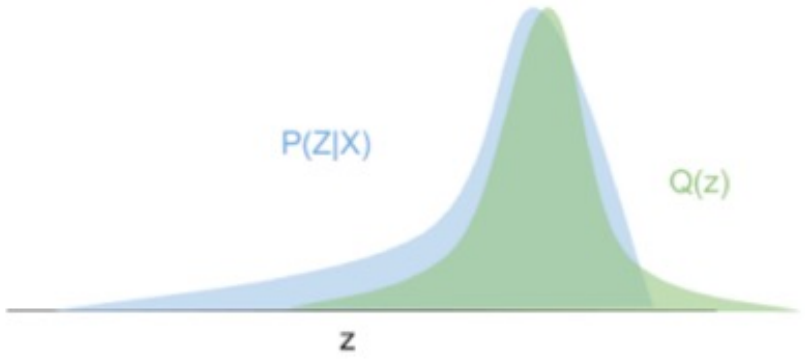
- Monte Carlo simulation
  - 대수의 법칙에 의해 모평균으로 수렴

$$\mathrm{E[H(x)]} \approx \frac{1}{N}\sum_{i=1}^{N} H(x_i)$$

$\frac{1}{N}\sum_{i=1}^{N} H(x_i)$: *Monte Carlo Simulation* 값

Background

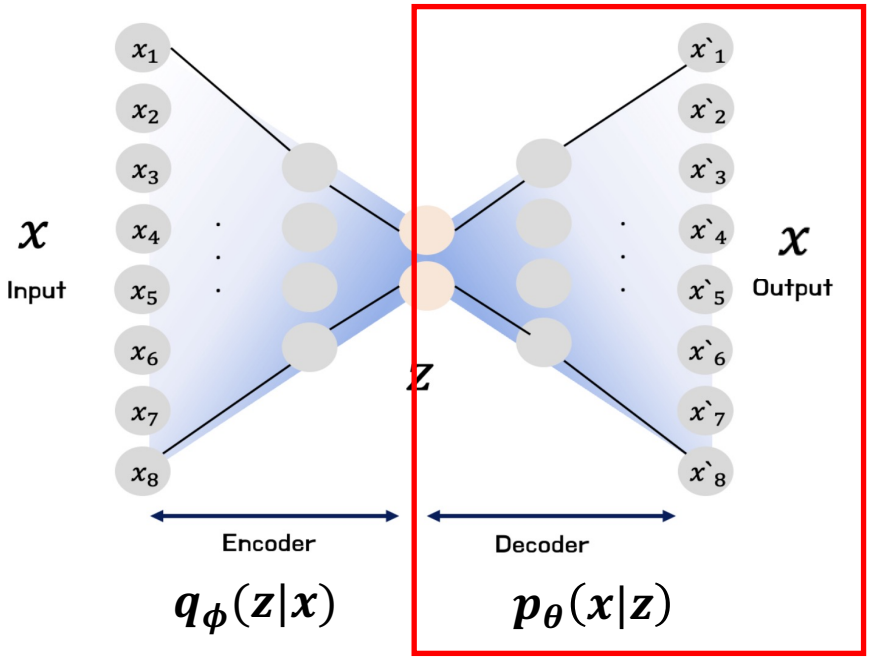
- Variational Inference
  - 추론 목적: Likelihood 계산 + Posterior(P(z|x)) 구하기
  - 대부분의 경우 P(z|x)의 정확한 분포를 모름
  - 사후확률(Posterior) 분포 p(z|x)를 다루기 쉬운 확률분포 q(z)로 근사



P(z|x) : 구하기 어려운 분포
Q(z) : 가우시안 분포

Paper #1 (VAE)

- Motivation
  - How can we perform efficient approximate inference and learning with directed probabilistic models whose continuous latent variables and/or parameters **have intractable posterior distributions?**



모든 경우를 적분하는 것이 어려움

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)\ dz$$

분포 가정 분포 가정

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$$

Intractable Posterior

구하기 어려움

$p_\theta(z|x)$ 대신 $q_\phi(z|x)$를 쓰자

Variational Inference

Paper #1 (VAE)

- Directed Graphical model

모든 z에 대해 적분하는 것이 어려움

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)\ dz$$

분포 가정 분포 가정

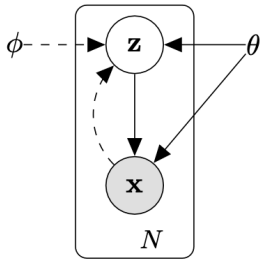
$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$$

Intractable Posterior

구하기 어려움

$p_\theta(z|x)$ 대신 $q_\phi(z|x)$를 쓰자

Variational Inference



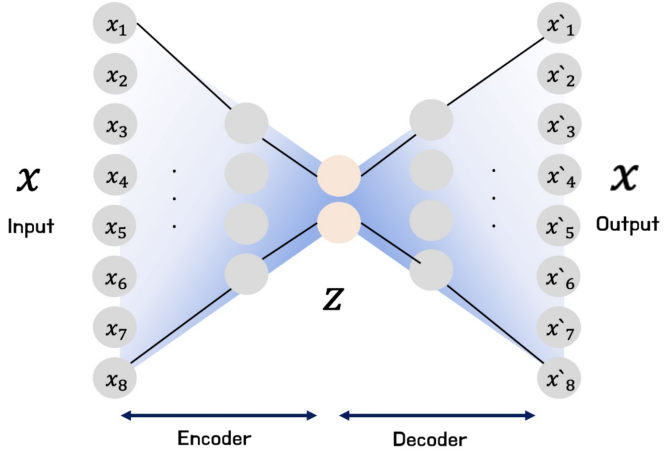


Figure 1: The type of directed graphical model under consideration. Solid lines denote the generative model $p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, dashed lines denote the variational approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ to the intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The variational parameters $\boldsymbol{\phi}$ are learned jointly with the generative model parameters $\boldsymbol{\theta}$.

Paper #1 (VAE)

- Variational Inference (Evidence Lower BOund)

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \quad (1)$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) + \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})] \quad (2)$$

$$log\ p_\theta(x) = log\ p_\theta(x) \underbrace{\int q_\phi(z|x)\ dz}_{=1} = \int log\ p_\theta(x)\ q_\phi(z|x)\ dz$$

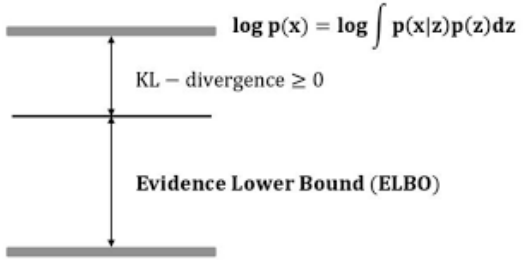
$$= \int log\ [\frac{p_\theta(x,z)}{p_\theta(z|x)}]\ q_\phi(z|x)\ dz$$

$$= \int log\ [\frac{p_\theta(x,z)}{q_\phi(z|x)} \frac{q_\phi(z|x)}{p_\theta(z|x)}]\ q_\phi(z|x)\ dz$$

$$= \int [\log\ (\frac{p_\theta(x,z)}{q_\phi(z|x)}) + \log(\frac{q_\phi(z|x)}{p_\theta(z|x)})\ ] q_\phi(z|x)\ dz$$

$$= \int log\ \frac{p_\theta(x,z) p_\theta}{q_\phi(z|x)} q_\phi(z|x)\ dz + \int log \frac{q_\phi(z|x)}{p_\theta(z|x)} q_\phi(z|x)\ dz$$

$$= \mathcal{L}(\theta, \phi; x) + \underbrace{D_{KL}(q_\phi(z|x)||p_\theta(z|x))}_{\text{Intractable}} \geq \underbrace{\mathcal{L}(\theta, \phi; x)}_{\text{Lower bound}}$$



Paper #1 (VAE)

- Variational Inference (Evidence Lower BOund)

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right] \quad (3)$$

$\mathcal{L}(\theta, \phi; x)$ = $\int log\ \frac{p_\theta(x|z)}{q_\phi(z|x)} q_\phi(z|x)\ dz = \int [\log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}] q_\phi(z|x)\ dz$

Lower bound

$= \int \log p_\theta(x|z)\, q_\phi(z|x) dz + \int \log \frac{p_\theta(x)}{q_\phi(z|x)} q_\phi(z|x)\ dz$

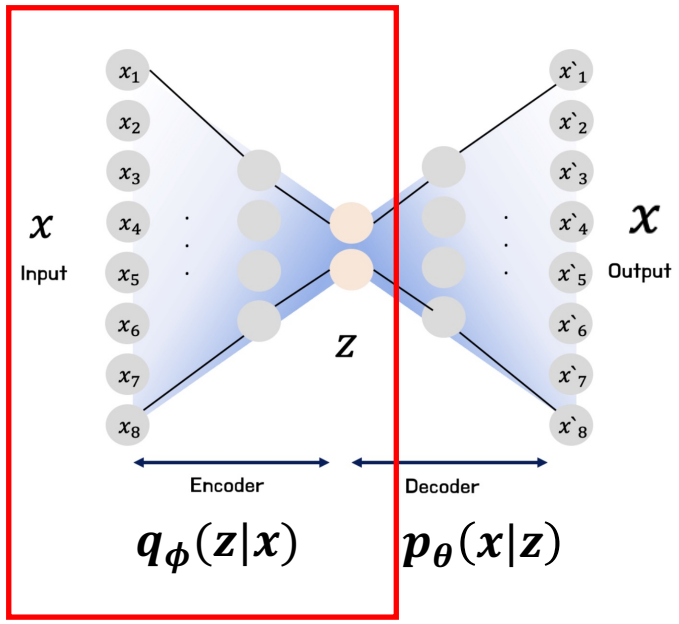
$= \mathrm{E}_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - \int \log \frac{q_\phi(z|x)}{p_\theta(x)} q_\phi(z|x)$

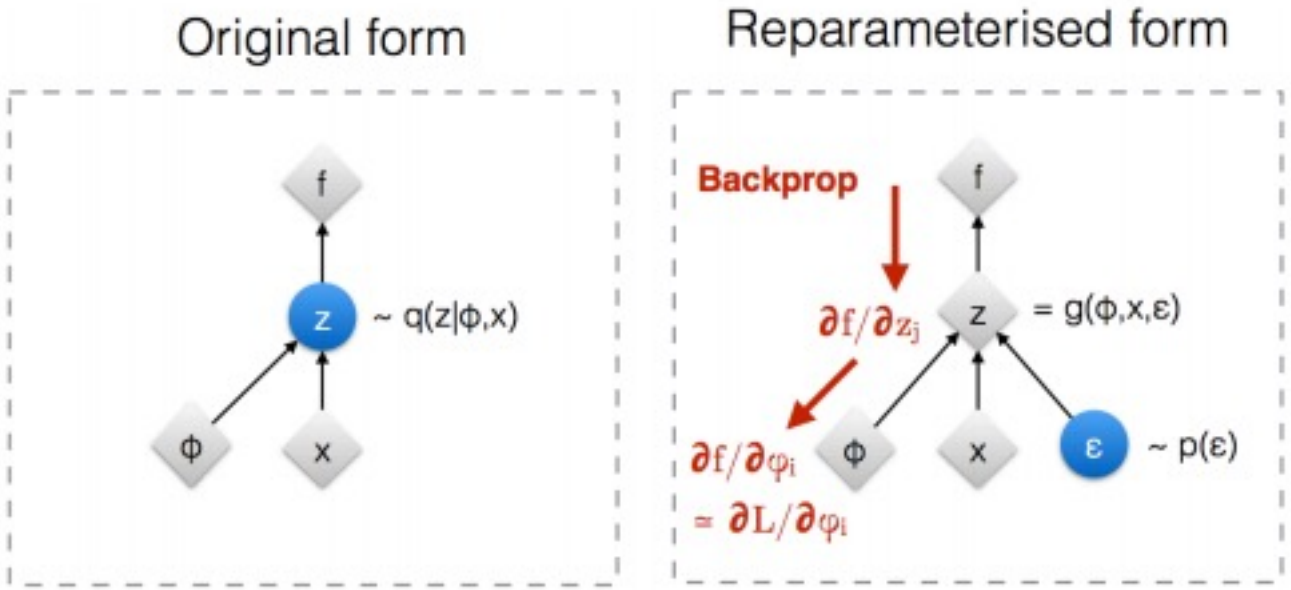
$= \mathrm{E}_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z))$

Reconstruction　　Regularization

Paper #1 (VAE)

- Reparametrization Trick





$z \sim q_\phi(z|x)$

High variance

$z = g_\phi(\epsilon, x)$

$z = \mu + \sigma\epsilon$

Paper #1 (VAE)

- ELBO Structure
  - $\mathcal{L}(\theta, \phi; x) = - D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathrm{E}_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)]$

Monte Carlo simulation

$$\widetilde{\mathcal{L}}^B(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \frac{1}{L}\sum_{l=1}^{L}(\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}))$$

where $\mathbf{z}^{(i,l)} = g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(i,l)}, \mathbf{x}^{(i)})$ and $\boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})$ (7)

Reparametrization Trick

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \simeq \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) + \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$$

where $\mathbf{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)}$ and $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})$ (10)

Paper #1 (VAE)

- Code

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim
from torchvision import datasets, transforms
from torchvision.utils import save_image
import matplotlib.pyplot as plt

device = 'cpu'
# 데이터셋 불러오기

train_dataset = datasets.MNIST(root='./mnist_data/', train=True, transform=transforms.ToTensor(), download=True)
test_dataset = datasets.MNIST(root='./mnist_data/', train=False, transform=transforms.ToTensor(), download=False)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=100, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=100, shuffle=False)
```

```python
def loss_function(recon_x, x, average, log_variance):
    # Minimize 상태로 만들기
    binary_cross_entropy = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    KL_divergence = - 0.5 * torch.sum(1 + log_variance - average ** 2 - log_variance.exp())
    return binary_cross_entropy + KL_divergence
```

```python
class VAE(nn.Module):
    def __init__(self, x_dim, h_dim1, h_dim2, z_dim):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(x_dim, h_dim1)
        self.fc2 = nn.Linear(h_dim1, h_dim2)
        self.fc_average = nn.Linear(h_dim2, z_dim)
        self.fc_log_variance = nn.Linear(h_dim2, z_dim)

        self.fc3 = nn.Linear(z_dim, h_dim2)
        self.fc4 = nn.Linear(h_dim2, h_dim1)
        self.fc5 = nn.Linear(h_dim1, x_dim)

    def encoder(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc_average(x), self.fc_log_variance(x)

    def reparameterization(self, average, log_variance):
        std = torch.exp(0.5 * log_variance)
        eps = torch.randn_like(std)
        return average + eps * std

    def decoder(self, z):
        z = F.relu(self.fc3(z))
        z = F.relu(self.fc4(z))
        return torch.sigmoid(self.fc5(z))

    def forward(self, x):
        average, log_variance = self.encoder(x.view(-1, 784))
        z = self.reparameterization(average, log_variance)
        return self.decoder(z), average, log_variance
```

Paper #1 (VAE)

- Code

```python
train_loss_list = []
test_loss_list = []

def train(epoch):
    model.train()
    train_loss = 0
    for data, _ in train_loader:
        data = data.to(device)
        optimizer.zero_grad()
        x_reconstruct, average, log_variance = model(data)
        loss = loss_function(x_reconstruct, data, average, log_variance)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
    print("{}'s avrage train loss : {}".format(epoch, train_loss / len(train_loader.dataset)))
    train_loss_list.append(train_loss / len(train_loader.dataset))

def test(epoch):
    model.eval()
    test_loss= 0
    with torch.no_grad():
        for data, _ in test_loader:
            data = data.to(device)
            x_reconstruct, average, log_variance = model(data)
            test_loss += loss_function(x_reconstruct, data, average, log_variance).item()
    print("{}'s avrage test loss : {}".format(epoch, test_loss / len(test_loader.dataset)))
    test_loss_list.append(test_loss / len(test_loader.dataset))
```

```python
z_list = [3, 5, 10, 20 ,200]
for z in z_list:
    train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=100, shuffle=True)
    test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=100, shuffle=False)
    model = VAE(x_dim = 28 ** 2, h_dim1 = 500, h_dim2 = 250, z_dim = z).to(device)
    optimizer = torch.optim.Adagrad(model.parameters(), lr = 0.01)
    train_loss_list = []
    test_loss_list = []
    for epoch in range(1, 21):
        train(epoch)
        test(epoch)

    epochs = range(1, 21)
    plt.plot(epochs, train_loss_list, label='Train Loss')
    plt.plot(epochs, test_loss_list, label='Test Loss')
    plt.ylim(100, 150)
    plt.xticks(range(1, 21))
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Train and Test Loss Over Epochs')
    plt.show()
```

Paper #1 (VAE)

- Code

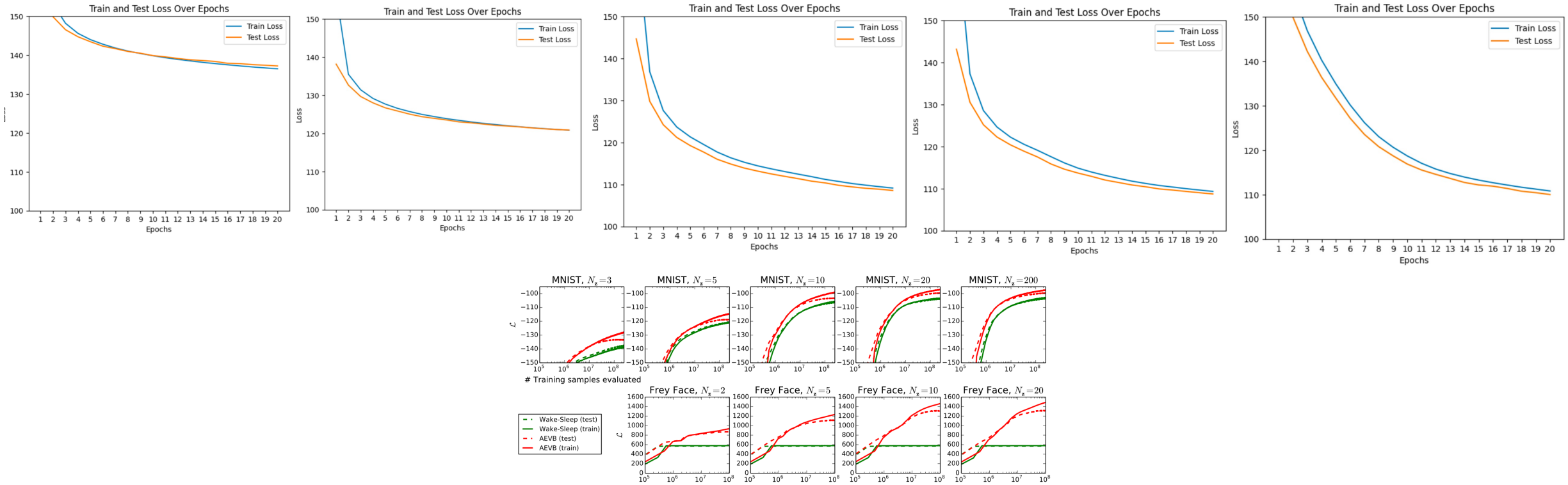


Figure 2: Comparison of our AEVB method to the wake-sleep algorithm, in terms of optimizing the lower bound, for different dimensionality of latent space ($N_{\mathbf{z}}$). Our method converged considerably faster and reached a better solution in all experiments. Interestingly enough, more latent variables does not result in more overfitting, which is explained by the regularizing effect of the lower bound. Vertical axis: the estimated average variational lower bound per datapoint. The estimator variance was small (< 1) and omitted. Horizontal axis: amount of training points evaluated. Computation took around 20-40 minutes per million training samples with a Intel Xeon CPU running at an effective 40 GFLOPS.

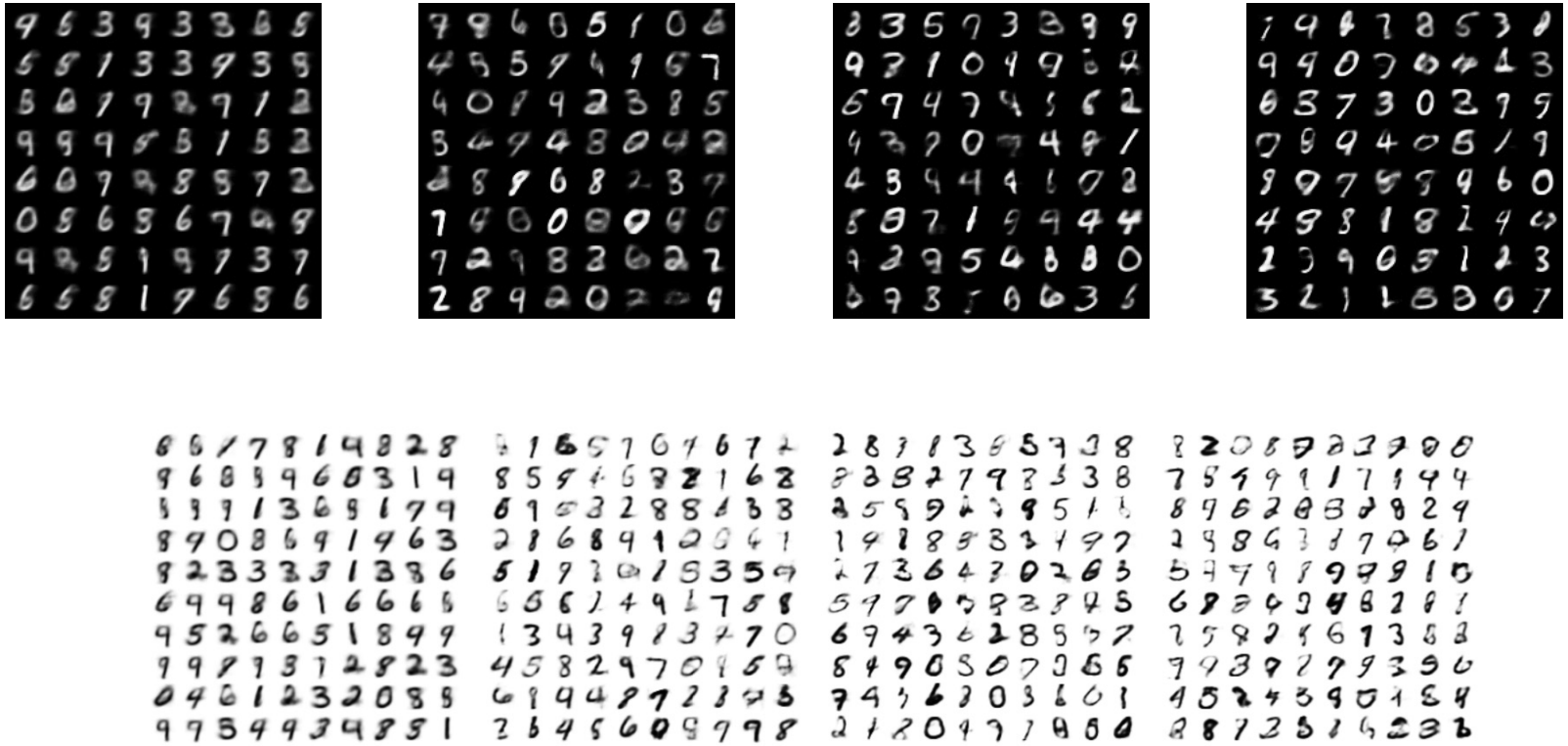
Paper #1 (VAE)

- Code

```python
z_list = [2, 5, 10, 20]
for z in z_list:
    train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=100, shuffle=True)
    test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=100, shuffle=False)
    model = VAE(x_dim = 28 ** 2, h_dim1 = 500, h_dim2 = 250, z_dim = z).to(device)
    optimizer = torch.optim.Adagrad(model.parameters(), lr = 0.01)
    train_loss_list = []
    test_loss_list = []
    for epoch in range(1, 21):
        train(epoch)
        test(epoch)

    with torch.no_grad():
        z_random = torch.randn(64, z).to(device)
        sample = model.decoder(z_random).to(device)

        save_image(sample.view(64, 1, 28, 28), './samples' + '{}.png'.format(z))
```

Paper #1 (VAE)

- Code

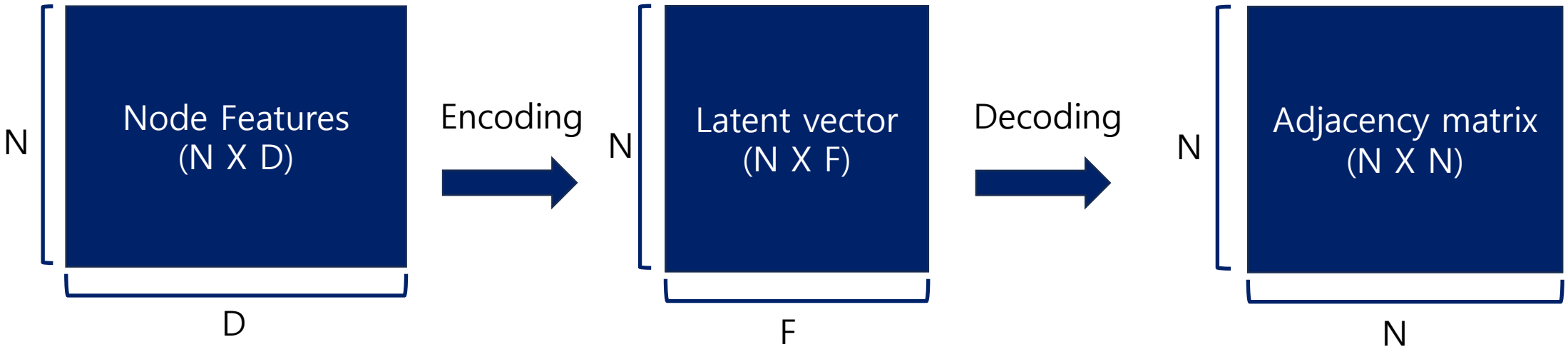


(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 5: Random samples from learned generative models of MNIST for different dimensionalities of latent space.

Paper #2 (VGAE)

- Motivation
  - We introduce the variational graph autoencoder (VGAE), a framework for unsupervised learning on graph-structured data based on the variational auto-encoder (VAE).
  - Use of latent variables
  - Link prediction problem



Paper #2 (VGAE)

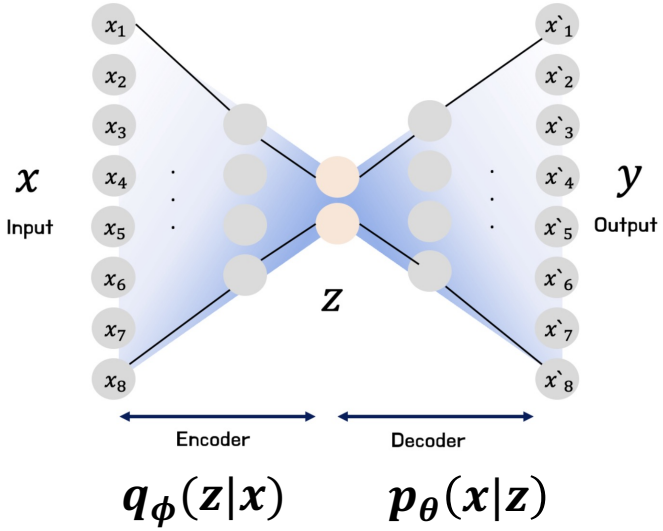
- Encoder
  - Using GCN
  - Reparametrization trick

$$q(\mathbf{Z}\,|\,\mathbf{X},\mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i\,|\,\mathbf{X},\mathbf{A}), \quad \text{with} \quad q(\mathbf{z}_i\,|\,\mathbf{X},\mathbf{A}) = \mathcal{N}(\mathbf{z}_i\,|\,\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2)).$$

$$\mathrm{GCN}(\mathbf{X},\mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}\big(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0\big)\mathbf{W}_1$$

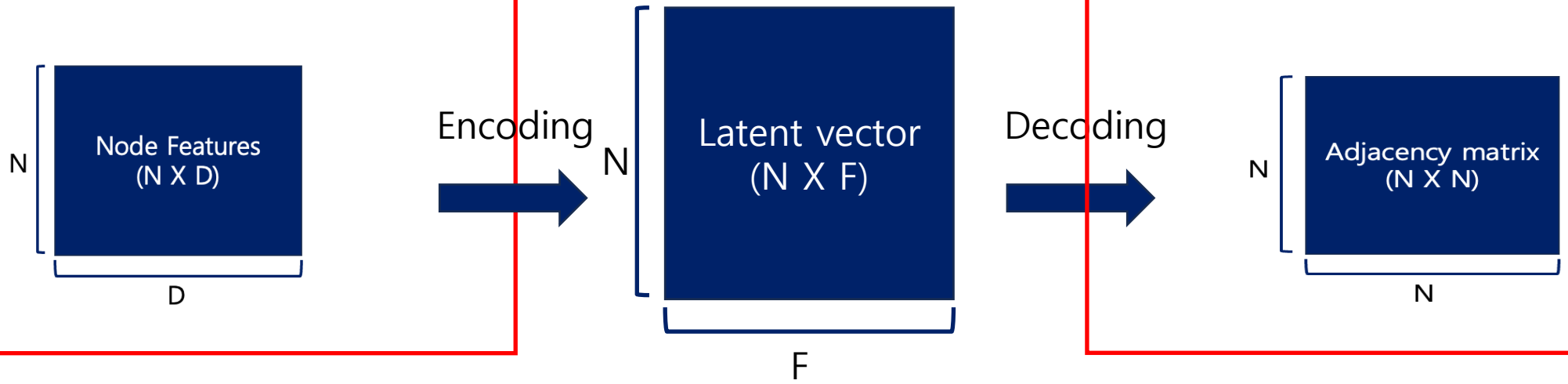
$$\boldsymbol{\mu} = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X},\mathbf{A})$$

$$\log \boldsymbol{\sigma} = \mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X},\mathbf{A})$$



- Decoder
  - Using inner product

$$p(\mathbf{A}\,|\,\mathbf{Z}) = \prod_{i=1}^{N}\prod_{j=1}^{N} p(A_{ij}\,|\,\mathbf{z}_i,\mathbf{z}_j), \quad \text{with} \quad p(A_{ij}=1\,|\,\mathbf{z}_i,\mathbf{z}_j) = \sigma(\mathbf{z}_i^{\top}\mathbf{z}_j).$$



Paper #2 (VGAE)

- Learning
  - Optimize function

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}\left[\log p\left(\mathbf{A} \mid \mathbf{Z}\right)\right] - \mathrm{KL}\left[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \,||\, p(\mathbf{Z})\right]$$

  - Featureless approach (VGAE*)
    - Replace X with the identity matrix in the GCN

$$\mathrm{GCN}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}\left(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0\right)\mathbf{W}_1$$

$$\boldsymbol{\mu} = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A})$$

$$\log \boldsymbol{\sigma} = \mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$$

Paper #2 (VGAE)

- GAE
  - A non-probabilistic variant of the VGAE model
  - Only use reconstruction error

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^{\top}\right), \quad \text{with} \quad \mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A})$$

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}\left[\log p\left(\mathbf{A} \mid \mathbf{Z}\right)\right] - \mathrm{KL}\left[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \,||\, p(\mathbf{Z})\right]$$

  - Featureless approach (GAE*)
    - Replace X with the identity matrix in the GCN

Paper #2 (VGAE)

- Experiments
  - To learn meaningful latent embeddings on a link prediction task on several popular citation network datasets.
  - Split Train / Validation / Test set **(disjoint)**
  - Use negative sampling

Table 1: Link prediction task in citation networks. See [1] for dataset details.

Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
DW [6]	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02	82.2 ± 0.01	87.4 ± 0.00
VGAE*	84.0 ± 0.02	87.7 ± 0.01	78.9 ± 0.03	84.1 ± 0.02	82.7 ± 0.01	87.5 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	$\mathbf{96.4} \pm 0.00$	$\mathbf{96.5} \pm 0.00$
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$	$\mathbf{90.8} \pm 0.02$	$\mathbf{92.0} \pm 0.02$	94.4 ± 0.02	94.7 ± 0.02

Paper #2 (VGAE)

- Code (Failure Case)

```python
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim
import dgl
from dgl.nn import GraphConv
from torch_geometric.datasets import Planetoid
from torch_geometric.transforms import RandomLinkSplit

device = 'cpu'

# 데이터 불러오기
dataset = Planetoid(root='/tmp/Cora', name='Cora')
data = dataset[0]

# Train / Validation / Test split
transform = RandomLinkSplit(is_undirected=True, num_val = 0.05, num_test = 0.1)
train_data, val_data, test_data = transform(data)

num_nodes = data.num_nodes
```

```python
# 인접행렬 만들기
train_edge_index = train_data.edge_index
train_adjacency_matrix = torch.zeros((num_nodes, num_nodes))
train_adjacency_matrix[train_edge_index[0], train_edge_index[1]] = 1
train_adjacency_matrix = (train_adjacency_matrix + np.identity(2708)).to(torch.float32)

test_edge_index = test_data.edge_index
test_adjacency_matrix = torch.zeros((num_nodes, num_nodes))
test_adjacency_matrix[test_edge_index[0], test_edge_index[1]] = 1
test_adjacency_matrix = (test_adjacency_matrix + np.identity(2708)).to(torch.float32)

node_features = data.x


# 그래프 데이터 만들기
train_edge_index = train_edge_index.tolist()
train_a = torch.tensor(train_edge_index[0])
train_b = torch.tensor(train_edge_index[1])
train_graph = dgl.graph((train_a, train_b))
train_graph = dgl.add_self_loop(train_graph)

test_edge_index = test_edge_index.tolist()
test_a = torch.tensor(test_edge_index[0])
test_b = torch.tensor(test_edge_index[1])
test_graph = dgl.graph((test_a, test_b))
test_graph = dgl.add_self_loop(test_graph)

train_graph = train_graph.to(device)
test_graph = test_graph.to(device)
```

Paper #2 (VGAE)

- Code (Failure Case)

```python
class VGAE(nn.Module):
    def __init__(self, in_dim = 1433, hidden1_dim = 32, z = 16):
        super().__init__()
        self.graph_conv1 = GraphConv(in_dim, hidden1_dim, activation = F.relu, allow_zero_in_degree = True)
        self.graph_conv_average = GraphConv(hidden1_dim, z, activation = lambda x: x, allow_zero_in_degree = True)
        self.graph_conv_log_variance = GraphConv(hidden1_dim, z, activation = lambda x: x, allow_zero_in_degree = True)

    def encoder(self, adj, features):
        z = self.graph_conv1(adj, features)
        average = self.graph_conv_average(adj, z)
        log_variance = self.graph_conv_log_variance(adj, z)
        return average, log_variance

    def reparameterization(self, average, log_variance):
        std = torch.exp(0.5 * log_variance)
        eps = torch.randn_like(std)
        return average + std * eps

    def decoder(self, z):
        new_adj = torch.sigmoid(torch.matmul(z, z.t()))
        return new_adj

    def forward(self, adj, features):
        average, log_variance = self.encoder(adj, features)
        z = self.reparameterization(average, log_variance)
        new_adj = self.decoder(z)
        return new_adj, average, log_variance
```

```python
def loss_function(new_adj, adj, average, log_variance):
    # Minimize 상태로 만들기
    binary_cross_entropy = F.binary_cross_entropy(new_adj, adj, reduction='sum')
    KL_divergence = - 0.5 * torch.sum(1 + log_variance - average ** 2 - log_variance.exp())
    return binary_cross_entropy + KL_divergence
```

```python
train_loss_list = []
test_loss_list = []
model = VGAE().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = 0.03)
```

Paper #2 (VGAE)

- Code (Failure Case)

```python
def train():
    with train_graph.local_scope():
        model.train()
        optimizer.zero_grad()
        new_adj, average, log_variance = model(train_graph, node_features)
        loss = loss_function(new_adj, train_adjacency_matrix, average, log_variance)
        loss.backward()
        optimizer.step()
        train_loss_list.append(loss.item())
        print("Train Loss: {}".format(loss.item()))


def test():
    model.eval()
    with test_graph.local_scope():
        with torch.no_grad():
            new_adj, average, log_variance = model(test_graph, node_features)
            loss = loss_function(new_adj, test_adjacency_matrix, average, log_variance)
            train_loss_list.append(loss.item())
            print("test Loss: {}".format(loss.item()))
    return new_adj
```

```python
for epoch in range(1,201):
    train()
    test()
```

```
Train Loss: 5259356.0
test Loss: 5216732.0
Train Loss: 5213807.0
test Loss: 5201317.5
Train Loss: 5201628.0
test Loss: 5217998.0
Train Loss: 5219848.0
test Loss: 5227190.5
Train Loss: 5227084.5

show more (open the raw output data in a text editor) ...

test Loss: 5157293.5
Train Loss: 5156840.0
test Loss: 5156893.5
Train Loss: 5156632.5
test Loss: 5156985.5
```

```python
new_adj = np.array(test())
binary_adj = (new_adj >= 0.5).astype(int)

test_adjacency_matrix = np.array(test_adjacency_matrix)

from sklearn.metrics import roc_auc_score, precision_score
auc_score = roc_auc_score(binary_adj.flatten(), test_adjacency_matrix.flatten())
precision = precision_score(binary_adj.flatten(), test_adjacency_matrix.flatten(), average='binary')
print("AUC Score:", auc_score)
print("Precision:", precision)
```

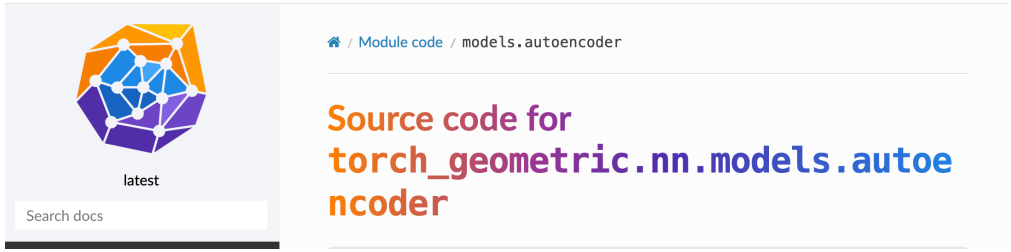
```
AUC Score: 0.5004482530894087
Precision: 0.6376740376740376
```

Paper #2 (VGAE)

- Code (Success Case)

```python
import numpy as np
import torch
import torch_geometric.transforms as T
from torch_geometric.datasets import Planetoid
from torch_geometric.nn import GCNConv, VGAE

device = 'cpu'
transform = T.Compose([
    T.NormalizeFeatures(),
    T.RandomLinkSplit(num_val=0.05, num_test=0.1, is_undirected=True, split_labels=True, add_negative_train_samples=False),
])

dataset = Planetoid('.', name='Cora', transform=transform)
train_data, val_data, test_data = dataset[0]
train_data, val_data, test_data = train_data.to(device), val_data.to(device), test_data.to(device)
```

```python
class Encoder(torch.nn.Module):
    def __init__(self, dim_in, hidden_dim = 32, dim_out = 16):
        super().__init__()
        self.fc1 = GCNConv(dim_in, hidden_dim)
        self.fc_average = GCNConv(hidden_dim, dim_out)
        self.fc_log_variance = GCNConv(hidden_dim, dim_out)

    def forward(self, x, edge_index):
        x = self.fc1(x, edge_index).relu()
        return self.fc_average(x, edge_index), self.fc_log_variance(x, edge_index)

model = VGAE(Encoder(dataset.num_features, 32, 16)).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```



Paper #2 (VGAE)

- Code (Success Case)

```python
def train():
    model.train()
    optimizer.zero_grad()
    z = model.encode(train_data.x, train_data.edge_index)
    loss = model.recon_loss(z, train_data.pos_edge_label_index) + (1 / train_data.num_nodes) * model.kl_loss()
    loss.backward()
    optimizer.step()
    return float(loss)

@torch.no_grad()
def test(data):
    model.eval()
    z = model.encode(data.x, data.edge_index)
    return model.test(z, data.pos_edge_label_index, data.neg_edge_label_index)


for epoch in range(201):
    loss = train()
    test_auc, test_ap = test(test_data)
    if epoch % 50 == 0:
        print('Train loss : {}'.format(loss))
        print('Test AUC : {} | Test AP : {}'.format(test_auc, test_ap))
```

```python
def recon_loss(self, z: Tensor, pos_edge_index: Tensor,                    [docs]
               neg_edge_index: Optional[Tensor] = None) -> Tensor:
    r"""Given latent variables :obj:`z`, computes the binary cross
    entropy loss for positive edges :obj:`pos_edge_index` and negative
    sampled edges.

    Args:
        z (torch.Tensor): The latent space :math:`\mathbf{Z}`.
        pos_edge_index (torch.Tensor): The positive edges to train against.
        neg_edge_index (torch.Tensor, optional): The negative edges to
            train against. If not given, uses negative sampling to
            calculate negative edges. (default: :obj:`None`)
    """
    pos_loss = -torch.log(
        self.decoder(z, pos_edge_index, sigmoid=True) + EPS).mean()

    if neg_edge_index is None:
        neg_edge_index = negative_sampling(pos_edge_index, z.size(0))
    neg_loss = -torch.log(1 -
                          self.decoder(z, neg_edge_index, sigmoid=True) +
                          EPS).mean()

    return pos_loss + neg_loss
```

```python
def test(self, z: Tensor, pos_edge_index: Tensor,                          [docs]
         neg_edge_index: Tensor) -> Tuple[Tensor, Tensor]:
    r"""Given latent variables :obj:`z`, positive edges
    :obj:`pos_edge_index` and negative edges :obj:`neg_edge_index`,
    computes area under the ROC curve (AUC) and average precision (AP)
    scores.

    Args:
        z (torch.Tensor): The latent space :math:`\mathbf{Z}`.
        pos_edge_index (torch.Tensor): The positive edges to evaluate
            against.
        neg_edge_index (torch.Tensor): The negative edges to evaluate
            against.
    """
    from sklearn.metrics import average_precision_score, roc_auc_score

    pos_y = z.new_ones(pos_edge_index.size(1))
    neg_y = z.new_zeros(neg_edge_index.size(1))
    y = torch.cat([pos_y, neg_y], dim=0)

    pos_pred = self.decoder(z, pos_edge_index, sigmoid=True)
    neg_pred = self.decoder(z, neg_edge_index, sigmoid=True)
    pred = torch.cat([pos_pred, neg_pred], dim=0)

    y, pred = y.detach().cpu().numpy(), pred.detach().cpu().numpy()

    return roc_auc_score(y, pred), average_precision_score(y, pred)
```

Paper #2 (VGAE)

- Code (Success Case)

```python
def train():
    model.train()
    optimizer.zero_grad()
    z = model.encode(train_data.x, train_data.edge_index)
    loss = model.recon_loss(z, train_data.pos_edge_label_index) + (1 / train_data.num_nodes) * model.kl_loss()
    loss.backward()
    optimizer.step()
    return float(loss)

@torch.no_grad()
def test(data):
    model.eval()
    z = model.encode(data.x, data.edge_index)
    return model.test(z, data.pos_edge_label_index, data.neg_edge_label_index)


for epoch in range(201):
    loss = train()
    test_auc, test_ap = test(test_data)
    if epoch % 50 == 0:
        print('Train loss : {}'.format(loss))
        print('Test AUC : {} | Test AP : {}'.format(test_auc, test_ap))
```

```
Train loss : 3.5164217948913574
Test AUC : 0.6655624727702184 | Test AP : 0.69503406826326
Train loss : 1.3143773078918457
Test AUC : 0.6430261153858617 | Test AP : 0.6786212637061367
Train loss : 1.146220326423645
Test AUC : 0.7295637113877196 | Test AP : 0.7447826307149207
Train loss : 1.0693825483322144
Test AUC : 0.7687889993482855 | Test AP : 0.7783319383978209
Train loss : 0.9703818559646606
Test AUC : 0.8702080085262971 | Test AP : 0.8629827566623429
```

```
(0.8641732568057945, 0.8772858317978459)
```

Takeaway

- Good
  - 전반적인 흐름 이해 (Bakckground, Methodology, …)
  - VAE 재현성 확인

- Weak
  - Link prediction 실험 설계 이해 부족
  - Failure Case 성공적으로 돌려보기

- Takeaway
  - Theoretical background의 중요성 (Directed graphical model, ELBO, …)
  - Library 패키지를 사용한 공부
  - GNN 흥미 & Use case..?

Thank you for watching!

- 코드 깃허브 업로드 완료
  - VAE / VGAE_failure / VGAE_success