Variational Autoencoder (VAE)
& Graph Variational Autoencoder(GVAE)

February 7, 2023

Jihwan Bae, Korea University

Contents

Introduction

Variational Inference + AutoEncoder = VAE

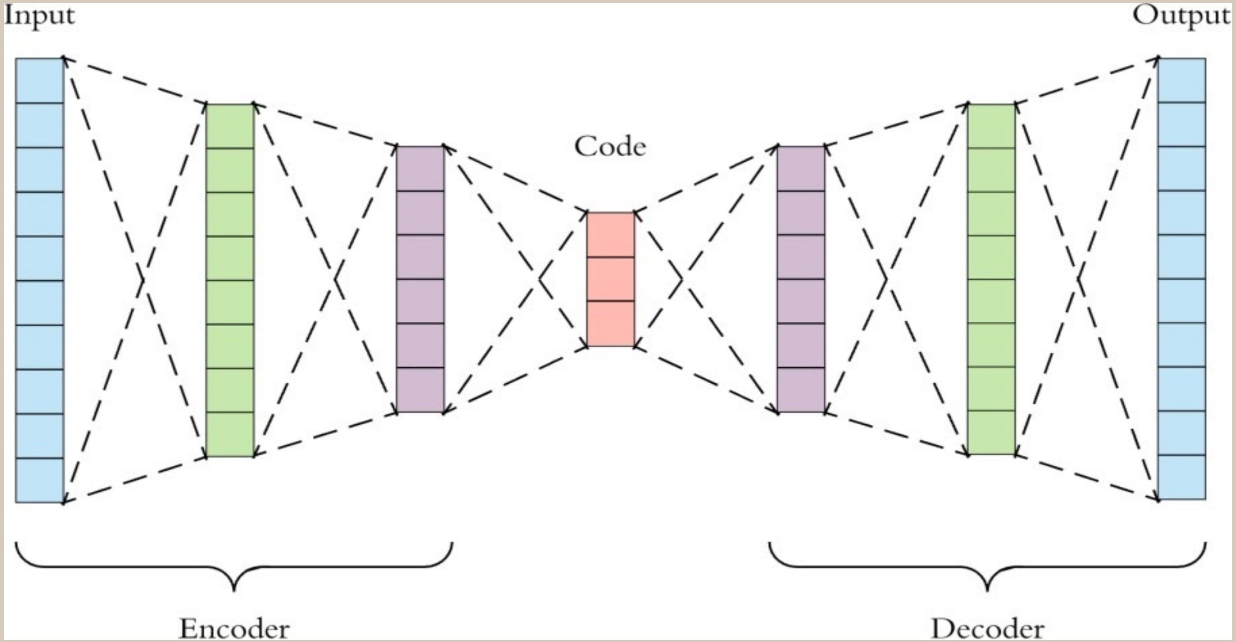
Variational Inference : Approximating target distribution with a simpler distribution

ex) 사후확률(posterior) 분포 $p(z\,|x)$ --> $q(z)$ ~ Normal

Introduction

Variational Inference + AutoEncoder = VAE

AutoEncoder : "The aim of an **autoencoder** is to learn a **representation (encoding) for a set of data** "

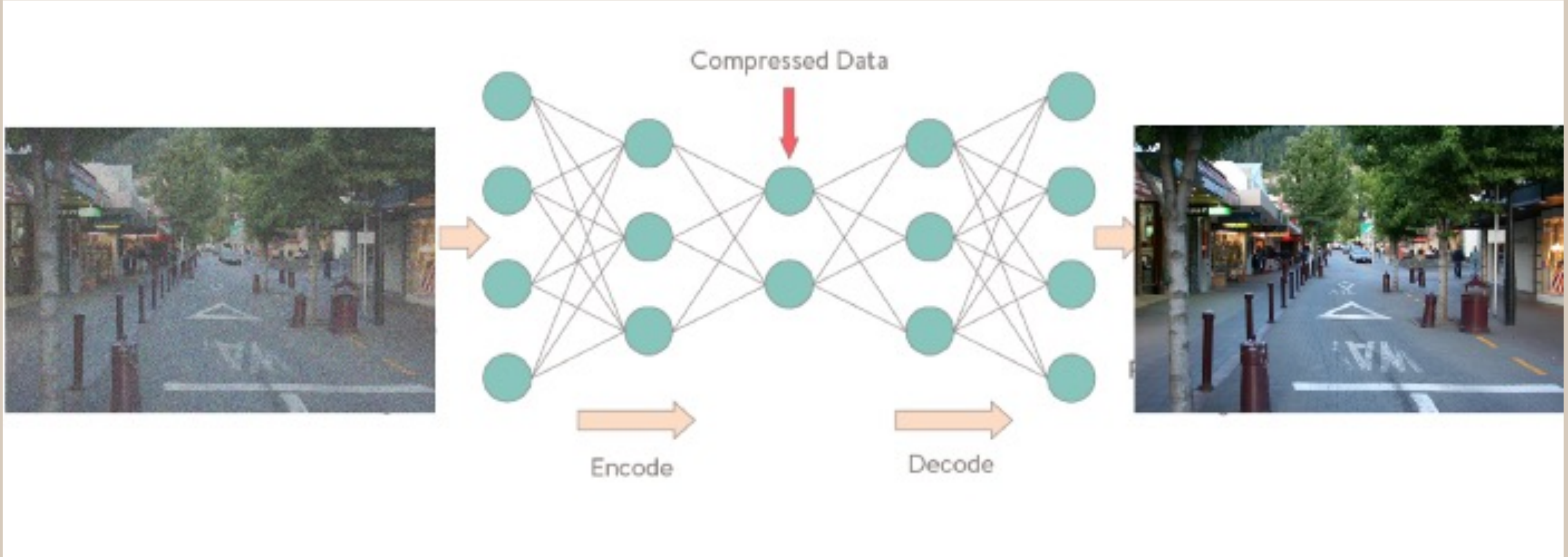


AutoEncoder (AE)

Introduction

Variational Inference + AutoEncoder = VAE

AutoEncoder : "The aim of an **autoencoder** is to learn a **representation (encoding) for a set of data** "

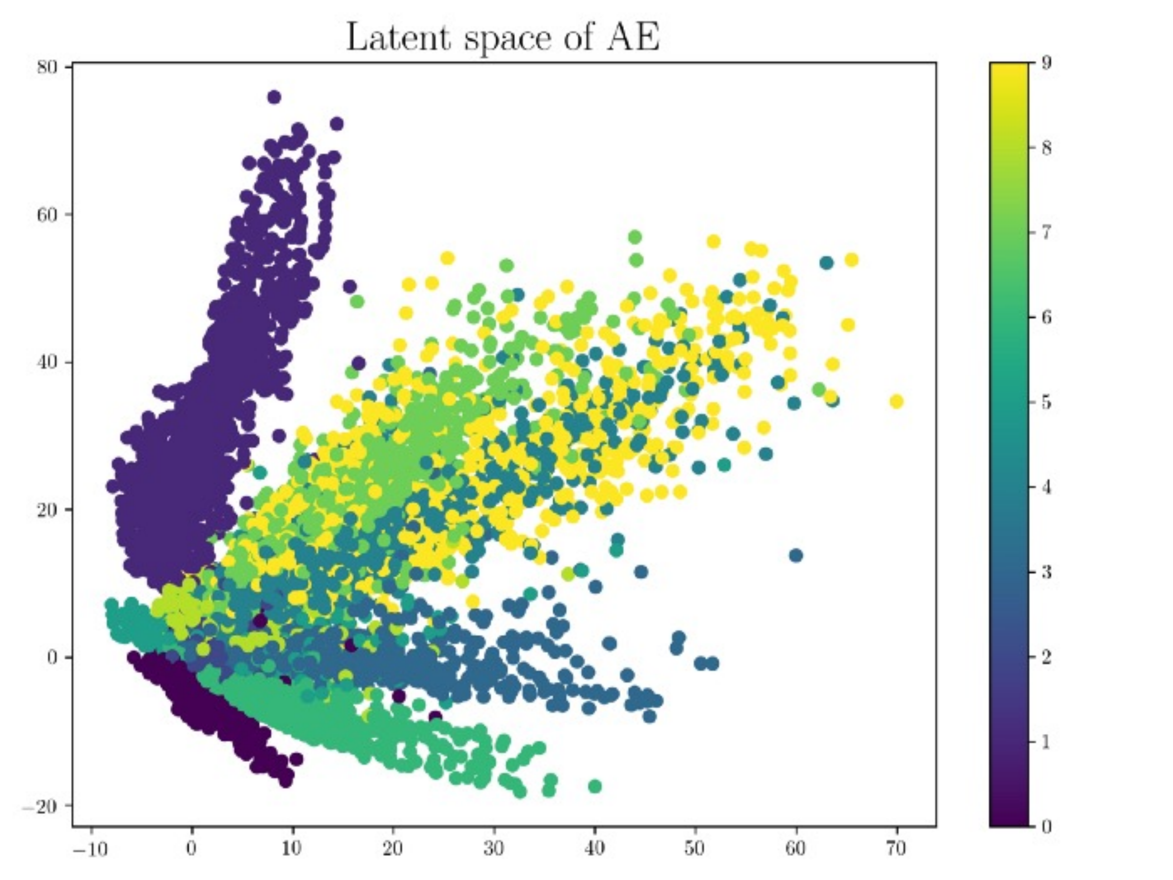


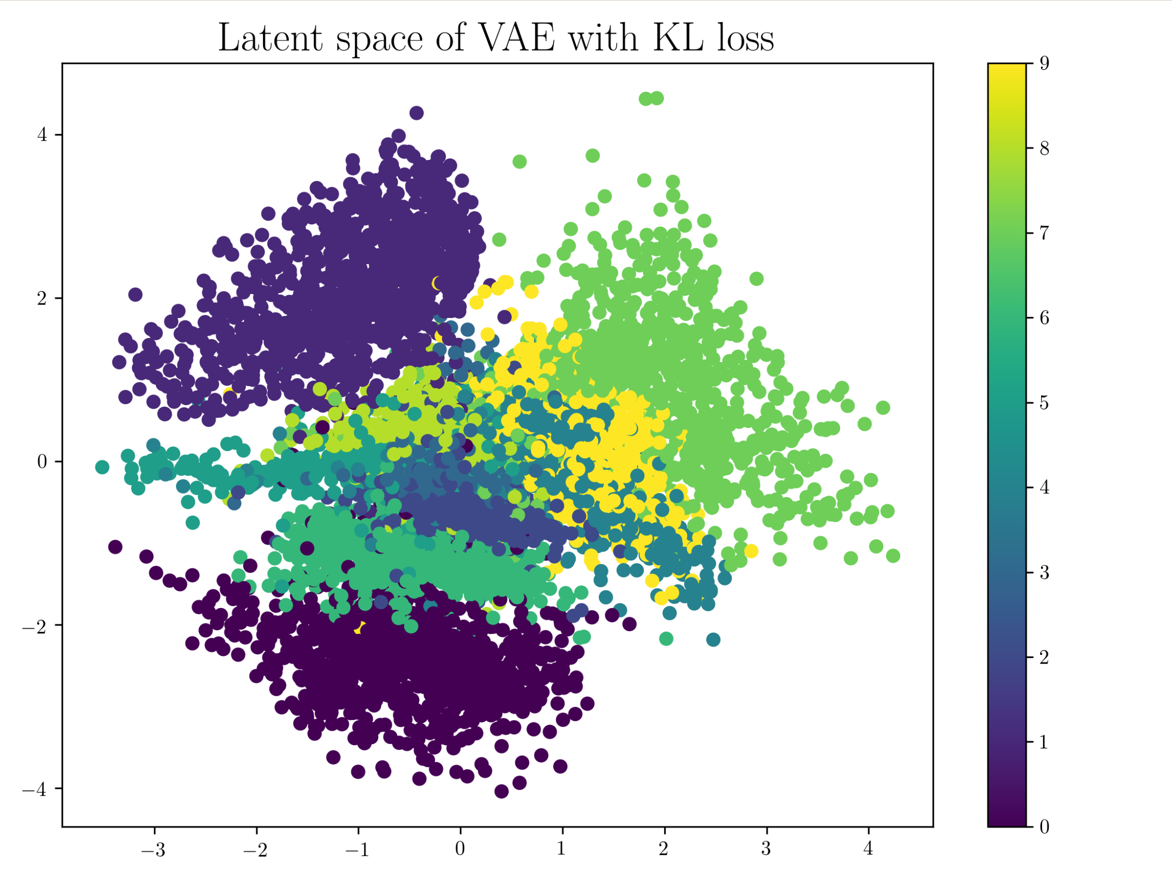
Common Applications:

1. Anomaly Detection
2. Image Denoising

Introduction

Why VAE?





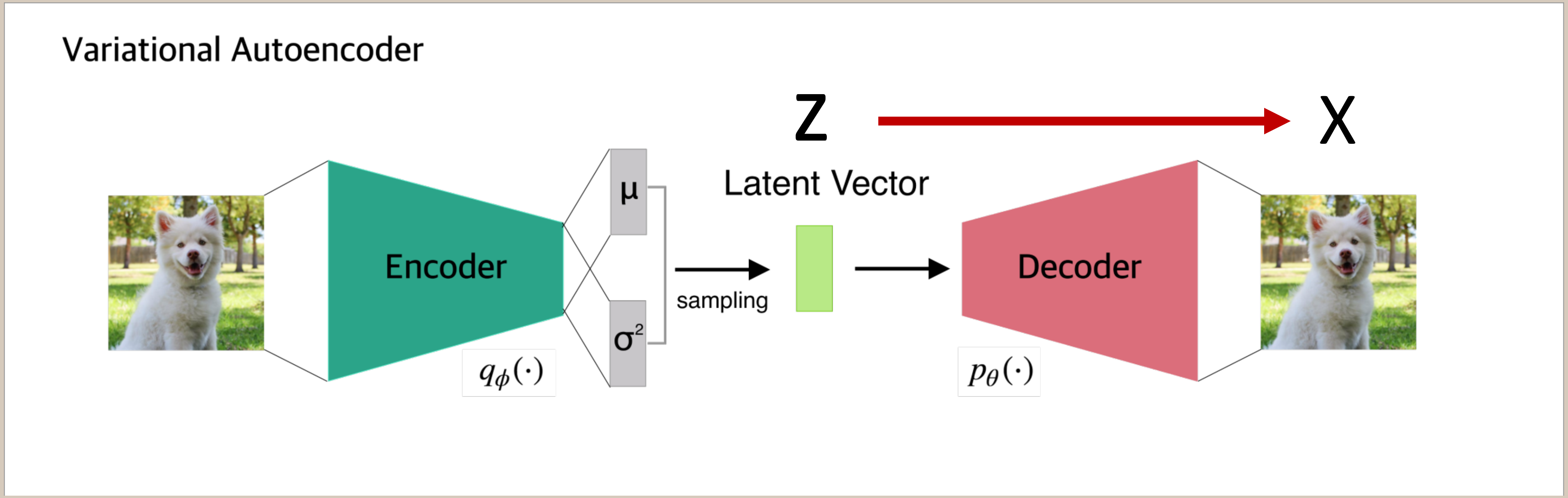
Introduction

(1) Encoder

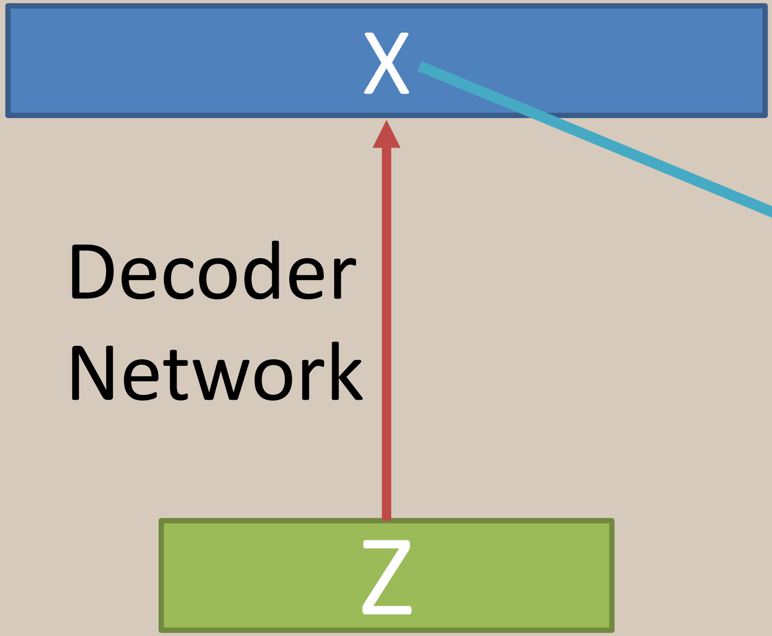
- inference network
- input : x, output : z
- $q(z \mid x, \phi)$

(2) Decoder

- generative network
- input : z, output : as closely as x
- $p(x \mid z, \theta)$



VAE



Given I.I.D data, we want to learn model parameters to maximize likelihood of training data

$X_1, X_2, \ldots, X_n$

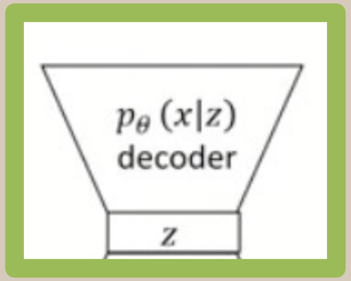
$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, d\mathbf{z}$$

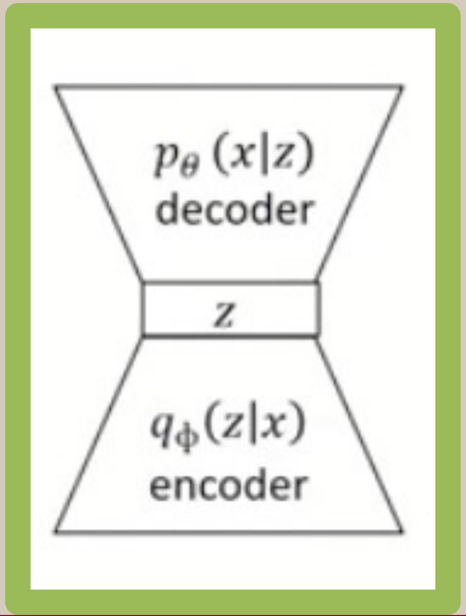
VAE

Data Likelihood : $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z})\, d\mathbf{z}$

<u>Intractable to compute for every Z</u>

Define additional encoder network q(z|x) that approximate p(x|z)





VAE

$$
\begin{aligned}
\log p_\theta(\mathbf{x}) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})} \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z} - KL(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z})) + KL(q_\phi(\mathbf{z}|\mathbf{x}) || p_\theta(\mathbf{z}|\mathbf{x}))
\end{aligned}
$$

$$
D_{KL}(P||Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx
$$

- $KL(q\|\, p) \geq 0$
- $KL(q\|\, p) = 0 \;\Leftrightarrow\; q = p$

VAE

$$= \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z} - KL(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) + \underbrace{KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))}_{\text{(Non-Negative)}}$$

$$\geq \underbrace{\int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}}_{\text{Reconstruction error}} - \underbrace{KL(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{\text{Regularization error}} \longleftarrow \text{Evidence lower bound (ELBO)}$$

Maximizing Evidence Lower Bound(ELBO)

probability a hypothesis is true given the evidence

probability a hypothesis is true (before any evidence is present)

probability of seeing the evidence if the hypothesis is true

$$P(H/E) = \frac{P(H)\,P(E/H)}{P(E)}$$

probability of observing the evidence

VAE

$$\arg\min_{\theta,\phi}\sum_i \overset{-\text{ ELBO}}{\boxed{\underbrace{-\mathbb{E}_{q_\phi(z|x_i)}\left[log\left(p(x_i|g_\theta(z))\right)\right]}_{\text{Reconstruction Error}} + \underbrace{KL(q_\phi(z|x_i)||p(z))}_{\text{Regularization}}}}$$

VAE

$$\arg\min_{\theta,\phi} \sum_i -\mathbb{E}_{q_\phi(z|x_i)}\left[log\left(p\left(x_i|g_\theta(z)\right)\right)\right] + KL\left(q_\phi(z|x_i)||p(z)\right)$$

$$\mathbb{E}_{q_\phi(z|x_i)}\left[\log\left(p_\theta(x_i|z)\right)\right] = \int \log\left(p_\theta(x_i|z)\right) q_\phi(z|x_i)dz \approx \frac{1}{L}\sum_{z^{i,l}} \log\left(p_\theta\left(x_i|z^{i,l}\right)\right)$$

VAE

$$\mathbb{E}_{q_\phi(z|x_i)}[\log(p_\theta(x_i|z))] = \int \log(p_\theta(x_i|z))q_\phi(z|x_i)dz \approx \frac{1}{L}\sum_{z^{i,l}} \log\left(p_\theta(x_i|z^{i,l})\right) \approx \log\left(p_\theta\left(x_i|z^i\right)\right)$$

$$p_\theta\left(x_i|z^i\right) \sim Bernoulli(p_i)$$

$$= \sum_{j=1}^{D} log\, p_{i,j}^{x_{i,j}} (1 - p_{i,j})^{1-x_{i,j}} \longleftarrow p_{i,j}\text{: network output}$$

$$= \sum_{j=1}^{D} x_{i,j} log\, p_{i,j} + (1 - x_{i,j}) \log(1 - p_{i,j})$$

Cross entropy

VAE

$$\log\left(p_\theta(x_i|z^i)\right) = \log(N(x_i; \mu_i, \sigma_i^2 I))$$

$$= -\sum_{j=1}^{D} \frac{1}{2}\log(\sigma_{i,j}^2) + \frac{(x_{i,j} - \mu_{i,j})^2}{2\sigma_{i,j}^2}$$

For gaussain distribution with identity covariance

$$\log\left(p_\theta(x_i|z^i)\right) \propto -\sum_{j=1}^{D} (x_{i,j} - \mu_{i,j})^2$$

Squared Error

VAE

Optimization Assumptions

$$q_\phi(z|x_i) \sim N(\mu_i, \sigma_i^2 I)$$ $$p(z) \sim N(0, I)$$

$$\arg\min_{\theta,\phi} \sum_i -\mathbb{E}_{q_\phi(z|x_i)}\left[\log\left(p(x_i|g_\theta(z))\right)\right] + \boxed{KL(q_\phi(z|x_i)||p(z))}$$

Regularization

VAE

Theorem: Let x be an $n \times 1$ random vector. Assume two multivariate normal distributions P and Q specifying the probability distribution of x as

$$\begin{aligned} P &: \; x \sim \mathcal{N}(\mu_1, \Sigma_1) \\ Q &: \; x \sim \mathcal{N}(\mu_2, \Sigma_2) \; . \end{aligned} \tag{1}$$

Then, the Kullback-Leibler divergence of P from Q is given by

$$\mathrm{KL}[P \,||\, Q] = \frac{1}{2} \left[(\mu_2 - \mu_1)^T \Sigma_2^{-1} (\mu_2 - \mu_1) + \mathrm{tr}(\Sigma_2^{-1} \Sigma_1) - \ln \frac{|\Sigma_1|}{|\Sigma_2|} - n \right] \; . \tag{2}$$

VAE

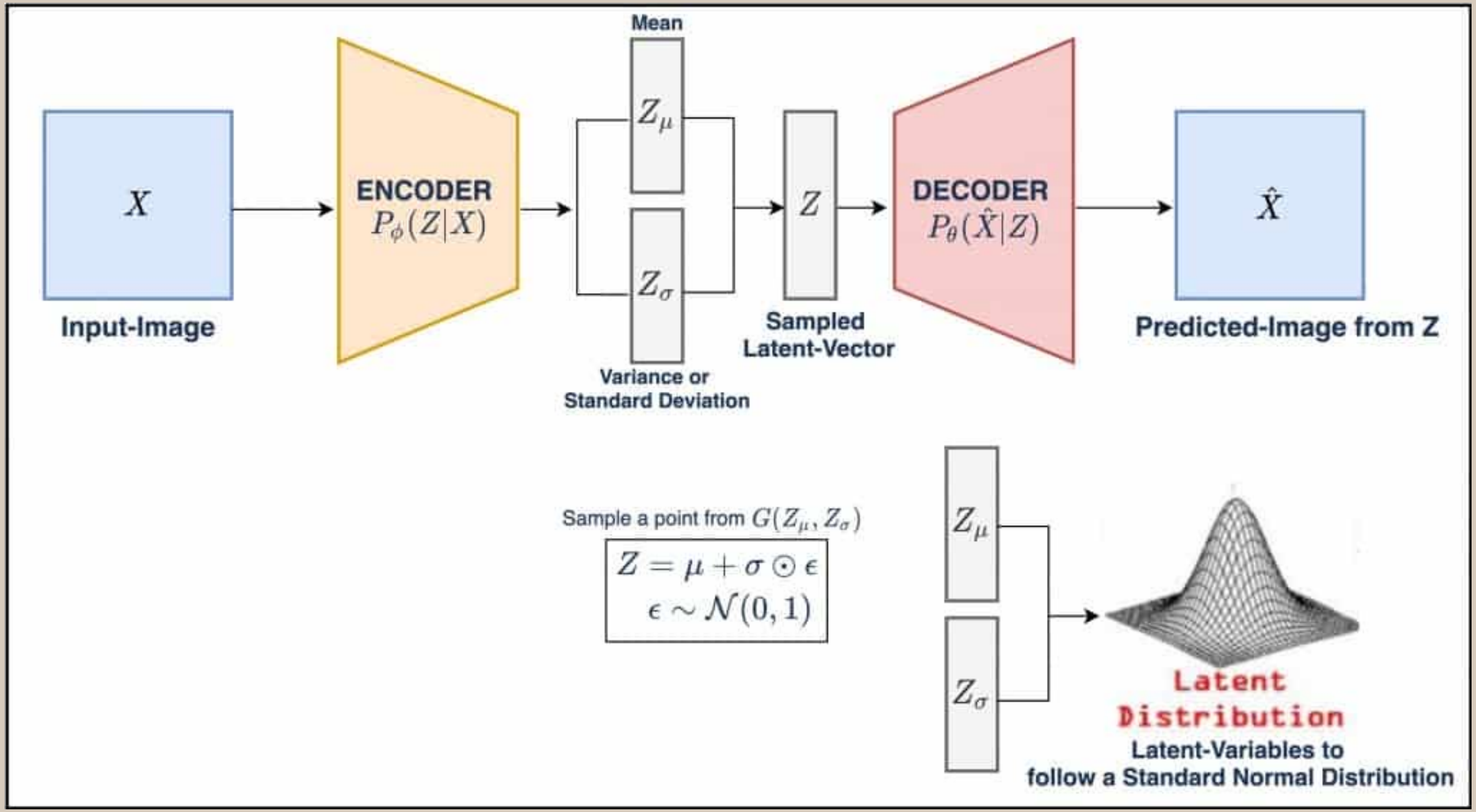
$$KL(q_\phi(z|x_i)||p(z)) = \frac{1}{2}\left\{tr(\sigma_i^2 I) + \mu_i^T \mu_i - J + ln\frac{1}{\prod_{j=1}^{J}\sigma_{i,j}^2}\right\}$$

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

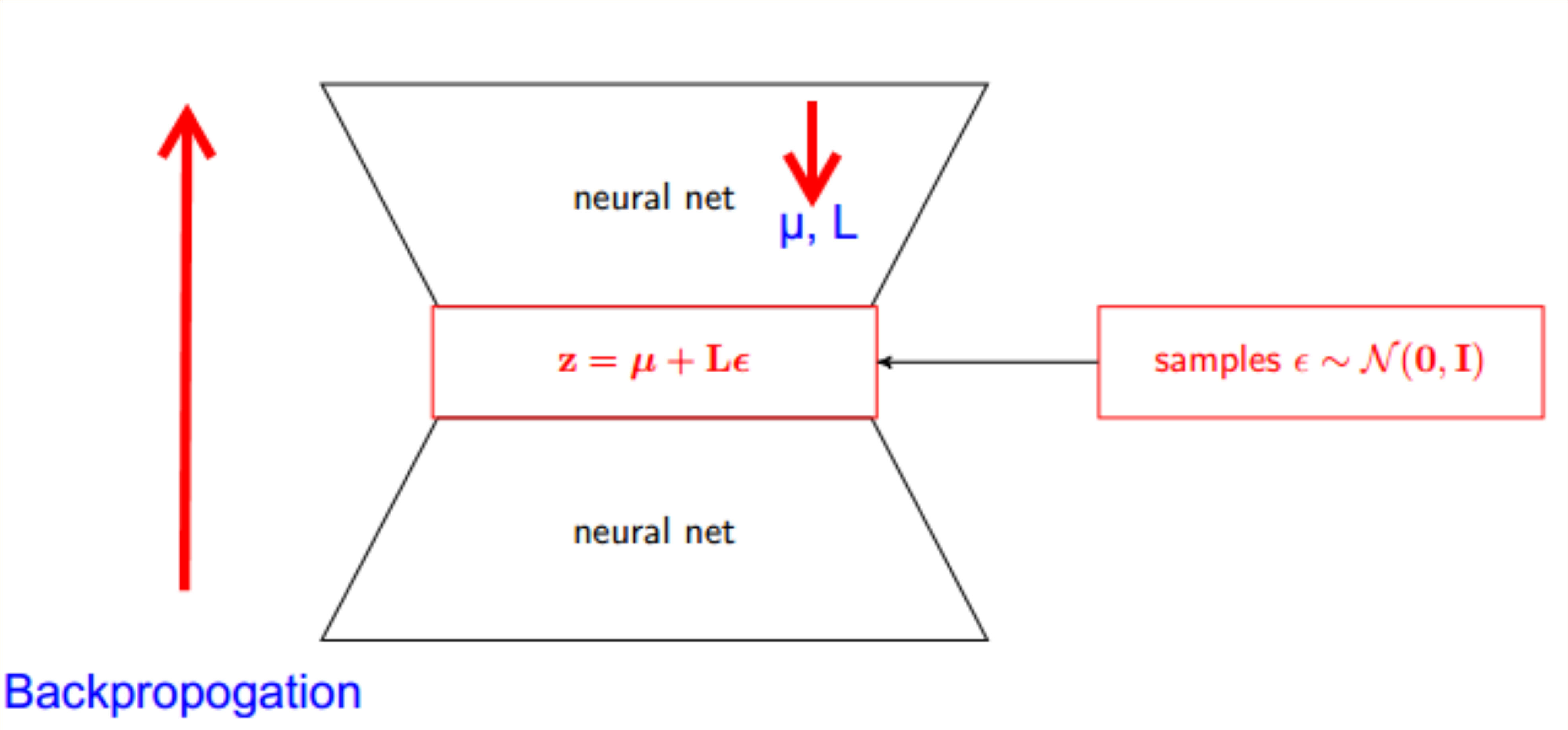
$$= \frac{1}{2}\sum_{j=1}^{J}(\mu_{i,j}^2 + \sigma_{i,j}^2 - ln(\sigma_{i,j}^2) - 1)$$

VAE

VAE Structure



VAE


neural net
μ, L
$\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$
samples $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
neural net
Backpropogation

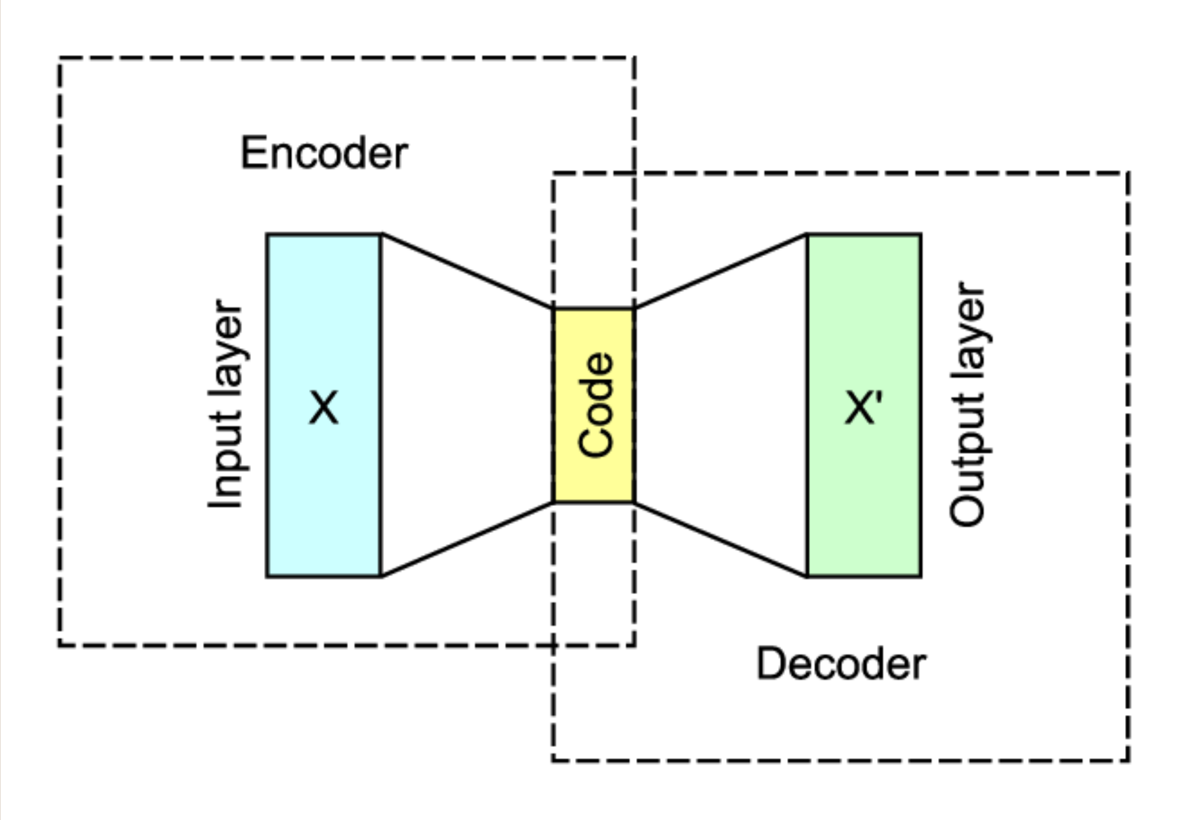
VAE

$$L_i(\phi, \theta, x_i) = \underbrace{-\mathbb{E}_{q_\phi(z|x_i)}\left[\log\left(p_\theta(x_i|z)\right)\right]}_{\text{Reconstruction Error}} + \underbrace{KL\left(q_\phi(z|x_i)\,||\,p(z)\right)}_{\text{Regularization}}$$

$$\underbrace{\log\left(p_\theta(x_i|z^l)\right) = \sum_{j=1}^{D} x_{i,j} \log p_{i,j} + (1 - x_{i,j}) \log(1 - p_{i,j})}_{\text{Cross entropy}}$$

$$KL(q_\phi(z|x_i)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(\mu_{i,j}^2 + \sigma_{i,j}^2 - \ln(\sigma_{i,j}^2) - 1)$$

GVAE



GVAE

First Neural Net of Encoder

$$\tilde{A} = D^{-1/2} A D^{-1/2}$$

$$\bar{X} = GCN(A, X) = ReLU(\tilde{A} X W_0)$$

GVAE

First Neural Net of Encoder

$$\tilde{A} = D^{-1/2} A D^{-1/2}$$

$$\bar{X} = GCN(A, X) = ReLU(\tilde{A} X W_0)$$

Second Neural Net of Encoder

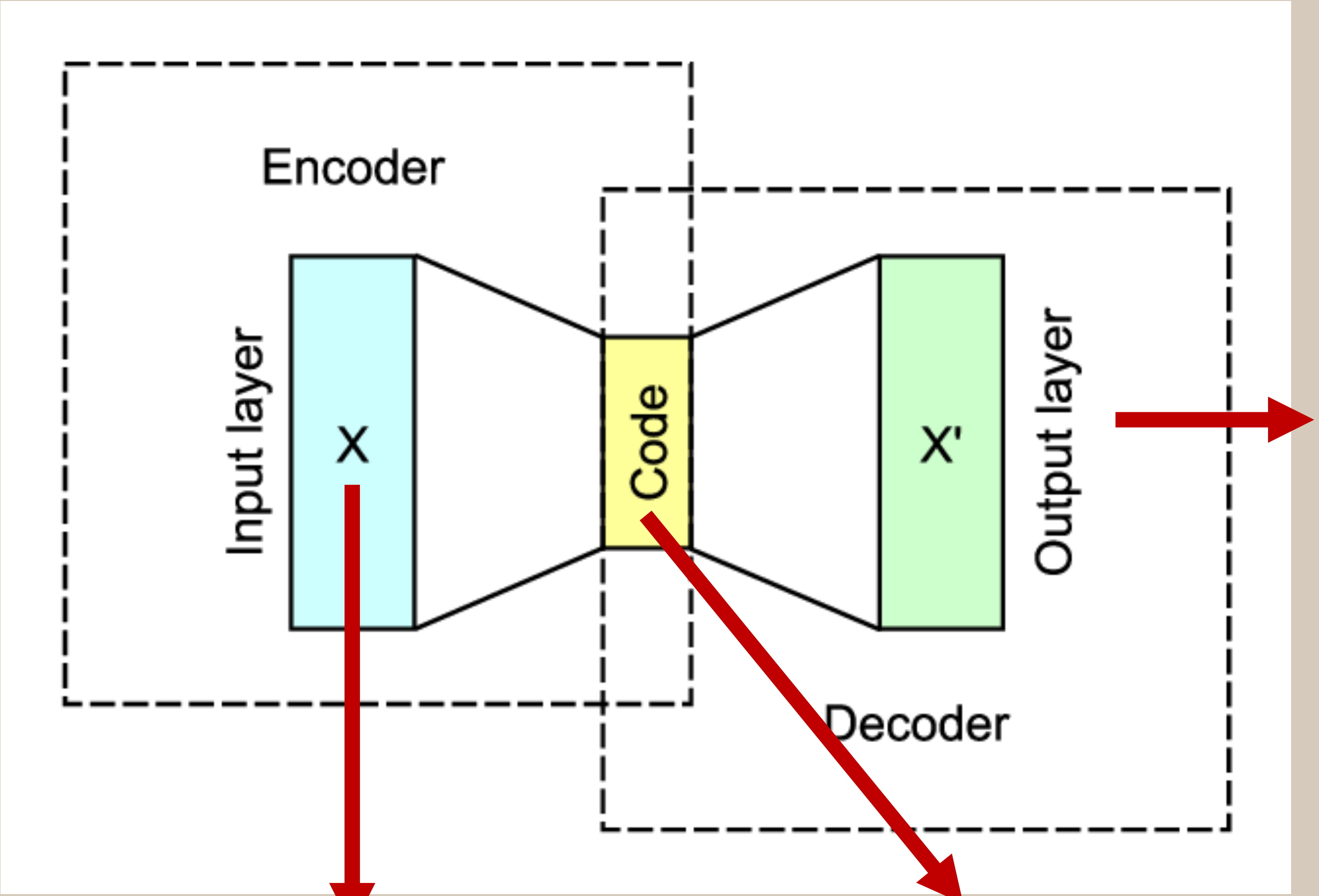
$$\mu = GCN_{\mu}(X, A) = \tilde{A} \bar{X} W_1$$

$$\log \sigma^2 = GCN_{\sigma}(X, A) = \tilde{A} \bar{X} W_1$$

$$GCN(A, X) = \tilde{A} ReLU(\tilde{A} X W_0) W_1$$

with $\tilde{A} = D^{-1/2} A D^{-1/2}$

GVAE



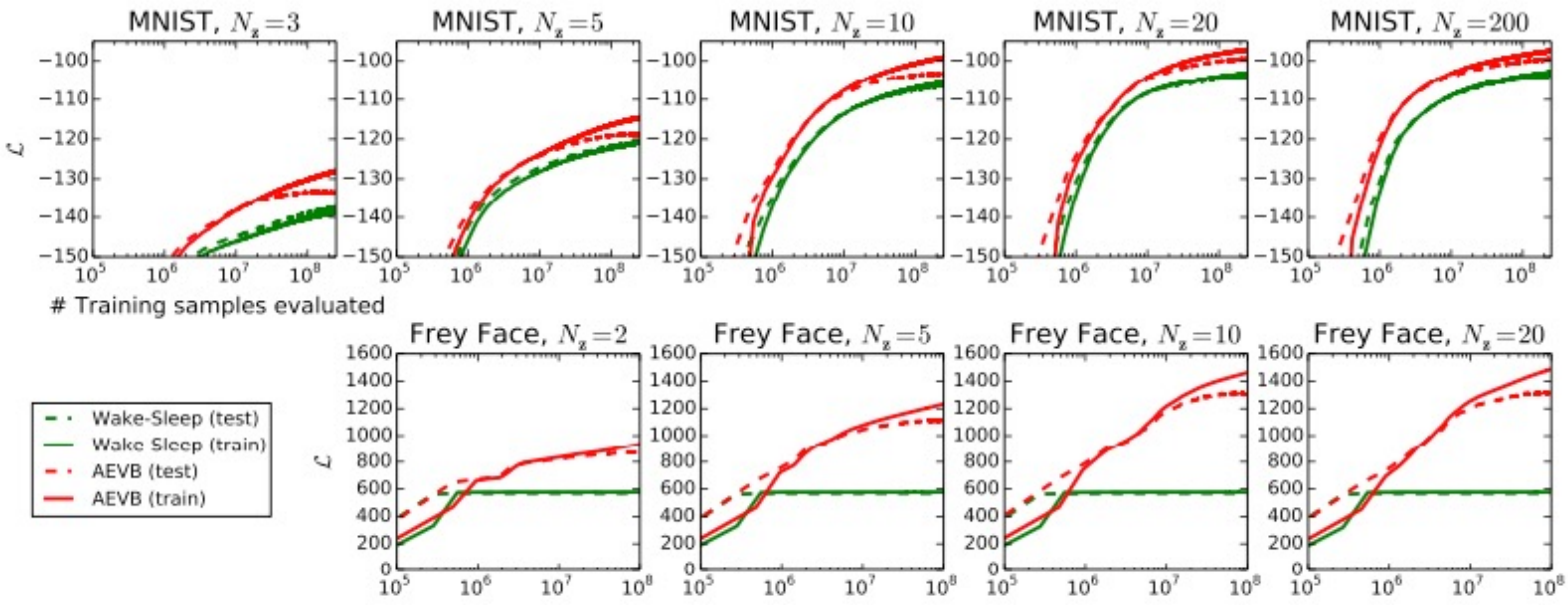
Inner product between latent variable Z

$$\hat{A} = \text{logistic sigmoid}\,(zz^T)$$

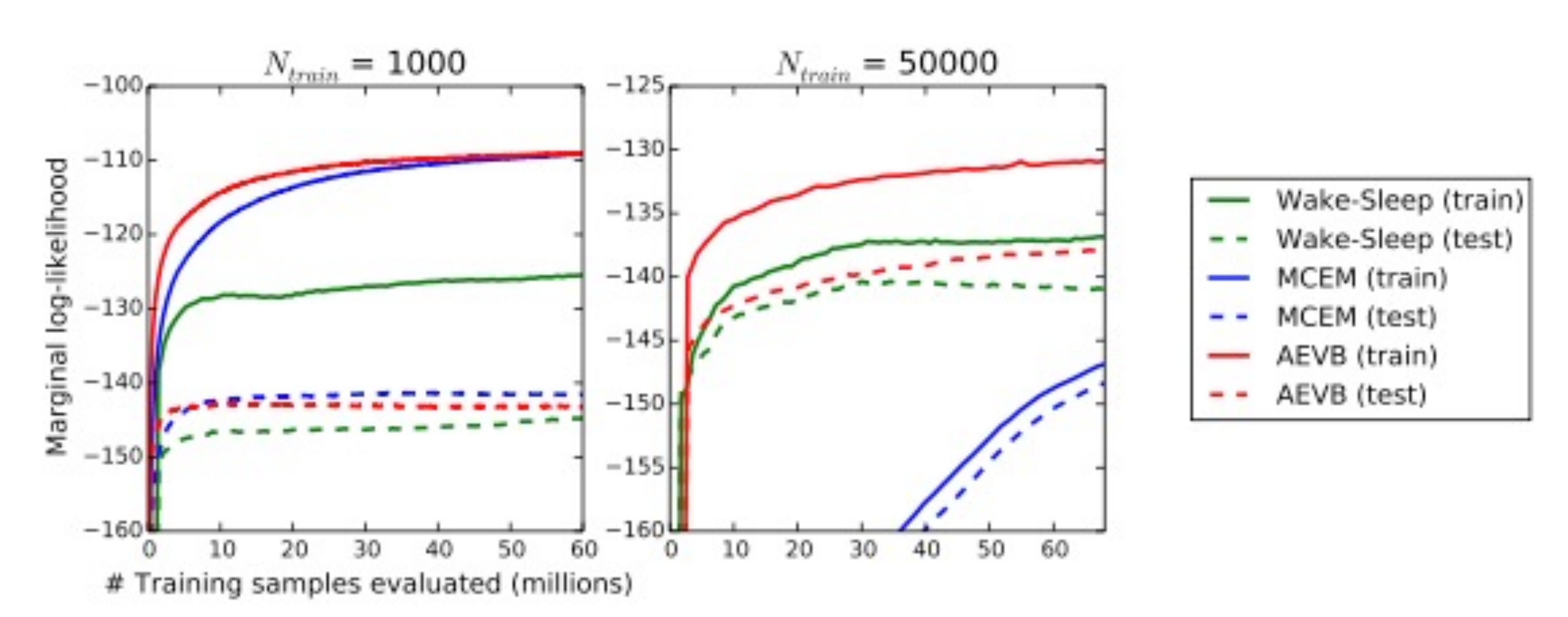
$$GCN(A, X) = \tilde{A}ReLU(\tilde{A}XW_0)W_1$$

$$Z = \mu + \sigma \odot \epsilon$$
$$\epsilon \sim Norm(0, 1)$$

Experiment


MNIST, $N_z=3$
MNIST, $N_z=5$
MNIST, $N_z=10$
MNIST, $N_z=20$
MNIST, $N_z=200$
$\mathcal{L}$
Training samples evaluated
Frey Face, $N_z=2$
Frey Face, $N_z=5$
Frey Face, $N_z=10$
Frey Face, $N_z=20$
Wake-Sleep (test)
Wake-Sleep (train)
AEVB (test)
AEVB (train)

Experiment

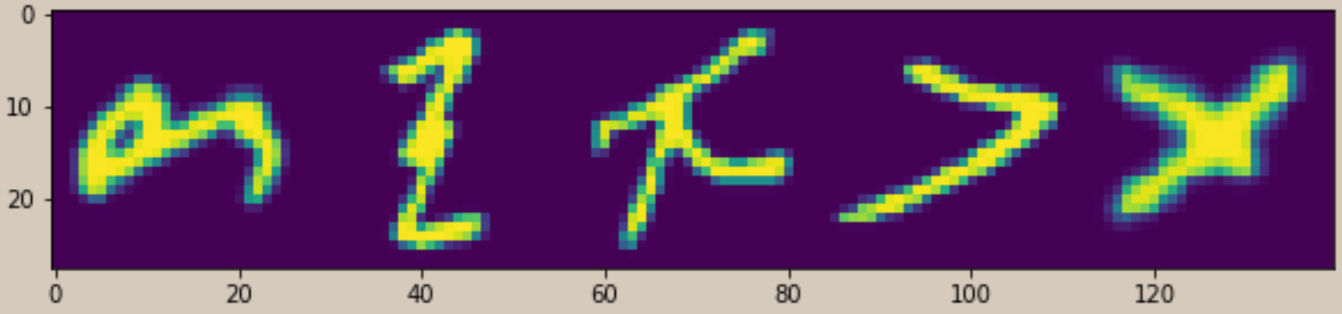


N_{train} = 1000
N_{train} = 50000
Marginal log-likelihood
Training samples evaluated (millions)
−100
−110
−120
−130
−140
−150
−160
−125
−130
−135
−140
−145
−150
−155
−160
0
10
20
30
40
50
60
Wake-Sleep (train)
Wake-Sleep (test)
MCEM (train)
MCEM (test)
AEVB (train)
AEVB (test)

Implementation – VAE

```python
for i in train_loader:
  plt.figure(figsize=(10,10))
  k = None
  for img in i[0][5:10]:
    img = img.permute(1, 2, 0)
    img = img.squeeze()
    if k == None:
      k = img
      print(k.shape)
    else:
      k = torch.cat((k,img), 1)
  plt.imshow(k)
  break
plt.show()
```

```
torch.Size([28, 28])
```



Implementation – VAE

```
class VAE(nn.Module):
    def __init__(self, input_dim, hid_dim1, hid_dim2, emb_dim):
        super(VAE, self).__init__()

        self.input_dim = input_dim

        # encoder part
        self.fc1 = nn.Linear(input_dim, hid_dim1)
        self.fc2 = nn.Linear(hid_dim1, hid_dim2)
        self.fc3_mean = nn.Linear(hid_dim2, emb_dim)
        self.fc3_var = nn.Linear(hid_dim2, emb_dim)
        # decoder part
        self.fc4 = nn.Linear(emb_dim, hid_dim2)
        self.fc5 = nn.Linear(hid_dim2, hid_dim1)
        self.fc6 = nn.Linear(hid_dim1, input_dim)

```

Implementation – VAE

```python
    def forward(self, x):

        #encoding
        temp = F.relu(self.fc1(x.view(-1, 784)))
        temp = F.relu(self.fc2(temp))
        mu = self.fc3_mean(temp)
        log_var = self.fc3_var(temp)

        #sampling
        std = torch.exp(0.5*log_var) # This is to constrain the variance to be positive (σ2∈R+)
                                     # But why learn log_var instead of learning log_sigma,
                                     # which does not require 0.5 multiplication?
        eps = torch.randn_like(std)
        z = eps.mul(std).add_(mu)

        #decoding
        temp = F.relu(self.fc4(z))
        temp = F.relu(self.fc5(temp))
        return F.sigmoid(self.fc6(temp)), mu, log_var
```

Implementation – VAE

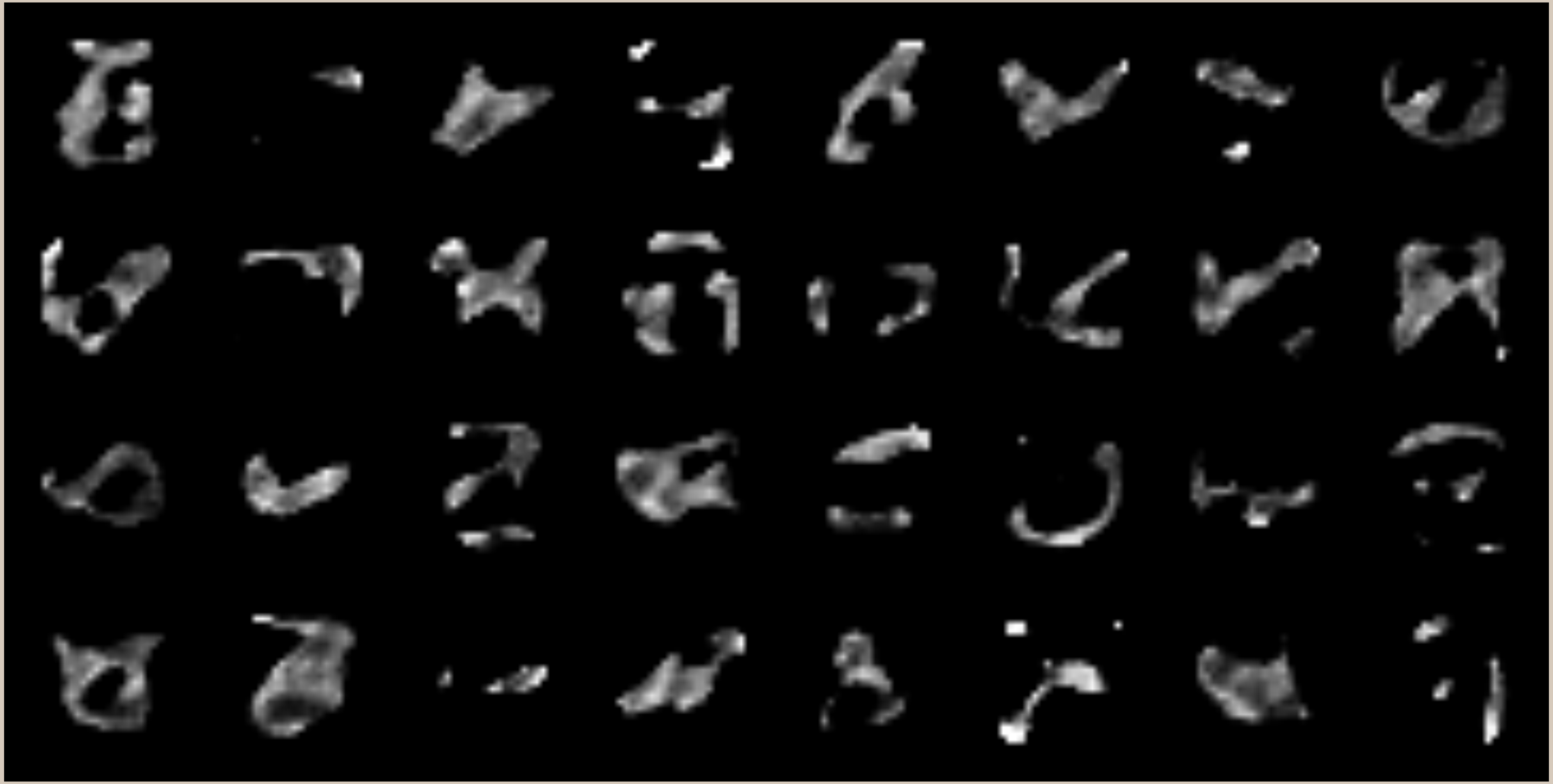
```
# build model
vae = VAE(784, 128, 32, 20)
if torch.cuda.is_available():
    vae.cuda()

optimizer = optim.Adam(vae.parameters())

```

```
# return reconstruction error + KL divergence losses
def loss_function(reconstructed_data, original_data, mu, log_var):
    Binary_CE = F.binary_cross_entropy(reconstructed_data, original_data.view(-1, 784), reduction='sum')
    KL_Divergence = -0.5 * torch.sum(1 + log_var - mu**2 - log_var.exp())
    return Binary_CE + KL_Divergence
```

Implementation – VAE



Implementation – GVAE

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np

class VGAE(nn.Module):
  def __init__(self, adj):
    super(VGAE,self).__init__()
    self.first_layer = GraphConv(input_dim, output_dim*2, adj, F.relu)
    self.mean = GraphConv(output_dim*2, output_dim, adj, activation=lambda x:x)
    self.logvar = GraphConv(output_dim*2, output_dim, adj, activation=lambda x:x)


  def forward(self, X):
    hidden = self.first_layer(X)
    self.mean = self.mean(hidden)
    self.logvar = self.logvar(hidden)
    epsilon = torch.randn(self.mean.size(0), output_dim)
    sampled_z = epsilon*torch.exp(self.logvar) + self.mean
    decoded = torch.sigmoid(torch.matmul(Z,Z.t()))
    return decoded
```

Implementation – GVAE

```
class GraphConv(nn.Module):
  def __init__(self, input_dim, output_dim, adj, activation, **kwargs):
    super(GraphConv, self).__init__(**kwargs)
    self.weight = torch.rand(input_dim, output_dim)
    self.adj = adj
    self.activation = activation

  def forward(self, inputs):
    X = inputs
    XW = torch.mm(x, self.weight)
    AXW = torch.mm(self.adj, XW)
    output = self.activation(AXW)
    return output
```

Conclusion

- VAE is a novel estimator of the variational lower bound, Stochastic Gradient VB (SGVB), for efficient approximate inference with continuous latent variables.
- Both GVAE and GAE achieve competitive results on the featureless task.
- Future work will investigate better-suited prior distributions, more flexible generative models and the application of a stochastic gradient descent algorithm for improved scalability.