Deep Learning

A course about theory & practice

Generative Networks:
Variational Auto-Encoders (VAE)

Marco Piastra

UNIVERSITÀ DI PAVIA

Auto-Encoders

Auto-Encoders

- **Auto-Encoder**

  Two main (composite) layers: **encoder** and **decoder**

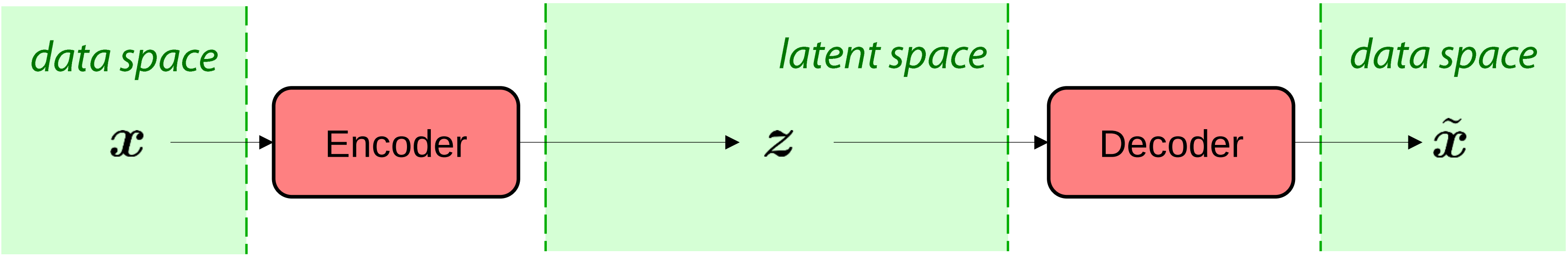
  One **hidden** or **latent** layer $\boldsymbol{z}$

  Each item in the dataset comprises the input only (*Unsupervised Learning*)

$$D := \{(\boldsymbol{x}^{(i)})\}_{i=1}^{N},$$

  The result of the optimization is $\boldsymbol{z}$:
  a compact (i.e. lower-dimensional) representation of the input $\boldsymbol{x}$

  This representation is also called the *latent space*

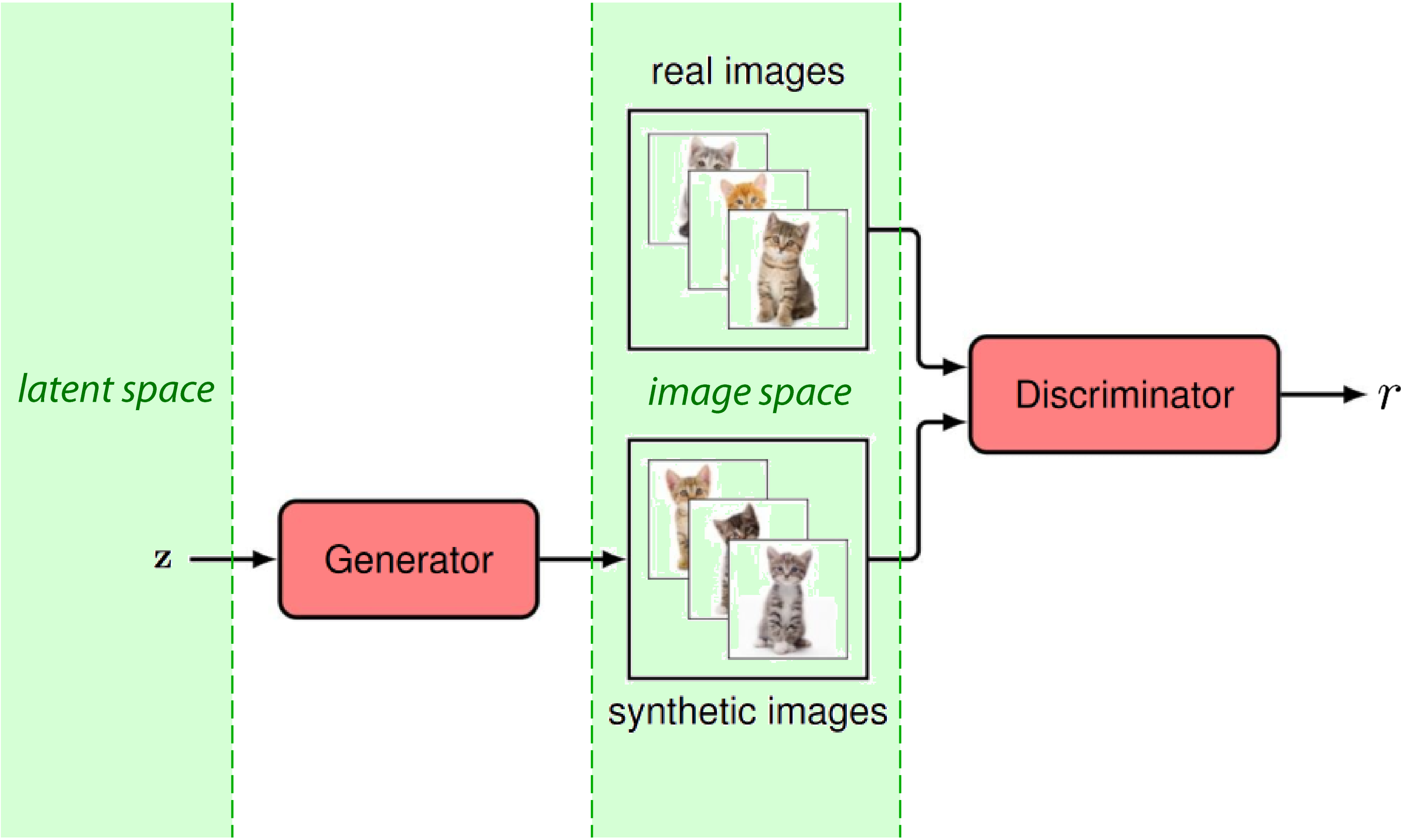


Generative Adversarial Networks

Basic idea: Decoder + Classifier

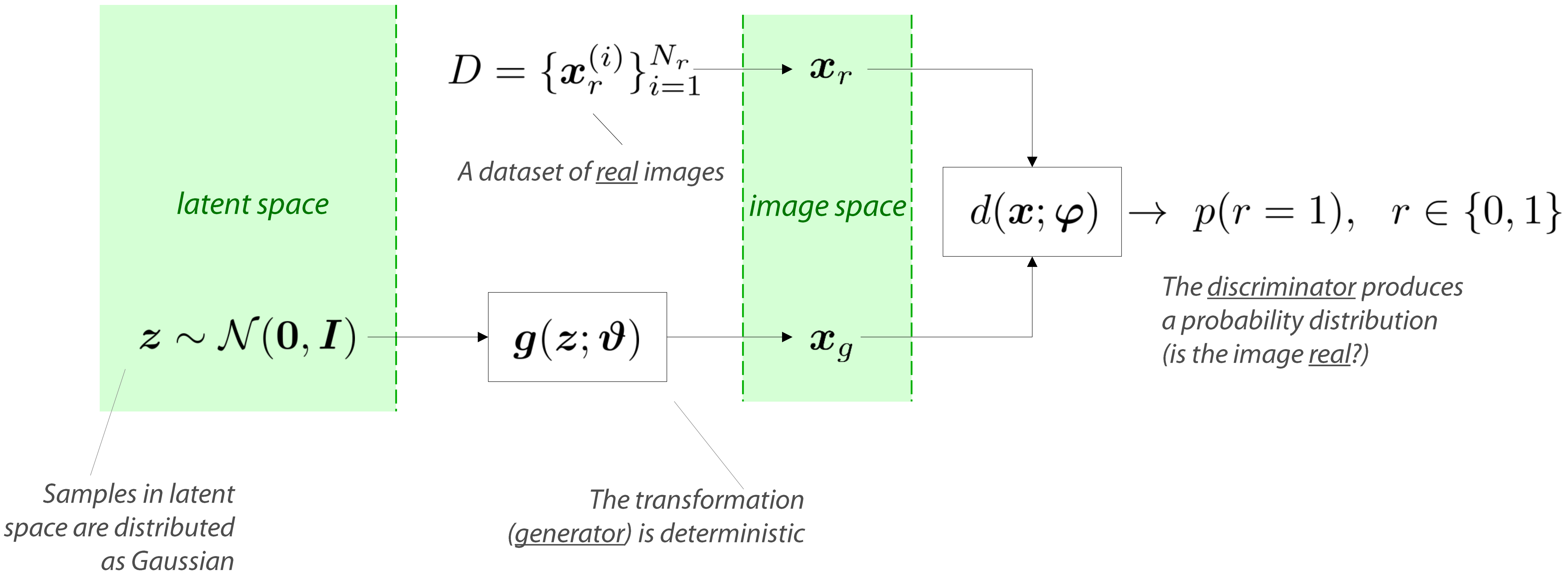
Objective:
creating a non-linear transformation from a latent space to a data space

Method:
training together a generator and a discriminator using a real dataset

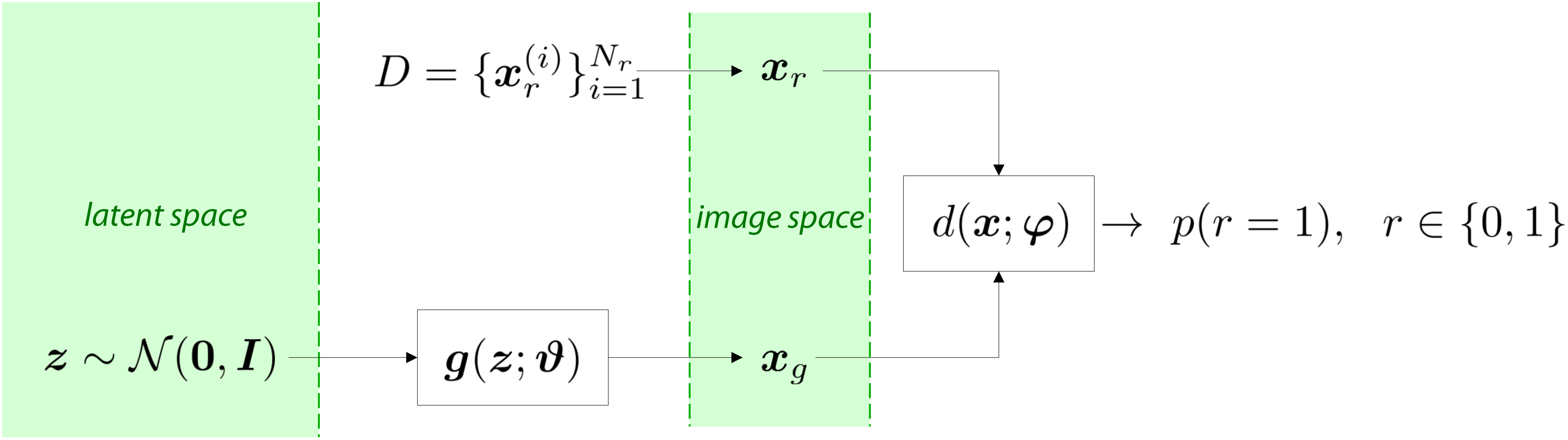


[Image from https://www.bishopbook.com/]

Generative Adversarial Network (GAN)



Generative Adversarial Network (GAN)



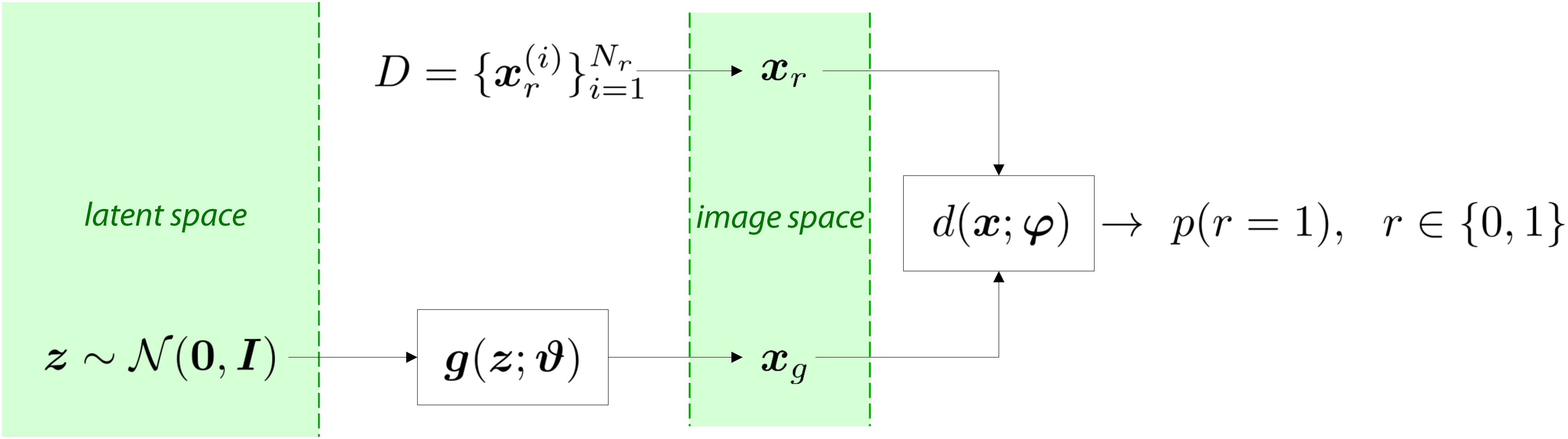
Loss function

$$L(\boldsymbol{\vartheta}, \boldsymbol{\varphi}) := -\frac{1}{N_r}\sum_{i \in \mathcal{R}} \ln(d(\boldsymbol{x}_r^{(i)}; \boldsymbol{\varphi})) - \frac{1}{N_g}\sum_{j \in \mathcal{G}} \ln(1 - d(\boldsymbol{g}(\boldsymbol{z}^{(j)}; \boldsymbol{\vartheta}); \boldsymbol{\varphi}))$$

Cross-entropy
(d should recognize real images)

Cross-entropy
(d should recognize 'false' images)

Generative Adversarial Network (GAN)



Loss function

$$L(\boldsymbol{\vartheta},\boldsymbol{\varphi}) := -\frac{1}{N_r}\sum_{i\in\mathcal{R}}\ln(d(\boldsymbol{x}_r^{(i)};\boldsymbol{\varphi})) - \frac{1}{N_g}\sum_{j\in\mathcal{G}}\ln(1-d(\boldsymbol{g}(\boldsymbol{z}^{(j)};\boldsymbol{\vartheta});\boldsymbol{\varphi}))$$

Gradients

$$\Delta\boldsymbol{\varphi} = -\,\eta\frac{\partial}{\partial\boldsymbol{\varphi}}L(\boldsymbol{\vartheta},\boldsymbol{\varphi})$$

Make the discriminator smarter

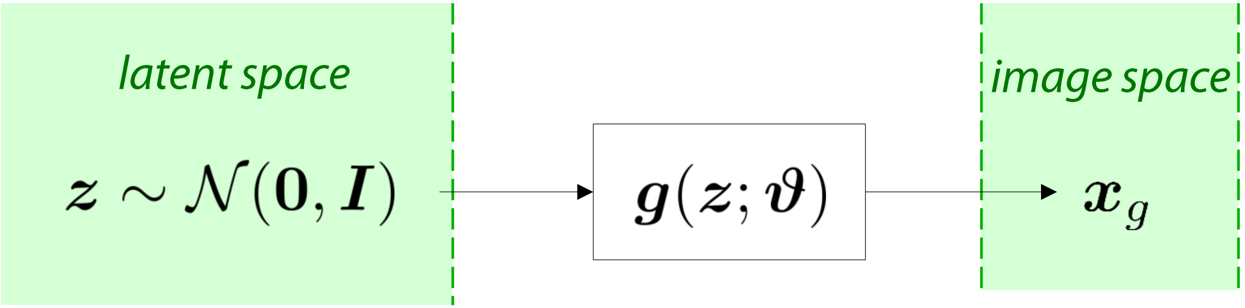
$$\Delta\boldsymbol{\vartheta} = +\,\eta\frac{\partial}{\partial\boldsymbol{\vartheta}}L(\boldsymbol{\vartheta},\boldsymbol{\varphi})$$

Make the generator smarter: the discriminator should be fooled

Generative Adversarial Network (GAN)

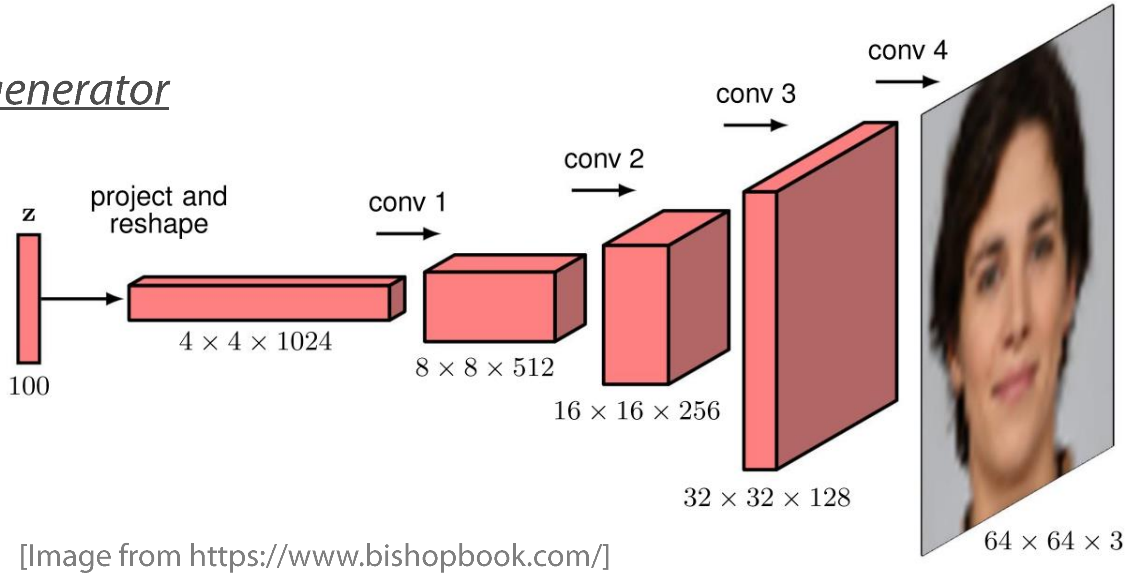
After training

The generator can be used to transform <u>random samples</u> in latent space into realistic data items



ImageGAN

Typically, a (de)convolutional network is used for the *<u>generator</u>*

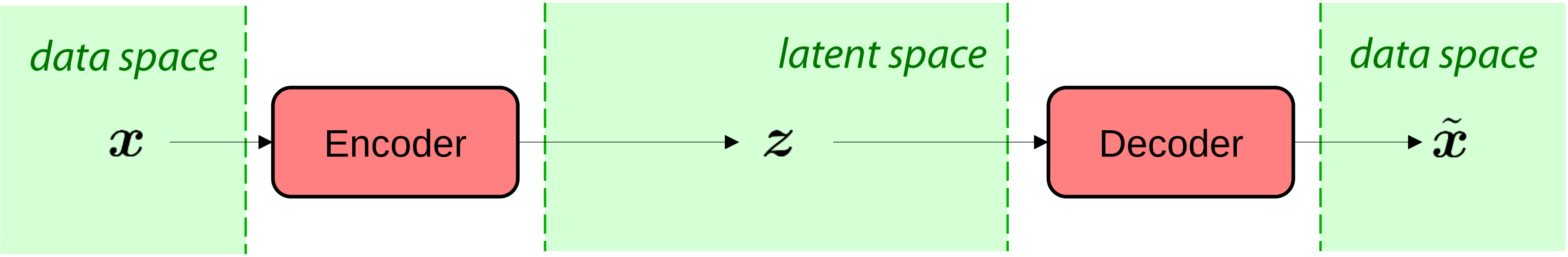


[Image from https://www.bishopbook.com/]

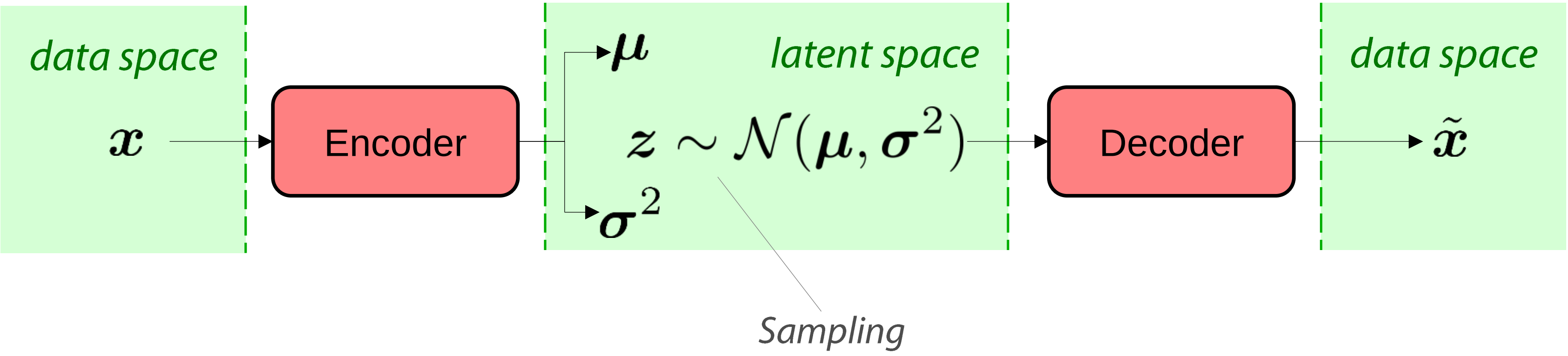
Variational Auto-Encoders

Basic idea

Auto-Encoder:
from data space
into latent space
then back



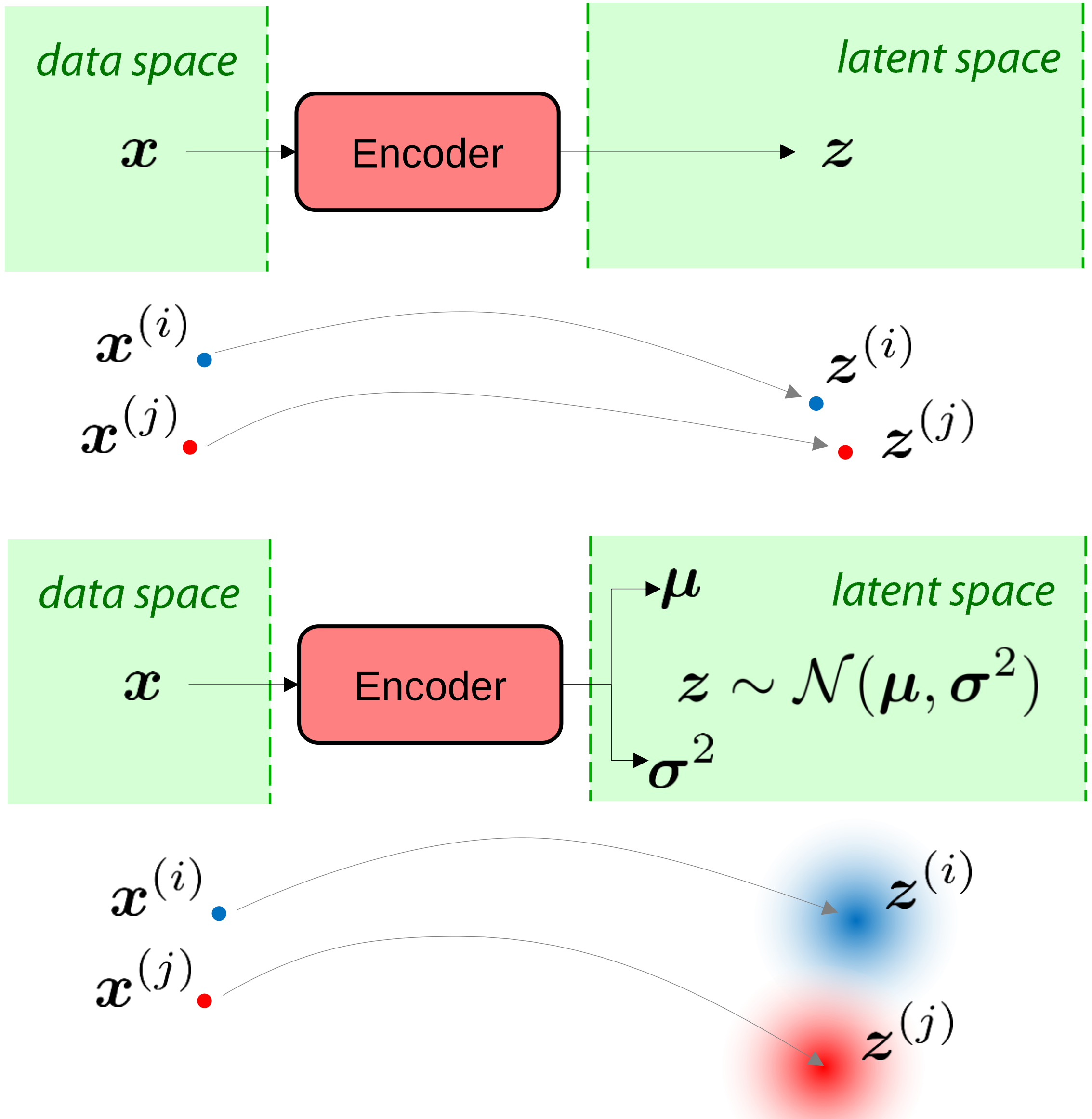
Variational
Auto-Encoder:
use a Gaussian <u>spread</u>
<u>function</u> to ***organize***
the latent space



Organizing the latent space

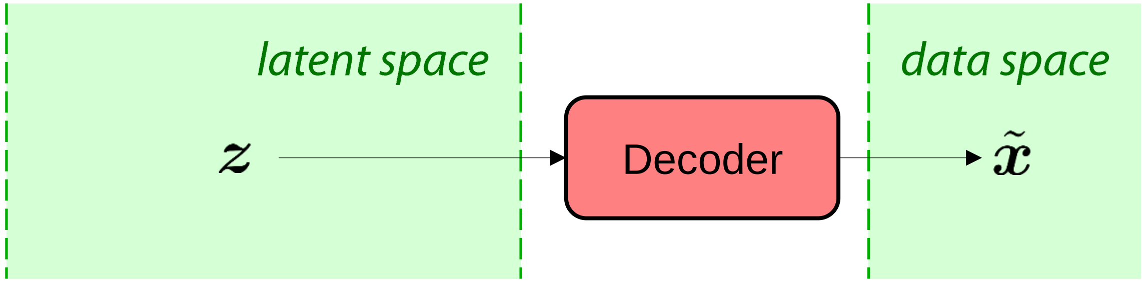
Auto-Encoder:
the correspondence between data space and latent space is one to one

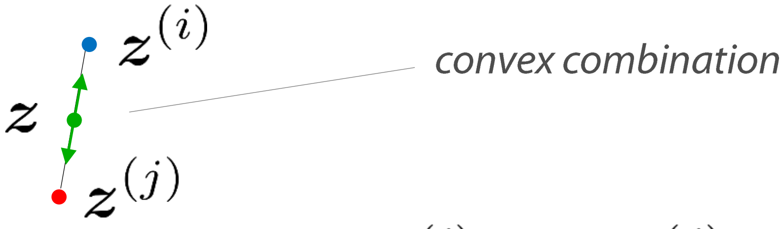
Variational Auto-Encoder :
the correspondence is one to many



Smooth generation

Variational Auto-Encoder:
after training, any convex combination of two points in latent space will generate a data item that changes smoothly from one extreme to the other



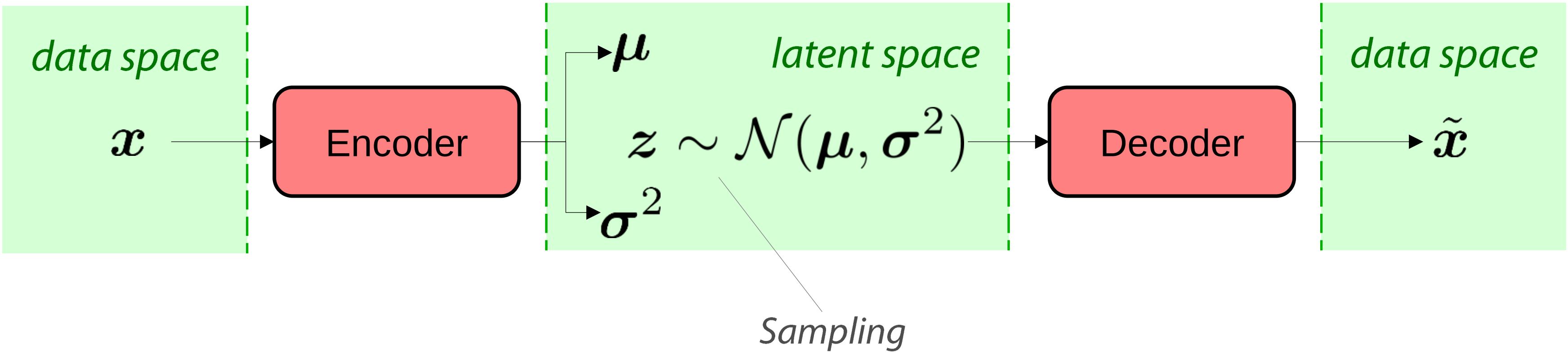


$$z = \alpha_i z^{(i)} + \alpha_j z^{(j)}$$

$$\alpha_i, \alpha_j \geq 0, \quad \alpha_i + \alpha_j = 1$$

A problem: which loss function?

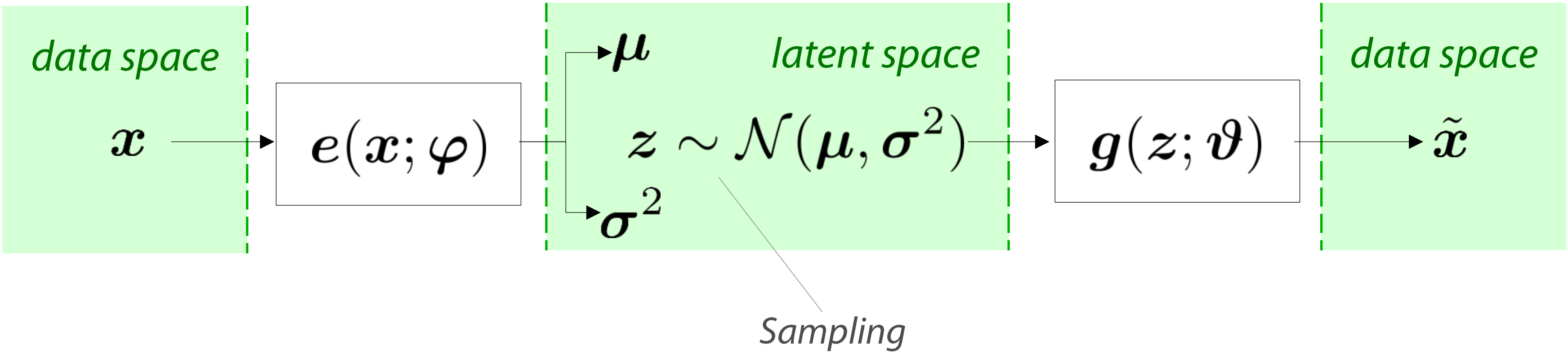
Variational Auto-Encoder :
use a Gaussian <u>spread function</u> to ***organize*** *the latent space*



This is what we want to train from a real dataset $D = \{\boldsymbol{x}_r^{(i)}\}_{i=1}^{N_r}$

A problem: which loss function?

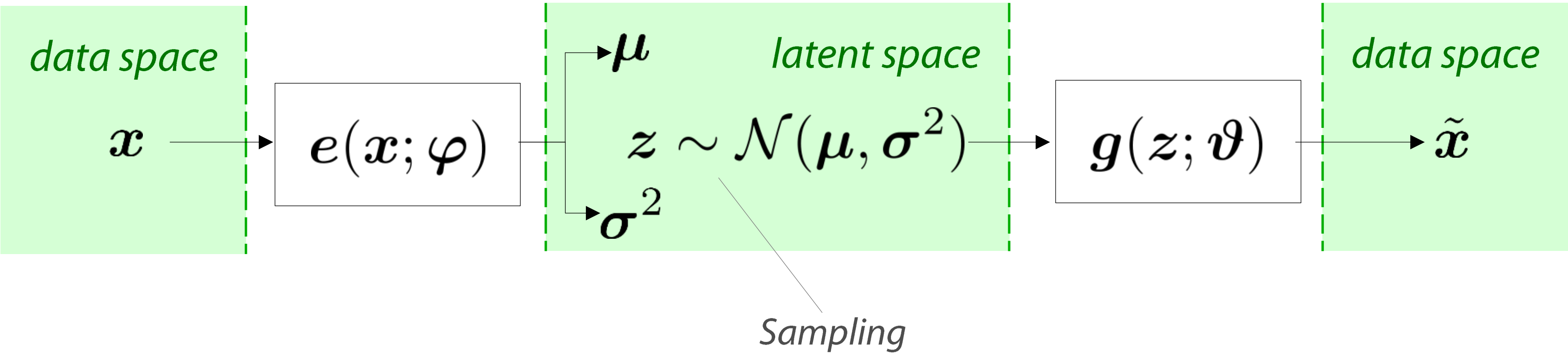
Variational Auto-Encoder :
*use a Gaussian <u>spread function</u> to **organize** the latent space*



This is what we want to train from a real dataset $D = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$

A problem: which loss function?

Variational Auto-Encoder :
*use a Gaussian <u>spread function</u> to **organize** the latent space*



This is what we want to train from a real dataset $D = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$

This is similar to the loss of a standard autoencoder

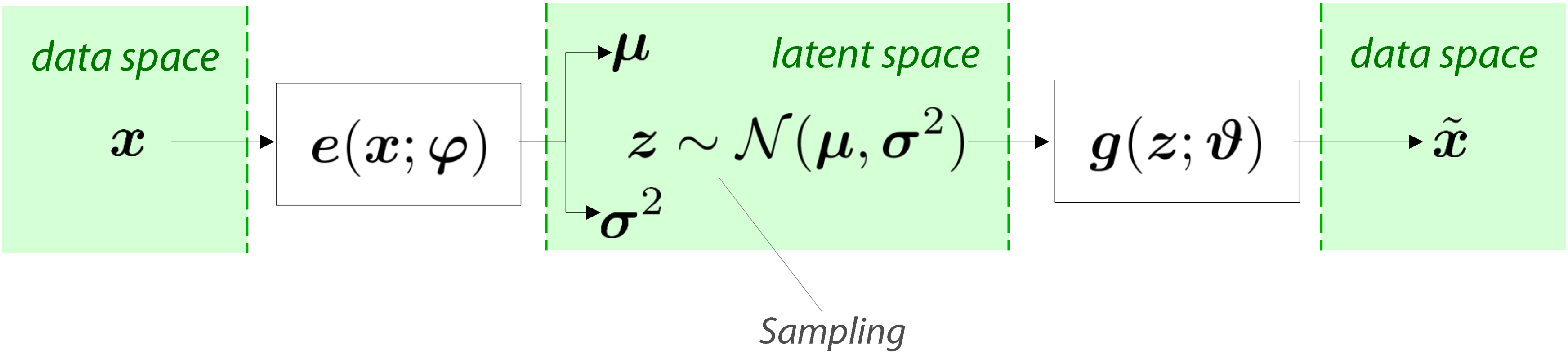
$$L(\boldsymbol{x};\boldsymbol{\varphi},\boldsymbol{\vartheta}) := \mathrm{KL}(q(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{\varphi}) \parallel p(\boldsymbol{z})) + \frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

This is an hyperparameter (see later)

Kullback-Leibler divergence

A problem: which loss function?

Variational Auto-Encoder :
*use a Gaussian <u>spread function</u> to **organize** the latent space*



This is what we want to train from a real dataset $D = \{x^{(i)}\}_{i=1}^{N}$

$$L(x;\varphi,\vartheta) := \mathrm{KL}(q(z \mid x, \varphi) \,\|\, p(z)) \; + \; \frac{1}{2}\frac{\|x - \tilde{x}\|^2}{c}$$

Design choices

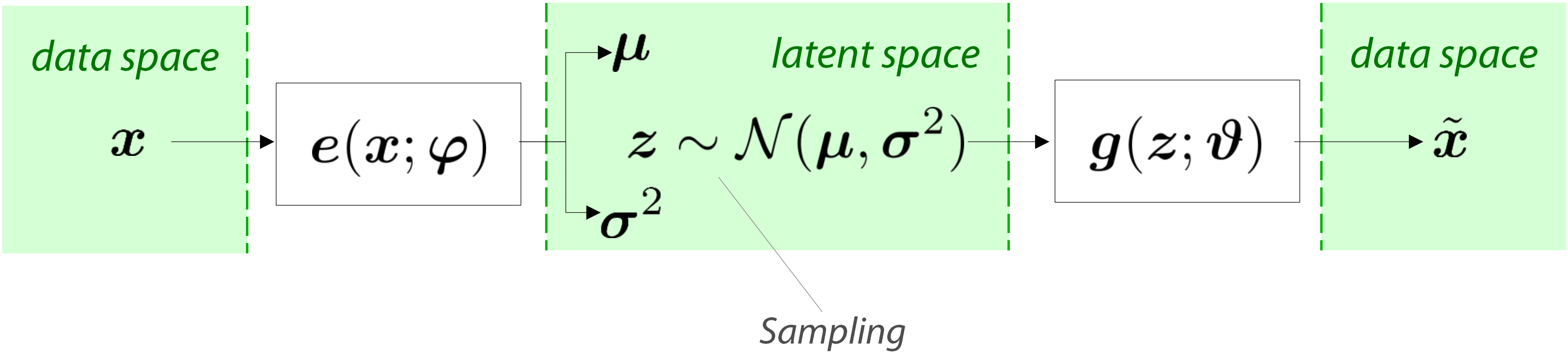
$$q(z \mid x, \varphi) := \mathcal{N}(\mu(x;\varphi), \sigma^2(x;\varphi) I)$$

$$p(z) := \mathcal{N}(0, I)$$

Normalization constraint: a soft limit against overspreading latent values

A problem: which loss function?

Variational Auto-Encoder :
*use a Gaussian <u>spread function</u> to **organize** the latent space*



In general:

$$\mathrm{KL}(q(\boldsymbol{z}) \parallel p(\boldsymbol{z})) := \int q(\boldsymbol{z}) \ln \frac{q(\boldsymbol{z})}{p(\boldsymbol{z})} \, \mathrm{d}\boldsymbol{z}$$

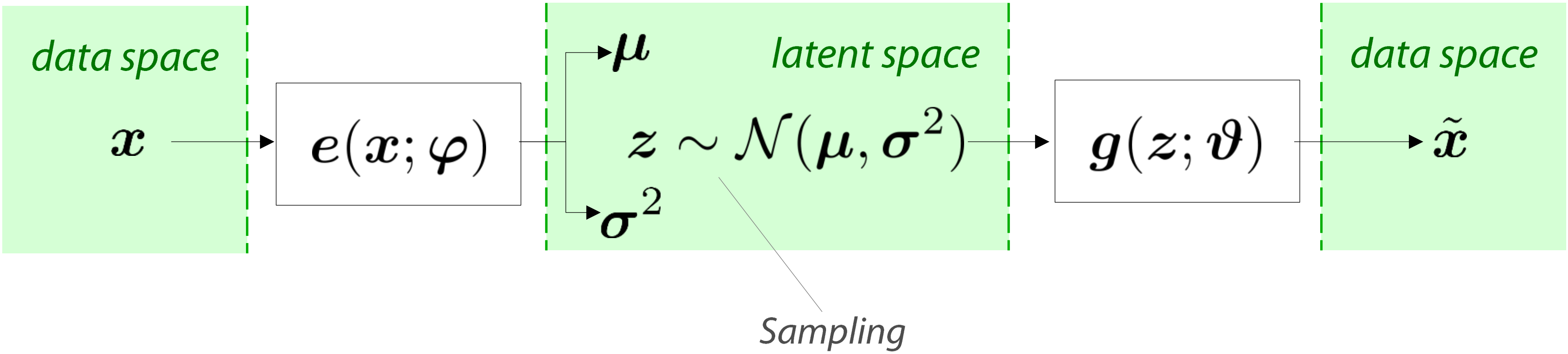
Kullback-Leibler divergence; always positive, zero when the two distributions are identical

Since both distributions are normal:

$$\mathrm{KL}(q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\varphi}) \parallel p(\boldsymbol{z})) = -\frac{1}{2} \sum_{j=1}^{\dim(\boldsymbol{z})} \left(1 + \ln \sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \mu_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi})\right)$$

A problem: which loss function?

Variational Auto-Encoder :
use a Gaussian <u>spread</u> <u>function</u> to ***organize*** *the latent space*



With a bit more of mathematics (omitted) it can be shown that the second term in the loss function

$$\frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

relates to an assumption of:

$$p(\tilde{\boldsymbol{x}}) := \mathcal{N}(\boldsymbol{x}, c\boldsymbol{I}), \;\; c > 0$$

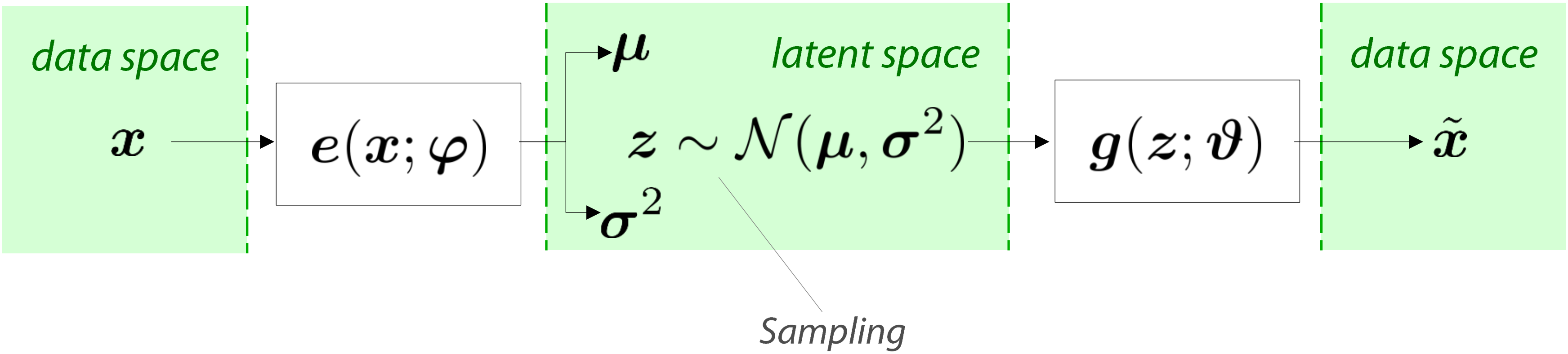
hyperparameter

Design choice

(hyper) spherical normal

Reparametrization Trick

Variational Auto-Encoder :
*use a Gaussian spread function to **organize** the latent space*



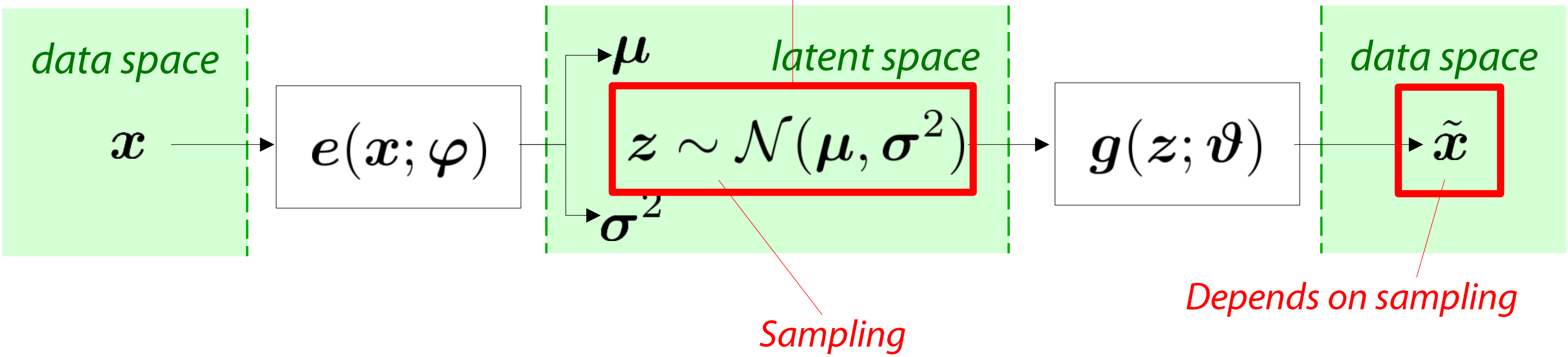
$$L(\boldsymbol{x};\boldsymbol{\varphi},\boldsymbol{\vartheta}) := -\frac{1}{2}\sum_{j=1}^{\dim(\boldsymbol{z})}\Big(1 + \ln\sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \mu_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi})\Big) \; + \; \frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

$$\Delta\boldsymbol{\varphi} = -\,\eta\frac{\partial}{\partial\boldsymbol{\varphi}}L(\boldsymbol{x};\boldsymbol{\vartheta},\boldsymbol{\varphi}) \qquad\qquad \Delta\boldsymbol{\vartheta} = -\,\eta\frac{\partial}{\partial\boldsymbol{\vartheta}}L(\boldsymbol{x};\boldsymbol{\vartheta},\boldsymbol{\varphi})$$

Reparametrization Trick

$\tilde{\boldsymbol{x}}$ *depends on both* $\boldsymbol{\vartheta}$ *and* $\boldsymbol{\varphi}$ *via* $\boldsymbol{z}$
yet, when $\boldsymbol{z}$ *is sampled, the derivative in* $\boldsymbol{\varphi}$ *is blocked*

Variational Auto-Encoder : *use a Gaussian spread function to* ***organize*** *the latent space*



$$L(\boldsymbol{x};\boldsymbol{\varphi},\boldsymbol{\vartheta}) := -\frac{1}{2}\sum_{j=1}^{\dim(\boldsymbol{z})}\left(1 + \ln\sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \mu_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi})\right) + \frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

The trick is:

$$\boldsymbol{z} = \boldsymbol{\mu}(\boldsymbol{x};\boldsymbol{\varphi}) + \varepsilon\,\boldsymbol{\sigma}(\boldsymbol{x};\boldsymbol{\varphi})$$

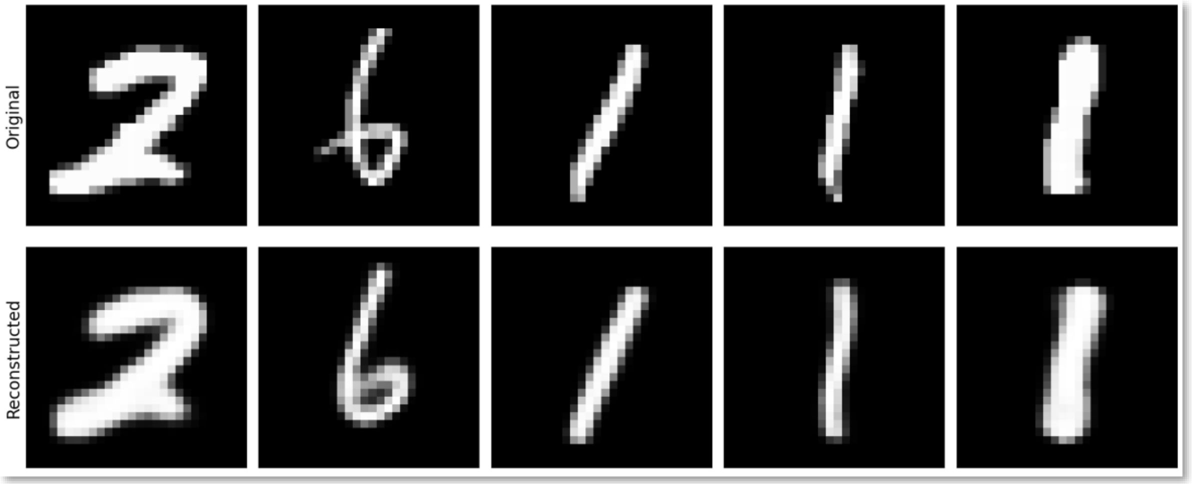
where:

$$\varepsilon \sim \mathcal{N}(0,1)$$

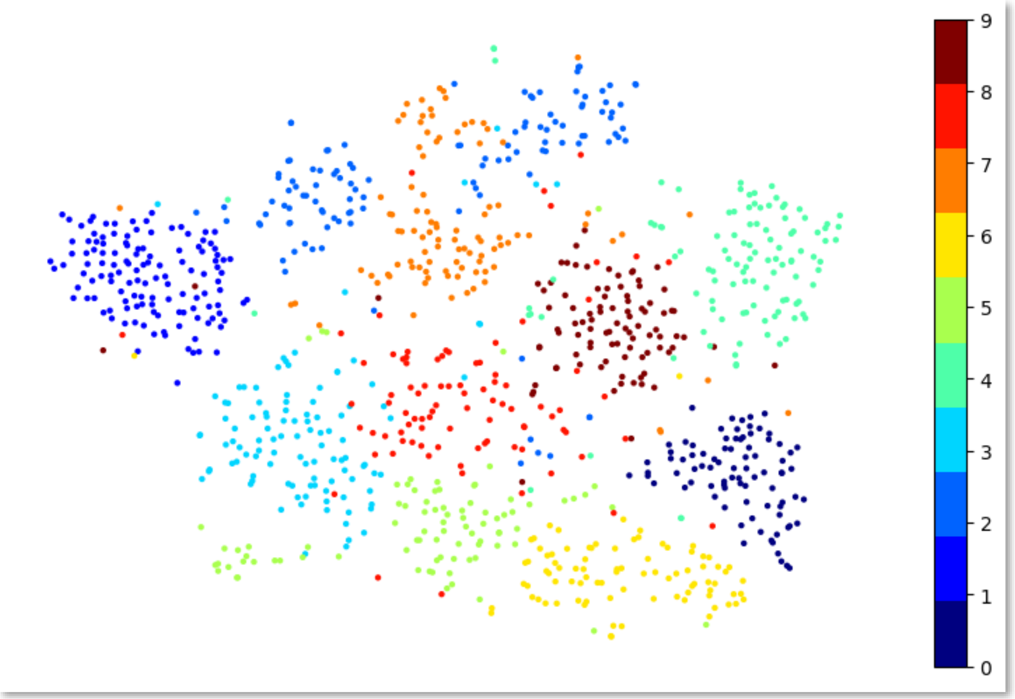
is an hyperparameter (although it changes at each forward pass) therefore it is constant to the derivative.
In plain words, during training and per each data item $\boldsymbol{x}^{(i)}$ *the system draws one random value* ε *and computes the derivatives*

VAE MNIST Experiments

Reconstruction



Latent space (2D TSNE)



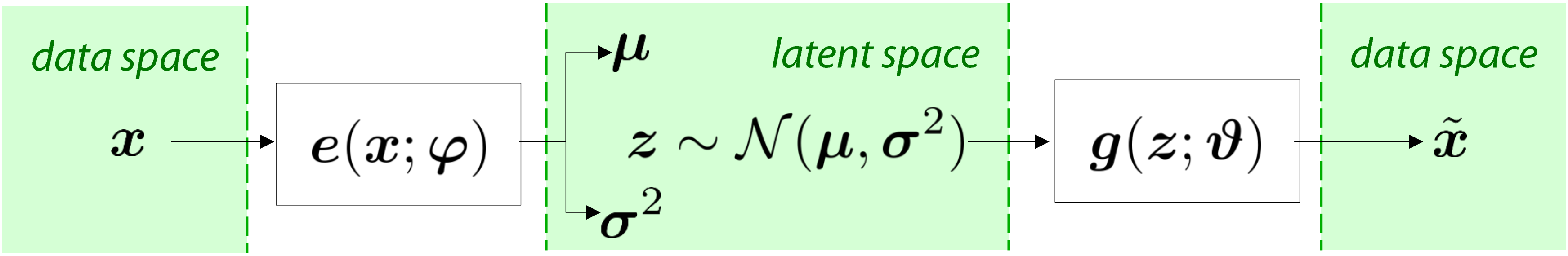
Links

https://johfischer.com/2022/09/18/denoising-score-matching/
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
https://en.wikipedia.org/wiki/Variational_autoencoder
https://mbernste.github.io/posts/vae/

Gaussian-Mixture Variational Auto-Encoder (GMVAE)

One VAE limitation: unimodal arrangement of latent space

Variational Auto-Encoder :
use a Gaussian <u>spread function</u> to ***organize*** *the latent space*



This is what we want to train from a real dataset $D = \{\boldsymbol{x}_r^{(i)}\}_{i=1}^{N_r}$

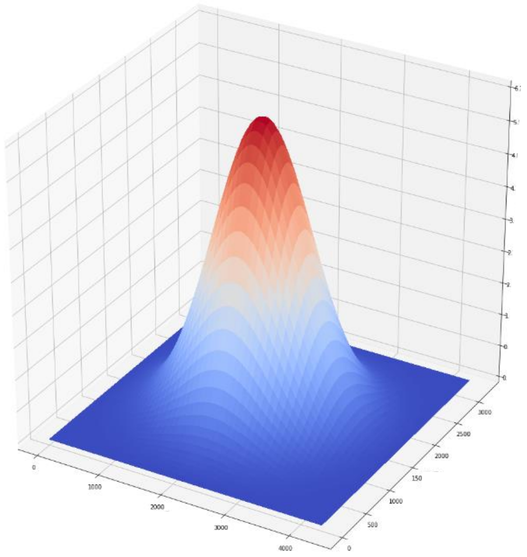
$$L(\boldsymbol{x};\boldsymbol{\varphi},\boldsymbol{\vartheta}) := \mathrm{KL}(q(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{\varphi}) \parallel p(\boldsymbol{z})) \; + \; \frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

Design choices

$$q(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{\varphi}) := \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x};\boldsymbol{\varphi}), \boldsymbol{\sigma}^2(\boldsymbol{x};\boldsymbol{\varphi})\boldsymbol{I})$$

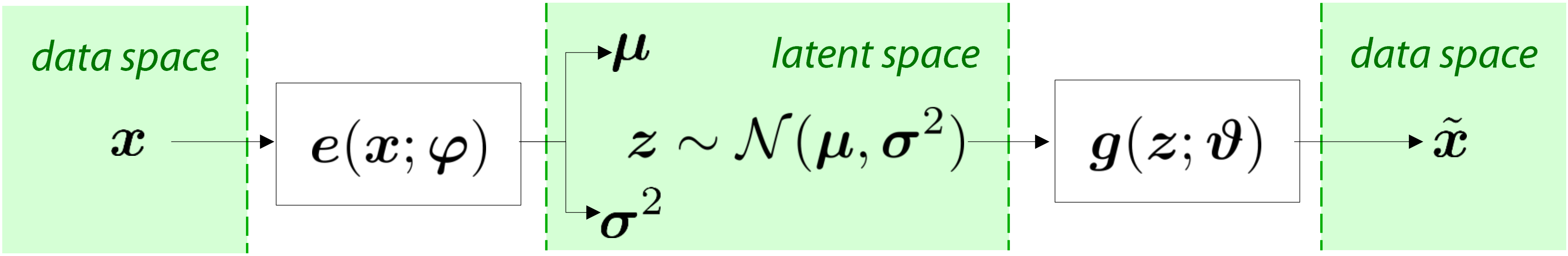
$$p(\boldsymbol{z}) := \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$$

Unimodal distribution

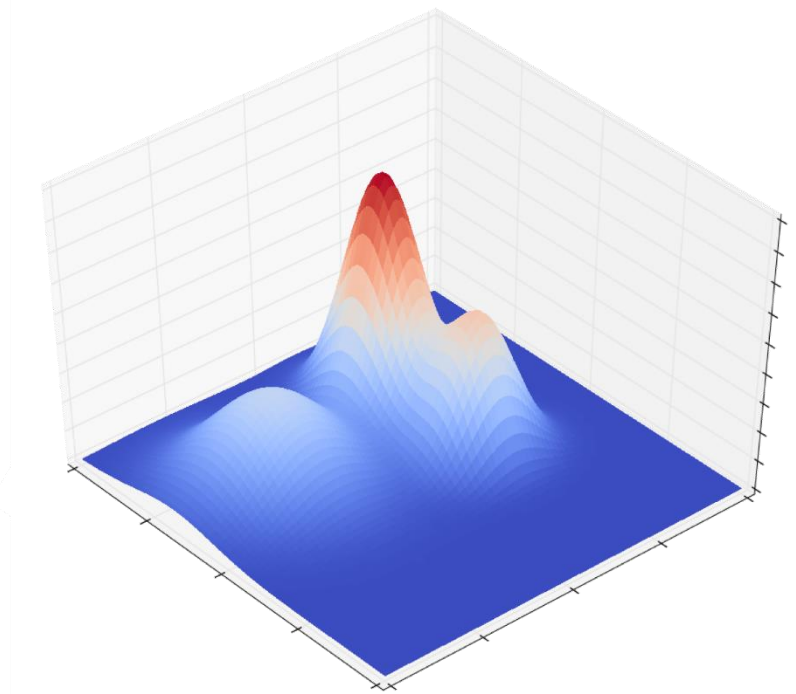


One VAE limitation: unimodal arrangement of latent space

Variational Auto-Encoder :
use a Gaussian <u>spread function</u> to ***organize*** *the latent space*

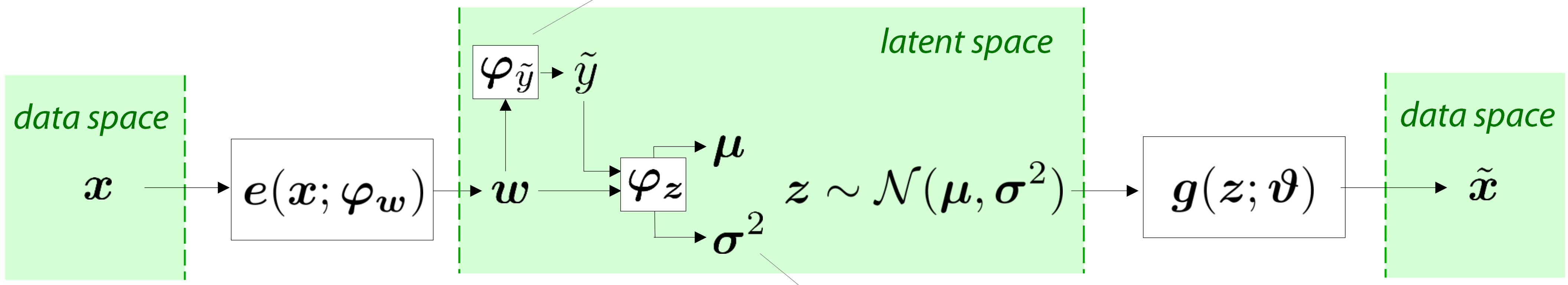


We may be wanting a more flexible arrangement of latent space



Multimodal distribution

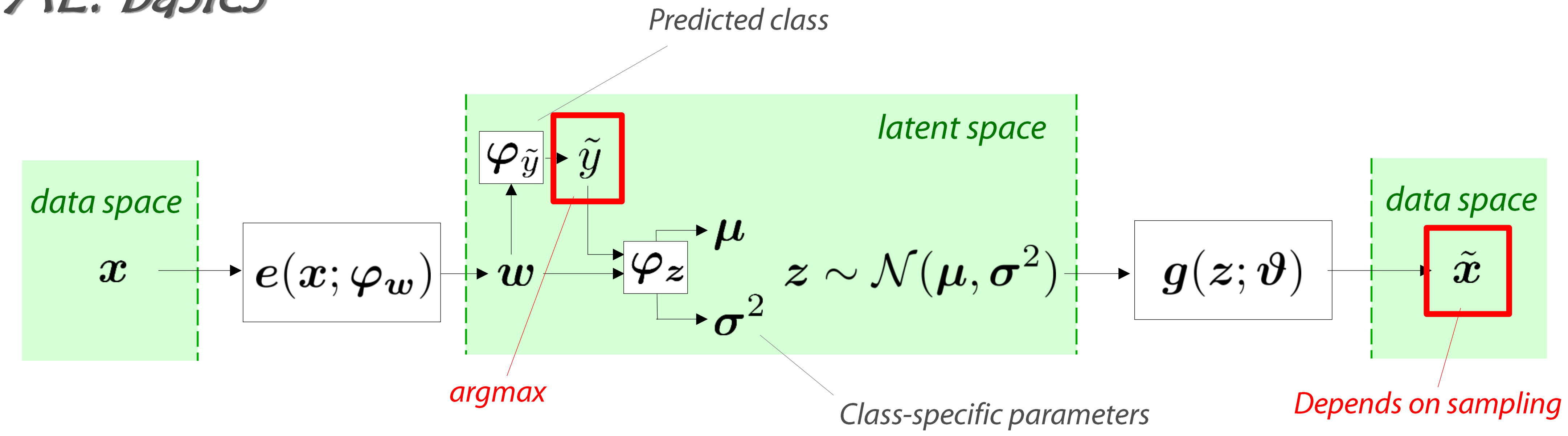
GMVAE: Basics



Training dataset $D = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$

Class label

GMVAE: Basics

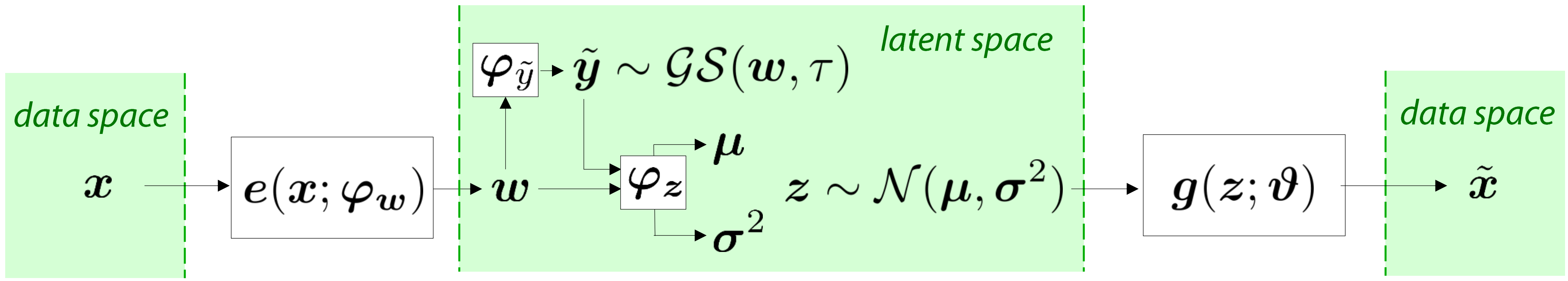


Similar problem as before:
'argmax' prevents backpropagation

Training dataset $D = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$

Class label

GMVAE: Gumbel-Softmax



Yet another reparametrization trick:

$$\boldsymbol{\lambda} = \operatorname{logit}(\boldsymbol{w}, \boldsymbol{\varphi}_{\tilde{y}}) \in \mathbb{R}^k$$

Number of classes

Gumbel distribution

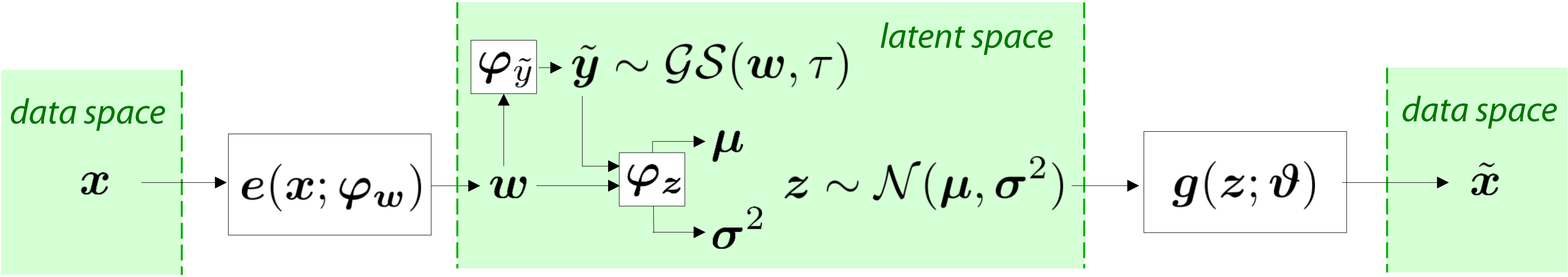
$$\boldsymbol{\xi} = -\log(-\log \boldsymbol{u}), \quad \boldsymbol{u} \sim \mathcal{U}(0, 1; k) \in \mathbb{R}^k$$

Uniform distribution over [0, 1]

$$\tilde{\boldsymbol{y}} = \mathcal{GS}(\boldsymbol{w}, \tau) = \frac{\exp((\boldsymbol{\lambda} + \boldsymbol{\xi})/\tau)}{\sum \exp((\boldsymbol{\lambda} + \boldsymbol{\xi})/\tau)}$$

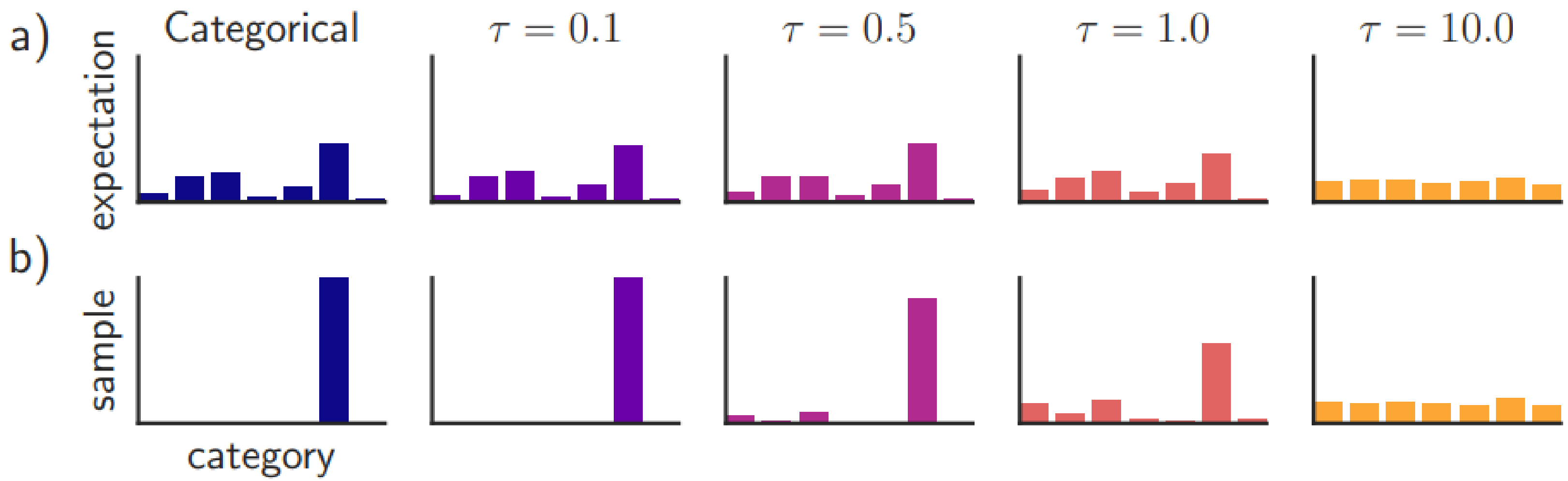
'Temperature' hyperparameter

GMVAE: Gumbel-Softmax



Gumbel-Softmax: derivable, 'soft-argmax'

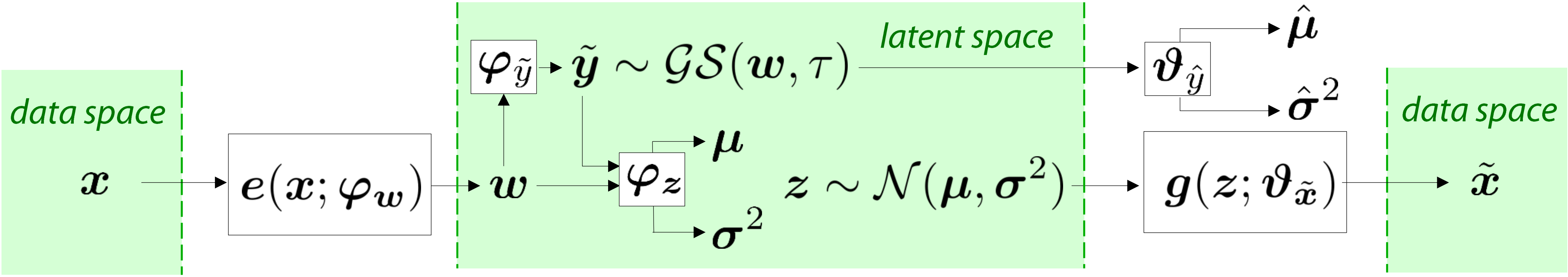
'Temperature' hyperparameter



The result is
a 'soft one-hot encoding'
of the argmax

[image from https://arxiv.org/pdf/1611.01144]

GMVAE: Loss Function



$$L(\boldsymbol{x}; \boldsymbol{\varphi}, \boldsymbol{\vartheta}) := \; c_r \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2 + c_g \left| \log \mathcal{N}(\boldsymbol{z}, \boldsymbol{\mu}, \boldsymbol{\sigma}^2) - \log \mathcal{N}(\boldsymbol{z}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}^2) \right| + c_e \, \mathrm{Ent}(\tilde{\boldsymbol{y}})$$

Reconstruction loss — *'Gaussian' loss* — *Entropy*

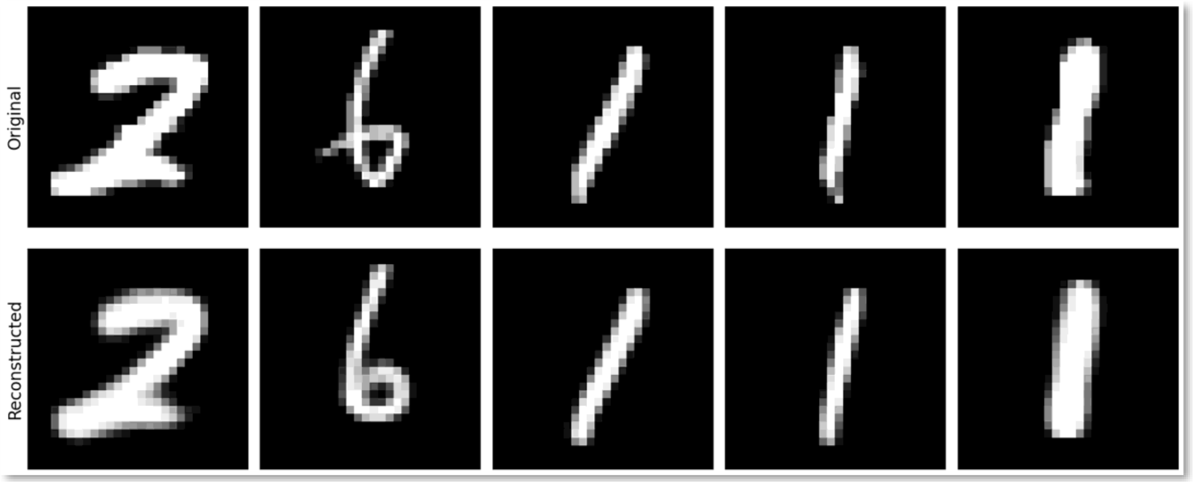
where:

$$\mathrm{Ent}(\tilde{\boldsymbol{y}}) = -\sum_{i=1}^{k} \tilde{y}_i \log \tilde{y}_i$$

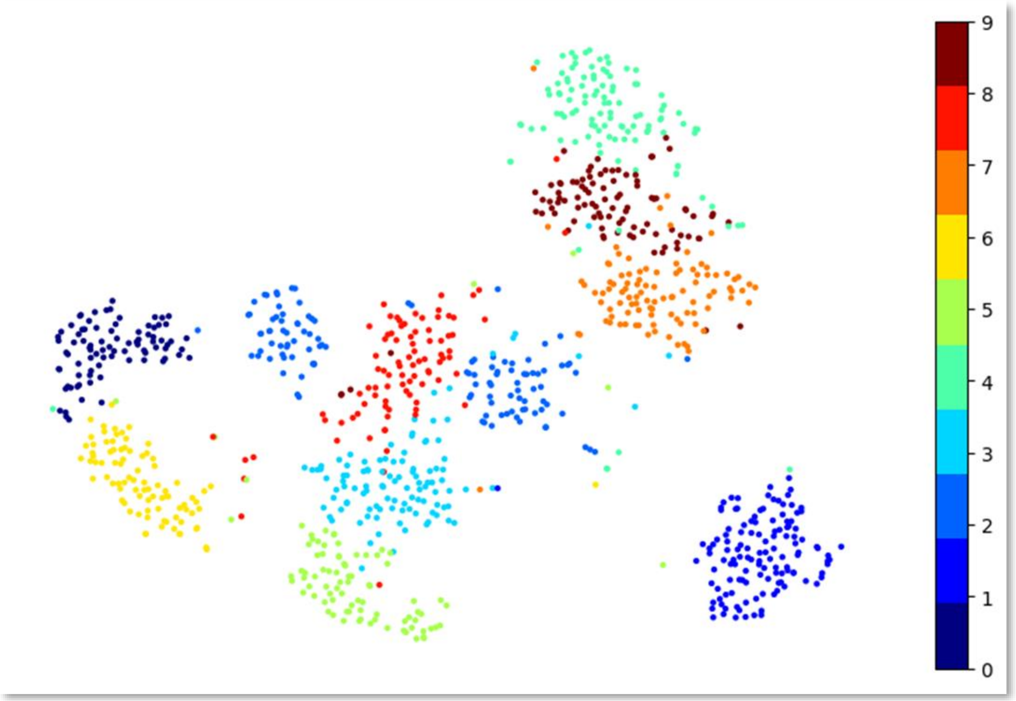
$\tilde{\boldsymbol{y}}$ *is generated by a softmax:*
each $\tilde{y}_i$ *is a probability*

GMVAE MNIST Experiments

Reconstruction

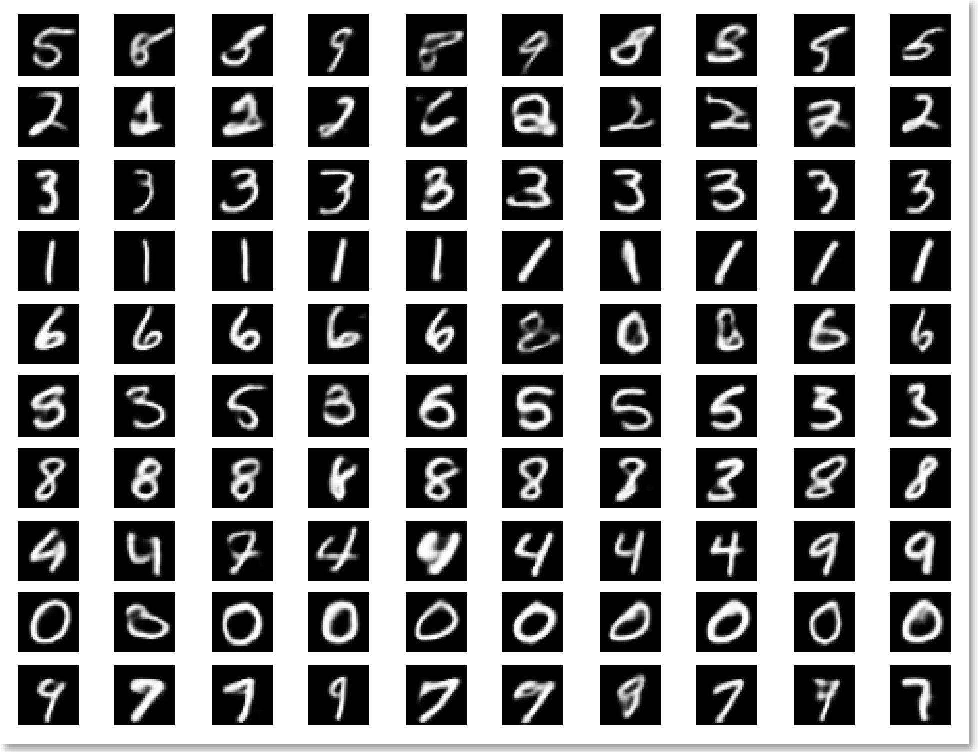


Latent space (2D TSNE)

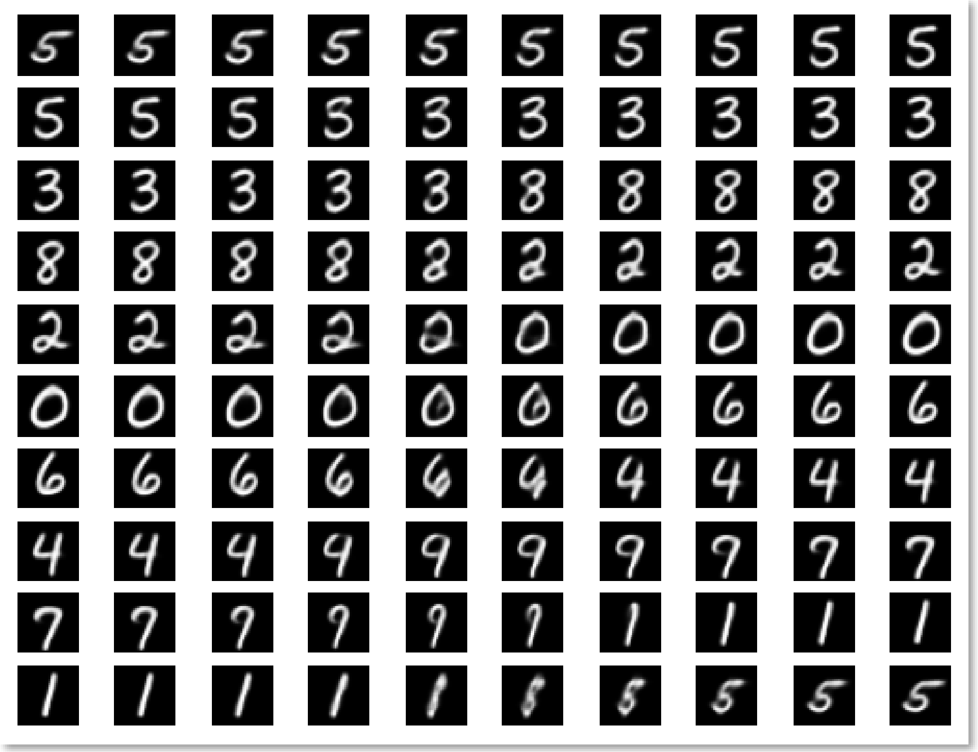


GMVAE MNIST Experiments

Random sampling (for each class μ and σ)



Convex interpolation (TSP tour)



Links

https://arxiv.org/pdf/1406.5298
https://arxiv.org/pdf/1611.05148
https://ruishu.io/2016/12/25/gmvae/
https://arxiv.org/pdf/2112.00976