

(Variational) autoencoders



Carl Herrmann
Heidelberg University



CHARLES
UNIVERSITY



SORBONNE
UNIVERSITÉ



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386



UNIVERSITY
OF WARSAW



UNIVERSITÀ
DEGLI STUDI
DI MILANO

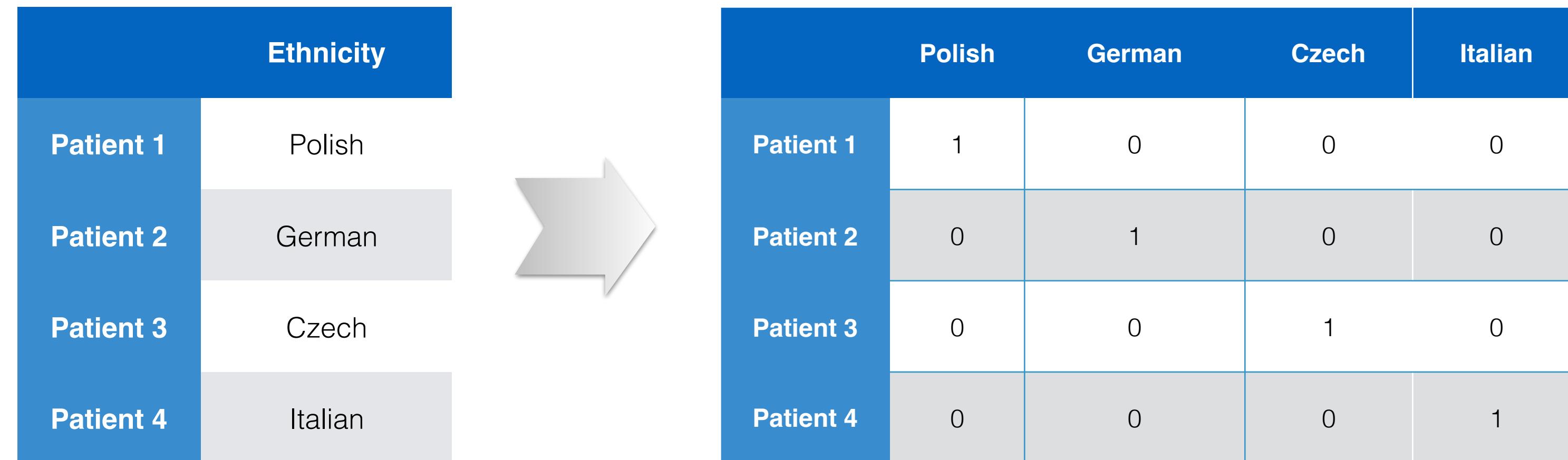


EUROPEAN
UNIVERSITY
ALLIANCE

1 - Embeddings

Embeddings

- Categorical variables are usually represented as **one-hot encoded** representations
- Example: citizenship of patients



The diagram illustrates the transformation of categorical data into one-hot encoded vectors. On the left, a table shows the ethnicity of four patients. An arrow points to the right, indicating the conversion process. On the right, a second table shows the same four patients represented as one-hot encoded vectors for Polish, German, Czech, and Italian ethnicities.

Ethnicity	
Patient	Ethnicity
Patient 1	Polish
Patient 2	German
Patient 3	Czech
Patient 4	Italian

	Polish	German	Czech	Italian
Patient				
Patient 1	1	0	0	0
Patient 2	0	1	0	0
Patient 3	0	0	1	0
Patient 4	0	0	0	1

- Limitation: there is no way to compute a similarity (or distance) between states

Embeddings

- **Embeddings** are continuous representations of categorical variables
- Embeddings can be used as (low-dimensional) representations of high-dimensional features
- Embeddings can be used to find **similarities** between objects
- Example: capitals → 2D coordinates

	<i>x</i>	<i>y</i>
Prag	→ [50.07 ; 14.41]	
Warsaw	→ [52.24 ; 21.02]	
Paris	→ [48.86 ; 2.35]	
Rom	→ [41.90 ; 12.50]	
Berlin	→ [52.52 ; 13.40]	
...	...	

D = 195

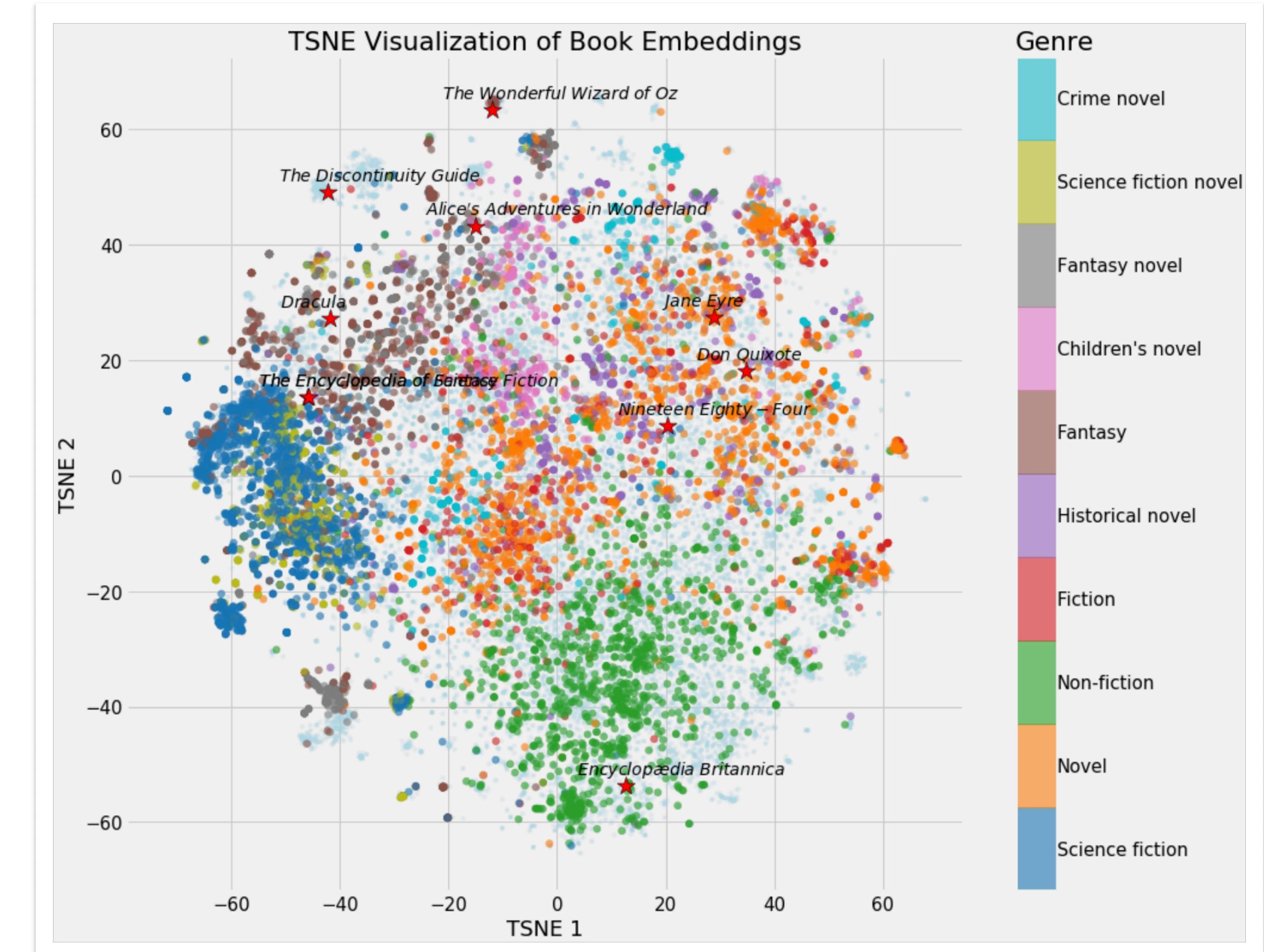
D = 2

similarity

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

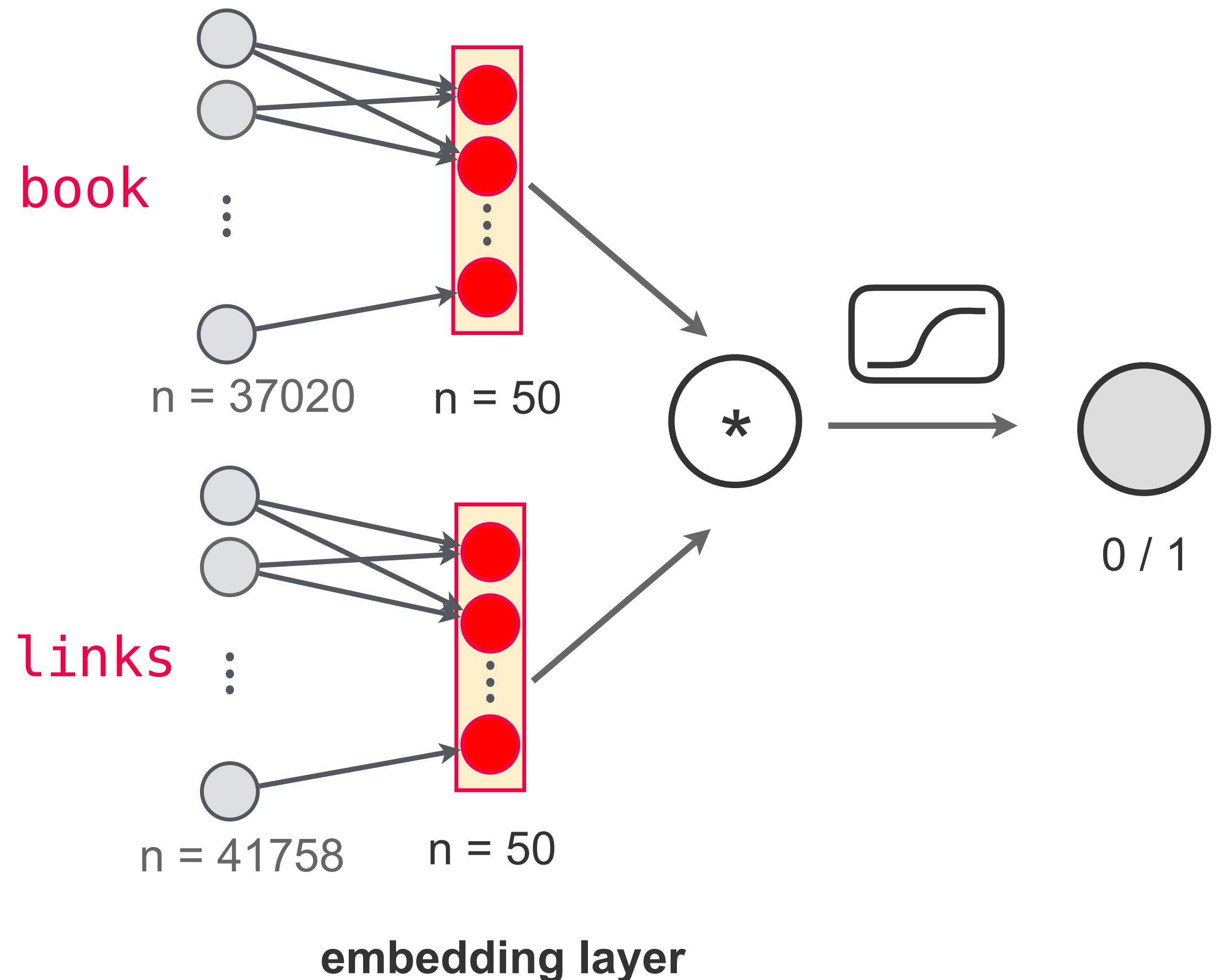
Embeddings

- **Embeddings** can correspond to a *natural representation* (e.g. geographical coordinates,...)
- or they can be learned through specific constraints (e.g. 2D representation of books, ...) → "*learned-representation*"



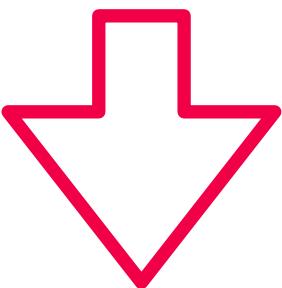
Example: book embedding

- Wikipedia articles about books → list of links contained in the wikipedia article [book, link]
- Task: predict if a link is contained in the wikipedia page of a book
 - [book, link] → 1 : article about book contains link
 - [book, link] → 0 : article about book does not contain link



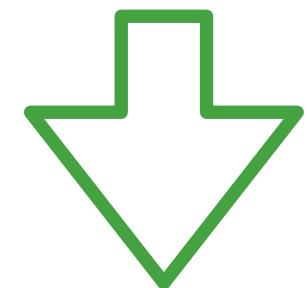
Example: book embedding

$n = 37020$



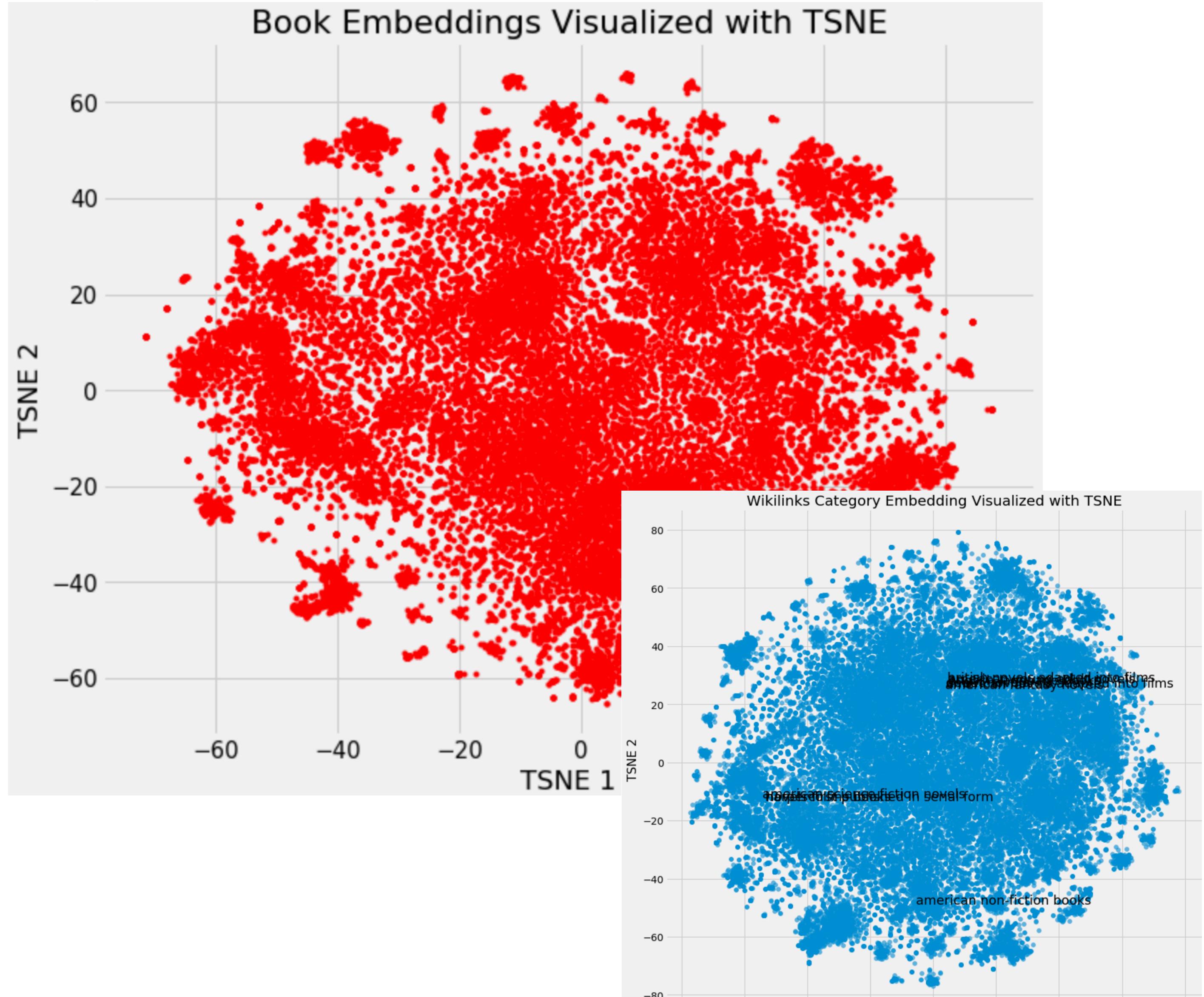
embedding

$n = 50$



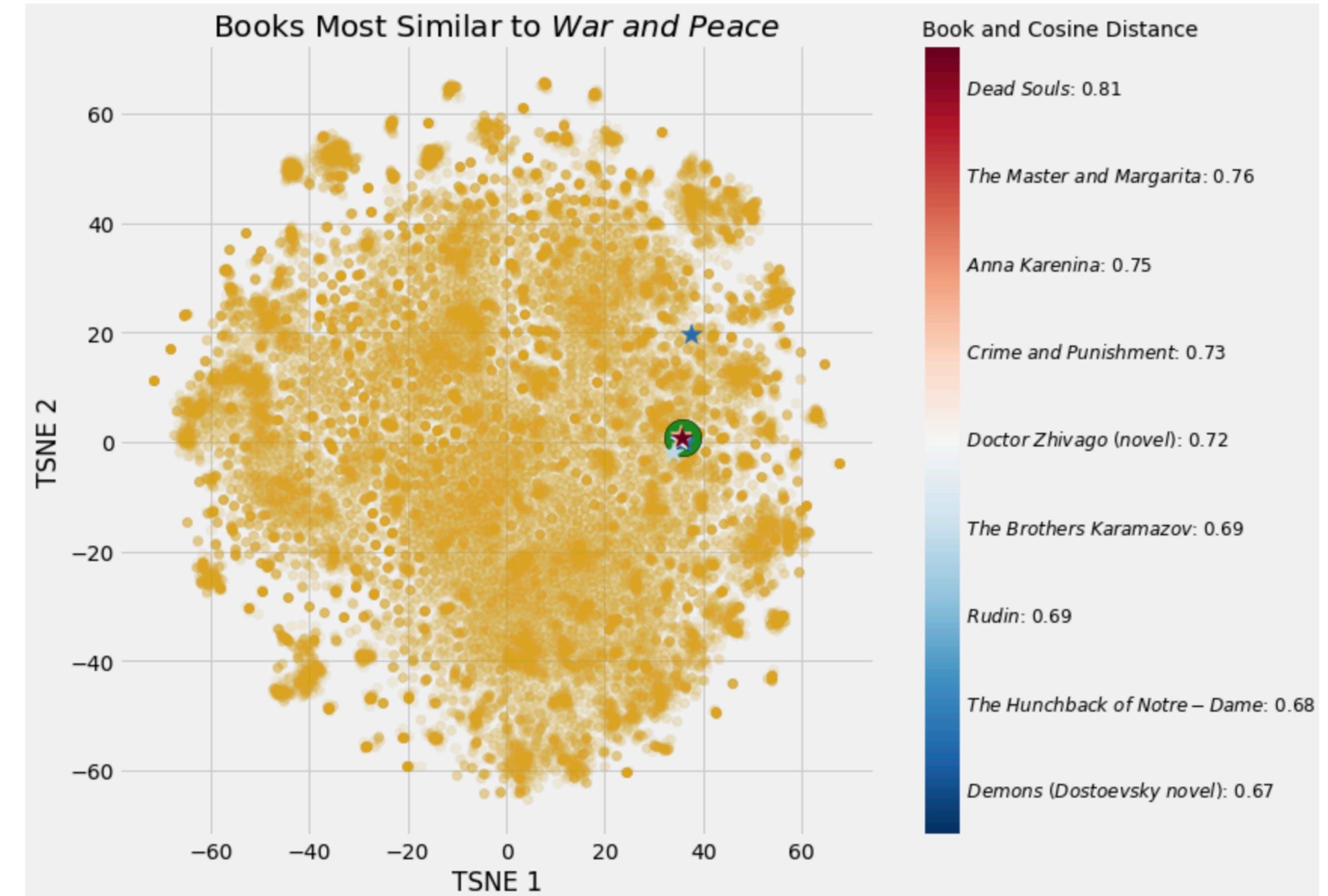
2D visualization
(t-SNE)

$n = 2$



Example: book embedding

- The notion of **distance/ similarity** in the embedding space allows to compute meaningful relationships between embedded objects
- Not possible with one-hot encoded representation!

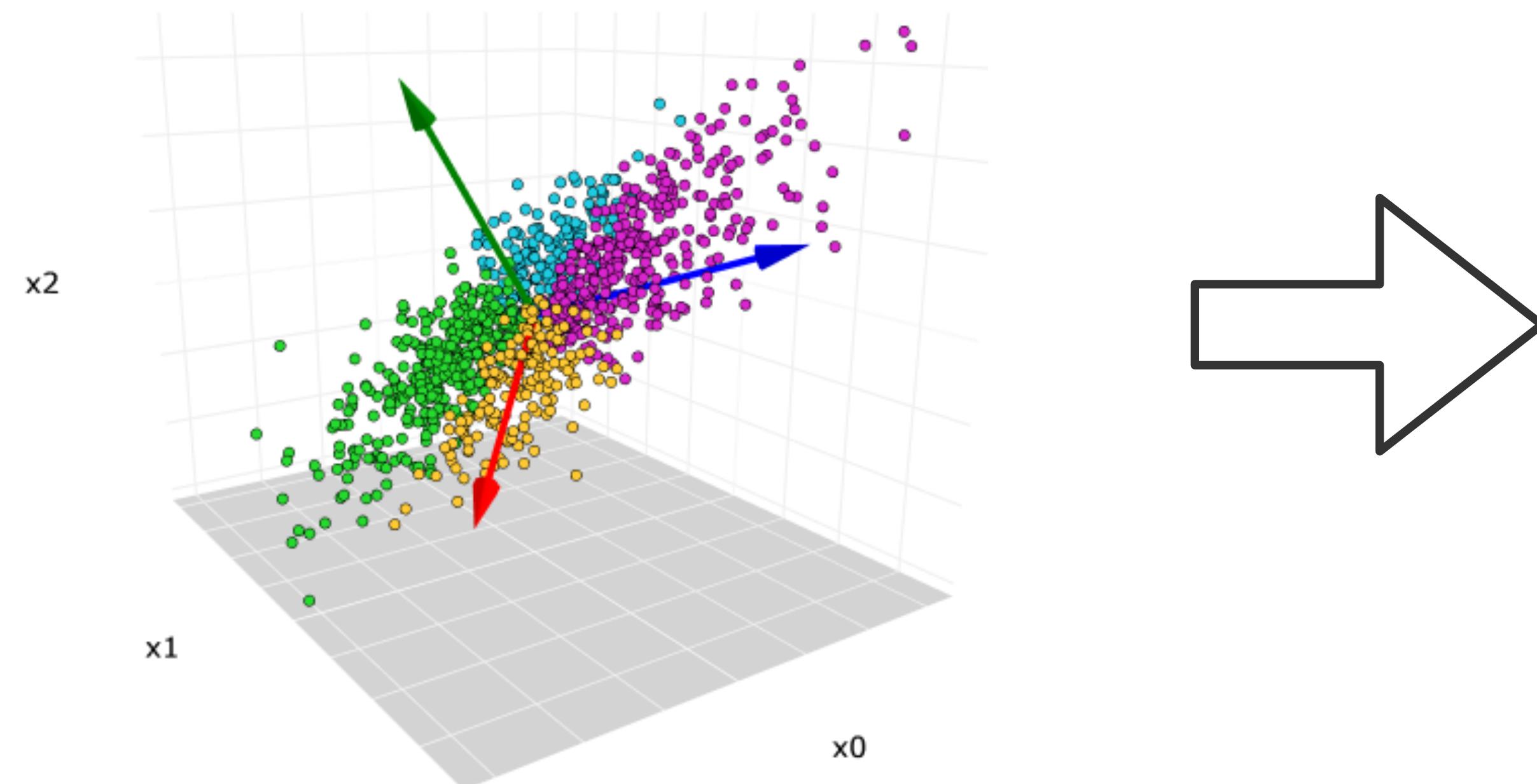


Some embedding models

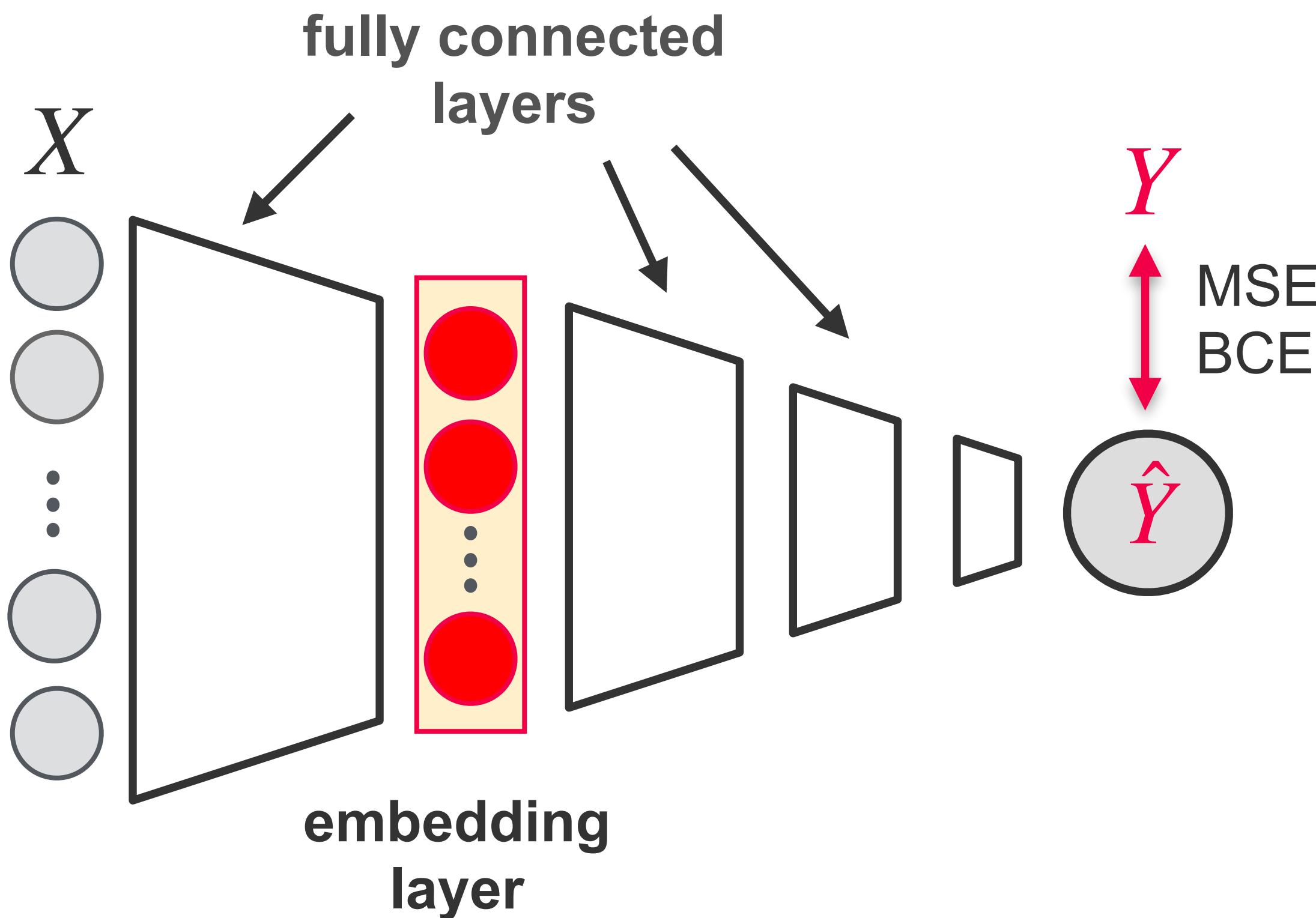
- **Principal component analysis (PCA)**

principal components = embedding learned from training dataset

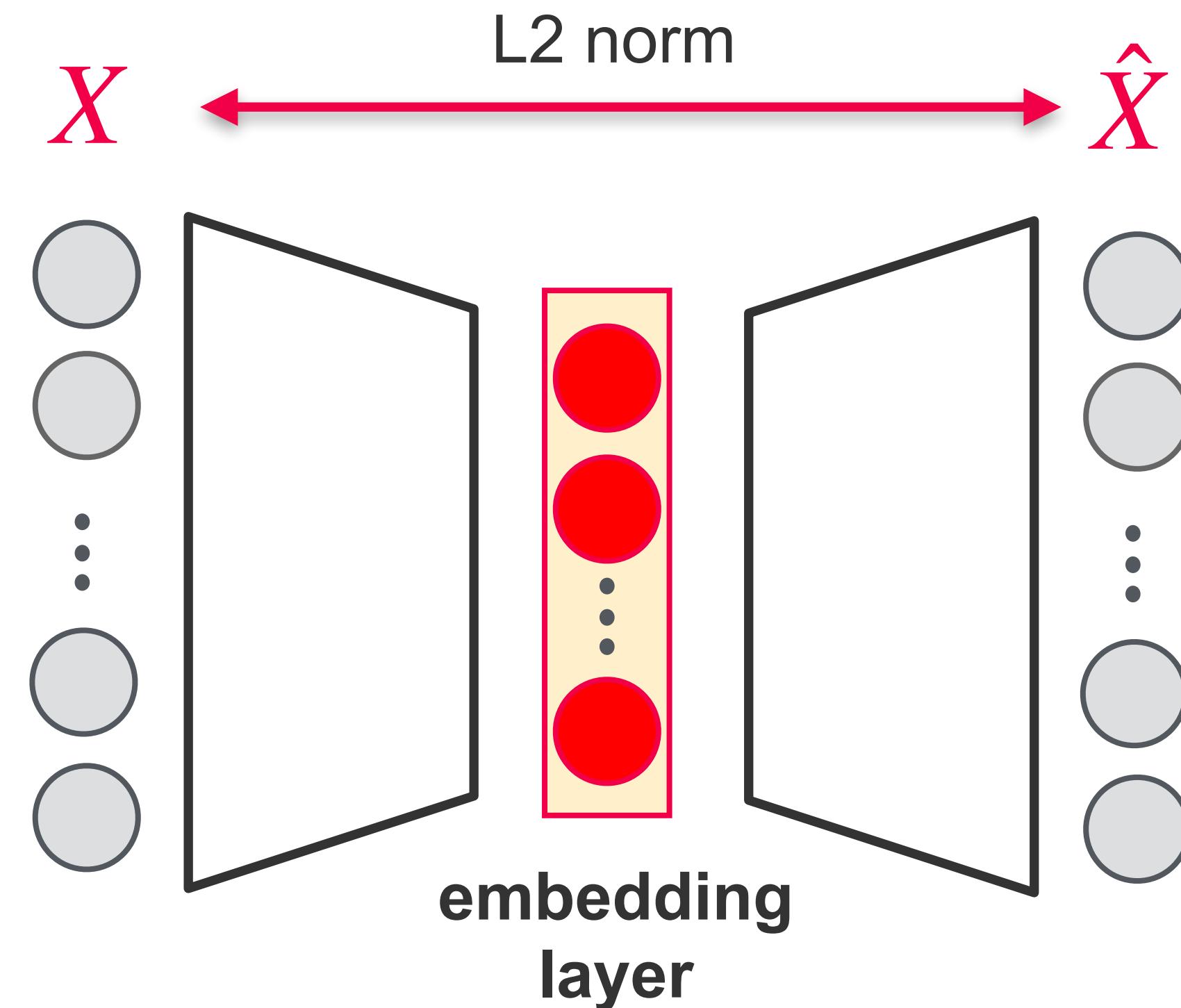
Project data from testing set in embedding space



Creating embeddings



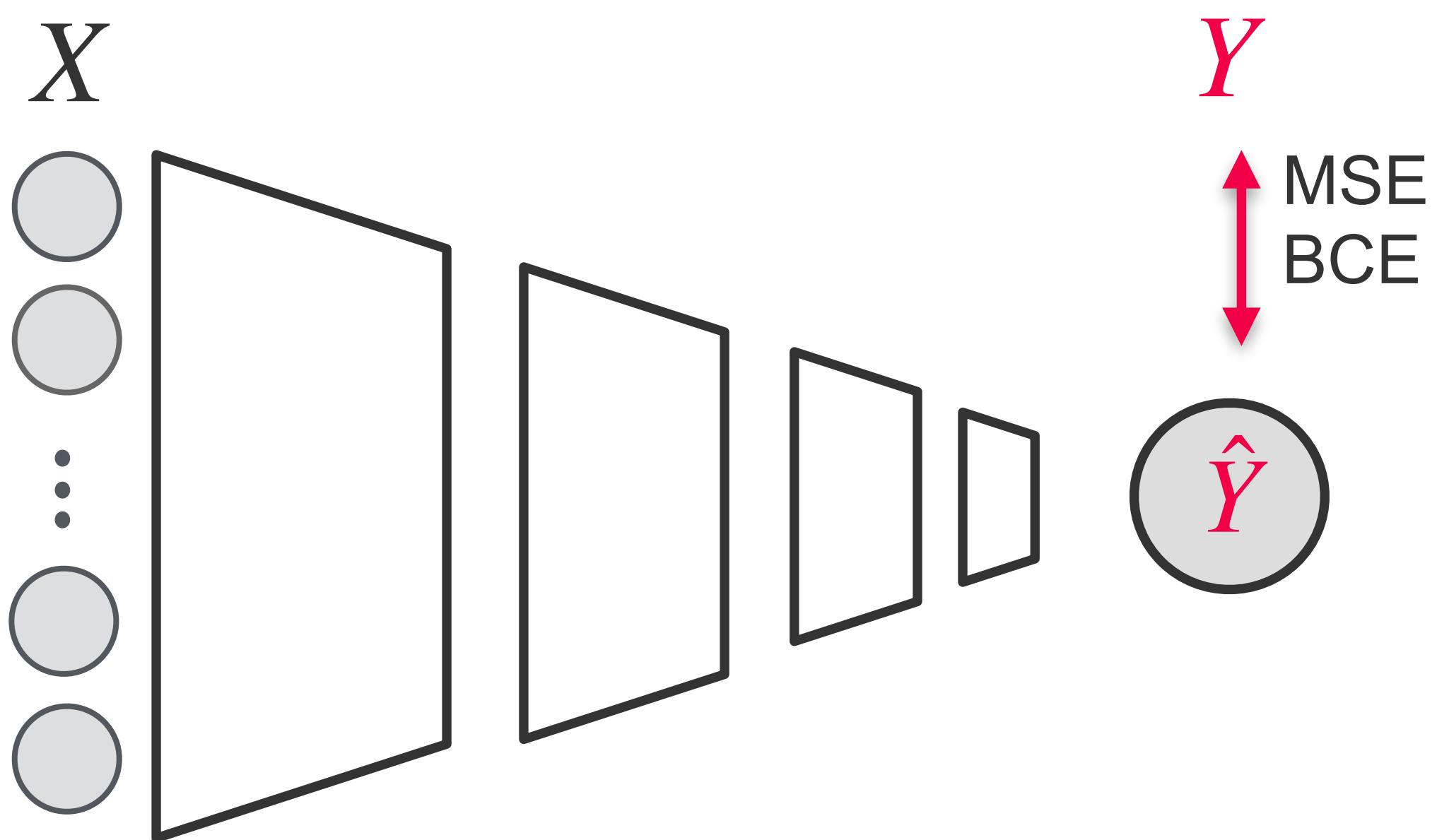
Regression/classification DNN:
create embedding
by optimizing classification/regression
problem for labelled data



Autoencoder:
create embedding
by minimizing reconstruction error

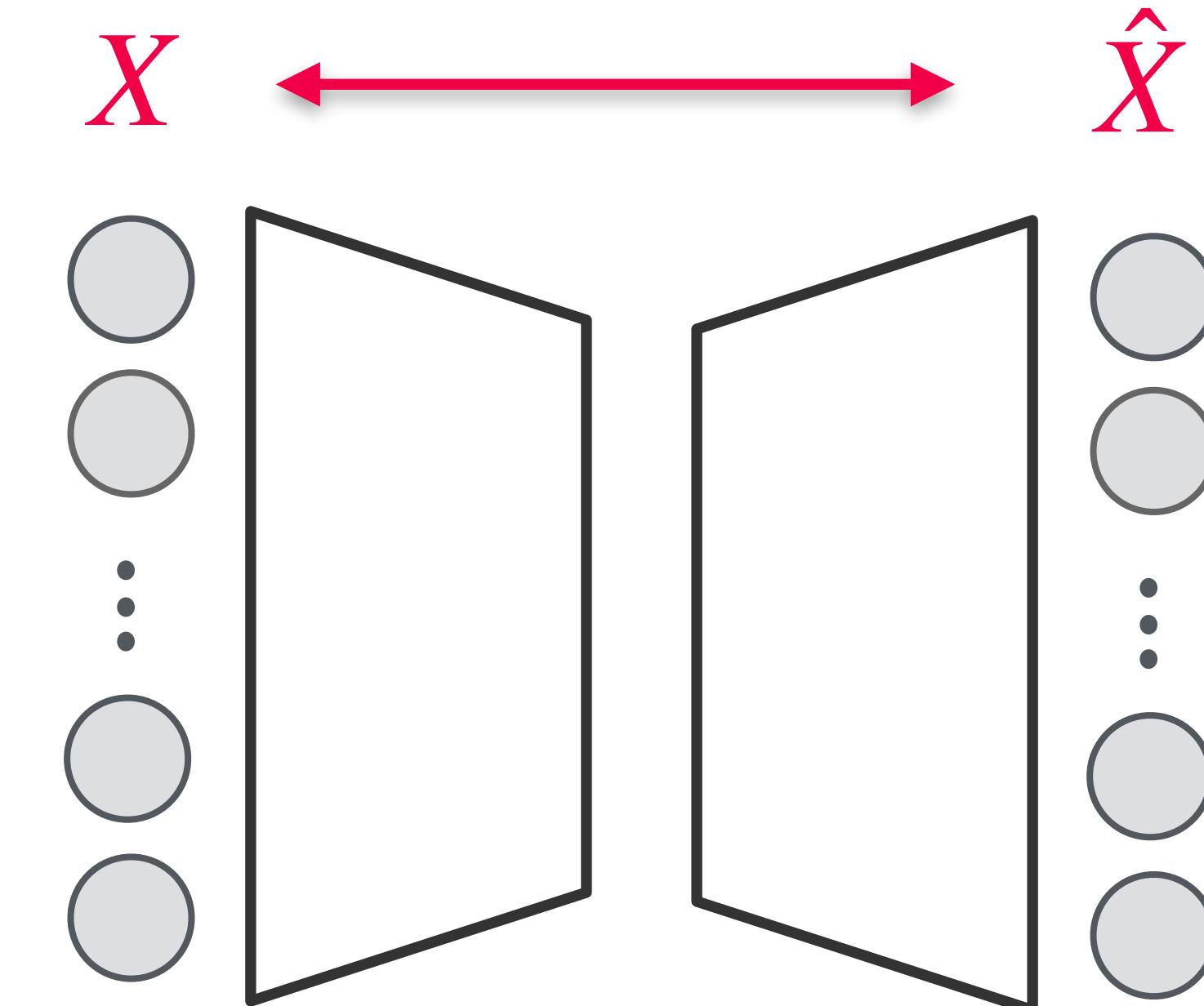
2 - Autoencoders

Supervised vs. unsupervised



Supervised approach

we have discrete (\rightarrow classification) or continuous (\rightarrow regression) labels



Unsupervised approach

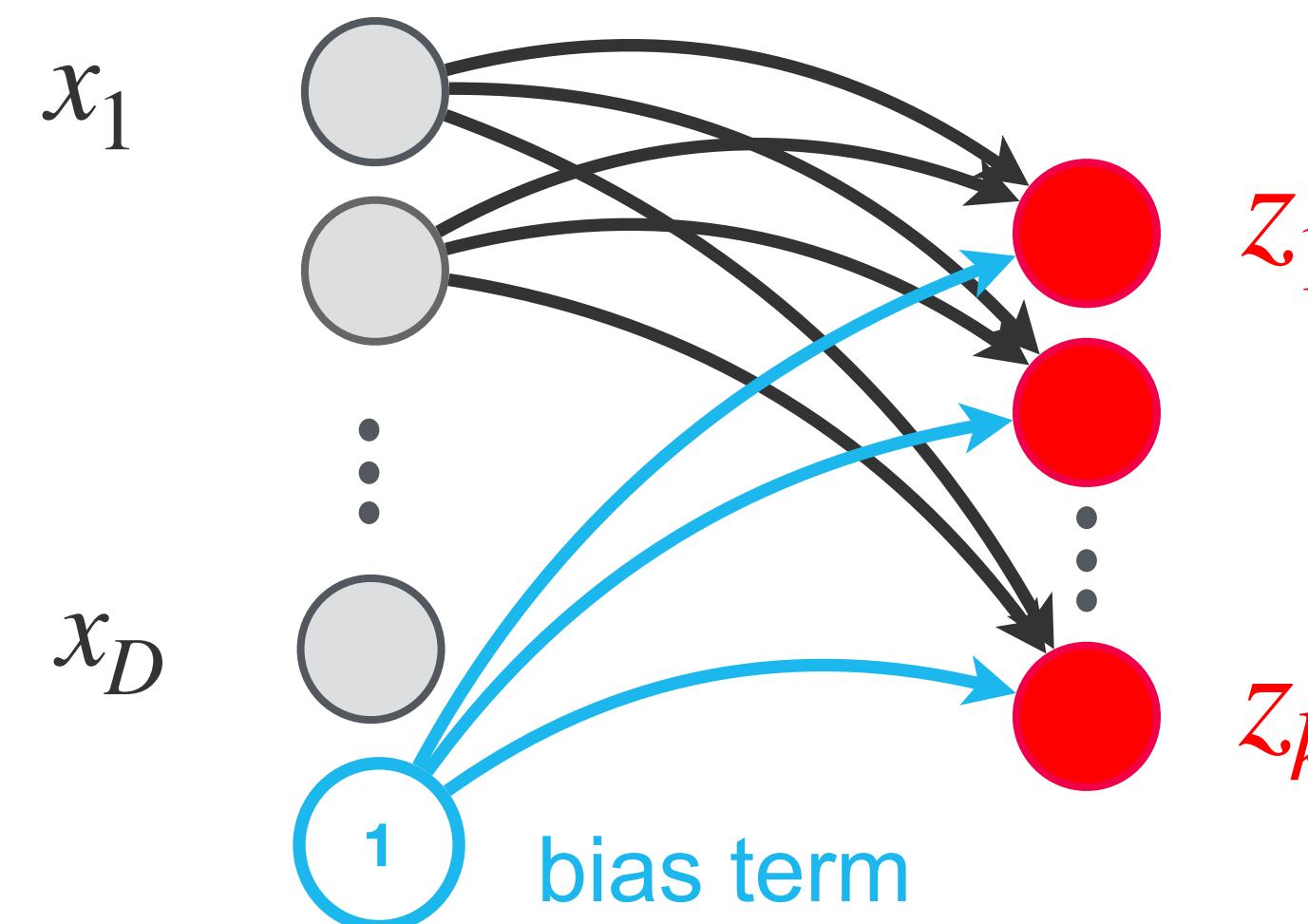
we do not have any labels for the data X

Linear layer

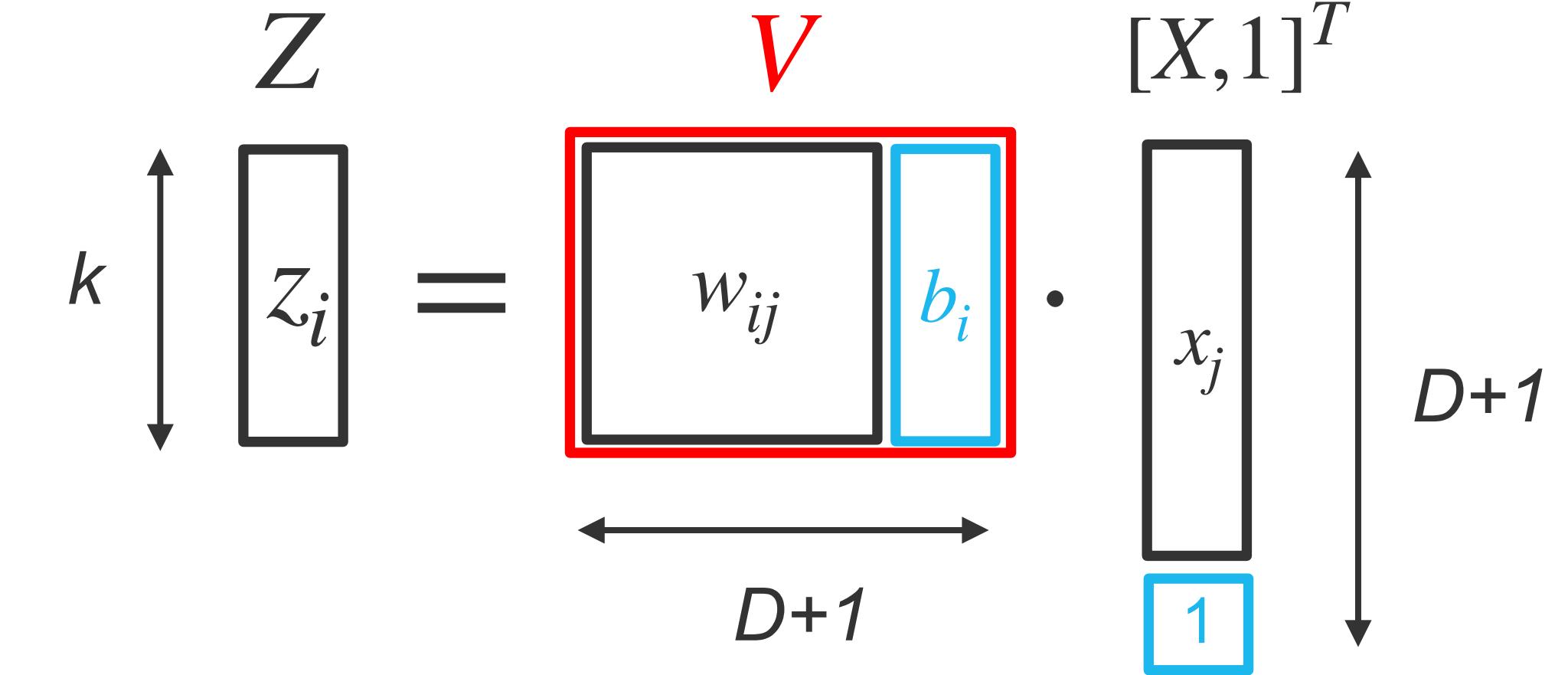
- Remember: fully-connected linear layer is equivalent to **matrix multiplication**

$$X = [x_1, x_2, \dots, x_D]^T$$

$$Z = [z_1, \dots, z_k]^T$$



$$z_i = \sum_{j=1}^D w_{ij} \cdot x_j + b_i$$



The diagram shows the matrix multiplication form of the linear layer. The output vector Z is represented as a column of height k with entries z_i . The weight matrix V is a $k \times D+1$ matrix where the columns are labeled w_{ij} (black) and b_i (blue). The input vector X is a row of width $D+1$ with entries x_j (black) and 1 (blue). The bias term b_i is added as a scalar to the result of the multiplication VX .

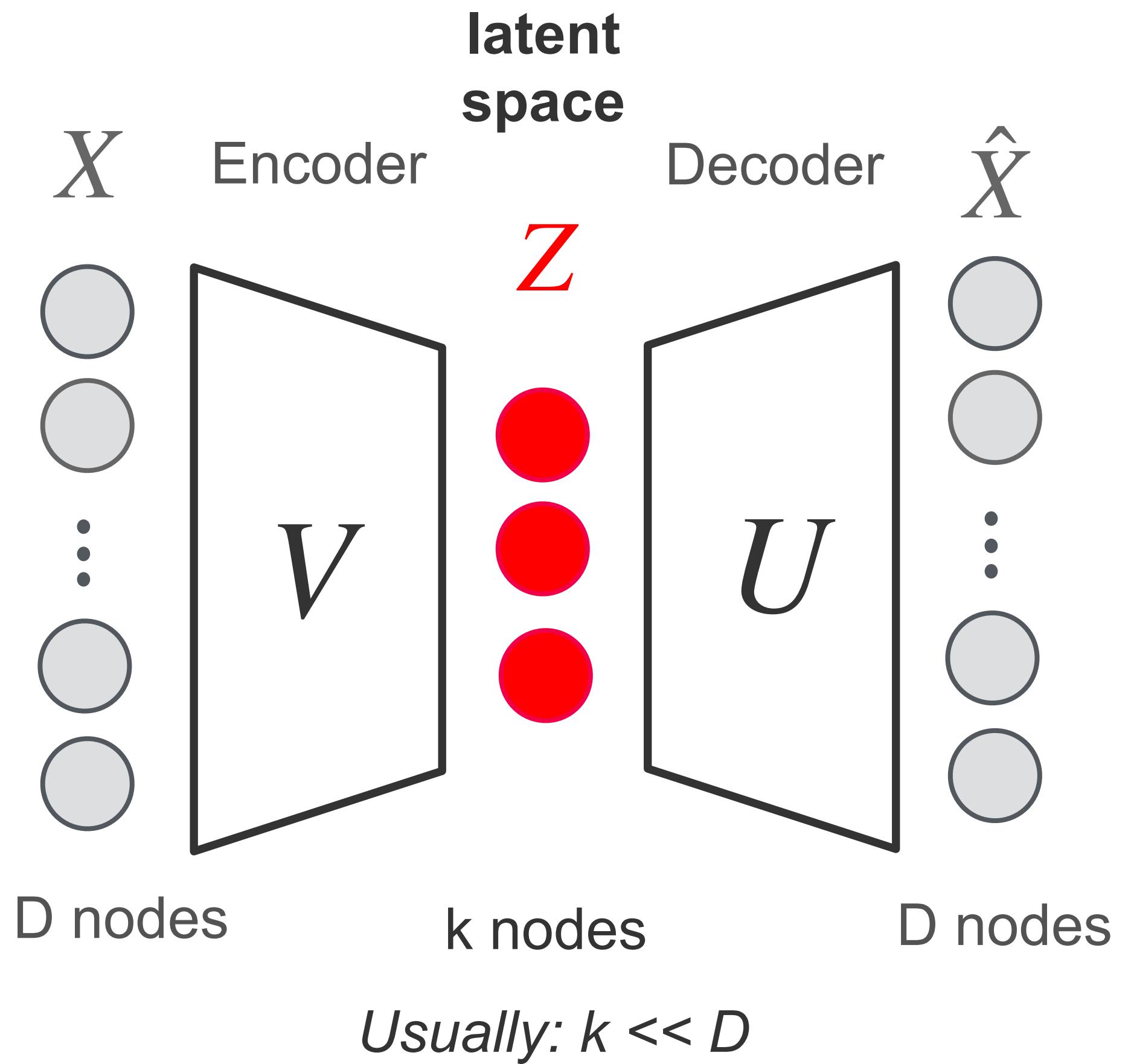
For simplification of notation, we will write (in matrix notation)

$$Z = VX$$

Shallow linear autoencoders

- Autoencoder = feed-forward neural net
- Input = X
- Task: reconstruct X through bottleneck layer
- Simplest example:
 - one input layer
 - latent hidden layer
 - output layer
 - **linear** activation
- Loss $\mathcal{L} = \|X - \hat{X}\|^2$

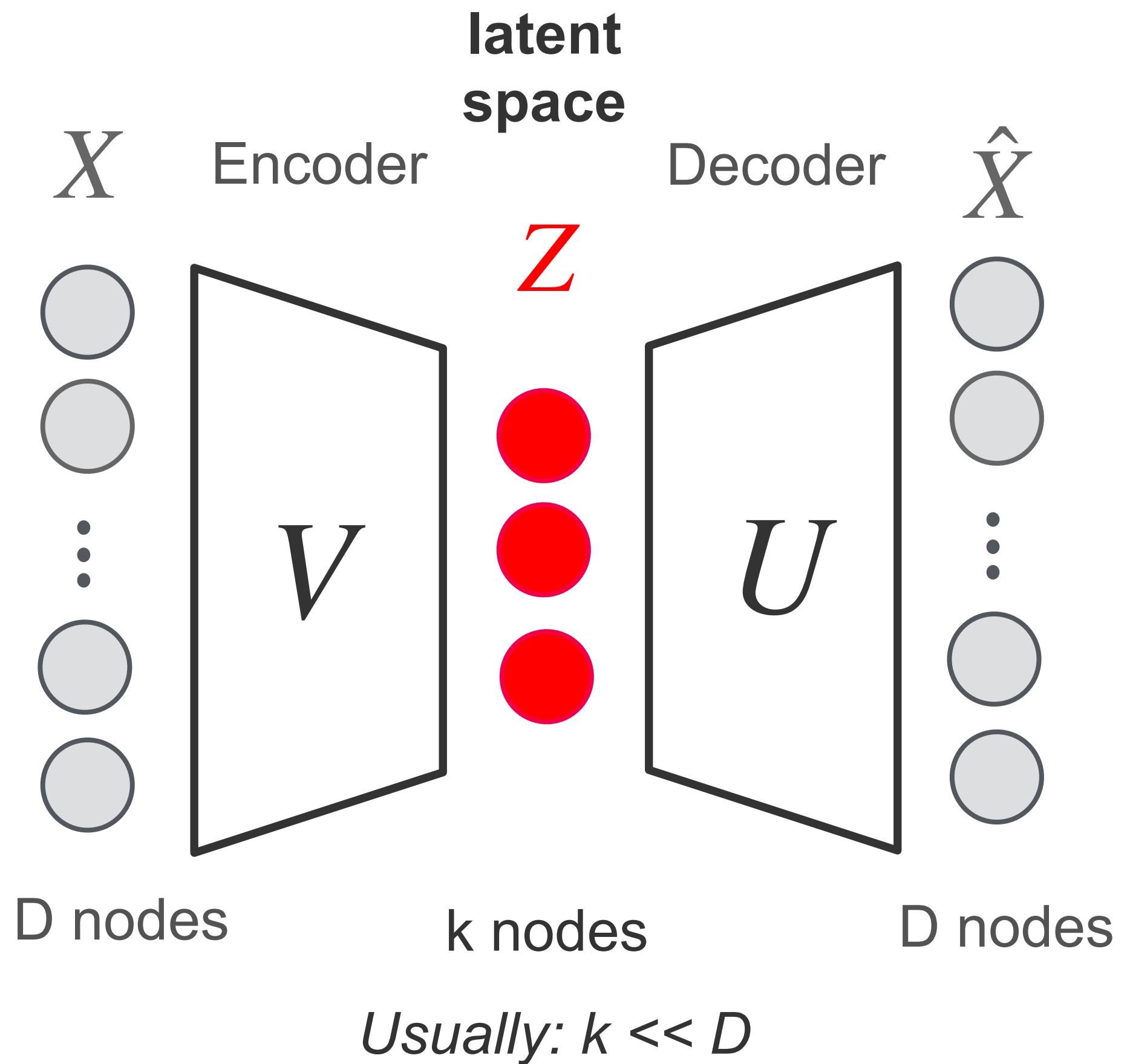
$$\hat{X} = UVX$$



This is the **reconstruction loss**

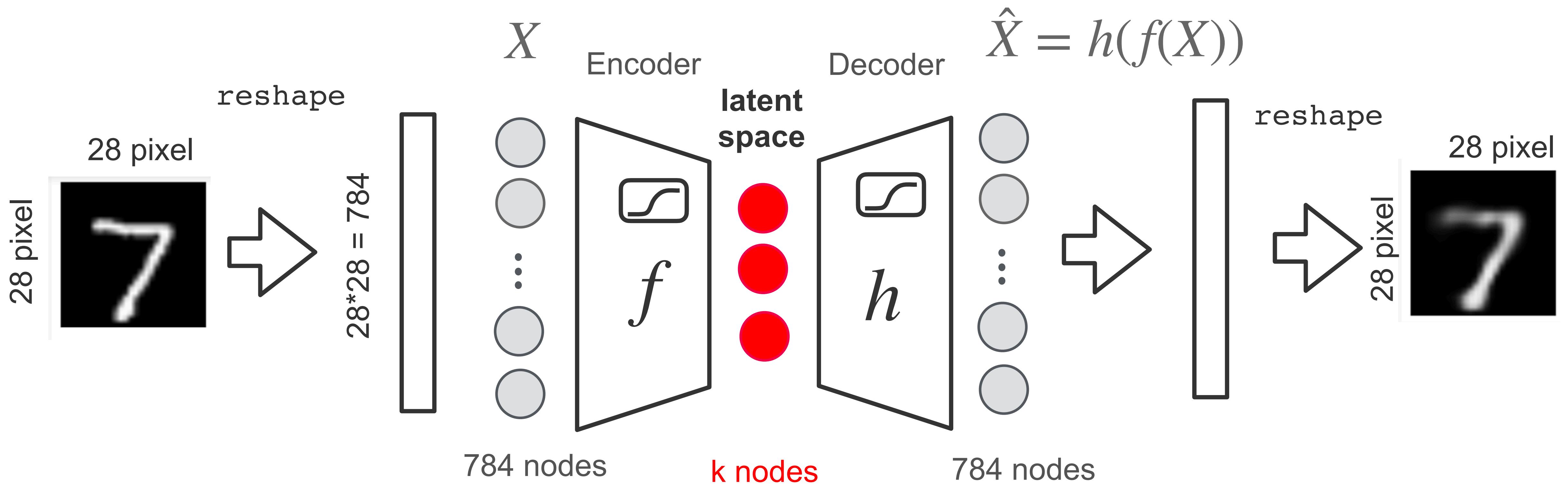
Relation to PCA

- Input matrix X
- In PCA, the original coordinates are rotated in a new coordinate system defined by the principal components
 $Z = VX$
- V is a **rotation** matrix
- then $\hat{X} = UZ = V^TZ$
- hence $\hat{X} = V^TVX$
- since V is a rotation matrix: $V^TV = I$
- hence $\hat{X} = X$



Example non-linear AE

- We use MNIST image data of hand-written digits
- This is a **shallow autoencoder** : just one hidden (latent) layer.
- Sigmoid activation functions (non-linear!)

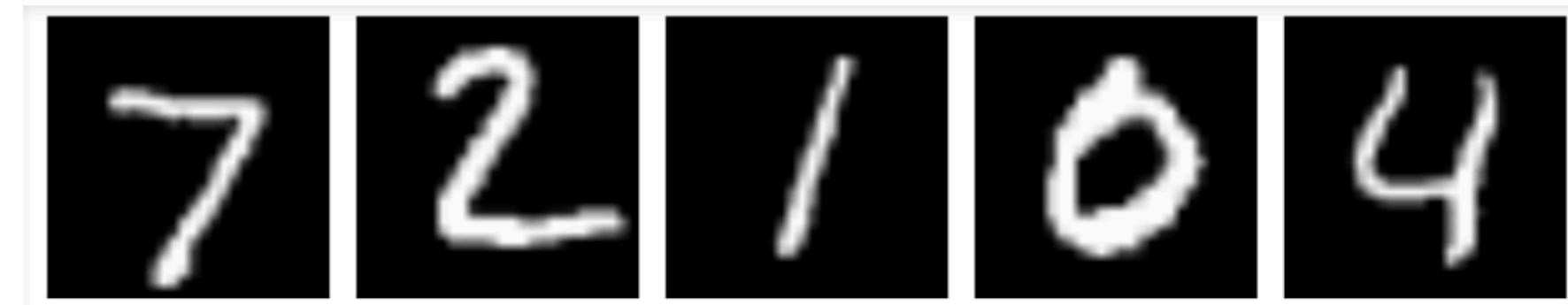


Implementation

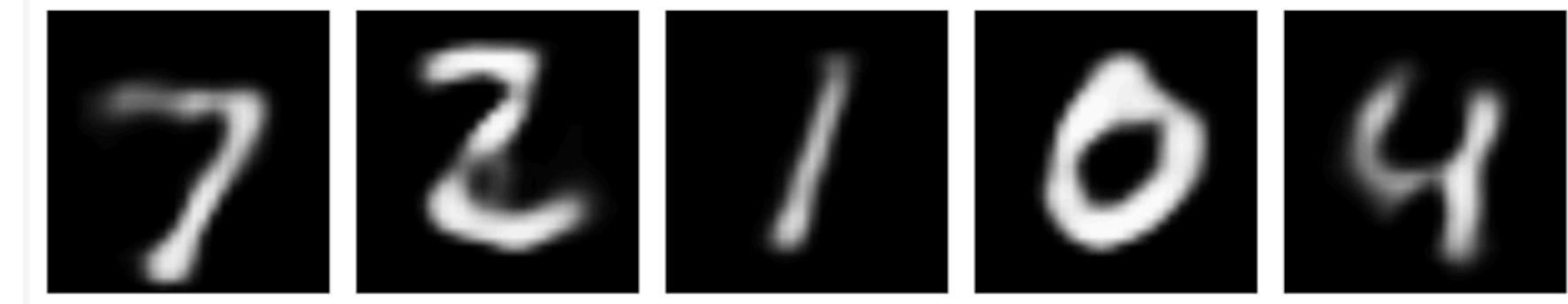
```
class AE_shallow(nn.Module):
    def __init__(self, **kwargs):
        super().__init__()
        self.encoder_hidden_layer = nn.Linear(
            in_features=kwargs["input_shape"], out_features=kwargs["z_size"]
        )
        self.decoder_output_layer = nn.Linear(
            in_features=kwargs["z_size"], out_features=kwargs["input_shape"]
        )

    def forward(self, features):
        code = self.encoder_hidden_layer(features)
        code = torch.sigmoid(code)
        activation = self.decoder_output_layer(code)
        reconstructed = torch.sigmoid(activation)
        return reconstructed
```

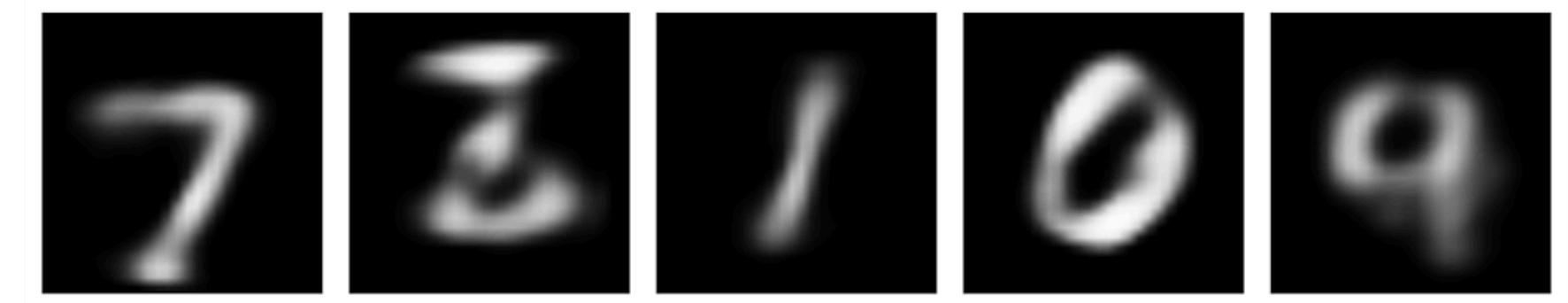
Example AE



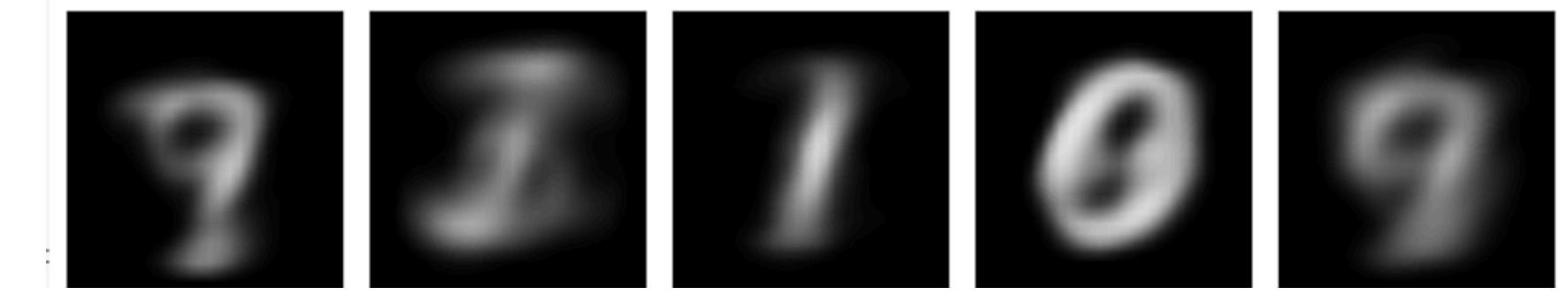
Original



Reconstructed
 $k = 20$



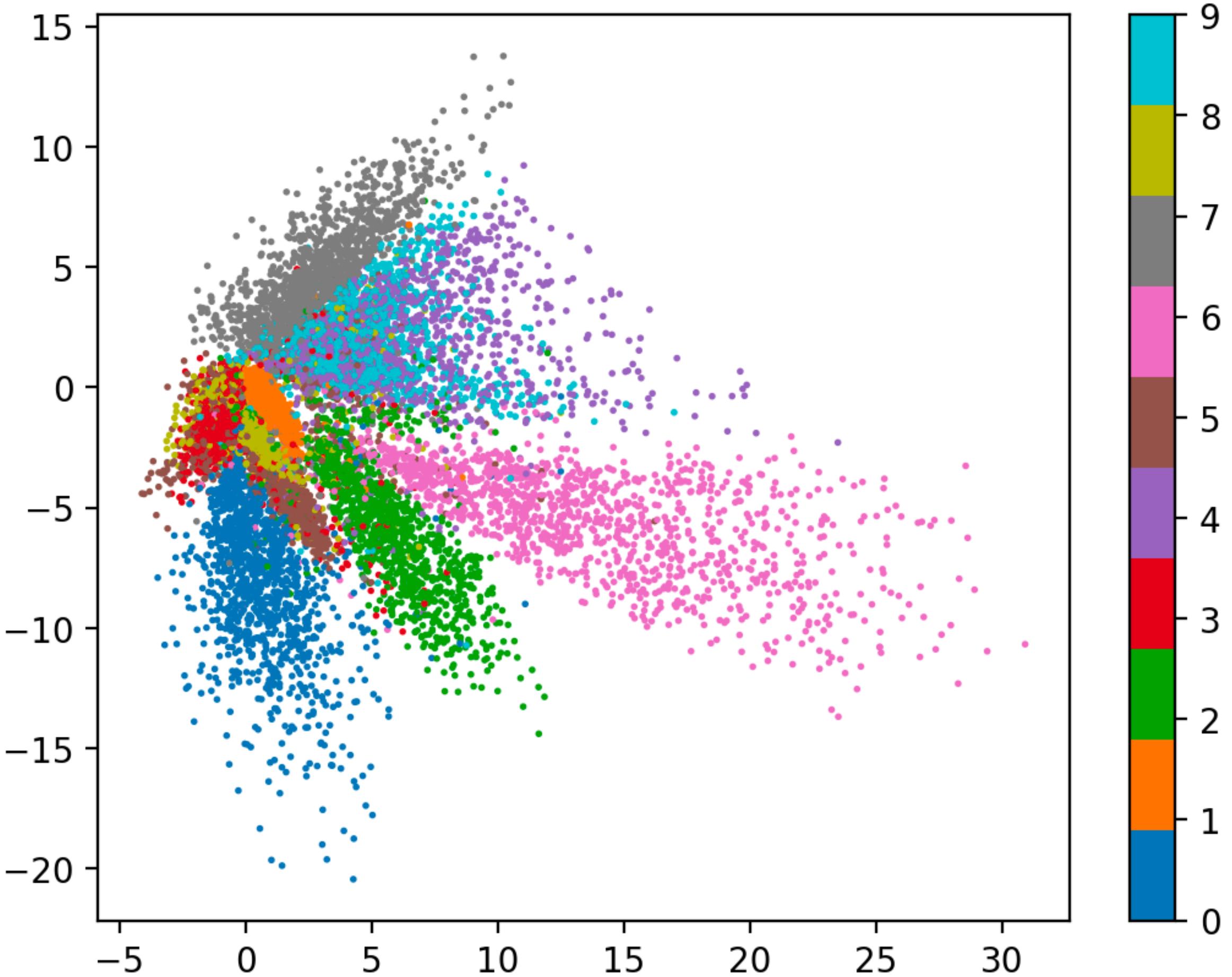
Reconstructed
 $k = 10$



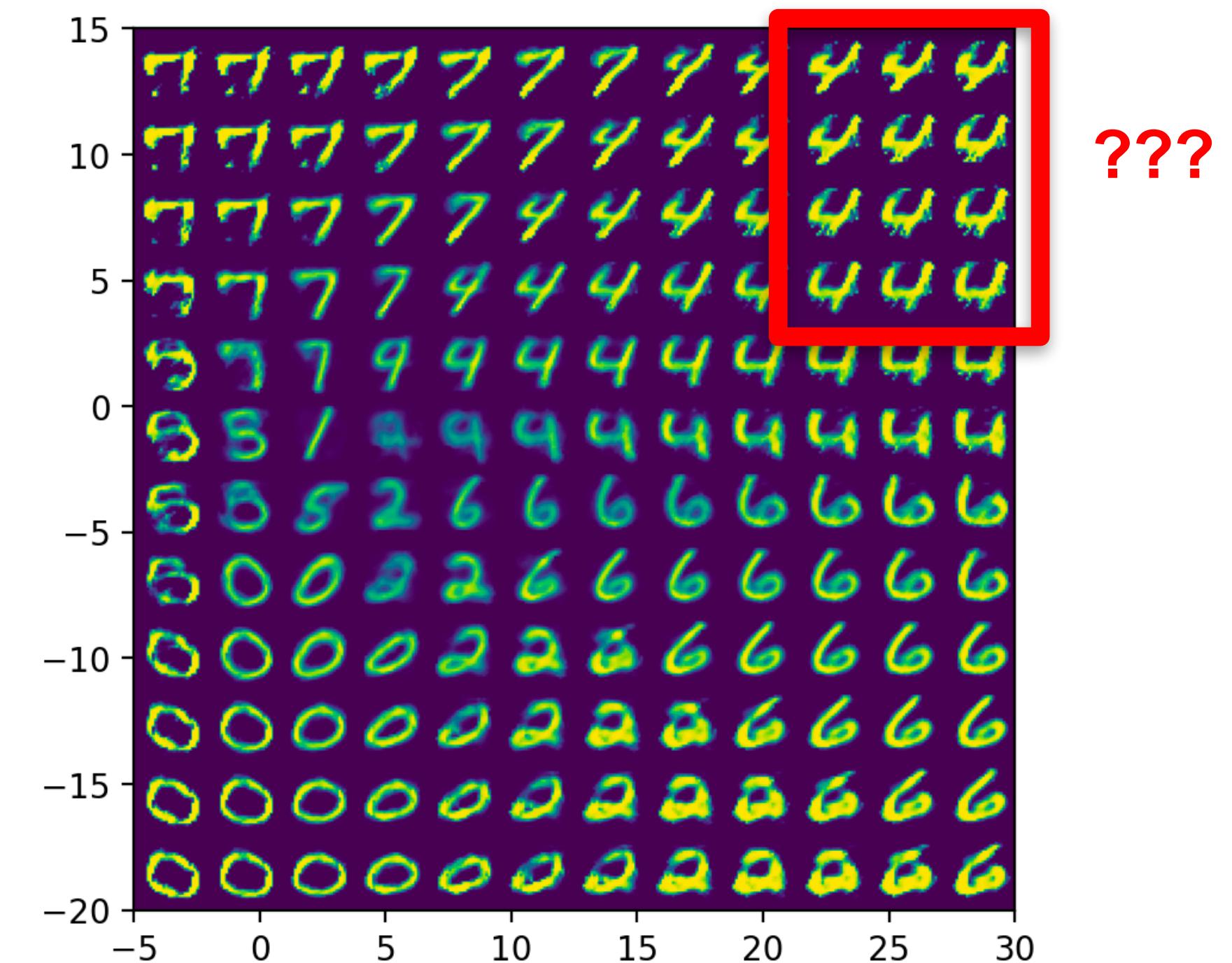
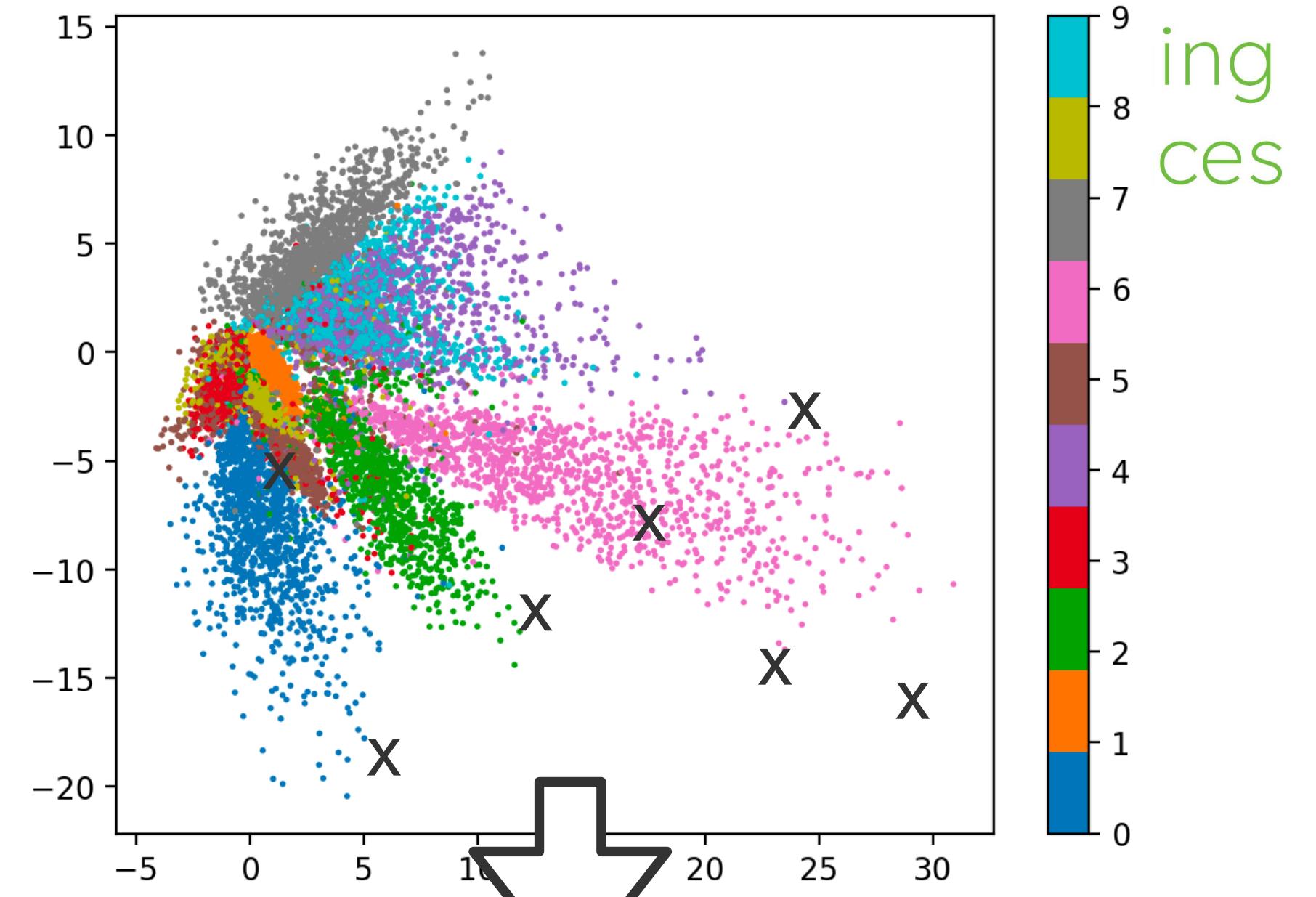
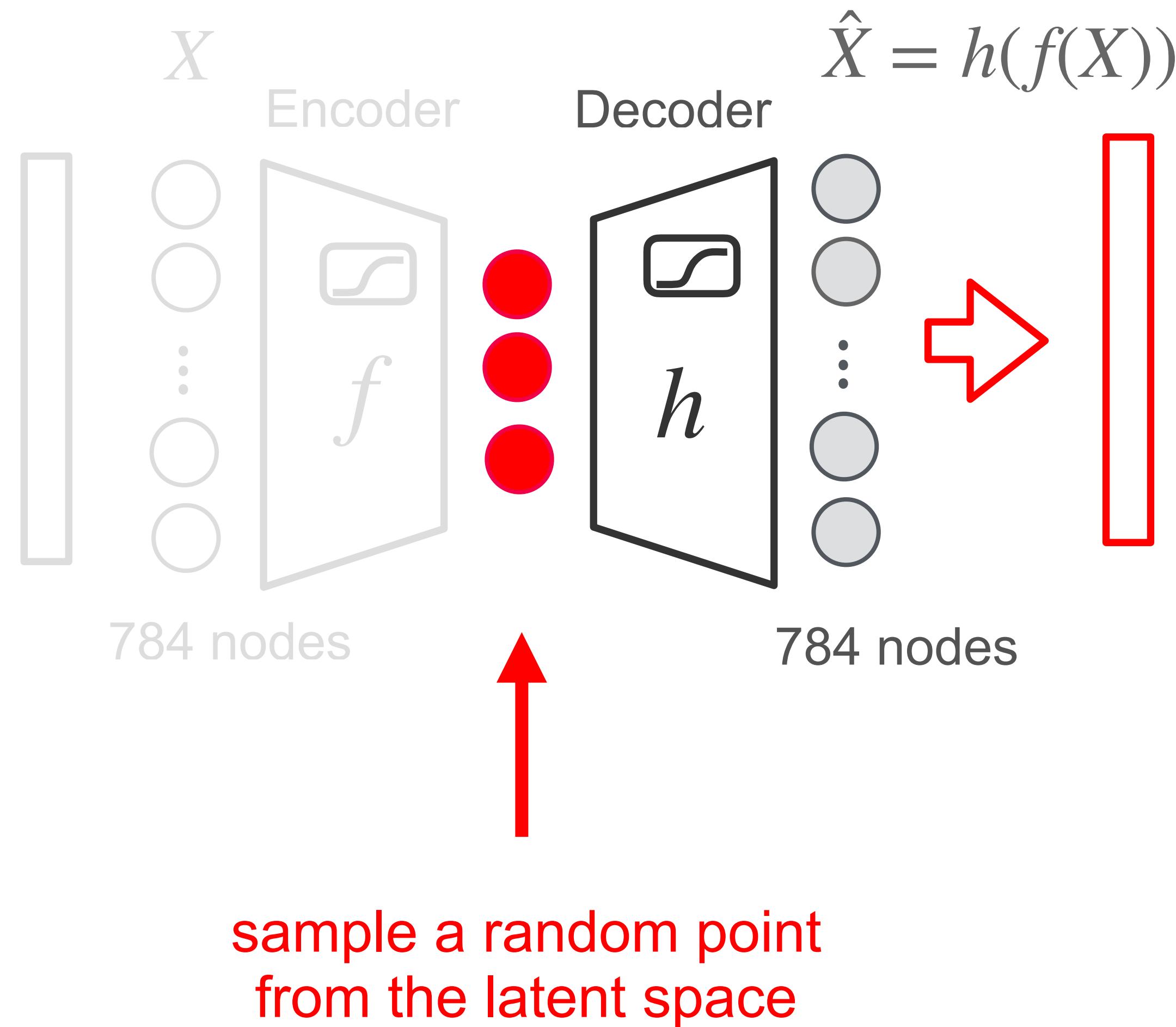
Reconstructed
 $k = 2$

Latent space

- The latent space Z is a lower dimensional representation of the input data X
- If we choose the latent dimension $k=2$, we can represent the input samples
- We can learn features from the latent space (bigger diversity in digit '6', digits '4' and '9' are close in latent space,...)
- even if $k>2$, we can project the latent space in 2d (UMAP, t-SNE,...)

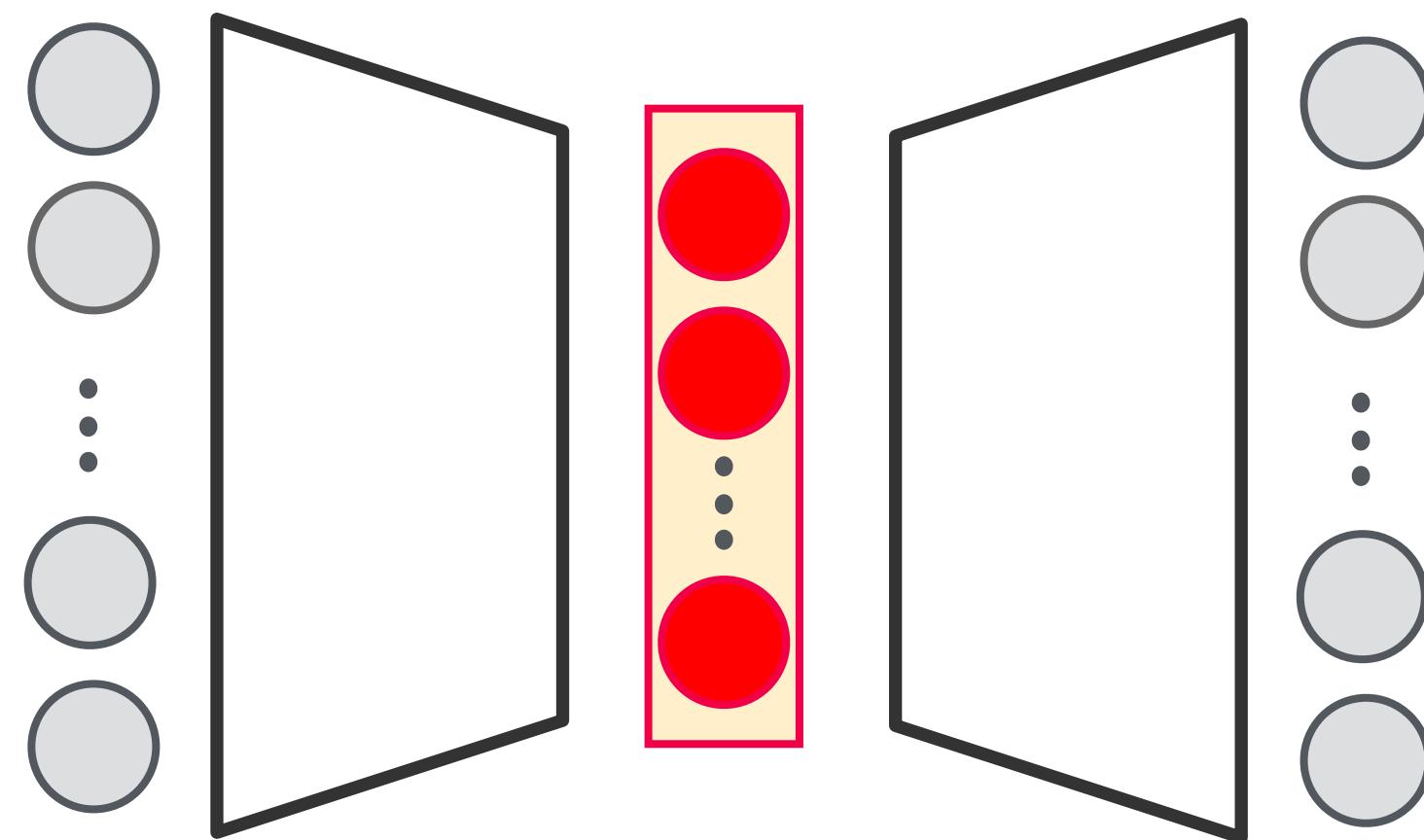


Sampling from latent space

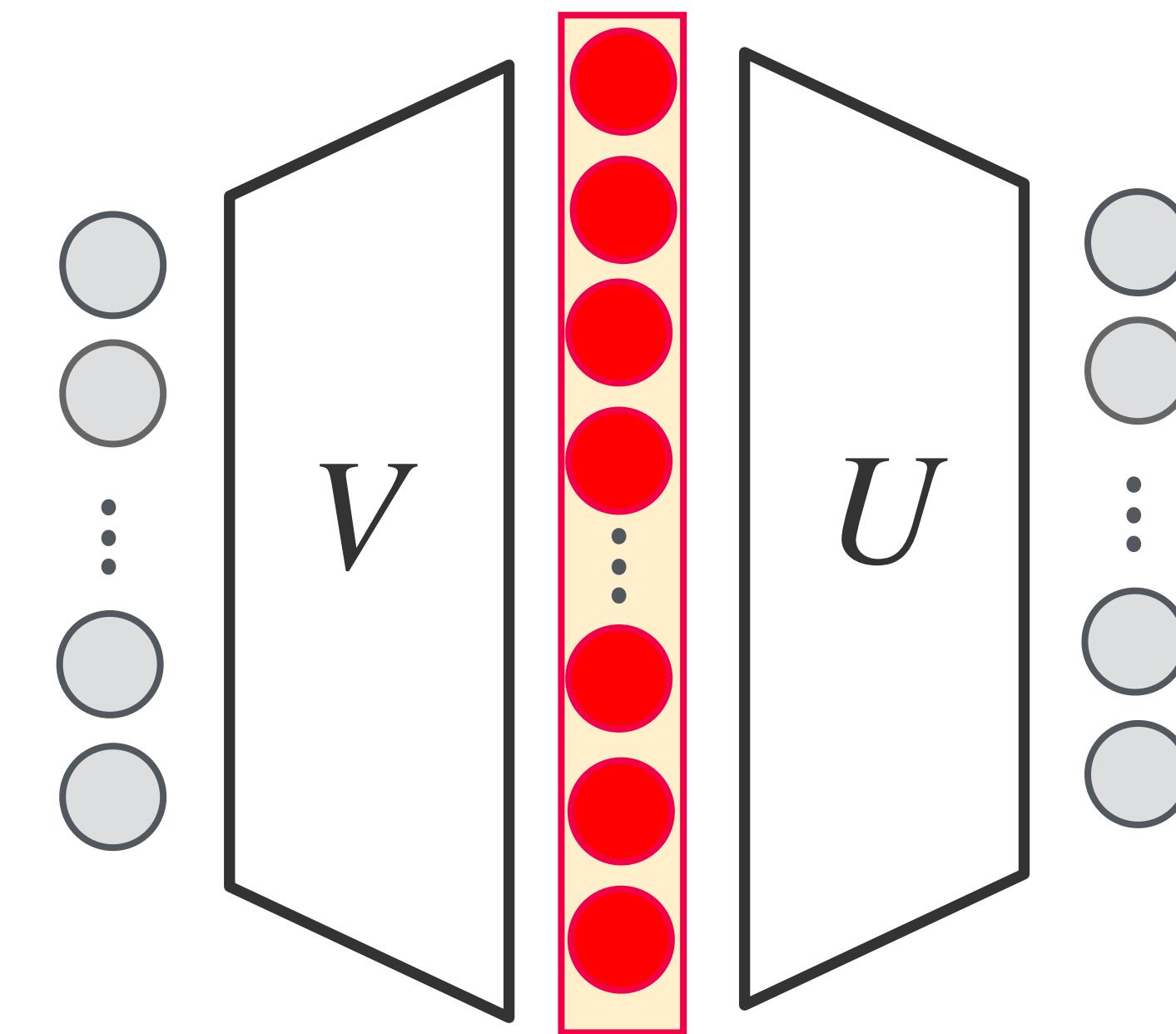


Over/Undercomplete AE

Undercomplete $k < D$



Overcomplete $k > D$

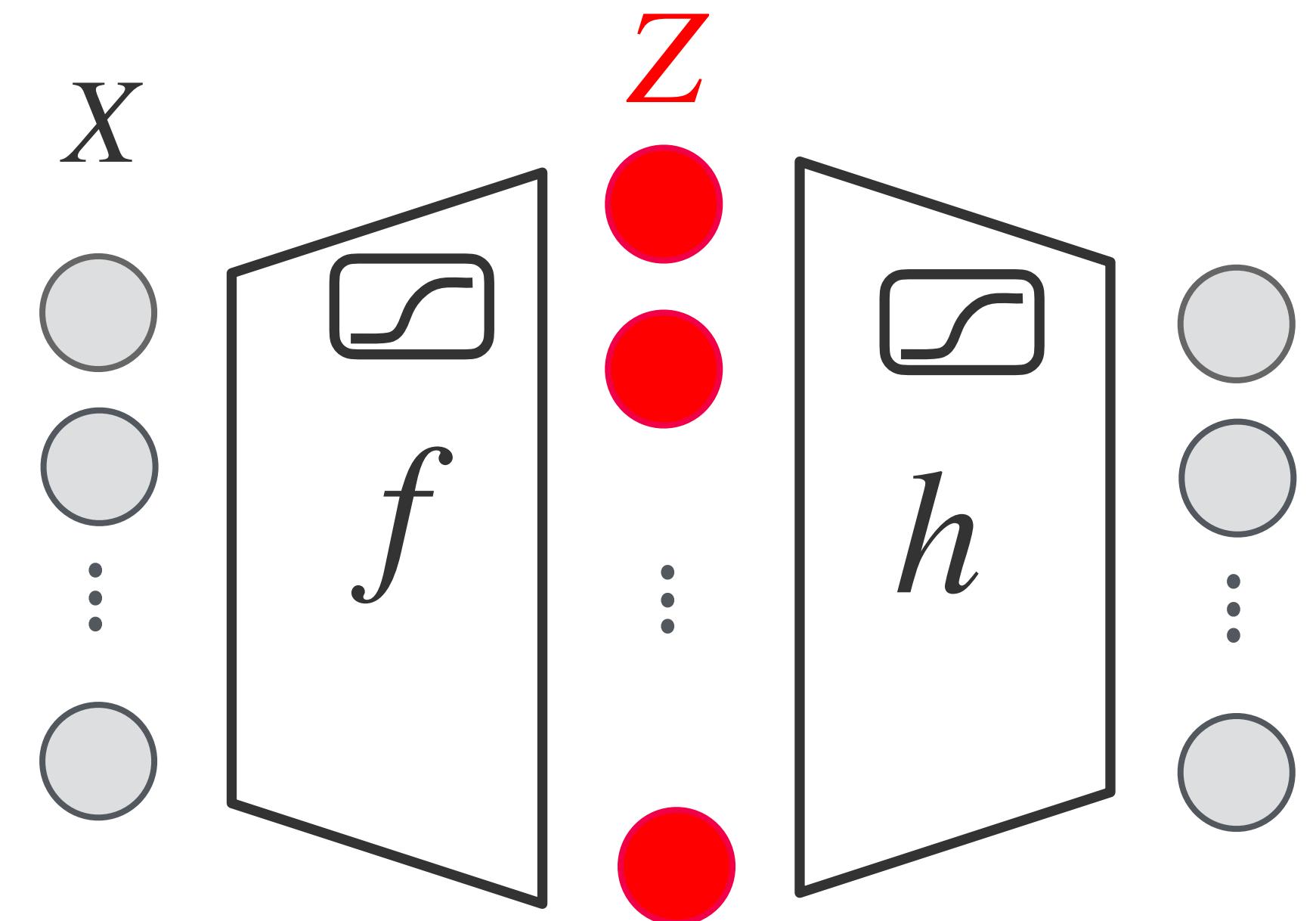


An overcomplete AE with linear layer could simply learn a **trivial mapping**

$$\hat{X} = UVX \quad \text{with} \quad UV = I$$

Regularization

- To avoid learning a trivial mapping in case of overcomplete AE, we should use additional constraints
→ **regularization**
- We can apply a **sparsity** constraint
 - penalize the norm of the activation of latent nodes
 - impose a number of zero-nodes
- Adapt the **loss function**



$$\mathcal{L} = \| X - h(f(X)) \|^2 + \lambda \cdot \Omega(Z) \quad (\Omega \geq 0)$$

λ is a **hyperparameter**

Regularization

$$L = \|X - h(f(X))\|^2 + \lambda \cdot \Omega(Z) \quad (\Omega \geq 0)$$

$\Omega(Z) = Z_{L1/L2}$ with  or 

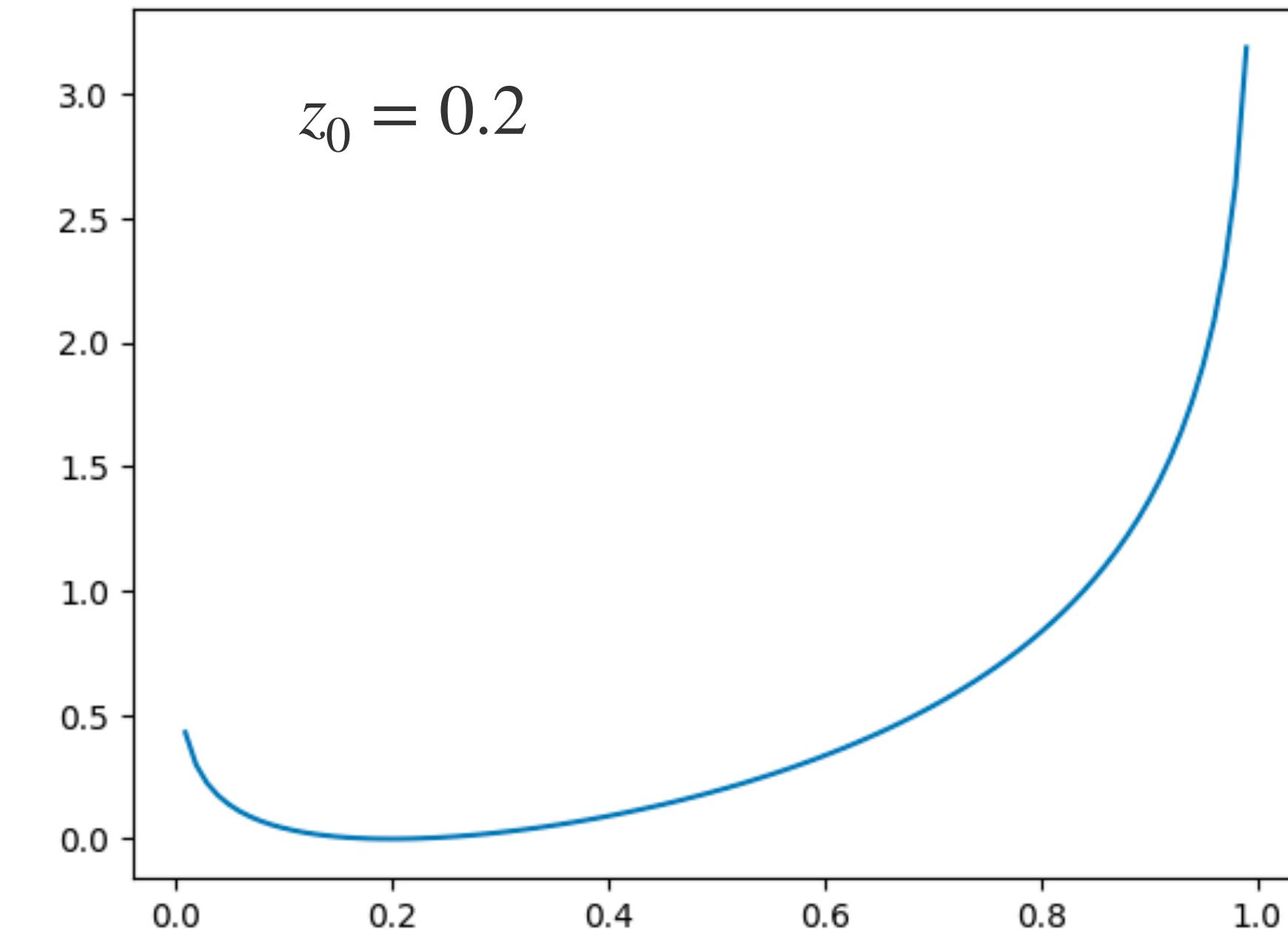
- **Sigmoid** activation function will lead to small values for the latent nodes (~ ridge regression)
- **ReLU** activation function will lead to zero latent nodes (~ LASSO regression)

Regularization

$$L = \|X - h(f(X))\|^2 + \lambda \cdot \Omega(Z) \quad (\Omega \geq 0)$$

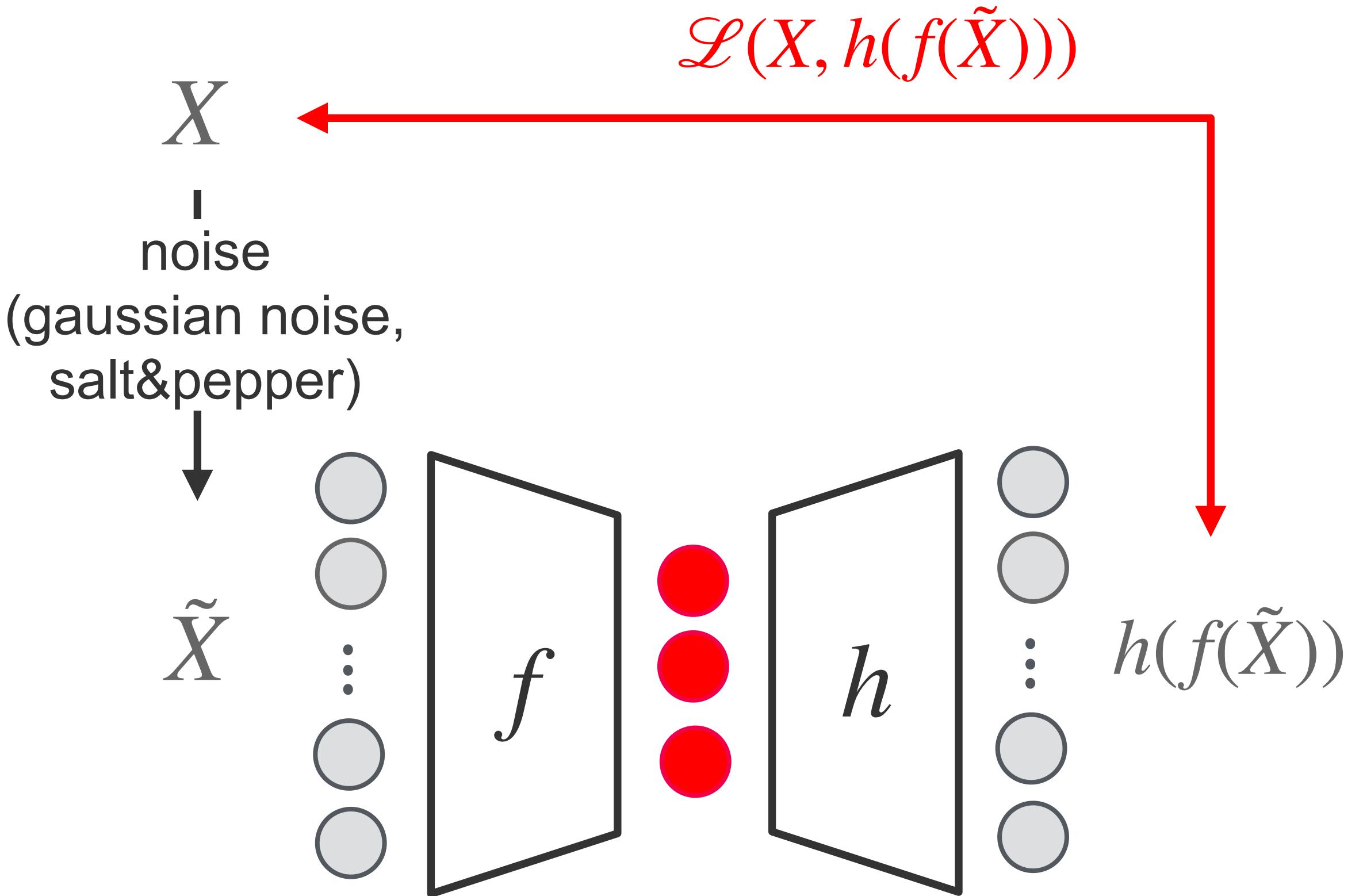
$$\Omega(Z) = \sum_{i=1}^k z_0 \log \frac{z_0}{z_i} + (1 - z_0) \log \frac{1 - z_0}{1 - z_i}$$

- z_0 is a small fixed value
- z_i is the average activation over training sample
- term is similar to the **Kullback-Leibler** divergence
- $\Omega(Z) = 0$ when $z_i = z_0$

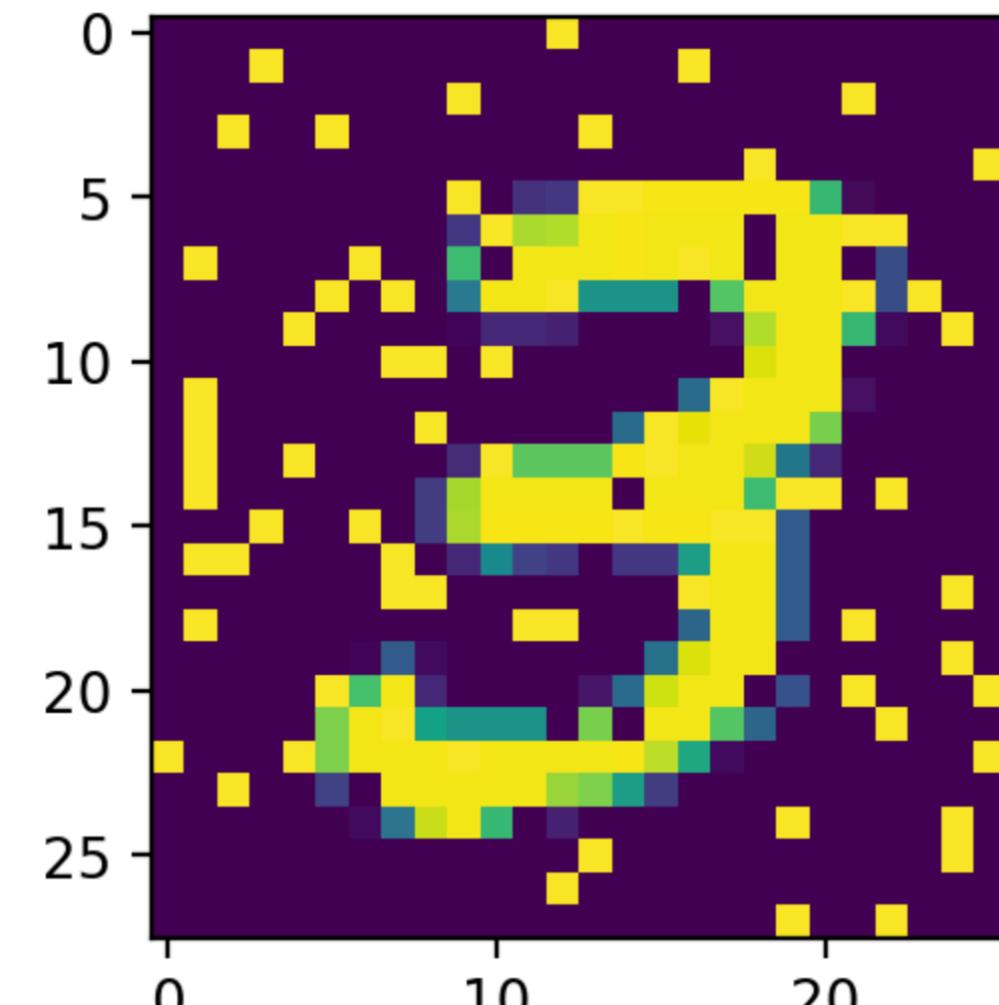
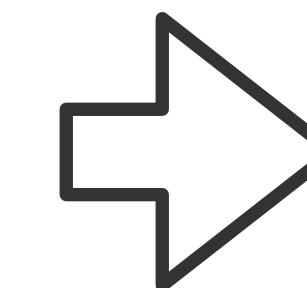
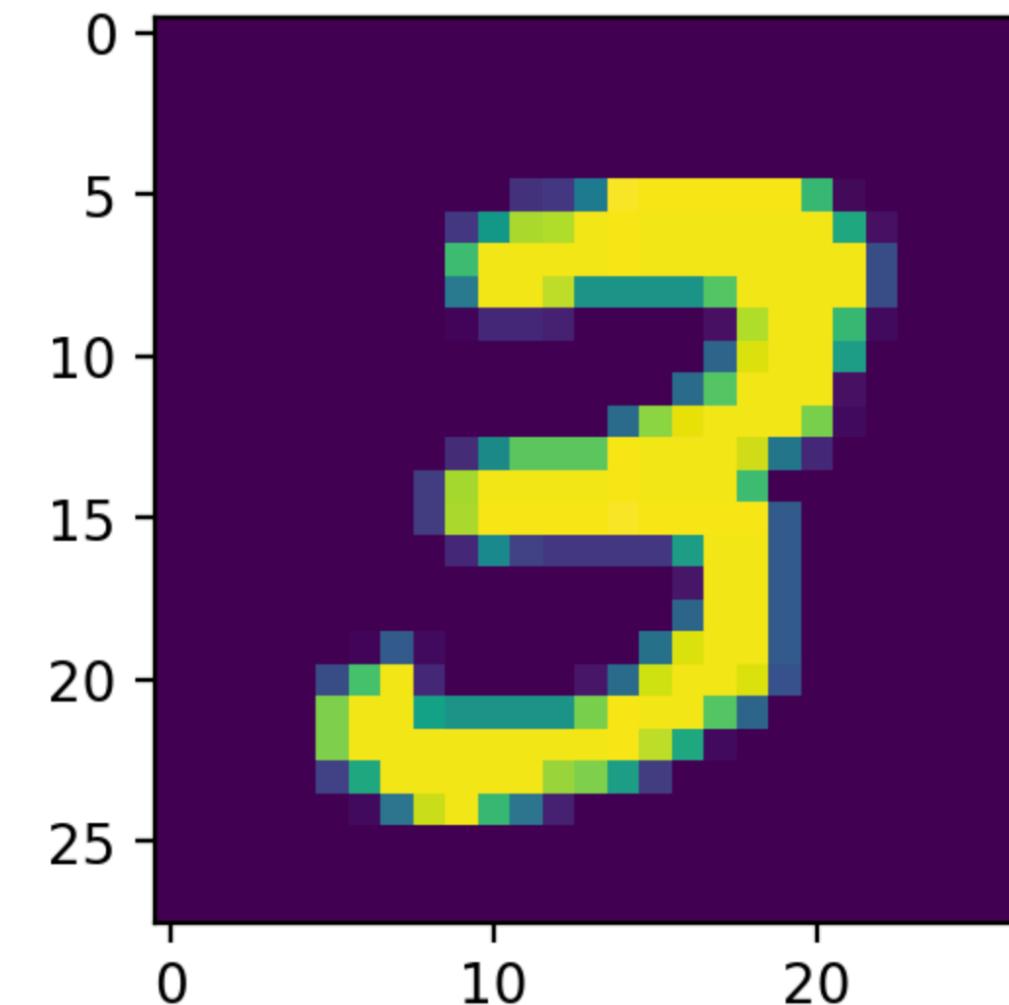


Denoising Autoencoders

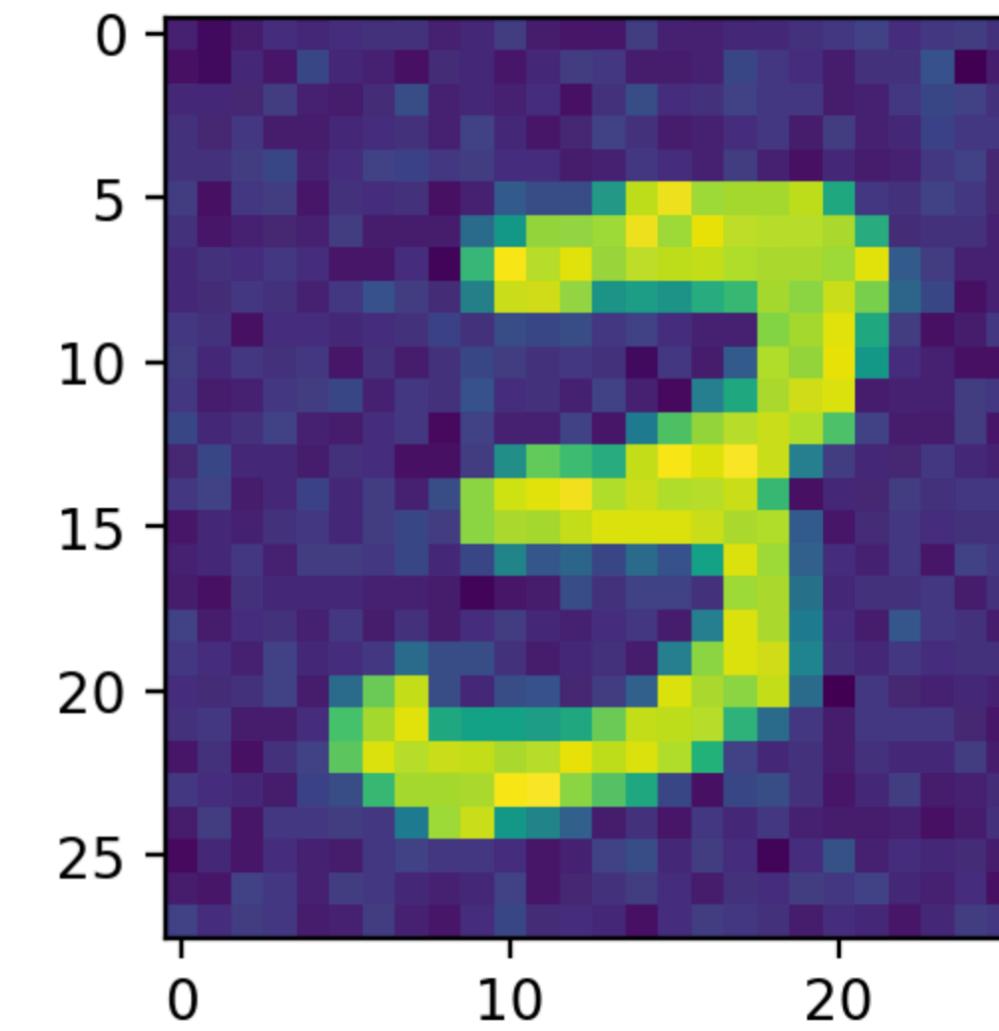
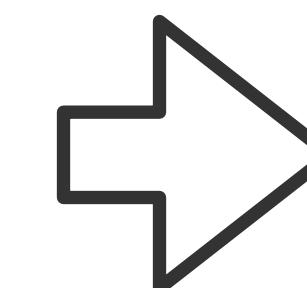
- Autoencoders are usually build to reconstruct the original input X using a loss function $\mathcal{L}(X, h(f(X)))$
- **Denoising autoencoders** are trained to reconstruct the original image X starting from a **corrupted version** \tilde{X} of the original input with a loss function $\mathcal{L}(X, h(f(\tilde{X})))$



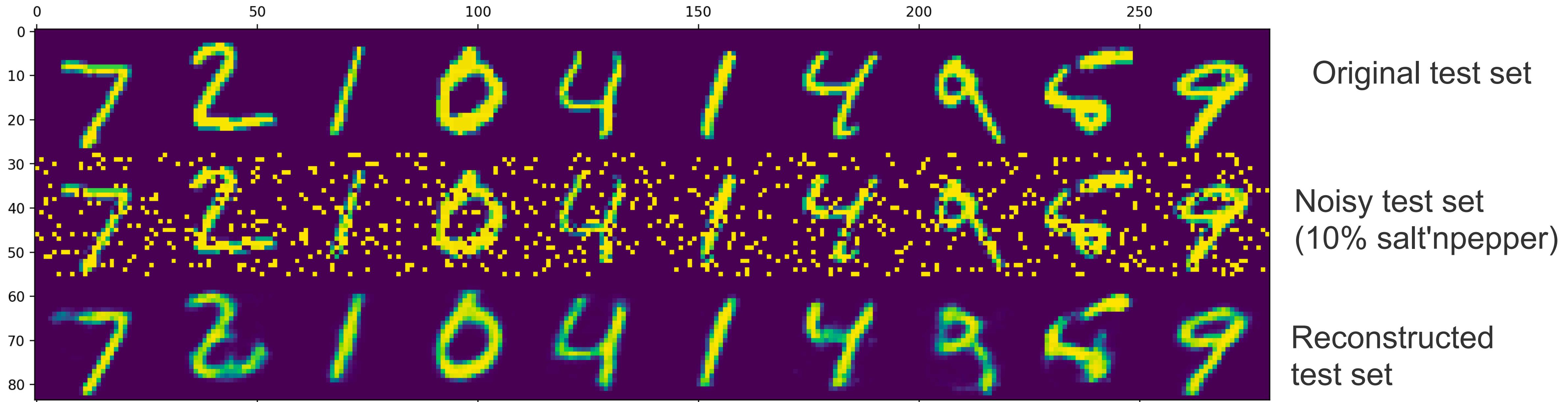
Denoising Autoencoders



put random pixels to max value ('salt')
or min value ('pepper')



add gaussian noise to all pixels



Example of reconstruction

Original test set

7	2	/	0	4	1	4	9	5	9	0	6
9	0	1	5	9	7	3	4	9	6	6	5
4	0	7	4	0	1	3	1	3	4	7	2

Noisy test set
(90% pixel deletion)

?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?

*Can you spot the mistakes
and understand why?*

Reconstructed test set

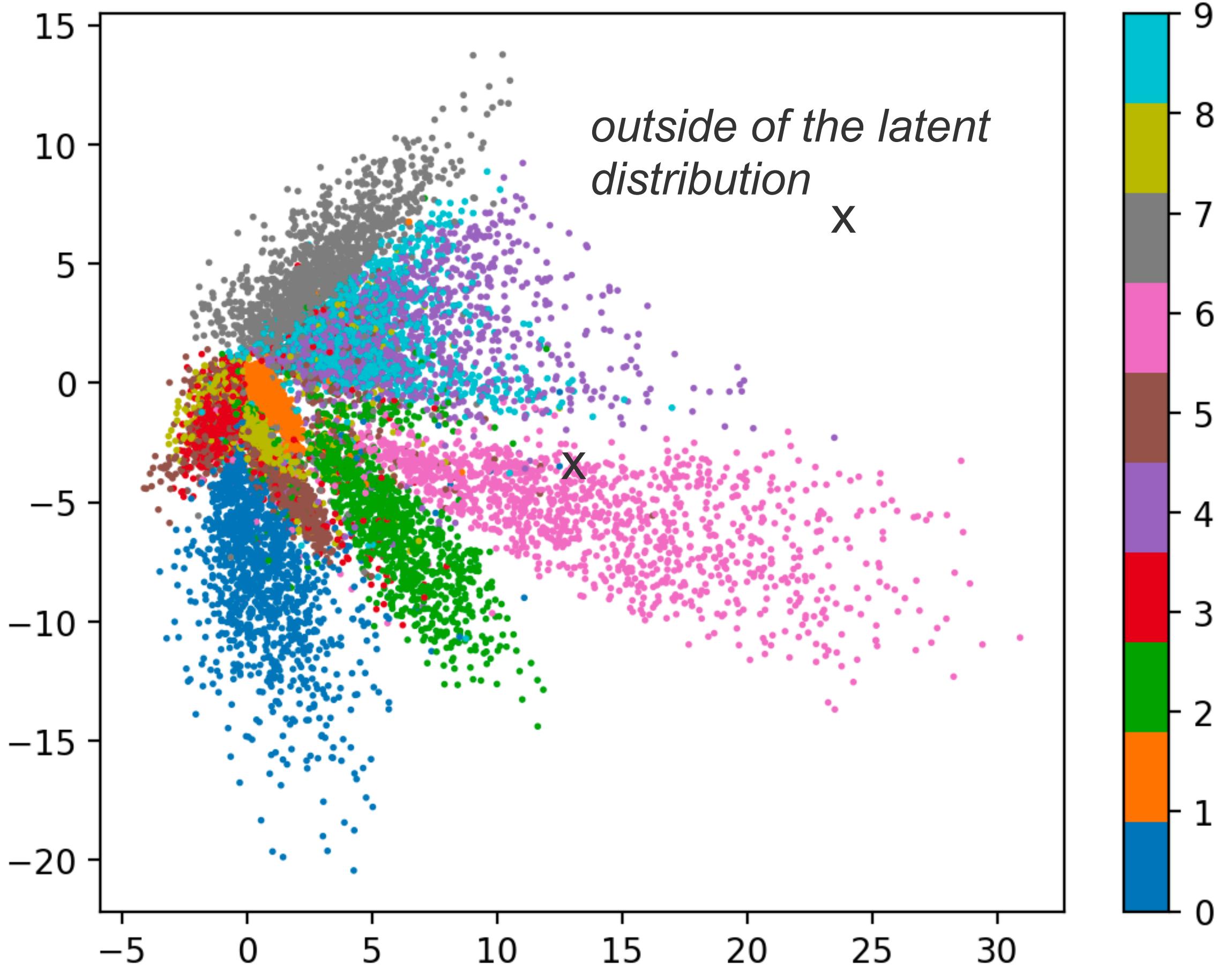
7	3	/	0	9	1	9	9	4	7	0	0
9	0	1	5	9	7	8	4	9	6	4	5
4	0	7	4	0	1	3	1	3	4	7	2

[François Fleuret]

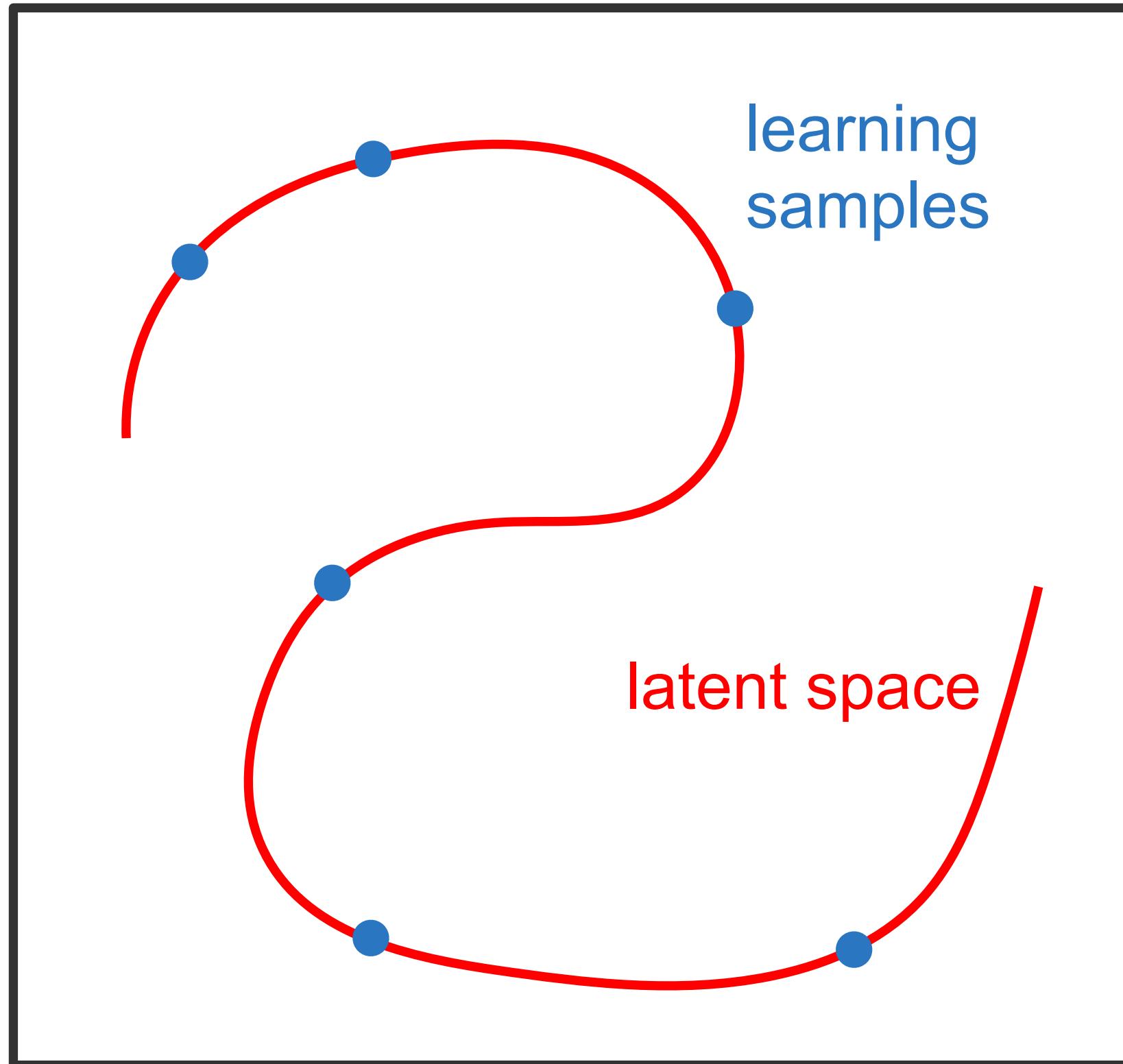
3 - Variational Autoencoders

Limitations of autoencoders

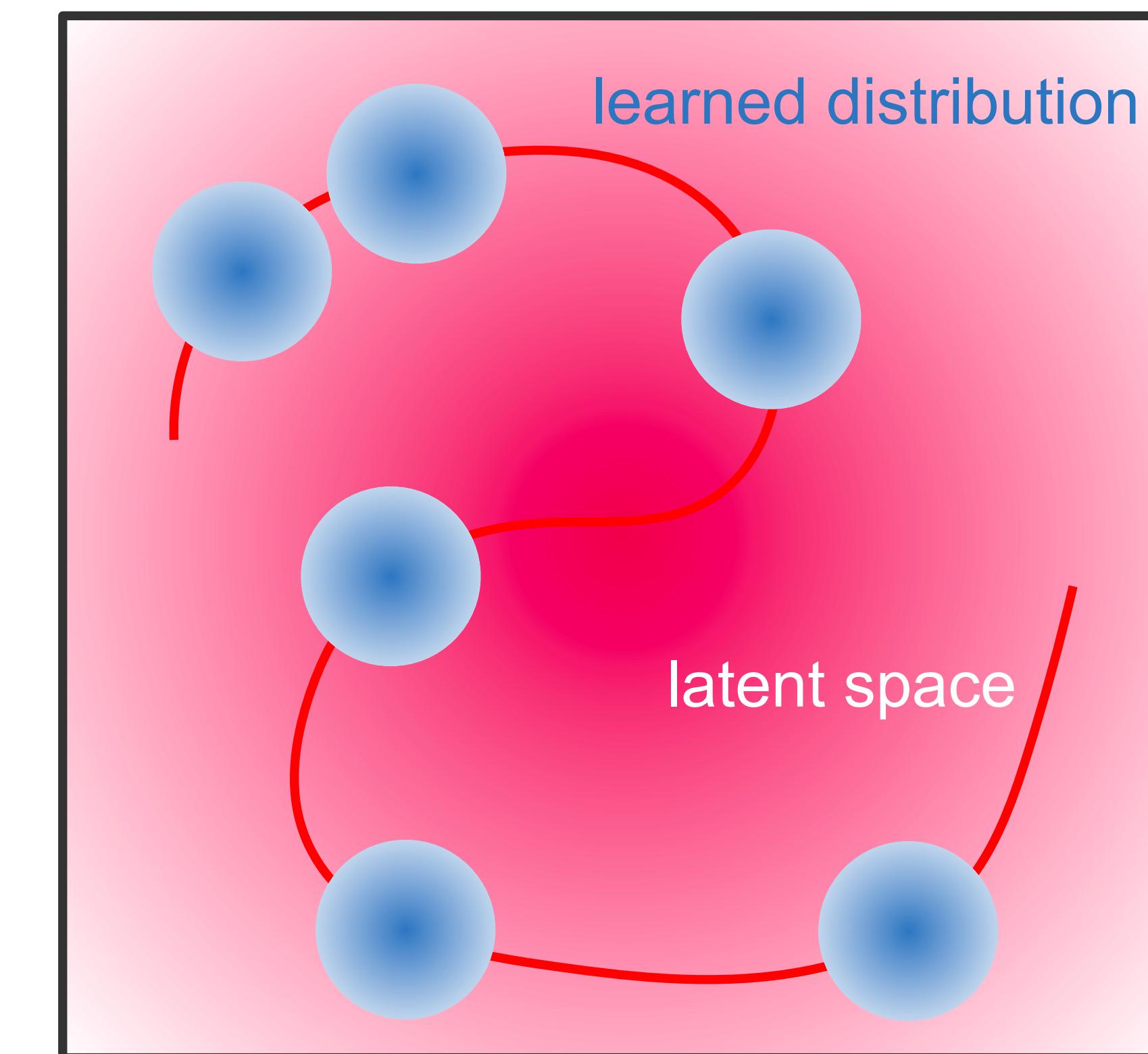
- Autoencoders define an embedding ('latent space') in a deterministic way
- The learned latent space has **zero measure** (almost all of the points in the latent space do not correspond to the data distribution)
- Hence, we **cannot sample new datapoints**, as we do not know the latent distribution
- Autoencoders are not generative models
→ **Variational autoencoders**



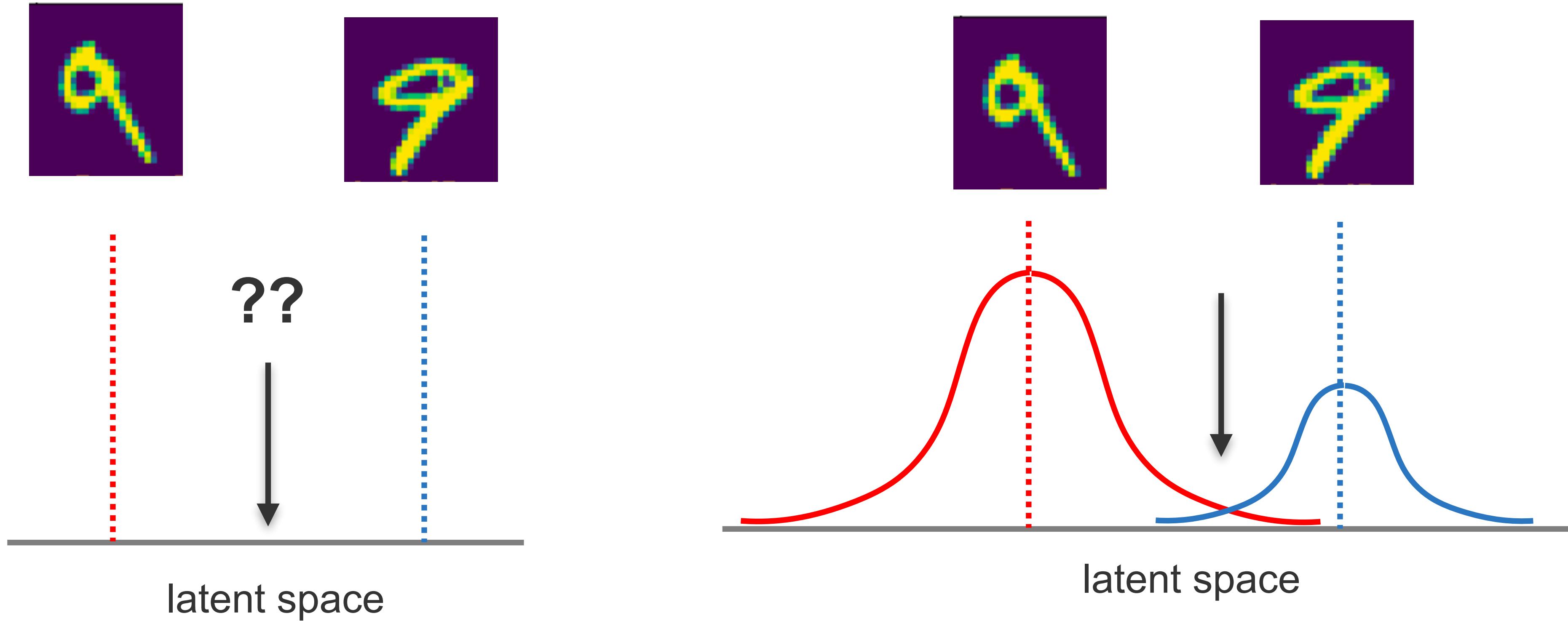
Limitations of autoencoders



autoencoder



variational autoencoder



- Autoencoders make point-estimates of the training samples
- VAEs learn a probability distribution

Variational autoencoder

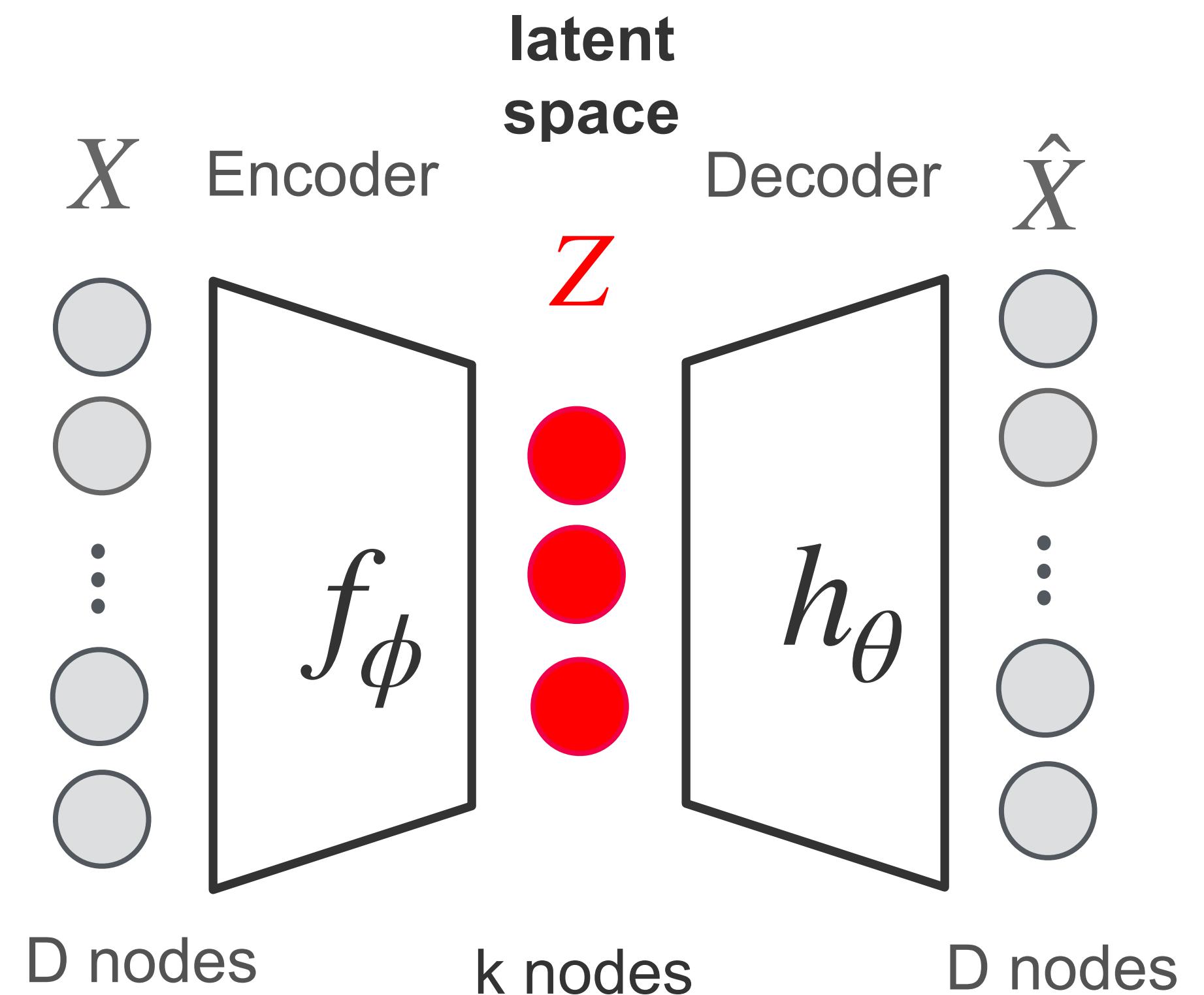
- The latent space is now sampled from a (posterior) distribution

$$Z \sim q_\phi(Z | X)$$

- New data points are then generated

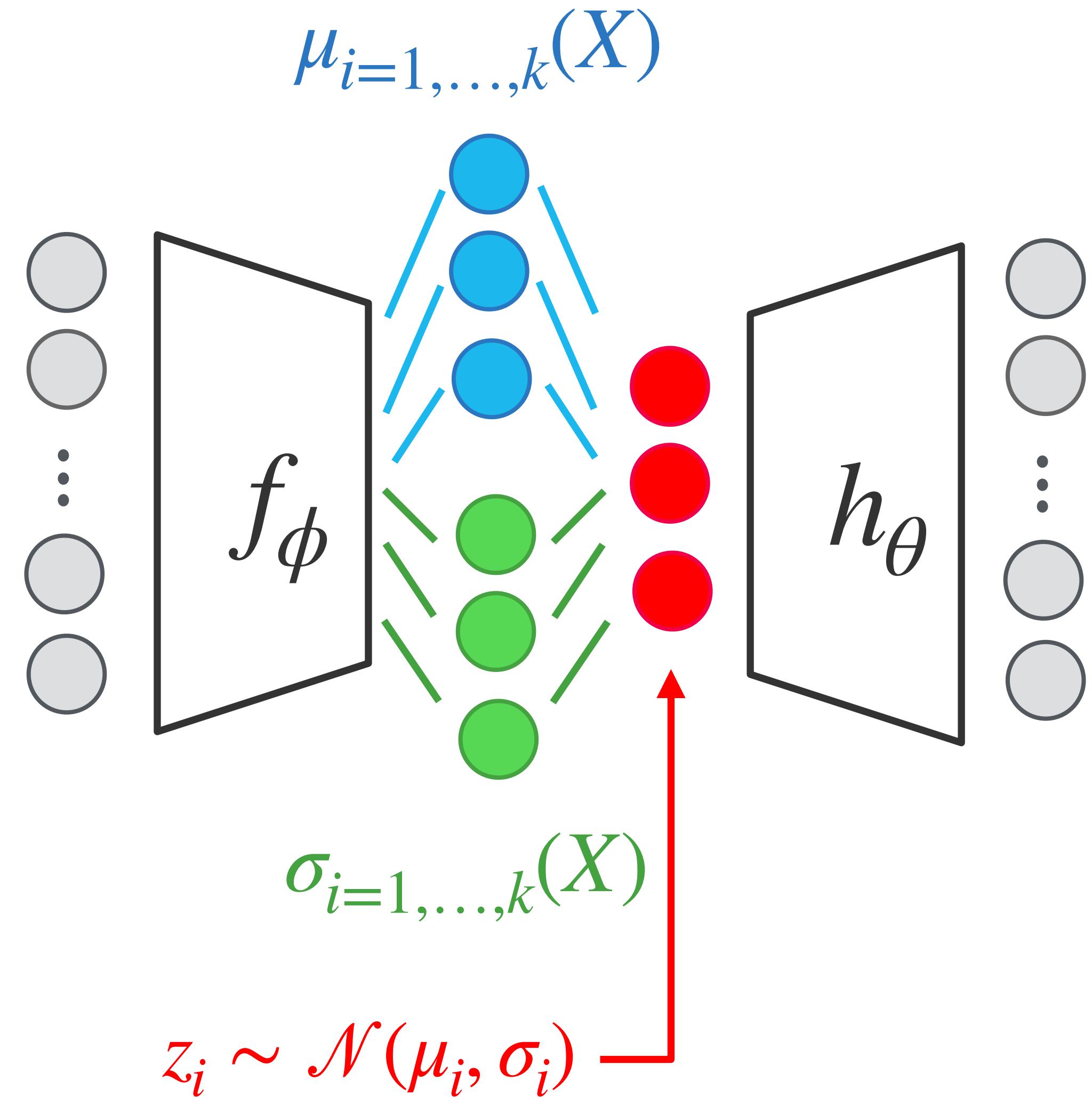
$$\hat{X} \sim p_\theta(X | Z)$$

- Hence, the latent space can be represented as **probability distributions** (for example normal distributions)



Variational autoencoder

- The model learns the parameters μ and σ of normal distributions
$$Z : z_i \sim \mathcal{N}(\mu_i, \sigma_i)$$
- New samples can be generated by sampling from the normal distribution
- VAEs belong to the class of **generative models**
- Beware that they are non-deterministic!



Loss function

- Let us look at the **loss function** that need to be minimized during training of a VAE

$$\mathcal{L}_{VAE} = - \sum_{i=1}^N \left(\boxed{\mathbb{E}_{Z^i \sim q_\phi(Z^i | X^i)} (\log p_\theta(X^i | Z^i))} - KL[q_\phi(Z^i | X^i) \| p(Z^i)] \right)$$

- The left term is the **log-likelihood** of the data X^i given the latent variable Z^i sampled from the distribution $q_\phi(Z^i | X^i)$
- This term will tend to push the reconstructed X^i towards the input vector X^i
- This is equivalent to the **reconstruction error** of the loss term in the AE

Loss function

- Let us look at the **loss function** that need to be minimized during training of a VAE

$$\mathcal{L}_{VAE} = - \sum_{i=1}^N \left(\mathbb{E}_{Z^i \sim q_\phi(Z^i | X^i)} (\log p_\theta(X^i | Z^i)) \boxed{- KL[q_\phi(Z^i | X^i) \| p(Z^i)]} \right)$$

- The right term is the **Kullback-Leibler divergence**, with $p(Z^i) \sim \mathcal{N}(0,1)$ the prior distribution of the latent variables
- KL divergence is always non-negative; it is zero if both distribution are equal
- This term forces $q_\phi(Z^i | X^i)$ to be 'not too far' from a standard normal distribution $\mathcal{N}(0,1)$

Kullback-Leibler divergence

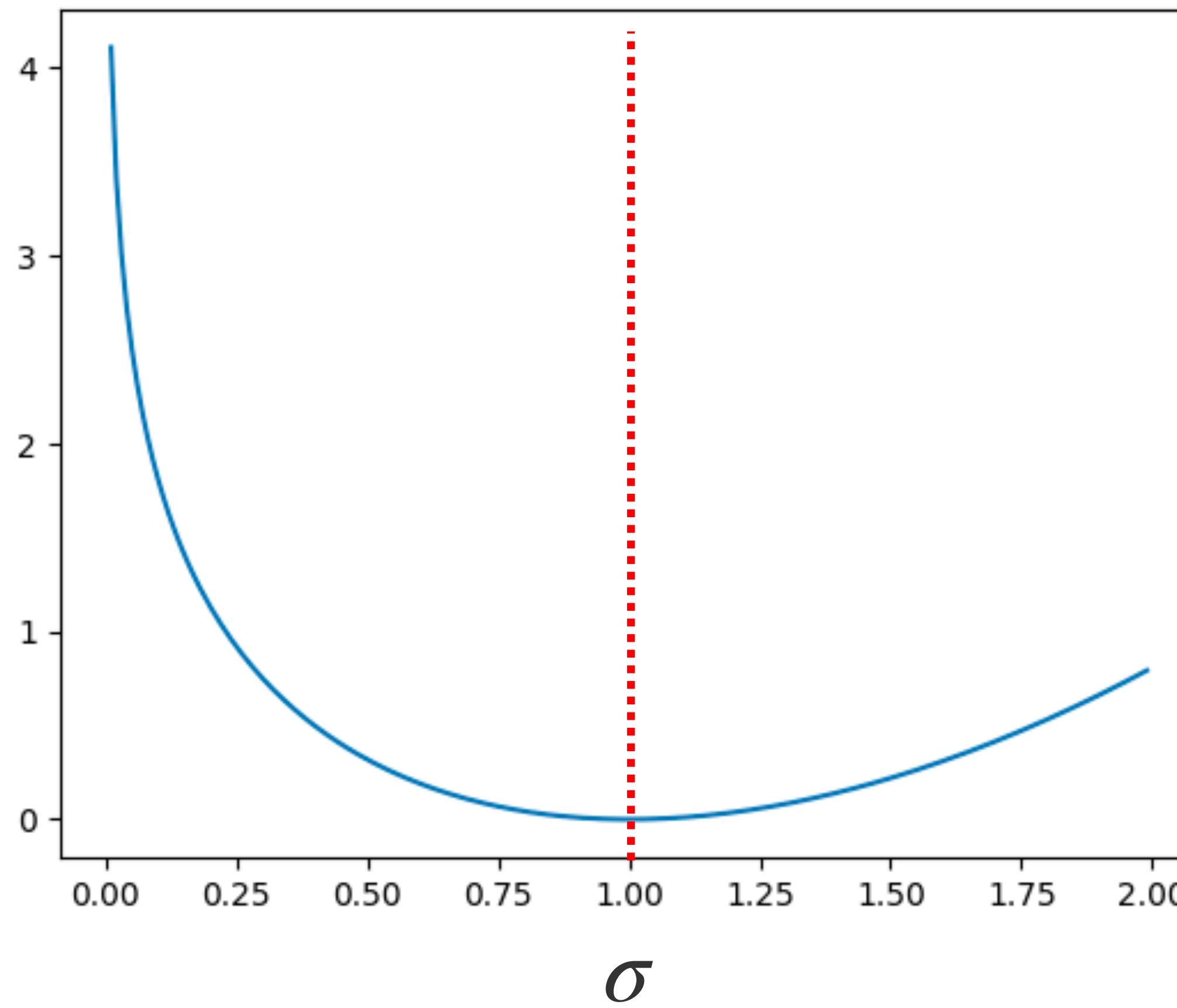
$$KL[p||q] = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

Note that the KL is not symmetric!

- The KL-divergence is **zero** when the 2 distributions are equal; otherwise $KL > 0$
- if $p \sim \mathcal{N}(\mu_0, \sigma_0)$ and $q \sim \mathcal{N}(\mu_1, \sigma_1)$, then
$$KL[p||q] = \log \frac{\sigma_1}{\sigma_0} + \frac{\sigma_0^2 + (\mu_0 - \mu_1)^2}{2\sigma_1^2} - \frac{1}{2}$$
- if $p \sim \mathcal{N}(\mu, \sigma)$ and $q \sim \mathcal{N}(0,1)$, then
$$KL[p||q] = -\log \sigma + \frac{1}{2}\mu^2 + \frac{1}{2}\sigma^2 - \frac{1}{2}$$
- Note that for $\sigma \rightarrow 0$ we have $KL \rightarrow +\infty$

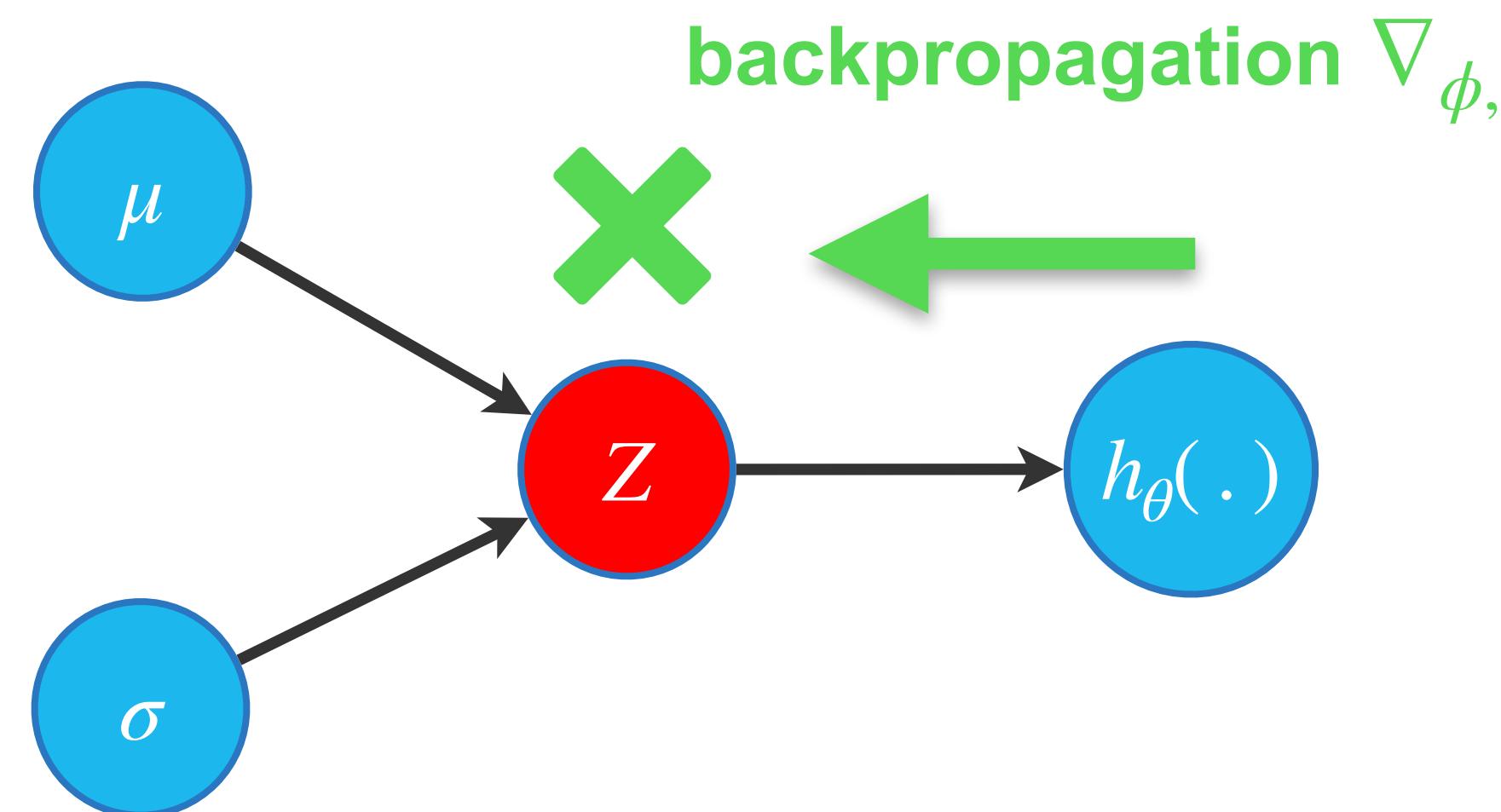
Kullback-Leibler divergence

$$KL[\mathcal{N}(0,\sigma) \parallel \mathcal{N}(0,1)]$$

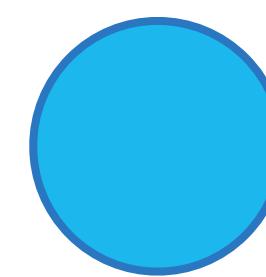
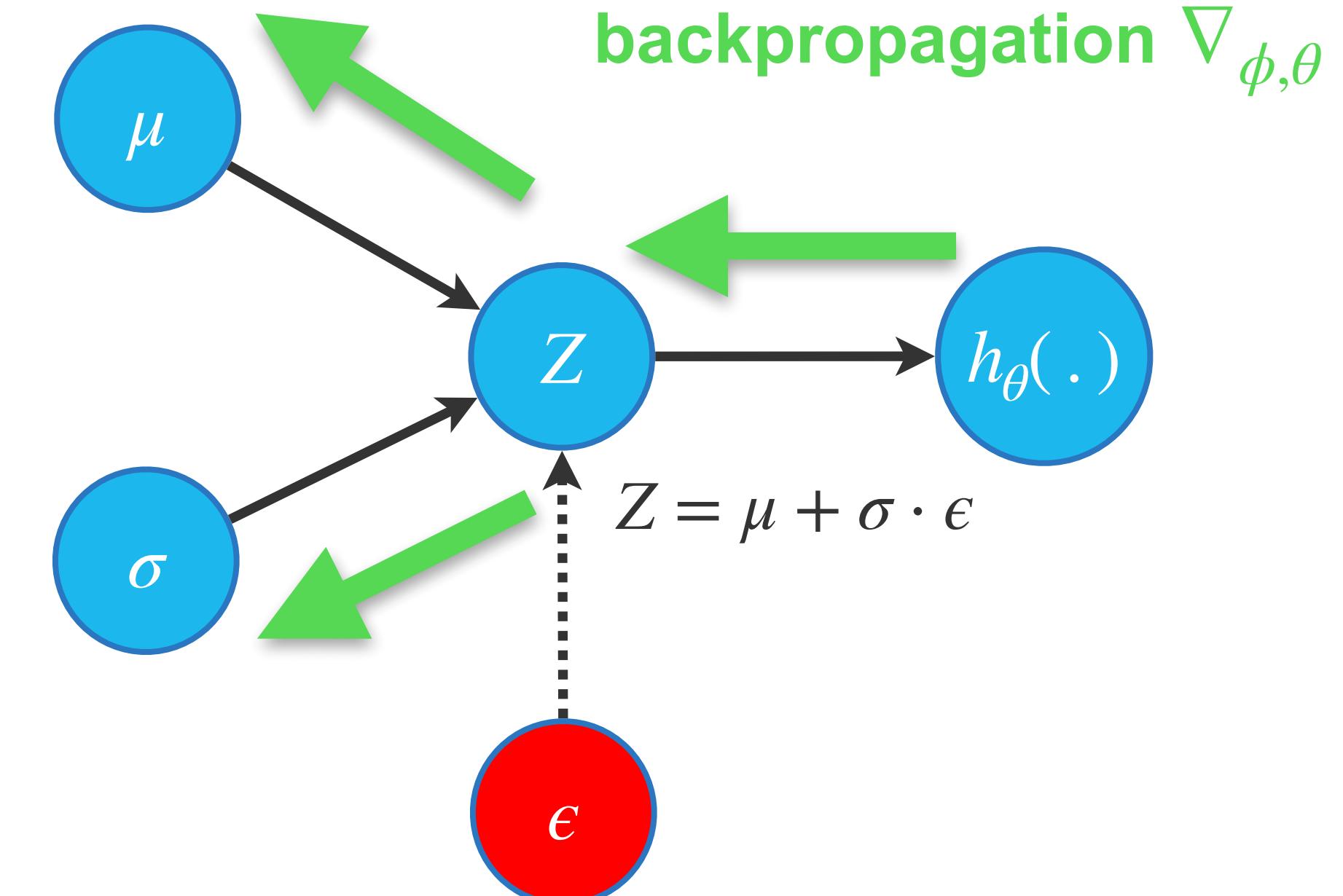


"Reparametrization trick"

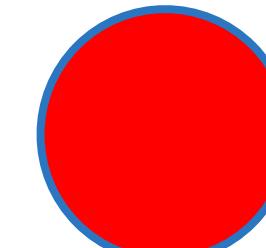
without reparametrization



with reparametrization



= deterministic variable



= random variable



= dependent on ϕ, θ



= independent of ϕ, θ

A bit of math.....

- Remember the following formula for the **expectation**

$$\mathbb{E}_{z \sim q(z)} (f(z)) = \int_z q(z) f(z) dz$$

- Expectations can be evaluated using **Monte-Carlo simulation**, i.e. by averaging over training samples (mean = unbiased estimator of expectation)

$$\mathbb{E}_{z \sim q(z)} (f(z)) \approx \frac{1}{L} \sum_{i=1}^L q(z_i) f(z_i)$$

- Backpropagation requires to compute the **gradient** of the loss function w.r.t. the model parameters: $\nabla_{\phi, \theta} \mathcal{L}$

"Reparametrization trick"

- Remember we need to minimize this loss function

$$\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2 = -\mathbb{E}_{Z \sim q_\phi(Z|X)} (\log p_\theta(X|Z)) - KL[q_\phi(Z|X) \| p(Z)]$$

- We need to compute the gradient $\nabla_{\phi,\theta}$ of this function; let's focus on the first term

$$\begin{aligned} \nabla_{\phi,\theta} \mathcal{L}_1 &= -\nabla_{\phi,\theta} \mathbb{E}_{Z \sim q_\phi(Z|X)} (\log p_\theta(X|Z)) = -\nabla_{\phi,\theta} \left(\int_Z q_\phi(Z|X) p_\theta(X|Z) dZ \right) \\ &= \boxed{- \int_Z (\nabla_{\phi,\theta} q_\phi(Z|X)) p_\theta(X|Z) dZ} - \boxed{\int_Z q_\phi(Z|X) \nabla_{\phi,\theta} p_\theta(X|Z) dZ} \\ &= -\mathbb{E}_{Z \sim q_\phi(Z|X)} (\nabla_{\phi,\theta} \log p_\theta(X|Z)) \end{aligned}$$

this is NOT an expectation
and cannot be simply evaluated...

this is an expectation and
can be computed through Monte-Carlo
(= taking the mean over training samples)

"Reparametrization trick"

- The trick consists in writing $Z = \mu_\phi(X) + \sigma_\phi(X) \cdot \epsilon = g_\phi(\epsilon, X)$ with $\epsilon \sim p(\epsilon) = \mathcal{N}(0,1)$
- we can now replace Z in the previous equations with $g_\phi(\epsilon, X)$

$$\nabla_{\phi,\theta} \mathcal{L}_1 = - \nabla_{\phi,\theta} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} (\log p_\theta(X | g_\phi(\epsilon, X))) = - \nabla_{\phi,\theta} \left(\int_{\epsilon} p(\epsilon) p_\theta(X | g_\phi(\epsilon, X)) d\epsilon \right)$$

$$= - \int_{\epsilon} p(\epsilon) \nabla_{\phi,\theta} p_\theta(X | g_\phi(\epsilon, X)) d\epsilon = - \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} \left(\nabla_{\phi,\theta} \log p_\theta(X | g_\phi(\epsilon, X)) \right)$$

*this is an expectation and
can be computed through Monte-Carlo !*

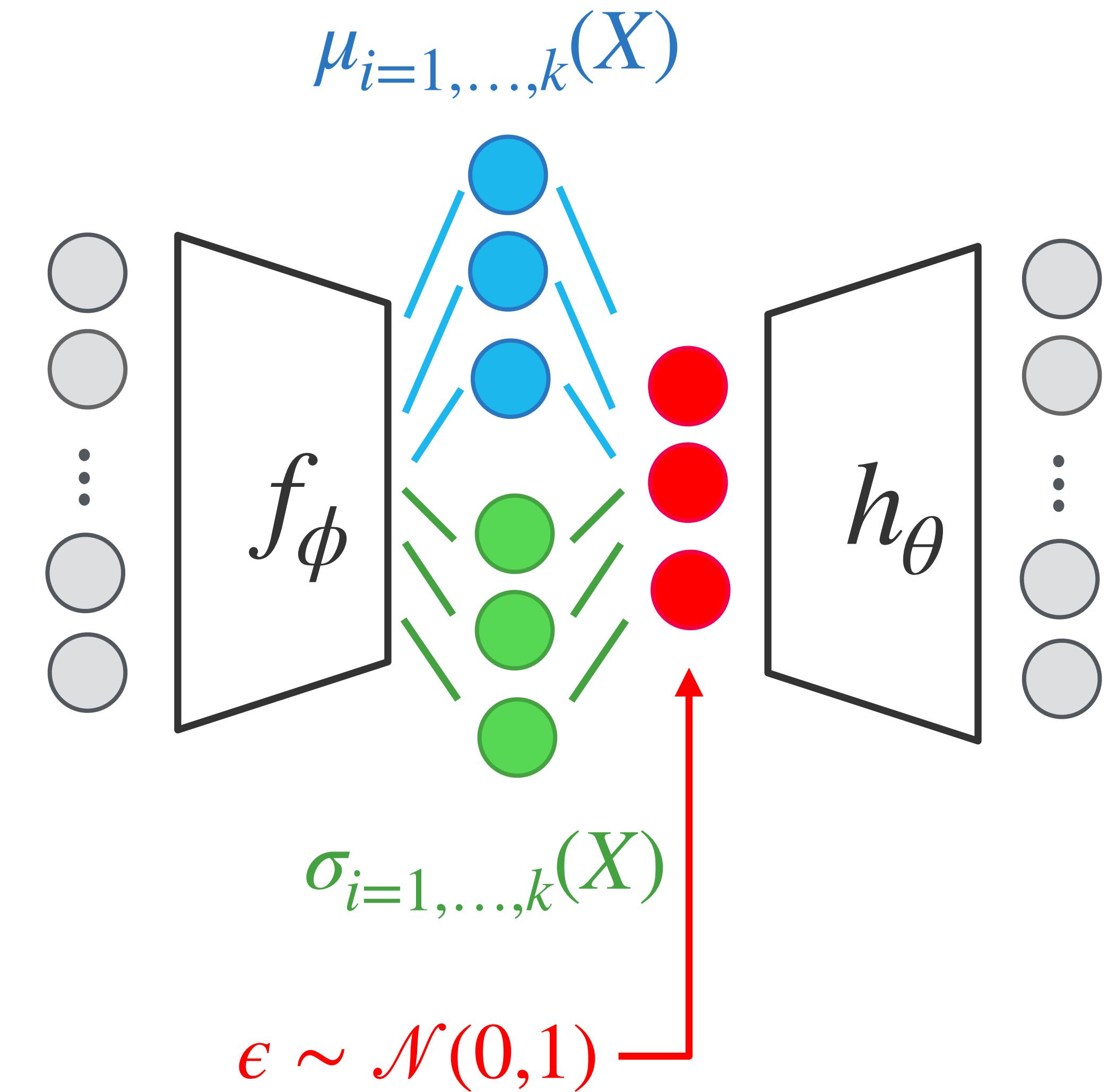
- The KL term can be computed analytically; hence the gradient of the KL term can be computed analytically without need to estimate it through Monte-Carlo

Variational autoencoder

- In order to be able to perform the backpropagation (using the gradients), we need to apply the **reparametrization trick**

$$Z : z_i = \mu_i + \epsilon \cdot \sigma_i$$

- Using this trick, gradients can be computed as the random variable ϵ does not depend on the parameters ϕ, θ of the network



Implementation

```
class VariationalEncoder(nn.Module):
    def __init__(self, latent_dims):
        super(VariationalEncoder, self).__init__()
        self.linear1 = nn.Linear(784, 512)
        self.linear2 = nn.Linear(512, latent_dims)
        self.linear3 = nn.Linear(512, latent_dims)

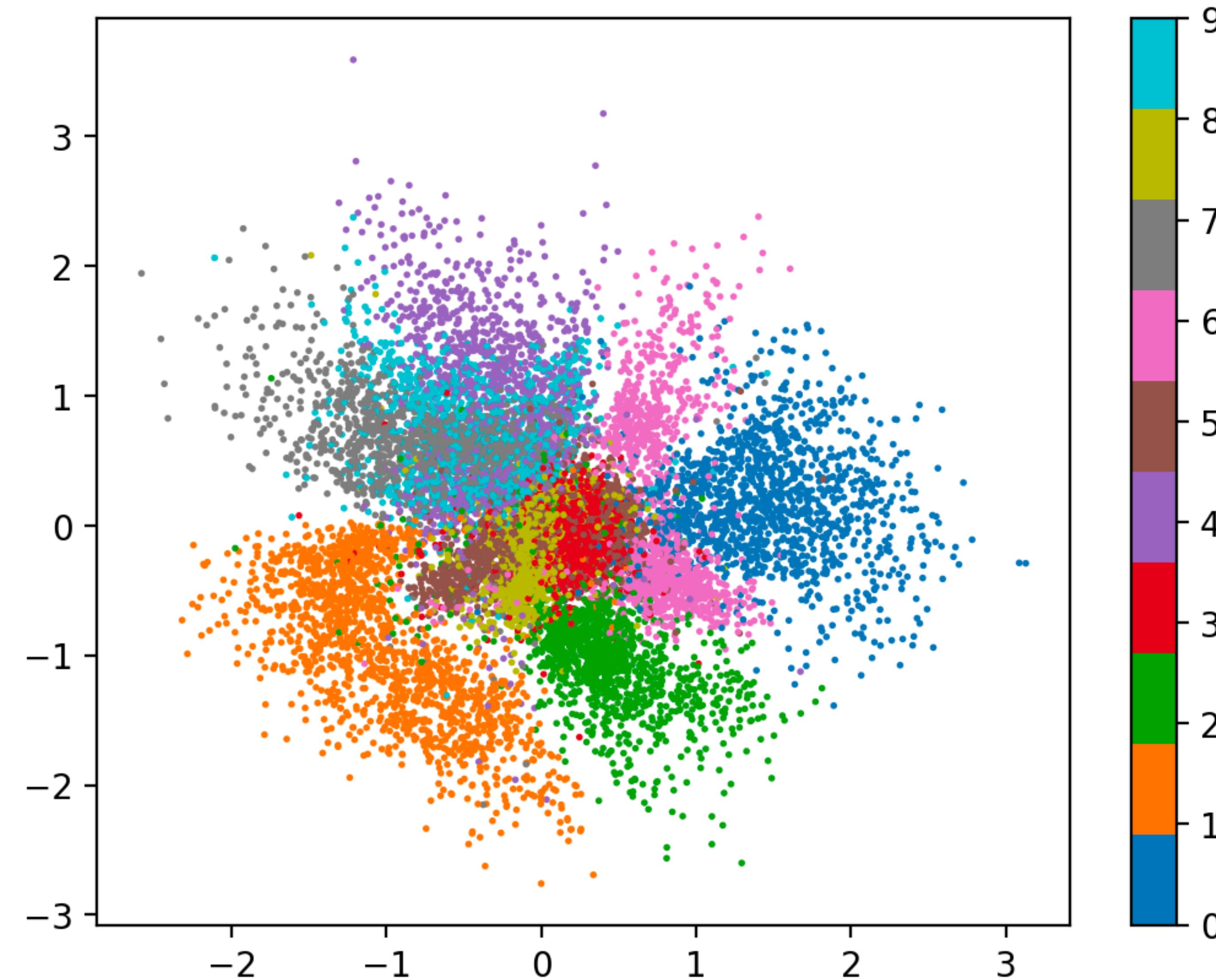
        self.N = torch.distributions.Normal(0, 1) ←  $\epsilon \sim \mathcal{N}(0, 1)$ 
        self.kl = 0

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.linear1(x))
        sigma = torch.exp(self.linear3(x)) ←  $z = \epsilon \cdot \sigma + \mu$ 
        mu = self.linear2(x)
        z = mu + sigma*self.N.sample(mu.shape)
        self.kl = (0.5*sigma**2 + 0.5*mu**2 - torch.log(sigma) - 1/2).sum()
        return z
```

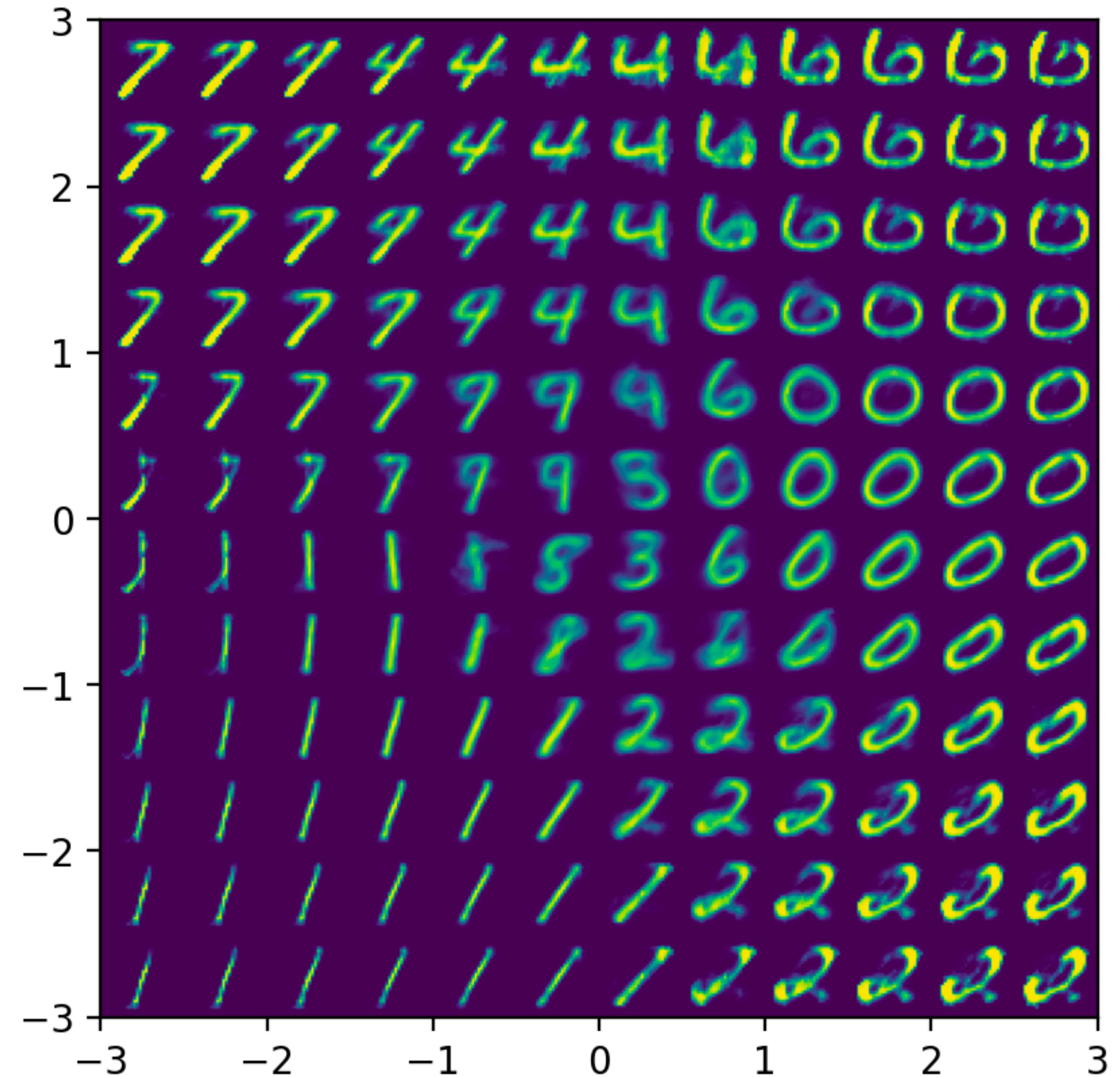
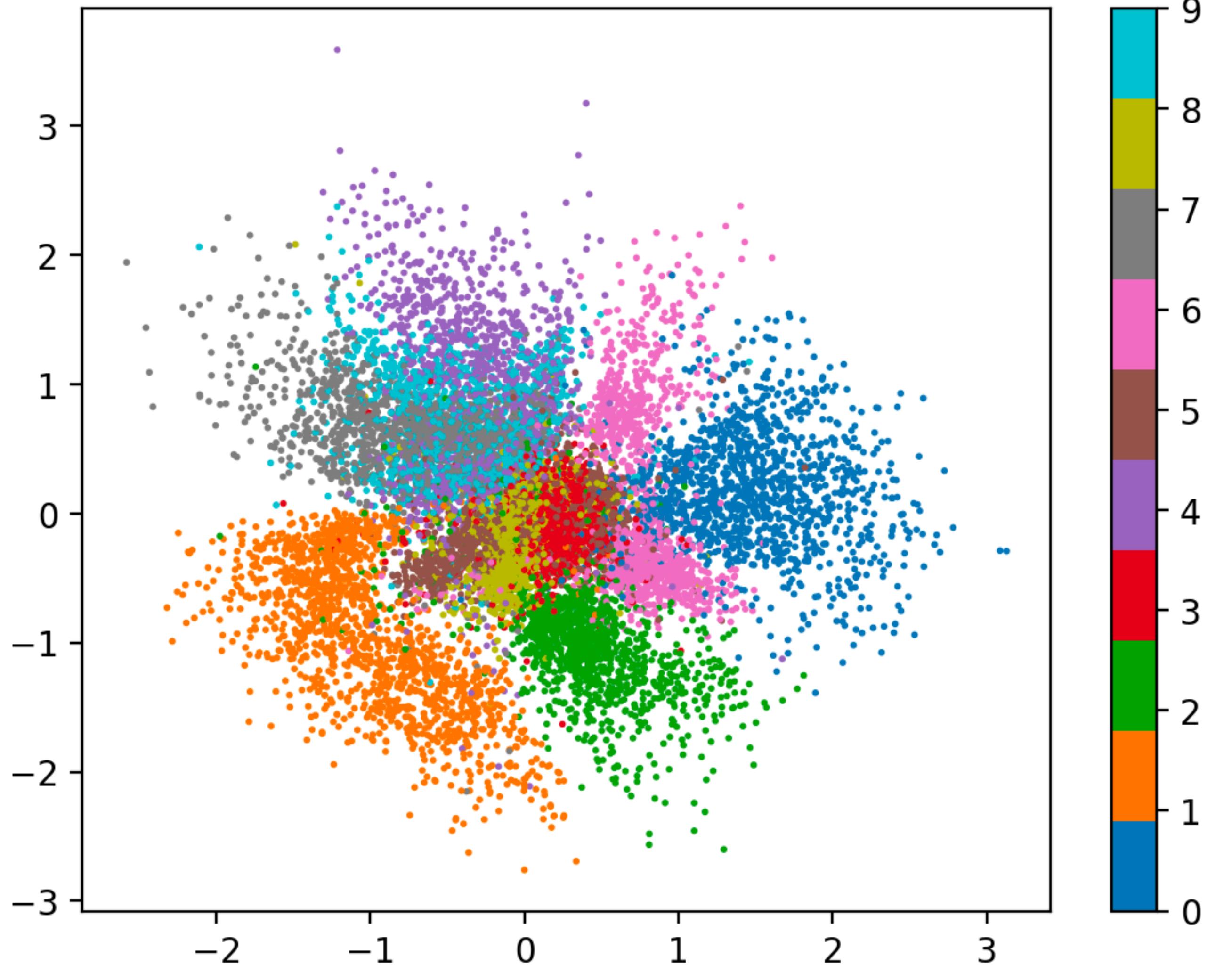
learn μ

learn σ

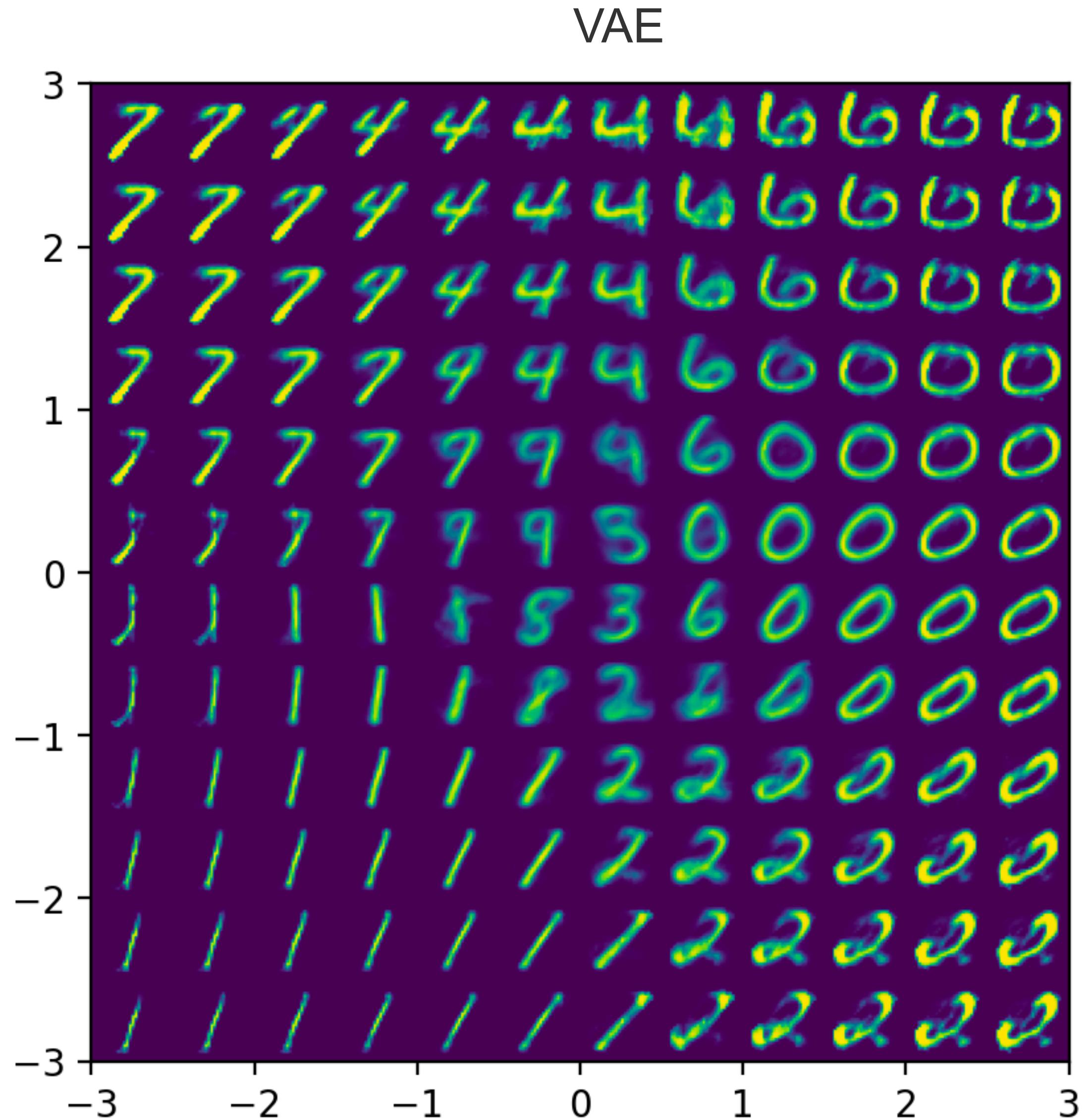
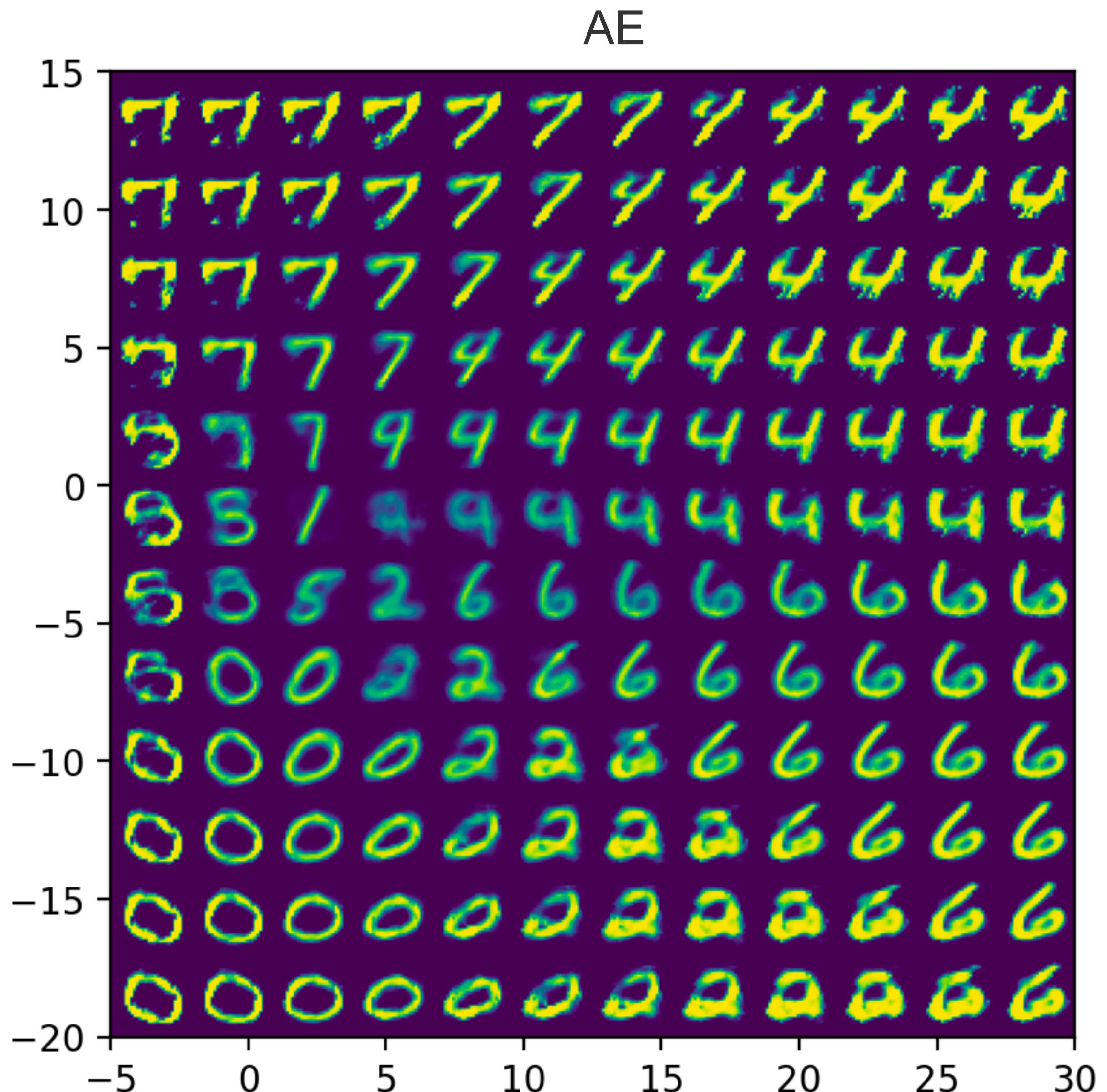
VAE latent space



Sampling from latent space

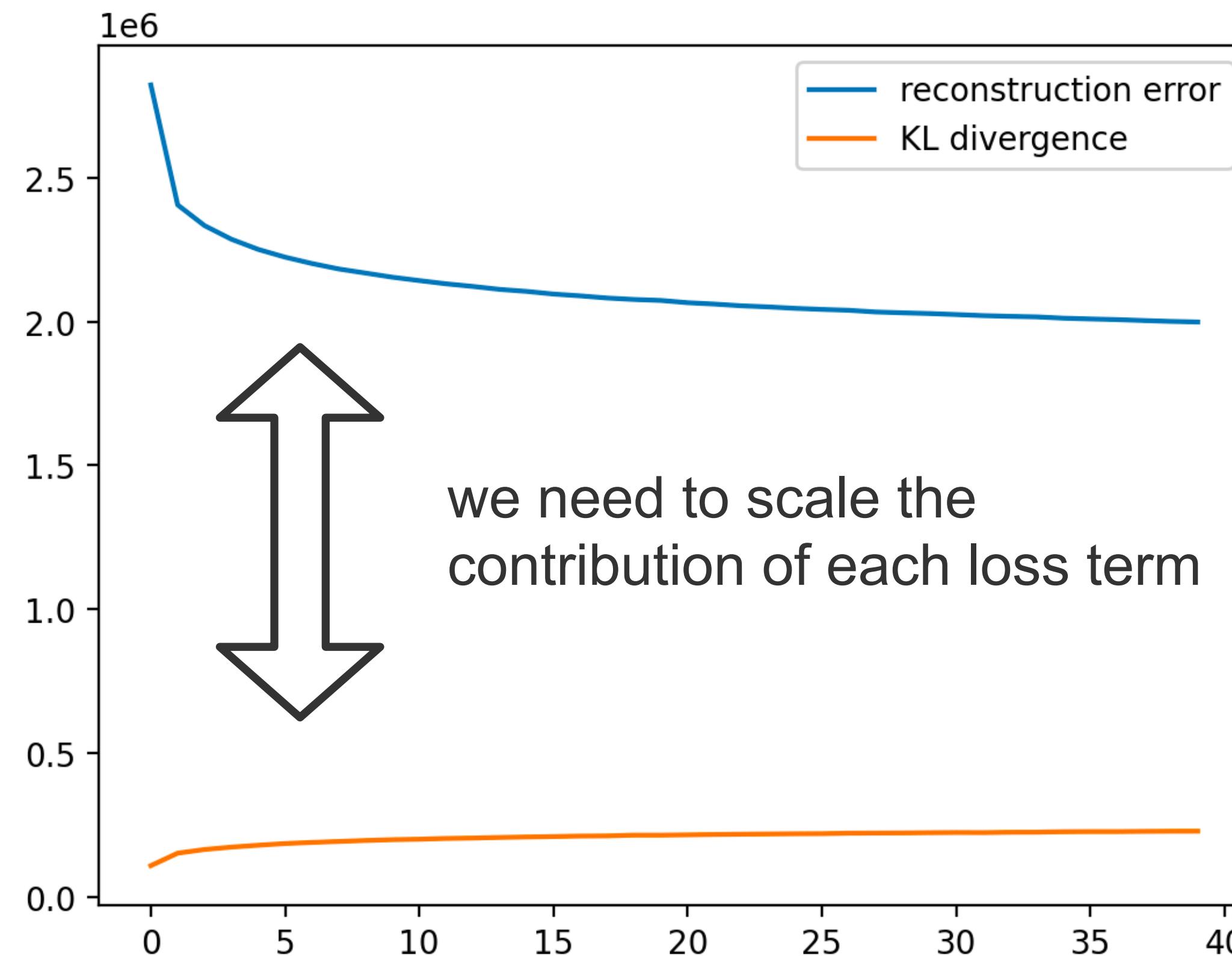


Compare to AE



Hyperparameter

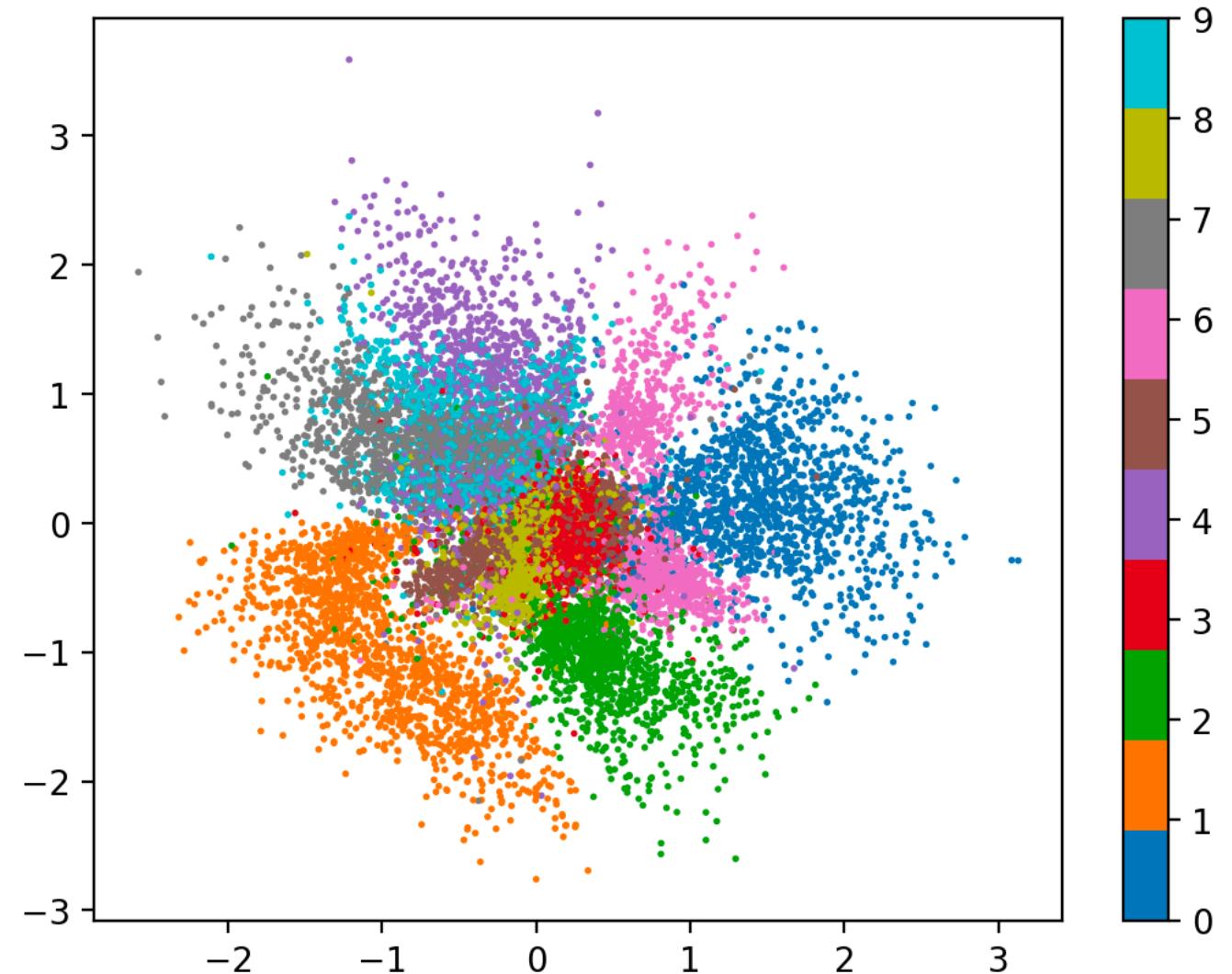
$$\mathcal{L}_{VAE} = \sum_{i=1}^N \left(\boxed{\|X^i - \hat{X}^i\|^2} + \boxed{KL[q_\phi(Z^i | X^i) \| p(Z^i)]} \right)$$



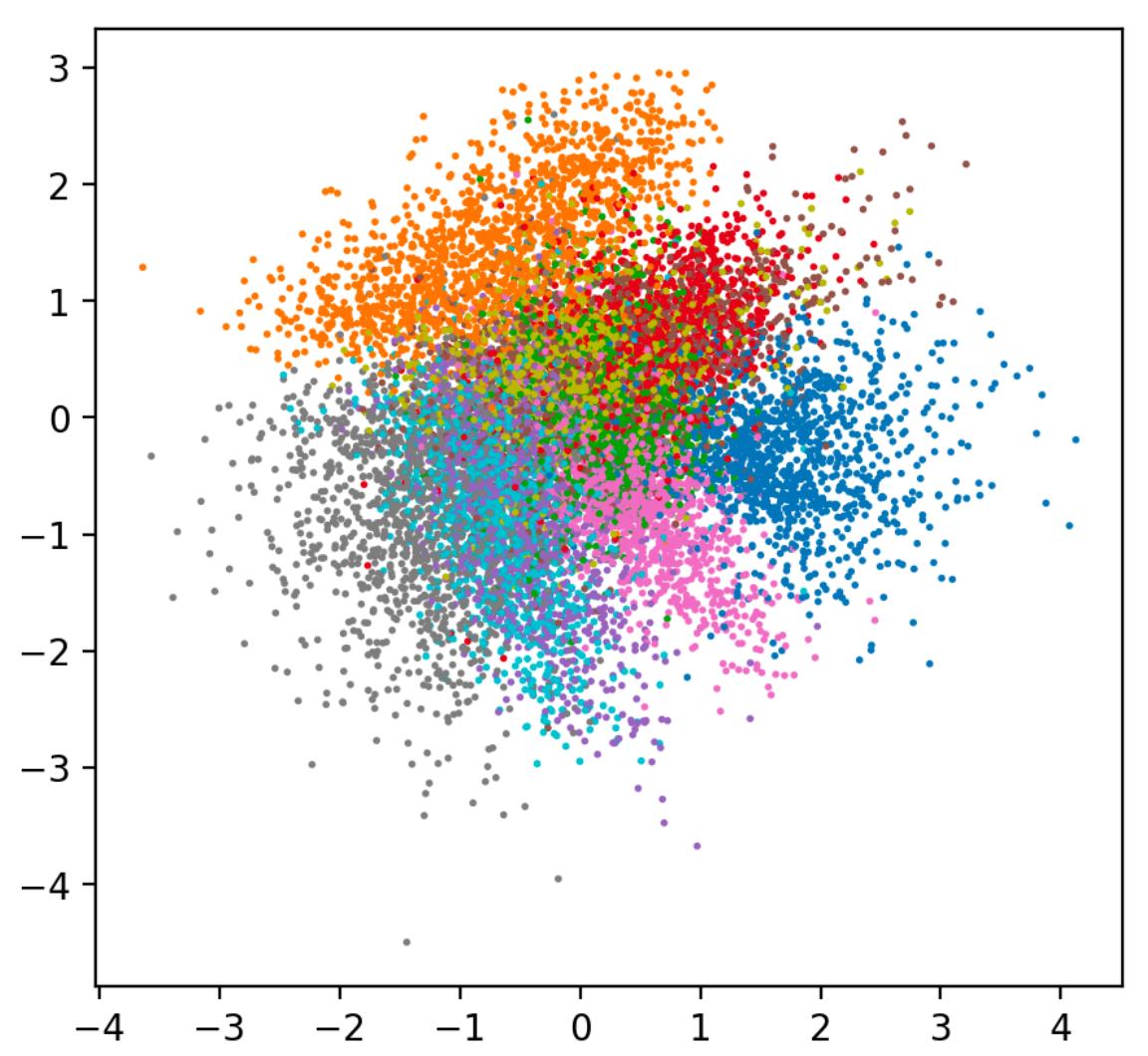
Hyperparameter

$$\mathcal{L}_{VAE} = \sum_{i=1}^N \left(\|X^i - \hat{X}^i\|^2 + \boxed{\lambda} \cdot KL[q_\phi(Z^i | X^i) \| p(Z^i)] \right)$$

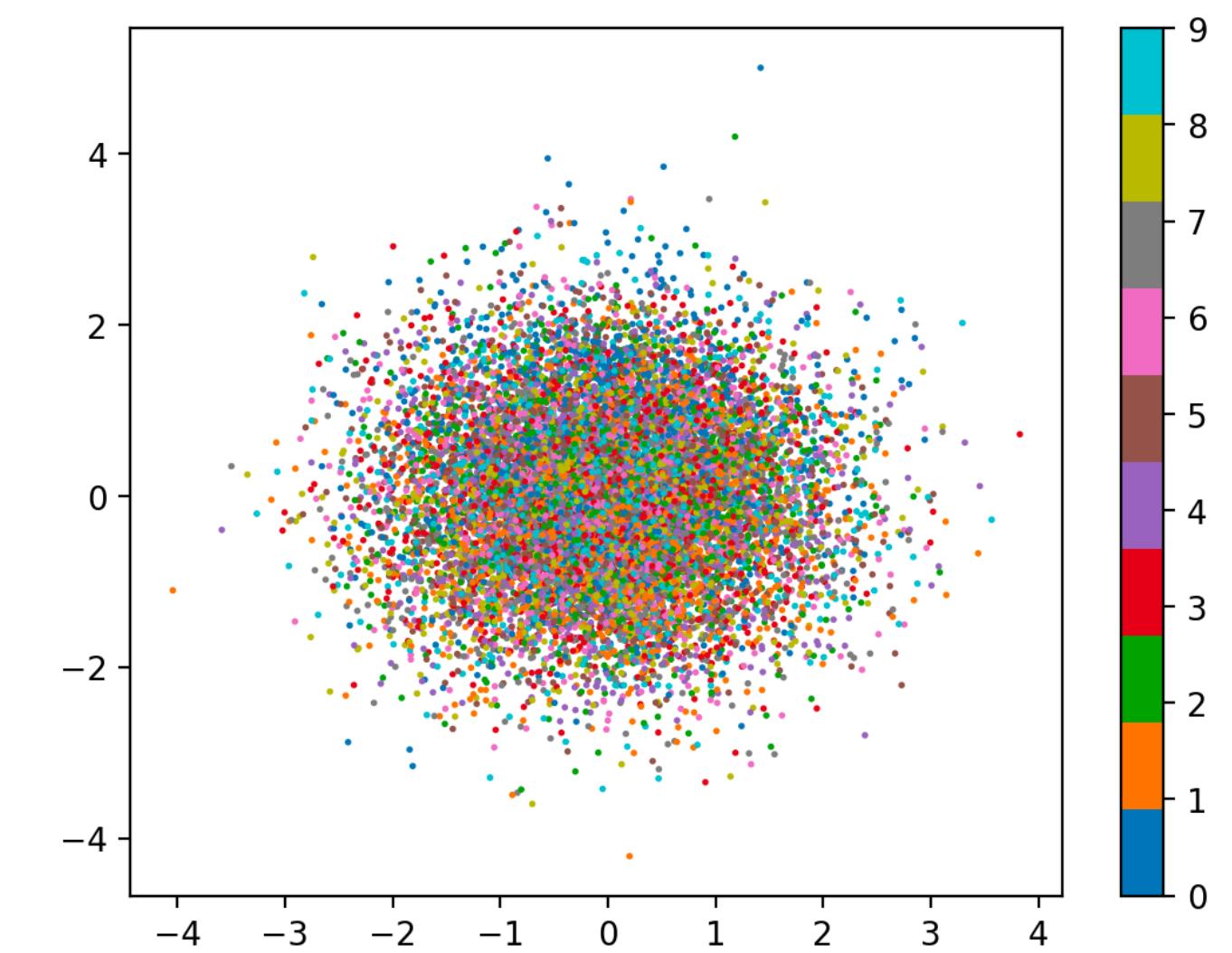
$$\lambda = 1$$



$$\lambda = 4$$



$$\lambda = 10$$



- As the contribution of the KL-divergence increases, the z get pushed towards $\mathcal{N}(0, 1)$

Take-home messages

- Autoencoders are **unsupervised** models → they do not need labelled data
- **Dimensional reduction** approach: find a low-dimensional representation of the data
- **Various flavors**: undercomplete / overcomplete / sparse / denoising
- Variational autoencoders are **generative, probabilistic** models
- VAEs learn a more **continuous representation** of the data, from which we can sample new realistic data points
- **Hyperparameters** of VAEs have a strong influence on the model behaviour

References

- **VAE original paper**
 - Kingma, Welling (2013) <https://arxiv.org/abs/1312.6114>
- **Different views on VAEs, with mathematical details**
 - <https://mbernste.github.io/posts/vae/>
- **Reparametrization trick**
 - Blog: <https://gregorygundersen.com/blog/2018/04/29/reparameterization/>
 - Video: <https://www.youtube.com/watch?v=vy8q-WnHa9A&t=447s>
- **Denoising AE**
 - Slides: <https://fleuret.org/dlc/materials/dlc-handout-7-3-denoising-autoencoders.pdf>
 - pyTorch implementation: <https://github.com/pranjaldatta/Denoising-Autoencoder-in-Pytorch/blob/master/DenoisingAutoencoder.ipynb>