



**Universität
Zürich^{UZH}**



Machine Learning and Data Analysis Unsupervised Learning: (Variational)- AutoEncoders and GANs

National University of Science and Technology of Moscow (NUST-MISIS)

Lecturer: Prof. Dr. Nico Serra^{1,2}

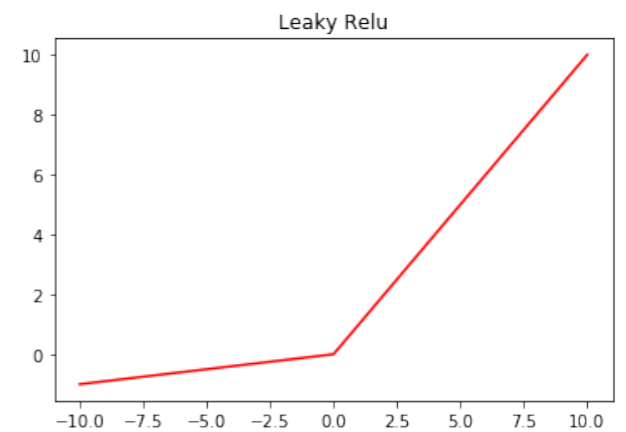
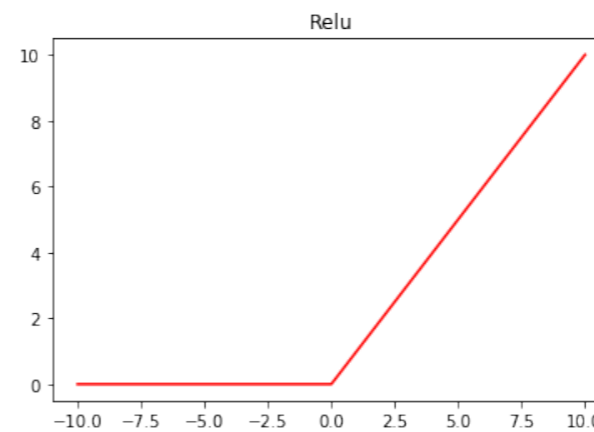
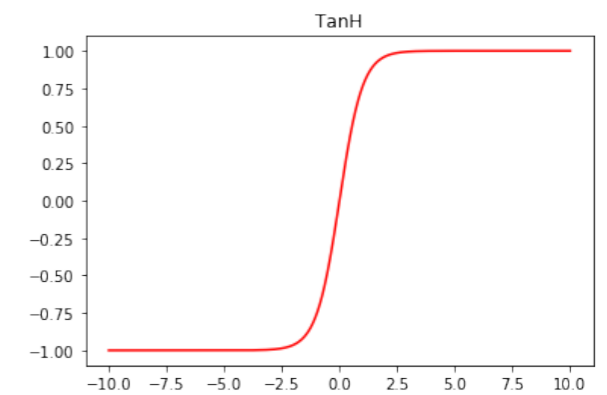
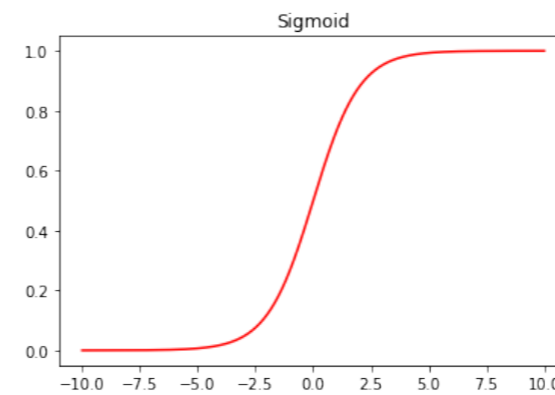
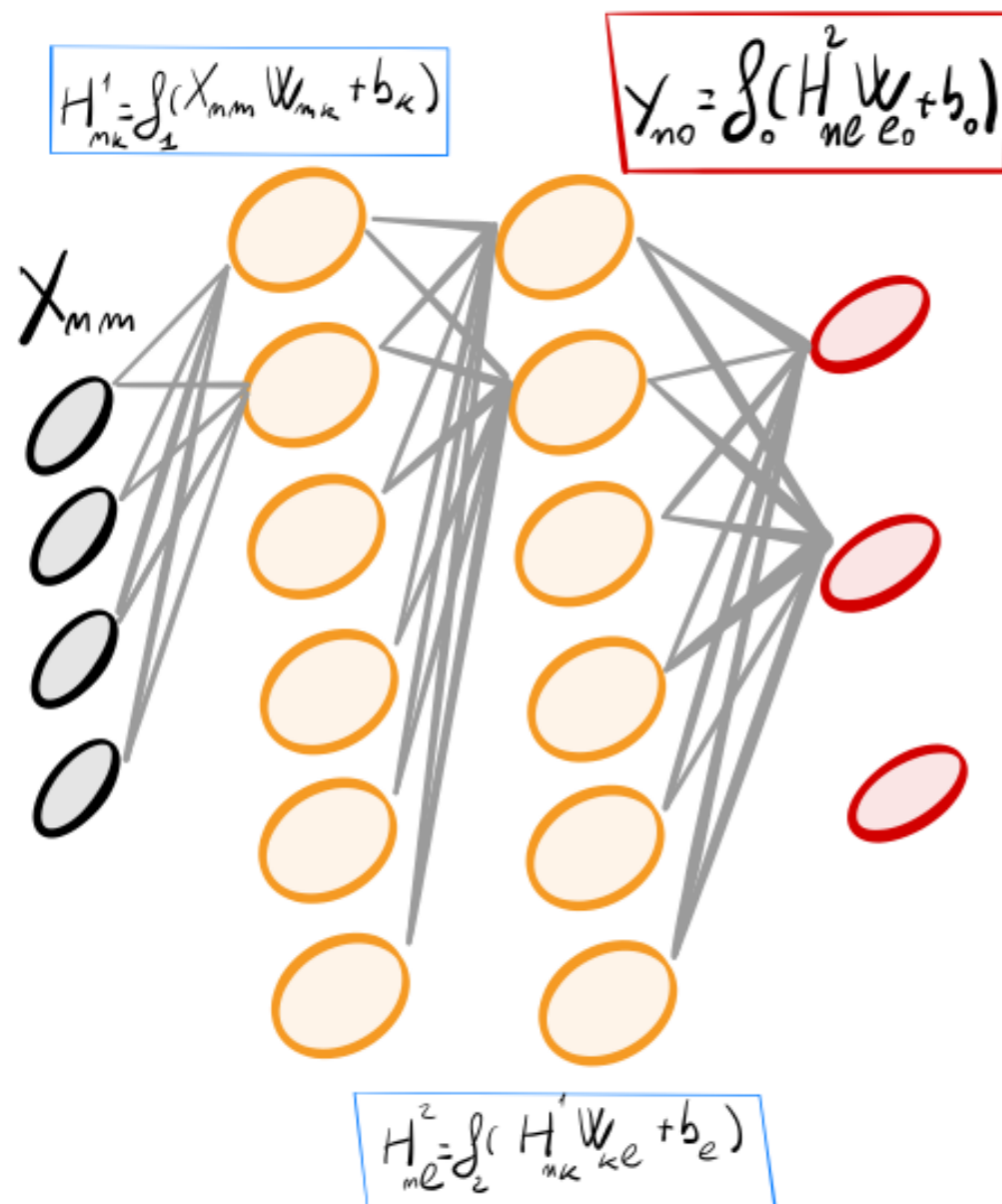
¹University of Zurich

²NUST-MISIS

<https://www.physik.uzh.ch/en/researcharea/lhcb/team/senior-researchers/Serra.html>

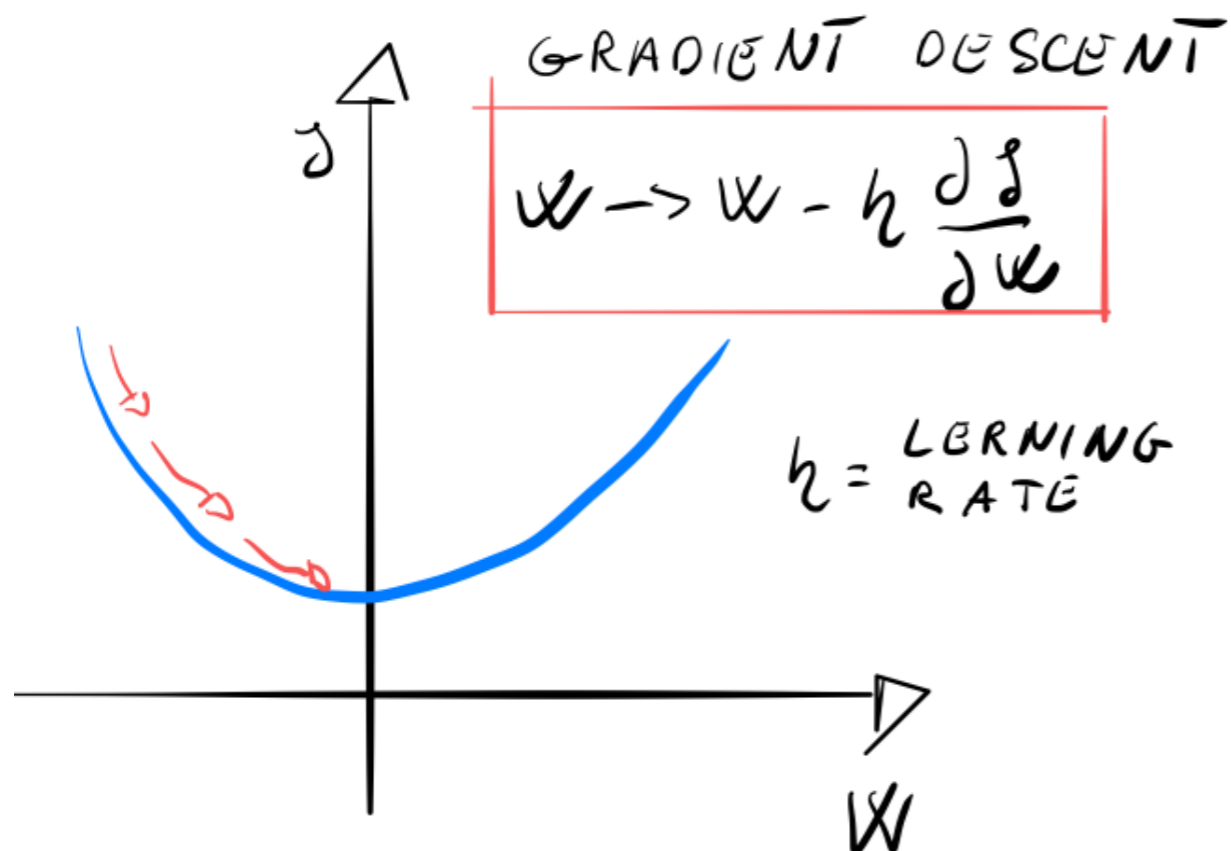
Summary

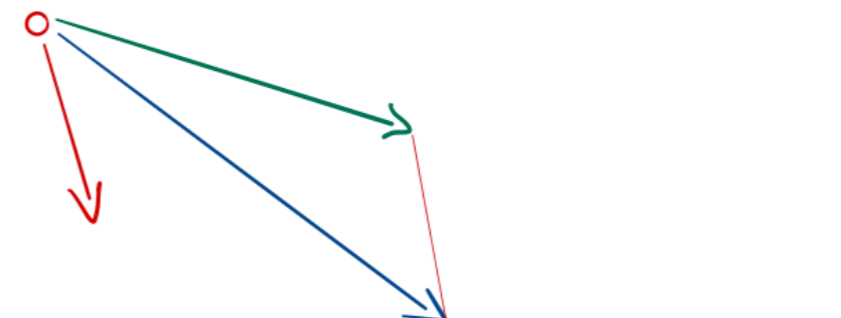
- We have seen that Neural Networks consist generally in several layers of neurones with non-linear activation function
- Output function depends on the goal, we have seen, linear, sigmoid, soft-max, but there could be more



Summary

- ANN are optimised using some variant of Gradient Descent:
 - Several algorithms improve vanilla SGD such as Momentum, RMSProp, Adam, ... all these are available in modern Deep Learning Packages



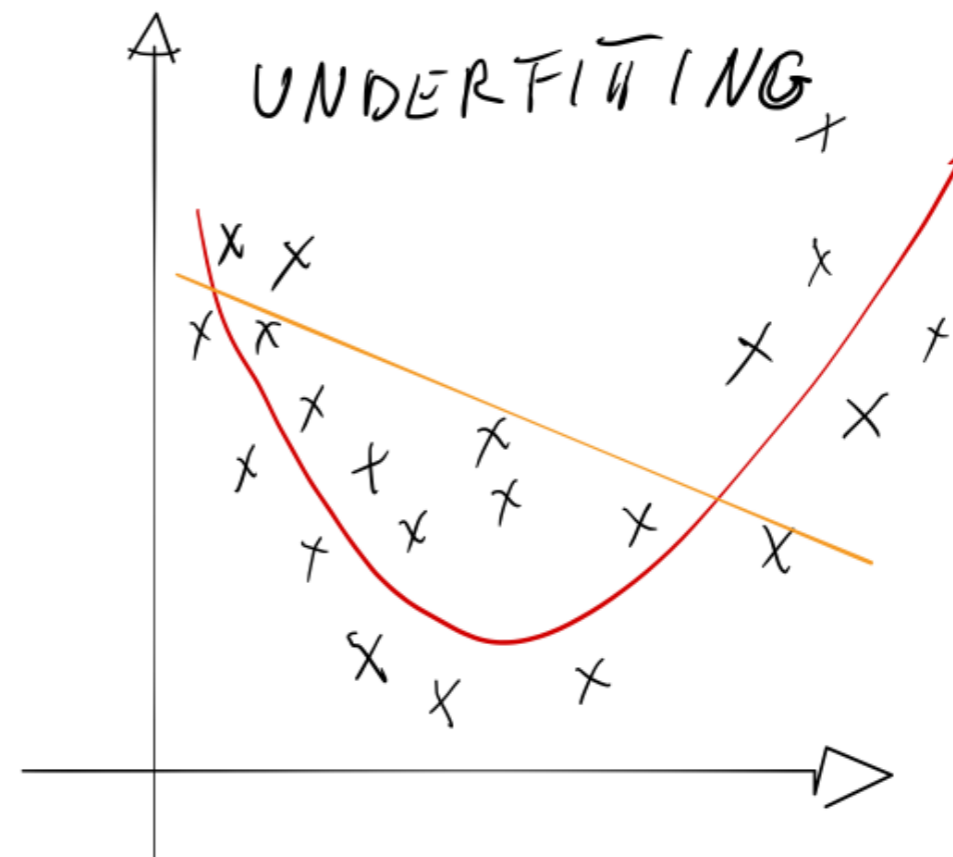
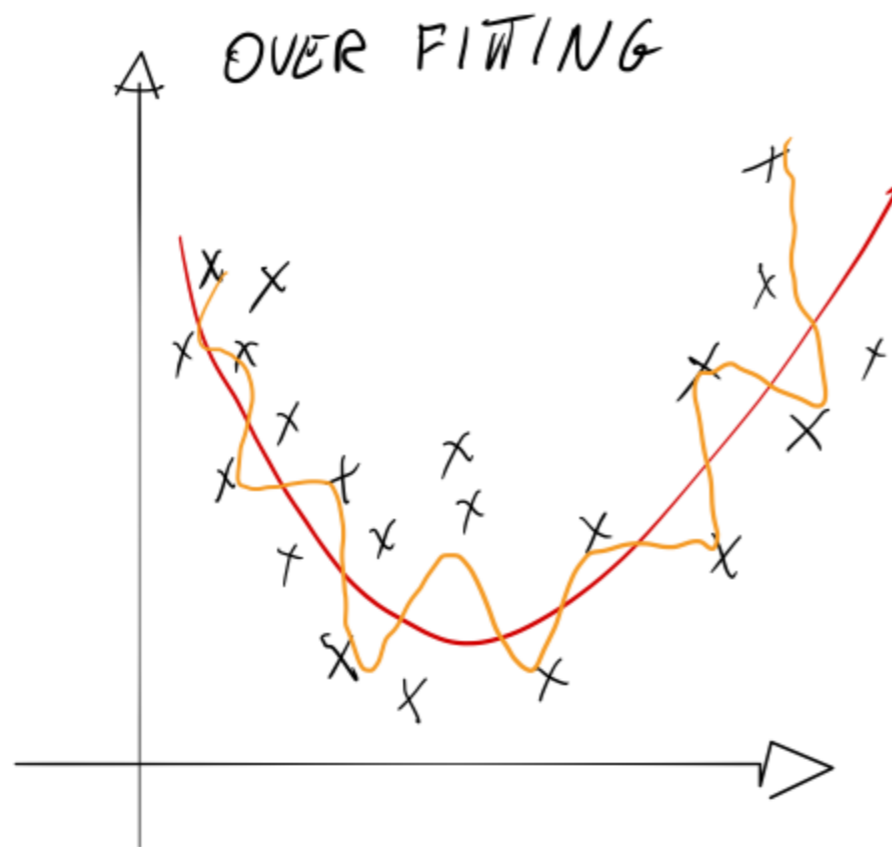


$$V_t = \beta V_{t-1} + (1 - \beta) \nabla_w L_{\text{loss}}$$

$$W \rightarrow W - \eta V_t$$

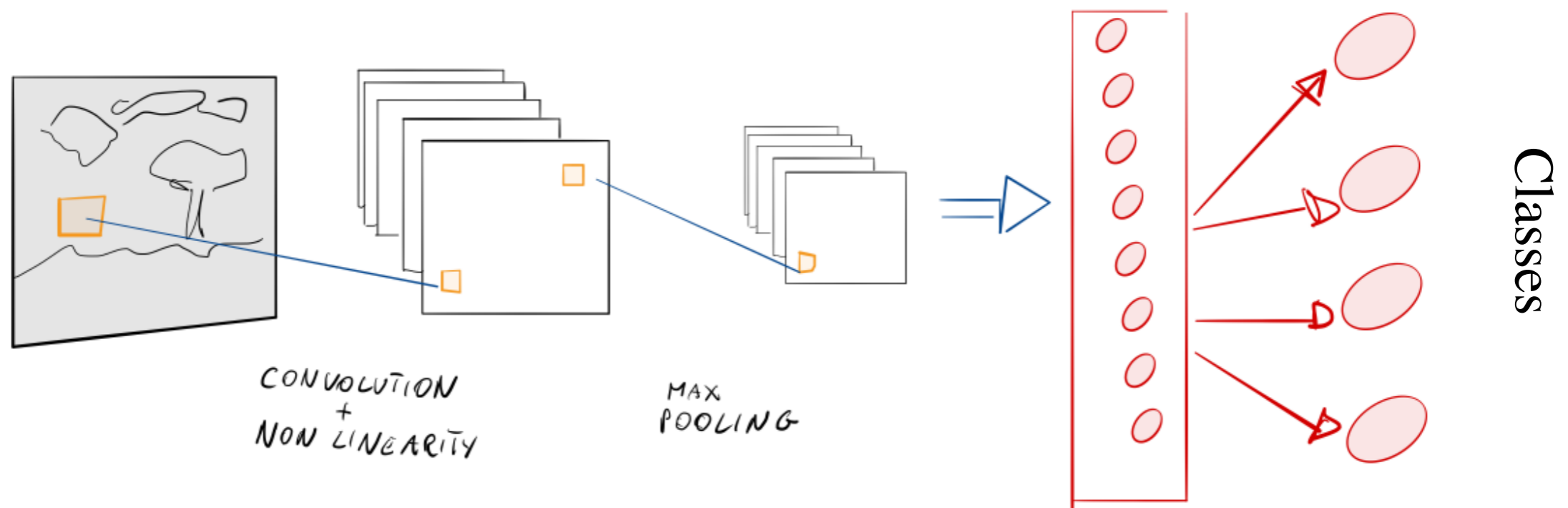
Summary

- The problem of over-fitting and under-fitting is a common problem in machine learning and can be attacked with several techniques:
 - Regularization, Data Augmentation, Early Stop, Dropout, ...



Summary

- We have seen that in addition to fully connected feed forward ANN there are other architecture, we have seen CNN which mimic the neuron connection pattern of the mammal visual cortex
- It is particularly suited in data structured like images, when the information in each pixel is not as important as the correlation among them

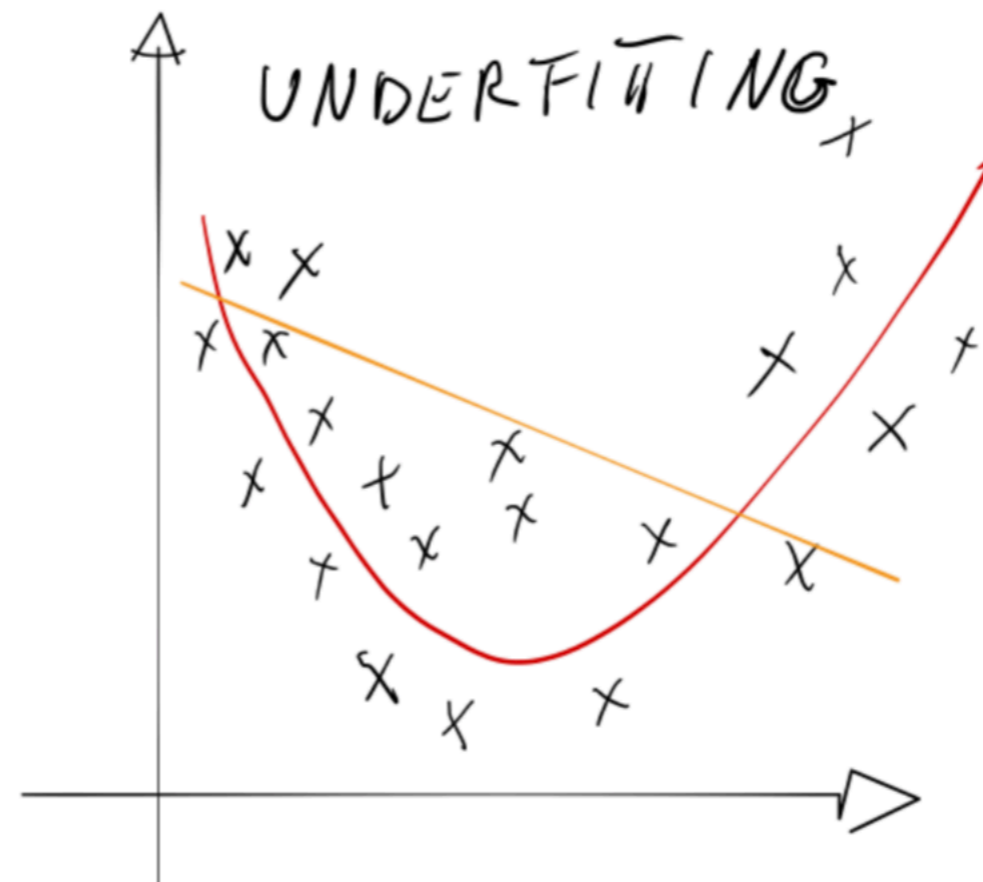
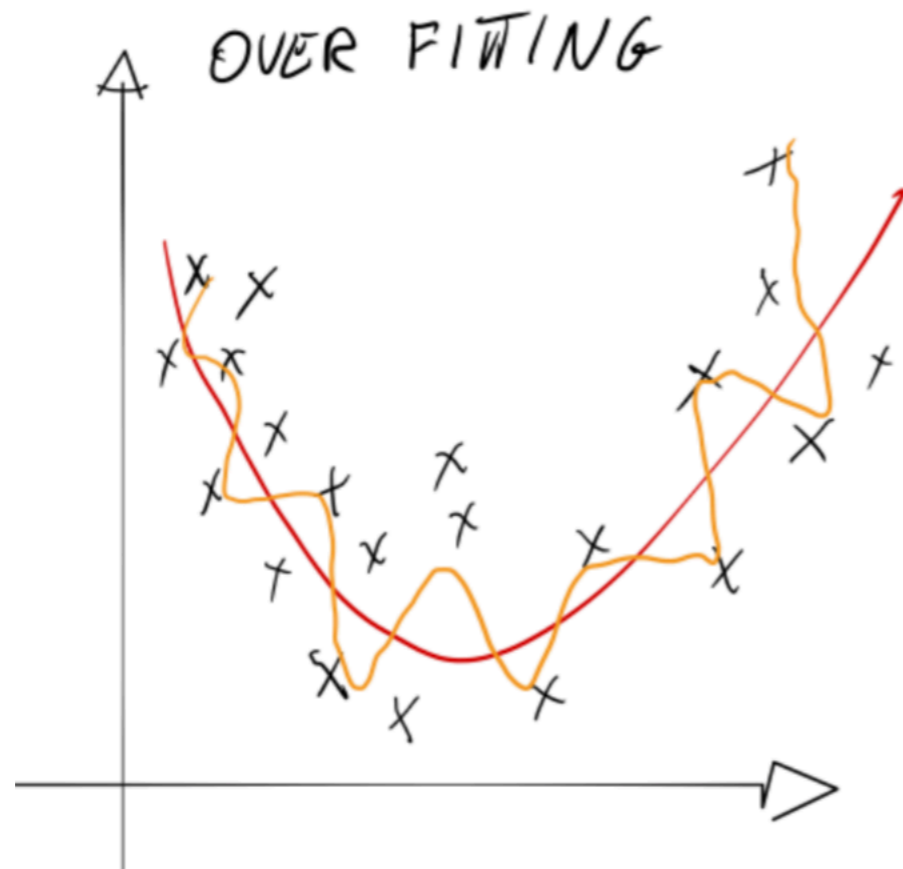


Over/Under-fitting

The problem of over-fitting and under-fitting is a common problem in machine

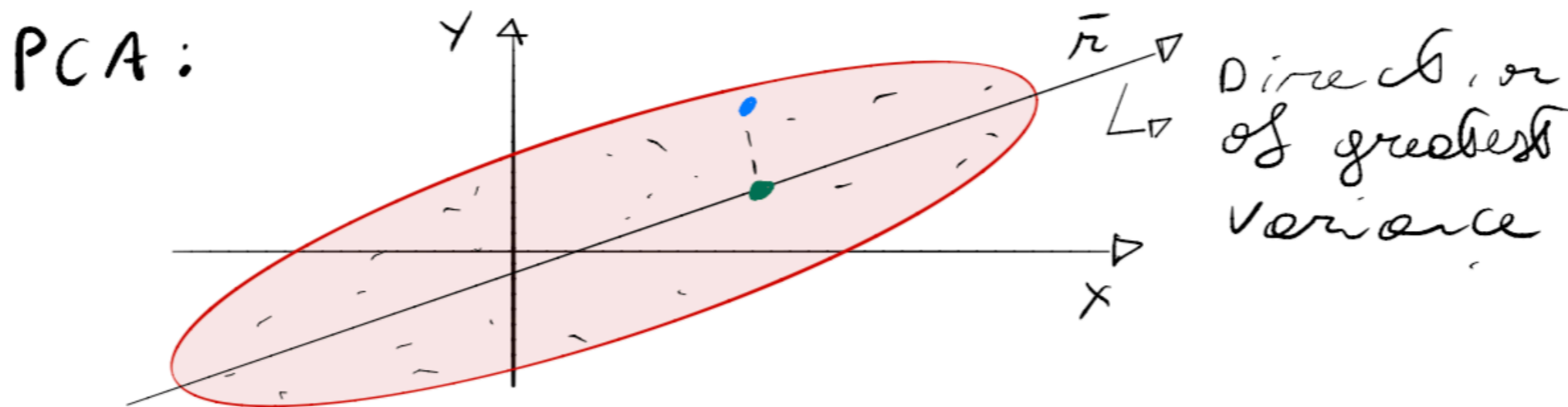
-learning and can be attacked with several techniques:

-Regularization, Data Augmentation, Early Stop, Dropout, ...



Unsupervised Learning

- Learn the structure of data
- Learn features in data
- Learn probability distribution of data
- Compress data



PCA: Suppose that I want to represent my data with a single number I chose the direction of greatest variance

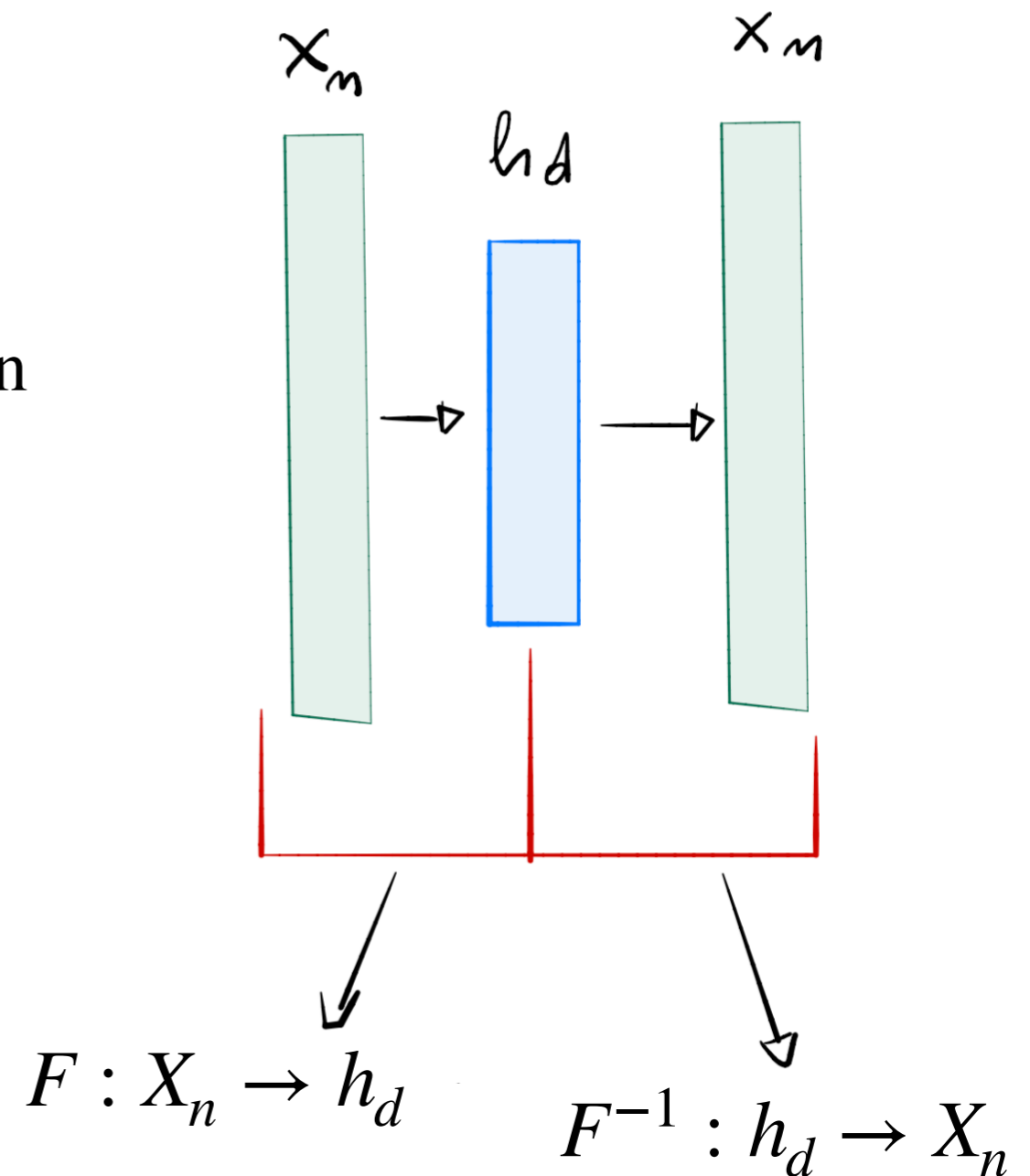
PCA

- The Principal Component Analysis (PCA) is a way of compressing the data
- If data are located on a linear manifold, it is convenient to “get rid” of redundant dimensions
- In order to find the best representation of data in d -dimensions ($d < n$), we choose the d dimensions with greatest variance
- PCA consists of finding the d orthogonal dimensions with greatest variance, equivalent to diagonalise an n -dimension matrix and take the d -dimensional sub-matrix

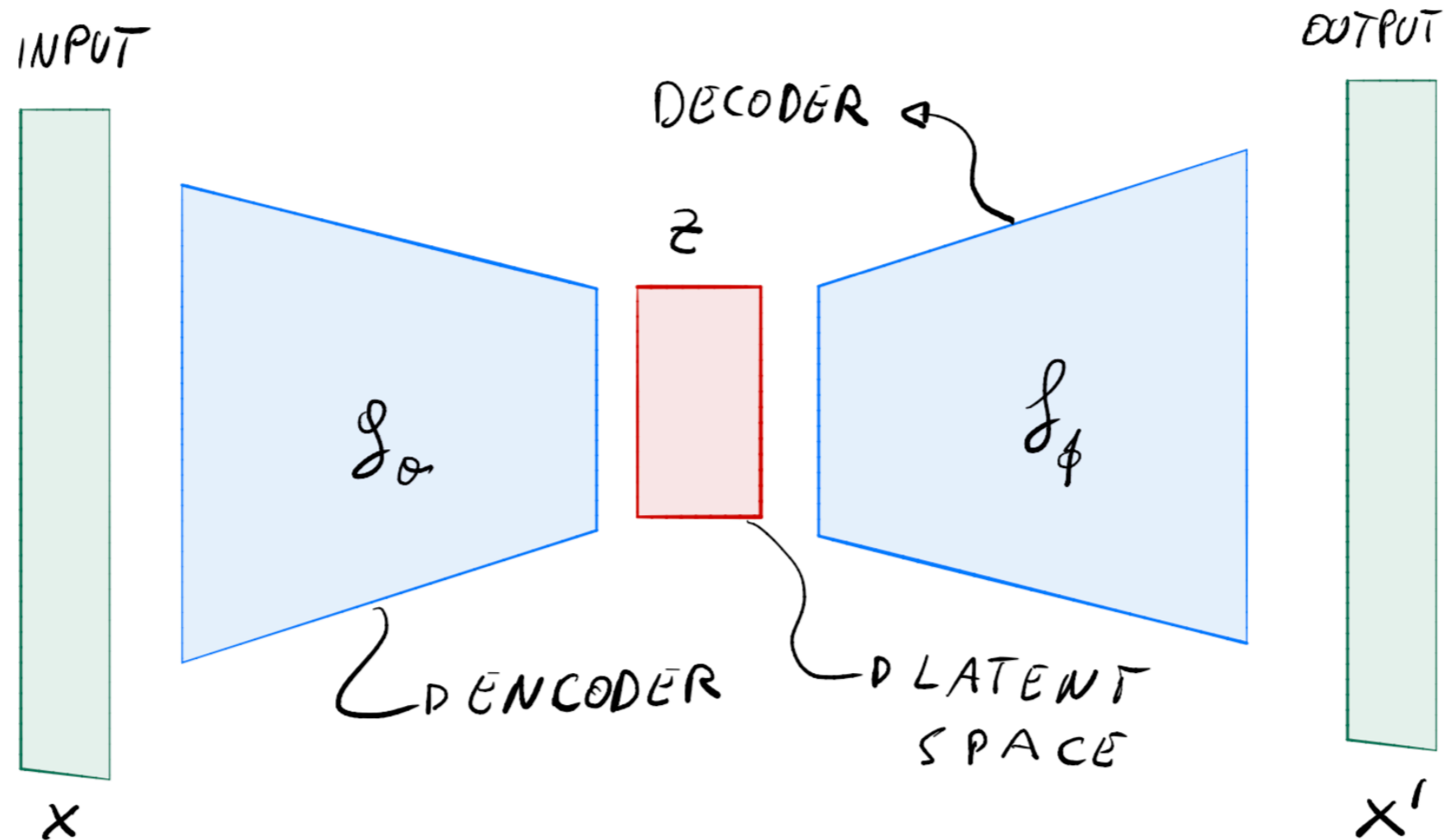
PCA

A PCA-like method can be applied with a simple ANN

- ANN with 1 hidden layer and no (linear) activation function
- The dimension of the hidden layer is $d < n$
- The loss consists in minimising the square error
- The hidden layer spans the same space at PCA, but the h_d neurons are NOT orthogonal
- ANN is not an efficient way to apply PCA



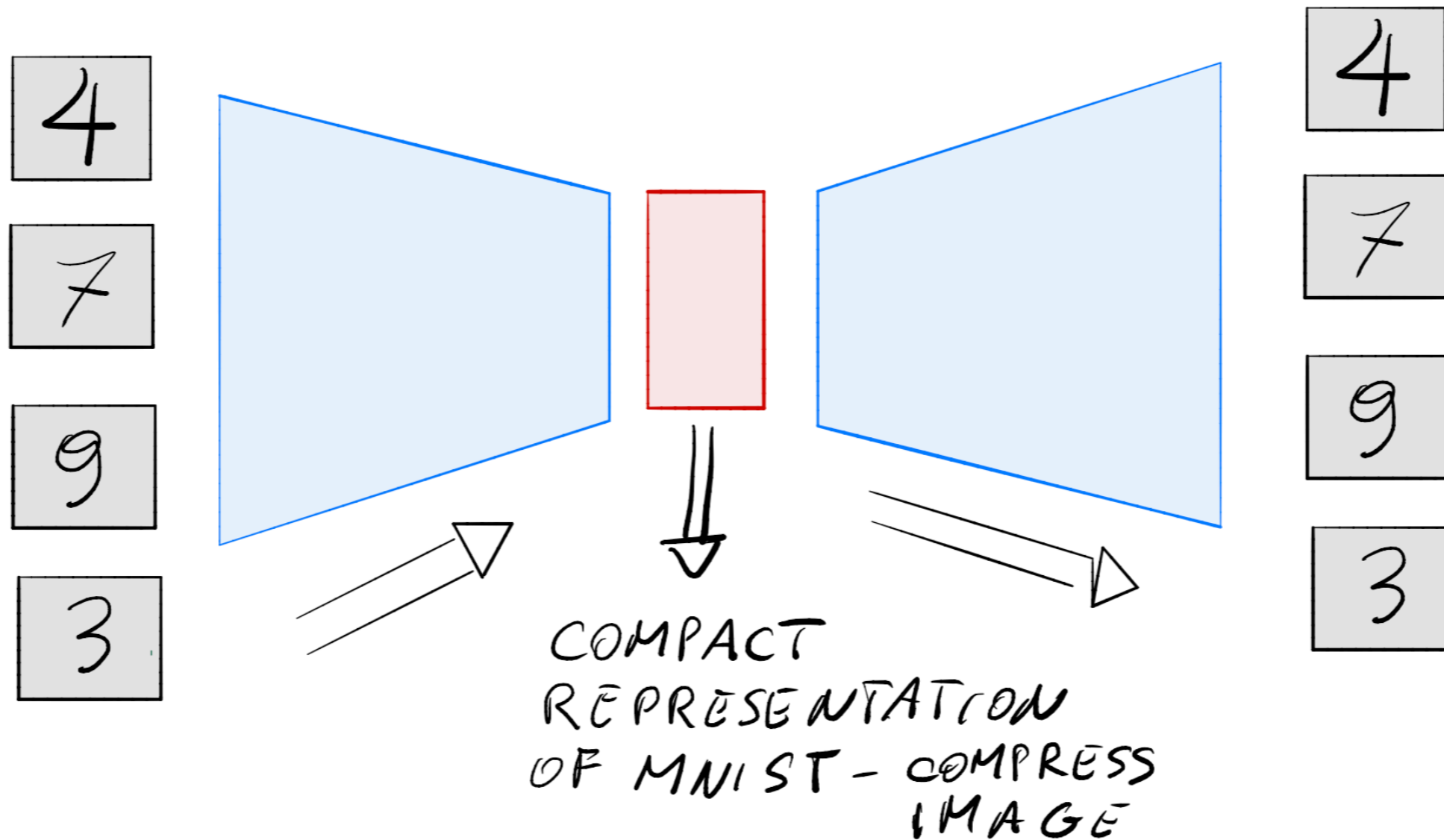
Autoencoders



$$X' = f_\phi [g_\theta(X)] \quad \text{Loss} : \mathcal{L}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^N [X_i - X'_i]^2$$

Autoencoders (AE) are trained to reproduce the input

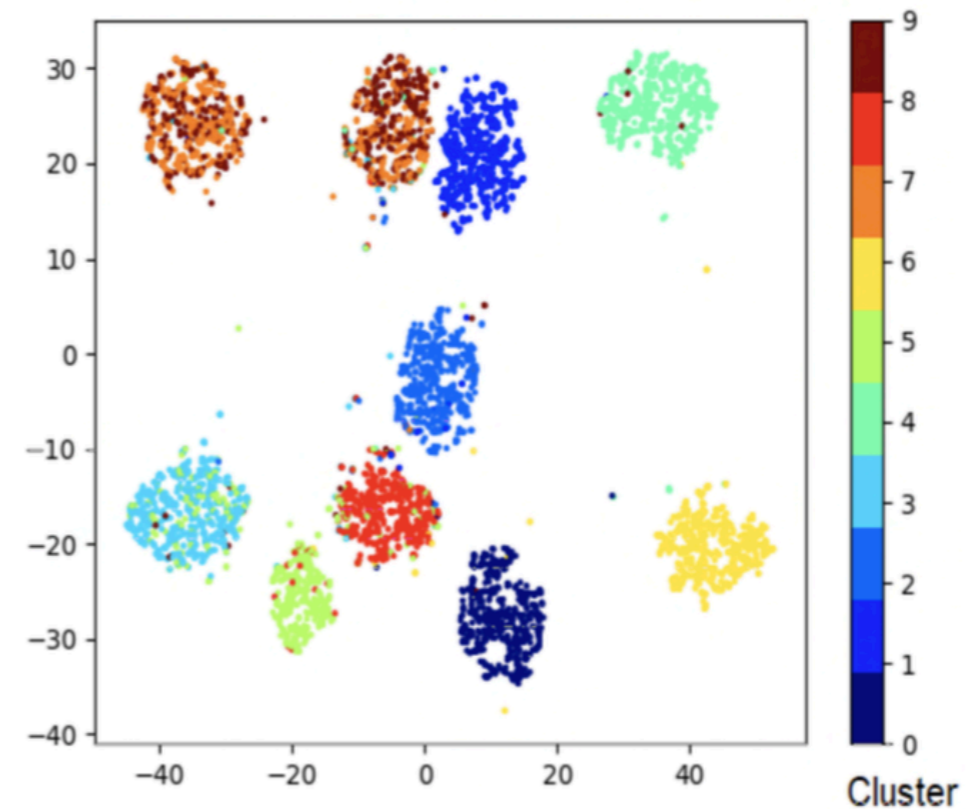
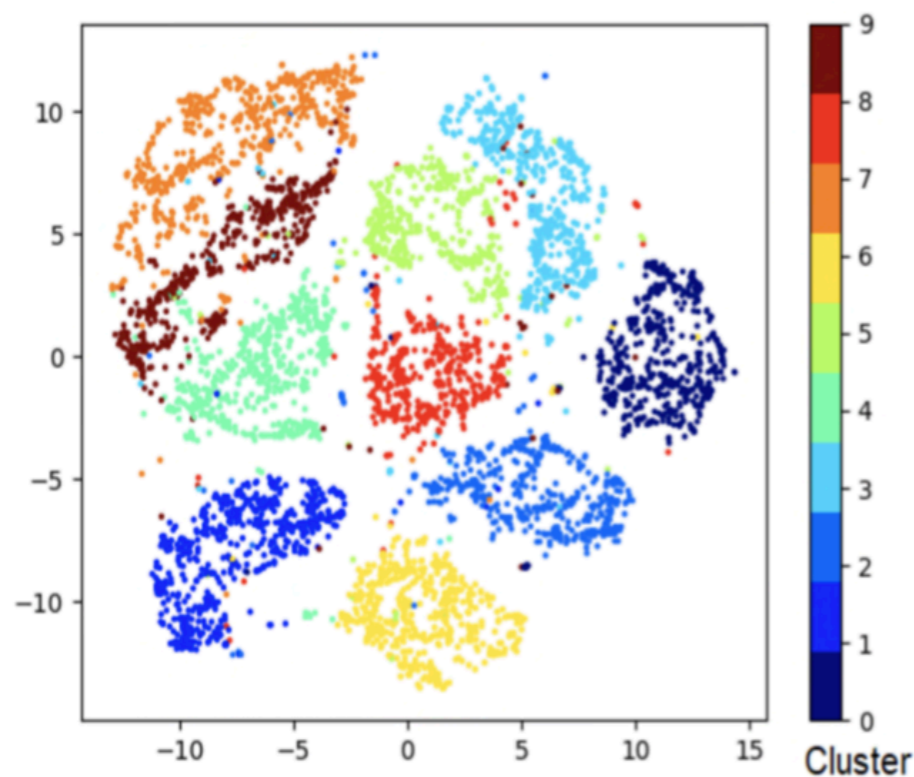
AE example



- For instance we can train the AE with the MNIST dataset to reproduce the input
- The latent space is a compact representation of the MNIST dataset

Latent Space

- We can visualise the latent space (in this case was a 2-d space)
- This is after training with MNIST, the color represent the different numbers



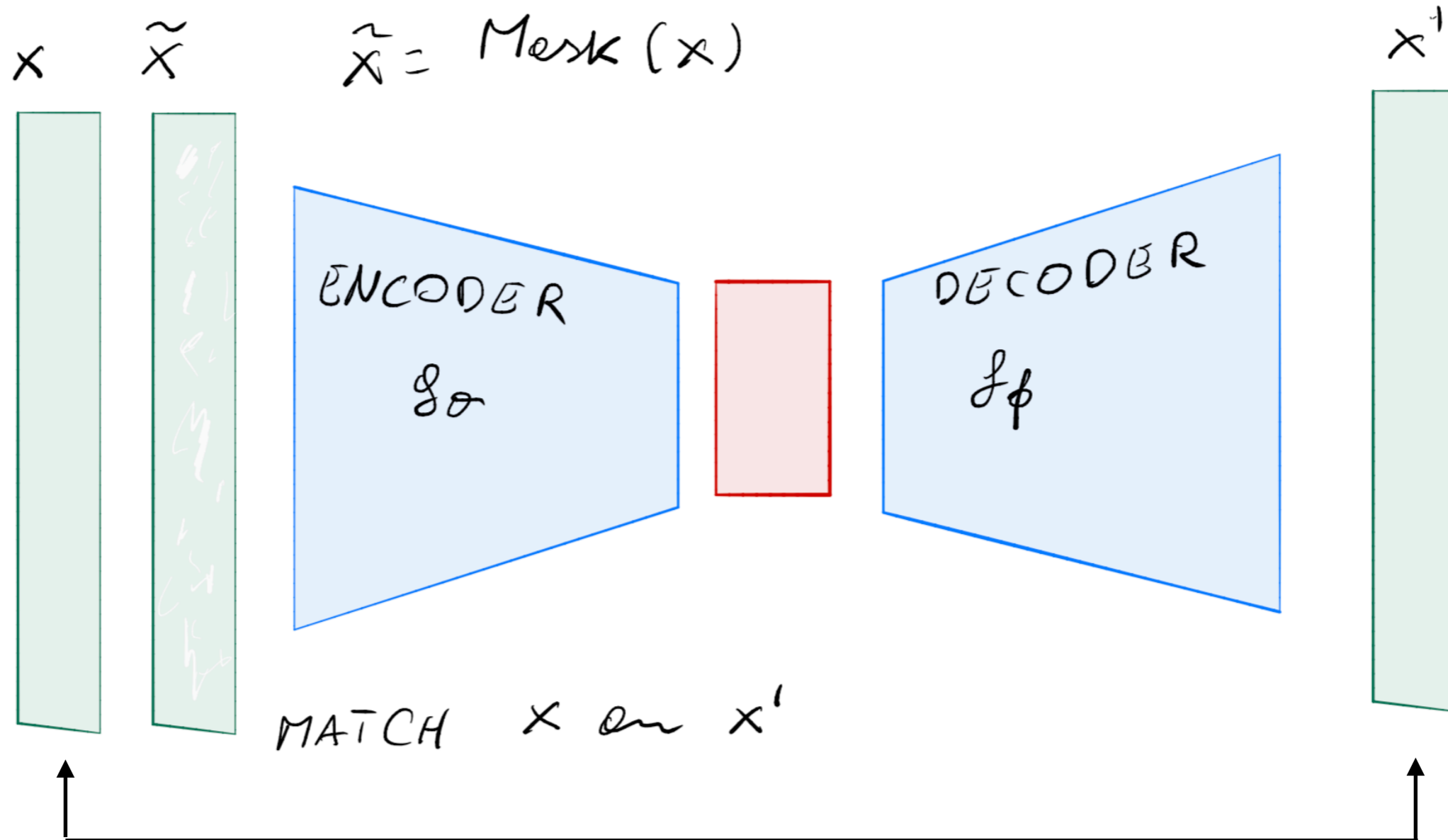
arXiv:1801.07648

Denoising AE

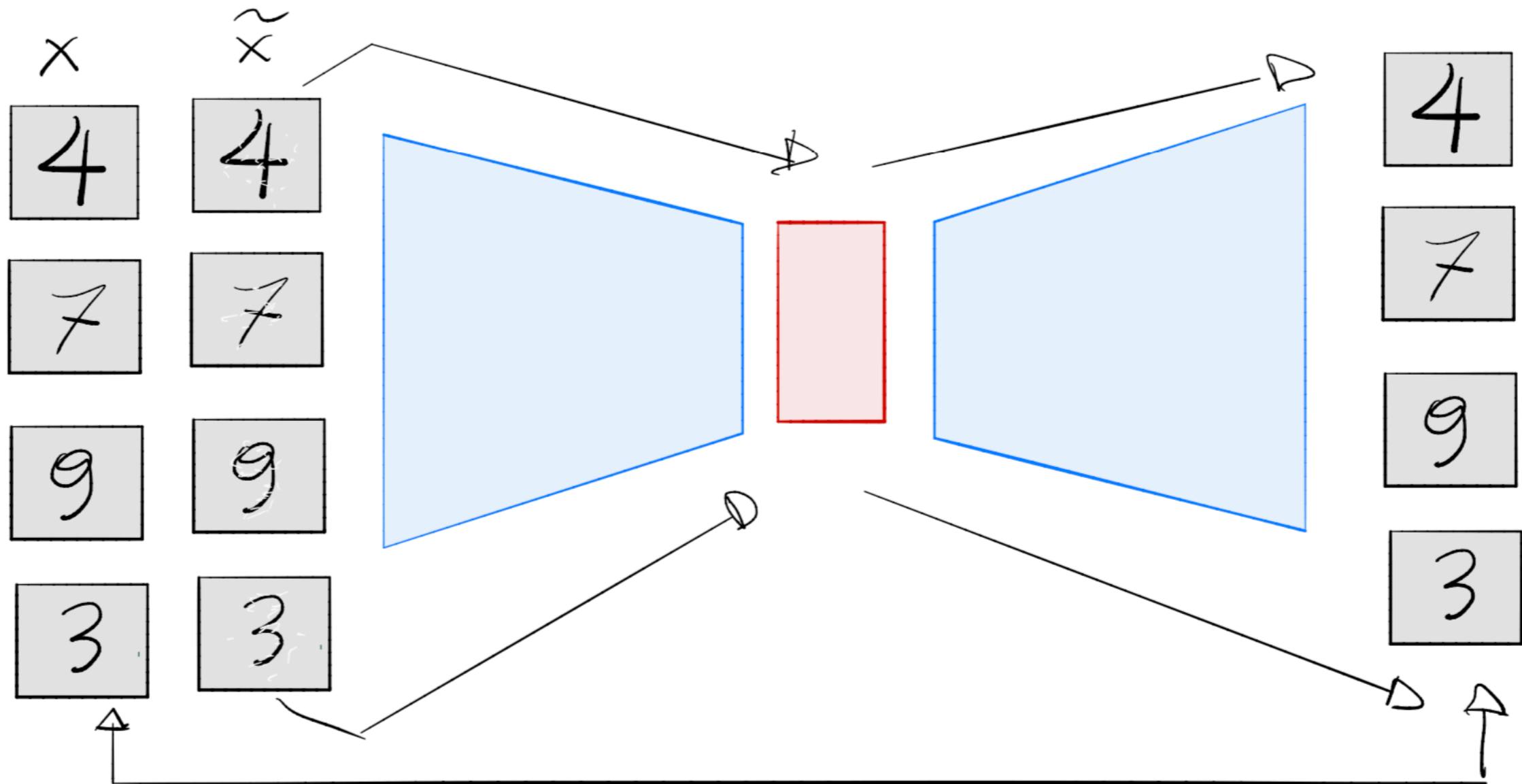
We can use AE to denoise the input:

- We apply a Mask (simulates noise)
- We predict X , giving as input \tilde{X}

$$Loss : \mathcal{L}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^N [X_i - X'_i]^2$$



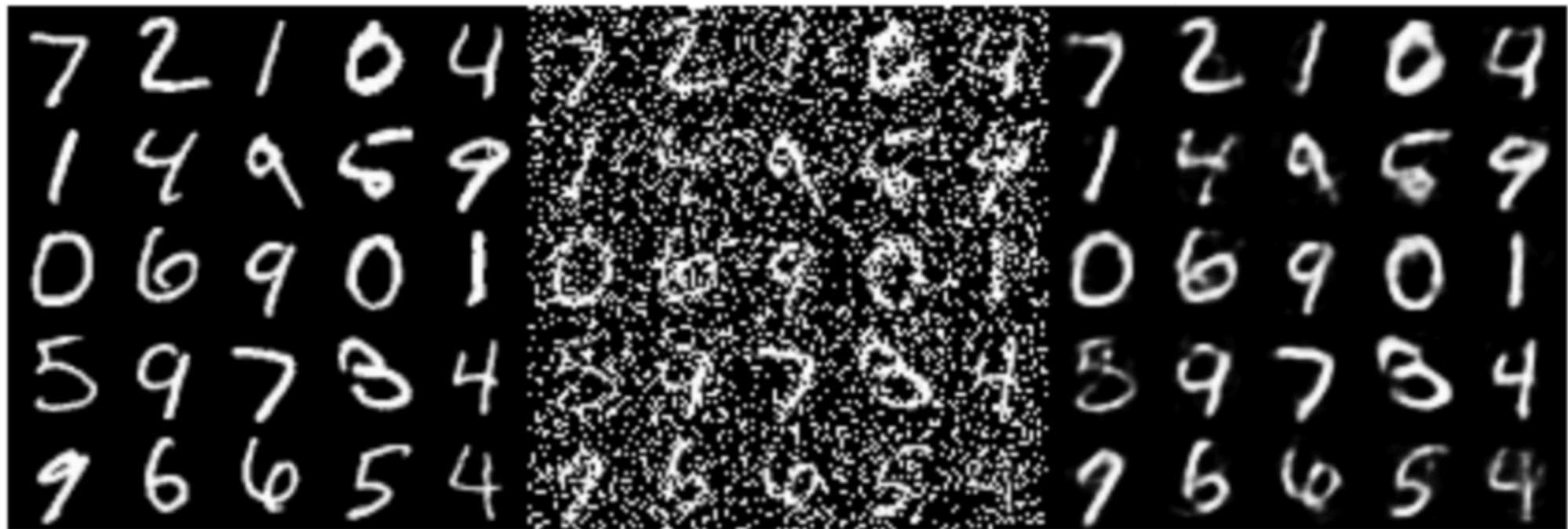
Example with MNIST



$$\text{Loss} : \mathcal{L}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^N [X_i - X'_i]^2$$

Example with MNIST

Application of denoising AE to corrupted MNIST sample



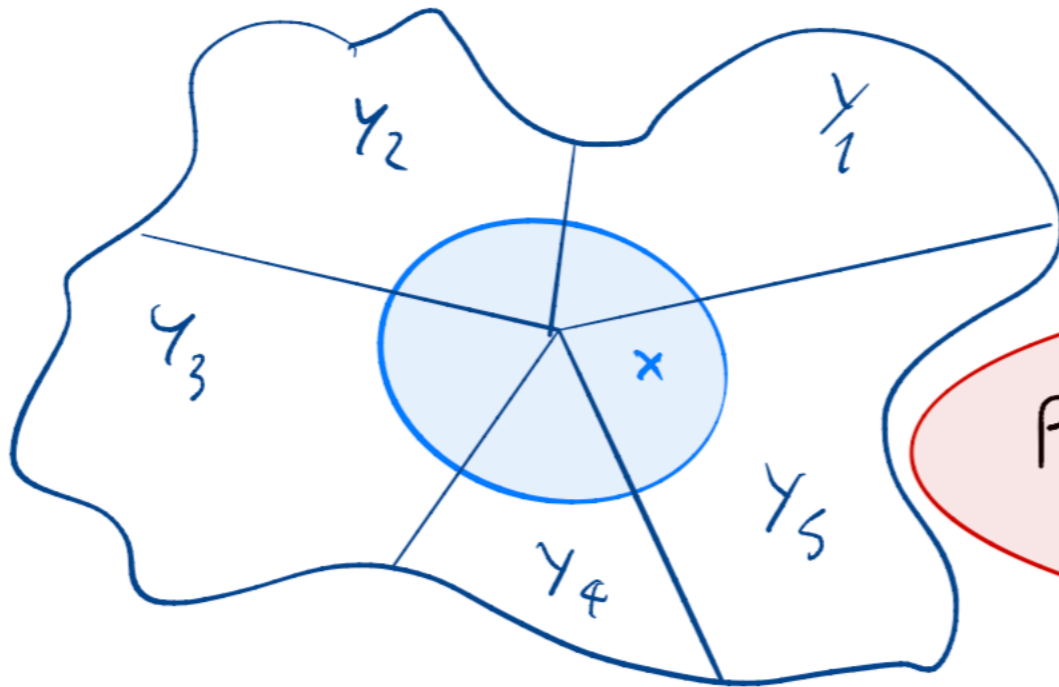
Original input, corrupted data, reconstructed data

Copyright by opendeep.org

Probability

$$\underbrace{P(Y|X)}_{\text{posterior}} = \frac{\underbrace{P(X|Y)}_{\text{likelihood ratio}} \underbrace{P(Y)}_{\text{prior}}}{P(X)} = \frac{P(X, Y)}{P(X)} \rightarrow \text{joint distribution}$$

→ Bayes' Theorem



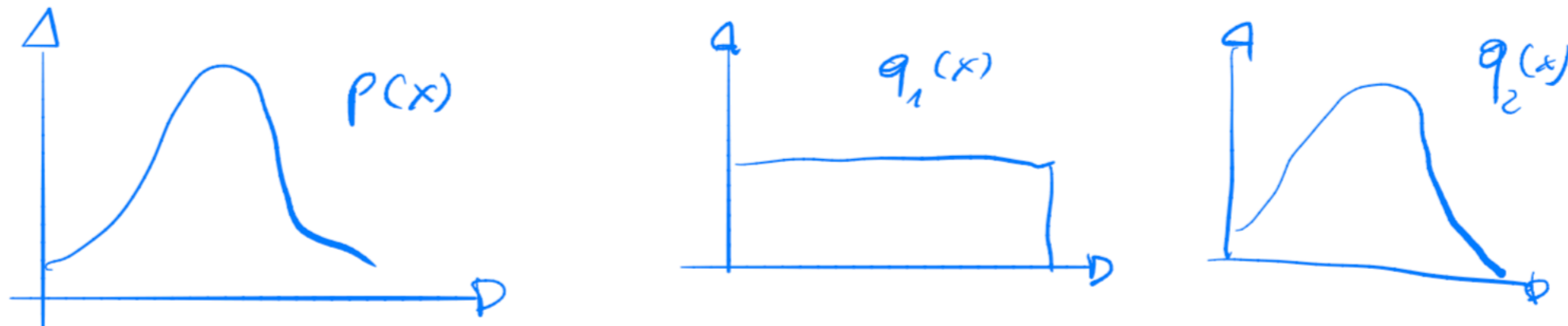
$$P(X) = \sum_i P(X|Y_i) P(Y_i)$$

for $Y_i \cap Y_j = \emptyset$

$$P(Y|X) = \frac{P(X|Y) P(Y)}{\sum_i P(X|Y_i) P(Y_i)}$$

KL divergence

- The Kullback-Leibler (KL) divergence measures how different are two probability density functions (Pdfs)



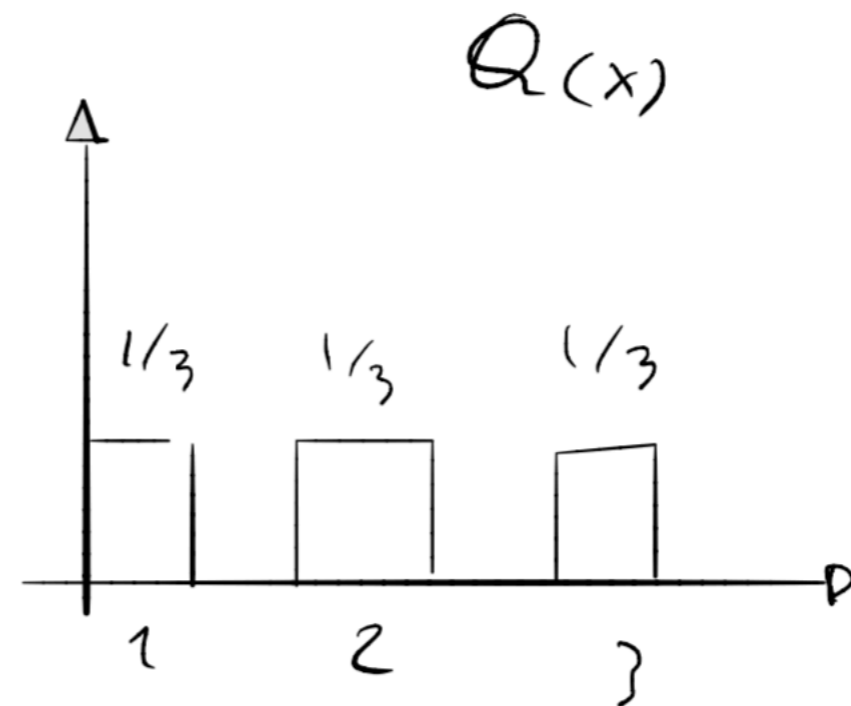
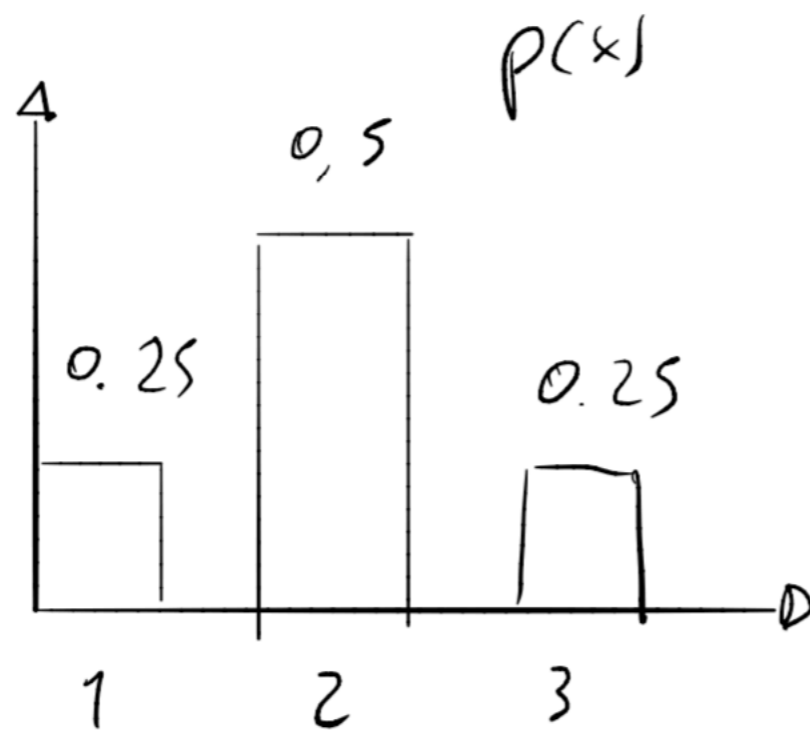
$$KL(P(x) \parallel q_1(x)) > KL(P(x) \parallel q_2(x))$$

$$KL [P(x) \parallel Q(x)] = \int P(x) \text{Log} \left(\frac{P(x)}{Q(x)} \right) dx$$

KLD example

For a discrete distribution we have

$$KL [P(x) || Q(x)] = \sum_i P(x) \text{Log} \left(\frac{P(x)}{Q(x)} \right)$$



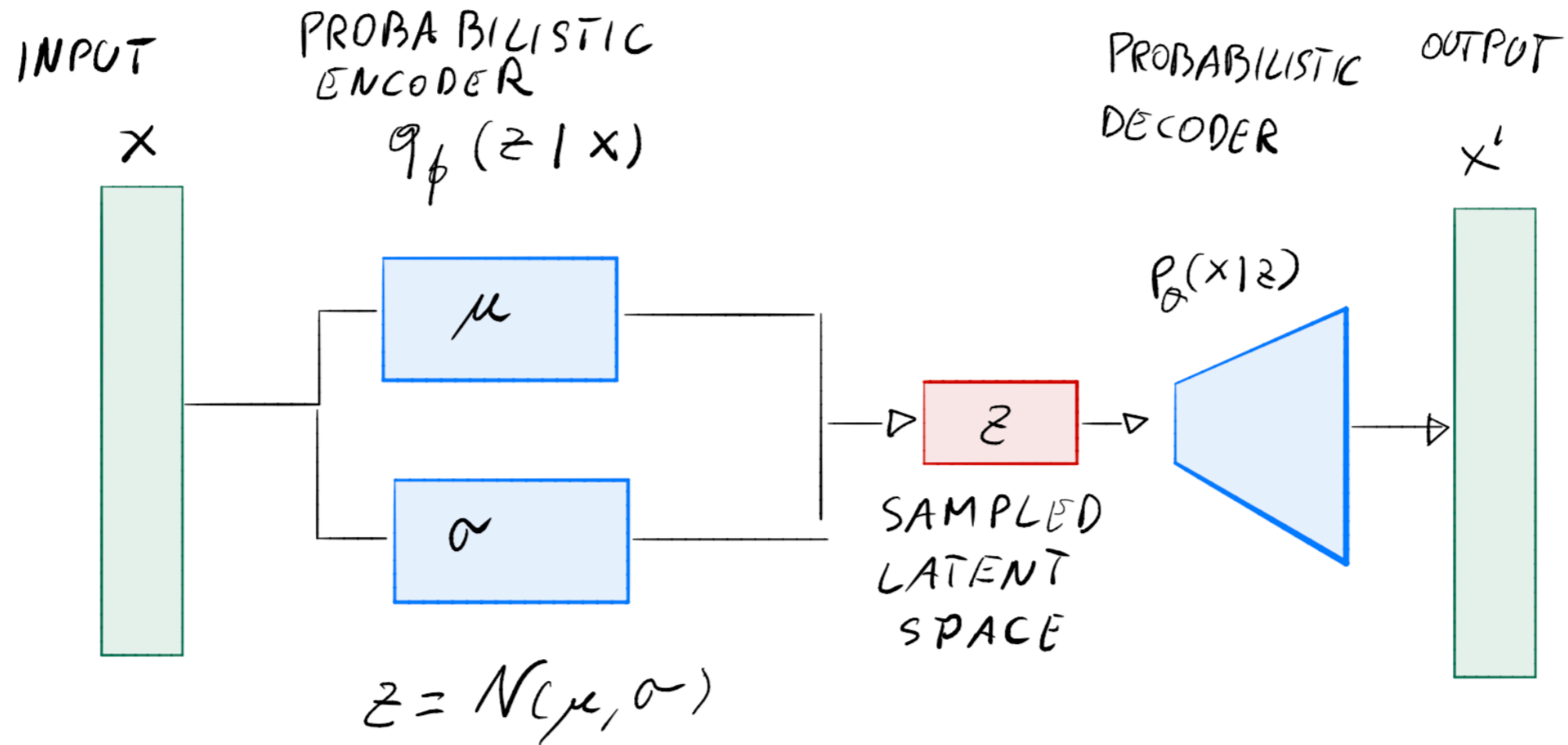
$$KL [P(x) || Q(x)] = 0.25 \text{Log} \left(\frac{0.25}{0.333} \right) + 0.5 \text{Log} \left(\frac{0.5}{0.333} \right) + 0.25 \text{Log} \left(\frac{0.25}{0.333} \right)$$

KLD Properties

- The KL divergence is always positive or zero (if the two PDFs are the same)
- $KL [P(x) || Q(x)] \neq KL [Q(x) || P(x)]$
- If $\mathcal{N}_1 = N(\mu_1, \Sigma_1)$ and $\mathcal{N}_2 = N(\mu_2, \Sigma_2)$ are multidimensional normal distributions

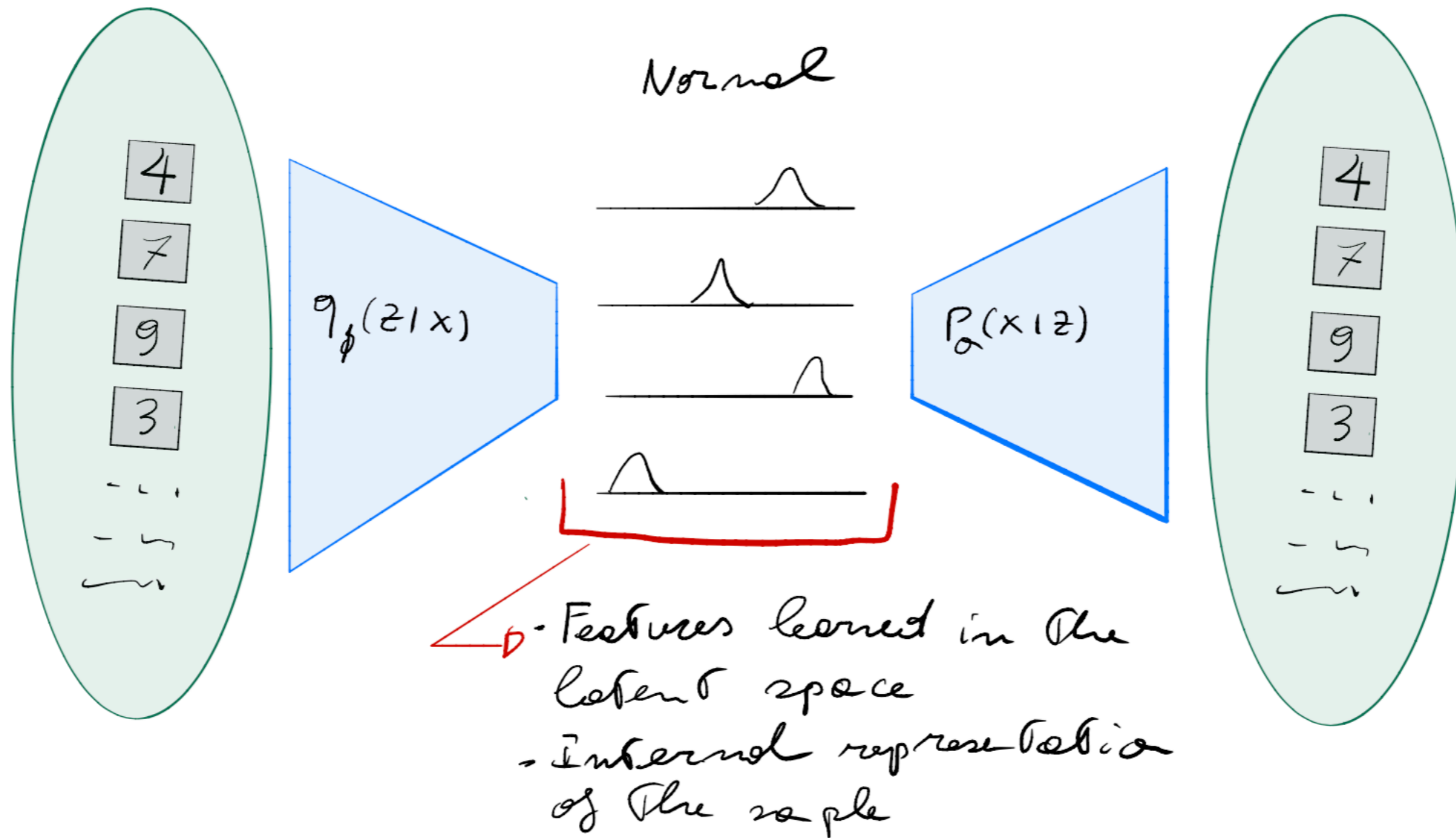
$$KL [\mathcal{N}_1 || \mathcal{N}_2] = \frac{1}{2} \left[\text{Log} \left(\frac{|\Sigma_1|}{|\Sigma_2|} - D + \text{Tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_1 - \mu_2)^T \Sigma_2^{-1} (\mu_2 - \mu_1) \right) \right]$$

Variational AE



The goal of Variational Autoencoders (VAE) aims to find the distribution $q_{\phi}(Z|X)$ such that we can generate the distribution for my sample X sampling from Z using $p_{\theta}(X|Z)$

VAE



VAE are used to generate samples with the distribution learned from data

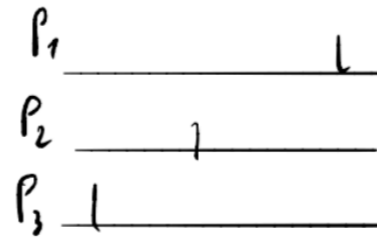
AE vs VAE

AE vs VAE

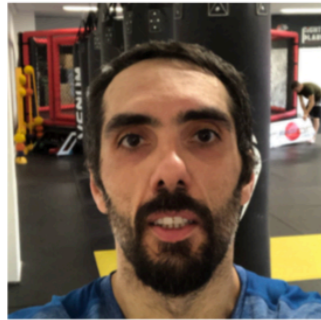


P_1

AE

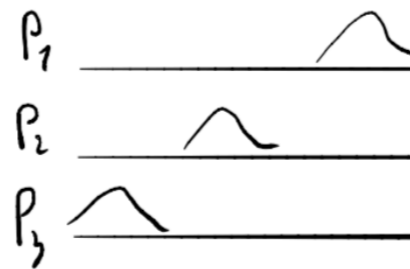


SMILE



P_2

VAE



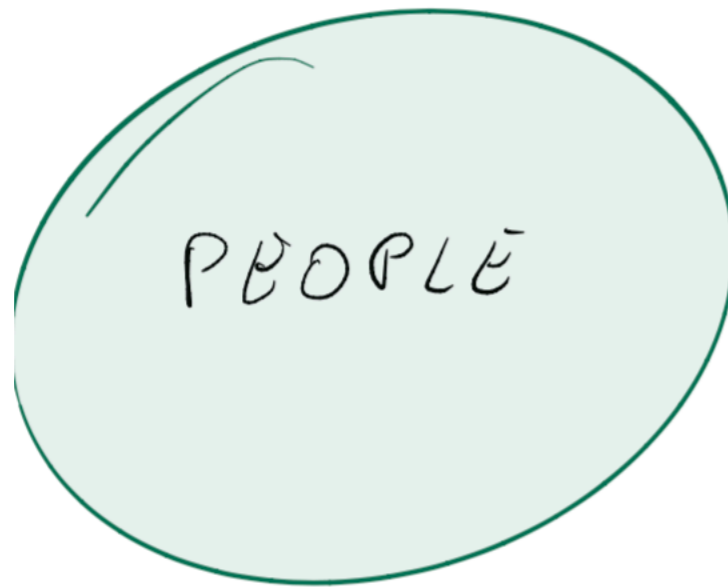
SMILE



P_3

AE vs VAE

SAMPLE



FEATURES IN LATENT SPACE

SMILE _____
 GENDER _____
 GLASSES _____
 HAIR COLOR _____
 - - - - _____

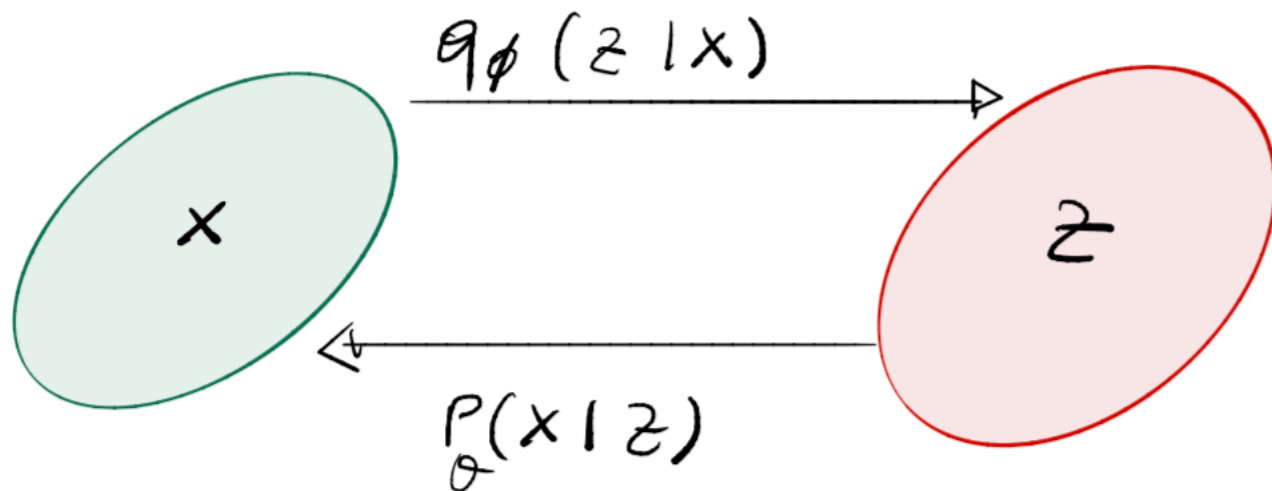
AE

SMILE _____|
 GENDER _____|
 GLASSES _____+
 HAIR COLOR _____|
 - - - - _____|

VAE

SMILE _____^
 GENDER _____^
 GLASSES _____^
 HAIR COLOR _____~
 - - - - _____^

VAE Loss



Our goal is to use gradient descent to learn the parameters ϕ, θ

Using Variational Inference it can be shown that the appropriate loss to train VAE is:

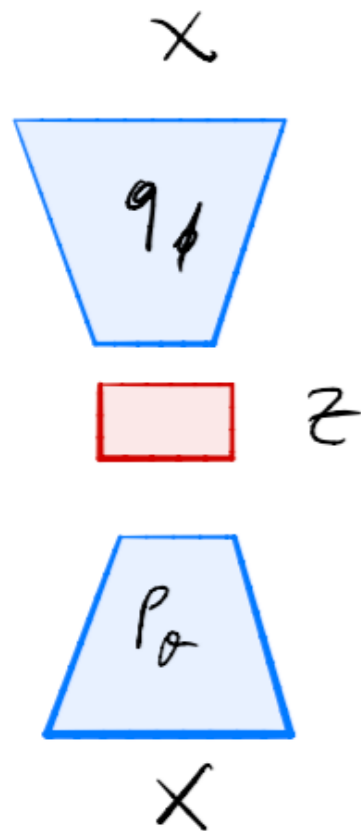
$$\mathcal{L} = -E_{Z \sim q_\phi(Z, X)} [\text{Log} P_\theta(X|Z)] + \text{KL} [q_\theta || P(z)]$$

VAE LOSS

$$\mathcal{L} = - E_{Z \sim q_{\phi}(Z, X)} [\text{Log} P_{\theta}(X | Z)] + \text{KL} [q_{\theta} || P(z)]$$

Reconstruction Loss

Regularizer



Sampling from Z with

$$q_{\phi}(Z | X)$$

We want to maximise the probability of obtaining

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

We need to have a prior for Z , $P(Z)$ because we need to know how to sample from Z , e.g. a normal distribution

- We map the distribution data X into a known distribution $P(Z)$
- We map the known distribution Z into the data distribution of X

VAE Loss

$$\mathcal{L} = -E_{Z \sim q_{\phi}(Z, X)} [\text{Log} P_{\theta}(X | Z)] + \text{KL} [q_{\theta} || P(z)]$$

Data Fidelity ←

→ KL Divergence

Ensure that sampling from Z
we get the data distribution X

This is equivalent to the
square error

Ensures that the latent space
is distributed according with
our prior

Avoid collapsing, i.e. ensure
that we do not get zero
variance, therefore that we
learn to generate new
samples

VAE pseudocode

VAE example

MNIST from VAE using Pyro (pyro.ai)

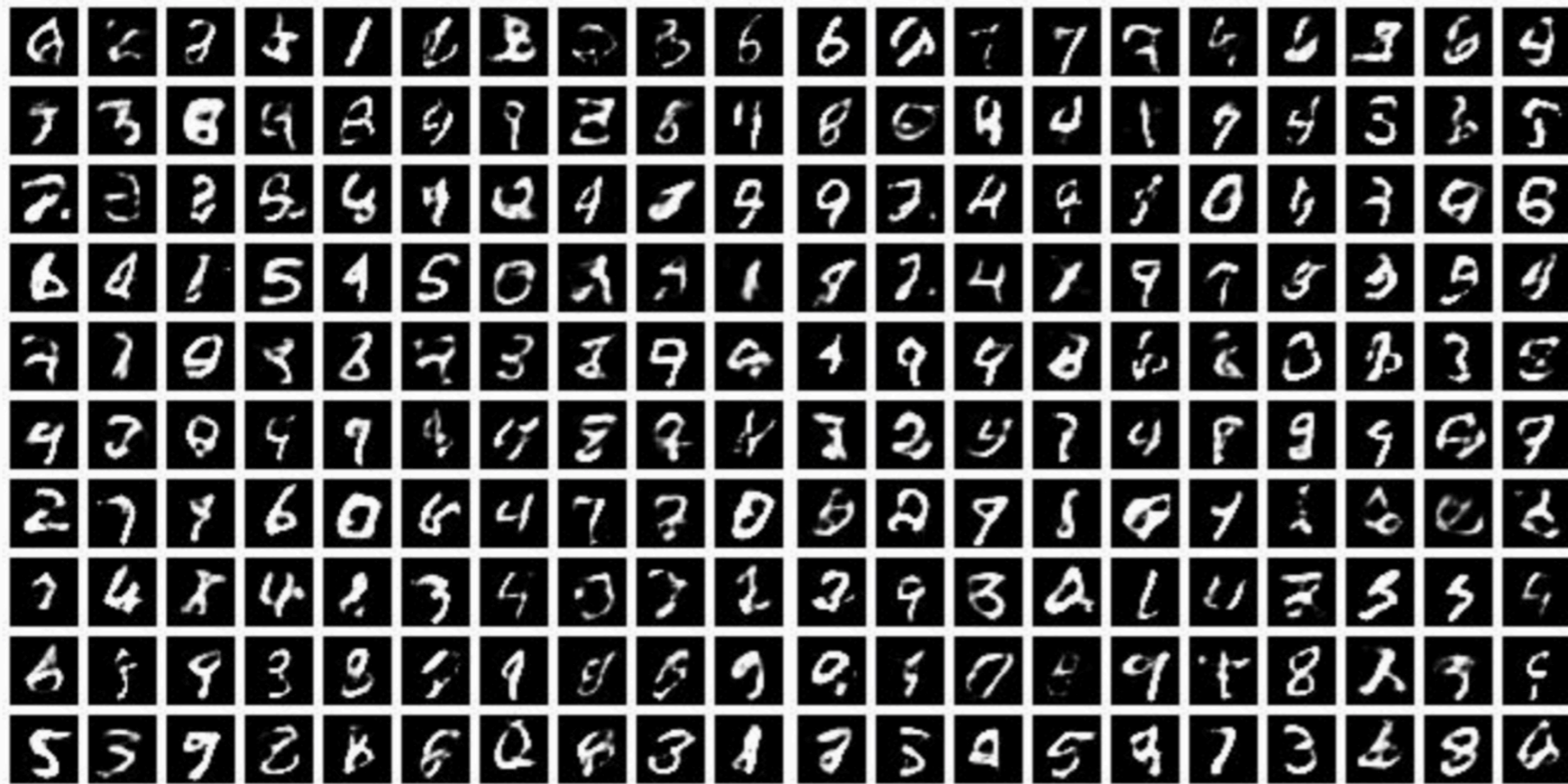
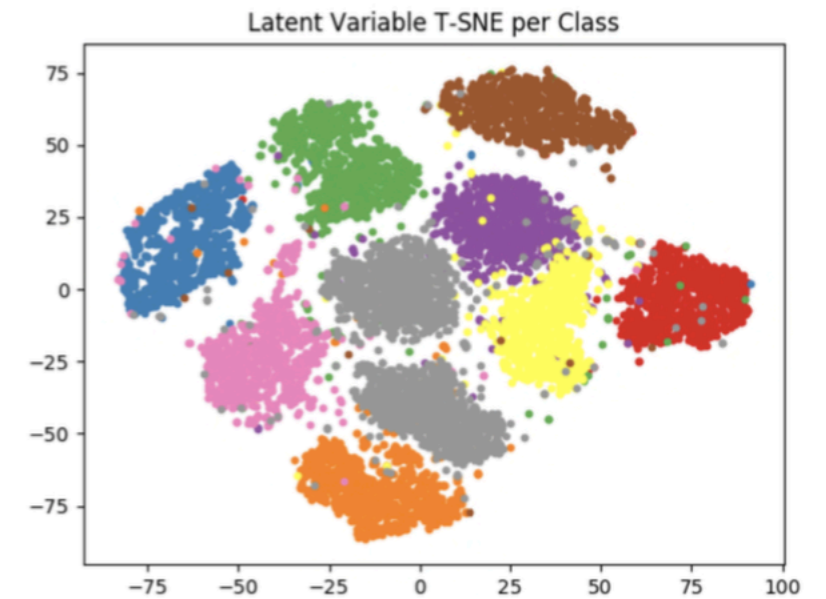


Figure 4: Samples from generative model.



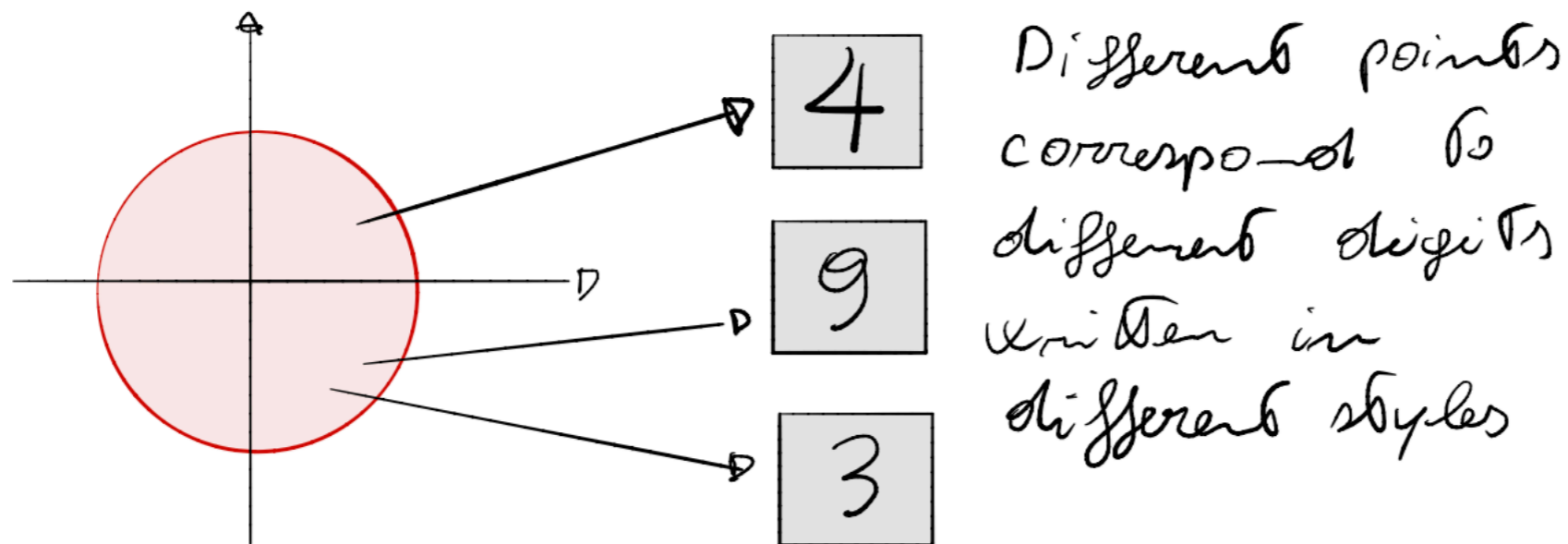
VAE Example



Manga characters generated
with VAE (mc.ai)

Conditional VAE

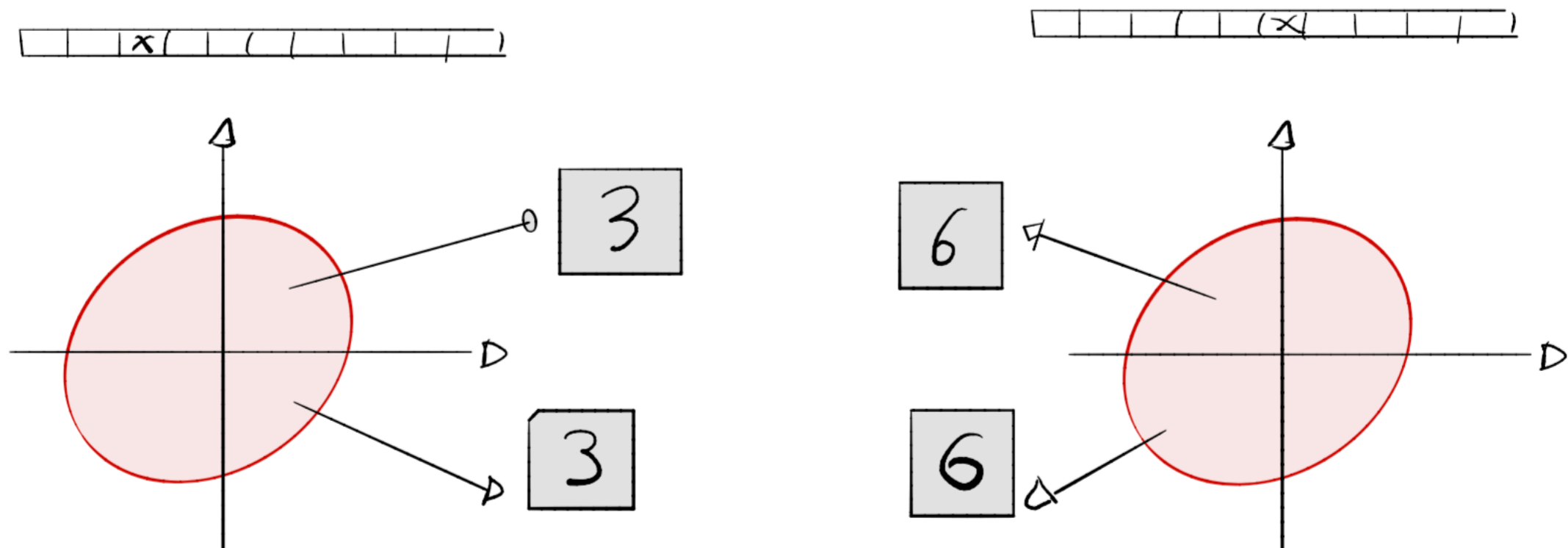
VAE allow to generate the data distribution learned from data



However, sometimes you want to generate a subset of data, e.g. you want to generate the number 6 with different handwritings

Conditional VAE

In addition to the latent space, you add an input, for instance the MNIST label



The latent space will specify style related things, while the number is given by the labels

CVAE Loss

The loss of the VAE is given by:

$$\mathcal{L} = -E_{Z \sim q_\phi(Z, X)} [\text{Log} P_\theta(X | Z)] + \text{KL} [q_\theta || P(z)]$$

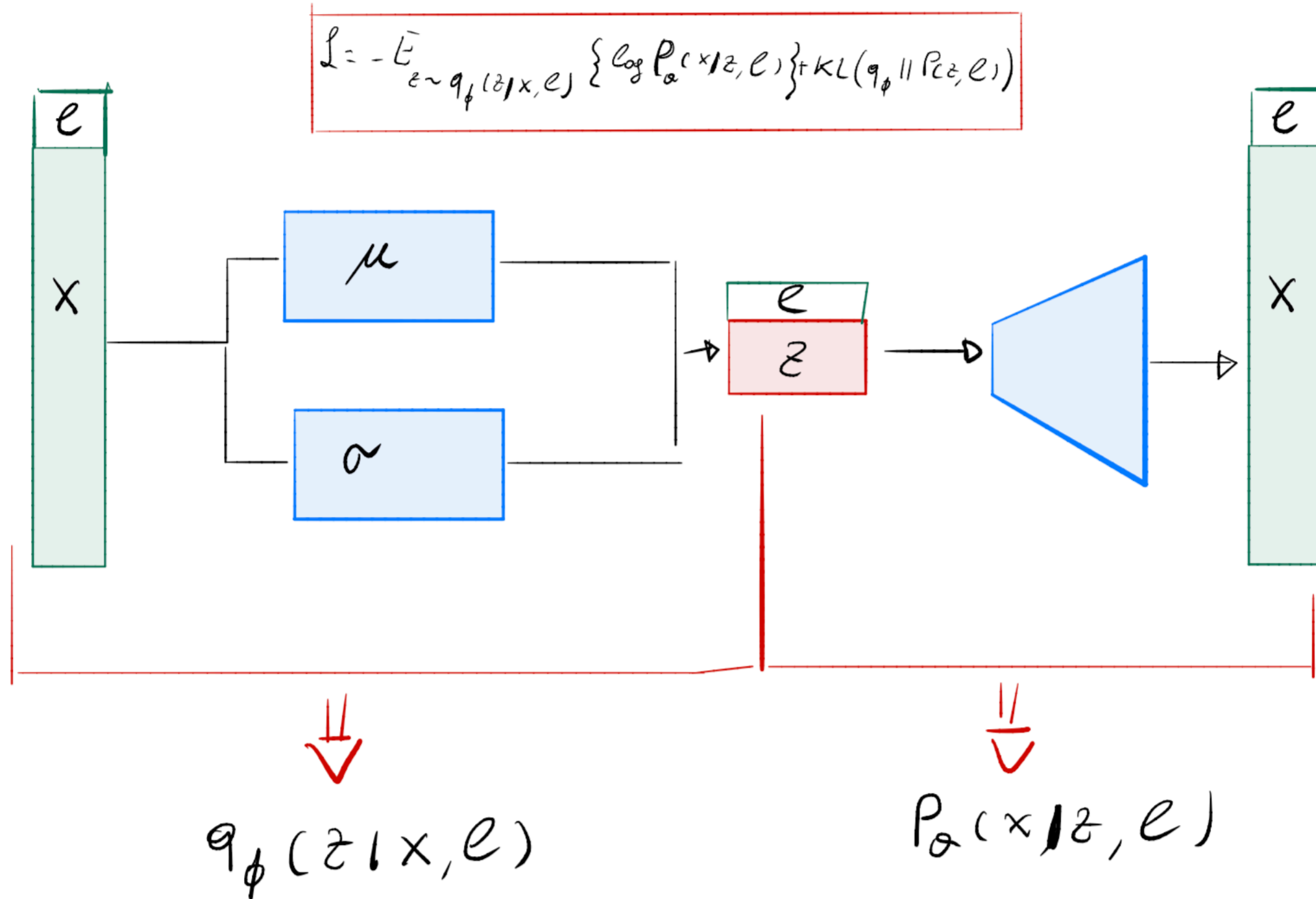
Now we need to condition this loss to a label or a new variable ℓ

$$\mathcal{L} = -E_{Z \sim q_\phi(Z, X, \ell)} [\text{Log} P_\theta(X | Z, \ell)] + \text{KL} [q_\theta || P(z, \ell)]$$

label

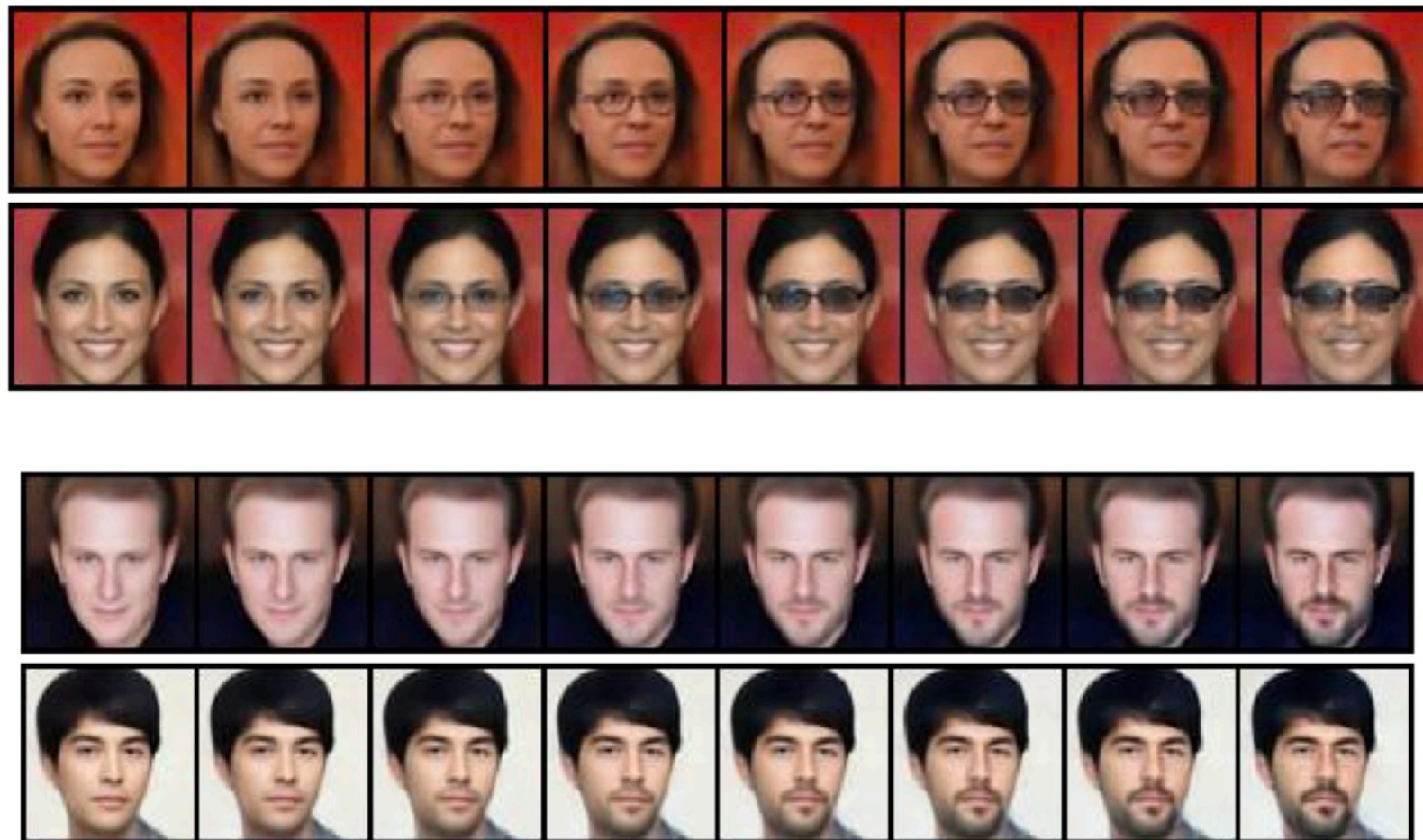


CVAE



CVAE Example

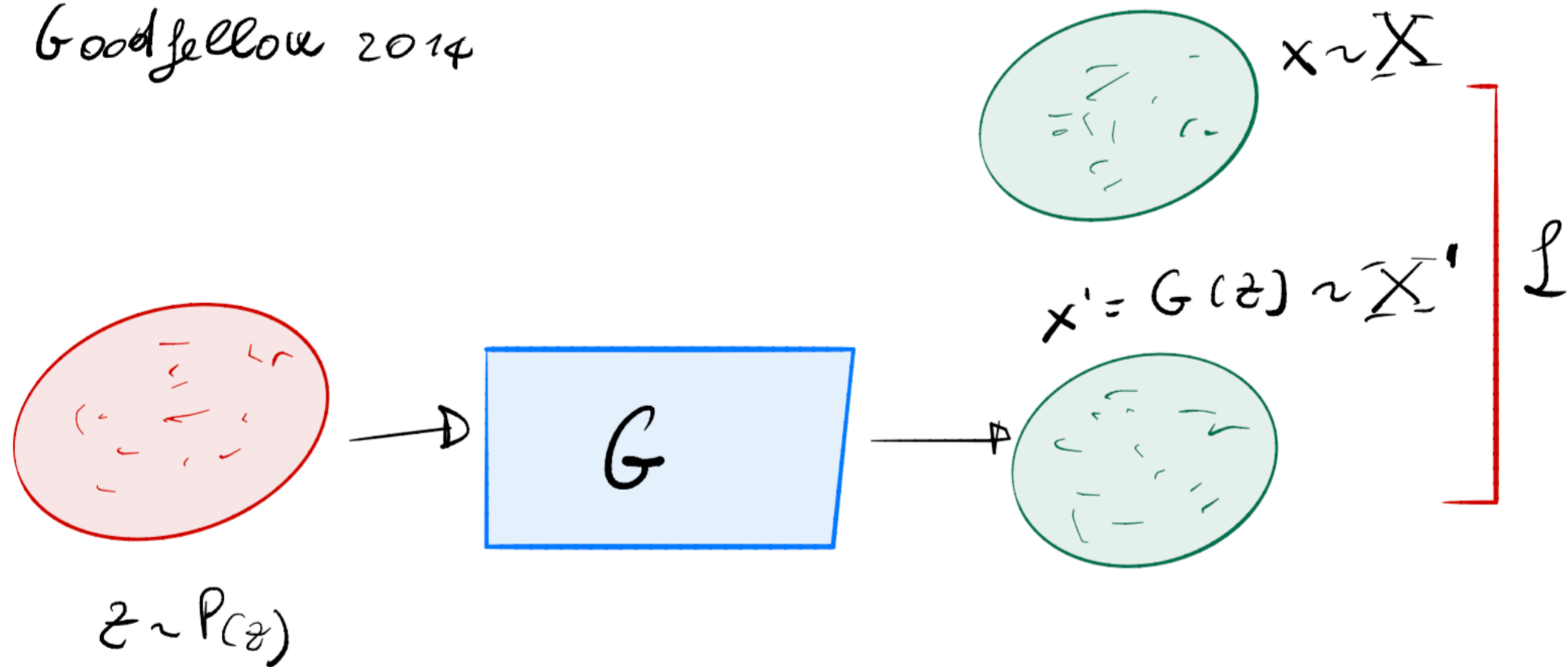
CVAE can be used, in addition to generate sub-samples of data, to add features to samples



J. Klys, J. Snell, R. Zanel - University of Toronto

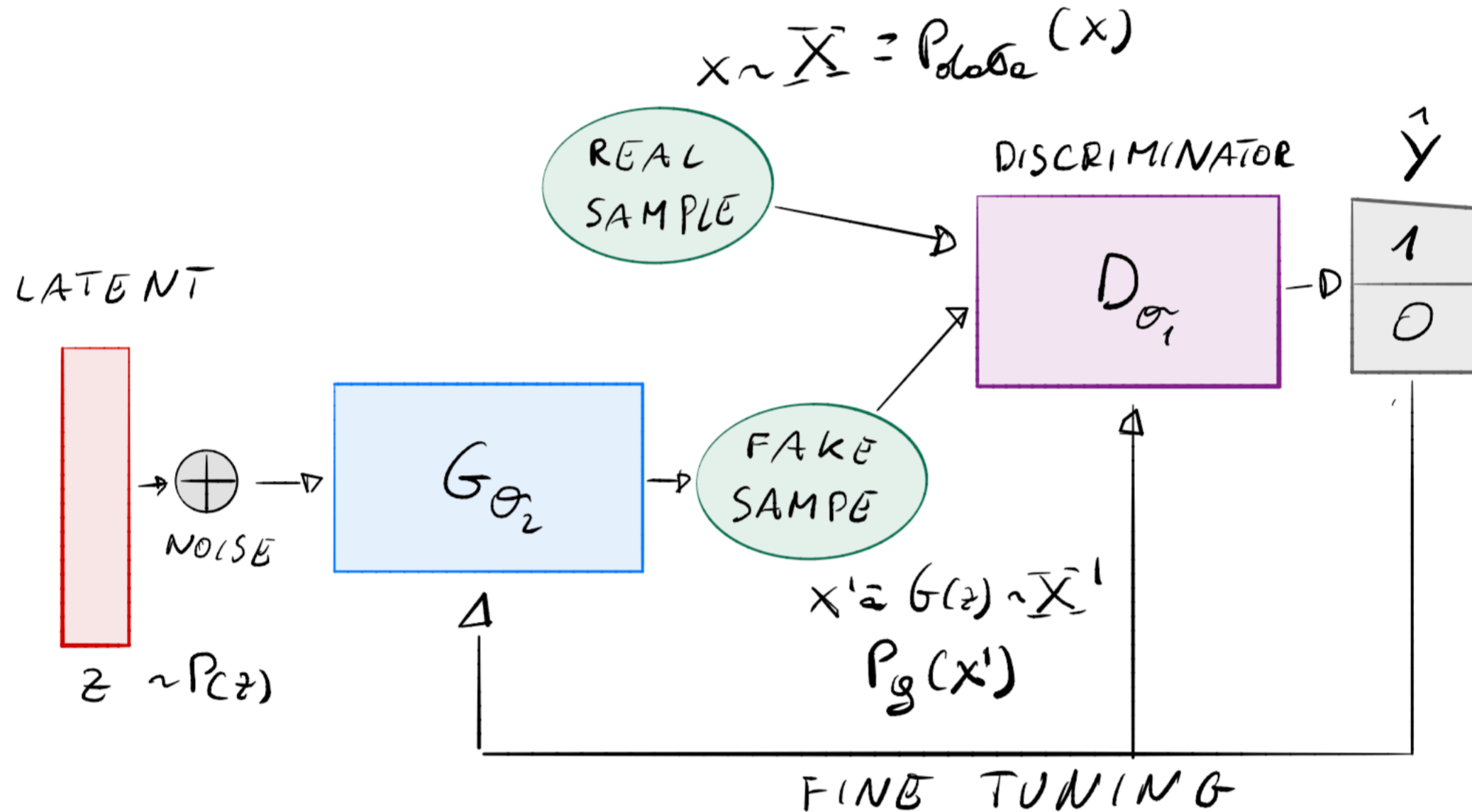
GANs

Goodfellow 2014



Generative Adversarial Networks (GANs) are becoming very popular to generate samples. The objective of GANs is to minimise the distance between X and X'

GANs



Goal: $P_g(X') \sim P_{data}(X)$

NB: X and X' must have the same shape

Training GANs

- GANs are trained alternating Generator and Discriminator
- The goal of the Generator is to generate samples that reproduce the data distribution, the goal of the Discriminator is to distinguish between “fake” and “real” samples
- The Discriminator is a classifier
- The Generator is trained to fool the Discriminator
- The weights of the Generator (W_G) are fixed when training the Discriminator (W_D) and vice versa
- When Generator and Discriminator reach the Nash equilibrium, the generated sample is undistinguishable from the data

Training GANs

Since D is a simple classifier its loss is the binary cross entropy

Label Output of D X samples are labelled as 1
 X' samples are labelled as 0

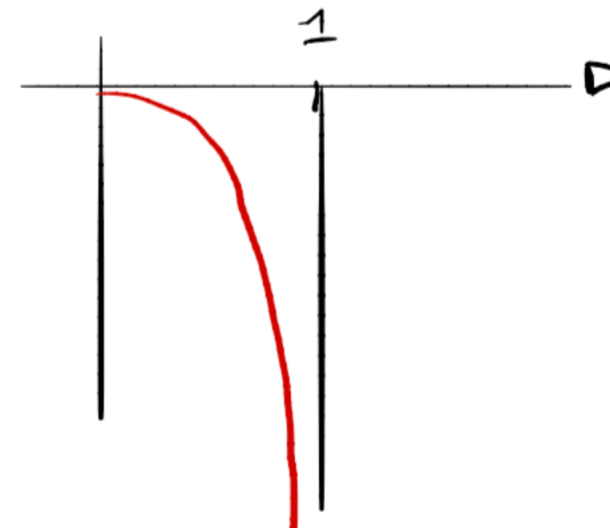
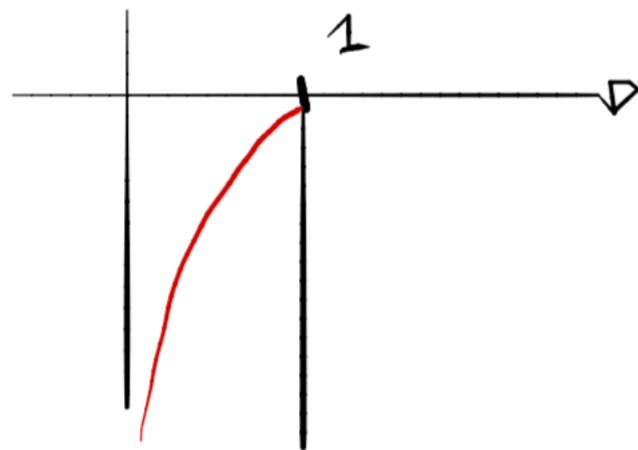
$$\mathcal{L}_D = \left[\underbrace{y \text{Log}(\hat{y})}_A + \underbrace{(1 - y) \text{Log}(1 - \hat{y})}_B \right]$$

$$\mathcal{L}(D, 1) = \text{Log}\{D(X)\} \quad A$$

$$\mathcal{L}(D, 0) = (1 - 0) \text{Log}\{1 - D(X')\} = \text{Log}\{1 - D(G(Z))\} \quad B$$

Training Discriminator

$$\mathcal{L}_D = \{ \text{Log}(D(X)) + \text{Log}(1 - D[G(Z)]) \}$$



To train the Generator

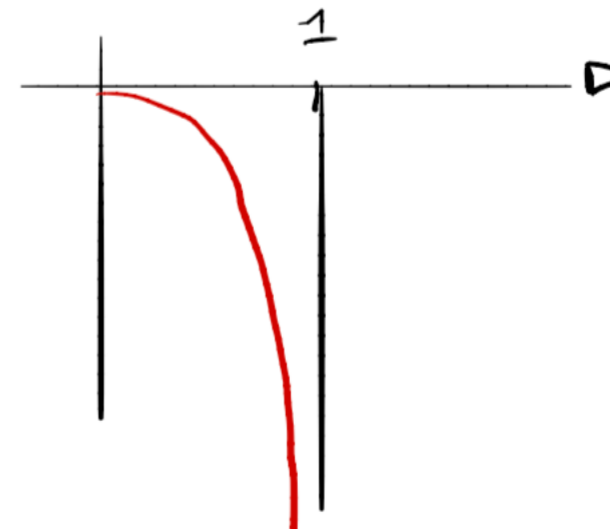
$$D(X) \rightarrow 1$$

$$D(G(Z)) \rightarrow 0$$

Training Generator

$$\mathcal{L}_D = \{ \text{Log}(D(X)) + \text{Log}(1 - D[G(Z)]) \}$$

This term does not depend on the generator



To train the Discriminator

$$\text{Log}[1 - D(G(Z))] \rightarrow -\infty$$

$$D(G(Z)) \rightarrow 1$$

GAN Optimization

$$\mathcal{L} = \min_G \max_D \{ \text{Log}[D(x)] + \text{Log}[1 - D(G(Z))] \}$$

Since we have a sample rather than a single event we get

$$\mathcal{L} = \min_G \max_D \{ E_{X \sim P_{data}} (\text{Log}[D(x)]) + E_{Z \sim P_g} \text{Log}[1 - D(G(Z))] \}$$

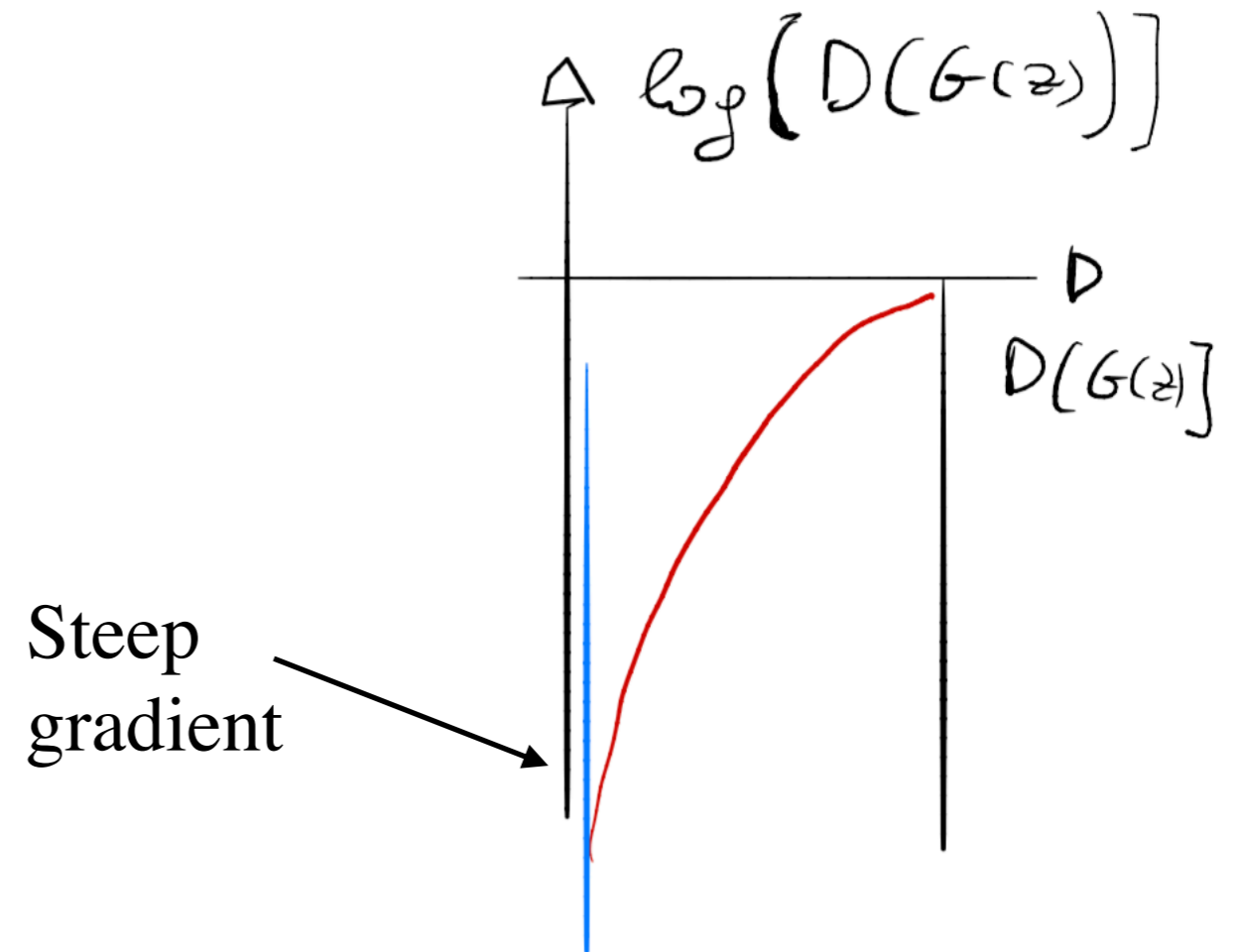
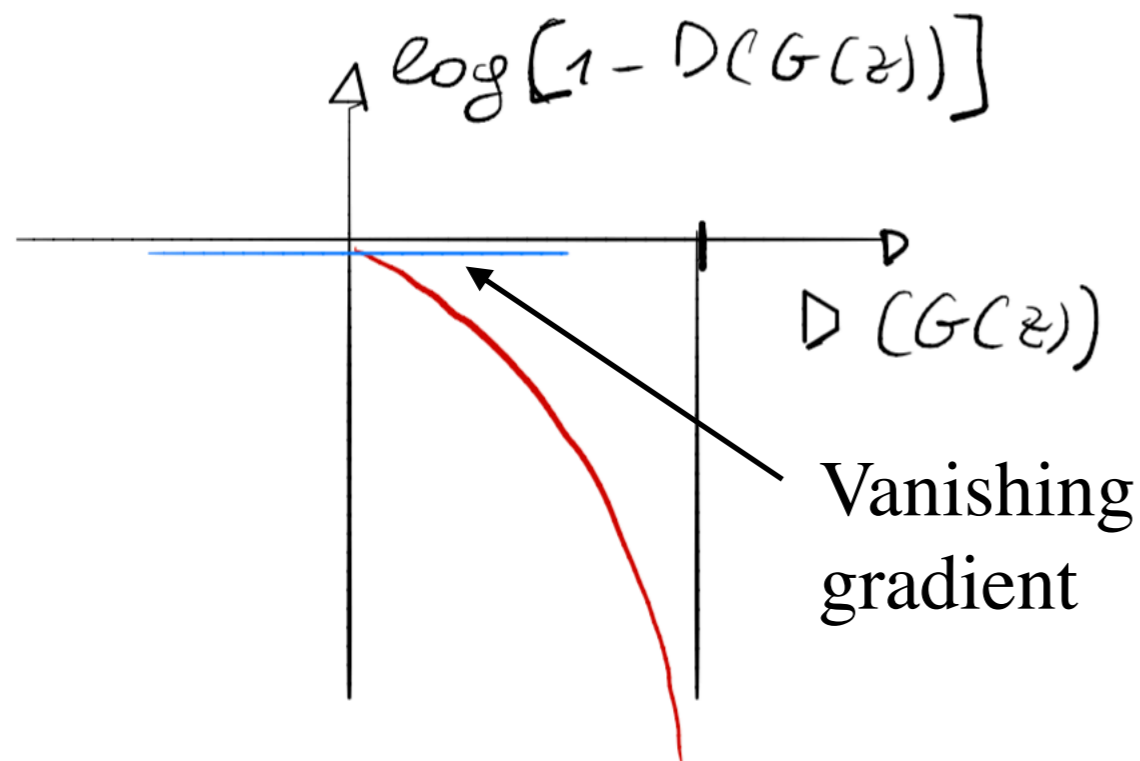
For epochs:

For k steps:

- Sample from P_g (Z) = (Z_1, Z_2, Z_3, \dots) and $y=0$
- Sample from P_{data} (X) = (X_1, X_2, X_3, \dots) and $y=1$
- Update D ascending $\nabla_{W_D} \left\{ \frac{1}{m} \sum_1^m [\text{Log}D(X_i) + \text{Log}(1 - D(G(Z_i)))] \right\}$
- Sample from P_g (Z) = (Z_1, Z_2, Z_3, \dots) and $y=0$
- Update G descending $\nabla_{W_G} \left\{ \frac{1}{m} \sum_1^m \text{Log}(1 - D(G(Z_i))) \right\}$

Vanishing Gradient

- If we start with $D[G(Z)] \sim 0$ the gradient is also ~ 0 , so the weights will not be updated



To solve this problem the cost for the generator can be changed to the equivalent

$$\mathcal{L}_G = \max_G \left(E_{Z \sim P_g} \{ \text{Log}[D(G(Z))]\} \right)$$

Final GAN cost

Cost for the Discriminator:

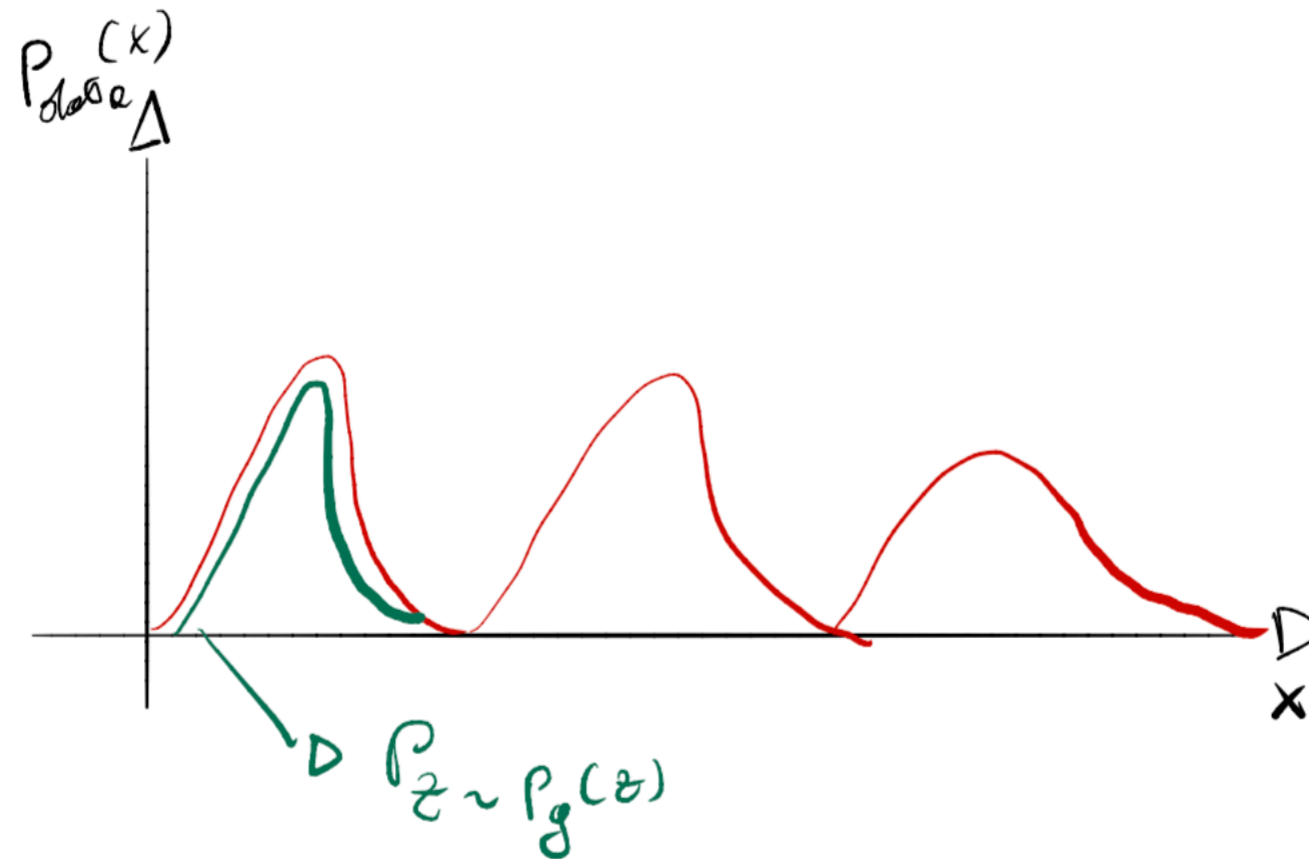
$$\mathcal{L} = \max_D \{ E_{X \sim P_{data}} (\text{Log}[D(x)]) + E_{Z \sim P_g} \text{Log}[1 - D(G(Z))]\}$$

Cost for the Generator:

$$\mathcal{L} = \max_G \{ E_{Z \sim P_g} \text{Log}[D(G(Z))]\}$$

Mode Collapse

- Suppose data is multimodal
- The generator can fool the discriminator by only learning 1 mode, in the worst case 1 example

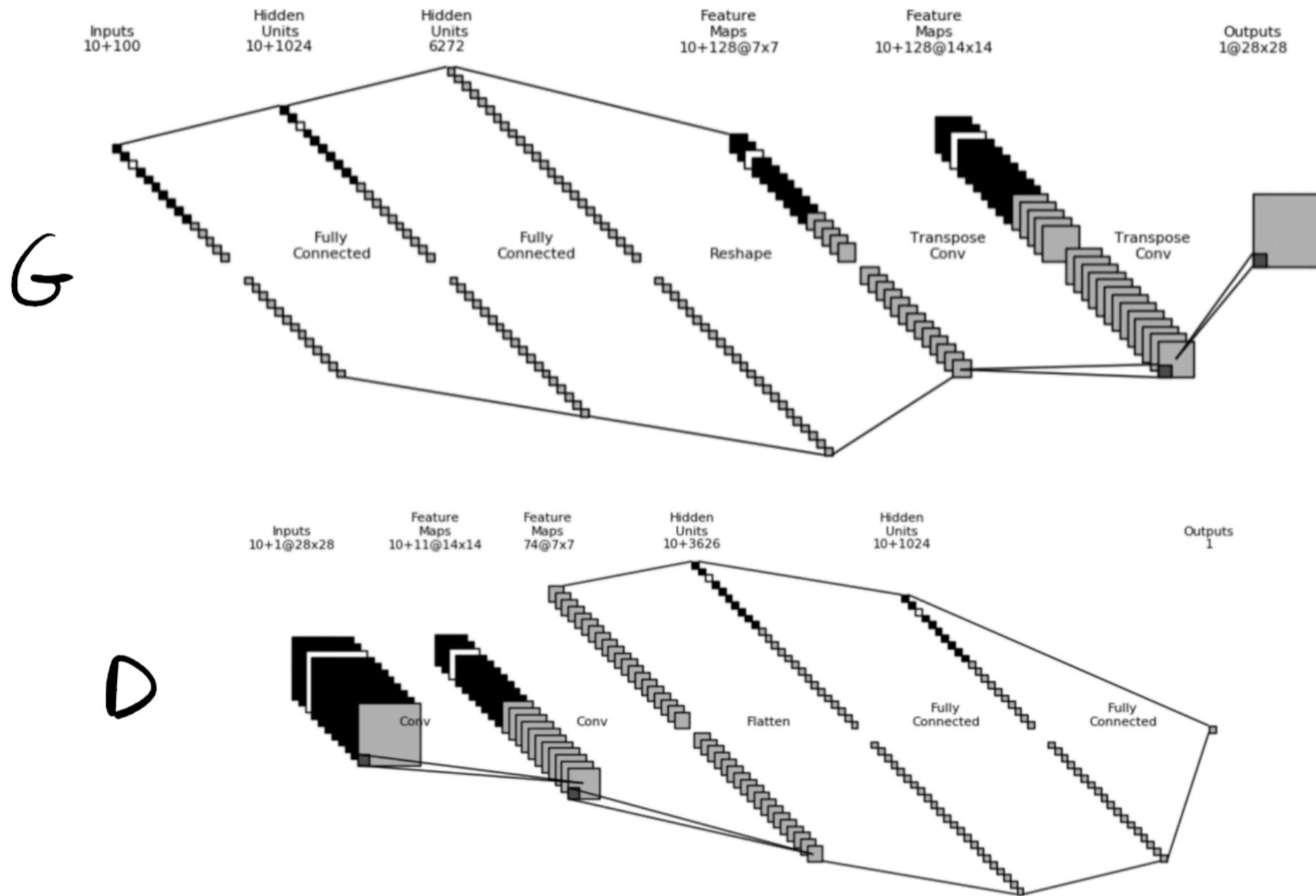


- There are several ways to prevent mode collapse, have a look here <https://medium.com/intel-student-ambassadors/reducing-mode-collapse-in-gans-using-guided-latent-spaces-36f52a08a668>

GAN Example

About Greg Heinrich

<https://devblogs.nvidia.com/tag/gan/>



GAN Example

About Greg Heinrich

<https://devblogs.nvidia.com/tag/gan/>

MNIST



GAN Example

About Greg Heinrich

<https://devblogs.nvidia.com/tag/gan/>

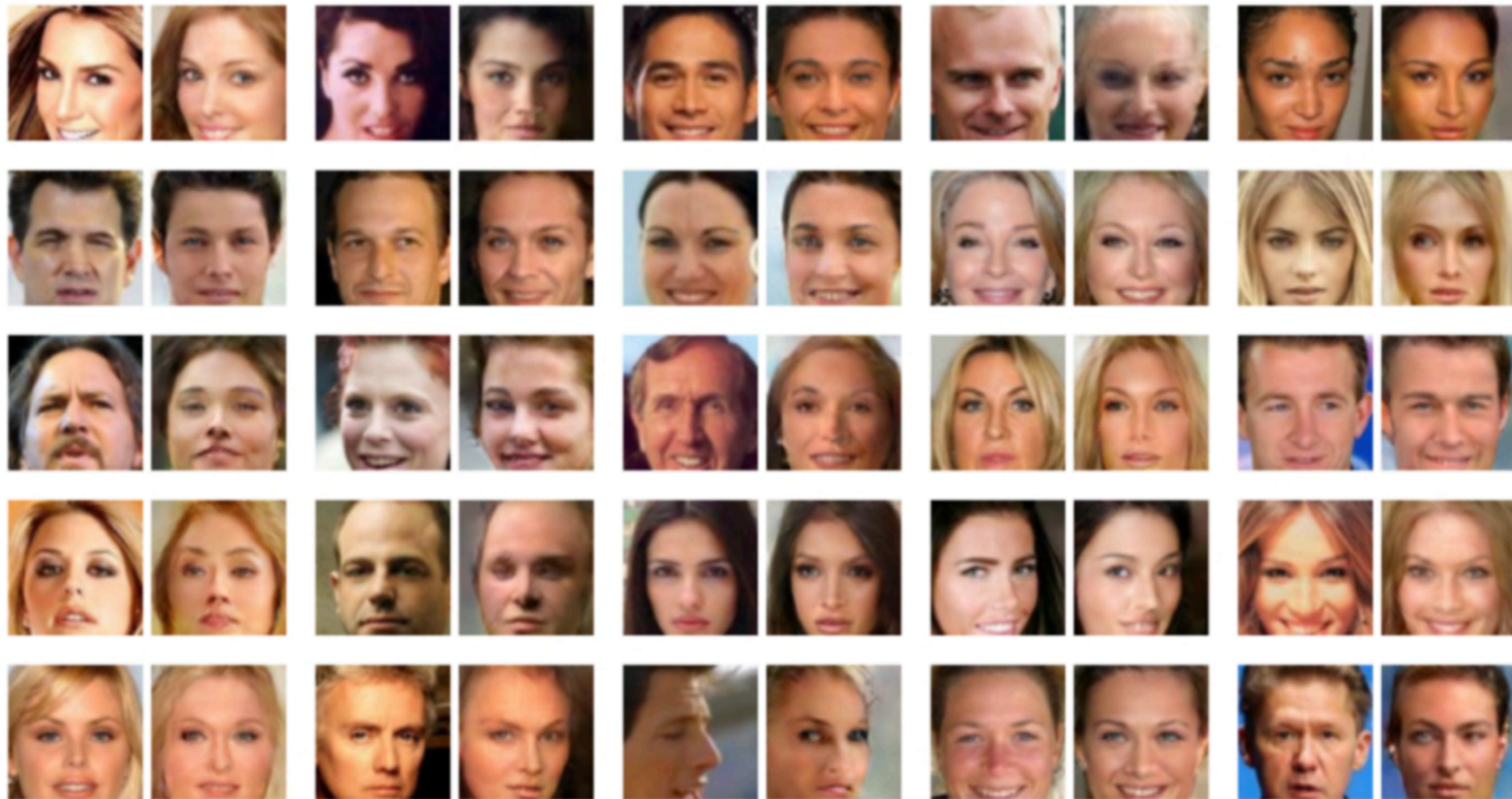


Figure 4: Each pair of images shows an image from the dataset and its reconstruction after going through Generative Adversarial Networks **E** and **G**. 25 images from the dataset were used. Images were not cherry picked.

Conclusion

- We have seen that unsupervised learning can be used to learn features in data, learn the structure of data, generate data, add features, denoise data, style transfer, ...
- We have seen different architectures:
 - Autoencoders
 - Variational Autoencoders (<https://www.tensorflow.org/tutorials/generative/cvae>)
 - Generative Adversarial Networks (<https://www.tensorflow.org/tutorials/generative/dcgan>, <https://www.tensorflow.org/tutorials/generative/cyclegan>, <https://www.tensorflow.org/tutorials/generative/pix2pix>)