3A FISE - Spécialité Informatique - Majeure ISIA

Deep Avancé : Generative Models

Part III: deep latent generative models

julien.Rabin (at) ensicaen.fr

Version : 1st December 2025

In previous episodes ...

Machine Learning and **Deep Neural Network (DNN)**

- In other course, **Image Classification:** AlexNet [Krizhevsky *et al.*'12], VGG-19 [Simonyan & Zisserman'14]**,** ResNet [He'15], Inception Network [Szegedy'16], ...

- In this course so far, applications to other computer vision problems:

  - **Filtering** (denoising, artifact removing), **super-resolution**, inpainting: deep image prior [Ulyanov *et al.*'17]
  - **Texture synthesis** and **style transfer:** [Gatys *et al.*'15] [Ulyanov *et al.*'16]

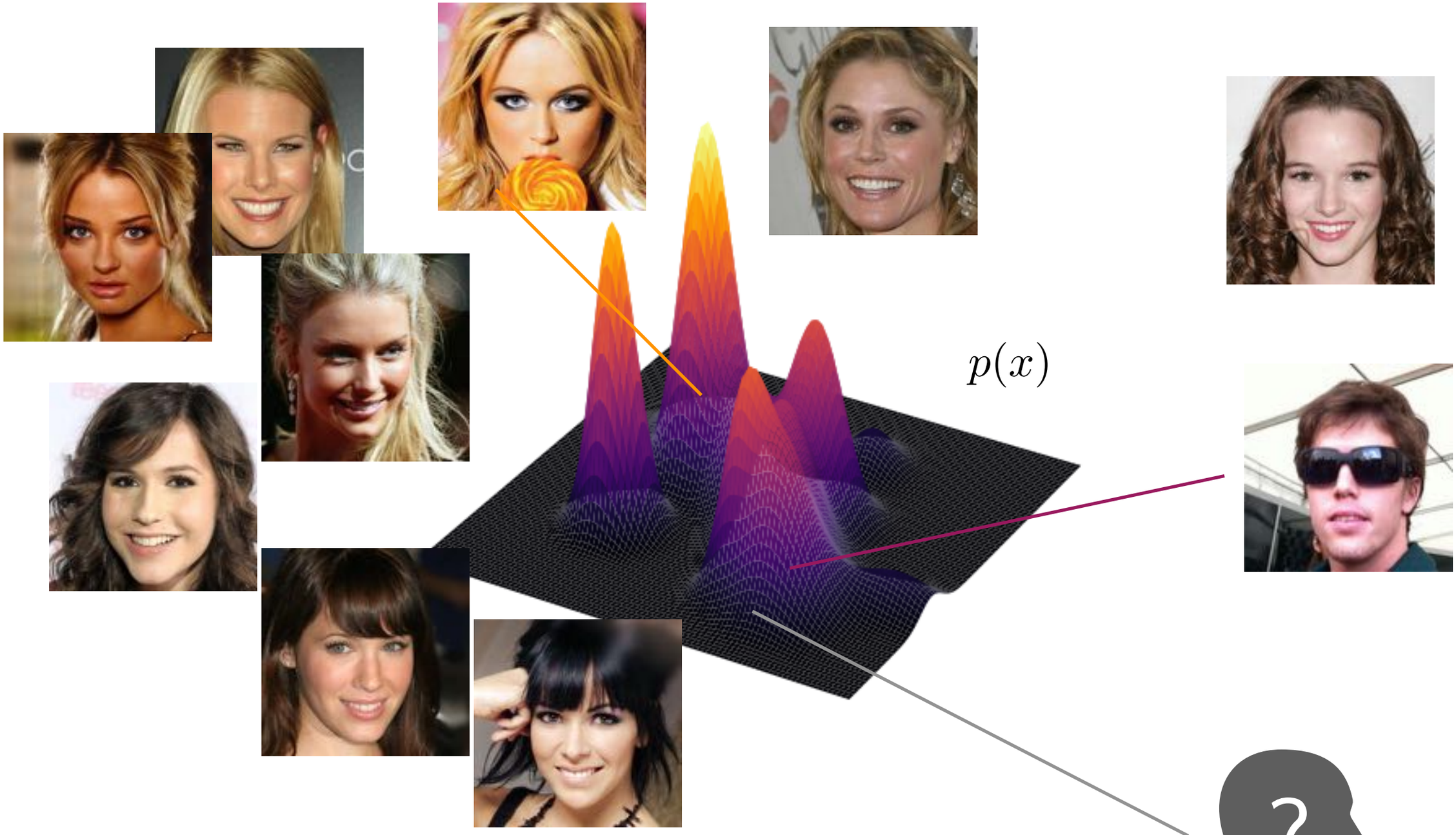
- Today : training DNN for **Image generation** with various models:
  GANs [Goodfellow'14], VAE [Kingma & Welling, '14], Transformers [Vaswani'17] with VQ-GAN [Esser'21], Diffusion Model [Ho.'20] and Latent Diffusion [Rombach'22] ...

What is a Generative Model ?

(Additional content in syllabus)

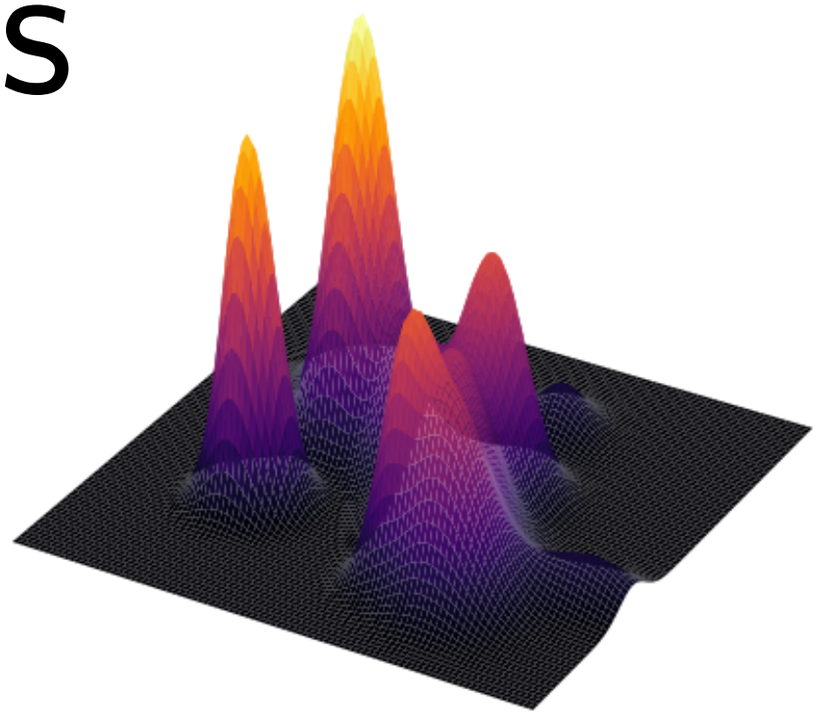
Goal of generative modelling

- 1) **Model** the distribution of data of interest (here registered images of celebrity portraits)



- 2) **Sample** from this model

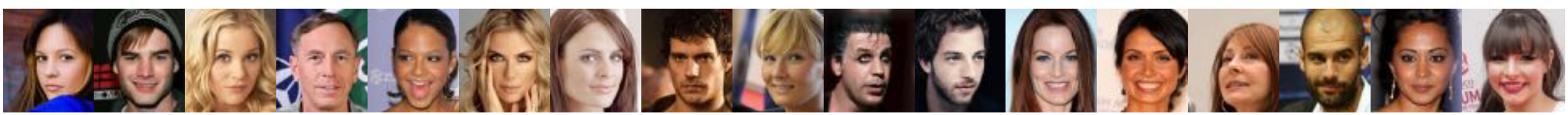
Challenges



- **High dimensional data:** $x \in R^d$ with $d = 10^6$
- **Unknown distribution:** $p(x) = ?$
- **Sampling is difficult,** even when p is known !

- **Only access to (a few) representative samples:**
  here samples x_i from celebA-HQ dataset

$$\mathcal{D} = \{x_1, ...x_n\} = \{x_i\}_{i \in [N]}$$



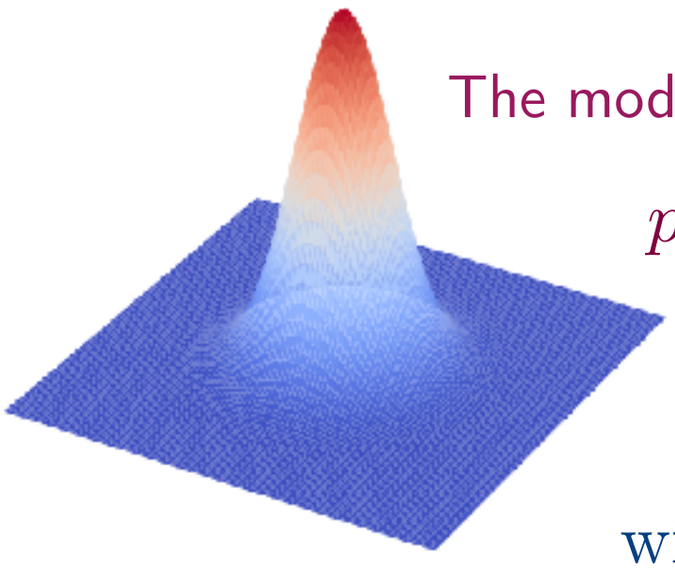
What is a latent generative model ?

- **Definitions:** denoting $p(y)$ or p_y the probability distribution of a random variable y
  - A **parametric model** is a parametric family of the distribution of datapoints x

    evidence (**marginal likelihood** of a datapoint x): $p_\theta(x) = p(x, \theta)$
  - A **latent variable model** is a joint model with a latent variable z (hidden state)

    joint likelihood*: $p_\theta(x, z) = p_\theta(z) \times p_\theta(x|z) = p_z(z, \theta)p_x(x|z, \theta)$
  - A **latent generative model** is a parametric function sampling x from the model

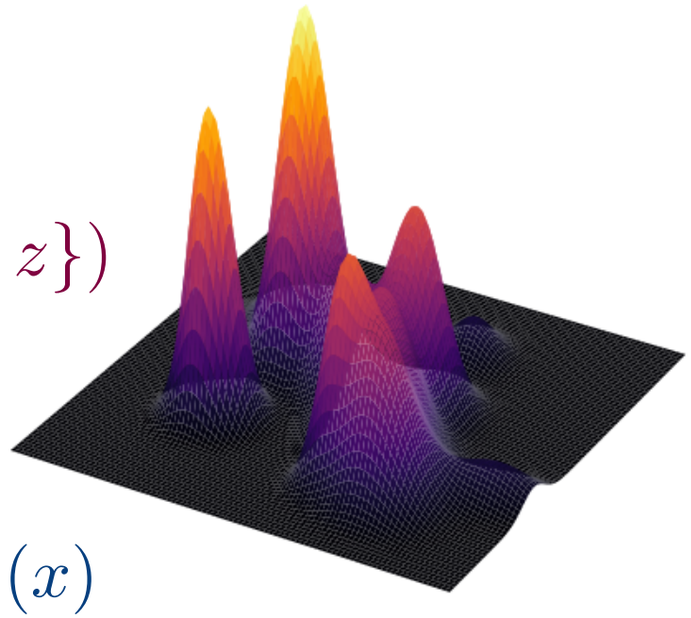
generated distribution: $p_x = G_\theta \sharp p_z$



The model evidence of a sample x is

$$p_\theta(x) = G_\theta \sharp p_z(x) = p_z(\{z, G_\theta^{-1}(x) = z\})$$

with $z \sim p_\theta(z)$ sample $\hat{x} = G_\theta(z) \sim p_\theta(x)$



latent prior: $p_\theta(z)$

$p_\theta(x)$

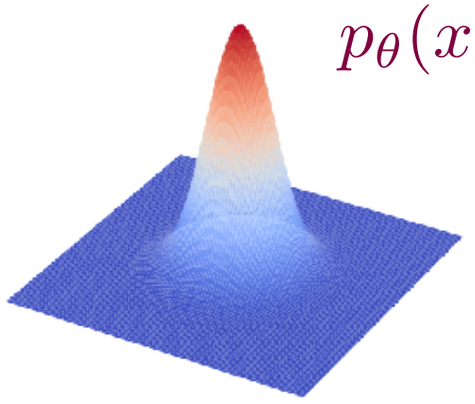
* with some conventional abuse of notations

Latent generative models

- **Miscellaneous remarks:**
  - The **latent variable** $z \in \mathbb{R}^\ell$ lives in small dimensions $\ell \ll d$
  - it can be seen as expression of hidden states or factors (e.g. age, gender, color ...)
  - in the general case of **unsupervised** learning, latent variables corresponds to **unknown** factors
  - with **supervised** training, it is possible to train conditional generative models
  - **deep** latent generative models use a neural network to parametrize G_θ
  - once trained, **generating random samples from $p_\theta(x)$ is simple** ...
  - ... but computing the likelihood $p_\theta(x)$ is generally **intractable**

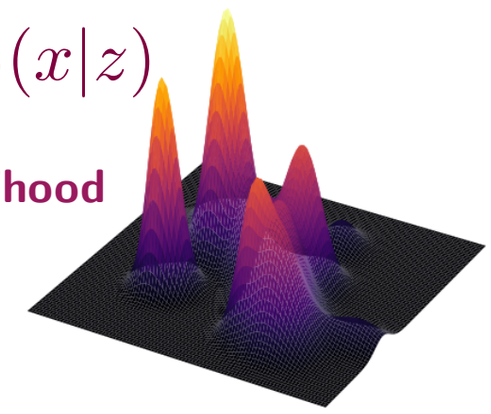
$$p_\theta(x) = G_\theta \sharp p_\theta(z) = \int_z p_\theta(x|z)p_\theta(z)dz = \mathbb{E}_{z\sim p_\theta(z)} p_\theta(x|z)$$

conditional likelihood



$p_\theta(z)$

with $z \sim p_\theta(z)$ sample $\hat{x} = G_\theta(z) \sim p_\theta(x)$



$p_\theta(x)$

Multi-variate gaussian example

- **Step 1 : Fitting parameters θ**
  - model **selection**: P_θ with $\theta = (\mu, \Sigma)$

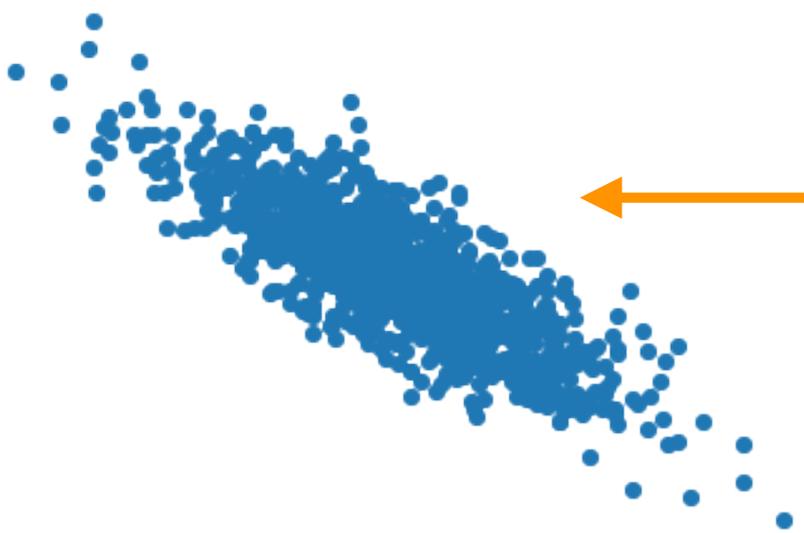
$$x \in \mathbb{R}^d \sim p_\theta = \mathcal{N}(\mu, \Sigma) : \qquad p_\theta(x) = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)}$$

  - model **fitting:** optimize the model parameters by maximising model evidence (MLE = maximum likelihood estimators of μ and Σ)

$$\text{(MLE)} \qquad \max_\theta \{\mathcal{L}_\mathcal{D}(\theta) := p_\theta(\mathcal{D}) := p_\theta(x_1, ...x_N)\} = \max_\theta p_\theta(x_1) \times p_\theta(x_2) \times ... \times p_\theta(x_N)$$

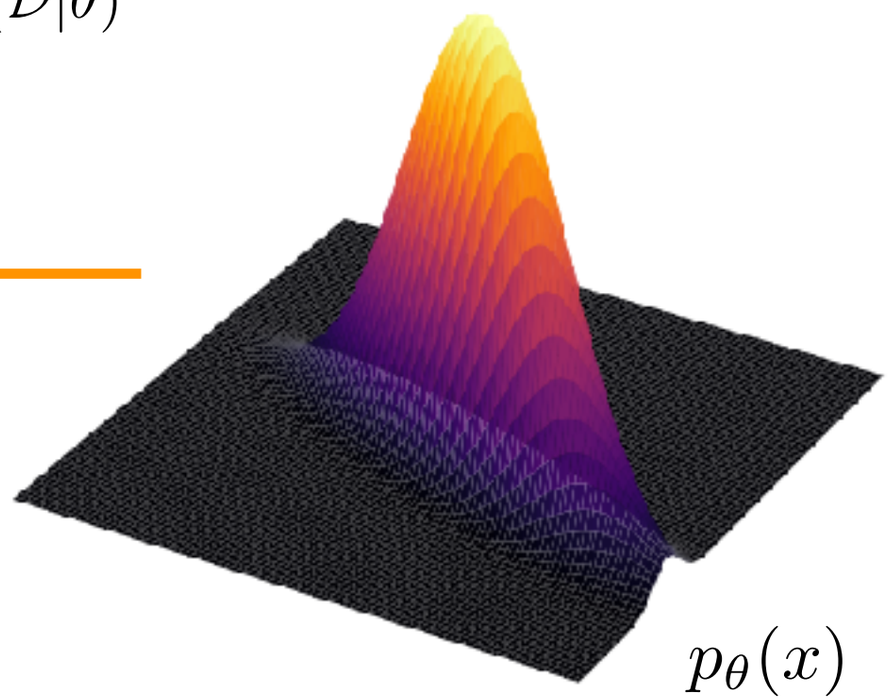
  - *remark* : variant with bayesian inference using some prior $p(\theta)$

$$\text{(MAP)} \qquad \max_\theta p(\theta) p(\mathcal{D}|\theta)$$



$$\mathcal{D} = \{x_1, ...x_n\} = \{x_i\}_{i \in [N]}$$

fit θ



$$p_\theta(x)$$

Multi-variate gaussian example

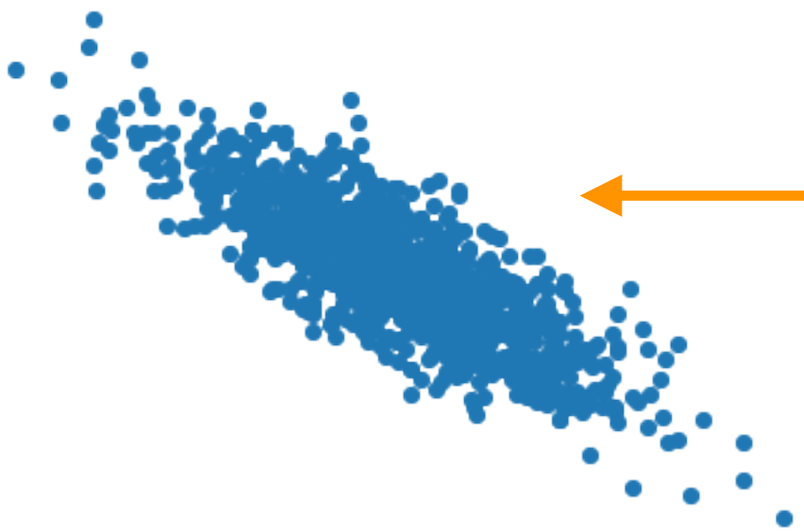
- **Step 1 :** model fitting of P_θ with $\theta = (\mu, \Sigma)$ by maximizing evidence

$$x \in \mathbb{R}^d \sim p_\theta = \mathcal{N}(\mu, \Sigma) : \qquad p_\theta(x) = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)}$$

$$\text{(NNL-minimization)} \qquad \min_\theta \left\{ -\frac{1}{N} \log \mathcal{L}_\mathcal{D}(\theta) = -\frac{1}{N} \log p_\theta(\mathcal{D}) = -\frac{1}{N} \sum_{i=1}^{N} \log p_\theta(x_i) \right\}$$

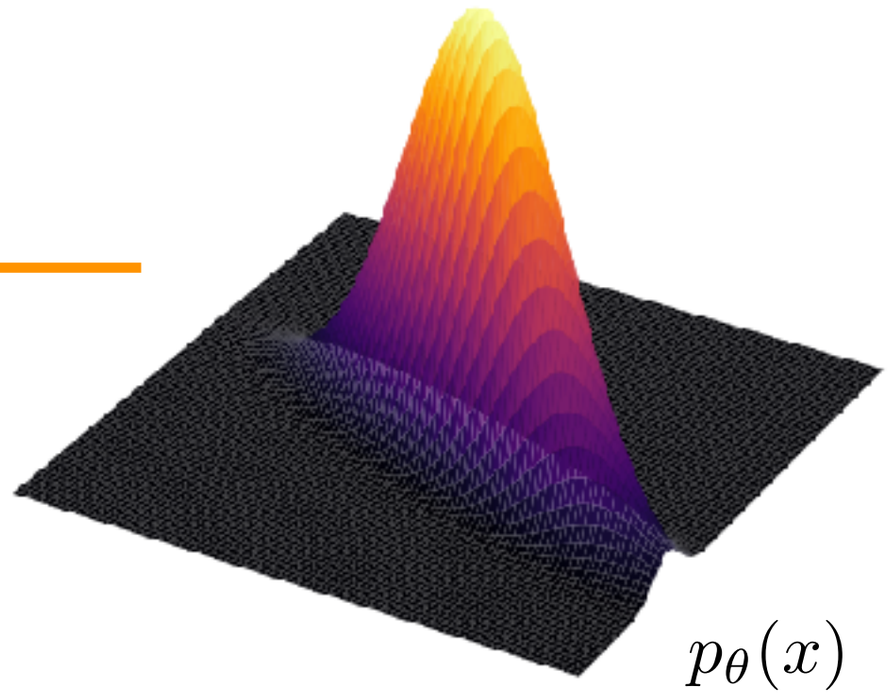
- **Result :** (see proof in syllabus)

$$\hat{\mu} = \bar{X} = \tfrac{1}{N} X \mathbf{1} = \tfrac{1}{N} \sum_{i=1}^{N} x_i \qquad \hat{\Sigma} = \tfrac{1}{N} \tilde{X} \tilde{X}^\top \text{ with } \tilde{X} = X - \bar{X} \mathbf{1}^\top$$



$\mathcal{D} = \{x_1, ...x_n\} = \{x_i\}_{i \in [N]}$

fit θ



$p_\theta(x)$

Multi-variate gaussian example

- **Step 2 : Sampling** $x \sim \mathcal{N}(\mu, \Sigma)$ using change of variable (« reparametrization trick »)

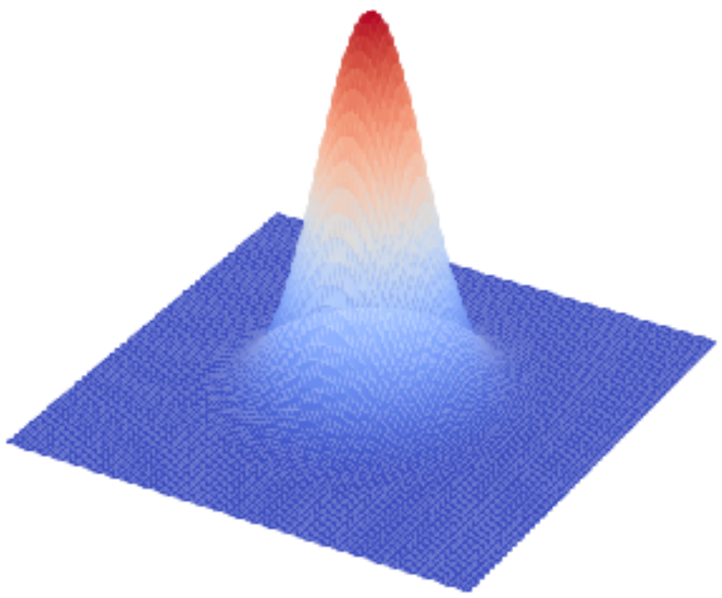
$$x = G_\theta(\varepsilon) = \hat{\mu} + R\,z \text{ where } z \sim p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Note : prior is unparameterised !

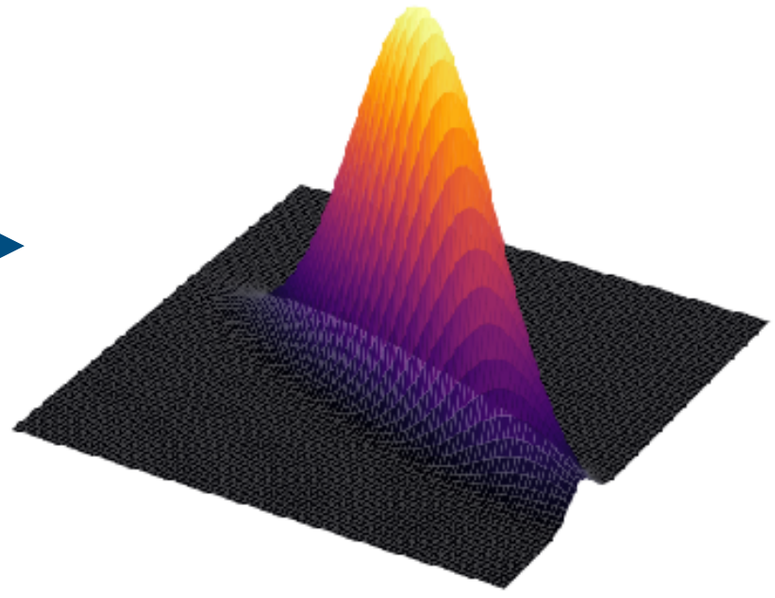
- **Computing R :** R is a solution of (see proof in syllabus)

$$R\,R^\top = \hat{\Sigma} = V S V^\top$$

$$R = V\sqrt{S} = [v_1, ..., v_d] \times \text{diag}(\sqrt{s_1}, ...\sqrt{s_d}) = [\sqrt{s_1}v_1, ..., \sqrt{s_d}v_d] = (\sqrt{s}^\top \mathbf{1}) \odot V$$



sample x from G_θ



Latent space $\mathcal{Z} \in \mathbb{R}^\ell$

$p_\theta(z) = p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

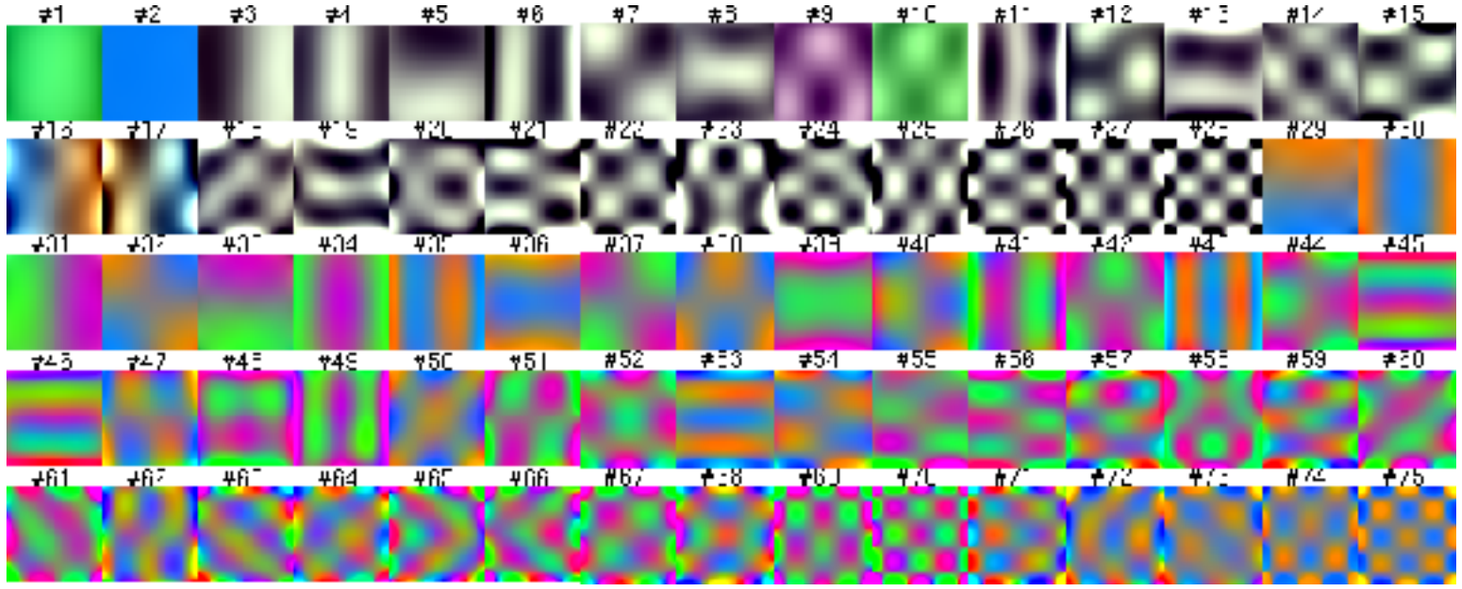
Data/Feature space $\mathcal{X} \in \mathbb{R}^d$

$p_\theta(x)$

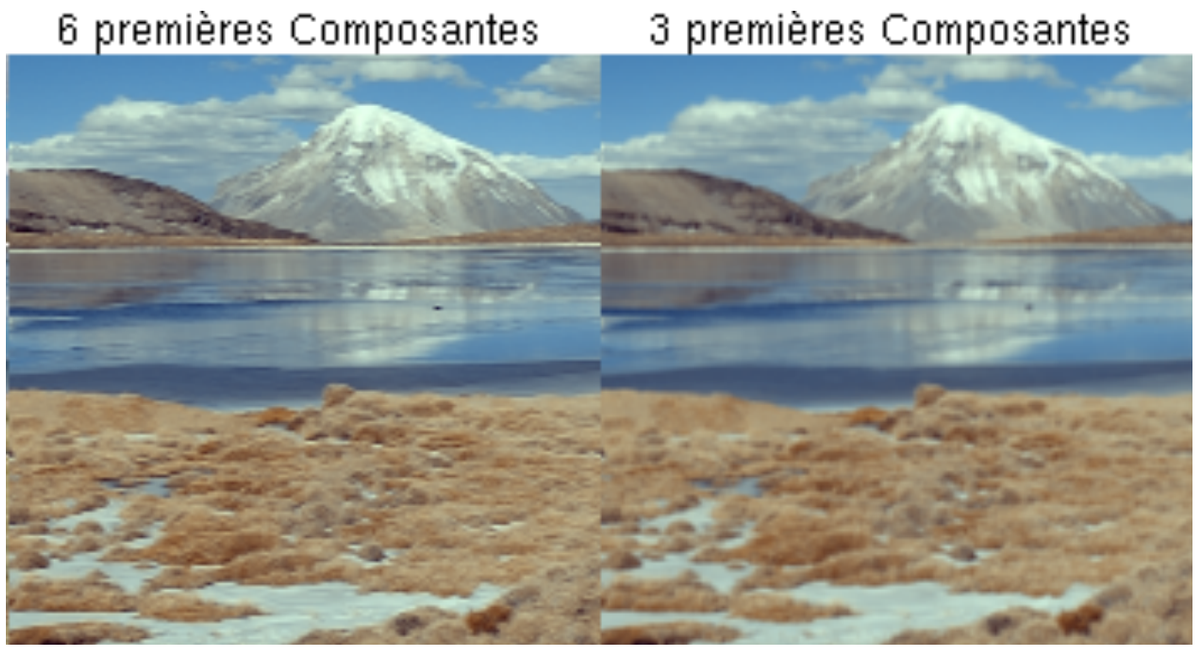
Example : back to NL-PCA on patches

- **Dataset $\mathscr{D}$:** a simple image

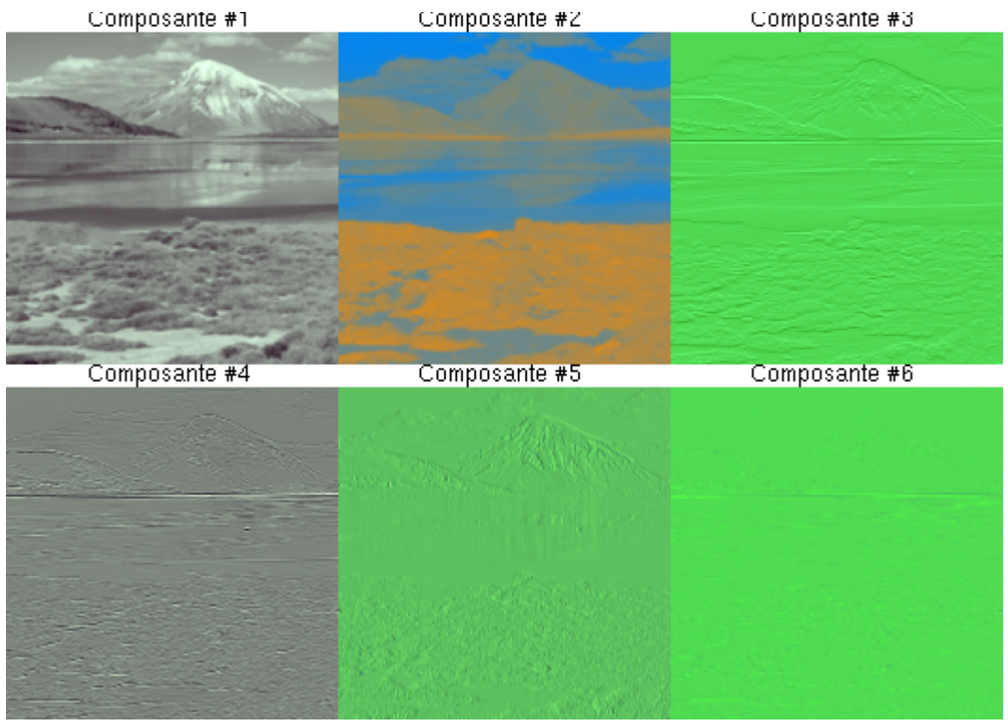
- **Eigen-patches : 75 dimensional** 5x5 RGB patches



- **Reconstruction:**



- **Projection:**



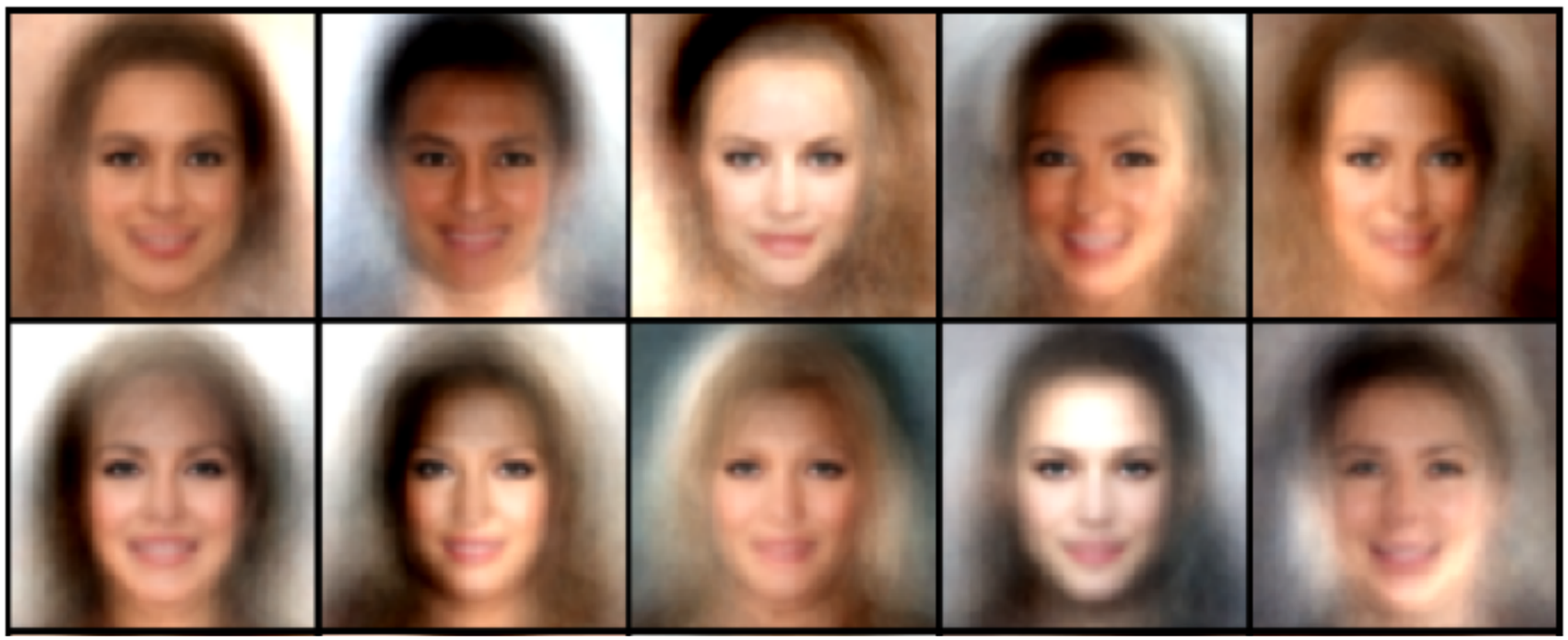
Demo : Gaussian model of portraits

- **Dataset $\mathscr{D}$:** excerpt from celebA-HQ (N=1k images)
- Method related to **EigenFACE [Sirovich & Kirby '87]** for face classification
- **Inference :** PCA or power-iterations (data space) d = $3 \times 96^2 = 27648$,
- **Sampling:** (latent space) l=100 (selecting largest eigenvalues)
- **Demo:** Pytorch+Numpy notebook *celeba_eigenface.ipynb*

Demo : Gaussian model of portraits

- **Sampling:** gaussian vectors using 100 eigenvalues

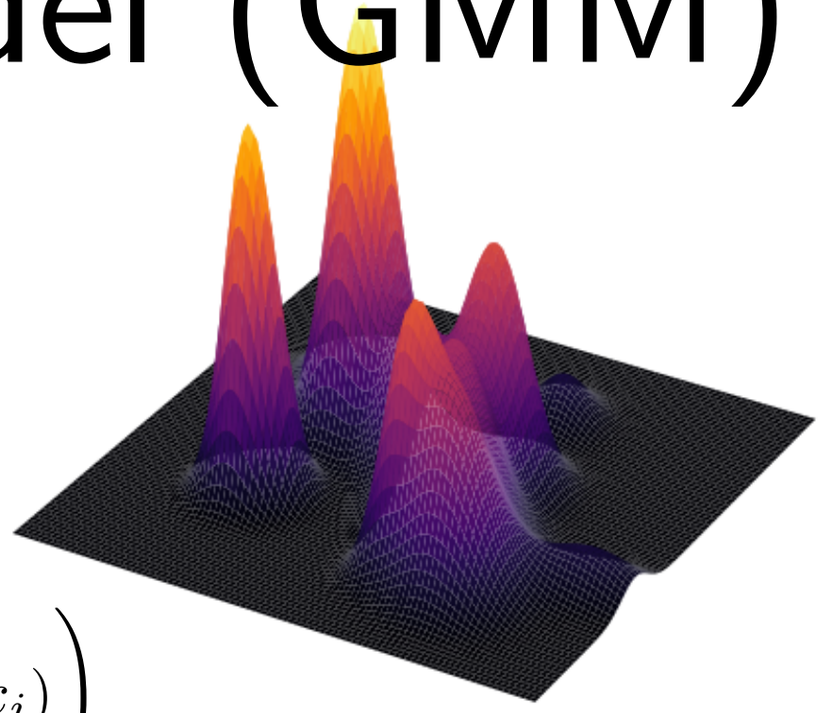
- **Sampling:** gaussian vectors using 20 eigenvalues



Limitations of the Gaussian model

- **Dimensionality:** for d-dimensional features, the gaussian model has d+d^2 parameters (for the mean and covariance matrix)
  - requires **dimensionality reduction** : eg multi-scale decomposition, PCA, AE ...
- **Euclidean Metric** is not adequate to compare full images
  - requires **image-based representation** learnt from a dataset
- **Variability/Diversity:** using only one mode is simplistic (considering for instance the mean which may not be representative)
  - requires **complex model** (multi-mode generation) : eg GMM, supervised learning (using labels : conditional generation)
- **Photo-realism:** images are too noisy or blurry, lacks details (texture) and sharpness (contour)
  - requires **image-based metric**

Gaussian Mixture Model (GMM)



- **Definition** $$p_\theta(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k)(x)$$

- **Fitting parameters θ :** (see syllabus)
  - using directly MLE (maximum likelihood estimators)

$$\log p_\theta(\mathcal{D}) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k)(x_i) \right)$$

  - using EM (Expectation Maximisation) with **Evidence Lower Bound (ELBO)***

$$\log p_\theta(x) \geq L(\theta, x) := \mathbb{E}_{z \sim q} \left[\log \left(\frac{p_\theta(x, z)}{q(z)} \right) \right]$$

- **Sampling x :** in two steps
  - sample $n \sim p(n) = \sum_{k=1}^{K} \pi_k \delta(n - k)$
  - sample $x \sim \mathcal{N}(\mu_n, \Sigma_n)$ using change of variable

* (proof later)

Demo : GMM

- **Demo:** Pytorch(+Scikit-learn) notebook *celeba_eigenface.ipynb*
- **Sampling:** GMM with K=36, N=1000, and 5 eigenvalues

Demo : GMM

- **Demo:** Pytorch(+Scikit-learn) notebook *celeba_eigenface.ipynb (**Lab session #6**)*
- **Comparison:** fake vs real samples using **Euclidean Nearest Neighbour** (NN)

Fake samples *Real samples*

Remark : shows again that euclidean metric is not **appropriate** for image comparison

Unfortunately, **LPIPS** give similar results ...

Deep Latent Models

- **Definition:** the parametric distribution is **implicitly** defined from a generative network

$$p_\theta(x) = G_\theta \sharp p_\theta(z) \qquad \text{with } z \sim p_\theta(z) \text{ sample } \hat{x} = G_\theta(z) \sim p_\theta(x)$$

- **Architectures :**
  - feed-forward network, recursive network ...
  - optimization using stochastic gradient descent

- **Fitting parameters θ :** (see syllabus)
  - different strategies and criteria
  - optimization using stochastic gradient descent

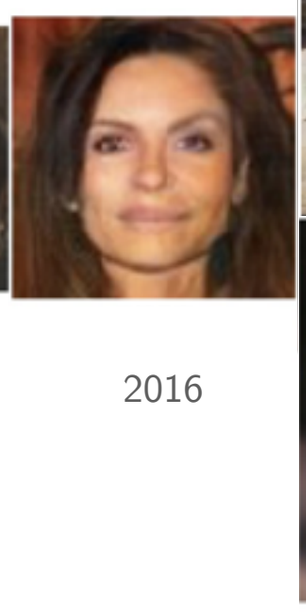
- **Sampling x :** in two steps
  - sample $z \sim p(z) = \mathcal{N}(0, \mathbf{Id})$ from a latent domain $\mathcal{X} \in \mathbb{R}^\ell$ (either $\ell \ll d$ or $\ell = d$)
  - compute $x = G_\theta(z)$

Deep Latent Models

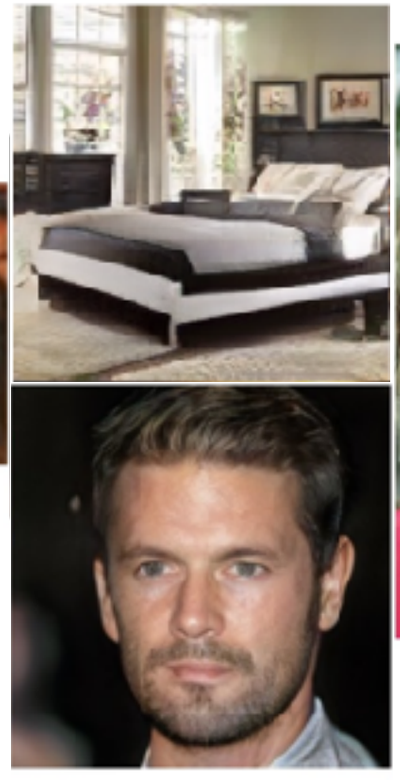
- **Paradigm** over the last decade: " *bigger is better* "

2014

2015



2016



2017

2018

2019

2022

- **Cost** $>$ 1 M$
- **Energy** $>$ 40 MWh / T.CO_2
- **Risks** (cf ***European AI Act***)

$\mathbf{P} = \#$ **parameters**
$\mathbf{D} = \#$ **data**
$\mathbf{J} = \#$ **GPU days**

$\mathbf{P = 40\ M}$
$\mathbf{D = 0.03\ M}$
$\mathbf{J = 4}$

$\mathbf{P = 60\ M}$
$\mathbf{D = 0.08\ M}$
$\mathbf{J = 64}$

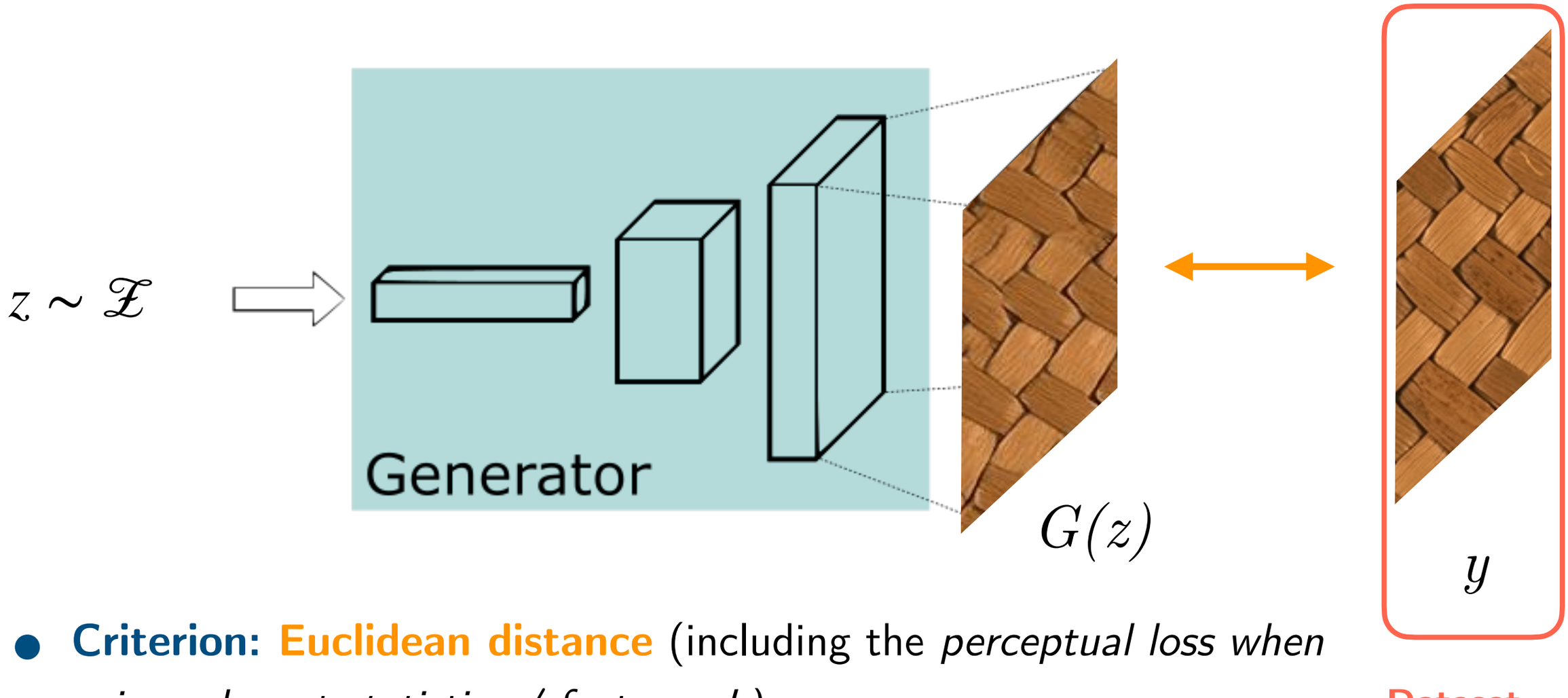
$\mathbf{P = 160\ M}$
$\mathbf{D = 300\ M}$
$\mathbf{J = 1000}$

$\mathbf{P = 1500\ M}$
$\mathbf{D = 400\ à\ 2000\ M}$
$\mathbf{J = 300\ à\ 6000}$

How to train Generative Models ?

How to train a generator ?

- **Recall:** with previous methods (Deep Image Prior, Texture Networks, SinGan ...), a generative network G was trained **on a single image**

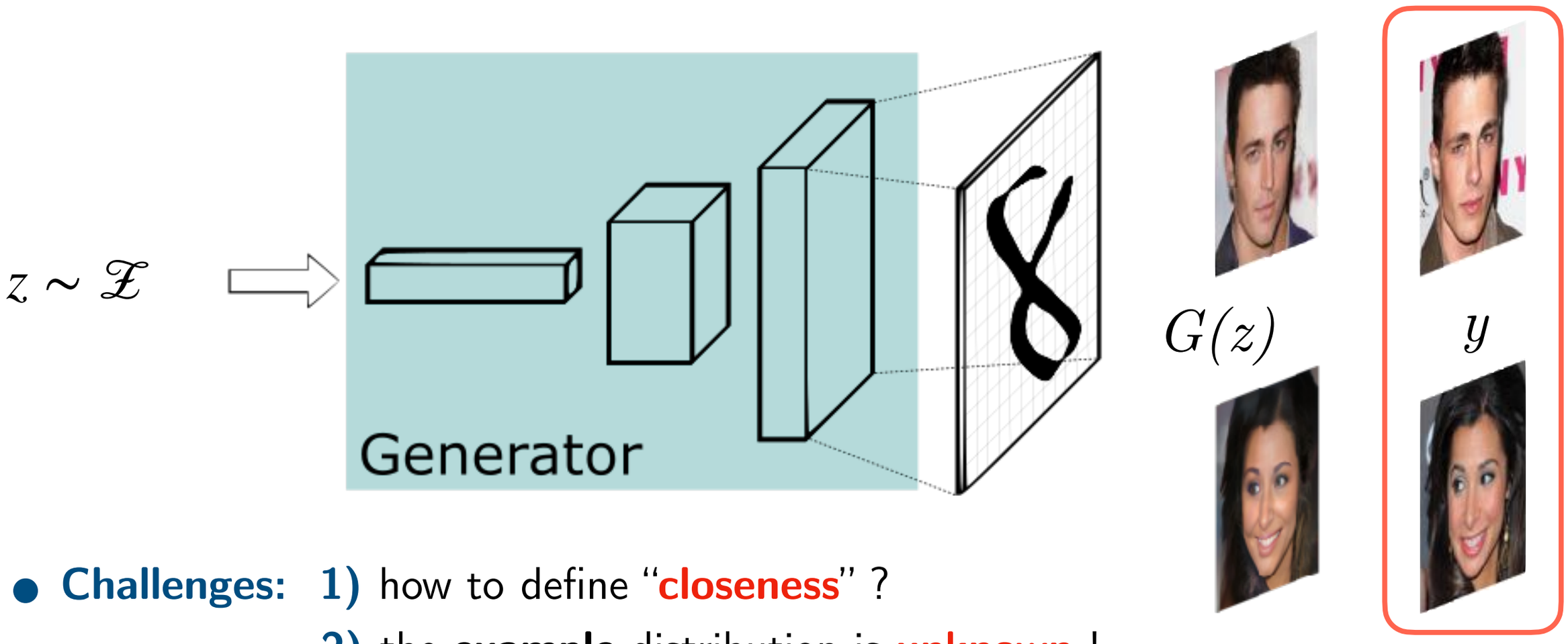


- **Criterion: Euclidean distance** (including the *perceptual loss when using relevant statistics / features ϕ)*

$$\|\phi(G(z)) - \phi(y)\|^2$$

How to train a generator ?

- **Objective:** generate images $G(z)$ using a neural network G (from random latent codes z) that **are close** to empirical **samples from a distribution**

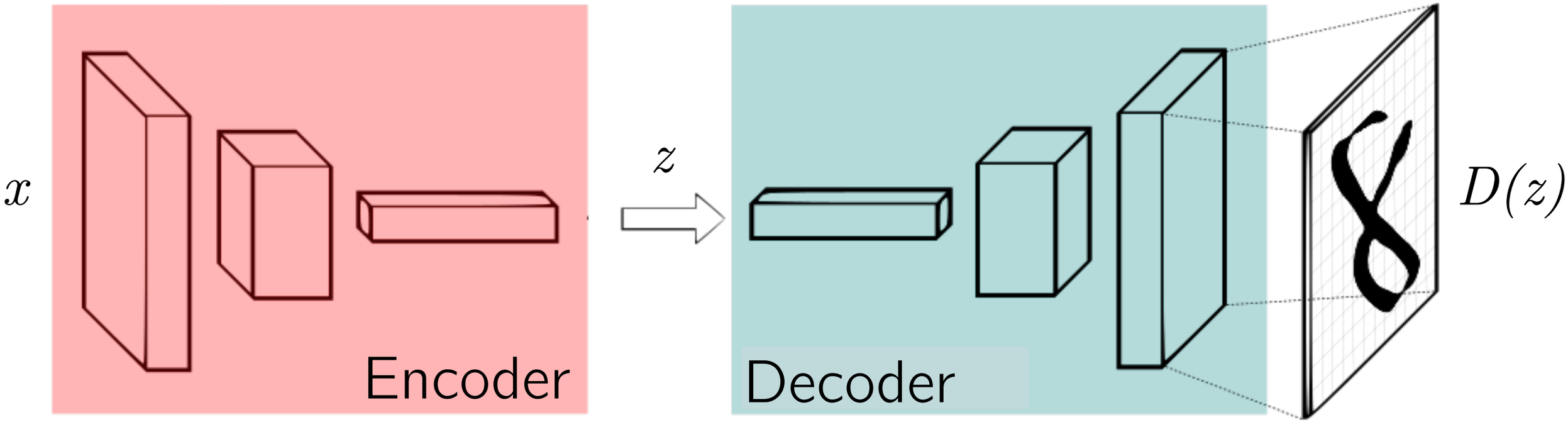


- **Challenges:** **1)** how to define "**closeness**" ?
  2) the **example** distribution is **unknown** !

- **Note:** we cannot use the *perceptual loss* anymore ...

How about Auto-Encoders ?

- **Principle:** Encode an input image x into a **latent code** $z{=}E(x)$ and decode with the Decoder D(z)



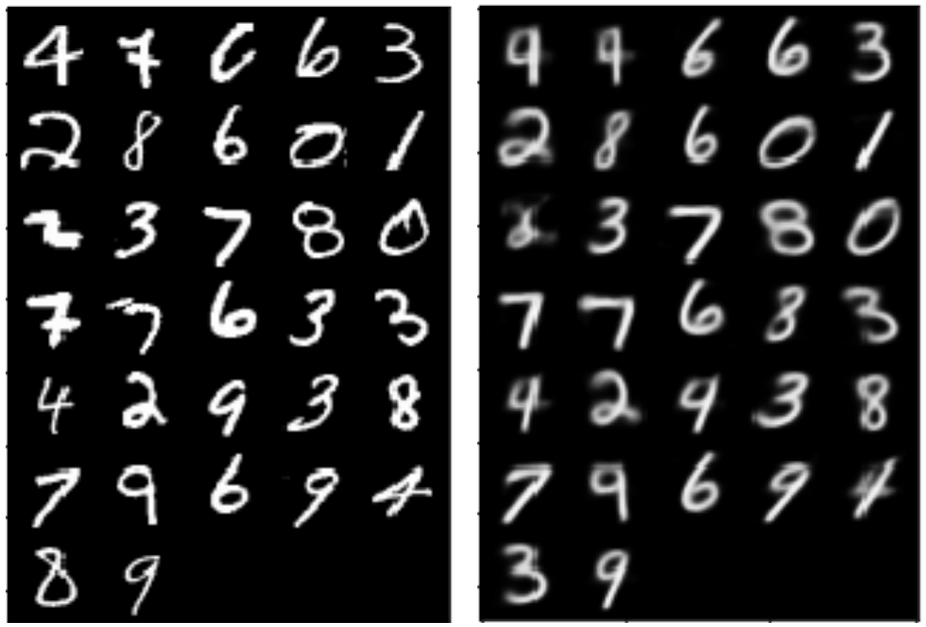
- **Optimization problem:** $$\min_{D,E} \sum_{x_i \in \mathcal{D}ata} \|D(E(x_i)) - x_i\|_2^2$$

- **Limitation :** decoder D **does not generalize** by training on data samples *i.e.* **sampling z outside** $\{E(x_i)\}$ gives non realistic samples D(z)

Auto-Encoders on MNIST Digits

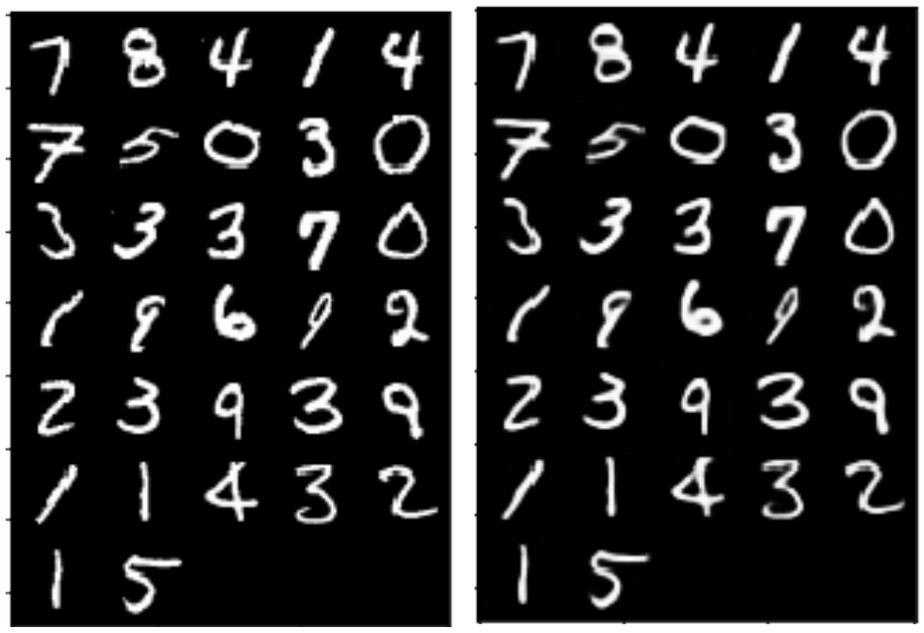
- **Illustrations on MNIST:** with MLP & ConvNet
  - MNIST_AutoEncoder.ipynb & MNIST_AutoEncoder_overfitting.ipynb

test data *reconstruction*



small latent space:
imperfect (blurry) reconstruction

test data *reconstruction*

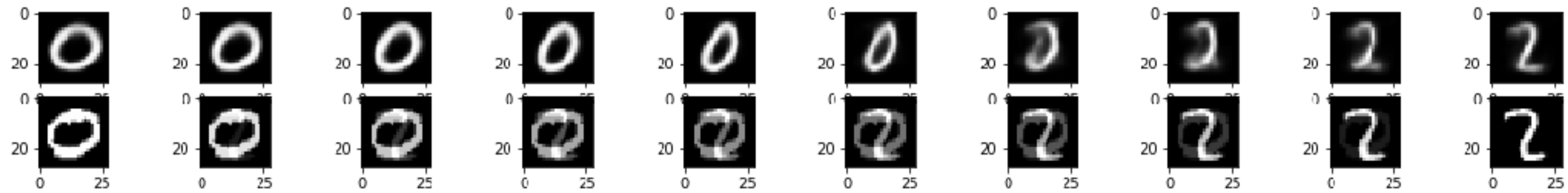


large latent space:
perfect reconstruction (even on test data)

Interpolation in latent space

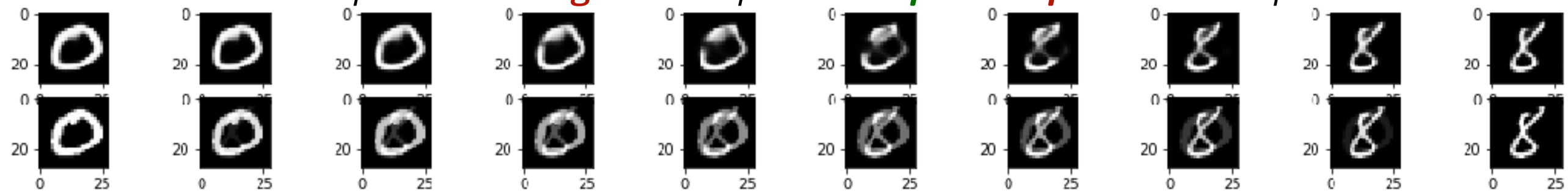
- **Illustrations on MNIST:** with MLP & ConvNet
  - MNIST_AutoEncoder.ipynb & MNIST_AutoEncoder_overfitting.ipynb

*Linear interpolation in **small** latent space : **blurry** but **plausible** interpolation*



Linear interpolation in pixel space

*Linear interpolation in **large** latent space : **sharp** but **improbable** interpolation*



Linear interpolation in pixel space

Sampling Latent codes of digits

- **Illustrations on MNIST:** with MLP & ConvNet
  - MNIST_AutoEncoder.ipynb & MNIST_AutoEncoder_overfitting.ipynb

random sampling with multi-variate gaussian sampling (see eigenface)

small latent space:

plausible but **blurry** samples (w/ label 5)

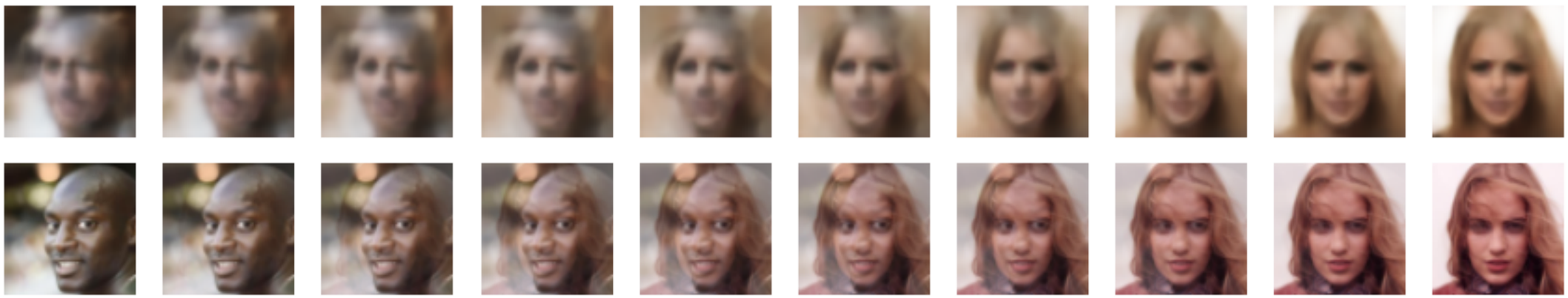
large latent space:

improbable but **sharp** samples (w/ label 3)

Auto-Encoders with portraits

- **Illustrations on CelebA-HQ:** with U-Net & ConvNet & MLP
  - CELEBA_AutoEncoder.ipynb

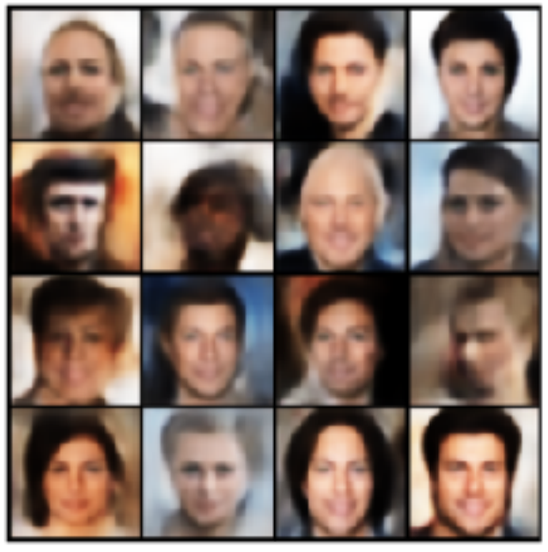
*Linear interpolation in **small** latent space : **blurry** but **plausible** interpolation*



Linear interpolation in pixel space

random sampling with multi-variate gaussian sampling

More realistic than in RGB space (eigenface) but still not photo-realistic



*sampling from 'male' distribution (**entangled** characteristics)*

☞ comparaison with VAE

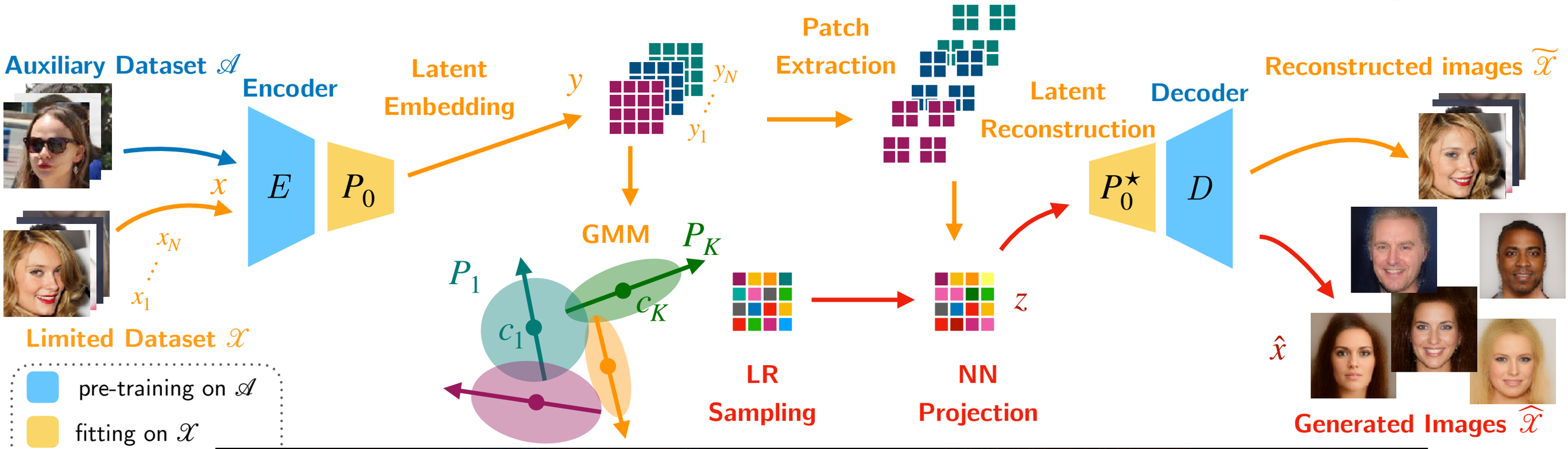
How about Auto-Encoders ?

- **Illustrations on MNIST & CelebA:**
  - MNIST_AutoEncoder.ipynb & MNIST_AutoEncoder_overfitting.ipynb
  - CelebaHQ_AutoEncoder.ipynb

- **Observations:**
  - **does not generalised** with high dimensional latent-space (**overfitting**, *ie* **discrepancy with test data**)
  - **tends to generalise** with smaller latent space (but **tradeoff** with quality of reconstruction)
  - latent space is **self-organised** but w/o explicit control (**entangled characteristics**)

Combining AE & GMM

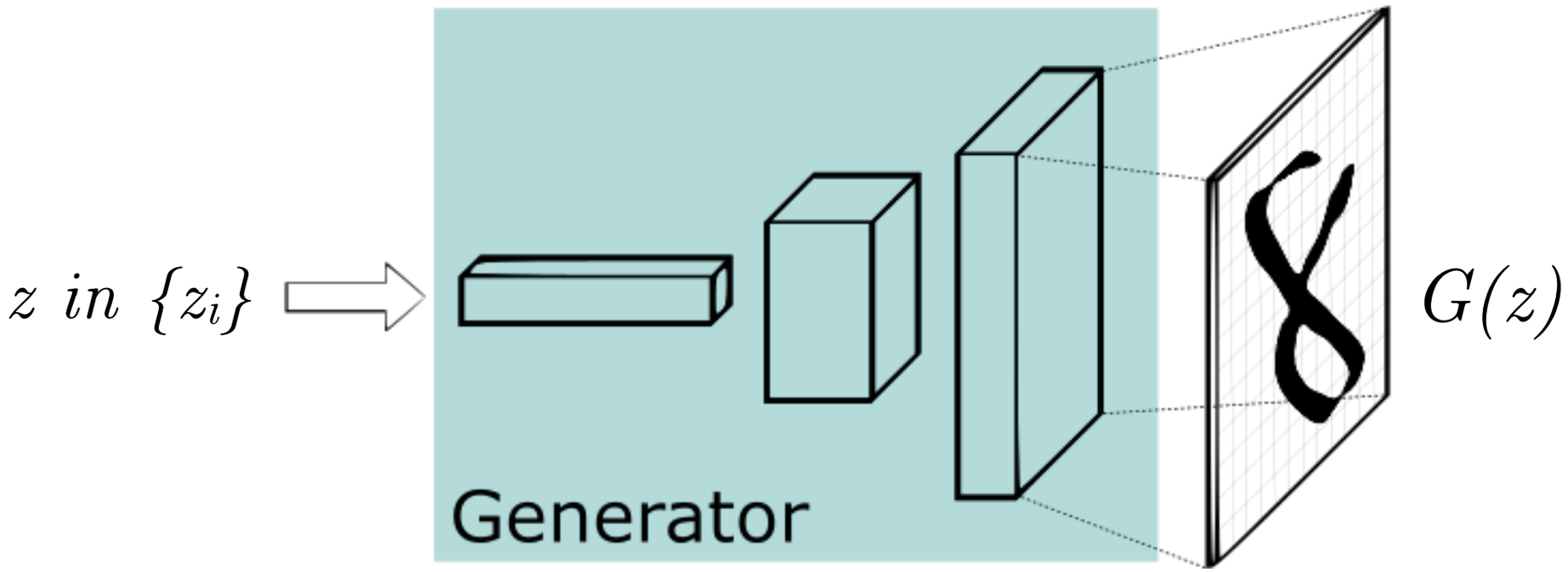
- **Examples from:** Latent-GMM **[Samuth'24]**



How about optimising latent codes ?

Generative Latent Optimization (GLO) **[Bojanowski'18]**

- **Principle:** P(z) is discrete and supported by a collection of latent codes $\{z_i\}$ that are optimised **simultaneously** with the generator G



- **Optimization problem:**

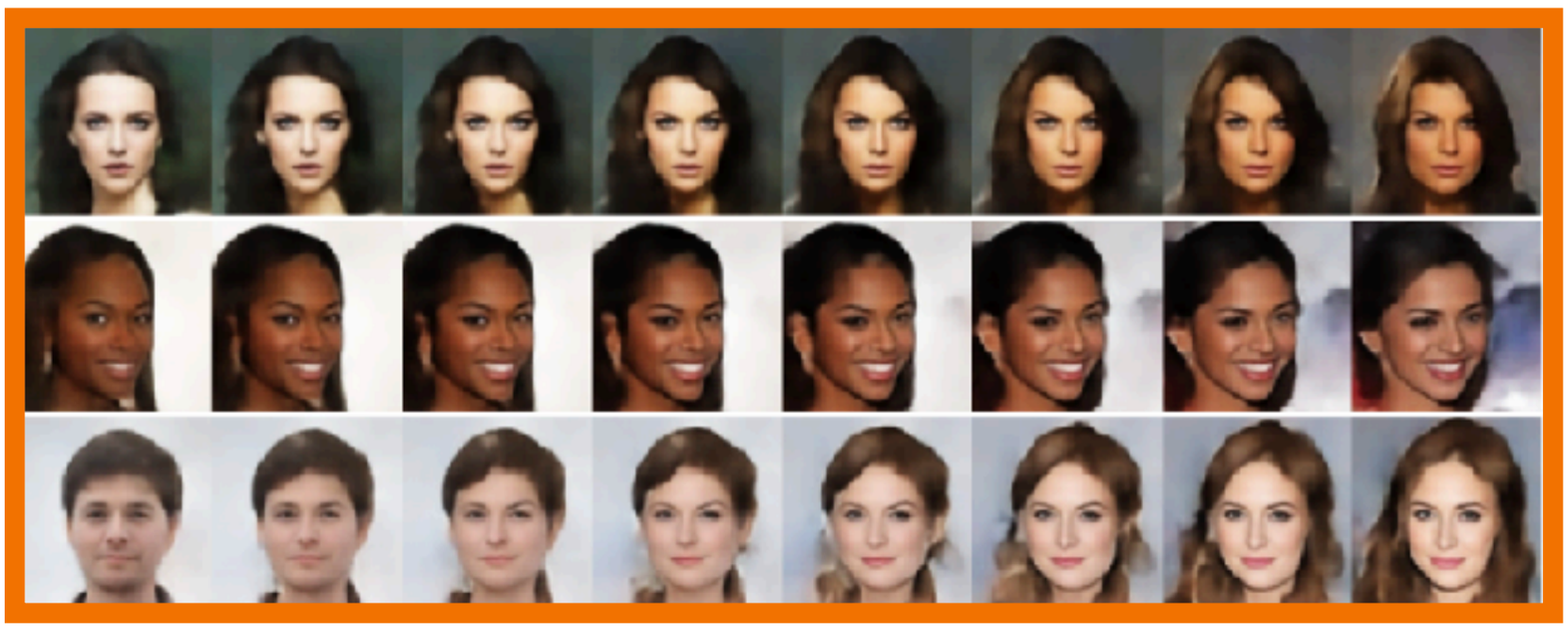
Laplacian Pyramid Δ

$$\min_{G,\{z_i\}} \sum_{(z_i,x_i),i=1..|\mathcal{D}|} \|G(z_i) - x_i\|_2^2 + \lambda \sum_{\ell} \|\Delta_\ell(G(z_i) - x_i)\|_1$$

- **Sampling:** latent codes are uniformly sampled from the unit-sphere

Generative Latent Optimization (GLO) [Bojanowski'18]

- **Results: interpolations** and **random samples** (CelebA - 128)



overfitting with small datasets [Webster'19]

Sliced-Wasserstein AE [Kolouri'18]

- **Variant using optimal transport:**

$$\min_{G,E} \sum_{x_i \in \mathcal{D}} \|G(E(x_i)) - x_i\|_2^2 + \lambda \mathrm{SW}_2^2(\mathcal{Z}, \mu)$$

where SW is the Sliced-Wasserstein Distance

and Z is the distribution of the encoded training data

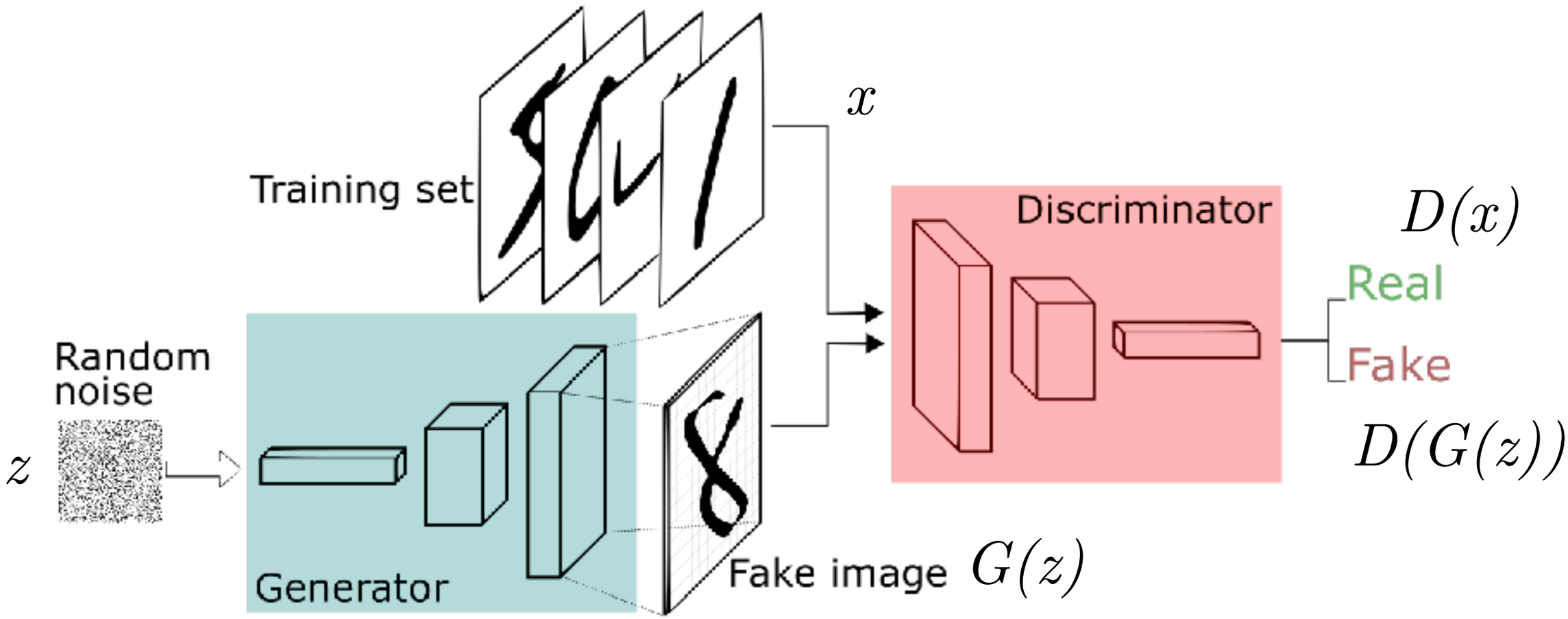
- **Interpolation** between **training samples** on CelebA

- Does not generalize well **(random samples)**

Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GAN)

- **Principle [Goodfellow'14]:** Adversarial optimization strategy between a generator G and a **Discriminator** D detecting real from fake images



- **Optimization:** $\max_D \min_G \mathbb{E}_{z\sim p_Z, x\sim p_{\mathcal{D}}} \log(D(x)) + \log(1 - D(G(z)))$

- **Synthesis:** $G(z)$ computed from a sample z from the training pdf p_Z

(some) improvements

- DC-GAN [Radford et al.'15]
- WGAN (Wasserstein penalty) [Arjovsky et al.'17], LS-GAN, Chi-2 GAN ...
- Gradient penalty : WGAN-GP [Gulrajani et al.'17]
- Progressive Growing of GANs [Karras et al.'18]
- BigGAN [Brock'19] (Conditional generation)
- Style-GAN [Karras et al.'19], Style-GAN v2, v3 ...
- ...

GAN training in details

Outline :

- **Supervised training** of an image classifier :
  some recalls about logistic regression

- **Unsupervised training** of the Discriminator using generated and real images

- **Unsupervised training** of the generative network using the discriminator

- Variant of GAN training (other metric, regularisation)

Recall: binary variable

- For a binary random variable Y, a prior **Bernoulli law** is parametrised by a parameter p

$$\mathbb{P}(y=1) = p \qquad\qquad \mathbb{P}(y=0) = 1-p$$

- Given this prior, the **likelihood** of Y=y writes

$$\mathcal{L}_\theta(y) = \mathcal{L}_p(y) = \mathbb{P}(Y=y) = \begin{cases} p & \text{if } y=1 \\ 1-p & \text{if } y=0 \end{cases} = p^y\,(1-p)^{1-y}$$

- Taking the natural logarithm, the **negative log-likelihood** writes

$$\ell_p(y) = -\log \mathcal{L}_p(y) = -y \log p - (1-y)\log(1-p)$$

- **Entropy** of the Bernoulli law is the **binary entropy function**

$$H(p) = -\sum_k \mathbb{P}(Y=y_k)\log \mathbb{P}(Y=y_k) = -p\log p - (1-p)\log(1-p)$$

Recall: cross entropy

- Entropy of the Bernoulli law is the **binary entropy function**

$$H(p) = -\sum_k \mathbb{P}(Y = y_k) \log \mathbb{P}(Y = y_k) = -p \log p - (1-p) \log(1-p)$$

- Cross - Entropy $\mathrm{H}(p, q) = -\sum_x p(x) \log q(x)$

- **Binary Cross Entropy (BCE)**: with Bernoulli distributions

$$\mathrm{H}(p, q) = -p \log q - (1-p) \log (1-q)$$

- Kullback-Leibler Divergence a.k.a the **relative entropy**

$$\mathrm{KL}(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = H(p, q) - H(p)$$

Recall on MLE

- **maximizing likelihood** is equivalent to minimising negative log-likelihood (NLL)

- the negative log-likelihood for an observation y given a model

$$\ell_p(y) = -\log \mathcal{L}_p(y) = -y \log p - (1-y)\log(1-p)$$

- For n **iid** observations $y = (y_1, ...y_n)$

$$\ell_p(y) = -\log \Pi_i \mathcal{L}_p(y_i) = -\sum_{i=1}^{n} y_i \log p + (1-y_i)\log(1-p)$$

- Example of **Maximum likelihood estimation (MLE)** of p :

$$\partial_p \ell_p(y) = 0 \Rightarrow p = \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Recall: Training a model with MLE

- Dataset $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^n$ is composed of data points x with labels $y(x)$
  data points follow the (**unknown**) distribution $p(x)$

- We want to train a model q parametrised by θ which aims at predicting the probability $q_\theta(x, y) = \mathbb{P}(Y = y \ \ x, \theta) \approx \mathbb{P}(Y = y \ \ x)$
  e.g. for classification $\hat{y}(x) = \arg\max_y q_\theta(x, y)$

- MLE : optimal parameters θ obtained by maximising likelihood

$$\max_\theta \{\mathcal{L}(\theta; \mathcal{D}) = \Pi_i \, \mathbb{P}(y_i | x_i, \theta)\}$$

- Using iid assumption and negative log-likelihood ℓ, this writes

$$\min_\theta \mathbb{E}_{x\sim p} \, \ell(\theta; x) \qquad \text{with} \qquad \ell(\theta; x) = -\log \mathbb{P}(Y = y(x) | x, \theta)$$

- Equivalent to **minimise cross entropy**

$$\min_\theta \{H(p, q_\theta) = -\mathbb{E}_{x\sim p} \, \log q_\theta(x, y(x))\}$$

Recall: Training a model with **pytorch**

- Multi-label classification (e.g. MNIST with K = 10 labels)

$$\min_{\theta} \left\{ -\mathbb{E}_{x\sim p} \log q_\theta(x, y(x)) \approx -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}_{y_i=k} \log q_\theta(x_i, y_i) \right\}$$

- Note: for a sample x_i with label y_i, q is only evaluated at $y = y_i$

- in practice, the output of the network may use a softmax layer to define a probability vector q_θ, or a 'sigmoid' function σ to normalise values in [0,1]

- ⚠ torch.nn.CrossEntropyLoss include softmax: "The *input* is expected to contain the **un-normalized** logits"

- In the binary case (K=2), the network only outputs the scalar value $q_\theta(x, y = 1)$ (resorting to $q_\theta(x, y = 0) = 1 - q_\theta(x, y = 1)$)

- ⚠ torch.nn.BCELoss vs BCEWithLogitsLoss (including sigmoid)

Training the discriminator

- For GAN the dataset is composed of datapoints that are **real** (label y=1) or **fake** (y=0) : this is a ***binary classification problem***

- The discriminator D is trained to predict the correct label

- Assuming equal repartition of real/fake, $\mathscr{D} = real \cup fake$
  Bayes' rule: $p(x) = \mathbb{P}(x|y=1)\mathbb{P}(y=1) + \mathbb{P}(x|y=0)\mathbb{P}(y=0) = \frac{1}{2}p_{\text{real}}(x) + \frac{1}{2}p_{\text{fake}}(x)$

- Setting $D(x) = q_\theta(x, y=1)$ (probability that x is a true sample) and using Binary Cross Entropy minimisation :

$$\min_\theta \left\{ -\mathbb{E}_{x\sim p} \log q_\theta(x, y(x)) \approx -\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{y_i=1} \log q_\theta(x_i) + \mathbb{1}_{y_i=0} \log(1 - q_\theta(x_i)) \right\}$$

- Equivalent to the following maximisation problem

$$\max_D \left\{ \mathbb{E}_{x\sim p_{\text{real}}(x)} \log D(x) + \mathbb{E}_{z\sim p_{\text{fake}}(z)} \log(1 - D(z)) \right\}$$

Recall: push-forward

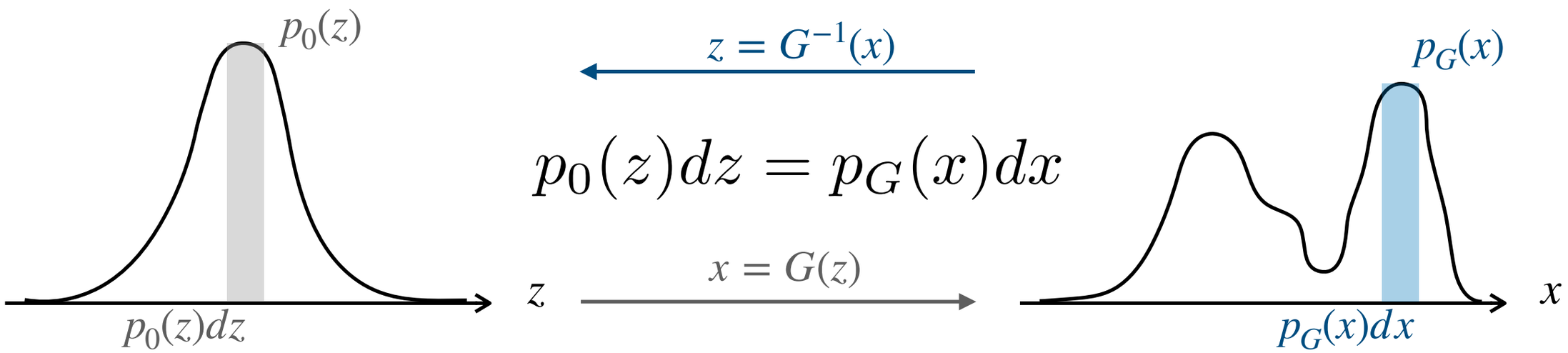
- Notation of probability distributions :

$$p = p_{\text{data}} \qquad p_{\text{latent}} = p_0 \qquad p_G = p_{\text{fake}} = G\sharp p_{\text{latent}}$$

- Definition

$$p_G(x) = G\sharp p_0(x) = p_0(\{z, G^{-1}(x) = z\})$$

- Illustration for 1D distributions
  and continuous, differentiable, bijective map G



Optimality for D

- objective loss function writes, considering a generative model G

$$L(G, D) = \mathbb{E}_{p(x)} \log D(x) + \mathbb{E}_{p_0(z)} \log(1 - D(G(z)))$$

- Considering the generated distribution p_G, we have

$$L(G, D) = \mathbb{E}_{x \sim p(x)} \log D(x) + \mathbb{E}_{x \sim p_G(x)} \log(1 - D(x))$$

- Under some conditions (existence of probability density functions)

$$L(G, D) = \int_x p(x) \log D(x) + p_G(x) \log(1 - D(x)) dx$$

- For a **fixed** G, the function $f(x) = p(x) \log D(x) + p_G(x) \log(1 - D(x))$ is optimal for

$$D^*(x) = \frac{\frac{1}{2} p(x)}{\frac{1}{2} p(x) + \frac{1}{2} p_G(x)}$$

Training the generator

- The generative network G is only trained using the discriminator supervision *solely on random (fake) samples*

$$\max_G \mathbb{E}_{z \sim p_{\text{latent}}(z)} \log D(G(z))$$

Optimality for G

- objective loss function $D^*(x) = \frac{\frac{1}{2}p(x)}{\frac{1}{2}p(x) + \frac{1}{2}p_G(x)}$

$$L(G, D) = \mathbb{E}_{x \sim p(x)} \log D(x) + \mathbb{E}_{x \sim p_{\mathrm{G}}(x)} \log(1 - D(x))$$

$$L(G, D^*) = \mathrm{KL}(p || \tfrac{1}{2}p(x) + \tfrac{1}{2}p_G(x)) + \mathrm{KL}(p_G || \tfrac{1}{2}p(x) + \tfrac{1}{2}p_G(x)) - \log 4$$

- relates to the Shannon-Jensen distance (symmetrized KL divergence)

$$L(G, D^*) = 2\,\mathrm{JS}(p, p_G) - \log 4$$

- Optimal G
  - minimum is $-\log 4$
  - achieved for $p_G = p$

Training a GAN

- Combining the two problems yields the minimax optimisation

$$\min_G \max_D \left\{ \mathbb{E}_{x \sim p_{\text{real}}(x)} \log D(x) + \mathbb{E}_{z \sim p_{\text{latent}}(z)} \log(1 - D(G(z))) \right\}$$

- Practical considerations
  - **unsupervised training** (no need for labelling, **overfitting ?**)
  - requires **lots of training samples** (*eg.* >10k for face generation)
  - mini-batch sampling of real/fake
  - adversarial training: D is trained in **inner loop** (*e.g.* 10 iterations)
  - gradient saturation: use instead $\max_G \mathbb{E}_{z \sim p_{\text{latent}}(z)} \log D(G(z))$
  - p_{latent} is a distribution that is easy to sample from, such as $\mathcal{U}([-1,1]^d)$ or $\mathcal{N}(O, \text{Id}_d)$
  - mode collapse ("helvetica scenario"): G **must not be trained too much** without updating D

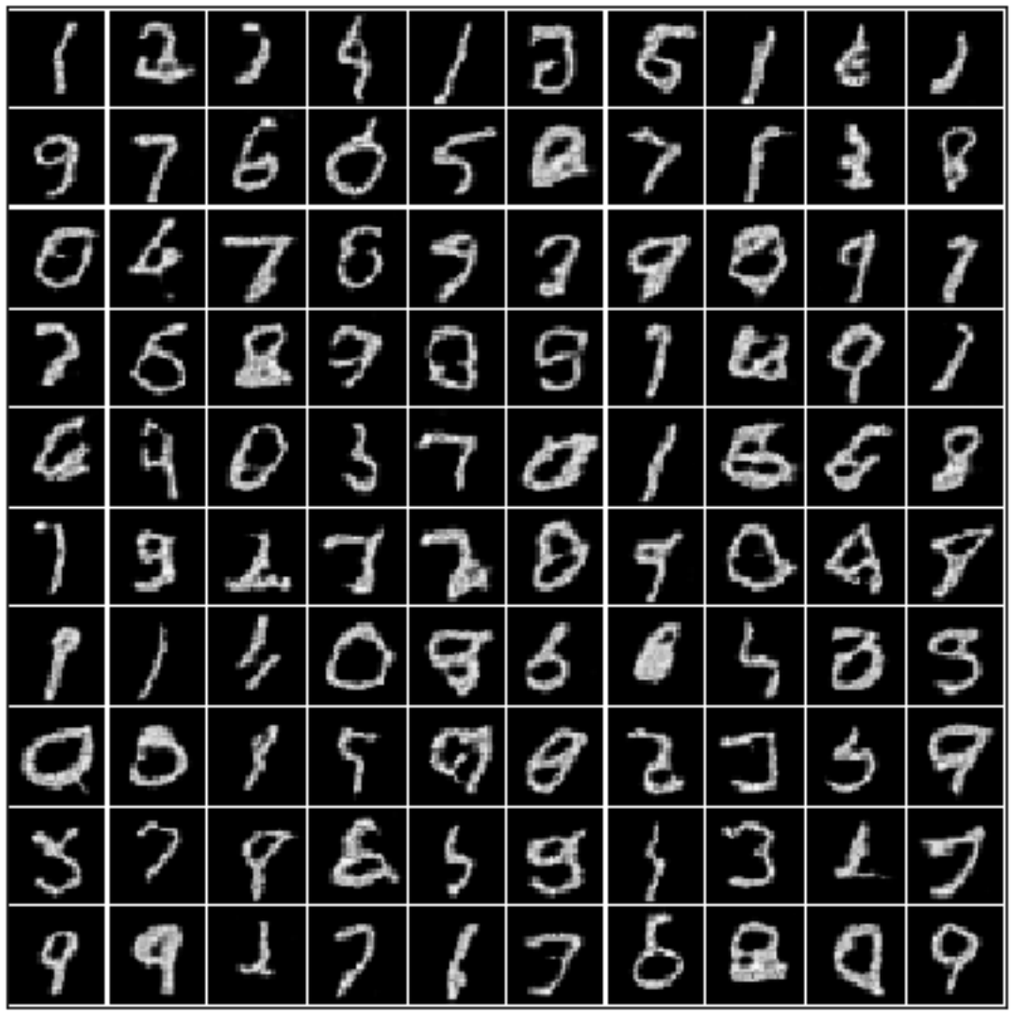
Optimality

- Practical observations:
  - D may **overfit** (memorise training data) ☞ **membership attacks**
  - instability due to overtraining G and/or D (requires checkpoints)
  - produce very sharp images
  - requires a lot of data

Demo

- **Illustrations on MNIST:** with MLP & CNN (same as AE)
  - MNIST_GAN.ipynb

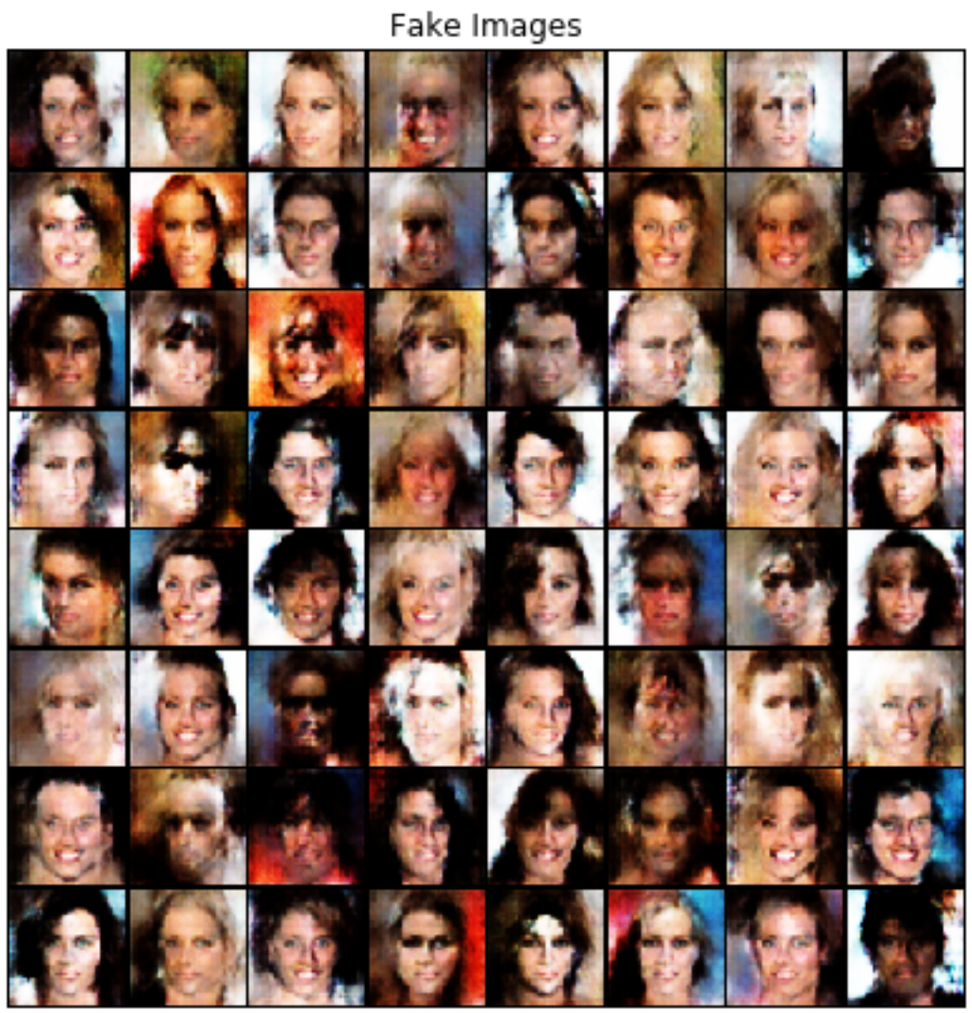
random generation with latent sampling



Demo

- **Illustrations on CelebA-HQ:** CNN & MLP
  - CELEBA_GAN.ipynb

random generation with latent multi-variate gaussian sampling



Some Variants (I)

- **LS-GAN** (for *least square*) **[Mao'16]**: $.^2$ instead of $\log(.)$

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{real}}(x)}(D(x))^2 + \mathbb{E}_{z\sim p_{\text{latent}}(z)}(1 - D(G(z)))^2$$

- **WGAN** (for *Wasserstein* distance W_1) **[Arjovsky'17]**: Lipschitz D

$$\min_G \max_{\text{1-Lip}D} \mathbb{E}_{x\sim p_{\text{real}}(x)} D(x) + \mathbb{E}_{z\sim p_{\text{latent}}(z)} 1 - D(G(z))$$

To enforce Lipschitz constraint: Weight-Clipping (WC) $\Gamma = \{\gamma \in R^n, \gamma \in [0, \tau]\}$

$$(\text{WGAN-WC}) \quad \min_{G_\theta} \max_{\gamma\in\Gamma s.t. \|D_\gamma\|_{Lip}\leq 1} \mathbb{E}_{y\sim\mathcal{Y}} \left\{D_\gamma(y))\right\} - \mathbb{E}_{x\sim\mathbb{P}_{G_\theta}} \left\{D_\gamma(x))\right\}$$

- **WGAN-GP [Gulrajani'17]**: gradient penalty to enforce the Lipschitz constraint

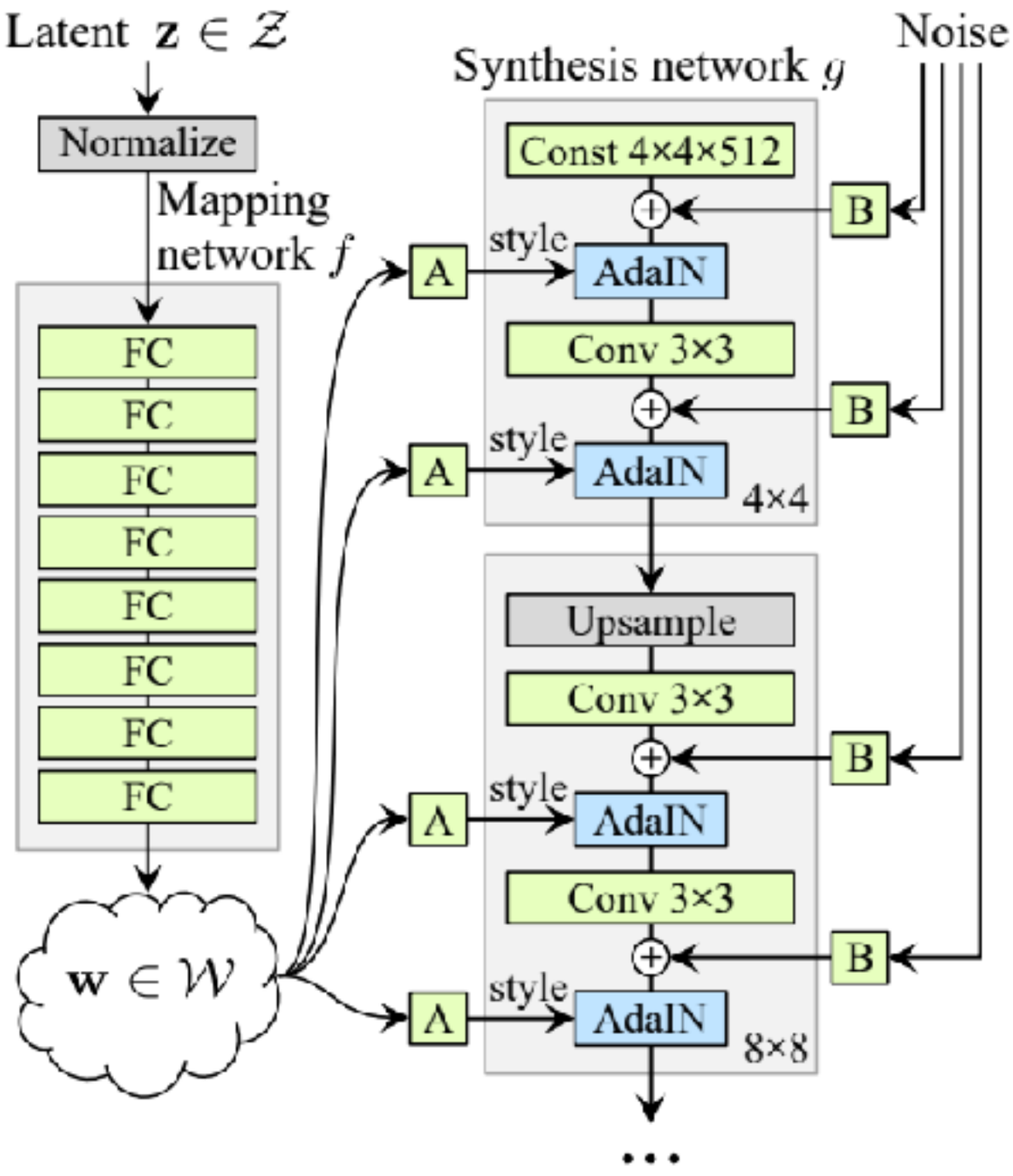
$$(\text{WGAN-GP}) \quad \min_{G_\theta} \max_{D_\gamma} \mathbb{E}_{y\sim\mathcal{Y}} \left\{D_\gamma(y))\right\} - \mathbb{E}_{x\sim\mathbb{P}_{G_\theta}} \left\{D_\gamma(x))\right\} + \lambda \mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{G}_\theta}} \left\{(\|\nabla D_\gamma(\hat{x})\| - 1)^2\right\}$$

Some Variants (II)

- **StyleGAN v1 [Karras'18] principle**:
  - adversarial optimisation (WGAN-GP)
  - novelty : Generative NN architecture
  - Inspired from **style transfer**
  - style **modulation** using adaptive instance normalization (**AdaIN**)

$$\mathrm{AdaIN}(f, s) = s_1 \frac{f - \mu(f)}{\sigma(f)} + s_2$$

  - noise injection



Some Variants (II)

- **StyleGAN v1 [Karras'18] principle**:
  - adversarial optimisation (WGAN-GP)
  - novelty : Generative NN architecture
  - Inspired from **style transfer**
  - style **modulation** using adaptive instance normalization (**AdaIN**)

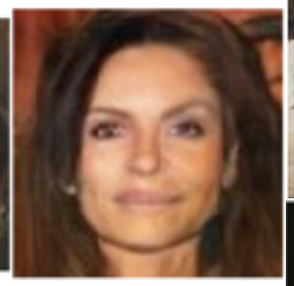
$$\mathrm{AdaIN}(f, s) = s_1 \frac{f - \mu(f)}{\sigma(f)} + s_2$$

  - noise injection

GAN performance

- “ bigger is better ”

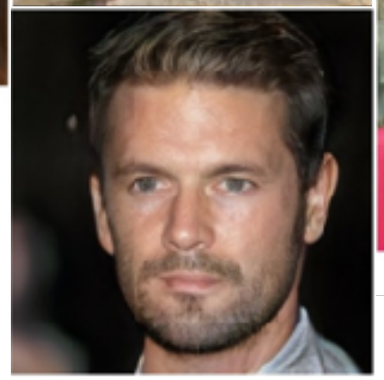
2014



2015

DC-GAN

2016



2017

PG-GAN

2018

Style-GAN

2019

Big-GAN

2023

GigaGAN

P = # paramètres	**P = 40 M**	**P = 60 M**	**P = 160 M**	**P = 1000 M**
D = # données	**D = 0.03 M**	**D = 0.08 M**	**D = 300 M**	**D = 2000 M**
J = # jours - GPU	**J = 4**	**J = 64**	**J = 1000**	**J = 5000 (A100)**

Maximum Likelihood Estimation (MLE) of latent models

Training Deep Latent Models with MLE

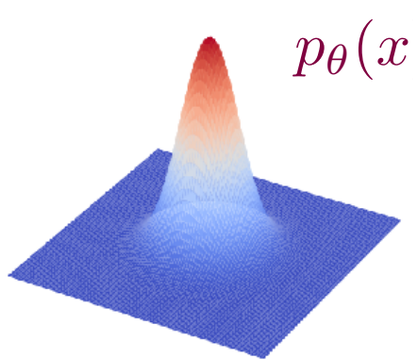
- **Motivations:**
  1. (*mixture of*) gaussian generative models p_θ can be fitted by optimising model parameters θ using **(*approximation*) of MLE**
  2. deep latent generative networks (deep image prior, texture networks, AE, GANs ...) are capable of **learning arbitrarily complex distributions**
  3. once trained, **generating random samples from p_θ is simple** ...

- **Goal:** directly optimise deep generative model parameters **to maximize model evidence on a dataset**

- **Challenges:** as seen with GMM, this is computationally involved !

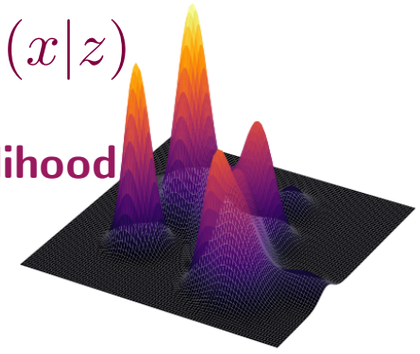
$$p_\theta(x) = G_\theta \sharp p_\theta(z) = \int_z p_\theta(x|z) p_\theta(z) dz = \mathbb{E}_{z \sim p_\theta(z)} p_\theta(x|z)$$

conditional likelihood



$p_\theta(z)$

with $z \sim p_\theta(z)$ sample $\hat{x} = G_\theta(z) \sim p_\theta(x)$



$p_\theta(x)$

Training Deep Latent Models with MLE

- **Motivations:**
  1. (*mixture of*) gaussian generative models p_θ can be fitted by optimising model parameters θ using **(*approximation*) of MLE**
  2. deep latent generative networks (deep image prior, texture networks, AE, GANs ...) are capable of **learning arbitrarily complex distributions**
  3. once trained, **generating random samples from p_θ is simple** ...

- **Goal:** directly optimise deep generative model parameters **to maximize model evidence on a dataset**

- **Challenges:** as seen with GMM, this is a difficult problem, even in the simplest setting

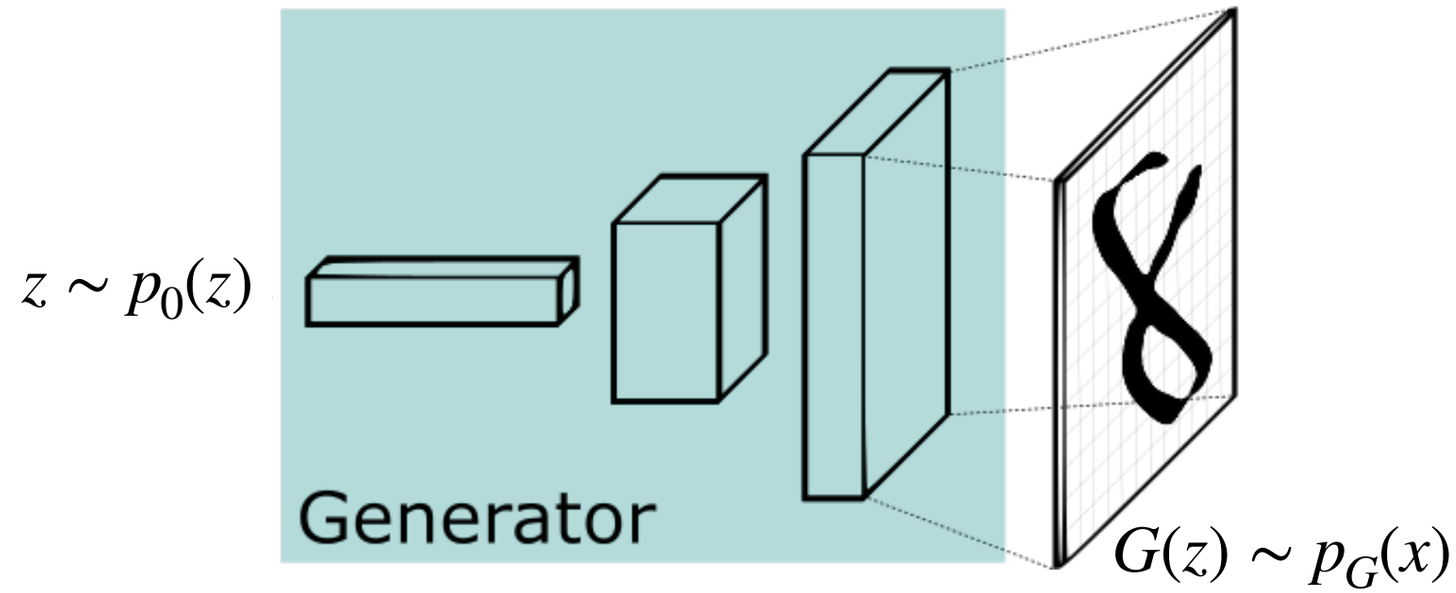
- **Related methods in the rest of the presentation:**
  - VAE **[Kingma & Welling, '14]**
  - Normalising Flow (**GLOW [Kingma'18]**)
  - Diffusion Models (Latent Diffusion **[Rombach'22]**), Flow matching, Rectified flow ...

Variational Auto-Encoders

(details in syllabus)

Variational Auto-Encoder (VAE)

- **Reference:** Auto-Encoding Variational Bayes **[Kingma & Welling, '14]**
- Used in **SOTA** models, combined with other techniques
  (Latent Diffusion [Rombach'22], **VQ - VAE 2** [Razavi'19], VD-VAE [Child'21])
- **Goal:** like GANs, generate **realistic** samples by sampling **latent codes** z from a known prior distribution that are decoded with the generator $G(z)$



- **Training:** unlike GANs, quality of samples is not assessed from an unsupervised classifier, but by evaluating the **likelihood $p_G(x)$ of samples** z, i.e. the **model evidence**

$$min_G - \mathcal{L}(G) = \mathbb{E}_{x \sim P_{data}} \{ - \log p_G(x) \}$$

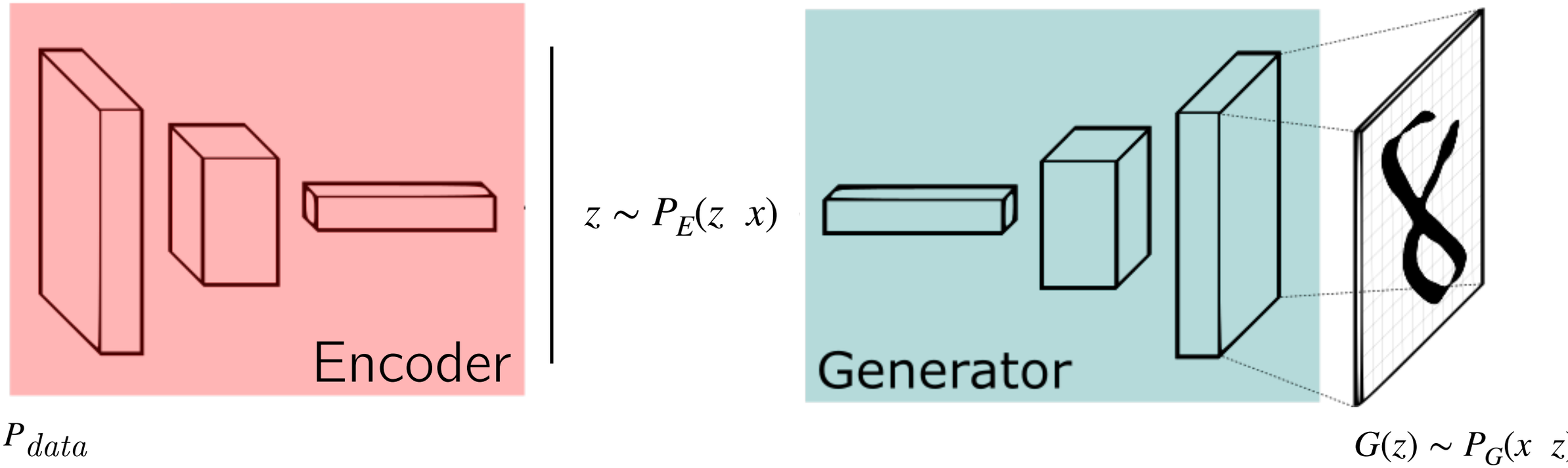
- **Gist**: using AE principle, batch of latent z are sampled **during training** from a pdf **parametrized by an auxiliary encoder**

VAE optimisation problem

- **(intractable) objective** with the marginal likelihood (evidence)

$$min_G - \mathcal{L}(G) = \mathbb{E}_{x \sim P_{data}} \{ - \log p_G(x) \}$$

- **Variational approach:** approximate the evidence by optimizating a **surrogate probability function** $P_E(z \mid x)$, parametrized by a neural network (**encoder E**)



- **Optimization problem:** based on **ELBO** (**Evidence Lower BOund**)

$$\mathcal{L}(G) \geq \text{ELBO}(G, E) = \mathbb{E}_{x \sim P_{data}} \left\{ \mathbb{E}_{z \sim P_E(z|x)} \log P_G(x|z) - \text{KL}(P_E(z|x) || p(z)) \right\}$$

likelihood of data from estimated latent + "distance" to the desired prior

Evidence Lower BOund

For a single datapoint x, the marginal likelihood of the latent model $p_\theta(x, z)$ satisfies for **any pdf prior** q on the latent variable (see syllabus for details):

$$\begin{aligned}
\log p_\theta(x) &= \int_{z\in\mathbb{R}^\ell} q(z)\log p_\theta(x)dz = \mathbb{E}_{z\sim q}\left[\log p_\theta(x)\right] \\
&= \int_{z\in\mathbb{R}^\ell} q(z)\log\left(\frac{p_\theta(x)p_\theta(z|x)}{p_\theta(z|x)}\right)dz = \int_{z\in\mathbb{R}^\ell} q(z)\log\left(\frac{p_\theta(x,z)}{p_\theta(z|x)}\right)dz \\
&= \int_{z\in\mathbb{R}^\ell} q(z)\log\left(\frac{p_\theta(x,z)}{p_\theta(z|x)}\frac{q(z)}{q(z)}\right)dz \\
&= \int_{z\in\mathbb{R}^\ell} q(z)\log\left(\frac{p_\theta(x,z)}{q(z)}\right)dz + \int_{z\in\mathbb{R}^\ell} q(z)\log\left(\frac{q(z)}{p_\theta(z|x)}\right)dz \\
&= \mathbb{E}_{z\sim q}\left[\log\left(\frac{p_\theta(x,z)}{q(z)}\right)\right] + \underbrace{\mathrm{KL}\left(q(z)||p_\theta(z|x)\right)}_{\geq 0} \\
&\geq \mathbb{E}_{z\sim q}\left[\log\left(\frac{p_\theta(x,z)}{q(z)}\right)\right]
\end{aligned}$$

In VAE, $q(z) = p_\phi(z \mid x)$ is a **differential surrogate** to the intractable **posterior** $p_\theta(z \mid x)$ where ϕ are the parameters of the encoder E

Variational Bayes Inference

- **Optimising** the parametric posterior distribution to reduce the KL gap, we can rewrite the lower bound

$$\min_{\theta} \mathbb{E}_{x\sim P} - \log p_\theta(x) \leq \min_{\theta} \min_{\phi} \mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(.|x)} \left[-\log \left(\frac{p_\theta(x,z)}{q_\phi(z|x)} \right) \right]$$

- The corresponding **Variational Bayes Inference** problem writes:

$$\begin{aligned}
\min_{\theta,\phi} \mathcal{L}(\theta,\phi) &:= -\mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \left(\frac{p_\theta(x,z)}{q_\phi(z|x)} \right) \right] \\
&= -\mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \left(p_\theta(x,z) \right) \right] + \mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \left(q_\phi(z|x) \right) \right] \\
&= \mathrm{KL}(q_\phi(z|x) \,||\, p_\theta(z)) - \mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \left(p_\theta(x|z) \right) \right]
\end{aligned}$$

- This problem has an interesting interpretation when training an AE:
  - the first term **regularises the latent distribution** by enforcing **the desired prior** with the Kullback-Leibler Divergence
  - the second term **maximise the expected conditional likelihood** of training data

Variational Bayes Inference for AE

- **Optimization problem:** minimisation w.r.t E and G simultaneously (resp. θ & ϕ)

$$\min_{\theta,\phi} \mathcal{L}(\theta,\phi) = \mathrm{KL}(q_\phi(z|x)\,||\,p_\theta(z)) - \mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta(x|z)\right)\right]$$

- **Requirements:** for deep latent model we need to **compute the gradient** of this objective function using auto-differentiation

- **Step 1 - Optimising the model's parameters θ:** when choosing a non-parametric prior $p_\theta(z) = p_0(z)$ (e.g. $\mathcal{N}(0,I)$), **only the second term is optimized** (*see the syllabus for the general case*)

$$\min_{\theta} f(\theta) = -\mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta(x|z)\right)\right]$$

with mini-batch approximation of the expectations

$$\nabla_\theta f(\theta) \approx -\frac{1}{B}\sum_{x_b\in\mathcal{B}} \nabla_\theta\left[\log p_\theta(x_b|z_b)\right] \text{ where } z_b \sim q_\phi(.|x_b) \text{ and } x_b \sim P$$

In VAE, $p_\theta(x\ \ z)$ is a differentiable pdf that is parametrised by the decoder (generator)

Variational Bayes Inference for AE

- **Optimization problem:** minimization w.r.t E and G simultaneously (resp. θ & ϕ)

$$\min_{\theta,\phi} \mathcal{L}(\theta,\phi) = \mathrm{KL}(q_\phi(z|x)\,||\,p_\theta(z)) - \mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta(x|z)\right)\right]$$

- **Step 2 - Optimising the posterior parameters ϕ** is **more challenging** as we need to differentiate the expectations which depend both on the posterior

- **" Reparametrization trick "** : (*see the syllabus*) with $\varepsilon \sim q_0(\varepsilon) = \mathcal{N}(\mathbf{0}, \mathrm{Id})$

$$z = h_\phi(\varepsilon, x) = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon \sim q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x)^2)$$

In VAE, posterior parameters $(\mu_\theta(x), \sigma^2(x)) = E(x)$ are the encoder outputs !
(this is not the latent representation of x ...)

- **Gradient expression** now can be estimated by sampling random ε and x

$$\nabla_\phi \mathcal{L}(\theta,\phi) \approx \tfrac{1}{B}\sum_{x_b\in B}\tfrac{1}{B'}\sum_{\varepsilon_b\in B'}\nabla_\phi\left[-\log p_\theta(h_\phi(\varepsilon_b,x_b)) - \log p_\theta(x_b|h_\phi(\varepsilon,x_b)) + \log\left(q_\phi(h_\phi(\varepsilon_b,x_b)|x_b)\right)\right]$$

KL divergence with 1D Gaussians

- Between two gaussian distributions, the **KL has a closed form expression** :

$$\mathrm{KL}\left(\mathcal{N}(\mu_1, \sigma_1^2) \,||\, \mathcal{N}(\mu_2, \sigma_2^2)\right) = \frac{1}{2}\left(\frac{(\mu_1 - \mu_2)^2 + \sigma_1^2}{\sigma_2^2} + \log\frac{\sigma_2^2}{\sigma_1^2} - 1\right)$$

- **In particular**

$$\mathrm{KL}\left(\mathcal{N}(\mu, \sigma^2) \,||\, \mathcal{N}(0, 1)\right) = \frac{1}{2}\left(\mu^2 + \sigma^2 - \log(\sigma^2) - 1\right)$$

KL divergence with multivariate Gaussians

- For **separable** (but not isotrope) multivariate Gaussians

$\mu = (\mu_i)_{i\in[d]}, \sigma^2 = (\sigma_i^2)_{i\in[d]} \in \mathbb{R}^d$ and $\Sigma = \mathrm{Id}\,\sigma^2$ is a diagonal covariance matrix

$$\mathrm{KL}\left(\mathcal{N}(\mu, \mathbf{I}\,\sigma^2)\,||\,\mathcal{N}(\mathbf{0}, 1)\right) = \mathbb{E}_{\mathcal{N}(\mu,\mathbf{I}\,\sigma^2)} \log\left[\frac{\mathcal{N}(\mu,\sigma^2)}{\mathcal{N}(\mathbf{0},\mathbf{I})}\right]$$

$$= \mathbb{E}_{z\sim\mathcal{N}(\mu,\mathbf{I}\,\sigma^2)} \log\left[\frac{\frac{1}{\sqrt{\Pi_i 2\pi\sigma_i^2}} e^{-\sum_i \frac{1}{2\sigma_i^2}(z_i-\mu_i)^2}}{\frac{1}{\sqrt{2\pi}^d} e^{-\frac{1}{2}\|z\|^2}}\right]$$

$$= \mathbb{E}_{z\sim\mathcal{N}(\mu,\mathbf{I}\,\sigma^2)} \left[\sum_i -\log(\sigma_i) - \tfrac{1}{2\sigma_i^2}(z_i-\mu_i)^2 + \tfrac{1}{2} z_i^2\right]$$

$$= \sum_i \mathbb{E}_{z_i\sim\mathcal{N}(\mu_i,\sigma_i^2)} \left[-\log(\sigma_i) - \tfrac{1}{2\sigma_i^2}(z_i-\mu_i)^2 + \tfrac{1}{2} z_i^2\right]$$

$$= \sum_{i=1}^{d} -\frac{1}{2} - \frac{1}{2}\log(\sigma_i^2) + \tfrac{1}{2}(\mu_i^2 + \sigma_i^2)$$

Training a VAE with Gaussians

- To simplify expressions, we can use Gaussians PDF for the **prior** $p_\theta(z)$, the approximate **posterior** $q_\theta(z \mid x)$ and the conditional **likelihood** $p_\theta(x \mid z)$!
  - **Gaussian prior**: $\varepsilon \sim q_0(\varepsilon) = \mathcal{N}(\mathbf{0}, \mathrm{Id})$
  - **Gaussian Posterior** with Reparametrization trick (allows back propagation)

$$z = h_\phi(\varepsilon, x) = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon \sim q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x)^2)$$

  - **Gaussian likelihood for generated samples:** with $P_G(x \mid z) = \mathcal{N}(G(z), s^2 Id)$

- **Optimization problem:** minimization w.r.t θ & ϕ

$$\min_{\theta,\phi} \mathcal{L}(\theta, \phi) = \mathrm{KL}(q_\phi(z|x) \,||\, p_\theta(z)) - \mathbb{E}_{x\sim P}\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log\left(p_\theta(x|z)\right)\right]$$

$$\mathcal{L}(\theta, \phi) = \tfrac{1}{2}\mathbb{E}_{x\sim P}\, \mathbb{E}_{\substack{\varepsilon\sim\mathcal{N}(\mathbf{0},\mathrm{Id}) \\ z=\mu_\phi(x)+\varepsilon\odot\sigma_\phi(x)}} \Big[\tfrac{1}{s^2}\|x - G_\theta(z)\|^2 + \|\mu_\phi(x)\|^2 + \|\sigma_\phi(x)\|^2 - \langle \mathbf{1}, \log(\sigma_\phi(x)^2)\rangle \Big] + Cst$$

Practical setting (VAE)

- **Practical trick I:** Enforcing the **positivity** of standard deviation σ can be **cumbersome** for posterior parameters $(\mu_\theta(x), \sigma^2(x)) = E(x)$: predict instead $(\mu_\theta(x), s(x) := \log \sigma^2(x)) = E(x)$ and then use $\sigma^2(x) = \exp s(x) > 0$

- **Practical trick II (conditional VAE):** Controlling **attributes** y of the generated samples x (*eg.* hair color) by implicit **supervised learning:** given an annoyed dataset $\{x_i, y_i\}_i$, the encoder inputs are both x and y

- **Advantages :**
  - compared to AE, increasing the latent dimension **does not let the model overfit**
  - **more stable** than GANs, requires less samples
  - **Interesting by-product :** the encoder can still be used after training !
  - **Sampling after training :** compared to AE, easy to sample the latent distribution as

$$q_\phi(z|x) \approx \mathcal{N}(0, Id) \text{ i.e. } z \sim \mathcal{N}(0, Id)$$

- **Drawbacks :**
  - **approximate** method to estimate the model likelihood
  - generated samples are **still blurry & not as sharp** as with GANs

Conclusion: VAE in a Nutshell

- **Sampling :**

Gaussian Latent

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

Encoder

Gaussian parameters

$$\mu_\phi(x)$$

$$x \sim p_{\text{data}} \quad E_\phi$$

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon$$

$$\sigma_\phi(x)$$

Decoder

$$D_\theta \quad \hat{x}$$

used for training

Gaussian Output

$$\hat{x} = D_\theta(z) + s\,\varepsilon$$

$$\hat{x} \sim \mathcal{N}(D_\theta(z), s^2\mathbf{I})$$

- **Training :**

$$\mathcal{L}(\theta, \phi) = \tfrac{1}{2}\mathbb{E}_{x\sim P}\,\mathbb{E}_{\substack{\varepsilon\sim\mathcal{N}(\mathbf{0},\mathrm{Id}) \\ z=\mu_\phi(x)+\varepsilon\odot\sigma_\phi(x)}}\Big[\tfrac{1}{s^2}\|x - G_\theta(z)\|^2 \longrightarrow \textbf{MSE (like Auto-Encoder)}$$

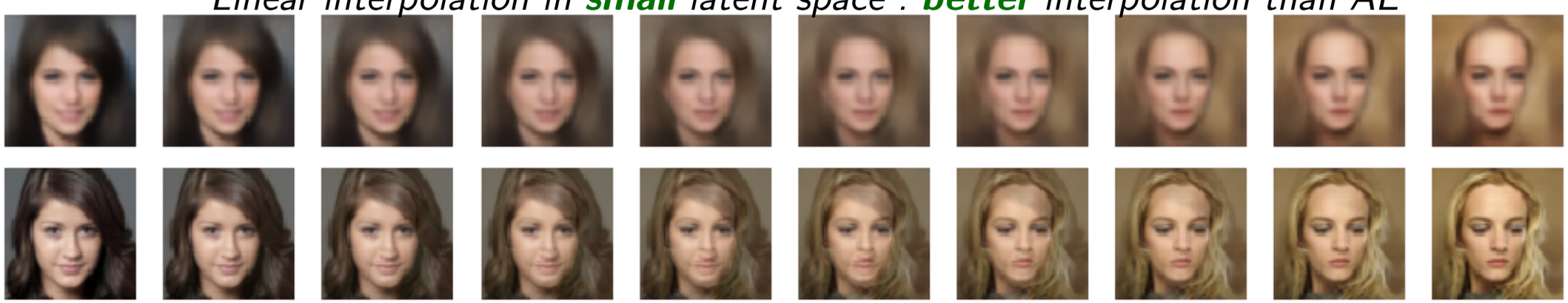
$$\textbf{Enforce } z \sim \mathcal{N}(0,I_d) \longleftarrow + \|\mu_\phi(x)\|^2 + \|\sigma_\phi(x)\|^2 - \langle \mathbf{1}, \log(\sigma_\phi(x)^2)\rangle\Big] + Cst$$

demo VAE

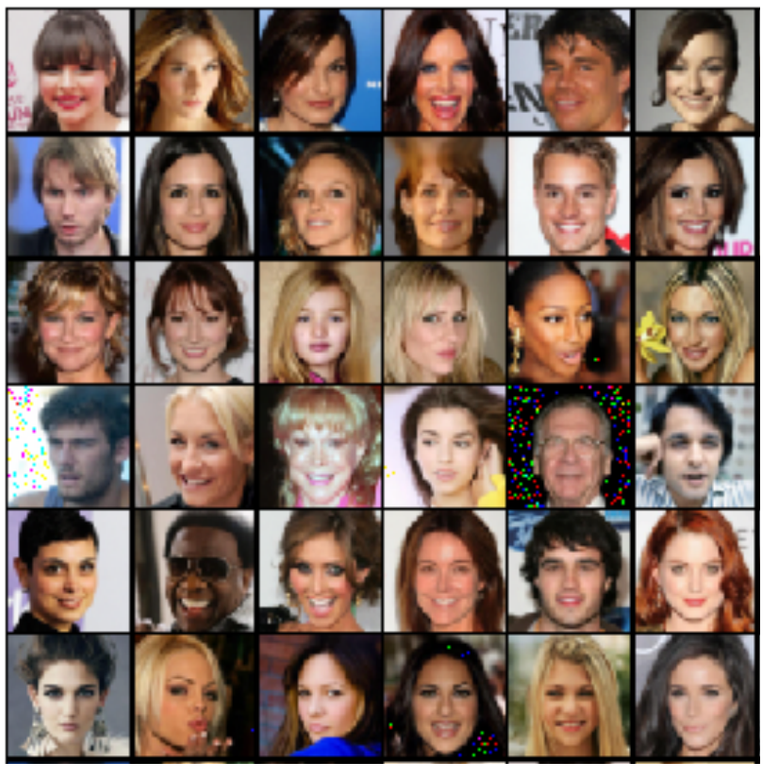
- **Illustrations on CelebA-HQ:** CNN & MLP
  - CELEBA_vae.ipynb

☞ comparaison with AE

*Linear interpolation in **small** latent space : **better** interpolation than AE*

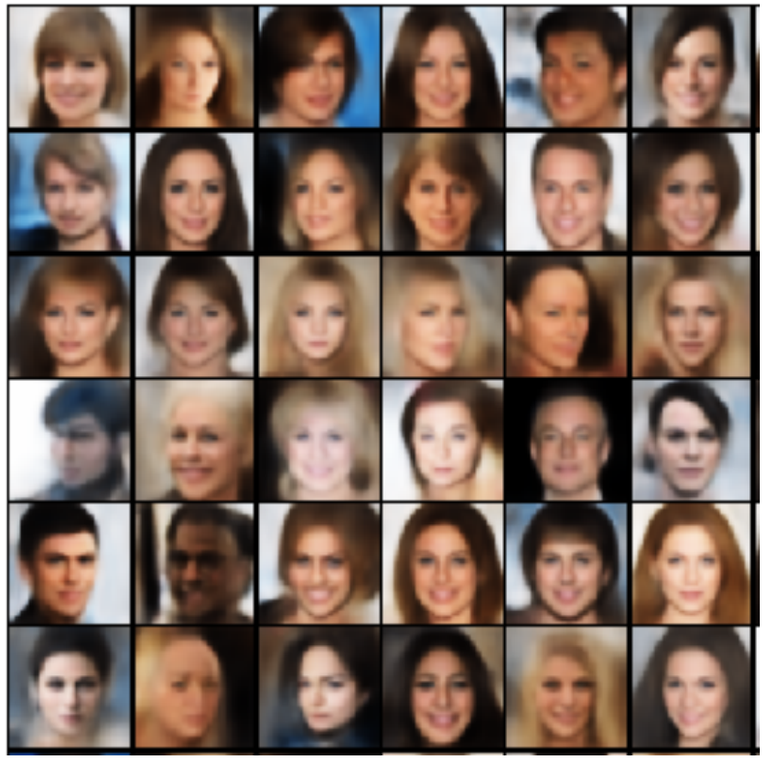


Linear interpolation in pixel space



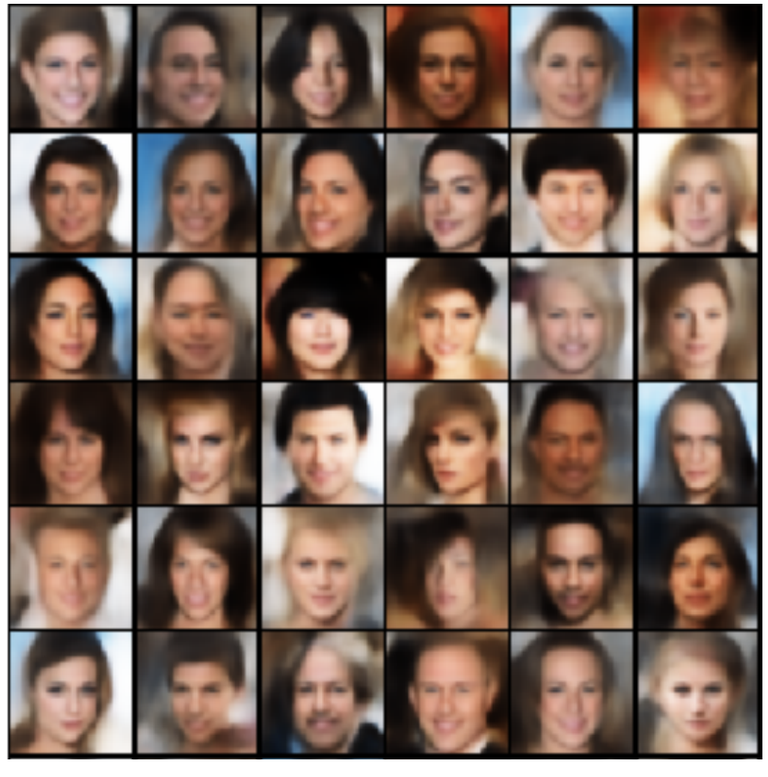
test images

x'



test reconstruction

$G(E(x'))$

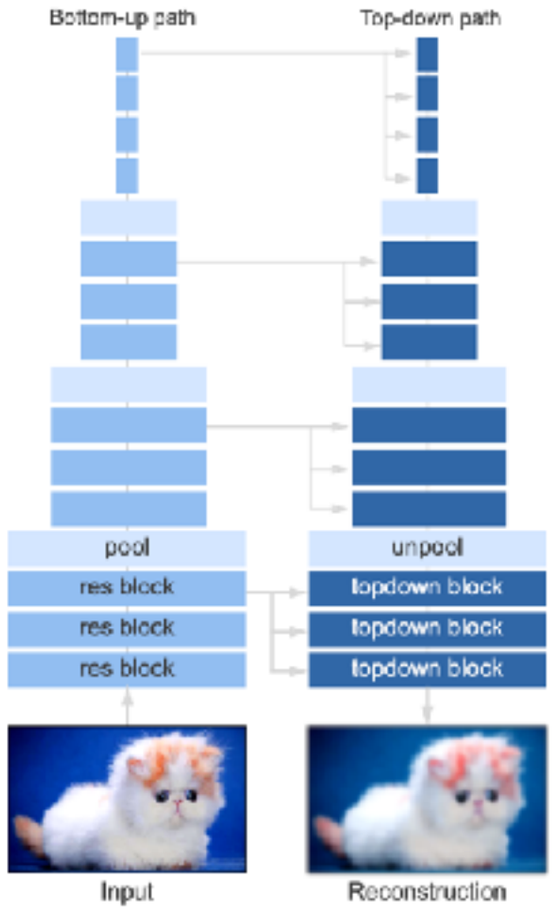


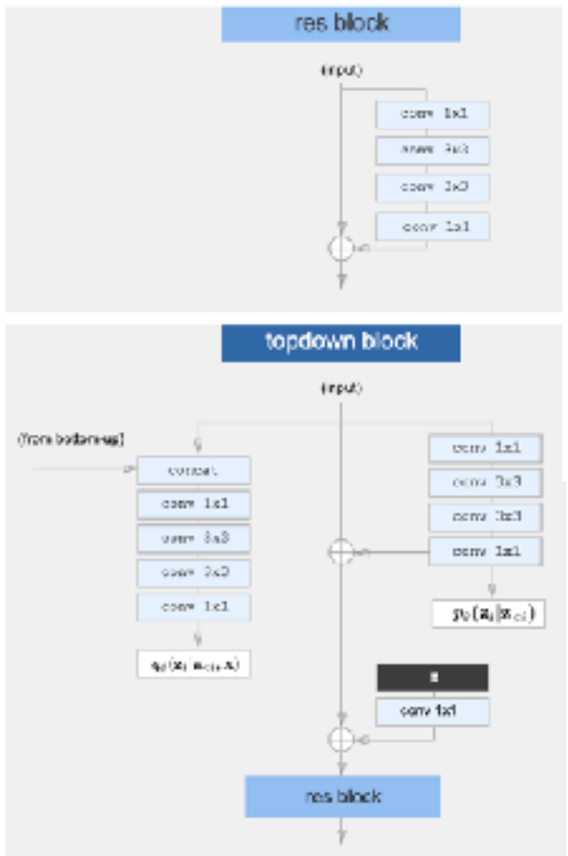
random samples

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

SOTA in VAE

- **Examples from:** VD-VAE [Child '21]





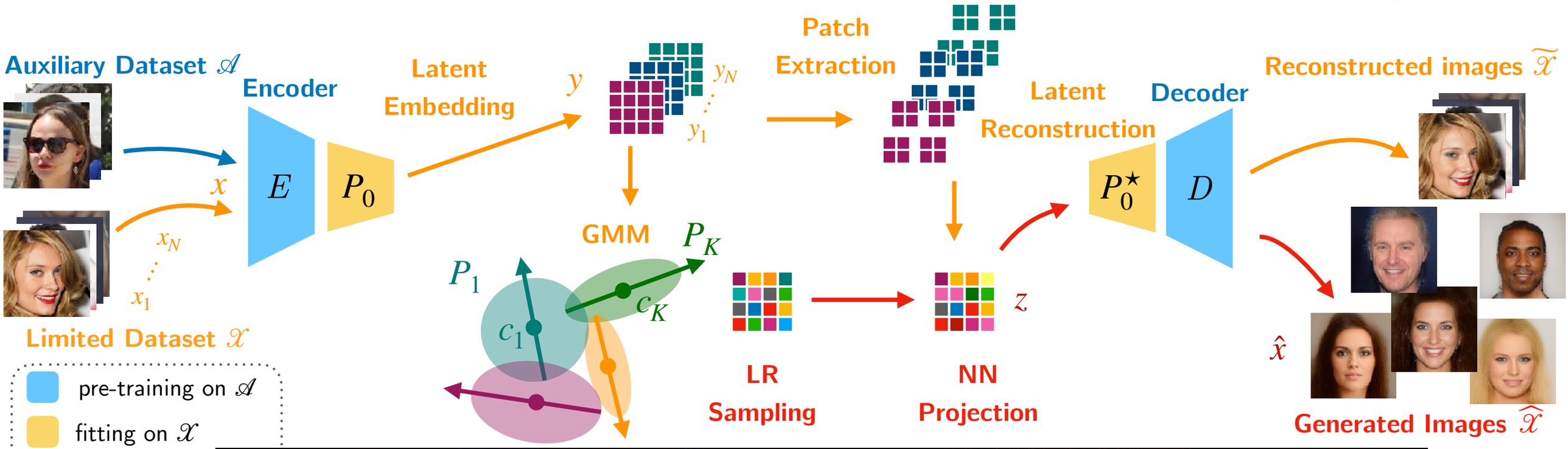
Deep Hierarchical VAE (48 layers, 100M params)

FFHQ-256

FFHQ-1024

Latent generative modelling

- **Examples from:** Latent-GMM **[Samuth'24]**

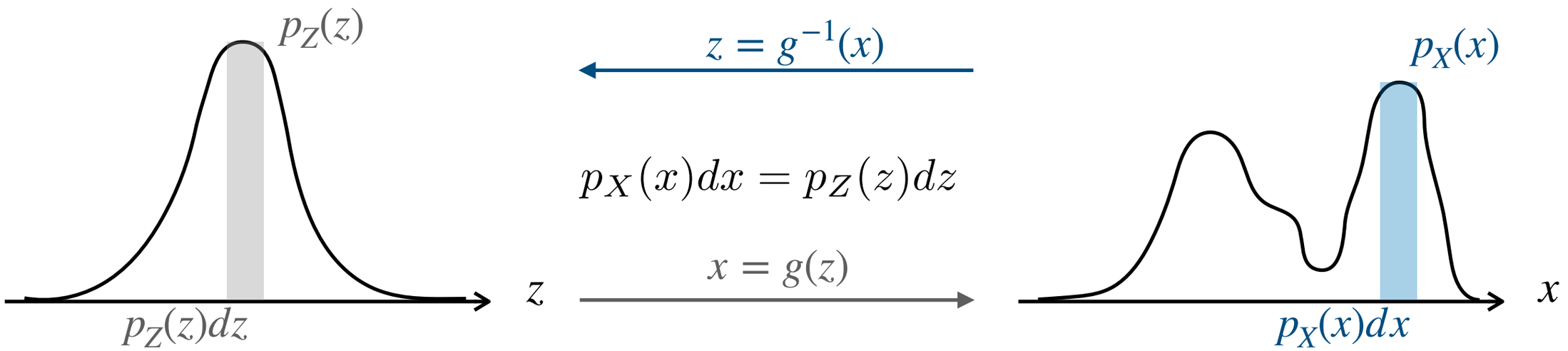


Normalising Flow

Flow-based generative models

NICE [Dinh et al., 2014] & RealNVP [Dinh et al., 2016]

- **Principle:** define an **invertible** network G to optimise explicitly the posterior distribution



- **1D illustration (provided g is monotonic):** $p_X(x) = p_Z(z)\frac{dg^{-1}(x)}{dx} = p_Z(z)\frac{1}{g'(z)}$
- **In general (for a bijective mapping g):** $p_X(x) = p_Z(z)\left|det\left(\nabla_x g^{-1}(x)\right)\right|$
- **Deep Neural network:** given $f = g^{-1} = f_1 \circ \ldots \circ f_L$ and $x_\ell = f_\ell(x_{\ell-1})$ and $x_0 = x$

Negative Log Likelihood (NNL) $-\log p_X(x) = -\log p_Z(f(x)) - \sum_{\ell=1}^{L} |det\left(\nabla_x f_\ell(x_\ell)\right)|$

Jacobian matrix

Flow-based generative models

- **Objective loss :** given $f = g^{-1} = f_1 \circ \ldots \circ f_L$ and $x_\ell = f_\ell(x_{\ell-1})$ and $x_0 = x$

$$min_f \; \mathbb{E}_{x \sim P_{data}} \left\{ -\log p_X(x) = -\log p_Z(f(x)) - \sum_{\ell=1}^{L} |det\left(\nabla_x f_\ell(x_\ell)\right)| \right\}$$

- **Optimisation :** stochastic mini-batch gradient descent

$$\mathbb{E}_{x \sim P_{data}} \log p_X(x) \approx \frac{1}{N} \sum_{i=1}^{N} \log p_X(x_i)$$

GLOW [Kingma'18]
using invertible 1x1 convolutions

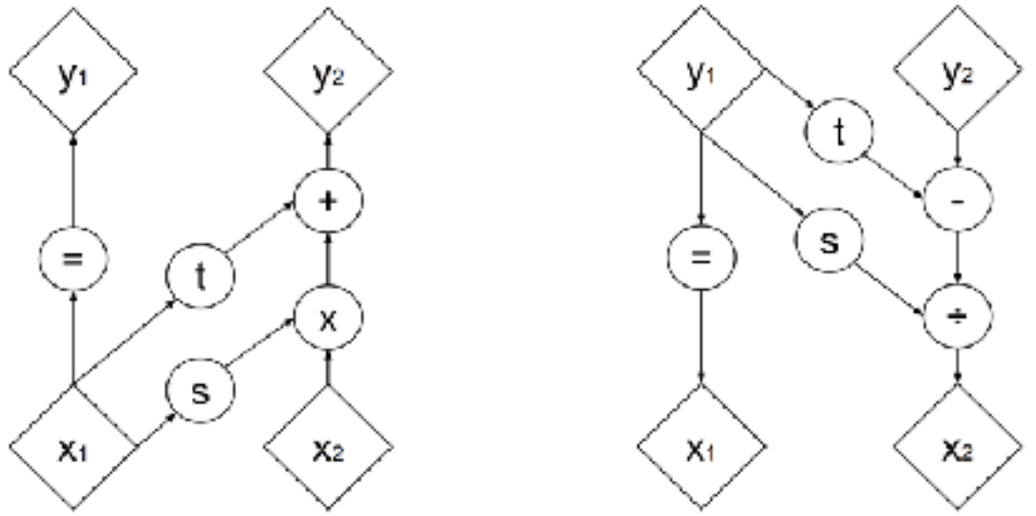
Random samples x = g(z)
Trained on CelebHQ

Note : latent code z sampled at low temperature (ie gaussian with lower variance)

- **Advantages : exact** method to estimate the NNL, can compare test/train **to detect overfitting**
- **Drawbacks :** restriction to **invertible** modules f_k with **simple Jacobian** expression, which **limits** the class of trainable generative model

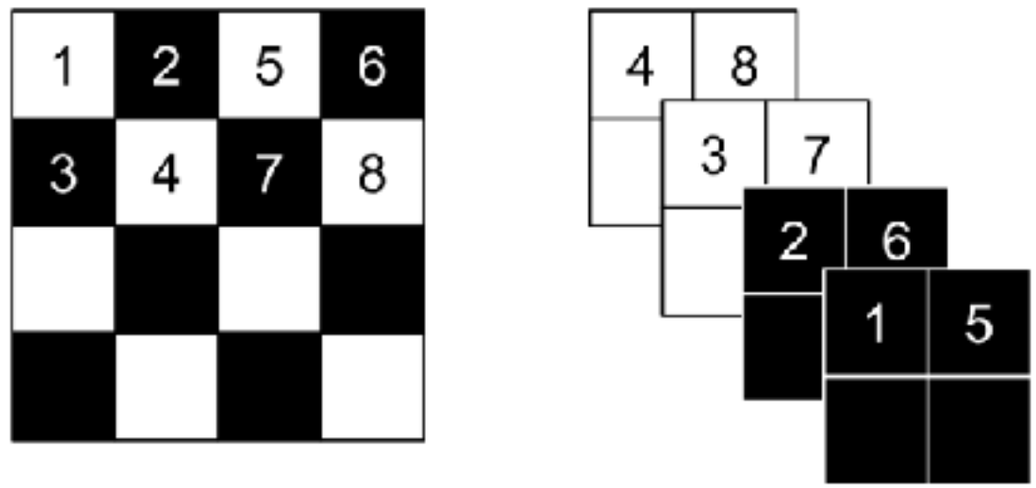
NN with invertible layers ?

- Example of the **Split & Copy Trick:**
  - vector x (first input signal, then latent feature maps) are **split** into $x = (x_1, x_2)$ along **channels dimensions**
  - x_1 is **copied**: $y_1 = x_1$
  - x_2 is **transformed with NNs:**
    $y_2 = x_2 \odot \exp s(x_1) + t(x_1)$
  - *Note: s and t are MLP altering each pixel independently*
  - **inversion:** $x_1 = y_1$ and
    $x_2 = (y_2 - t(y_1)) \odot \exp(-s(y_1))$
  - then **permutation** of y_1 & y_2
- To mix **spatial dimensions**, masking schemes are used :



(a) Forward propagation (b) Inverse propagation

Density estimation using Real NVP
[Dinh ICLR'17]



Invertible Masking schemes for mixing spatial information

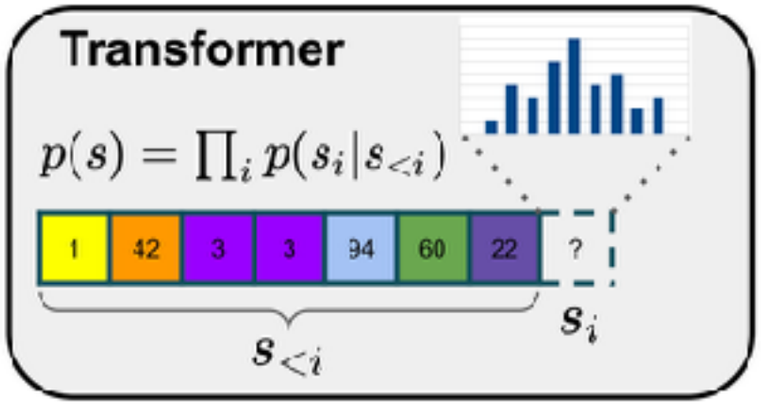
Auto-Regressive Models & Transformers

Motivation

- Popular approach for text & image generation : Recurrent neural network [Graves'13], PixelRNN & PixelCNN [van den Oord'16], Transformers [Vaswani'17] ...

- SOTA for LLM (large language models) and image generation when combined with other techniques (AE, GAN, VQ ...) : **VQ-GAN [Esser'21]**

- **Principle** : given a (causal) sequence of tokens (RGB, latent VQ, ...) predict the **conditional distribution of the next one**



- Training : this is again a **likelihood-based** method aiming at maximizing

$$P(X) = \int_z P(X|z)P(z)dz = \mathbb{E}_{z\sim \mathcal{Z}} P(X|z) \approx \frac{1}{B}\sum_{b=1}^{B} P(X|z_b)$$

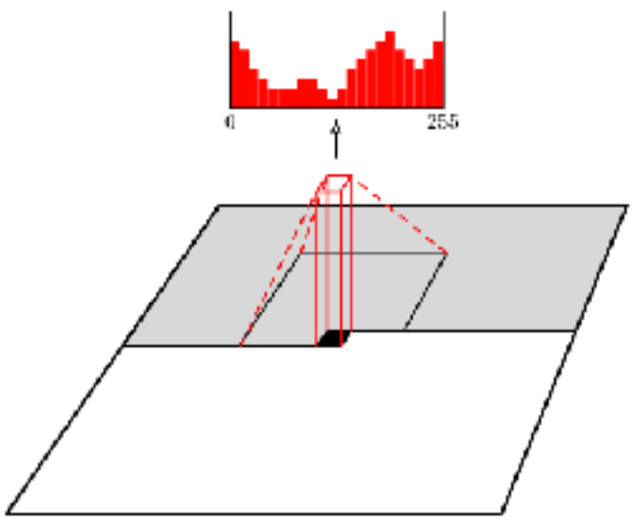
Auto-Regressive Models

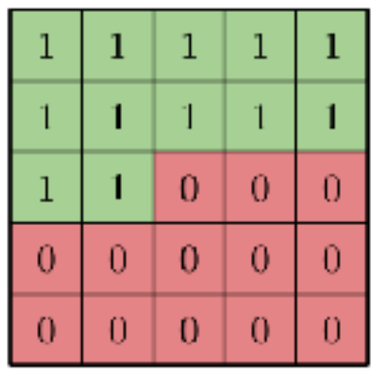
- **Problem** : model the probability distribution of a **sequence** $x = (x_1, x_2, \dots x_t)$

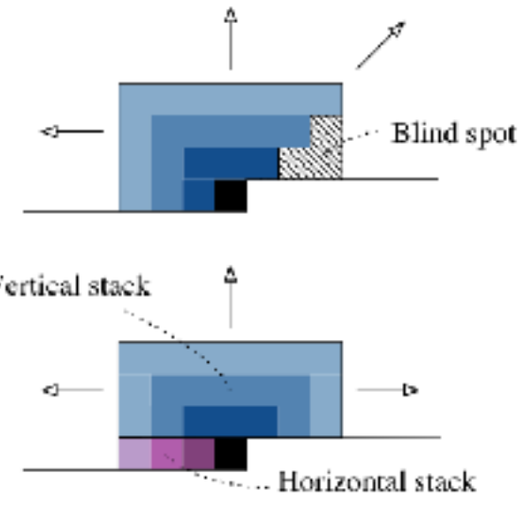
$$\begin{aligned}\mathbb{P}(x) &= \mathbb{P}(x_1, x_2, \dots x_t) = \mathbb{P}(x_1) \times \mathbb{P}(x_2|x_1) \times \dots \mathbb{P}(x_t|x_1, \dots x_{t-1}) \\ &= \Pi_{n=1}^{t} \mathbb{P}(x_n|x_{<n})\end{aligned}$$

- Data **sequence** in machine learning:
  - Naturally arise in **natural language processing (NLP)**, time series ... which require to model **causal phenomena**.
  - **Less natural for images** where **sequences can be very long** (requires masked patch/ sliding window) and **with arbitrary ordering** (e.g. raster scan order)

from PixelCNN Decoder [Oord'16]







Likelihood Modeling with Neural Networks

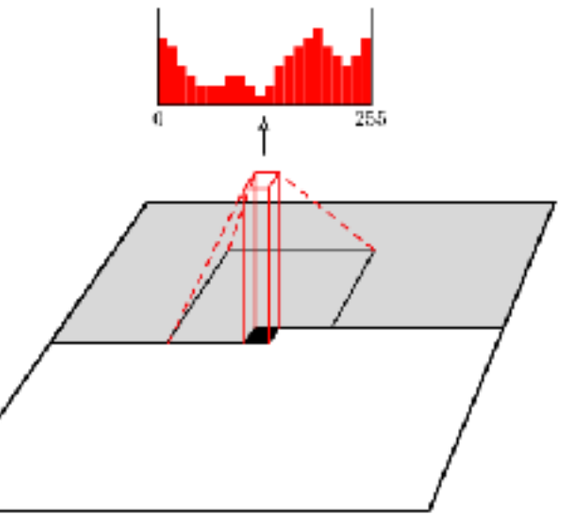
- **Problem** : model the probability distribution of a **sequence** $x = (x_1, x_2, \dots x_t)$

$$\begin{aligned}\mathbb{P}(x) &= \mathbb{P}(x_1, x_2, \dots x_t) = \mathbb{P}(x_1) \times \mathbb{P}(x_2|x_1) \times ...\mathbb{P}(x_t|x_1, ...x_{t-1}) \\ &= \Pi_{n=1}^{t} \mathbb{P}(x_n|x_{<n})\end{aligned}$$

- **Design**: how to define the likelihood $\mathbb{P}(x_n \ \ x_{<n})$?
  - **continuous case**: with **density** probability function can be used
    - For GMM: explicit optimisation of the parameters of gaussians (mean vectors, covariance matrices, mixture weights)
    - For VAE: a Neural Network (encoder) predicts the means and variances
  - **discrete case**: where $x_n \in \mathscr{C}$ takes values in a finite codebook (8-bit Grey level values, 32-bit sound amplitude, ASCII symbols ...)
    - A **neural network with a final softmax layer** can be used to parametrize probabilities $\mathbb{P}(x_n = c \in \mathscr{C} \ \ x_{<n})$

Examples of discrete time series

- Pixel Recurrent Neural Networks (Pixel RNN) and later PixelCNN **[van den Oord '16]**:
  - RGB values are quantified in 256 values
  - R are first predicted using previous values, then G, then B



- GPT **[Radford '18]**:
  - sub-words are **quantified** in tokens
  - next tokens are predicted using a classifier

Tokens : among 50,257

Coucou les amis !

Tokens ID:

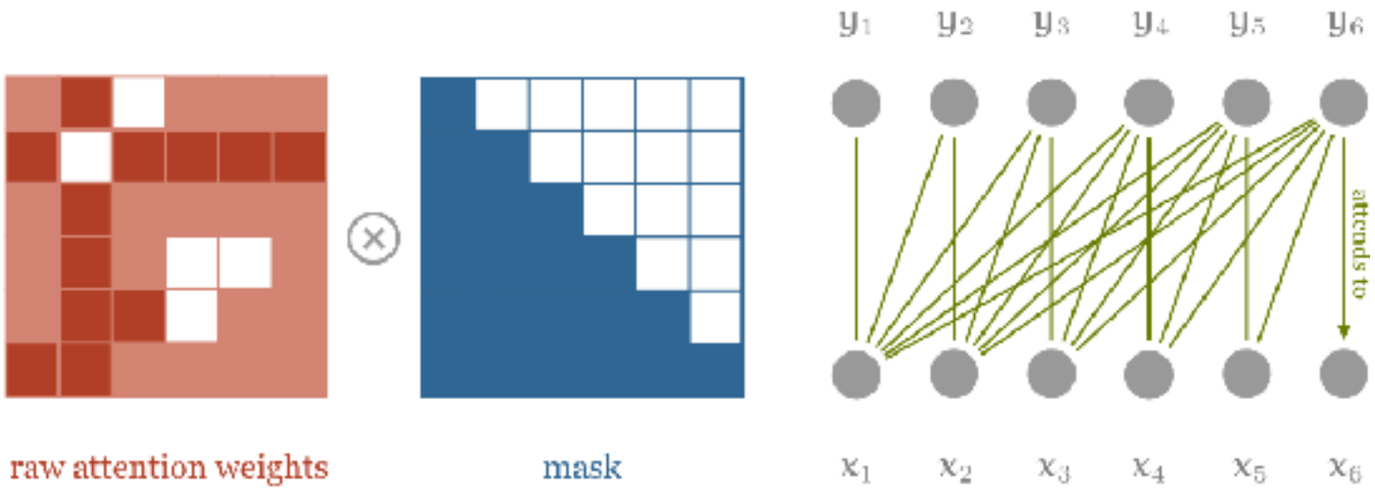
[69310, 23045, 3625, 87892, 758]

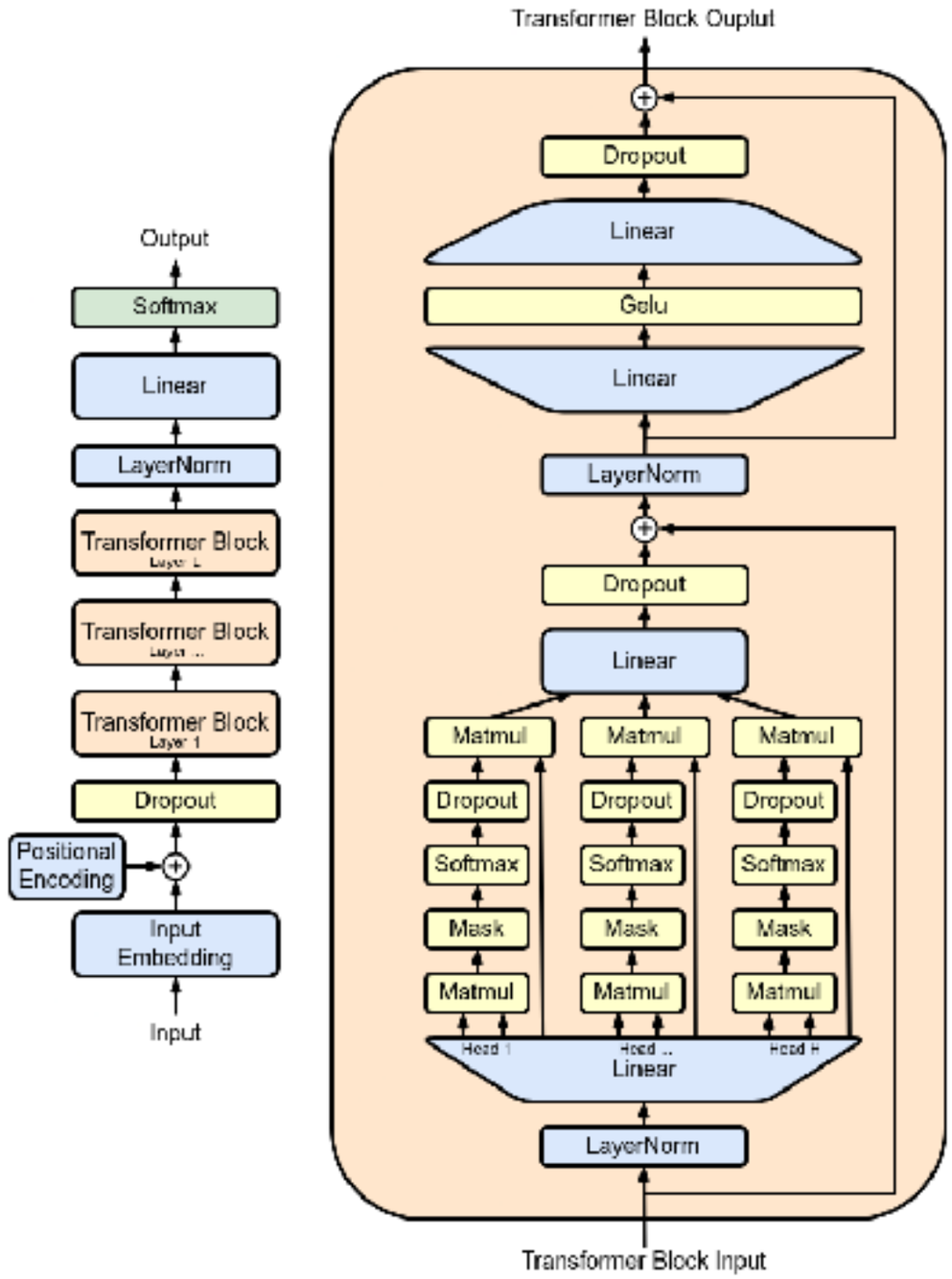
$$\begin{pmatrix} 23.4 \\ 55.2 \\ \dots \\ -128.4 \end{pmatrix} \quad \dots \quad \begin{pmatrix} -66.7 \\ 14.9 \\ \dots \\ 1.7 \end{pmatrix}$$

Token embedding in 12,888 dimensions (GPT3)

Generative Pretrained Transformer (GPT) for NLP

- **Motivation:** AR models are difficult to train **on long context** (recursive back-propagation top generate several symbols) *(see also: LSTM architecture)*
- **Inputs: [BxTxE]** batch of B of time-sequences of T symbols/ tokens of E dimensions with **positional embedding**
- **Principle:** **Attention**-Based **Decoder** architecture *(ie. no encoder + cross-attention as in seq2seq)*
  - Each token is processed independently in **shared MLPs** (here only 2 layers), but Tokens are compared with **multi-head self-attention** (see Part I) with different embeddings for K,Q,V = tokens
- Last layer is a **shared classifier** (linear + softmax) **predicting next token in a causal fashion**
- causality is obtained by design using **masked attention**

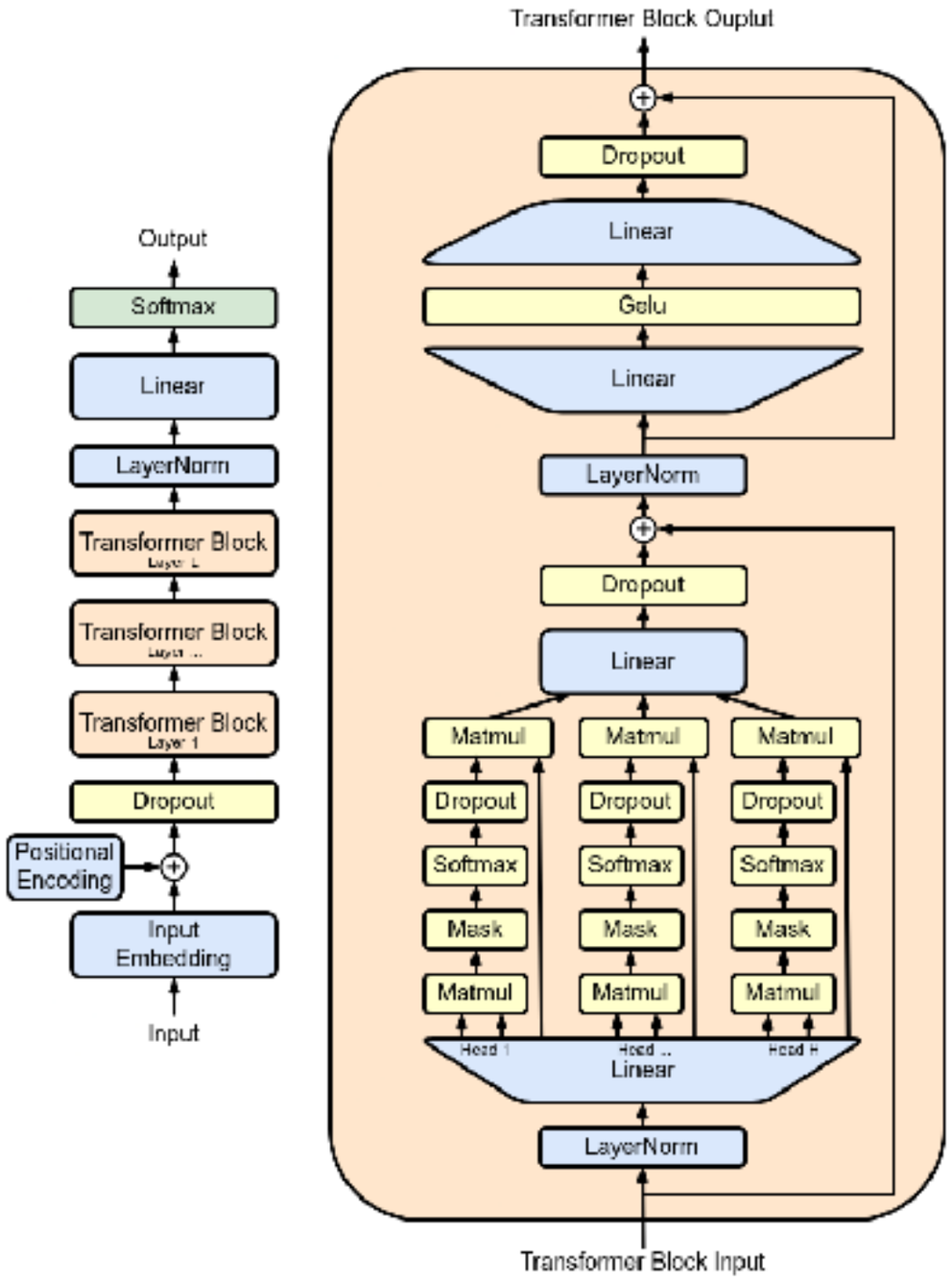




Decoder Transformer Architecture (i.e. w/o encoder)

Generative Pretrained Transformer (GPT) for NLP

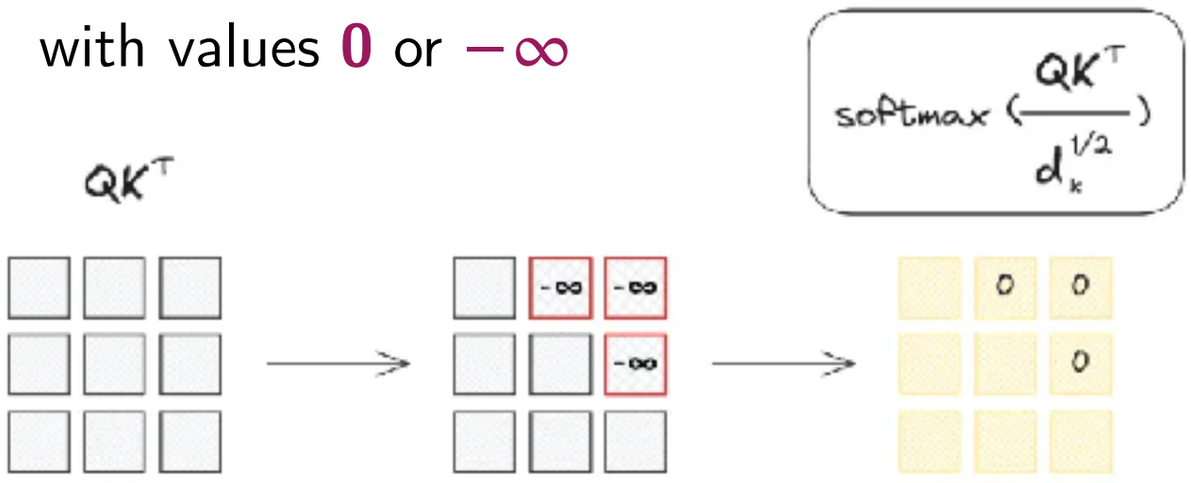
- **Inputs x: [BxTxE]** batch of B of time-sequences of T symbols/tokens of E dimensions with **positional embedding**
- **Training trick:** using **masked attention,** each output y at time positions t predict the probabilities of the next token $t+1$, considering only the sequence $y_t = \mathbb{P}_\theta(x_{t+1} \mid x_1, \ldots x_t)$
  - ➡ computing the loss for the full sequence ***at once*** *is* equivalent to use T different sentences
  - ➡ only one back-propagation is required
- **self-supervised Training:**
  - **ground-truth:** shifted inputs to the left
  - **loss:** conventional **cross-entropy**

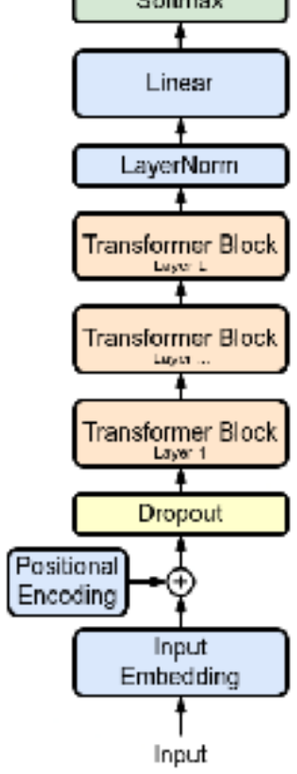


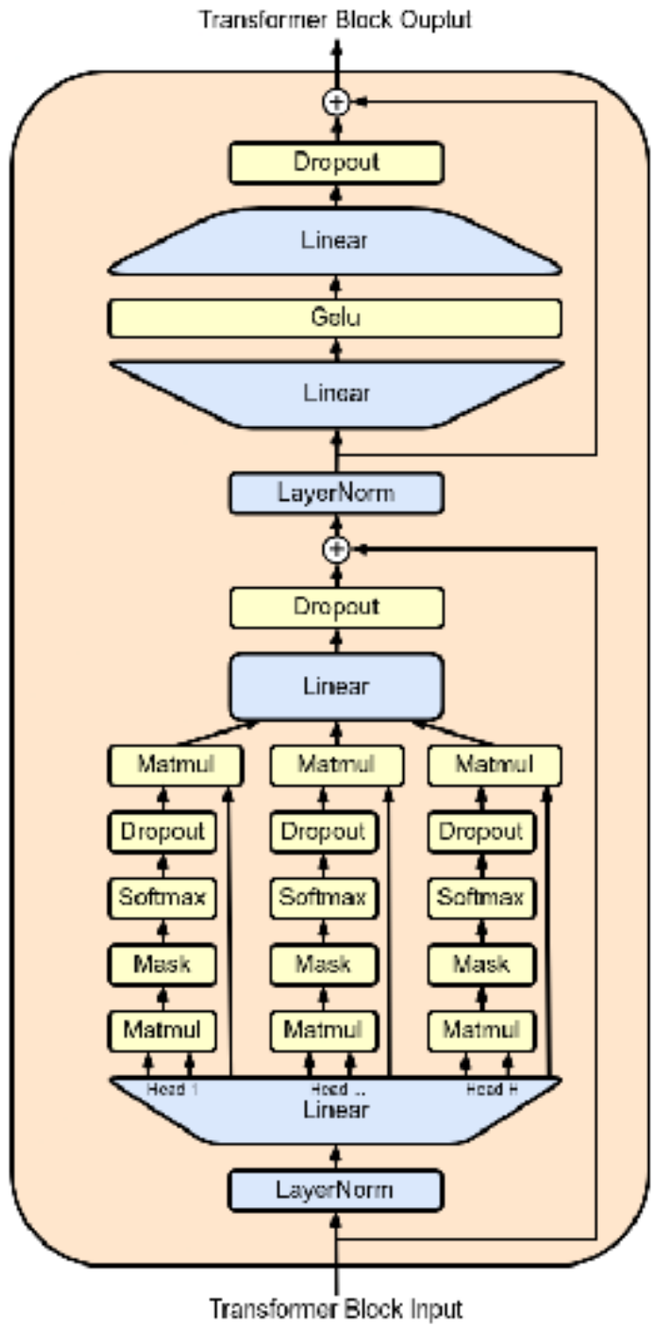
Decoder Transformer Architecture (i.e. w/o encoder)

Lab #9: tiny GPT

- **Illustrations on Jules-Verne Dataset: tiny_GPT.ipynb**
  - **tokens:** alpha-numerical symbols (*e.g.* a,ë,1,!, ,% ...)
  - naïve positional encoding: index position of symbols (*e.g.* 1,2,3,... T) + pytorch nn.embedding layer
  - use of pytorch encoder nn.TransformerEncoderLayer rather than a decoder which takes an additional memory input (from the encoder)
  - causal mask: **additive** mask with values **0** or $-\infty$

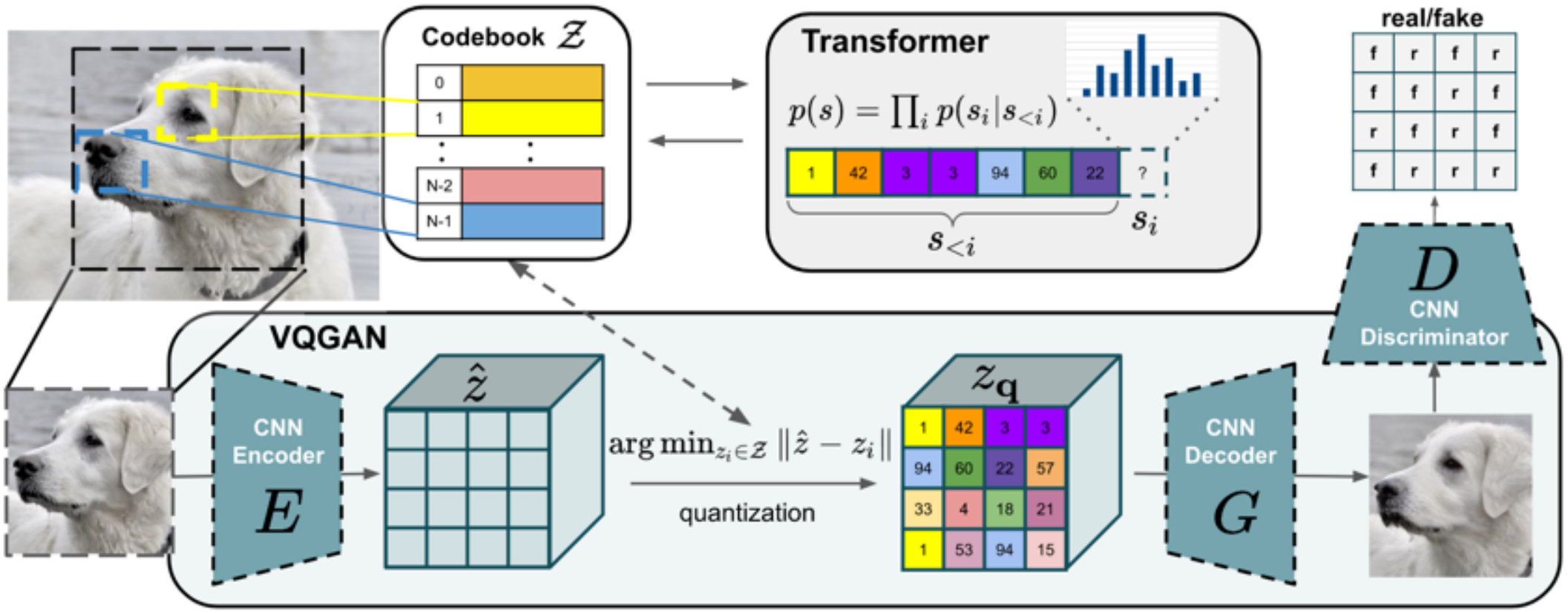






VQ-GAN [Esser'21]

- Like latent diffusion (see next part !), two-stage approach:
  1. train a **VQ-AE with adversarial loss (latent "VQ-GAN" representation)**
  2. train a **latent GPT ("taming transformers")**



- For very large 2D images, use raster scan order with **sliding window** (like pixel RNN)

Vector-Quantization (VQ) for discrete representation

- **VQ - VAE :** (from VQ-VAE2 [Razavi'18])
  train simultaneously an encoder E, a decoder D, and a codebook {e}
  to minimise

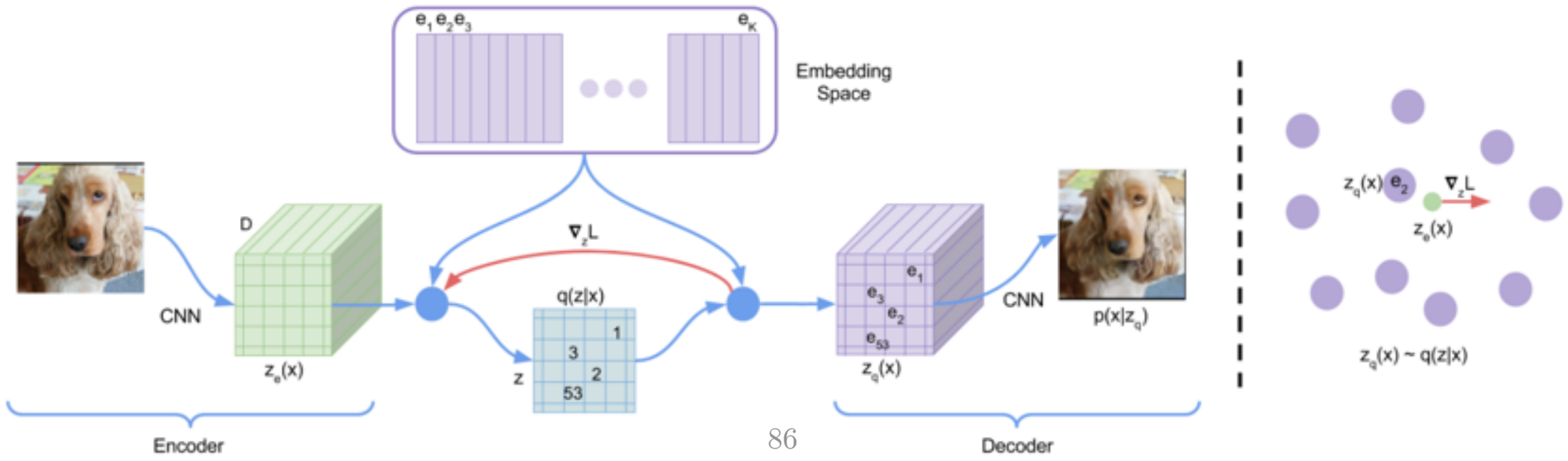
$$\mathcal{L}(x, D(e)) = \|x - D(e)\|^2 + \|sg[E(x)] - e\|^2 + \beta\|sg[e] - E(x)\|^2$$

where the quantized latent representation e of x is the NN

$$\text{Quantize}(E(x)) = e_k \text{ where } k = \arg\min_j \|E(x) - e_j\|$$

and sg indicates 'stop gradient' as argmin operator is not differentiable

- **VQ principle :** (from VQ-VAE [Oord'18])



SOTA AR-Models

Exemples with Face Generation (≈1B parameters)

VQ-VAE 2 [Razavi'19] with VQ-AE + PixelCNN

VQ-GAN [Esser'21] with transformers (GPT)

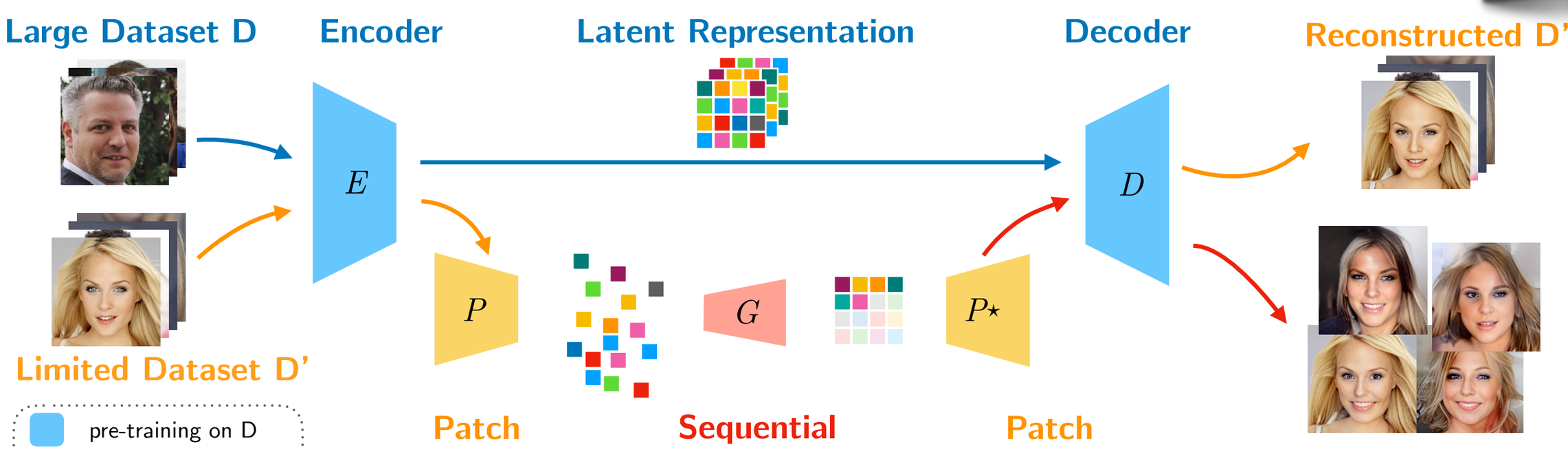
Overfitting in Auto-regressive models

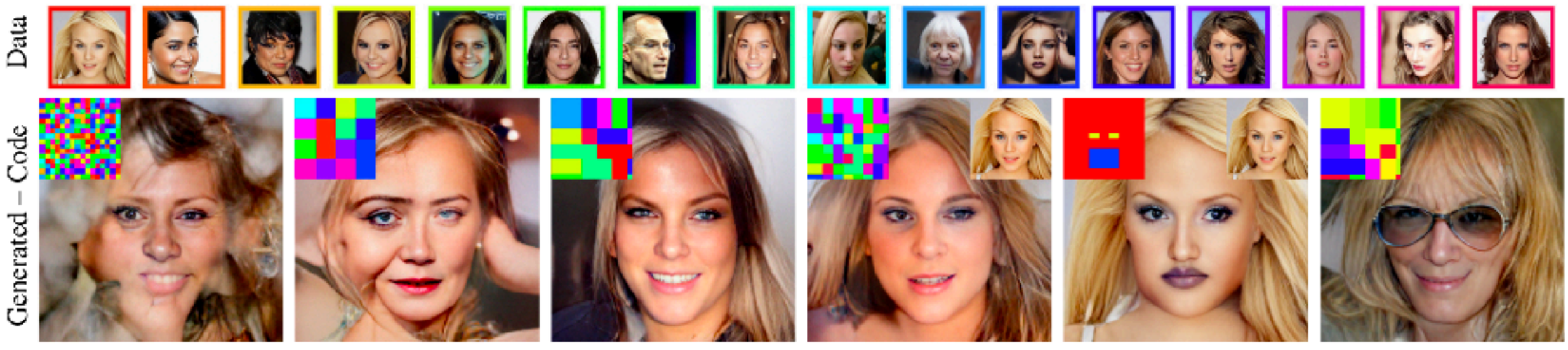
Overfitting with large auto-regressive model (from VQ-GAN [Esser'21])

Generated Nearest Neighbours in validation & training datasets using a perceptual loss (LPIPS)

Application to few shot generative modelling

- **Examples from:** Latent-Patch **[Samuth'23]**





(a) Random Code
1×1 Patch copy

(b) Random Code
4×4 Patch copy

(c) LATENTPATCH
2×2 Patch sequence

(d) Using reference
First image, in red

(e) Face edition
Eyes and mouth

(f) Selected attributes
Woman, Blond, Glasses

Diffusion Models

(simple version)

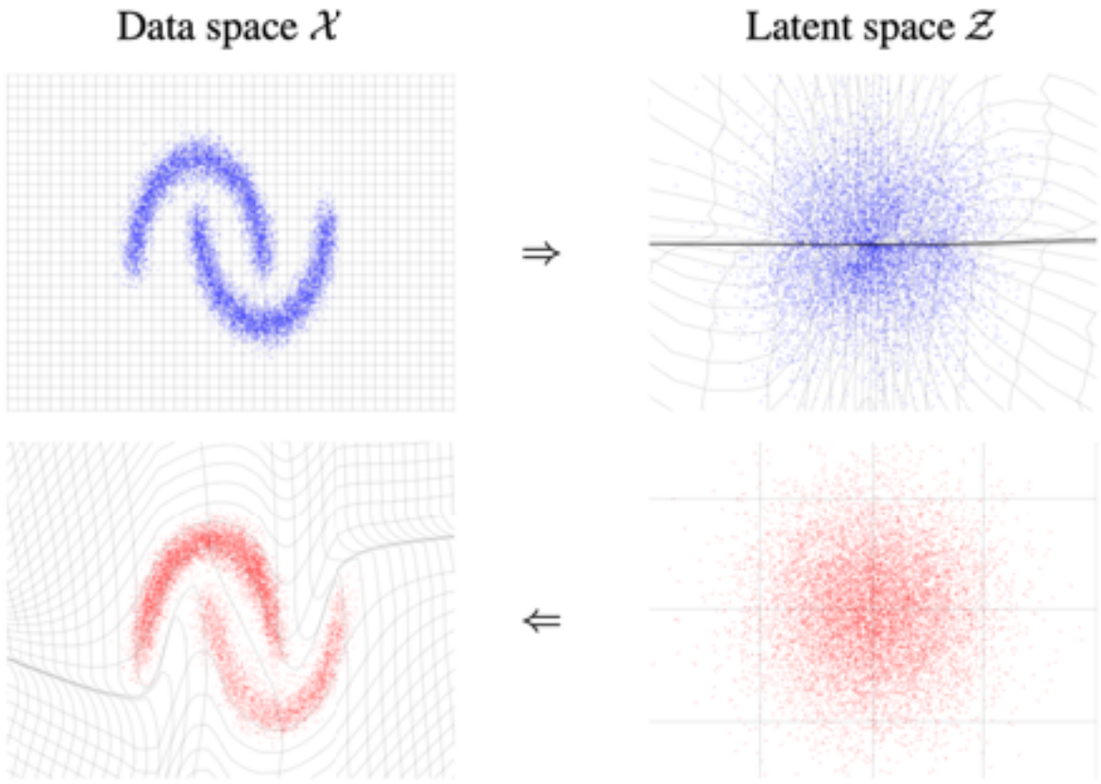
Diffusion Models: Motivation

Motivations

- **Avoids the trade-off in quality** from VAE, due to the joint-training of a **low-dimensional** encoder
- Achieves **SOTA** performance in various domains (Text, Image, Video ...)
- Allows for efficient **conditional generation**

Strategy

- **Design** a simple **high-dimensional encoder** (rather than training)
- focus on training the **generative model** (decoder+prior)



References

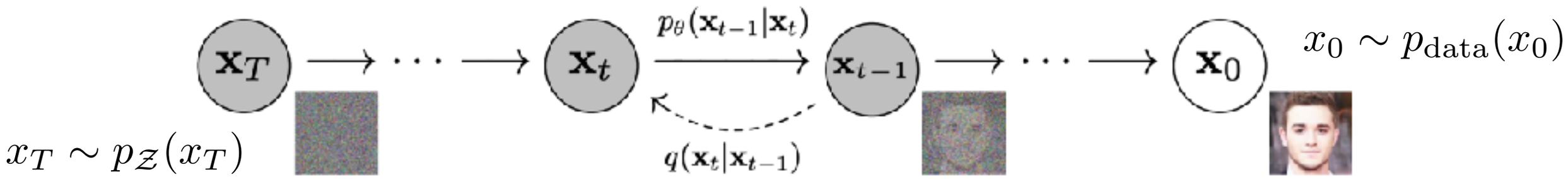
- "Deep Unsupervised Learning using Nonequilibrium Thermodynamics" [Sohl-Dickstein et al., '15]
- **DDPM** [Ho et al., '20], **DDIM** [Song et al., '20], **SDE** [Song et al., '21]

Diffusion Models: Encoder

Key Concepts [Sohl-Dickstein et al., '15]

- **1 - Forward process: Encoding by diffusion**
  - **Encoding:** The latent representation z of a data point $x_0 \in \mathbb{R}^d$ is a (***time***) **sequence of latent representations** $z = x_{1:T} := (x_1, x_2, \ldots, x_t, \ldots x_T)$ where $x_t \in \mathbb{R}^d$

⚠ *'time' does not refer to data dimension !*



  - **Noise Diffusion:** Markov chain model for the posterior $q(x_t \mid x_{0:t-1}) = q(x_t \mid x_{t-1})$ where each latent x_t is obtained by reducing SNR by adding noise $\varepsilon \sim \mathcal{N}(0, I_d)$

$$x_t = a\, x_{t-1} + b\, \varepsilon$$

$a \in (0, 1]$

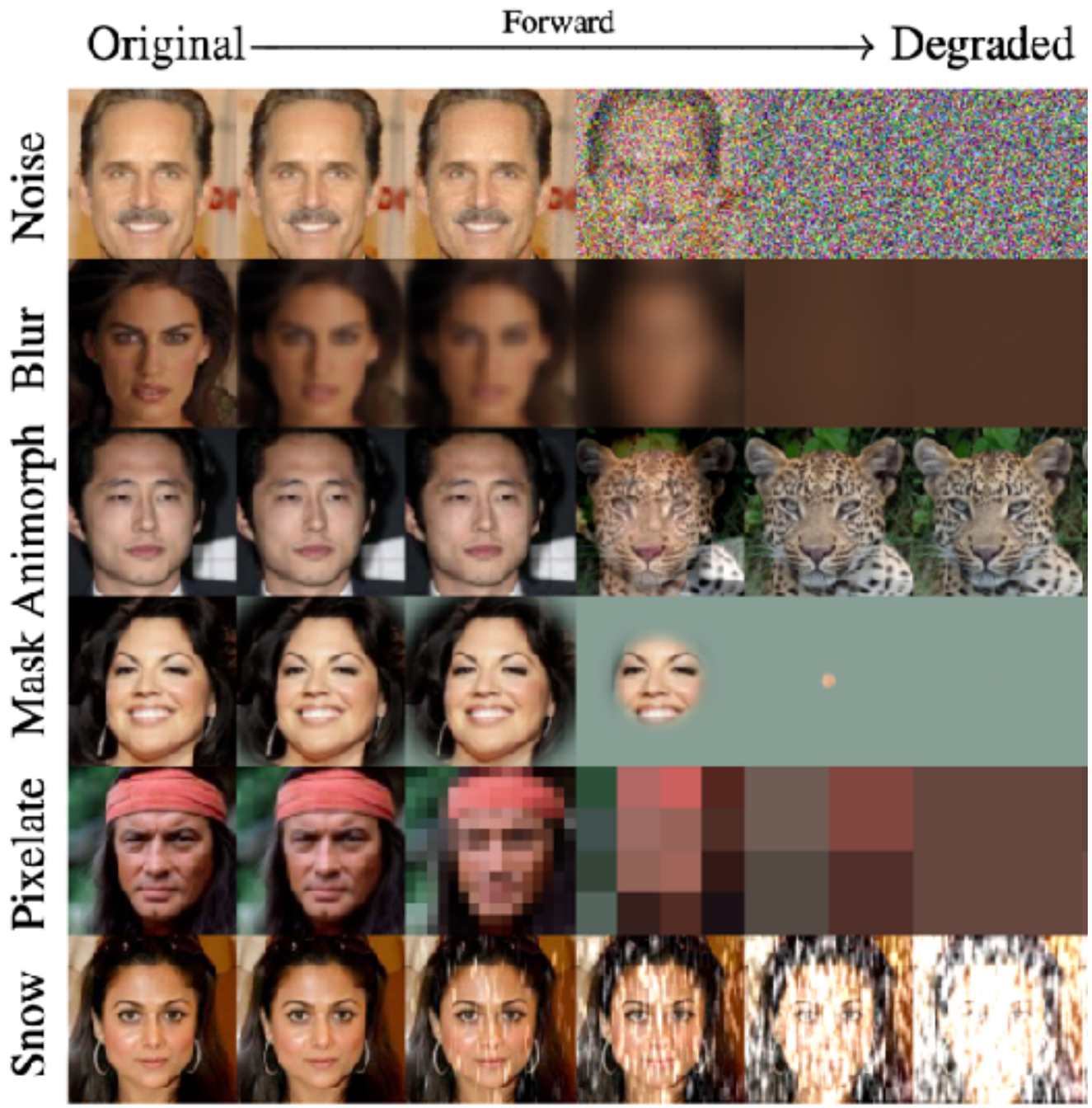
$b > 0$

Diffusion Models: Encoders in littérature

Remark: alternative diffusion process ("Cold Diffusion" [Bansal et al. '22])

- **1 - Forward process: simple Encoding by diffusion**

$x_0 \sim p_{\text{data}}(x_0)$



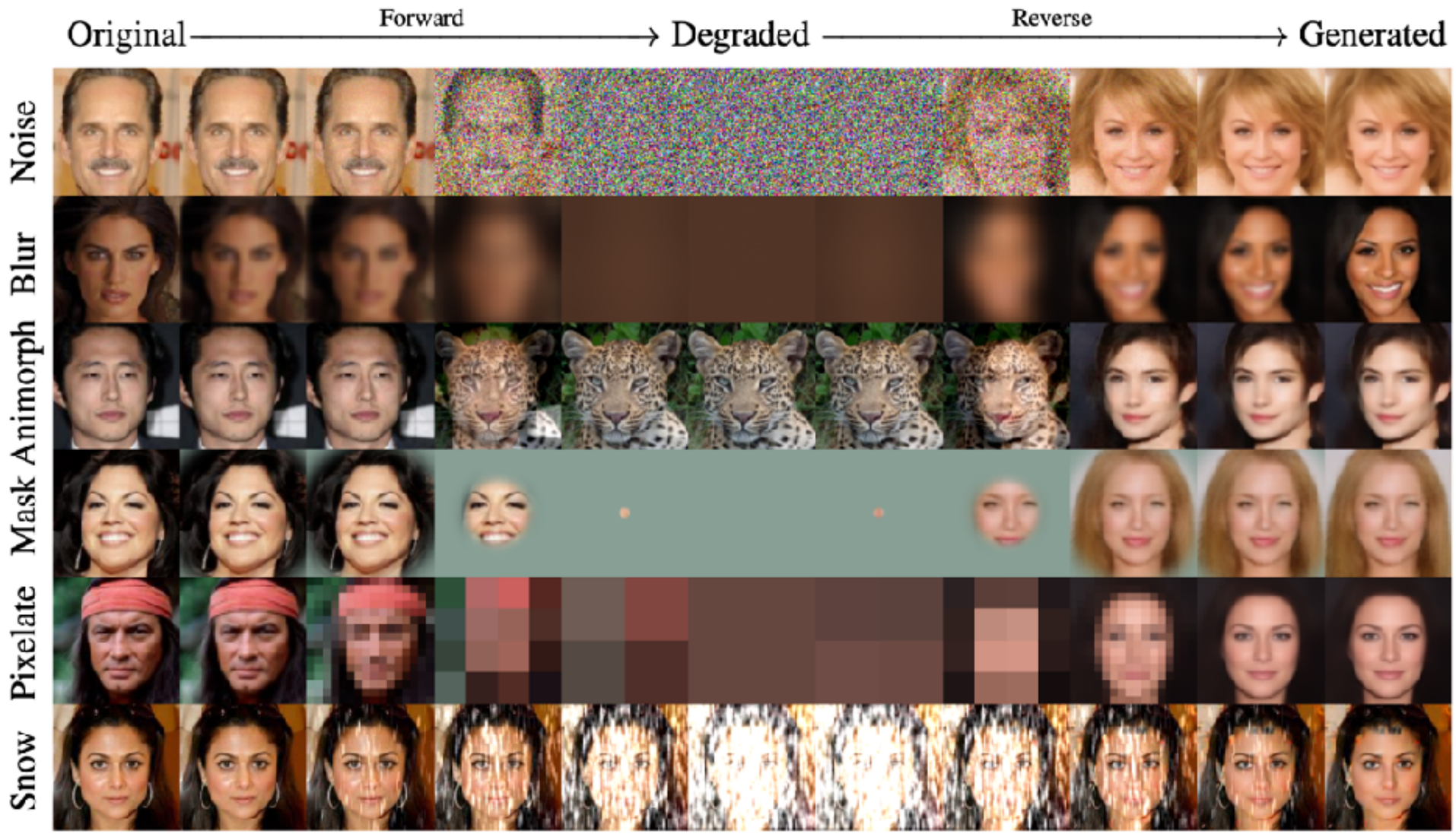
$x_T \sim p_{\mathcal{Z}}(x_T)$

Diffusion Models: reversing the diffusion

Remark: alternative diffusion process ("Cold Diffusion" [Bansal et al. '22])

- **1 - Forward process: decoding by reverse diffusion ?**

$x_0 \sim p_{\text{data}}(x_0)$ $\qquad\qquad$ $x \sim p_{\text{gen}}(x)$

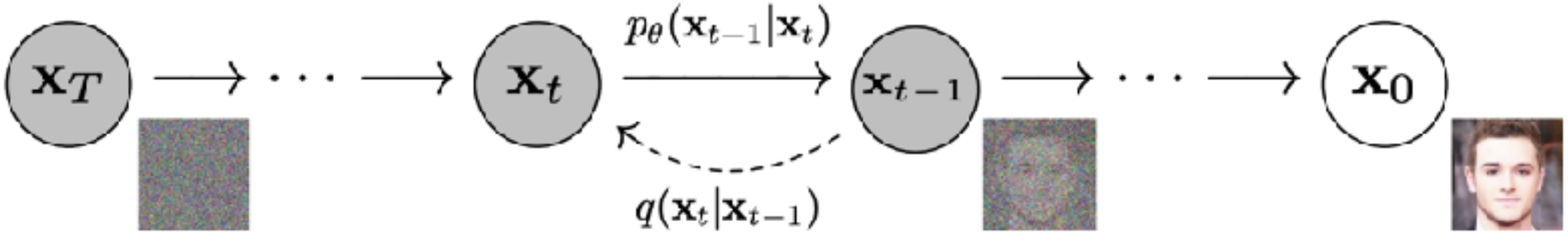


$x_T \sim p_{\mathcal{Z}}(x_T)$ $\qquad$ **How to decode ?**

Diffusion Models: backward diffusion

Key Concepts [Sohl-Dickstein et al., '15]

- **2 - Backward process: Decoding by reversing the diffusion**
  - **Decoding:** we want to recover **sequentially** a signal x from the latent representations x_T using the conditional probability $p(x_{t-1} \mid x_t)$ which is **intractable**



  - For small time-steps it can be **approximated by a multi-variate gaussian**
  - **Training:** like before, we learn a parametric conditional model (decoder)

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_{\theta,t}(x_t), \Sigma_{\theta,t}(x_t))\,(x_{t-1})$$

where $\mu_{\theta,t}, \Sigma_{\theta,t}$ are deep neural networks **depending on time** t

- $p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_{\theta,t}(x_t), \Sigma_{\theta,t}(x_t))\,(x_{t-1})$

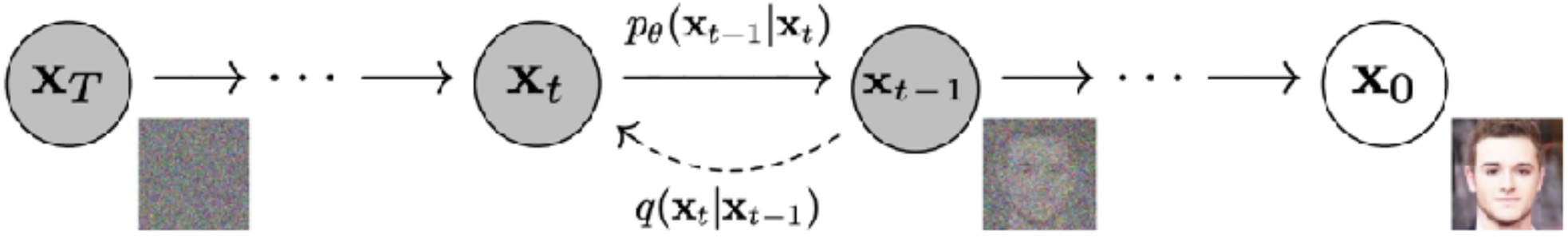
Diffusion Models: backward diffusion

Key Concepts [Sohl-Dickstein et al., '15]

- **2 - Backward process: Decoding by reversing the diffusion**
  - **Denoising:** sampling $p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_{\theta,t}(x_t), \Sigma_{\theta,t}(x_t))\,(x_{t-1})$

    is equivalent to: $x_{t-1} = \mu_{\theta,t}(x_t) + R_{\theta,t}(x_t)\varepsilon, \text{ where } \varepsilon \sim \mathcal{N}(0, I_d)$

    $R_{\theta,t}R_{\theta,t}^T = \Sigma_{\theta,t}$



$\Sigma_{\theta,t} = \sigma_t^2 I_d$

- **Spherical gaussian simplification**: considering a diagonal covariance, we get a similar generator than VAE where the *decoder only predicts the mean of the output distribution*

$$x_{t-1} = \underbrace{\mu_{\theta,t}(x_t)} + \sigma_t\varepsilon, \text{ where } \varepsilon \sim \mathcal{N}(0, I_d)$$

how to train such NN ?

Diffusion Models vs Stochastic Differential Equation

Related Approaches

- **Diffusion** [Sohl-Dickstein et al., '15], **DDPM** [Ho et al., '19], **DDIM** [J. Song et al., '20]
- **Score Matching with Langevin dynamics process** [Y. Song et al., '21]
- **Variational Diffusion Models** [Kingma et al., '23]
- **Flow matching** [Lipman et al., '24] ...

SDE

$$dx(t) = f(x(t), t)dt + g(t)dW(t)$$

- **A unifying view:** *discrete* or *continuous formulations* of a **stochastic differential equation** (SDE)
- **... with some differences:** forward model (e.g. variance preserving/exploding schedule), sampling, loss formulation ...
- **Implication:** SDE interpretation useful for
  - **reformulation for inverse problems** where the generative model is used as a prior
  - **more efficient (faster) sampling ODE scheme for generation** (Euler, Heun, Runge-Kutta ...)

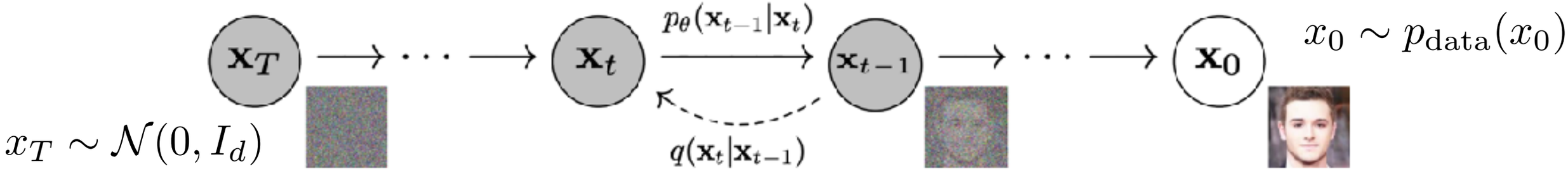
Denoising Diffusion Probabilistic Models

Training DDPM in a nutshell [Ho et al., '20]

- **1 - Variance preserving** diffusion and **direct sampling** at time t:

$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1-\alpha_t}\varepsilon \quad \Rightarrow \quad x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon \qquad \text{with} \qquad \bar{\alpha}_t = \Pi_{1 \leq s \leq t} \alpha_s$$

- **2 - Backward diffusion: Decoding by a denoising network** ε_θ



- **Strategy:** we want to recover x_0 by estimating unknown noise ε using **a network** ε_θ

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\varepsilon) \qquad \text{with} \qquad \hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\varepsilon_\theta(x_t, t))$$

- **Training:** like AE, using MSE between x_0 and $\hat{x}_0$ to train the denoising network yields

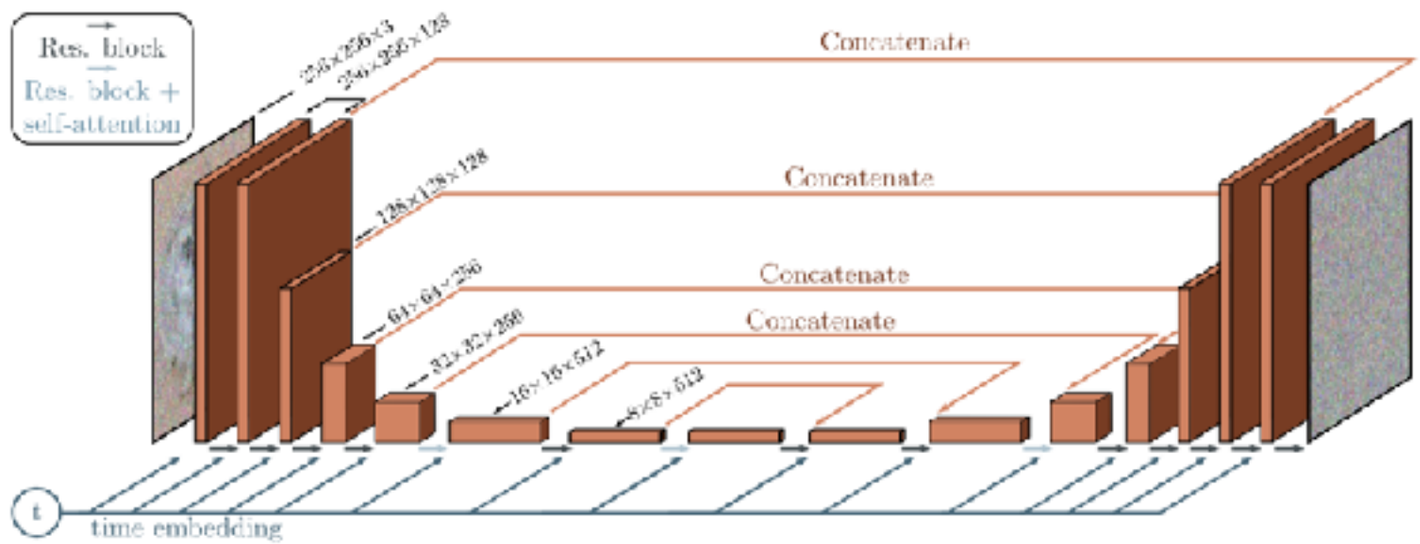
$$\min_\theta \mathbb{E}_{x_0,\varepsilon,t} \|\varepsilon - \varepsilon_\theta(x_t, t)\|^2$$

Denoising Diffusion Probabilistic Models

Training DDPM [Ho et al., '20]

- **Same denoising network** ε_θ used for all time steps but conditioned on time t
  In practice, latent x_t and time t are concatenated using time embedding

source: S.J.D. Prince



Example of a U-Net denoiser architecture using time embedding.

- **How the training relates to negative log-likelihood (NLL) minimisation ?**

$$\min_\theta \mathbb{E}_{x_0,\varepsilon,t} \|\varepsilon - \varepsilon_\theta(x_t, t)\|^2$$

Corresponds to (up to normalising factors) a **lower-bound on the log-likelihood (ELBo)**

- **What about sampling the generative model ?**

Denoising Diffusion Probabilistic Models

Sampling DDPM [Ho et al., '20]

- **Stochastic** sampling: from t=T to t=1, starting from $x_T \sim \mathcal{N}(0, I_d)$

Recall that the decoder parametrizes the gaussian conditional probability $p_\theta(x_{t-1} \ x_t)$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\big(\mu_{\theta,t}(x_t), \sigma_t^2 I_d\big)\,(x_{t-1})$$

This yields

$$x_{t-1} = \mu_{\theta,t}(x_t) + \sigma_t \varepsilon, \text{ where } \varepsilon \sim \mathcal{N}(0, I_d)$$

Based on ELBo minimisation, authors show that the expectation $\mu_{\theta,t}(x_t)$ writes

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(x_t, t)\right) + \sigma_t \varepsilon$$

where $\varepsilon_\theta(x_t, t)$ is the trained denoising network

Sampling DDIM [Song et al., '20]

- **Faster Deterministic** sampling: using the estimated noise $\varepsilon_\theta(x_t, t)$ rather than ε for the next time step

$$\begin{aligned} x_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}\hat{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\varepsilon_\theta(x_t, t) \\ &= \sqrt{\bar{\alpha}_{t-1}}\frac{1}{\sqrt{\bar{\alpha}_t}}\left(x_t - \sqrt{1-\bar{\alpha}_t}\varepsilon_\theta(x_t, t)\right) + \sqrt{1-\bar{\alpha}_{t-1}}\varepsilon_\theta(x_t, t) \end{aligned}$$

Denoising Diffusion Probabilistic Models

Advantages

- **high quality** sampling
- **stability** of training (analogous to auto-encoder)
- **tighter lower-bound** on data likelihood

DDPM with CelebA-HQ [Ho et al., '20]

Limitations and Challenges

- Iterative sampling process is **slow** (typically T=1000 steps with DDPM)
- latent representation is **large** (larger than the data points !)
- no deterministic latent representation of the data (**encoder is stochastic**)