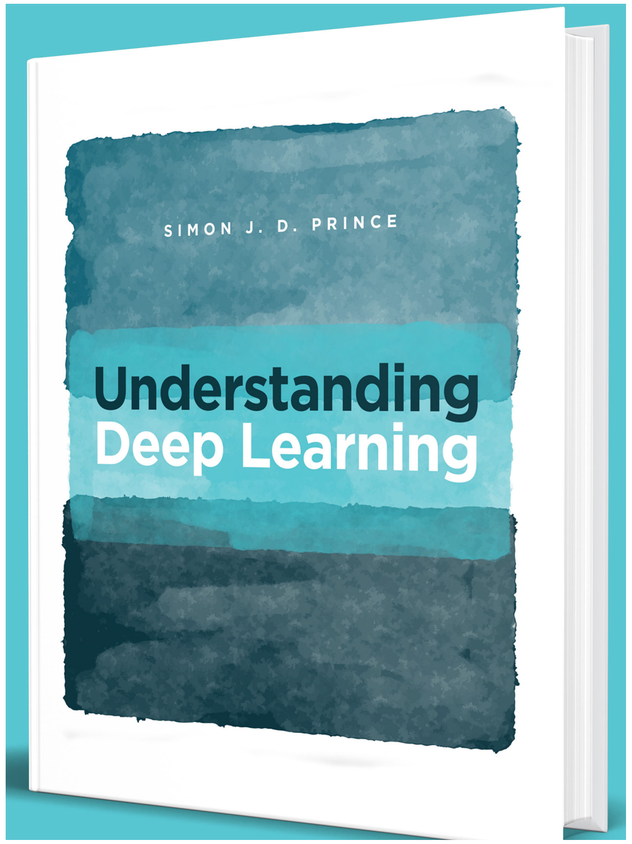
Lecture 10: Generative Models

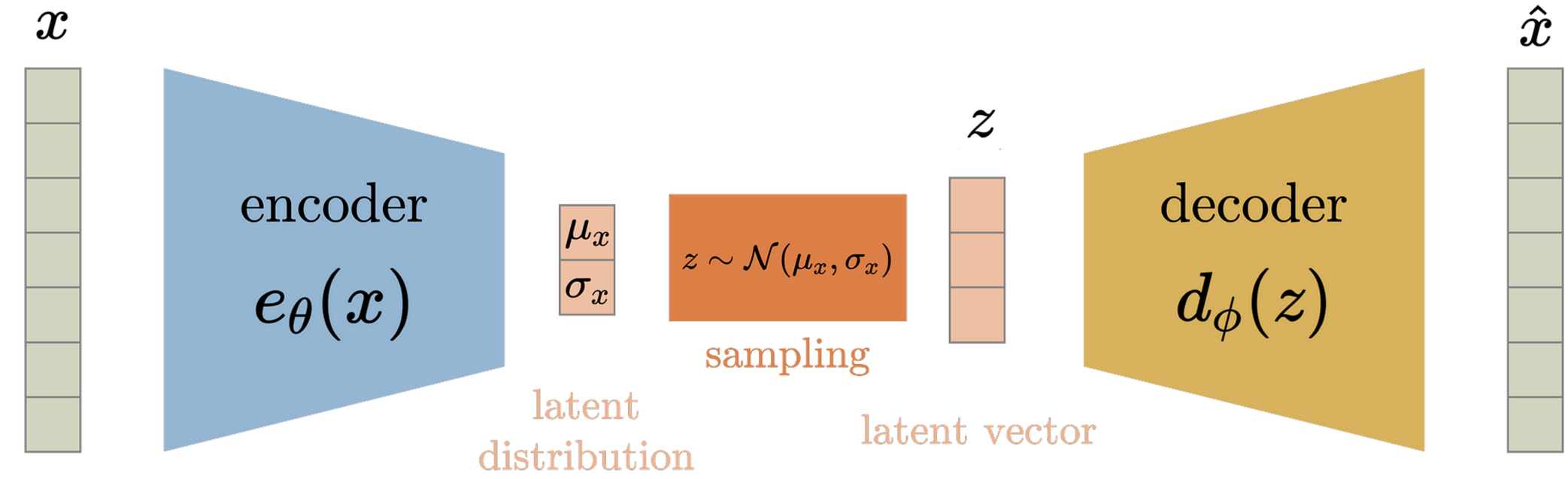
Maurizio Pierini (CERN), Thea Aarrestad (ETH)

Program for Today



Date	Topic	Tutorial
Sep 16	Intro & class description	
Sep 23	probability Theory + Basic of machine learning + Dense NN	*Basic jupyter + DNN on mnist (give jet dnn as homework)*
Sep 30	Statistics + Convolutional NN	*Convolutional NNs with MNIST*
Oct 7	Training in practice: regularization, optimization, etc	*Free Practice*
Oct 14	Unsupervised learning and anomaly detection	*Autoencoders with MNIST*
Oct 21	Graph NNs	*Tutorial on Graph NNs*
Oct 28	Transformers	
Nov 4	Network compression (pruning, quantization, Knoweldge Distillation)	
Nov 11	Tutorial on hls4ml	
→ Nov 18	Generative models: GANs, VAEs, etc	*GAN and VAE*
Nov 25	Reinforcement Learning	
Dec 2	Normalizing flows	*TBD*
Dec 9	Quantum Machine Learning	
Dec 16	**Exams To Be Confirmed**	

The VAE loss decrypted



$$\text{reconstruction loss} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2$$

$$\mu_x, \sigma_x = e_\theta(x), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$similarity\ loss = KL\ Divergence = D_{KL}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

$$loss = reconstruction\ loss + similarity\ loss$$

Shannon Entropy and Kullback-Leibler Divergence

- Shannon Entropy: $\mathcal{H}[p(x)] = -\mathbb{E}[log(p(x))] = -\int dx p(x) \log p(x)$ (which becomes $\mathcal{H}[p(x)] = -\sum p(x) \log p(x)$ when approximating the integral by a sum, as in our case)
  - When the log is in base 2, Shannon Entropy has units of bits and is a measurement of amount of information
- Kullaback-Leibler divergence $D_{KL}(p(x), q(x))$: number of bits required to transform q(x) into p(x)

$$D_{KL}(p(x), q(x)) = \int dx\ p(x) \log \frac{p(x)}{q(x)}$$

- From the definition

$$D_{KL}(p(x), q(x)) = \mathcal{H}_p(q) - \mathcal{H}_p(p)$$

- which explains why $D_{KL}(p(x), q(x))$ is the relative Shannon entropy between p(x) and q(x)

Shannon Entropy and Kullback-Leibler Divergence

- *The Kullaback-Leibler divergence is not a distance:*
  - $D_{KL}(p(x), q(x)) \neq D_{KL}(q(x), p(x))$
- *Jensen-Shannon divergence is a distance metric derived from the KL divergence*
  - $D_{JS}(p(x), q(x)) = \frac{D_{KL}(p(x), q(x)) + D_{KL}(q(x), p(x))}{2}$

ELBO and VAE loss

- Normally, we would train the VAE minimizing $-\log p(x)$, or maximizing $\log p(x)$, where we want to enforce the dependence on some z with some prior
  - We don't know the prior and we want to approximate it via the encoder $z \sim q(x)$
- An effective training strategy consists in maximizing a function $\mathcal{L}(q) \leq \log p(x)$ (q being the encoder function $q(z|x)$). Doing so, p(x) will also be pushed to its maximum
- In particular we choose

$$\mathcal{L}(q) = \int dz q(z|x) \log p(z,x) + \mathcal{H}(q(z|x)) = \int dz q(z|x)[\log p(z|x)p(x)] - \int dz q(z|x) \log q(z|x)$$

$$= \int dz q(z|x)[\log p(x)] + \int dz q(z|x) \log \frac{p(z|x)}{q(z|x)} = \log p(x) - \int dz q(z|x) \log \frac{q(z|x)}{p(z|x)} =$$

$$\log p(x) - D_{KL}(q(z|x), p_{model}(z|x)) \leq \log p(x)$$

- Since D_{KL} is positive defined, the last inequality holds

ELBO and VAE loss

- To understanding what this ELBO is doing, we should write the loss differently, keeping in mind that we have a discrete set of z distributed as $z \sim q(z|x)$ and that $\int dz q(z|x) f(z) = E_{z\sim q(z|x)} f(z)$

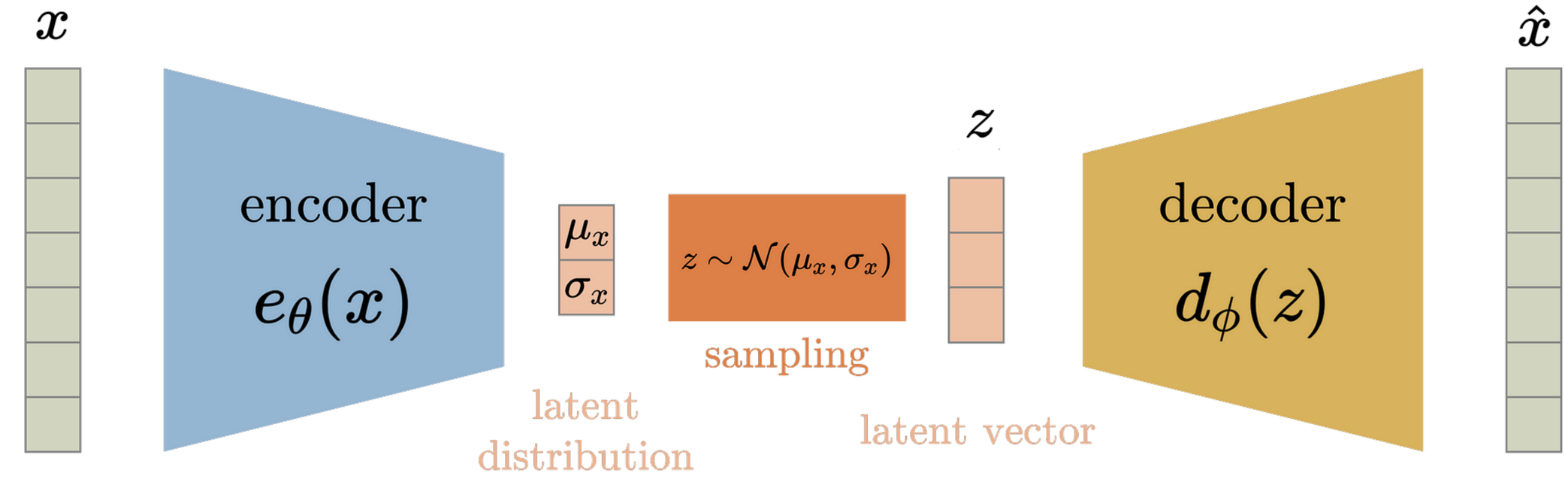
$$\mathscr{L}(q) = \mathbb{E}_{z\sim q(z|x)} \log p(z,x) + \mathscr{H}(q(z|x)) = \mathbb{E}_{z\sim q(z|x)}[\log p(x|z)p(z)] - \mathbb{E}_{z\sim q(z|x)} \log q(z|x)$$

$$= \mathbb{E}_{z\sim q(z|x)} \log p(x|z) + \mathbb{E}_{z\sim q(z|x)} \log p(z) - \mathbb{E}_{z\sim q(z|x)} \log q(z|x)$$

$$= \mathbb{E}_{z\sim q(z|x)} \log p(x|z) + \mathbb{E}_{z\sim q(z|x)} \log \frac{p(z)}{q(z|x)} = \mathbb{E}_{z\sim q(z|x)} \log p(x|z) - D_{KL}(q(z|x), p(z))$$

- This looks like the sparse autoencoder loss, where now the second term is forcing the approximate prior $q(z|x)$ to be as close as possible to some specific prior choice $p_{model}(z)$
- Usually, one uses a Gaussian for $p_{model}(z)$, so that the KL divergence is differentiable
- Usually, one puts a hyperparameter β in front of the second term. This is called β–VAE

The VAE loss decrypted



$$\text{reconstruction loss} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2$$

$$\mu_x, \sigma_x = e_\theta(x), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$similarity\ loss = KL\ Divergence = D_{KL}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

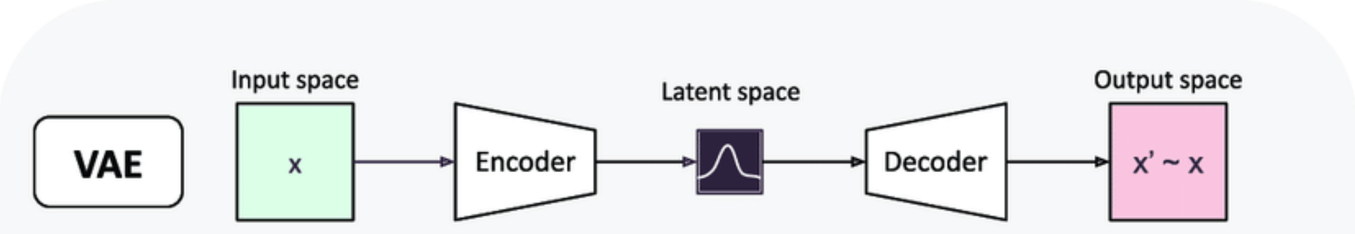
$$loss = reconstruction\ loss + similarity\ loss$$

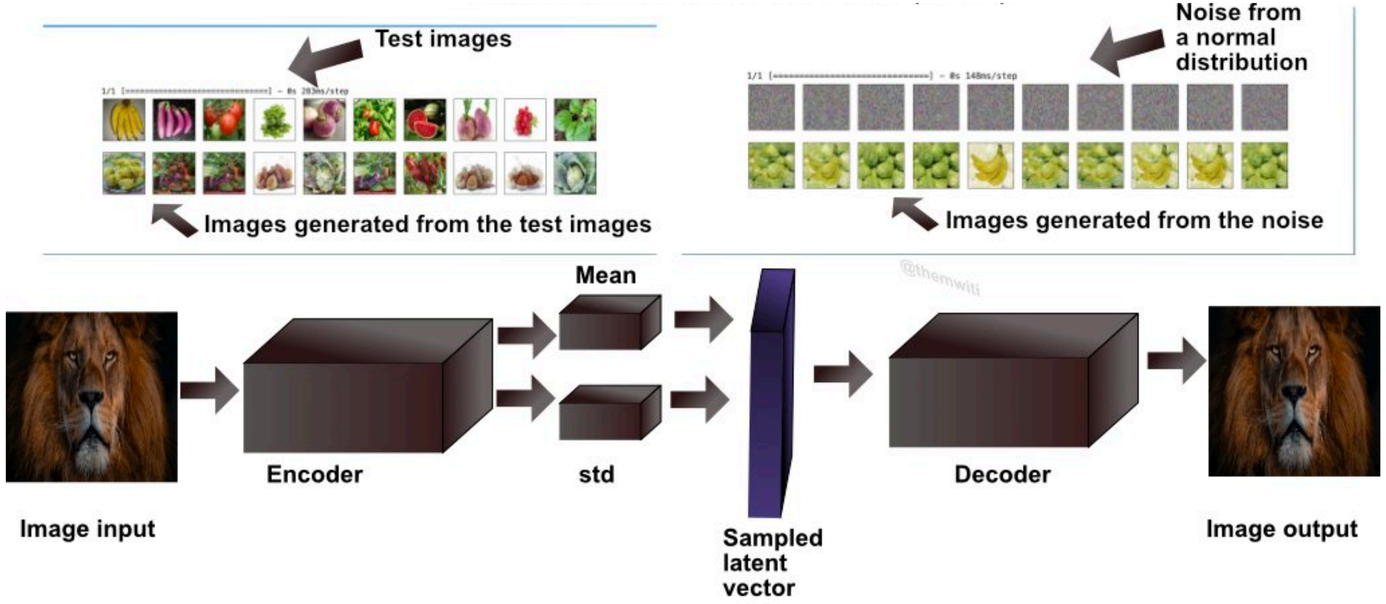
What the VAE loss does

- The reconstruction term ensures that the decoded data are close to the input (as for the plain AE)
  - We have a metric of distance (the loss) that can be used for Anomaly Detection
- The similarity term ensures that the convergence of the training prefers a solution in which the latent variables are distributed ~ Gaussian
  - We can then sample from a Gaussian and obtain new examples, similar to the input data
    - The VAE is a generative model

VAEs as Generative Models

- *In inference, the VAE returns an output starting from a randomly sampled Gaussian vector*
  - *The inference is not deterministic: for a given input one can have various outputs*
- *One can then use the decoder of a VAE as a generative algorithm and creates new examples*



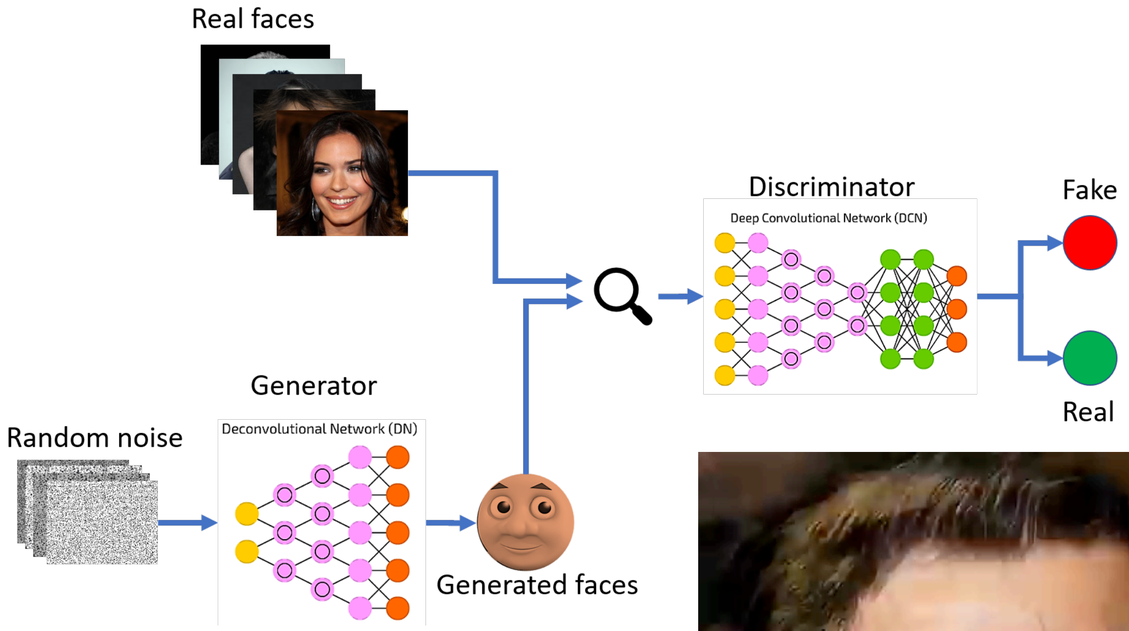


VAEs as Generative Models

- *Passing all information through the bottle neck is complicated*
  - *This typically results in blurred images for convolutional VAE*
- *This is why VAEs to be used for generation are usually designed to be over complete*
- *Also, one plays with β to enhance the generating capability over the reconstructing capability*
  - *Which makes them great for generation, but bad for anomaly detection*
- *We will see next week that there are other options for generative models*

Generative Adversarial Networks

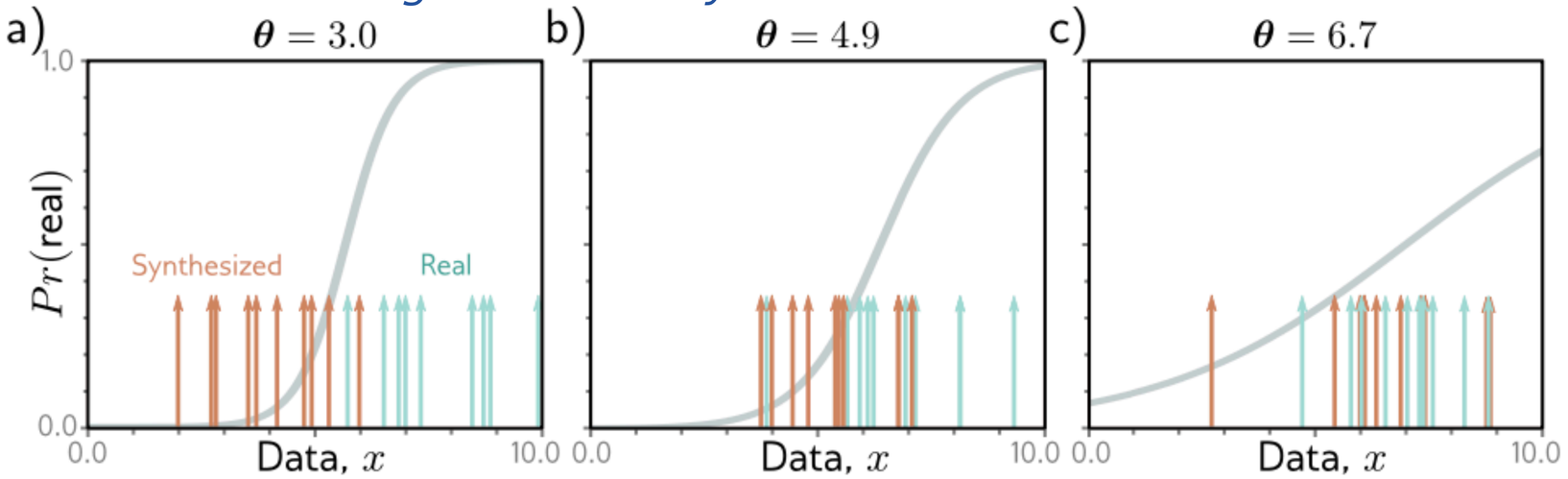
- A GAN is a network that aims at creating new samples, undistinguishable from those contained in a given reference dataset
- GANs are trained through a two-network construction
  - A Generator: starts from random noise and generates the candidate new examples
  - A discriminator: receives an example (real or from generator, we will call it “fake”) and has to identify which are the real ones



https://thispersondoesnotexist.com/

Generative Adversarial Networks

- *The generator:*
  - *Starts from some latent space z*
  - *Creates the data x applying a network on z*
  - *The generator is a map from the space of z to that of x*
- *The discriminator:*
  - *A traditional classifier*
- *The GAN construction defines similarity between generated and real data through their statistical indistinguishability*



The GAN loss function

We start from the binary cross entropy of the discriminator

$$\mathscr{L} = \sum_i -(1-y_i)\log(1-\hat{y}_i) - y_i \log(\hat{y}_i)$$

where y_i is the ground-truth label and $\hat{y}_i = D(x_i|\phi)$, ϕ indicating the network parameters we want to determine

- We look for

$$\hat{\phi} = \text{argmin}_\phi \sum_i -\log(1-D(x^*|\phi)) - \log(D(x|\phi))$$

fixing $y_i = 1$ for real examples (labelled x), 0 for the fake ones (labelled as x^*)

- We now explicitly impose that x^* come from the generator $x^* = G(z|\theta)$ and we fix θ so that G maximizes confusion (i.e., increases the loss) of the discriminator

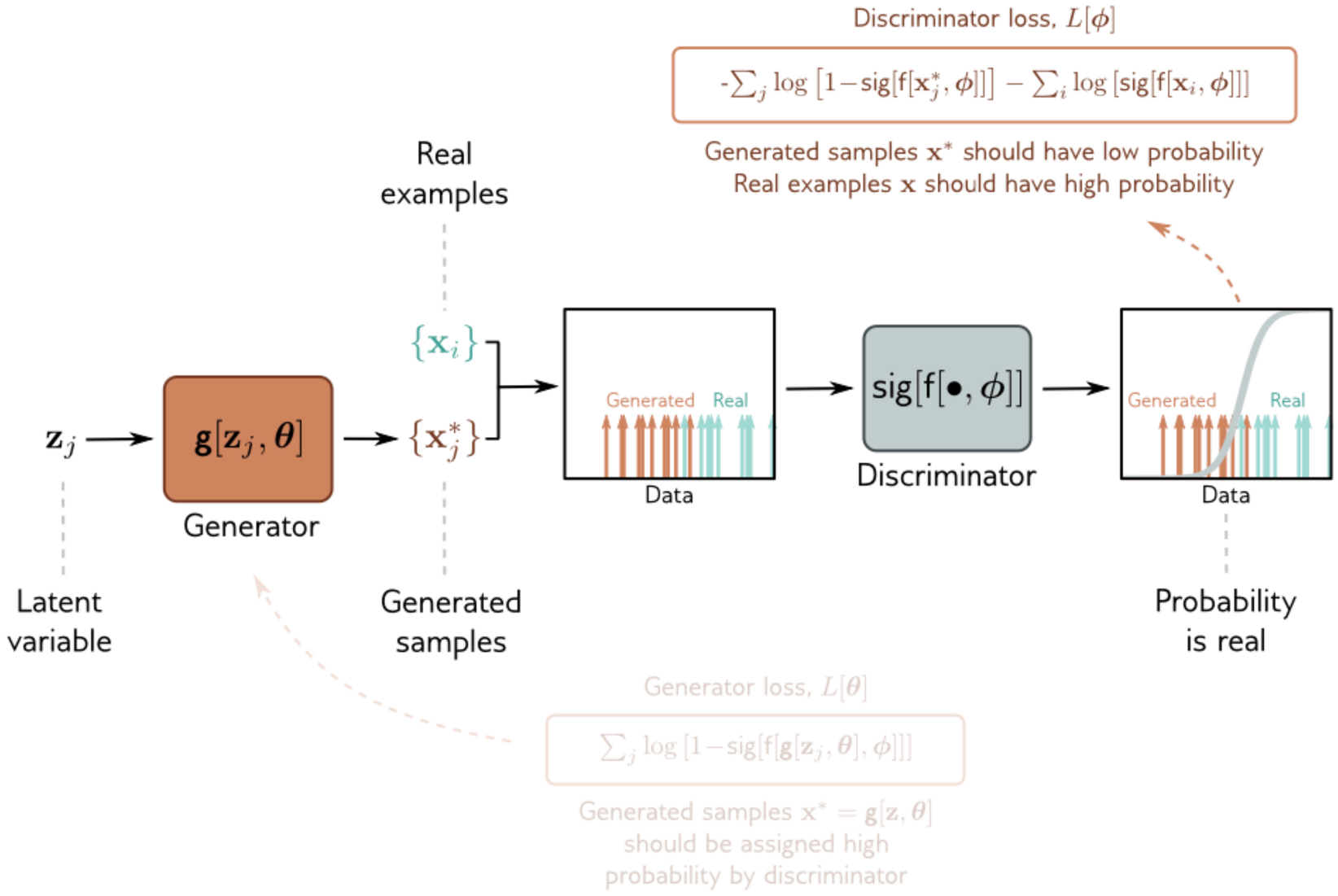
$$\hat{\phi} = \text{argmax}_\theta \left[\text{argmin}_\phi \sum_i -\log(1-D(G(z|\theta)|\phi)) - \log(D(x|\phi)) \right]$$

The GAN loss function

- More complicated than what we saw so far
  - MinMax game converging to a so-called Nash equilibrium: a solution which is the maximum of a function and the minimum of another one
  - Training is more delicate (and unstable)
  - It requires a specific procedure to handle this double task on two networks
- To define such a procedure, we define two loss functions to be minimized (notice the minus sign on $\mathscr{L}_\theta$)
  - $\mathscr{L}_\theta = \sum_i \log(1 - D(G(z|\theta)|\phi))$ Trains the Generator
  - $\mathscr{L}_\phi = \sum_i -\log(1 - D(G(z|\theta)|\phi)) - \log(D(x|\phi))$ Trains the Discriminator

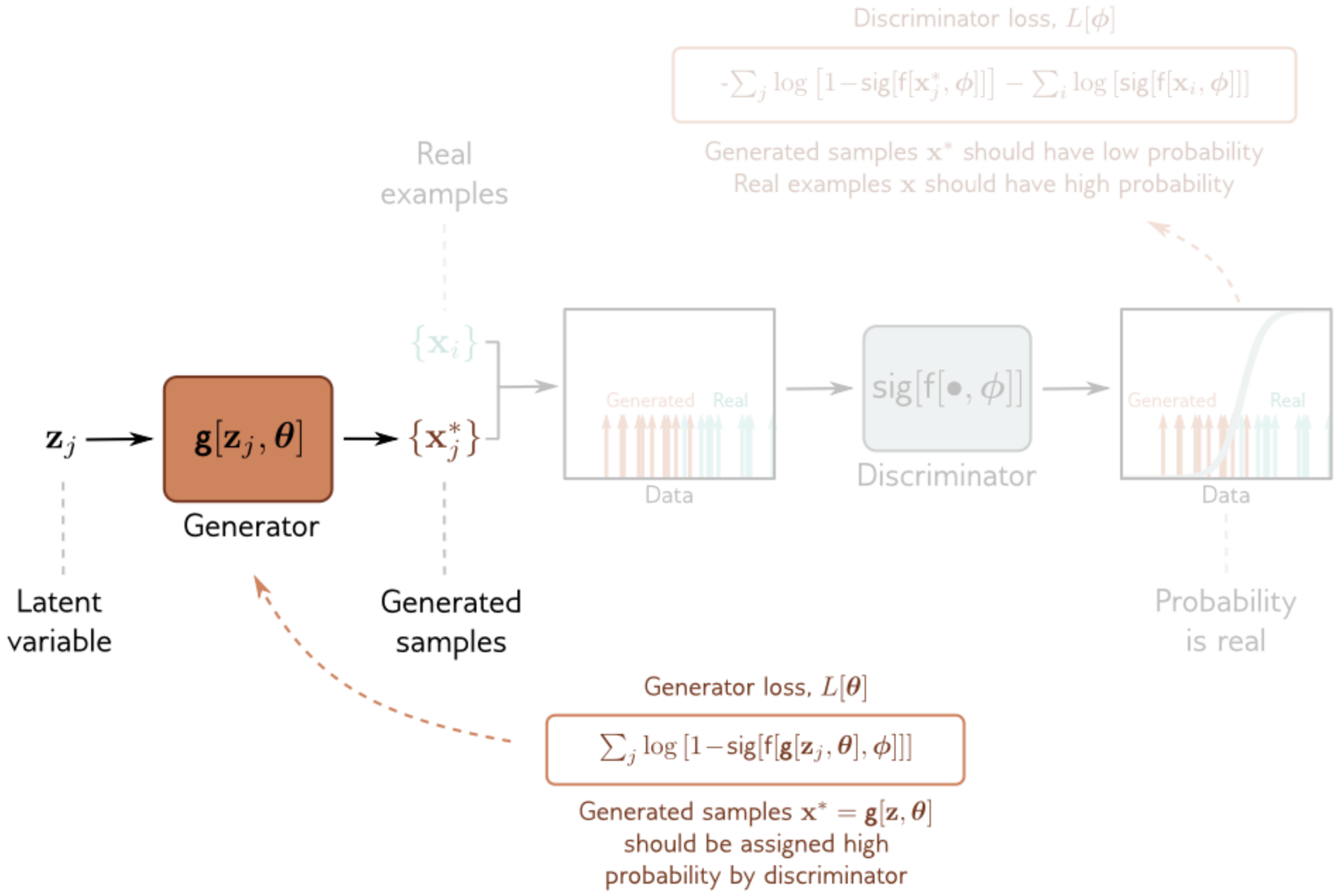
Training a GAN

- *Step 1: for a given batch*
  - *Release the Discriminator parameters*
  - *Freeze the Generator parameters*
  - *Train the Discriminator minimizing $\mathscr{L}_\phi$*



Training a GAN

- *Step 2: for a given batch*
  - *Freeze the Discriminator parameters*
  - *Release the Generator parameters*
  - *Train the Generator minimizing* $\mathscr{L}_\theta$



Training a GAN

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

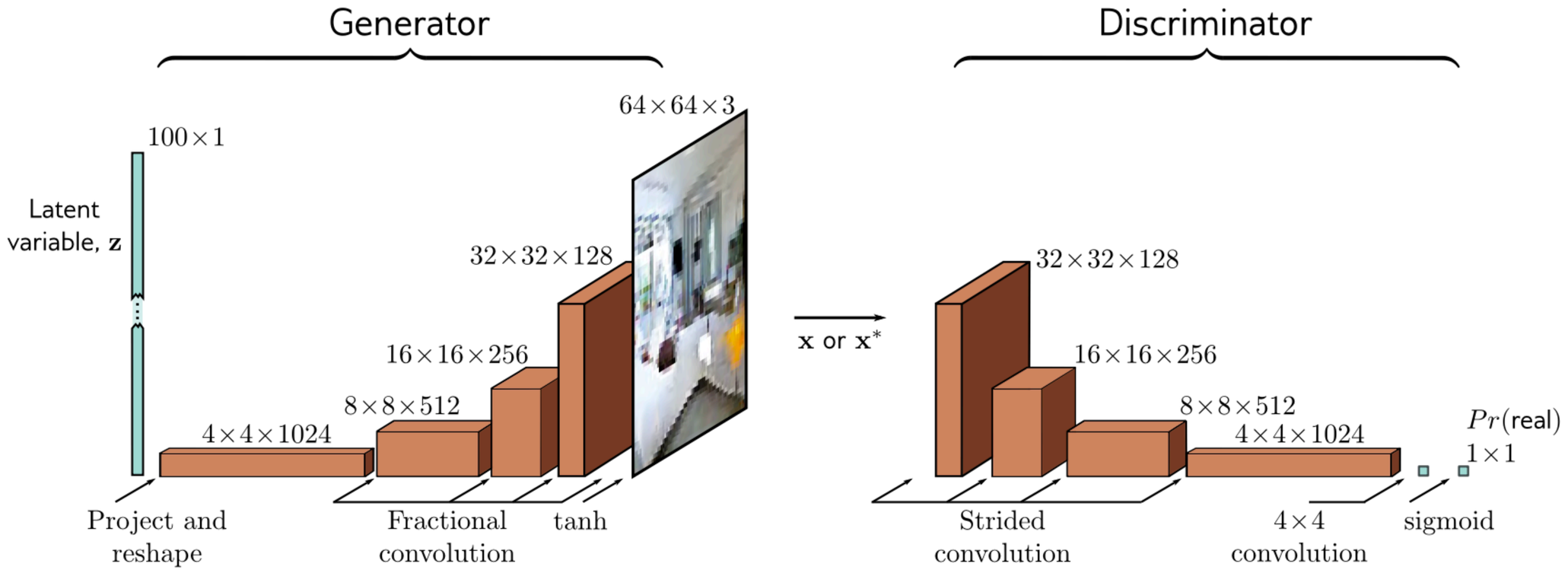
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

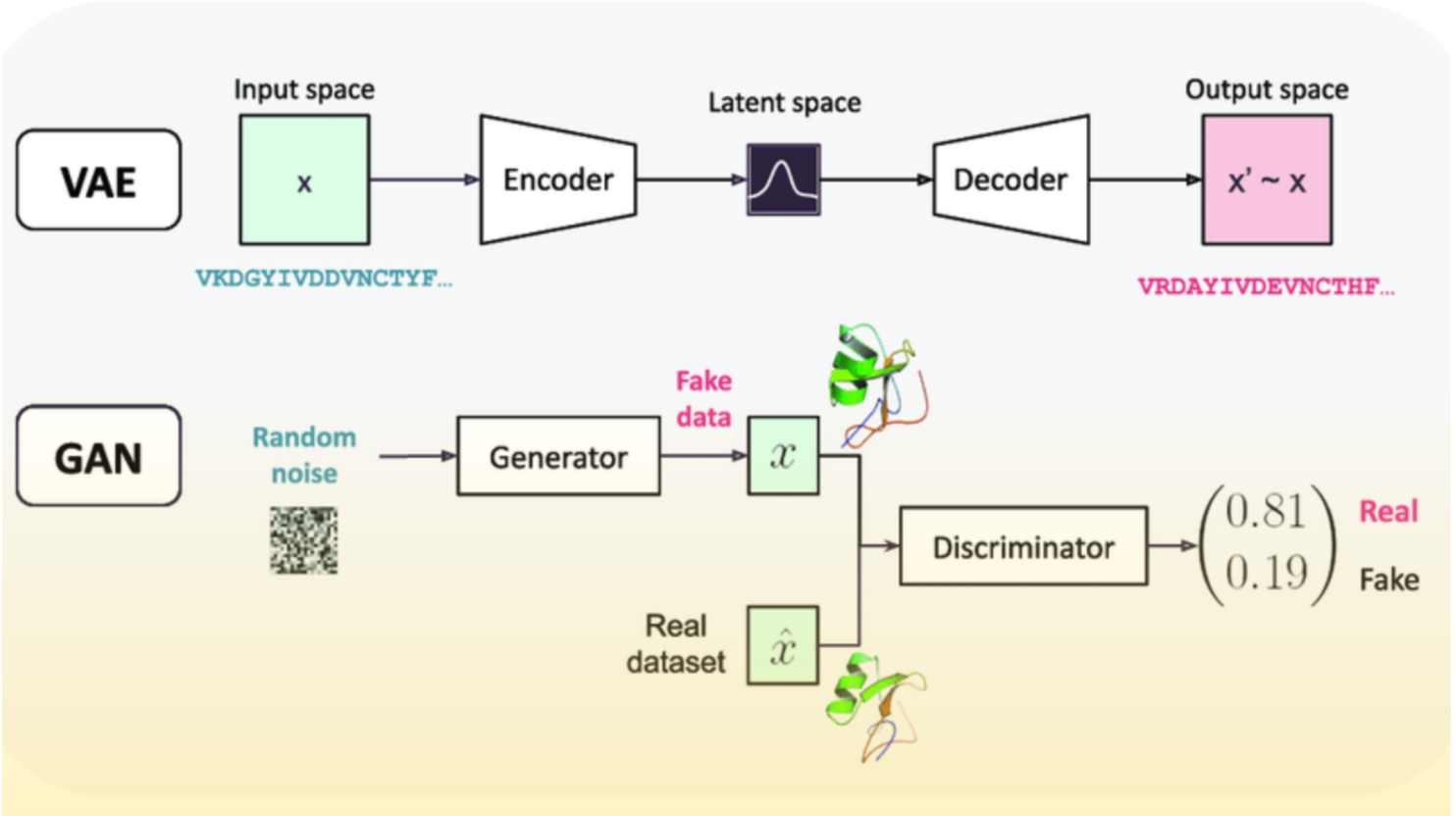
Example: DC Gan

- *A Convolutional Generator from random noise to images*
- *A convolutional binary classifier from images to probability(true)*



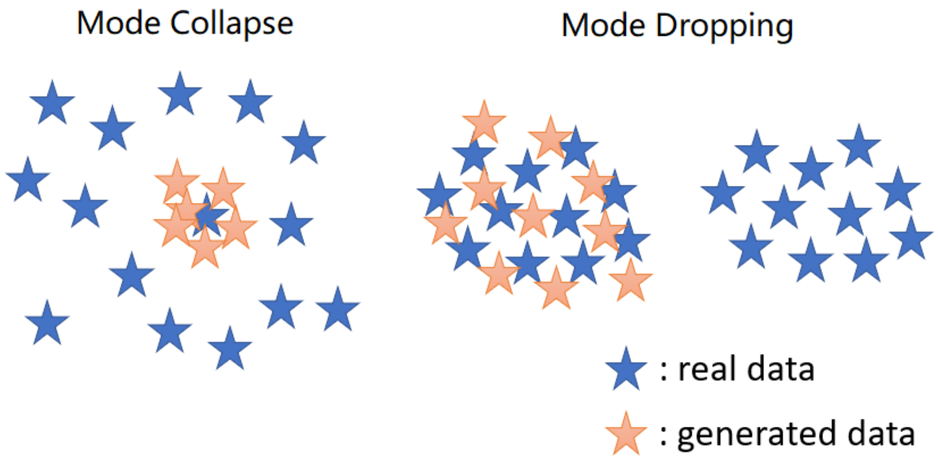
Differences Between GANs and VAEs

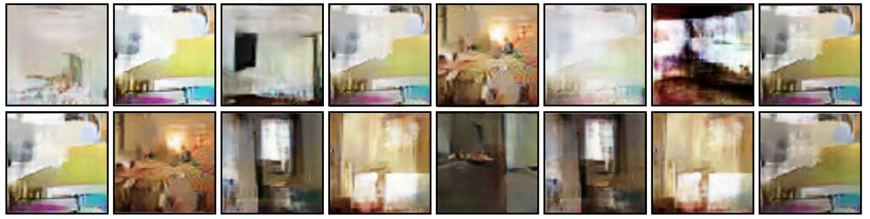
- *Both the decoder of a VAE and a GAN start from the latent space z and generate data in x*
- *The training objective is different*
  - *VAEs use data to determine the pdf of the latent space variables*
    - ➡ *New examples can be sampled from it*
  - *GANs learn a map from latent space to data space*
    - ➡ *examples can be generated, but not sampled (we don't have the measure of how probable each point in the latent space is)*
- *We will see in one of the next lectures how Normalizing Flows try to merge these two concepts*

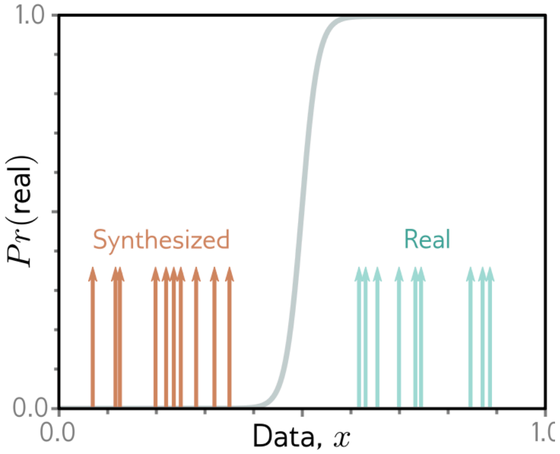


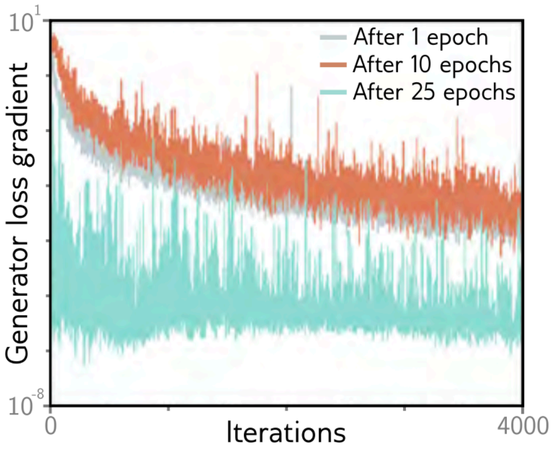
Training a GAN is practice

- ***A GAN training can fail in many ways***
  - *Mode dropping: converge to plausible generations, but only spanning a subset of the data types (e.g., generate faces, but never with specific features, like moustache or glasses)*
  - *Mode collapse: the network ignores the randomness of the input and generates always the same image with minimal variations*
  - *Vanishing gradient: if (particularly at beginning) the generator gives easy-to-identify fake images, a small change in the generator will not overcome the easy task of the discriminator. Happens whenever the pdf (in z) of the real and generated images is very different (very often).*
- *To avoid these issues, people have to go to tricky trial&error procedures, resulting in often ad-hoc prescriptions*
  - *e.g., DCGAN required BatchNorm in G and D, stride>1, leaky ReLu in D, Adam optimizer with reduced momentum. A traditional network would maybe not train optimally for different choices, but the training would not fail completely*
- ***A DELICATE BALANCE:** In general, one needs a good discriminator to challenge the generator, but not a too good discriminator so that the generator can progress learning*









Wasserstein GANs

- A way to fix the GAN training instability is to use a more robust loss.

- Stable results were obtained very early, using

$$\mathscr{L}[\phi] = \sum_j f(x_j^* \,|\, \phi) - \sum_i f(x_i \,|\, \phi) = \sum_i f(G(x_i \,|\, \theta) \,|\, \phi) - \sum_i f(x_i \,|\, \phi)$$

where the discriminator $f(x)$ is forced to have a gradient norm

$$\left| \frac{\partial f(x \,|\, \phi)}{\partial x} \right| < 1$$

- Let's see where this comes from

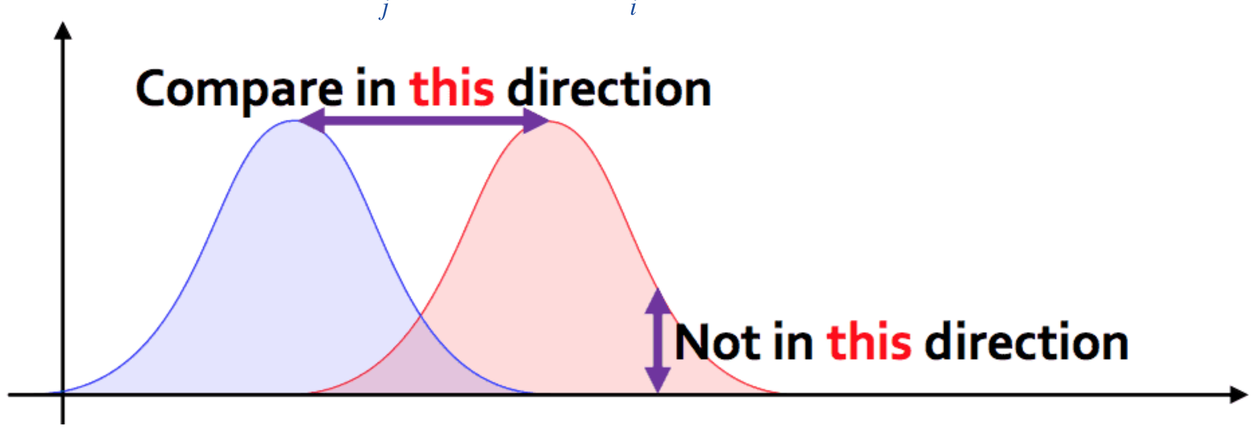
Wasserstein distance

- The Wasserstein distance (or earth mover distance) is a measure of distance between functions
  - An alternative to other metrics, like the KL divergence (and JS distance) that we already discussed
- It corresponds to the work needed to move probability mass from a function f to make it coincide with some other function g
- For discrete quantities, it is defined as

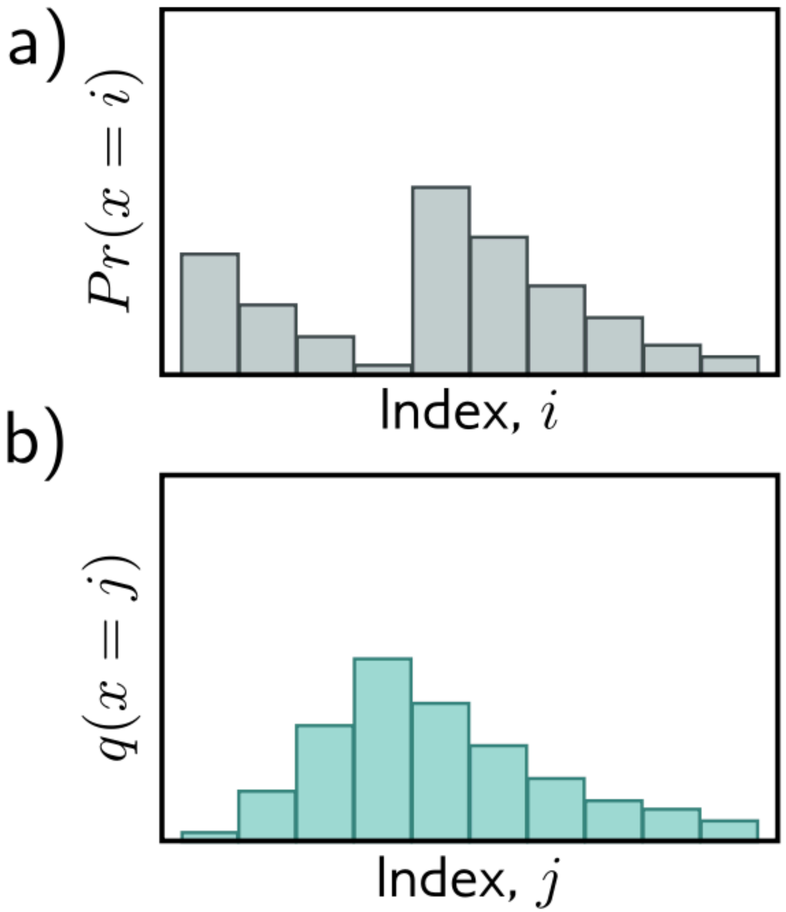
$$D_w[Pr_i, q_j] = \min_P \left[\sum_{i,j} P_{ij} |i - j| \right]$$

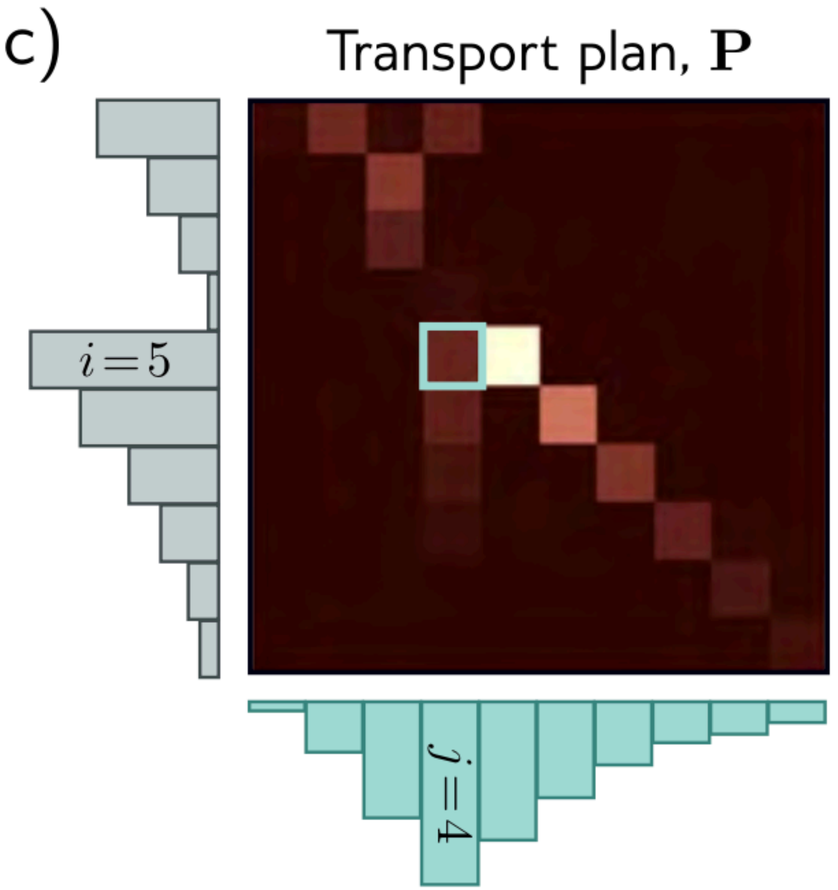
where the matrix P store the amount of probability moved from q to Pr, defined such that

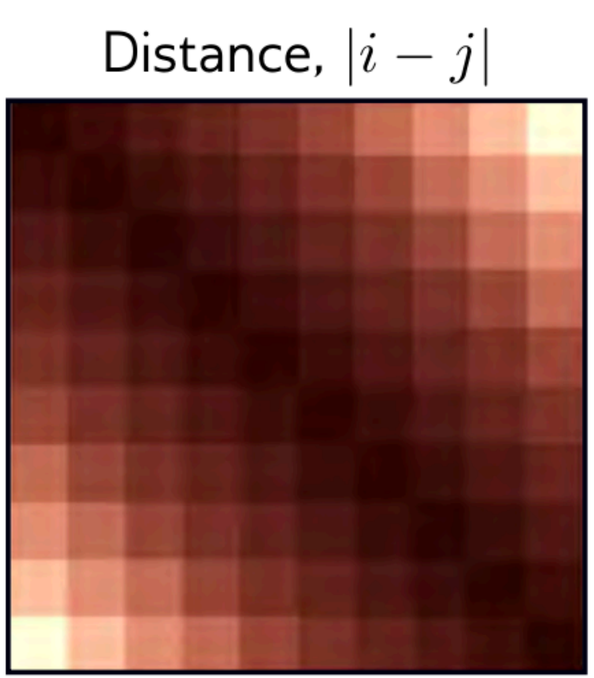
$$\sum_j P_{ij} = Pr(i) \ , \ \sum_i P_{ij} = q_j \ , \ P_{ij} \geq 0$$



Wasserstein distance







Wasserstein distance
$= \sum \mathbf{P} \cdot |i - j|$

Wasserstein distance

- It can be shown (we skip the math there) that given a set of discrete functions f one can write

$$D_w[Pr, q] = \min_P \left[\sum_{i,j} P_{ij} \, | \, i - j \, | \right] = \max_f \left[\sum_i Pr_i f_i - \sum_j q_j f_j \right]$$

with $|f_{i+1} - f_i| < 1$

- This can be generalized to continuous functions. The Wasserstein distance is written as

$$D_w[Pr(x), q(x)] = \min_\pi \left[\int\int \pi(x_1, x_2) \, | \, x_2 - x_1 \, | \, dx_1 dx_2 \right]$$

which can computed maximizing the following expression over an ensemble of functions $f(x)$

$$D_w[Pr(x), q(x)] = \max_{f(x)} \left[\int Pr(x) f(x) dx - \int q(x) f(x) dx \right]$$

with $\left| \frac{\partial f(x \, | \, \phi)}{\partial x} \right| < 1$

Wasserstein Loss

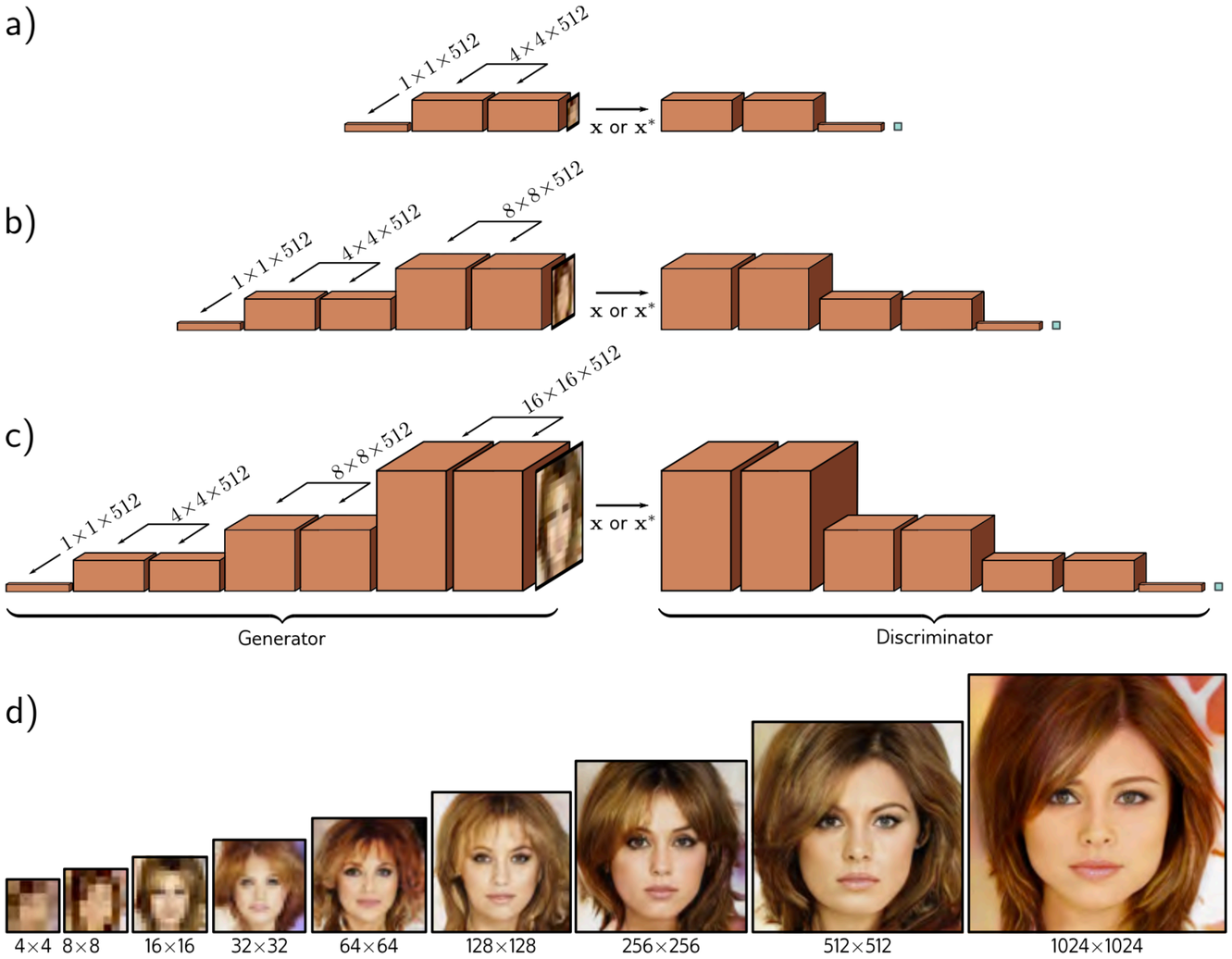
- Our loss is then understood in terms of the Wasserstein distance
  - We optimize over the ensemble of functions f that can be expressed by our neural network f, by changing the parameters ϕ of the network
  - Pr is the pdf of the generated data
  - q is the pdf of the real data
  - We approximate the two integrals as numerical sums over the data in the generated and real dataset
- We then obtain

$$\mathscr{L}[\phi] = \sum_j f(x_j^* \,|\, \phi) - \sum_i f(x_i \,|\, \phi) = \sum_i f(G(x_i \,|\, \theta) \,|\, \phi) - \sum_i f(x_i \,|\, \phi)$$

and the gradient penalty is imposed with a regularisation term

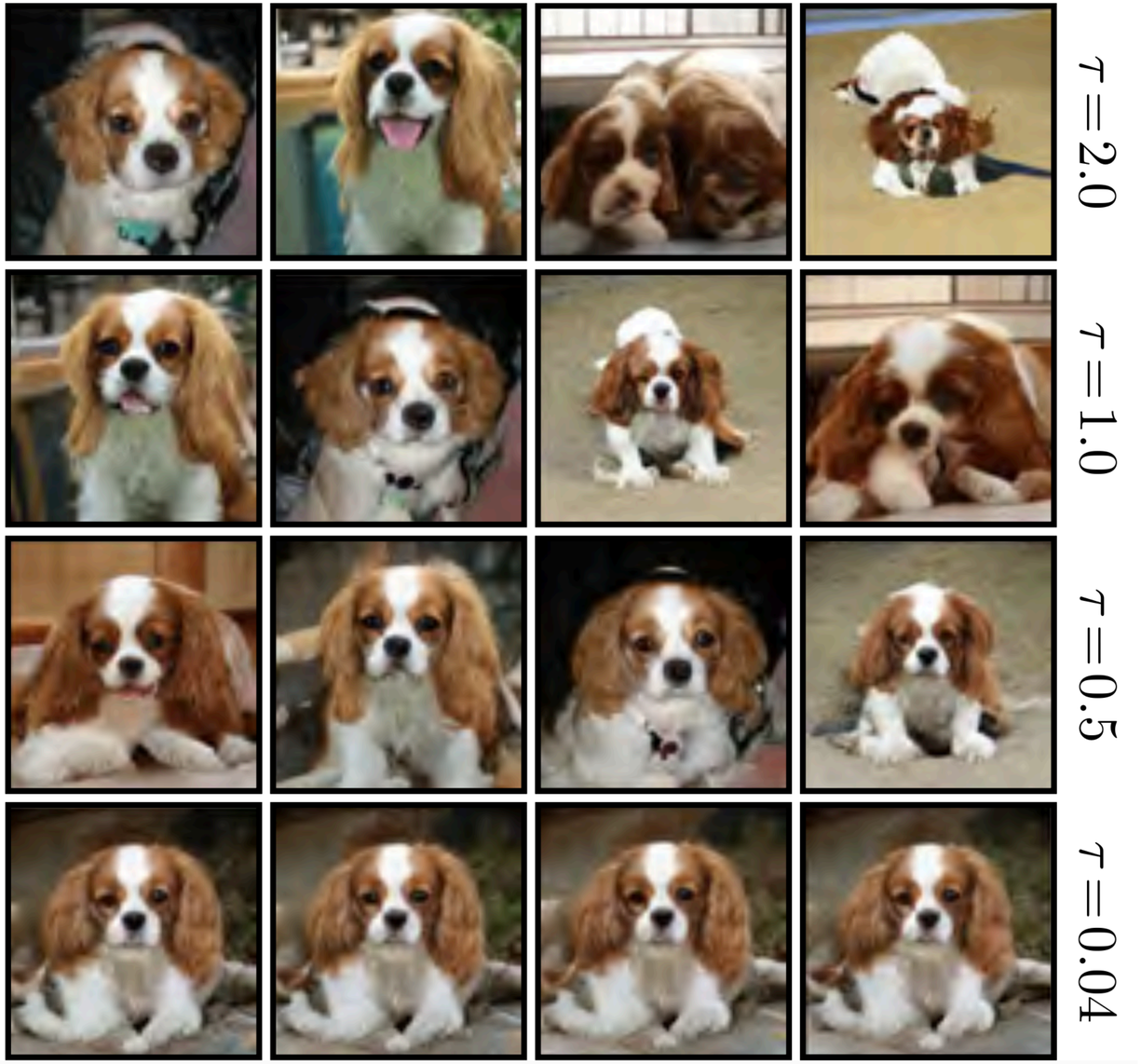
Training strategies

- *Progressive growing: one starts with low resolution (4x4) images, and progressively moves to higher resolution adding upsampling layers to the generator and extra "first" layers to the discriminators*
  - *This approach opens the way to high-resolution generators*



Training strategies

- *Truncation: z are sampled around their high probability (approximated by the mean). Less variation in the sample, but higher resolution is obtained*

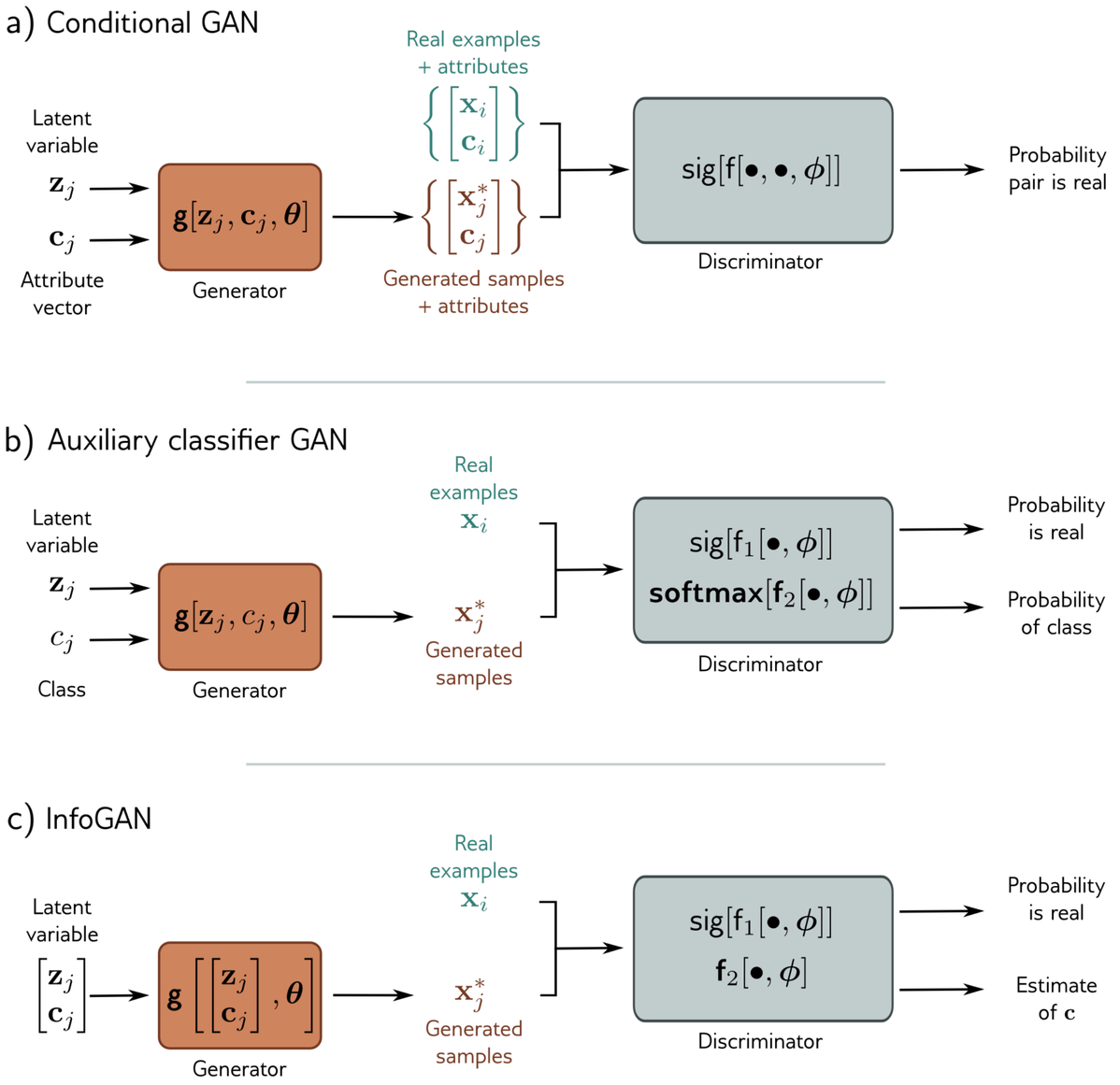


Training strategies

- *Minibatch discrimination: the discriminator is provided with additional information, corresponding to the variation of the input across a mini batch*
  - *e.g., the pixel-by-pixel L1 norm between inputs*
- *With that, the discriminator can easily identify generated images that are too similar to each other (small variation) and force the generator to diversify the images*
  - *This pushes the generator far from mode collapsing*

Conditional Generation

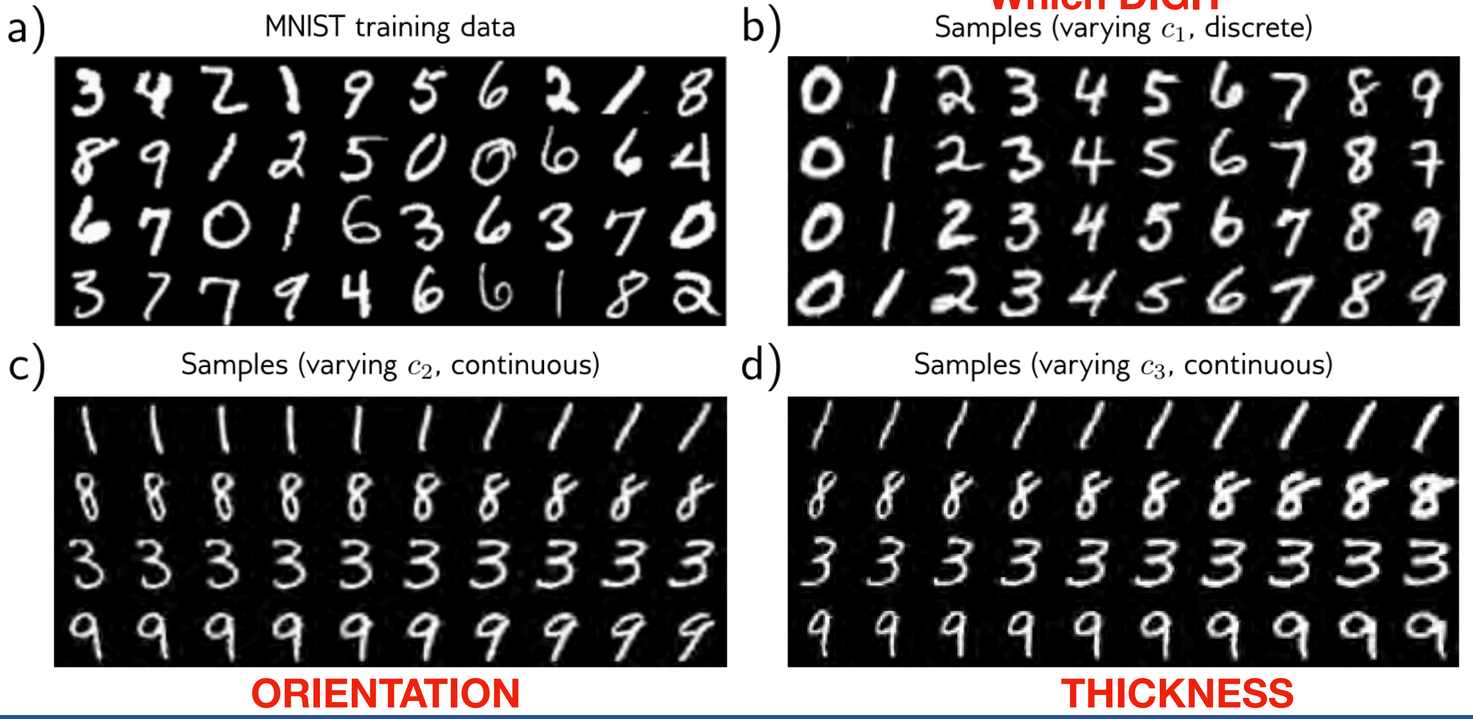
- *One can force the GAN towards specific tasks, injecting information as part of the latent-space representation*
  - *Conditional GANs: additional attributed to Generators, passed fwd to the discriminator. Forces generating specific aspects*
  - *Auxiliary GANs: similar to conditional GANs, but for discrete classes. Class is not passed to discriminator, who has to guess it (double classification task)*
  - *InfoGAN: formally similar to Auxiliary GAN, but the input information is random: in an unsupervised fashion, the network learns to associate it to specific features*



Conditional Generation

- *Example of Auxiliary GAN for ImageNet: can be used to ask not a generic image, but of a specific class (e.g., "generate an image of a monarch butterfly")*
- *Example of InfoGAN: additional info can be discrete or continuous*

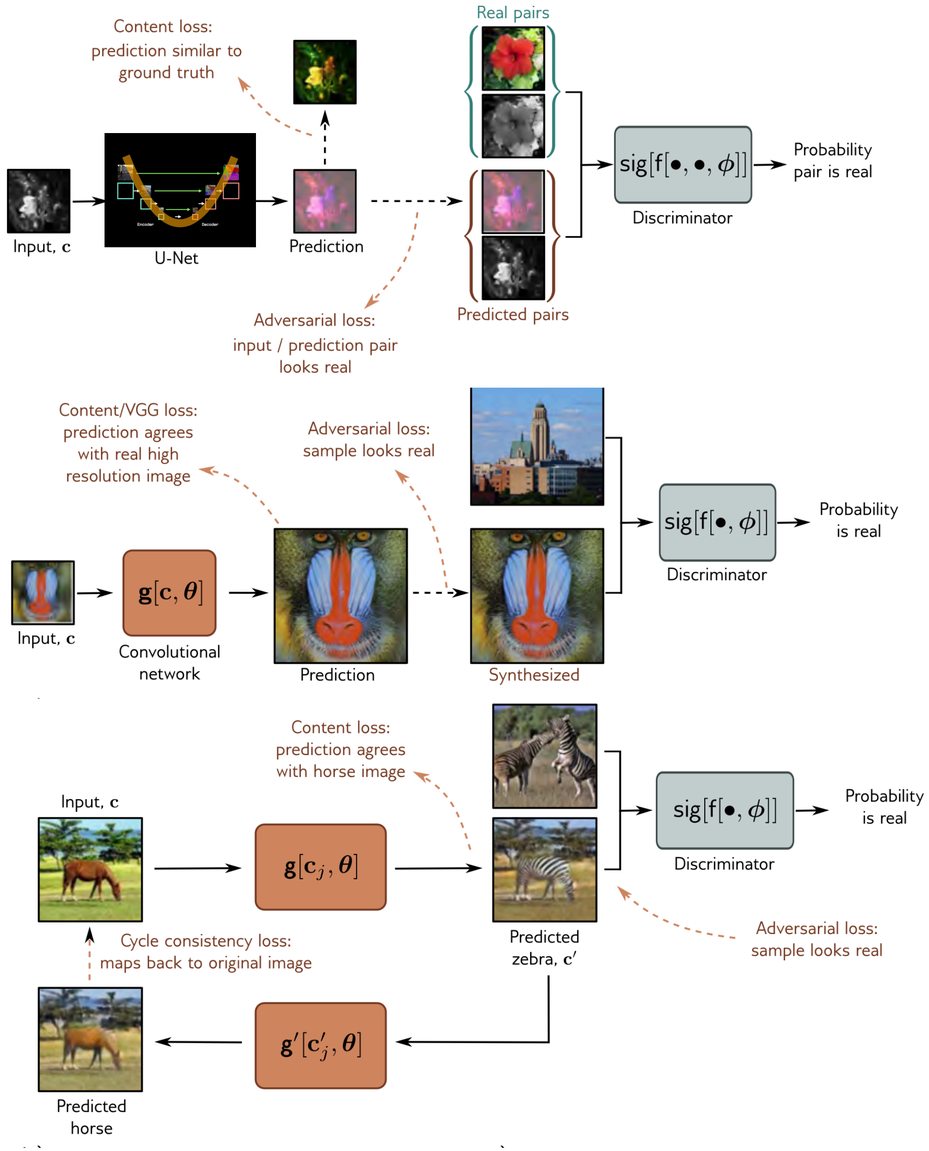
Which DIGIT



ORIENTATION **THICKNESS**

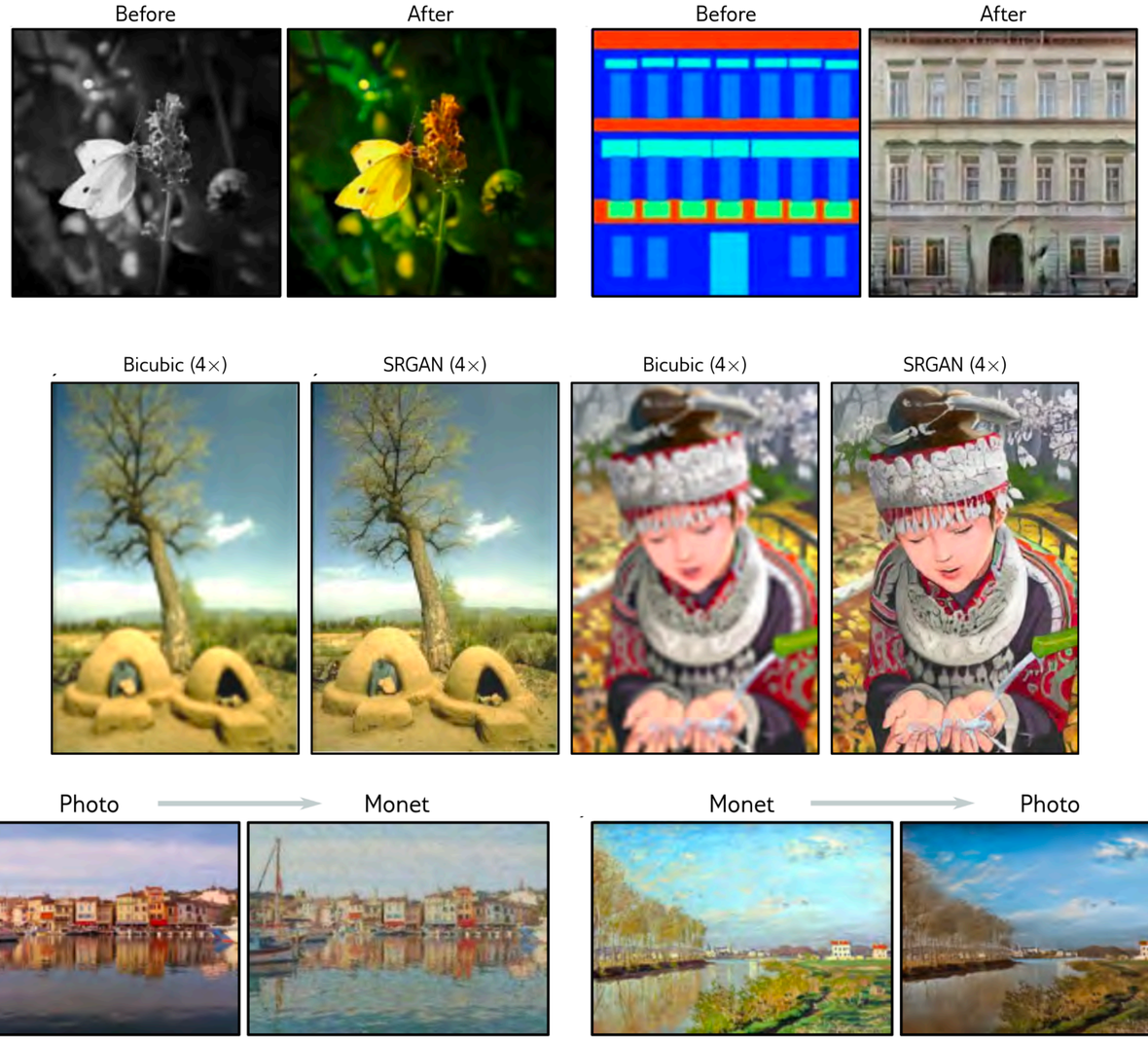
Other Usage of Adversarial Training

- *Pix2Pix is a U-Net that applies style to images (e.g., adding colors to b&w images)*
- *SRGAN returns high-resolution images from low-resolution ones*
- *CycleGAN applies style from reference dataset to a given dataset. It is trained without having examples of before/after correspondence (Monet pairings, images, but no images of landscapes painted by Monet)*



Other Usage of Adversarial Training

- *Pix2Pix is a U-Net that applies style to images (e.g., adding colors to b&w images)*
- *SRGAN returns high-resolution images from low-resolution ones*
- *CycleGAN applies style from reference dataset to a given dataset. It is trained without having examples of before/after correspondence (Monet pairings, images, but no images of landscapes painted by Monet)*



Beware: GANs hallucinate