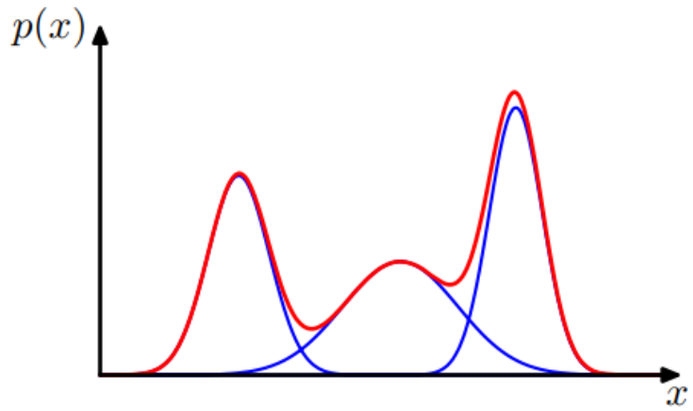
Deep Learning
6. Generative neural networks

A course @EDHEC
by Romain Tavenard (Prof. @Univ. Rennes 2)

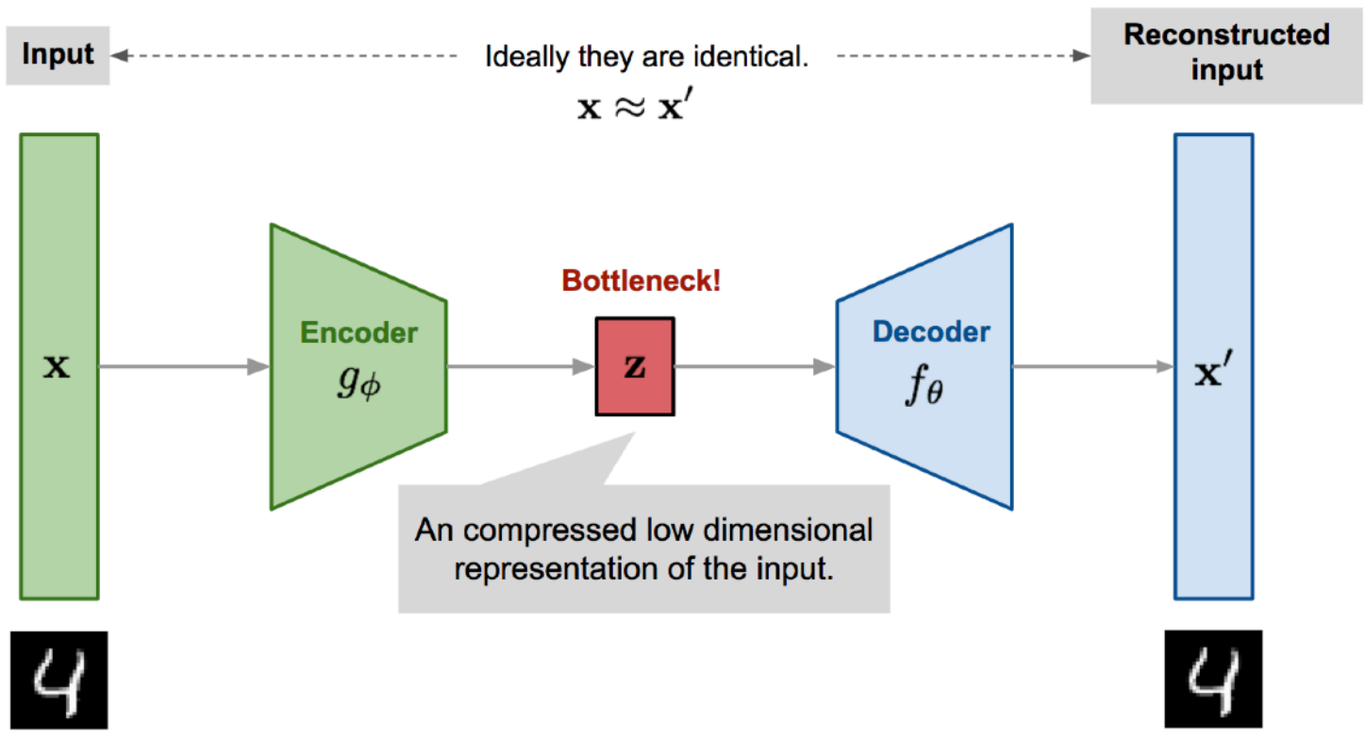
Generative models in a nutshell

- Goal: model $p(x)$ or $p(x \mid y)$
  - explicitly
    - *eg.* Gaussian Mixture Models
  - **implicitly**
    - at least allow for sampling (*i.e.* generate new data)
    - *e.g.* Variational Auto Encoders, Generative Adversarial Networks



Auto-encoders
[Hinton & Salakhutdinov, 2006]

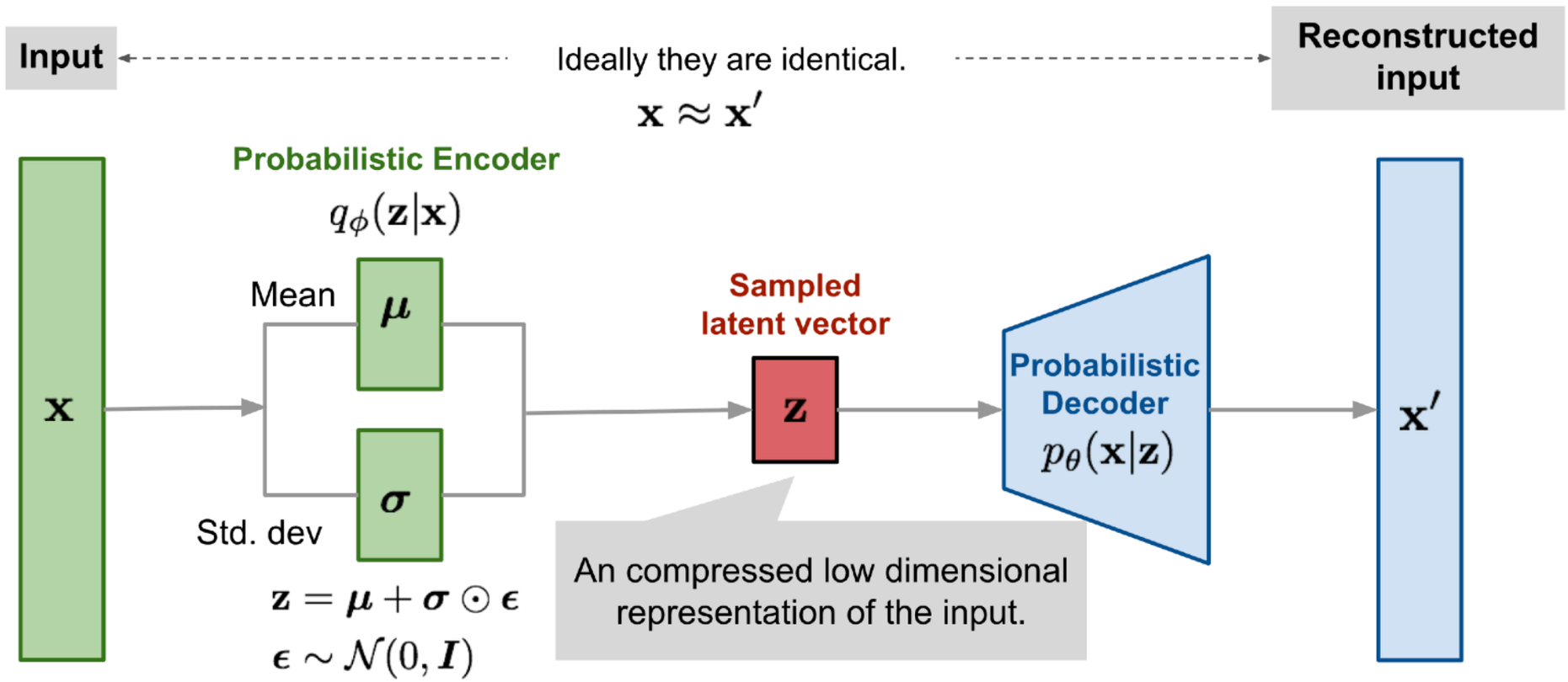
- Encode information in a latent space
  - typically lower-dimensional
  - similar in spirit to a non-linear PCA
  - not a generative model *per se*!



Source: lilianweng.github.io

Variational auto-encoders
[Kingma & Welling, 2014]

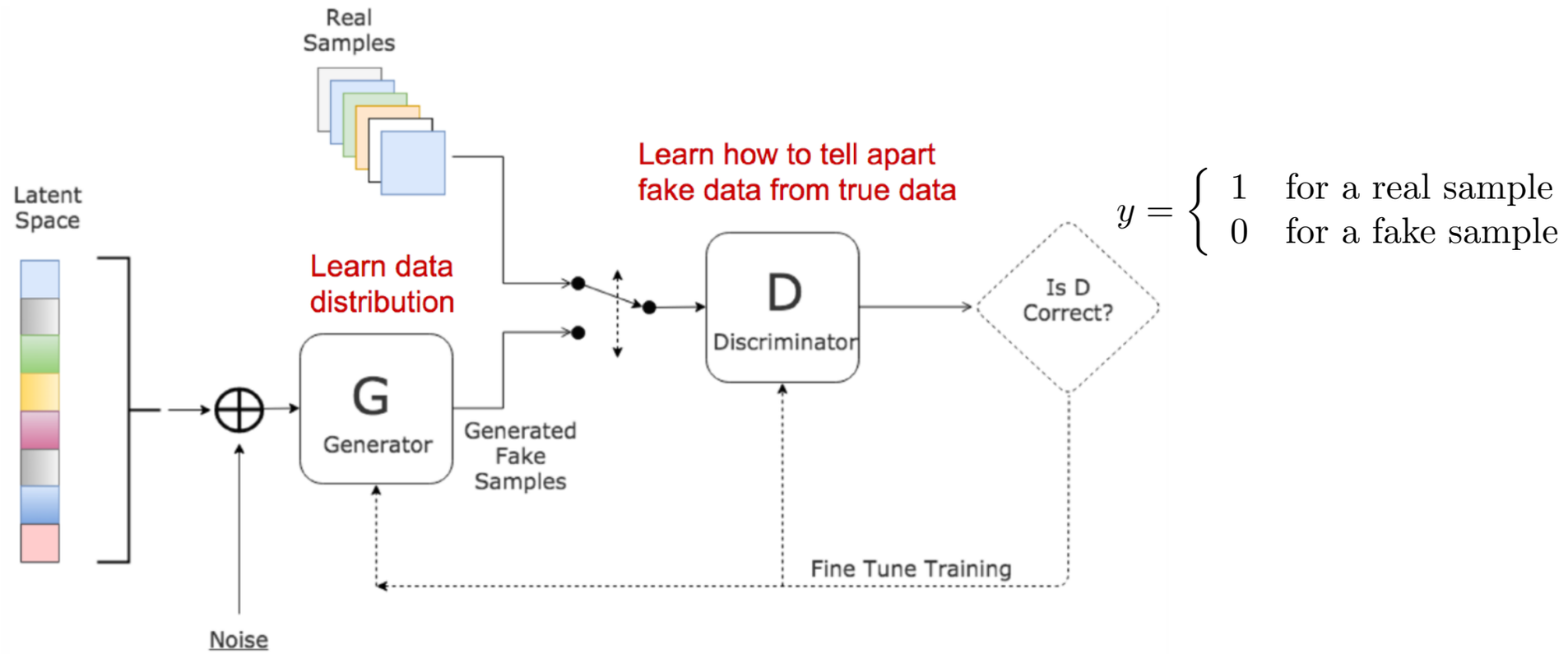
- Goal: turn auto-encoders into generative models
- Idea: set a prior on the distribution of z
  (through penalization of the loss function)
- Generative process: draw z from the prior, and decode it



Source: lilianweng.github.io

Generative Adversarial Networks [Goodfellow *et al.*, 2014]

- Model:



Source: lilianweng.github.io

- Loss function:

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$
$$= \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$$

Generative Adversarial Networks [Goodfellow *et al.*, 2014]

- Generative process
  - Draw z from p_z
  - Pass it to the generator to compute $G(z)$
- Optimization
  - Alternate between generator and discriminator
  - Very unstable process in practice

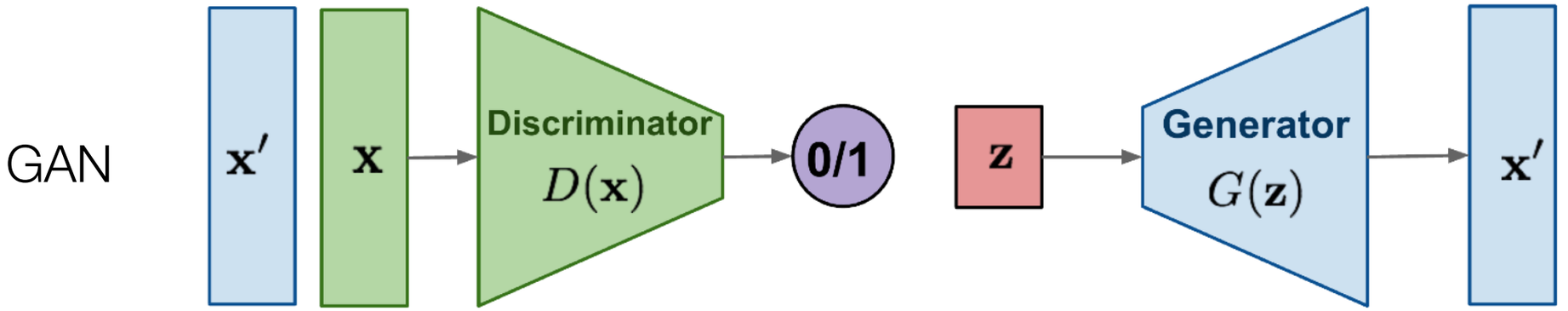
$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$
$$= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log(1-D(x))]$$

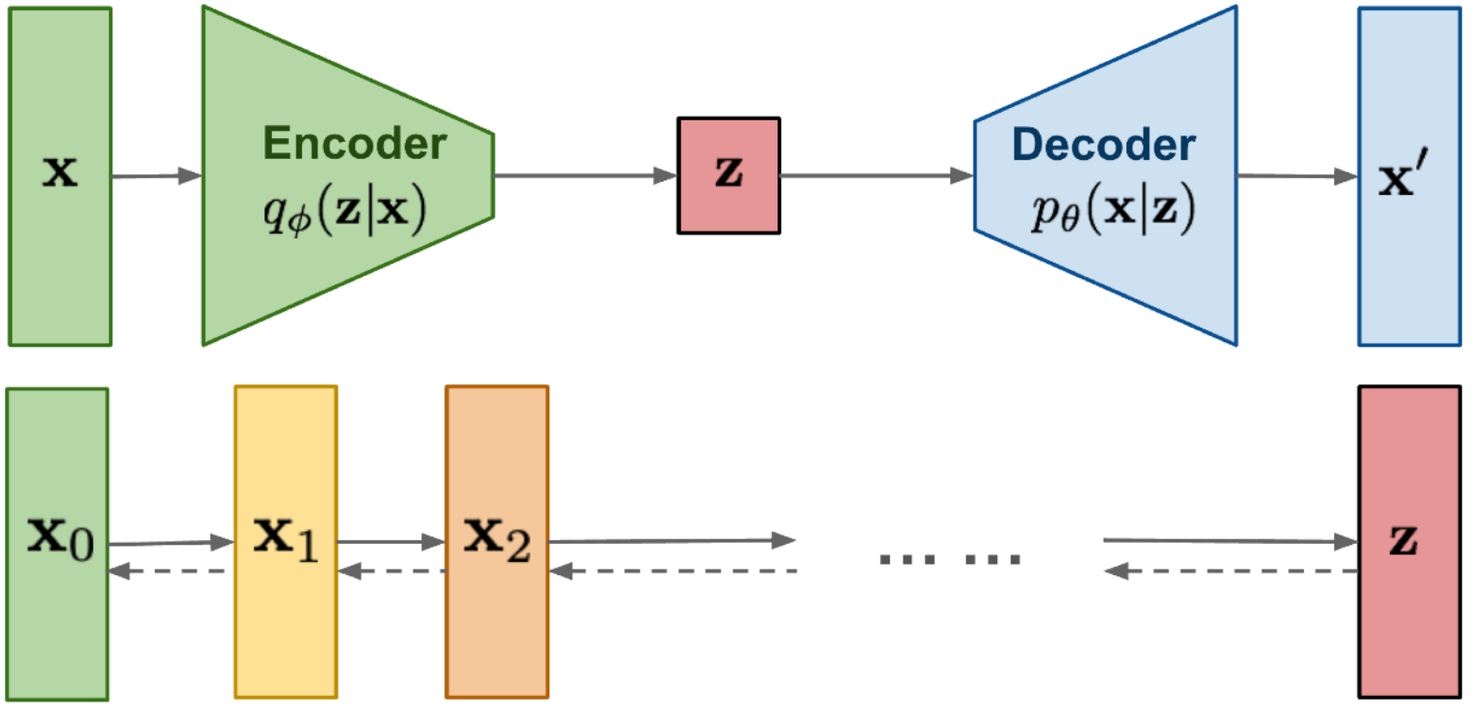
Generative Adversarial Networks

- Many variants to the original model
  - Class-conditional variants
  - Different losses
  - Different structures

- Very realistic samples generated (BigGAN, StyleGAN)

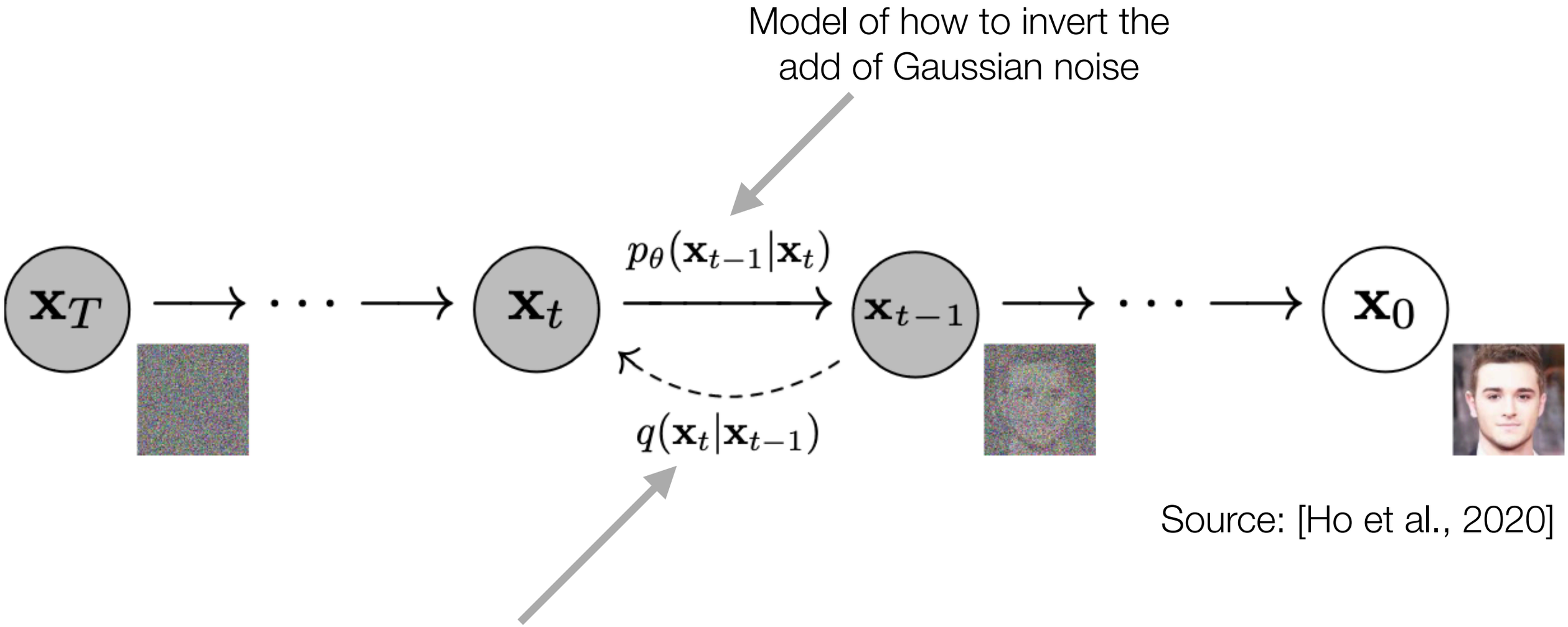
Diffusion models
[Ho *et al.*, 2020]





Source: lilianweng.github.io

Diffusion models
[Ho *et al.*, 2020]



Source: [Ho et al., 2020]

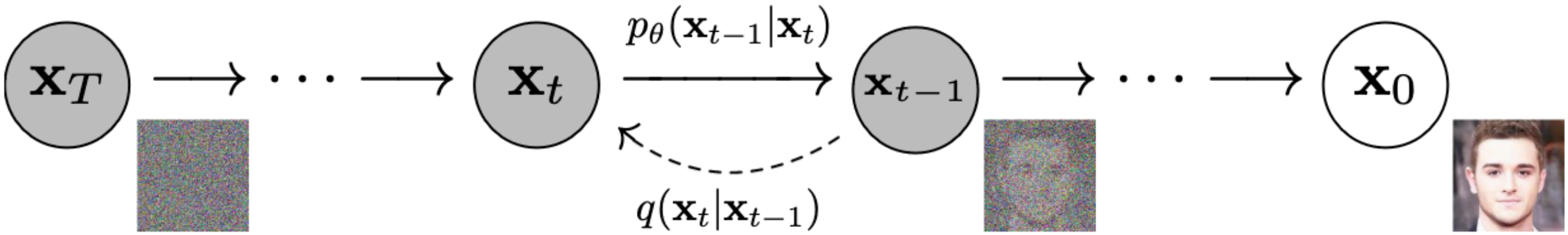
Diffusion models
[Ho *et al.*, 2020]

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$



Source: [Ho et al., 2020]

Conditional Flow Matching [Lipman *et al.*, 2023]

- Principle: learn a vector field that transports noise samples (t=0) to data samples (t=1)
- Loss

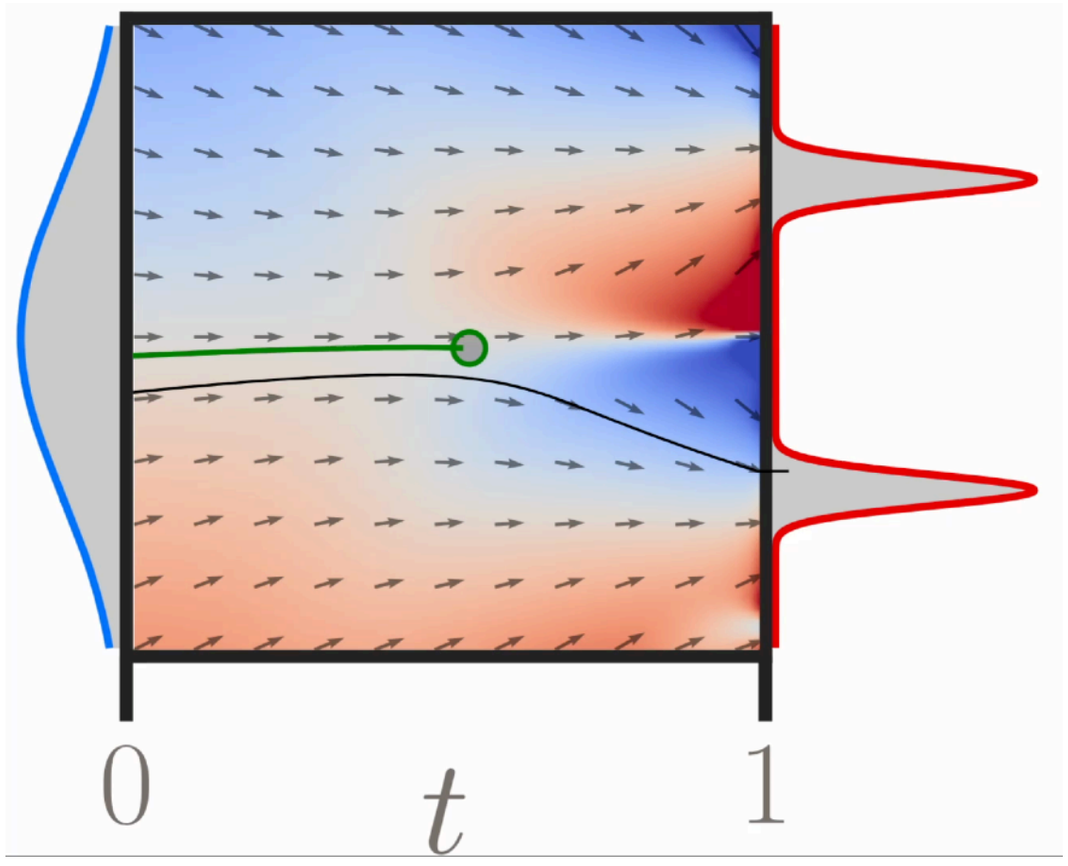
$$\mathbb{E}_{x_0,x_1,t} \left\| u^{\theta}(x,t) - (x_1 - x_0) \right\|^2$$

"Ground truth" velocity

$t \cdot x_0 + (1-t) \cdot x_1$

- At generation time, use an Ordinary Differential Equation solver (eg. Euler scheme):

$$x_{t+\varepsilon} \leftarrow x_t + \varepsilon u^{\theta}(x_t, t)$$



Source: [Gagneux et al., 2025]