Likelihood-Free Generative Models

Advanced Statistical Inference

Simone Rossi

Likelihood models

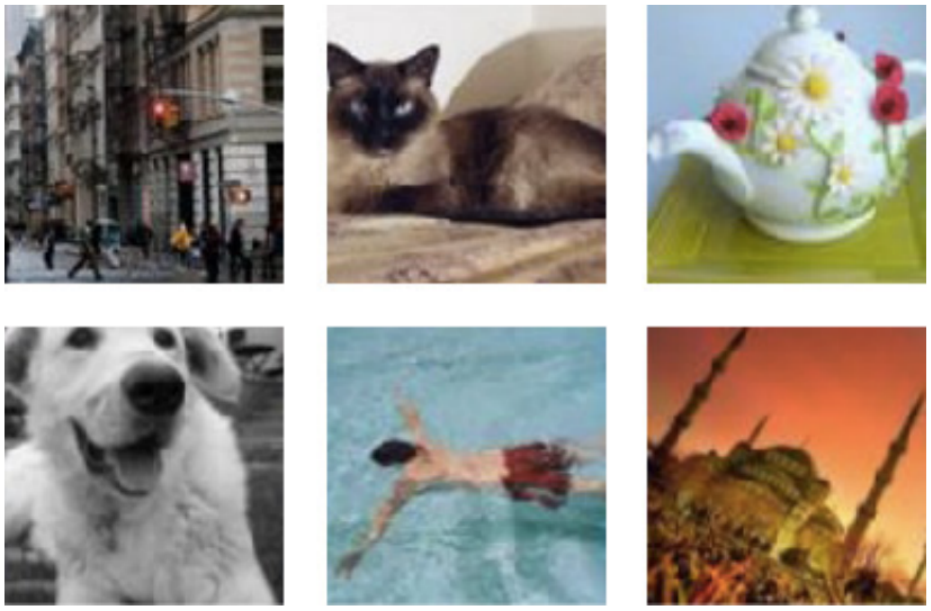
- So far we have focused on likelihood-based generative models, where we maximize the likelihood of the data given the model parameters.
- Likelihood-based models are powerful, but they have some limitations:
  - We run on the assumption that higher likelihood means better generated samples, but this is not always true.

Figure 1: 256×256-pixel samples generated by NVAE, trained on CelebA HQ [28].

- **Likelihood-free** learning consider alternative training objectives that do not depend directly on a likelihood

Comparing distributions via samples

$S_1 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \mid \boldsymbol{x}_i \sim p(\boldsymbol{x})\}$



$S_2 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \mid \boldsymbol{x}_i \sim q(\boldsymbol{x})\}$

Given a finite set of samples from two distributions (S_1 and S_2), how can we tell if these samples are from the same distribution or not?

Two-sample tests

Set up a statistical test to compare the two sets of samples S_1 and S_2:

- **Null hypothesis**: the two sets of samples are drawn from the same distribution, i.e., $p = q$.
- Test statistic T computes a S_1 and S_2, for example the difference in the empirical means:

$$T(S_1, S_2) = \frac{1}{N_1} \sum_{x \in S_1} x - \frac{1}{N_2} \sum_{x \in S_2} x$$

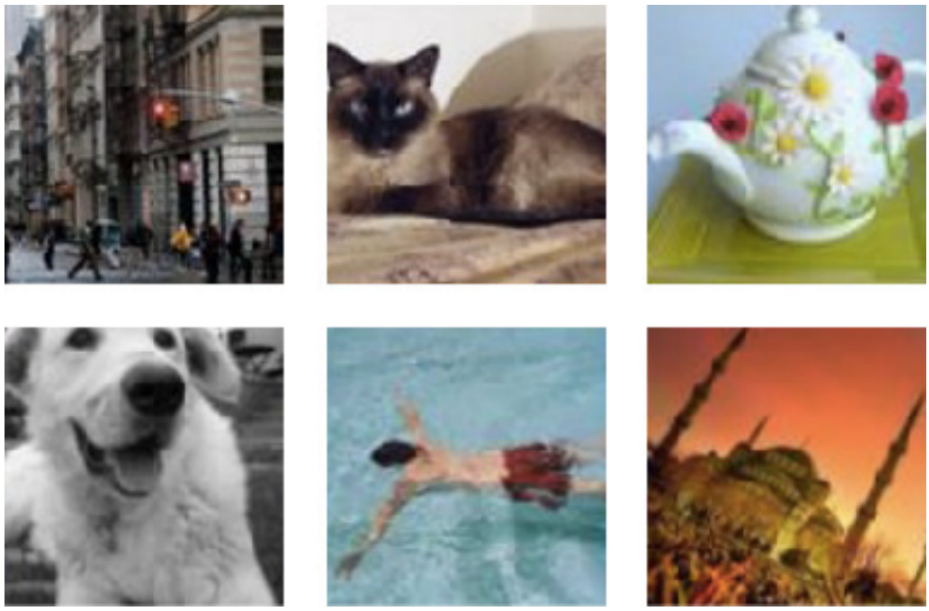
- If T is larger than some threshold α, we reject H_0

. . .

Key observation: Test statistic is **likelihood-free** since it does not depend on p and q directly, but only on the samples from these distributions.

Generative models and two-sample tests

Assume $p_{\text{data}}(x)$ is the true data distribution and we have training samples S_1

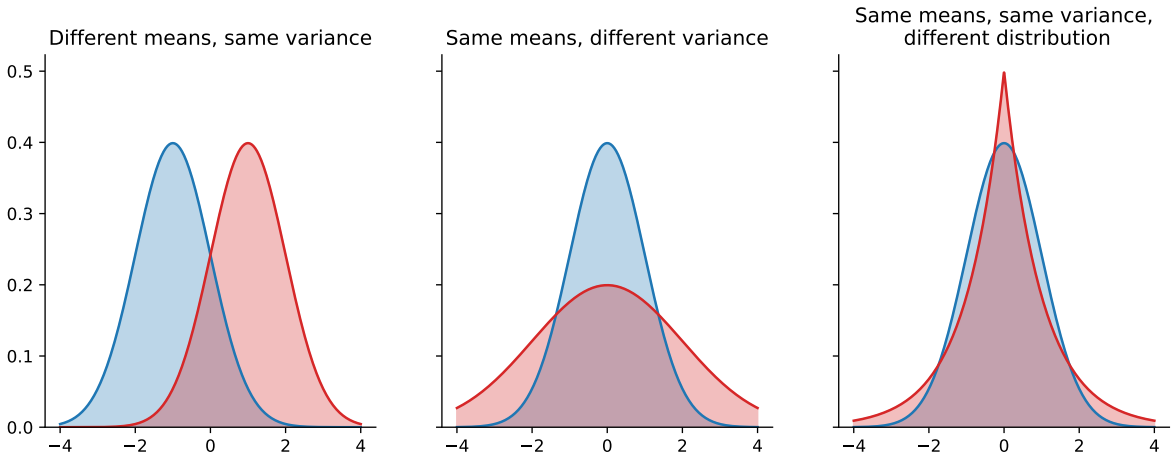


Assume we have a model $p(\boldsymbol{x};\boldsymbol{\theta})$ that permits us to generate samples for S_2

. . .

Idea: formulate the training of the generative model to minimize a two-sample test statistic between S_1 and S_2

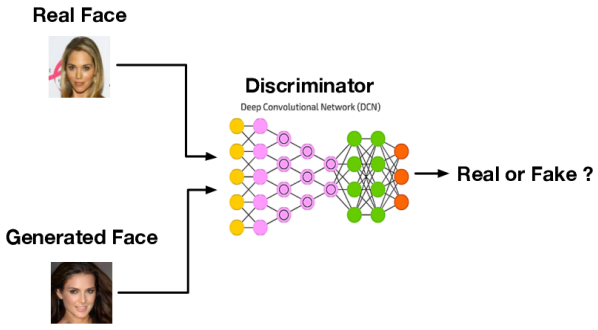
Two-sample tests are hard



- In the generative model setup, we know that S_1 and S_2 come from different complex distributions, $p_{\text{data}}(\boldsymbol{x})$ and $p(\boldsymbol{x}; \boldsymbol{\theta})$. Simple test statistics based on moments (e.g., mean, variance) may not be able to distinguish between the two distributions.
- **Idea**: **Learn** a statistic to automatically identify in what way the two sets of samples differ
- How? Train a classifier (which we will call **discriminator**)

Discriminator

Build binary classifier $D(\boldsymbol{\phi}, \boldsymbol{x})$ (e.g., neural network with parameters $\boldsymbol{\phi}$) that tries to distinguish "real" ($y = 1$) samples from the dataset and "fake" ($y = 0$) samples generated from the model



Test statistic: negative loss of the classifier.

- Low loss means real and fake samples are easy to distinguish (distributions are different).
- High loss means real and fake samples are hard to distinguish (distributions are similar).

. . .

Goal: Maximize the two-sample test statistic or equivalently minimize the classification loss.

Loss for the discriminator

Do you remember what we used for a binary classification task? **Bernoulli**

. . .

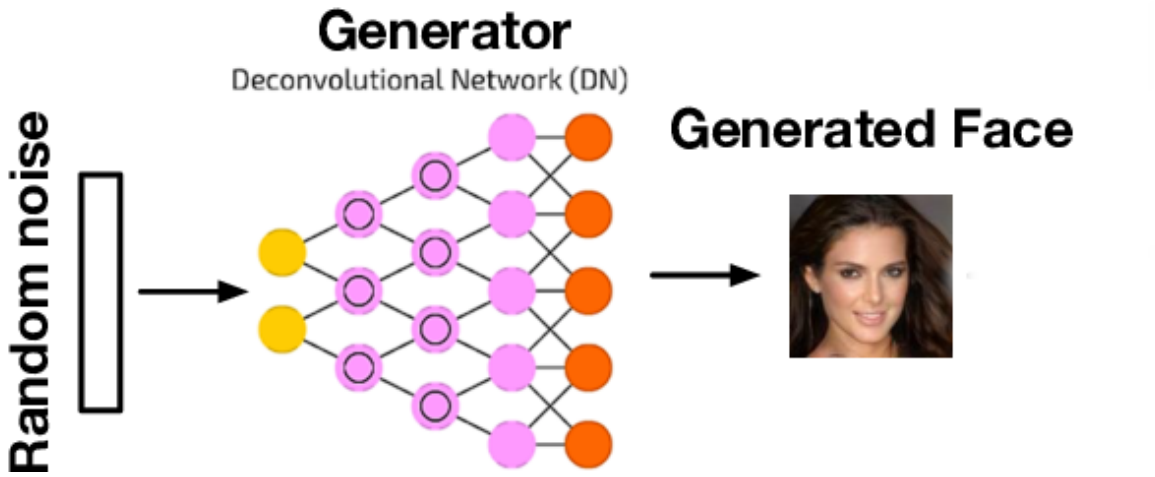
$$\arg\max_{\phi} \mathcal{L}(\phi) := \mathbb{E}_{p_{\text{data}}(x)} \log D(\phi, x) + \mathbb{E}_{p(x;\theta)} \log(1 - D(\phi, x))$$
$$\approx \frac{1}{N_1} \sum_{x \in S_1} \log D(\phi, x) + \frac{1}{N_2} \sum_{x \in S_2} \log(1 - D(\phi, x))$$

For a fixed generative model $p(x; \theta)$, we can train the discriminator $D(\phi, x)$ to distinguish between real and fake samples:

- Assign probability 1 to real samples from the dataset S_1 (i.e., from $p_{\text{data}}(x)$)
- Assign probability 0 to fake samples from the model S_2 (i.e., from $p(x; \theta)$)

Generator

Still missing: how do we generate the "fake" samples S_2 from the model $p(x; \theta)$?



Generator:

- Latent variable model with a **deterministic mapping** from a latent variable z to the data space x:
  - Define a function $G(\theta, z)$ as a neural network with parameters θ
  - Sample $z \sim p(z)$, where $p(z)$ is the prior distribution (e.g., Gaussian)
  - Compute $x = G(\theta, z)$,

How do we train the generator?

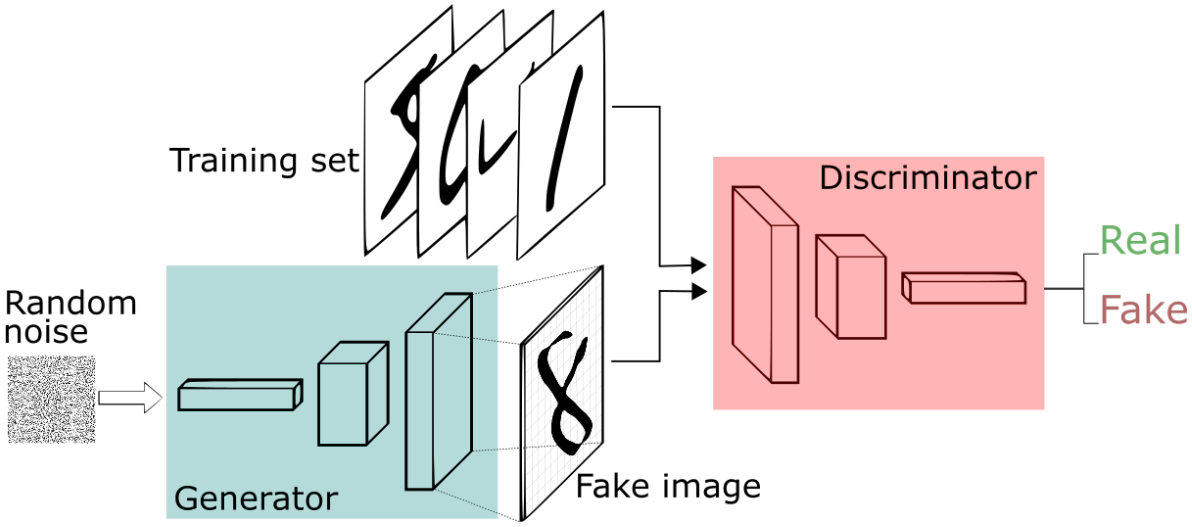
- Previously we trained the discriminator to maximize the two-sample test statistic (i.e., correctly classify real and fake samples).
- But for the generator, we want to **minimize** the two-sample test statistic (i.e., make it hard for the discriminator to distinguish between real and fake samples).

$$\min_{\theta} \max_{\phi} \mathcal{L}(\theta, \phi) := \mathbb{E}_{p_{\text{data}}(x)} \log D(\phi, x) + \mathbb{E}_{p(z)} \log(1 - D(\phi, G(\theta, z)))$$

This is a **minimax** optimization problem

Generative adversarial networks (GANs)

This model configuration and training procedure is known as **Generative Adversarial Networks (GANs)**



Generative adversarial networks (GANs)

GANs have been successfully applied to several domains and tasks

Generative adversarial networks (GANs)

Pros:

- Loss only requires samples, no likelihood needed
- Lots of flexibility in the choice of the neural network architecture for the generator and discriminator
- Can generate high-quality samples
- Fast sampling from the model (just sample from noise and pass through the generator)

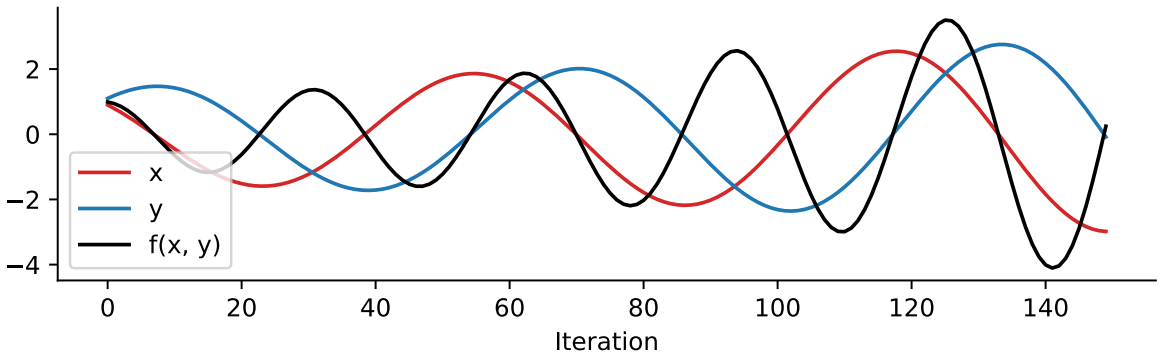
Cons:

- Training is unstable, they are hard to train
- No guarantee that the generator will learn the true data distribution (see Nash equilibrium)
- Hard to *numerically* evaluate the quality of the generated samples (no likelihood, no ELBO)

Training GANs is hard (example)

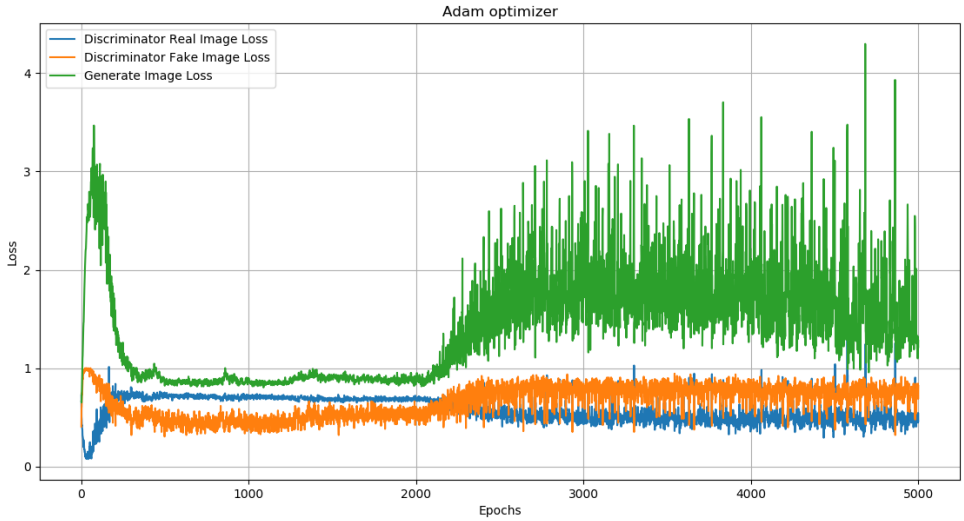
Imagine $f(x, y) = xy$ and we want to comput $\min_x \max_y f(x, y)$. A gradient descent-ascent algorithm would look like this:

$$x \leftarrow x - \eta \nabla_x f(x, y) = x - \eta y$$
$$y \leftarrow y + \eta \nabla_y f(x, y) = y + \eta x$$



Training GANs is hard

- The generator and discriminator loss keep oscillating during GAN training
- Difficult to assess if training is converging or not



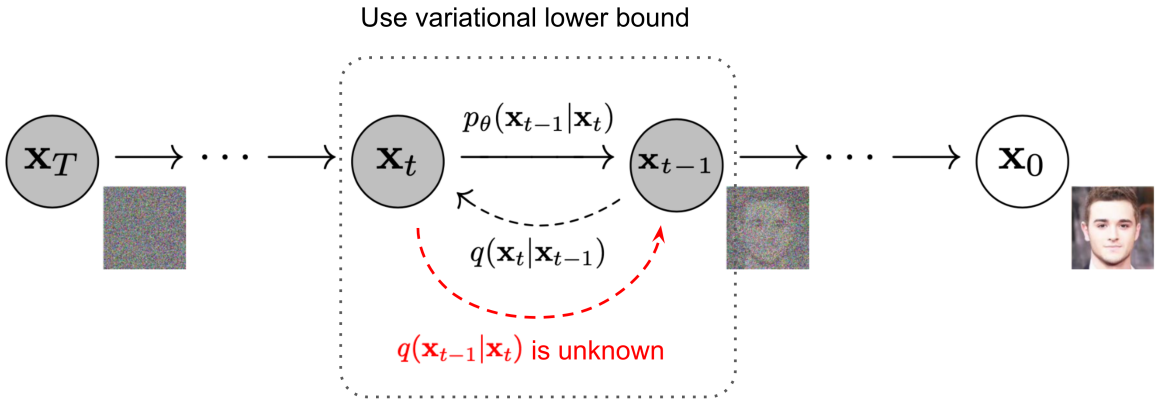
Diffusion models

Diffusion models

Currently the state-of-the-art generative models, especially for image/video generation tasks

Diffusion models

Key idea: corrupt the data x into noise by adding Gaussian noise in a series of steps, and then learn to reverse this process to generate new samples.



For example:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \boldsymbol{I})$$

where β_t is a small positive constant that controls the amount of noise added at each step.

Continuous diffusion process

The diffusion process can be viewed as a continuous-time stochastic process, described by a **stochastic differential equation (SDE)** of the form:

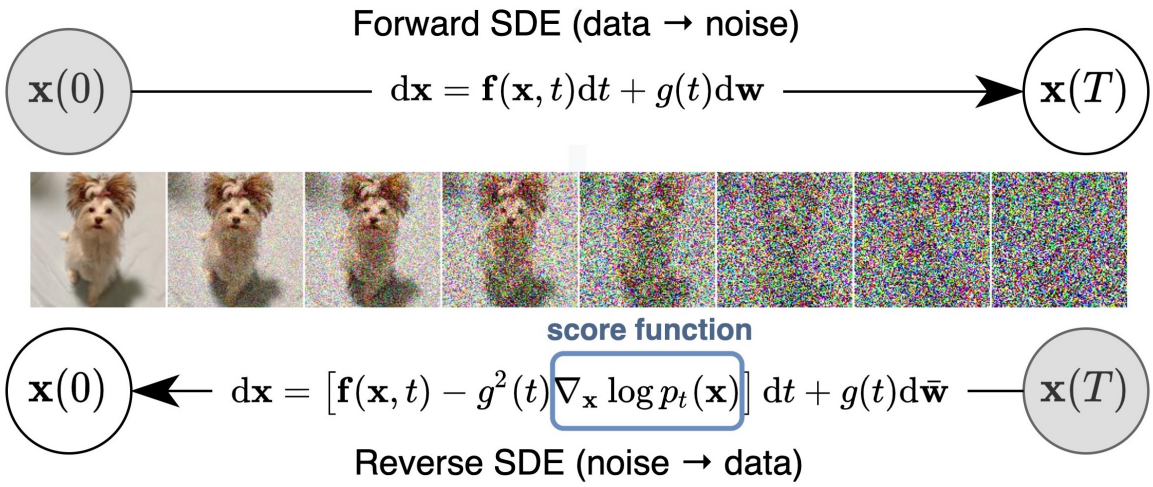
$$\mathrm{d}\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)\mathrm{d}t + g(t)\,\mathrm{d}\boldsymbol{B}_t$$

Reverse diffusion process

The reverse diffusion process is also described by an SDE:

$$\mathrm{d}\boldsymbol{x}_t = \left(\boldsymbol{f}(\boldsymbol{x}_t, t) - g(t)^2\nabla_{\boldsymbol{x}_t}\log p(\boldsymbol{x}_t)\right)\mathrm{d}t + g(t)\,\mathrm{d}\boldsymbol{B}_t$$

Training diffusion models



To train the diffusion model, we learn a neural network $\boldsymbol{s}(\boldsymbol{\theta}, \cdots)$ to approximate the score function $\nabla_{\boldsymbol{x}_t}\log p(\boldsymbol{x}_t)$:

$$\mathrm{KL}\left(p_0(\boldsymbol{x})\|p(\boldsymbol{x};\boldsymbol{\theta})\right) \leq \frac{T}{2}\mathbb{E}_{t\in\mathcal{U}(0,T)}\mathbb{E}_{p_t(\boldsymbol{x})}[\lambda(t)\|\nabla_{\boldsymbol{x}}\log p_t(\boldsymbol{x}) - \boldsymbol{s}(\boldsymbol{\theta}, \boldsymbol{x}, t)\|_2^2] + \mathrm{KL}\left(p_T(\boldsymbol{x})\|p_{\mathrm{prior}}(\boldsymbol{x})\right)$$