First steps to deep-learning

From the perceptron to the deep generative models

Nicolas Desassis, Ferdinand Bhavsar

nicolas.desassis@minesparis.psl.eu

January 2022

Sources

- Coursera: GAN specialization
- Lil'blog

Outline

1. The problem
2. Deep Generative Models: a geostatistical point of view
3. Training generative models
   - GAN
   - WGAN
4. Conditioning (later)

Example - Dimension reduction (or compression)

Starting from a set of examples $\mathbf{x}_1, \ldots, \mathbf{x}_N$, find two functions

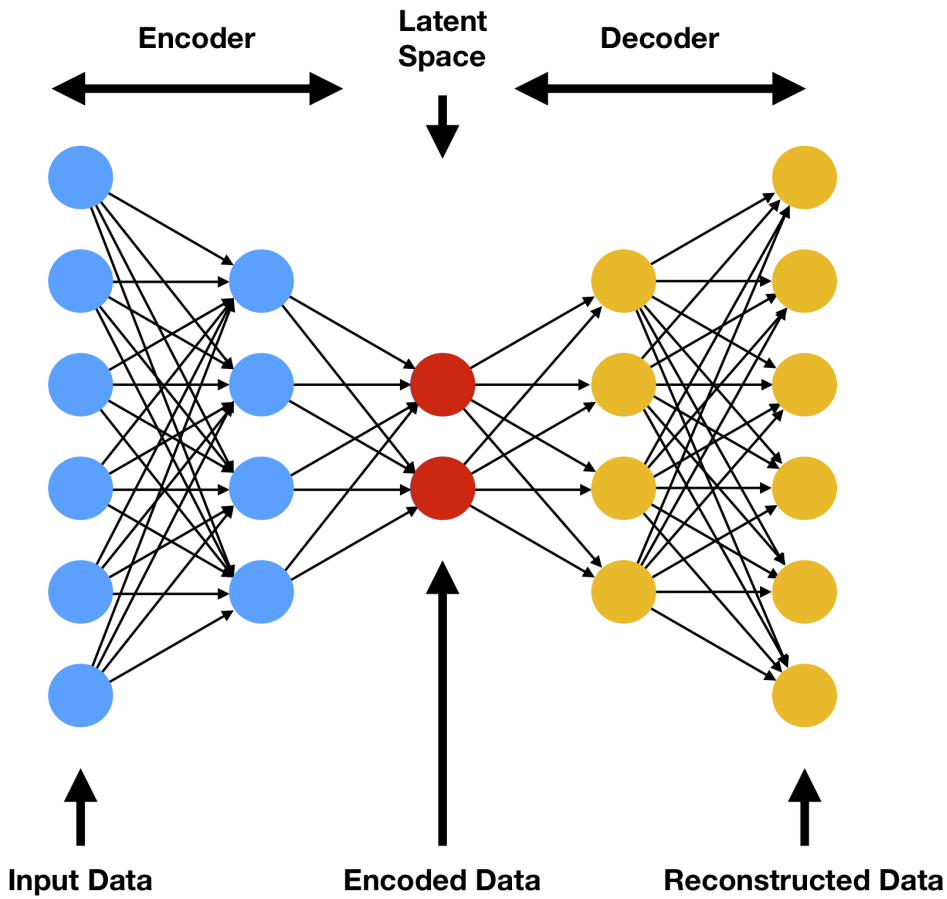
- an encoder f_e from $\mathbf{R}^n$ to $\mathbf{R}^d$ with d much smaller than n
- a decoder f_d from $\mathbf{R}^d$ to $\mathbf{R}^n$

The aim is to minimize the reconstruction error over all the examples

$$T(f_e, f_d) = \sum_{i=1}^{N} ||\mathbf{x}_i - f_d(f_e(\mathbf{x}_i))||^2$$

Example - Dimension reduction (or compression)

Autoencoders



[https://www.compthree.com/blog/autoencoder/]

The problem

Aim

Given a (large?) set of training examples, generate new examples which "look like" the training examples in terms of

- fidelity or realism
- diversity or variability

The problem

Aim

Given a (large?) set of training examples, generate new examples which "look like" the training examples in terms of

- fidelity or realism
- diversity or variability

In other words, given an empirical distribution $P_{\mathcal{D}}$ over $\mathbb{R}^n$, find a probabilistic distribution (the model) P_G with its simulation algorithm (the generator) which is close to $P_{\mathcal{D}}$ (or its theoretical version P)

Small dimensions

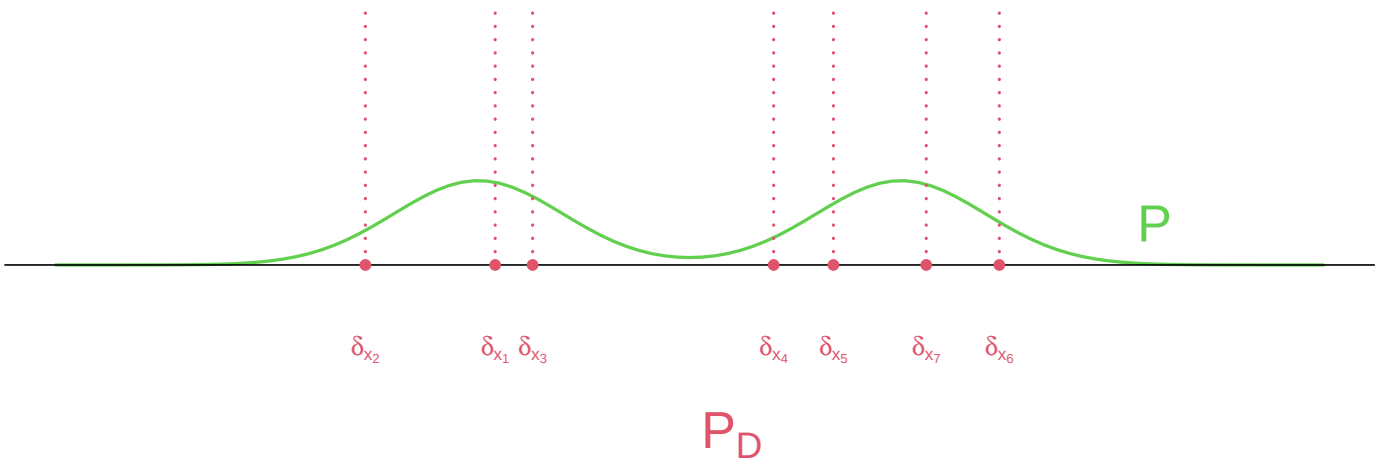
Kernel density estimation (KDE)

- Replace the empirical distribution

$$P_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathbf{x}_i}$$

by its convolution with a (gaussian) kernel

$$P_{G_\sigma}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} g\left(\frac{\mathbf{x}-\mathbf{x}_i}{\sigma}\right)$$



Small dimensions

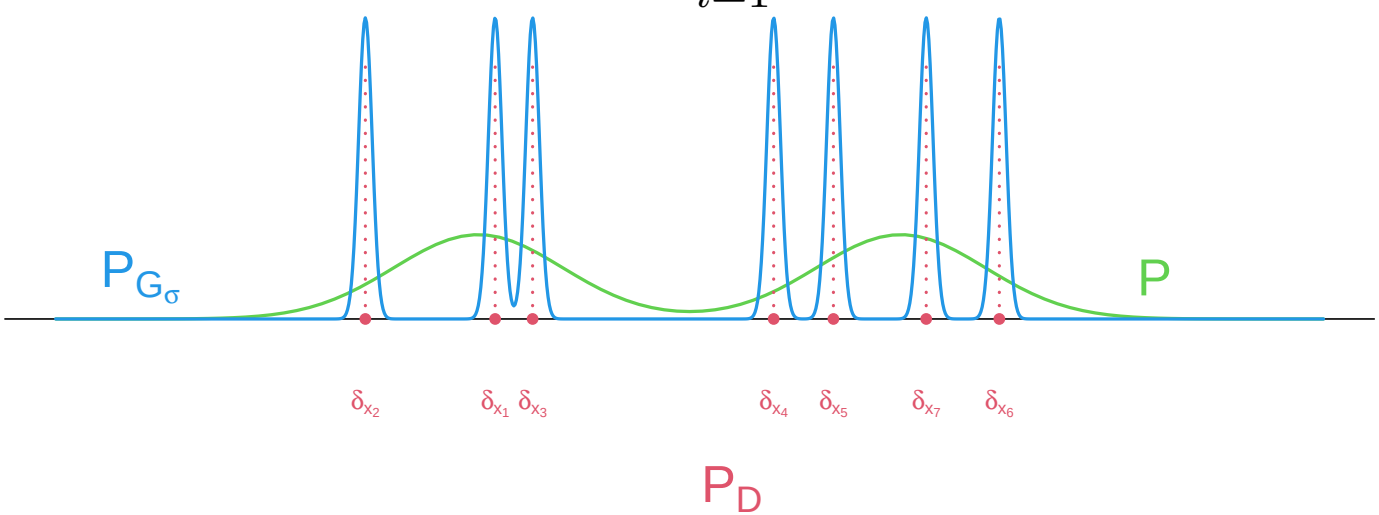
Kernel density estimation (KDE)

- Replace the empirical distribution

$$P_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathbf{x}_i}$$

by its convolution with a (gaussian) kernel

$$P_{G_\sigma}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} g\left(\frac{\mathbf{x}-\mathbf{x}_i}{\sigma}\right)$$



Small dimensions

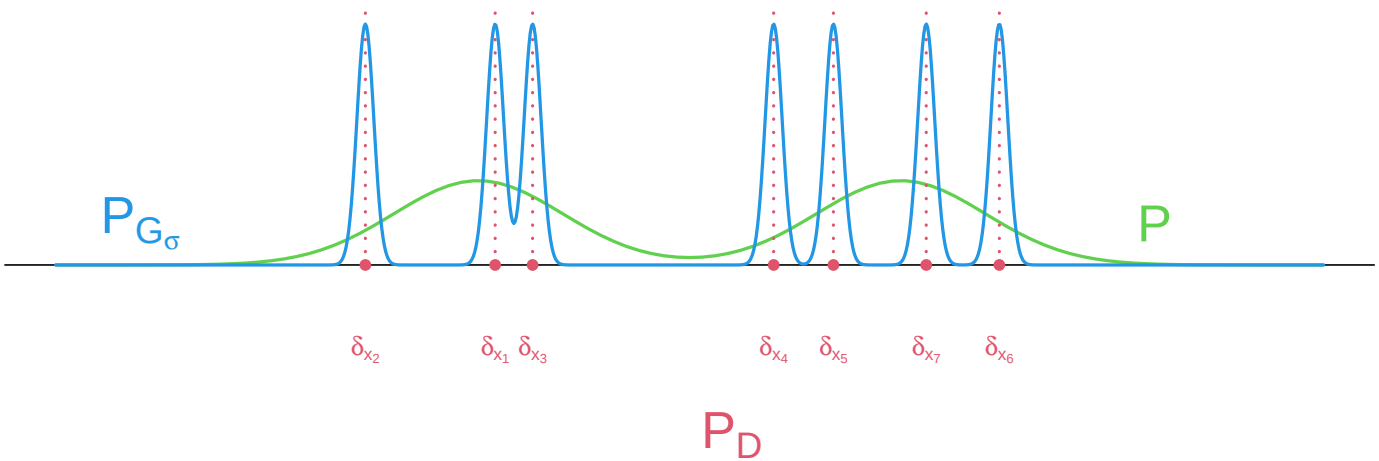
Kernel density estimation (KDE)

- Replace the empirical distribution

$$P_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathbf{x}_i}$$

by its convolution with a (gaussian) kernel

$$P_{G_\sigma}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} g\left(\frac{\mathbf{x}-\mathbf{x}_i}{\sigma}\right)$$



Small dimensions

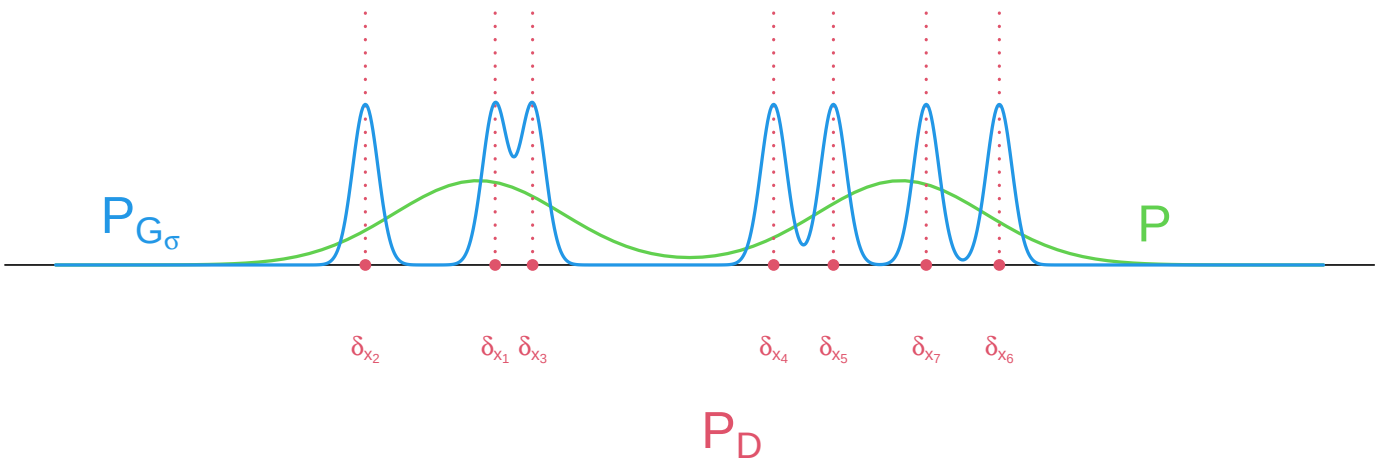
Kernel density estimation (KDE)

- Replace the empirical distribution

$$P_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathbf{x}_i}$$

by its convolution with a (gaussian) kernel

$$P_{G_\sigma}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} g\left(\frac{\mathbf{x}-\mathbf{x}_i}{\sigma}\right)$$



Small dimensions

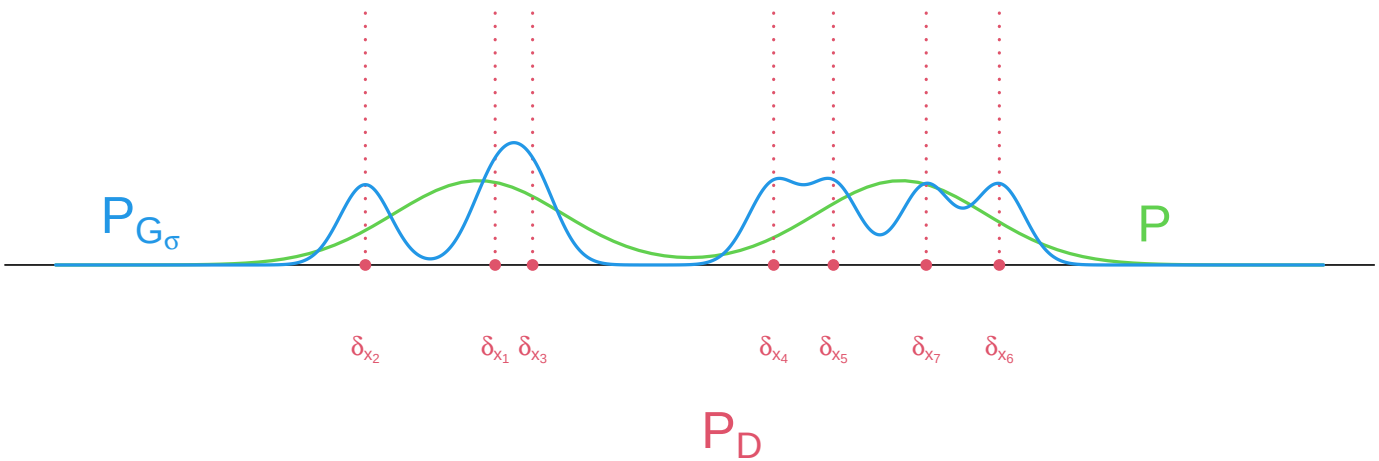
Kernel density estimation (KDE)

- Replace the empirical distribution

$$P_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathbf{x}_i}$$

by its convolution with a (gaussian) kernel

$$P_{G_\sigma}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} g\left(\frac{\mathbf{x}-\mathbf{x}_i}{\sigma}\right)$$



Small dimensions

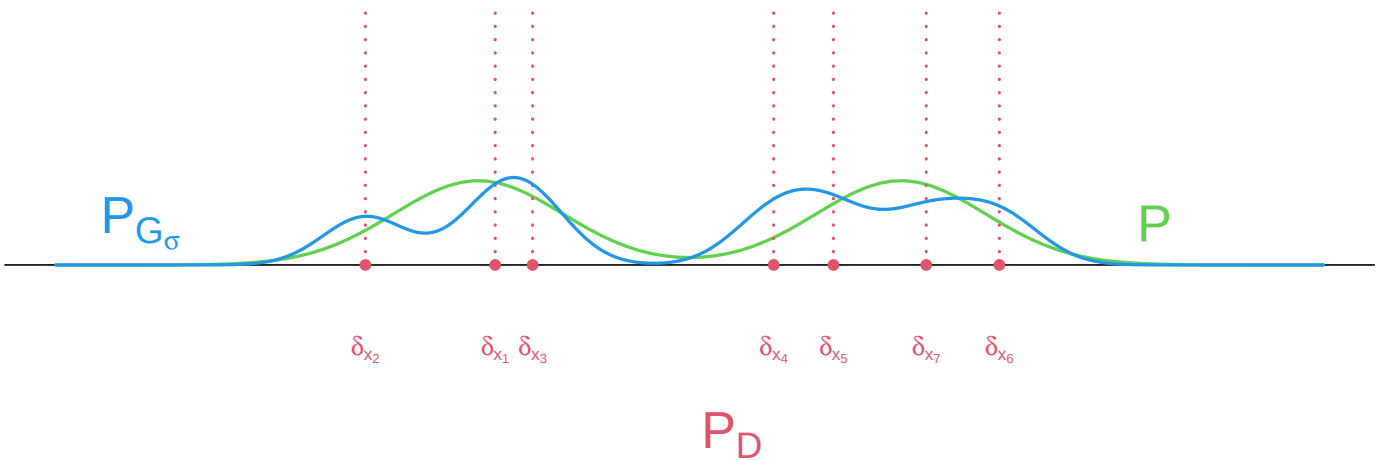
Kernel density estimation (KDE)

- Replace the empirical distribution

$$P_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathbf{x}_i}$$

by its convolution with a (gaussian) kernel

$$P_{G_\sigma}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} g\left(\frac{\mathbf{x}-\mathbf{x}_i}{\sigma}\right)$$



Small dimensions

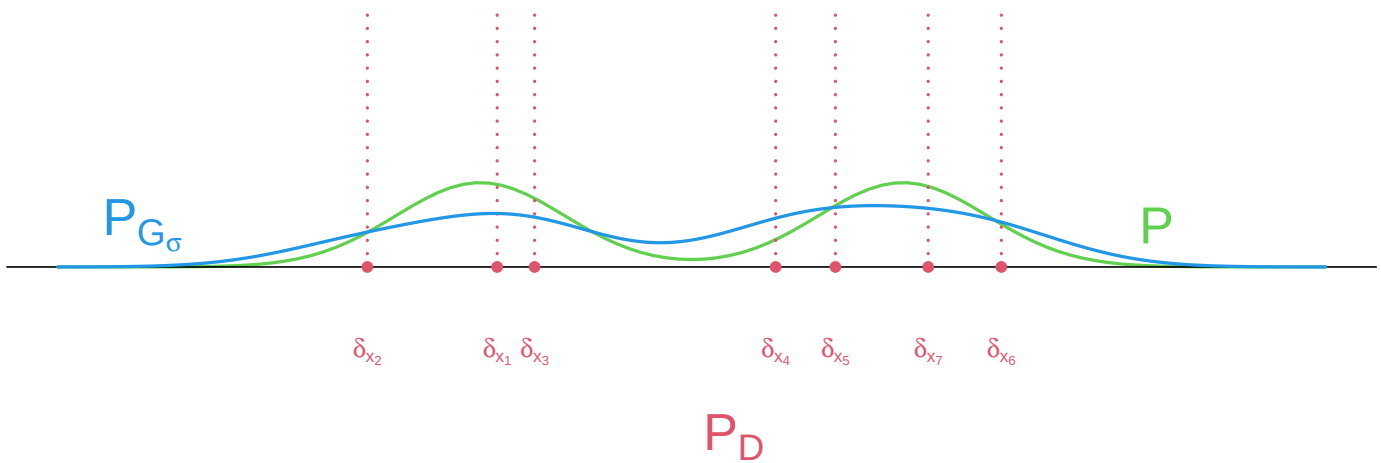
Kernel density estimation (KDE)

- Replace the empirical distribution

$$P_{\mathcal{D}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathbf{x}_i}$$

by its convolution with a (gaussian) kernel

$$P_{G_\sigma}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} g\left(\frac{\mathbf{x}-\mathbf{x}_i}{\sigma}\right)$$

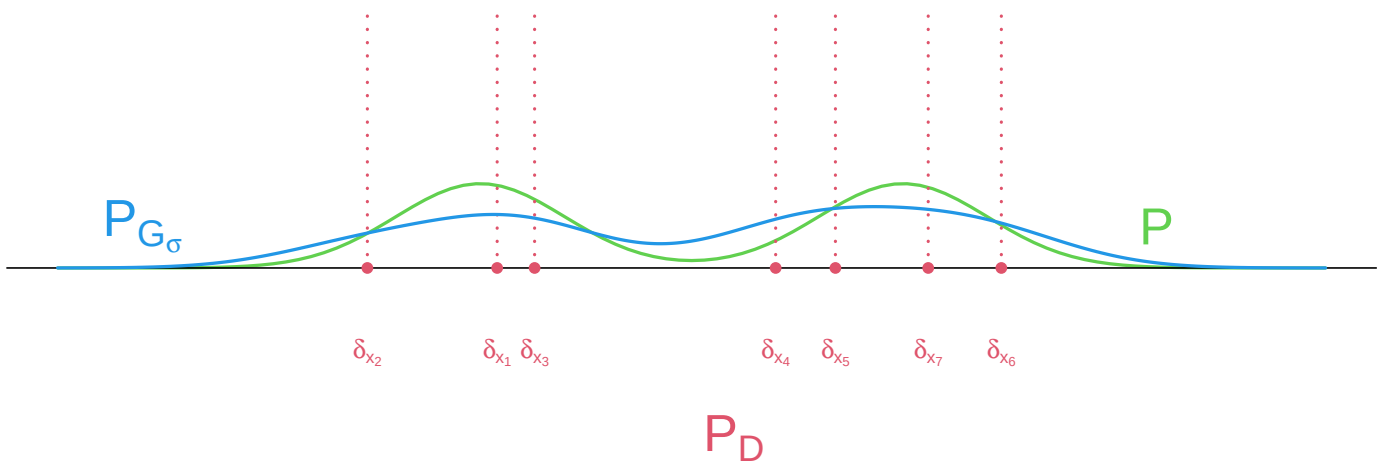


Small dimensions

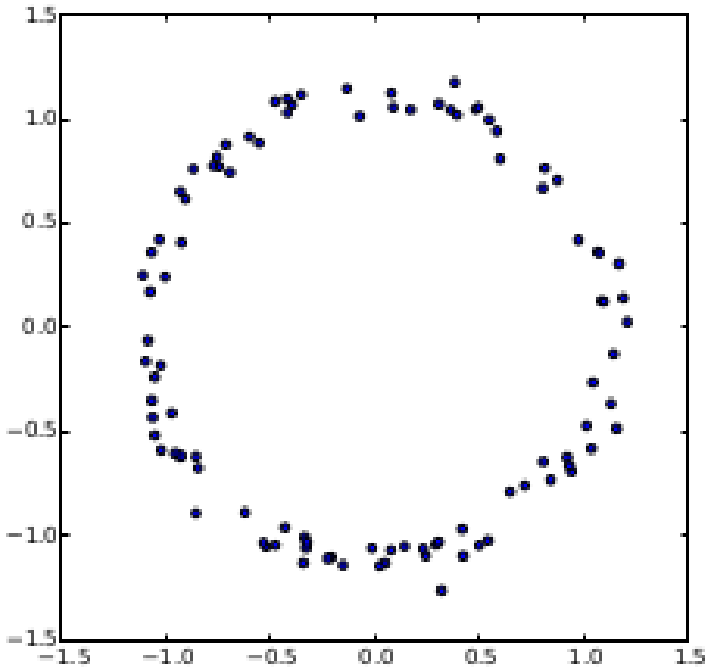
Kernel density estimation (KDE)

Algorithm:

- Choose a data $\mathbf{x}_i$ with probability $\frac{1}{N}$
- Generate a value around $\mathbf{x}_i$ by adding a gaussian noise with standard deviation σ

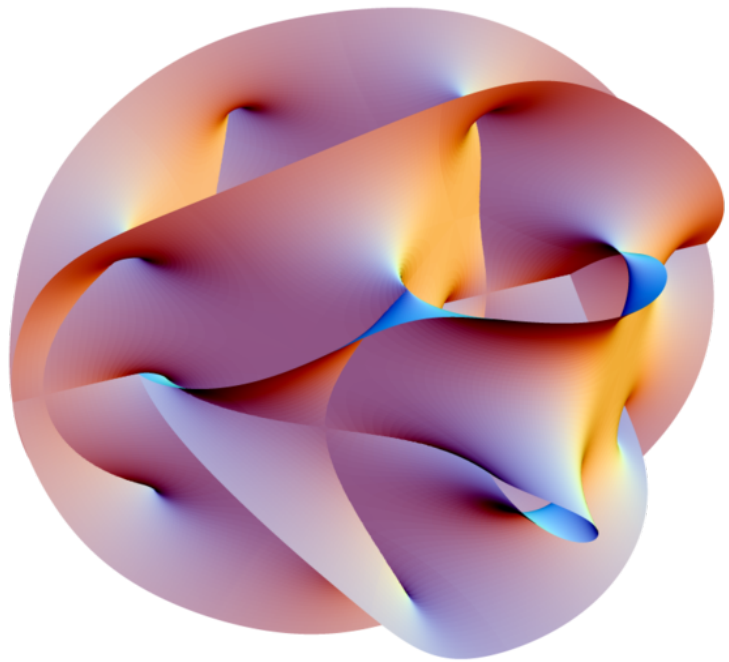


It could work in 2D



Problem: How to choose the kernel in high dimension?

The data belongs to a manifold



[https://en.wikipedia.org/wiki/File:Calabi-Yau.png]

First idea: use an anamorphosis like approach

- Generate a latent vector

$$\mathbf{z} \sim \mathcal{N}_n(0, I_n)$$

- Compute

$$\mathbf{x} = G(\mathbf{z})$$

where $G = f_{\mathbf{W}}$ is a neural network with weights $\mathbf{W}$ to determine

First idea: use an anamorphosis like approach

- Generate a latent vector

$$\mathbf{z} \sim \mathcal{N}_n(0, I_n)$$

- Compute

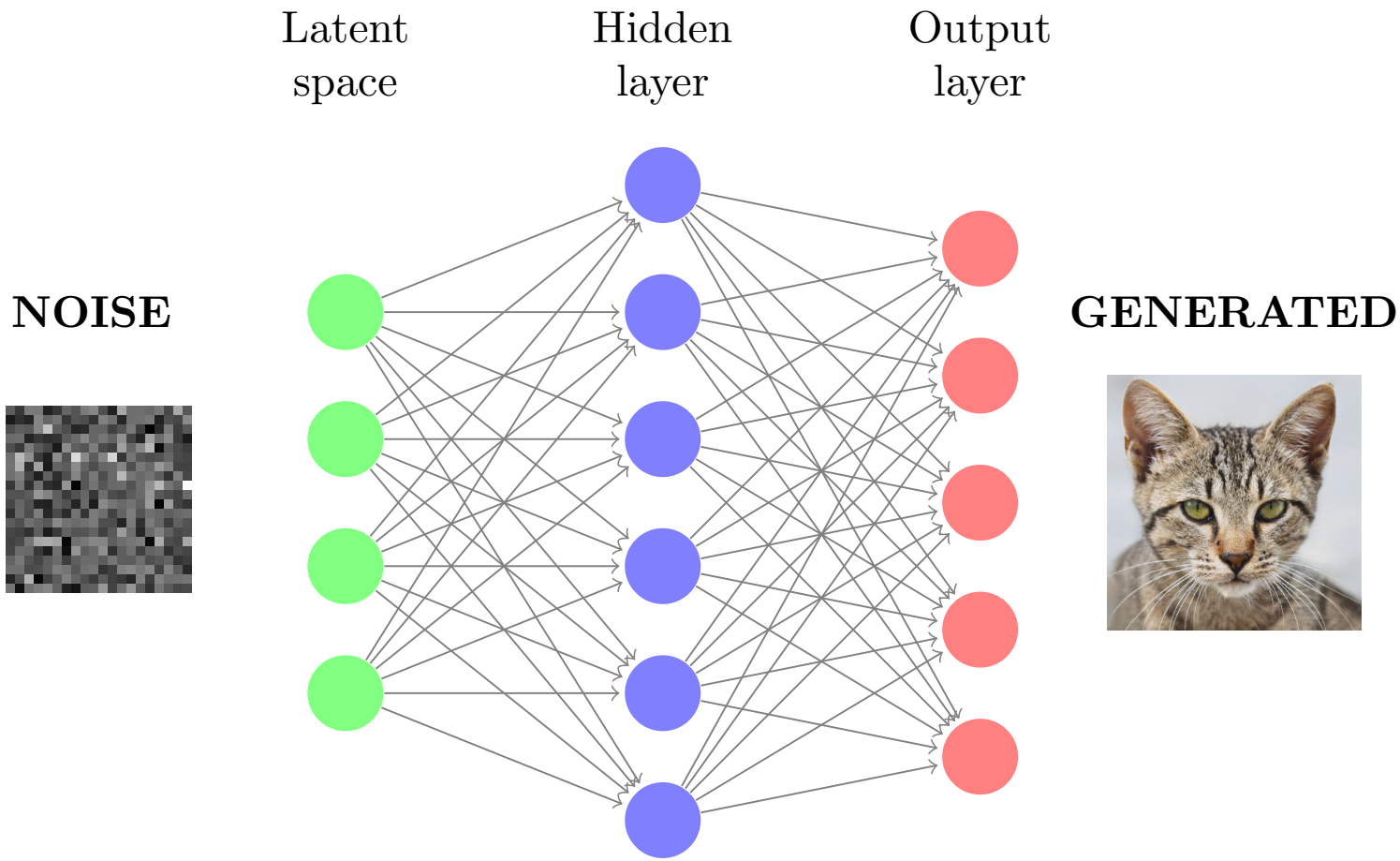
$$\mathbf{x} = G(\mathbf{z})$$

  where $G = f_{\mathbf{W}}$ is a neural network with weights $\mathbf{W}$ to determine

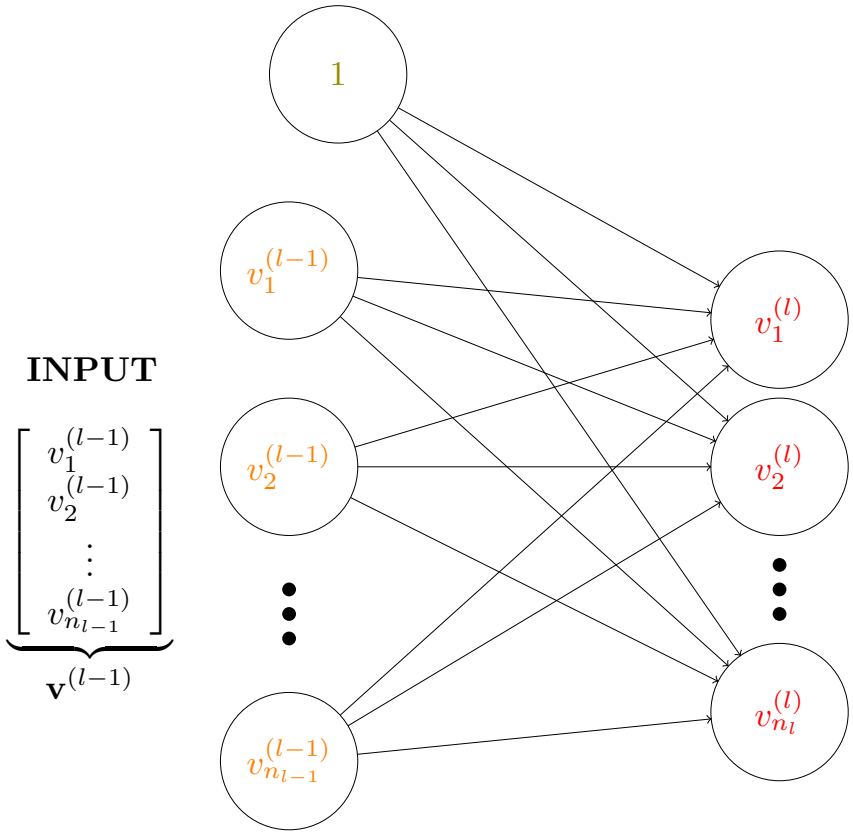
Mathematical justifications:

- Most of the distributions of $\mathbb{R}^n$ can be written as a function of a Gaussian vector of $\mathbb{R}^n$ with independant components (inversion method + decomposition of multivariate densities as a product of conditional distributions)
- By using a high-capacity neural network G, we can approximate the target function (universal approximation theorem)

Generator



Remind



INPUT

$$\underbrace{\begin{bmatrix} v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\mathbf{v}^{(l-1)}}$$

OUPUT

$$\underbrace{\begin{bmatrix} v_1^{(l)} \\ v_2^{(l)} \\ \vdots \\ v_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{v}^{(l)}} = \underbrace{\begin{bmatrix} h_{(l)}\left(s_1^{(l)}\right) \\ h_{(l)}\left(s_2^{(l)}\right) \\ \vdots \\ h_{(l)}\left(s_{n_l}^{(l)}\right) \end{bmatrix}}_{h_{(l)}(\mathbf{s}^{(l)})}$$

2 attributes for layer l:
1) A weight matrix $\tilde{W}^{(l)}$
2) An activation function $h_{(l)}$

$$s_i^{(l)} = \omega_{i,0}^{(l)} \cdot 1 + \sum_{j=1}^{n_{l-1}} \omega_{i,j}^{(l)} v_j^{(l-1)}$$

3 steps:
1) Stack a 1 to the input
2) Compute $\mathbf{s}^{(l)} = \tilde{W}^{(l)} \tilde{\mathbf{v}}^{(l-1)}$
3) Return $\mathbf{v}^{(l)} = h_{(l)}(\mathbf{s}^{(l)})$

$$\underbrace{\begin{bmatrix} s_1^{(l)} \\ s_2^{(l)} \\ \vdots \\ s_{n_l}^{(l)} \end{bmatrix}}_{\mathbf{s}^{(l)}} = \underbrace{\begin{bmatrix} \omega_{1,0}^{(l)} & \omega_{1,1}^{(l)} & \omega_{1,2}^{(l)} & \dots & \omega_{1,n_{l-1}}^{(l)} \\ \omega_{2,0}^{(l)} & \omega_{2,1}^{(l)} & \omega_{2,2}^{(l)} & \dots & \omega_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & & \\ \omega_{n_l,0}^{(l)} & \omega_{n_l,1}^{(l)} & \omega_{n_l,2}^{(l)} & \dots & \omega_{n_l,n_{l-1}}^{(l)} \end{bmatrix}}_{\tilde{W}^{(l)}} \underbrace{\begin{bmatrix} 1 \\ v_1^{(l-1)} \\ v_2^{(l-1)} \\ \vdots \\ v_{n_{l-1}}^{(l-1)} \end{bmatrix}}_{\tilde{\mathbf{v}}^{(l-1)}}$$

Example of geostatistical deep generative model

1) Cholesky algorithm (a.k.a SGS)

Gaussian vector $\mathbf{x}$ with mean vector μ and covariance matrix Σ

- Compute L (lower triangular) such as

$$\Sigma = LL^T$$

- Sample

$$\mathbf{z} \sim \mathcal{N}_n(0, I_n)$$

- Return

$$\mathbf{x} = \mu + L\mathbf{z}$$

Example of geostatistical deep generative model

1) Cholesky algorithm (a.k.a SGS)

Gaussian vector $\mathbf{x}$ with mean vector μ and covariance matrix Σ

- Compute L (lower triangular) such as

$$\Sigma = LL^T$$

- Sample

$$\mathbf{z} \sim \mathcal{N}_n(0, I_n)$$

- Return

$$\mathbf{x} = \mu + L\mathbf{z}$$

- Anamorphosis can be handled by the activation function

Example of geostatistical deep generative model

2) Chebyshev algorithm (a.k.a algo de Mike)

SPDE

- Compute $P(M)$ such as

$$\Sigma = P(M)^2$$

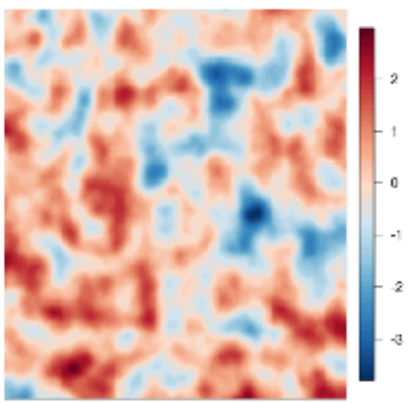
with M very sparse

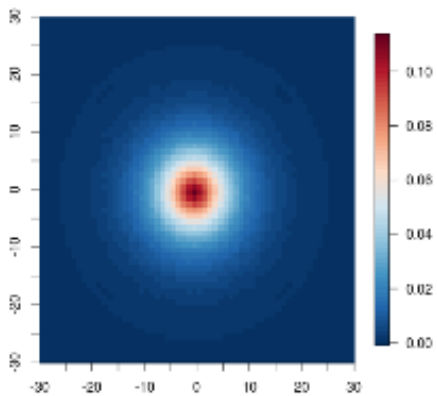
- Sample

$$\mathbf{z} \sim \mathcal{N}_n(0, I_n)$$

- Return

$$\mathbf{x} = \mu + P(M)\mathbf{z}$$





Example of geostatistical deep generative model

3) Chebyshev algorithm implementation

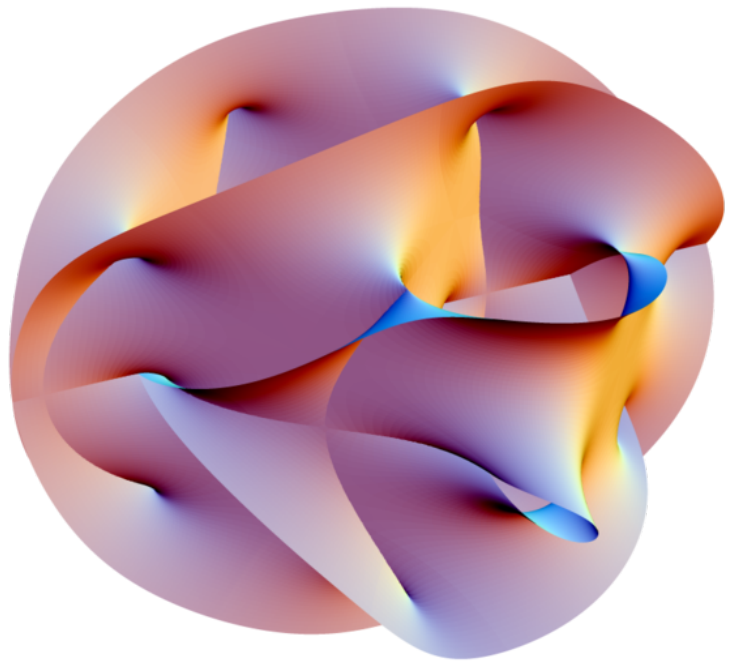
Hörner scheme on $P(X) = \sum_{i=0}^{K} a_i X^i$

- Initialization $\mathbf{x}^{(K)} = a_K \mathbf{z}$
- Iterate
$$\mathbf{x}^{(i)} = M\mathbf{x}^{(i+1)} + a_i \mathbf{z}$$
- Return $\mathbf{x}^{(0)}$

Very deep neural network with convolutional layers and few parameters

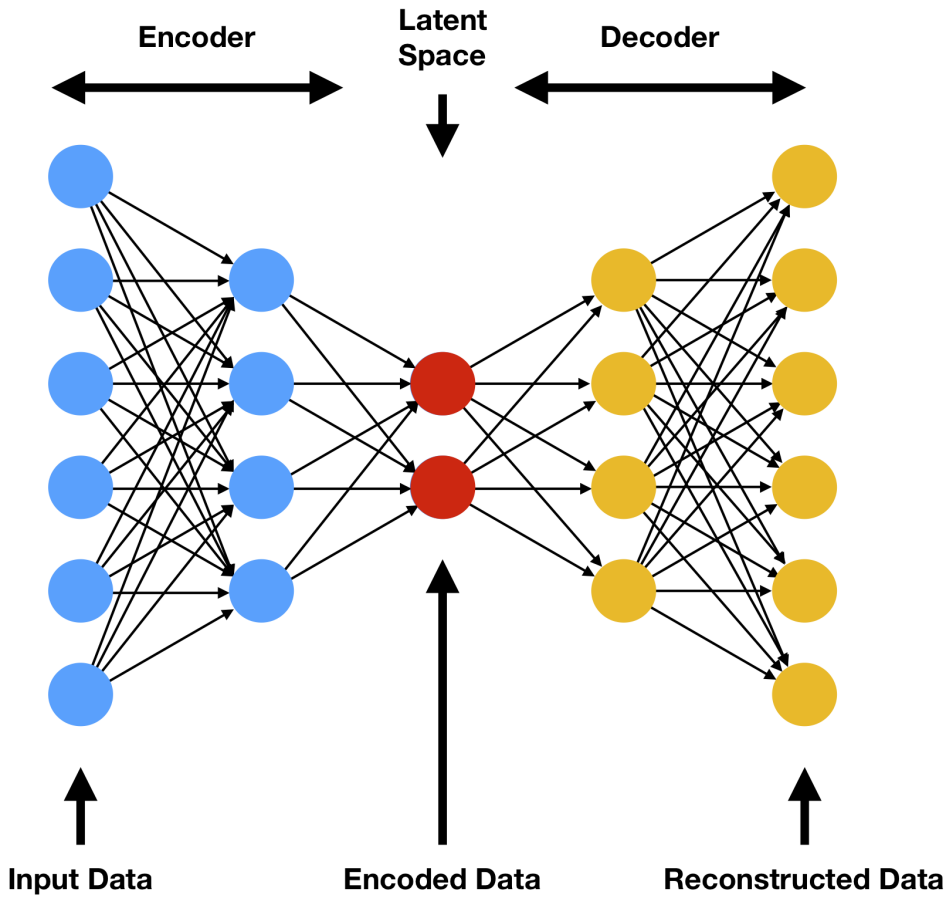
Problem: How to handle the large number of parameters?

The data belongs to a manifold



[https://line.17qq.com/articles/cmmcohgcpv.html]

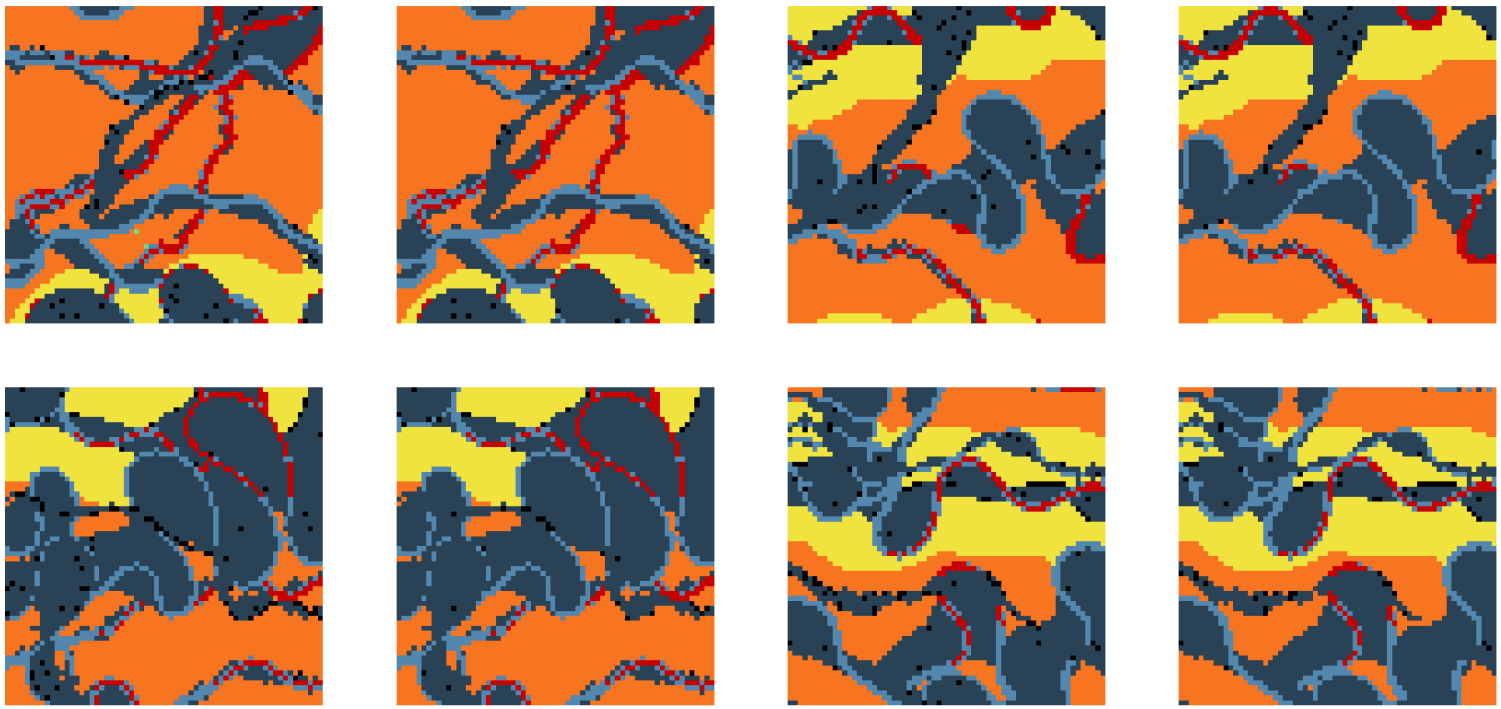
Second idea: Dimension reduction



[https://www.compthree.com/blog/autoencoder/]

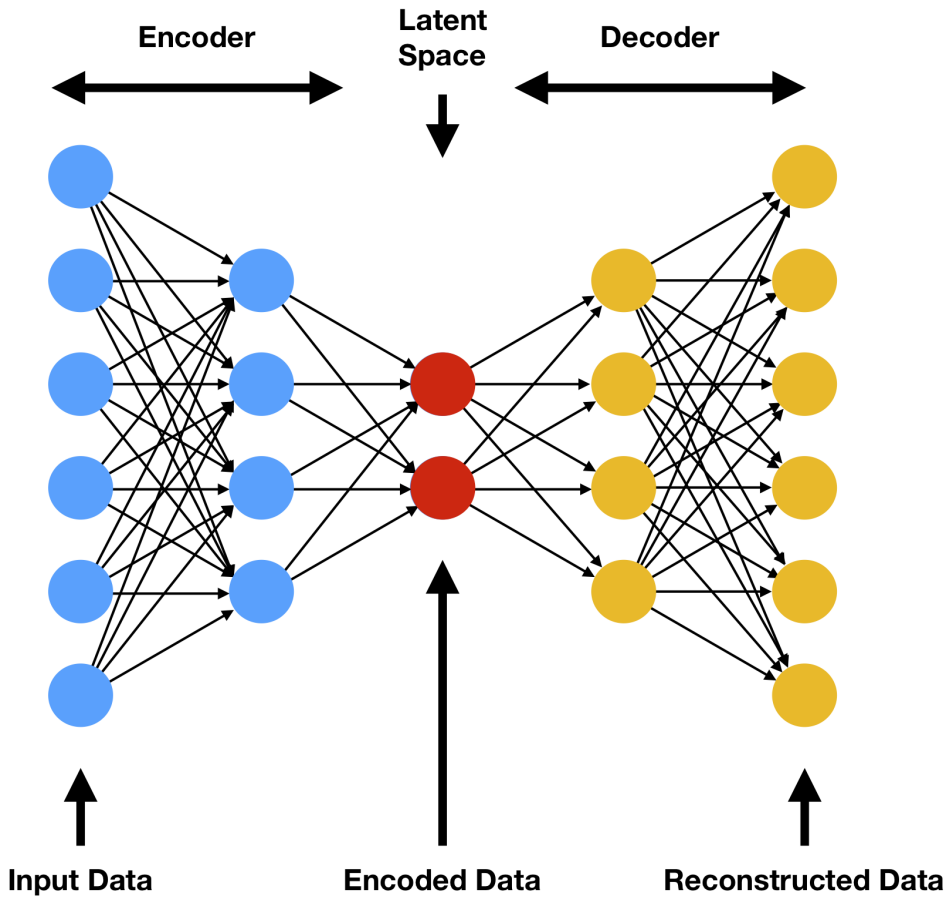
The network "learns" a low dimensional representation of the inputs

Example: compression of Flumy images

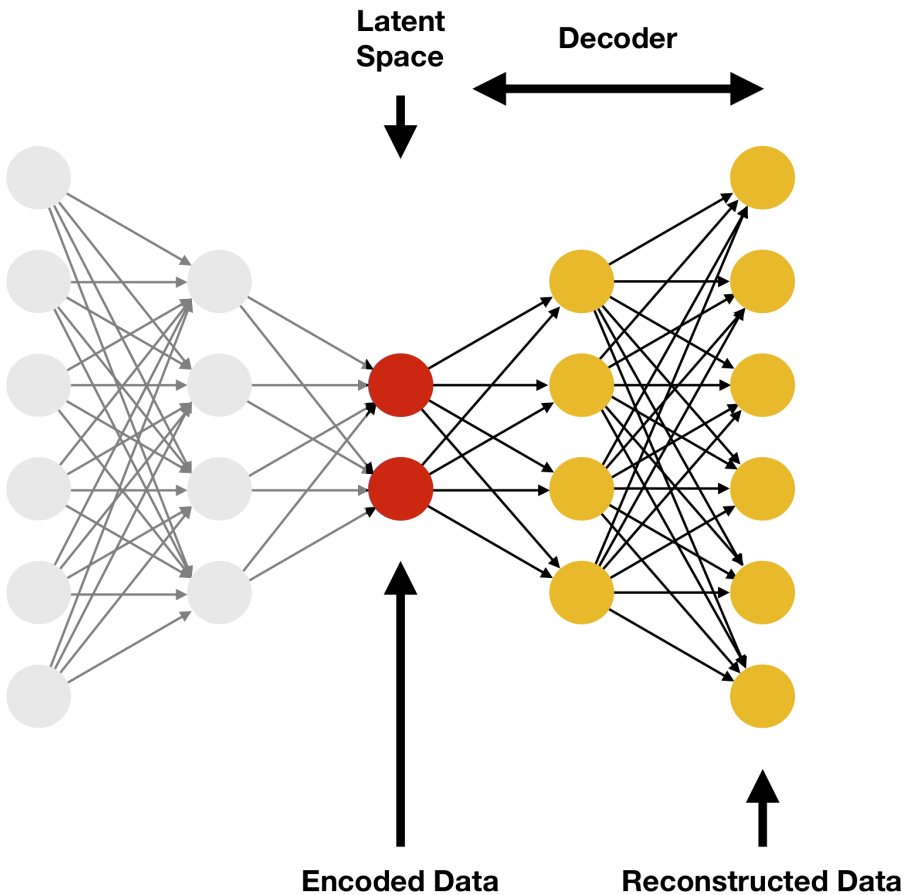


Original 64×64 Flumy images (left) and reconstructed version (right) after a compression in a latent space of 121 units

Simulate in the latent space



Simulate in the latent space



A last geostatistical example

Spectral method (a.k.a Shinozuka algorithm)

Simulate a Gaussian random field with Gaussian covariance function $C(h) = e^{-h^2}$ on a $n_1 \times n_2$ grid

- Simulate $(\omega_1^{(1)}, \omega_2^{(1)}), \ldots, (\omega_1^{(K)}, \omega_2^{(K)})$, K independent Gaussian vectors
- Simulate $\phi^{(1)}, \ldots, \phi^{(K)}$, K independent and uniform values on $[0, 1]$ (independent of the $\omega_j^{(i)}$)
- For each node (u_1, u_2), return

$$\mathbf{x}(u_1, u_2) = \frac{\sqrt{2}}{\sqrt{K}} \sum_{i=1}^{K} \cos(\omega_1^{(i)} u_1 \sqrt{2} + \omega_2^{(i)} u_2 \sqrt{2} + 2\pi\phi^{(i)})$$

A last geostatistical example

Spectral method (a.k.a Shinozuka algorithm)

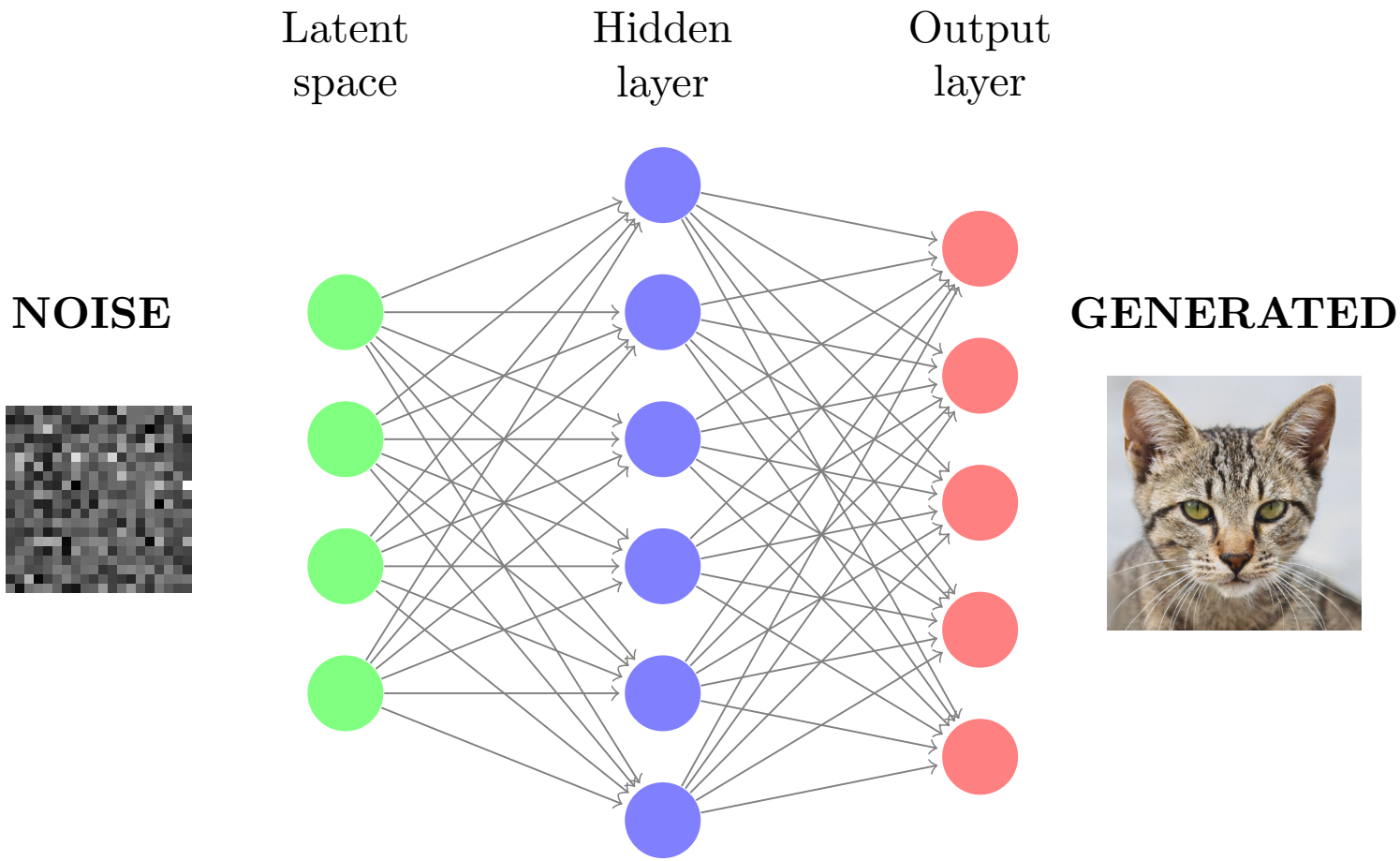
Simulate a Gaussian random field with Gaussian covariance function $C(h) = e^{-h^2}$ on a $n_1 \times n_2$ grid

- Simulate $(\omega_1^{(1)}, \omega_2^{(1)}), \ldots, (\omega_1^{(K)}, \omega_2^{(K)})$, K independent Gaussian vectors
- Simulate $\phi^{(1)}, \ldots, \phi^{(K)}$, K independent and uniform values on $[0, 1]$ (independent of the $\omega_j^{(i)}$)
- For each node (u_1, u_2), return

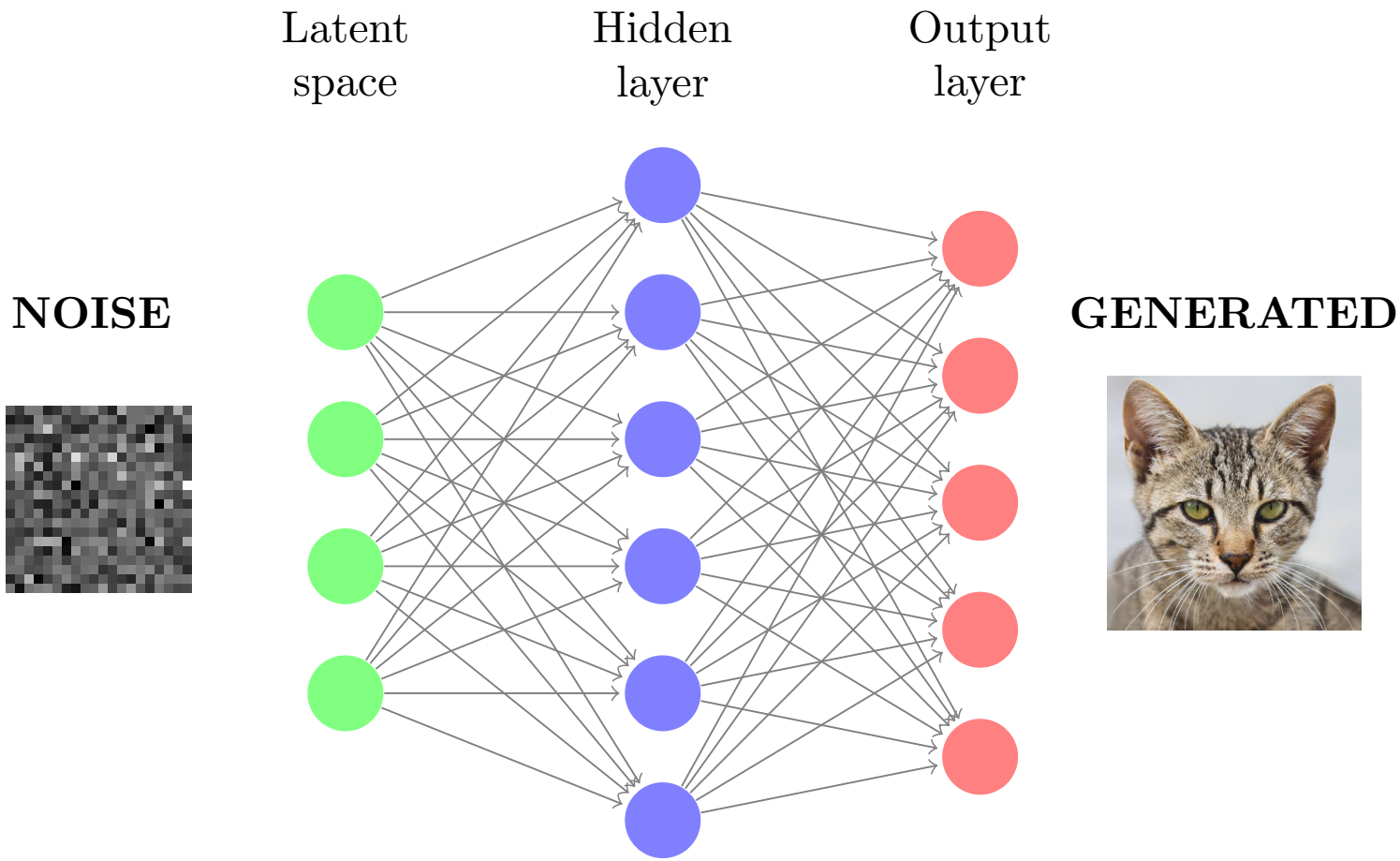
$$\mathbf{x}(u_1, u_2) = \frac{\sqrt{2}}{\sqrt{K}} \sum_{i=1}^{K} \cos(\omega_1^{(i)} u_1 \sqrt{2} + \omega_2^{(i)} u_2 \sqrt{2} + 2\pi\phi^{(i)})$$

A very sparse neural network with $3K$ inputs, one hidden layer (activation function cosinus) with $K \times n_1 \times n_2$ outputs and an output layer with $n_1 \times n_2$ output (and linear activation function)

Generator



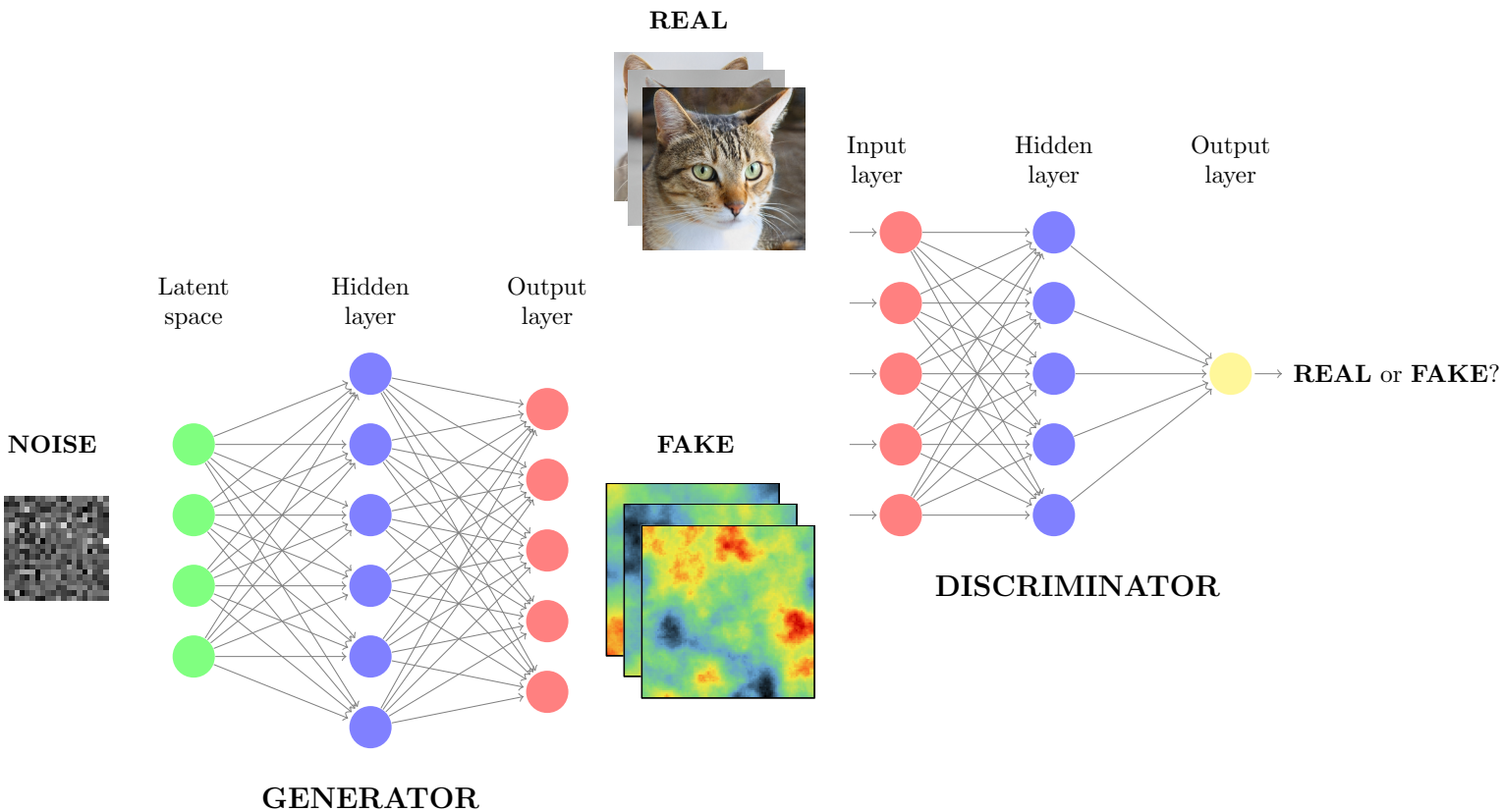
Generator



How to train the network?

Generative Adversarial Network (GAN)

A two players game



Generative Adversarial Networks (GAN)

Two adversarial neural networks

- A generator $X = G_\theta(z)$ where z is a Gaussian noise $\mathcal{N}(0, I_d)$ and θ are the weights
- A discriminator $D_\phi(X)$ which classifies images "true" or "fake" (ϕ are the weights)

Generative Adversarial Networks (GAN)

Two adversarial neural networks

- A generator $X = G_\theta(z)$ where z is a Gaussian noise $\mathcal{N}(0, I_d)$ and θ are the weights
- A discriminator $D_\phi(X)$ which classifies images "true" or "fake" (ϕ are the weights)

- Idea: Train G_θ and D_ϕ in order that G_θ misleads D_ϕ and D_ϕ tries to become better to detect fakes

Generative Adversarial Networks (GAN)

Two adversarial neural networks

- A generator $X = G_\theta(z)$ where z is a Gaussian noise $\mathcal{N}(0, I_d)$ and θ are the weights
- A discriminator $D_\phi(X)$ which classifies images "true" or "fake" (ϕ are the weights)

- D_ϕ is close to 1 if the discriminator classifies an image as true and close to 0 otherwise

Generative Adversarial Networks (GAN)

Two adversarial neural networks

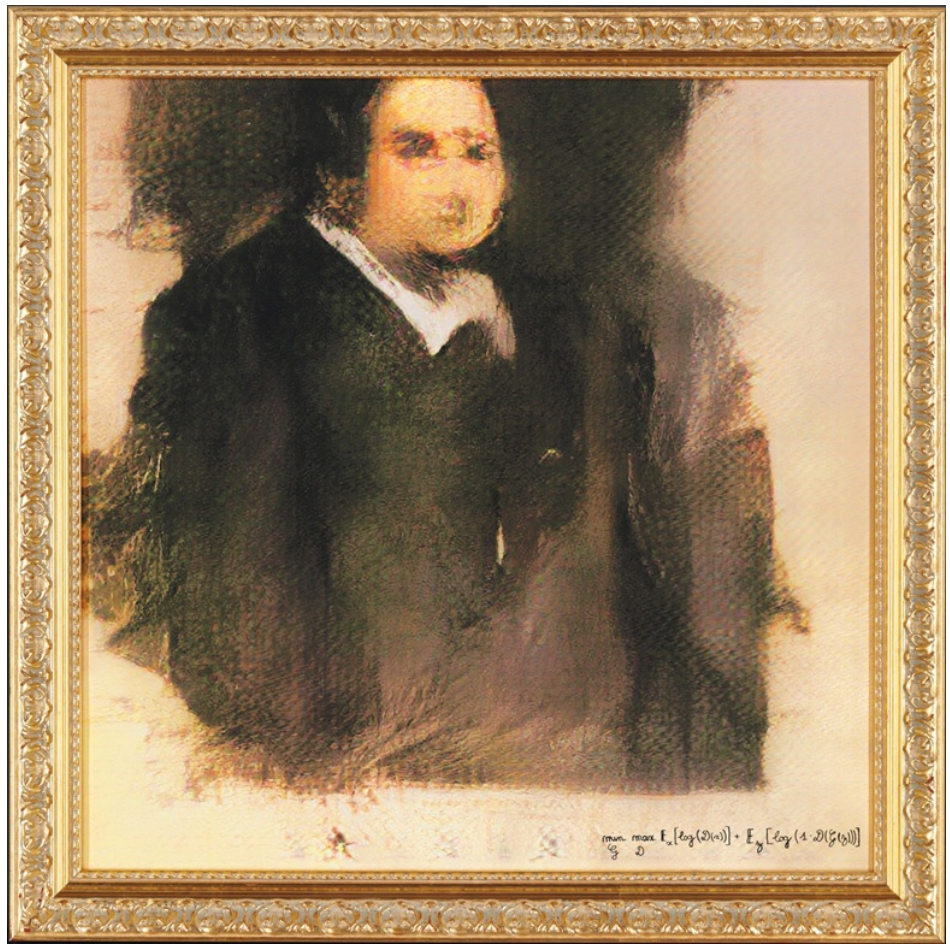
- A generator $X = G_\theta(z)$ where z is a Gaussian noise $\mathcal{N}(0, I_d)$ and θ are the weights
- A discriminator $D_\phi(X)$ which classifies images "true" or "fake" (ϕ are the weights)

- Minmax game with value function

$$\min_{G_\theta} \max_{D_\phi} V(D_\phi, G_\theta)$$

with

$$V(D_\phi, G_\theta) = E_{X \sim \text{Data}}[\log D_\phi(X)] + E_{z \sim \mathcal{N}(0, I_d)}[\log(1 - D_\phi(G_\theta(z))]$$


$\min_G \max_D E_x[\log(D(x))] + E_y[\log(1 - D(G(y)))]$

Training

Goodfellow et al. (2014)

for Number of training iterations **do**

  for k states **do**

    Sample minibatch of m_b noise samples $\{z_1, \ldots, z_{m_b}\} \sim \mathcal{N}(0, I_d)$

    Sample minibatch of m_b examples $\{x_1, \ldots, x_{m_b}\} \sim \text{Data}$

    Update the discriminator by ascending its stochastic gradient:

$$\nabla_\phi \frac{1}{m_b} \sum_{i=1}^{m_b} [\log D_\phi(x_i) + \log(1 - D_\phi(G_\theta(z_i)))]$$

  end for

  Sample minibatch of m_b noise samples $\{z_1, \ldots, z_{m_b}\} \sim \mathcal{N}(0, I_d)$

  Update the generator by descending its stochastic gradient

$$\nabla_\theta \frac{1}{m_b} \sum_{i=1}^{m_b} \log(1 - D_\phi(G_\theta(z_i)))$$

end for

Theory

GAN cost function:

$$V(D,G) = E_{X\sim \text{Data}}[\log D(X)] + E_{z\sim\mathcal{N}(0,I_d)}[\log(1-D(G(z))]$$

For a given discriminator D, it measures a dissimilarity between the data distribution $P_{\mathcal{D}}$ and the generated distribution P_G:

$$L_D(P_{\mathcal{D}},P_G) = E_{X\sim P_{\mathcal{D}}}[\log D(X)] + E_{X\sim P_G}[\log(1-D(X))]$$

which is equal to

$$L_D(P_{\mathcal{D}},P_G) = \int (P_{\mathcal{D}}(x)\log D(x) + P_G(x)\log(1-D(x)))\,dx$$

which is minimum for

$$D^\star(x) = \frac{P_{\mathcal{D}}(x)}{P_{\mathcal{D}}(x)+P_G(x)}$$

Theory

Maximizing

$$L_{D^\star}(P_{\mathcal{D}}, P_G)$$

with respect to P_G, we obtain that the global optimum of $L_{D^\star}$ is reached for

$$P_G^\star = P_{\mathcal{D}}$$

leading to $D^\star(x) = \frac{1}{2}$ and

$$\begin{aligned} L_{D^\star}(P_{\mathcal{D}}, P_G^\star) &= \int \left(P_{\mathcal{D}}(x) \log D^\star(x) + P_G^\star(x) \log(1 - D^\star(x))\right) dx \\ &= -2 \log 2 \end{aligned}$$

Theory

Maximizing

$$L_{D^\star}(P_{\mathcal{D}}, P_G)$$

with respect to P_G, we obtain that the global optimum of $L_{D^\star}$ is reached for

$$P_G^\star = P_{\mathcal{D}}$$

leading to $D^\star(x) = \frac{1}{2}$ and

$$\begin{aligned} L_{D^\star}(P_{\mathcal{D}}, P_G^\star) &= \int \left(P_{\mathcal{D}}(x) \log D^\star(x) + P_G^\star(x) \log(1 - D^\star(x))\right) dx \\ &= -2 \log 2 \end{aligned}$$

Under quite strong assumptions on the training algorithm, it converges towards the global optimum. In practice, it is not so easy.

Difficult training

- Instabilities (min-max game)
- Disjoint supports of the two distributions often occur (low dimensional manifolds)
- Vanishing gradient
- Mode collapse

Idea: Change the metric $\rightarrow$ Wasserstein GAN (Arjovsky et al., 2017)

Theory

- Kullback-Leibler Divergence

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

- Jensen-Shannon Divergence

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$

- GAN cost function with optimal discriminator

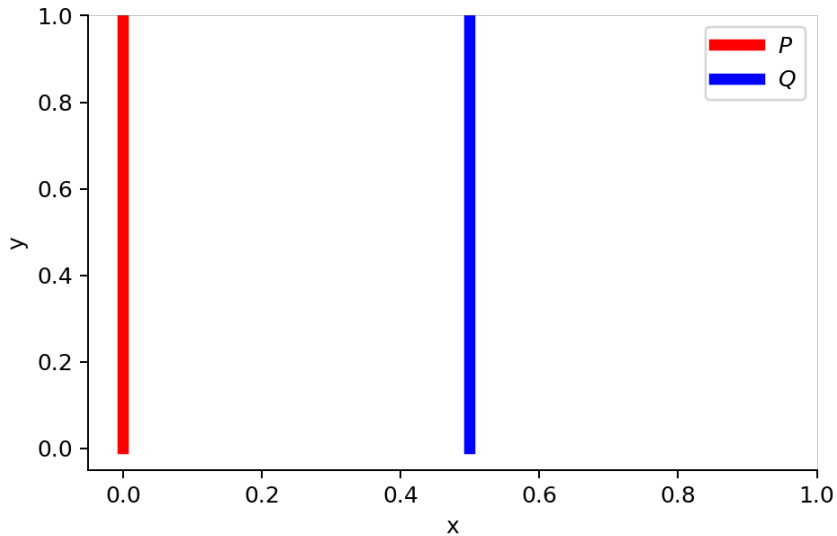
$$L(p, q) = 2D_{JS}(q||p) - 2\log 2$$

- Wasserstein (or earth mover) distance

$$W(p, q) = \inf_{\gamma \sim \Pi(p,q)} E_{(x,y)\sim\gamma}[||x - y||]$$

where $\Pi(p, q)$ is the set of "bivariate" distributions with p and q as marginal and $||.||$ is a norm over $\mathbb{R}^n$.

Example of two distributions P and Q



- (x, y) with $x = 0$ and $y \sim U[0, 1]$ with P
- (x, y) with $x = \theta$ and $y \sim U[0, 1]$ with Q

$D_{JS}(P||Q) = \frac{1}{2}$ and $W(P||Q) = \theta$

Kantorovich-Rubinstein Duality

- Wasserstein

$$W(p,q) = \inf_{\gamma \sim \Pi(p,q)} E_{(x,y)\sim\gamma}[||x-y||]$$

  where $\Pi(p,q)$ is the set of "bivariate" distributions with p and q as marginal and $||.||$ is a norm over $\mathbb{R}^n$.

- It can be rewritten

$$W(p,q) = \sup_{f \in L_{1-lipz}} E_{x\sim p}[f(x)] - E_{x\sim q}[f(x)]$$

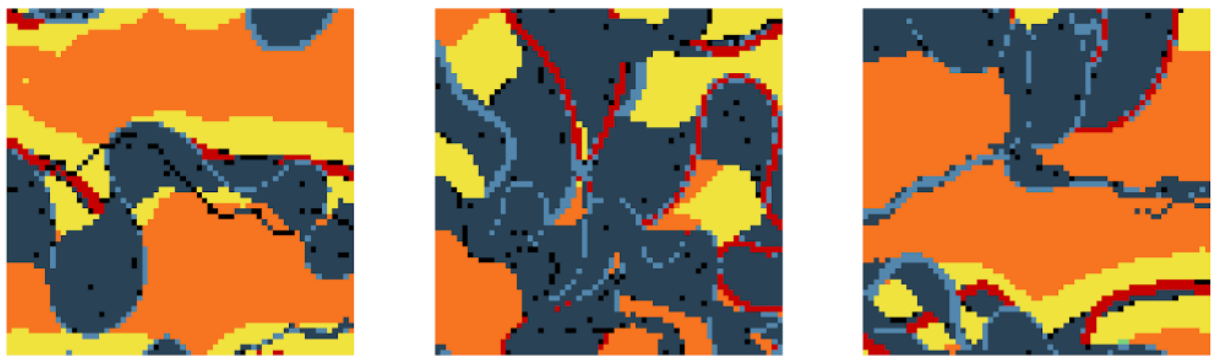
- f will be replaced by a neural network and $f \in L_{1-lipz}$ is "controlled" by using gradient penalty (WGAN-GP). See Gulrajani et al. (2017)

To play

Google GAN lab

Flumy generator

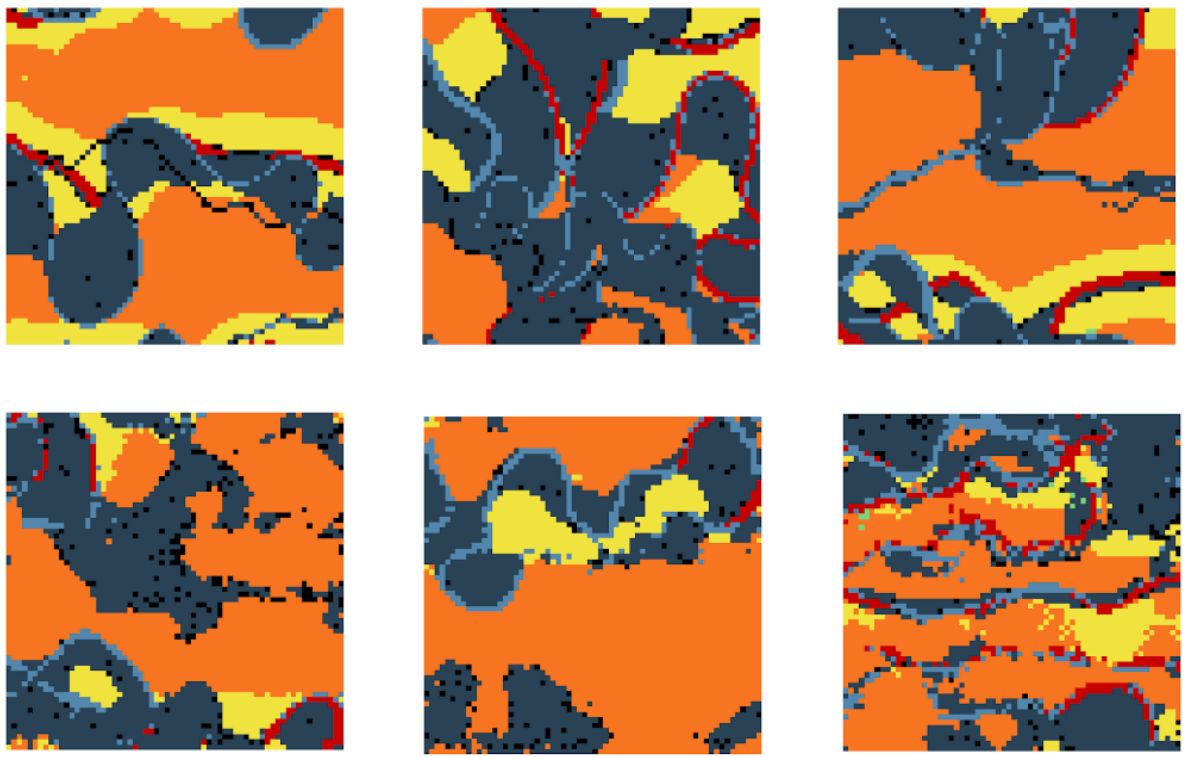
- Generator is trained on 10000 small 64×64 training images
- The architecture of the generator (fully convolutional neural network) leads to stationary process



Example of 64×64 Flumy generated images

Results

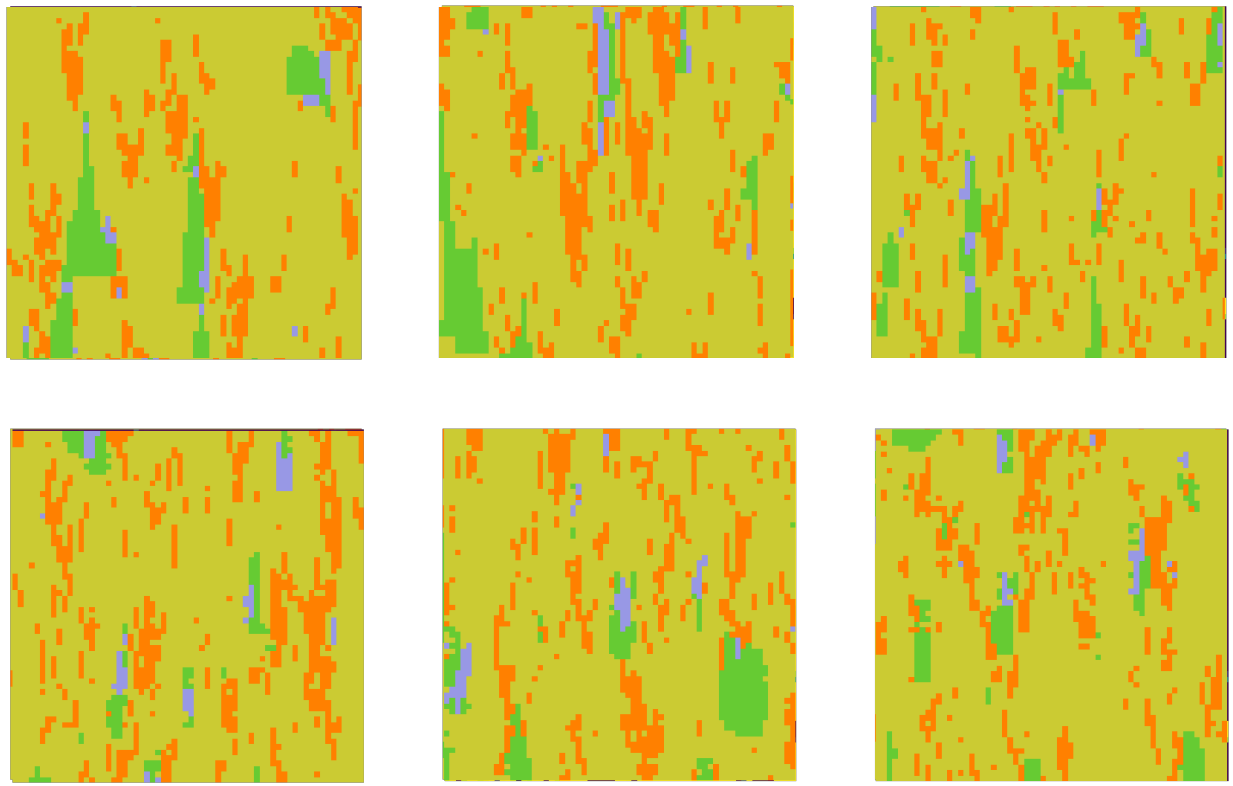
Horizontal section



Top: Flumy. Bottom: GAN

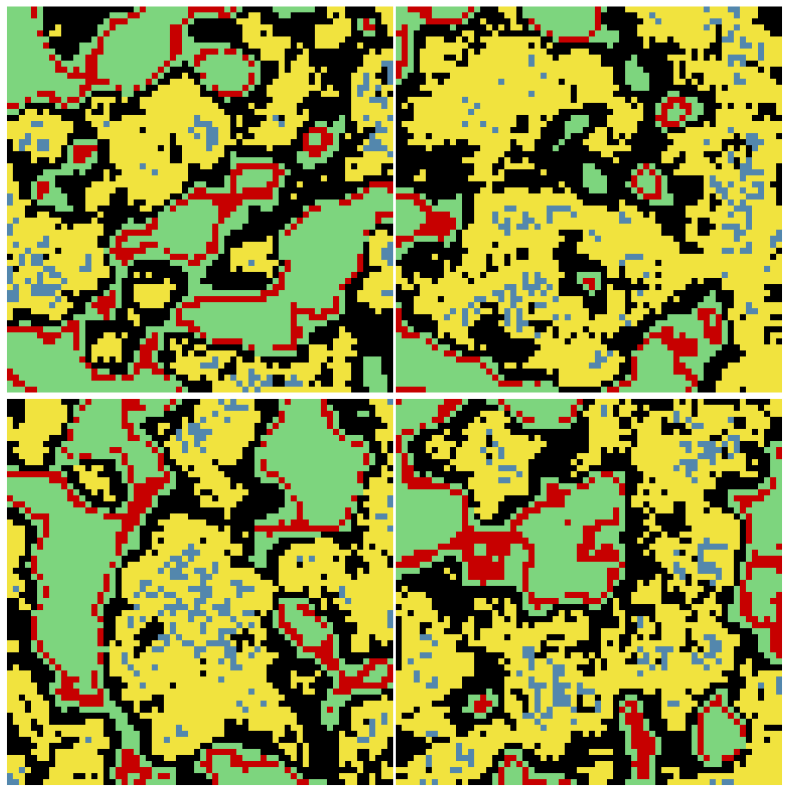
Results

Vertical section



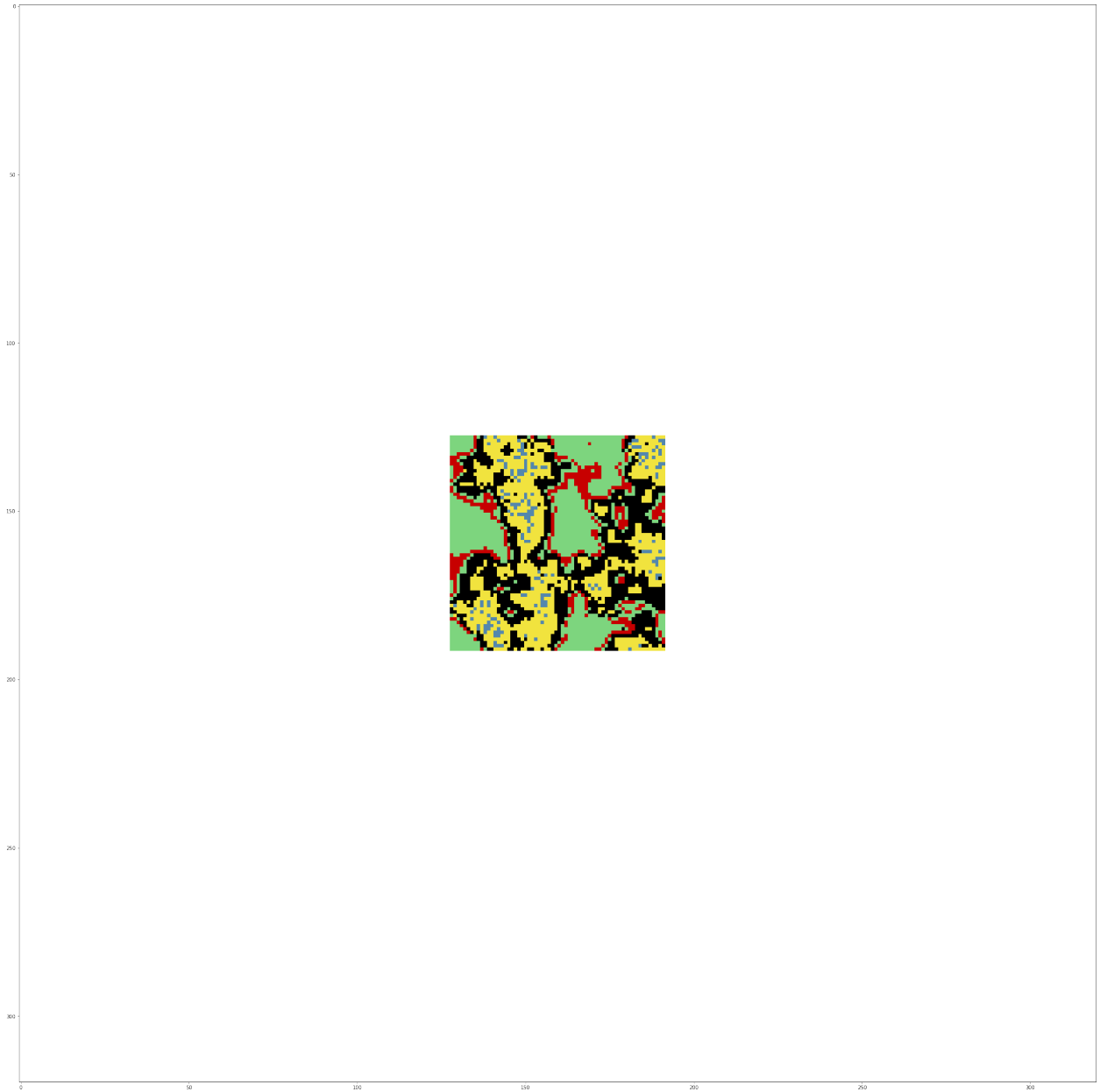
Top: Flumy. Bottom: GAN

Example on Plurigaussian

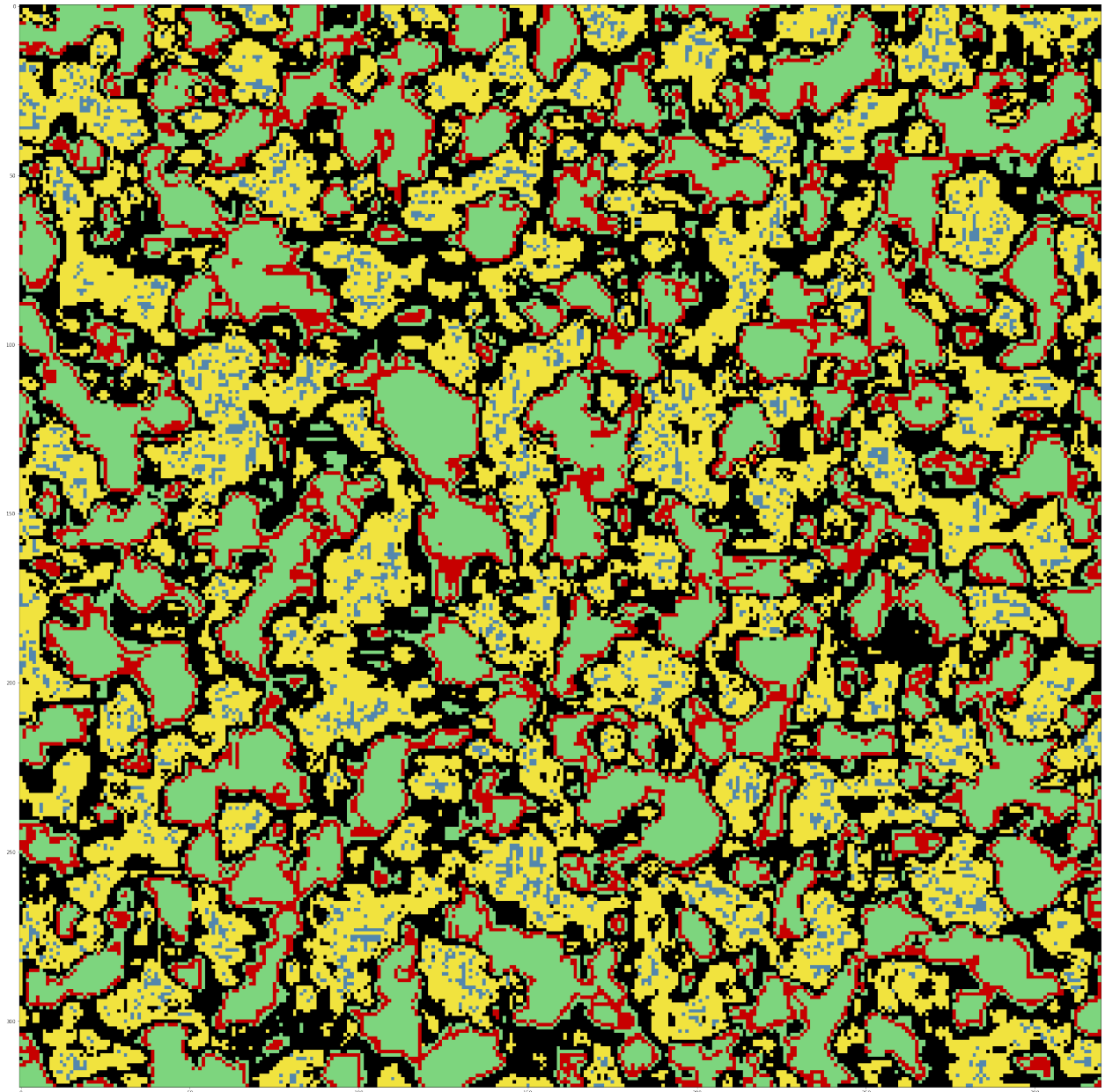


4 images among 1000 training images

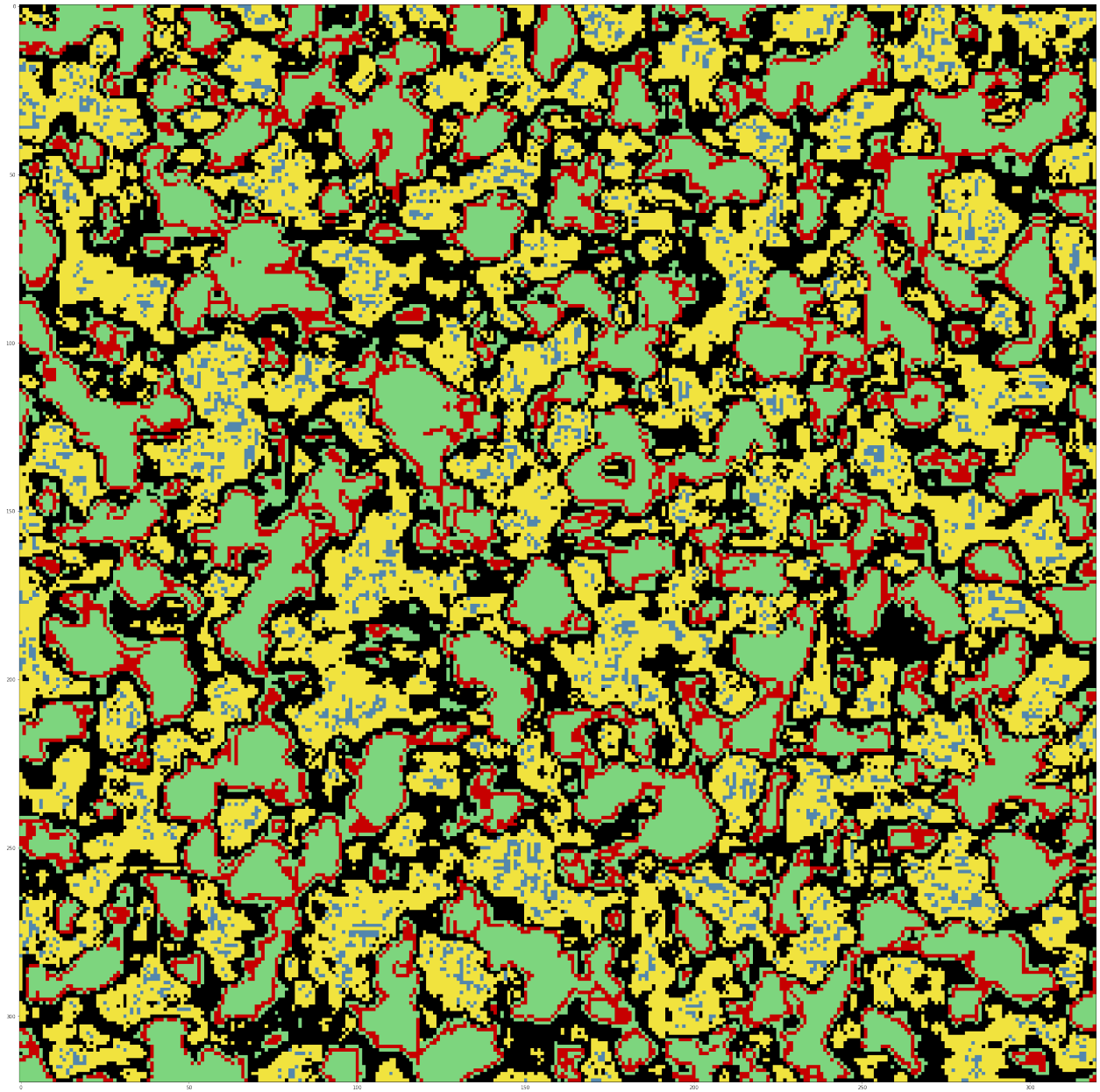
Local adaptation



Local adaptation



Local adaptation



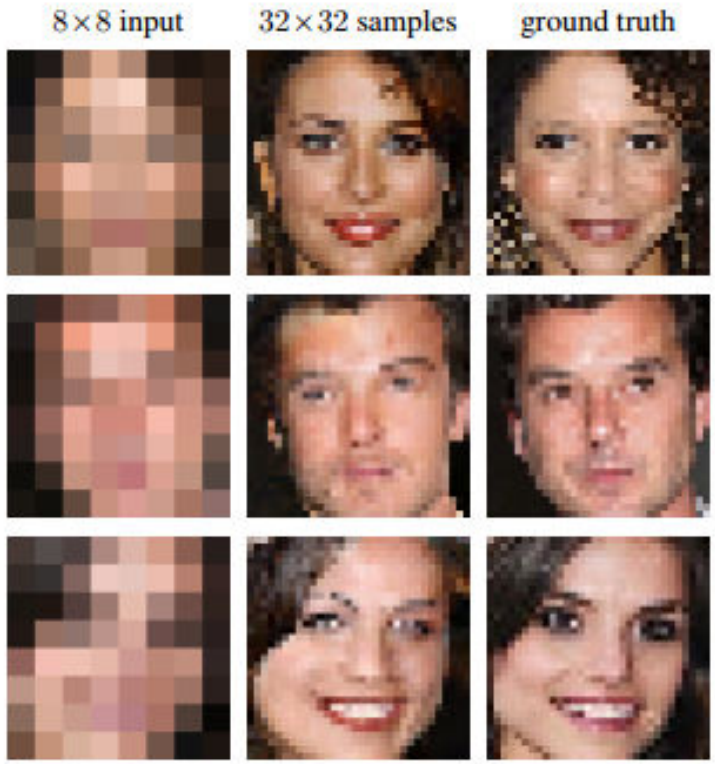
Conditioning

Variational bayesian formulation

Perspectives

Condition by proportions

Dahl et al. (2017)

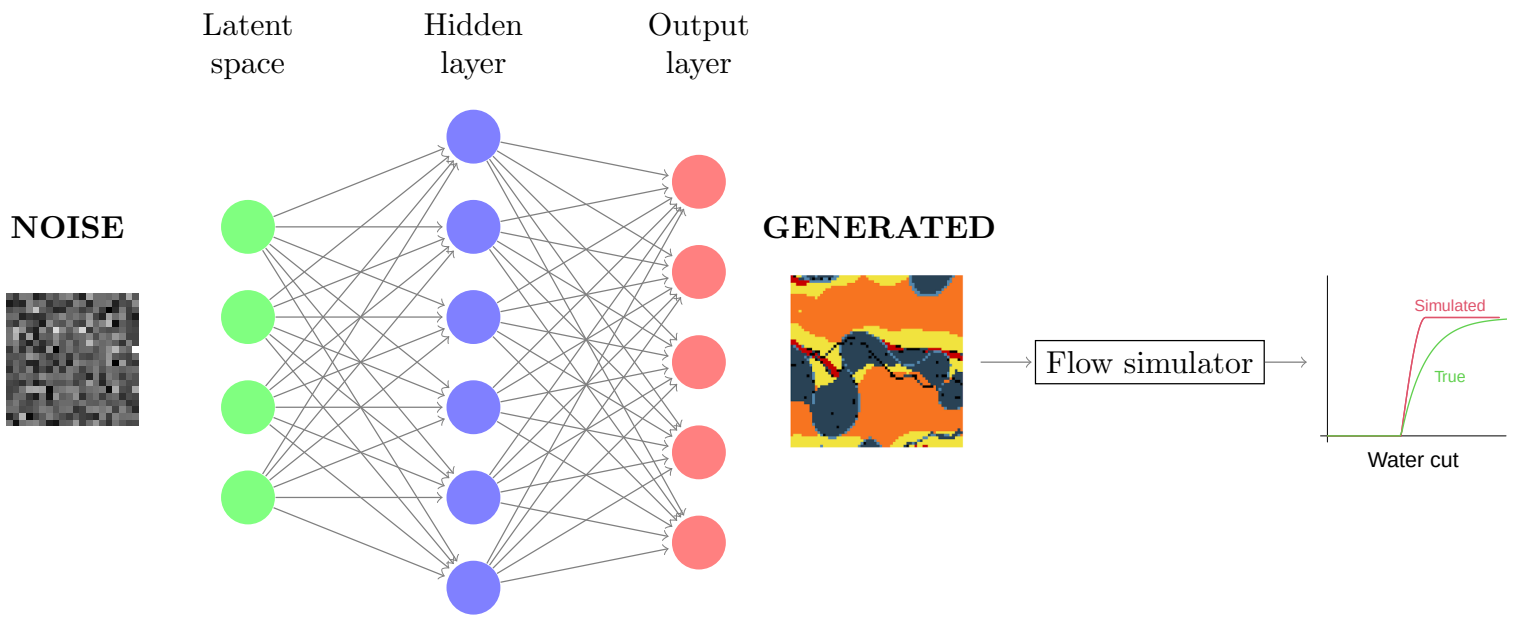


Super Resolution GAN (SRGAN)

Inversion

Mosser et al. (2020)

- Condition by indirect data (seismic, production, ...)
- Minimize the misfit function with respect to the latent coordinates
- Use back-propagation, possibly coupled with the adjoint-state method



References

Arjovsky, M., S. Chintala, and L. Bottou (2017). Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. PMLR.

Dahl, R., M. Norouzi, and J. Shlens (2017). Pixel recursive super resolution. In *Proceedings of the IEEE international conference on computer vision*, pp. 5439–5448.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680.

Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville (2017). Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

Mosser, L., O. Dubrule, and M. J. Blunt (2020). Stochastic seismic waveform inversion using generative adversarial networks as a geological prior. *Mathematical Geosciences 52*(1), 53–79.

Pereira, M. and N. Desassis (2019). Efficient simulation of gaussian markov random fields by chebyshev polynomial approximation. *Spatial Statistics 31*, 100359.