

# Learning distributions : Generative Adversarial Networks and Autoencoders

---

Titouan Vayer

October 4, 2018

# Overview

Introduction : learning a probability distribution

Original GAN [GPM<sup>+</sup>14]

Wasserstein GAN's

Optimal transport and Wasserstein distance

Wasserstein GAN

Autoencoder

Variational autoencoder

## Introduction : learning a probability distribution

---

# Learning a probability measure

The aim of generative models is to learn a data distribution  $\mathbb{P}_r$ , on  $\mathcal{X}$  the "data". The underlying paradigm is Unsupervised Learning.

Some applications of generative models :

- Text to image synthesis [RAY<sup>+</sup>16]
- Text to speech, Image and content generation [GDG<sup>+</sup>15]
- Data augmentation [ASE17]
- Future simulation, DeepFake [CGZE18]
- Drawing cats

this white and yellow flower have thin white petals and a round yellow stamen

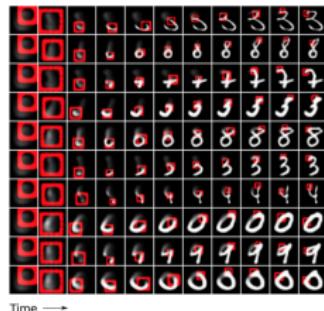
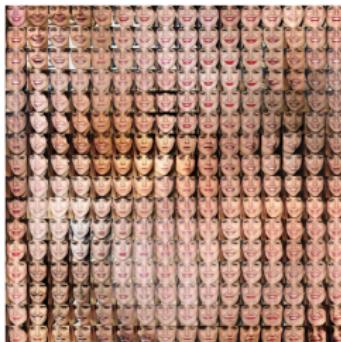


Figure 1: [RAY<sup>+</sup>16, ASE17, GDG<sup>+</sup>15]

## Probability measures and histograms

Let  $\Omega$  be a measurable space.

- A probability measure is a measure  $\mu$  on  $\Omega$  such that  $\int_{\Omega} d\mu = 1$ .
- We note  $\mathcal{P}(\Omega)$  the space of all probability measures on  $\Omega$ .

In the discrete case where  $\Omega = \{x_1, \dots, x_n\}$  a probability measure can be described through histograms such that :

$$\mu = \sum_{i=1}^n \mu_i \delta_{x_i}$$

where  $\delta_{x_i}(x) = 1$  if  $x = x_i$  else 0 and  $\sum_{i=1}^n \mu_i = 1$ .  $\mu_i$  is the probability of the event  $x_i$ .

## Probability measures and histograms

- Bernoulli law : toss a coin  $\Omega = \{\text{pile, face}\}$  and  $\mu_{i=1,2} \in \{0.5, 0.5\}$
- Color histograms in image : pixels as empirical distribution

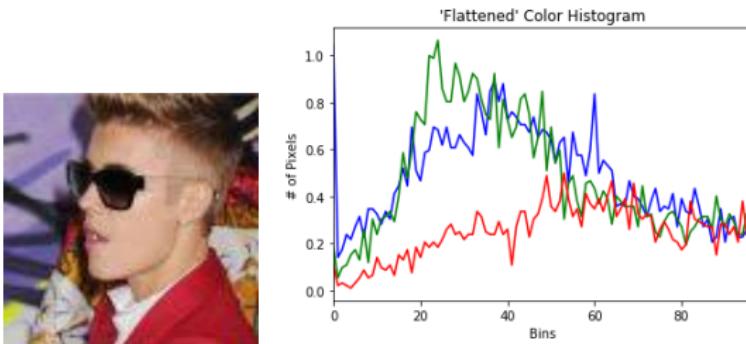


Figure 2: Histogramme de Bieber

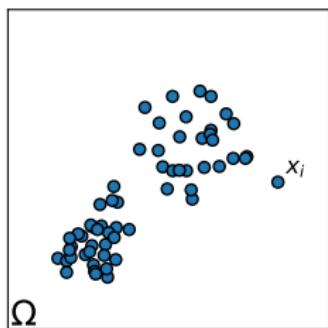
- An image is a realization of a higher probability distribution where each pixel follows a distribution in the RGB color
- Distribution of all the images of Bieber faces

## Everything leads to distributions: Empirical vs Histogram

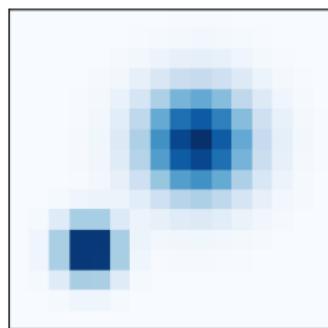
Many problems in which the output of the learning machine is both non-negative and multi-dimensional might be cast as predicting a measure

Discrete measure:  $\mu = \sum_{i=1}^n \mu_i \delta_{x_i}, \quad x_i \in \Omega, \quad \sum_{i=1}^n \mu_i = 1$

Lagrangian (point clouds)



Eulerian (histograms)



- Constant weight:  $\mu_i = \frac{1}{n}$
- Quotient space:  $\Omega^n, \Sigma_n$
- Fixed positions  $x_i$  e.g. grid
- Convex polytope  $\Sigma_n$  (simplex):  
 $\{(\mu_i)_i \geq 0; \sum_i \mu_i = 1\}$

## Prerequisite : divergences

In order to learn from distributions one needs to define a way to compare distributions.

-> One possibility to achieve this is the notion of divergence of distributions.

In order to learn from distributions one needs to define a way to compare distributions.

-> One possibility to achieve this is the notion of divergence of distributions.

$D : \mathcal{P}(\Omega) \times \mathcal{P}(\Omega) \rightarrow \mathbb{R}$  is a divergence if it satisfies :

- for any distributions  $P$  and  $Q$ ,  $D(P||Q) \geq 0$
- $D(P||Q) = 0 \iff P = Q$  a.e

In order to learn from distributions one needs to define a way to compare distributions.

-> One possibility to achieve this is the notion of divergence of distributions.

$D : \mathcal{P}(\Omega) \times \mathcal{P}(\Omega) \rightarrow \mathbb{R}$  is a divergence if it satisfies :

- for any distributions  $P$  and  $Q$ ,  $D(P||Q) \geq 0$
- $D(P||Q) = 0 \iff P = Q$  a.e

Divergences are ways to define a similarity between distributions. They are not distances (may not be symmetric and no triangular inequality)

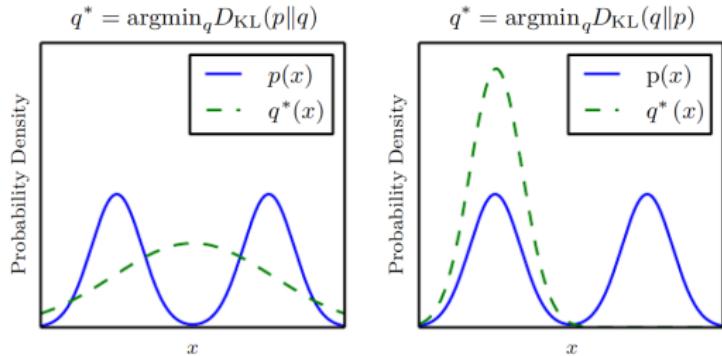
## Kullback Leiber divergence

If we have two distributions  $P(x)$ ,  $Q(x)$  over the same random variable  $x$  we can define :

$$\text{KL}(P||Q) = \mathbb{E}_{x \sim P} \left[ \log \left( \frac{P(x)}{Q(x)} \right) \right] = \int_{\Omega} \log \left( \frac{P(x)}{Q(x)} \right) P(x) dx \quad (1)$$

Information theory : KL it is the extra amount of information needed to send a message containing symbols drawn from  $P$ , when we use a code that was designed to minimize the length of messages drawn from  $Q$ .

## Kullback Leiber divergence is asymmetric



**Figure 3:** Suppose we wish to approximate  $p(x)$  with another distribution  $q(x)$ . (Left) The effect of minimizing  $\text{KL}(p\|q)$  :  $q$  has high probability where  $p$  has high probability. (Right) The effect of minimizing  $\text{KL}(q\|p)$  :  $q$  has low probability where  $p$  has low probability [LBH15]

We can derive the symmetric Jensen-shannon divergence from the KL-divergence

$$\text{JSD}(P\|Q) = \frac{1}{2} \text{KL}(P\|P_A) + \frac{1}{2} \text{KL}(Q\|P_A) \quad (2)$$

where  $P_A = \frac{P+Q}{2}$

For discrete histograms with same number of bins  $P = (p_i)_i, Q = (q_i)_i$  :

- Minkowski distance or  $\ell_p$  distances :  $d_{L_r}(P, Q) = \left( \sum_i |p_i - q_i|^r \right)^{\frac{1}{r}}$
- Histogram intersection :  $d_h(P, Q) = 1 - \frac{\sum_i \min(p_i, q_i)}{\sum_i q_i}$
- $\chi^2$  statistic :  $d_{\chi^2}(P, Q) = \sum_i \frac{(p_i - q_i)^2}{h_i}$ , where  $m_i = \frac{p_i + q_i}{2}$
- Quadratic form distance :  $d_A(P, Q) = \sqrt{(P - Q)^T A (P - Q)}$  with  $A = (a_{ij})$  the "cross-bin" information.
- Kolmogorov-Smirnov distance, match distance ...

For discrete histograms with same number of bins  $P = (p_i)_i, Q = (q_i)_i$  :

- Minkowski distance or  $\ell_p$  distances :  $d_{L_r}(P, Q) = \left( \sum_i |p_i - q_i|^r \right)^{\frac{1}{r}}$
- Histogram intersection :  $d_h(P, Q) = 1 - \frac{\sum_i \min(p_i, q_i)}{\sum_i q_i}$
- $\chi^2$  statistic :  $d_{\chi^2}(P, Q) = \sum_i \frac{(p_i - q_i)^2}{h_i}$ , where  $m_i = \frac{p_i + q_i}{2}$
- Quadratic form distance :  $d_A(P, Q) = \sqrt{(P - Q)^T A (P - Q)}$  with  $A = (a_{ij})$  the "cross-bin" information.
- Kolmogorov-Smirnov distance, match distance ...

How to choose ? Each one has its advantages and its drawbacks (some clues coming...)

For discrete histograms with same number of bins  $P = (p_i)_i, Q = (q_i)_i$  :

- Minkowski distance or  $\ell_p$  distances :  $d_{L_r}(P, Q) = \left( \sum_i |p_i - q_i|^r \right)^{\frac{1}{r}}$
- Histogram intersection :  $d_h(P, Q) = 1 - \frac{\sum_i \min(p_i, q_i)}{\sum_i q_i}$
- $\chi^2$  statistic :  $d_{\chi^2}(P, Q) = \sum_i \frac{(p_i - q_i)^2}{h_i}$ , where  $m_i = \frac{p_i + q_i}{2}$
- Quadratic form distance :  $d_A(P, Q) = \sqrt{(P - Q)^T A (P - Q)}$  with  $A = (a_{ij})$  the "cross-bin" information.
- Kolmogorov-Smirnov distance, match distance ...

How to choose ? Each one has its advantages and its drawbacks (some clues coming...)

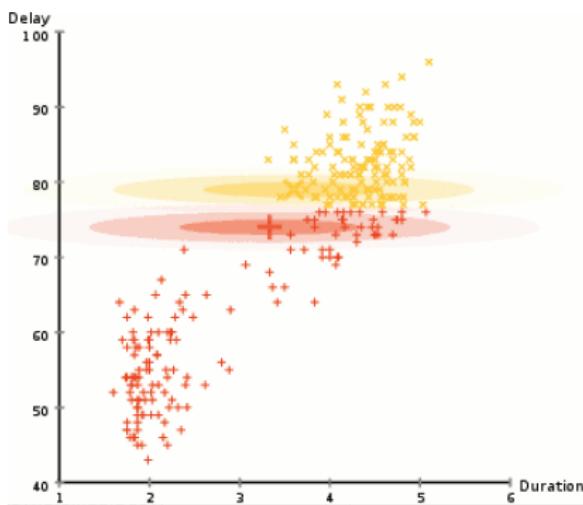
Now that we now how to compare distributions how do we learn them ?

## Classical Log likelihood approach

Usually to learn a distribution  $\mathbb{P}_r$  we use a parametric family of densities  $(P_\theta)_{\theta \in \Theta}$  and try to maximize the following log-likelihood :

$$\max_{\theta \in \Theta} \mathbb{E}_{x \sim \mathbb{P}_r} [\log(P_\theta(x))] \approx \max_{\theta \in \Theta} \frac{1}{m} \sum_{i=1}^m \log P_\theta(x_i) \quad (3)$$

Example : gaussian mixture, logistic regression where we want to learn the distribution  $P(Y = 1|X = x)$  using the parametric family  $(\sigma(\theta^T x))_{\theta \in R^d} \dots$



### Asymptotic behavior

If  $\mathbb{P}_r$  has a density, and  $\mathbb{P}_{\theta^*}$  is the density maximizing (3) then asymptotically  $\mathbb{P}_{\theta^*}$  is such that :

$$\text{KL}(\mathbb{P}_r || \mathbb{P}_{\theta^*}) = \min_{\theta \in \Theta} \text{KL}(\mathbb{P}_r || P_\theta)$$

Solving (3) is equivalent to find the "closest" distribution among the given parametric family which fits the target distribution (*w.r.t* the KL-divergence)

Original GAN [GPM<sup>+</sup>14]

---

# Generative model

## Generative problem

We want to construct a distribution  $\mathbb{P}_g$ , called a generative distribution, which mimicks the real distribution  $\mathbb{P}_r$ .

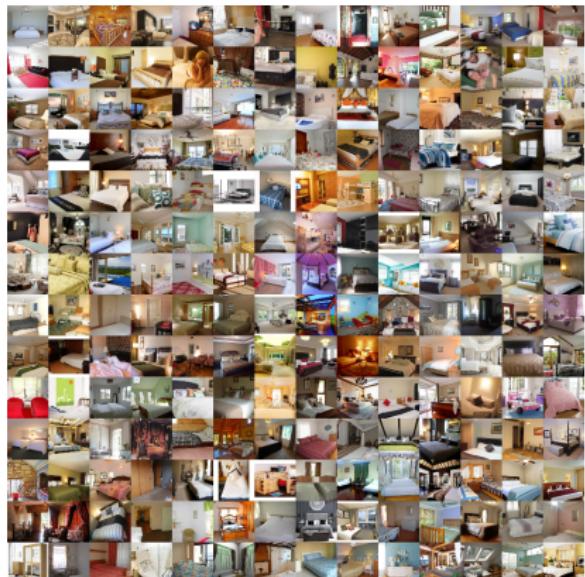
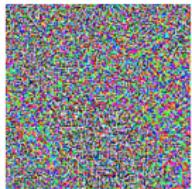


Figure 4: [RMC15]

## The adversarial Setting

Neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed to fool the network.

For e.g by searching an input  $x'$  near a data point  $x$  such that the model output is very different at  $x'$  : it is adversarial examples [GSS14]

$$\begin{array}{ccc} \text{} & + .007 \times & \text{} \\ \text{$x$} & & = \\ \text{"panda"} & & \text{sign}(\nabla_x J(\theta, x, y)) \\ 57.7\% \text{ confidence} & & \text{"nematode"} \\ & & 8.2\% \text{ confidence} \\ & & \text{$x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$} \\ & & \text{"gibbon"} \\ & & 99.3 \% \text{ confidence} \end{array}$$

In this approach, a generative model (the generator) is trained to learn  $\mathbb{P}_g$  by fooling a feedforward classifier (the discriminator):

- The discriminator attempts to recognize all samples from the generative model as being fake, and all samples from the training set as being real.
- Based on a game theoretic scenario in which the generator network must compete against the discriminator  
([https://en.wikipedia.org/wiki/A\\_Beautiful\\_Mind\\_\(film\)](https://en.wikipedia.org/wiki/A_Beautiful_Mind_(film))) :

In this approach, a generative model (the generator) is trained to learn  $\mathbb{P}_g$  by fooling a feedforward classifier (the discriminator):

- The discriminator attempts to recognize all samples from the generative model as being fake, and all samples from the training set as being real.
- Based on a game theoretic scenario in which the generator network must compete against the discriminator

([https://en.wikipedia.org/wiki/A\\_Beautiful\\_Mind\\_\(film\)](https://en.wikipedia.org/wiki/A_Beautiful_Mind_(film))) :

A function  $L(g, d)$  determines the payoff of the discriminator. The generator receives  $-L(g, d)$  as its own payoff. Each player attempts to maximize its own payoff.

$$\max_d \min_g L(g, d)$$

What is  $L$  ?

## Problem setting

In the most classical GAN setting we have to solve the following problem :

$$\max_D \min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))] \quad (4)$$

## Problem setting

In the most classical GAN setting we have to solve the following problem :

$$\max_D \min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))] \quad (4)$$

- $\max_D \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))]$  : force  $D$  to differentiate all sample from  $\mathbb{P}_r$  and from  $\mathbb{P}_g$

## Problem setting

In the most classical GAN setting we have to solve the following problem :

$$\max_D \min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))] \quad (4)$$

- $\max_D \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))]$  : force  $D$  to differentiate all sample from  $\mathbb{P}_r$  and from  $\mathbb{P}_g$
- $\min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))]$  : the generator learns to fool  $D$

## Problem setting

In the most classical GAN setting we have to solve the following problem :

$$\max_D \min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))] \quad (4)$$

- $\max_D \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))]$  : force  $D$  to differentiate all sample from  $\mathbb{P}_r$  and from  $\mathbb{P}_g$
- $\min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))]$  : the generator learns to fool  $D$

This raise two questions : how do we model " $G$  produces  $\mathbb{P}_g$ " and how do we model  $D$  ??

## How to model the generator G : latent variable

- When training a generative model, the more complicated the dependencies between the dimensions, the more difficult the models are to train.
- For e.g if we care about modeling the digits 0 – 9. If the left half of the character contains the left half of a 5, then the right half cannot contain the left half of a 0.
- This a priori helps if the model first decides which character to generate before it assigns a value to any specific pixel

This kind of decision is called a latent variable : before our model draws anything, it first randomly samples a digit value.

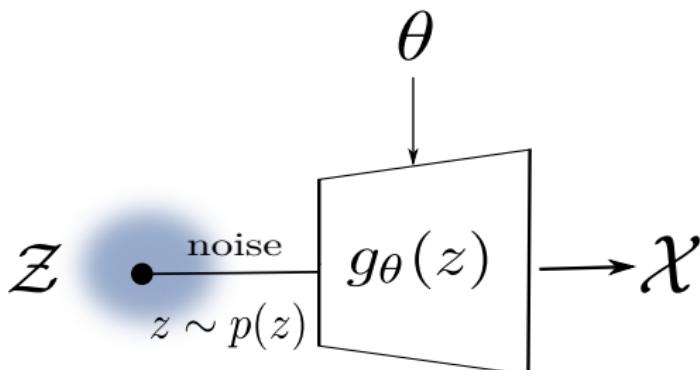
0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9

## How to model the generator G : latent variable

Let  $\mathcal{X}$  be the space of the data.

- We have a latent space  $\mathcal{Z}$  which we can sample from using some probability density  $p(z)$ . (typically gaussian noise)
- We then use parametric functions  $g_\theta : \mathcal{Z} \rightarrow \mathcal{X}$ .
- $g_\theta$  is now a random variable in the space of our data  $\mathcal{X}$

The goal is to optimize  $\theta$  such that we can sample  $z$  from  $p(z)$  and, with high probability,  $g_\theta(z)$  will be like the  $X$ 's in our dataset.

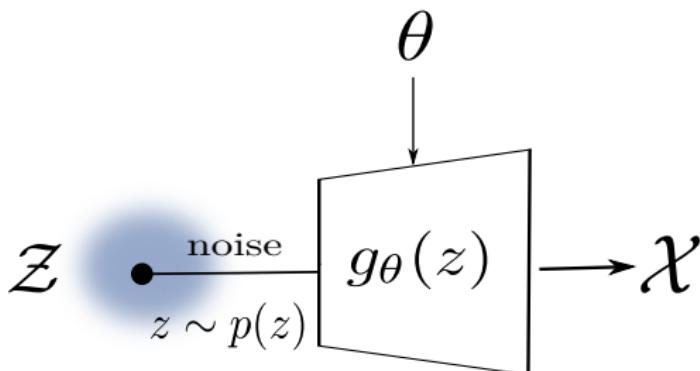


## How to model the generator G : latent variable

Let  $\mathcal{X}$  be the space of the data.

- We have a latent space  $\mathcal{Z}$  which we can sample from using some probability density  $p(z)$ . (typically gaussian noise)
- We then use parametric functions  $g_\theta : \mathcal{Z} \rightarrow \mathcal{X}$ .
- $g_\theta$  is now a random variable in the space of our data  $\mathcal{X}$

The goal is to optimize  $\theta$  such that we can sample  $z$  from  $p(z)$  and, with high probability,  $g_\theta(z)$  will be like the  $X$ 's in our dataset.



What parametric family  $g_\theta$  do we choose ? -> neural network

## Why using neural network ?

### Representer theorem

Let  $g$  be a bounded, continuous and non decreasing (activation) function. Let  $K_d$  be some compact set in  $R_d$  and  $C(K_d)$  the set of continuous functions on  $K_d$ . Let  $f \in C(K_d)$ .

Then for all  $\epsilon > 0$ , there exists  $N \in \mathbb{N}$ , real numbers  $v_i, b_i$  and  $R_d$ vectors  $w_i$  such that the function

$$F(x) = \sum_{i=1}^N v_i g(\langle w_i, x \rangle + b)$$

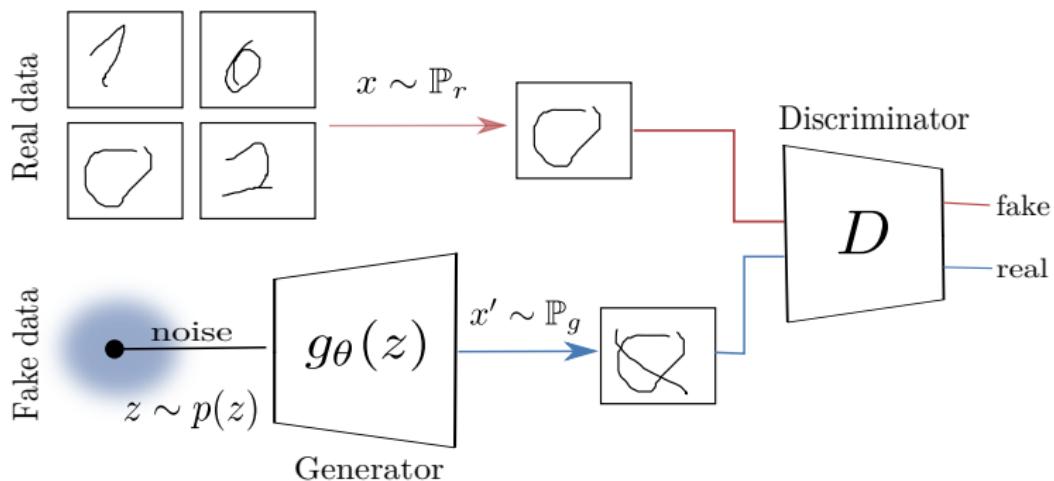
satisfies  $\forall x \in K_d$ ,  $|F(x) - f(x)| \leq \epsilon$

## Putting everything together

We use a parametric function (neural network)  $g_\theta$  and a noise  $p(z)$  such that  $x' = g_\theta(z), z \sim p(z)$ :

$$\max_D \min_{g_\theta} L(D, g_\theta) = \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{z \sim p(z)} [\log(1 - D(g_\theta(z)))] \quad (5)$$

The discriminator is also a neural network.



## Theoretical results

To solve this problem we use alternate updates of  $D$  and  $G$  by iterating the two following steps :

- First train the discriminator  $D$  by maximizing (5) w.r.t  $D$
- Train the generator  $G$  by minimizing (5) w.r.t  $G$

One can show that the first step (with generator fixed) leads to an optimal discriminator :

$$D^*(x) = \frac{\mathbb{P}_r(x)}{\mathbb{P}_r(x) + \mathbb{P}_g(x)}$$

The second step is equivalent at finding the generator which minimizes :

$$2\text{JSD}(\mathbb{P}_r, \mathbb{P}_g) - 2 \log 2$$

so that we have :

### Solution to (5)

The global minimum of (5) is achieved if and only if  $\mathbb{P}_r = \mathbb{P}_g$  and at that point  $L(D^*, G^*) = -2 \log 2$

# Implementation of GAN

---

**Algorithm 1** pseudo code of GAN

---

- 1: Discriminator with parameters  $\theta_d$  and generator with parameters  $\theta_g$
- 2: **for** number of training iterations **do**
- 3:   **for**  $k$  steps **do**
- 4:     sample minibatches  $\{z_1, \dots, z_m\}$  from noise prior  $p(z)$
- 5:     sample minibatches  $\{x_1, \dots, x_m\}$  from data  $\mathbb{P}_r$
- 6:     Update the discriminator by ascending its stochastic gradient

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \log[D(x_i)] + \log[1 - D(\mathbf{g}_{\theta_g}(z_i))]$$

- 7:   **end for**
- 8:   sample minibatches  $\{z_1, \dots, z_m\}$  from noise prior  $p(z)$
- 9:   Update the generator by ascending its stochastic gradient

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log[1 - D(\mathbf{g}_{\theta_g}(z_i))]$$

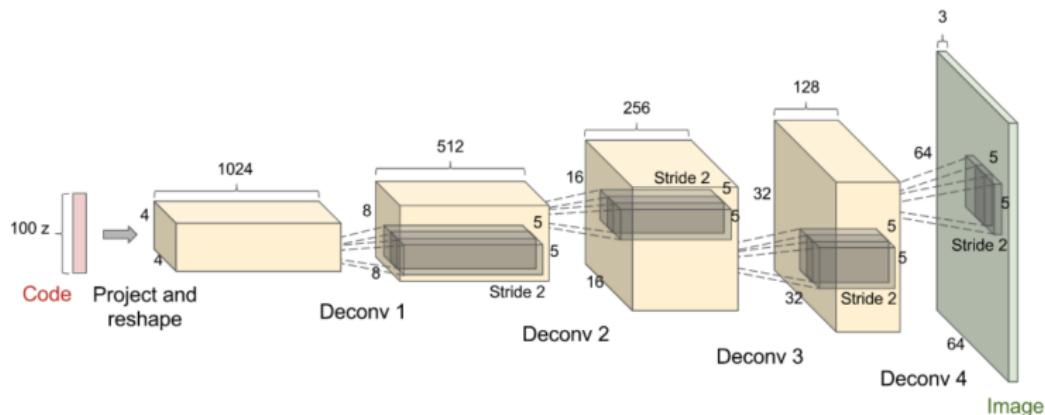
- 10: **end for**
-

# How to build the discriminator and generator ?

One rule : don't be a hero. There exist a lot of GAN architectures : GAN, AC-GAN, BiGAN, BGAN, CC-GAN, CGAN (conditionnal)

We will focus on the classical GAN ([GPM<sup>+</sup>14]) and DCGAN ([RMC15]) :

- Original GAN doesn't use convolution : only dense layer with ReLu/sigmoid/maxout activation and batch normalization
- DCGAN :



- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator). In other words, when the generator needs to increase the spatial dimension of the representation.
- Use Batch Normalization in both the generator (except at the output layer) and the discriminator (except at the input layer)
- Remove fully connected hidden layers for deeper architectures
- Use ReLU activation in generator for all layers except for the output
- Use LeakyReLU activation in the discriminator for all layers

## Downside

---

GAN training is theoretically guaranteed to converge if we can modify the density functions directly, but:

- Instead, we modify G and D : have they enough capacity to represent the distributions ?
- We represent G and D as highly non-convex parametric functions
- “Oscillation”: can train for a very long time, generating very many different categories of samples, without clearly generating better samples

## Downside

---

Multimodal low dimensionnal manifold undifined

## Some enlightening response [AB17]

## Evaluation of GAN's

There is not a good way to quantify how good samples are : about 24 quantitative and 5 qualitative measures.

Most widely adopted : Inception Score.

- It uses a pre-trained neural network (Inception Net) to capture the desirable properties of generated samples by running a classification on them
- We desire highly classifiable and diversity of the samples : expected to have low entropy for easily classifiable samples and high entropy on the classes.

$$\text{IS} = \exp(H(y) - \mathbb{E}_x[H(y|x)])$$

where  $y$  denote the classes and  $x$  the samples.

# Evaluation of GAN's [Bor18]

Measure	Description
Quantitative	<ul style="list-style-type: none"> <li>Log likelihood of explaining realworld held out/test data using a density estimated from the generated data (e.g., using KDE or Parzen window estimation). <math>L = \frac{1}{N} \sum_i \log P_{model}(\mathbf{x}_i)</math></li> <li>The probability mass of the true data "covered" by the model distribution <math>C := P_{data}(dP_{model} &gt; t)</math> with <math>t</math> such that <math>P_{model}(dP_{model} &gt; t) = 0.95</math></li> <li>KLD between conditional and marginal label distributions over generated data. <math>\exp(E_{\mathbf{x}} [\text{KL}(p(\mathbf{y}   \mathbf{x}) \  p(\mathbf{y}))])</math></li> <li>Encourages diversity within images sampled from a particular category. <math>\exp(E_{\mathbf{x}_i} [E_{\mathbf{x}_j} [\{\text{KL}(P(\mathbf{y}   \mathbf{x}_i))   P(\mathbf{y}   \mathbf{x}_j)\}]])</math></li> <li>Similar to IS but also takes into account the prior distribution of the labels over real data. <math>\exp(E_{\mathbf{x}} [\text{KL}(p(\mathbf{y}   \mathbf{x}) \  p(\mathbf{y}^{train}))] - \text{KL}(p(\mathbf{y}) \  p(\mathbf{y}^{train})))</math></li> <li>Takes into account the KLD between distributions of training labels vs. predicted labels, as well as the entropy of predictions. <math>\text{KL}(p(\mathbf{y}   \mathbf{x}) \  p(\mathbf{y}^{train})) + E_{\mathbf{x}} [H(p(\mathbf{y}   \mathbf{x}))]</math></li> <li>Wasserstein-2 distance between multi-variate Gaussians fitted to data embedded into a feature space <math>FID(r, g) =   \mu_r - \mu_g  _2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{\frac{1}{2}})</math></li> <li>Measures the dissimilarity between two probability distributions <math>P_r</math> and <math>P_g</math> using samples drawn independently from each distribution. <math>M_h(P_r, P_g) = E_{\mathbf{x}, \mathbf{x}' \sim P_r} [k(\mathbf{x}, \mathbf{x}')] - 2E_{\mathbf{x} \sim P_r, \mathbf{y} \sim P_g} [k(\mathbf{x}, \mathbf{y})] + E_{\mathbf{y}, \mathbf{y}' \sim P_g} [k(\mathbf{y}, \mathbf{y}')]</math></li> <li>The critic (e.g., an NN) is trained to produce high values at real samples and low values at generated samples <math>\hat{W}(\mathbf{x}_{test}, \mathbf{x}_g) = \frac{1}{N} \sum_{i=1}^N \hat{f}(\mathbf{x}_{test}[i]) - \frac{1}{N} \sum_{i=1}^N \hat{f}(\mathbf{x}_g[i])</math></li> <li>Measures the support size of a discrete (continuous) distribution by counting the duplicates (near duplicates)</li> <li>Answers whether two samples are drawn from the same distribution (e.g., by training a binary classifier)</li> <li>An indirect technique for evaluating the quality of unsupervised representations (e.g., feature extraction; FCN score). See also the GAN Quality Index (GQI) [122].</li> <li>Measures diversity of generated samples and covariate shift using classification methods.</li> <li>Given two sets of samples from the same distribution, the number of samples that fall into a given bin should be the same up to sampling noise</li> <li>Measures the distributions of distances to the nearest neighbors of some query images (i.e., diversity)</li> <li>Compares two GANs by having them engaged in a battle against each other by swapping discriminators or generators. <math>p(\mathbf{x}   \mathbf{y} = 1; M_1) p(\mathbf{x}   \mathbf{y} = 1; M_2) = (p(\mathbf{y} = 1   \mathbf{x}; D_1) p(\mathbf{x}; G_2)) / (p(\mathbf{y} = 1   \mathbf{x}; D_2) p(\mathbf{x}; G_1))</math></li> <li>Implements a tournament in which a player is either a discriminator that attempts to distinguish between real and fake data or a generator that attempts to fool the discriminators into accepting fake data as real.</li> <li>Compares <math>n</math> GANs based on the idea that if the generated samples are closer to real ones, more epochs would be needed to distinguish them from real samples.</li> <li>Adversarial Accuracy. Computes the classification accuracies achieved by the two classifiers, one trained on real data and another on generated data, on a labeled validation set to approximate <math>P_g(y \mathbf{x})</math> and <math>P_r(y \mathbf{x})</math>. Adversarial Divergence: Computes <math>\text{KL}(P_g(y \mathbf{x}), P_r(y \mathbf{x}))</math></li> <li>Compares geometrical properties of the underlying data manifold between real and generated data.</li> <li>Measures the reconstruction error (e.g., <math>L_2</math> norm) between a test image and its closest generated image by optimizing for <math>z</math> (i.e., <math>\min_z   G(\mathbf{z}) - \mathbf{x}^{(test)}  ^2</math>)</li> <li>Evaluates the quality of generated images using measures such as SSIM, PSNR, and sharpness difference</li> <li>Evaluates how similar low-level statistics of generated images are to those of natural scenes in terms of mean power spectrum, distribution of random filter responses, contrast distribution, etc.</li> <li>These scores are used to quantify the degree of overfitting in GANs, often over toy datasets.</li> </ul>
	<ul style="list-style-type: none"> <li>Nearest Neighbors</li> <li>To detect overfitting, generated samples are shown next to their nearest neighbors in the training set</li> <li>In these experiments, participants are asked to distinguish generated samples from real images in a short presentation time (e.g., 100 ms); i.e., real v.s fake</li> <li>Participants are asked to rank models in terms of the fidelity of their generated images (e.g., pairs, triples)</li> <li>Over datasets with known modes (e.g., a GMM or a labeled dataset), modes are computed as by measuring the distances of generated data to mode centers</li> <li>Regards exploring and illustrating the internal representation and dynamics of models (e.g., space continuity) as well as visualizing learned features</li> </ul>
Qualitative	1. Rapid Scene Categorization [33]
	3. Preference Judgment [40, 128, 120, 124]
	4. Mode Drop and Collapse [99, 59]
	5. Network Internals [81, 13, 38, 69, 125, 5]

## Wasserstein GAN's

---

666. MÉMOIRES DE L'ACADEMIE ROYALE

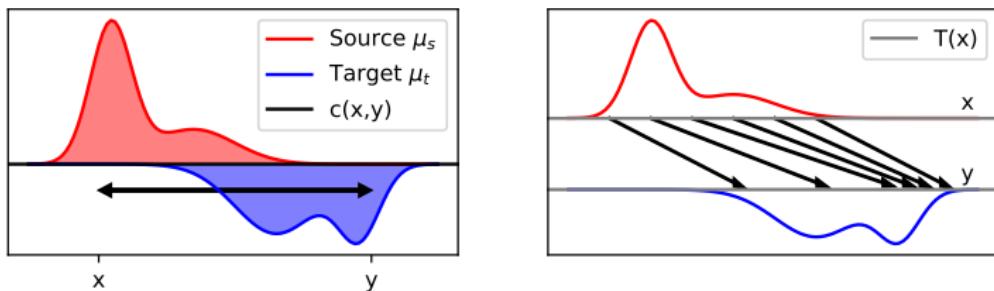
MÉMOIRE  
SUR LA  
THÉORIE DES DÉBLAIS  
ET DES REMBLAIS.  
Par M. MONGE.



## Problem

- How to move dirt from one place (déblais) to another (remblais) while minimizing the effort ?
- Find a mapping  $T$  between the two distributions of mass (transport).
- Optimize with respect to a displacement cost  $c(x, y)$  (optimal).

# The origins of optimal transport

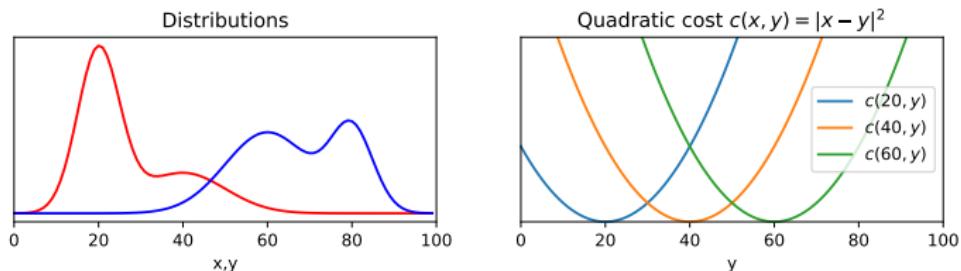


## Problem

- How to move dirt from one place (déblais) to another (remblais) while minimizing the effort ?
- Find a mapping  $T$  between the two distributions of mass (transport).
- Optimize with respect to a displacement cost  $c(x,y)$  (optimal).

## Optimal transport (Monge formulation)

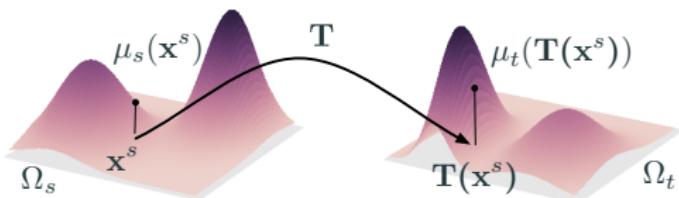
- Mathematical tools aiming at comparing distributions



- Probability measures  $\mu_s$  and  $\mu_t$  on  $\Omega_s$ ,  $\Omega_t$  with a cost function  $d : \Omega_s \times \Omega_t \rightarrow \mathbb{R}^+$ .
- The Monge formulation aim at finding a mapping  $T : \Omega_s \rightarrow \Omega_t$

$$\inf_{T \# \mu_s = \mu_t} \int_{\Omega_s} d(x, T(x)) \mu_s(x) dx \quad (6)$$

## What is $T\#\mu_s = \mu_t$ ?



- $T\#$  is the so called push forward operator
- it transfers measures from one space  $\Omega_s$  to another space  $\Omega_t$
- it is equivalent to:

$$\mu_t(A) = \mu_s(T^{-1}(A))$$

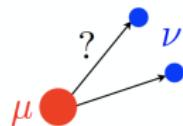
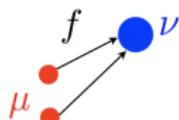
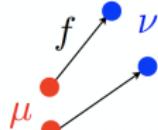
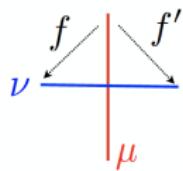
- consists simply in moving the positions of all the points in the support of the measure : for  $\mu_s = \sum_{i=1}^n a_i \delta_{x_i}$

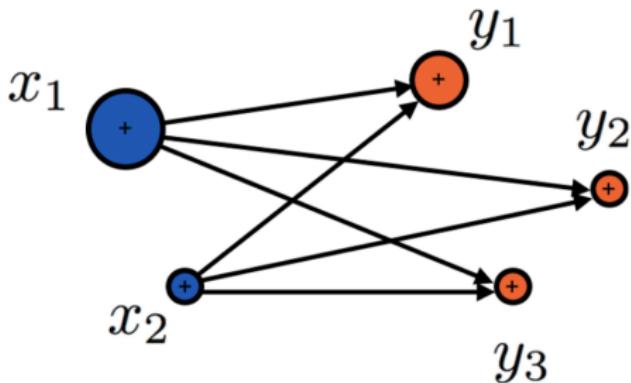
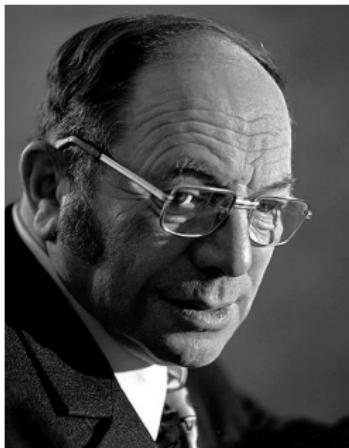
$$T\#\mu_s = \sum_{i=1}^n a_i \delta_{T(x_i)}$$

## Non-existence Non-uniqueness

Solving for this push-forward operator is a non-convex optimization problem,

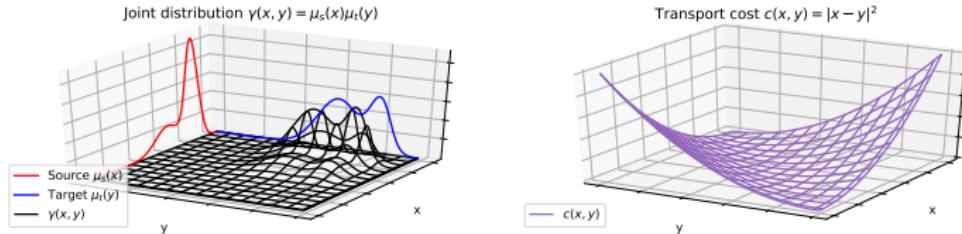
- for which existence is not guaranteed,
- nor unicity





- Leonid Kantorovich (1912–1986), Economy nobelist in 1975, proposed a different formulation of the problem
- with applications mainly for ressource allocation problems

# Kantorovich relaxation



$\mu_s = \sum_{i=1}^n a_i \delta_{x_i}$  and  $\mu_t = \sum_{j=1}^m b_j \delta_{y_j}$  on a common ground space equipped with a distance

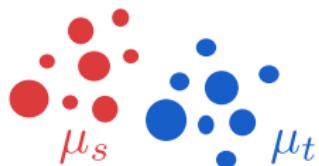
- The Kantorovich formulation seeks for a probabilistic coupling  $\pi \in \Pi(\mu_s \times \mu_t)$  between  $\mu_s$  and  $\mu_t$ .
- $\pi$  is a joint probability measure with prescribed marginals  $\mu_s$  and  $\mu_t$ .
- Computes the Wasserstein distance :

$$\mathcal{W}_p(\mu_s, \mu_t) = \left( \min_{\pi \in \Pi(\mu_s, \mu_t)} \sum_{i,j} d(x_i, y_j)^q \pi_{i,j} \right)^{\frac{1}{p}} \quad (7)$$

Can be solved by linear programming

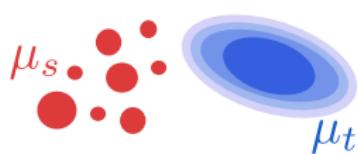
The resulting coupling  $\pi$  associates in a "fuzzy" way the points of the distributions.

Discrete



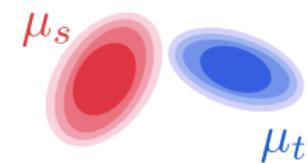
$$\pi$$

Semi discrete



$$\pi$$

Continuous



$$\pi$$

## The Wasserstein distance

- The  $p$ -Wasserstein distance is always defined even for arbitrary measures even singular ones.
- The  $p$ -Wasserstein distance defines a metric over  $\mathcal{P}(\Omega)$ . It means that for  $\alpha, \beta \in \mathcal{P}(\Omega)$ ,  $\mathcal{W}_p(\alpha, \beta) = 0$  iff  $\alpha = \beta$  and  $\mathcal{W}_p$  satisfies the triangle inequality
- The  $p$ -Wasserstein distance is a weak metric, that means  $(\alpha_k)_k$  converges weakly (in law) to some  $\alpha$  in  $\mathcal{P}(\Omega)$  iff  $\mathcal{W}_p(\alpha_k, \alpha) \rightarrow 0$  as  $k \rightarrow \infty$

The first point is one of the most interesting since "classical" distances (or divergences) are not even defined between discrete distributions.

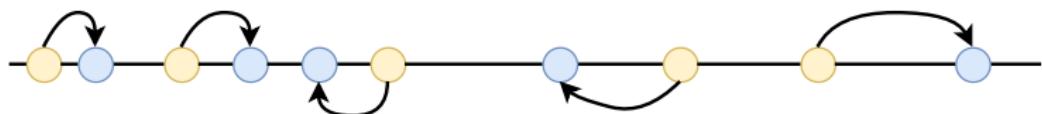
- The L2 norm can only be applied to continuous measures with a density with respect to a base measure
- The discrete  $\ell_2$  norm requires that positions  $(x_i, y_j)$  take values in a predetermined discrete set to work properly
- Idem KL divergence is not defined if the supports of the measures does not overlap

## Special case: 1D distribution

We consider the case where  $d(x, y) = |x - y|$

- if  $x_1 < x_2$  and  $y_1 < y_2$ , it is easy to check that  
 $d(x_1, y_1) + d(x_2, y_2) < d(x_1, y_2) + d(x_2, y_1)$
- As such, any optimal transport plan respects the ordering of the elements, and the solution is given by the monotone rearrangement of  $\mu_s$  onto  $\mu_t$

This gives very simple algorithm to compute the transport in  $O(N \log N)$ , by sorting both  $x_i$  and  $y_i$  and summing the absolute values of differences.



In the case  $\Omega = \mathbb{R}^d$ ,  $d(x, y) = \|x - y\|$  and  $p = 1$

---

The optimal transport problem then aim to find  $\phi \in \text{Lip}^1$  (set of 1-Lipschitz functions) as

$$\sup_{\phi \in \text{Lip}^1} \int \phi d(\mu_s - \mu_t) = \sup_{\phi \in \text{Lip}^1} \mathbb{E}_{x \sim \mu_s} [\phi(x)] - \mathbb{E}_{y \sim \mu_t} [\phi(y)] \quad (8)$$

- known as **Kantorovich-Rubinstein duality**
- $\phi$  can be learnt as a neural network constrained to the set  $\text{Lip}^1$ , see next section on GAN

Numerous applications for the Wasserstein distance :

- Learning with Wasserstein Loss [FZM<sup>+</sup>15]
- Wasserstein GAN's [ACB17]
- Domain Adaptation [CFTR17]
- Image colorization [FPPA14], Dictionary Learning [RCP16] ...

$$\min_{\theta \in \Theta} \mathcal{W}_1(\mathbb{P}_r, \mathbb{P}_{\theta})$$

idea :  $\mathbb{P}_{\theta} = g_{\theta}(z), z \sim Z$ ,  $g_{\theta}$  is a neural network.

$$\iff \max_{w \in \mathcal{W}} \min_{\theta \in \Theta} \mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{z \sim P_Z} [f_w(g_{\theta}(z))]$$

$$f_w = D, g_{\theta} = G$$

---

**Algorithm 2** pseudo code of GAN

---

- 1: Require :  $\alpha$  the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_c$ , the number of iterations of the critic per generator iteration
- 2: Require :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.
- 3: **while**  $\theta_g$  has not converged **do**
- 4:   **for**  $t = 0, \dots, n_c$  **do**
- 5:     sample minibatches  $\{z_1, \dots, z_m\}$  from noise prior  $p(z)$
- 6:     sample minibatches  $\{x_1, \dots, x_m\}$  from data  $\mathbb{P}_r$ ,
- 7:     Update  $f_w$  by ascending its stochastic gradient

$$\nabla_w \frac{1}{m} \sum_{i=1}^m f_w(x_i) - \frac{1}{m} \sum_{i=1}^m f_w(\mathbf{g}_{\theta_g}(z_i))$$

- 8:      $w \leftarrow \text{clip}(-c, c)$
- 9:   **end for**
- 10:   sample minibatches  $\{z_1, \dots, z_m\}$  from noise prior  $p(z)$
- 11:   Update the generator by ascending its stochastic gradient  
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m f_w(\mathbf{g}_{\theta_g}(z_i))$$
- 12: **end while**

## WGAN: the devil in the approximation

### Neural network belonging to $\text{Lip}^1$ ?

- Not really! [?] proposes to do weight clipping that force an upper bound on the Lipschitz constant.
- It is actually the supremum over K-Lipschitz functions that is approximated by a neural network

$$\max_{f \in \text{NN class}} L_{\text{WGAN}}(f, G) \leq \sup_{\|\phi\|_L \leq K} L_{\text{WGAN}}(\phi, G) = K \cdot W_1^1(G(z), \mu_d)$$

- Actually **not** equivalent to solve the optimal transport, but gradients are aligned.

### Improved WGAN [GAA<sup>+</sup>17]

$$\min_G \sup_{f \in \text{NN class}} \mathbb{E}_{x \sim \mu_d}[f(x)] - \mathbb{E}_{z \sim \mathcal{N}(0, I)}[f(G(z))] + \lambda \mathbb{E}_{x \sim \mu_d}[(\|\nabla f(x)\|_2 - 1)^2]$$

Relaxation of the constraint (for  $W_1$  the gradient of the potential is 1 almost everywhere).

## Autoencoder

---

An autoencoder is a neural network that is trained to attempt to copy its input to its output. It has two parts :

- An encoder function  $h_{\theta_e} : \mathcal{X} \rightarrow \mathcal{Z}$  that pushes the inputs  $x$  in a smaller dimensional space.
- A decoder function  $g_{\theta_d} : \mathcal{Z} \rightarrow \mathcal{X}$  that reconstructs from the low dimensional space to the initial space

Very generally autoencoders aim at solving :

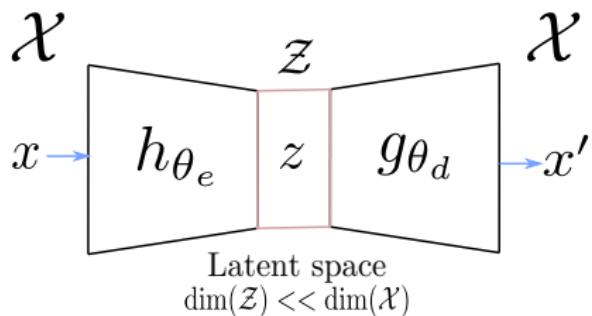
$$\min_{\theta_e, \theta_d} \mathbb{E}_{x \sim \mathbb{P}_r} [L(x, g_{\theta_d}, h_{\theta_e})] \quad (9)$$

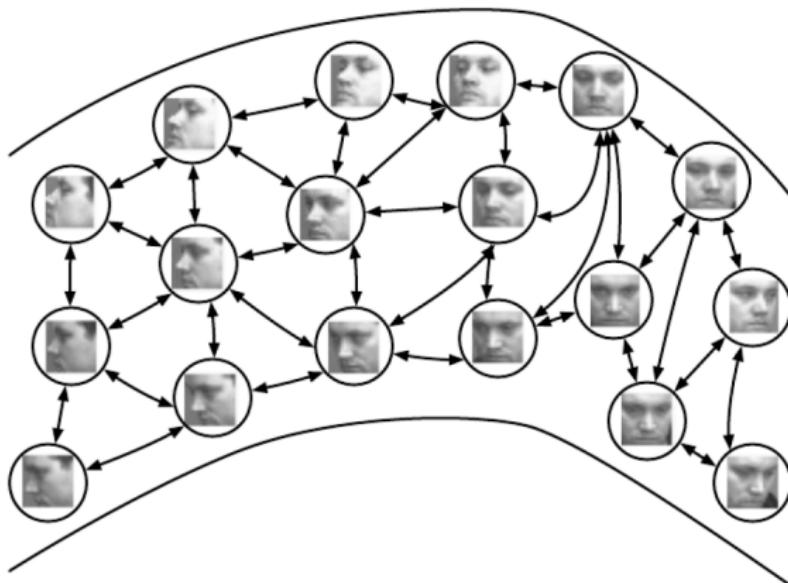
## Traditional autoencoders

The most classical autoencoders are undercomplete autoencoders : the latent space has a smaller dimension  $\dim(\mathcal{Z}) \ll \dim(\mathcal{X})$  (typically subset of  $\mathbb{R}^d$  with  $d$  small) :

$$\min_{\theta_e, \theta_d} \mathbb{E}_{x \sim \mathbb{P}_r} [L(x, g_{\theta_d}(h_{\theta_e}(x)))] \quad (10)$$

- $L$  is chosen to be a reconstruction error as  $\ell_2$  error
- Forces the network to act as the identity. The latent space acts as a dimensional reduction of  $\mathcal{X}$ .
- When the decoder is linear and  $L$  is the mean squared error, an undercomplete autoencoder learns to span the same subspace  $\mathcal{Z}$  as PCA.





**Figure 5:** Non-parametric manifold learning procedures build a nearest neighbor graph in which nodes represent training examples and directed edges indicate nearest neighbor relationships. The coordinate system associates each training example with a real-valued vector position in the embedding  $\mathcal{Z}$ . [GMP00]

- Lower-dimensional representations can improve performance on many tasks, such as classification.
- Models of smaller spaces consume less memory and runtime.
- Interesting for e.g in information retrieval : finding entries in a database that resemble a query entry. We can store a full complicated database in a hash table mapping binary code vectors to entries and do fast queries.

- Regularized autoencoders : if  $\dim(\mathcal{Z}) \approx \dim(\mathcal{X})$ 
  - In this case classical autoencoders usually fail to learn anything useful.
  - Regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output : sparsity of the representation, smallness of the derivative of the representation, and robustness to noise or to missing inputs
  - See VAE
- Sparse autoencoders : the learning process aim at minimizing

$$L(x, g_{\theta_d}(h_{\theta_e}(x))) + \Omega(z)$$

- $\Omega$  is a regularization function in the latent space  $z = h_{\theta_e}(x)$ .
- Sparse autoencoders are typically used to learn features for another task such as classification

- Denoising autoencoders : the learning process aim at minimizing

$$L(x, g_{\theta_d}(h_{\theta_e}(\tilde{x})))$$

- $\tilde{x}$  is a copy of  $x$  that has been corrupted by some form of noise.
- Denoising training forces  $h_{\theta_e}$  and  $g_{\theta_d}$  to implicitly learn the structure of  $\mathbb{P}_r$
- Contracting autoencoders : the learning process aim at minimizing

$$L(x, g_{\theta_d}(h_{\theta_e}(x))) + \Omega(z, x)$$

- $\Omega(z, x) = \lambda \sum_i \|\nabla_x h_{\theta_e}(x)_i\|$
- This forces the model to learn a function that does not change much when  $x$  changes slightly
- It forces the autoencoder to learn features that capture information about the training distribution.

## Variational autoencoder

---

Encoder and decoder previously defined can be seen as distributions :

- From an input  $x \in \mathcal{X}$  of the data and from parameters  $\theta_e$ , the encoder outputs a hidden representation  $z \in \mathcal{Z}$ . This process defines a conditional distribution  $q_{\theta_e}(z|x)$
- In the same way from parameters  $\theta_d$  and a hidden representation  $z \in \mathcal{Z}$  the decoder output a point  $x$  in the space  $\mathcal{X}$ . This process defines a conditional distribution  $p_{\theta_d}(x|z)$

In the digit example with each pixel as 0 and 1 : the probability distribution of a single pixel can be then represented as a Bernoulli distribution.

- The encoder gets as input the latent representation of a digit  $z$  and outputs 784 Bernoulli parameters, one for each of the 784 pixels in the image.
- The decoder decodes the real-valued numbers in  $z$  into 784 real-valued numbers between 0 and 1.

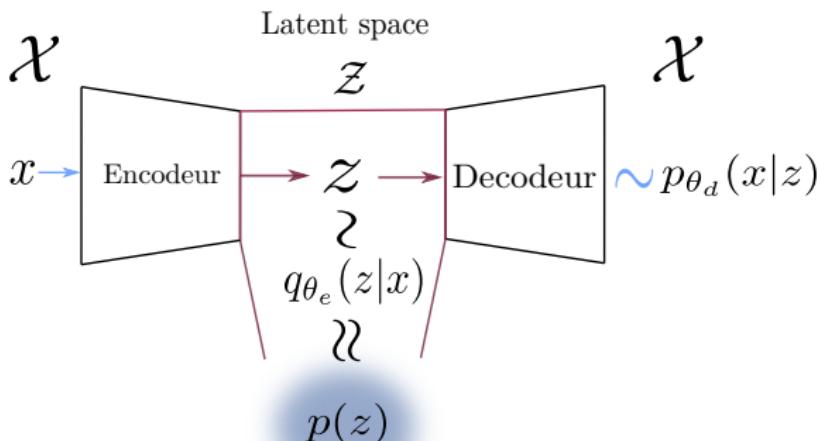
## Problem setting

The basic idea of VAE is to force the distribution  $q_{\theta_e}(z|x)$  coming from the encoder to follow a certain distribution  $p(z)$  (typically a normal distribution) in order to generate samples.

In this case we want to minimize w.r.t  $\theta_e$  and  $\theta_d$  the following quantity :

$$-\mathbb{E}_{z \sim q_{\theta_e}(z|x)} [\log(p_{\theta_d}(x|z))] + \text{KL}(q_{\theta_e}(z|x)||p(z)) \quad (11)$$

for  $x \sim \mathbb{P}_r$



- $-\mathbb{E}_{z \sim q_{\theta_e}(z|x)} [\log(p_{\theta_d}(x|z))]$  is the reconstruction loss : encourages the decoder to learn to reconstruct the data  $\mathbb{P}_r$ .
- $\text{KL}(q_{\theta_e}(z|x) || p(z))$  is the regularization term : it measures how much information is lost when using  $q_{\theta_e}(z|x)$  to represent  $p(z)$ . It encourages  $q_{\theta_e}(z|x)$  to be equal to  $p(z)$ .

We usually choose  $p(z) = \mathcal{N}(0, I)$ . Once the network is trained it is easy to generate samples : pick a  $z \sim \mathcal{N}(0, I)$  (easy) a feed the learned generator  $p_{\theta_d}(x|z)$  with it.

-  Martín Arjovsky and Léon Bottou, [Towards principled methods for training generative adversarial networks](#), CoRR [\*\*abs/1701.04862\*\*](#) (2017).
-  Martín Arjovsky, Soumith Chintala, and Léon Bottou, [Wasserstein GAN](#), CoRR [\*\*abs/1701.07875\*\*](#) (2017).
-  A. Antoniou, A. Storkey, and H. Edwards,  
[Data Augmentation Generative Adversarial Networks](#), ArXiv e-prints  
(2017).
-  Ali Borji, [Pros and cons of GAN evaluation measures](#), CoRR  
[\*\*abs/1802.03446\*\*](#) (2018).
-  Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy,  
[Optimal transport for domain adaptation](#), IEEETPAMI **39** (2017), no. 9,  
1853–1865.
-  C. Chan, S. Ginosar, T. Zhou, and A. A. Efros, [Everybody Dance Now](#),  
ArXiv e-prints (2018).

-  Sira Ferradans, Nicolas Papadakis, Gabriel Peyré, and Jean-François Aujol, Regularized discrete optimal transport, SIAM Journal on Imaging Sciences 7 (2014), no. 3, 1853–1882.
-  C. Frogner, C. Zhang, H. Mobahi, M. Araya-Polo, and T. Poggio, Learning with a Wasserstein Loss, ArXiv e-prints (2015).
-  Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville, Improved training of wasserstein gans, CoRR [abs/1704.00028](https://arxiv.org/abs/1704.00028) (2017).
-  K. Gregor, I. Danihelka, A. Graves, D. Jimenez Rezende, and D. Wierstra, DRAW: A Recurrent Neural Network For Image Generation, ArXiv e-prints (2015).
-  Shaogang Gong, Stephen J. McKenna, and Alexandra Psarrou, Dynamic vision: From images to face recognition, 1st ed., Imperial College Press, London, UK, UK, 2000.

-  Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio, Generative adversarial nets, Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada, 2014, pp. 2672–2680.
-  I. J. Goodfellow, J. Shlens, and C. Szegedy, Explaining and Harnessing Adversarial Examples, ArXiv e-prints (2014).
-  Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton, Deep learning, Nature **521** (2015), no. 7553, 436–444.
-  Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee, Generative adversarial text to image synthesis, CoRR **abs/1605.05396** (2016).

-  Antoine Rolet, Marco Cuturi, and Gabriel Peyré, Fast dictionary learning with a smoothed wasserstein loss, Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (Cádiz, Spain) (Arthur Gretton and Christian C. Robert, eds.), Proceedings of Machine Learning Research, vol. 51, PMLR, 09–11 May 2016, pp. 630–638.
-  A. Radford, L. Metz, and S. Chintala,  
Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ArXiv e-prints (2015).
-  Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas, The earth mover's distance as a metric for image retrieval, International Journal of Computer Vision **40** (2000), no. 2, 99–121.