Towards more Deep Learning

Titouan Vayer

October 1, 2019

Overview

Generative models

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 [Reed et al., 2016]
- Text to speech, Image and content generation [Gregor et al., 2015]
- Data augmentation [Antoniou et al., 2017]
- Future simulation, DeepFake [Chan et al., 2018]
- Drawing cats

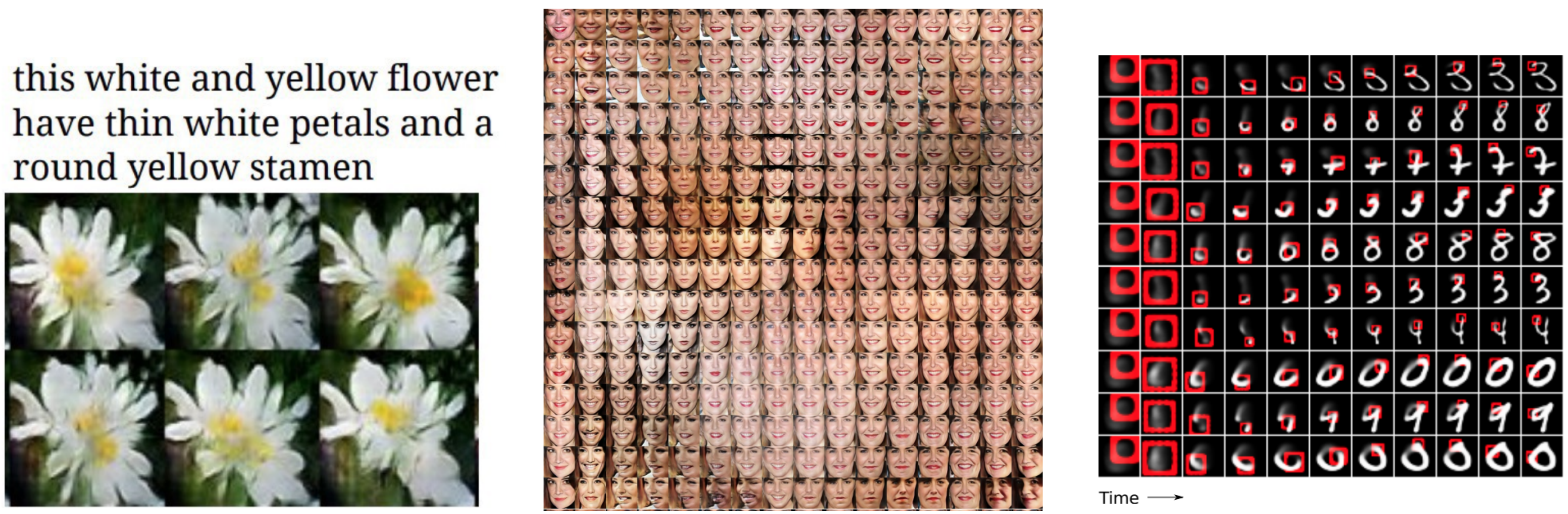


Figure 1: [Reed et al., 2016, Antoniou et al., 2017, Gregor et al., 2015]

Probability measures and histograms

Let Ω be a measurable space.

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

In the discrete case where $\Omega = \{x_1, .., x_n\}$ 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 $\sum_{i=1}^{n} \mu_i = 1$. μ_i is the probability of the event x_i.

Probability measures and histograms

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

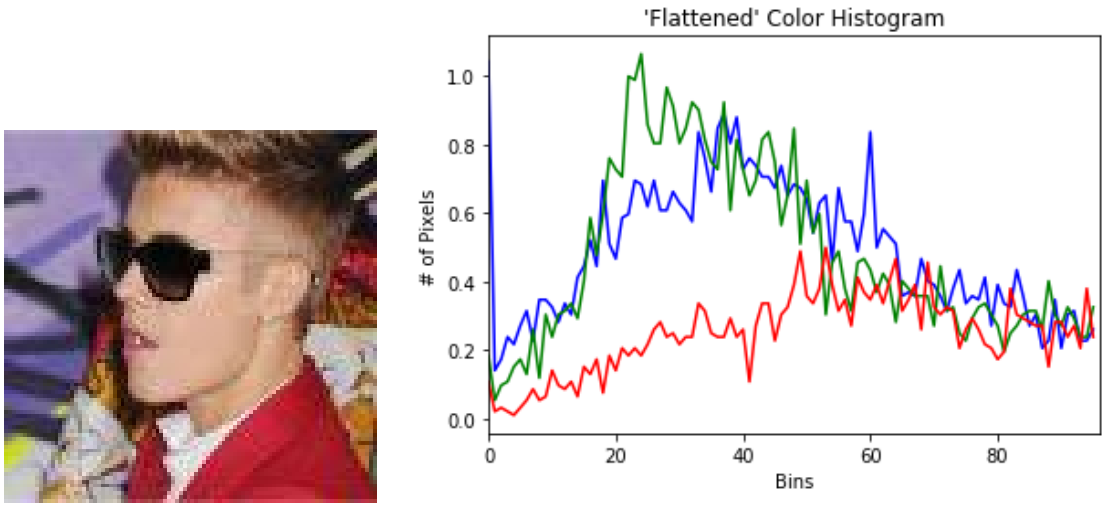


Figure 2: Histogram of Bieber

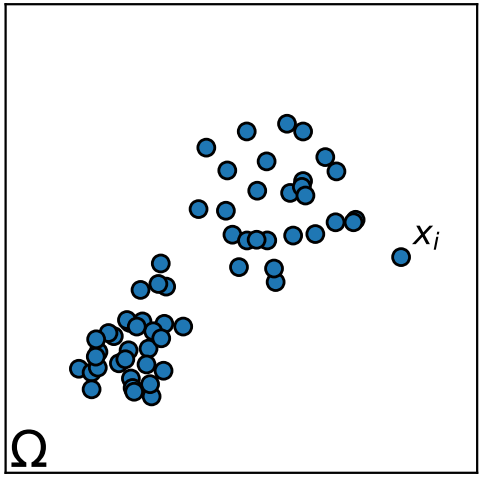
- An image is a realization of a higher probability distribution where each pixel follows a distribution in the RGB color
- Distribution of all the images of Bieber faces

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

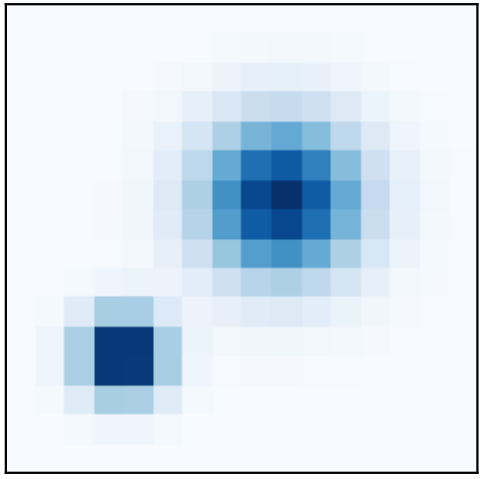
Discrete measure: $\mu = \sum_{i=1}^{n} \mu_i \delta_{\mathbf{x}_i}, \quad \mathbf{x}_i \in \Omega, \quad \sum_{i=1}^{n} \mu_i = 1$

Lagrangian (point clouds)



- Constant weight: $\mu_i = \frac{1}{n}$
- Quotient space: Ω^n, Σ_n

Eulerian (histograms)



- Fixed positions $\mathbf{x}_i$ e.g. grid
- Convex polytope Σ_n (simplex): $\left\{(\mu_i)_i \geq 0; \sum_i \mu_i = 1\right\}$

Prerequisite: Expectation and machine learning

For a real random variable x following the probability measure $\mu \in \mathcal{P}(\mathbb{R})$ we define its expectation as:

$$\mathop{\mathbb{E}}_{x\sim\mu}[x] = \int_{x\in\Omega} x d\mu(x) \tag{1}$$

Prerequisite: Expectation and machine learning

For a real random variable x following the probability measure $\mu \in \mathcal{P}(\mathbb{R})$ we define its expectation as:

$$\mathop{\mathbb{E}}_{x\sim\mu}[x] = \int_{x\in\Omega} x d\mu(x) \tag{1}$$

For any measurable function f we have:

$$\mathop{\mathbb{E}}_{x\sim\mu}[f(x)] = \int_{x\in\Omega} f(x) d\mu(x)$$

It represents the "average" of the random variable x under the law μ

Prerequisite: Expectation and machine learning

Carefull all random variable are not necessarily real !

For example:

- If μ is "toss" a coin $\mathop{\mathbb{E}}_{x\sim\mu}[x] = -0.5 + 0.5 = 0 \in \mathbb{R}$
- If $\mu = \sum_{i=1}^{n} \mu_i \delta_{\mathbf{x}_i} \in \mathcal{P}(\Omega)$, $\mathop{\mathbb{E}}_{x\sim\mu}[x] = \sum_i \mu_i x_i \in \Omega$
- If $\mu = \mathcal{N}(m, \Sigma) \in \mathcal{P}(\mathbb{R}^d)$, $\mathop{\mathbb{E}}_{x\sim\mu}[x] = m \in \mathbb{R}^d$
- ....

Prerequisite: Expectation and machine learning

The expectation is at the core of any machine learning process.

Suppose the goal is to find $f : X \to Y$ such that $f(X)$ is a good proxy for Y. If we have a loss L:

$$f = \underset{g \in \mathcal{H}}{\operatorname{argmin}} \underset{(x,y)\sim\mathcal{D}}{\mathbb{E}}[L(g(x), y)] \tag{2}$$

where $\mathcal{D}$ is the law that generates the data, $\mathcal{H}$ a space of functions.

Prerequisite: Expectation and machine learning

The expectation is at the core of any machine learning process.

Suppose the goal is to find $f : X \to Y$ such that $f(X)$ is a good proxy for Y. If we have a loss L:

$$f = \underset{g \in \mathcal{H}}{\operatorname{argmin}} \underset{(x,y)\sim\mathcal{D}}{\mathbb{E}}[L(g(x), y)] \tag{2}$$

where $\mathcal{D}$ is the law that generates the data, $\mathcal{H}$ a space of functions.

On average on all the data f will be the "best" possible function *w.r.t* the loss L

Prerequisite: Expectation and machine learning

In practice we don't have access to the "true" law $\mathcal{D}$ that generates the data so that we can evaluate (2) but only on discrete samples $(x_i, y_i)_{i \in \{1,\dots,n\}} \sim \mathcal{D}$.

Prerequisite: Expectation and machine learning

In practice we don't have access to the "true" law $\mathcal{D}$ that generates the data so that we can evaluate (2) but only on discrete samples $(x_i, y_i)_{i \in \{1,\dots,n\}} \sim \mathcal{D}$.

In usual ML we minimize the empirical risk:

$$\frac{1}{n} \sum_{i=1}^{n} L(g(x_i), y_i) \approx \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} [L(g(x), y)]$$

Prerequisite: divergences

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

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

Prerequisite: divergences

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

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

$D : \mathcal{P}(\Omega) \times \mathcal{P}(\Omega) \to \mathbb{R}$ 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

Prerequisite: divergences

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

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

$D : \mathcal{P}(\Omega) \times \mathcal{P}(\Omega) \to \mathbb{R}$ 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 triangle inequality)

Kullback Leiber divergence

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

$$\mathrm{KL}(P||Q) = \mathop{\mathbb{E}}_{x\sim P}\big[\log(\frac{P(x)}{Q(x)})\big] = \int_{\Omega} \log(\frac{P(x)}{Q(x)})P(x)dx \tag{3}$$

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

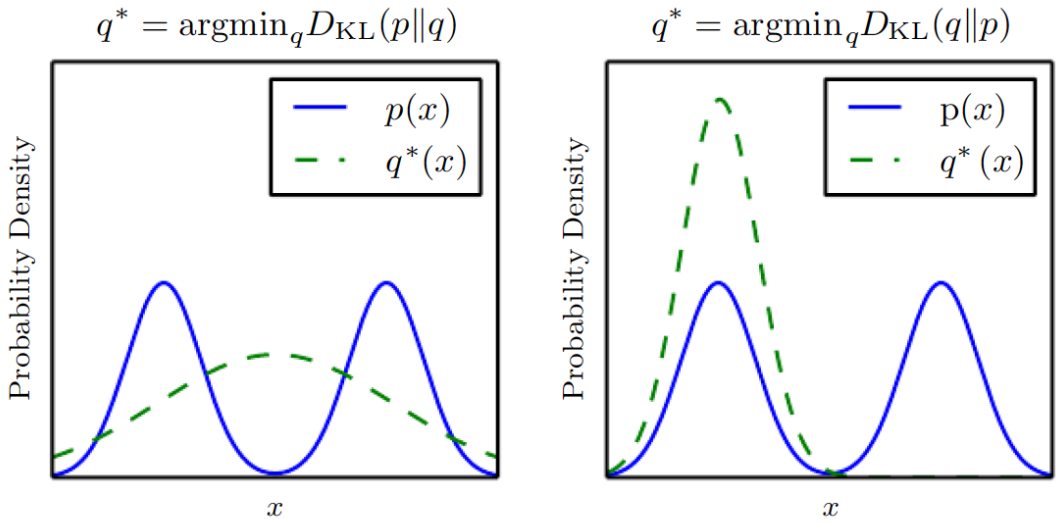


Figure 3: Suppose we wish to approximate $p(x)$ with another distribution $q(x)$. (Left)The effect of minimizing KL$(p||q)$: q has high probability where p has high probability. (Right)The effect of minimizing KL$(q||p)$: q has low probability where p has low probability [LeCun et al., 2015]

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

$$\text{JSD}(P||Q) = \frac{1}{2}\text{KL}(P||P_A) + \frac{1}{2}\text{KL}(Q||P_A) \tag{4}$$

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

A lot of similarity measures... [Rubner et al., 2000]

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

- Minkowski distance or ℓ_p distances : $d_{L_r}(P,Q) = \left(\sum_i |p_i - q_i|^r\right)^{\frac{1}{r}}$
- Histogram intersection : $d_h(P,Q) = 1 - \frac{\sum_i \min(p_i, q_i)}{\sum_i q_i}$
- χ^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 ...

A lot of similarity measures... [Rubner et al., 2000]

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

- Minkowski distance or ℓ_p distances : $d_{L_r}(P,Q) = \left(\sum_i |p_i - q_i|^r\right)^{\frac{1}{r}}$
- Histogram intersection : $d_h(P,Q) = 1 - \frac{\sum_i \min(p_i, q_i)}{\sum_i q_i}$
- χ^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...)

A lot of similarity measures... [Rubner et al., 2000]

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

- Minkowski distance or ℓ_p distances : $d_{L_r}(P,Q) = \left(\sum_i |p_i - q_i|^r\right)^{\frac{1}{r}}$
- Histogram intersection : $d_h(P,Q) = 1 - \frac{\sum_i \min(p_i, q_i)}{\sum_i q_i}$
- χ^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...)

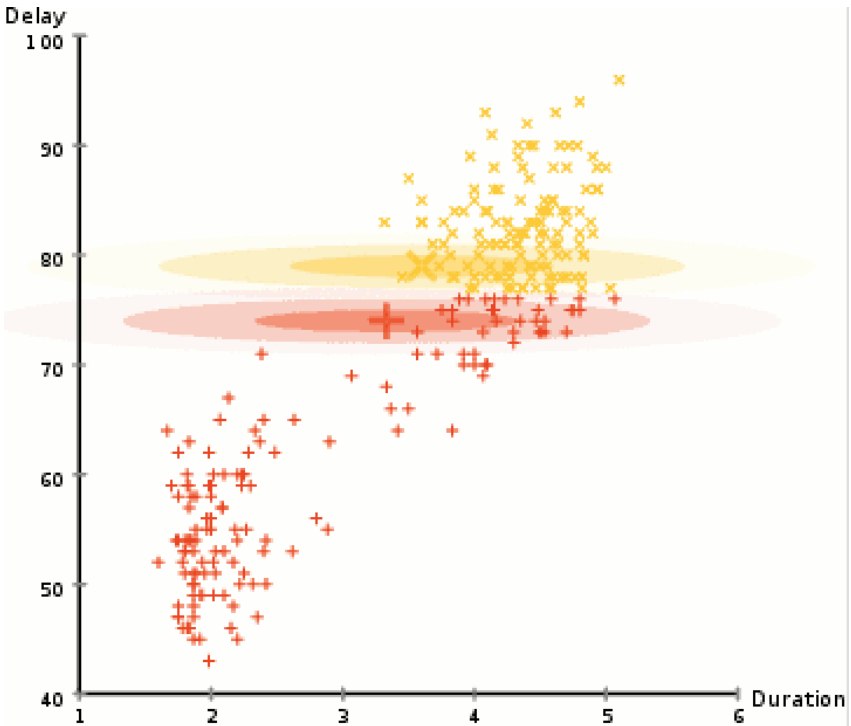
Now that we now how to compare distributions how do we learn them ?

Classical Log likelihood approach

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

$$\max_{\theta\in\Theta} \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r}[\log(P_\theta(x))] \approx \max_{\theta\in\Theta} \frac{1}{m}\sum_{i=1}^{m} \log P_\theta(x_i) \tag{5}$$

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



Asymptotic behavior

If $\mathbb{P}_r$ has a density, and $\mathbb{P}_{\theta^*}$ is the density maximizing (5) then asymptotically $\mathbb{P}_{\theta^*}$ is such that :

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

Solving (5) is equivalent to find the "closest" distribution among the given parametric family which fits the target distribution (*w.r.t* the KL-divergence)

Generative problem

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

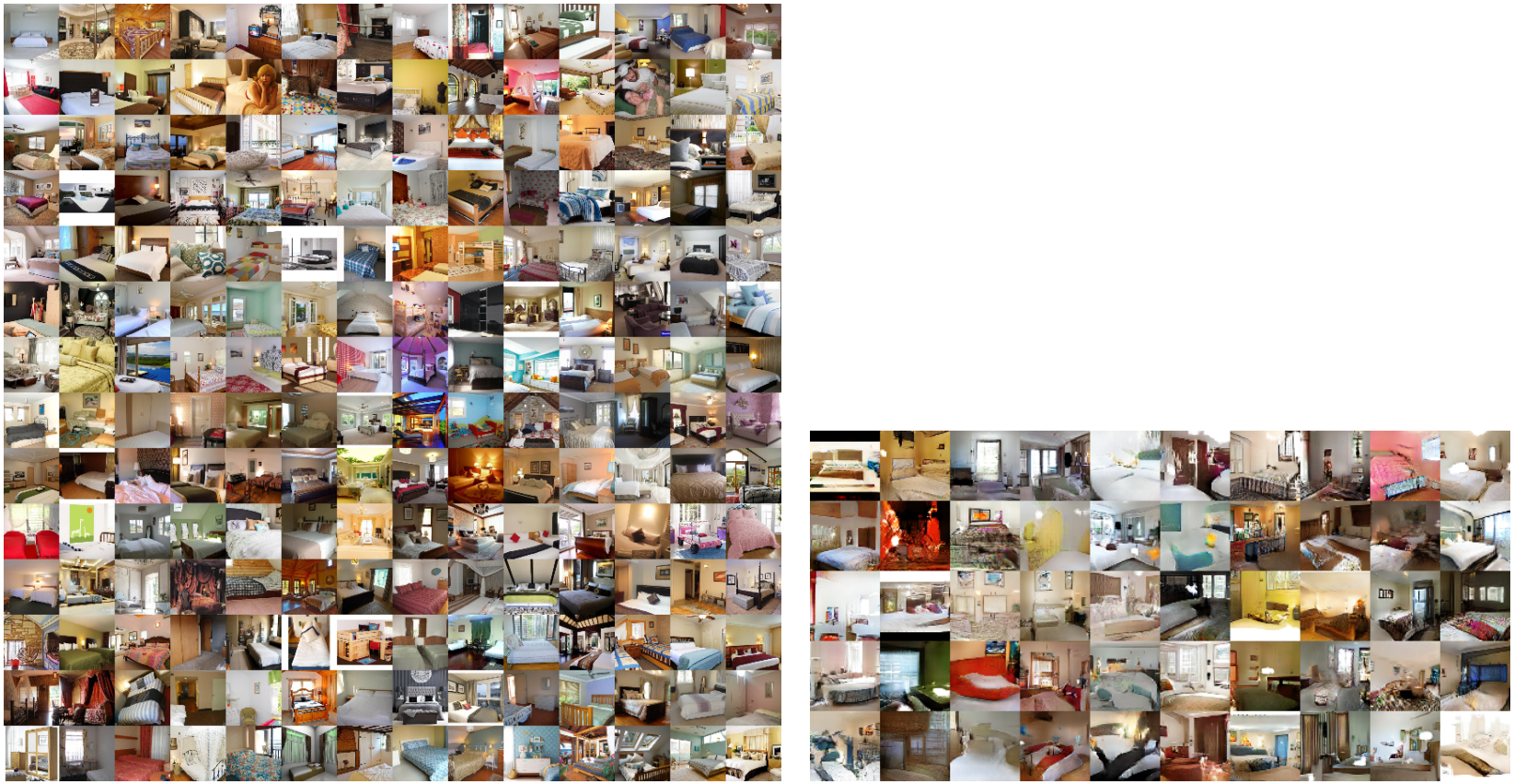
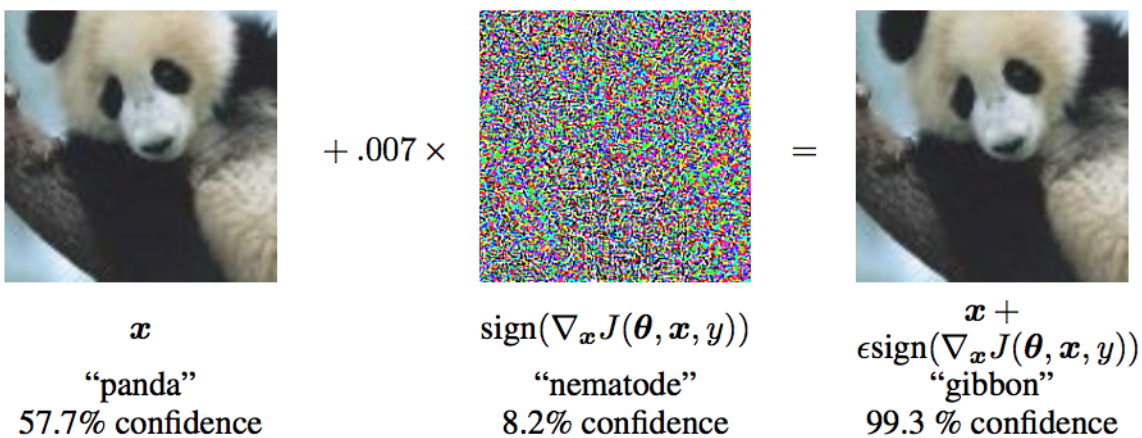


Figure 4: [Radford et al., 2015]

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 [Goodfellow et al., 2014]



Problem setting

In this approach, a generative model (the generator) is trained to learn $\mathbb{P}_g$ by fooling a feed-forward 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.

Problem setting

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

Problem setting

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

$$\max_{D} \min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x \sim \mathbb{P}_r} [\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g} [\log(1 - D(x'))] \tag{6}$$

Problem setting

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

$$\max_{D} \min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x \sim \mathbb{P}_r}[\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g}[\log(1 - D(x'))] \tag{6}$$

- $\max_{D} \mathbb{E}_{x \sim \mathbb{P}_r}[\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g}[\log(1 - D(x'))]$: 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 :

$$\max_{D} \min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x \sim \mathbb{P}_r}[\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g}[\log(1 - D(x'))] \tag{6}$$

- $\max_{D} \mathbb{E}_{x \sim \mathbb{P}_r}[\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g}[\log(1 - D(x'))]$: 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 :

$$\max_{D} \min_{G \text{ produces } \mathbb{P}_g} \mathbb{E}_{x \sim \mathbb{P}_r}[\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g}[\log(1 - D(x'))] \tag{6}$$

- $\max_{D} \mathbb{E}_{x \sim \mathbb{P}_r}[\log(D(x))] + \mathbb{E}_{x' \sim \mathbb{P}_g}[\log(1 - D(x'))]$: 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

- The more complicated the dependencies between the dimensions, the more difficult the models are to train.

How to model the generator G : latent variable

- 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.

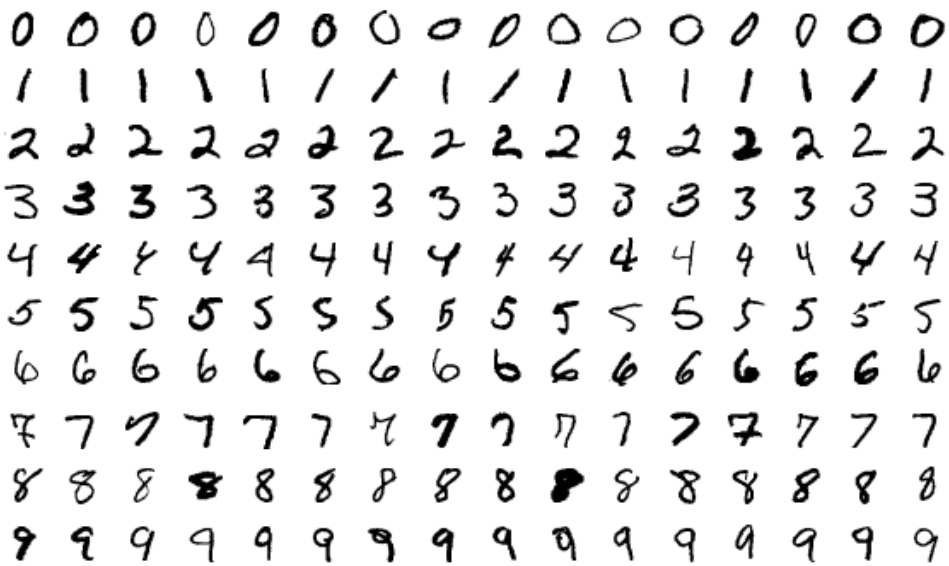
How to model the generator G : latent variable

- 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 prior helps if the model first decides which character to generate before it assigns a value to any specific pixel

How to model the generator G : latent variable

- 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 prior 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.

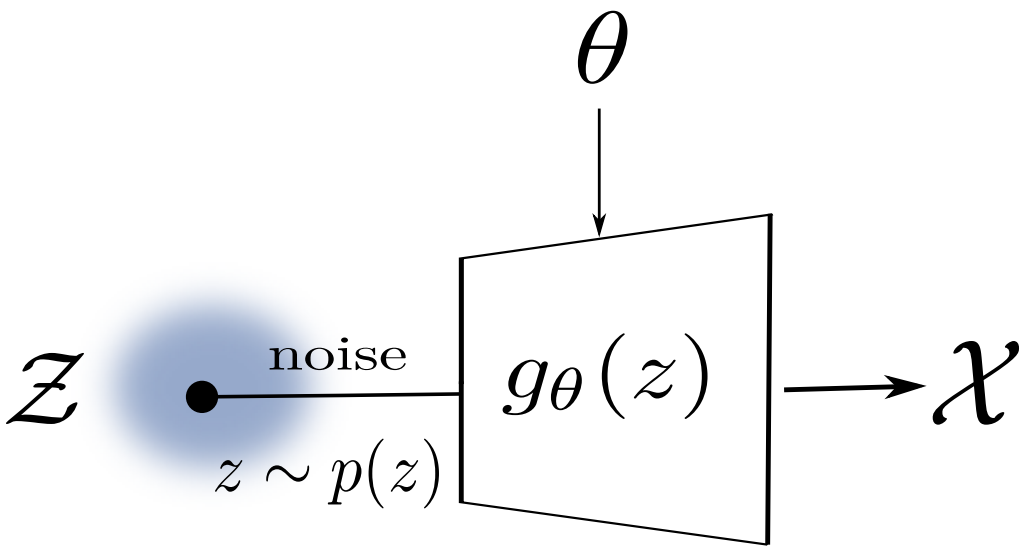


How to model the generator G : latent variable

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

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

The goal is to to optimize θ 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.

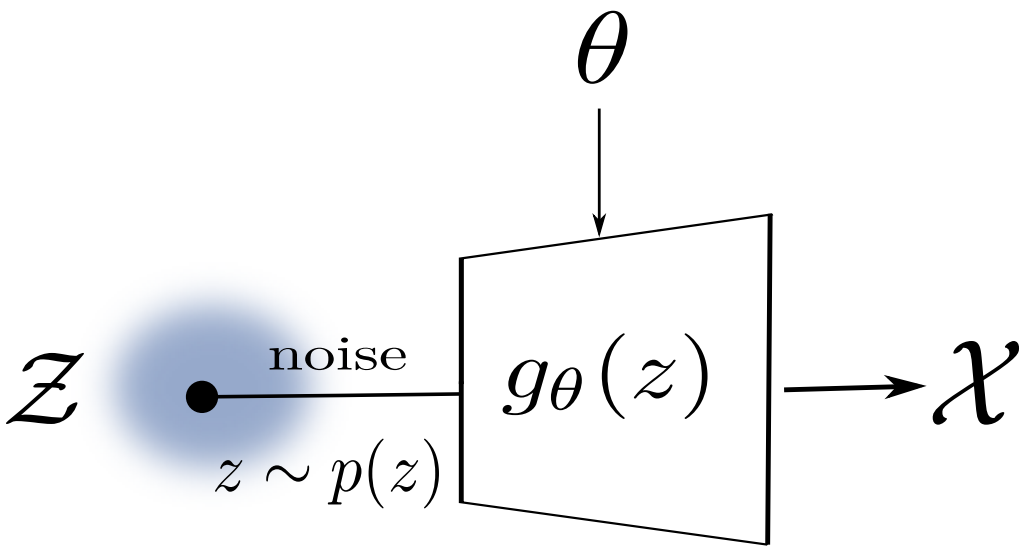


How to model the generator G : latent variable

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

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

The goal is to to optimize θ 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_θ 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

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

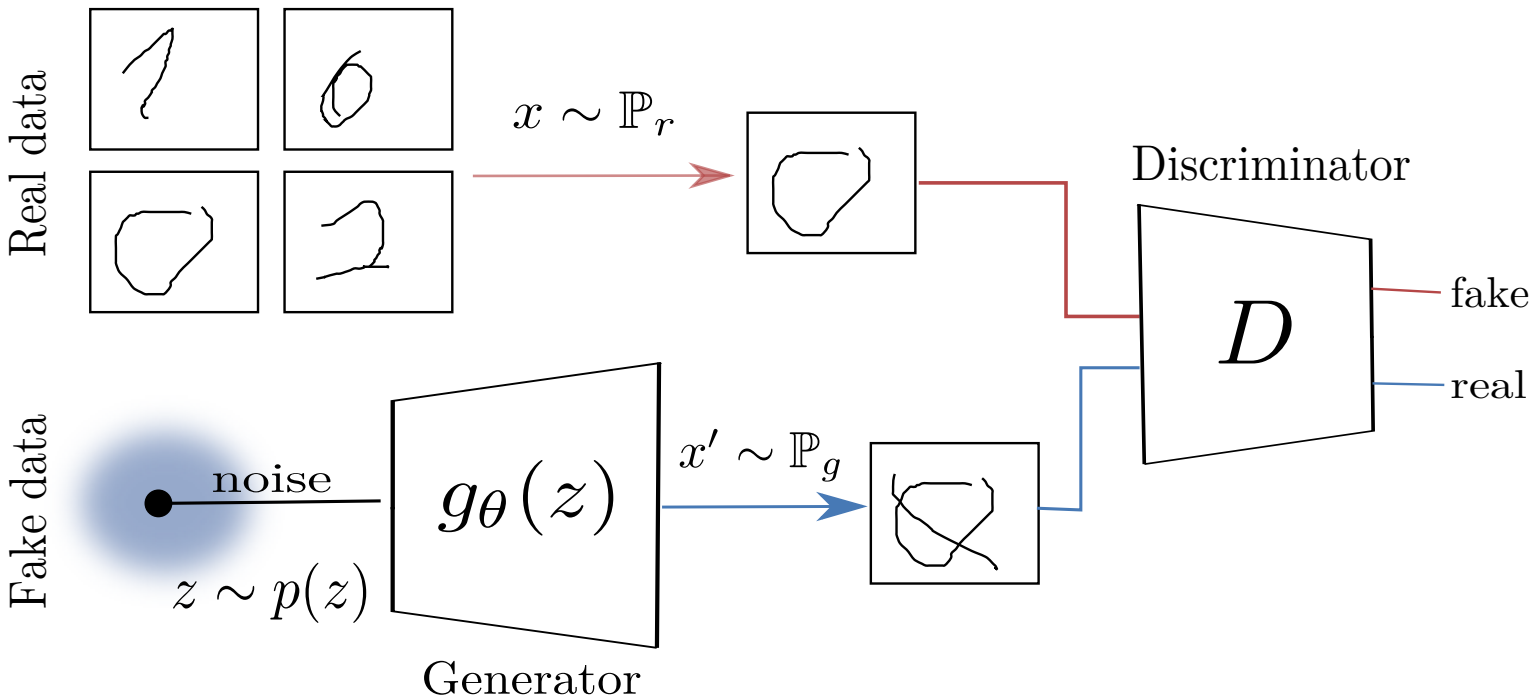
satisfies $\forall x \in K_d,\ |F(x) - f(x)| \leq \epsilon$

Putting everything together

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

$$\max_D \min_{g_\theta} L(D, g_\theta) = \mathop{\mathbb{E}}_{x \sim \mathbb{P}_r}[\log(D(x))] + \mathop{\mathbb{E}}_{z \sim p(z)}[\log(1 - D(g_\theta(z)))] \quad (7)$$

The discriminator is also au 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 (7) *w.r.t* D
- Train the generator G by minimizing (7) *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 :

$$2\text{JSD}(\mathbb{P}_r, \mathbb{P}_g) - 2\log 2 \tag{8}$$

so that we have :

Solution to (7)
The global minimum of (7) 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 θ_d and generator with parameters θ_g
2: **for** number of training iterations **do**
3: **for** k steps **do**
4: sample minibatches $\{z_1, ..., z_m\}$ from noise prior $p(z)$
5: sample minibatches $\{x_1, ..., x_m\}$ 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 $\{z_1, ..., z_m\}$ from noise prior $p(z)$
9: Update the generator by ascending its stochastic gradient

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

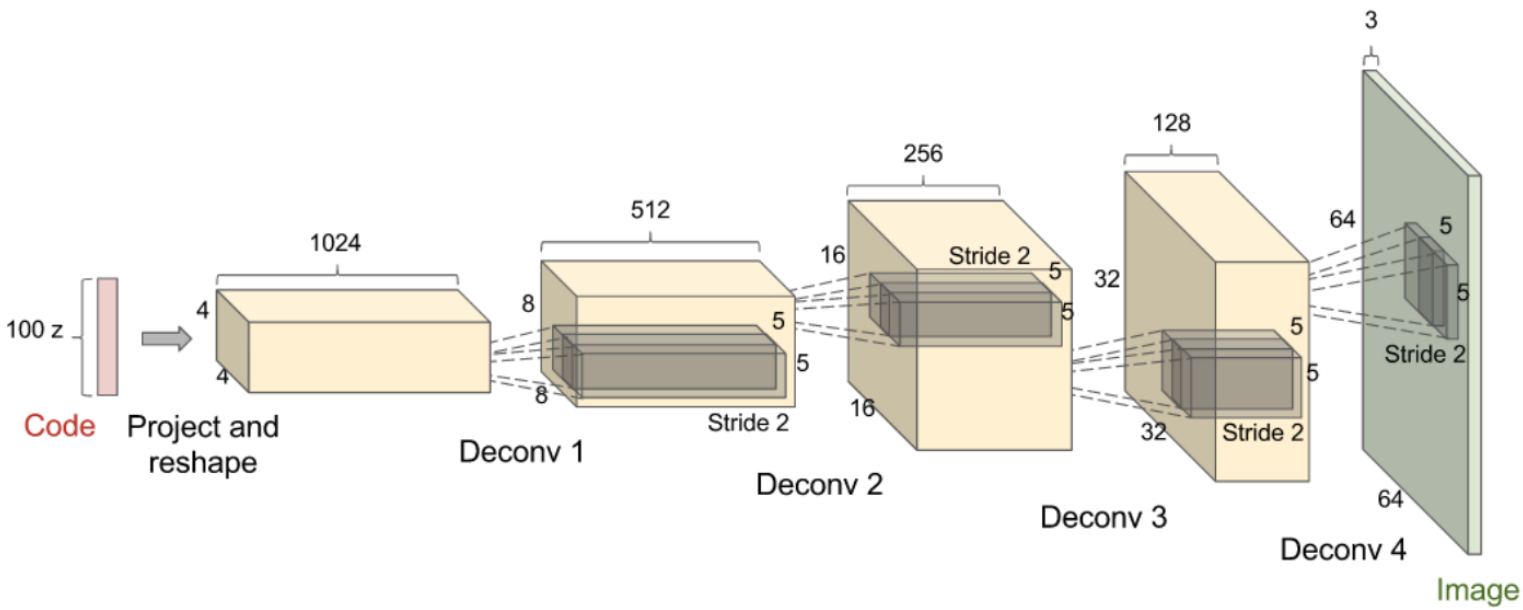
10: **end for**

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 (conditional)

We will focus on the classical GAN ([Goodfellow et al., 2014]) and DCGAN ([Radford et al., 2015]) :

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



DCGAN

- Replace any pooling layers with strided convolutions : 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 of traditional GAN's

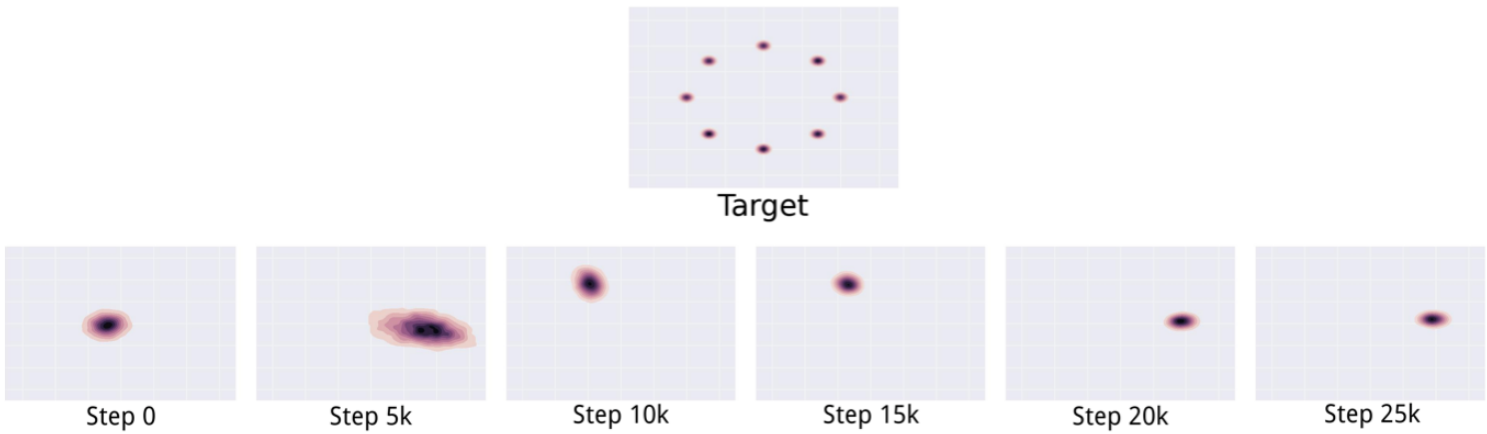
GAN training is theoretically guaranteed to converge if we can modify the density functions directly, but:

- Instead, we modify G and D : have they enough capacity to represent the distributions ?
- G and D are 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

Mode collapse or mode dropping : most severe form of non convergence

GAN's fail to learn the entire distribution but just modes of the distribution.

- When training D : convergence to the correct "best" distribution
- When training G in the inner loop : place all mass on most likely points



What is the core of mode dropping ?

Getting back to the KL divergence :

$$\mathrm{KL}(\mathbb{P}_r||\mathbb{P}_g) = \int_{\mathcal{X}} P_r(x) \log \frac{P_r(x)}{P_g(x)} dx$$

- If $P_r(x) > P_g(x)$, then x is a point with higher probability of coming from the data than being a generated sample.
- Mode dropping is when there are large regions with high values of P_r, but small or zero values in P_g.
- The integrand inside the KL grows quickly to infinity : the KL cost assigns an extremely high cost to a generator's distribution not covering parts of the data.

In practice the KL divergence occurs in training GAN's so that this situation happens.

Mode collapse causes low output diversity

this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch

the flower has petals that are bright pinkish purple with white stigma

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

Why ? Some enlightening responses [Arjovsky and Bottou, 2017]

First reason : the distributions $\mathbb{P}_r$ and $\mathbb{P}_g$ lie on small dimensional subsets that don't intersect so that KL divergences are undefined.

Why ? Some enlightening responses [Arjovsky and Bottou, 2017]

First reason : the distributions $\mathbb{P}_r$ and $\mathbb{P}_g$ lie on small dimensional subsets that don't intersect so that KL divergences are undefined.

In theory : the trained discriminator have cost at most $2 \log 2 - 2\text{JSD}(\mathbb{P}_r || \mathbb{P}_g)$.

- In practice, if we just train D till convergence, its error will go to 0
- The only way this can happen is if the distributions are not continuous, or they have disjoint supports.

Why ? Some enlightening responses [Arjovsky and Bottou, 2017]

Theorem

Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions whose support lies in two manifolds that don't have full dimension and don't perfectly align. Then :

- $\mathrm{JSD}(\mathbb{P}_r||\mathbb{P}_g) = \log 2$
- $\mathrm{KL}(\mathbb{P}_r||\mathbb{P}_g) = \infty$
- $\mathrm{KL}(\mathbb{P}_g||\mathbb{P}_r) = \infty$

Why ? Some enlightening responses [Arjovsky and Bottou, 2017]

Theorem

Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions whose support lies in two manifolds that don't have full dimension and don't perfectly align. Then :

- $\text{JSD}(\mathbb{P}_r||\mathbb{P}_g) = \log 2$
- $\text{KL}(\mathbb{P}_r||\mathbb{P}_g) = \infty$
- $\text{KL}(\mathbb{P}_g||\mathbb{P}_r) = \infty$

Use this divergences to test similarities between the distributions is a terrible : these divergences are always maxed out -> minimizing them by gradient descent is not really possible.

Yet (8) shows that the generator step is minimizing a JSD

Why ? Some enlightening responses [Arjovsky and Bottou, 2017]

Second reason : vanishing gradients

Theorem

Let $g_\theta : \mathcal{Z} \to \mathcal{X}$ be be a differentiable function that induces a distribution $\mathbb{P}_g$. Let $\mathbb{P}_r$ be the real data distribution. Suppose that the supports lies in two manifolds that don't have full dimension and don't perfectly align. Let D be a differentiable discriminator. Then :

$$\lim_{\|D-D^*\|\to 0} \nabla_\theta \mathop{\mathbb{E}}_{z\sim p(z)} [\log(1 - D(g_\theta(z))] = 0$$

Why ? Some enlightening responses [Arjovsky and Bottou, 2017]

Second reason : vanishing gradients

Theorem

Let $g_\theta : \mathcal{Z} \to \mathcal{X}$ be be a differentiable function that induces a distribution $\mathbb{P}_g$. Let $\mathbb{P}_r$ be the real data distribution. Suppose that the supports lies in two manifolds that don't have full dimension and don't perfectly align. Let D be a differentiable discriminator. Then :

$$\lim_{\|D-D^*\|\to 0} \nabla_\theta \mathop{\mathbb{E}}_{z\sim p(z)} [\log(1 - D(g_\theta(z))] = 0$$

- As our discriminator gets better, the gradient of the generator vanishes.
- This makes it difficult to train using this cost function : decide the precise amount of training dedicated to the discriminator before the vanishing effect.

The $-\log D$ alternative

Instead of considering $\underset{z\sim p(z)}{\mathbb{E}}[\log(1 - D(g_\theta(z)))]$ we rather consider :

$$\max_D \min_{g_\theta} L(D, g_\theta) = \underset{x\sim \mathbb{P}_r}{\mathbb{E}}[\log(D(x))] + \underset{z\sim p(z)}{\mathbb{E}}[-\log(D(g_\theta(z)))] \tag{9}$$

- Does not necessarily suffer from vanishing gradients : more stable cost function
- Still suffers from mode dropping

Evaluation of GAN's

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

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.

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 [Borji, 2018]

	Measure	Description
Quantitative	1. Average Log-likelihood [33, 102]	• 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). $L = \frac{1}{N}\sum_i \log P_{model}(\mathbf{x}_i)$
	2. Coverage Metric [104]	• The probability mass of the true data "covered" by the model distribution $C := P_{data}(dP_{model} > t)$ with t such that $P_{model}(dP_{model} > t) = 0.95$
	3. Inception Score (IS) [89]	• KLD between conditional and marginal label distributions over generated data. $\exp\left(\mathbb{E}_{\mathbf{x}}\left[\mathbb{KL}\left(p\left(y \mid \mathbf{x}\right) \parallel p\left(y\right)\right)\right]\right)$
	4. Modified Inception Score (m-IS) [36]	• Encourages diversity within images sampled from a particular category. $\exp(\mathbb{E}_{\mathbf{x}_i}[\mathbb{E}_{\mathbf{x}_j}[(\mathbb{KL}(P(y\|\mathbf{x}_i)\|P(y\|\mathbf{x}_j))]])$
	5. Mode Score (MS) [11]	• Similar to IS but also takes into account the prior distribution of the labels over real data. $\exp\left(\mathbb{E}_{\mathbf{x}}\left[\mathbb{KL}\left(p\left(y \mid \mathbf{x}\right) \parallel p\left(y^{train}\right)\right)\right] - \mathbb{KL}\left(p\left(y\right) \parallel p\left(y^{train}\right)\right)\right)$
	6. AM Score [132]	• Takes into account the KLD between distributions of training labels vs. predicted labels, as well as the entropy of predictions. $\mathbb{KL}(p(y^{\text{train}}) \parallel p(y)) + \mathbb{E}_{\mathbf{x}}\left[H(y\|\mathbf{x})\right]$
	7. Fréchet Inception Distance (FID) [37]	• Wasserstein-2 distance between multi-variate Gaussians fitted to data embedded into a feature space $FID(r, g) = \|\mu_r - \mu_g\|_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}})$
	8. Maximum Mean Discrepancy (MMD) [34]	• Measures the dissimilarity between two probability distributions P_r and P_g using samples drawn independently from each distribution. $M_k(P_r, P_g) = \mathbb{E}_{\mathbf{x},\mathbf{x}'\sim P_r}[k(\mathbf{x},\mathbf{x}')] - 2\mathbb{E}_{\mathbf{x}\sim P_r,\boldsymbol{y}\sim P_g}[k(\mathbf{x},\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{y},\boldsymbol{y}'\sim P_g}[k(\boldsymbol{y},\boldsymbol{y}')]$
	9. The Wasserstein Critic [2]	• The critic (e.g., an NN) is trained to produce high values at real samples and low values at generated samples $\hat{W}(\mathbf{x}_{test}, \mathbf{x}_g) = \frac{1}{N}\sum_{i=1}^{N}\hat{f}(\mathbf{x}_{test}[i]) - \frac{1}{N}\sum_{i=1}^{N}\hat{f}(\mathbf{x}_g[i])$
	10. Birthday Paradox Test [3]	• Measures the support size of a discrete (continuous) distribution by counting the duplicates (near duplicates)
	11. Classifier Two Sample Test (C2ST) [55]	• Answers whether two samples are drawn from the same distribution (e.g., by training a binary classifier)
	12. Classification Performance [81, 44]	• An indirect technique for evaluating the quality of unsupervised representations (e.g., feature extraction; FCN score). See also the GAN Quality Index (GQI) [122].
	13. Boundary Distortion [90]	• Measures diversity of generated samples and covariate shift using classification methods.
	14. Number of Statistically-Different Bins (NDB) [84]	• 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
	15. Image Retrieval Performance [112]	• Measures the distributions of distances to the nearest neighbors of some query images (i.e., diversity)
	16. Generative Adversarial Metric (GAM) [42]	• Compares two GANs by having them engaged in a battle against each other by swapping discriminators or generators. $p(\mathbf{x}\|y=1;M_1')/p(\mathbf{x}\|y=1;M_2') = \big(p(y=1\|\mathbf{x};D_1)p(\mathbf{x};G_2)\big)/\big(p(y=1\|\mathbf{x};D_2)p(\mathbf{x};G_1)\big)$
	17. Tournament Win Rate and Skill Rating [78]	• 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.
	18. Normalized Relative Discriminative Score (NRDS) [130]	• Compares n 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.
	19. Adversarial Accuracy and Divergence [121]	• 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 $P_g(y\|\mathbf{x})$ and $P_r(y\|\mathbf{x})$. Adversarial Divergence: Computes $\mathbb{KL}(P_g(y\|\mathbf{x}), P_r(y\|\mathbf{x}))$
	20. Geometry Score [48]	• Compares geometrical properties of the underlying data manifold between real and generated data.
	21. Reconstruction Error [119]	• Measures the reconstruction error (e.g., L_2 norm) between a test image and its closest generated image by optimizing for z (i.e., $min_{\mathbf{z}}\|\|G(\mathbf{z}) - \mathbf{x}^{(test)}\|\|^2$)
	22. Image Quality Measures [115, 85, 46]	• Evaluates the quality of generated images using measures such as SSIM, PSNR, and sharpness difference
	23. Low-level Image Statistics [126, 47]	• 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.
	24. Precision, Recall and F_1 score [67]	• These scores are used to quantify the degree of overfitting in GANs, often over toy datasets.
Qualitative	1. Nearest Neighbors	• To detect overfitting, generated samples are shown next to their nearest neighbors in the training set
	2. Rapid Scene Categorization [33]	• 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
	3. Preference Judgment [40, 128, 120, 124]	• Participants are asked to rank models in terms of the fidelity of their generated images (e.g., pairs, triples)
	4. Mode Drop and Collapse [99, 59]	• 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
	5. Network Internals [81, 13, 38, 69, 125, 5]	• Regards exploring and illustrating the internal representation and dynamics of models (e.g., space continuity) as well as visualizing learned features

The origins of optimal transport

666 · MÉMOIRES DE L'ACADÉMIE ROYALE

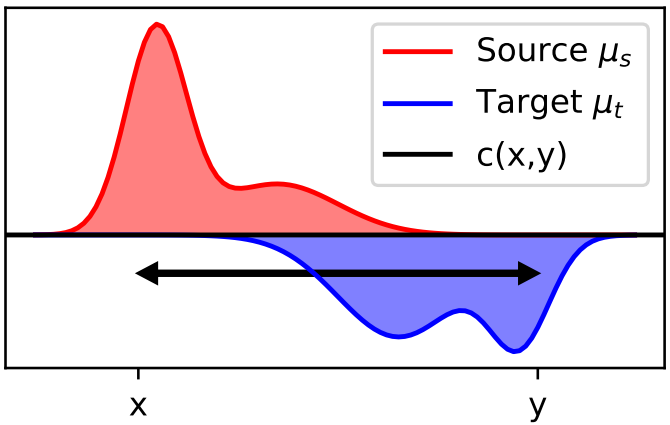
MÉMOIRE
SUR LA
THÉORIE DES DÉBLAIS
ET DES REMBLAIS.

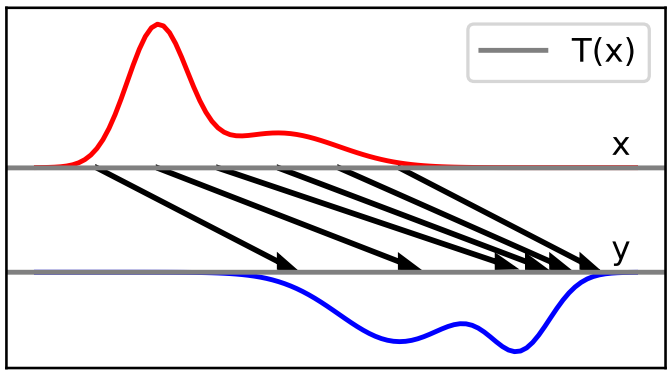
Par M. MONGE.

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



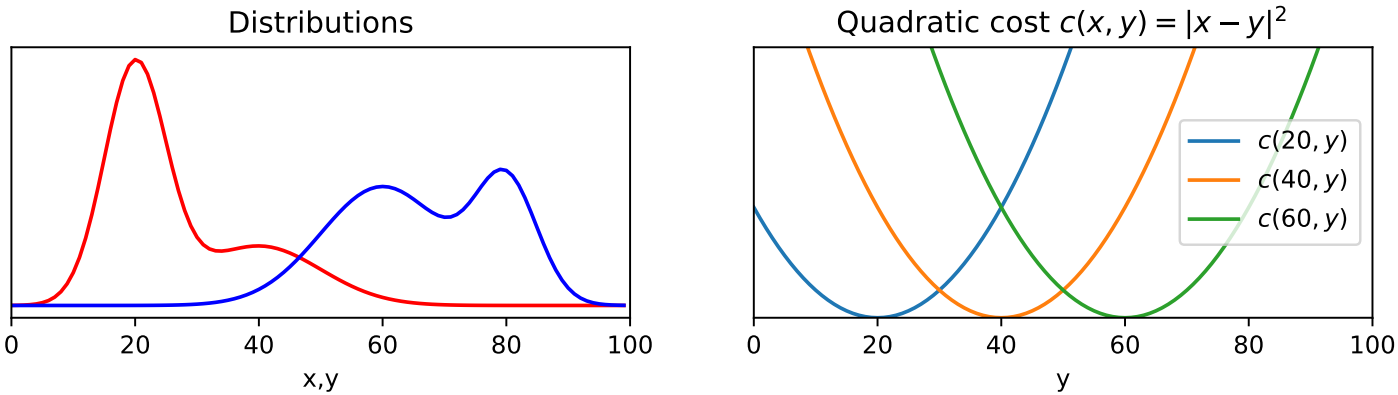


Problem

- 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)

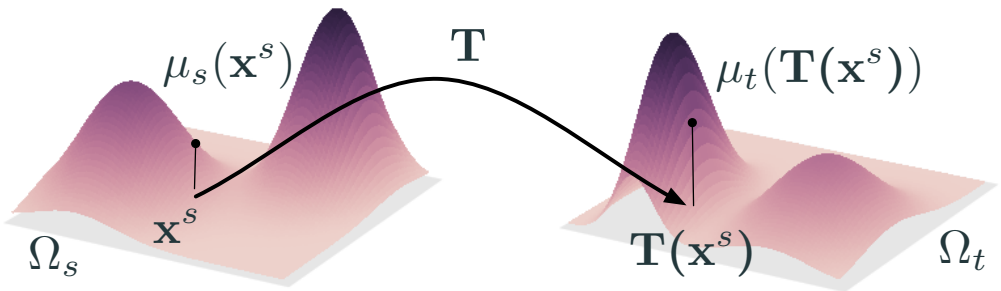
- Mathematical tools aiming at comparing distributions



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

$$\inf_{T\#\mu_s = \mu_t} \int_{\Omega_s} d(x, T(x))\mu_s(x)dx \tag{10}$$

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



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

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

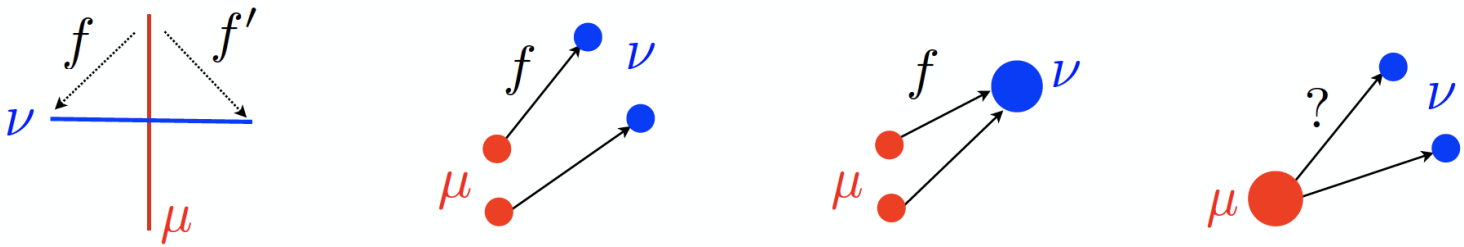
- consists simply in moving the positions of all the points in the support of the measure : for $\mu_s = \sum_{i=1}^{n} a_i \delta_{x_i}$

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

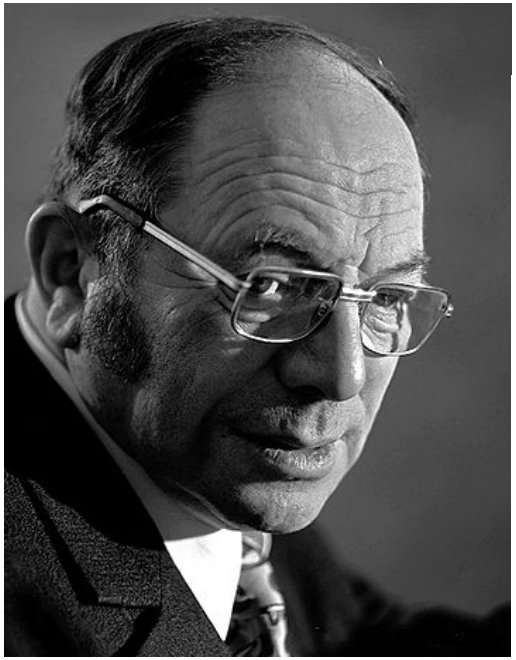
Non-existence Non-uniqueness

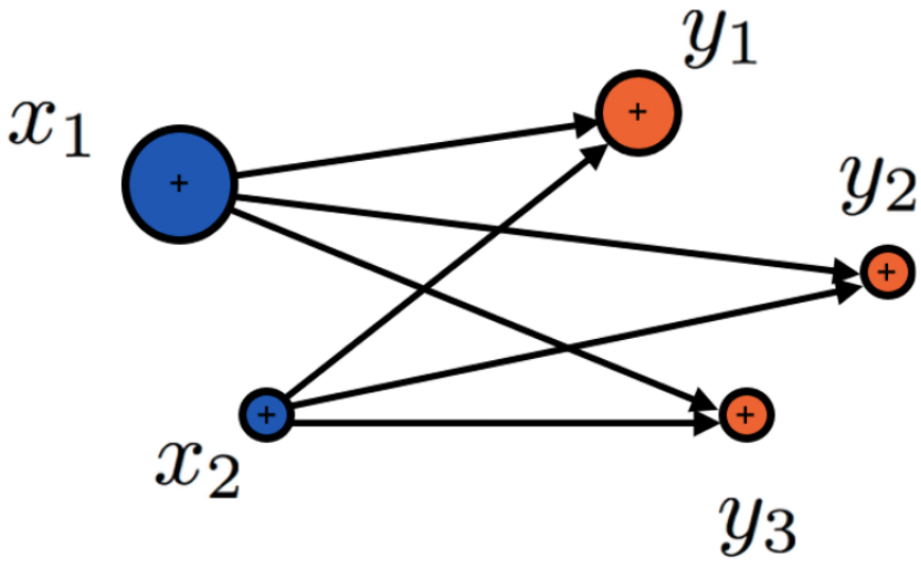
Solving for this push-forward operator is a non-convex optimization problem,

- for which existence is not guaranteed,
- nor unicity



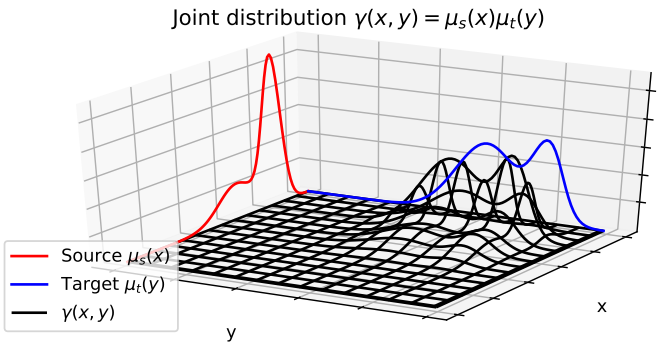
Kantorovich relaxation

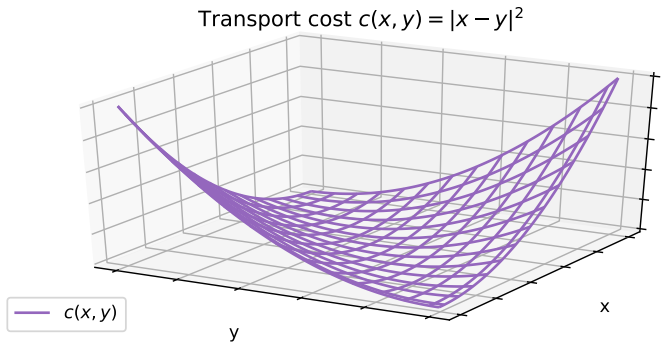




- Leonid Kantorovich (1912–1986), Economy nobelist in 1975, proposed a different formulation of the problem
- with applications mainly for resource allocation problems

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

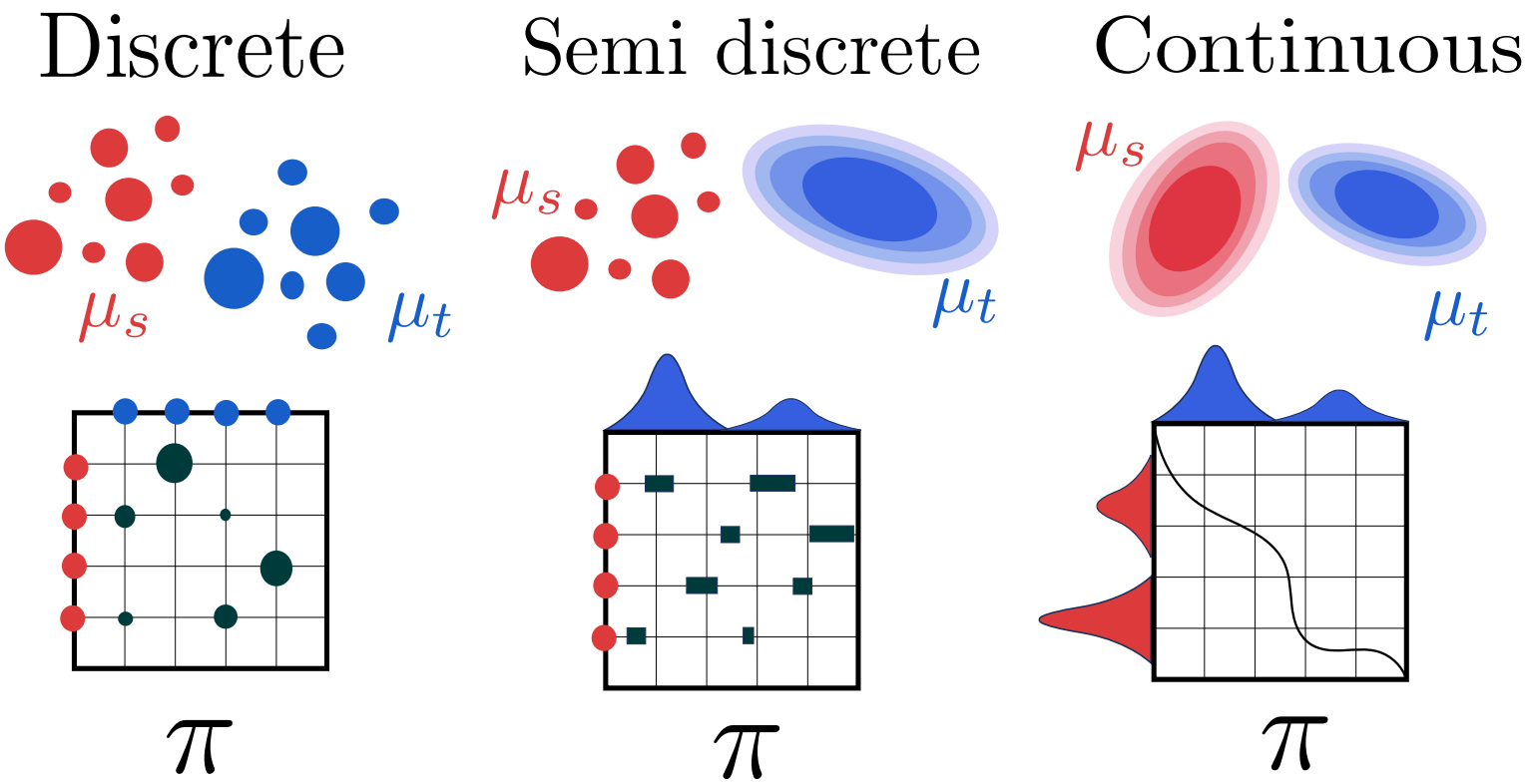
- The Kantorovich formulation seeks for a probabilistic coupling $\pi \in \Pi(\mu_s \times \mu_t)$ between μ_s and μ_t.
- π is a joint probability measure with prescribed marginals μ_s and μ_t.
- Computes the Wasserstein distance :

$$\mathcal{W}_p(\mu_s, \mu_t) = \left(\min_{\pi \in \Pi(\mu_s, \mu_t)} \sum_{i,j} d(x_i, y_j)^q \pi_{i,j} \right)^{\frac{1}{p}} \tag{11}$$

Can be solved by linear programming

Probabilistic couplings

The resulting coupling π associates in a "fuzzy" way the points of the distributions.



Properties of Wasserstein distance

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 α in $\mathcal{P}(\Omega)$ *iff* $\mathcal{W}_p(\alpha_k, \alpha) \to 0$ as $k \to \infty$

Properties of Wasserstein distance

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 α in $\mathcal{P}(\Omega)$ *iff* $\mathcal{W}_p(\alpha_k, \alpha) \to 0$ as $k \to \infty$

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

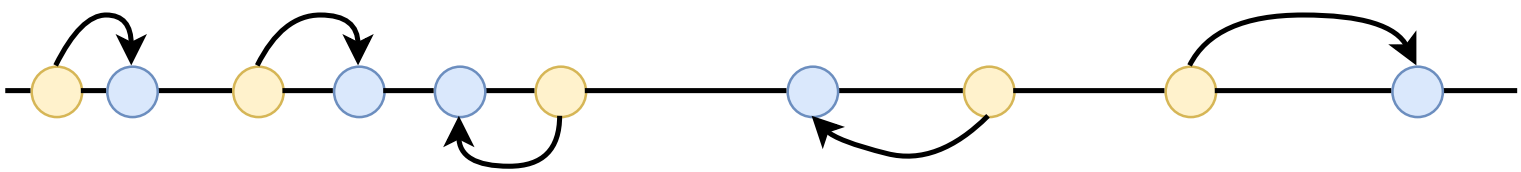
- The L2 norm can only be applied to continuous measures with a density with respect to a base measure
- The discrete ℓ_2 norm requires that positions (x_i, y_j) take values in a predetermined discrete set to work properly
- Idem KL divergence is not defined if the supports of the measures does not overlap

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 μ_s onto μ_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 $f \in \text{Lip}^1$ (set of 1-Lipschitz functions) as

$$\sup_{f \in \text{Lip}^1} \int f d(\mu_s - \mu_t) = \sup_{f \in \text{Lip}^1} \mathbb{E}_{x \sim \mu_s}[f(x)] - \mathbb{E}_{y \sim \mu_t}[f(y)] \qquad (12)$$

- known as **Kantorovich-Rubinstein duality**
- ϕ can be learnt as a neural network constrained to the set Lip^1.

Numerous applications for the Wasserstein distance :

- Learning with Wasserstein Loss [Frogner et al., 2015]
- Wasserstein GAN's [Arjovsky et al., 2017]
- Domain Adaptation [Courty et al., 2017]
- Image colorization [Ferradans et al., 2014], Dictionary Learning [Rolet et al., 2016] ...

Problem Setting [Arjovsky et al., 2017]

We consider a parametric family $(\mathbb{P}_\theta)_{\theta \in \Theta}$ and a real data distribution $\mathbb{P}_r$. We want to solve :

$$\min_{\theta \in \Theta} \mathcal{W}_1(\mathbb{P}_r, \mathbb{P}_\theta) \tag{13}$$

Solving (13) is equivalent to find the closest distribution (*w.r.t* the Wasserstein distance) among the given parametric family which fits the target distribution.

Problem Setting [Arjovsky et al., 2017]

The minimum of (13) is highly intractable but the Kantorovich-Rubinstein duality tells :

$$\mathcal{W}_1(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{f \in \mathbf{Lip^1}} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{y \sim \mathbb{P}_\theta}[f(y)]$$

If we rather consider $f \in \text{Lip}^K$ we end up with $K.\mathcal{W}_1(\mathbb{P}_r, \mathbb{P}_\theta)$

So, idea :

- $\mathbb{P}_\theta$ is coming from a neural network g_θ in the same way as traditional GAN's.
- We consider another neural network f_w called the critic (alike a discriminator)

And we solve (13) *via* :

$$\max_w \min_\theta \mathop{\mathbb{E}}_{x \sim P_r}[f_w(x)] - \mathop{\mathbb{E}}_{z \sim p(z)}[f_w(g_\theta(z))] \tag{14}$$

How do we find the function f^* that solves the maximization problem ?

How do we force f_w to be K-Lip ??

- f_w is parameterized by a neural network with weights w lying in a compact space $\mathcal{W}$ imply that f_w is K-Lip.
- K only depend on $\mathcal{W}$
- To force f_w to be K-Lip we can clamp the weights to a fixed box (*e.g* $\mathcal{W} = [-0.01, 0.01]^d$) after each gradient update

How do we find the function f^* that solves the maximization problem ?

How do we force f_w to be K-Lip ??

- f_w is parameterized by a neural network with weights w lying in a compact space $\mathcal{W}$ imply that f_w is K-Lip.
- K only depend on $\mathcal{W}$
- To force f_w to be K-Lip we can clamp the weights to a fixed box (*e.g* $\mathcal{W} = [-0.01, 0.01]^d$) after each gradient update

But :

- Weight clipping is a clearly terrible way to enforce a Lipschitz constraint
- If the clipping parameter is large : long time for any weights to reach their limit, making it harder to train the critic till optimality
- If the clipping is small : this can easily lead to vanishing gradients

Implementation of WGAN

Algorithm 2 pseudo code of GAN

Require :α the learning rate. c, the clipping parameter. m, the batch size. n_c, the number of iterations of the critic per generator iteration

Require : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ_g has not converged **do**

  for $t = 0, ..., n_c$ **do**

    sample minibatches $\{z_1, ..., z_m\}$ from noise prior $p(z)$

    sample minibatches $\{x_1, ..., x_m\}$ from data $\mathbb{P}_r$

    Update f_w by ascending its stochastic gradient

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

    $w \leftarrow \text{clip}(-c, c)$

  end for

  sample minibatches $\{z_1, ..., z_m\}$ from noise prior $p(z)$

  Update the generator by ascending its stochastic gradient $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} f_w(\mathbf{g}_{\theta_g}(z_i))$

end while

WGAN: the devil in the approximation

Neural network belonging to Lip^1 ?

- Not really!
- It is actually the supremum over K-Lipschitz functions that is approximated by a neural network
- Actually **not** equivalent to solve the optimal transport, but gradients are aligned.

Improved WGAN [Gulrajani et al., 2017]

$$\min_{\theta} \sup_{f \in \text{NN class}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}[f(g_\theta(\mathbf{z}))] + \lambda \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_r}[(||\nabla f(\mathbf{x})||_2 - 1)^2]$$

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

Autoencoder & Variational autoencoders

Problem setting

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

Very generally autoencoders aim at solving :

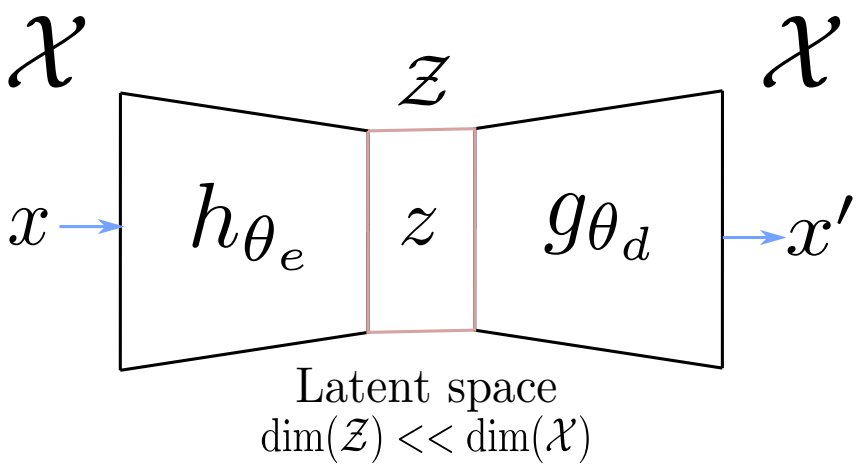
$$\min_{\theta_e, \theta_d} \mathop{\mathbb{E}}_{x \sim \mathbb{P}_r} [L(x, g_{\theta_d}, h_{\theta_e})] \tag{15}$$

Traditional autoencoders

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

$$\min_{\theta_e, \theta_d} \mathop{\mathbb{E}}_{x \sim \mathbb{P}_r} [L(x, g_{\theta_d}(h_{\theta_e}(x)))] \tag{16}$$

- L is chosen to be a reconstruction error as ℓ_2 error
- Forces the network to acts as the indentity. 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.



Learning manifolds

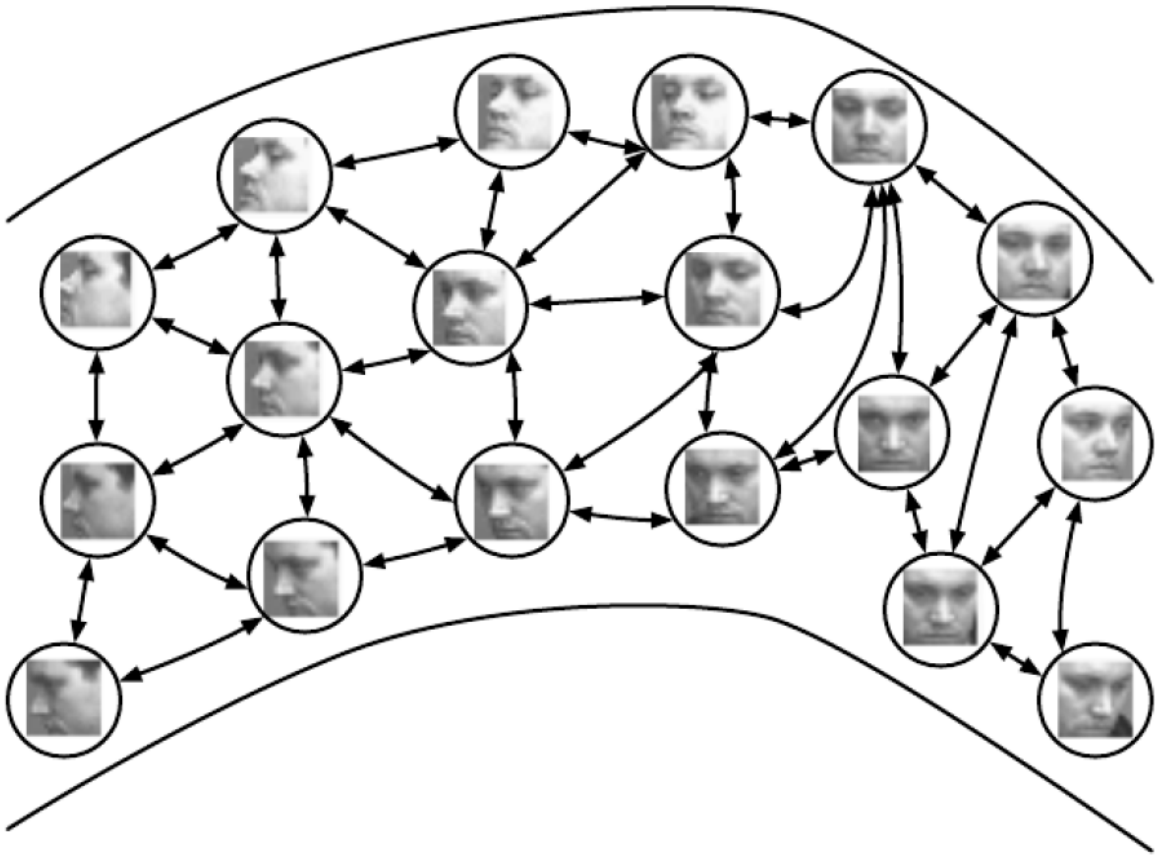


Figure 5: Non-parametric manifold learning procedures build a nearest neighbor graph in which nodes represent training examples a directed edges indicate nearest neighbor relationships. The coordinate system associates each training example with a real-valued vector position in the embedding $\mathcal{Z}$. [Gong et al., 2000]

Dimensionality reduction

- 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.

Many variants

- 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

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

  - Ω 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

Many variants

- 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_{θ_e} and g_{θ_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.

The probabilistic point of view

Encoder and decoder previously defined can be seen as distributions :

- From an input $x \in \mathcal{X}$ of the data and from parameters θ_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 θ_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.

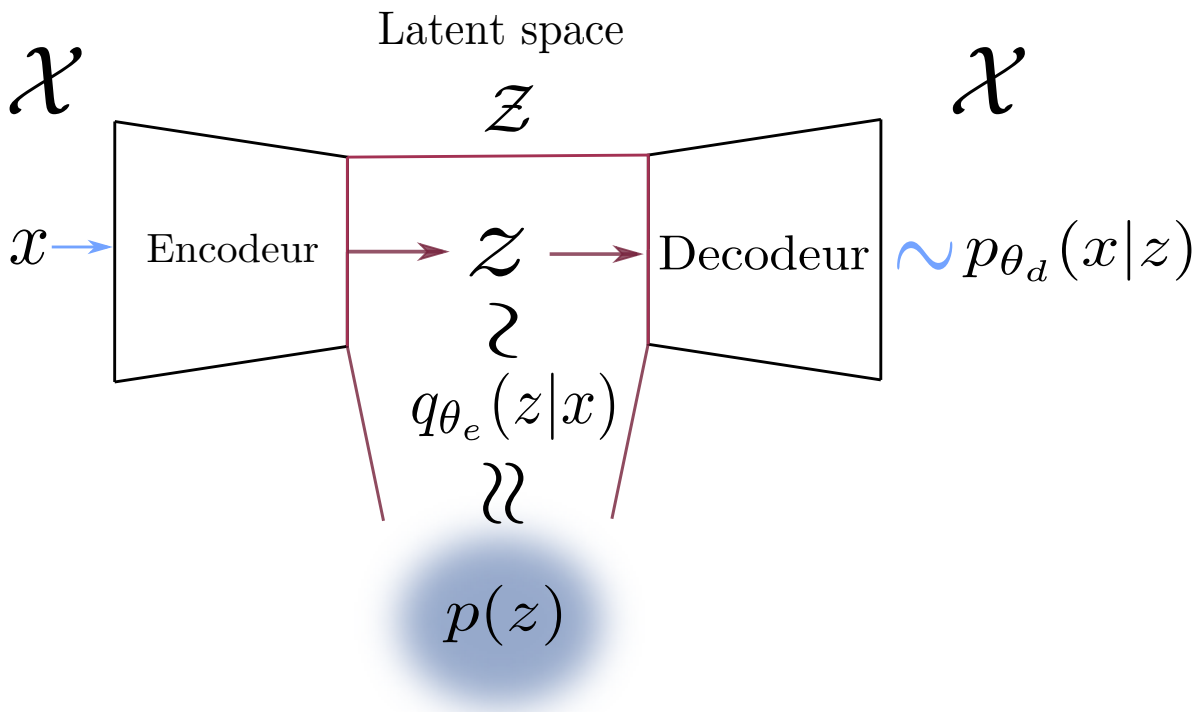
Problem setting

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* θ_e and θ_d the following quantity :

$$\underset{z\sim q_{\theta_e}(z|x)}{-\mathbb{E}}\left[\log(p_{\theta_d}(x|z))\right] + \mathrm{KL}(q_{\theta_e}(z|x)||p(z)) \tag{17}$$

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



Problem setting

- $-\mathbb{E}_{z\sim q_{\theta_e}(z|x)}[\log(p_{\theta_d}(x|z))]$ is the reconstruction loss : encourages the decoder to learn to reconstruct the data $\mathbb{P}_r$.
- $\mathrm{KL}(q_{\theta_e}(z|x)||p(z))$ 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.

Minimize the loss

- $-\mathbb{E}_{z\sim q_{\theta_e}(z|x)}[\log(p_{\theta_d}(x|z))]$: is a maximum likelihood estimation. Given an input z and a output x we want to maximize for example log loss or regression loss.
- What about $\mathrm{KL}(q_{\theta_e}(z|x)||\mathcal{N}(0,I))$? Let us say that we want $q_{\theta_e}(z|x) = \mathcal{N}(\mu(x), \Sigma(x))$ then

$$\mathrm{KL}(q_{\theta_e}(z|x)||\mathcal{N}(0,I)) = \frac{1}{2}\left(\mathrm{tr}(\Sigma(x)) + \mu(x)^T\mu(x) - d - \log\det(\Sigma(x))\right)$$

Deep Learning in practice: Fine-Tuning

Problem setting

In most of the Deep learning problems the first question is: which architecture to use ?

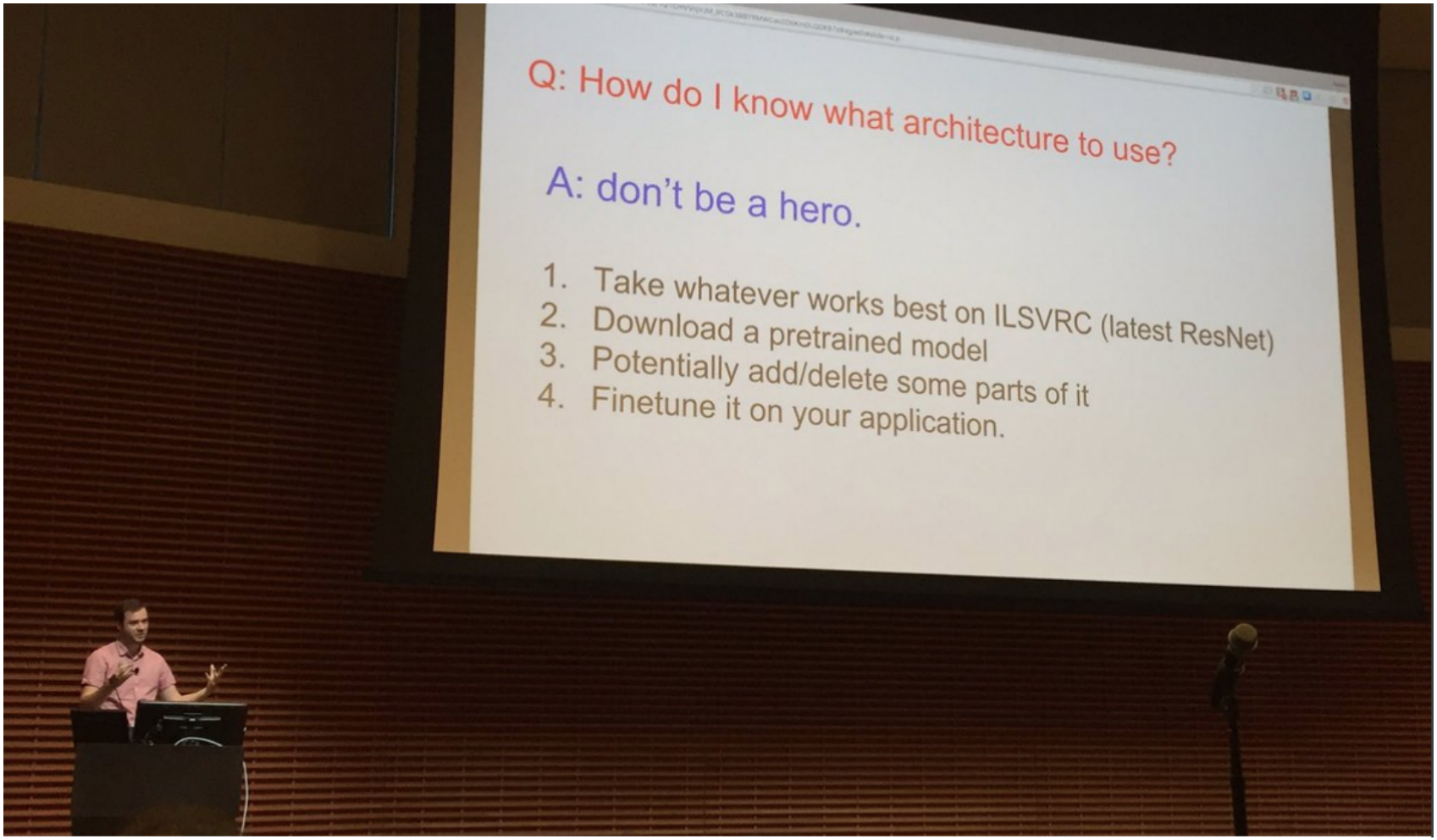
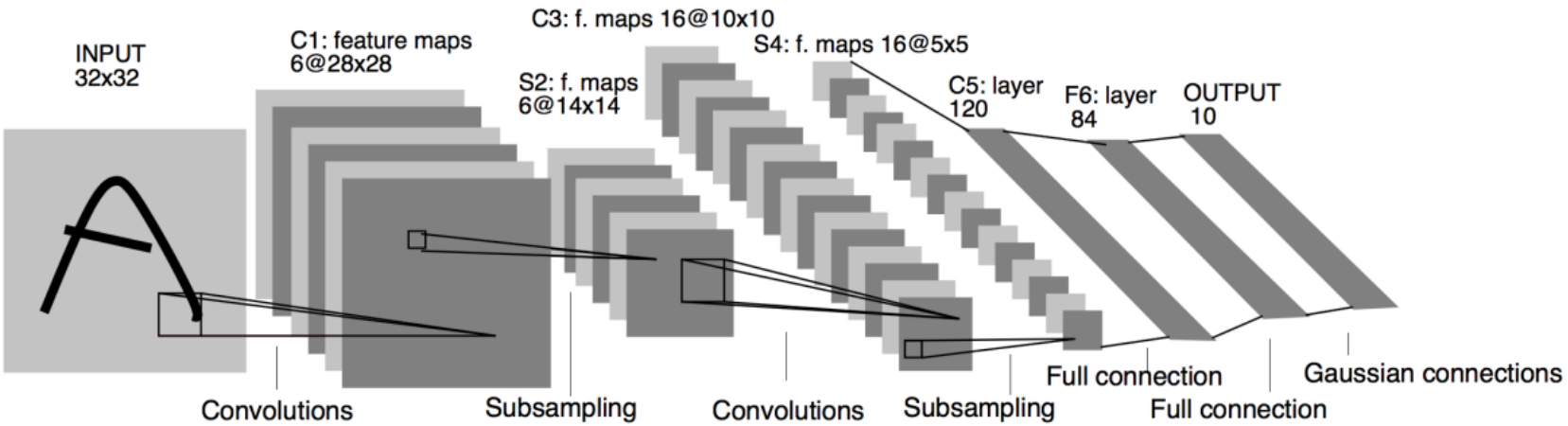


Figure 6: Andrej Karpathy (Imagenet)

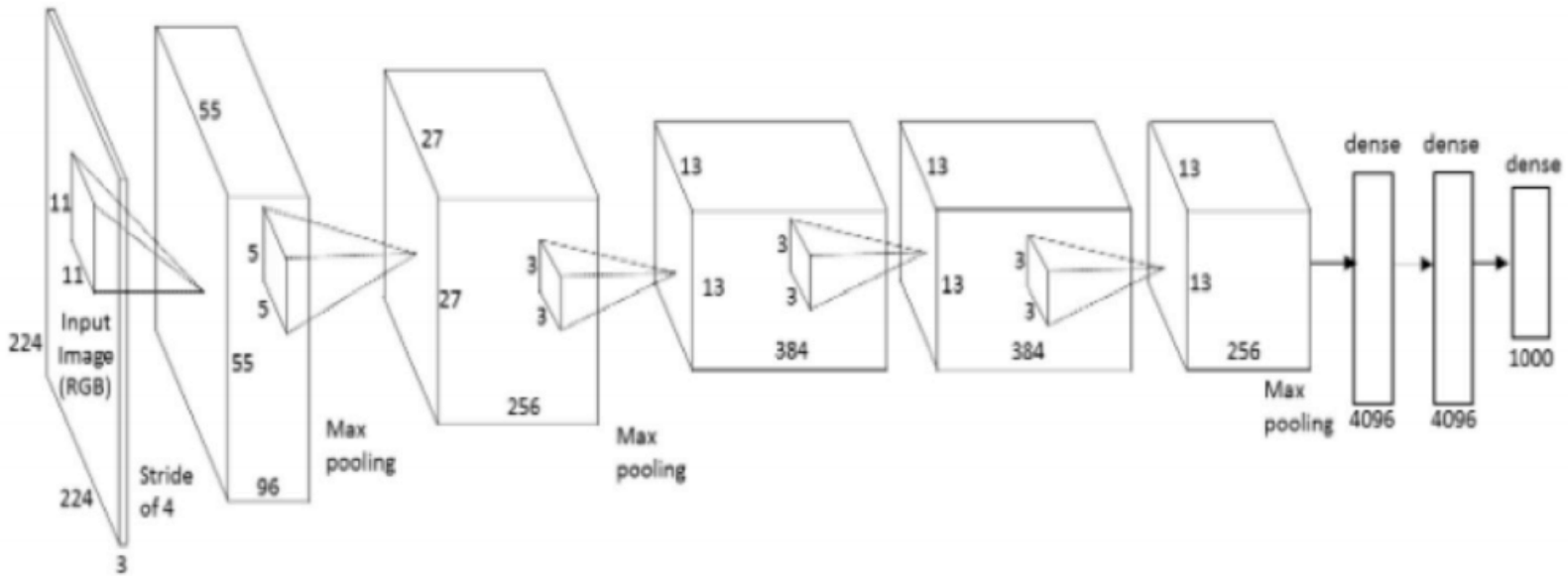
Some widely known architectures

Maybe the first architecture for images LeNet (60k parameters)
[LeCun et al., 1989]



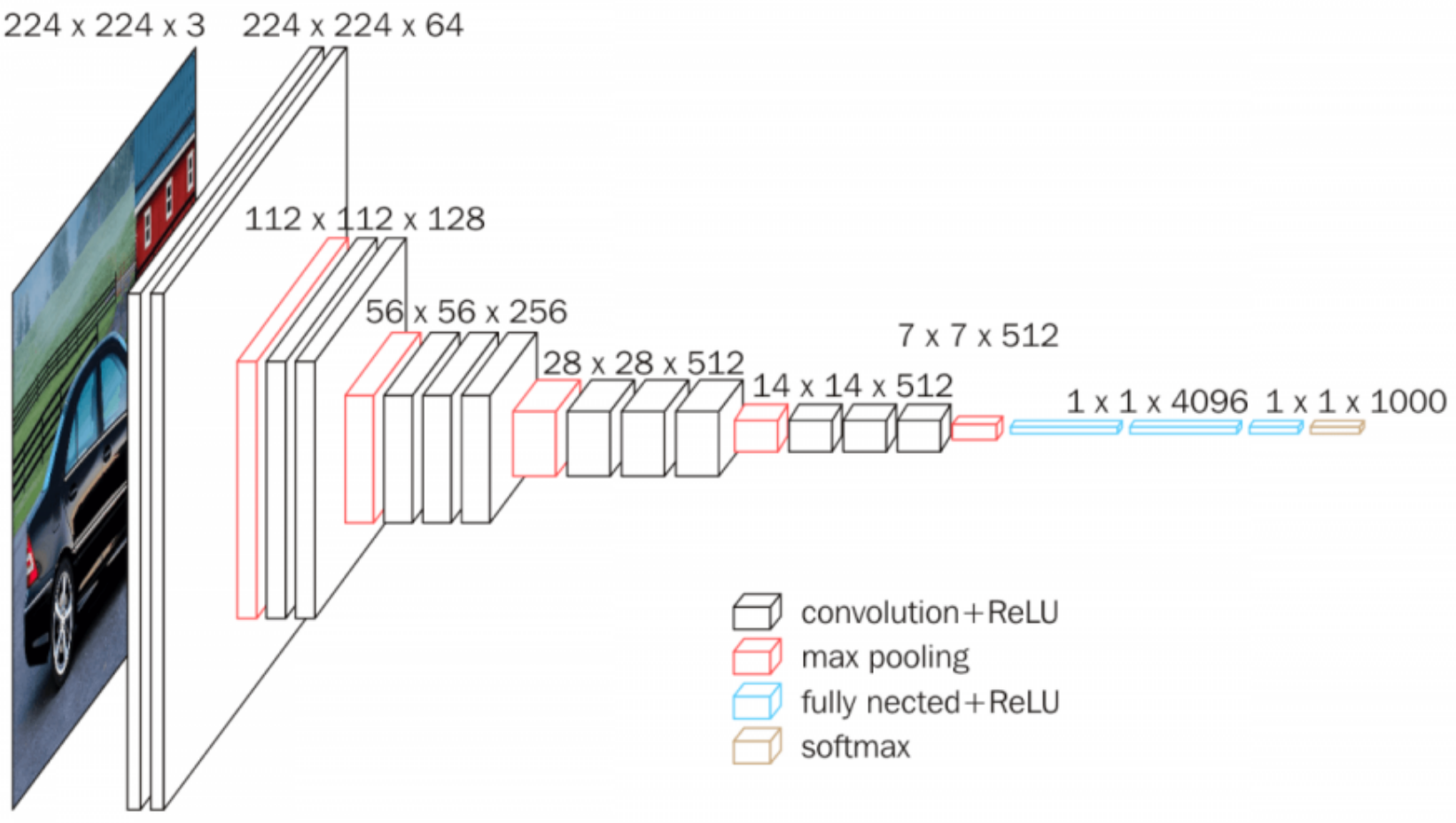
Some widely known architectures

Alexnet (60M parameters) [Krizhevsky et al., 2012]



Some widely known architectures

VGG (138M parameters) [Simonyan and Zisserman, 2014]



Some widely known architectures

Resnet (avoids vanishing gradients)
[He et al., 2015, Zagoruyko and Komodakis, 2016]

Model	top-1 err, %	top-5 err, %	#params	time/batch 16
ResNet-50	24.01	7.02	25.6M	49
ResNet-101	22.44	6.21	44.5M	82
ResNet-152	22.16	6.16	60.2M	115
WRN-50-2-bottleneck	21.9	6.03	68.9M	93
pre-ResNet-200	21.66	5.79	64.7M	154

How to use the models for your own application ? Fine-tuning

Very often "famous" models are trained on ImageNet and your image classification problem will not match ImageNet classes. Typical strategy consists is fine-tuning.

How to use the models for your own application ? Fine-tuning

Very often "famous" models are trained on ImageNet and your image classification problem will not match ImageNet classes. Typical strategy consists is fine-tuning.

- Truncate the last layer of the pre-trained network and replace it with your new softmax layer relevant to your problem.
- Use a smaller learning rate ($\sim$ 10 times smaller): pre-trained weights should not be distorted a lot.
- Freeze the weights of the first few layers: first few layers capture "universal features" that we want to keep. And it is expensive to retrain all the model!

How to use the models for your own application ? Fine-tuning

Very often "famous" models are trained on ImageNet and your image classification problem will not match ImageNet classes. Typical strategy consists is fine-tuning.

- Truncate the last layer of the pre-trained network and replace it with your new softmax layer relevant to your problem.
- Use a smaller learning rate ($\sim$ 10 times smaller): pre-trained weights should not be distorted a lot.
- Freeze the weights of the first few layers: first few layers capture "universal features" that we want to keep. And it is expensive to retrain all the model!

All Deep Learning framework (Keras/Pytorch) has pretrained popular models that can be loaded.

References i


Antoniou, A., Storkey, A., and Edwards, H. (2017).
Data Augmentation Generative Adversarial Networks.
ArXiv e-prints.


Arjovsky, M. and Bottou, L. (2017).
Towards principled methods for training generative adversarial networks.
CoRR, abs/1701.04862.


Arjovsky, M., Chintala, S., and Bottou, L. (2017).
Wasserstein GAN.
CoRR, abs/1701.07875.


Borji, A. (2018).
Pros and cons of GAN evaluation measures.
CoRR, abs/1802.03446.

References ii


Chan, C., Ginosar, S., Zhou, T., and Efros, A. A. (2018).
Everybody Dance Now.
ArXiv e-prints.


Courty, N., Flamary, R., Tuia, D., and Rakotomamonjy, A. (2017).
Optimal transport for domain adaptation.
IEEETPAMI, 39(9):1853–1865.


Ferradans, S., Papadakis, N., Peyré, G., and Aujol, J.-F. (2014).
Regularized discrete optimal transport.
SIAM Journal on Imaging Sciences, 7(3):1853–1882.


Frogner, C., Zhang, C., Mobahi, H., Araya-Polo, M., and Poggio, T. (2015).
Learning with a Wasserstein Loss.
ArXiv e-prints.

References iii


Gong, S., McKenna, S. J., and Psarrou, A. (2000).
Dynamic Vision: From Images to Face Recognition.
Imperial College Press, London, UK, UK, 1st edition.


Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014).
Generative adversarial nets.
In Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada, pages 2672–2680.


Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014).
Explaining and Harnessing Adversarial Examples.
ArXiv e-prints.


Gregor, K., Danihelka, I., Graves, A., Jimenez Rezende, D., and Wierstra, D. (2015).
DRAW: A Recurrent Neural Network For Image Generation.
ArXiv e-prints.


Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017).
Improved training of wasserstein gans.
CoRR, abs/1704.00028.


He, K., Zhang, X., Ren, S., and Sun, J. (2015).
Deep residual learning for image recognition.
CoRR, abs/1512.03385.


Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012).
Imagenet classification with deep convolutional neural networks.
In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc.


LeCun, Y., Bengio, Y., and Hinton, G. E. (2015).
Deep learning.
Nature, 521(7553):436–444.


LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989).
Backpropagation applied to handwritten zip code recognition.
Neural Comput., 1(4):541–551.


Radford, A., Metz, L., and Chintala, S. (2015).
Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.
ArXiv e-prints.


Reed, S. E., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016).
Generative adversarial text to image synthesis.
CoRR, abs/1605.05396.


Rolet, A., Cuturi, M., and Peyré, G. (2016).
Fast dictionary learning with a smoothed wasserstein loss.
In Gretton, A. and Robert, C. C., editors, Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, volume 51 of Proceedings of Machine Learning Research, pages 630–638, Cadiz, Spain. PMLR.


Rubner, Y., Tomasi, C., and Guibas, L. J. (2000).
The earth mover's distance as a metric for image retrieval.
International Journal of Computer Vision, 40(2):99–121.


Simonyan, K. and Zisserman, A. (2014).
Very deep convolutional networks for large-scale image recognition.
CoRR, abs/1409.1556.


Zagoruyko, S. and Komodakis, N. (2016).
Wide residual networks.
CoRR, abs/1605.07146.