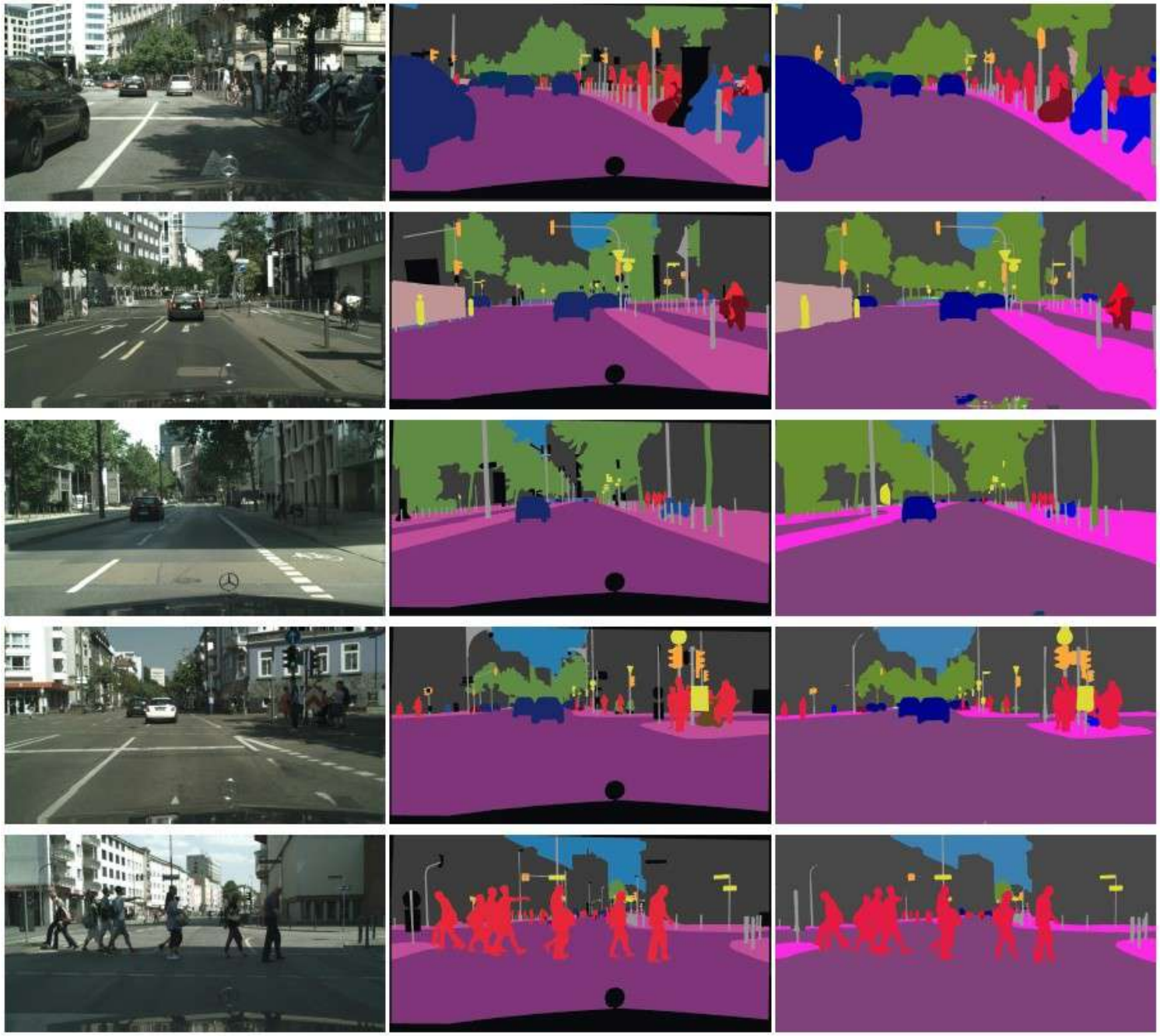
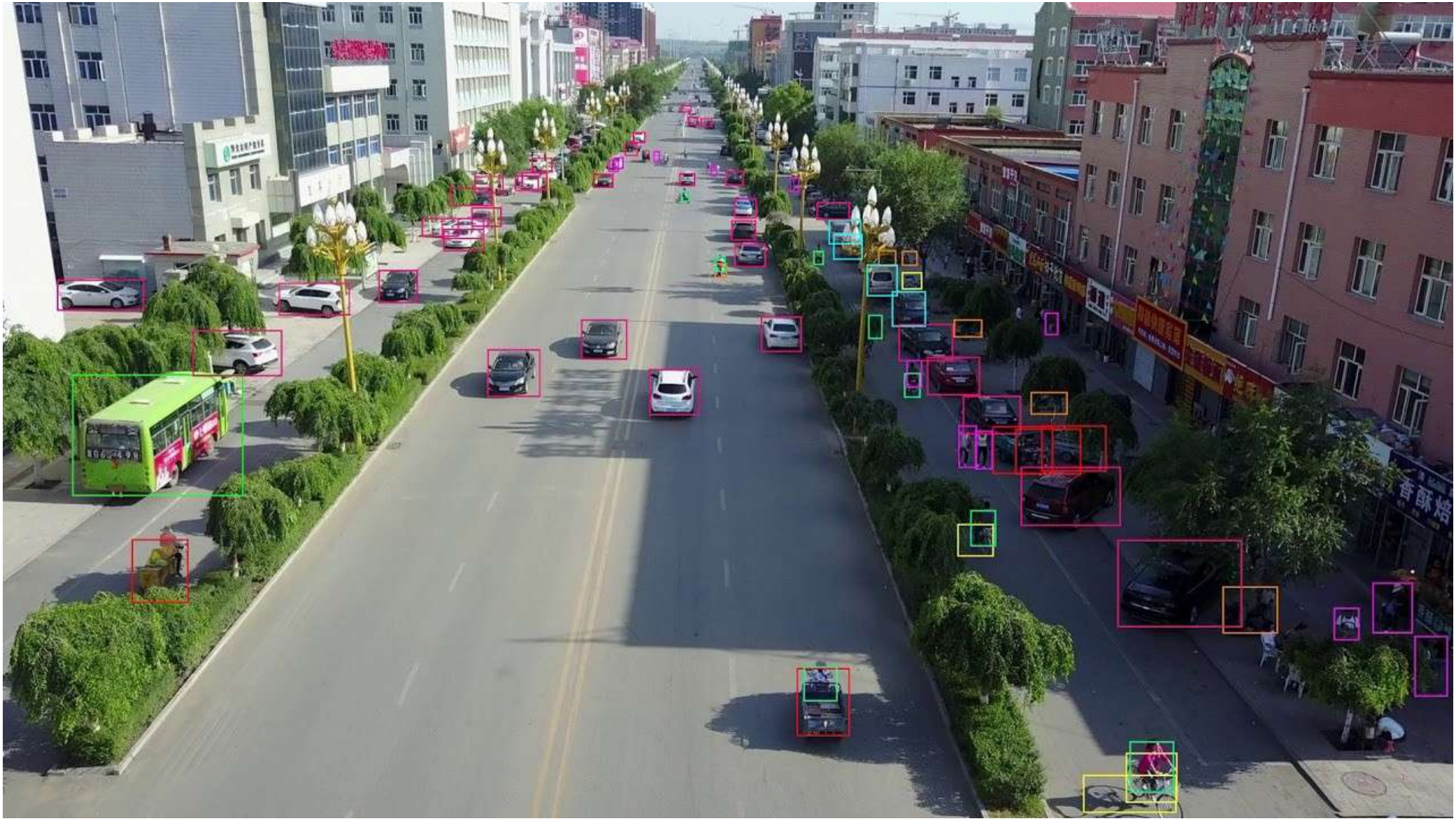
# Deep learning

Adrien CHAN-HON-TONG
ONERA

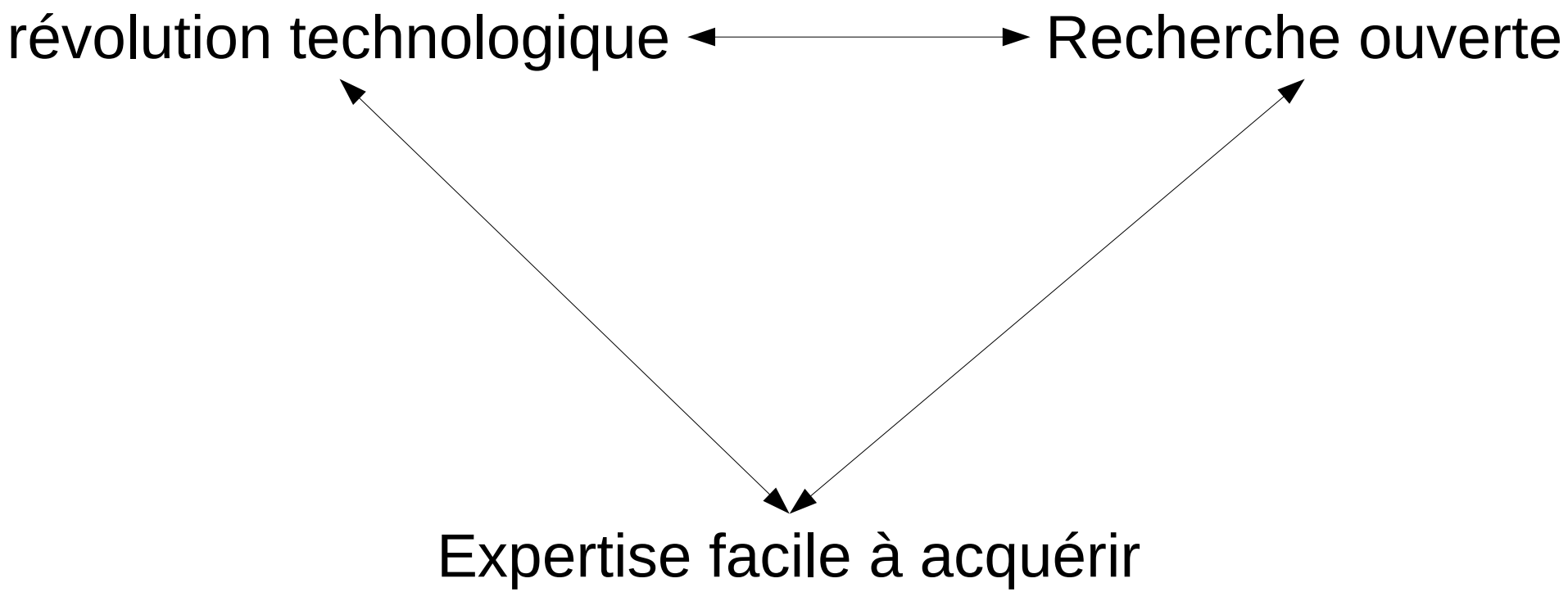
# Introduction



# Introduction



# Introduction

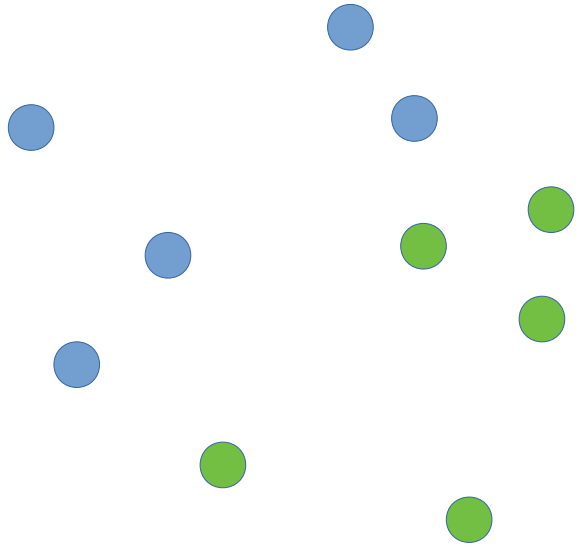


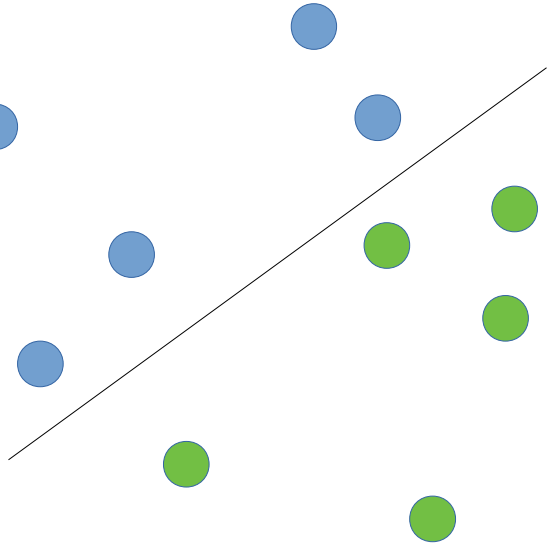
# Avant

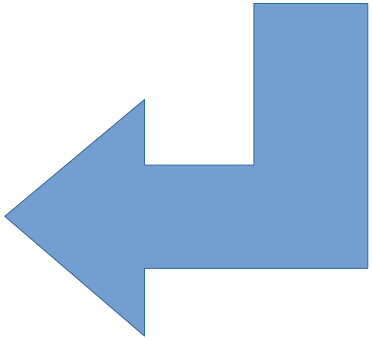
Données annotées et structurées
en dimension 100000000

Descriptions des données (features)
faites à la main
(hand crafted)

Nuage de points (D=10000)





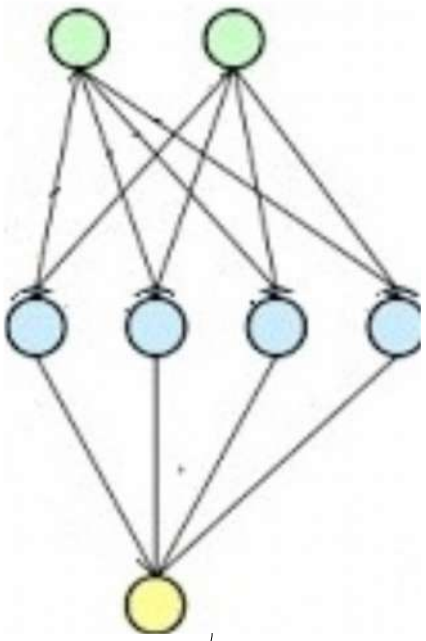


Classification de points

chat/chien

# Après

Données annotées et structurées
en dimension 100000000



Le réseau de neurones fait TOUT

chat/chien

# Important

Si vos données sont des points

→ Le deep learning aura peu d'avantage
par rapport à SVM, Decision forest, XGboost …

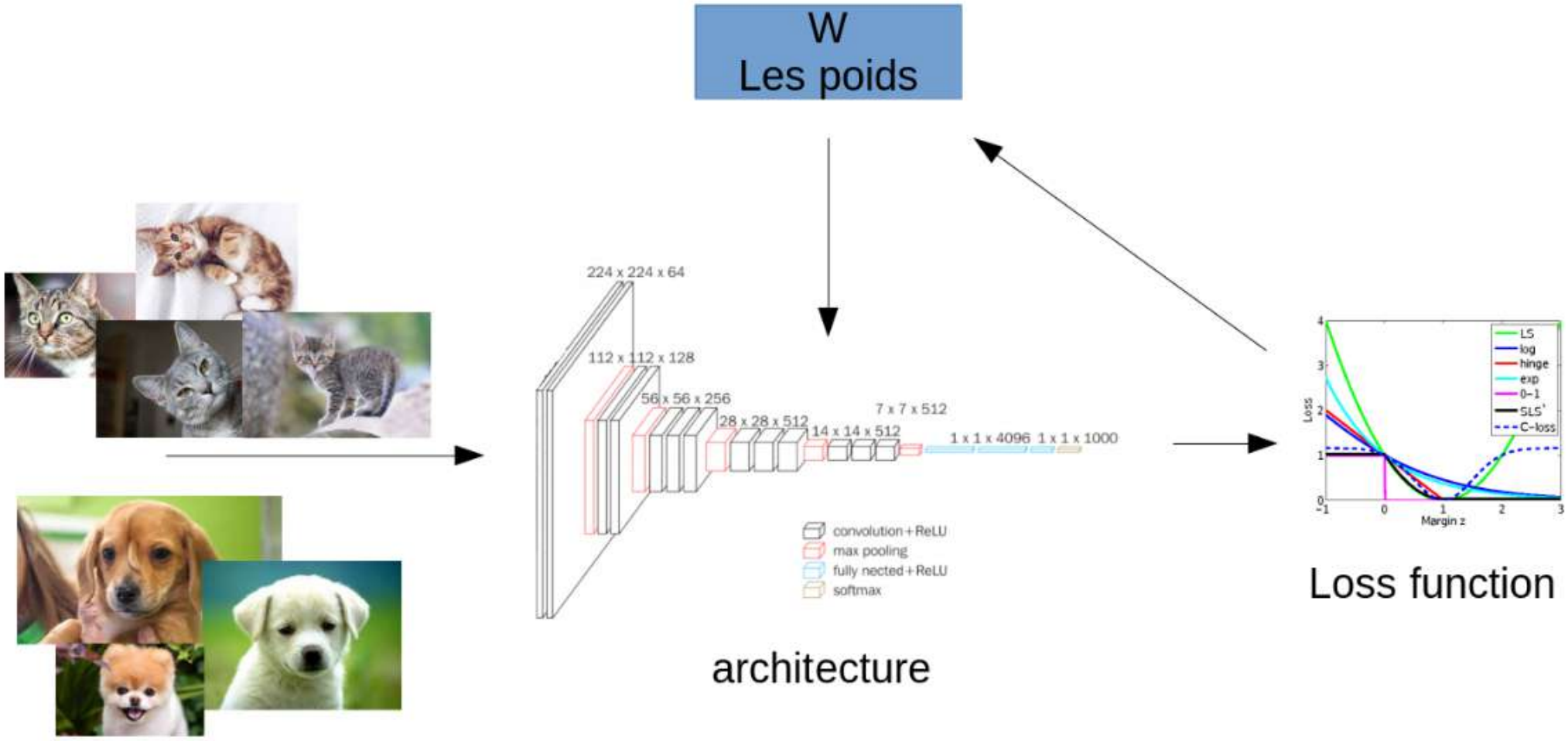
Si vos données sont structurées

→ Aujourd'hui on a aucun exemple de problème
où le deep learning n'est pas meilleur !

# Plan

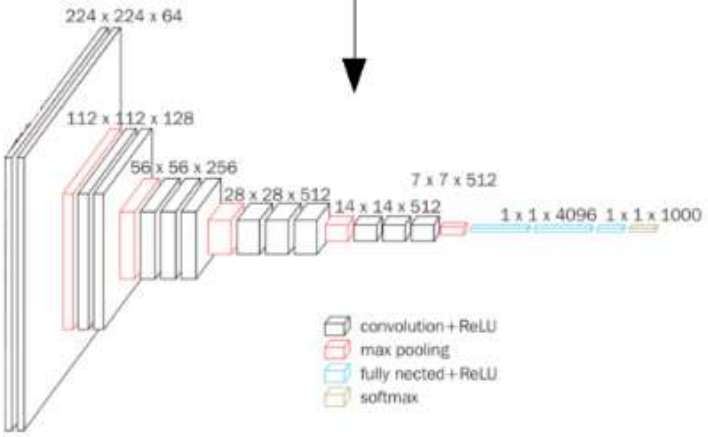
- Classification pure :
  - Apprentissage et test
  - Neurones et réseau
  - Universalité et erreur de généralisation
- Double descente :
  - Vapnik et régularisation
  - Méthode ensembliste et double descente
  - Descente de gradient stochastique
- Traitement de données structurées :
  - Le neurone convolutif
  - Les données structurées
  - La robustesse des réseaux

# 2 phases :
# apprentissage puis test



# 2 phases :
# apprentissage puis test

W
Les poids

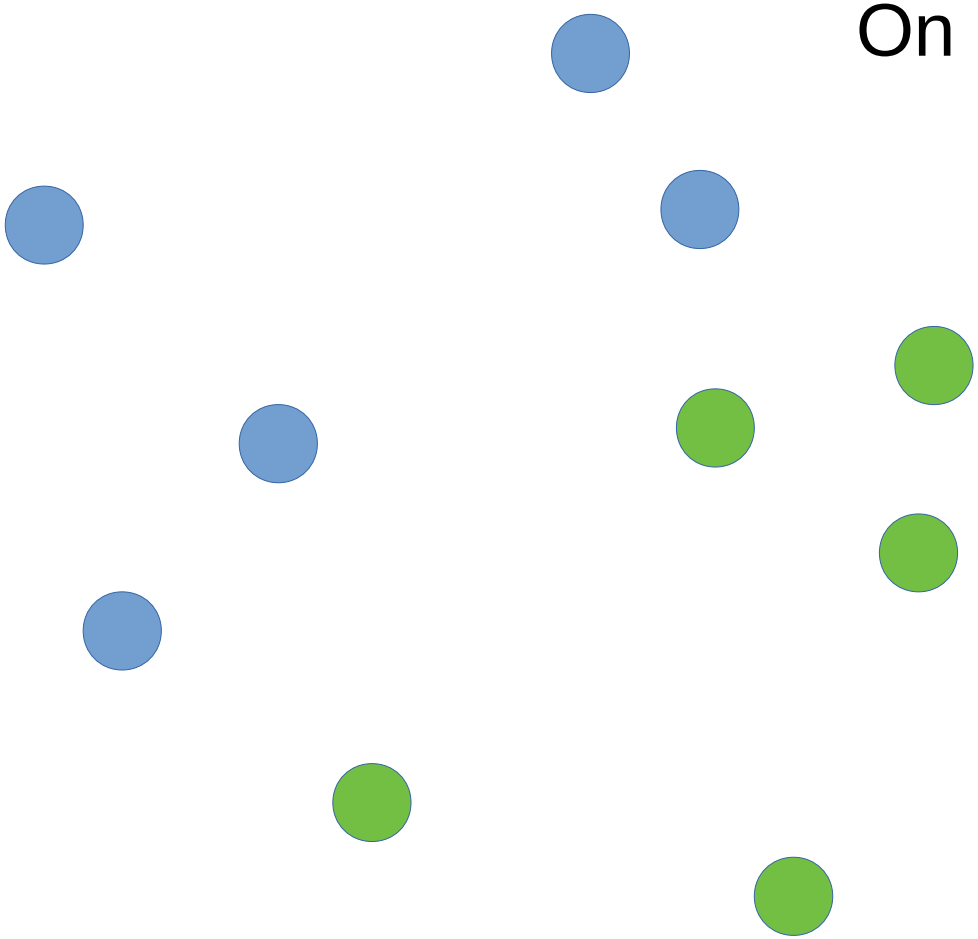



architecture

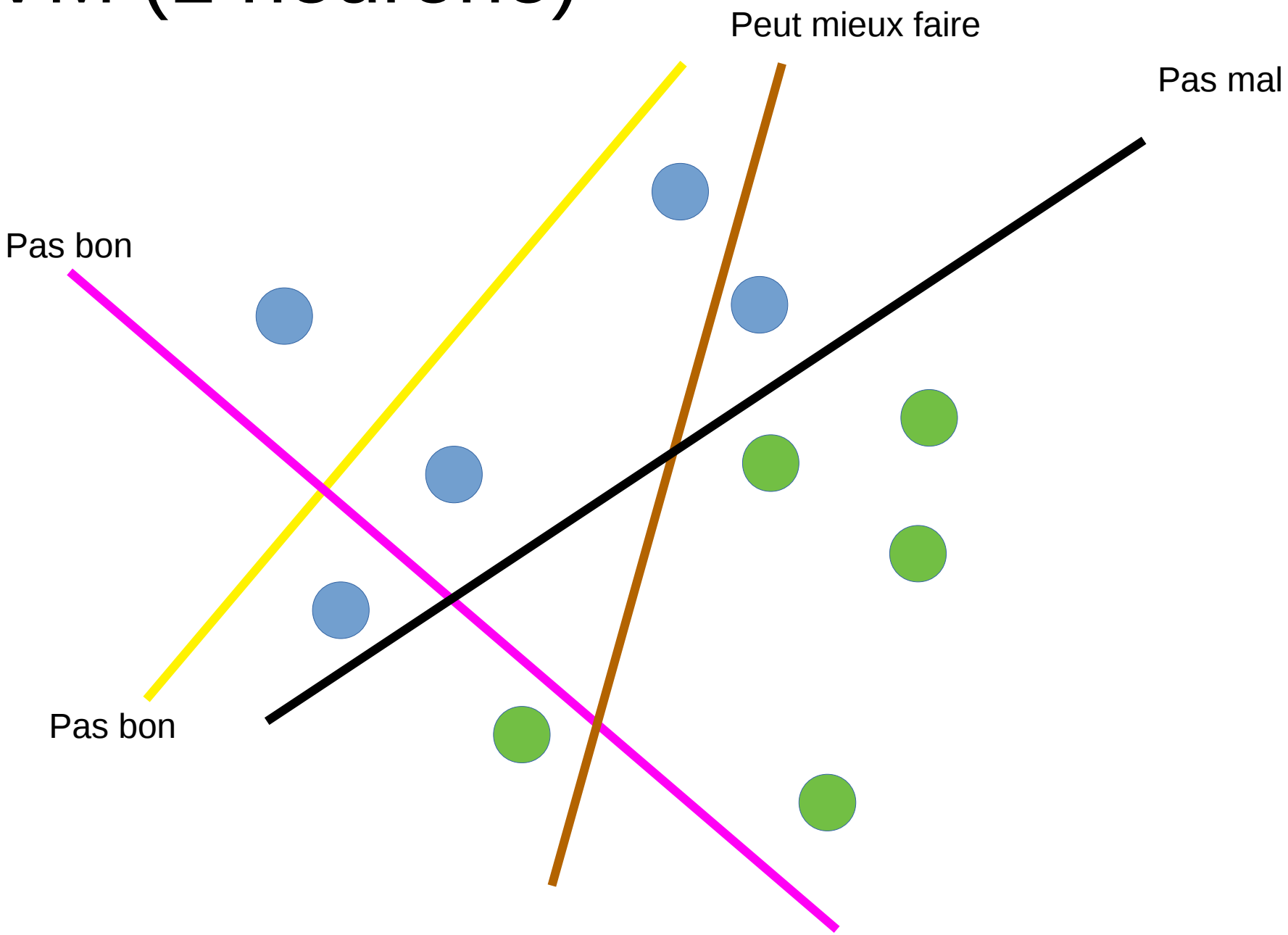
Décision
chat ou chien ?

# SVM (1 neurone)
# Apprentissage

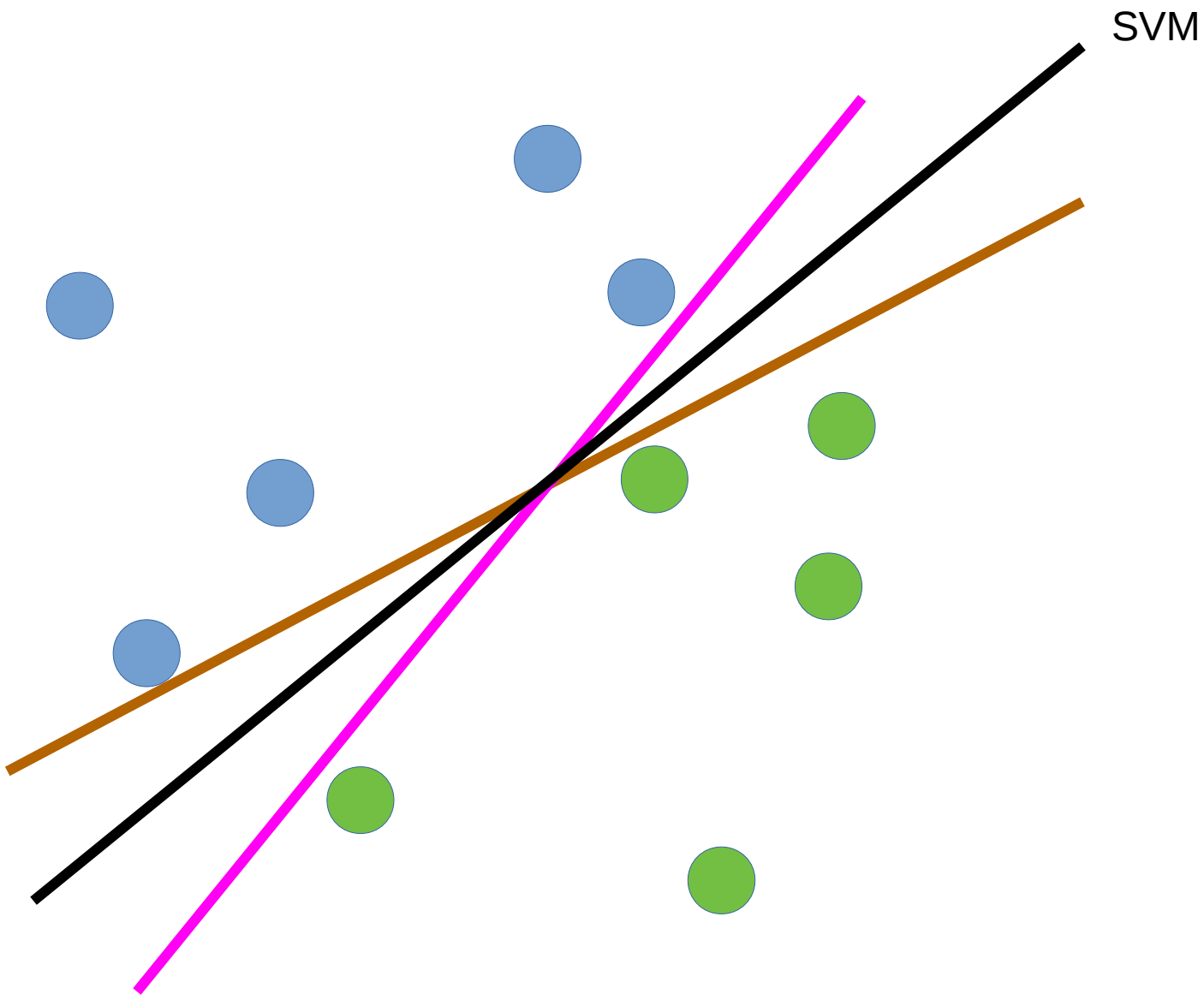
On cherche 1 droite qui sépare bleu et vert



# SVM (1 neurone)



# SVM (1 neurone)



# SVM (1 neurone)

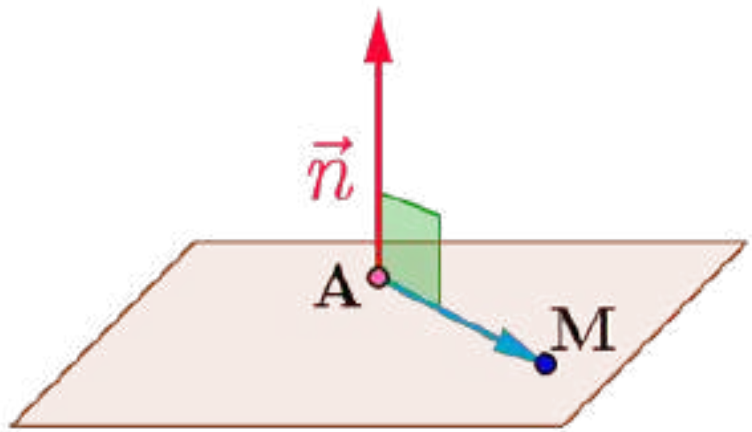
## Qu'est ce que ça donne avec des maths ?

On a des points $x_1, ..., x_N \in \mathbb{R}^D$ avec une couleur (bleu ou pas bleu) $(y_1, ..., y_N) \in \{-1, 1\}^N$.
Et on cherche un plan qui sépare les points.

# SVM (1 neurone)

## Rappel de math

On peut le paramétrer par un vecteur normal $w \in \mathbb{R}^D$ et un bias $b \in \mathbb{R}$. L'équation du plan est $w^T x + b = 0$.
Le plan sépare alors l'espace en 2 : $\{x \ / \ w^T x + b > 0\}$ et $\{x \ / \ w^T x + b < 0\}$.




ici $w = \vec{n}$, $M$ est dans le plan car $\vec{n}.\vec{AM} = 0$, si $\vec{n}.\vec{AM} > 0$ on est au dessus et sinon en dessous.

# SVM (1 neurone)

## Linear feasibility

On a des points $x_1, ..., x_N \in \mathbb{R}^D$ avec une couleur (bleu ou pas bleu) $(y_1, ..., y_N) \in \{-1, 1\}^N$.
Et on cherche un plan $w, b$ tel que $y_n = 1 \rightarrow w^T x_n + b > 0$ et $y_n = -1 \rightarrow w^T x_n + b < 0$, ce qu'on peut résumer en

$$\forall n \in \{1, ..., M\},\ y_n(w^T x_n + b) > 0$$

# SVM (1 neurone)

## SVM

On a des points $x_1, ..., x_N \in \mathbb{R}^D$ avec une couleur (bleu ou pas bleu) $(y_1, ..., y_N) \in \{-1, 1\}^N$.
Et le plan $w, b$ qui sépare les points en étant le plus distant possible

$$\max_{w,b,\delta} \delta$$

$$sc : \forall n \in \{1, ..., M\},\ y_n(w^T x_n + b) > \delta$$

$$sc : w^T w = 1$$

# SVM (1 neurone)

## SVM

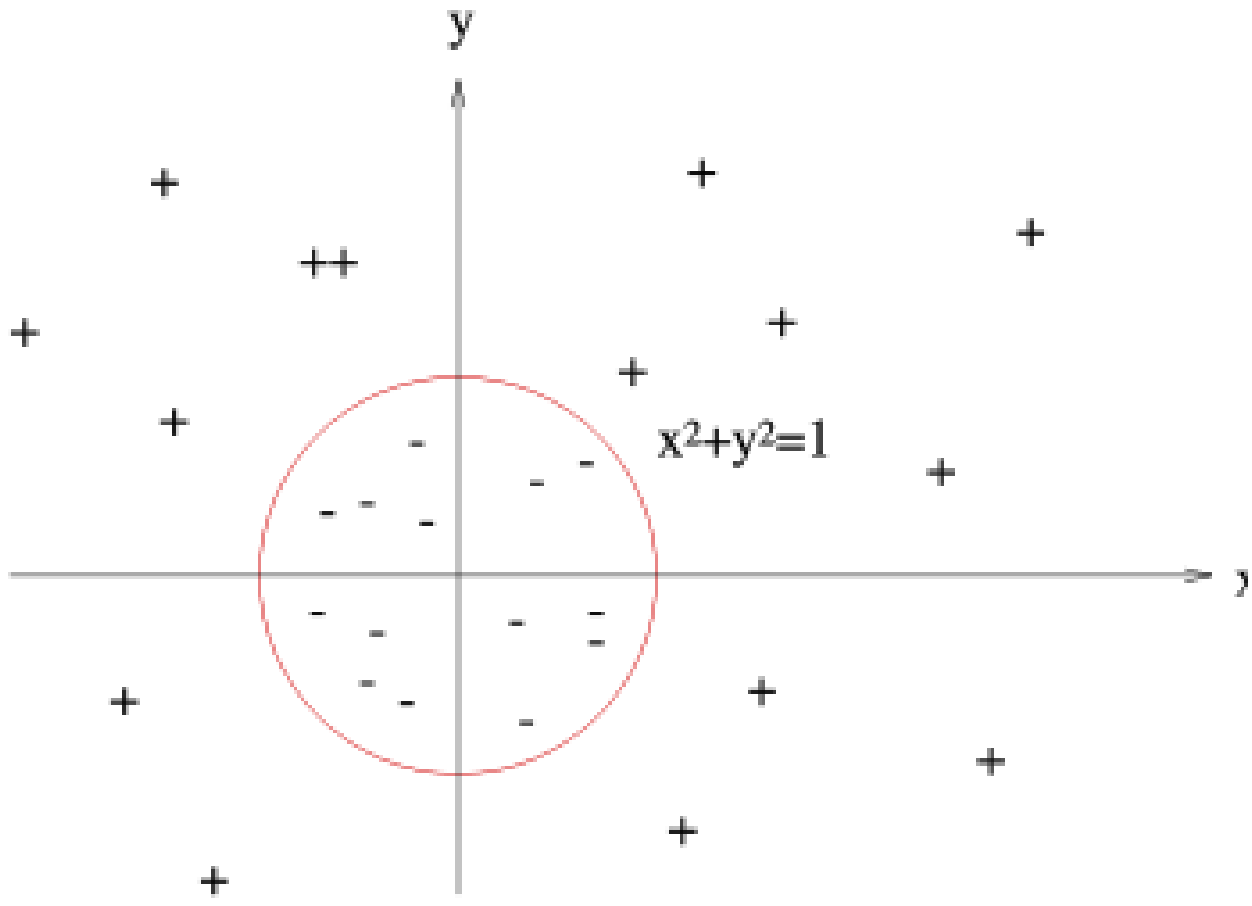
On a des points $x_1, ..., x_N \in \mathbb{R}^D$ avec une couleur (bleu ou pas bleu) $(y_1, ..., y_N) \in \{-1, 1\}^N$.
Et le plan $w, b$ qui sépare les points en étant le plus distant possible

$$\min_{w,b} w^T w$$

$$sc : \forall n \in \{1, ..., M\},\ y_n(w^T x_n + b) \geq 1$$

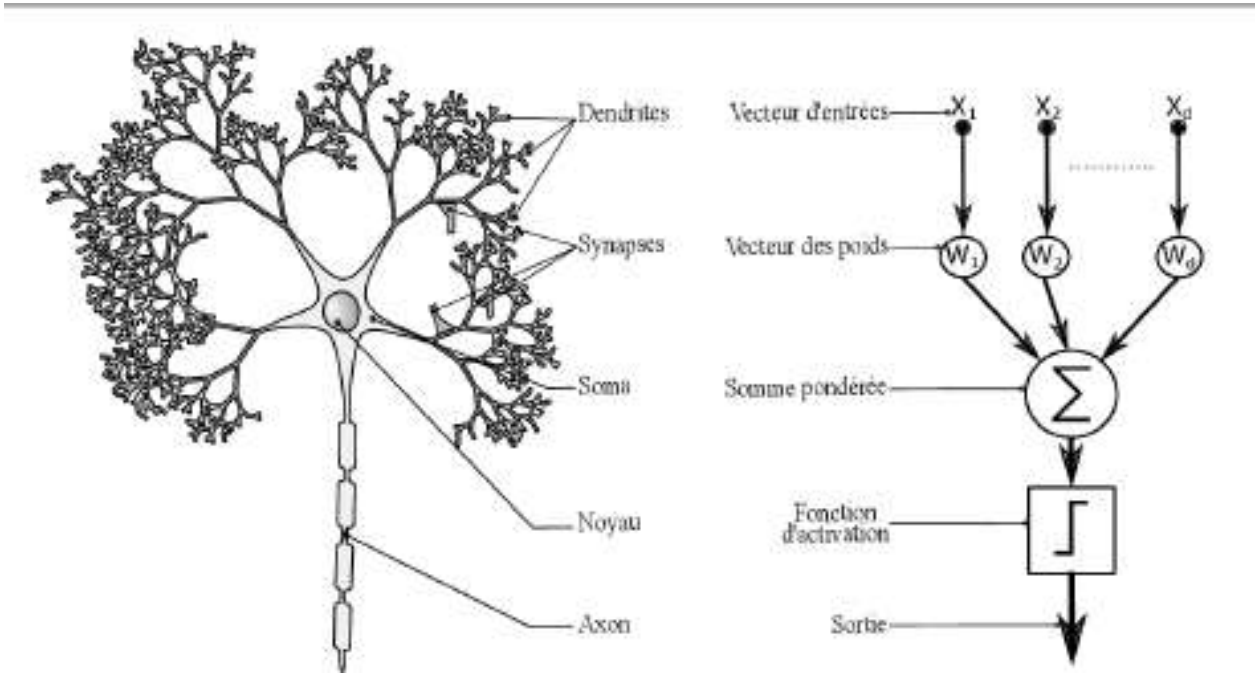
# Problème avec le SVM (1 neurone)

## Parfois on ne peut PAS apprendre les données



# Pause culturelle

## Pourquoi 1 plan = 1 neurone ?



# Du neurone au réseau

## Le neurone

$||x - q||_1 = |x_1 - q_1| + |x_2 - q_2| =$
$relu(x_1 - q_1) + relu(-x_1 + q_1) + relu(x_2 - q_2) + relu(-x_2 + q_2)$
Dans les architectures de réseaux de neurones (profond ou pas), le neurone est un filtre linéaire :

$$neurone_{\alpha,\beta} : \begin{array}{ccc} \mathbb{R}^{\phi} & \rightarrow & \mathbb{R} \\ u & \rightarrow & \alpha.u + \beta \end{array}$$

$\alpha \in \mathbb{R}^{\phi}$ et $\beta \in \mathbb{R}$ sont les **poids** du neurones.

Attention, maintenant, le neurone n'est plus nécessairement connecté à $x$ l'entrée. Il a une entré de taille $\phi$ indépendante de $D$.

# Du neurone au réseau

## La couche de neurone

Une couche de $\psi$ de neurones est une séquence de $\psi$ neurones prenant la même entrée, et, dont les $\psi$ sorties sont regroupées en 1 vecteur :

$$\begin{array}{rccc} & \mathbb{R}^{\phi} & \rightarrow & \mathbb{R}^{\psi} \\ couche_{A,b} : & u & \rightarrow & \begin{pmatrix} neurone_{A_1,b_1}(u) \\ \dots \\ neurone_{A_\psi,b_\psi}(u) \end{pmatrix} \end{array}$$

$A \in \mathbb{R}^{\psi \times \phi}$ et $b \in \mathbb{R}^{\psi}$ sont les **poids** de chacun des $\psi$ neurones.

La couche de neurone est aussi linéaire : $couche_{A,b}(u) = Au + b$ mais avec des tailles arbitraires en entrée et sorti.

# Du neurone au réseau

## Le réseau de neurone

Si on empile 2 couches de neurones c'est exactement comme s'il y en avait qu'une :

$$A'(Au + b) + b' = (A'A)u + (A'b + b')$$

Oui mais si on met une non linéarité entre les 2 c'est différents.

$$A'\mathrm{activation}(Au + b) + b' \neq \mathrm{activation}((A'A)u + (A'b + b'))$$

# Du neurone au réseau

## Le réseau de neurone

On empile 2 couches de neurones AVEC activation :

$$A' \mathrm{activation}(Au + b) + b'$$

Sur pytorch : il y en a un certain nombre relu, elu, leaky-relu, sigmoide, arctan, hard sigmoid, hard arctan, prelu, relu6, rrelu, celu, selu, gelu, hard shirk, soft shirk, log sigmoid, soft sign, tanh, tanhshirk

globalement avant on utilisait une sigmoide car le gradient existe partout. De 2012 à aujourd'hui, c'est plutôt relu
$relu(u) = [u]_+ = max(u, 0)$ car ça laisse passer le gradient tout en étant simple. Des architectures robustes commencent à utiliser SortUp (voir après).

# Du neurone au réseau

## Le réseau de neurone

Un réseau de neurones entièrement connectées **MLP** (multi layer perceptron en anglais) de profondeur $Q$ est un empilement de $Q$ couche de neurones - séparé par des activations - la dernière est classiquement un seul neurone :

$$\begin{array}{rccc} reseau_w : & \mathbb{R}^D & \rightarrow & \mathbb{R} \\ & x & \rightarrow & C_{w_Q}(relu(C_{w_{Q-1}}(...relu(C_{w_1}(x))...))) \end{array}$$

c'est à dire

$$reseau_w(x) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$$

$w_1, ..., w_Q$, $Q$ matrices dont la seule chose imposée étant que $w_1$ ait $D$ colonnes, et $w_Q$ 1 ligne (et que les tailles soit cohérentes entre elles)

# Bilan

## SVM vs DL

Dans le cas du SVM, on a $f(x, w) = w^T x + w_{biais}$ qui donne un signe

$$f(x, w) > 0 \text{ ou } f(x, w) < 0$$

pour dire de quel coté on est.
Dans un réseau de neurone c'est PAREIL sauf que

$$f(x, w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$$

# Universalité des réseaux Relu

## Valeur absolue D=1

$|x| = relu(x) + relu(-x)$

# Universalité des réseaux Relu

## Valeur absolue D=2

$||x||_1 = |x_1| + |x_2| = relu(x_1) + relu(-x_1) + relu(x_2) + relu(-x_2)$
marche en fait quelque soit la dimension $D$ (besoin de $2 \times D + 1$ neurones).

# Universalité des réseaux Relu

## Valeur absolue D=2 avec biais

$||x - q||_1 = |x_1 - q_1| + |x_2 - q_2| =$
$relu(x_1 - q_1) + relu(-x_1 + q_1) + relu(x_2 - q_2) + relu(-x_2 + q_2)$

## Le pseudo-dirac

$g_q(x) = relu(1 - ||x - q||_1)$ verifie

$$g_q : \begin{cases} g_q(q) = 1 \\ \forall x \ / \ ||x - q||_1 > 1, \ g_q(x) = 0 \end{cases}$$

# Universalité des réseaux Relu

$\forall x_1, ..., x_N \in \mathbb{Z}^D,\ \forall (y_1, ..., y_N) \in \{-1, 1\}^N,$
il suffit de $2 \times N \times D + N + 1$ neurones pour apprendre par coeur la base de données avec $f(x, w)$

$$= \sum_n y_n \times \left( relu \left( 1 - \sum_d \left( relu(x_d - x_{n,d}) + relu(-x_d + x_{n,d}) \right) \right) \right)$$

# Bilan

L’apprentissage c’est des points colorées dans un espace qu’on cherche à séparer

1 plan = SVM

Des couches linéaires séparés par des activations = réseau de neurones

→ Le SVM ne peut apprendre que des distributions linéairement séparables

→ Avec 2ND neurones, on peut apprendre n’importe qu’elle base d’apprentissage

# Bilan

L’apprentissage c’est des points colorées dans un espace qu’on cherche à séparer

1 plan = SVM

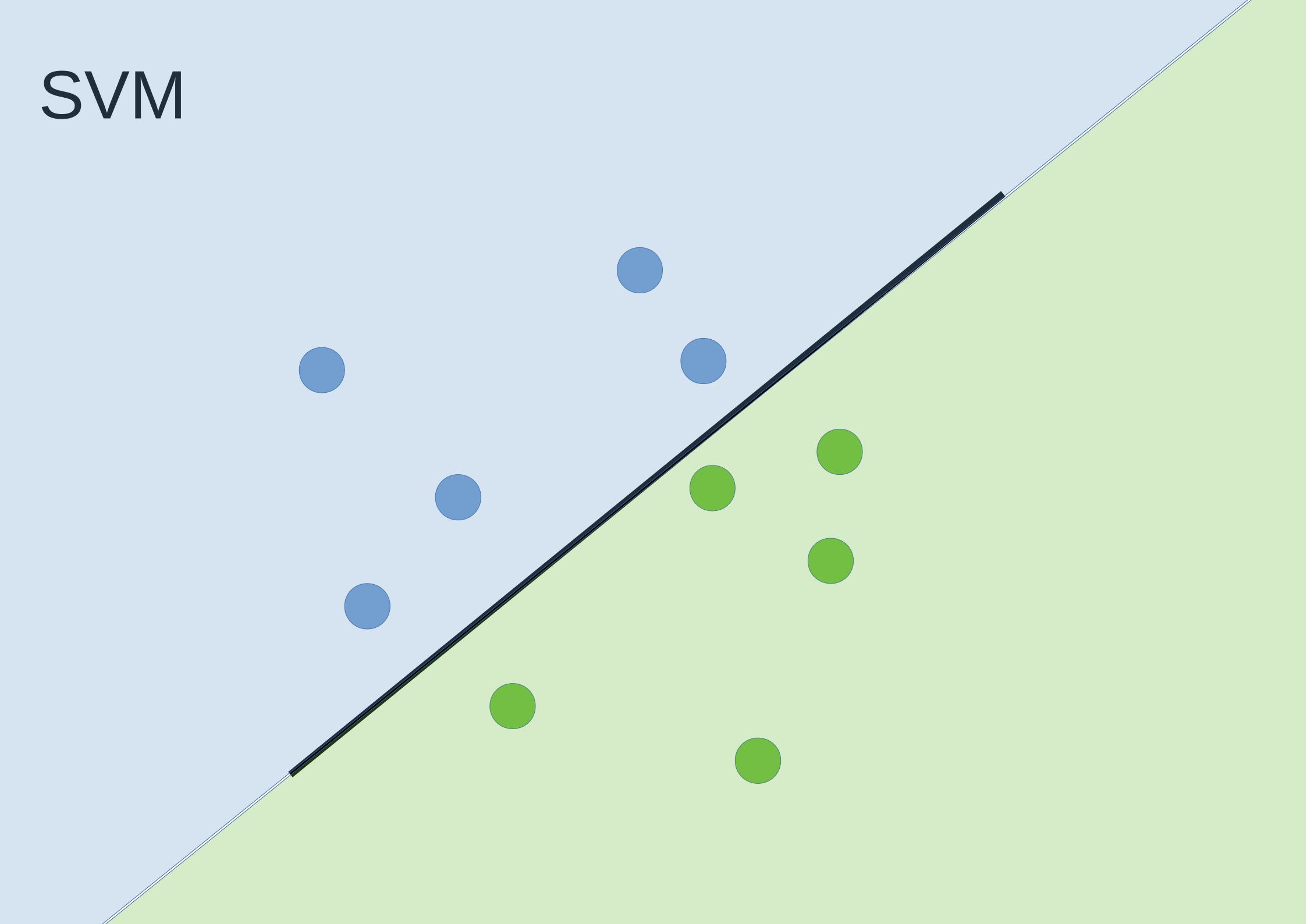
Des couches linéaires séparés par des activations = réseau de neurones

→ Le SVM ne peut apprendre que des distributions linéairement séparables

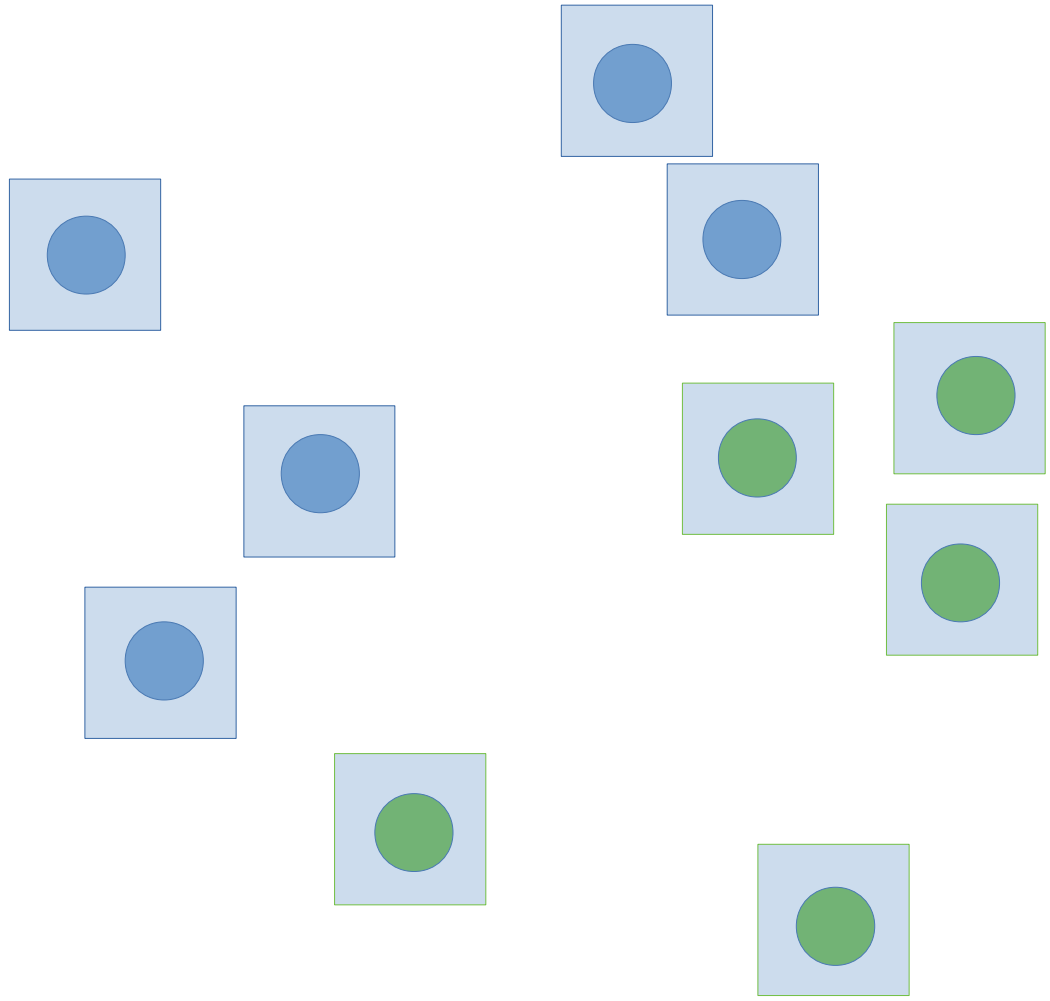
→ Avec 2ND neurones, on peut apprendre n’importe qu’elle base d’apprentissage

→ **ATTENTION : apprendre par cœur n’est PAS forcément pertinent**

# SVM

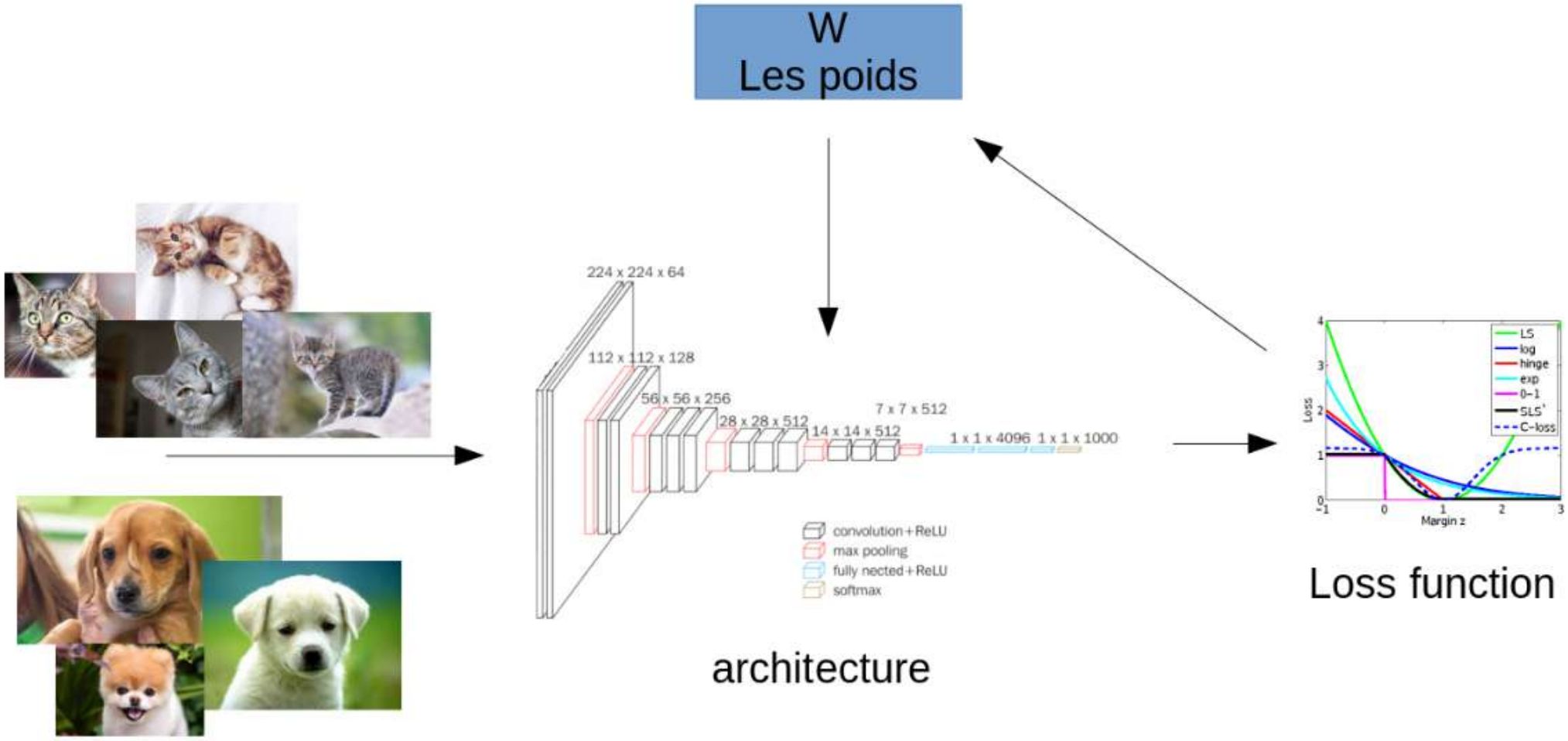


# Théorème d’universalité



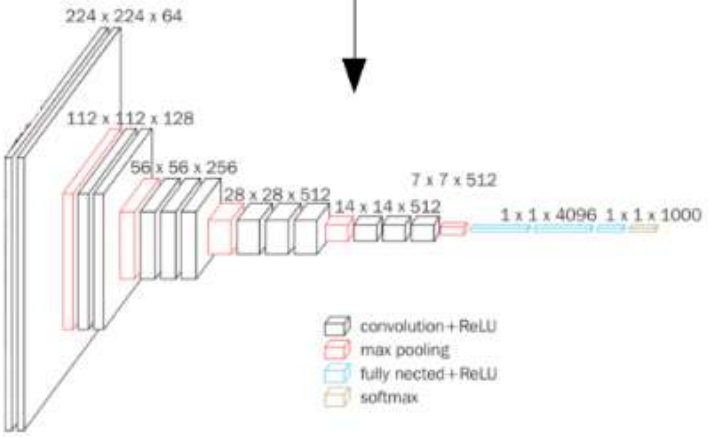
AUCUNE information loin des points de la base d’apprentissage !!!

# 2 phases :
# apprentissage puis test



# 2 phases :
# apprentissage puis test

W
Les poids




architecture

Décision
chat ou chien ?

# Ce qui compte c’est la performance en test !

# Ce qui compte c’est la performance en test !

L’apprentissage est un problème non formalisable/mathématique

# Ce qui compte c’est la performance en test !

L’apprentissage est un problème non formalisable/mathématique

On ne peut pas prouver la valeur de la constante de gravitation.
On ne peut que la mesurer

# Ce qui compte c’est la performance en test !

L’apprentissage est un problème non formalisable/mathématique

On ne peut pas prouver la valeur de la constante de gravitation.
On ne peut que la mesurer

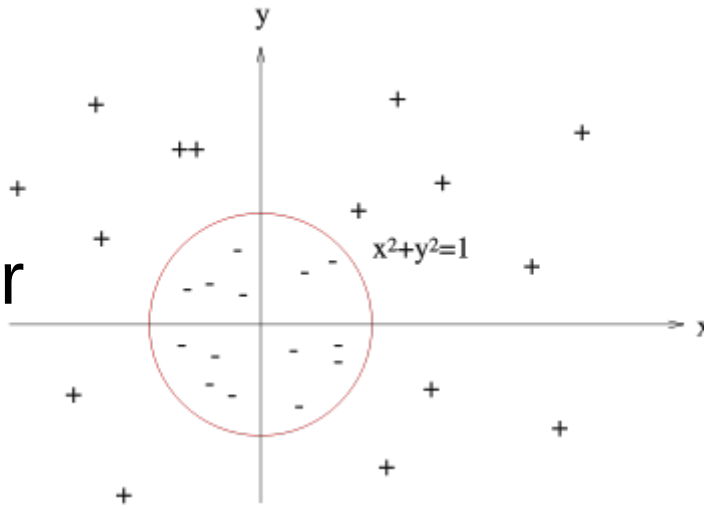
**En pratique, le DeepLearning ça marche**
**mais on ne pourra jamais le prouver.**

# Plan

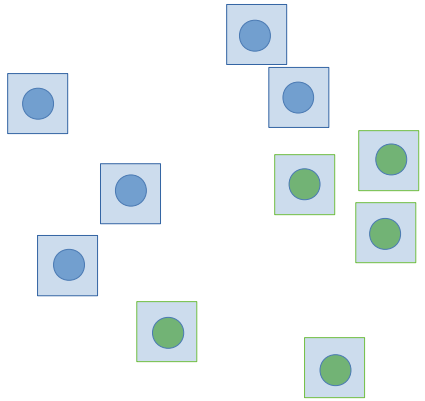
- ~~Classification pure :~~
  - ~~Apprentissage et test~~
  - ~~Neurones et réseau~~
  - ~~Universalité et erreur de généralisation~~
- Double descente :
  - Vapnik et régularisation
  - Méthode ensembliste et double descente
  - Descente de gradient stochastique
- Traitement de données structurées :
  - Le neurone convolutif
  - Les données structurées
  - La robustesse des réseaux

# L’ancien paradigme

Trop petit → tu peux pas apprendre : SVM sur

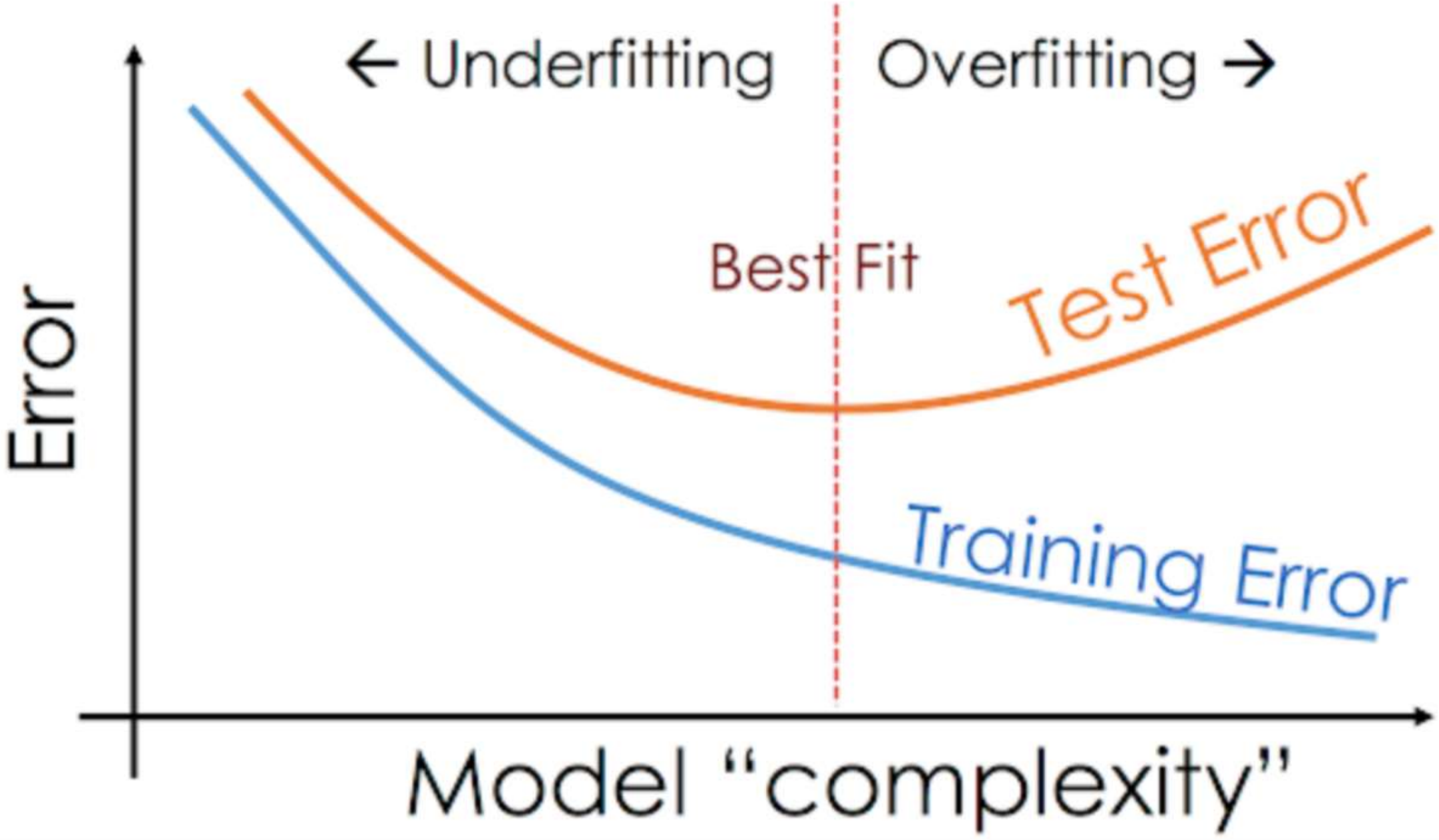



Trop gros → tu apprends par coeur

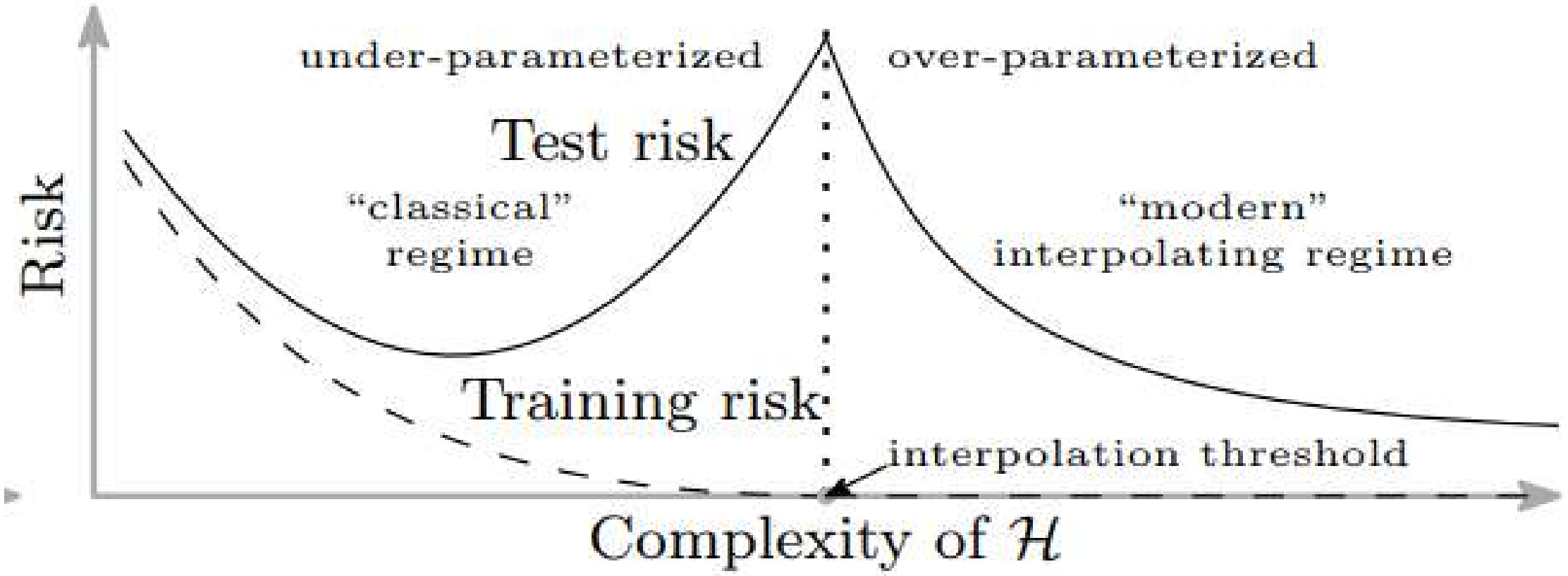


→ faut être entre les deux (théorie Vapnik)

# L’ancien paradigme

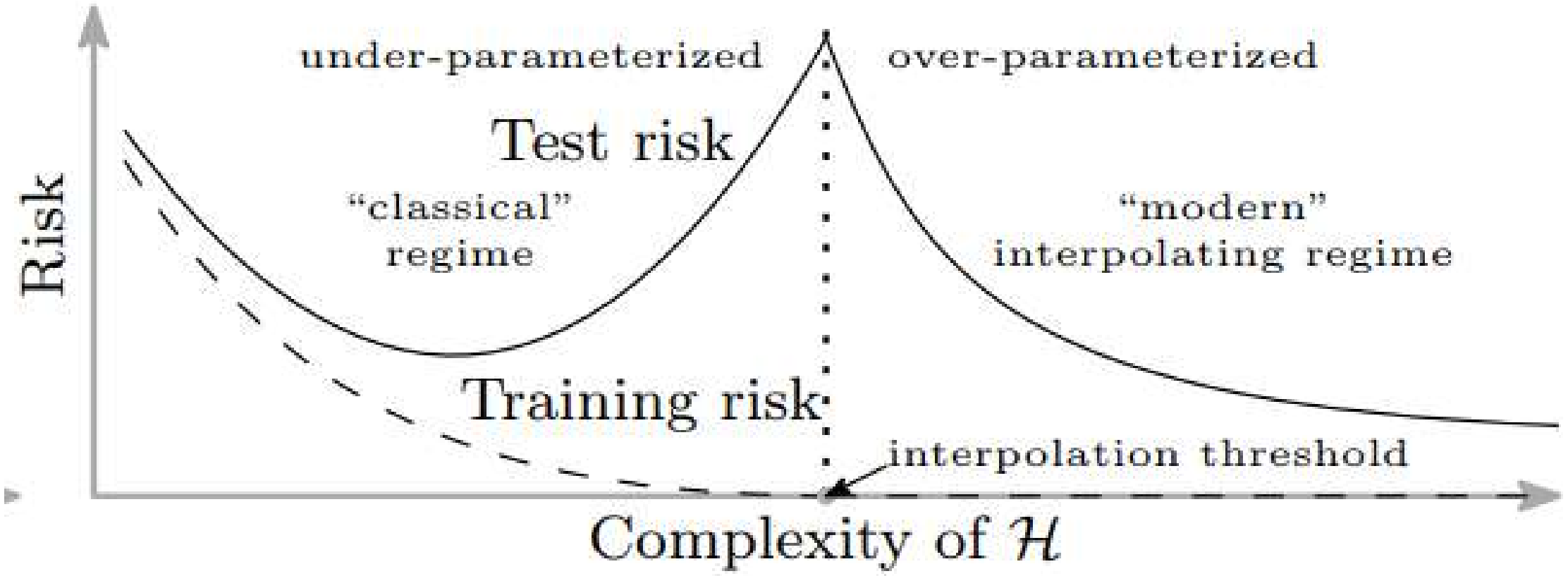


# Le nouveau paradigme




https://www.lesswrong.com/posts/FRv7ryoqtvSuqBxuT/understanding-deep-double-descent

# Le nouveau paradigme




*Prends le réseau de neurones le plus gros que tu peux*
*et t'aura les meilleurs résultats possibles*
*(pas nécessairement parfait mais pas trop mal normalement)*

# Le nouveau paradigme

*Prends le réseau de neurones le plus gros que tu peux*
*et t'aura les meilleurs résultats possibles*
*(pas nécessairement parfait mais pas trop mal normalement)*

→ C'est triste mais c'est que c'est ce qu'on observe !

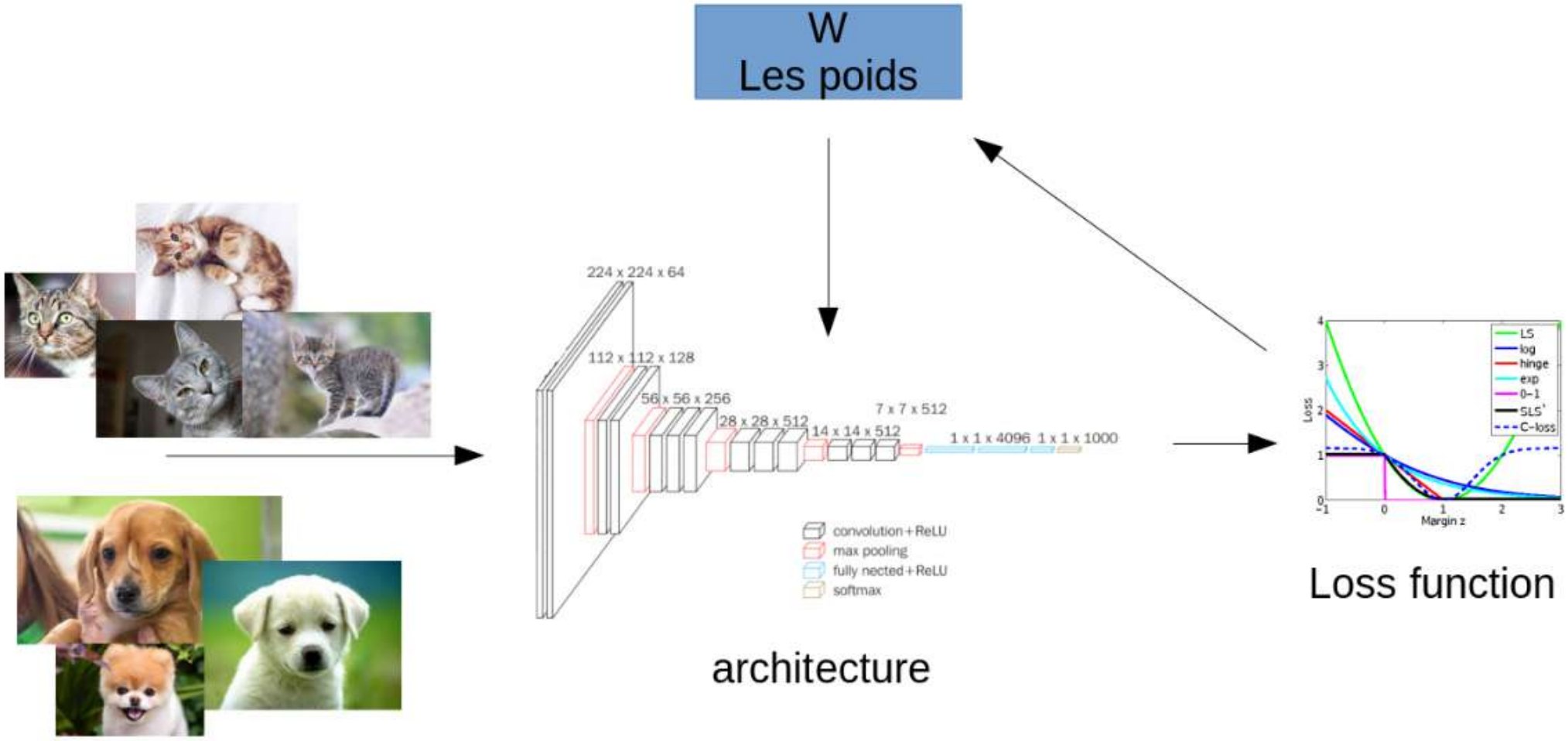
Il n'y a PAS de prime à la finesse ou à l'intelligence

# Le nouveau paradigme

D’où vient ce nouveau paradigme ?

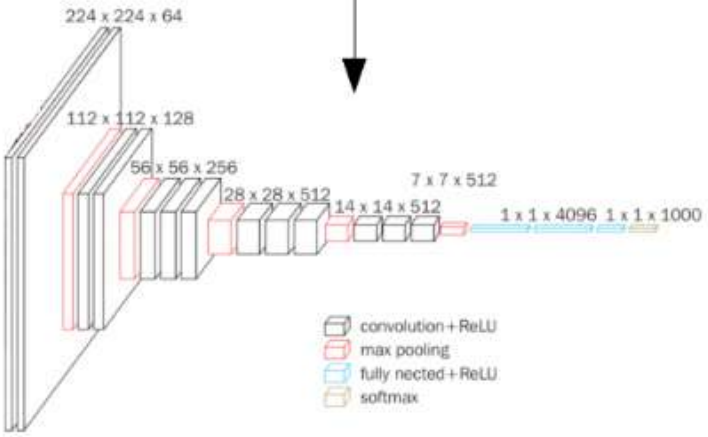
→ les réseaux de neurones agissent nativement comme des méthodes ensemblistes

# 2 phases :
# apprentissage puis test



# 2 phases :
# apprentissage puis test

W
Les poids




architecture

Décision
chat ou chien ?

L’apprentissage : on prend x1, …, xN, y1, …, yN et ça produit w

L’apprentissage : on prend x1, …, xN, y1, …, yN et ça produit w

Et si on le fait K fois, on aura w1, …, wK

L’apprentissage : on prend x1, …, xN, y1, …, yN et ça produit w

Et si on le fait K fois, on aura w1, …, wK

Et si on moyenne les K modèles, on tendra vers w*

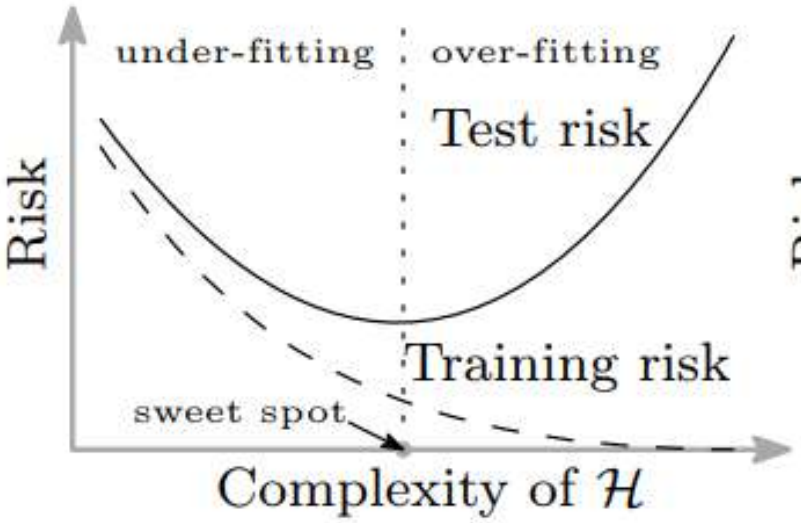
L’apprentissage : on prend x1, …, xN, y1, …, yN et ça produit w

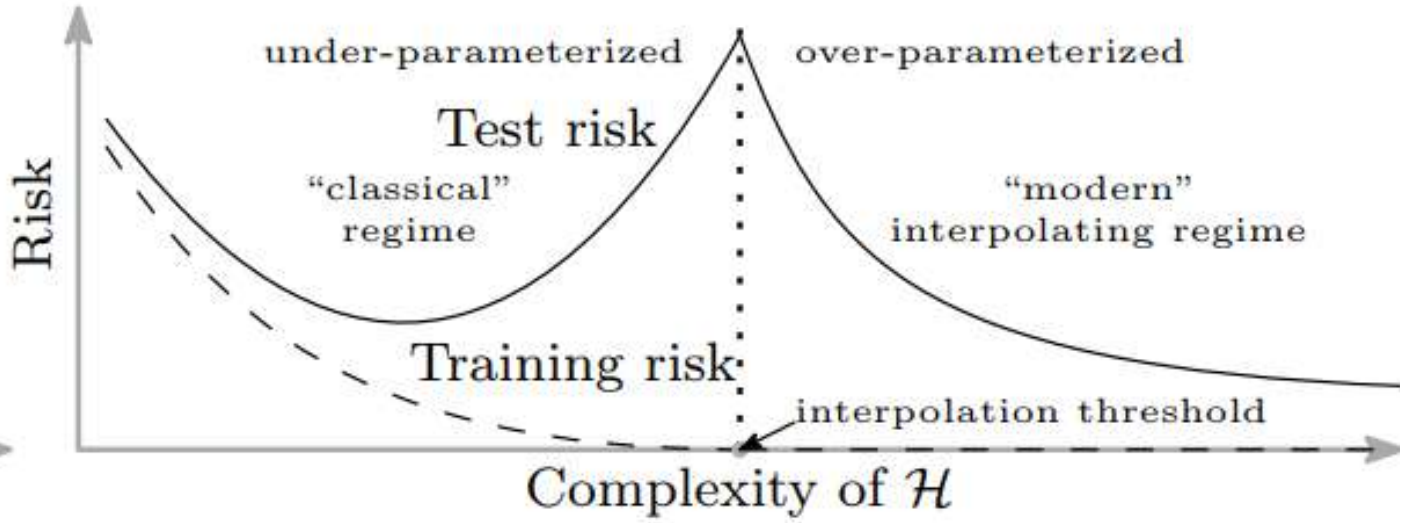
Et si on le fait K fois, on aura w1, …, wK

Et si on moyenne les K modèles, on tendra vers w*

**ATTENTION** : w* n’est pas parfait, il s’agit juste du modèle moyen quand on enlève la partie aléatoire de l’apprentissage

d’ailleurs si pas d’aléatoire w* = w1 = … = wK par exemple pour les SVM !




Si la complexité se mesure en nombre de modèle
Les 2 courbes convergent vers la performance du modèle moyen

**L’ancien paradigme :**

Si on considère K réseaux de Q neurones

→ Q crée de l’overfitting
→ mais à Q fixé, K n’en crée pas

cependant, il ne peut pas apprendre de base nécessitant + de Q neurones

**Le nouveau paradigme :**

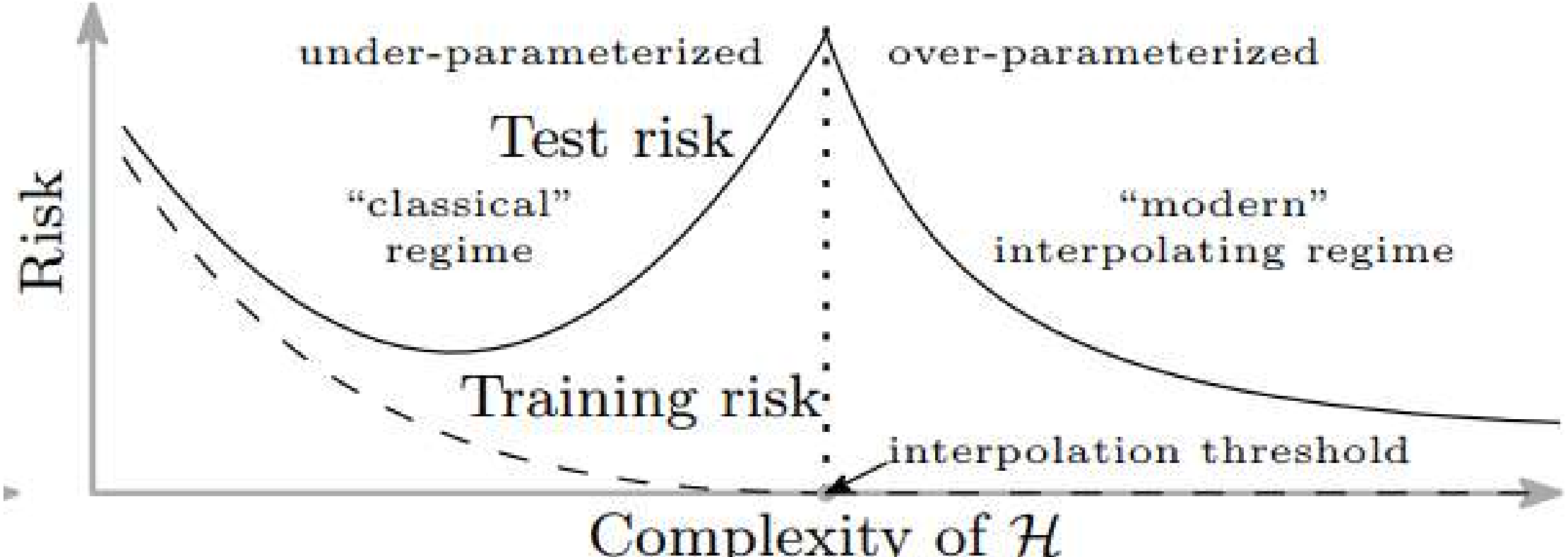
Si on considère 1 réseaux de P neurones

→ P va naturellement se décomposer en K et Q
→ *de sorte que Q soit suffisant*

Personnellement,
mes expériences me laissent penser
qu'il y a quand même de l'overfitting
quand on utilise un nombre « stupidement » grand de neurones

mais sur une très grande plage de valeurs

« plus c'est mieux »



Pourquoi les réseaux de neurones
aurait-il cette particularité d'ajuster leur complexité
au problème traiter ??

→ parce que leur apprentissage est
fondamentalement stochastique
et régularisant !

# La descente de gradient

$F$ est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$ alors
$\forall u, h \in \mathbb{R}^D$, $F(u+h) = F(u) + \nabla F_u | h + o(h)$
avec $ho(h) \underset{h \to 0}{\to} 0$ (notation petit $o$ classique)
Donc si $\nabla F_u \neq 0$ alors il existe $\lambda > 0$ tel que $F\left(u - \lambda \nabla F_u\right) < F(u)$

# La descente de gradient

## pseudo code

input : $F$, $u_0$

1. $u = u_0$
2. calculer $\nabla F_u$
3. si $\nabla F_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $F\left(u - \lambda \nabla F_u\right) \geq F(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla F_u$
7. go to 2

# La descente de gradient

## pseudo code

input : $F$, $u_0$

1. $u = u_0$
2. calculer $\nabla F_u$
3. si $\nabla F_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $F\left(u - \lambda \nabla F_u\right) \geq F(u)$ faire $\lambda = 0.5\lambda$
6. $u = u - \lambda \nabla F_u$
7. go to 2

cet algorithme converge vers un point $u^*$ tel que $\nabla F_u = 0$

# Apprentissage et descente de gradient

Appliquer à l'apprentissage :

- la variable $u$ de la descente de gradient est les poids $w$ du réseau
- la fonctionnelle $(F)$ est (+/-) l'erreur d'apprentissage :
  $F(w) \approx \sum_n \mathbf{1}_-(y(x_n)f(x_n, w))$
  avec $\mathbf{1}_-(t) = 1$ si $t \leq 0$ et $\mathbf{1}_-(t) = 0$ si $t > 0$

# Apprentissage et descente de gradient

## Test :

$w$ fixé, on prend $\chi$, et, on doit calculer $f(\chi, w)$

## Apprentissage :

On prend $x_1, ..., x_N$, et, on doit **approximer**

$$\min_w \sum_n \mathbf{1}_-(y(x_n) f(x_n, w))$$

# Fonction de perte

La descente de gradient ne marche qu'avec des fonctions globalement lisse.
Utiliser $F(w) = \min_w \sum_n \mathbf{1}_-(y(x_n)f(x_n, w))$ ne peut pas marcher

# Fonction de perte

La descente de gradient ne marche qu'avec des fonctions globalement lisse.
Utiliser $F(w) = \min_w \sum_n \mathbf{1}_-(y(x_n)f(x_n, w))$ ne peut pas marcher
Il faut lisser l'erreur d'apprentissage via une loss function

$$F(w) = loss\,(w) = \sum_n l(y(x_n)f(x_n, w))$$

# Fonction de perte

$$F(w) = loss\,(w) = \sum_n l(y(x_n)f(x_n, w))$$

- $l$ doit être assez lisse
- $l$ doit avoir une valeur proche de 0 si $y(x_n)f(x_n, w)$ est grand
- $l$ doit avoir une valeur très supérieure à 0 si $y(x_n)f(x_n, w)$ est très petit

# Fonction de perte

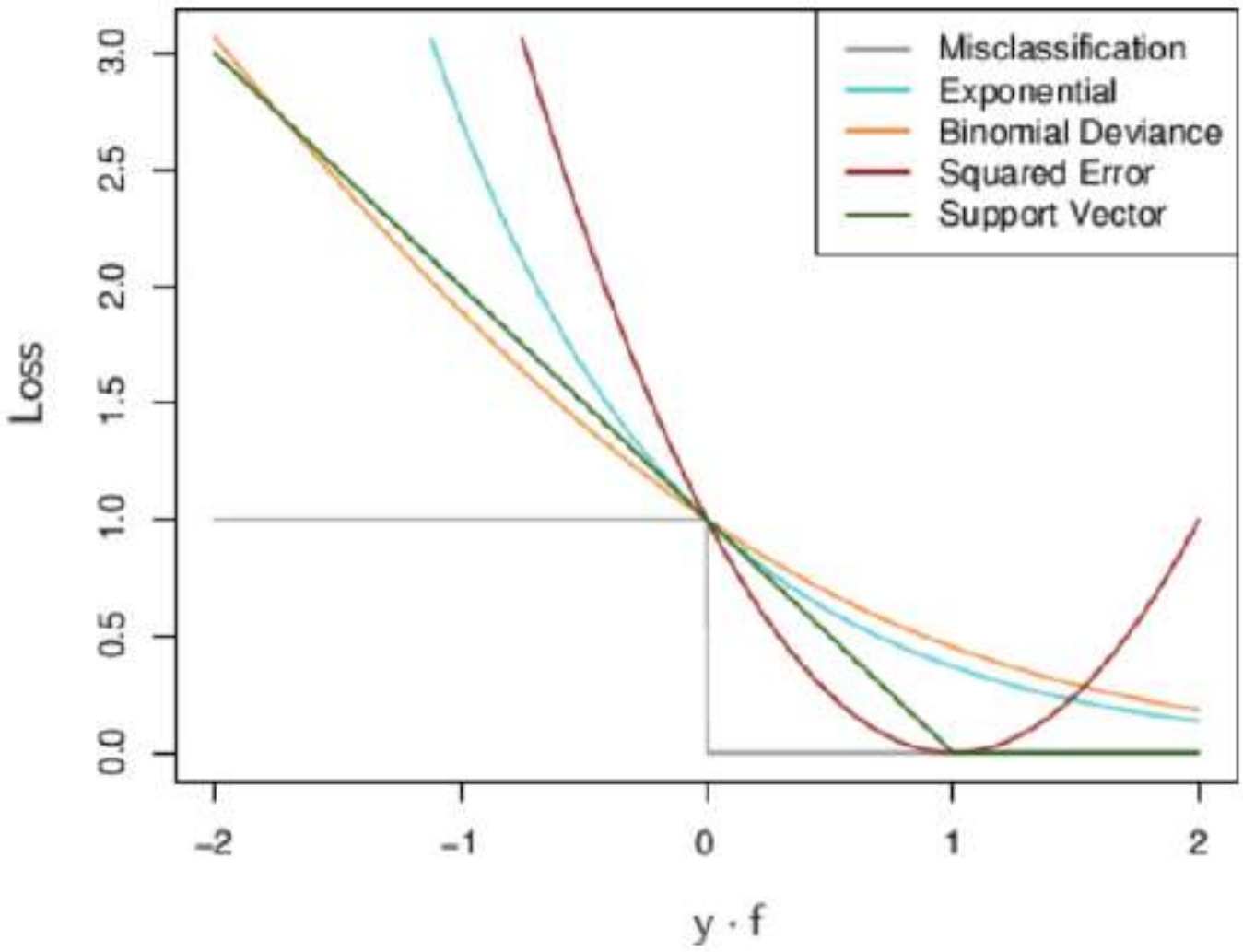
$$F(w) = loss\,(w) = \sum_n l(y(x_n)f(x_n, w))$$

- $l$ doit être assez lisse
- $l$ doit avoir une valeur proche de 0 si $y(x_n)f(x_n, w)$ est grand
- $l$ doit avoir une valeur très supérieure à 0 si $y(x_n)f(x_n, w)$ est très petit

**hinge loss :**

$$loss\,(w) = \sum_n relu(1 - y_n f(x_n, w))$$

# Fonction de perte



## Limite de la descente de gradient

$$loss\,(w) = \sum_n relu(1 - y_n f\,(x_n, w))$$

Si $N = 1000000$ ça veut dire que pour calculer $loss(w)$ je dois appliquer $f$ (plusieurs couches) à 1000000 points !

# Descente de gradient stochastique

*loss* est une fonction dérivable de $\mathbb{R}^D$ dans $\mathbb{R}$
et que $\mathit{loss}(u) = \sum_{i=1} q_i(u)$
alors **dans le cas convexe**, il est possible de minimiser *loss* en faisant comme une descente de gradient mais en prenant une sous sommes des $q_i$ tirée aléatoirement avec une politique $\lambda(t)$ fixée a priori (qui doit quand même vérifier certaines conditions).

# Descente de gradient stochastique

## pseudo code

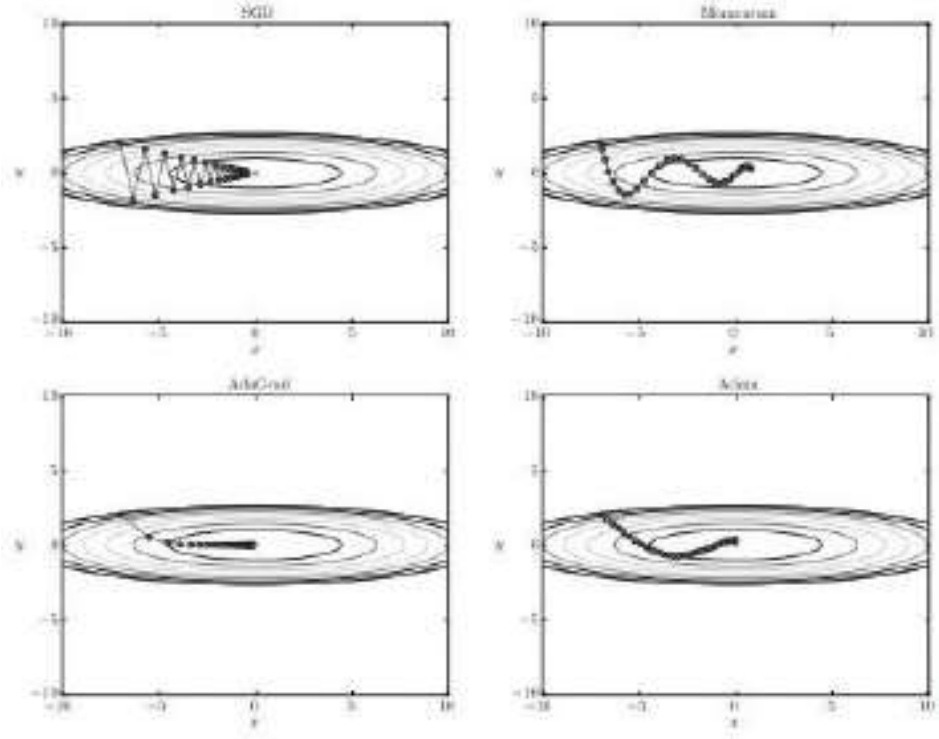
input : $x_1, y_1, ..., x_n, y_n,\ w_0$

1. $w = w_0$
2. $iter = 0$
3. tirer $n$ au hasard dans 1,...,N
4. $partial\_loss = relu(1 - y_n f(x_n, w))$
5. calculer $\nabla_w partial\_loss$
6. $w = w - \lambda_{iter} \nabla_w partial\_loss$
7. $iter = iter + 1$
8. si condition d'arrêt alors sortir
9. go to 3

# Descente de gradient stochastique

## Optimizer

$w = w - \lambda_{iter} \nabla_w partial\_loss$ est une possibilité mais il y en a d'autres :



# Synthèse

## L'apprentissage

$$f(x, w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$$

L'apprentissage consiste à appliquer la méthode de la descente de gradient stochastique (optimiseur à choisir) à une fonction de perte (à choisir) qui approxime l'erreur d'apprentissage
Par exemple

$$partial\_loss\,(w) = \sum_{n \in Batch} relu(1 - y_n f(x_n, w))$$

$$w = w - \lambda_{iter} \nabla_w partial\_loss$$

# Forward - Backward

Mais ça suppose qu'on sache calculer le gradient ! ! ! !

# Forward - Backward

## objectif

$$partial\_loss(w) = \sum_{n \in Batch} relu(1 - y_n f(x_n, w))$$

avec $f(x, w) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$

$\Rightarrow$ on veut calculer

$$\partial \frac{\partial partial\_loss(w)}{\partial w_{t,i,j}}$$

# Forward - Backward

## Forward

```
for t
    for i
        for j
            A[t][i] += relu(A[t-1][j])*w[t-1][i][j]
```

# Forward - Backward

## Réduction $w$ - $\alpha$

$$\frac{\partial loss}{\partial w_{t,i,j}} = \frac{\partial loss}{\partial \alpha_{t,i,j}} \frac{\partial \alpha_{t,i}}{\partial w_{t,i,j}} = \frac{\partial loss}{\partial \alpha_{t,i}} x_{t,j}$$

# Forward - Backward

## Réduction $\alpha$ - $\alpha$

$$\frac{\partial loss}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} w_{t,i,j} relu'(\alpha_{t,j})$$

relu est une fonction linéaire par morçeau, sa *dérivé* est donc une constante par morçeau

# Forward - Backward

## Attention

La somme dans $\frac{\partial loss}{\partial \alpha_{t,j}} = \sum_i \frac{\partial loss}{\partial \alpha_{t+1,i}} \frac{\partial \alpha_{t+1,i}}{\partial \alpha_{t,j}}$ ne vient **pas** de la somme dans $\alpha_{t+1,i} = \sum_j x_{t,j} w_{t,i,j}$.

Elle vient de $f(u) = a(b(u), c(u))$ implique $\frac{\partial f}{\partial u} = \frac{\partial a}{\partial b}\frac{\partial b}{\partial u} + \frac{\partial a}{\partial c}\frac{\partial c}{\partial u}$.

Lui même vient de $f(u+h) = f(u) + f'(u)h$

# Forward - Backward

```
for t
    for i
        for j
            A[t][i] += relu(A[t-1][j])*w[t-1][i][j]
DA[z][1] = partial_loss
for t from z to 1
    for j
        for i
            DA[t][j] += DA[t+1][i]*w[t][i][j]*relu'(A[t][j])
```

## ou en pytorch

```
z = net(x)
loss = criterion(z,y)
loss.backward
```

# Non monotonicité

Plus de neurones ça peut aggraver ou pas la performance

Plus de données ça peut augmenter ou pas la performance

# Le réglage du learning rate !

Plus de neurones ça peut aggraver ou pas la performance

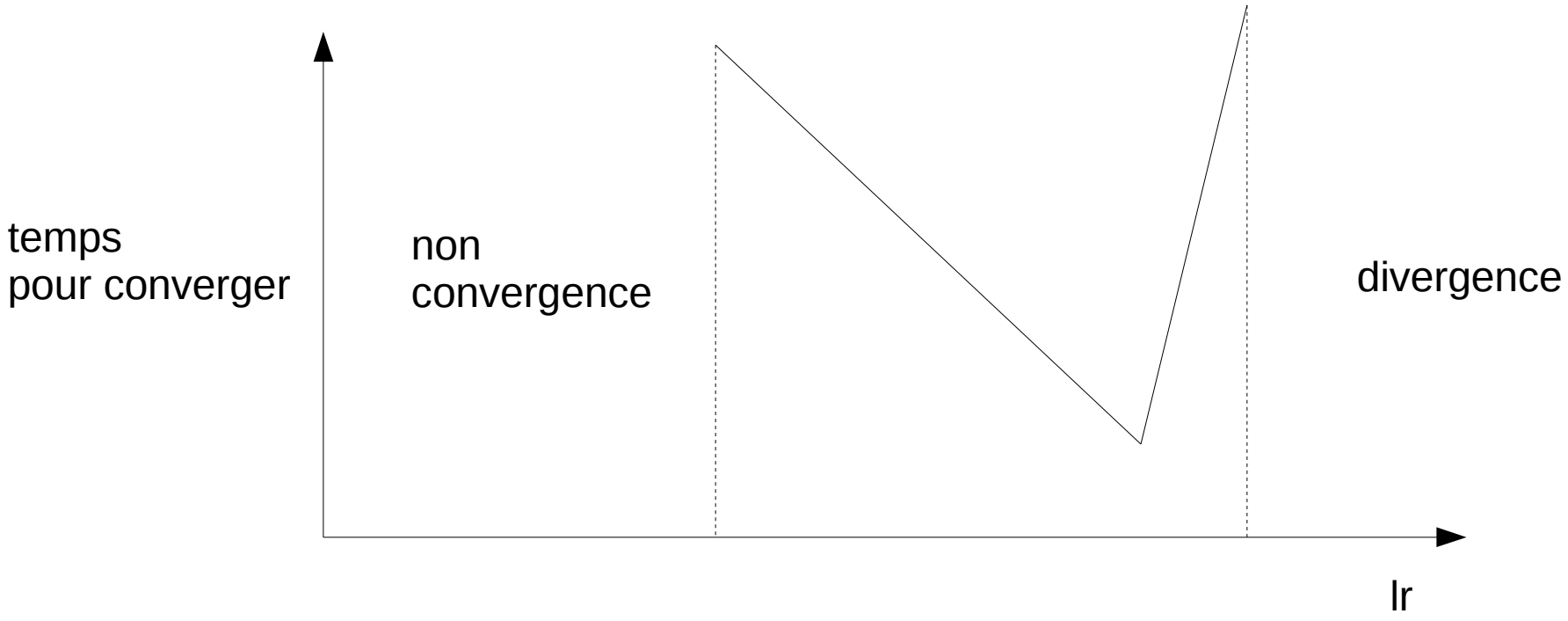
Plus de données ça peut augmenter ou pas la performance

Un learning rate plus haut ça peut diminuer ou pas la performance

$$W = W - lr \cdot \nabla F$$

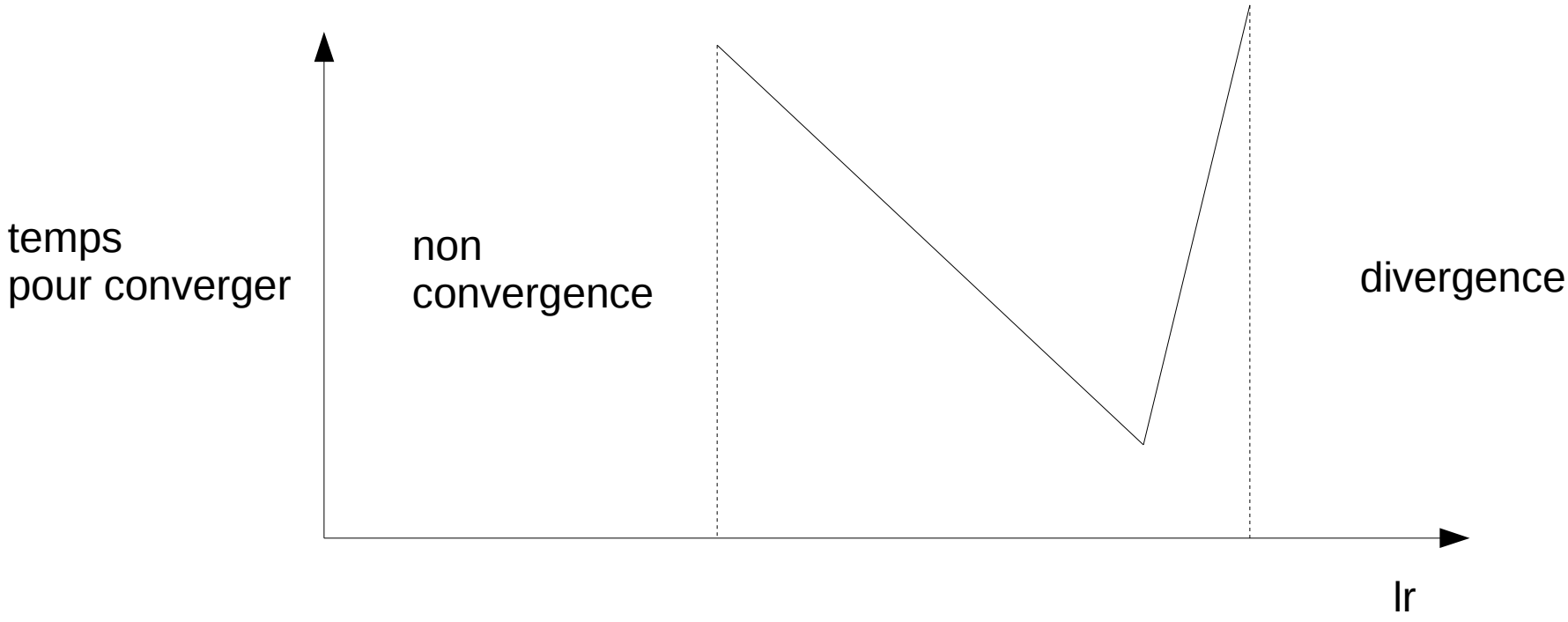
# Le réglage du learning rate !

Précisément, le temps pour converger diminue puis remonte



# Le réglage du learning rate !

Précisément, le temps pour converger diminue puis remonte



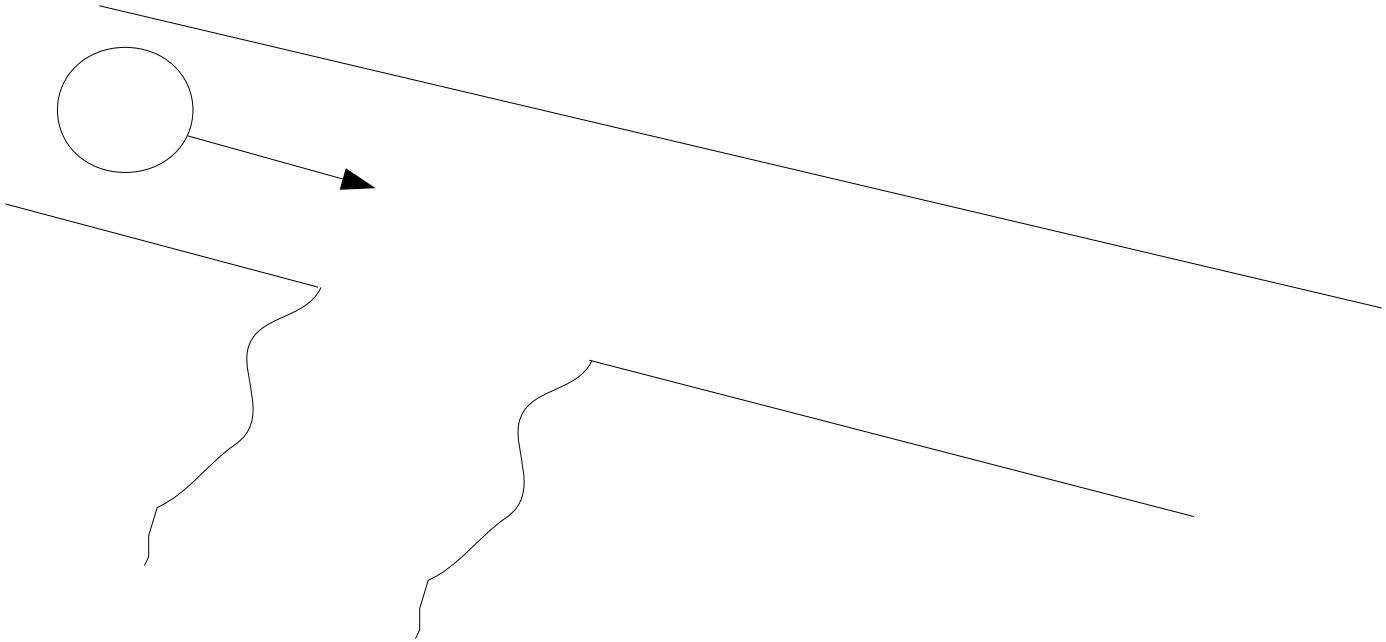

→ MAIS la généralisation est meilleur avec un learning rate élevé !

Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks

# Le réglage du learning rate !

Précisément, le temps pour converger diminue puis remonte

→ MAIS la généralisation est meilleur avec un learning rate élevé !



Un learning rate plus faible permet de prendre des « raccourcis » :
on converge plus vite mais on converge moins bien !

# Le réglage du learning rate !

Encore une fois c’est une **prime à la puissance de calcul** :
il ne faut pas avoir peur de calculer longtemps !

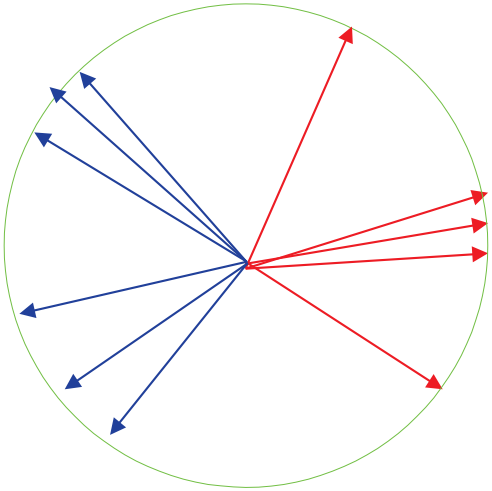
→ CIFAR10 : vous pouvez atteindre 2 % d’erreur à l’apprentissage en 5mins

Pourtant les meilleurs codes prennent des nuits
(pour obtenir une performance égale à l’apprentissage)

https://github.com/kuangliu/pytorch-cifar

# Le réglage du learning rate !

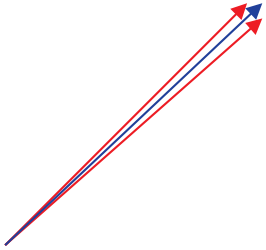
Grande dimension



Probablement transférable

Facile à apprendre si le LR est juste
sous une valeur critique qui fait diverger

2D



Probablement non transférable

Mais facile à apprendre si le LR est faible

Dur à apprendre à LR fort

# Plan

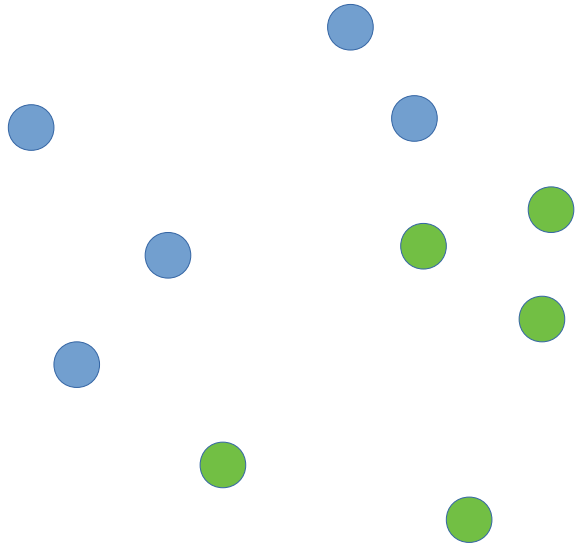
- ~~Classification pure :~~
  - ~~Apprentissage et test~~
  - ~~Neurones et réseau~~
  - ~~Universalité et erreur de généralisation~~
- ~~Double descente :~~
  - ~~Vapnik et régularisation~~
  - ~~Méthode ensembliste et double descente~~
  - ~~Descente de gradient stochastique~~
- Traitement de données structurées :
  - Le neurone convolutif
  - Les données structurées
  - La robustesse des réseaux

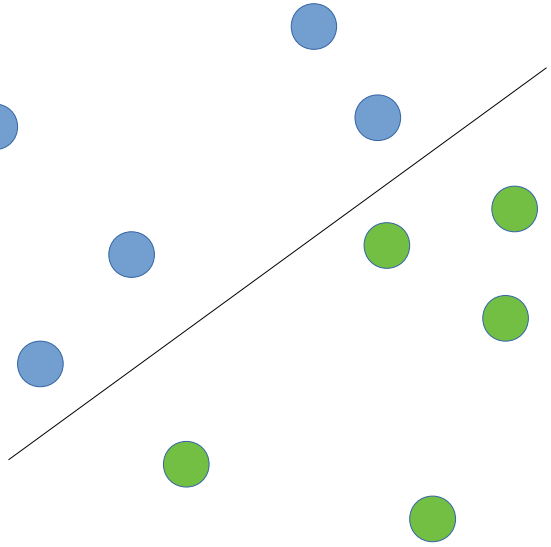
# Avant

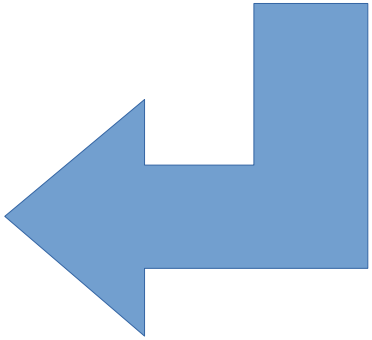
Données annotées et structurées
en dimension 100000000

Descriptions des données (features)
faites à la main
(hand crafted)

Nuage de points (D=10000)





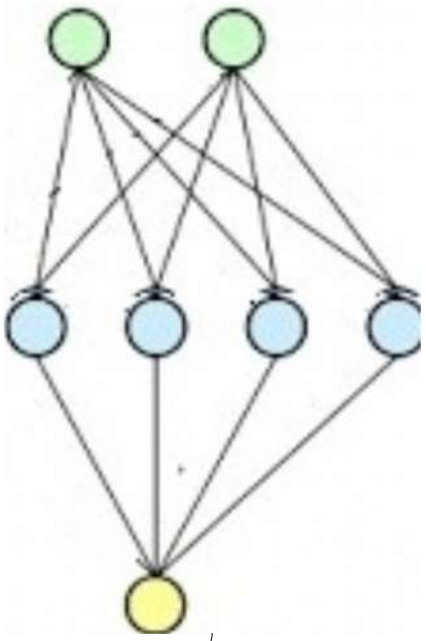


Classification de points

chat/chien

# Après

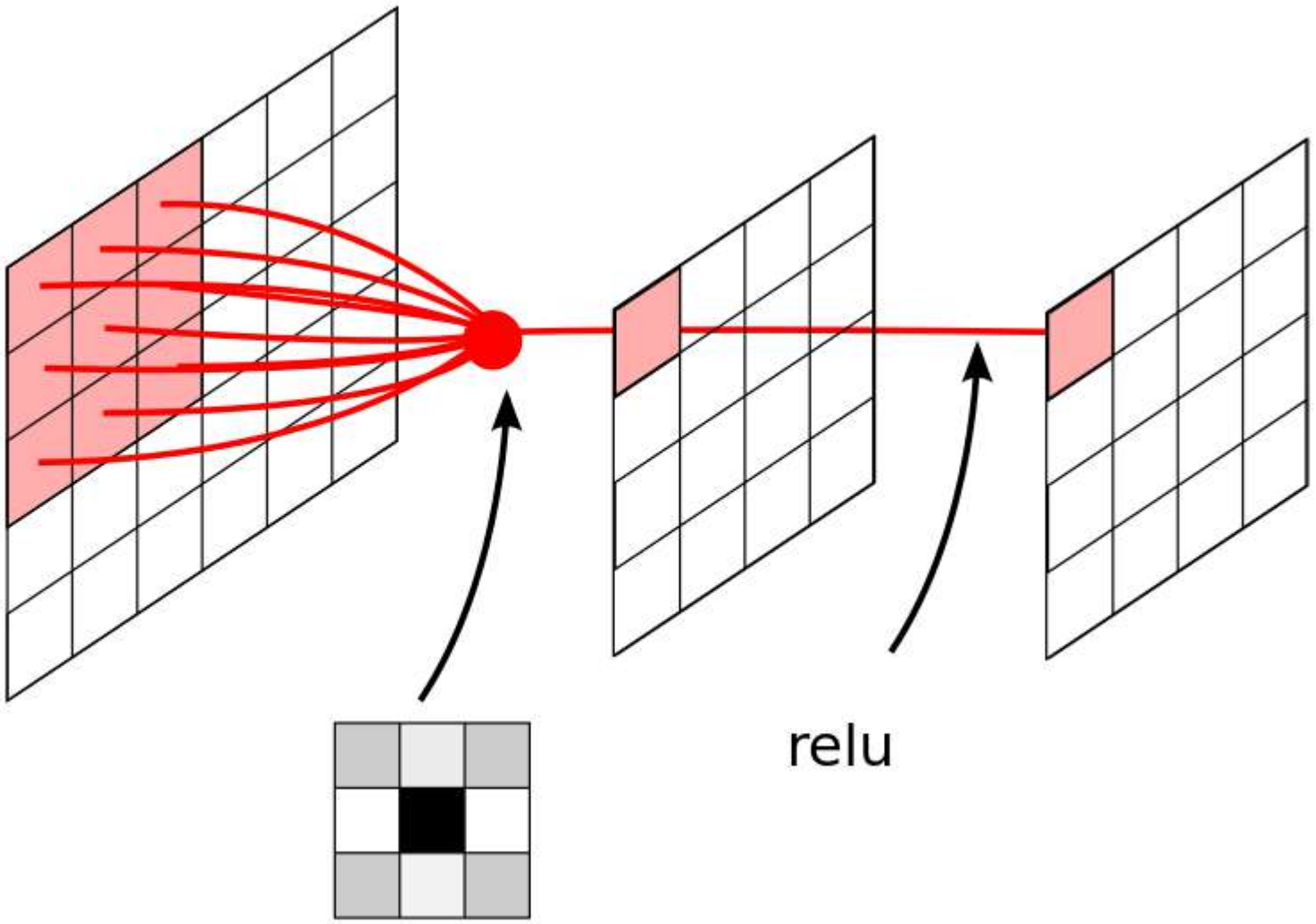
Données annotées et structurées
en dimension 100000000



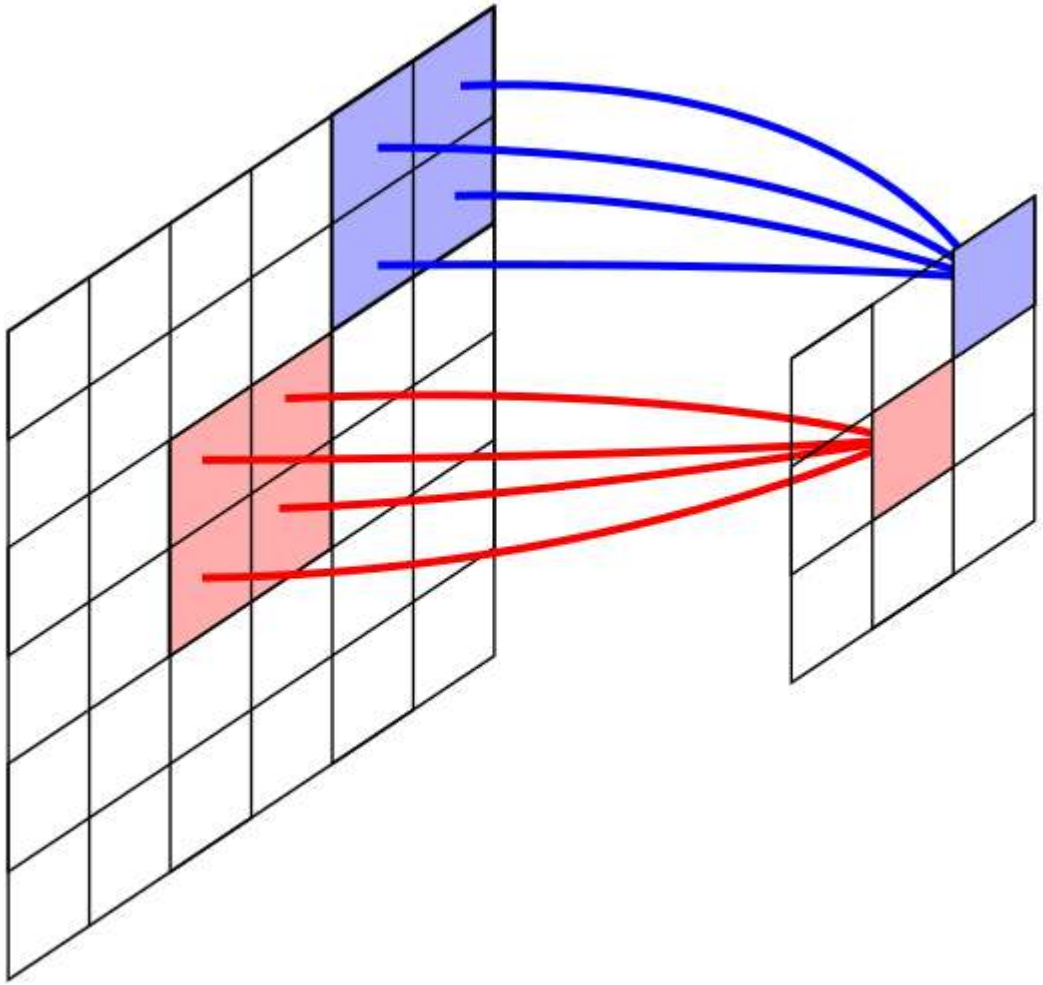
Le réseau de neurones fait TOUT

chat/chien

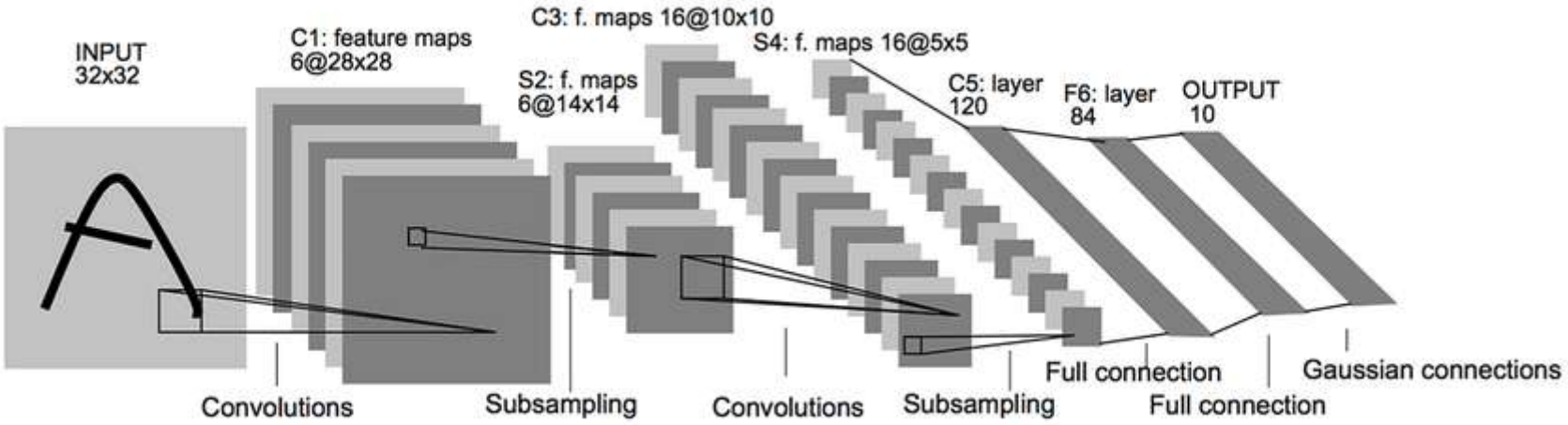
# Le neurone convolutif



# Le pooling

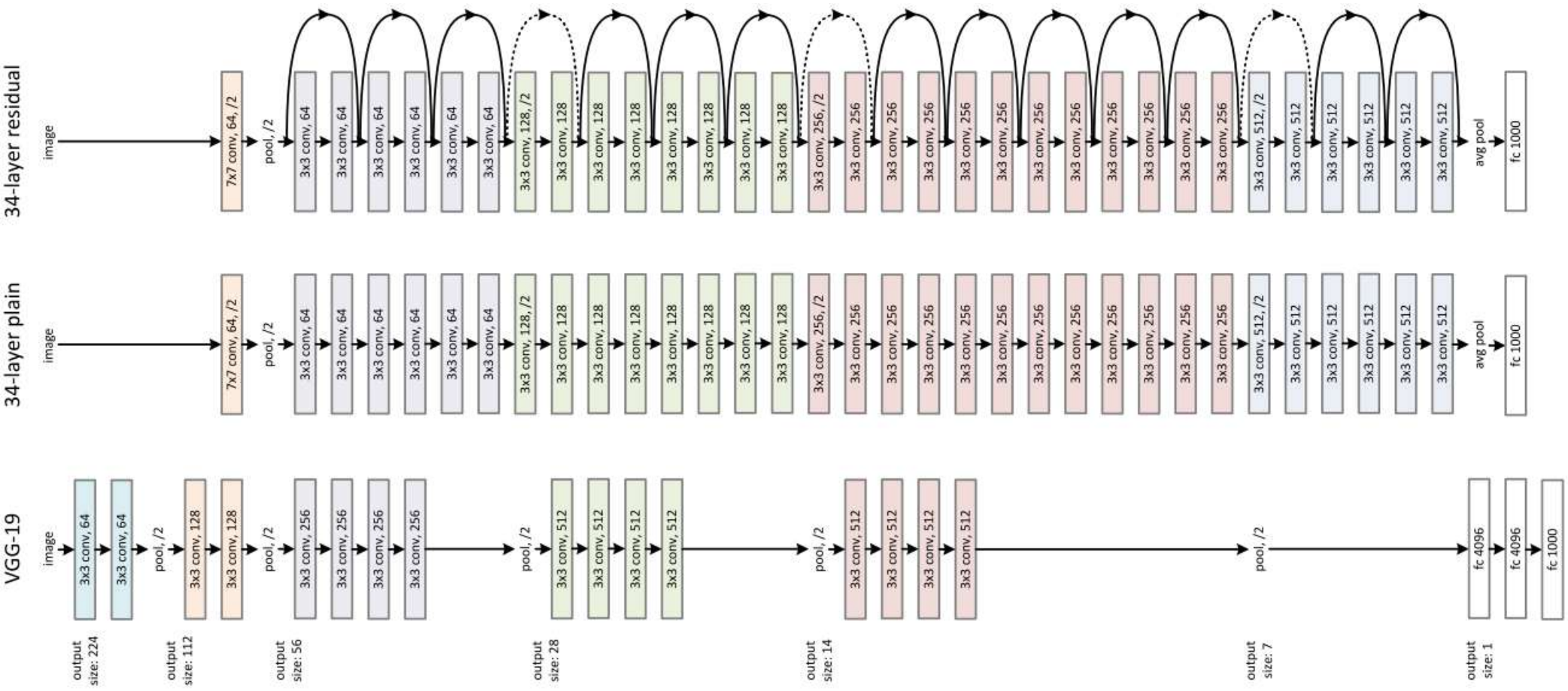


# Lenet

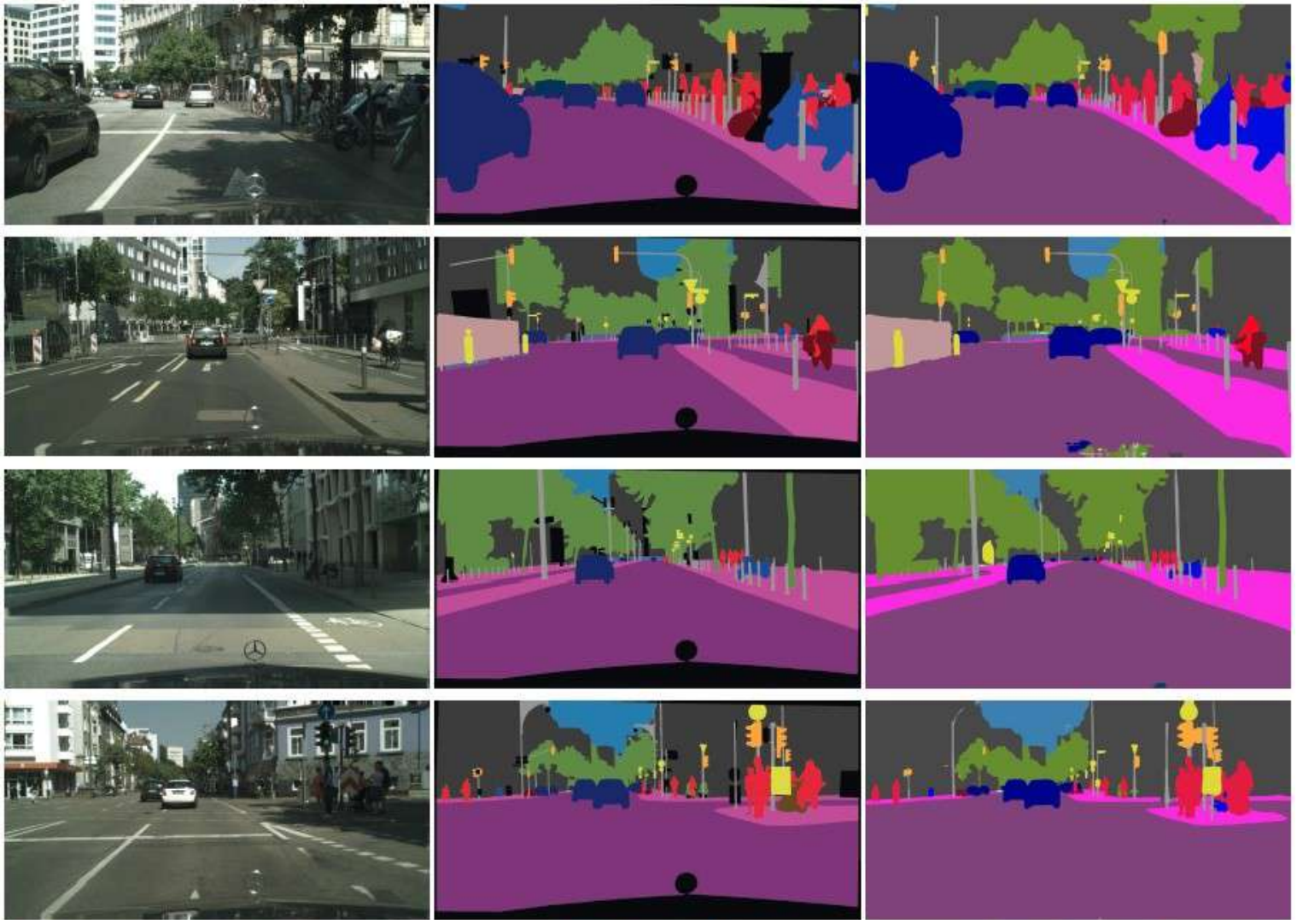

INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

# Deep Learning

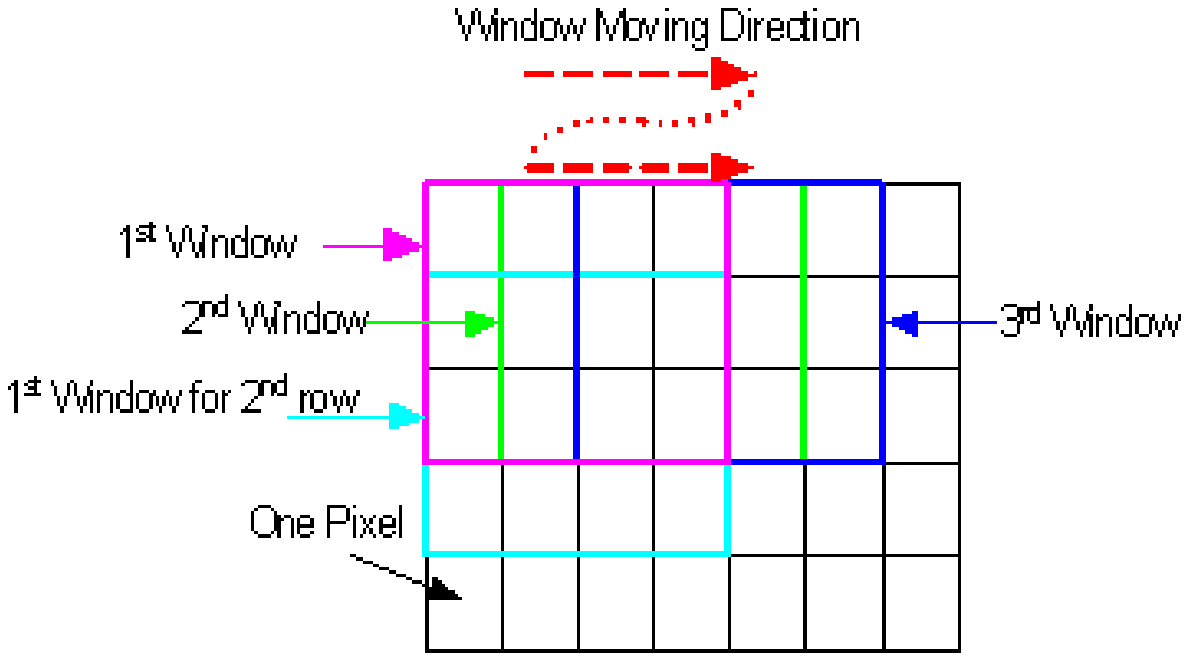
# Deep Learning



# Segmentation ?

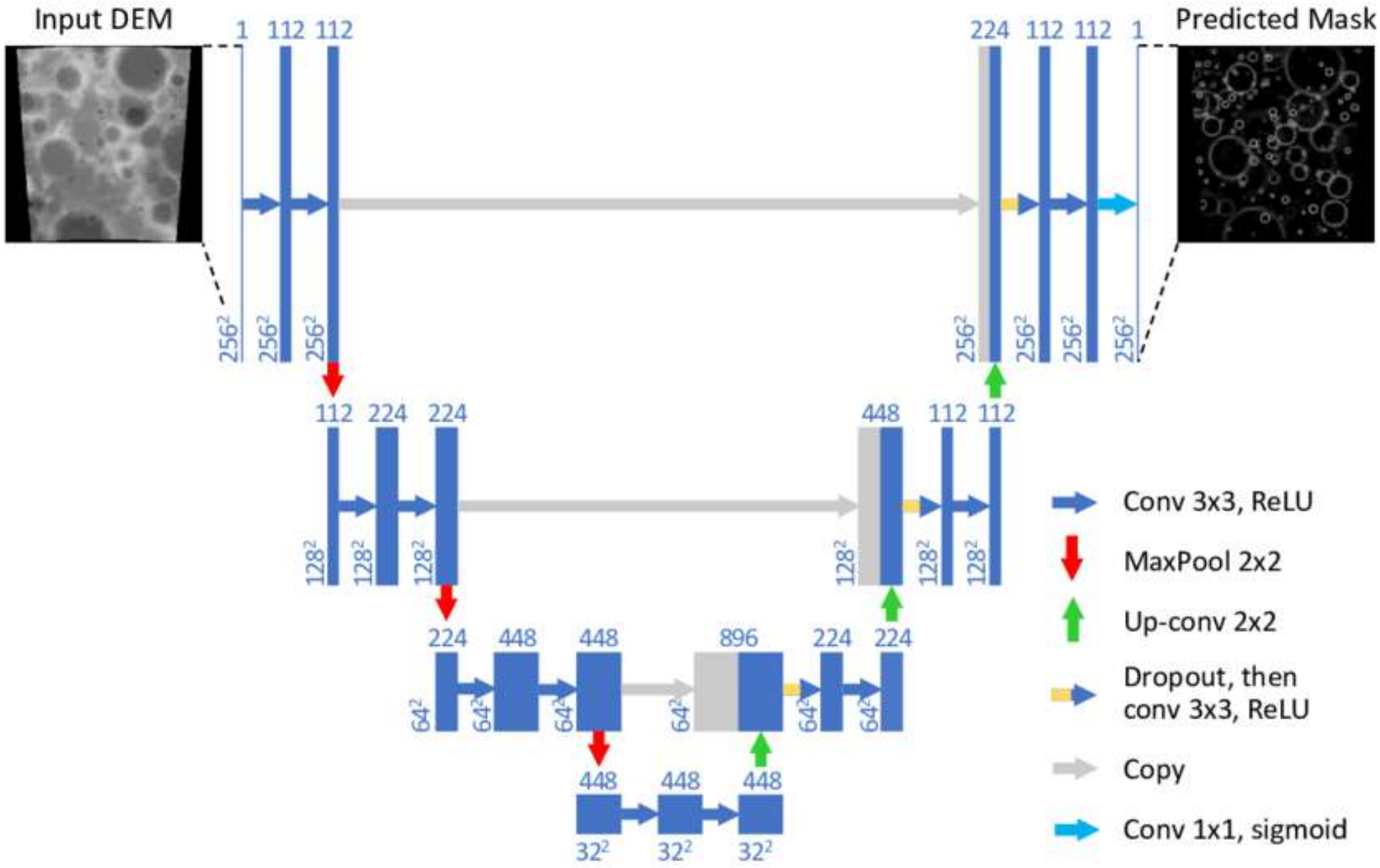


# La fenêtre glissante

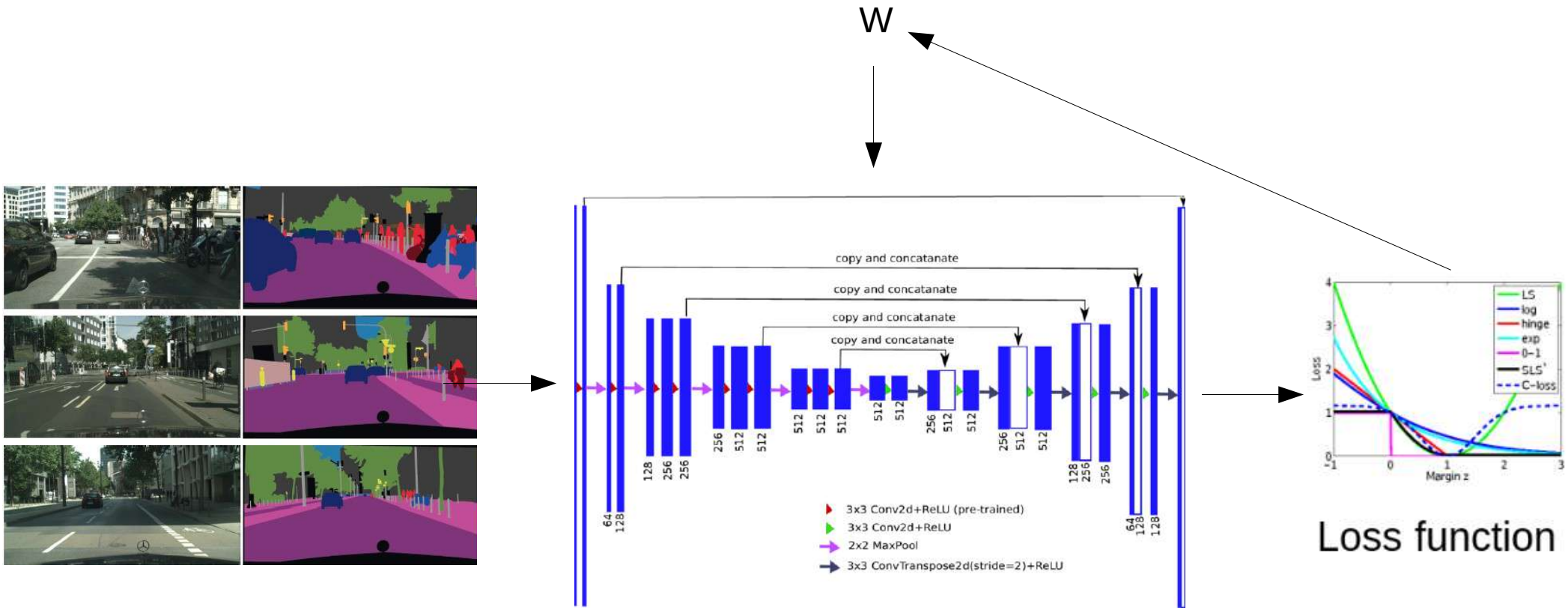



Plutôt que de raisonner par fenêtre, il est plus rapide de raisonner par couche !

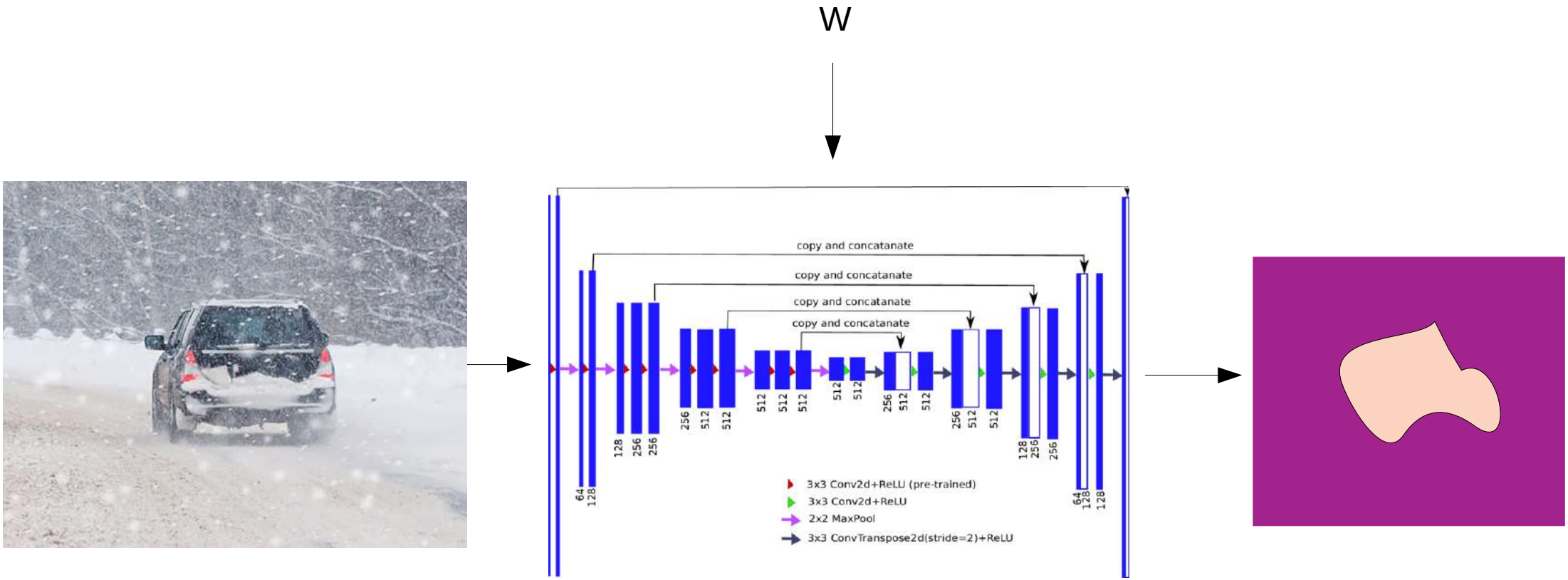
# Segmentation



# 2 phases :
# apprentissage puis test



# 2 phases :
# apprentissage puis test



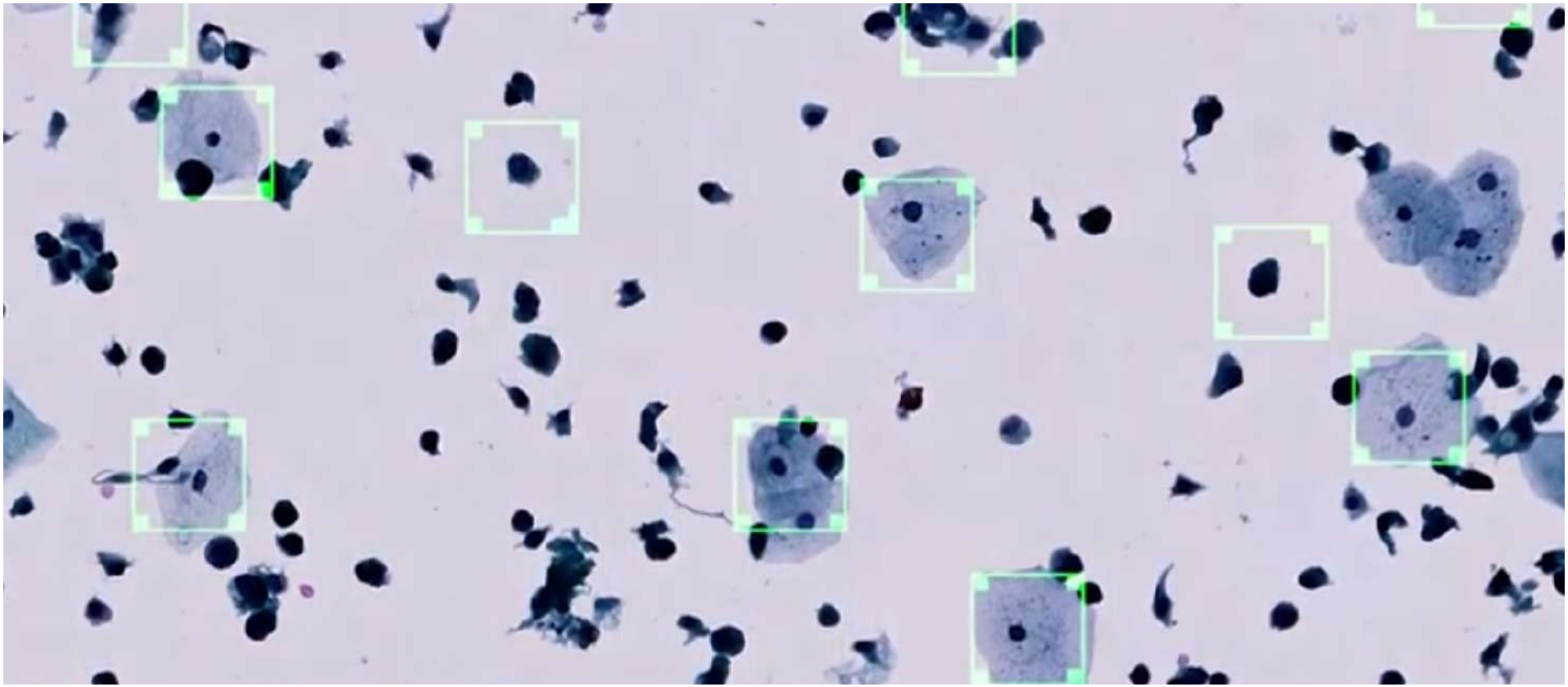
Image, vidéo, son, radar, classification, segmentation, détection

texte, classification, traduction

**Le deep learning est meilleur PARTOUT !**

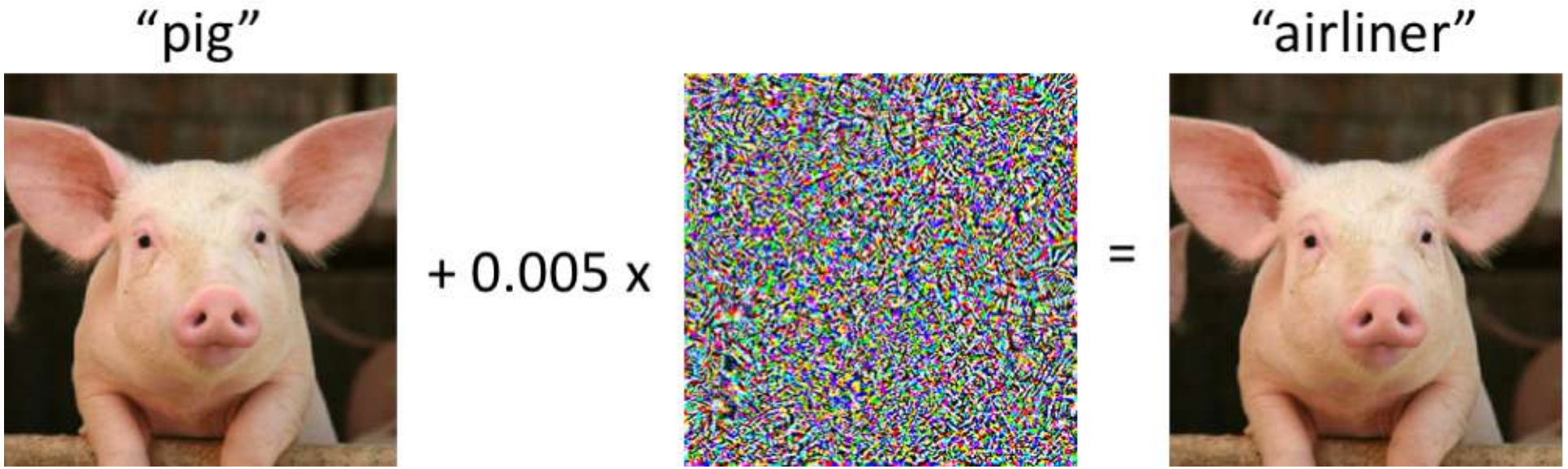
# Disclaimer

Quand on n’a pas beaucoup de données, c’est quand même pas toujours le mieux…



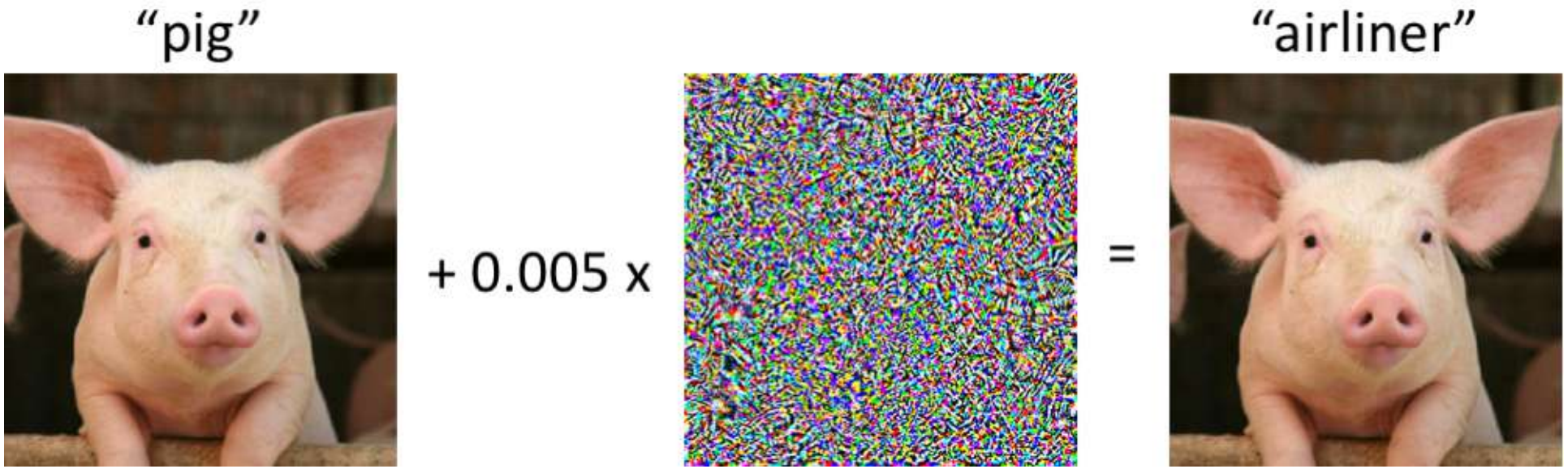
Hybridiser deeplearning et connaissance médicale !
https://vitadx.com/en/node/3

# Grande dimension et instabilité




Les réseaux de neurones naifs sont semble à des perturbations invisibles !

# Grande dimension et instabilité




Les réseaux de neurones naifs sont semble à des perturbations invisibles !

→ en réalité, c’est pas clair que ce soit grave
car ces perturbations ne sont pas forcément physique

# Grande dimension et instabilité

Durant l’apprentissage, on calcule

$$\text{Grad\_w } F$$

Mais on peut tout à fait calculer de la même façon

$$\text{Grad\_x } F$$

Et donc faire une « montée » de gradient sur x !

$$X = X + lr \,.\, \text{Grad\_x } F$$

Comme les gradients sont très grands

$$f(X) - f(X + lr \,.\, \text{Grad\_x } F) >> 1$$

# Rendre les réseaux robustes

f un réseau binaire

Ce qu'on ne veut pas c'est f(x)>0 et f(x+delta)<0 (ou l'inverse) sur le testing set

→ on veut donc apprendre au réseau à considérer que x est bien classé

**non** pas si f(x)>0 **mais** si f(x+delta)>0 (delta<epsilon)

# Rendre les réseaux robustes

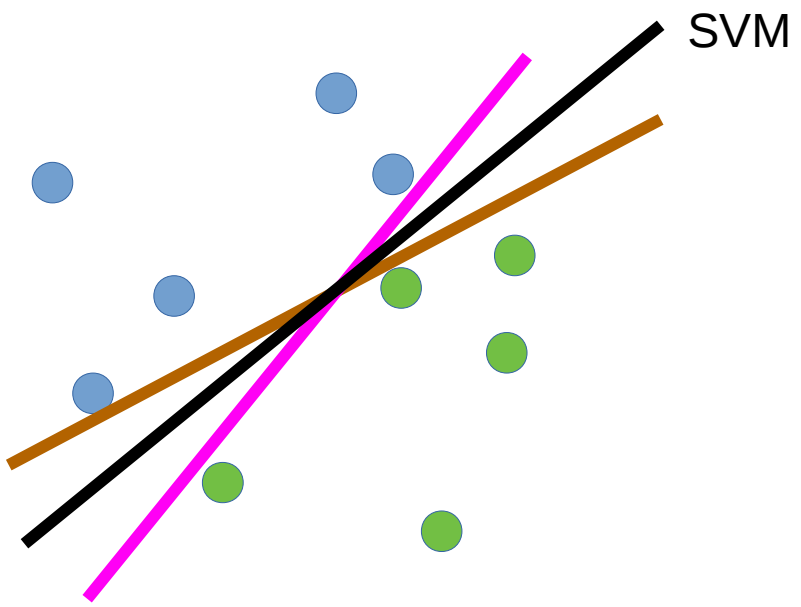
f un réseau binaire

Ce qu'on ne veut pas c'est f(x)>0 et f(x+delta)<0 (ou l'inverse) sur le testing set

→ on veut donc apprendre au réseau à considérer que x est bien classé

**non** pas si f(x)>0 **mais** si f(x+delta)>0 (delta<epsilon)

→ Le SVM fait exactement ça !



# Rendre les réseaux robustes

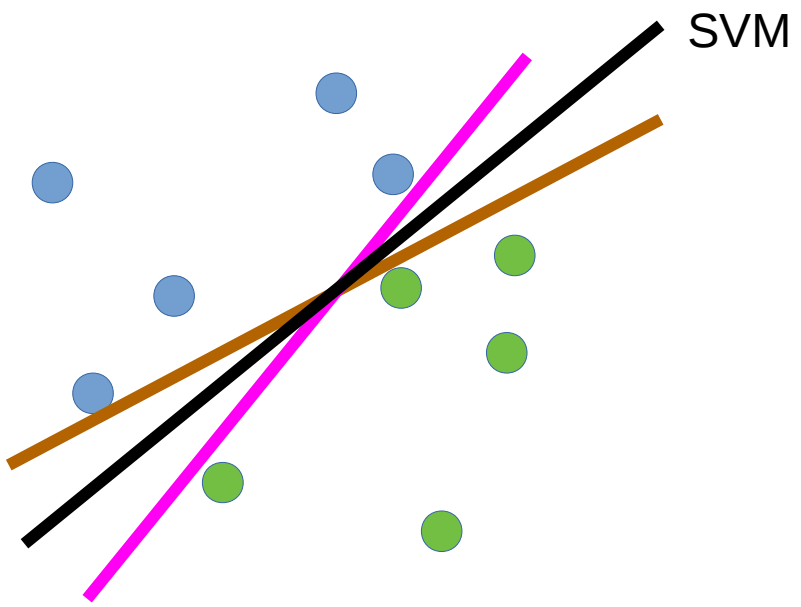
f un réseau binaire

Ce qu'on ne veut pas c'est f(x)>0 et f(x+delta)<0 (ou l'inverse) sur le testing set

→ on veut donc apprendre au réseau à considérer que x est bien classé

**non** pas si f(x)>0 **mais** si f(x+delta)>0 (delta<epsilon)

→ Le SVM fait exactement ça !



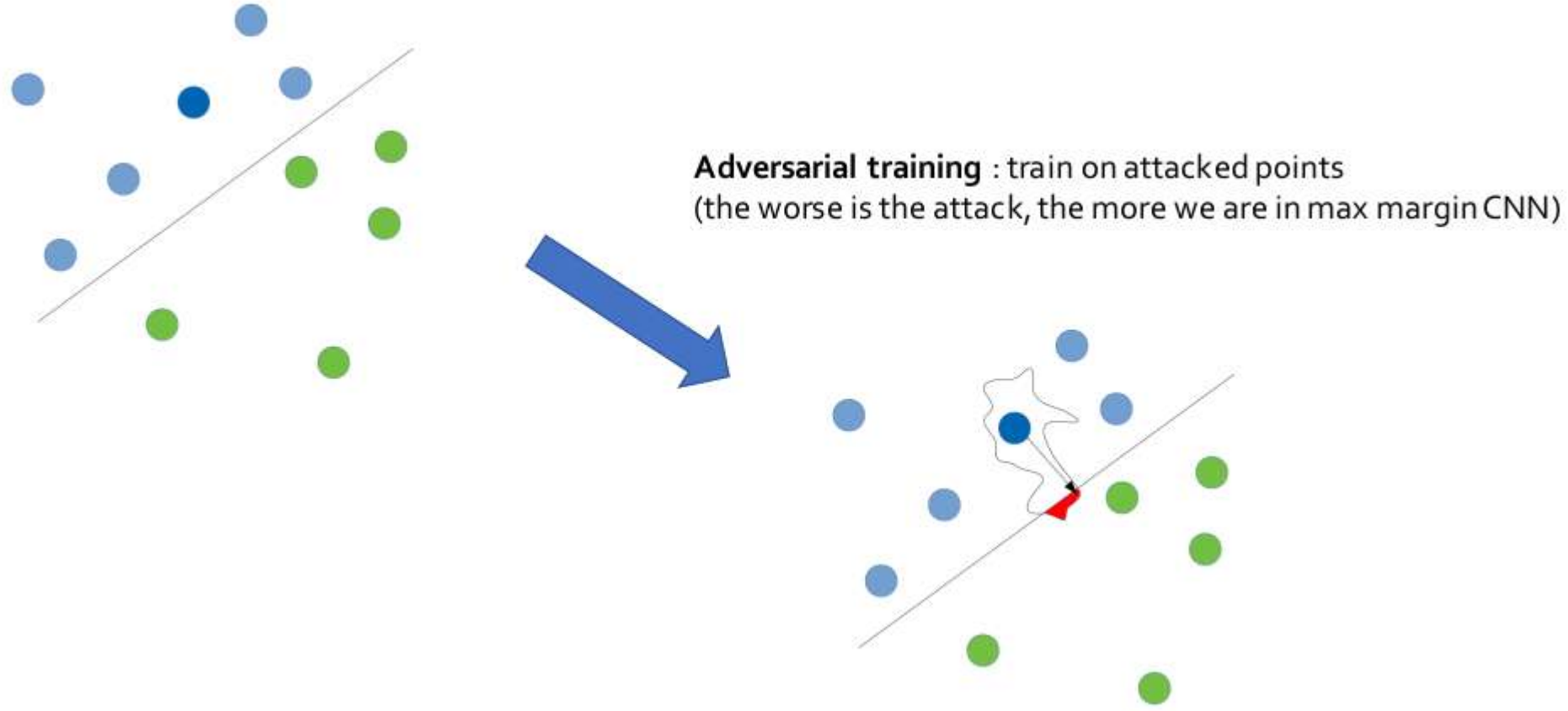

Sauf que calculer la marge est NP-Complet sur un réseau de neurones

# Rendre les réseaux robustes

2 options :

sur évaluer la marge de façon heuristique

sous évaluer la marge via une enveloppe convexe



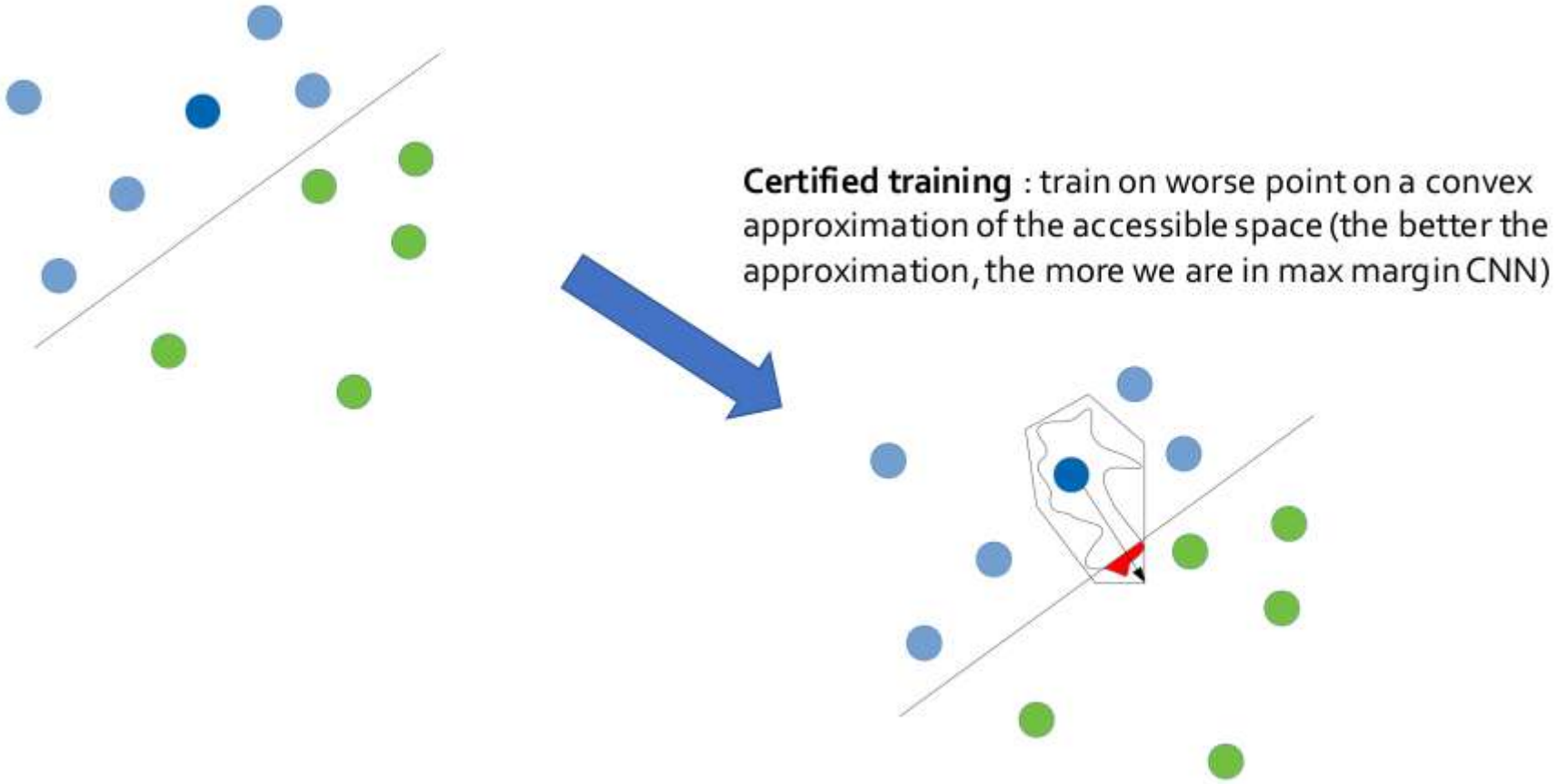

Si l'attaque n'est pas assez performance, le réseau n'est que peu protégé

# Rendre les réseaux robustes

2 options :

sur évaluer la marge de façon heuristique

sous évaluer la marge via une enveloppe convexe



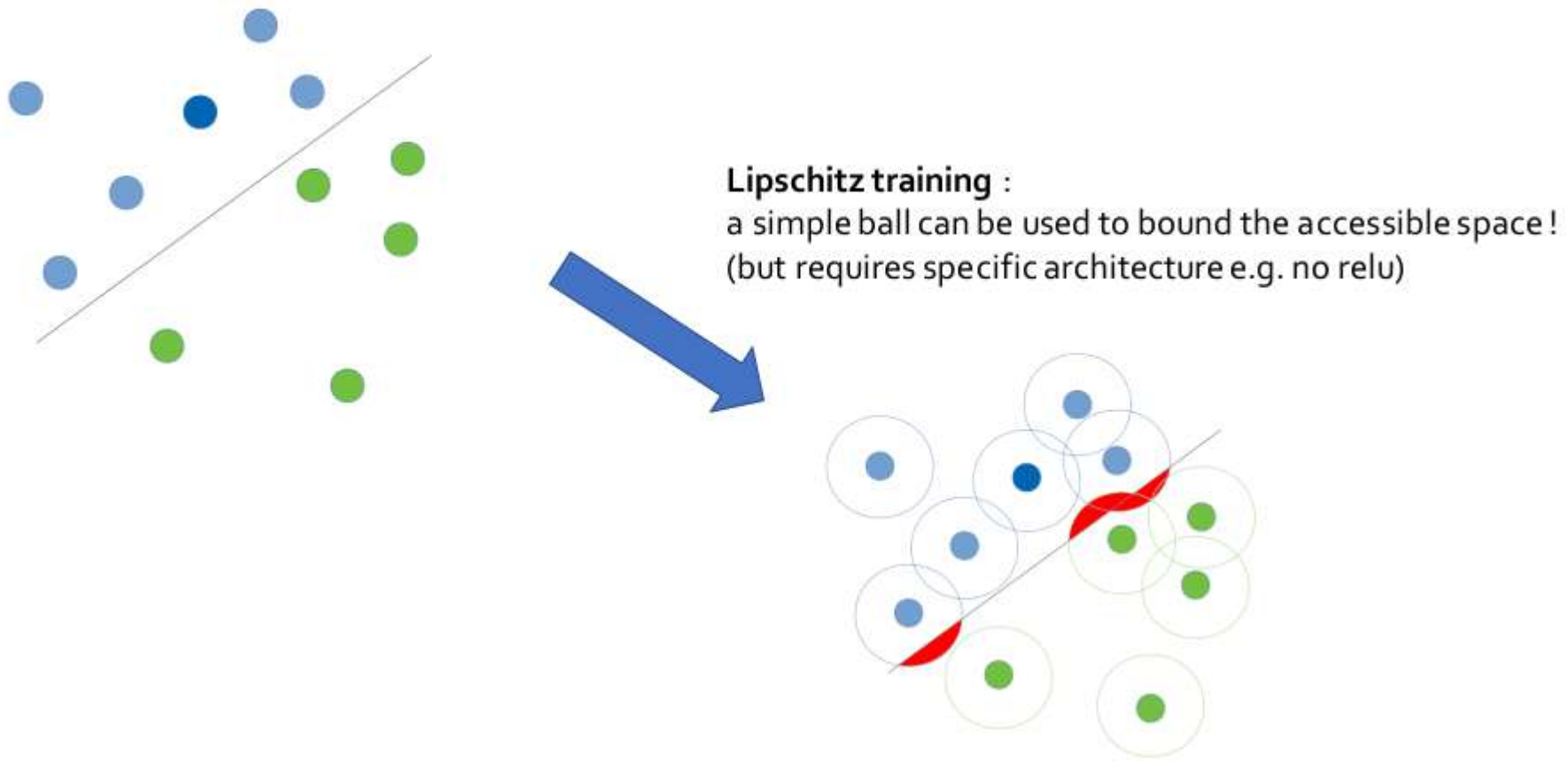

provable defenses against adversarial examples via the convex outer adversarial polytope

# Rendre les réseaux robustes

2 options :

sur évaluer la marge de façon heuristique

sous évaluer la marge via une enveloppe convexe




Sorting Out Lipschitz Function Approximation

# Rendre les réseaux robustes

relu $\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} \max(A, 0) \\ \max(B, 0) \\ \max(C, 0) \\ \max(D, 0) \end{pmatrix}$

Les valeurs n'augmentent pas
(1 lips)
mais peuvent diminuer

minmax $\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} \min(A, B) \\ \min(C, D) \\ \max(A, B) \\ \max(C, D) \end{pmatrix}$

Les valeurs ne changent pas

Seul leurs places changent

# Rendre les réseaux robustes

$$\text{relu}\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} \max(A, 0) \\ \max(B, 0) \\ \max(C, 0) \\ \max(D, 0) \end{pmatrix}$$

Les valeurs n'augmentent pas
(1 lips)
mais peuvent diminuer

$$\text{minmax}\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} \min(A, B) \\ \min(C, D) \\ \max(A, B) \\ \max(C, D) \end{pmatrix}$$

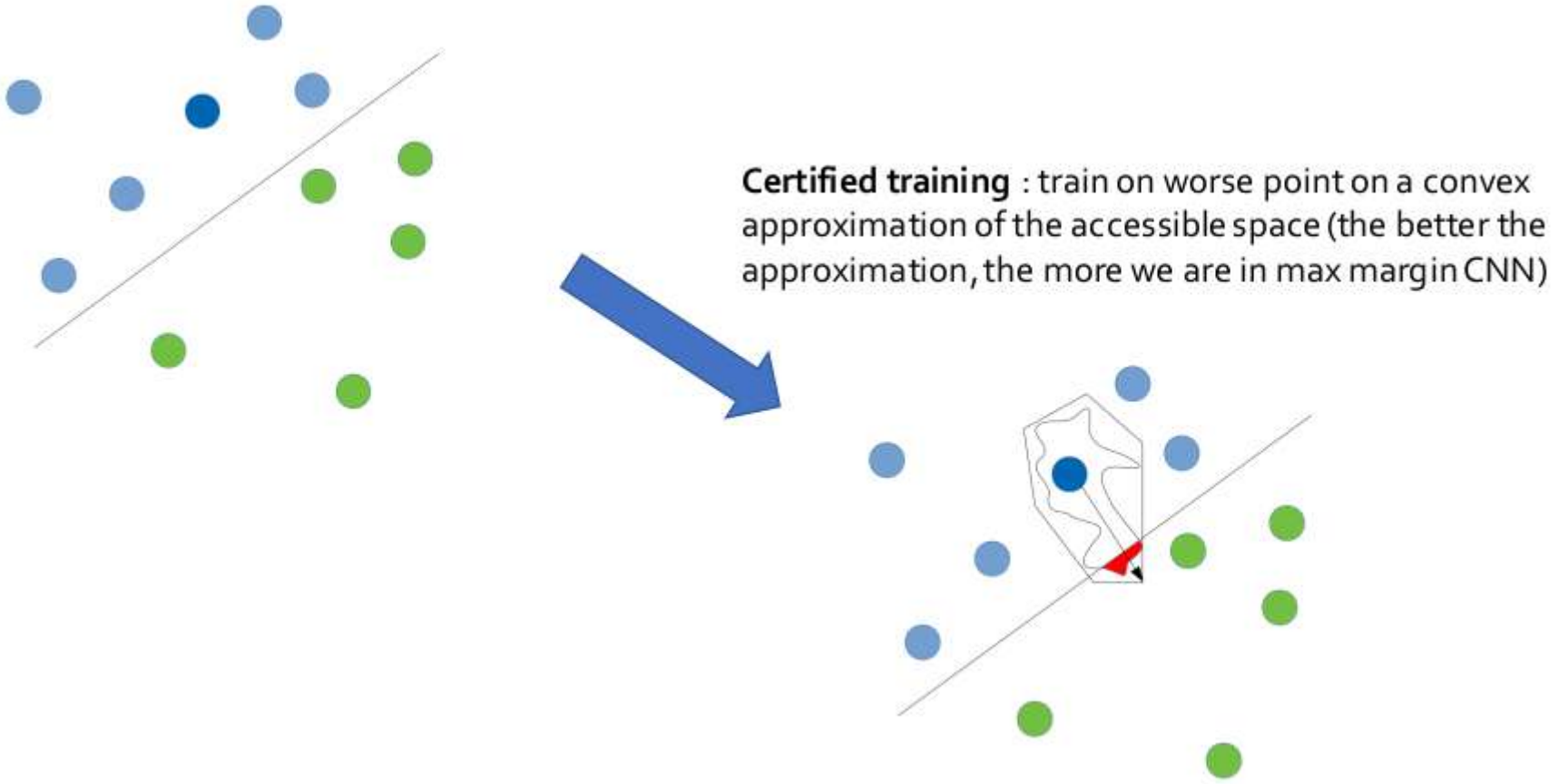
Les valeurs ne changent pas

Seul leurs places changent

$$\text{minmax}\begin{pmatrix} A \\ 0 \\ C \\ 0 \end{pmatrix} = \begin{pmatrix} \min(A, 0) \\ \min(C, 0) \\ \max(A, 0) \\ \max(C, 0) \end{pmatrix} = \begin{pmatrix} \text{-relu}\begin{pmatrix} A \\ C \end{pmatrix} \\ \text{relu}\begin{pmatrix} A \\ C \end{pmatrix} \end{pmatrix}$$

# Rendre les réseaux robustes

La encore une grosse prime à la puissance de calcul !




200 neurones pour apprendre 20 points !
32Go de ram (5000€) pour calculer le gradient sur 2 image 32x32 avec VGG13 !

# Les limites

Apprentissage avec peu de données (frugal learning)

Apprentissage de nouvelles classes à la volé (incremental learning)

Apprentissage éthique (~~robustness~~, fairness, privacy preserving)

Apprentissage et langage (explainability)

Apprentissage hybride (physically informed neural network)

Apprentissage de représentation (self supervised learning, representation learning)

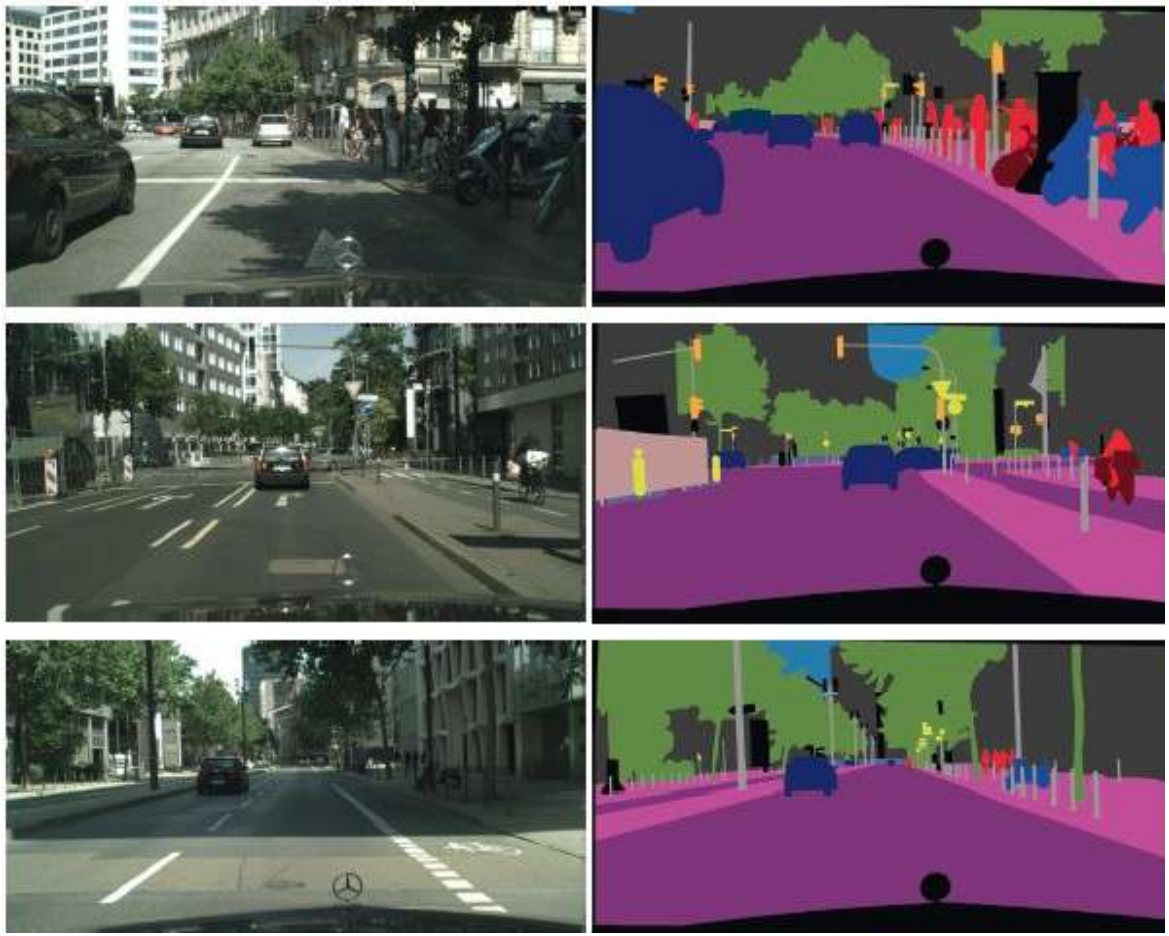
Adaptation de domaine (transfer learning)

MAIS le problème de la généralisation n’est pas réglée

# Les limites

Le problème de la généralisation n’est pas réglée
(malgré la double descente)

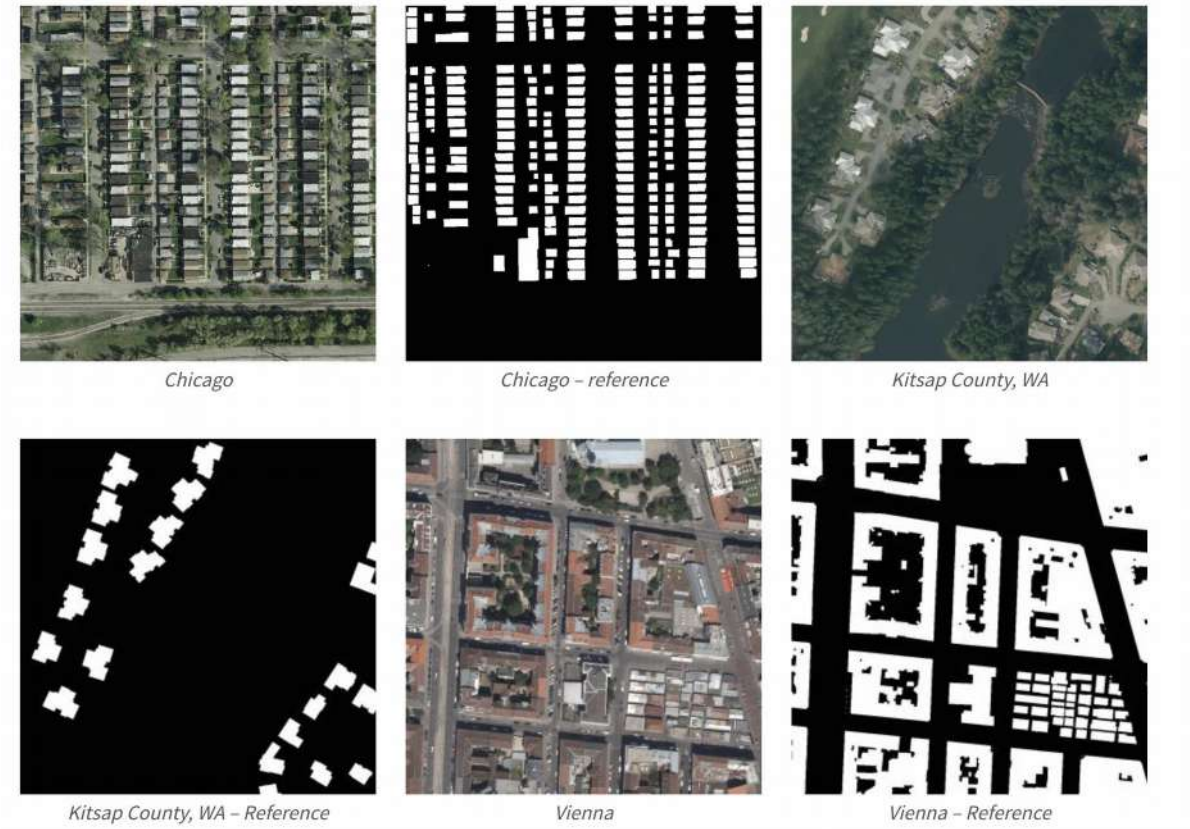
**→ c’est un problème d’apprentissage mais aussi d’industrialisation de la collecte de données**

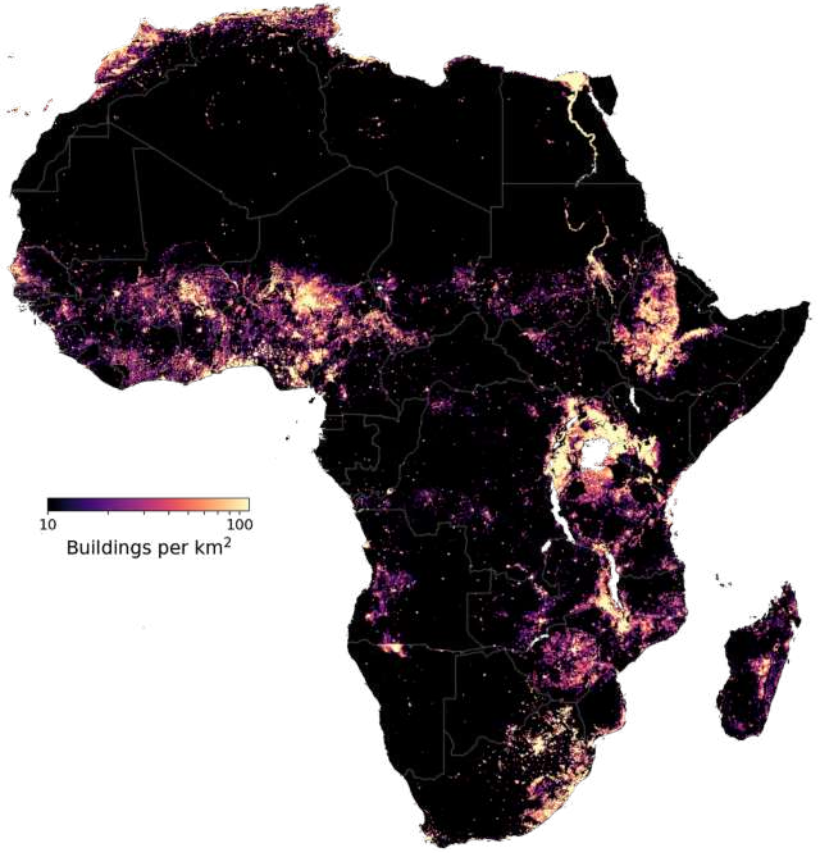


# Les limites

Le problème de la généralisation n’est pas réglée
(malgré la double descente)

→ **c’est un problème d’apprentissage mais aussi d’industrialisation de la collecte de données**




Inria Aerial Dataset : 6 villes

Open building dataset : l’Afrique en entier !

# Messages to take home

- D’abord apprentissage puis test sur d’autres données
- Et ce qui compte c’est la performance en test !

- Réseau de neurones = couches « linéaire + activation »
- Apprentissage avec SGD

- Le deep learning est surtout pertinent sur les données structurées
- Il ne souffre pas trop d’overfitting → prime à la puissance de calcul

- Sensibilité des réseaux → tempête dans un verre d’eau
- Mais le problème de la généralisation n’est pas vraiment résoluble

# TP

https://playground.tensorflow.org pas d’overfitting observé

Notebook python CIFAR10 et influence du learning rate

# Questions ?