Introduction au Deep learning

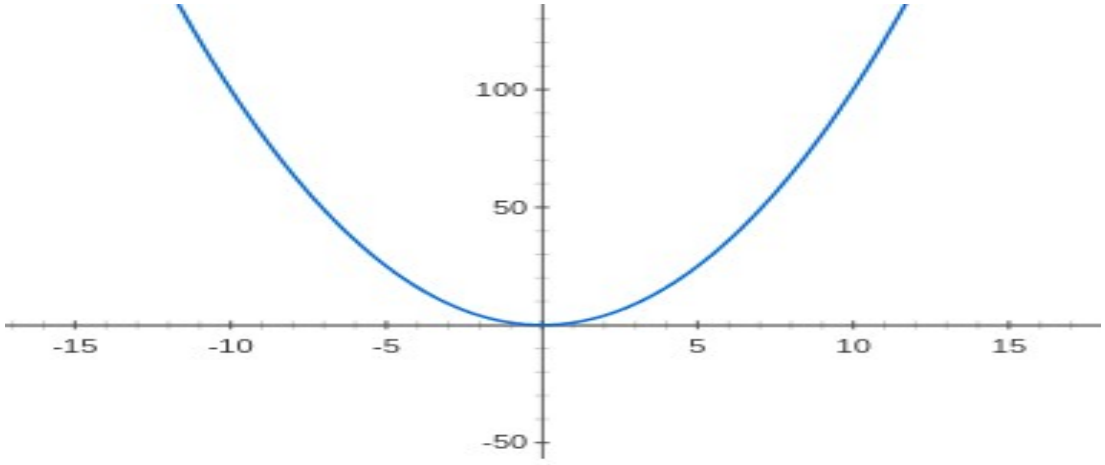
Adrien CHAN-HON-TONG
ONERA

EUROSAE septembre 2022

Avant-propos

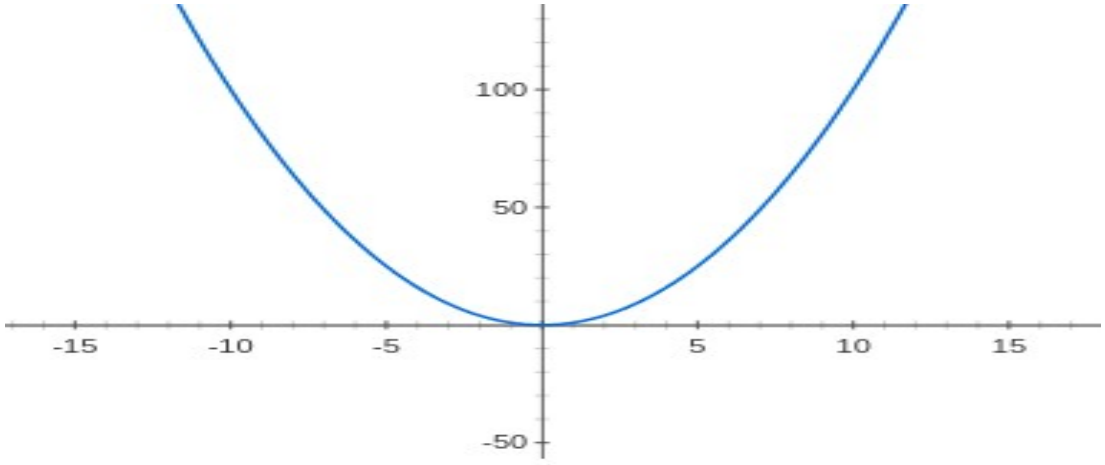
Définir une fonction ?

f(x) = x*x



Définir une fonction ?

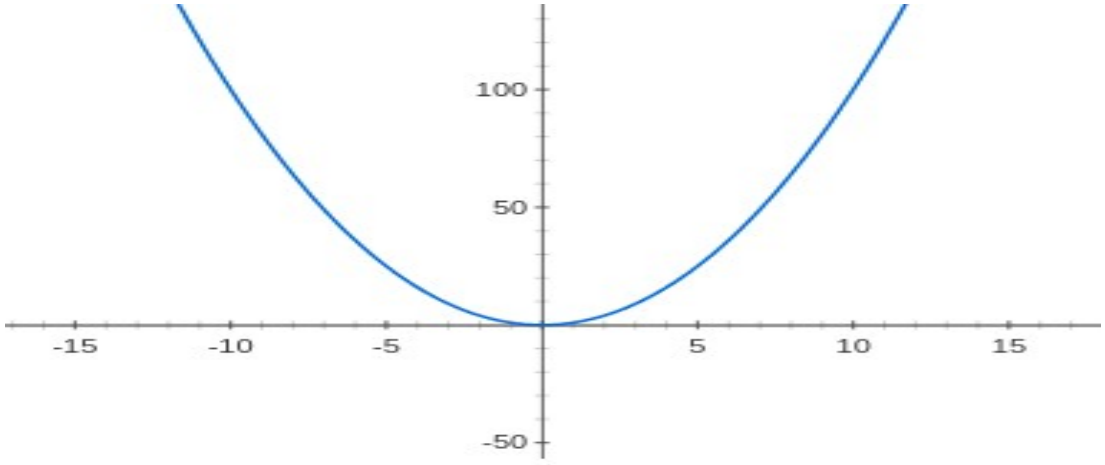
$f(x) = x*x$



Cette fonction est définie **partout.**

Définir une fonction ?

$$f(x) = x*x$$



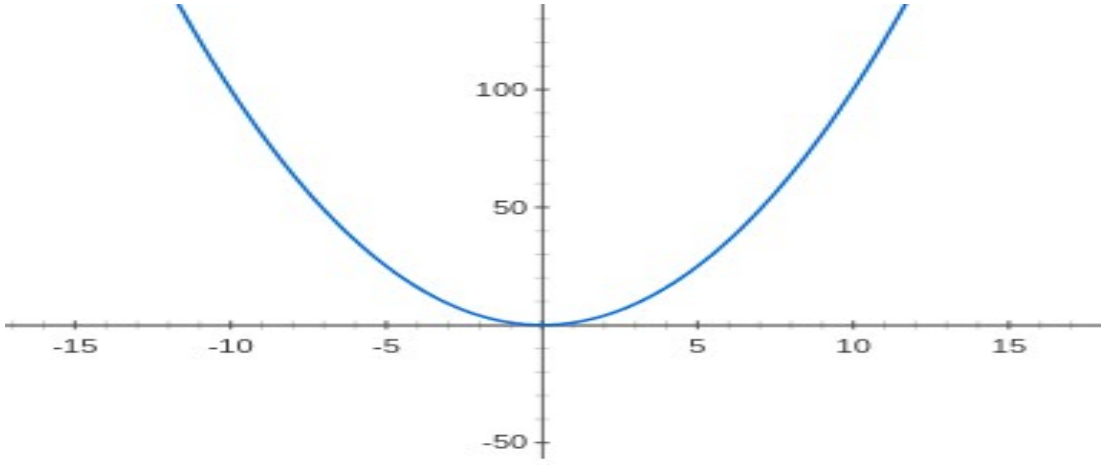
Cette fonction est définie **partout.**

Même si je considère une fonction plus complexe, elle peut être définie partout.

$$G(A,b,c) = \min_{x,\ Ax \geq b} cx$$

Définir une fonction ?

$$f(x) = x*x$$



Cette fonction est définie **partout.**

Même si je considère une fonction plus complexe, elle peut être définie partout.

$$G(A,b,c) = \min_{x,\ Ax \geq b} cx$$

Si un programme implémente la fonction, on peut espérer démontrer son exactitude.
Car la fonction est définie partout.

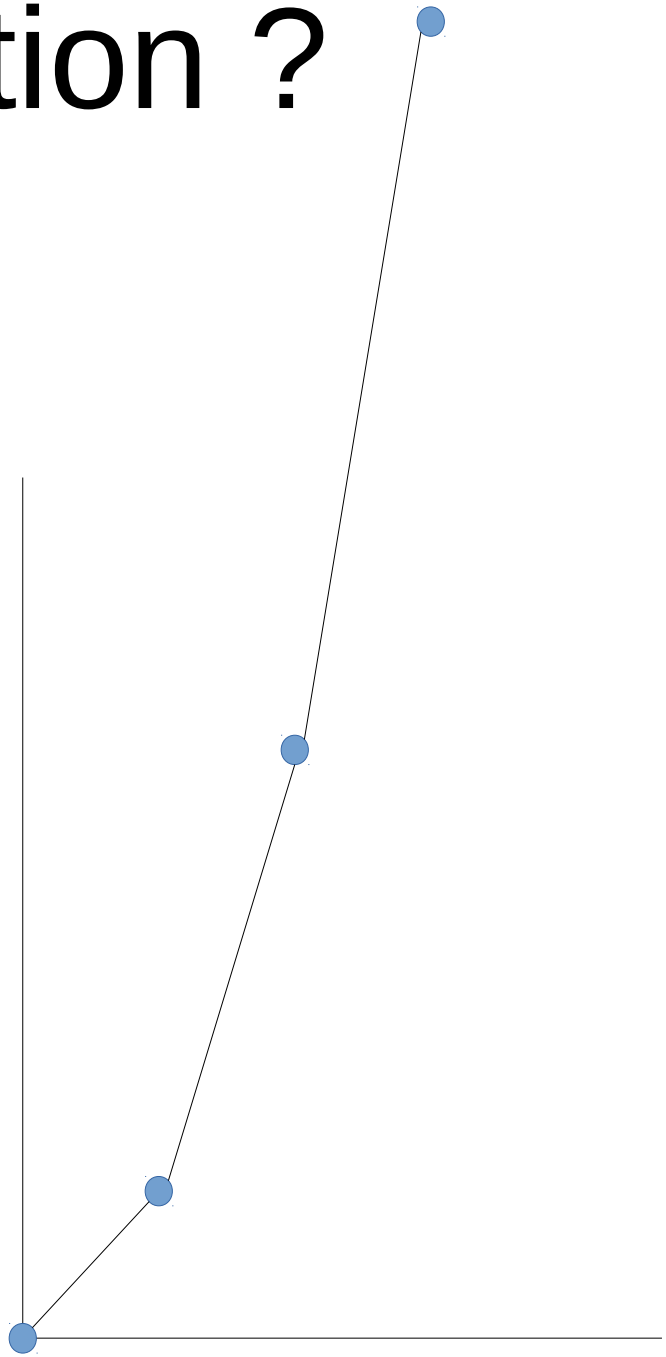
Définir une fonction ?

Maintenant si on définit partiellement la fonction ?

Définir une fonction ?

Maintenant si on définit partiellement la fonction ?

f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9

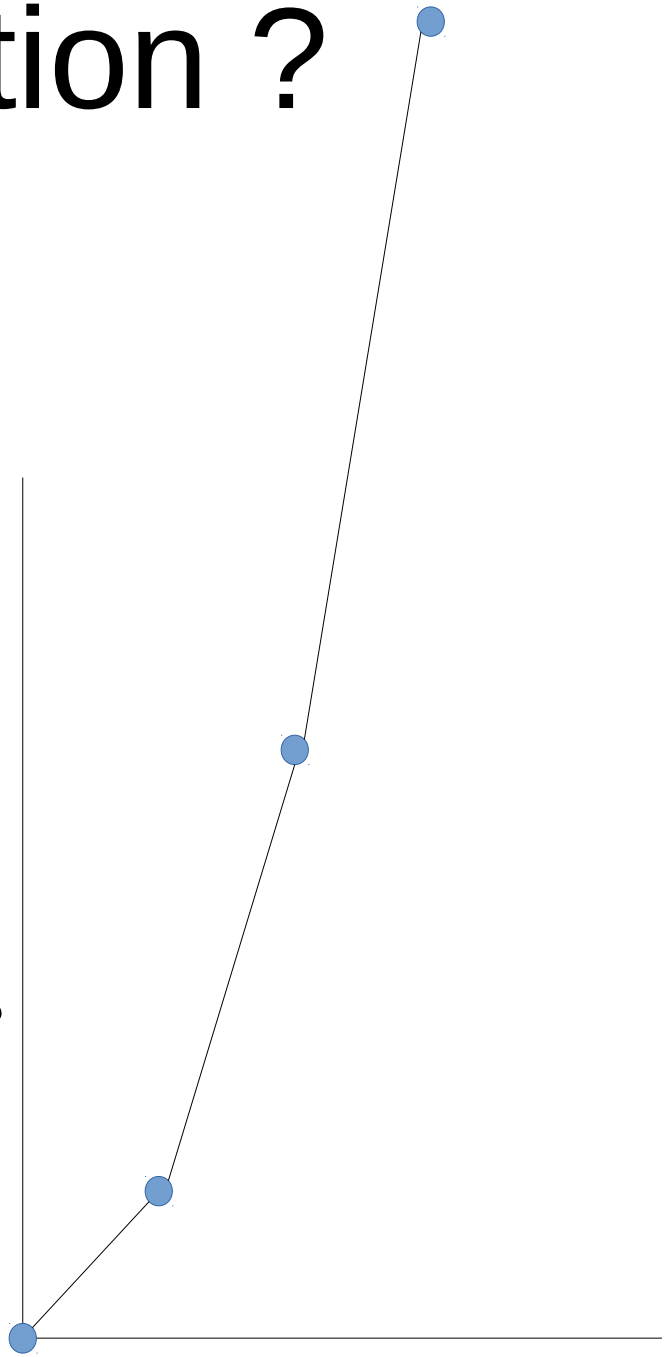


Définir une fonction ?

Maintenant si on définit partiellement la fonction ?

f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9

Y a-t-il une bonne raison de penser que f(x)=x*x ?



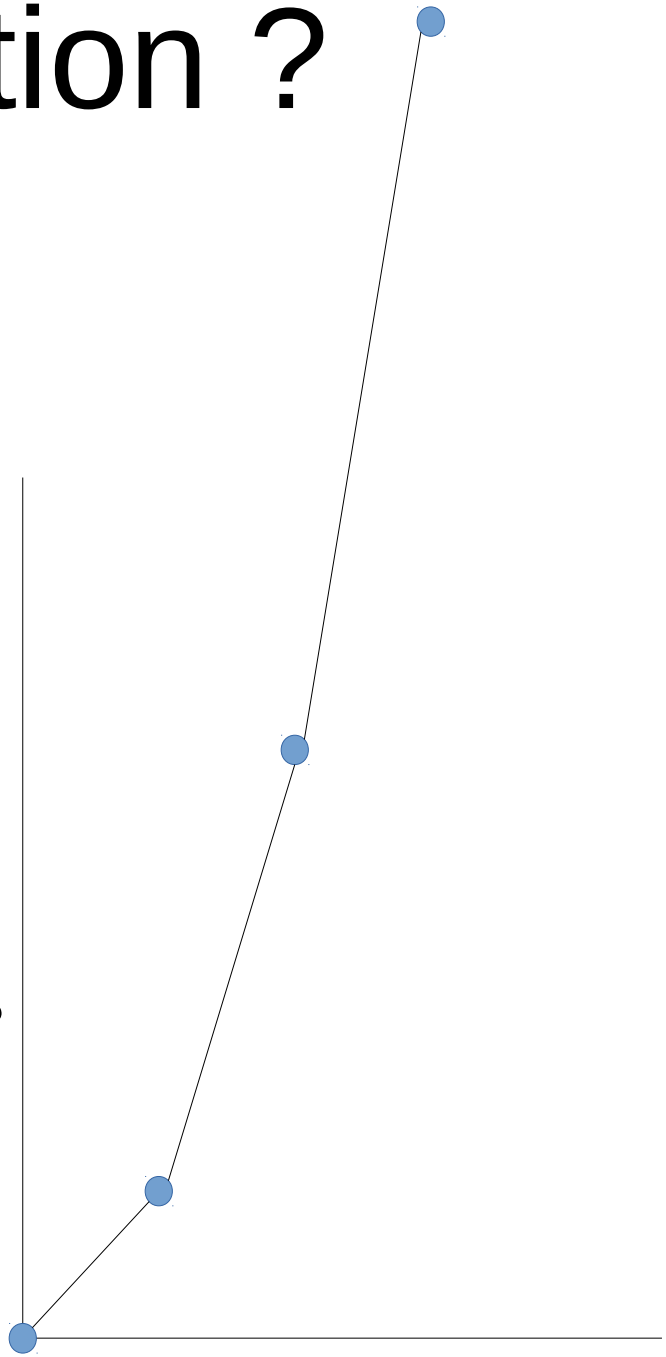
Définir une fonction ?

Maintenant si on définit partiellement la fonction ?

f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9

Y a-t-il une bonne raison de penser que f(x)=x*x ?

Oui et non …



Définir une fonction ?

f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9

Y a-t-il une bonne raison de penser que f(x)=x*x ?

Définir une fonction ?

f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9

Y a-t-il une bonne raison de penser que f(x)=x*x ?

Oui car

$$P\left(\text{erreur}_{\text{reelle}} \leq \text{erreur}_{\text{empirique}} + \sqrt{\frac{-\log(\delta)}{2K}}\right) \geq 1 - \delta$$

Définir une fonction ?

f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9

Y a-t-il une bonne raison de penser que f(x)=x*x ?

Oui car

$$P\left(\text{erreur}_{\text{reelle}} \leq \text{erreur}_{\text{empirique}} + \sqrt{\frac{-\log(\delta)}{2K}}\right) \geq 1 - \delta$$

Mais non car

Le No Free lunch theorem prouve que
si on considère l'ensemble des problèmes, alors toutes les fonctions sont aussi mauvaises.

Apprentissage par ordinateur

définir des fonctions qu’on ne sait pas définir autrement
(mais qu’on sait évaluer)
par des exemples.

Apprentissage par ordinateur

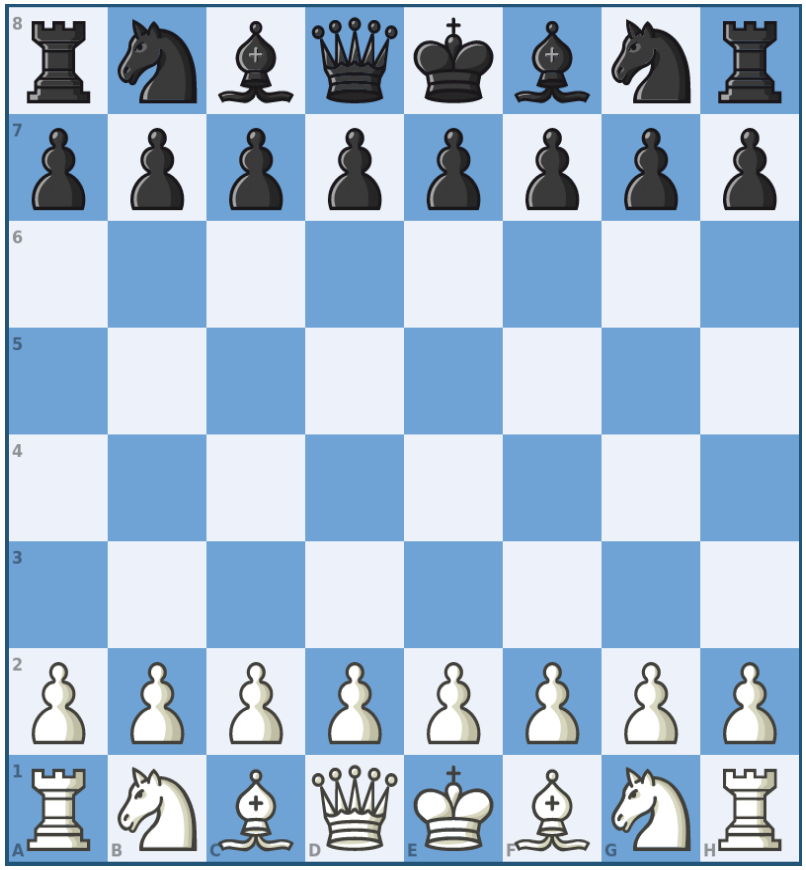
définir des fonctions qu'on ne sait pas définir autrement
(mais qu'on sait évaluer)
par des exemples.

Apprentissage vs Intelligence







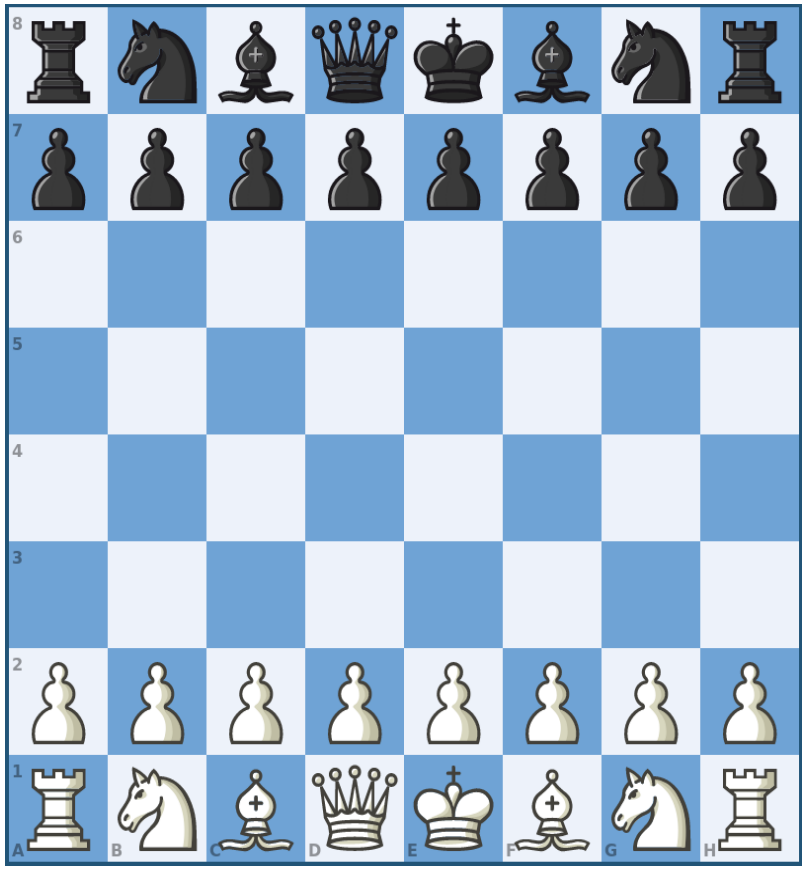


Apprentissage vs Intelligence









Jouer aux échecs c’est « dur » mais c’est quelque chose de cadré, formel, bien défini.

Expliquer pourquoi une image est une image de chats, c’est juste « impossible ».

Apprentissage par ordinateur

Le No Free lunch theorem prouve que
si on considère l’ensemble des problèmes, alors toutes les fonctions sont aussi mauvaises.

Apprentissage par ordinateur

Le No Free lunch theorem prouve que
si on considère l'ensemble des problèmes, alors toutes les fonctions sont aussi mauvaises.

→ Mais, l'ensemble des problèmes « réels » ne contient pas tous les problèmes !

Apprentissage par ordinateur

Le No Free lunch theorem prouve que
si on considère l'ensemble des problèmes, alors toutes les fonctions sont aussi mauvaises.

→ Mais, l'ensemble des problèmes « réels » ne contient pas tous les problèmes !

Il y a des méthodes d'apprentissages qui marchent mieux que d'autres sur les problèmes « réels ».

Mais on ne le découvre qu'en essayant !

Apprentissage par ordinateur

Le No Free lunch theorem prouve que
si on considère l'ensemble des problèmes, alors toutes les fonctions sont aussi mauvaises.

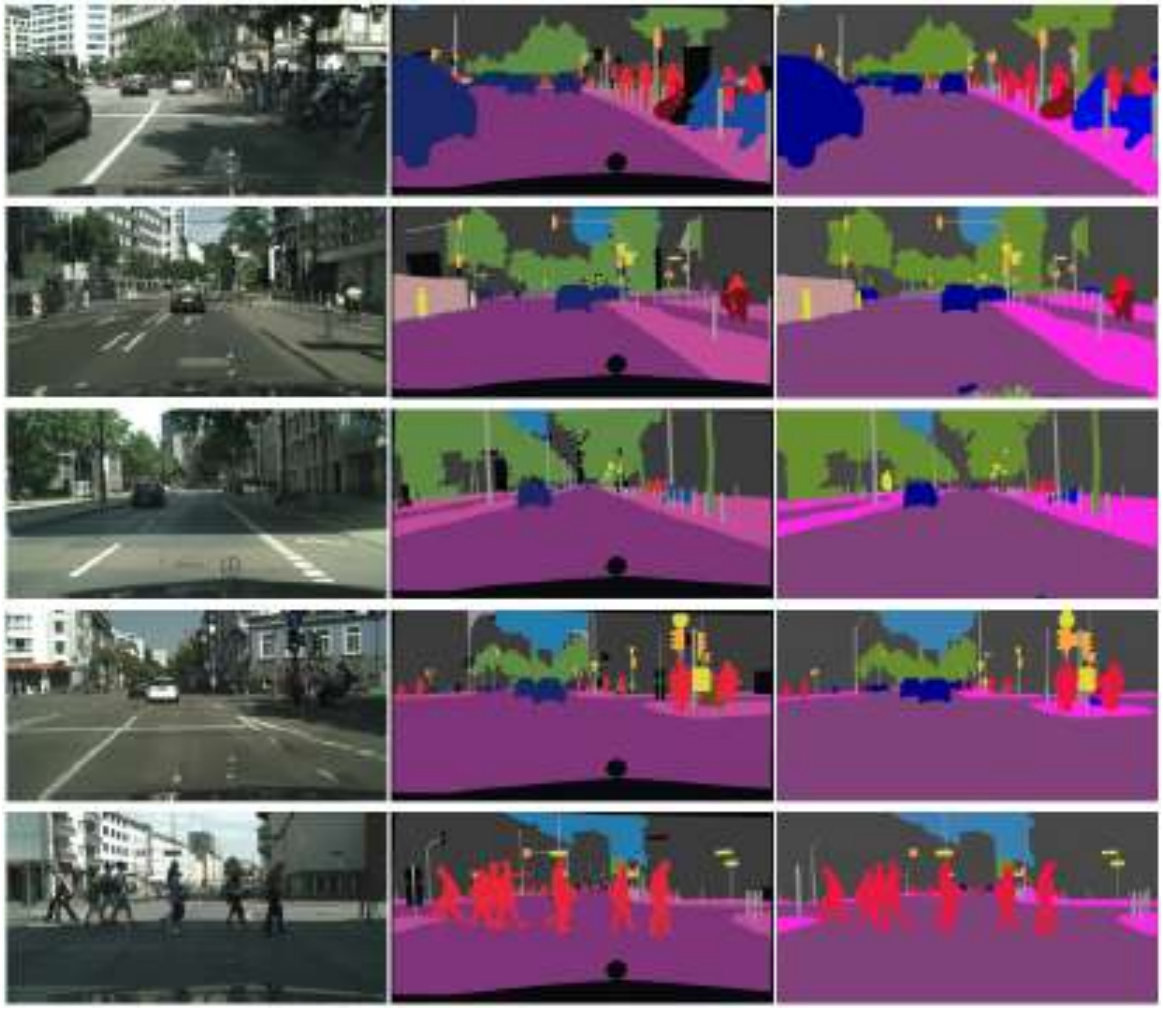
→ Mais, l'ensemble des problèmes « réels » ne contient pas tous les problèmes !

Il y a des méthodes d'apprentissages qui marchent mieux que d'autres sur les problèmes « réels ».

Mais on ne le découvre qu'en essayant !

La méthode qui aujourd'hui marche le mieux sur les problèmes structurées :
c'est le deeplearning.

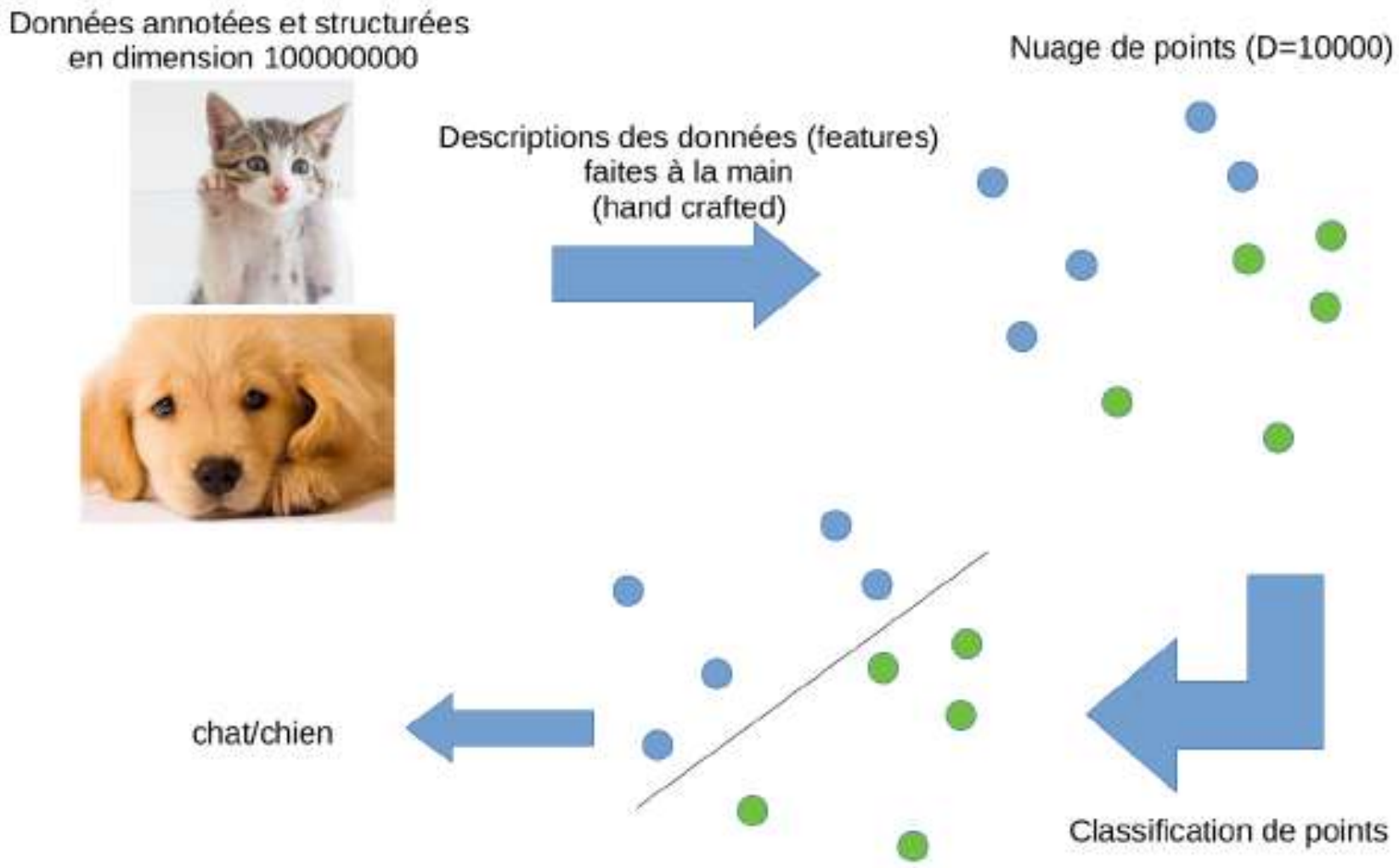
Introduction



Introduction

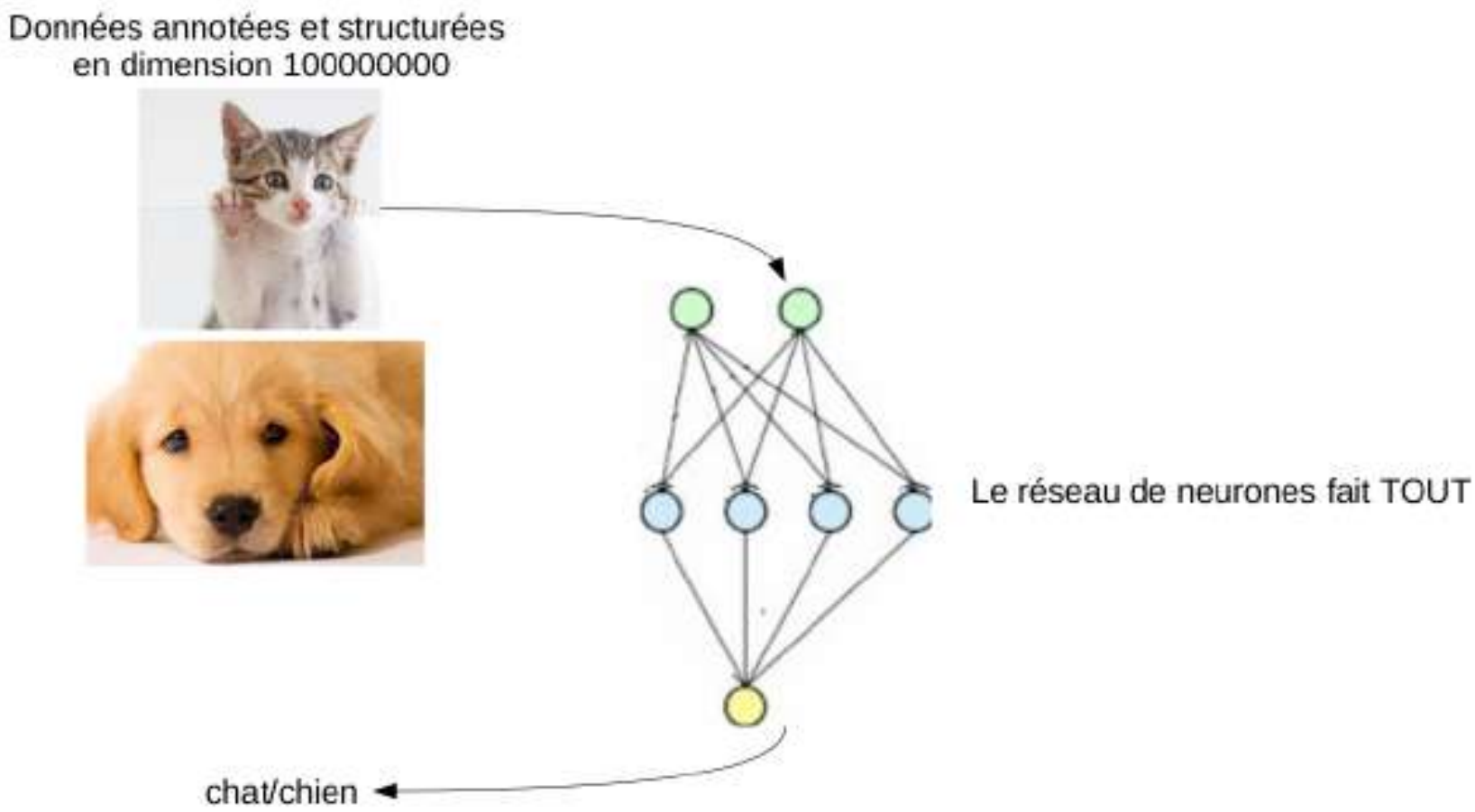
Introduction

Avant le deep learning



Introduction

Après



Introduction

Important

Le deep learning n'est pas vraiment supérieur à d'autres algorithmes (forêt de décision, XGboost ...) sur la classification de **points**.

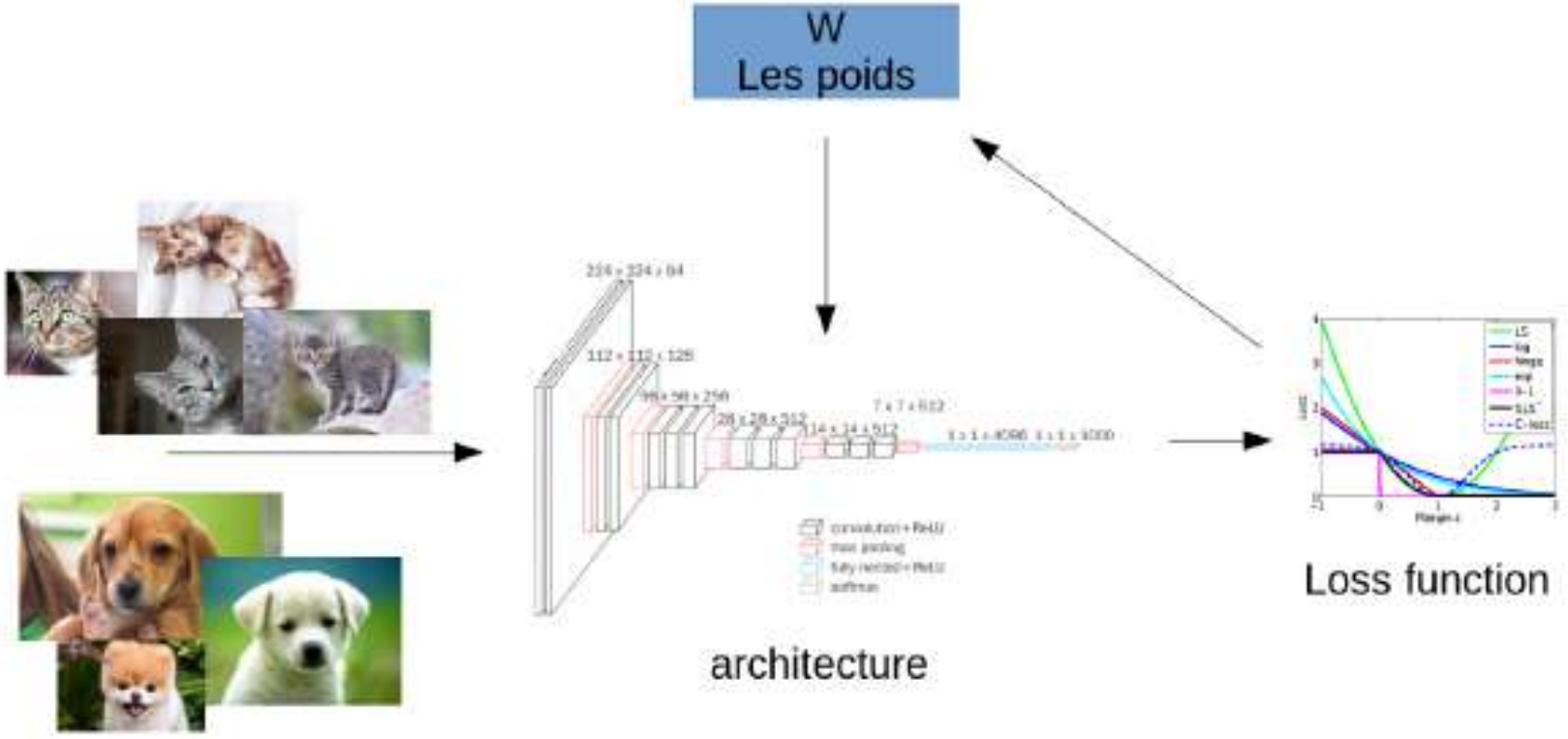
Le deep learning est par contre incontournable pour le traitement de données structurées (son, image, vidéo, texte, signal 1D...)

Plan

- Classification de points
  - Apprentissage et test
  - Neurone et réseau
  - Universalité et erreur de généralisation
- Double descente
  - Compromis simplicité/complexité
  - Méthodes par ensemble, double descente
  - Descente de gradient Stochastique
- Traitement de données structurées
  - Neurones convolutif
  - Détection, segmentation, génération...
  - Problème de stabilité

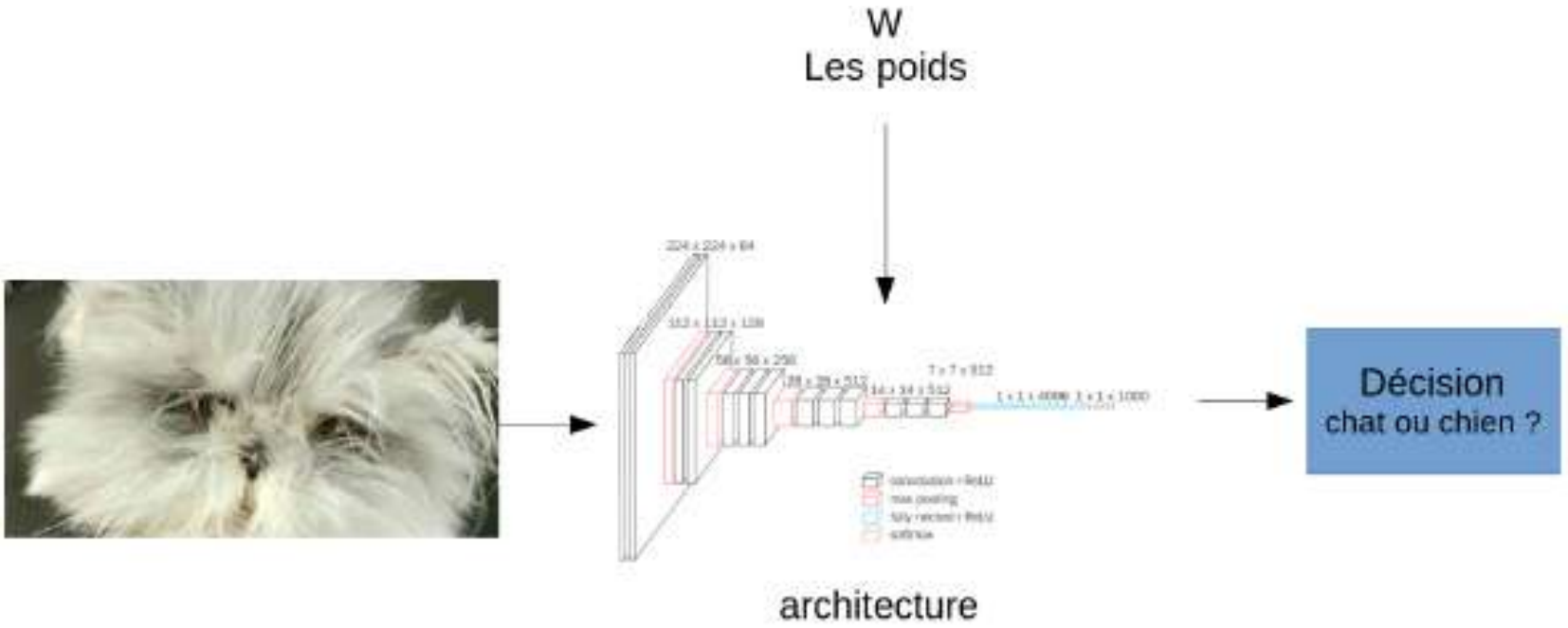
2 phases

Apprentissage



2 phases

Test et/ou production et/ou inférence



2 phases

Important

Il est essentiel de bien avoir en tête le problème global

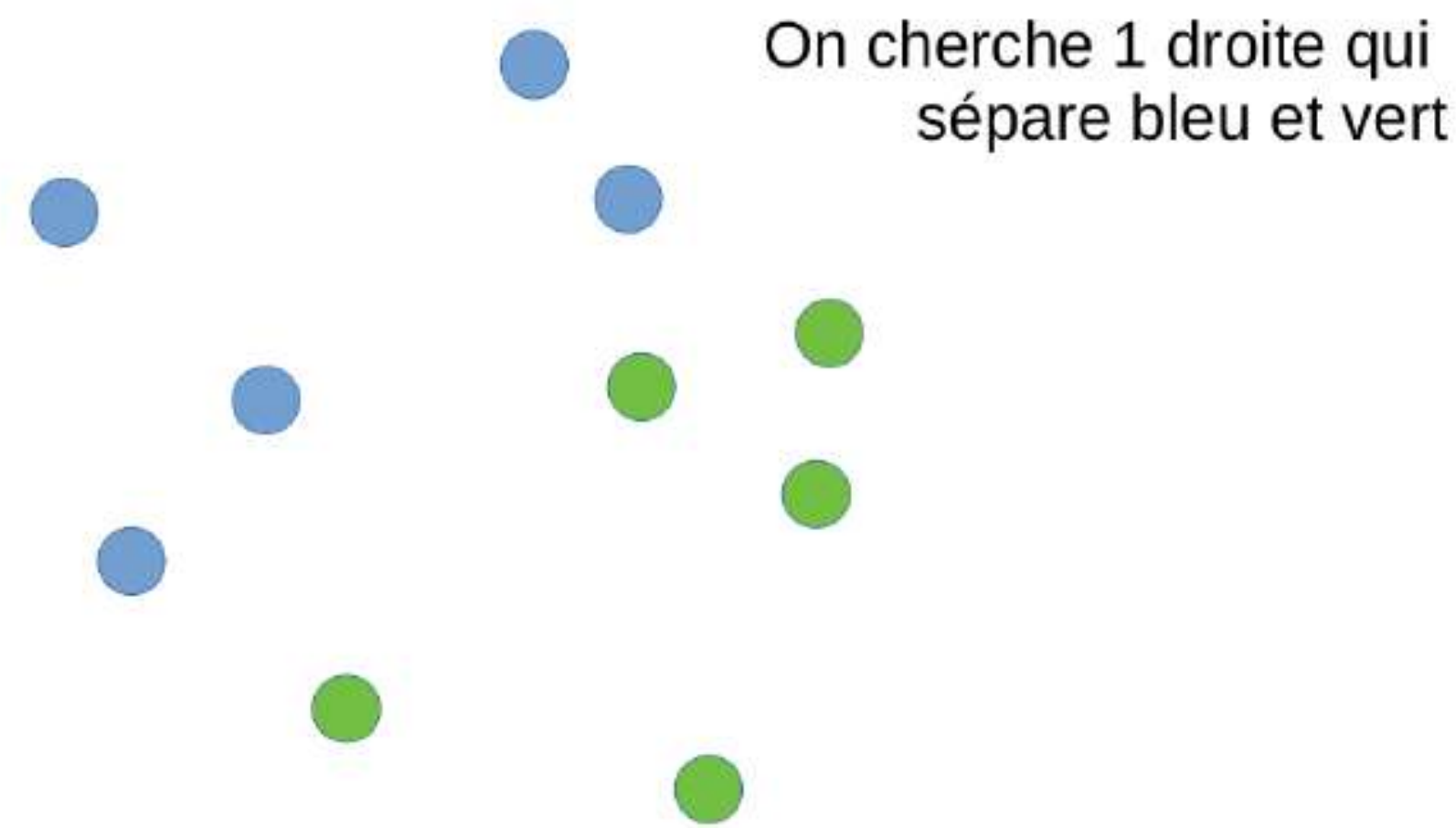
L'optimisation des poids w est un problème difficile.

Mais il n'a de sens que dans l'ensemble : choix d'une architecture, chois d'une fonction de coût, et objectif de performance en test.

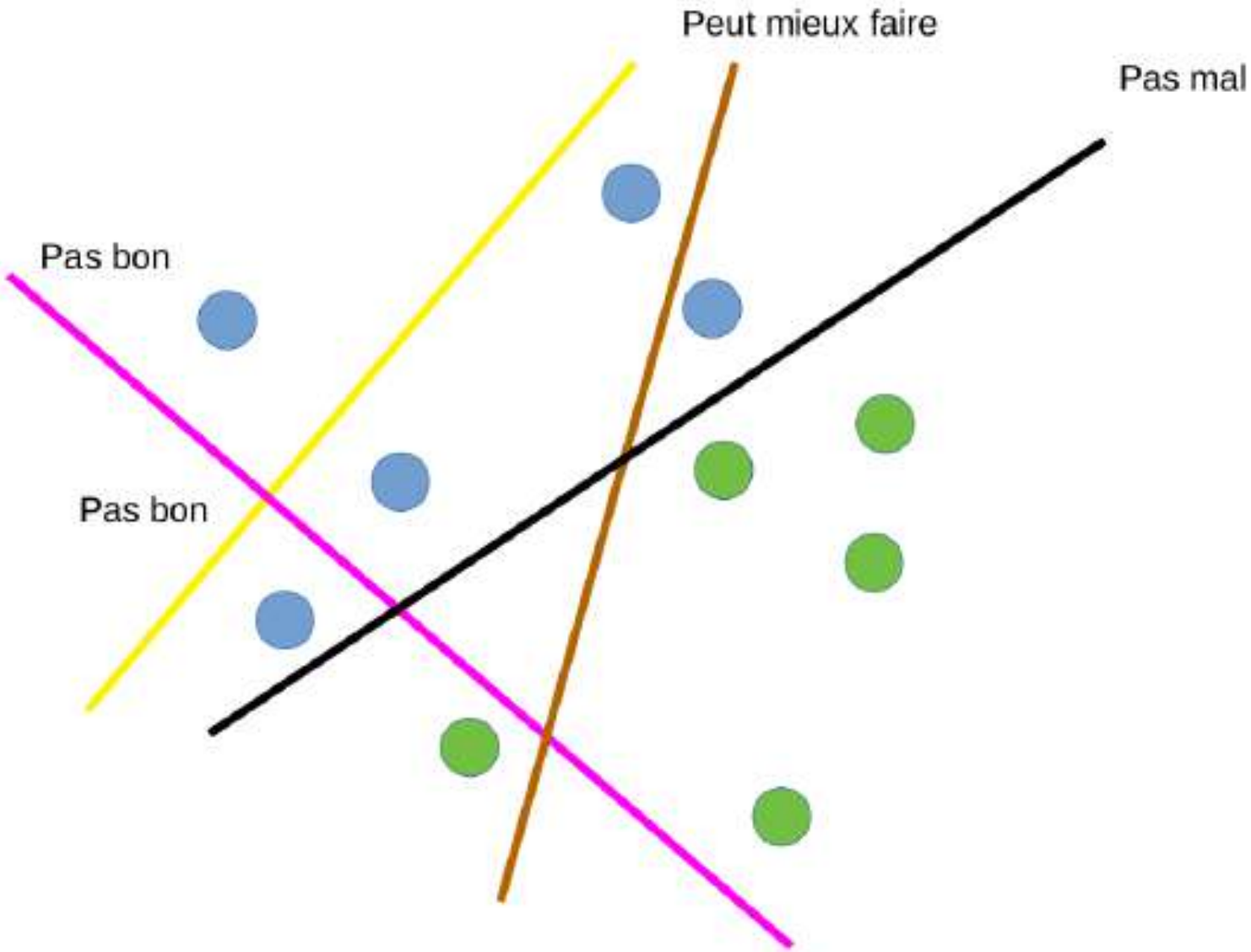
Plan

- Classification de points
  - Apprentissage et test
  - Neurone et réseau
  - Universalité et erreur de généralisation
- Double descente
  - Compromis simplicité/complexité
  - Méthodes par ensemble, double descente
  - Descente de gradient Stochastique
- Traitement de données structurées
  - Neurones convolutif
  - Détection, segmentation, génération...
  - Problème de stabilité

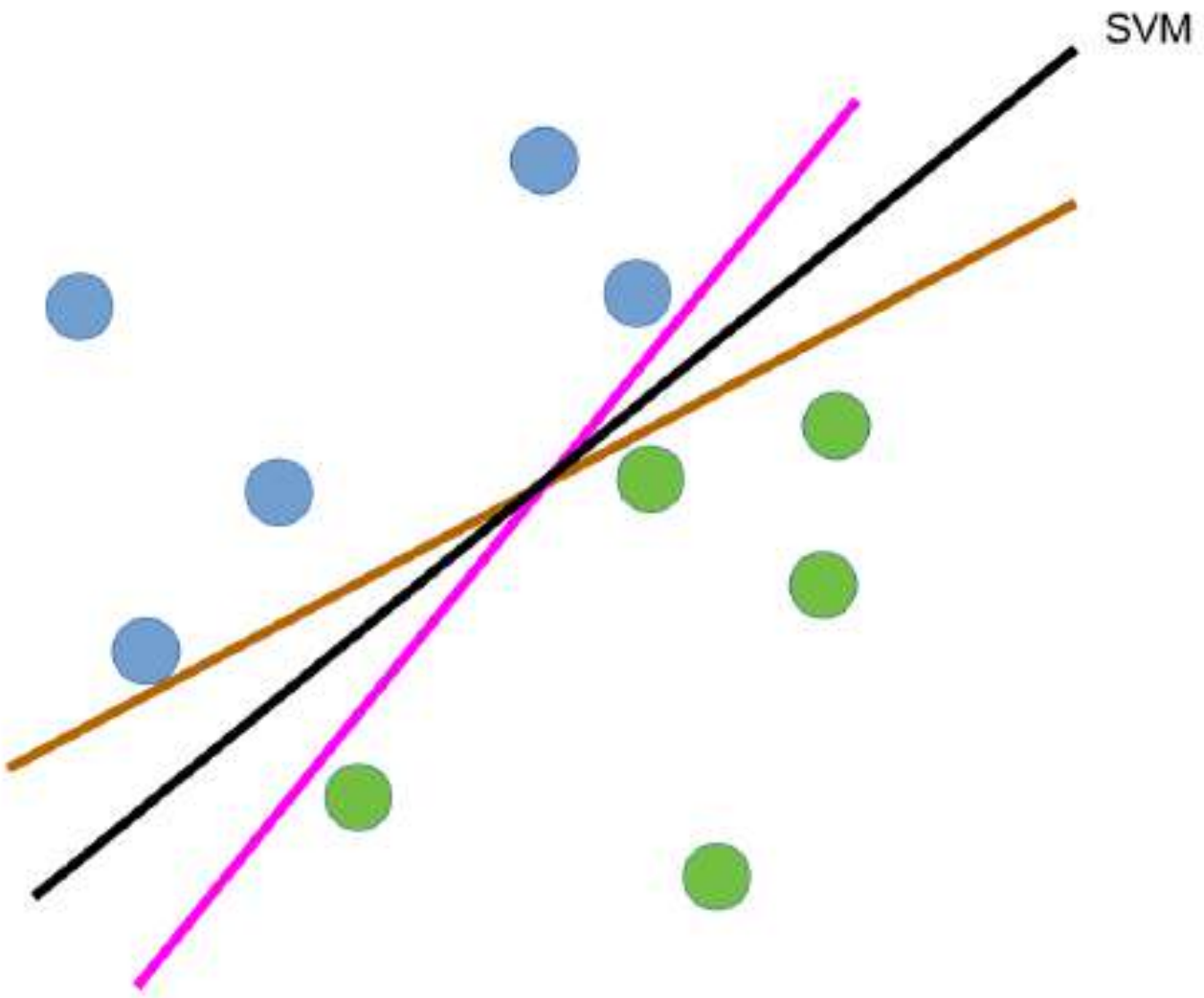
Apprentissage d'une séparation hyperplane



Apprentissage d'une séparation hyperplane



Apprentissage d'une séparation hyperplane



Apprentissage d'une séparation hyperplane

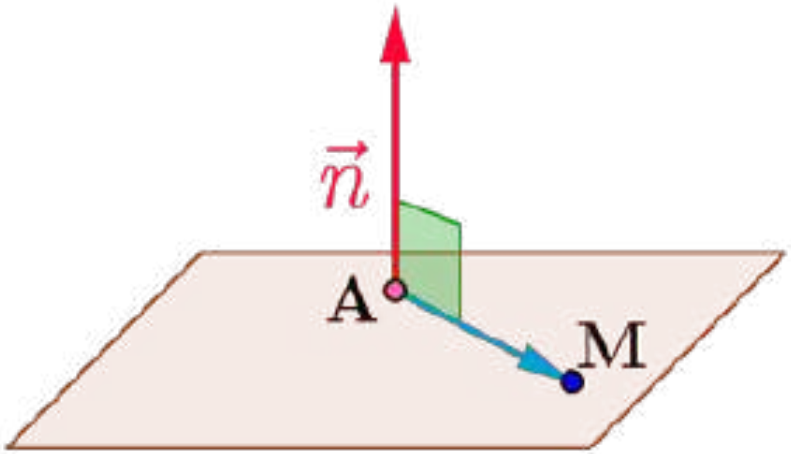
Qu'est ce que ça donne avec des maths ?

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.

Apprentissage d'une séparation hyperplane

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.

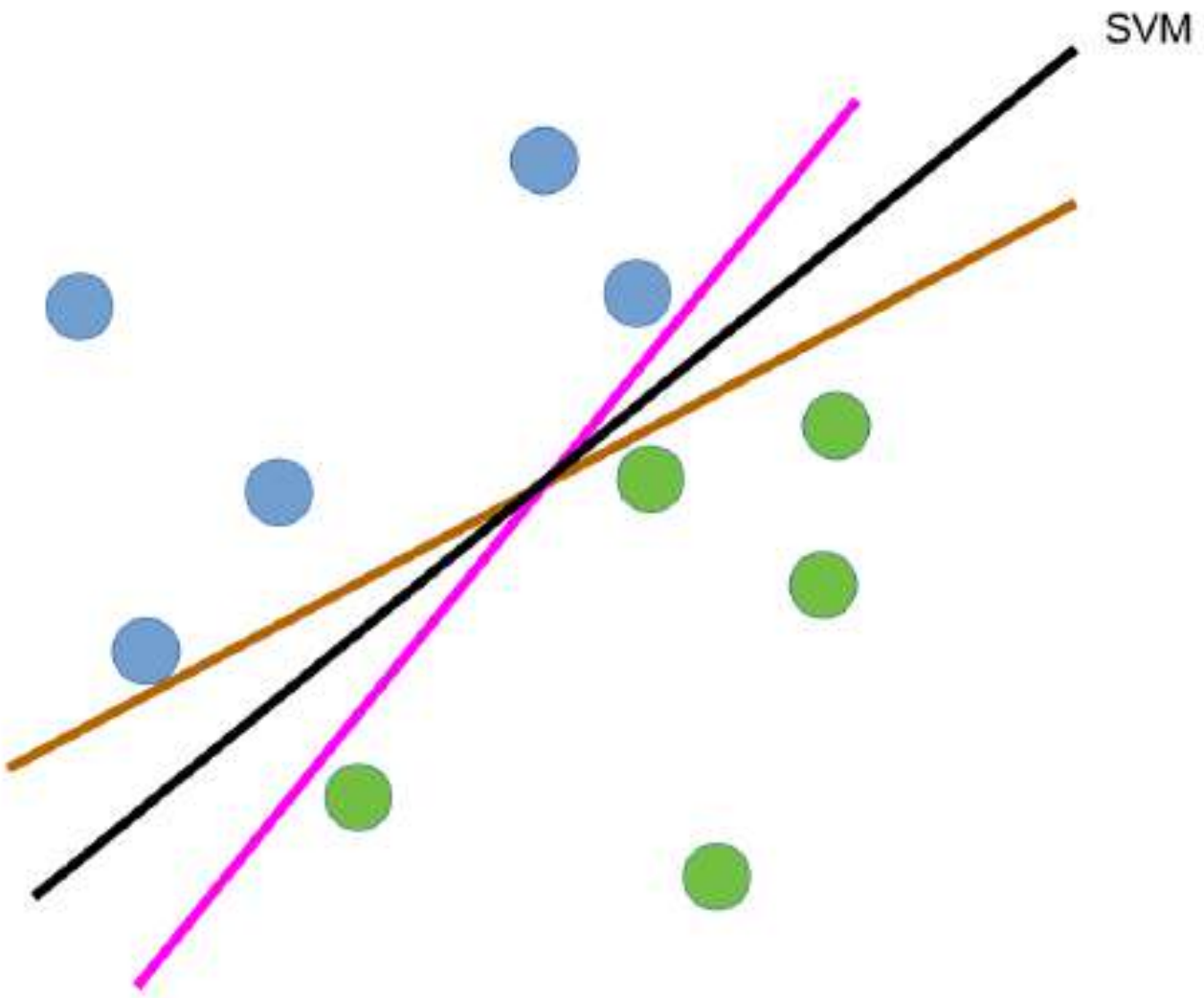
Apprentissage d'une séparation hyperplane

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

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

Apprentissage d'une séparation hyperplane



Apprentissage d'une séparation hyperplane

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

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

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

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

Apprentissage d'une séparation hyperplane

SVM

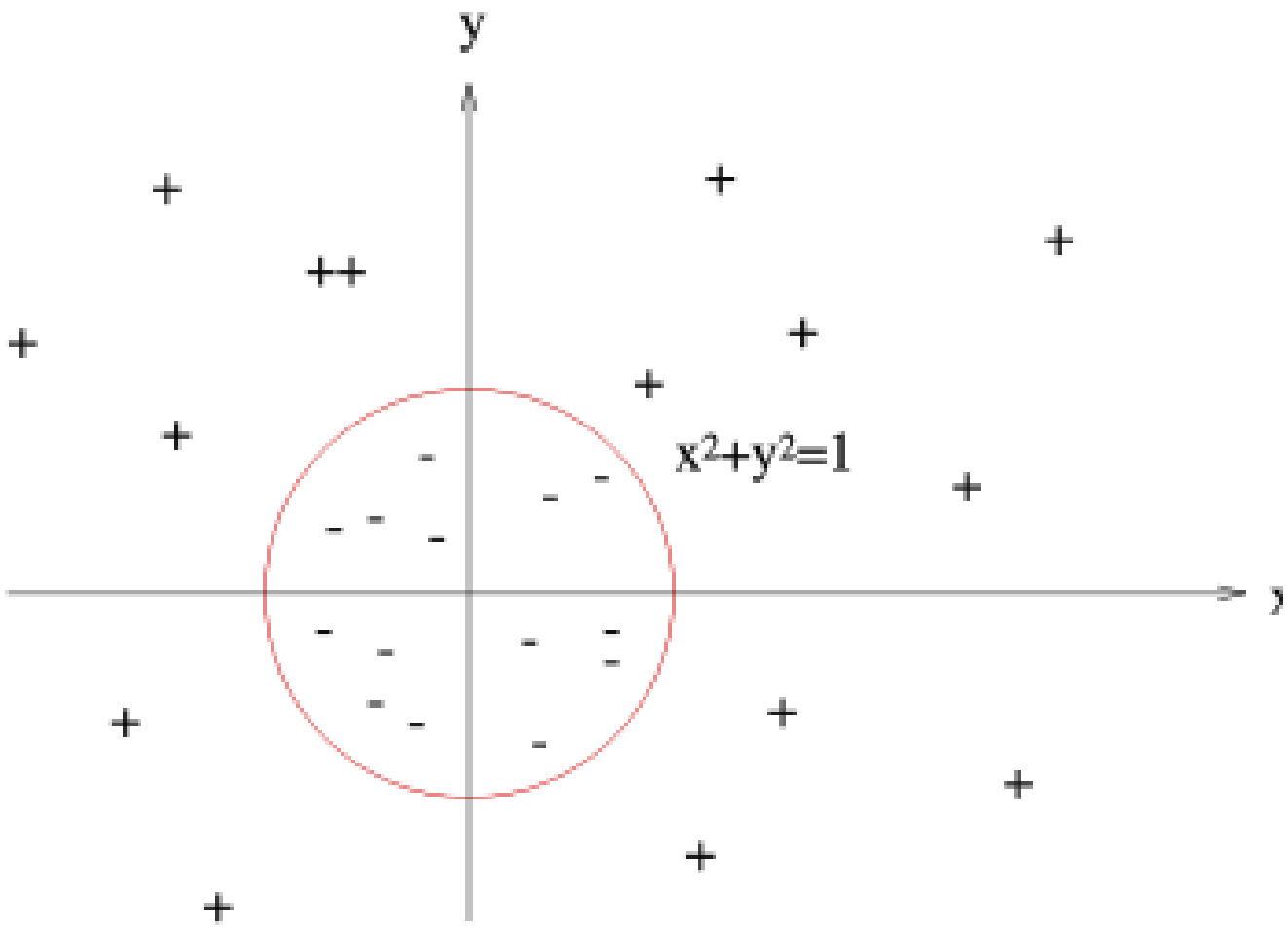
On a des points $x_1, ..., x_N \in \mathbb{R}^D$ avec une couleur (bleu ou pas bleu) $(y_1, ..., y_N) \in \{-1, 1\}^N$.
Et le plan w, b qui sépare les points en étant le plus distant possible

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

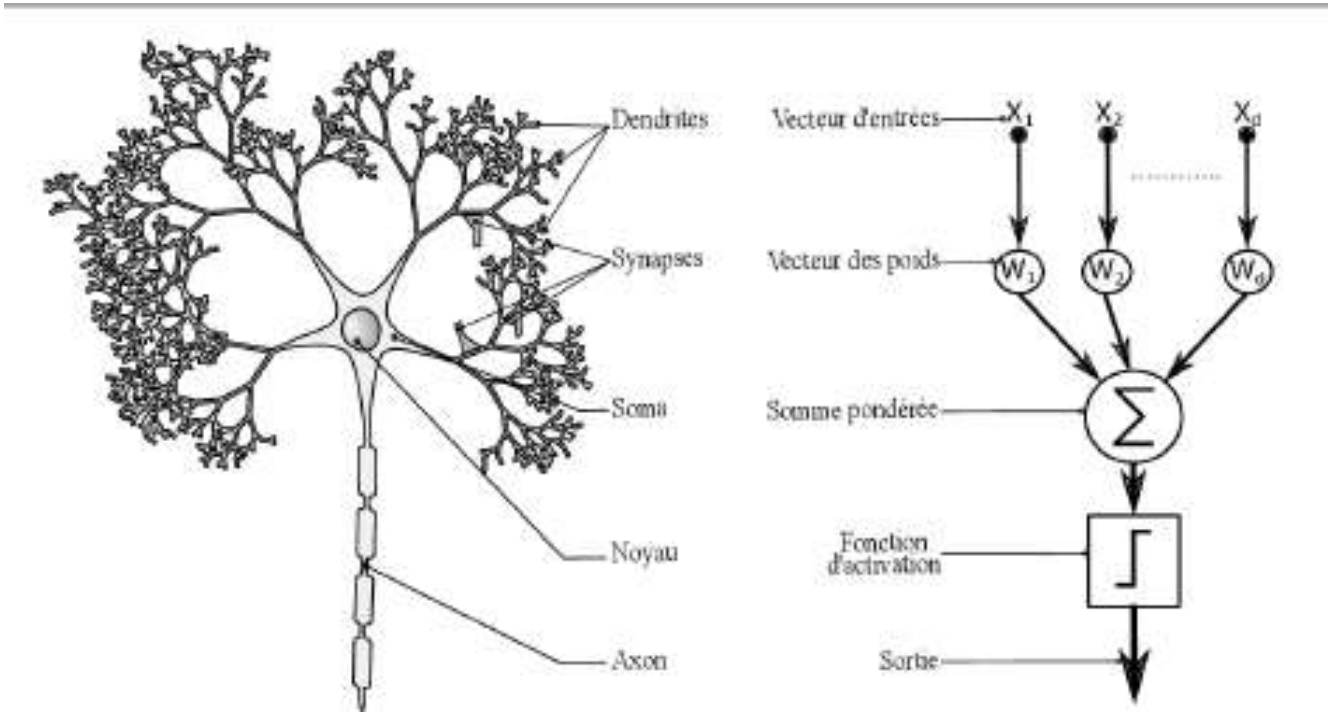
$$sc : \forall n \in \{1, ..., M\},\ y_n(w^T x_n + b) \geq 1$$

Problème

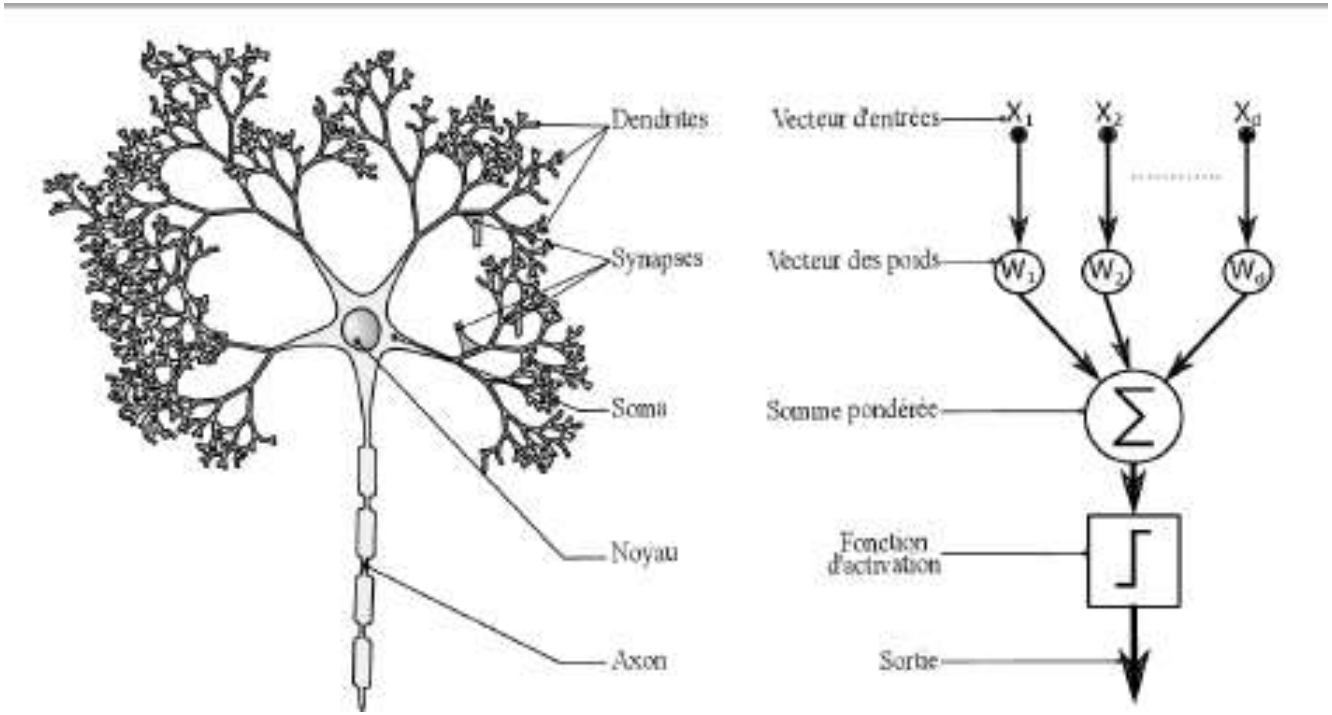
Parfois on ne peut PAS séparer les données avec un hyperplan



Le modèle du neurone



Le modèle du neurone



1 neurone c'est comme une frontière hyperplane (ex SVM)

Du neurone au réseau

Le neurone

Dans les architectures de réseaux de neurones (profond ou pas), le neurone est un filtre linéaire :

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

$\alpha \in \mathbb{R}^{\phi}$ 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 ϕ indépendante de D.

La couche de neurone

Une couche de ψ de neurones est une séquence de ψ neurones prenant la même entrée, et, dont les ψ 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 ψ neurones.

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

Le réseau de neurone

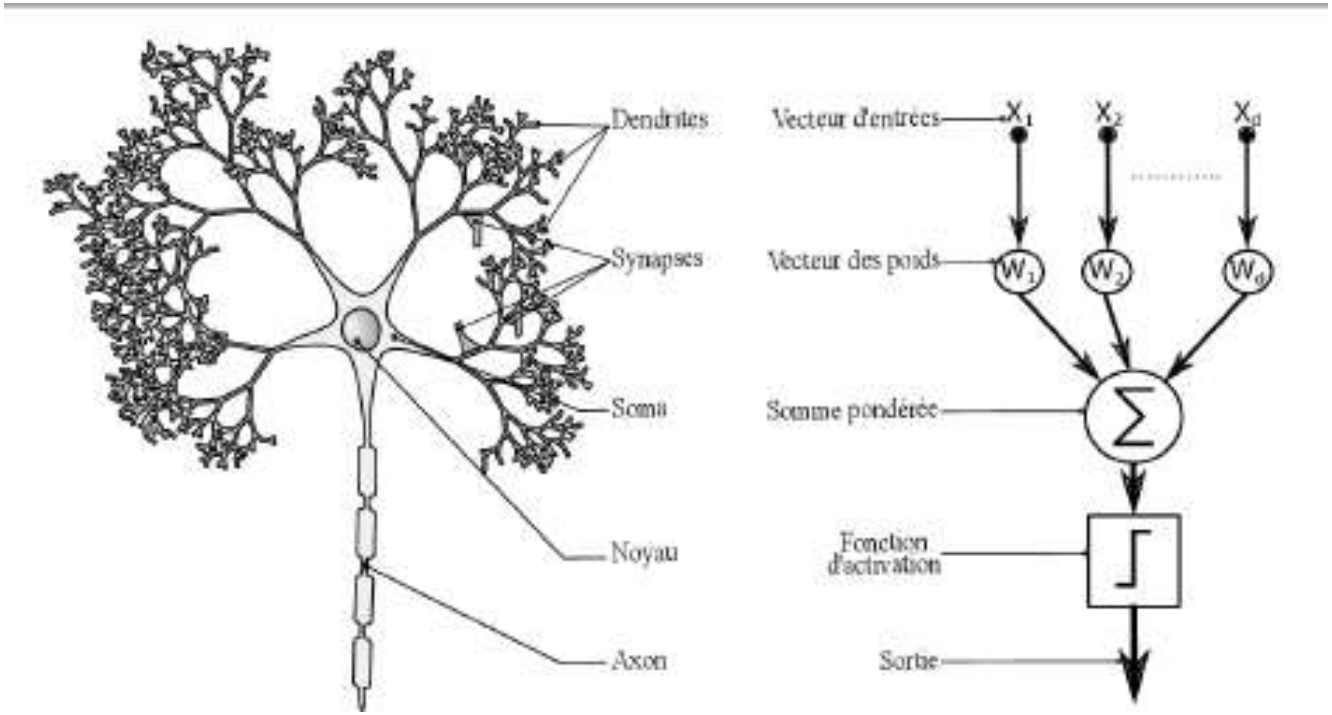
Si on empile 2 couches de neurones c'est exactement comme s'il y en avait qu'une :

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

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

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

Le modèle du neurone



1 neurone c'est comme 1 frontière hyperplane

Mais 2 neurones connecté c'est différent à cause de la fonction d'activation !

Le réseau de neurone

On empile 2 couches de neurones AVEC activation :

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

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 MinMax (voir partie 3).

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

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

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

Apprentissage

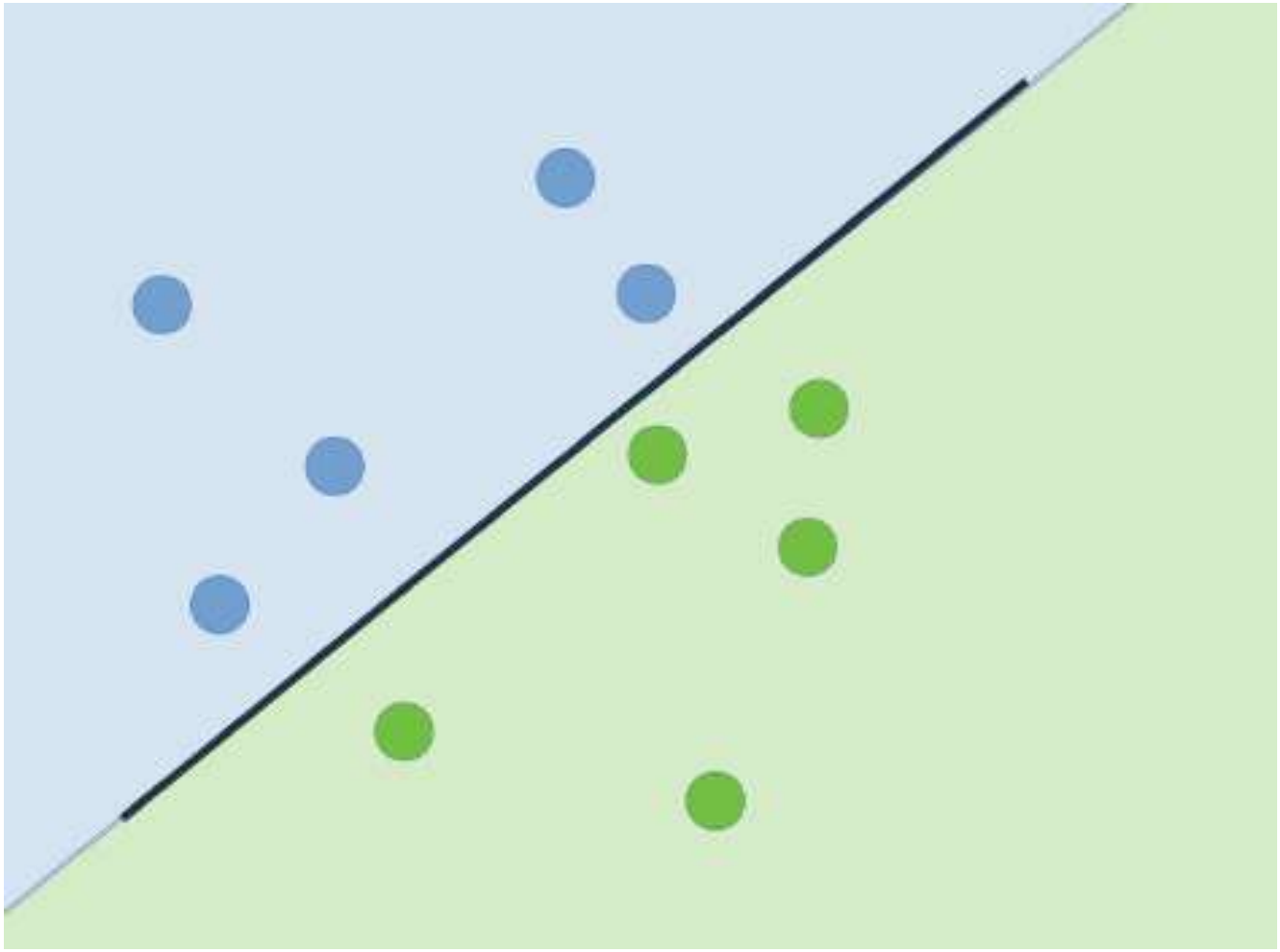
- On a des points colorés qu'on veut séparer
- Avec 1 neurone
  - Ça revient à cherche un hyperplan séparateur
  - $f_w(x) = w^T x + w_{biais}$
  - Mais il peut ne pas exister de séparation
- Avec un réseau de neurones
  - $f_w(x) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$
  - Avec $O(ND)$ neurones et 3 couches, on peut tout apprendre !

MAIS

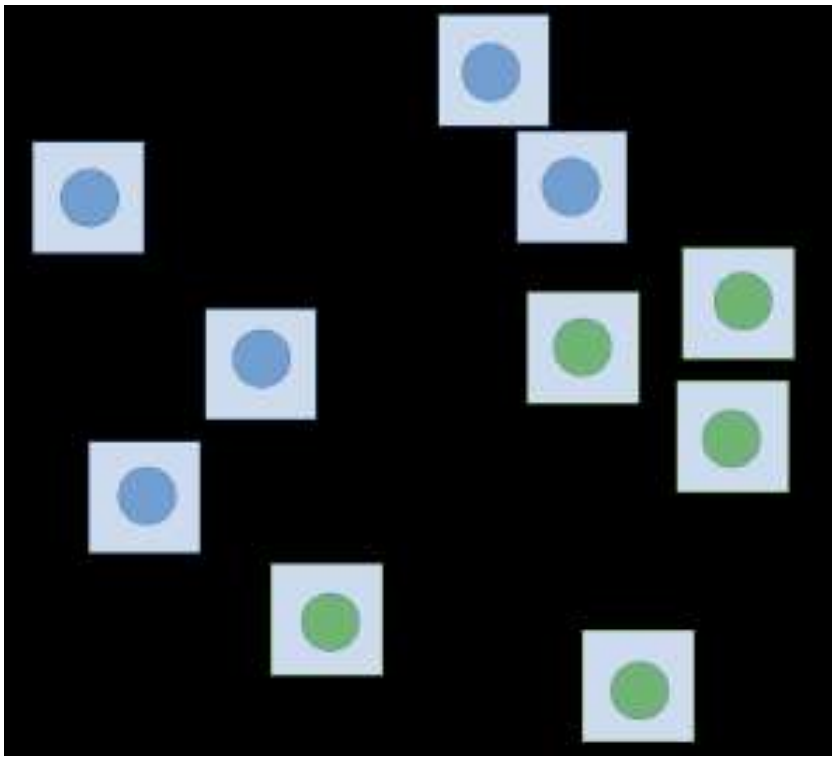
- On a des points colorés qu'on veut séparer
- Avec 1 neurone
  - Ça revient à cherche un hyperplan séparateur
  - $f_w(x) = w^T x + w_{biais}$
  - Mais il peut ne pas exister de séparation
- Avec un réseau de neurones
  - $f_w(x) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$
  - Avec $O(ND)$ neurones et 3 couches, on peut tout apprendre !

Attention : tout apprendre ce n'est pas nécessairement bien !

SVM



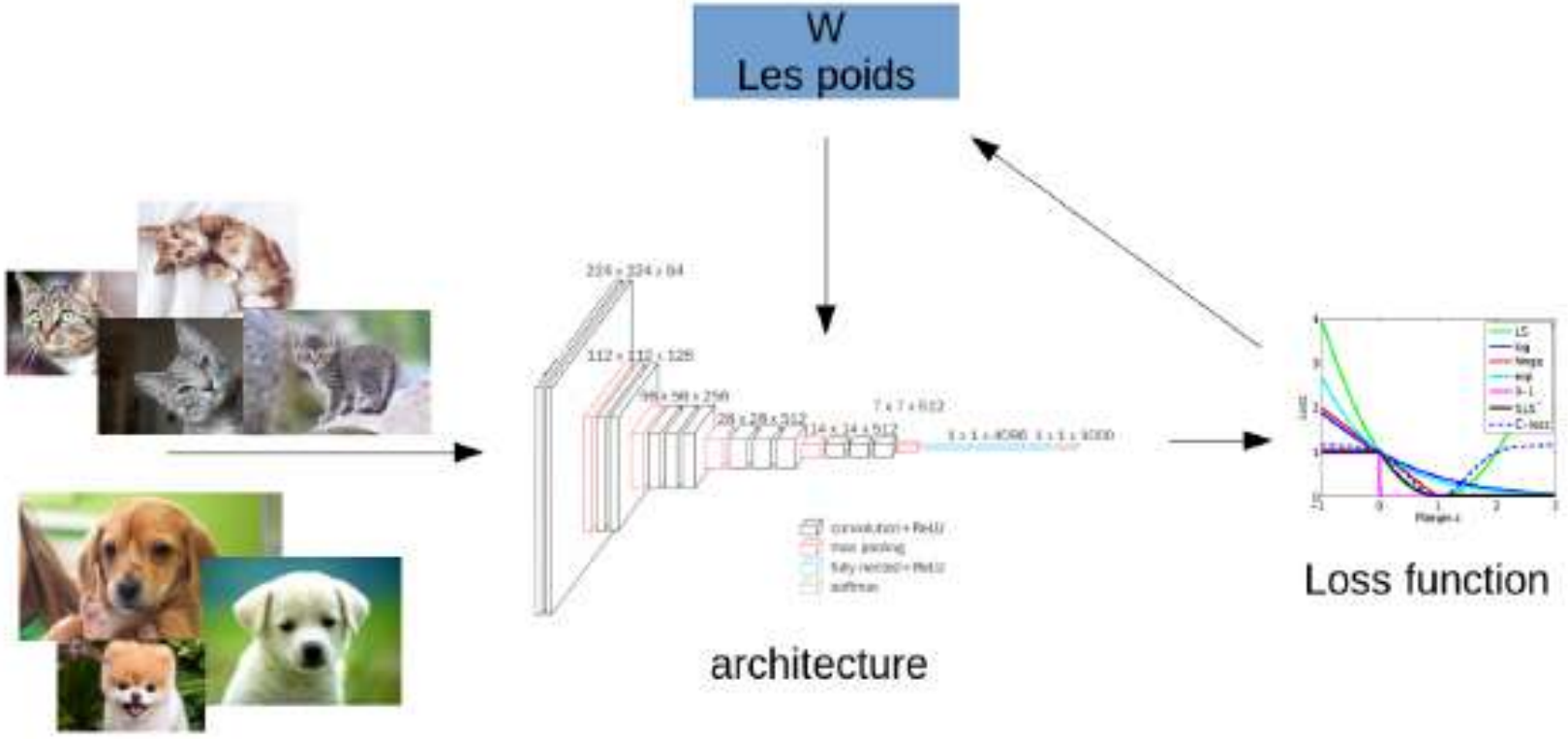
Théorème d'universalité



On ne prend aucune décision en dehors de la base d'apprentissage.
Le classifier est **inutile** : il n'a fait qu'encoder la base d'apprentissage, et, ne donne aucune information sur des points hors de cette base.

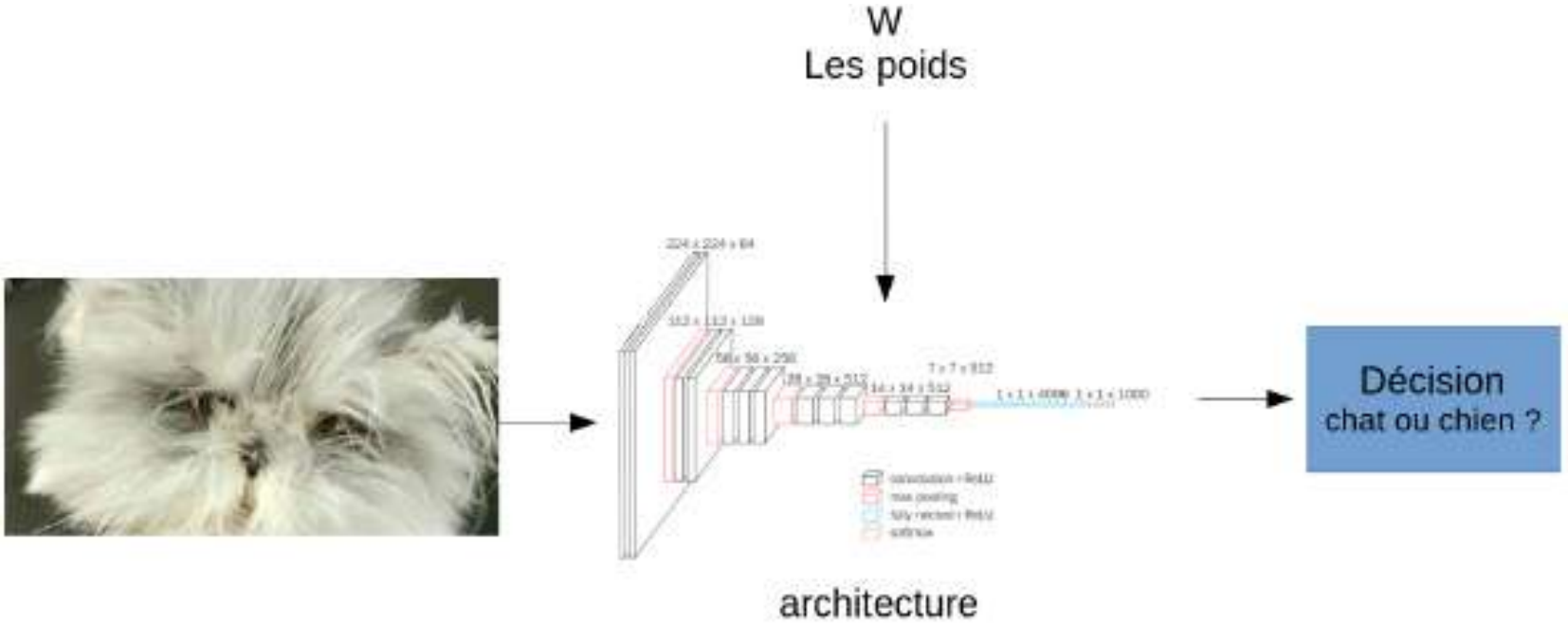
2 phases

Apprentissage



2 phases

Test et/ou production et/ou inférence



Ce qui compte c'est la performance sur les données de test !

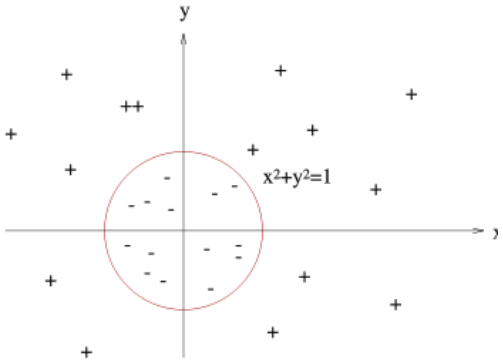
Plan

- Classification de points
  - Apprentissage et test
  - Neurone et réseau
  - Universalité et erreur de généralisation
- Double descente
  - Compromis simplicité/complexité
  - Méthodes par ensemble, double descente
  - Descente de gradient Stochastique
- Traitement de données structurées
  - Neurones convolutif
  - Détection, segmentation, génération...
  - Problème de stabilité

L'ancien paradigme

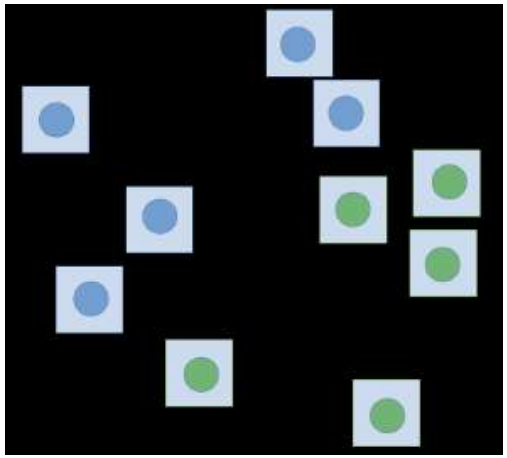
Vapnik Cherickov

- Pour avoir une *bonne* performance en test, il faut
- ni trop peu de *paramètres*



sinon on ne peut pas apprendre

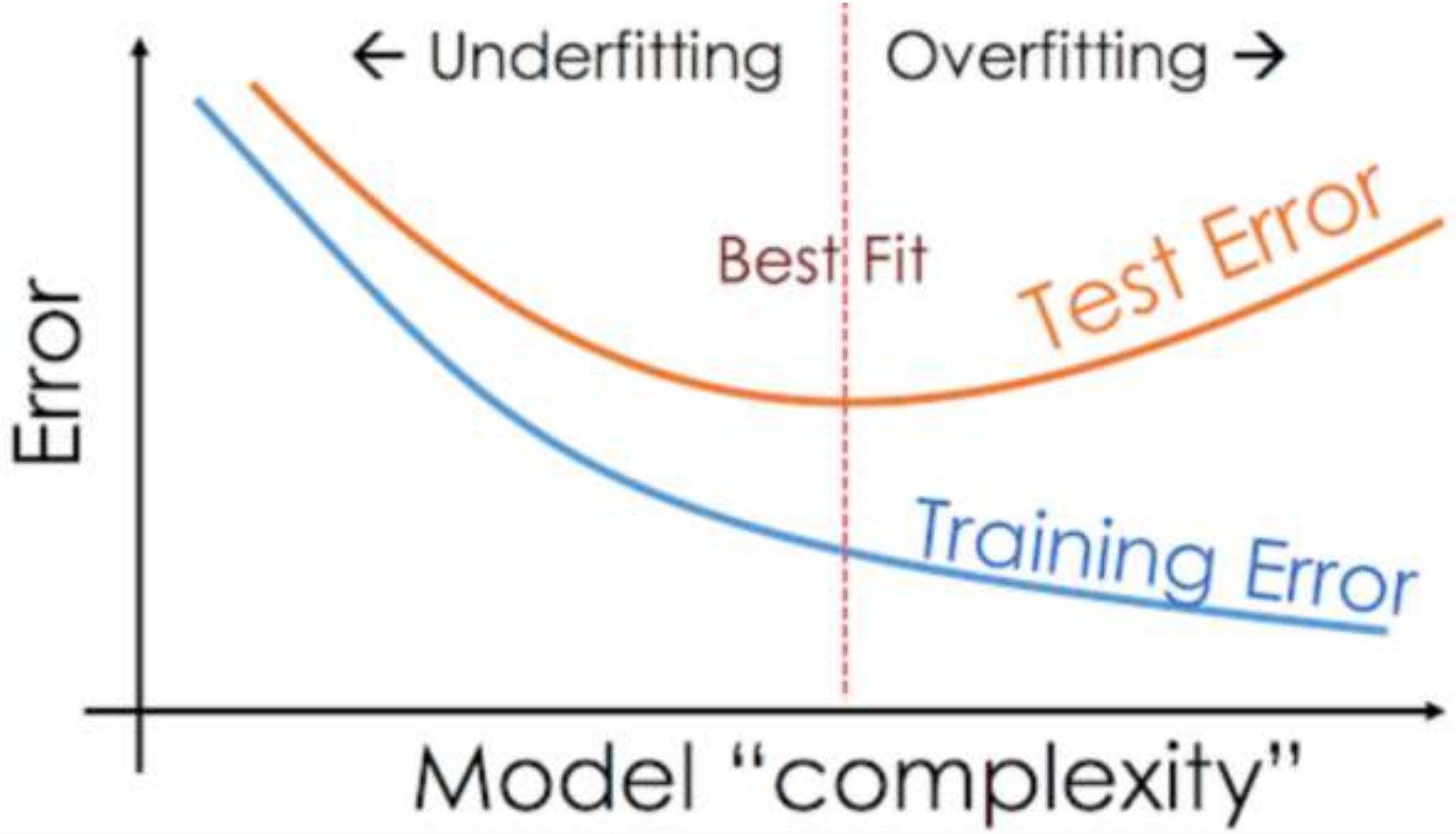
- ni trop de *paramètres*



sinon on apprend par coeur

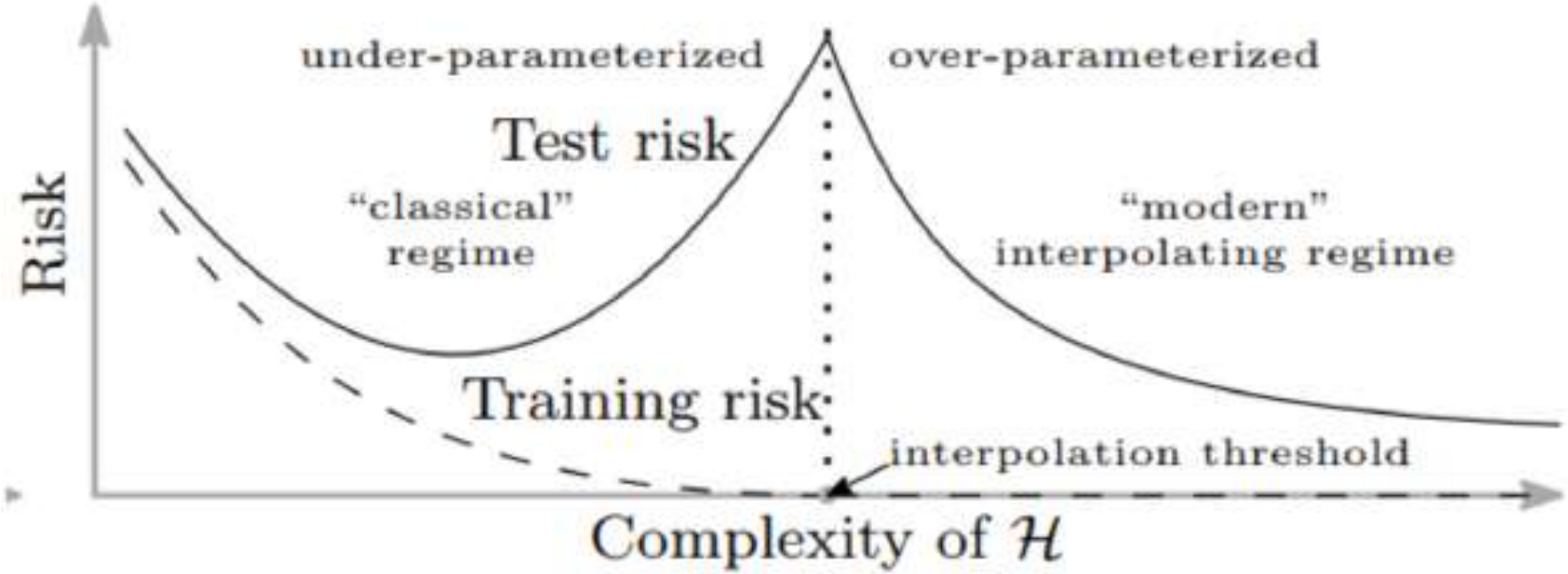
L'ancien paradigme

Vapnik Cherickov



Le NOUVEAU paradigme

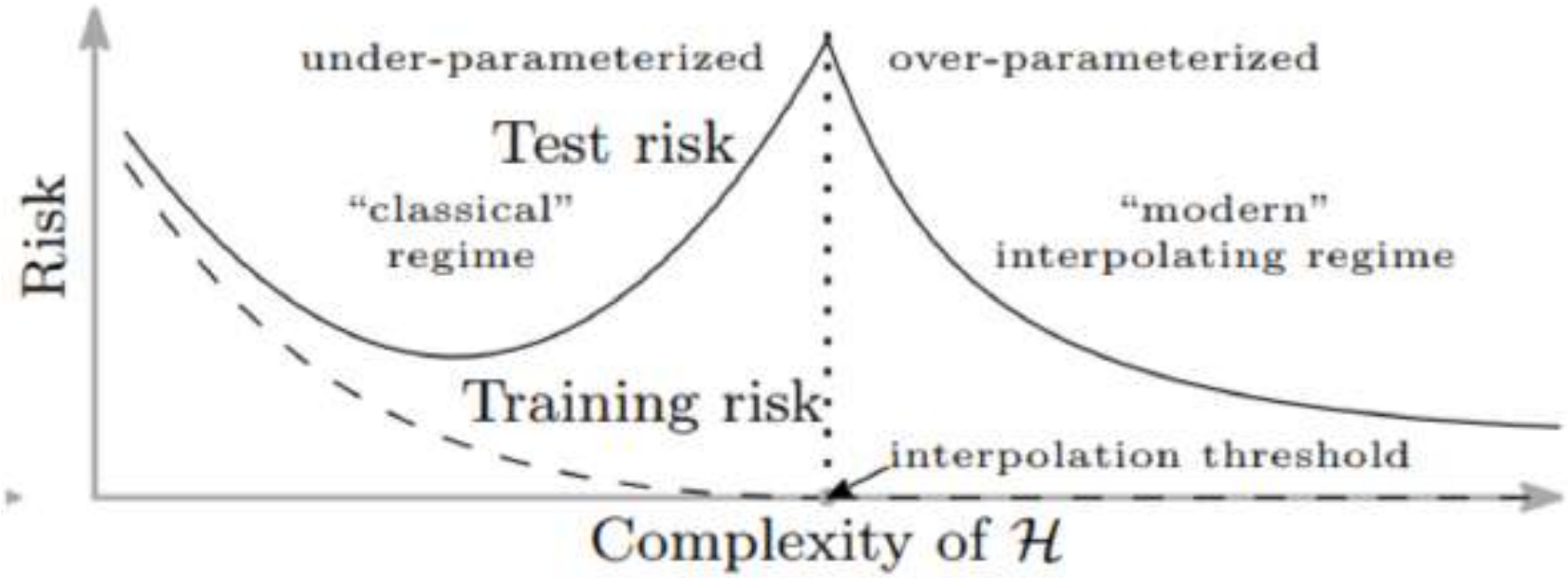
Double descente



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

Le NOUVEAU paradigme

Double descente



Toujours prendre le plus gros réseaux possible c'est mieux !

Une prime à la puissance plutôt qu'à l'intelligence :-(

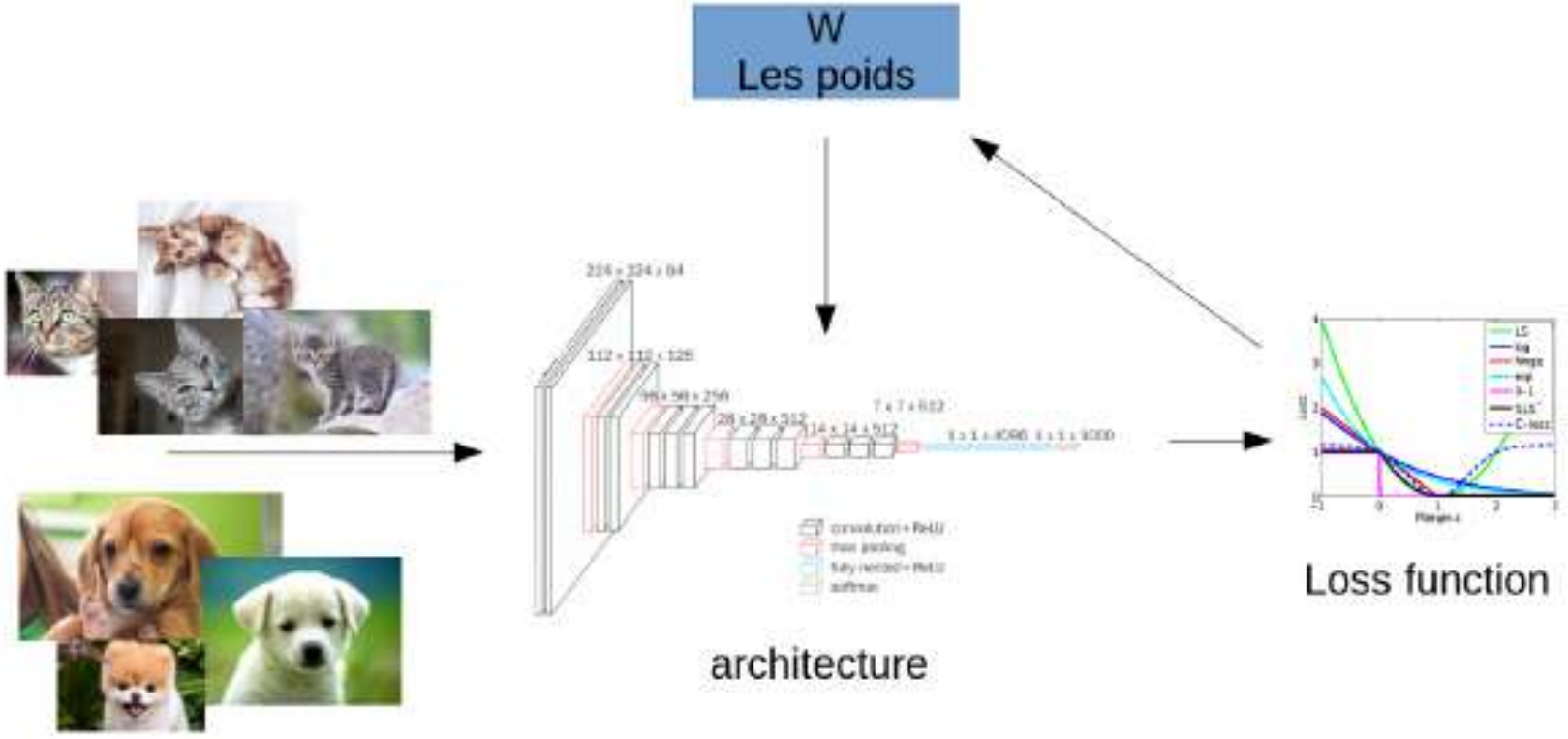
Le NOUVEAU paradigme

Pourquoi une double descente ?

Quand on apprend un réseau, on apprendrait en réalité un ensemble de sous réseau dont la complexité s'adapterait au problèmes ? ! ?

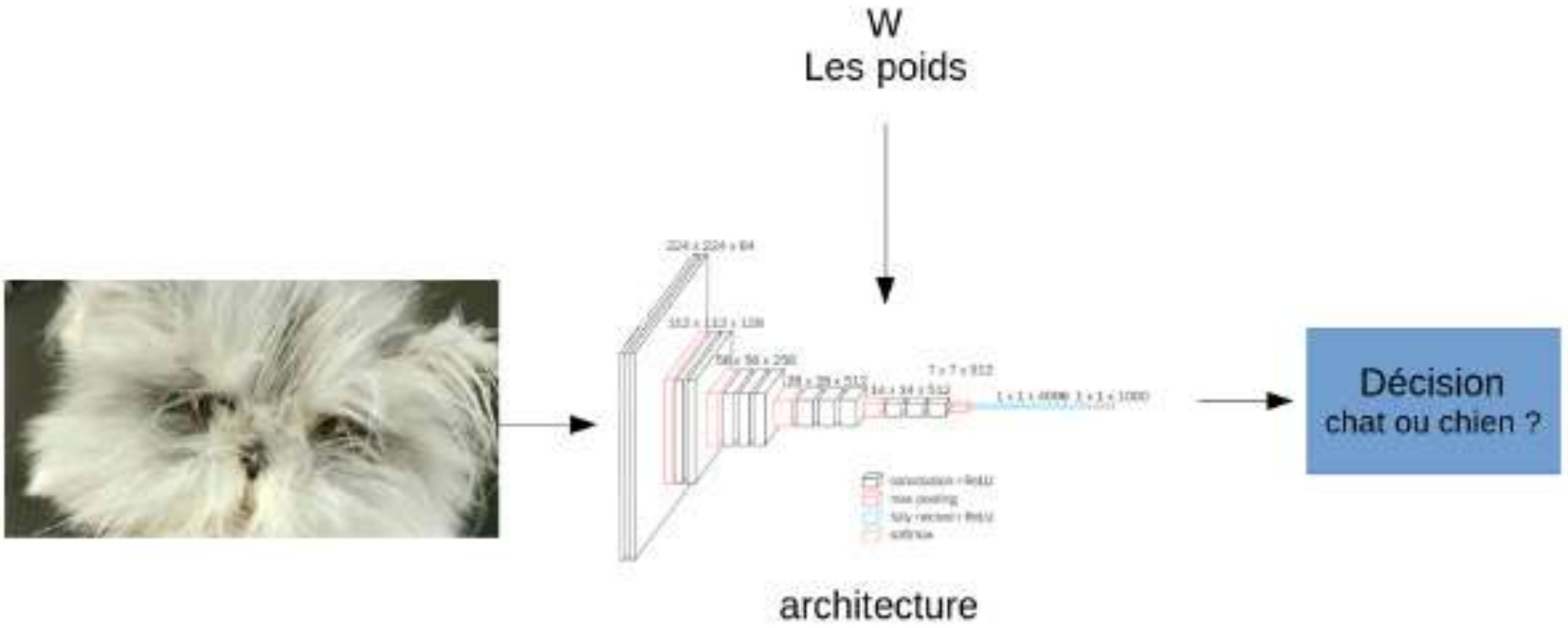
2 phases

Apprentissage



2 phases

Test et/ou production et/ou inférence



Méthodes ensemblistes

- durant l'apprentissage
  - j'ai $x_1, ..., x_N$, $y_1, ..., y_N$
  - je forme w
- en test, j'utilise w

Méthodes ensemblistes

- si j'apprends plusieurs fois
  - j'ai $x_1, ..., x_N$, $y_1, ..., y_N$
  - je forme w_1, ...,w_K
- en test, je peux fusionner les différents modèles
- je tends alors vers les performances d'un modèle moyen w^*

Méthodes ensemblistes

- si j'apprends $K \gg 1$ fois
  - j'ai $x_1, ..., x_N$, $y_1, ..., y_N$
  - je forme w_1, ...,w_K
- en test, je peux fusionner les différents modèles
- je tends alors vers les performances d'un modèle moyen w^*
- **Attention** w^* n'est pas optimal, il est juste moyen (typiquement pour le SVM dont l'apprentissage est déterministe $w_1 = ... = w_K = w^*$)

Méthodes ensemblistes

La complexité n'augmente **pas** avec le nombre de modèle mais **seulement** avec leur capacité.

Créer plein de modèle équivalent n'augmente pas l'overfitting !

La spécificité des réseaux de neurones ?

ancien paradigme

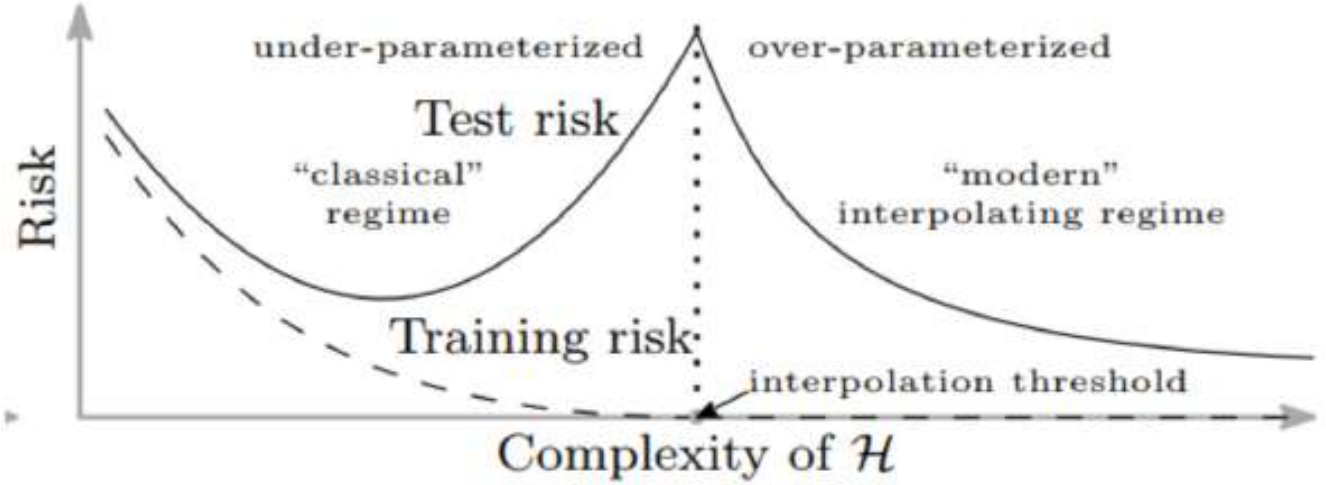
- K réseaux de Q neurones ce n'est pas comme 1 réseau de KQ neurones
- K ne crée pas d'overfitting
- Q trop petit on n'apprend pas, Q trop grand on overfit

nouveau paradigme

- Un réseau de H neurones va se décomposer nativement en K réseau de $\frac{H}{K}$ neurones
- $\frac{H}{K}$ serait nativement adapté au problème !

Ça n'engage que moi

Personnellement j'ai déjà fait overfitté des réseaux sur des problèmes jouets...
Mais il est vrai que sur une gammes très très grande de taille de réseau, on n'observe pas d'overfitting



D'où viendrait cette spécificité ?

de la façon de les apprendre :
de la descente de gradient stochastique

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 ∇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 ∇F_u
3. si $\nabla F_u \approx 0$ ou early stopping alors sortir
4. $\lambda = 1$
5. tant que $F(u - \lambda \nabla F_u) \geq F(u)$ 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 χ, 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

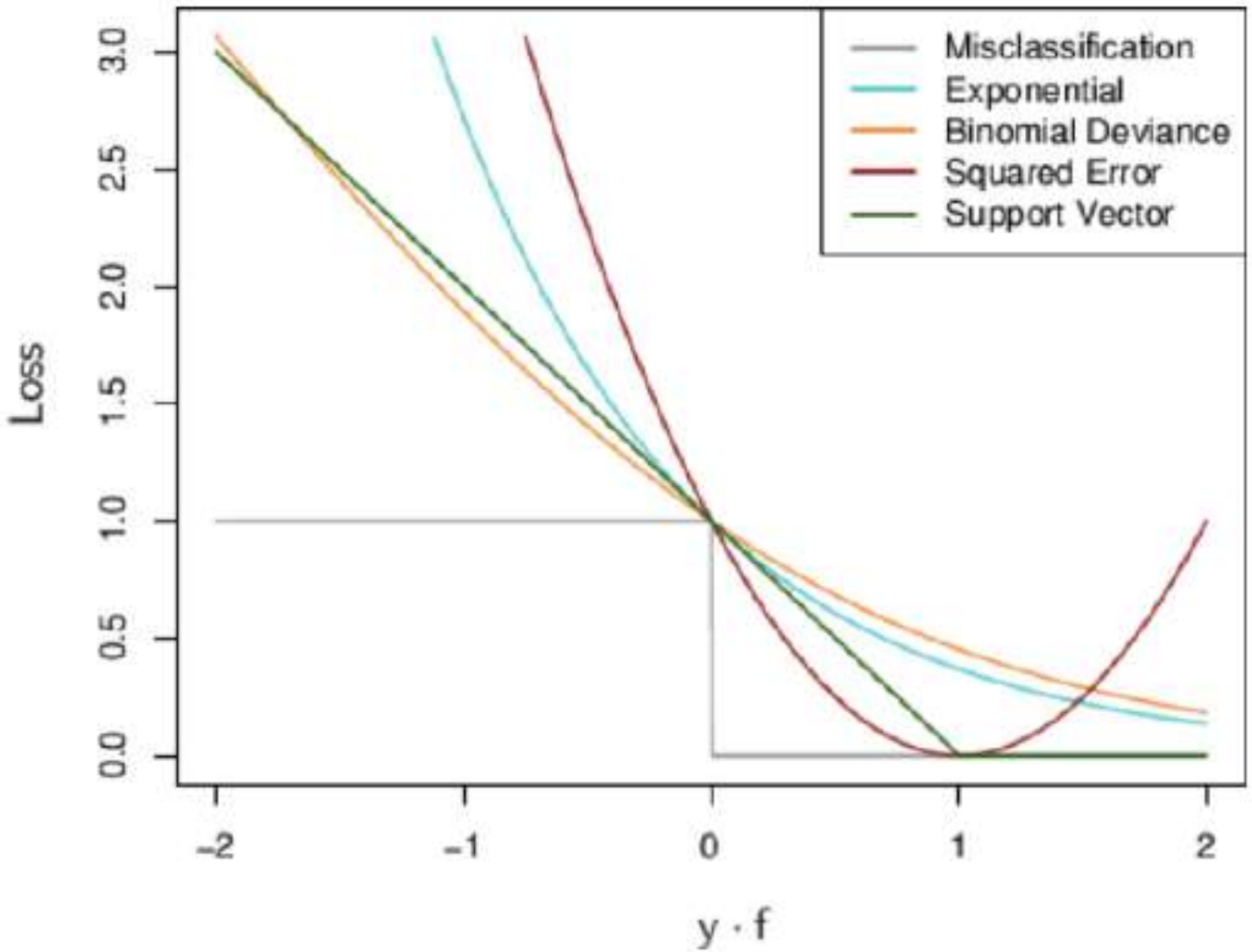
$$F(w) = loss\,(w) = \sum_n l(y(x_n)f(x_n, w))$$

- 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

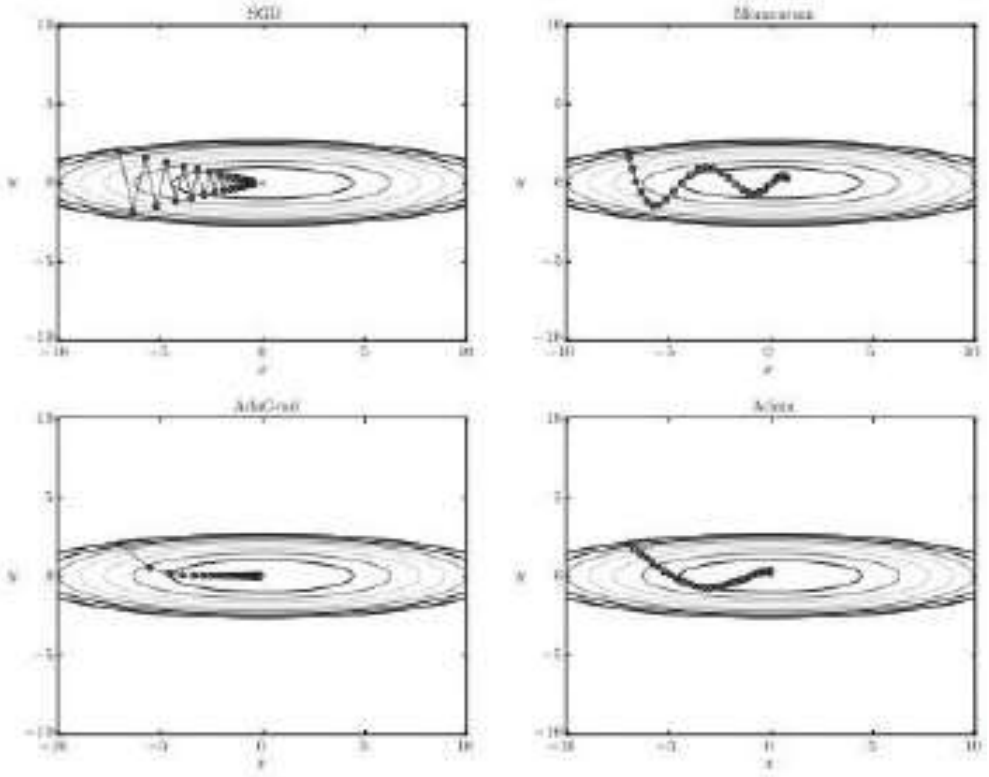
input : $x_1, y_1, ..., x_n, y_n,\ w_0$

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 :



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 - α

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

Forward - Backward

Réduction α - α

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

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
```

Message to take home

- Ce qui est embêtant avec le DL c'est que rien n'est *monotone* :
  - sur le nombre de neurones (parfois plus c'est mieux)
  - sur le volume de données (parfois plus c'est moins bien)

  $\Rightarrow$ Tous les réglages sont longs et nécessitent beaucoup de ressource de calcul

Message to take home

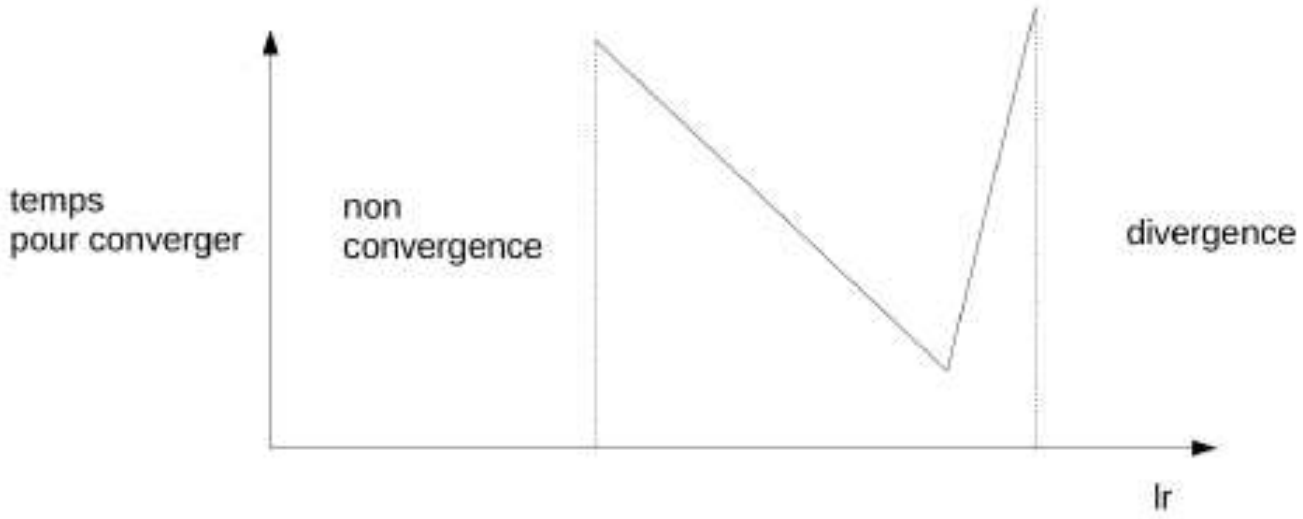
- Ce qui est embêtant avec le DL c'est que rien n'est *monotone* :
  - sur le nombre de neurones (parfois plus c'est mieux)
  - sur le volume de données (parfois plus c'est moins bien)

$\Rightarrow$ Tous les réglages sont longs et nécessitent beaucoup de ressource de calcul

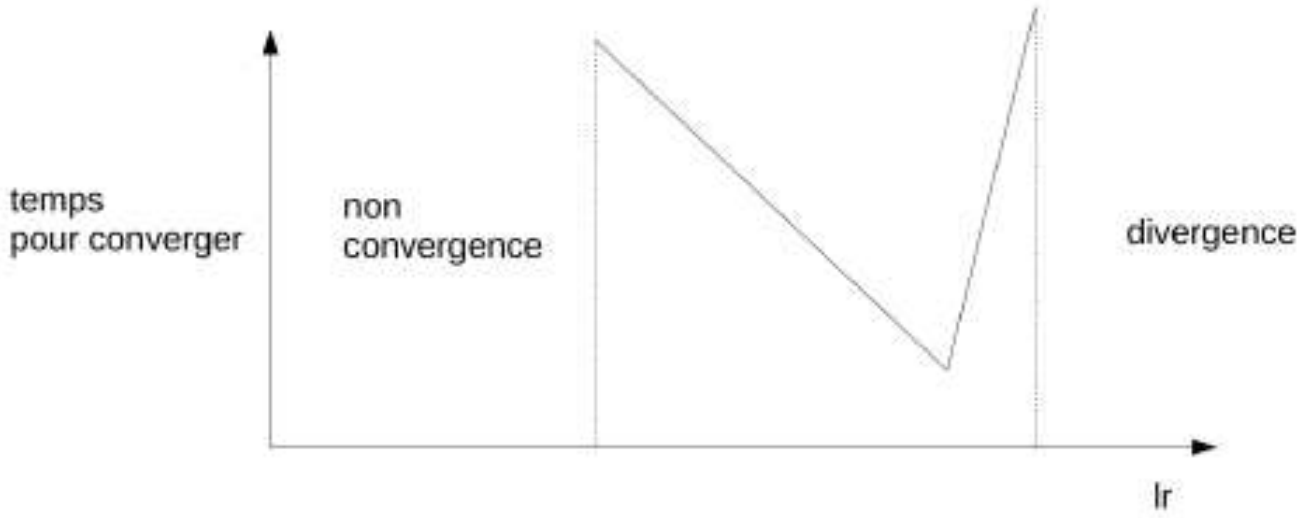
$\Rightarrow$ **exemple du learning rate**

$(w = w - lr \times \nabla partial\_Loss)$

Réglage du learning rate



Réglage du learning rate



Oui MAIS : la performance en test est meilleure quand la convergence a lieu avec un lr fort !

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

Réglage du learning rate

Un learning rate plus faible permet de prendre des raccourcis qui accélère l'apprentissage mais pousse à apprendre du bruit !

(Un learning rate plus faible augmenterait la capacité du réseau !)

Réglage du learning rate

Vous pouvez obtenir 2% d'erreurs en apprentissage en 5mins sur CIFAR10.
Pourtant les meilleurs codes CIFAR10 tournent pendant des nuits pour gagner quelques % en test !

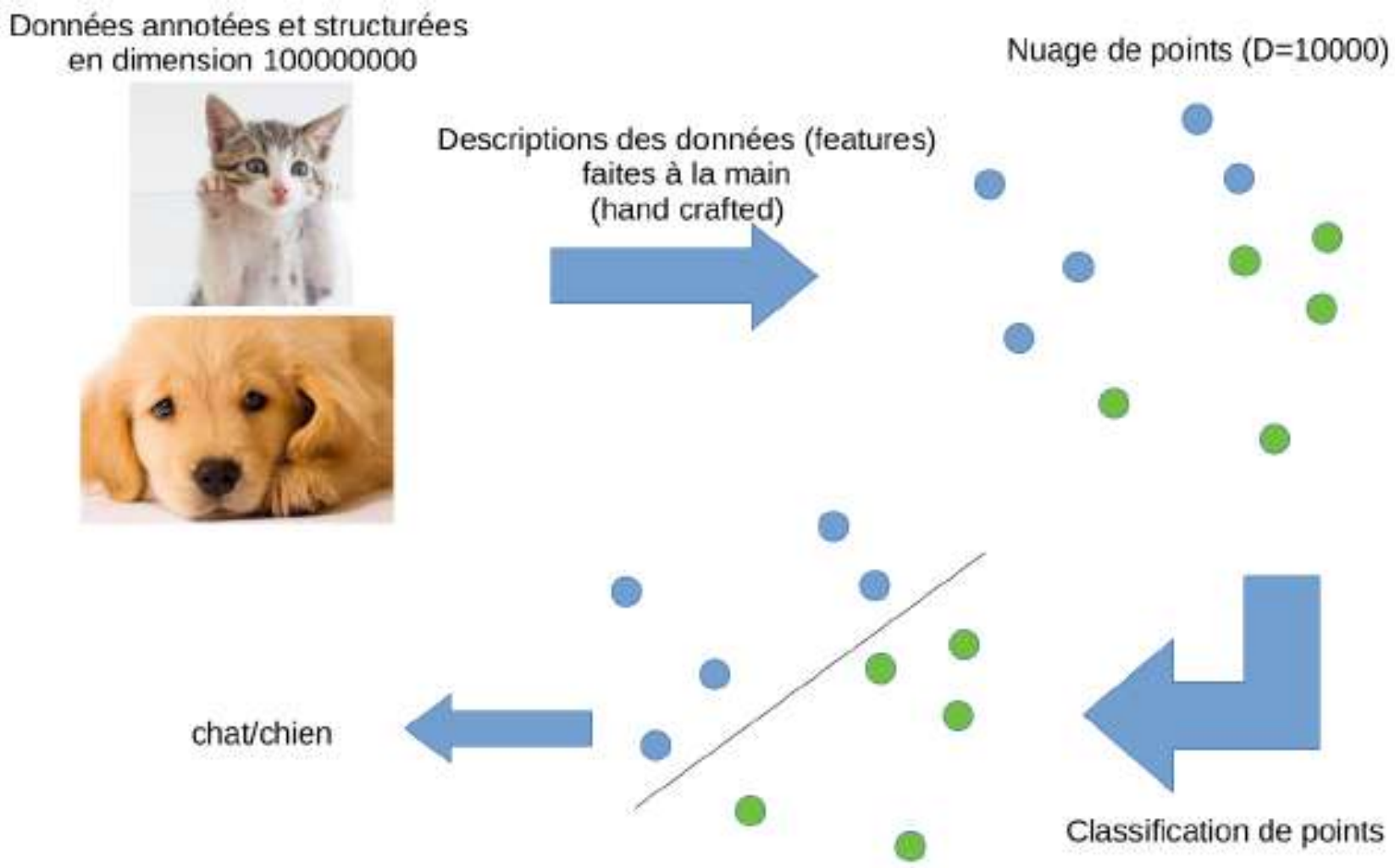
Encore une fois une grosse prime à la puissance plus qu'à l'intelligence :-(

Plan

- Classification de points
  - Apprentissage et test
  - Neurone et réseau
  - Universalité et erreur de généralisation
- Double descente
  - Compromis simplicité/complexité
  - Méthodes par ensemble, double descente
  - Descente de gradient Stochastique
- Traitement de données structurées
  - Neurones convolutif
  - Segmentation
  - Problème de stabilité

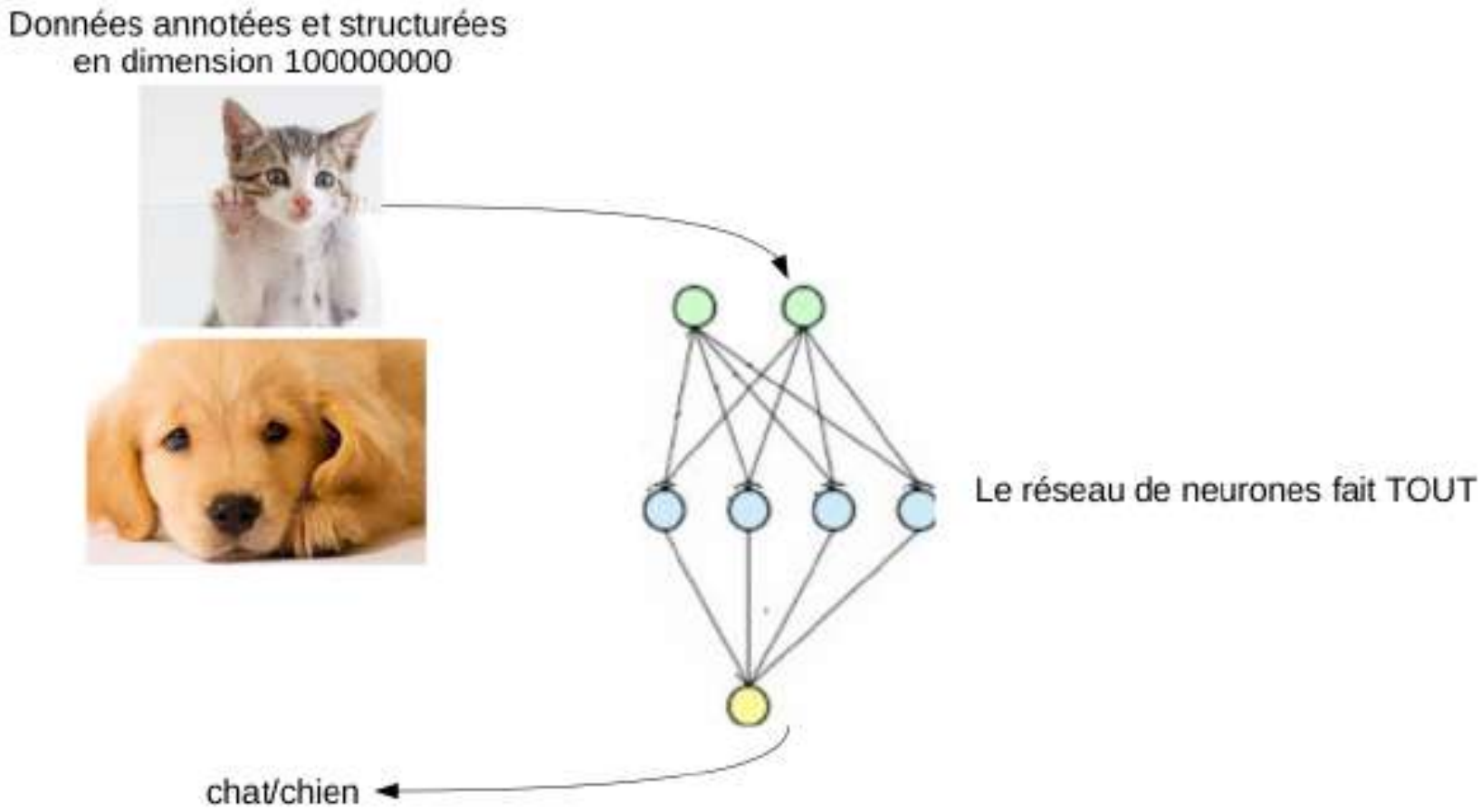
Données structurées

Avant le deep learning



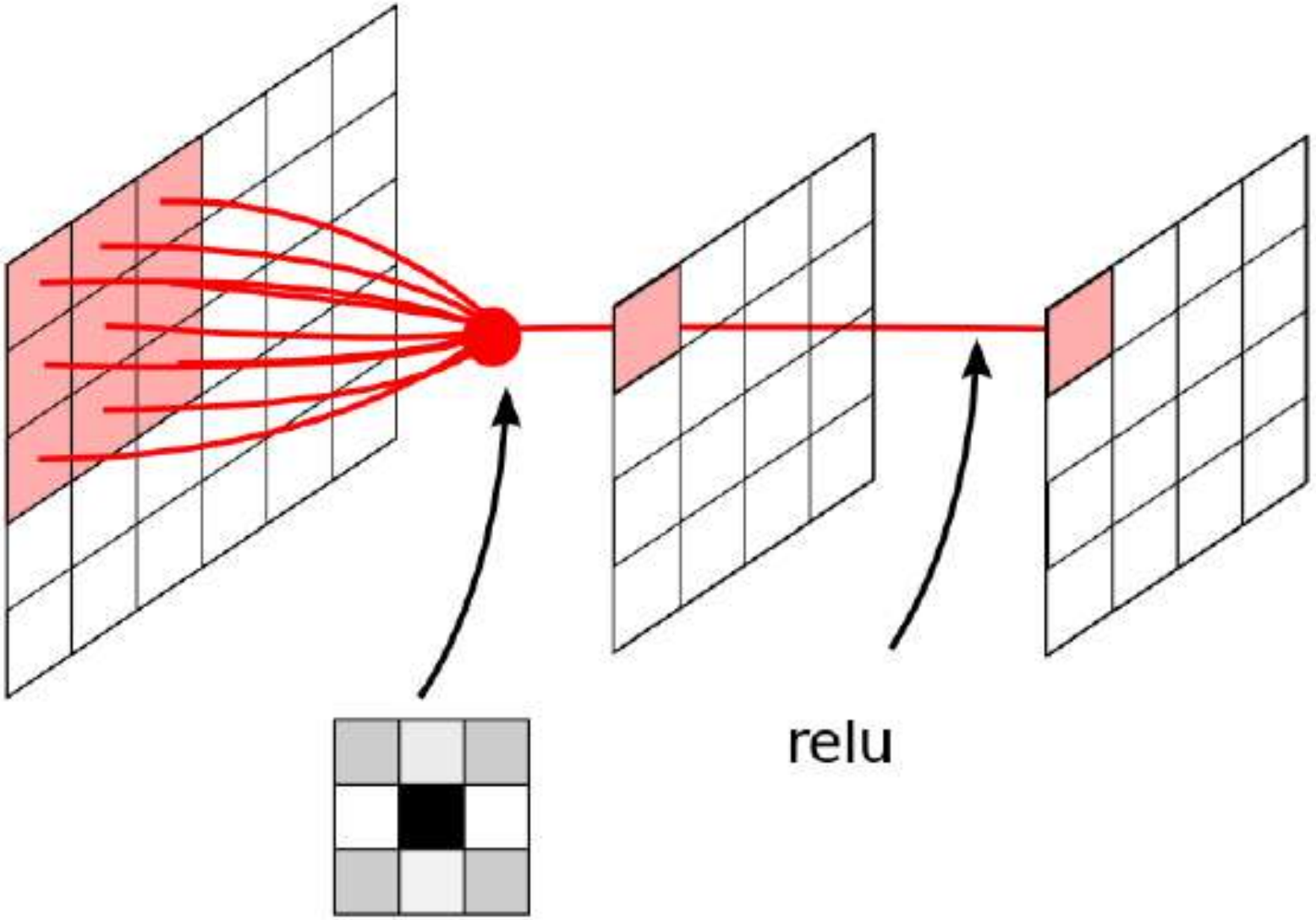
Données structurées

Après



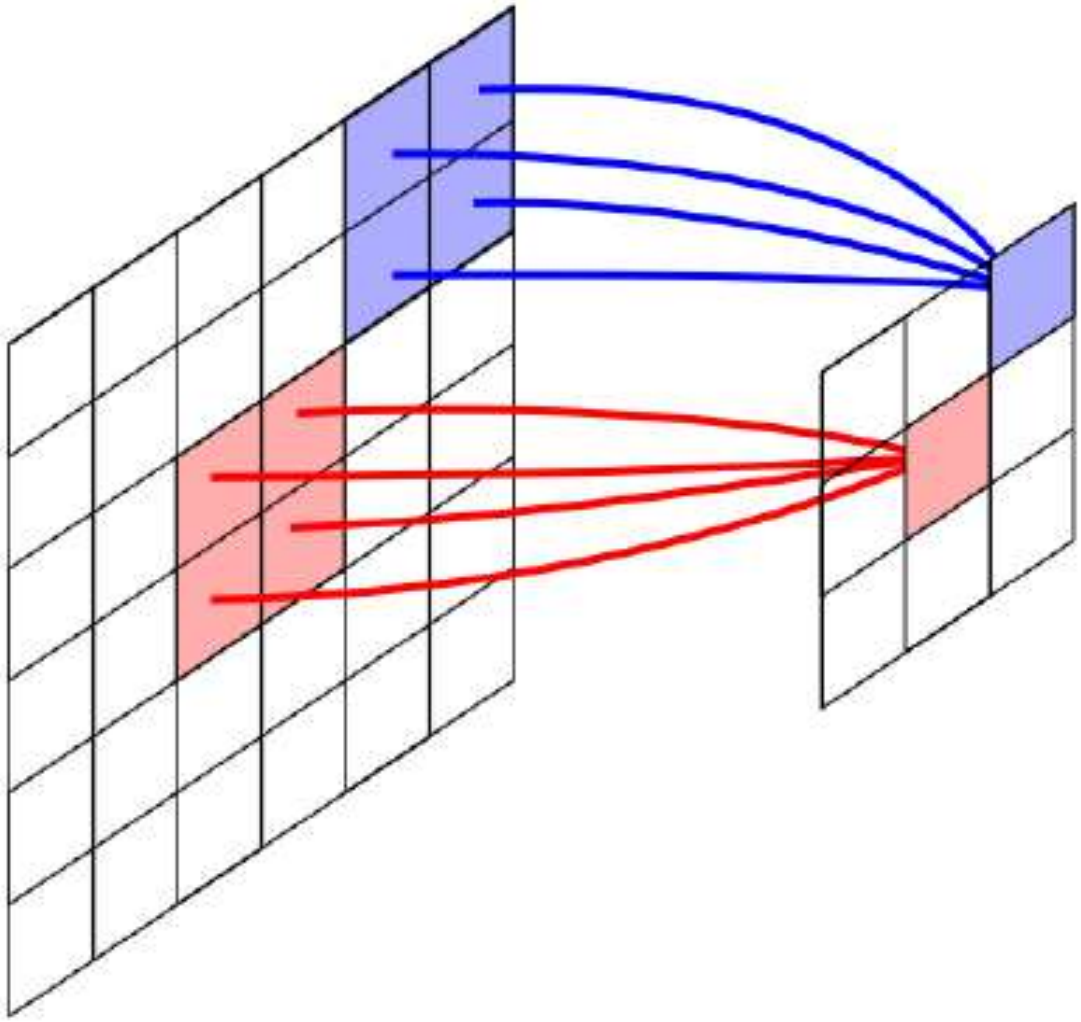
Données structurées

Le neurone convolutif



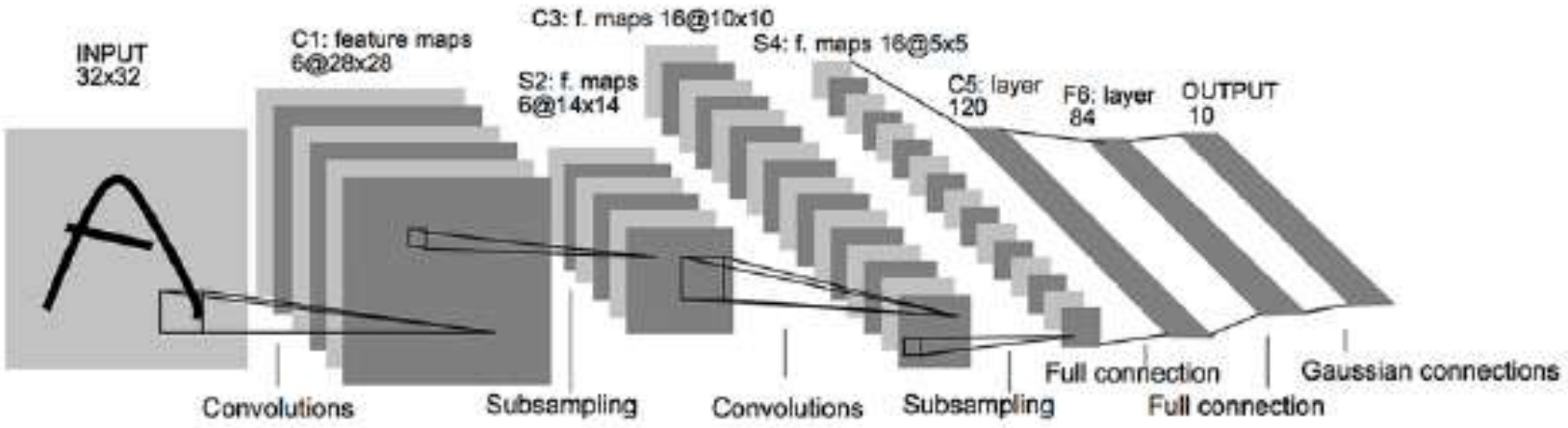
Données structurées

Le pooling



Données structurées

Lenet

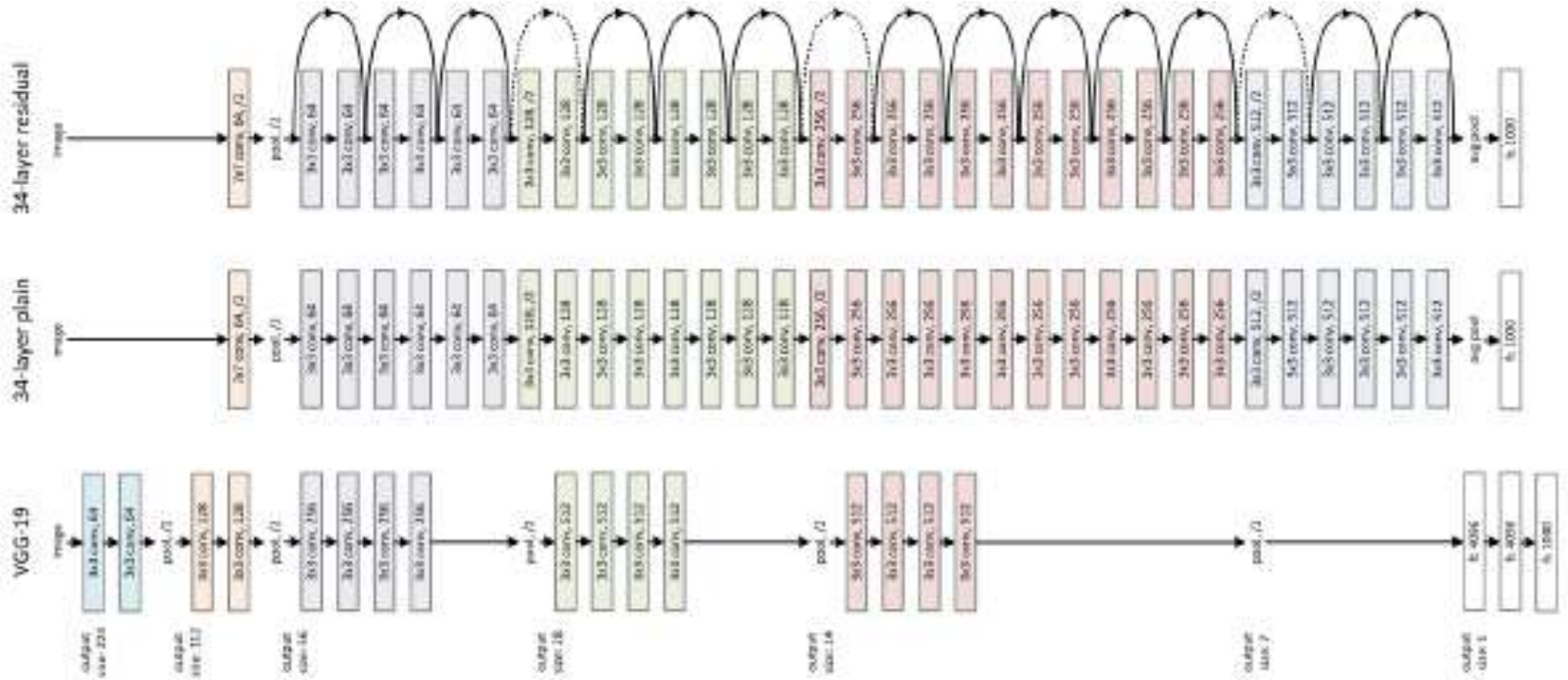


Données structurées

Lenet, Alexnet, VGG

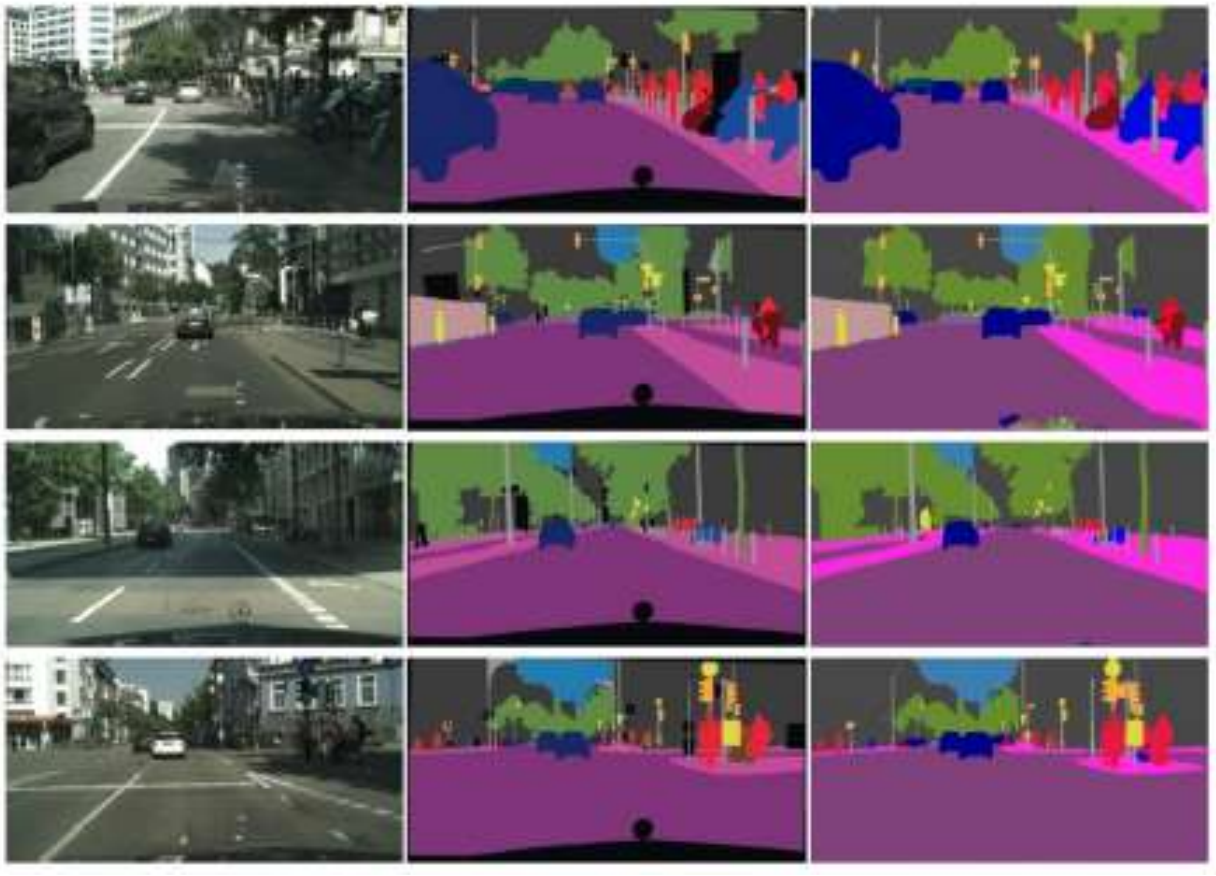
Données structurées

VGG, Resnet

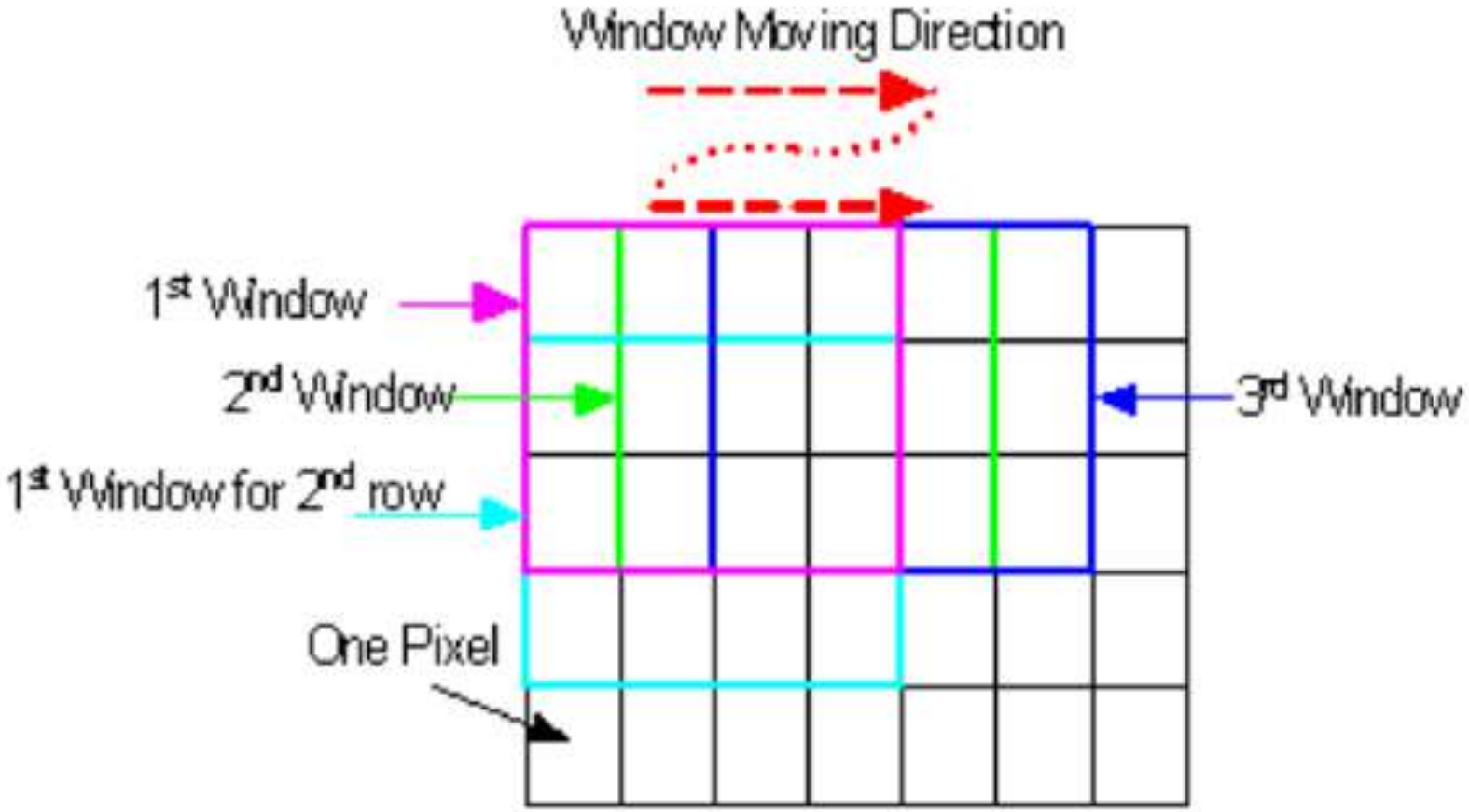


problème structurées

segmentation sémantique ?

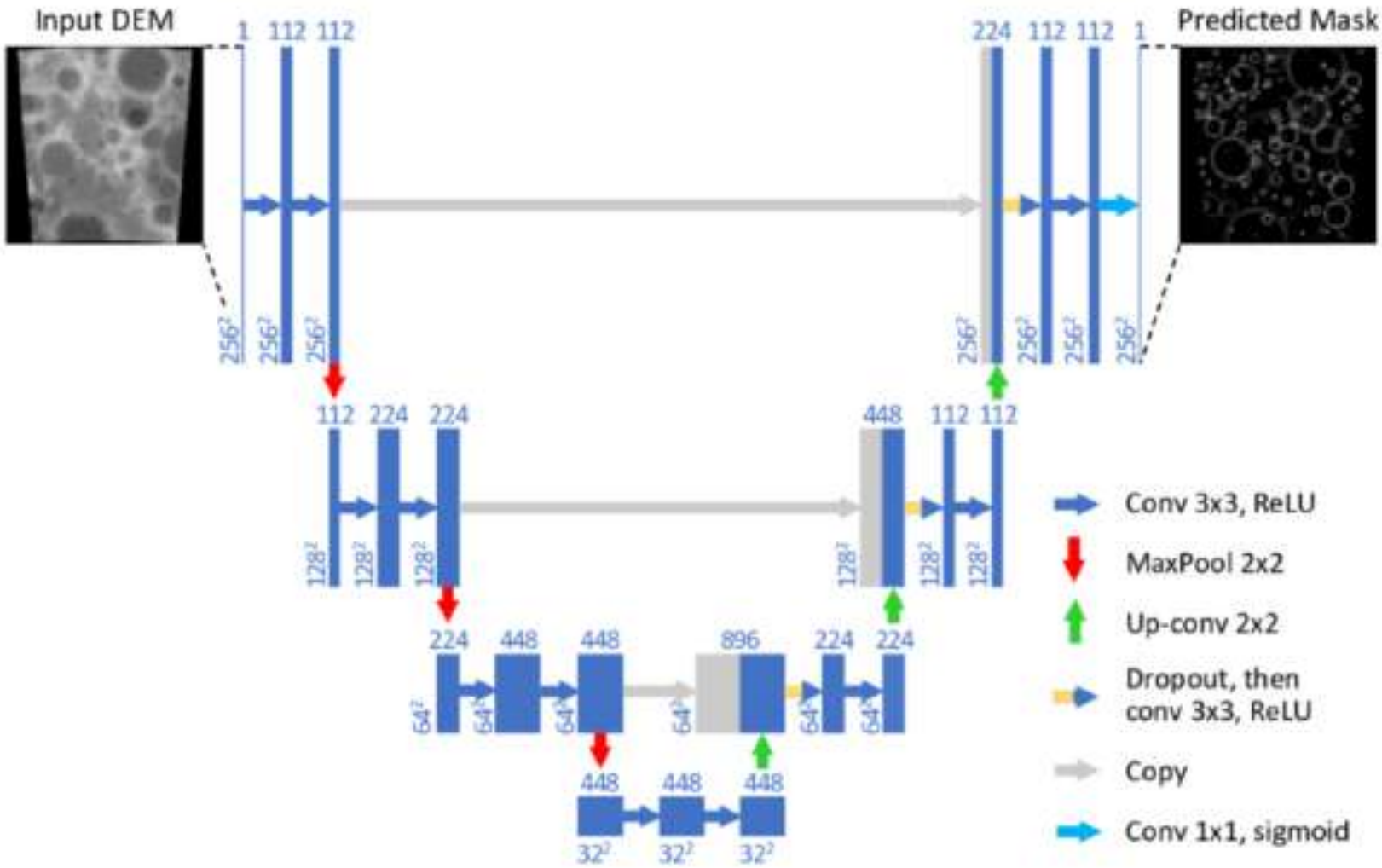


segmentation sémantique



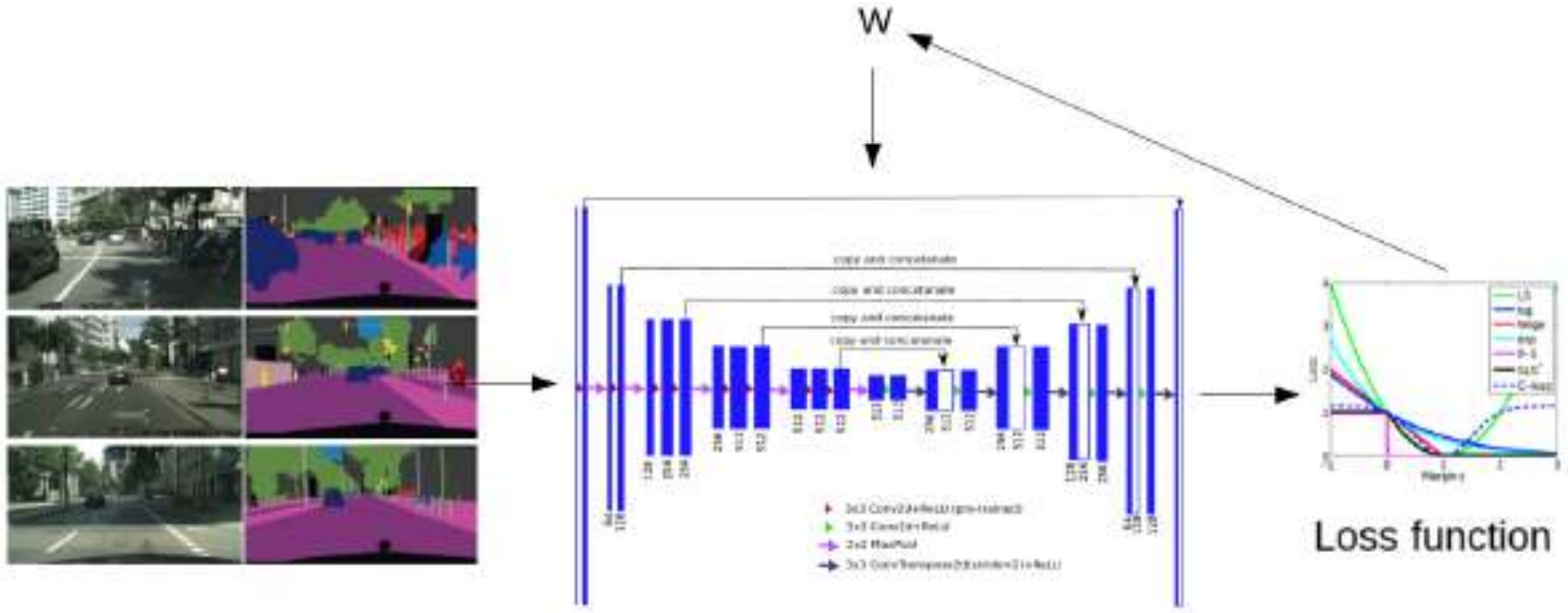
raisonner par fenêtre est une mauvaise idée : il faut raisonner par couche !

segmentation sémantique



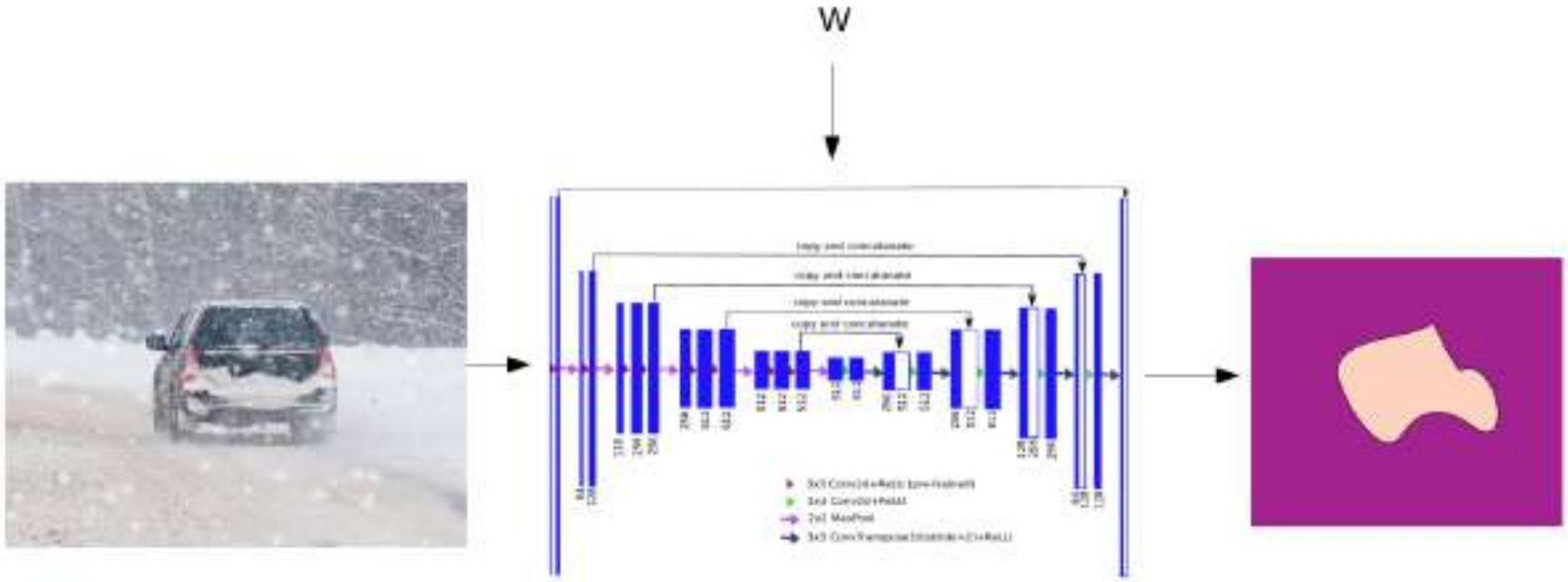
segmentation sémantique : 2 phases aussi

Apprentissage



segmentation sémantique : 2 phases aussi

Test



Le deep learning est incontestablement l'état de l'art sur les données/problèmes structurées
(son, image, vidéo, texte, détection, segmentation, génération...)