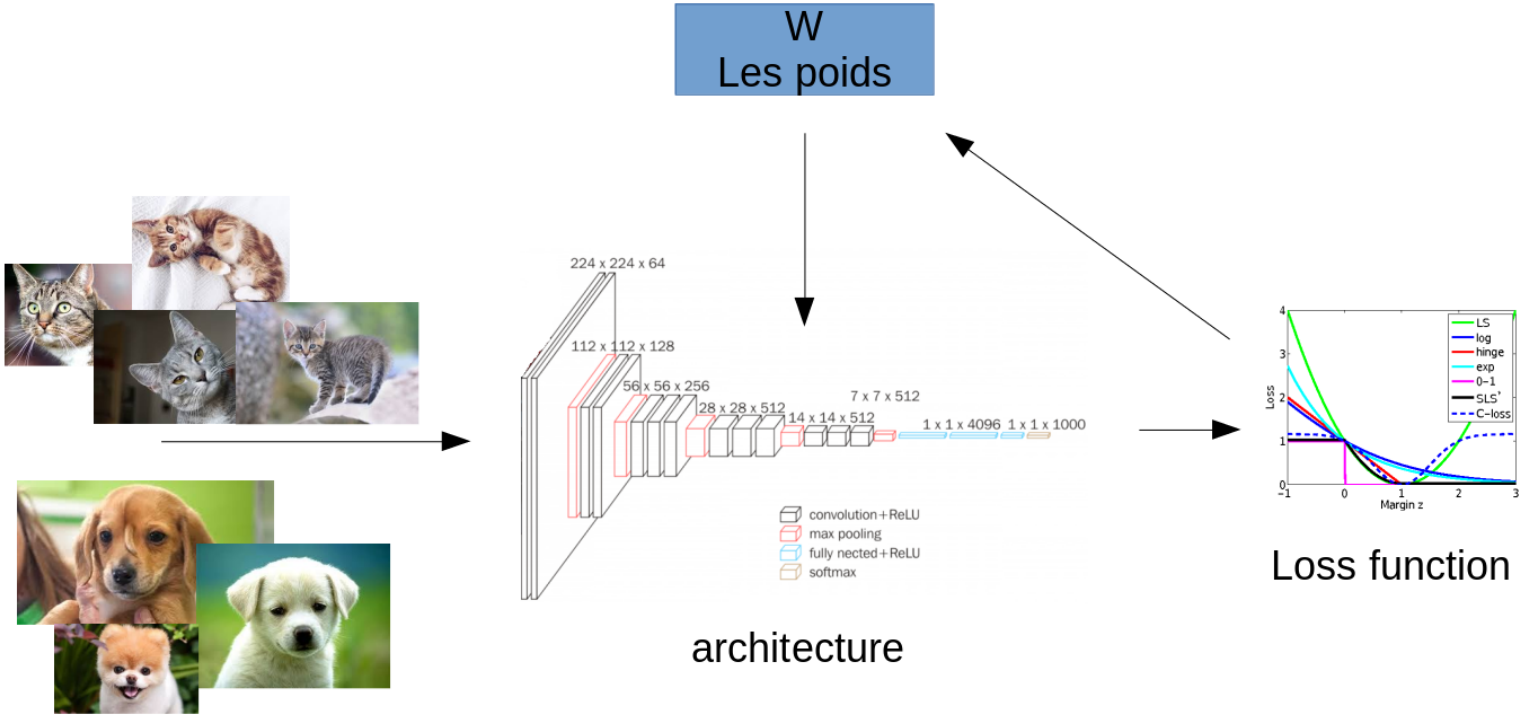
Introduction aux réseau de neurones

mots clés : réseau de neurones, théorème d'universalité, double descente, backpropagation

Adrien CHAN-HON-TONG
ONERA

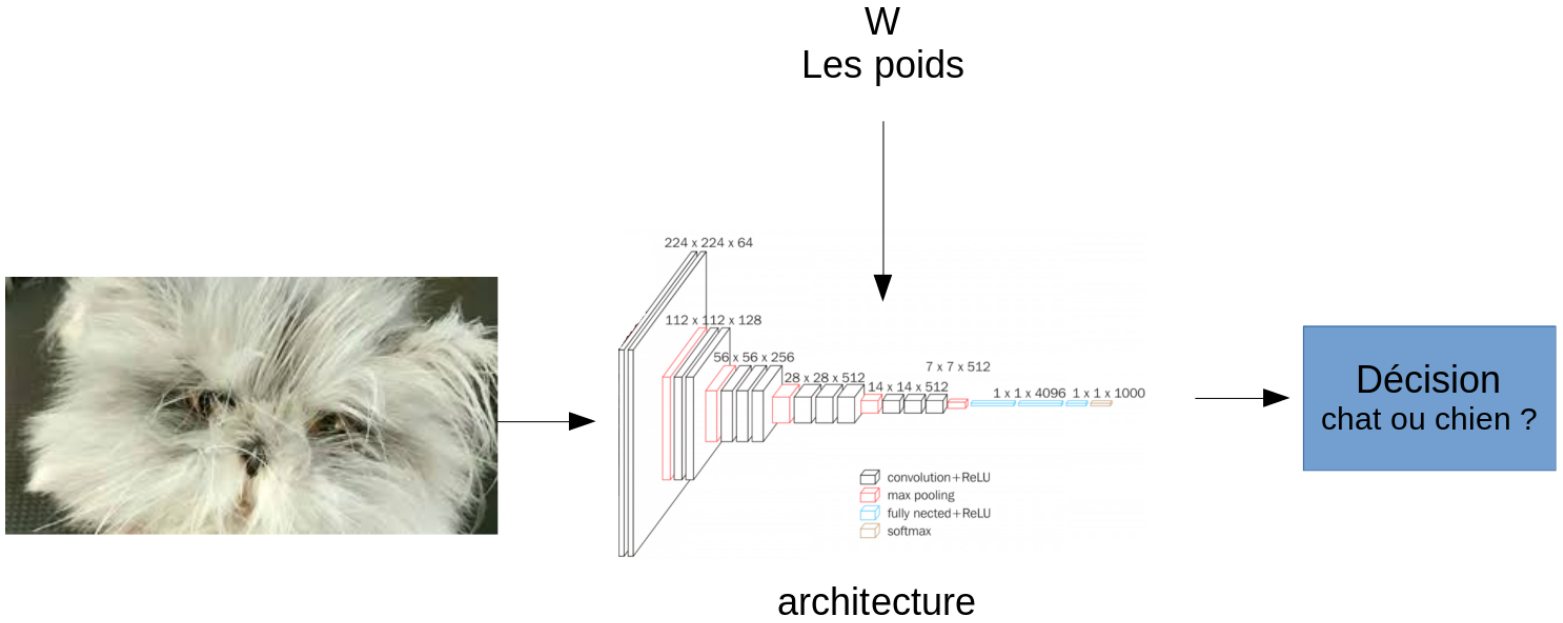
Rappel : apprentissage vs test

Apprentissage

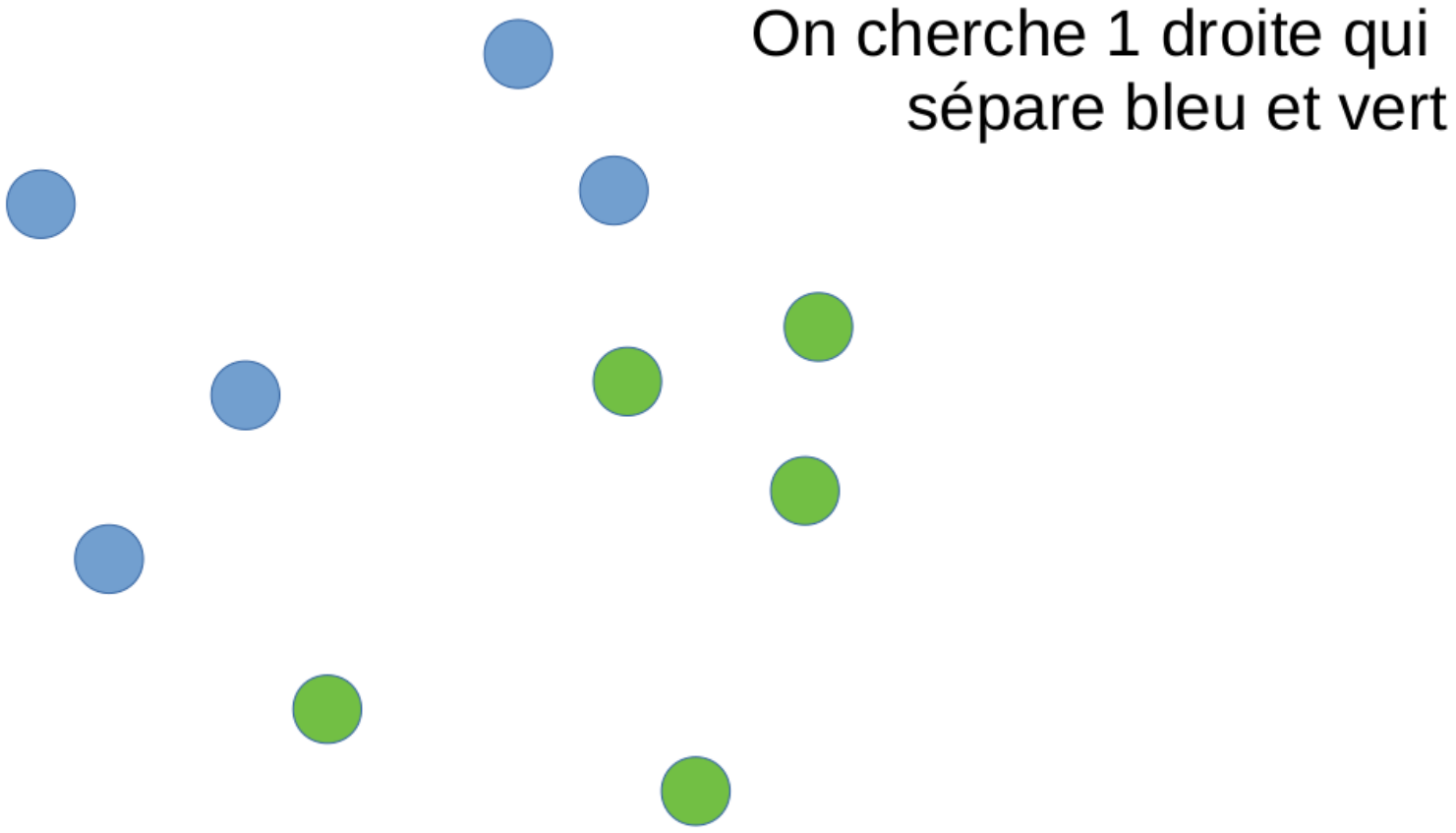


Rappel : apprentissage vs test

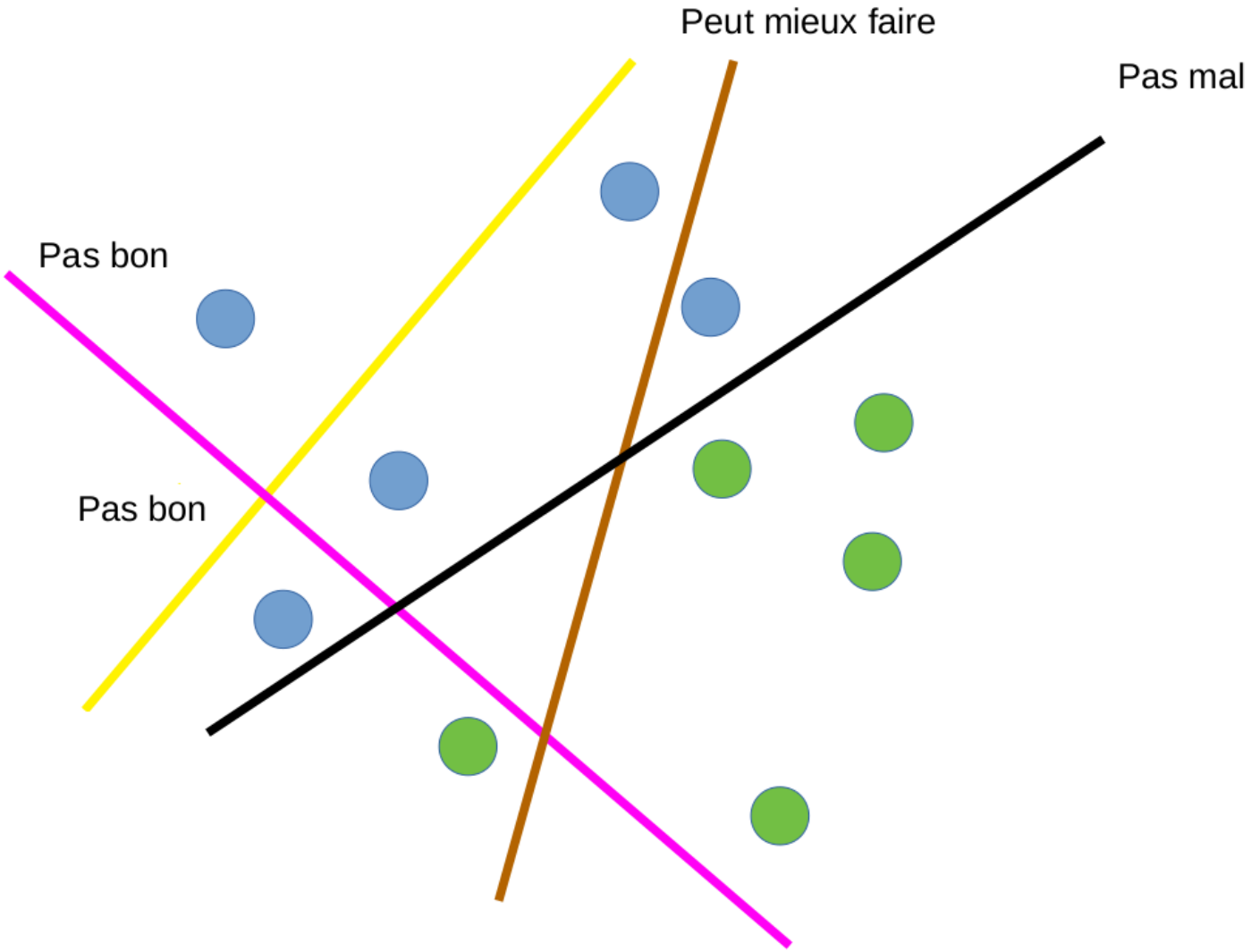
Test et/ou production et/ou inférence



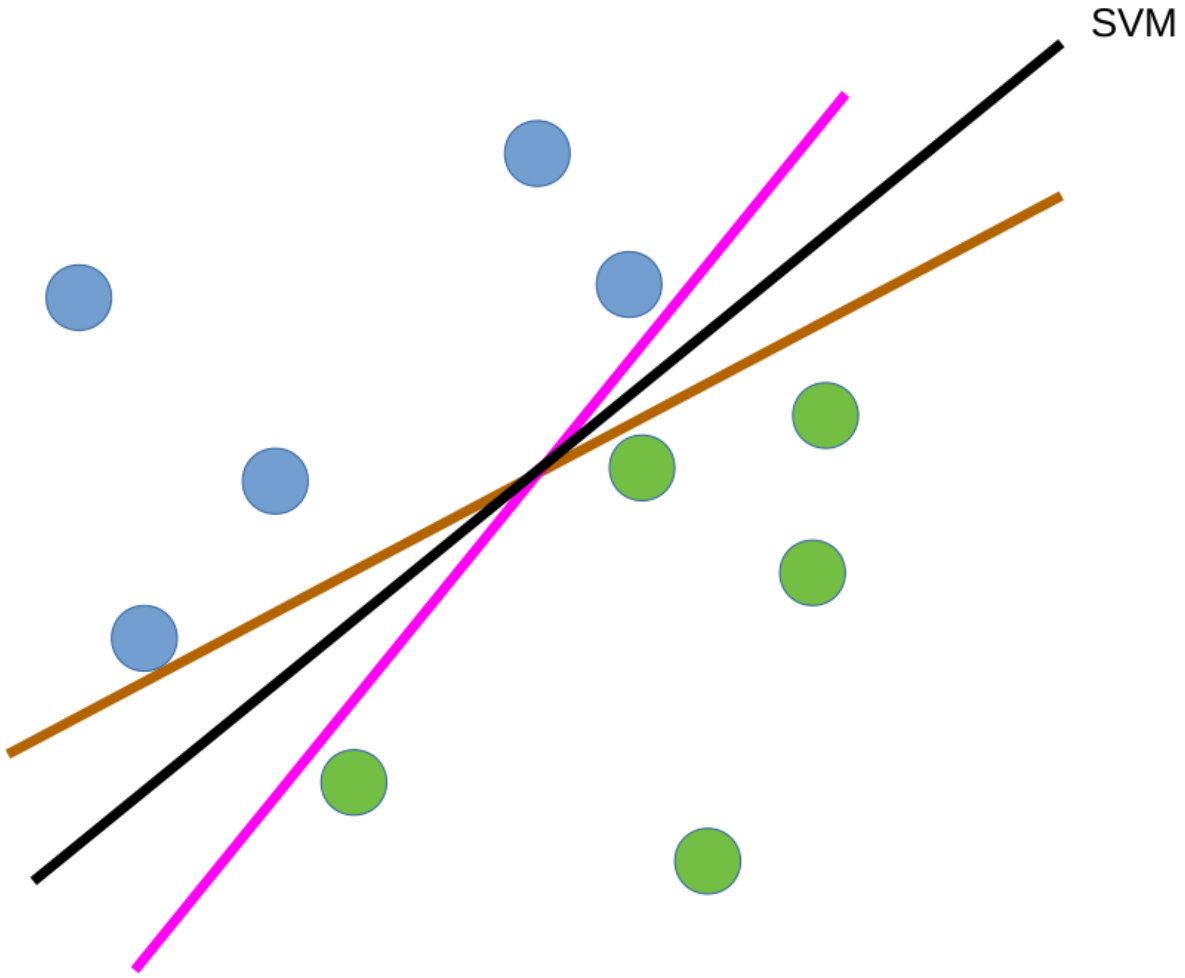
Rappel : cas d'une séparation hyperplane



Rappel : cas d'une séparation hyperplane



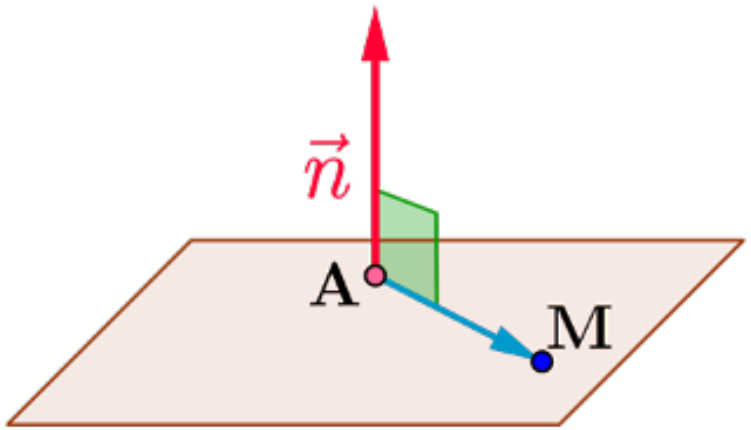
Rappel : cas d'une séparation hyperplane



Rappel : cas 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.

Rappel : cas 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

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

Rappel : cas d'une séparation hyperplane

Le 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

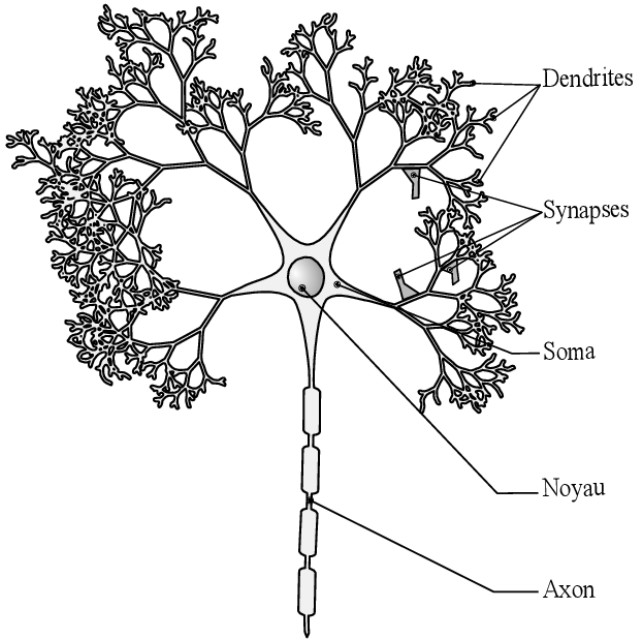
$$\min_{w,b} w^T w$$

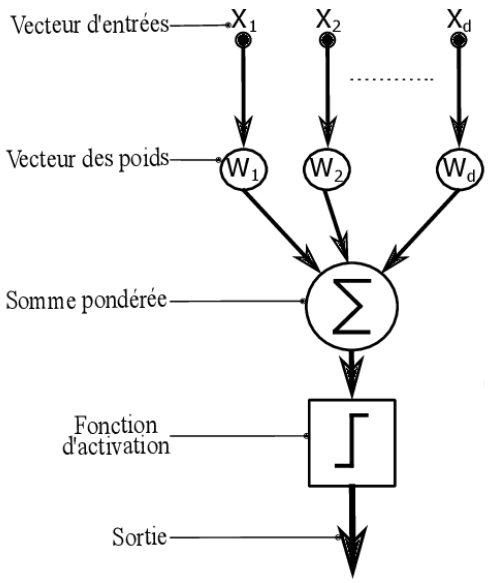
$$sc : \forall n \in \{1, ..., N\},\ y_n(w^T x_n + b) \geq 1$$

Plan

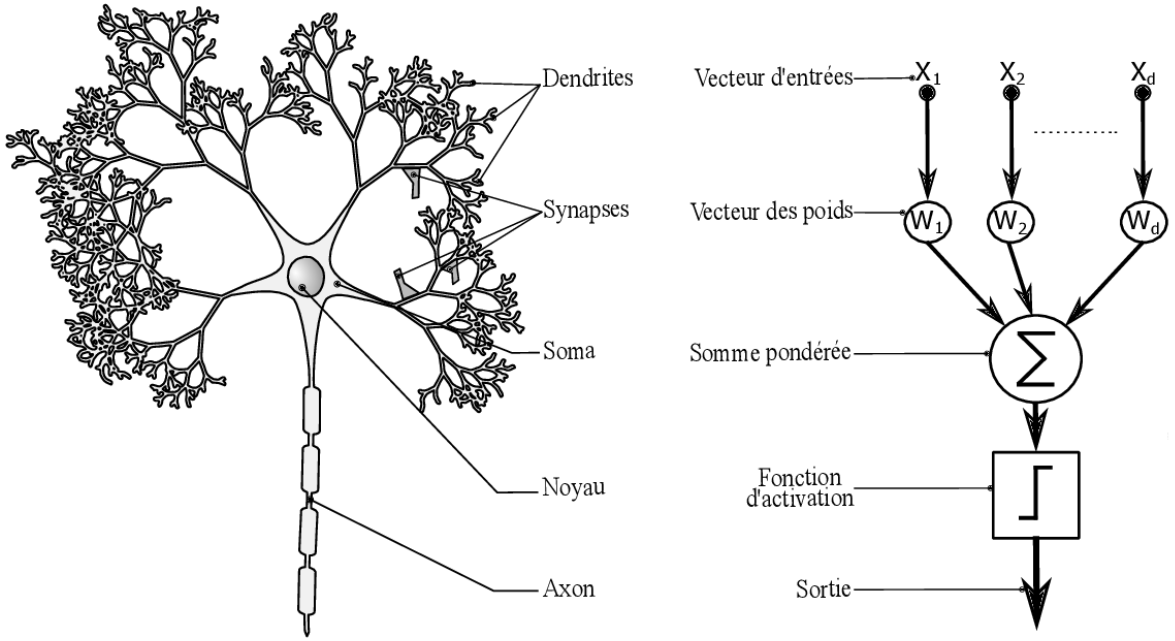
- Rappel : SVM
- Séparation hyperplane et modèle du neurone
- Neurone et réseau
- Théorème d'universalité
- Rappel : Compromis simplicité/complexité
- Méthodes par ensemble, double descente
- Descente de gradient Stochastique

Le neurone


Dendrites
Synapses
Soma
Noyau
Axon

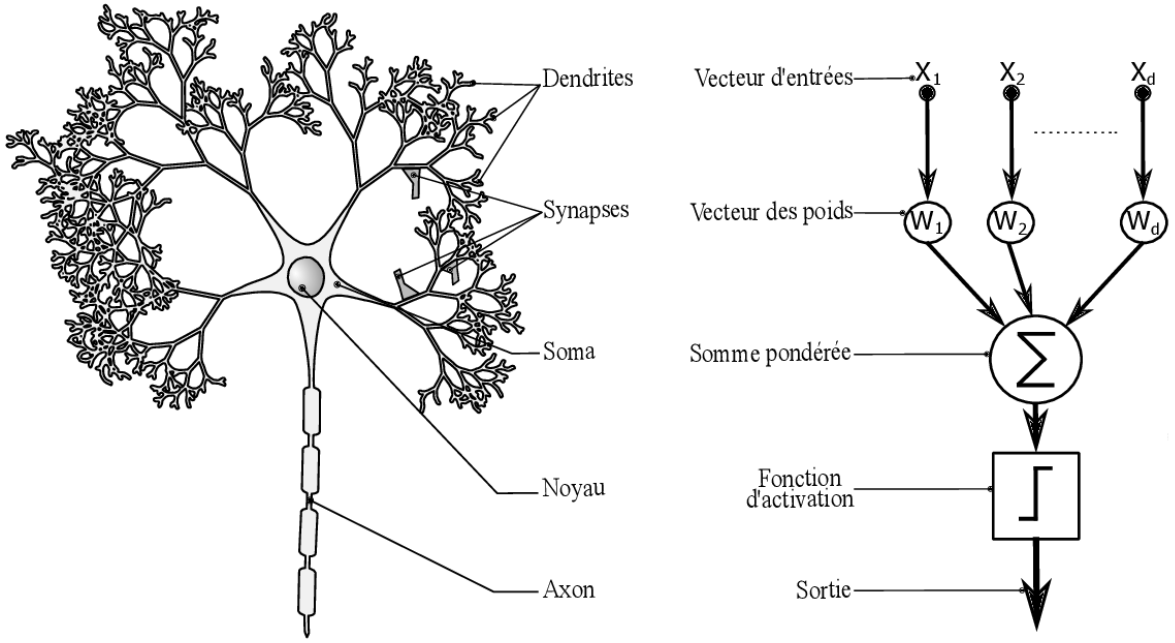

Vecteur d'entrées
X_1
X_2
X_d
Vecteur des poids
W_1
W_2
W_d
Somme pondérée
Fonction d'activation
Sortie

Le neurone



Le neurone sépare l'ensemble des situations où il ne s'active pas et l'ensemble des situations où il s'active.
Biologiquement, l'excitation du neurones n'est **PAS** une simple somme pondérés de ses signaux entrants.

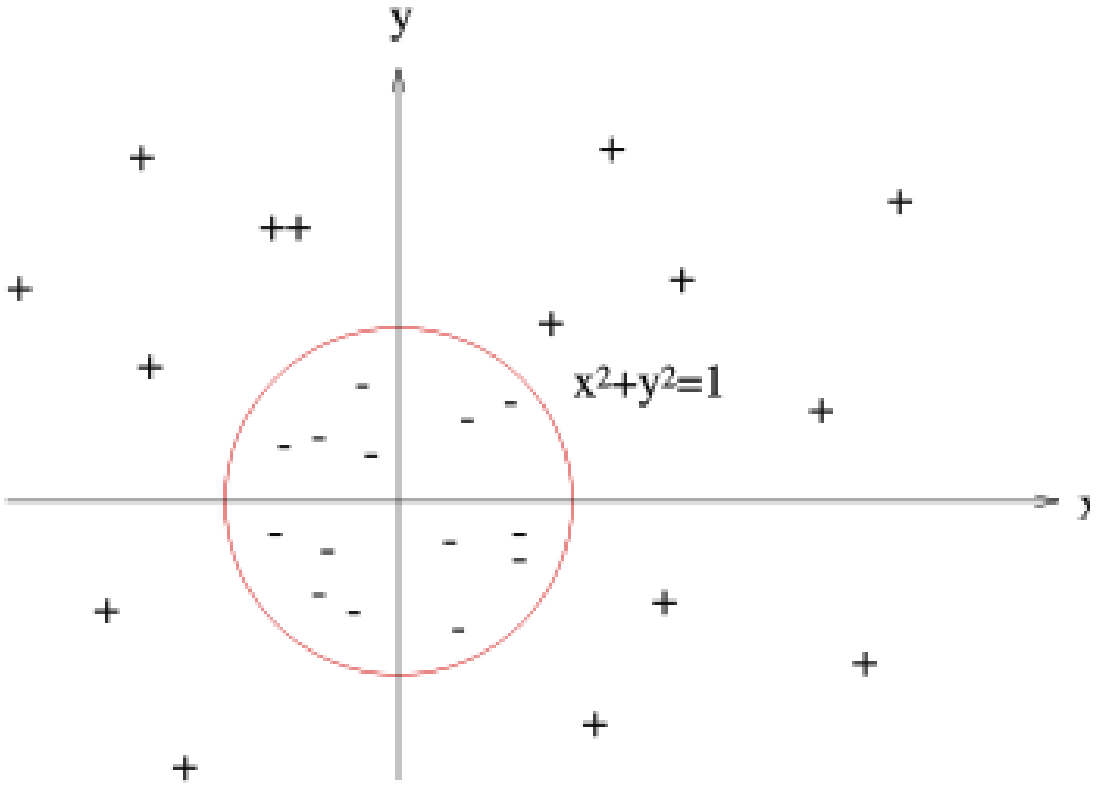
Le neurone



Le neurone sépare l'ensemble des situations où il ne s'active pas et l'ensemble des situations où il s'active.
Biologiquement, l'excitation du neurones n'est **PAS** une simple somme pondérés de ses signaux entrants.
Mais c'est un modèle simplifié : **neurone = hyperplan**

Le neurone

Si un neurone est un hyperplan, quel différence avec les SVM ?
$\Rightarrow$ Parfois on ne peut PAS séparer les données avec un seul hyperplan



Du neurone au réseau

Le neurone

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

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

Attention, ici, un neurone n'est pas nécessairement connecté à la donnée à traiter.

Du neurone au réseau

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 :

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

$A \in \mathbb{R}^{\psi \times \phi}$ et $b \in \mathbb{R}^{\psi}$ sont les **poids** de l'ensemble 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.

Du neurone au réseau

Le réseau de neurone

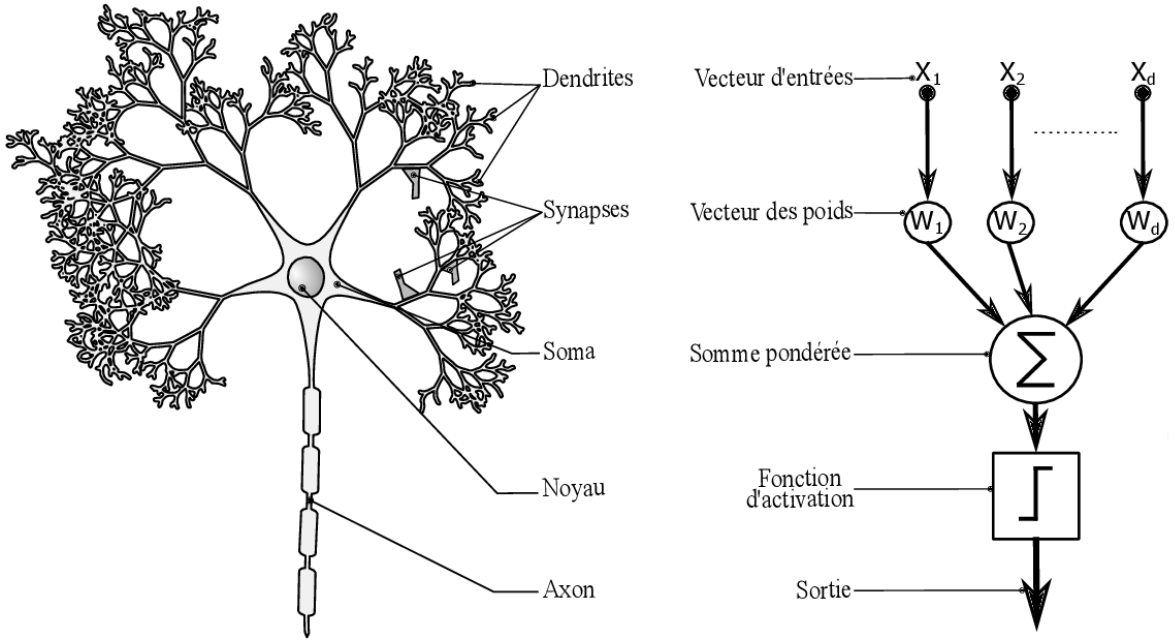
Si on empile 2 couches de neurones c'est exactement comme s'il y en avait qu'une :

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

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

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

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 !

Du neurone au réseau

Le réseau de neurone

On empile des 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.
Par extention, le relu d'un vecteur c'est le vecteur des relu !

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 :

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

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

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

Plan

- Rappel : SVM
- Séparation hyperplane et modèle du neurone
- Neurone et réseau
- Théorème d'universalité
- Rappel : Compromis simplicité/complexité
- Méthodes par ensemble, double descente
- Descente de gradient Stochastique

Universalité des réseaux Relu

Valeur absolue pour $x \in \mathbb{R}$

$$|x| = relu(x) + relu(-x) = \mathbf{1} \cdot relu\left(\begin{pmatrix} 1 \\ -1 \end{pmatrix} x\right)$$

Universalité des réseaux Relu

norme 1 pour $x \in \mathbb{R}^2$

$$||x||_1 = |(1,0)\cdot x| + |(0,1)\cdot x| = relu((1,0)\cdot x) + relu((-1,0)\cdot x) +$$

$$relu((0,1)\cdot x) + relu((0,-1)\cdot x) = \mathbf{1}\cdot relu\left(\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} x\right)$$

Universalité des réseaux Relu

norme 1 pour $x \in \mathbb{R}^D$

$$||x||_1 = \mathbf{1} \cdot relu \left(\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 1 \\ -1 & 0 & \dots & 0 \\ 0 & -1 & 0 & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & -1 \end{pmatrix} x \right) =$$

$$\mathbf{1} \cdot relu \left(\begin{pmatrix} \mathbf{I} \\ -\mathbf{I} \end{pmatrix} x \right)$$

Universalité des réseaux Relu

Avec biais

$\forall x, q \in \mathbb{R}^D$

$$||x - q||_1 = \mathbf{1} \cdot relu\left(\begin{pmatrix} \mathbf{I} \\ -\mathbf{I} \end{pmatrix} x + \begin{pmatrix} -q \\ q \end{pmatrix}\right)$$

On peut coder la distance en norme 1 (à un point constant q) avec une simple couche de 2D neurones et 1 neurone pour sommer !

Le pseudo-dirac

$g_q(x) = \mathbf{1} \cdot relu(1 - ||x - q||_1)$ 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

$$f(x, w) = \sum_n y_n \times relu(1 - ||x - x_n||_1)$$

$$= (y_1 \ \ ... \ \ y_N) \times \left(relu(-\mathbf{I} \times relu \left(\begin{pmatrix} \mathbf{I} \\ ... \\ \mathbf{I} \\ -\mathbf{I} \\ ... \\ -\mathbf{I} \end{pmatrix} x + \begin{pmatrix} -x_1 \\ ... \\ -x_N \\ x_1 \\ ... \\ x_N \end{pmatrix} \right) + \mathbf{1} \right)$$

Apprentissage

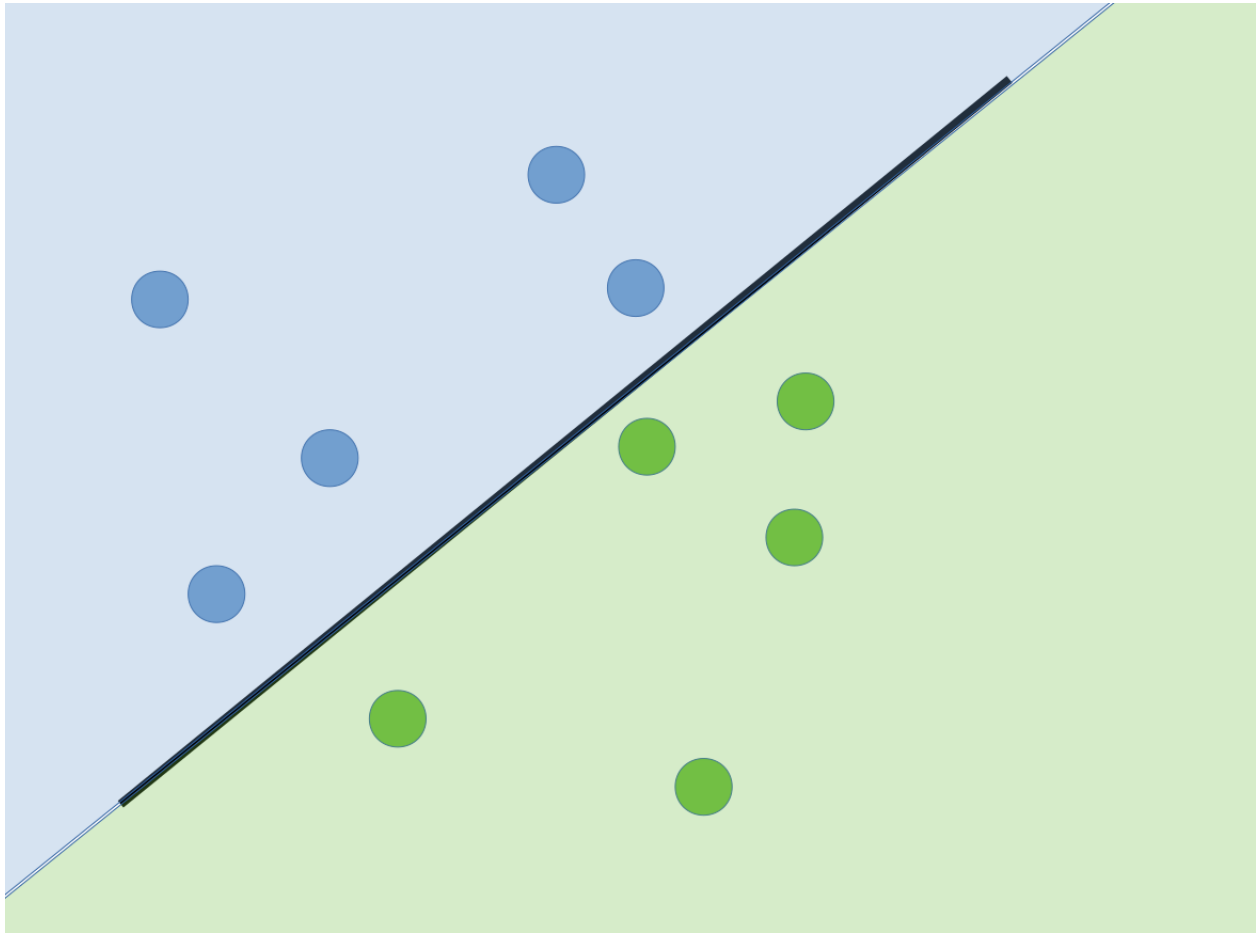
- On a des points colorés qu'on veut séparer
- Avec 1 neurone
  - Ça revient à cherche un hyperplan séparateur
  - $f_w(x) = w^T x + w_{biais}$
  - Mais il peut ne pas exister de séparation
- Avec un réseau de neurones
  - $f_w(x) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$
  - Avec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

MAIS

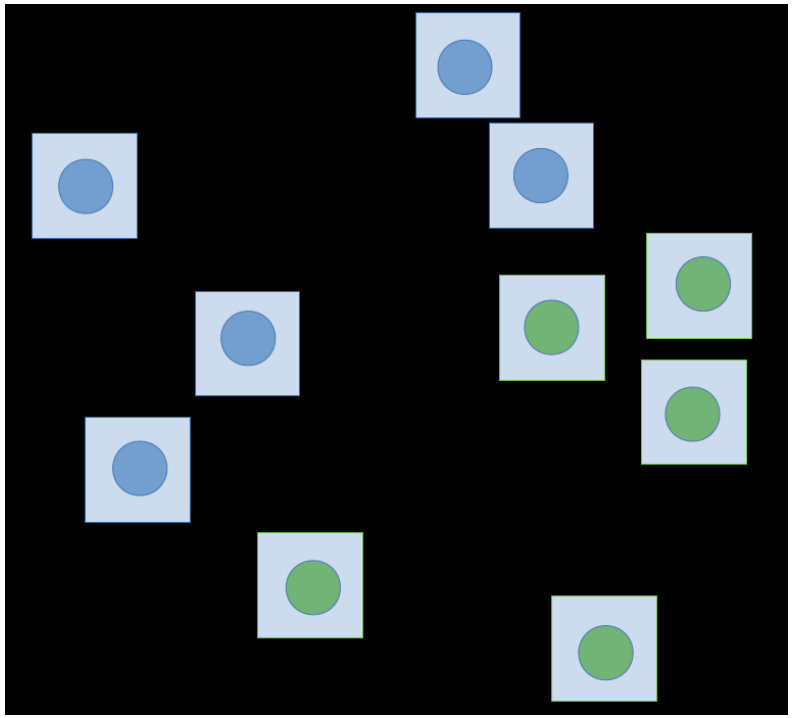
- On a des points colorés qu'on veut séparer
- Avec 1 neurone
  - Ça revient à cherche un hyperplan séparateur
  - $f_w(x) = w^T x + w_{biais}$
  - Mais il peut ne pas exister de séparation
- Avec un réseau de neurones
  - $f_w(x) = w_Q \times relu(w_{Q-1} \times relu(...(relu(w_1 \times x))))$
  - Avec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

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

SVM



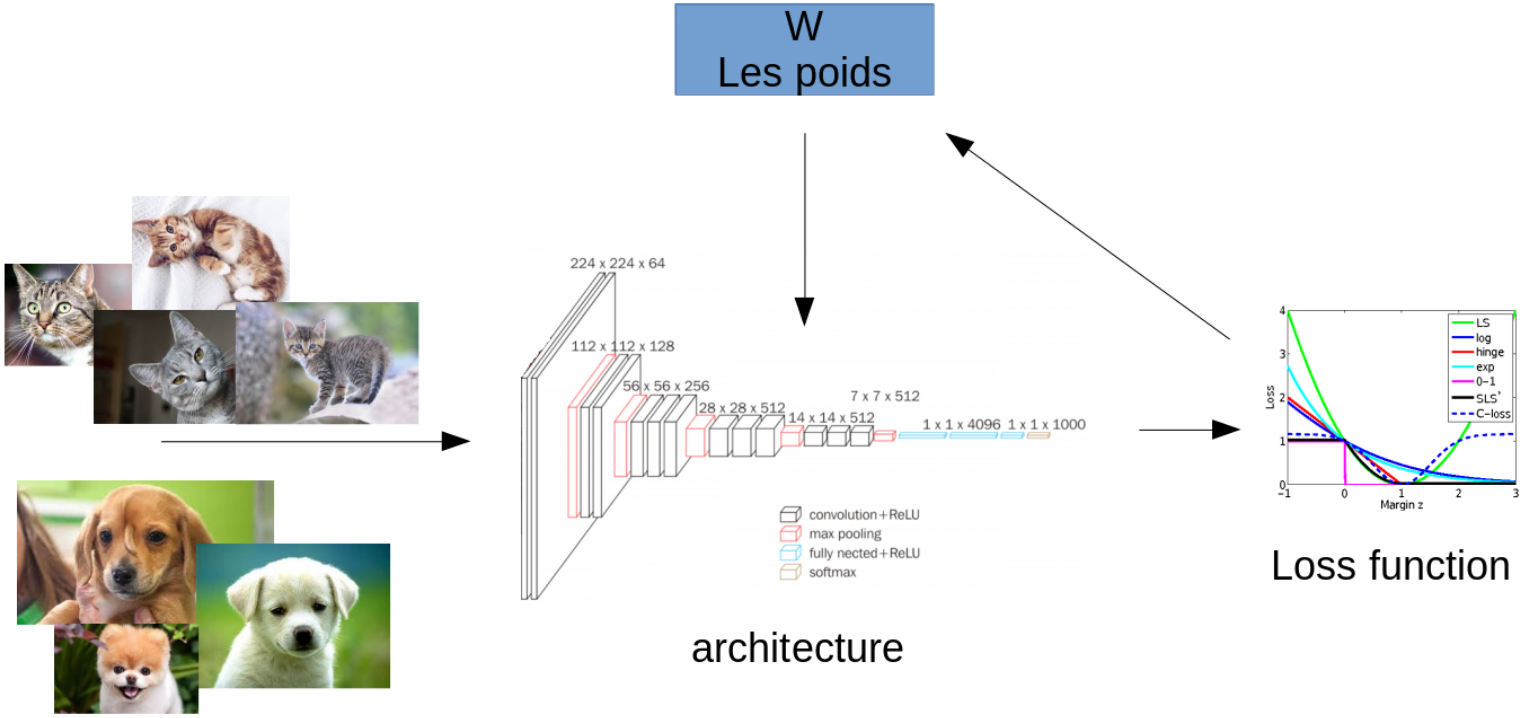
Théorème d'universalité



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.

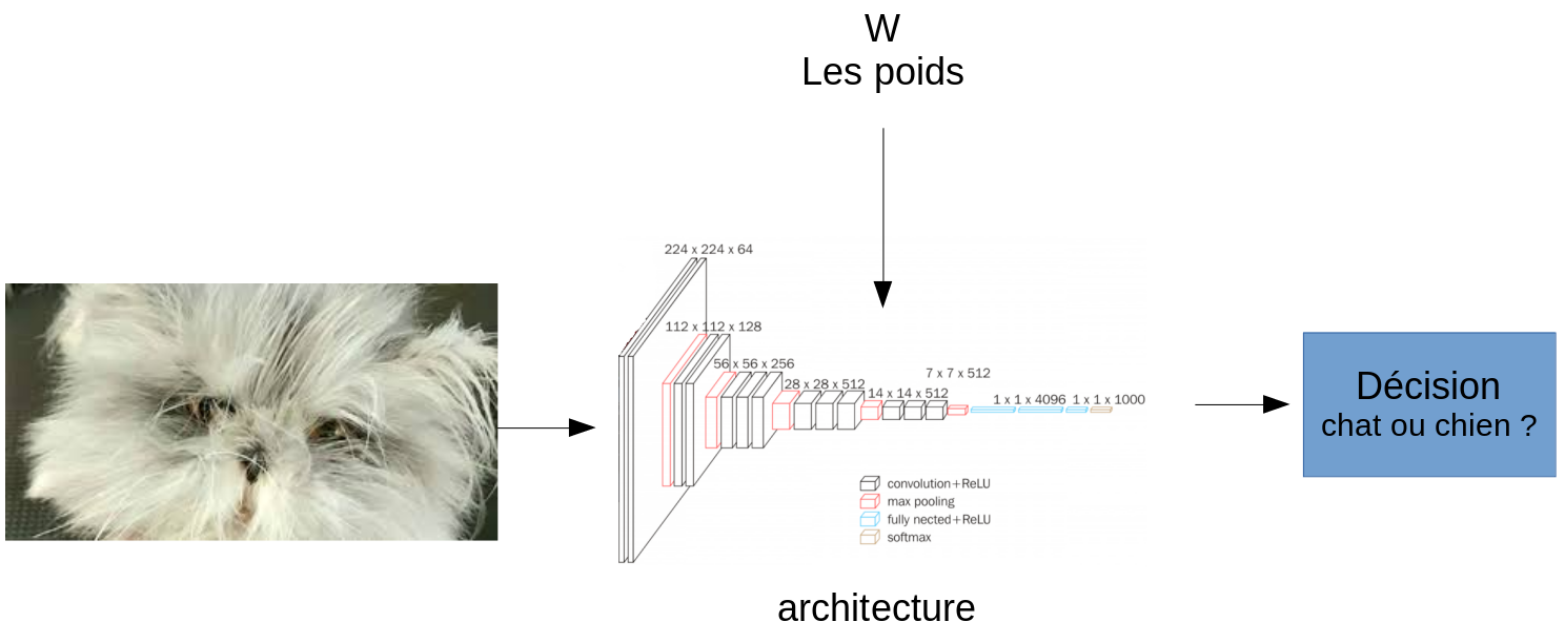
2 phases

Apprentissage



2 phases

Test et/ou production et/ou inférence



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

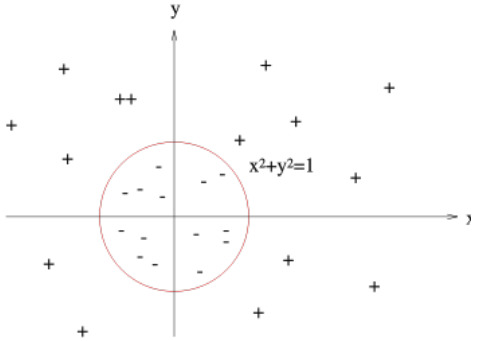
Plan

- Rappel : SVM
- Séparation hyperplane et modèle du neurone
- Neurone et réseau
- Théorème d'universalité
- Rappel : Compromis simplicité/complexité
- Méthodes par ensemble, double descente
- Descente de gradient Stochastique

Rappel : l'ancien paradigme

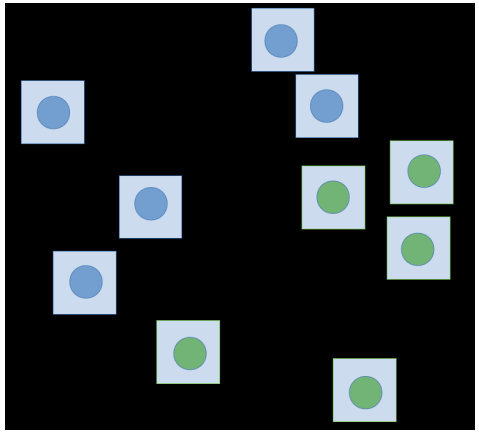
Vapnik Cherickov

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



sinon on ne peut pas apprendre

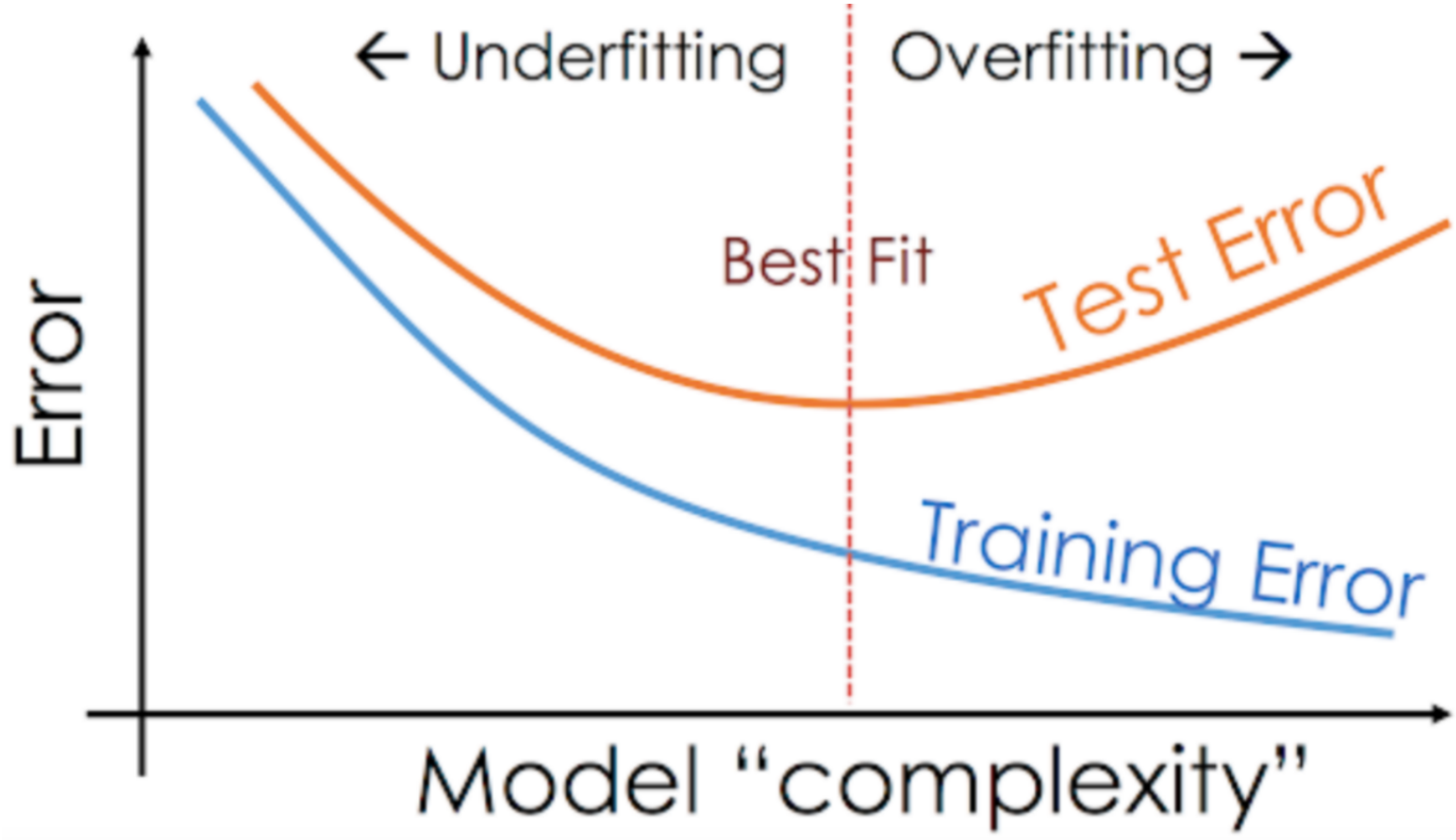
- ni trop de *paramètres*



sinon on apprend par coeur

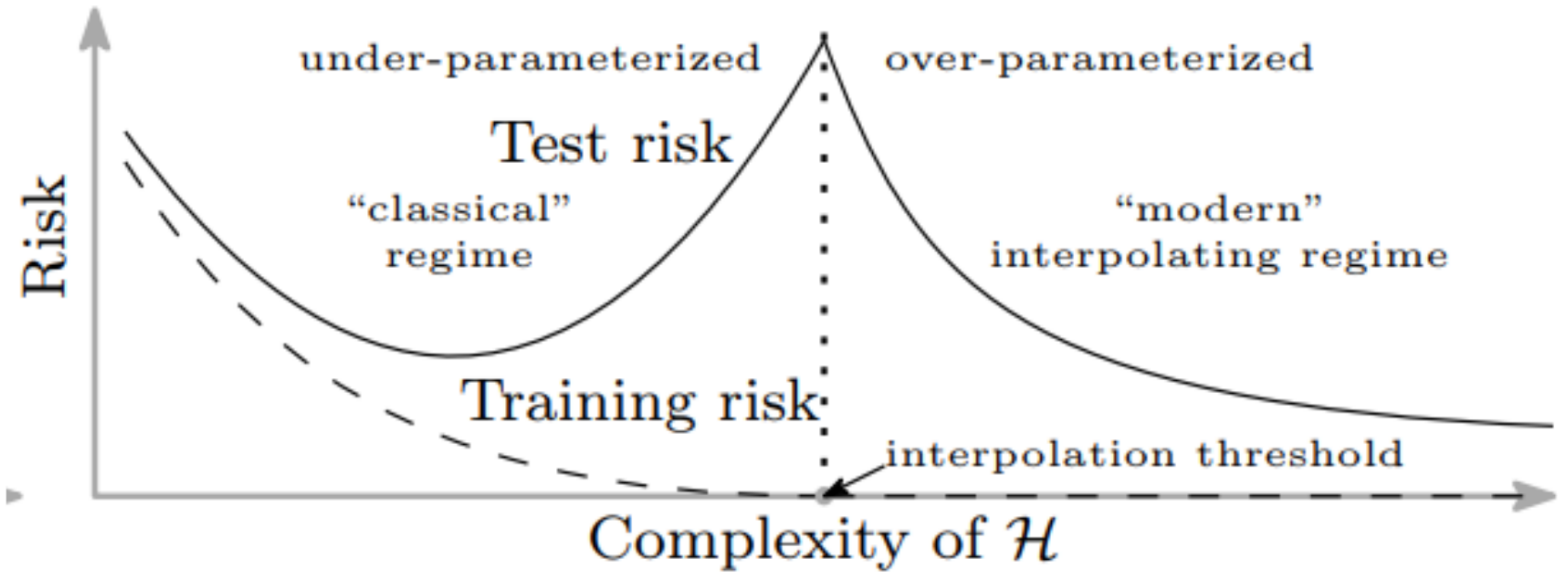
Rappel : l'ancien paradigme

Vapnik Cherickov



Le NOUVEAU paradigme

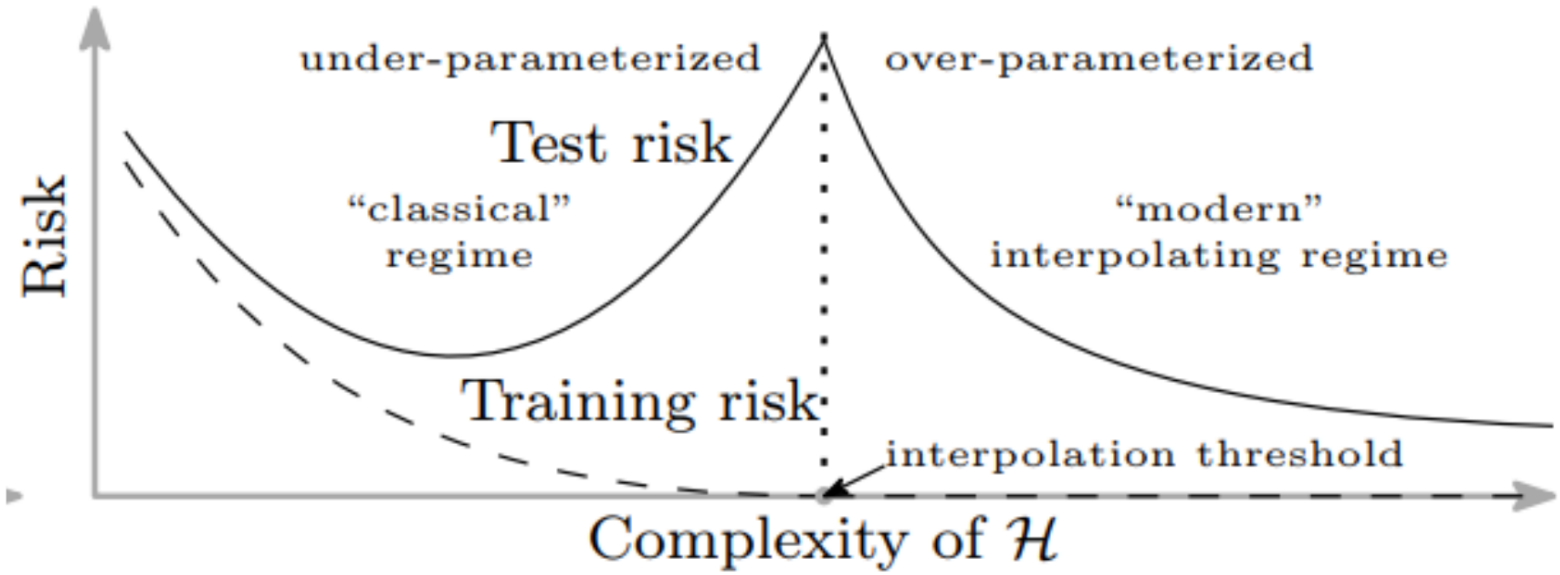
Double descente



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 ? ! ?

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

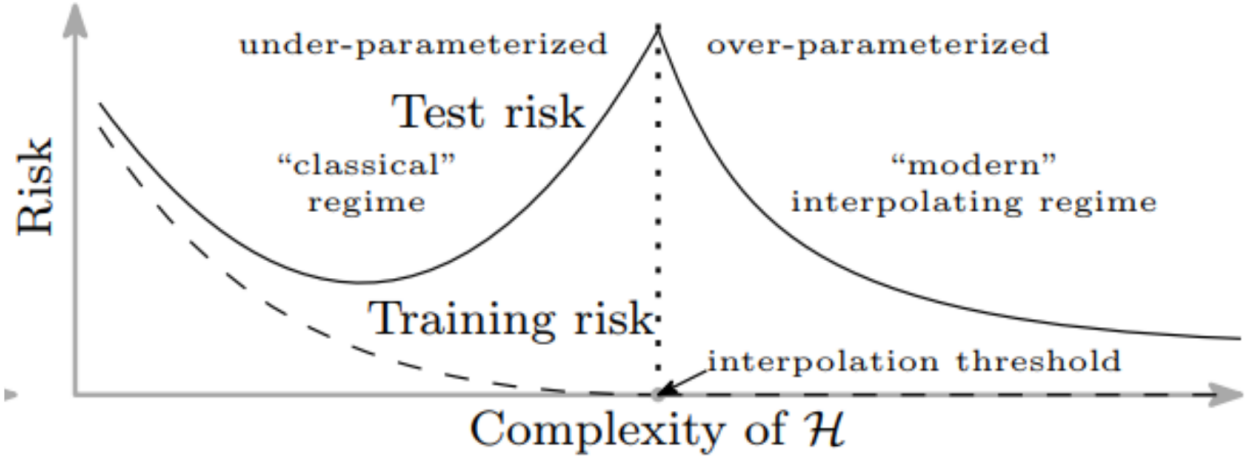
- K réseaux de Q neurones ce n'est pas comme 1 réseau de KQ neurones
- K ne crée pas d'overfitting
- Q trop petit on n'apprend pas, Q trop grand on overfit

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 !

ATTENTION

C'est une explication *plausible* - rien de plus !



D'où viendrait cette spécificité des réseau de neurones ?

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

Plan

- Rappel : SVM
- Séparation hyperplane et modèle du neurone
- Neurone et réseau
- Théorème d'universalité
- Rappel : Compromis simplicité/complexité
- Méthodes par ensemble, double descente
- 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 $F\left(u - \lambda \nabla F_u\right) \geq F(u)$ 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 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

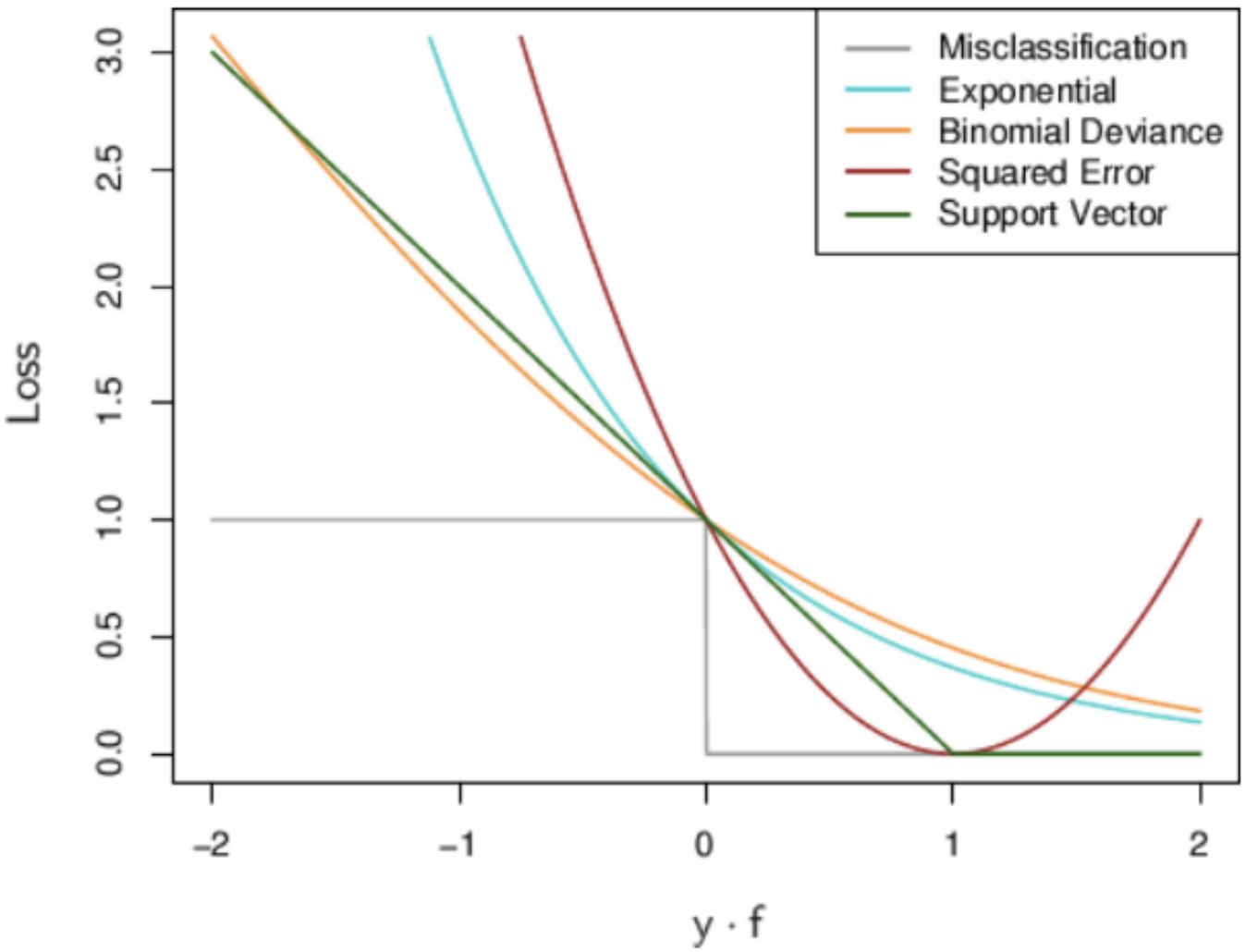
$$F(w) = loss\,(w) = \sum_n l(y(x_n)f(x_n, w))$$

- 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 $loss\,(u) = \sum_{i=1} q_i\,(u)$
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

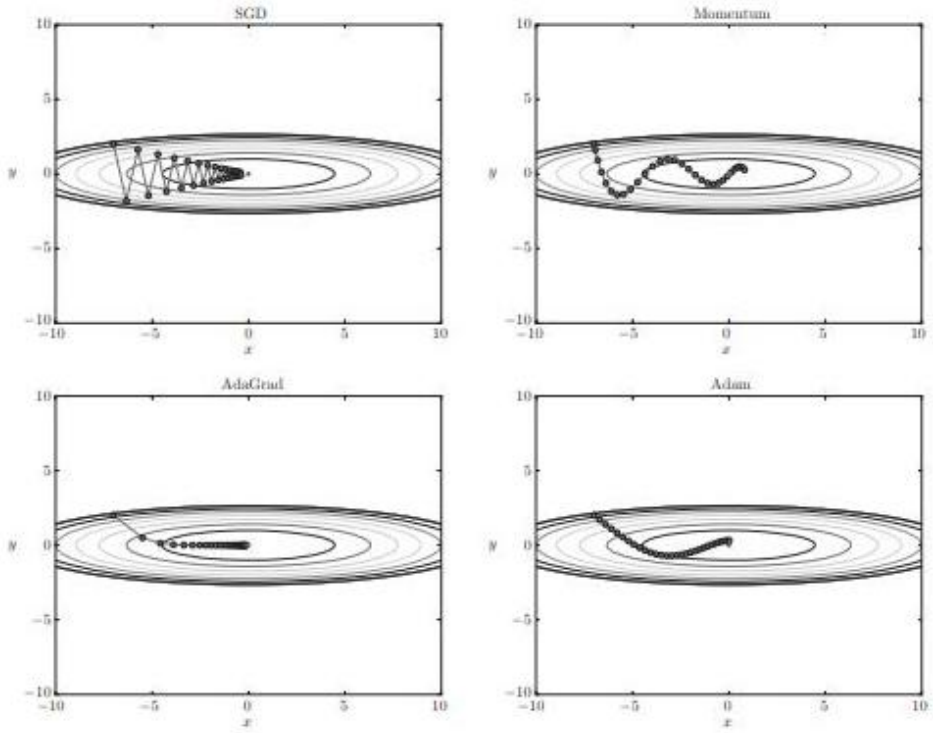
input : $x_1, y_1, ..., x_n, y_n,\ w_0$

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 :



Bilan

L'apprentissage en pratique

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

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

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

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])
```

Questions ?