# ENSTA : apprentissage automatique

## Introduction aux réseau de neurones

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

Adrien CHAN-HON-TONG
ONERA

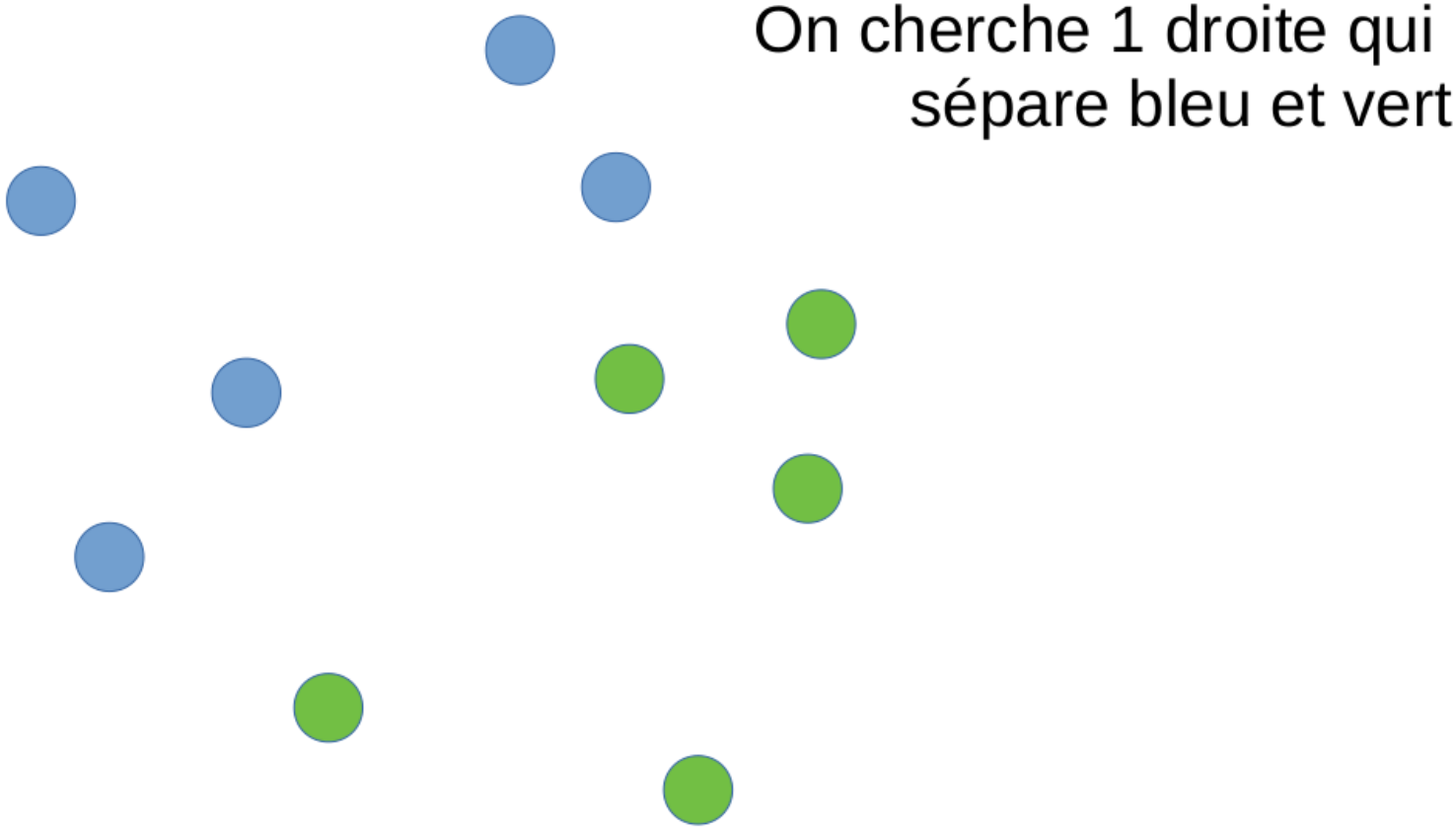
# Rappel : apprentissage vs test

## Apprentissage

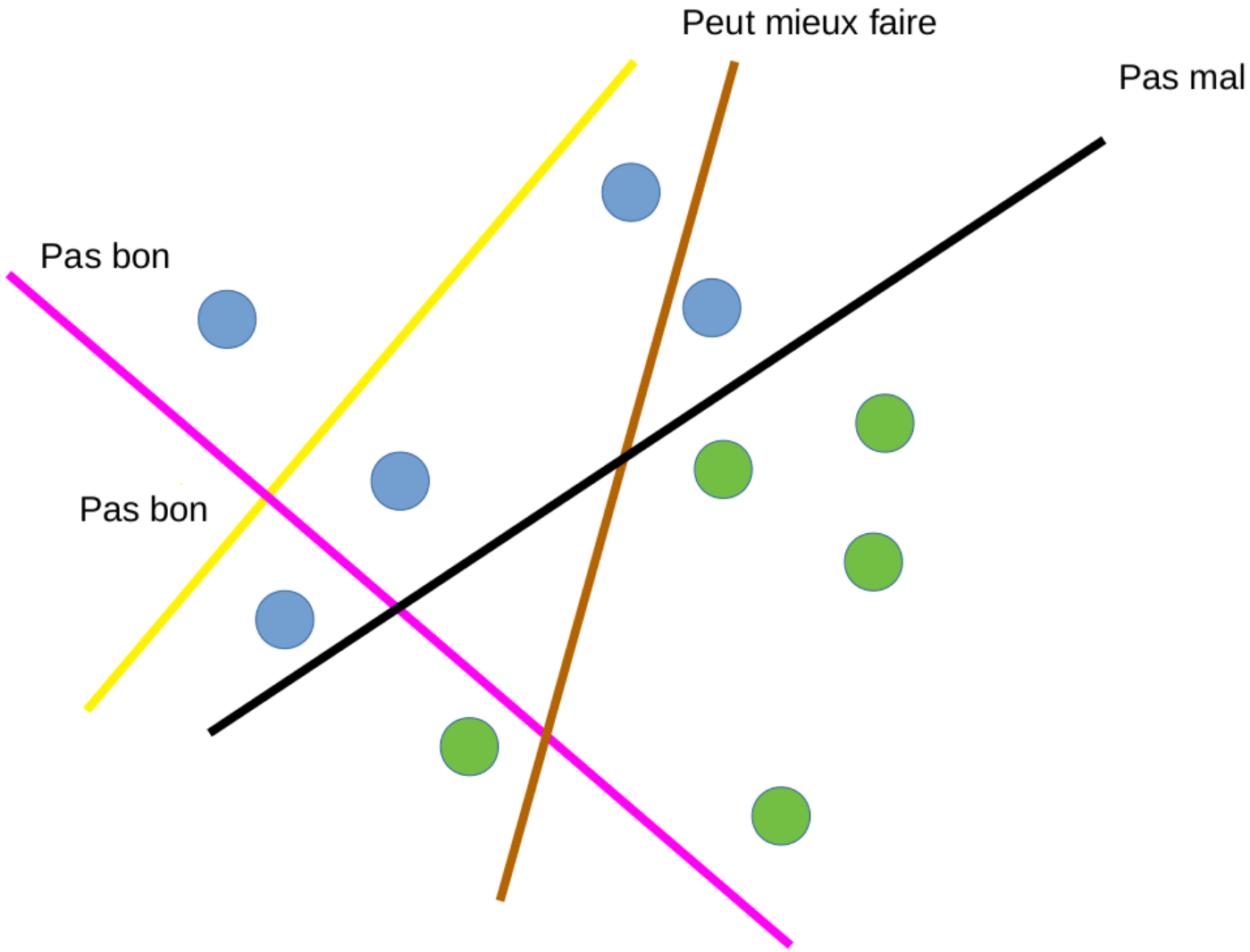
# Rappel : apprentissage vs test

Test et/ou production et/ou inférence

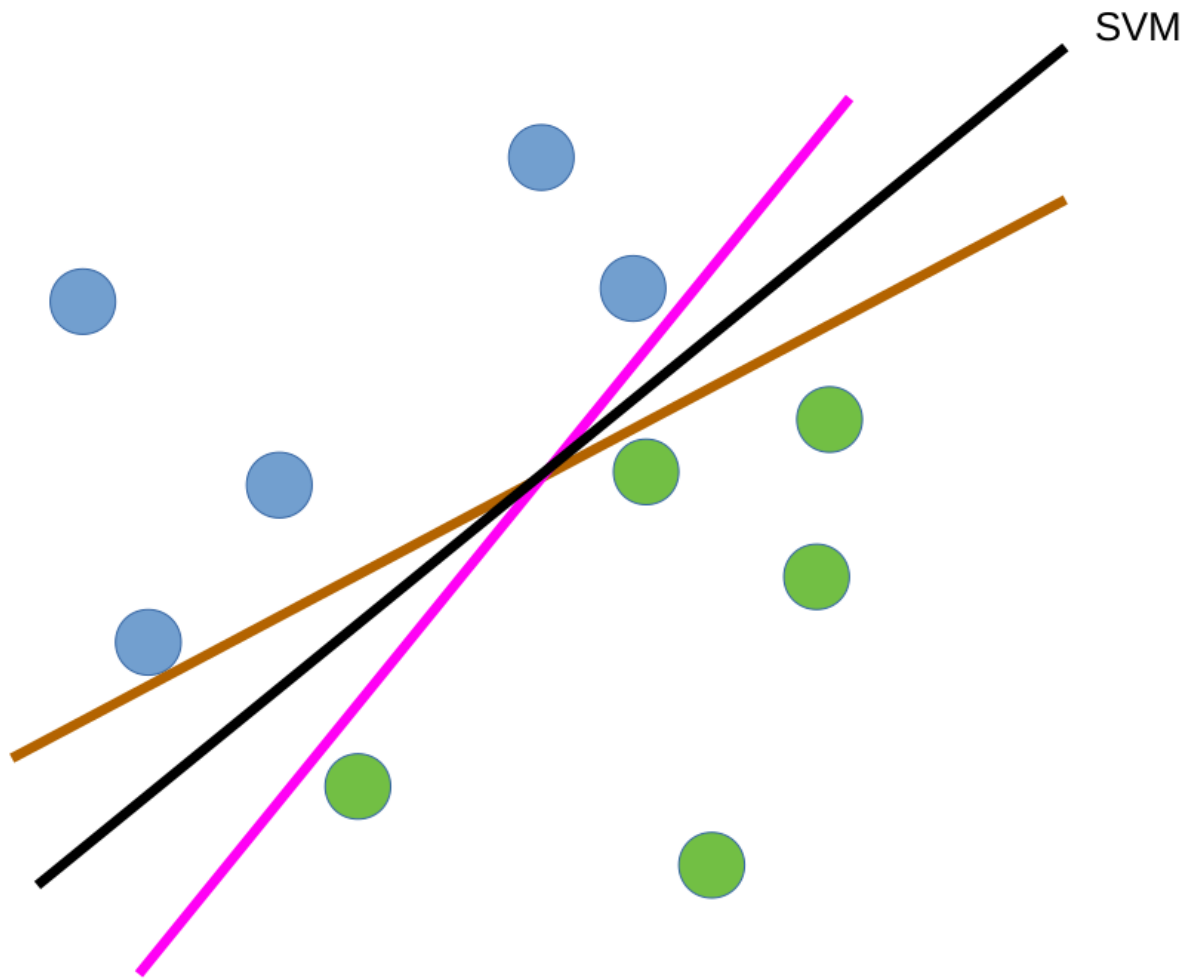
# Rappel : cas d'une séparation hyperplane



# Rappel : cas d'une séparation hyperplane



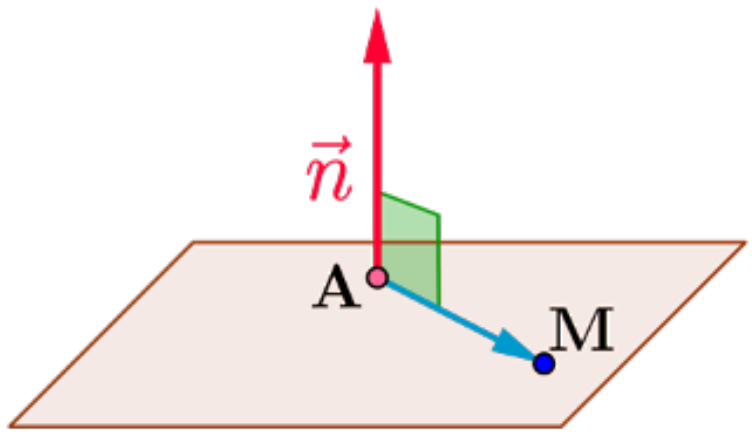
# Rappel : cas d'une séparation hyperplane



# 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 $\vec{n} \cdot \vec{AM} = 0$, si $\vec{n} \cdot \vec{AM} > 0$ on est au dessus et sinon en dessous.

# Rappel : cas d'une séparation hyperplane

## Linear feasibility

On a des points $x_1; :::; x_N \ 2 \ \mathrm{R}^D$ avec une couleur (bleu ou pas bleu) $(y_1; :::; y_N) \ 2 \ f \ \ 1; 1g^N$.
Et on cherche un plan $w; b$ tel que $y_n = 1 \ \ ! \ \ w^T x_n + b > 0$ et $y_n = \ \ 1 \ \ ! \ \ w^T x_n + b < 0$, ce qu'on peut résumer en

$$8n \ 2 \ f1; :::; Ng; \ y_n(w^T x_n + b) > 0$$

# Rappel : cas d'une séparation hyperplane

## Le SVM

On a des points $x_1; :::; x_N \ 2 \ \mathrm{R}^D$ avec une couleur (bleu ou pas bleu) $(y_1; :::; y_N) \ 2 \ f \ \ 1; 1g^N$.
Et le plan $w; b$ qui sépare les points en étant le plus distant possible

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

$$sc : 8n \ 2 \ f1; :::; Ng; \ y_n(w^T x_n + b) \quad 1$$

# 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

# 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 ?

) Parfois on ne peut PAS séparer les données avec un seul hyperplan

# Du neurone au réseau

## Le neurone

$$\textit{neurone}_{;} : \begin{matrix} \mathbb{R} & ! & \mathbb{R} \\ u & ! & :u + \end{matrix}$$

$2\ \mathbb{R}$ et $2\ \mathbb{R}$ 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 de neurones est une séquence de neurones prenant la même entrée, et, dont les sorties sont regroupées en 1 vecteur :

$$couche_{A;b} : \begin{array}{ccc} \mathbb{R} & ! & \mathbb{R} \\ u & ! & \begin{array}{c} 0 \\ @ \\ 1 \end{array} \begin{array}{c} neurone_{A_1;b_1}(u) \\ \vdots \\ neurone_{A\ ;b\ }(u) \end{array} \begin{array}{c} \\ \\ A \end{array} \end{array}$$

$A \in \mathbb{R}$ et $b \in \mathbb{R}$ sont les poids de l'ensemble des 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 :

$$A^0(Au + b) + b^0 = (A^0A)u + (A^0b + b^0)$$

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

$$A^0\text{activation}(Au + b) + b^0 \neq \text{activation}((A^0A)u + (A^0b + b^0))$$

# 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 des couches de neurones AVEC activation :

$$A^{0}\text{activation}(Au + b) + b^{0}$$

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 :

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

c'est à dire

$$reseau_w(x) = w_Q \quad relu(w_{Q\ 1} \quad relu(:::(relu(w_1 \quad x))))$$

$w_1; :::; w_Q$, $Q$ 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

$$f(x; w) = w_Q \quad relu(w_{Q\ 1} \quad relu(:::(relu(w_1 \quad x))))$$

# 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 $x \in \mathbb{R}$

$$|x| = relu(x) + relu(-x) = 1 \quad relu \quad \begin{matrix} 1 \\ -1 \end{matrix} \quad x$$

# Universalité des réseaux Relu

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

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

$$relu((0;1) \cdot x) + relu((0;-1) \cdot x) = 1 \; relu \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} x$$

# Universalité des réseaux Relu

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

$$\|x\|_1 = 1 \quad \text{relu}\left(\begin{pmatrix} 0 & 0 \\ & 1 \\ & 0 \\ & \vdots \\ & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ & \\ & \\ & \\ & \\ & \\ & \\ & \end{pmatrix} x \right) =$$

$$1 \quad \text{relu}\left( \begin{vmatrix} & \\ & \end{vmatrix} \quad x \right)$$

# Universalité des réseaux Relu

## Avec biais

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

$$\|x - q\|_1 = 1 \quad \text{relu}\left( \begin{vmatrix} & \\ & \end{vmatrix} x + \begin{matrix} q \\ q \end{matrix} \right)$$

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

$g_q(x) = 1 \quad \text{relu}(1 \quad \|x - q\|_1)$ vérifie

$g_q$ : $g_q(q) = 1$

$\forall x = \|x - q\|_1 > 1;\ g_q(x) = 0$

# Universalité des réseaux Relu

$8x_1; :::; x_N \ 2 \ Z^D$ tous distincts et $8(y_1; :::; y_N) \ 2 \ f \quad 1; 1g^N$,
il su t de $2 \quad N \quad D + N + 1$ neurones pour apprendre par coeur la base de données avec

$$f(x; w) = \sum_n y_n \quad \mathrm{relu}(1 \ jj \ x \quad x_n jj_1)$$

$$= (y_1 \ ::: \ y_N) \begin{pmatrix} 0 \\ \vdots \\ @ \end{pmatrix} \mathrm{relu}\left( \begin{pmatrix} I \\ \vdots \\ I \end{pmatrix} \mathrm{relu}\begin{pmatrix} 0 & 0 \\ \vdots & \vdots \\ @ & @ \end{pmatrix} \begin{pmatrix} I \\ \vdots \\ I \end{pmatrix} x + \begin{pmatrix} 1 \\ \vdots \\ A \end{pmatrix} \right) + \begin{pmatrix} 0 \\ \vdots \\ @ \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} + \begin{pmatrix} 1 \\ \vdots \\ A \end{pmatrix}$$

# 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
  - $f_w(x) = w_Q \quad \text{relu}(w_{Q\ 1} \quad \text{relu}(:::(\text{relu}(w_1 \quad x))))$
  - 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
  - $f_w(x) = w_Q \quad \text{relu}(w_{Q\ 1} \quad \text{relu}(:::(\text{relu}(w_1 \quad x))))$
  - 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 classi er 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 di érents modèles
- je tends alors vers les performances d'un modèle moyen

# Méthodes ensemblistes

- si j'apprends $K$ 1 fois
  - j'ai $x_1, \ldots, x_N, y_1, \ldots, y_N$
  - je forme $w_1, \ldots, w_K$
- 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 $w_1 = \ldots = w_K = w$ )

# 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'over tting !

# La spéci cité 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'over tting
- $Q$ trop petit on n'apprend pas, $Q$ trop grand on over t

## 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 spéci cité 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
$\forall u, h \in \mathbb{R}^D, F(u+h) = F(u) + \nabla F_u \cdot h + o(h)$
avec $o(h) \underset{h \to 0}{\to} 0$ (notation petit o classique)
Donc si $\nabla F_u \neq 0$ alors il existe $\epsilon > 0$ tel que $F(u - \epsilon \nabla F_u) < F(u)$

# La descente de gradient

## pseudo code

input : $F$, $u_0$

1. $u = u_0$
2. calculer $r$ $F_u$
3. si $r$ $F_u$ $0$ ou early stopping alors sortir
4. $= 1$
5. tant que $F(u \quad r \; F_u) \quad F(u)$ faire $= 0{:}5$
6. $u = u \quad r \; F_u$
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 $F(u - \lambda \nabla F_u) \geq F(u)$ 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 $r F_u = 0$

# 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 :
  $F(w) \quad \sum_{n}^{p} 1 \quad (y(x_n) f(x_n; w))$
  avec $1 \quad (t) = 1$ si $t \quad 0$ et $1 \quad (t) = 0$ si $t > 0$

# Apprentissage et descente de gradient

## Test :

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

## Apprentissage :

On prend $x_1; :::; x_N$, et, on doit approximer

$$\min_w \sum_n 1 \quad (y(x_n) f(x_n; w))$$

# Fonction de perte

La descente de gradient ne marche qu'avec des fonctions globalement lisse.
Utiliser $F(w) = \min_w \sum_n 1 \quad (y(x_n) f(x_n; w))$ ne peut pas marcher

# Fonction de perte

La descente de gradient ne marche qu'avec des fonctions globalement lisse.
Utiliser $F(w) = \min_w \sum_n 1 \quad (y(x_n) f(x_n; w))$ 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

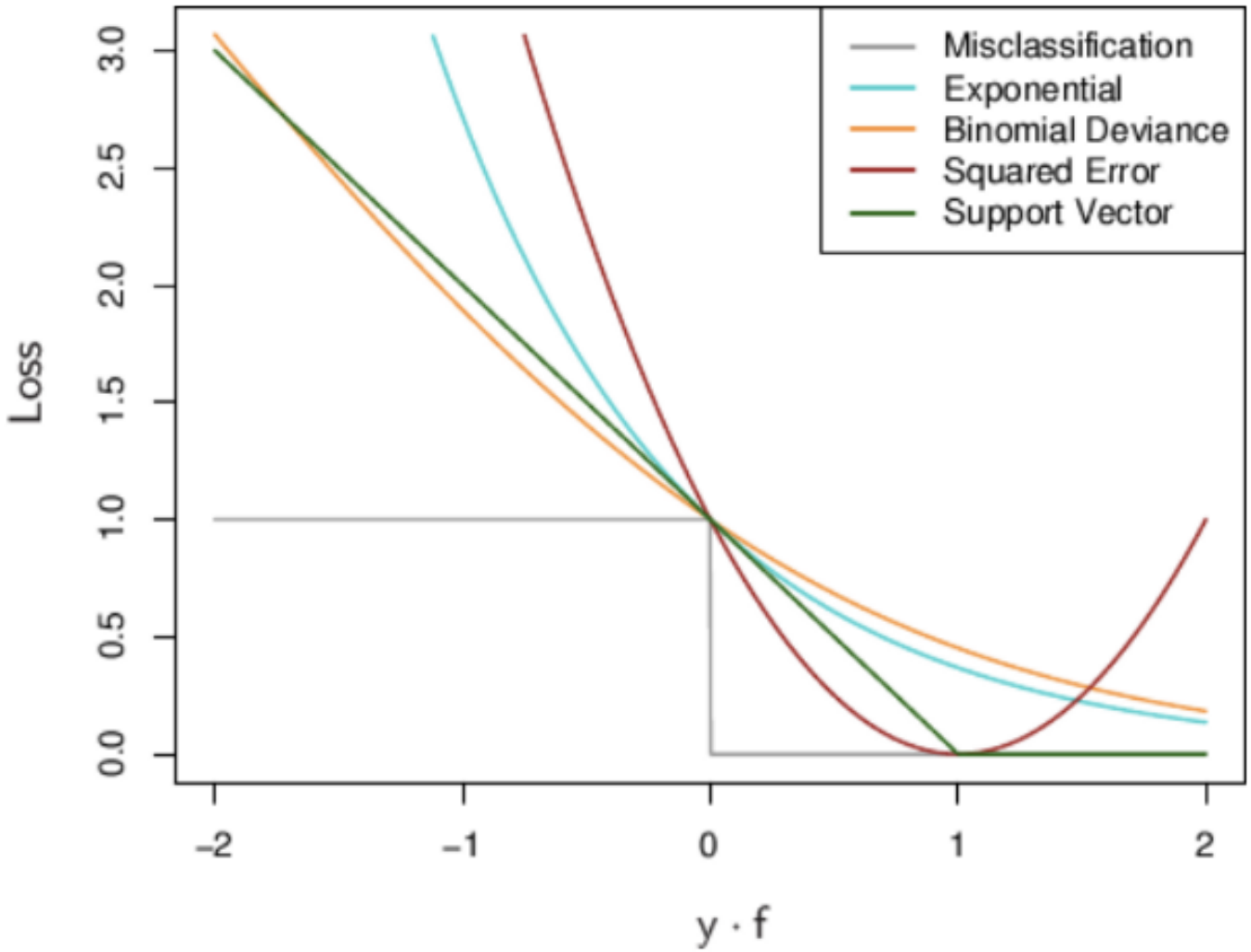
$$F(w) = loss(w) = \sum_n l(y(x_n) f(x_n; w))$$

- $l$ 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 :

$$loss(w) = \sum_n relu(1 \quad y_n f(x_n; w))$$

# Fonction de perte



## Limite de la descente de gradient

$$loss\,(w) = \sum_{n} relu(1 \quad y_n f\,(x_n; w))$$

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 $\mathit{loss}(u) = \sum_{i=1}^{p} q_i(u)$
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 $(t)$ fixée a priori (qui doit quand même vérifier certaines conditions).

# Descente de gradient stochastique

## pseudo code

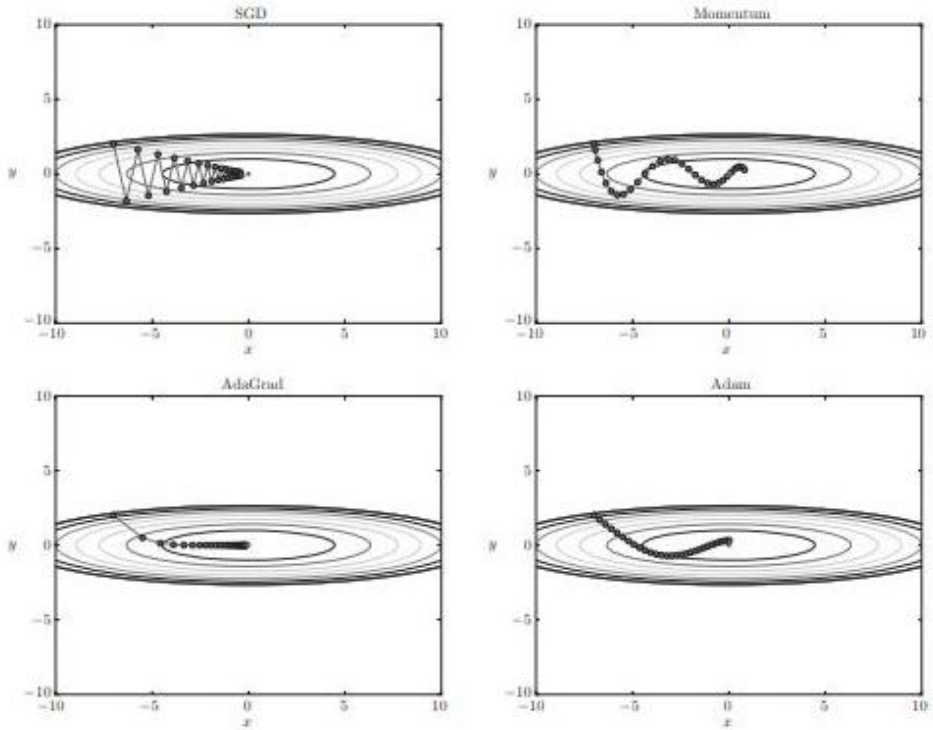
input : $x_1; y_1; :::; x_n; y_n$, $w_0$

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

# Descente de gradient stochastique

## Optimizer

$w = w \quad _{iter}\, r\, _w partial\_loss$ est une possibilité mais il y en a d'autres :



# Bilan

## L'apprentissage en pratique

$$f(x;w) = w_Q \quad relu(w_{Q\ 1} \quad relu(:::(relu(w_1 \quad x))))$$

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

$$partial\_loss(w) = \sum_{n2Batch} relu(1 \quad y_n f(x_n;w))$$

$$w = w \quad _{iter}\, r\, _w partial\_loss$$

# Forward - Backward

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

# Forward - Backward

## objectif

$$partial\_loss(w) = \sum_{n2Batch} relu(1 \quad y_n f(x_n; w))$$

avec $f(x; w) = w_Q \quad relu(w_{Q\ 1} \quad relu(:::(relu(w_1 \quad x))))$

) on veut calculer

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

# 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 $w$ -

$$\frac{@loss}{@w_{t;i;j}} = \frac{@loss}{@\ _{t;i;j}} \frac{@\ _{t;i}}{@w_{t;i;j}} = \frac{@loss}{@\ _{t;i}} x_{t;j}$$

# Forward - Backward

## Réduction -

$$\frac{@loss}{@\ _{t;j}} = \sum_i \frac{@loss}{@\ _{t+1;i}} \frac{@\ _{t+1;i}}{@\ _{t;j}} = \sum_i \frac{@loss}{@\ _{t+1;i}} w_{t;i;j} relu'(\ _{t;j})$$

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

# Forward - Backward

## Attention

La somme dans $\frac{@loss}{@\ _{t;j}} = \sum_i \frac{@loss}{@\ _{t+1;i}} \frac{@\ _{t+1;i}}{@\ _{t;j}}$ ne vient pas de la somme dans $\ _{t+1;i} = \sum_j x_{t;j} w_{t;i;j}$.

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

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 ?