agaetis
Data Science & Big Data

Introduction à l'apprentissage automatique

Léo Beaucourt

Contenu du cours

À propos de ce cours

- Introduction à l'apprentissage automatique (ou *machine learning* en anglais)
- En pratique: *Python*, *Jupyter*. Packages *numpy* *pandas*, *matplotlib*, *scikit-learn* et *tensorflow*.
- Pas de pré-requis mathématiques (à part les dérivées partielles, l'algébre linéaire, ...)
- Largement inspiré de l'excellent (et complet!) cours de **Andrew Ng** sur Coursera.

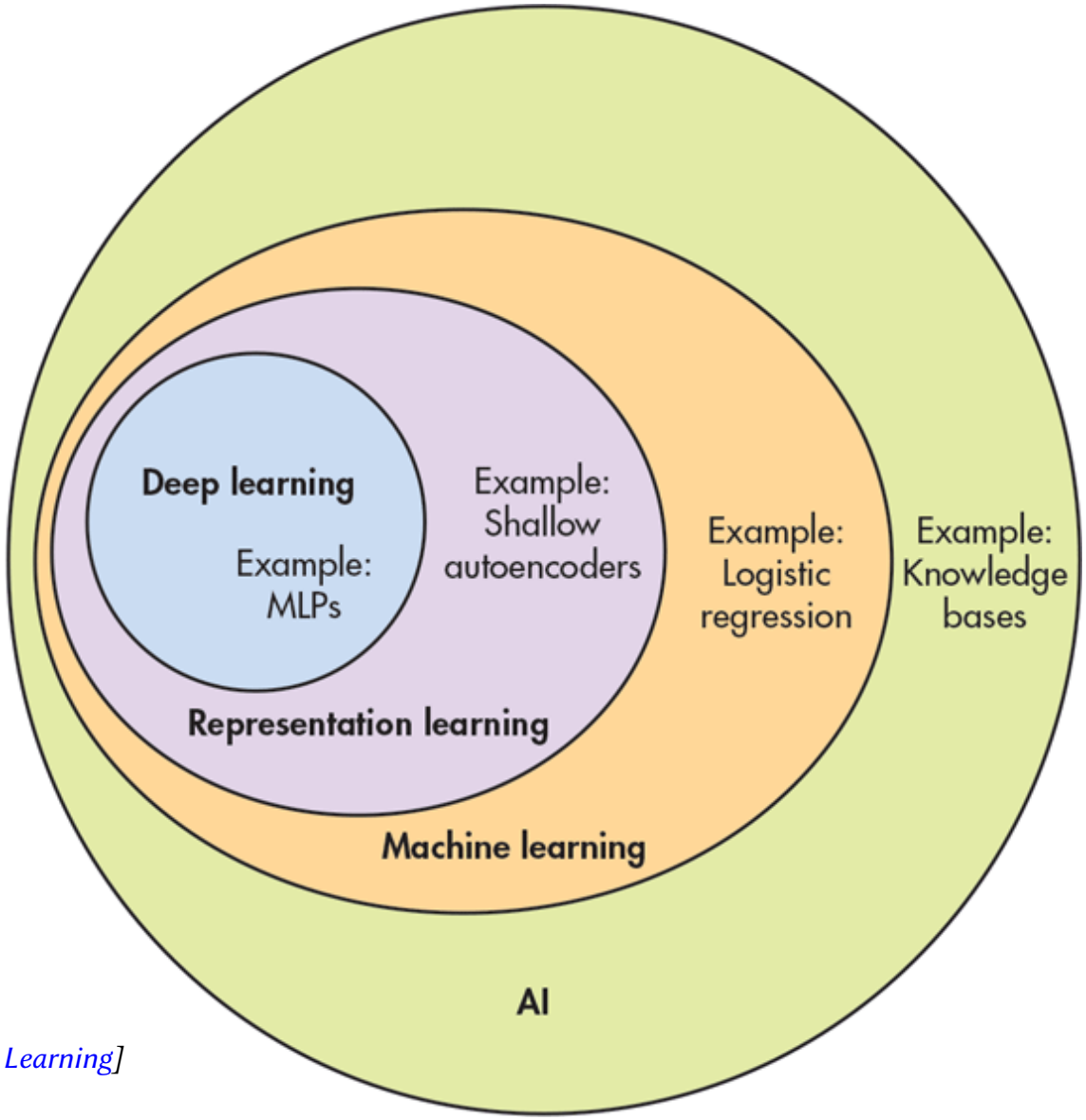
Allez, on démarre en douceur!

1. Intro: Qu'est ce que le *Machine Learning*

- **Arthur Samuel:**
  - *The field of study that gives computers the ability to learn without being explicitly programmed.*
- **Tom Mitchell:**
  - *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

- **L'idée:** Une machine apprend *seule* à réaliser une tache complexe à l'aide de processus itératifs simple.

1. Intro: Qu'est ce que le *Machine Learning*



[From MIT Press book Deep Learning]

1. Intro: Qu'est ce que le *Machine Learning*

- Les principaux types d'apprentissage:

Supervisé

- Utilise des données *labélisées*
- La machine apprend par l'exemple
- *Prédis* le résultat pour de nouveaux événements
- Problèmes de régression et de classification
- Regression linéaire et logistique
- Réseaux de Neurones
- Arbres de décisions

Non-supervisé

- Données non *labélisées*
- La machine apprend par elle même à indentifier une structure
- Évaluation des performances compliqué.
- Problèmes de classification, réduction de dimensions
- K-means
- Analyse en Composante Principale

Par renforcement

- Un agent A, effectue une action Ac, l'environnement E lui renvoie une récompense.
- Récompenses à court et long terme
- Utilisé par Deepmind (alphaGo)

2. L'apprentissage supervisé

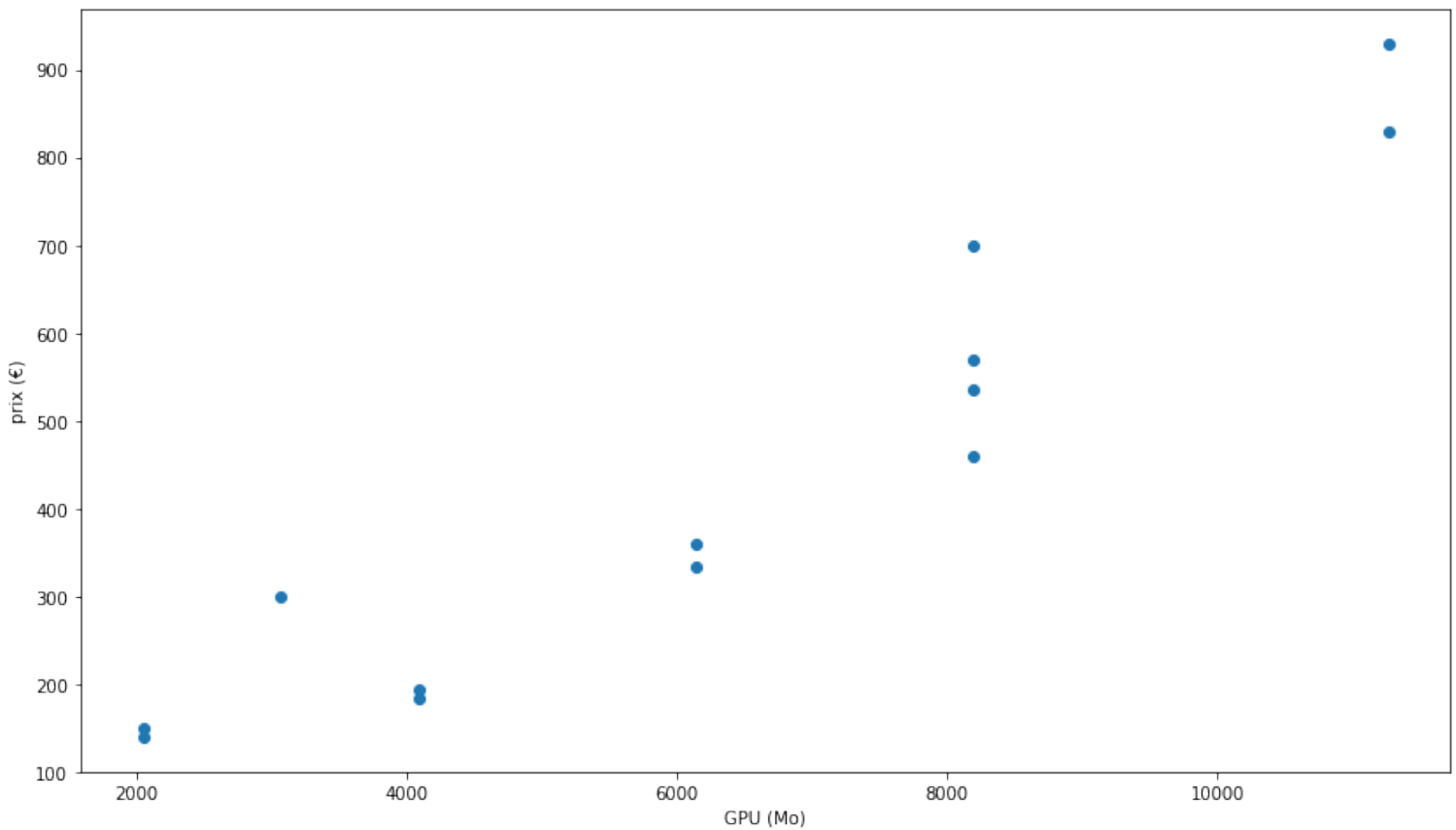
- Utilise des données *labélisées*
- La machine apprend par l'exemple
- *Prédis* le résultat pour de nouveaux événements
- Problèmes de régression et de classification
- Regression linéaire et logistique
- Réseaux de Neurones
- Arbres de décisions

2.1 La régression linéaire

- Mais... À quoi ça sert en vrai?
  - Prédiction d'une valeur en fonction de paramètres (prix de quelque chose)
  - Très utilisé en science (physiques des particules, sciences sociales) pour mettre en évidence des relations entre des variables ou ajuster un modèle
  - Dans le domaine médicales: les études épidémiologique
  - Dans la finance: prédictions des tendances, *Capital Asset Pricing Model*
  - ...

2.1 Un exemple: le prix d'une carte graphique

- Ce prix va dépendre de la taille de la mémoire vive (GPU) de la carte (entre autre ...)
- On a un jeu de données, cad une liste de carte graphique dont on connait le couple $\{GPU; prix\}$:



2.1 Construire un modèle (regression linéaire)

- Appelons x_1 la taille de la mémoire vive de nos m carte graphiques, et y le prix correspondant.
- On cherche à trouver le modèle qui permet de prédire un prix $\hat{y}$ à partir x_1:

$$\hat{y} = h_\theta(x_1)$$

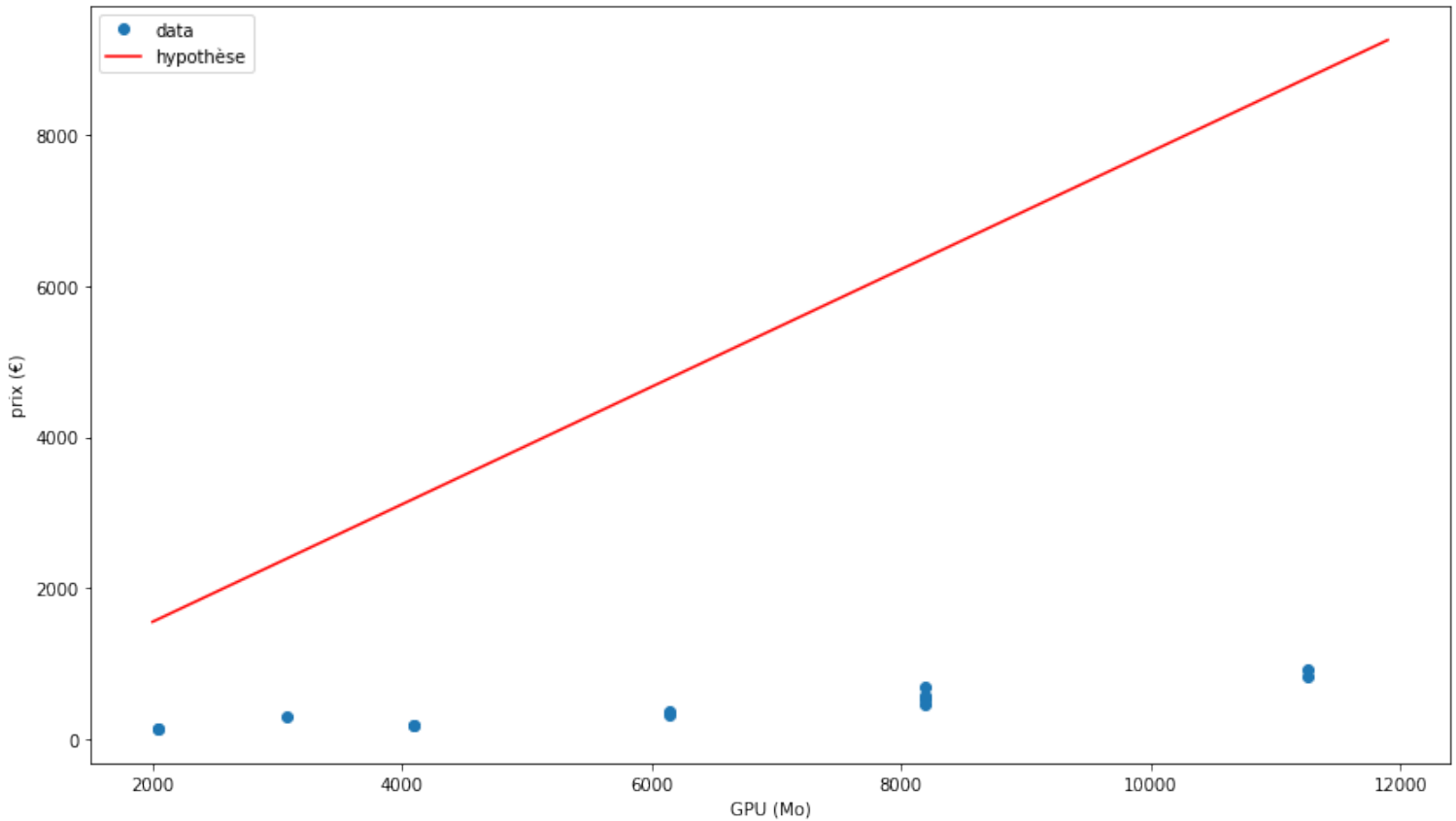
- On défini le paramètre θ_1 qui va *lier* x_1 à $\hat{y}$:

$$h_\theta(x) = \theta_1 x_1$$

- Rappel math: **fonction linéaire** $f(x) = kx$

2.1 Construire un modèle (regression linéaire)

- Initialisons aléatoirement la valeur de θ_1

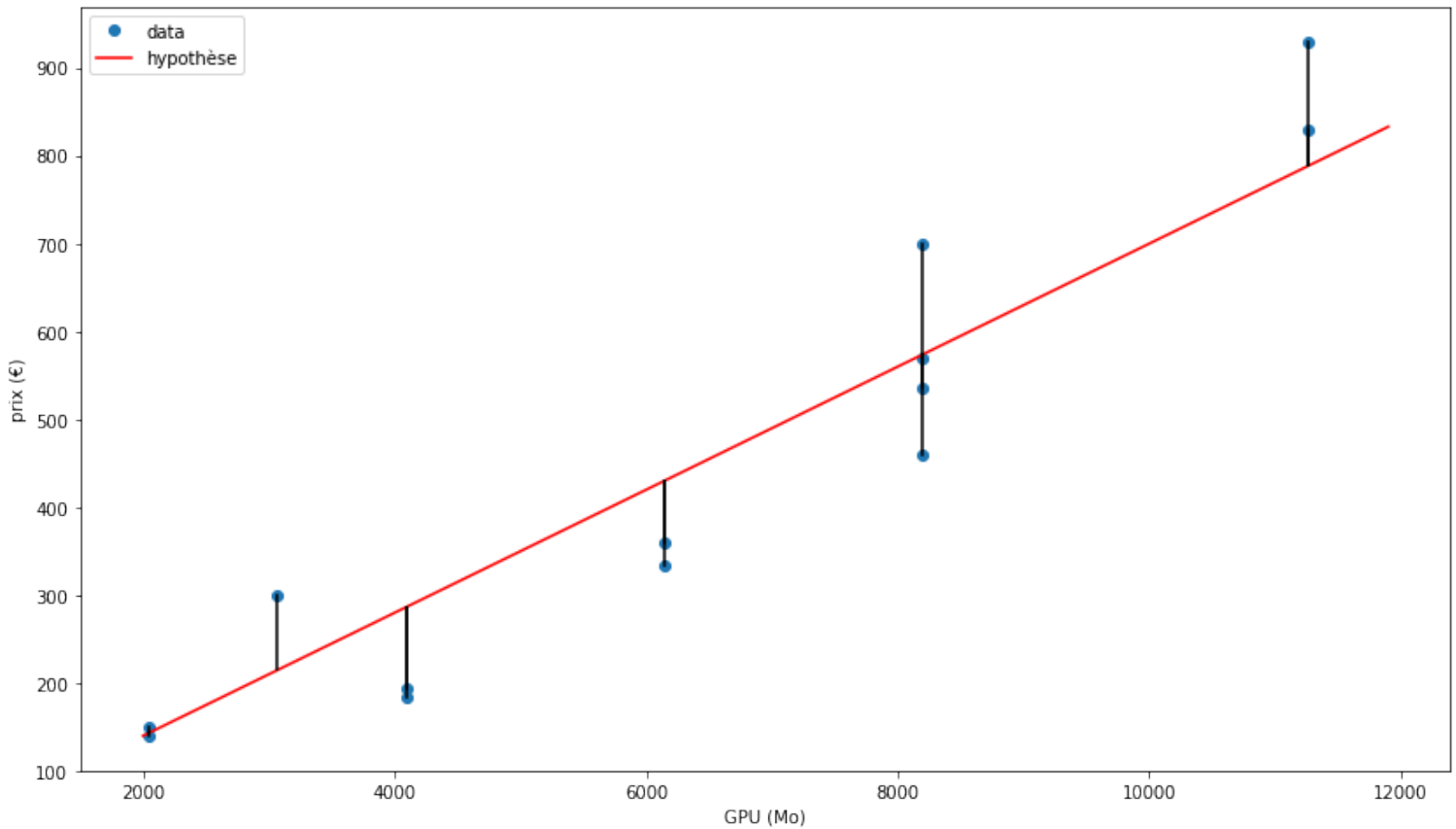


- C'est pas encore ça ...

2.1 La fonction de coût

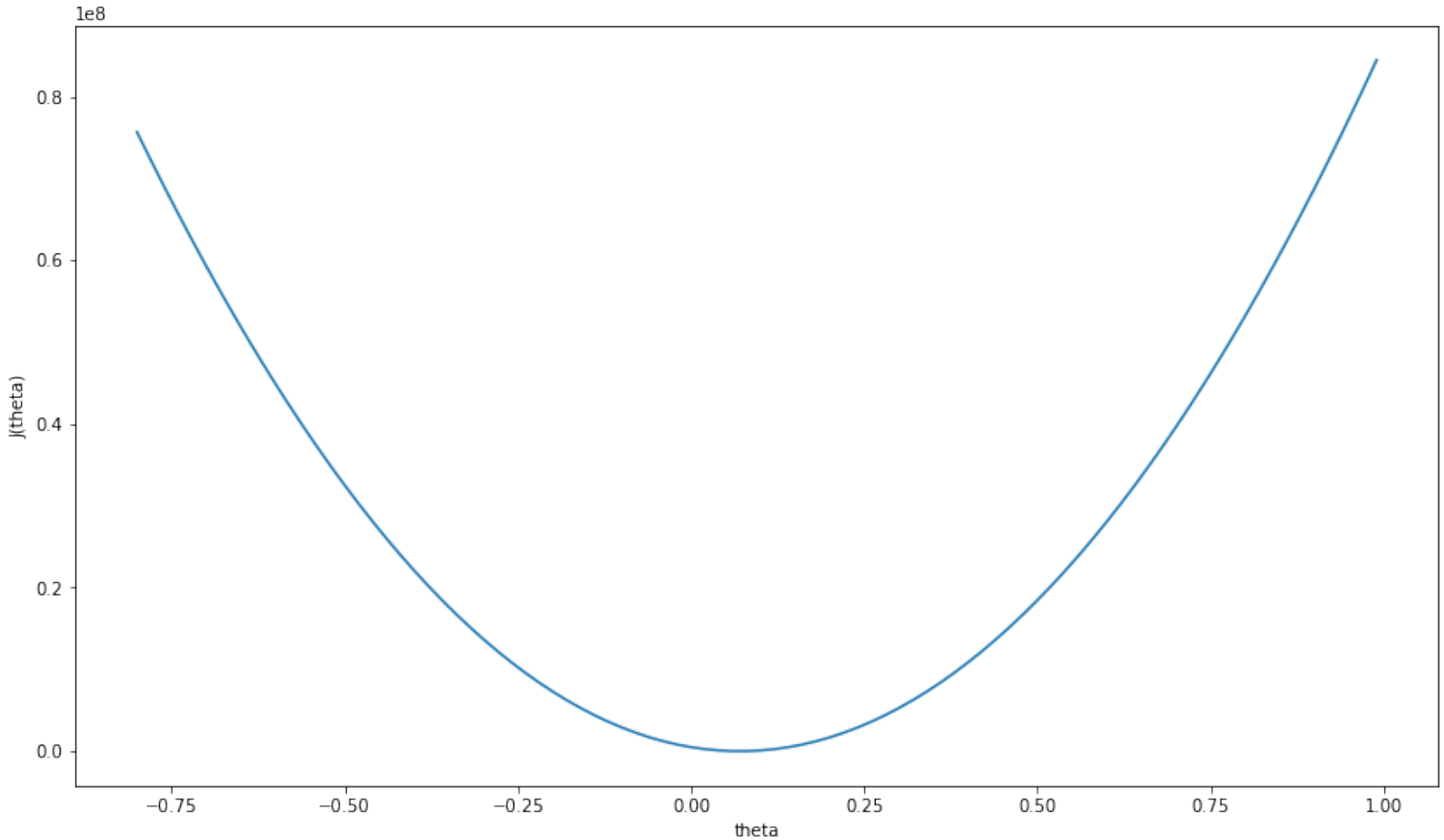
- Comment estimer la *véracité* de notre modèle?
  - La **Fonction de coût**: $\mathcal{J}(\theta)$
  - Une définition possible: somme quadratique des erreurs

$$\mathcal{J}(\theta) = \frac{1}{2m} \sum_{i=0}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$



2.1 La fonction de coût

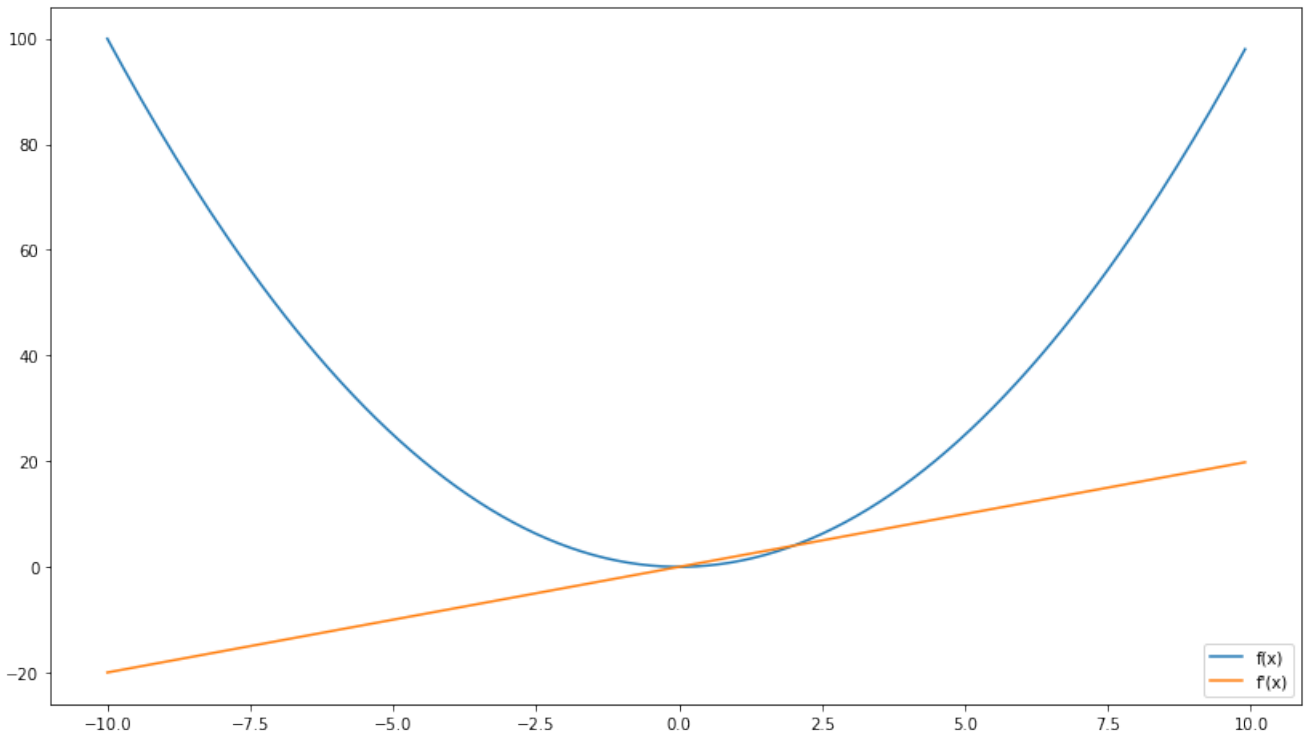
- On cherche à trouver la valeur de θ_1 qui **minimise** $\mathcal{J}(\theta)$
- En Brute ...



- ... essayons d'optimiser

2.1 La descente de gradient

- C'est l'algorithme qui va nous permettre d'arriver "*rapidement*" au minimum de $\mathcal{J}(\theta)$
- On va utiliser la *dérivation*: $\frac{d}{d\theta_1}\mathcal{J}(\theta)$:
  - Si $\mathcal{J}(\theta)$ est croissant: $\frac{d}{d\theta_1}\mathcal{J}(\theta) > 0$
  - Si $\mathcal{J}(\theta)$ est décroissant: $\frac{d}{d\theta_1}\mathcal{J}(\theta) < 0$



2.1 La descente de gradient

- (Encore) un peu de math, la descente de gradient s'écrit:

Descente de gradient

Répéter jusqu'à convergence: $\{$

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} \mathcal{J}(\theta)$$

$\}$

- α s'appelle le taux d'apprentissage (*learning rate*) et c'est le **seul** paramètre de l'algorithme.
- On va itérativement modifier la valeur de θ_1 en fonction de la dérivée de $\mathcal{J}(\theta)$, jusqu'à minimiser $\mathcal{J}(\theta)$ (*convergence*).

2.1 La descente de gradient

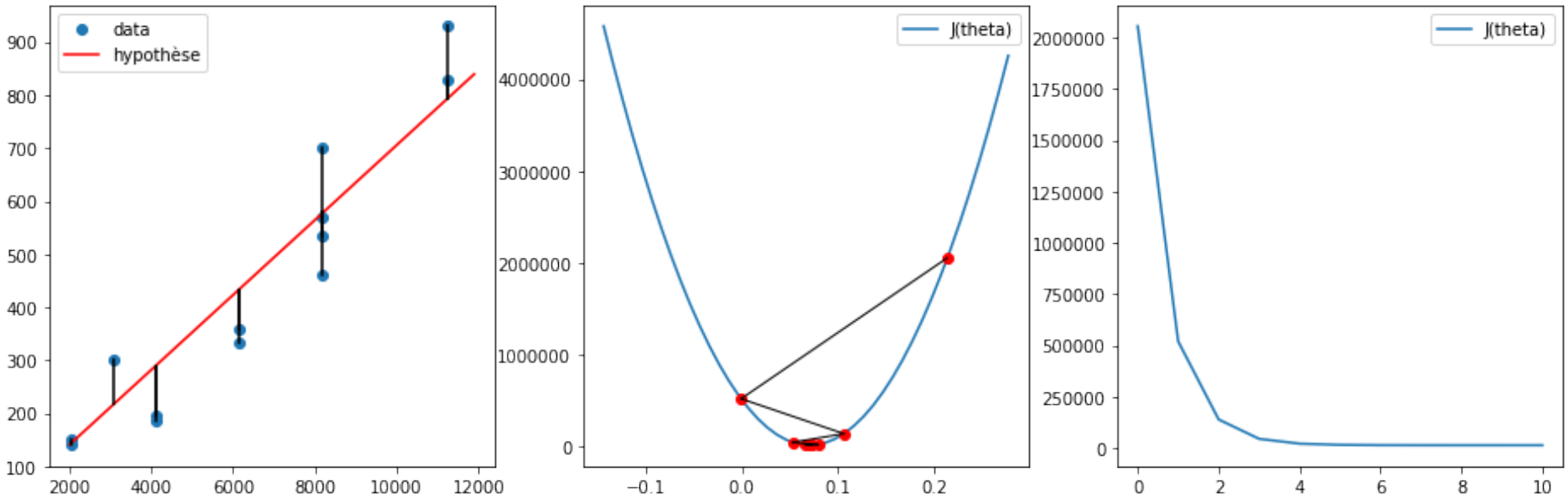
- Dérivons donc notre fonction de coût:

$$\mathcal{J}(\theta) = \frac{1}{2m}\sum_{i=0}^{m}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m}\sum_{i=0}^{m}(\theta_1 x_1^{(i)} - y^{(i)})^2$$

$$\frac{d}{d\theta_1}\mathcal{J}(\theta) = \frac{1}{m}\sum_{i=0}^{m}(\hat{y}^{(i)} - y^{(i)})x_1^{(i)}$$

- Un peu de *hand-tunning*:
  - Le learning rate (α) est fixé à 0.00000003
  - Défissons une précision $\epsilon = 0.001$ qui nous servira a arrêter la descente de gradient

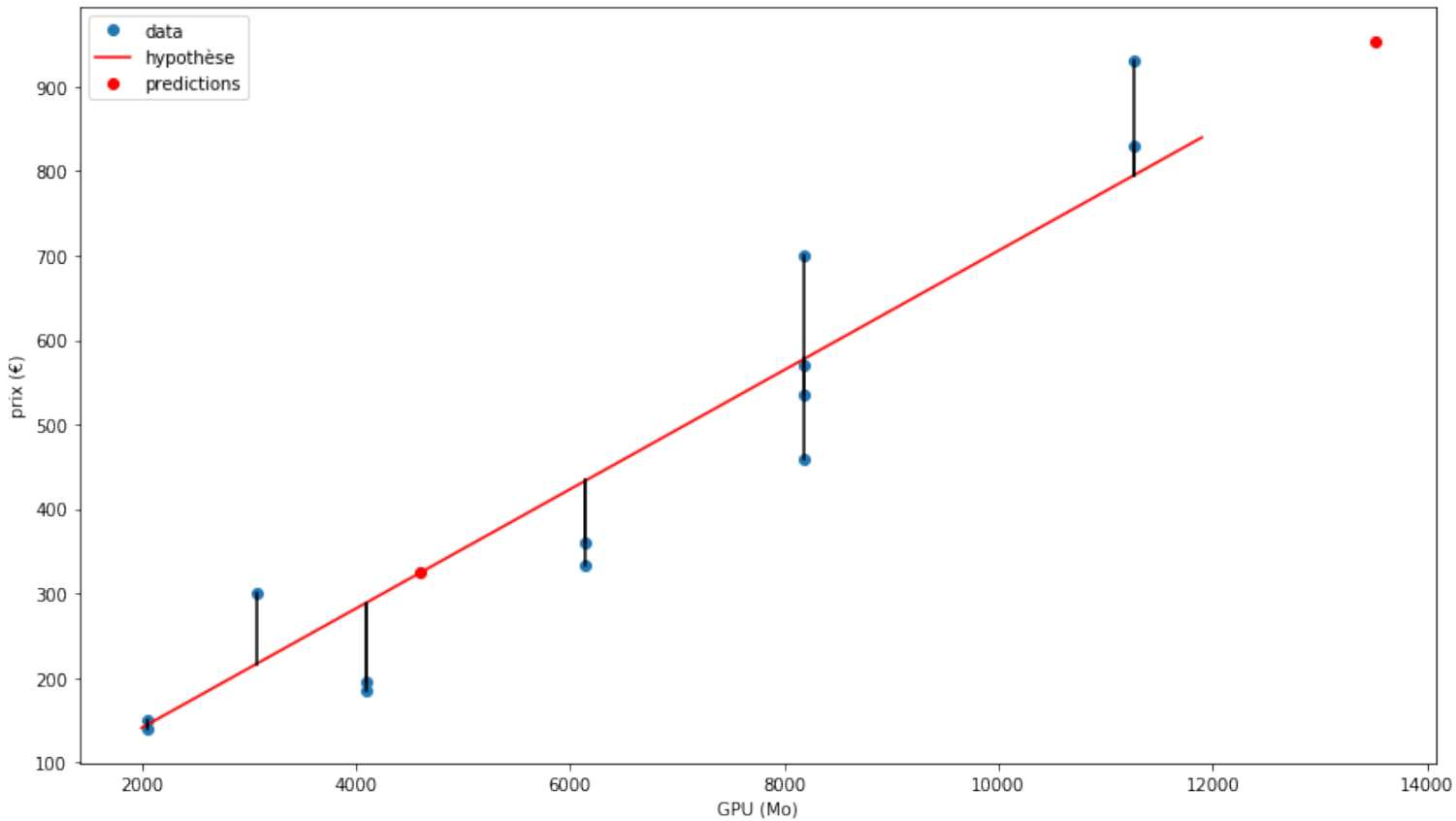
2.1 C'est parti !



- La descente de gradient c'est achevée au bout d'une dizaine d'itérations
- La valeur de notre paramètre θ_1 est 0.0706
- On peut voir que $\mathcal{J}(\theta)$ a continuellement diminué à chaque itération

2.1 On peut maintenant faire une prédiction

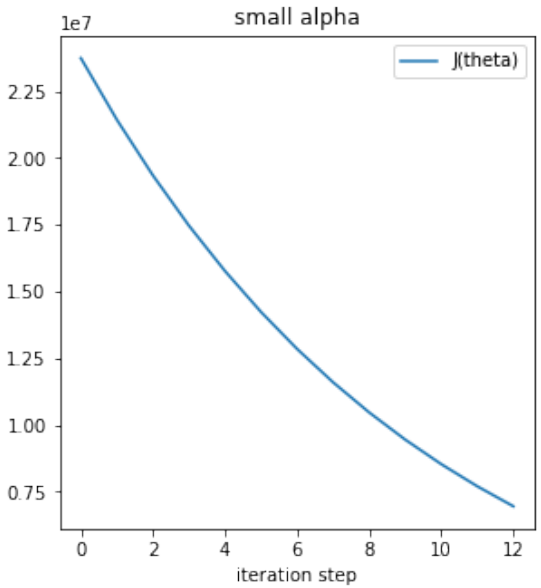
- Quel serait le prix d'une carte avec 4608 et 13516 Mo de GPU? (ce qui n'a pas de sens, on est d'accord)

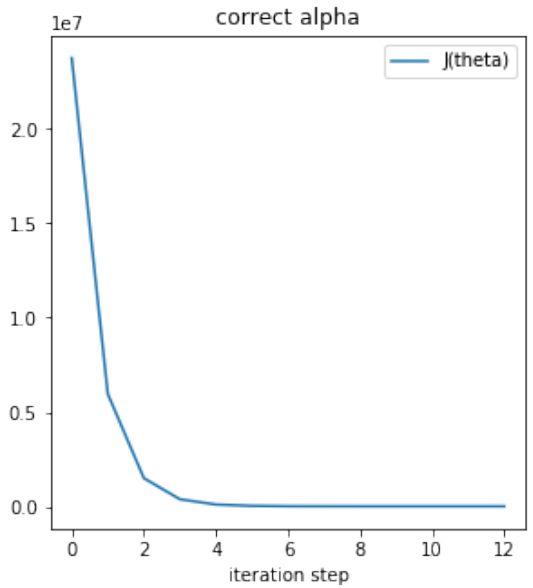


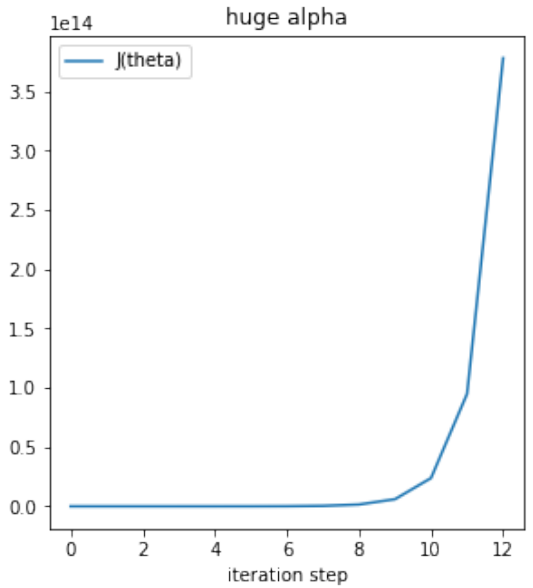
- On pourra les vendre autour de 325.10 et 953.62 euros !

2.1 Le choix du taux d'apprentissage

- **Learning Rate Très important:**
  - α **Trop grand:** la descente de gradient diverge
  - α **Trop petit:** la descente de gradient est très longue
- Pour choisir, on regarde l'évolution de la fonction de coût $\mathcal{J}(\theta)$ en fonction du nombre d'itérations:







2.1 Un mot sur la regression linéaire multivariables

- Ici, nous avons vu le cas avec une seule variable x_1 (on aurait pu rajouter un biais θ_0, cad un terme constant: $\hat{y} = \theta_0 + \theta_1 x_1$)
- Le principe est le même, mais avec plusieurs variables x_i (donc plusieurs paramètres θ_i)
- Notre fonction hypothèse s'écrie alors:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

- **Astuce:** On définit $x_0 = 1$, et on re-écrit la fonction hypothèse:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 \cdots + \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i$$

- La fonction de coût reste inchangée

2.1 Un mot sur la regression linéaire multivariables

- Dans le cas multivariables, la descente de gradient devient:

Descente de gradient (cas multivariables)

Répéter jusqu'à convergence: $\{$

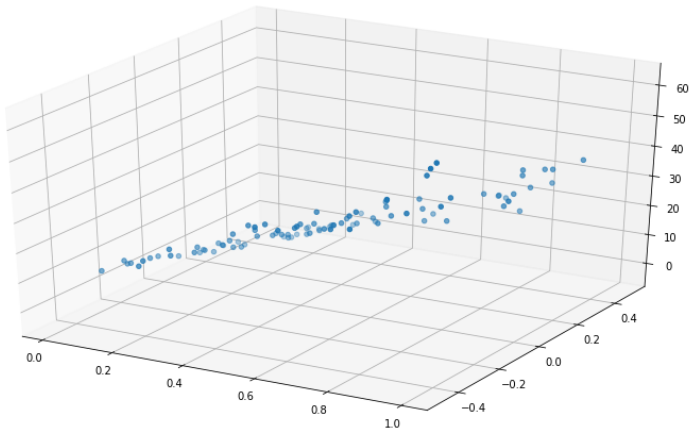
$$\theta_0 := \theta_0 - \alpha \frac{d}{d\theta_0} \mathcal{J}(\theta)$$
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} \mathcal{J}(\theta)$$
$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} \mathcal{J}(\theta)$$
$$\dots$$

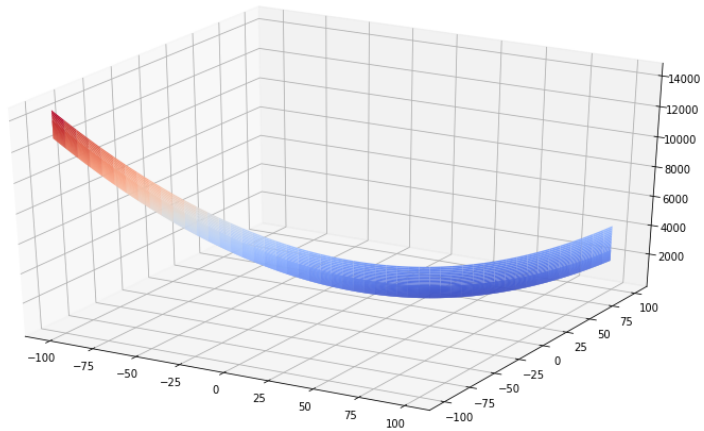
$\}$

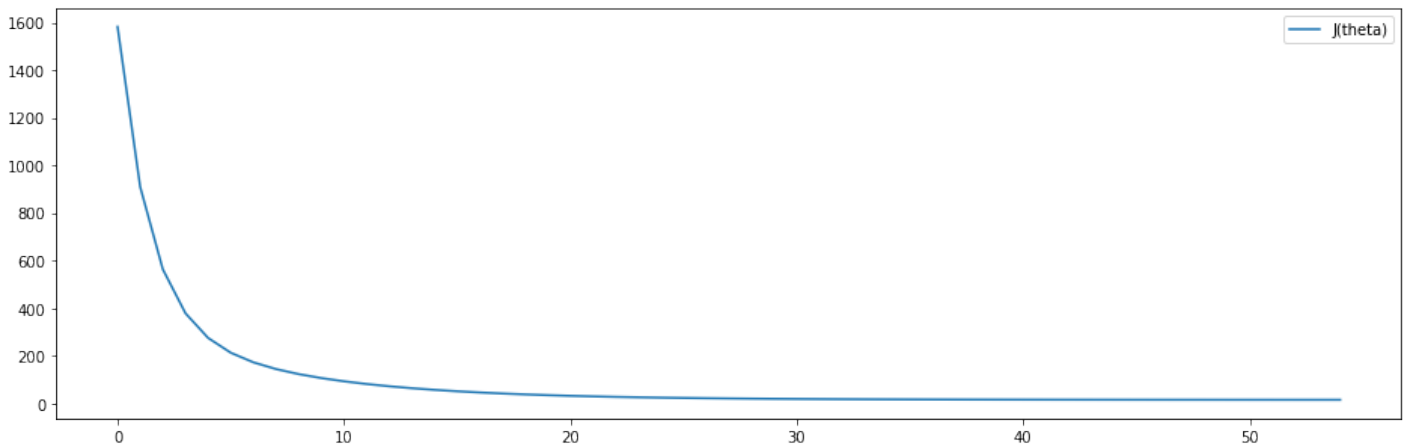
- Il est très important de simultanément changer les valeurs des paramètres.

2.1 Un mot sur la regression linéaire multivariables

- Pour illustrer: régression linéaire à deux dimensions







2.1 Features scaling

- Lorsque les variables n'ont pas la même échelle, la descente de gradient peut prendre beaucoup de temps
- Normaliser les variables: $-1 \leq x_i \leq 1$

	Feature Scaling	**Mean normalization**
Start range	$x_{min} \leq x \leq x_{max}$	$x_{min} \leq x \leq x_{max}$
Transformation	$x := \frac{x - x_{min}}{x_{max} - x_{min}}$	$x := x - x_{mean}$
New range	$0 \leq x \leq 1$	$(x_{min} - x_{mean}) \leq x \leq (x_{max} - x_{mean})$

- En combinant les deux: $\boldsymbol{x := \frac{x - x_{mean}}{x_{max} - x_{min}} \Rightarrow -1 \leq x \leq 1}$
- **Remarque:** Il est possible de remplacer $\boldsymbol{x_{max} - x_{min}}$ par l'écart type:

$$\sigma_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

2.2 La régression logistique

- **Classification**: prédire un nombre limités de valeurs discrètes
  - **Classification binaire**: Deux valeurs possibles: Vrai ou Faux (spam / non spam)
  - **Classification multiclasse**: plusieurs valeurs possibles (camion, voiture, piétons, vélos, ...)

- **Classification binaire:** En utilisant la régression linaire?
  - $y \in \{0, 1\} \Rightarrow$ On défini un seuil S pour $h_\theta(x)$:

$$h_\theta(x) \geq S \rightarrow y = 1$$
$$h_\theta(x) < S \rightarrow y = 0$$

- **Problème:** On voudrait que $0 \leq h_\theta(x) \leq 1$

$\Rightarrow$ Il faut redéfinir notre fonction hypothèse!

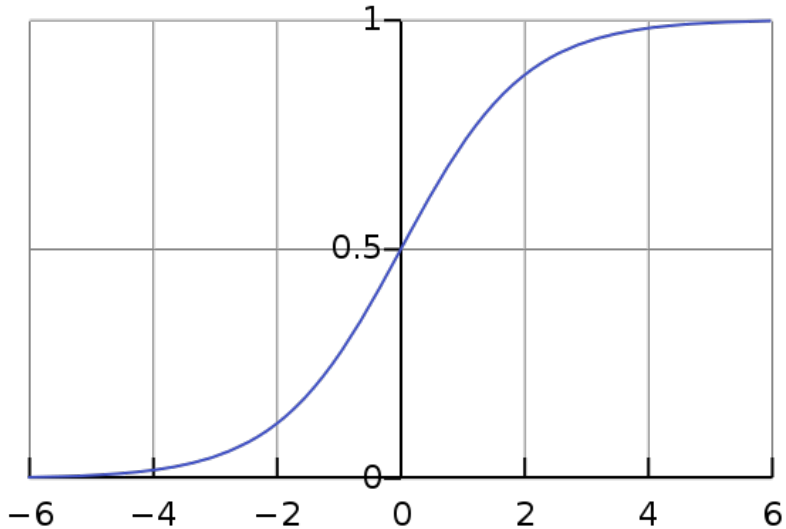
2.2 La régression logistique

- Modèle de la régression logistique:
  - On utilise la **Fonction Sigmoïde** $g(z) = \frac{1}{1+e^{-z}}$

$$z = \sum_{i=0}^{n} \theta_i x_i$$

$$h_\theta(x) = g(z)$$

$$0 \leq h_\theta(x) \leq 1$$



$g(z)$ depuis Wikipedia

- **Classification:** On prédit une **probabilité**

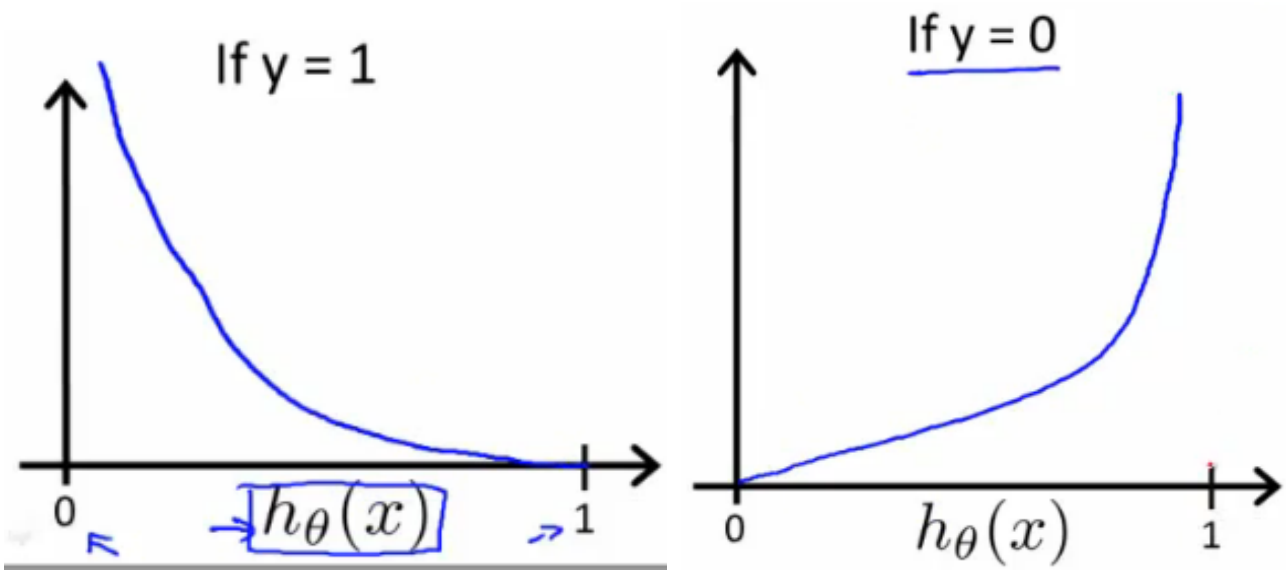
2.2 La régression logistique

- On change également la fonction de coût:

$$\mathcal{J}(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

- Avec:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



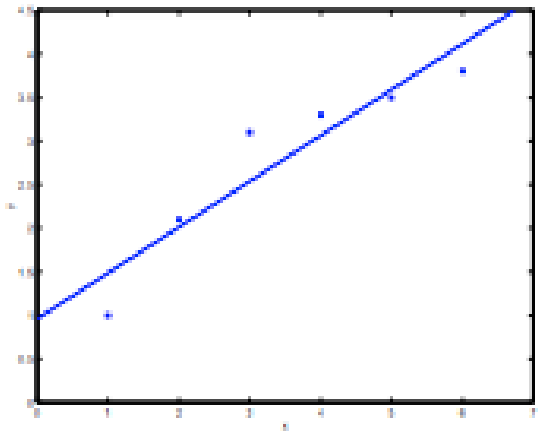
2.2 La régression logistique

- Et la **Classification multiclasse** ? $y \in \{0, 1, 2, \ldots, n\}$
- La **descente de gradient** ne permet pas de la résoudre
- On peut *tricher* en utilisant la méthode *'One-vs-all'*:
  - On remplace le problème multiclasse par $n + 1$ problèmes binaire
  - Probabilité que y soit dans une classe, les autres sont regroupés dans une deuxième classe *virtuelle*:

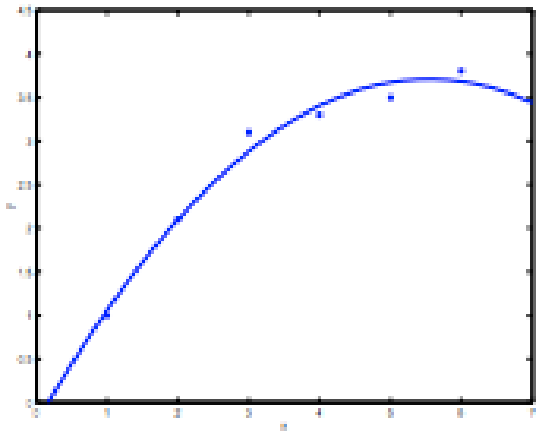
$$
\begin{aligned}
h_\theta^{(0)} &= P(y = 0|x; 0) \\
h_\theta^{(1)} &= P(y = 1|x; 0) \\
&\ldots \\
h_\theta^{(n)} &= P(y = n|x; 0) \\
\text{prediction} &= max_i(h_\theta^{(i)}(x))
\end{aligned}
$$

- Algorithme d'optimisation avancés: *Conjugate gradient*, *(L-)BFGS*, ...

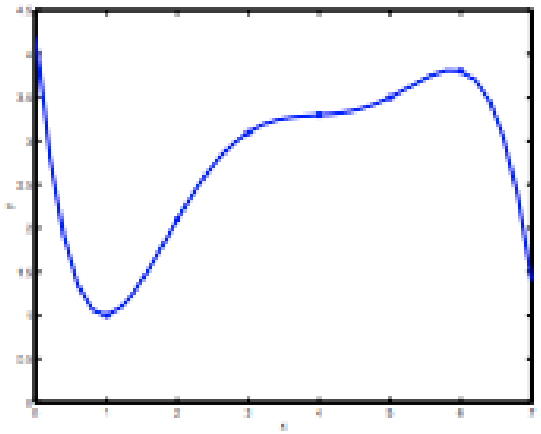
2.3 Les problèmes de biais et de variance



Underfitting (Bias)



Just fine!



Overfitting (Variance)

- **Underfitting**: modèle trop simple (pas assez de variables)
- **Overfitting**: modèle trop complexe, deux manière de résoudre:
  - Réduire le nombres de variables ou changer les paramètres du modèle
  - Utiliser des méthodes de **régularisation**

2.3 La régularisation

- Contraindre les paramètres θ_j sans réduire le nombre de variables
- On ré-écrit la fonction de coût avec le terme de régularisation:

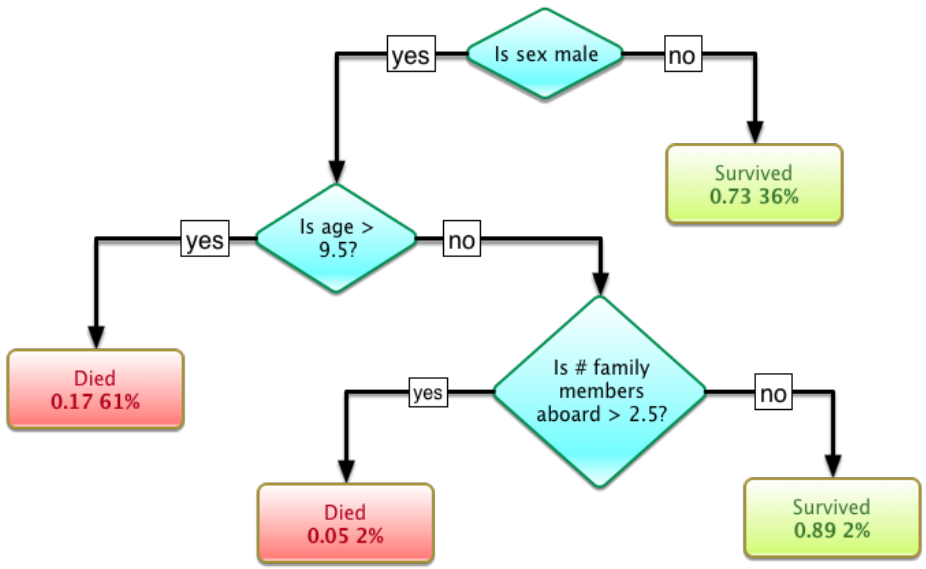
$$\mathcal{J}(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right)$$

- $\boldsymbol{\lambda}$: Paramètre de régularisation
- Si $\boldsymbol{\lambda}$ est trop grand: risque d'*underfitting*
- Si $\boldsymbol{\lambda = 0}$: pas de régularisation.
- **Remarque:** On ne régularise pas le terme constant $\boldsymbol{\theta_0}$

2.4 Les arbres de décisions

- Effectue une prédiction (regression ou classification) par une succession de decision simples: *if-else-then* sur les différentes variables
- *Arbre* = suite de décisions (*noeuds*) ammenant à une prédiction (*feuille*)

- La complexité de la structure de données qu'il est possible de représenter avec une arbre de décision est proportionelle à la profondeur de l'arbre



Un arbre pour déterminer la probabilité de survie des passagers du Titanic (depuis Wikipedia)

2.4 Les arbres de décisions: Apprentissage

- La *meilleure* séparation possible est déterminée:
  - Toutes les variables/sélection possibles
  - Celle qui minimise une fonction de coût
- L'échantillon d'entrainement est séparé suivant cette décision, créant ainsi deux feuilles (sous-échantillon)
- On recommence l'opération tant que l'on a pas atteint la profondeur voulue ou bien que la pureté de chaque feuille atteint un niveau satisfaisant (à déterminer)

Remarque: Il est possible d'utiliser la même variables pour différents noeud.

2.4 Les arbres de décisions

- Ces algorithmes sont faciles à interpréter (et à visualiser)
- Tout les types de données (catégorielle, numérique, ...) peuvent être mélangés
- Rapide et peu gourmand en ressource pour comprendre des structures de données complexes:

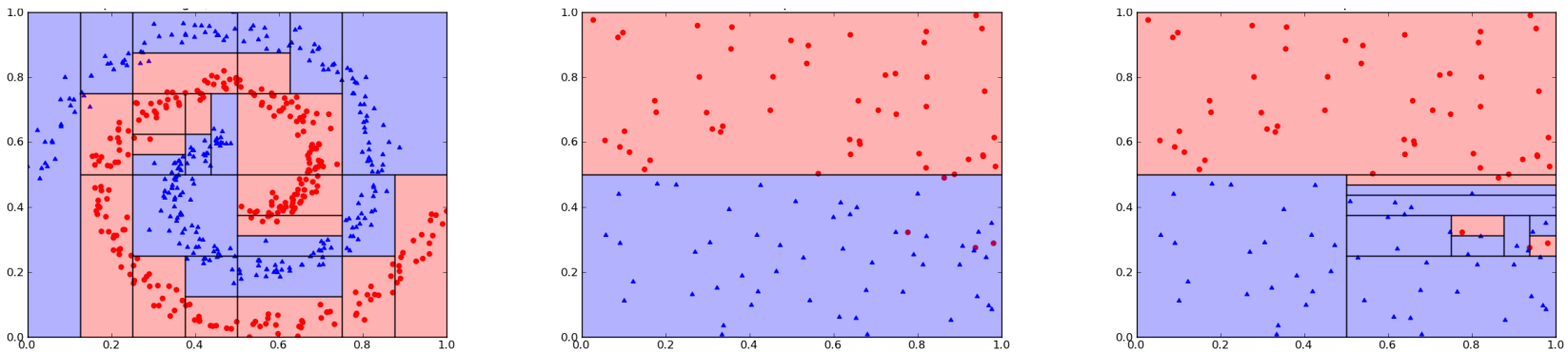


Illustration depuis University of Chicago

- Risque d'*overfitting* (sur-apprentissage) du modèle qui se généralise mal
- Algorithmes très sensibles au jeu de données d'entrainement

2.4 Les arbres de décisions: méthodes d'ensemble

- Pour pallier les faiblesse des arbres de décision, on les utilise comme *base learners* dans des méthodes qui en regroupe plusieurs
- Il existe deux familles de méthodes:
  - **averaging**: plusieurs estimateurs (*learners*) indépendant dont on moyenne les prédictions
    (*Random Forests*, . . .)
  - **boosting**: plusieurs estimateurs combinés
    (*AdaBoost*, *XGBoost*, . . .)

2.4 Exemple d'algorithme d'*averaging*: **Random Forests**

- *Bagging*: Chaque estimateur (arbre) de l'ensemble est construit à partir d'un sous-échantillon aléatoire (avec replacement) de l'échantillon d'entrainement (*Bootstrap aggregating*)
- *Features bagging*: La *meilleure* séparation possible sur un sous-échantillon aléatoire des variables
- Soit f_b, l'arbre entrainé sur le sous-échantillons b ($b \in [1, B]$), on calcule la prédiction globale en moyennant les prédictions de tout les arbres:

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x)$$

- Cette méthode obtient de meilleure performance car elle réduit la *variance* du modèle sans accroitre le *biais*

2.4 Exemple d'algorithme de *boosting*: **AdaBoost**

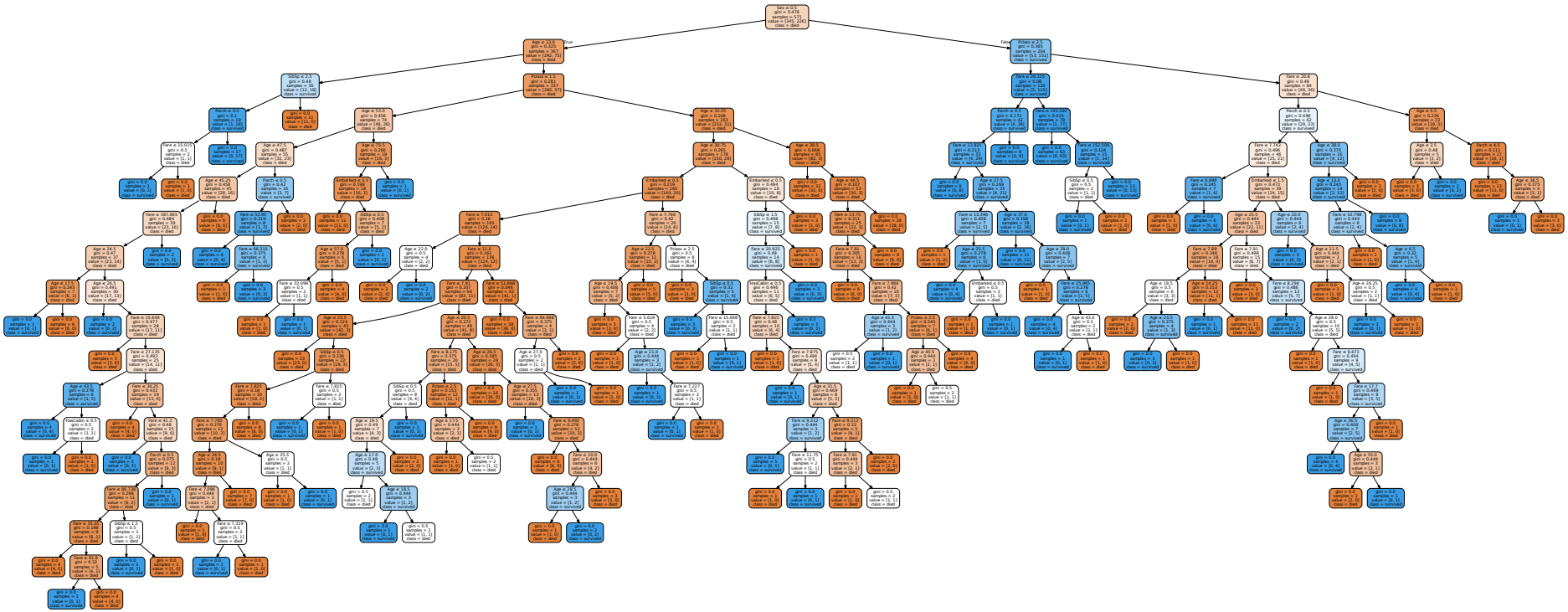
- L'apprentissage d'un même *learner* est répété en modifiant le jeu de données à chaque itération
- On utilise des poids: w_i pour $i \in [1, N]$. Initialement: $w_i = 1/N$
- À chaque itération, le poids des exemples correctement prédits diminue tandis que celui des exemples dont les prédictions sont fausse augmente
- L'algorithme devient plus sensibles aux exemples difficiles à prédire

2.4 Exemple d'algorithme de *boosting*: **XGBoost**

- Obtient d'excellent résultats sur la plupart des cas d'usages (classification et régression)
- Des arbres sont générés aléatoirement comme pour les *random forests*
- Mais au lieu de moyenner les prédictions, on additionne les prédictions
  - À chaque étape, des arbres sont générés et on sélectionne celui qui optimise la fonction d'objectif (une fonction de coût + une fonction qui mesure la complexité du modèle)
  - On additionne la prédiction de cet arbre à la prédiction du modèle et on recommence jusqu'à atteindre une performance suffisante

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= \hat{y}_i^{(0)} + f_1(x_i) = f_1(x_i) \\
\hat{y}_i^{(2)} &= \hat{y}_i^{(1)} + f_2(x_i) = f_1(x_i) + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \hat{y}_i^{(t-1)} + f_t(x_i) = \sum_{k=1}^{t} f_k(x_i)
\end{aligned}
$$

2.4 Arbres de décision: *Titanic*



	Decision Tree	Random Forest	AdaBoost	XGBoost
Accuracy (%)	77.6	81.1	83.2	83.9

3 L'apprentissage non-suppervisé

- Données non *labélisées*
- La machine apprend par elle même à indentifier une structure
- Évaluation des performances compliqué.
- Problèmes de classification, réduction de dimensions
- *K-means*, *Mean Shift*, *Gaussian Mixture Model*
- Analyse en Composante Principale

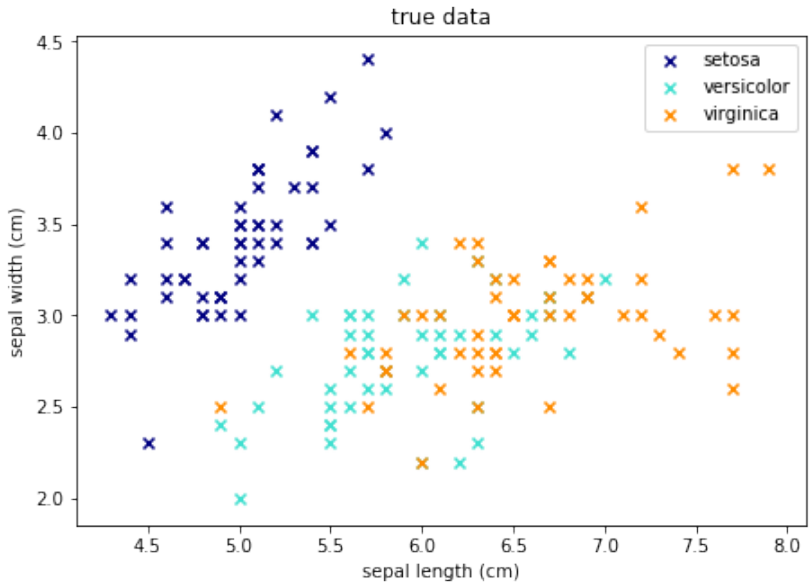
3.1 Les algorithmes de clustering

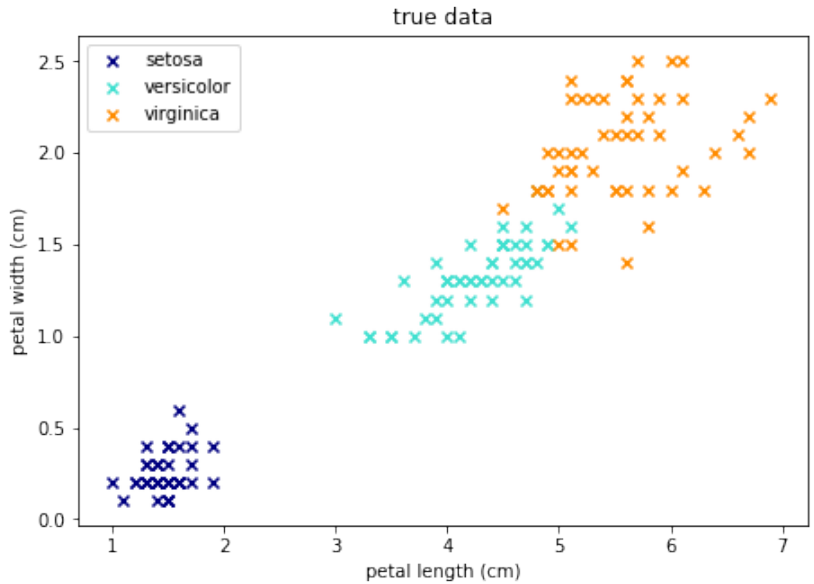
- *Clustering*: Se rapproche d'un problème de classification (sans labels)
- Ces algorithmes cherchent à rassembler les exemples en cluster
- À la différence des arbres de décision, le choix du cluster ne s'effectue pas par une suite de décisions simple, mais en déterminant la plus petite distance possible dans l'espace des variables
- Utilisés pour la segmentation d'utilisateurs/marchés dans le commerce en ligne mais aussi en génétique
- Il faut (dans la majorité des cas) définir au préalable le nombre de clusters à construire (hyperparamètre du modèle)

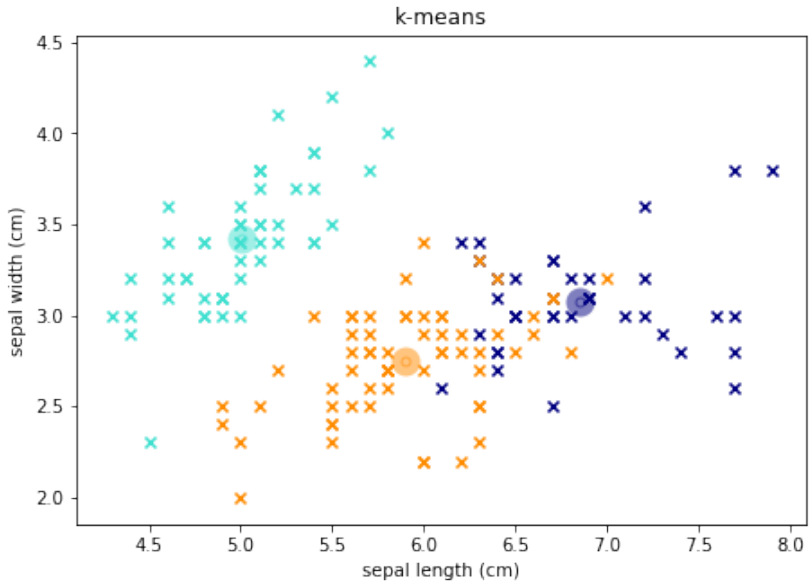
3.1 K-means

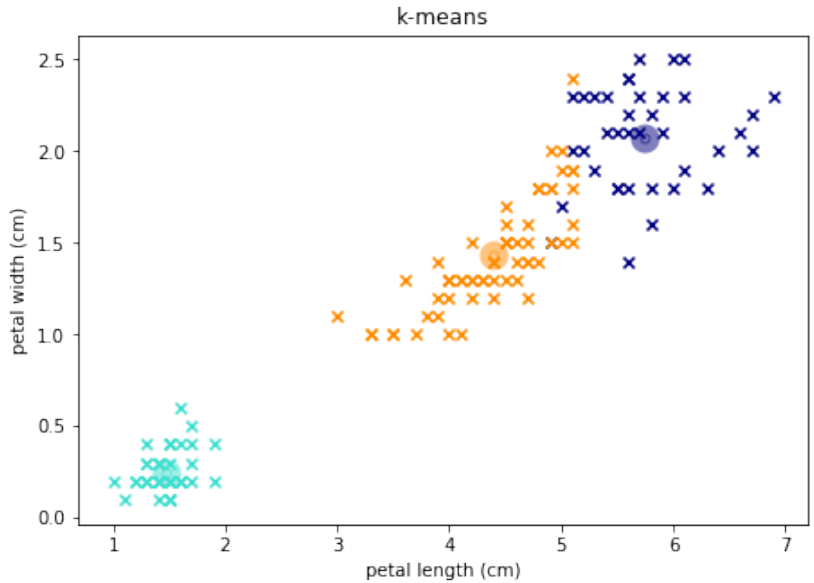
- Sépare les données en K clusters C d'égale variance (dispersion)
- Soit les *centroïdes*, la moyenne des échantillons dans chaque cluster
- *K-means* modifie la position des *centroïdes* jusqu'à trouver la valeur qui minimise l'écart moyens des échantillons vis-à-vis de son cluster correspondant
- Critère de minimisation: **inertie**: $I = \sum_{k=0}^{K} \sum_{x \in C_k} ||x - \mu_k||^2$
- La performance du *K-means* est fortement dépendante de son initialisation *(Solution: plusieurs initialisation $\rightarrow$ moyenne)*
- Le *K-means* ne fonctionne pas avec des variables catégorielles (il existe des adaptations). Il est préférable de normaliser les variables.

3.1 Clustering: Iris dataset, K-means result





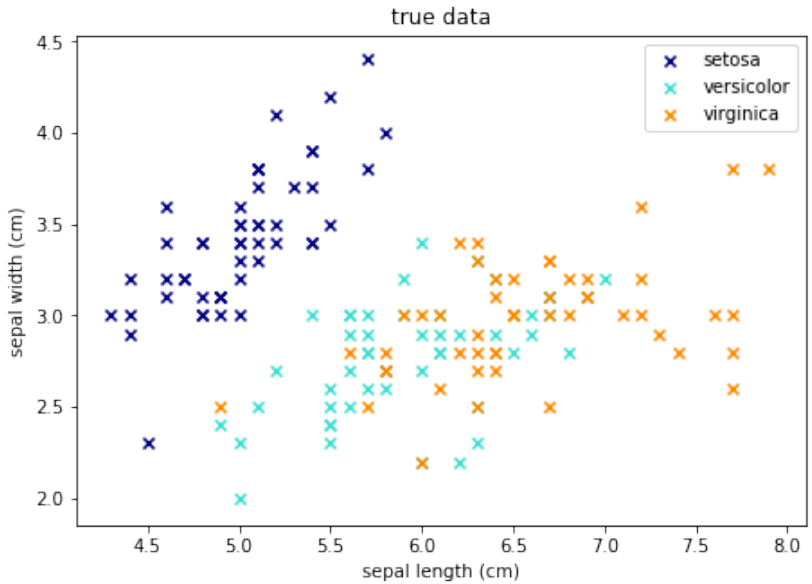


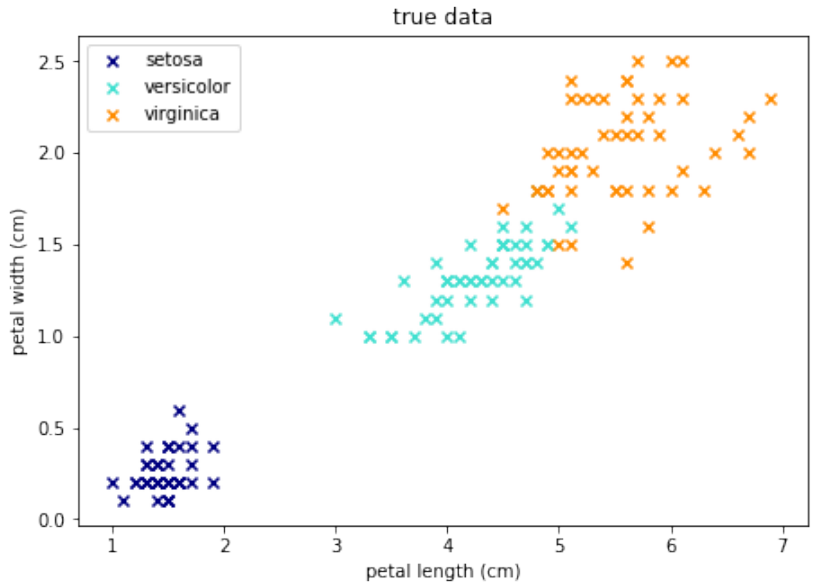


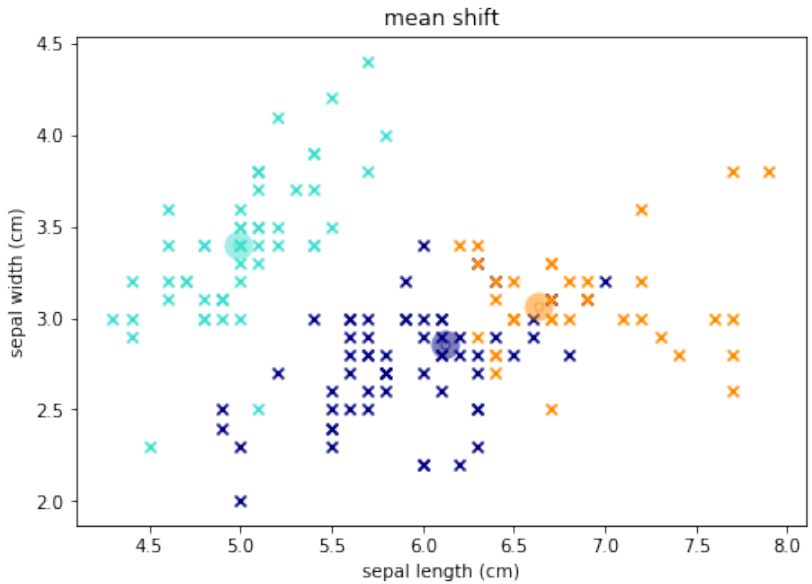
3.1 Mean Shift

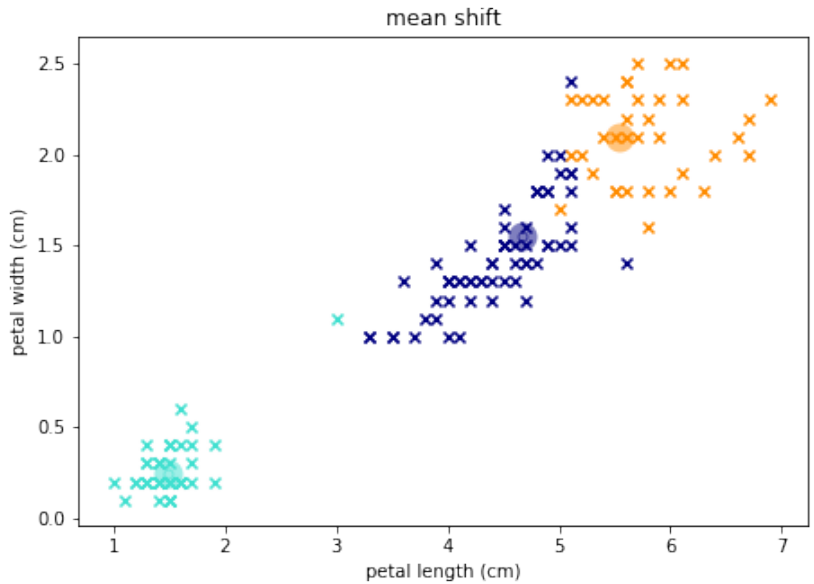
- Cherche les zones de fortes densité en modifiant itérativement la position de *centroïdes*
- Des *centroïdes* de rayons R sont aléatoirement initialisés
- Ils sont déplacés vers la région de plus haute densité (nombres de points dans le rayon R)
- On continue jusqu'à maximiser la densité de chaque *centroïde*
- Plusieurs *centroïdes* dans une zone: celui avec la plus haute densité est conservé
- l'ensemble du dataset est labelisé (plus petite distance)
- Le *Mean Shift* détermine le nombre optimal de clusters pour la valeur du rayon choisie (R)

3.1 Clustering: Iris dataset, Mean Shift result





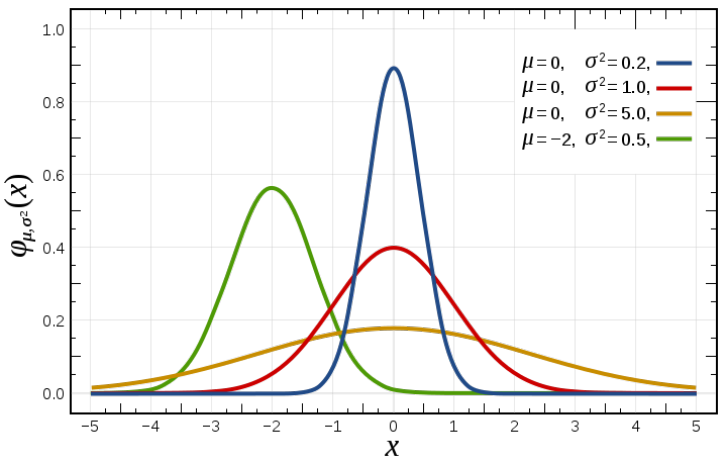




3.1 Gaussian mixture models

- **Hypothèse:** la structure des données est compatible avec un mélange de gaussiènne

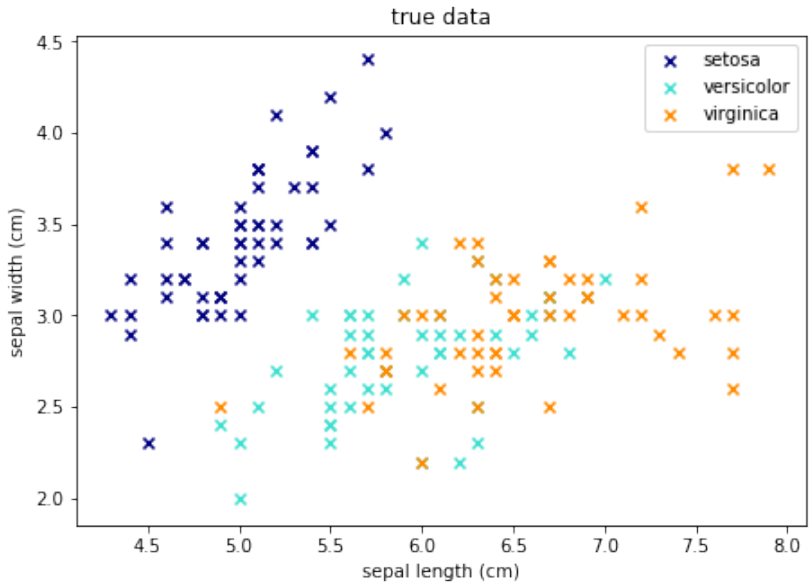
- On ajuste les paramètres de N gaussiènnes jusqu'à trouver ceux qui *collent* le mieux aux données

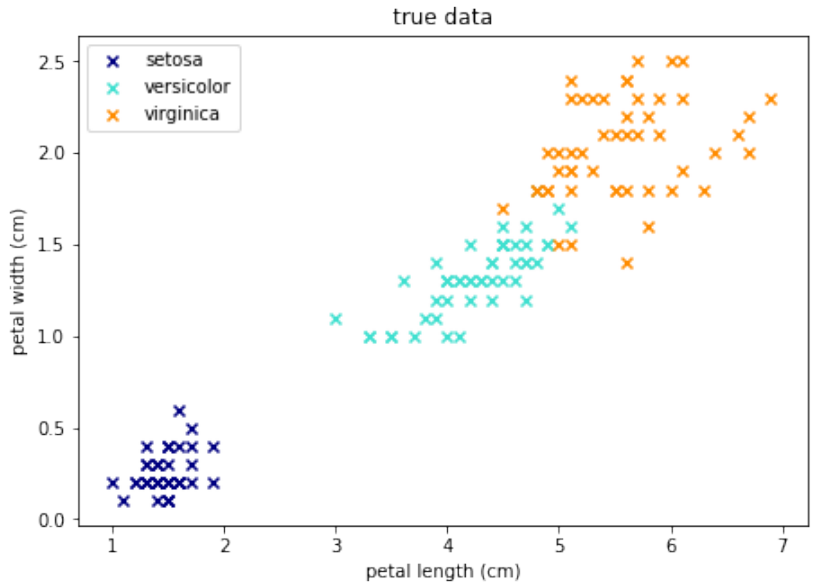


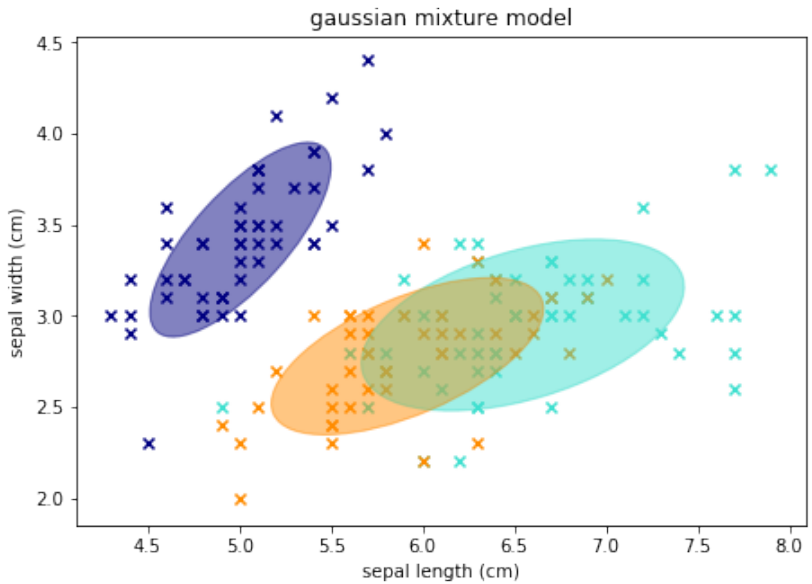
depuis Wikipedia

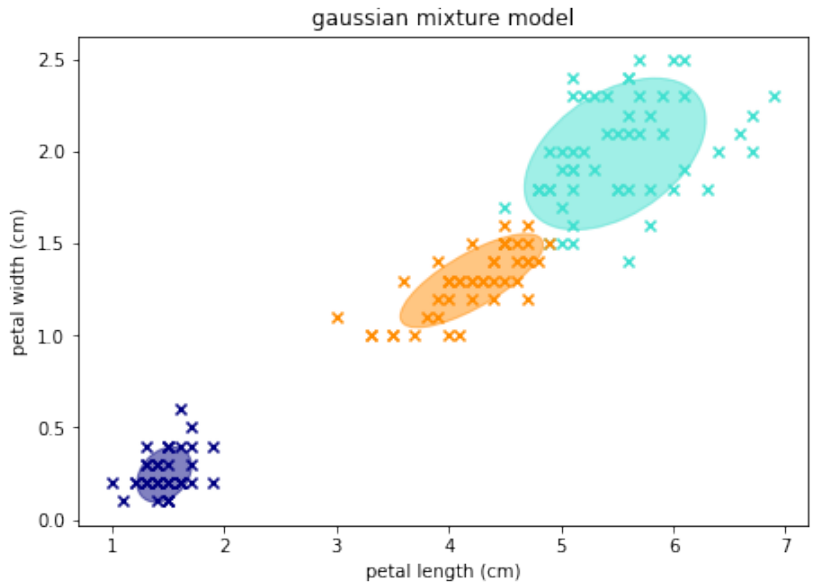
- Algorithme d'ajustement: **espèrance-maximisation**, on cherche à maximiser la fonction de *log-likelihood* (fonction de vraissemblance)
- *Intuitivement:* on cherche les paramètres qui rendent la distribution de données observée la plus probable

3.1 Clustering: Iris dataset, GMM result









3.2 Analyse en composantes principales

- **PCA** (*Composants principal analysis*): Algorithme de réduction de dimensions
- Transforme un problème à n variables en un problème à n' variables (avec $n' < n$)
- Explore les corrélations entre les variables pour les *regrouper* (pondération): **Pertes d'informations**
- Très utile pour:
  - Pre-stage d'un algorithme de clustering (peut améliorer les performances)
  - Pour faire de la visualisation en 2D de données à plus haute dimension

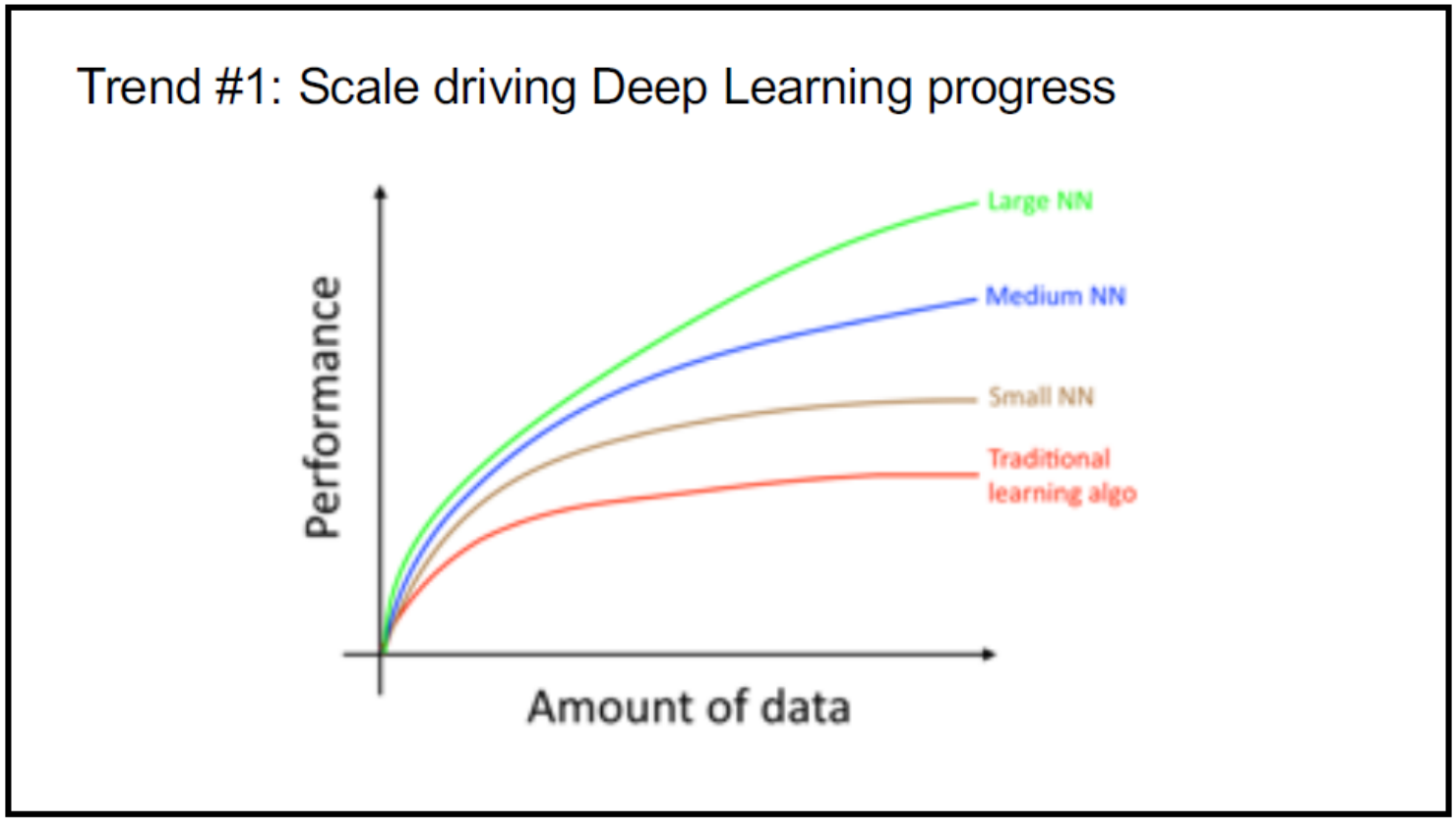
3.2 Analyse en composantes principales

- **Exemple d'utilisation:** digit dataset:
  - 64 variables $\Rightarrow$ 10 pour initialiser le K-means
  - 64 $\Rightarrow$ 2 pour la visualisation

4. L'apprentissage profond

- **Deep Learning** = *Réseaux de Neurones* (avec plus d'une couche cachée)
- Conceptualisés dans les années 80 et début 90, l'explosion de la puissance de calcul disponibles a rendu possible leur exploitation
- **Qu'est ce que ça a à voir avec le cerveaux?** Pas grand chose en fait ... à part une analogie avec la structure des neurones
- Utilisé principalement dans le cadre de l'**apprentissage supervisé**
  - Image classifiers, Object detection
  - Speech recognition
  - Machine traduction (DeepL)
  - Voiture autonomes
  - ...
- Données structurées/non-structurées:
  - Les humains sont bons pour interpréter des données non-structurées

4. L'apprentissage profond

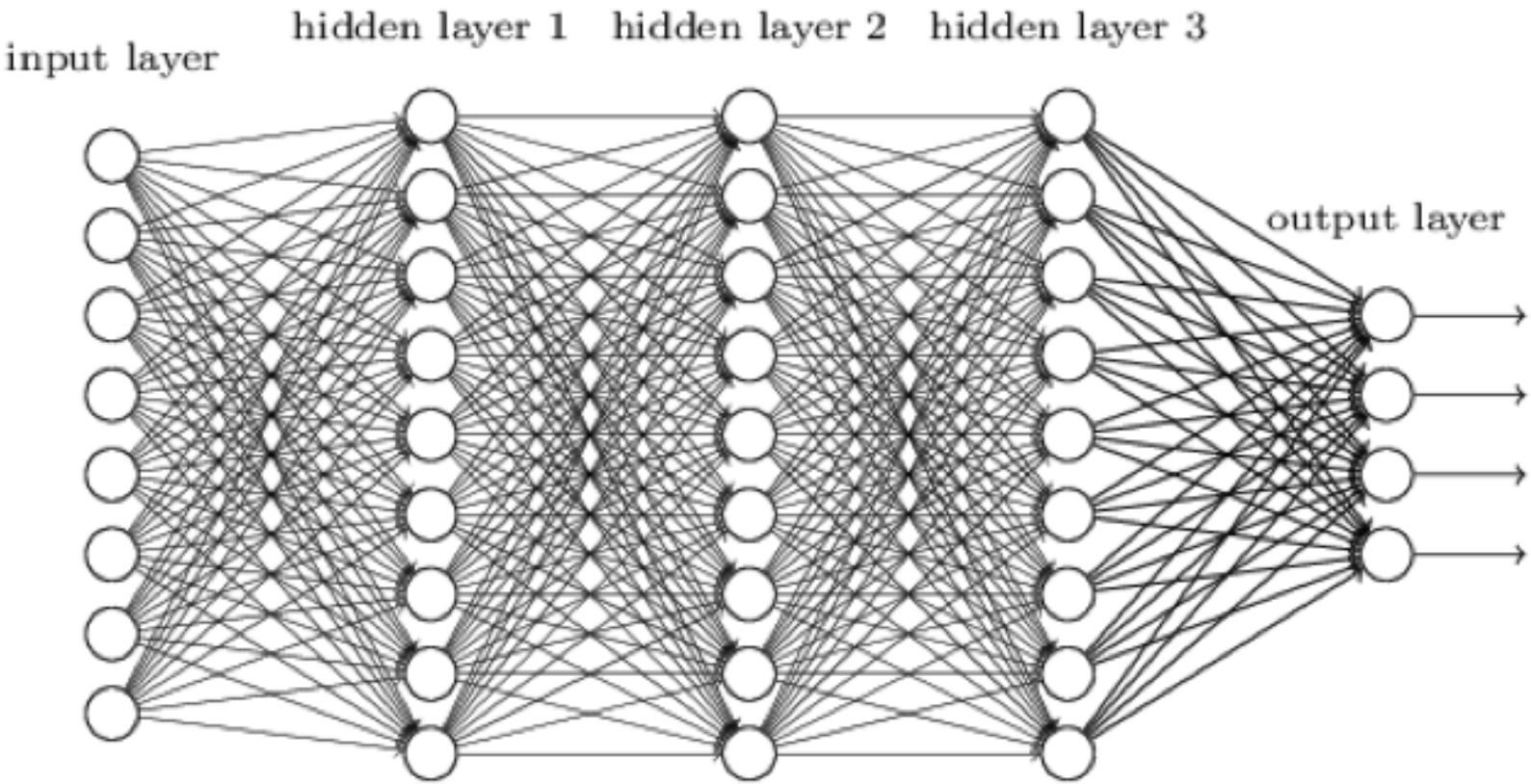


Large NN, Medium NN, Small NN, Traditional learning algo

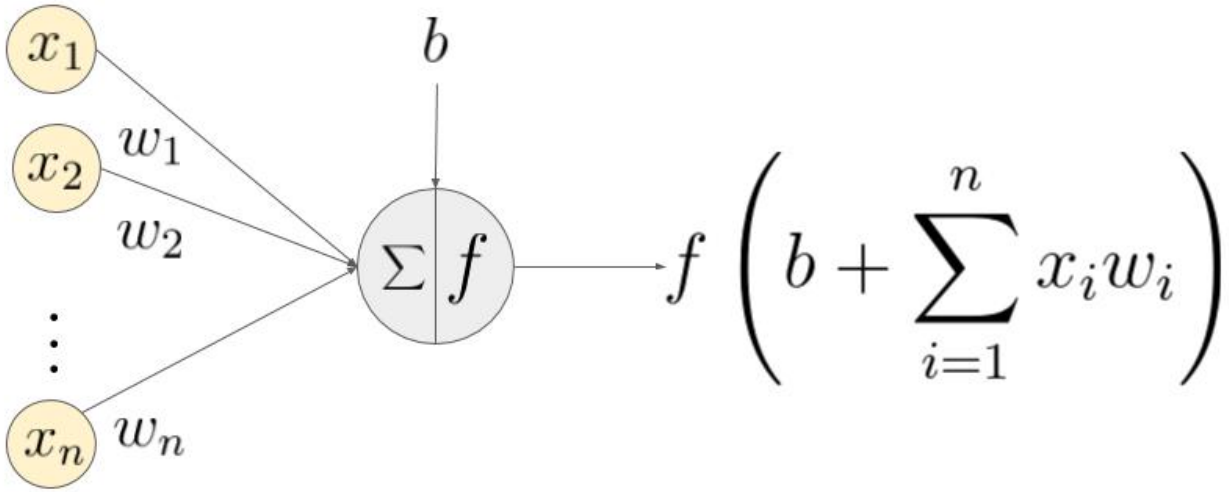
Depuis deeplearning.ai coursera

4.1 Les réseaux neurones

- 1 input layer $\Rightarrow L-1$ hidden layer $\Rightarrow$ 1 output layer
- $n^{[l]}$ cellules (neurones) pour la couche l, m variables (input layer: $n^{[0]}$)
- $W^{[l]}$, $b^{[l]}$: paramètres de la couche l ($W^{[l]} \in \mathbb{R}^{(n^{[l]} \times n^{[l-1]})}$, $b^{[l]} \in \mathbb{R}^{(n^{[l]} \times 1)}$)



4.1 Un neurone i de la couche l



- 2 étapes de calculs:
  - $z_i^{[l]} = \sum_{j=0}^{n^{[l-1]}} w_{ij}^{[l]} \times a_j^{[l-1]} + b_i^{[l]}$ Où, $a^{[0]} = x$
  - $a_i^{[l]} = f^{[l]}(z_i^{[l]})$ Où, $f^{[l]}(z)$ est la **fonction d'activation**

4.1 Une couche l de neurones

- On répète l'opération pour chaque neurone i de la couche l
- Plus efficace de réfléchir en multiplication de matrices:

$$z^{[l]} = \begin{bmatrix} z_1^{[l]} \\ z_2^{[l]} \\ \vdots \\ a_{n^{[l]}}^{[l]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[l]} & w_{12}^{[l]} & \dots & w_{1n^{[l-1]}}^{[l]} \\ w_{21}^{[l]} & w_{22}^{[l]} & \dots & w_{2n^{[l-1]}}^{[l]} \\ & & \ddots & \\ w_{n^{[l]}1}^{[l]} & w_{n^{[l]}2}^{[l]} & \dots & w_{n^{[l]}n^{[l-1]}}^{[l]} \end{bmatrix} \begin{bmatrix} a_1^{[l-1]} \\ a_2^{[l-1]} \\ \vdots \\ a_{n^{[l-1]}}^{[l-1]} \end{bmatrix} + \begin{bmatrix} b_1^{[l]} \\ b_2^{[l]} \\ \vdots \\ b_{n^{[l]}}^{[l]} \end{bmatrix}$$

- Que l'on réécrira: $\boldsymbol{z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}}$
- Puis: $\boldsymbol{a^{[l]} = f^{[l]}(z^{[l]})}$

4.1 Tous les exemples à la fois

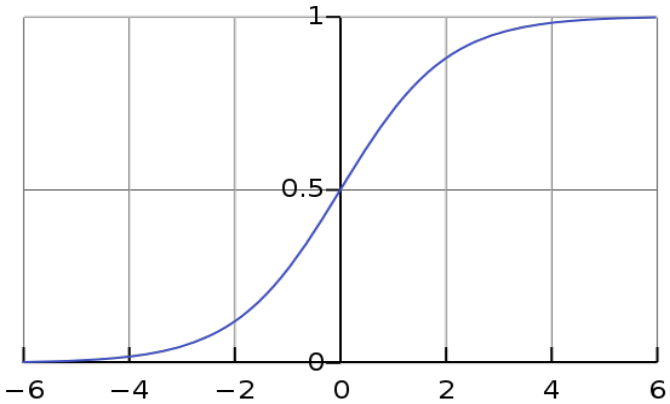
- On a vu le cas avec un exemple, mais si on veux faire les m exemples en une fois? `for-loop`? $\Rightarrow$ **Matrices**!
- On vectorise: $X \in \mathbb{R}^{n \times m}$, plus généralement: $A^{[l]}(Z^{[l]}) \in \mathbb{R}^{n^{[l]} \times m}$

$$A^{[l]} = \begin{bmatrix} a_1^{[l](1)} & a_1^{[l](2)} & \dots & a_1^{[l](m)} \\ a_2^{[l](1)} & a_2^{[l](2)} & \dots & a_2^{[l](m)} \\ & & \ddots & \\ a_{n^{[l]}}^{[l](1)} & a_{n^{[l]}}^{[l](2)} & \dots & a_{n^{[l]}}^{[l](m)} \end{bmatrix}$$

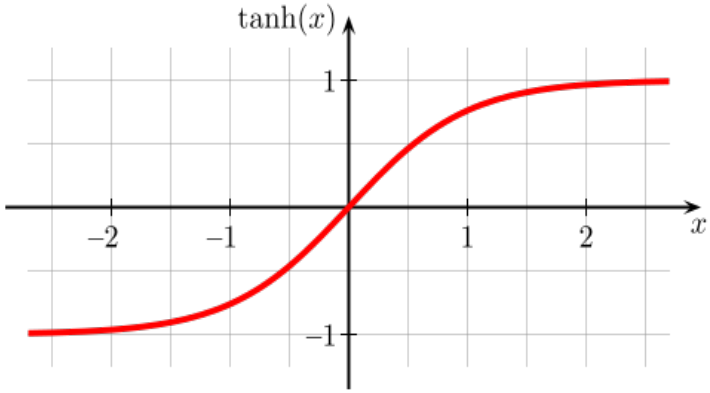
- Les étapes de calculs s'écrivent: $\boldsymbol{Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}}$
- Puis: $\boldsymbol{A^{[l]} = f^{[l]}(Z^{[l]})}$

4.1 Les 4 principales fonctions d'activations

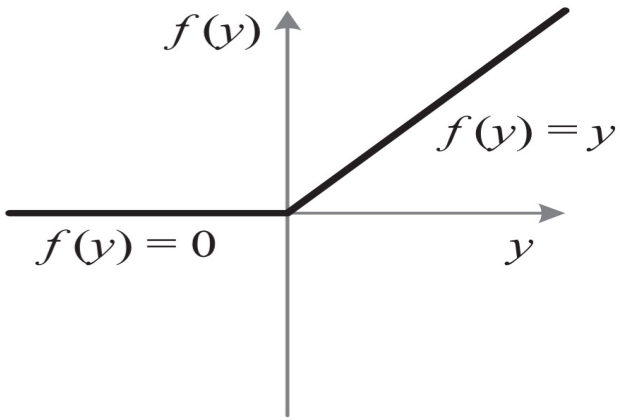
- **Logistique:**



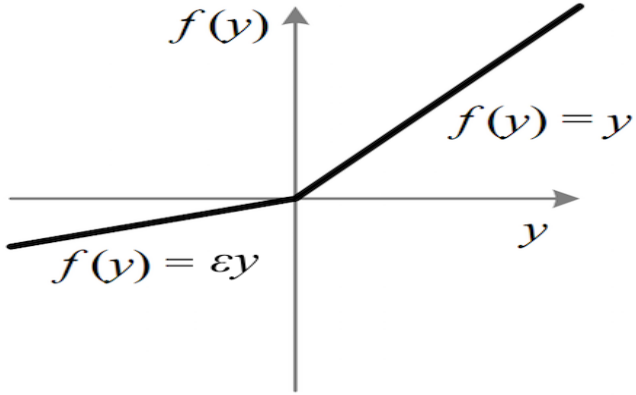
- **Tangente-hyperbolique:**



- **Rectified Linear Unit:**



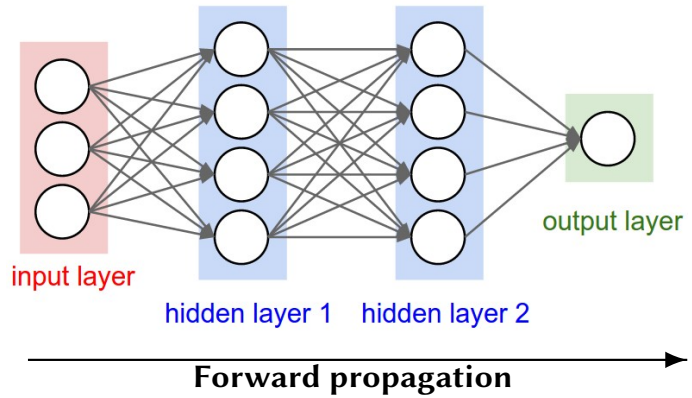
- **Leaky ReLU:**



- Si $\hat{y} \in \{0, 1\}$: **Logistique**. Les autres neurones: **ReLU**

4.1 Forward propagation

- Input $\boldsymbol{X}$: 3 variables ($n^{[0]} = 3$)
- 2 couches cachées ($n^{[1]} = n^{[2]} = 4$): **ReLU**
- Output: $n^{[3]} = 1$: **Logistique**



$$
\begin{array}{lcllcl}
X & \in & \mathbb{R}^{3\times m} & Y & \in & \mathbb{R}^{1\times m} \\
W^{[1]} & \in & \mathbb{R}^{4\times 3} & b^{[1]} & \in & \mathbb{R}^{4\times 1} \\
Z^{[1]} & \in & \mathbb{R}^{4\times m} & A^{[1]} & \in & \mathbb{R}^{4\times m} \\
W^{[2]} & \in & \mathbb{R}^{4\times 4} & b^{[2]} & \in & \mathbb{R}^{4\times 1} \\
Z^{[2]} & \in & \mathbb{R}^{4\times m} & A^{[2]} & \in & \mathbb{R}^{4\times m} \\
W^{[3]} & \in & \mathbb{R}^{1\times 4} & b^{[3]} & \in & \mathbb{R}^{1\times 1} \\
Z^{[3]} & \in & \mathbb{R}^{1\times m} & A^{[3]} & \in & \mathbb{R}^{1\times m}
\end{array}
$$

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= f^{[1]}(Z^{[1]}) = relu(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= f^{[2]}(Z^{[2]}) = relu(Z^{[2]}) \\
Z^{[3]} &= W^{[3]}A^{[2]} + b^{[3]} \\
\hat{Y} &= A^{[3]} = f^{[3]}(Z^{[3]}) = \sigma(Z^{[3]})
\end{aligned}
$$

4.1 Backward propagation

- C'est l'algorithme d'*apprentissage* des réseaux de neurones
- On définit la fonction de coût:

$$\mathcal{J}(W^{[1]}, \ldots, W^{[L]}, b^{[1]}, \ldots, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

- On utilise la **descente de gradient**:

Répéter: {

Calculez: $\hat{Y}$

$dW^{[L]} := \frac{d\mathcal{J}}{dW^{[L]}}, \quad db^{[L]} := \frac{d\mathcal{J}}{db^{[L]}}$

$W^{[L]} := W^{[L]} - \alpha dW^{[L]}$

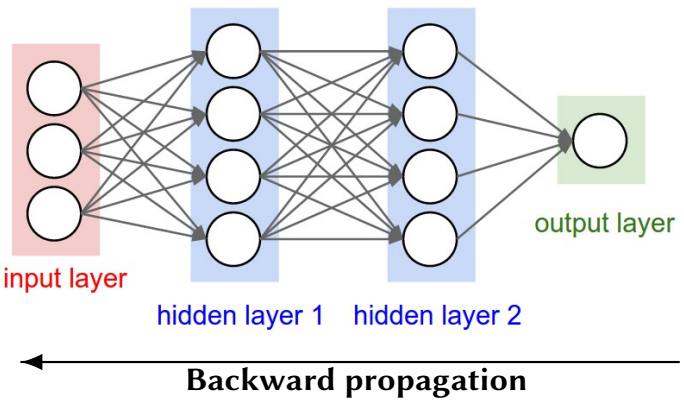
$b^{[L]} := b^{[L]} - \alpha db^{[L]}$

$\vdots$

}

4.1 Backward propagation

- On va propager l'erreur $\hat{Y} - Y$ pour modifier les valeurs des paramètres $W^{[l]}$ et $b^{[l]}$ du réseau de neurones



$$
\begin{aligned}
dZ^{[3]} &= A^{[3]} - Y \\
dW^{[3]} &= \frac{1}{m} dZ^{[3]} A^{[2]\,T} \\
db^{[3]} &= \frac{1}{m} \sum dZ^{[3]} \\
dZ^{[2]} &= W^{[3]\,T} dZ^{[3]} * relu'(Z^{[2]}) \\
dW^{[2]} &= \frac{1}{m} dZ^{[2]} A^{[1]\,T} \\
db^{[2]} &= \frac{1}{m} \sum dZ^{[2]} \\
dZ^{[1]} &= W^{[2]\,T} dZ^{[2]} * relu'(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} X^{T} \\
db^{[1]} &= \frac{1}{m} \sum dZ^{[1]}
\end{aligned}
$$

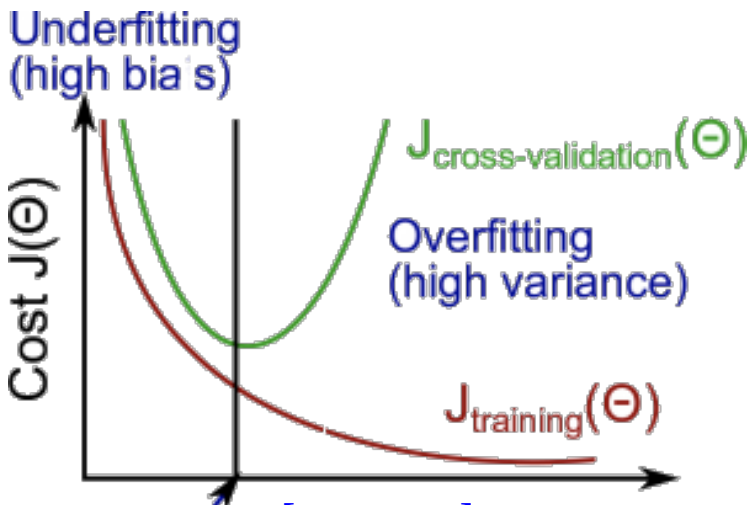
- Pour que l'apprentissage fonctionne: **Initialisation aléatoire** W, b

4.2 Les bonnes pratiques: préparer les donneés

- On sépare les données en trois sous-échantillons:
  - **Train:** (60%) échantillon d'entrainement avec lequel on applique la *forward-backward propagation*
  - **Validation:** (20%) échantillon qui nous permet de mesurer les performances du modèle pour différentes valeurs d'hyperparamètres et différentes architectures
  - **Test:** (20%) échantillon de test qui donne la performance du modèle final
- Il est important de séparer les données en différents sous-échantillons de test pour être sûr que le modèle de généralise bien
- S'assurer que les sous-échantillons proviennent de la même source et soient représentatifs

4.2 Les bonnes pratiques: biais et variance

- Si $\mathcal{J}_{train}(W, b) >> \mathcal{J}_{valid}(W, b)$: problème de variance: **Overfitting**
- Si $\mathcal{J}_{train}(W, b) \approx \mathcal{J}_{valid}(W, b) >> 0$: problème de biais: **Underfitting**
- Comment diagnostiquer? deux valeurs à regarder:
  - Erreur de l'échantillon d'entrainement: t_{err}
  - Erreur de l'échantillon de validation: v_{err}
- On estime le cas idéal (Bayes ou opérateur humain): $\approx 0\%$

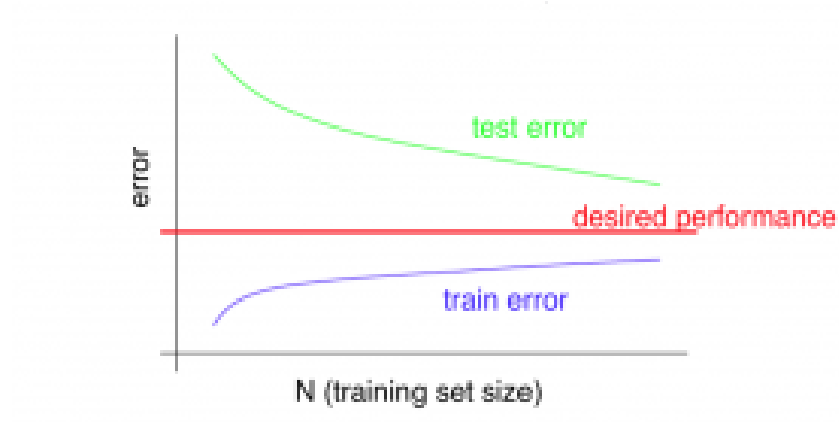


[Coursera]

t_{err}	v_{err}	problème
1%	11%	haute variance
15%	16%	haut biais
15%	30%	haut biais & haute variance
0.5%	1%	bas biais & basse variance

4.2 Les bonnes pratiques: courbes d'apprentissage

- Entrainer un algo en augmentant le nombre d'exemples et monitorer l'évolution de l'erreur:



Biais et variance [Coursera]

4.2 Les bonnes pratiques: Que faire?

- **Haut Biais?** → **Oui** → Réseau plus profond
  Entrainer plus longtemps
  Changer architecture du NN
  ↓
  Recommencer! →

(Performance entrainement)

↓

Non

↓

- **Haute Variance?** → **Oui** → Plus de données
  Regularisation
  Changer architecture du NN
  ↓
  Recommencer! →

(Performance validation)

↓

Non

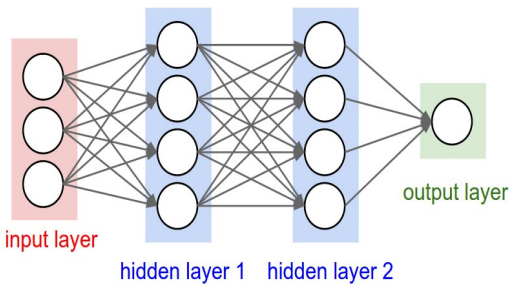
↓

OK!

4.3 Les différentes architectures de NN

- Différentes architectures pour différents usages:

Std NN



Regression
Classification

Convolutional NN

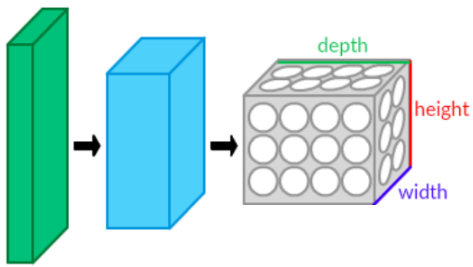
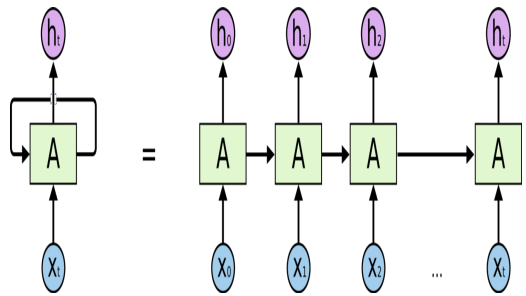


Image classifier
Object detection

Recurent NN



Audio recognition
Time Series analysis

Merci!

Des questions?

Léo Beaucourt: lbeaucourt@agaetis.fr