

Vincent Gigliobianco

Data Scientist

Rappels de mathématiques

a) Dérivées

b) Minimum d'une fonction

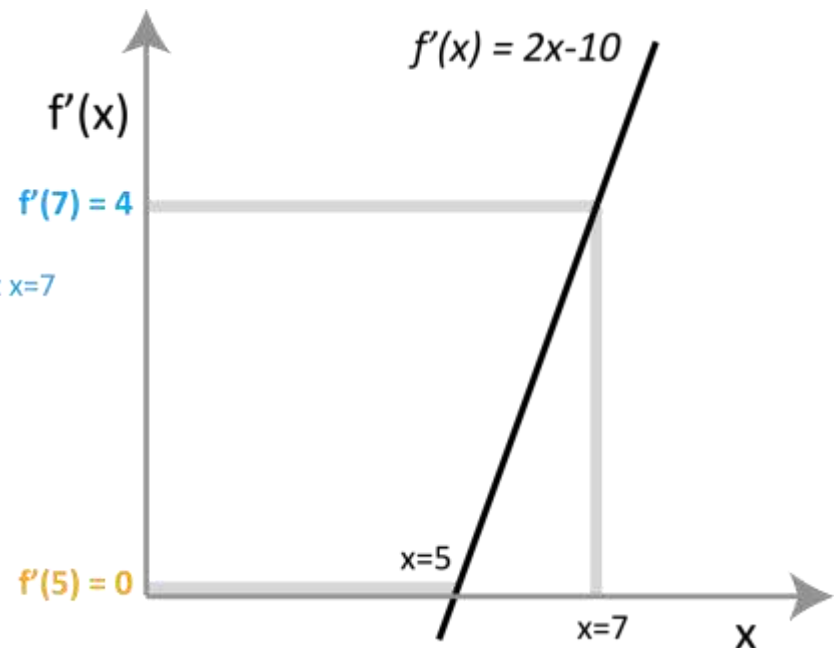
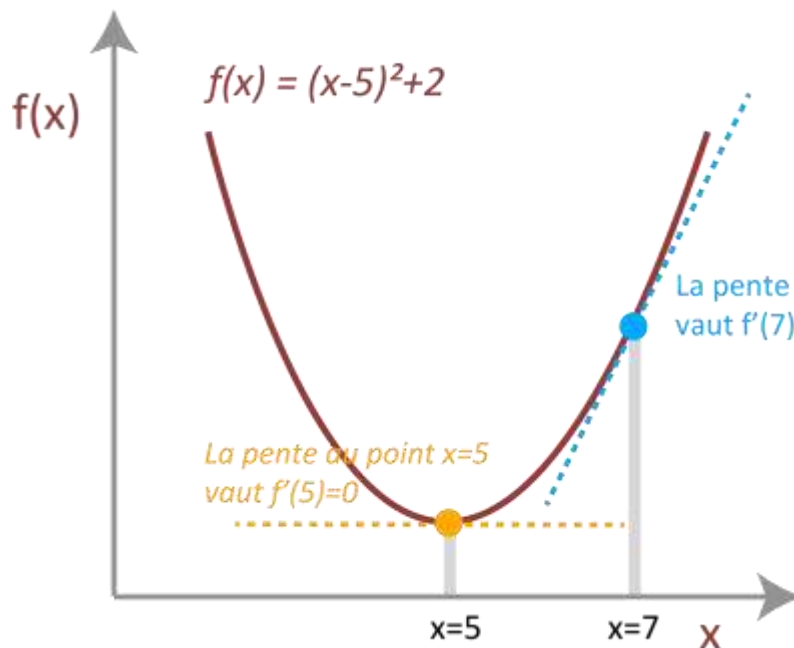
c) Fonction convexe

d) Matrices

Dérivées

Signification de la dérivée

- La pente en un point x_0 d'une fonction $f(x_0)$, est donné par sa dérivée $f'(x_0)$



Dérivées

Fonctions	Dérivées
Constante k	0
x^n	nx^{n-1}
$\frac{1}{x}$	$\frac{(-1)}{x^2}$
Log(x) (log népérien)	$\frac{1}{x}$
e^x	e^x

Dérivées

Règle 1:

« La dérivée d'une somme est la somme des dérivées »

$$f(x) = x^2 + 2x + 1$$

$$f'(x) = \text{dérivée de } (x^2) + \text{dérivée de } (2x) + \text{dérivée de } (1)$$

$$f'(x) = 2x + 2 = 2(x + 1)$$

Règle 2:

« la dérivée de λf , c'est $\lambda f'$ »

Exemple:

$$f(x) = 5(x^2 + 1)$$

$$f'(x) = 5(2x) = 10x$$

Règle 3 :

« la dérivée d'un produit : $(u.v)' = u'v + uv'$ »

Règle 4 :

« la dérivée d'un quotient: $\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$ »

Règle 5 des dérivées composées:

« la dérivée de $g(u)$ est $g'(u) \cdot u'$ »

Exemple : dériver

$$f(x) = \log(-2x^2 + x - 1)$$

$$\text{On pose : } u(x) = -2x^2 + x - 1$$

Et on dérive ...

Dérivées partielles

Soit la fonction F à 2 variables x et y :

$$F(x,y) = \frac{xy}{x^2 - y^2}$$

On définit sa dérivée par rapport à x

Règle à appliquer !

« x est une variable et y est une constante »

On va montrer que :

$$\frac{\partial F(x,y)}{\partial x} = \frac{-y(x^2 + y^2)}{(x^2 - y^2)^2}$$

On définit la dérivée de F par rapport à y

Règle à appliquer !

« x est une constante et y est une variable »

On va montrer que :

$$\frac{\partial F(x,y)}{\partial y} = \frac{x(x^2 + y^2)}{(x^2 - y^2)^2}$$

Minimum d'une fonction

Minimum de la parabole d'équation $Y = x^2 + 2x + 1$

- (cf Exercice 2 notebook python « Rappels_mathématiques »)

Son minimum peut s'obtenir :

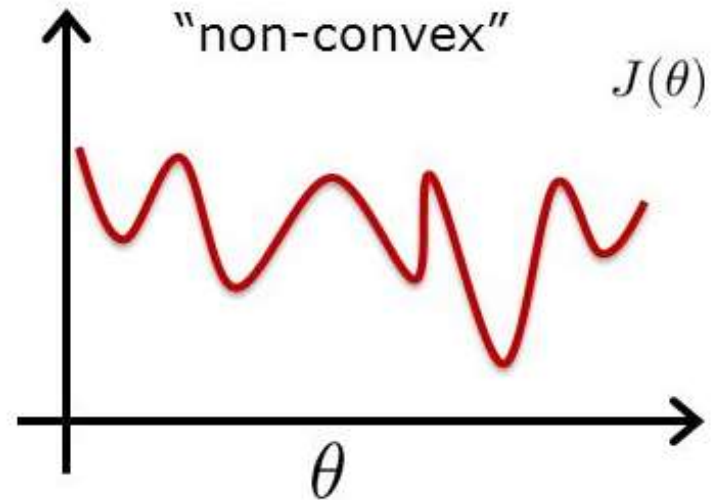
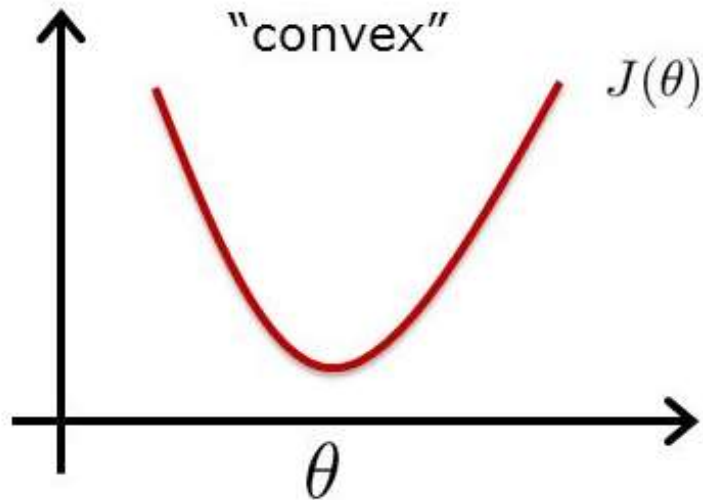
- En dérivant $f(x)$
- Puis résolvant l'équation $f'(x) = 0$

cad on cherche l'abscisse x où la pente de la courbe est nulle

C'est un minimum local qui est aussi un minimum global: c'est une fonction **convexe**

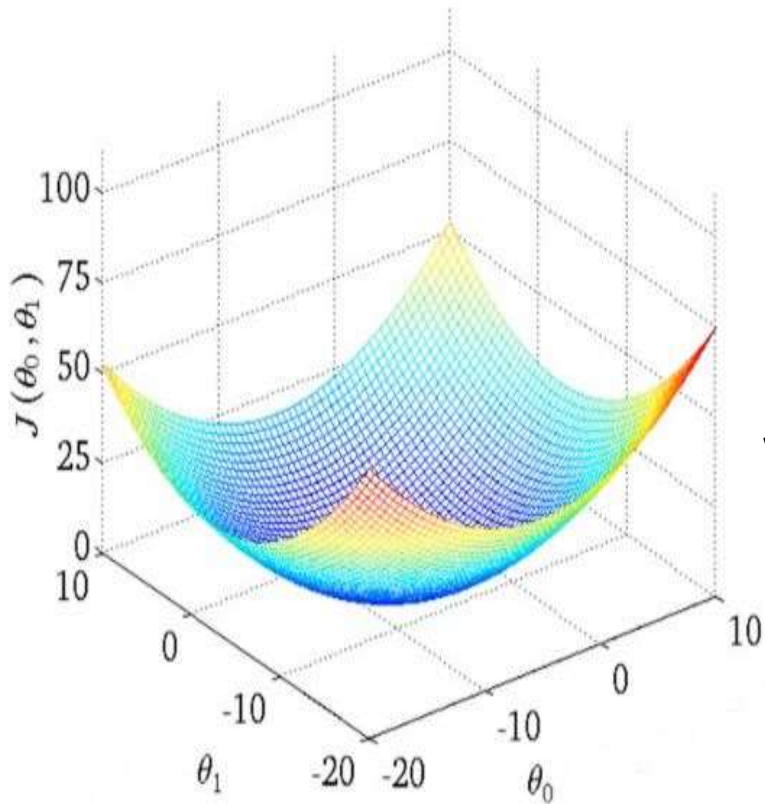
Fonction convexe

✓

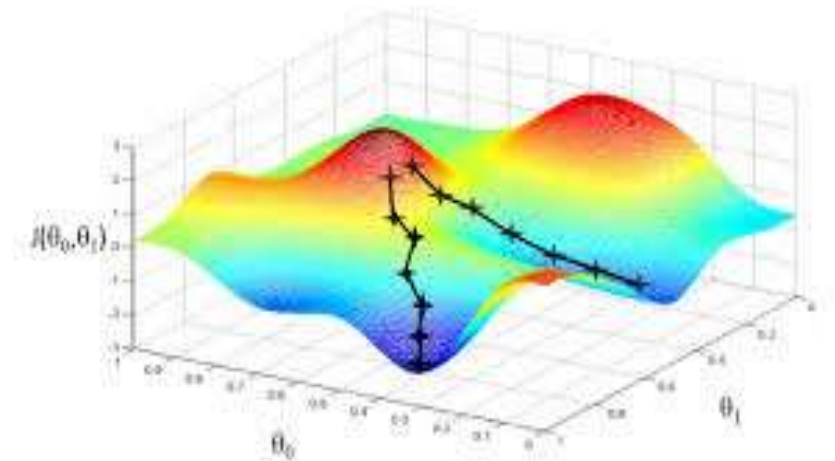


Une fonction est convexe si tout minimum local est aussi un minimum global

Fonction convexe



versus



Non Convexe

Convexe

Matrices

Éléments d'une matrice

$$M = \begin{pmatrix} 10 & 41 & 54 \\ 3 & 12 & 39 \\ 47 & 8 & 20 \\ 32 & 87 & 47 \end{pmatrix}$$

On parle de l'élément à la ligne i et à la ligne colonne j :

On peut le noter m_{ij}

Exemples: $m_{11} = 10$, $m_{12} = 41$, $m_{13} = 54$
 $m_{21} = 3$, $m_{22} = 12$, $m_{23} = 39$
 $m_{31} = 47$, $m_{32} = 8$, $m_{33} = 20$
 $m_{41} = 32$, $m_{42} = 87$, $m_{43} = 47$

Matrices

Une matrice c'est un tableau rectangulaire de valeurs numériques

On peut avoir n lignes et p colonnes, avec $n \geq 1$ et $p \geq 1$

Les valeurs sont des valeurs numériques de \mathbb{R} (nombres réels)

Exemple :

$$M = \begin{pmatrix} 10 & 41 & 54 \\ 3 & 12 & 39 \\ 47 & 8 & 20 \\ 32 & 87 & 47 \end{pmatrix}$$

M matrice à valeurs dans $\mathbb{R}^4 \times \mathbb{R}^3$

On dit que la dimension de M est : 4 x 3

Vecteur et matrice carrée

Vecteur : c'est une matrice $n \times 1$, n lignes et 1 colonne

Exemple :

$$\theta = \begin{pmatrix} 0.8 \\ 1.2 \\ 0.7 \\ 0.5 \\ 1.1 \end{pmatrix}$$

θ est un vecteur de dimension 4, à valeurs dans \mathbb{R}^4

Une matrice est dite carrée si elle possède le même nombre de lignes et de colonnes

Notation : A est carrée si sa dimension est de la forme $n \times n$

Somme de matrices

La somme de 2 matrices nécessite que les 2 matrices soient de même dimensions !
Sommer 2 matrices revient à sommer les éléments des matrices possédant les mêmes indices de lignes et de colonnes

Exemple:

$$\begin{pmatrix} 4 & 5 \\ 8 & 9 \\ 7 & 8 \end{pmatrix} + \begin{pmatrix} 12 & 22 \\ 8 & 30 \\ 10 & 11 \end{pmatrix} = \begin{pmatrix} 16 & 27 \\ 16 & 39 \\ 17 & 19 \end{pmatrix}$$

$$(A + B)_{ij} = A_{ij} + B_{ij}$$

Soustraction de matrices

$$(A - B)_{ij} = A_{ij} - B_{ij}$$

Multiplier une matrice par un scalaire

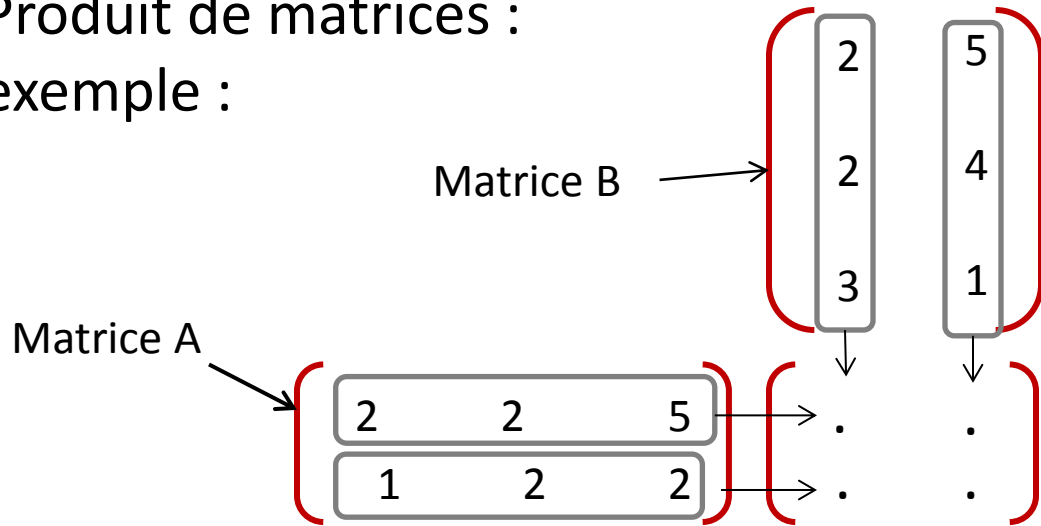
$$5 \begin{pmatrix} 5 & 1 & 2 \\ 4 & 5 & 4 \\ 3 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 25 & 5 & 10 \\ 20 & 25 & 20 \\ 15 & 15 & 5 \end{pmatrix}$$

Multiplier une matrice par scalaire λ revient à multiplier chaque terme de la matrice par λ :

$$(\lambda A)_{ij} = \lambda A_{ij}$$

Produit de matrices

Produit de matrices :
exemple :



Le nombre de colonnes de la matrice A = nombre de lignes de la matrice B

$$AB = \begin{pmatrix} 2 \times 2 + 2 \times 2 + 5 \times 3 & 2 \times 5 + 2 \times 4 + 5 \times 1 \\ 1 \times 2 + 2 \times 2 + 2 \times 3 & 1 \times 5 + 2 \times 4 + 2 \times 1 \end{pmatrix}$$

$$AB = \begin{pmatrix} 23 & 23 \\ 12 & 15 \end{pmatrix}$$

Produit de matrices

Chaque terme de AB s'obtient en faisant le produit scalaire d'une ligne de A par une colonne de B :



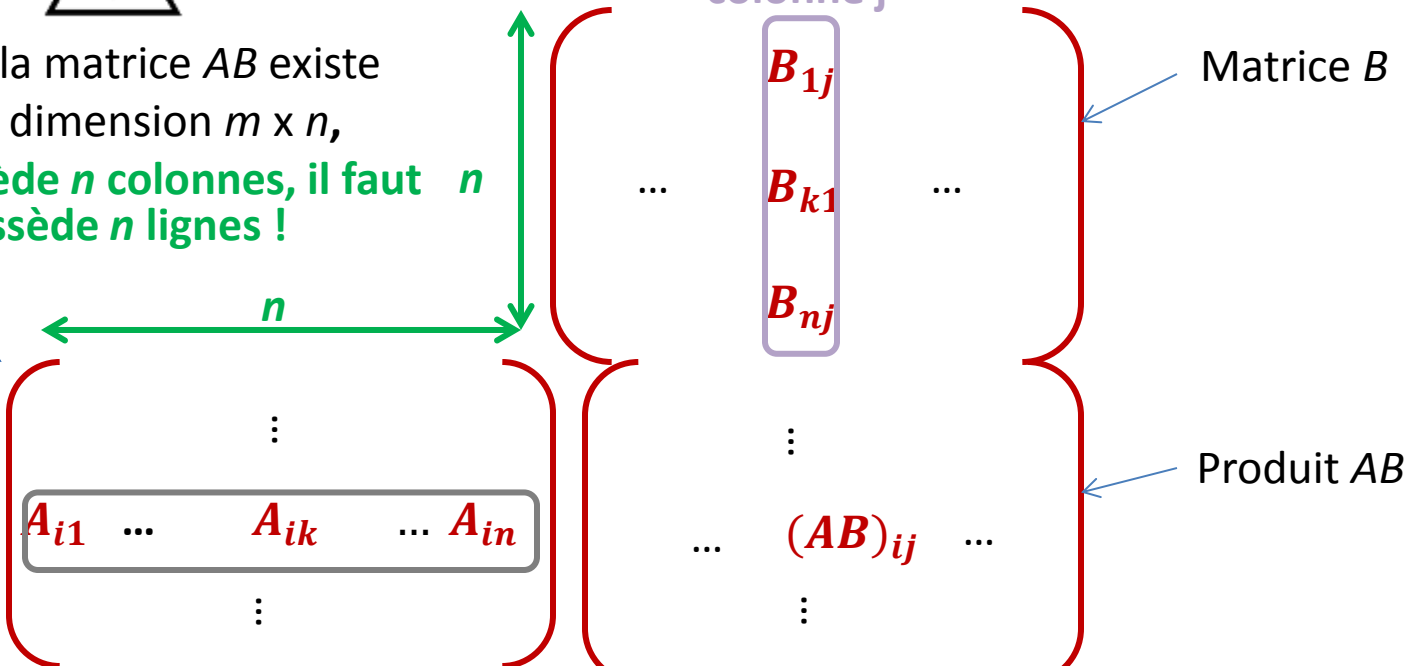
Pour que la matrice AB existe avec A de dimension $m \times n$,

Si A possède n colonnes, il faut que B possède n lignes !

Matrice A

colonne j

ligne i



Le terme de ligne i et de colonne j de AB s'obtient en multipliant le vecteur de la i ème ligne de A par le vecteur de la j ème colonne de B

$$(AB)_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$$

Produit de matrice et vecteur

Produit de matrice et vecteur:

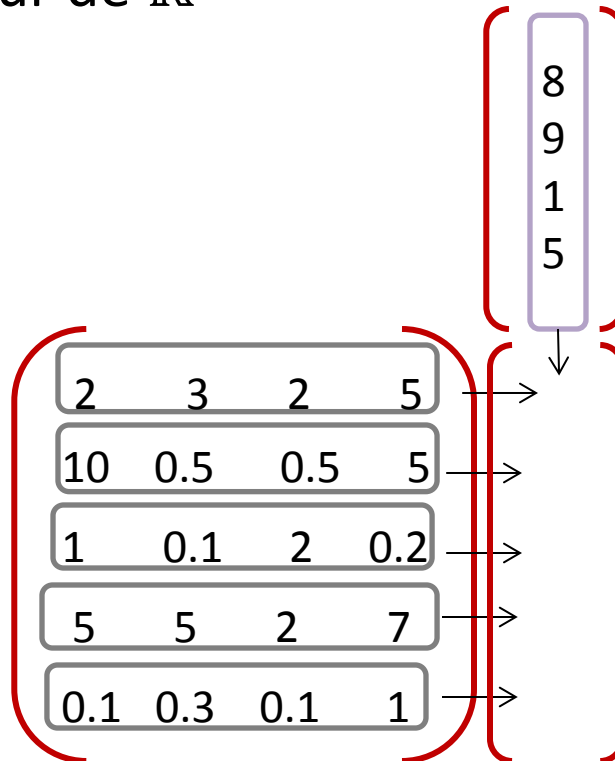
Comment multiplier la matrice M et le vecteur V ?

Sachant que M est une matrice à valeurs dans $\mathbb{R}^5 \times \mathbb{R}^4$

Et V est un vecteur de \mathbb{R}^4



**Le vecteur est
une matrice de
dimension 4 x 1**



Matrice identité

C'est une matrice carrée I telle que : $A \cdot I = I \cdot A = A$

Notation : si la dimension de I est n , I peut se noter $I_{n \times n}$

Elle est composée de 1's sur sa diagonale :

Exemples de matrices identité:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$I_{2 \times 2}$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$I_{3 \times 3}$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$


$I_{4 \times 4}$

Conséquence : si A est une matrice $m \times n$

$$A \cdot I_{n \times n} = I_{m \times m} \cdot A = A$$

Matrice transposée

Soit une matrice A de dimension $m \times n$


$$\mathbf{A} = \begin{pmatrix} 2 & 2 & 5 \\ 1 & 2 & 2 \end{pmatrix} \quad \mathbf{A}^T = \begin{pmatrix} 2 & 1 \\ 2 & 2 \\ 5 & 2 \end{pmatrix}$$

OU

Transposer une matrice revient créer une nouvelle matrice en mettant **les lignes de A** en colonnes ou en mettant **les colonnes de A** en lignes

$$\mathbf{A} = \begin{pmatrix} \begin{matrix} 2 \\ 1 \end{matrix} & \begin{matrix} 2 \\ 2 \end{matrix} & \begin{matrix} 5 \\ 2 \end{matrix} \end{pmatrix}$$

Le terme de ligne i et de colonne j de la matrice A^T s'obtient en prenant le terme de la ligne j et de la colonne i de la matrice A :



$$A_{ij}^T = A_{ji}$$

Matrice inverse

Une matrice inverse est une matrice A^{-1} telle que :

A soit une matrice carrée $n \times n$

avec $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{A}^{-1} \cdot \mathbf{A} = \mathbf{I}$



Toute matrice carrée n'est pas toujours inversible !
Dans ce cas on parle de matrice **singulière**

Exemple de matrice inverse:

$$A = \begin{pmatrix} 3 & 4 \\ 2 & 16 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{pmatrix}$$



```
Librairie Python : numpy.linalg.inv  
from numpy.linalg import inv  
import numpy as np
```

```
a = np.array([[3, 4], [2, 16]])  
ainv = inv(a)
```

Thank you

Machine Learning

Régression Linéaire
Régression Logistique
Gradient Descent

Régression linéaire & Gradient Descent

V. Gigliobianco



C'est quoi la régression linéaire?

- Un modèle qui approxime la réalité
- Salaire = $\theta_0 + \theta_1 * \text{Sexe} + \theta_2 * \text{Expérience} + \theta_3 * \text{Années d'études}$

*Les θ_i sont appelés coefficients du modèle.
Ou encore poids, paramètres, weights, etc.*



Mike est un homme, 1 an d'expérience, bac+5

$$\text{Salaire (Mike)} = 33 + 1 * 77 + 1 * 2.5 + 5 * 36 = 51.3K$$

! Le sexe, l'expérience et les années d'études sont les features du modèle

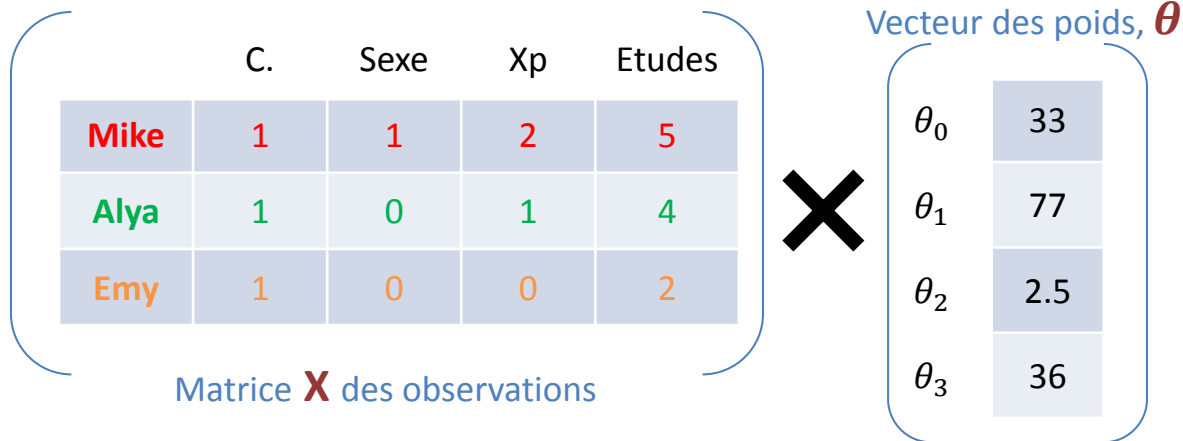
Le modèle se formalise par :

$$h_{\theta}(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \dots + \theta_n x_n^{(i)}$$



Prédiction des salaires

grâce au produit matriciel



Prédictions des salaires, $h(\theta)$

Détails du produit matriciel

Vecteur des poids

$X * \theta$

Matrice des observations



Mike

$$1*33 + 1*77 + 2*2.5 + 5*36 = 38K$$

Alya

$$1*33 + 0*77 + 1*2.5 + 4*36 = 31K$$

Emy

$$1*33 + 0*77 + 0*2.5 + 2*36 = 28K$$

Vecteur des prédictions



Prédiction des salaires

Sur Python

- Les données ici : Notebook « Exercices_cours_régression_linéaire.ipynb »
- Q1 – Prédisez les salaires de Marina, José, Hélène et Chris via un modèle de régression linéaire à l'aide d'une opération matricielle sachant
 - La matrice X représentant les features des 4 individus
 - Le vecteur Θ représentant les poids du modèle

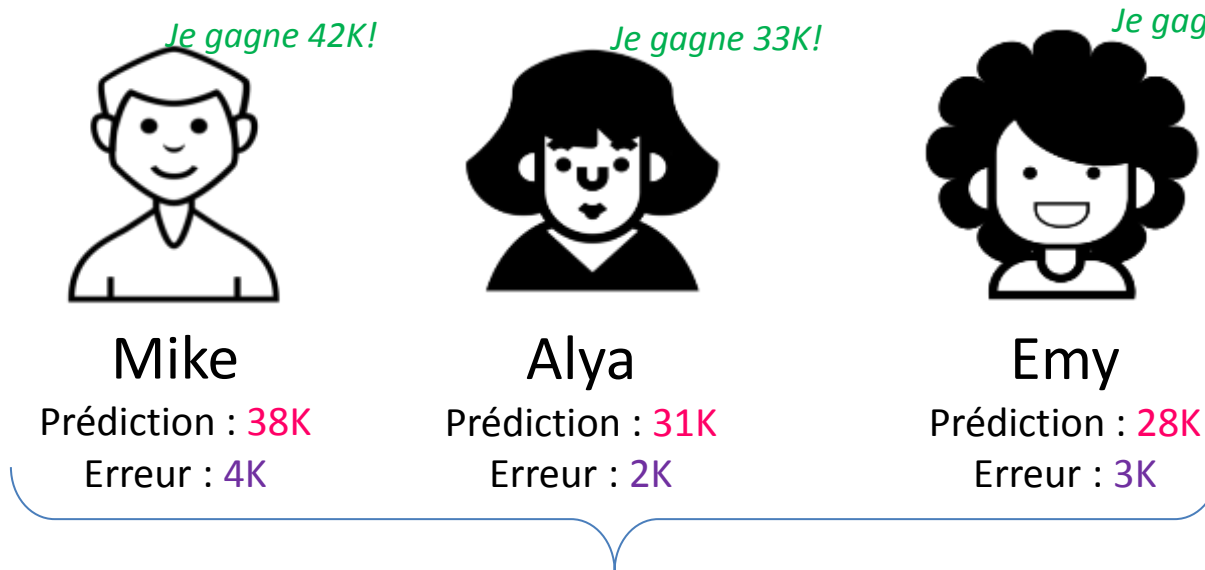
Tips

- Importer la librairie Numpy
 - `import numpy as np`
- Définir une matrice et un vecteur
 - `X = np.array([[1,1,2,5],[1,0,1,4],[1,0,0,2]])`
 - `theta = np.array([33,77,2.5,36])`
- Produit matriciel dans numpy
 - `predictions = X.dot(theta)`
- Afficher un objet dans Python 3x
 - `print(predictions)`



Qu'est ce que l'erreur du modèle?

- Erreur du modèle = Moyenne des erreurs des prédictions au carré
- Erreur de prédiction de i = La prédiction $h_{\theta}(x^i)$ – La réalité y^i



$$\text{Erreur du modèle} : (4^2 + 2^2 + 3^2)/3 = 9.7$$



Qu'est ce que l'erreur du modèle?

- Erreur du modèle $J(\theta)$ = Moyenne des erreurs des prédictions au carré
- Erreur de prédiction de i = La prédiction $h_{\theta}(x^i)$ – La réalité y^i

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (y^i - h_{\theta}(x^i))^2$$

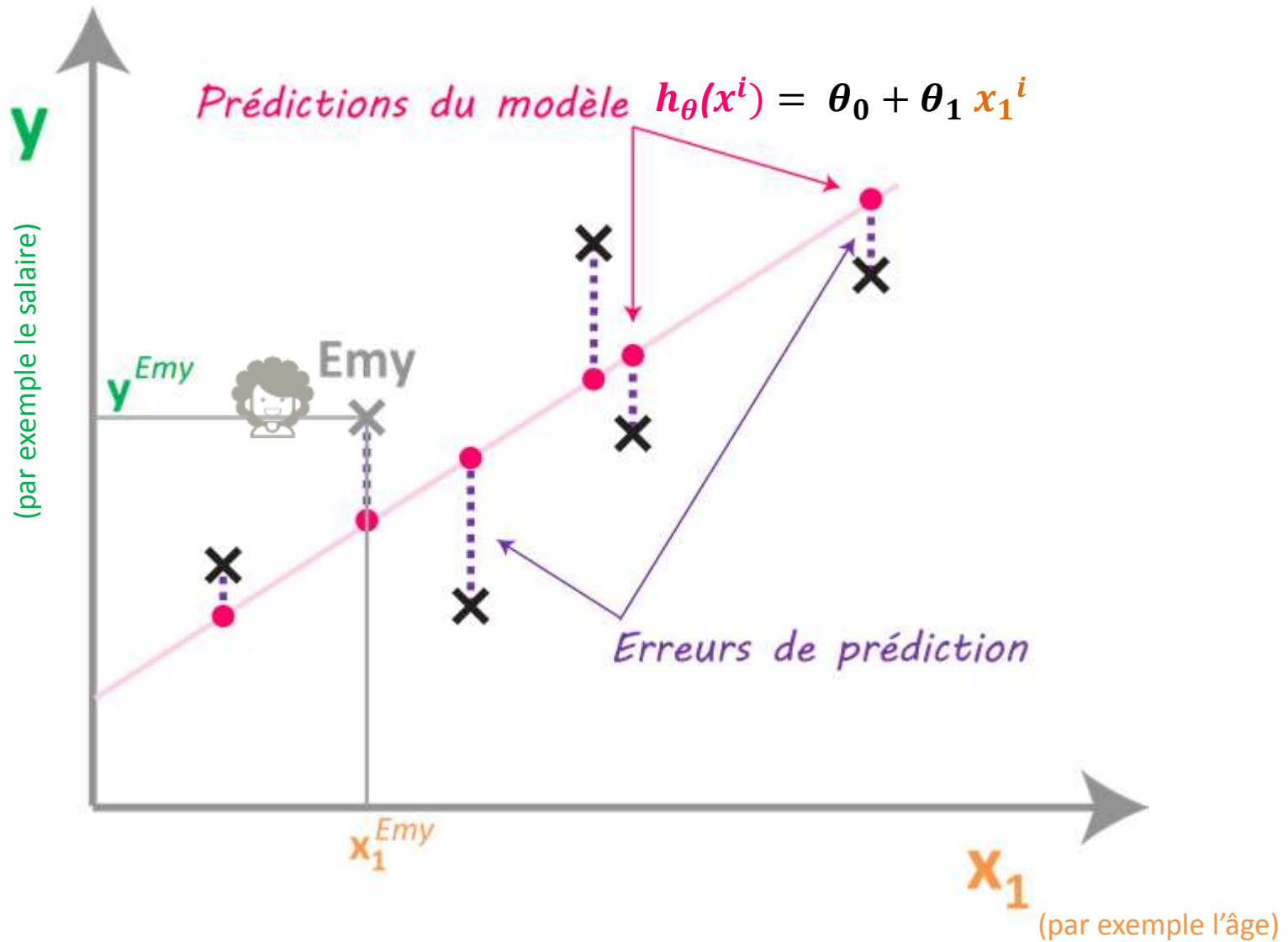
- $J(\theta)$ est l'erreur du modèle en fonction des θ_i
- y^i est la réalité de l'observation i
- $h_{\theta}(x^i)$ est la prédiction de l'observation i
- m est le nombre d'observations

Ecriture matricielle de J

$$J(\theta) = \frac{1}{m} (X * \theta - y)^T (X * \theta - y)$$



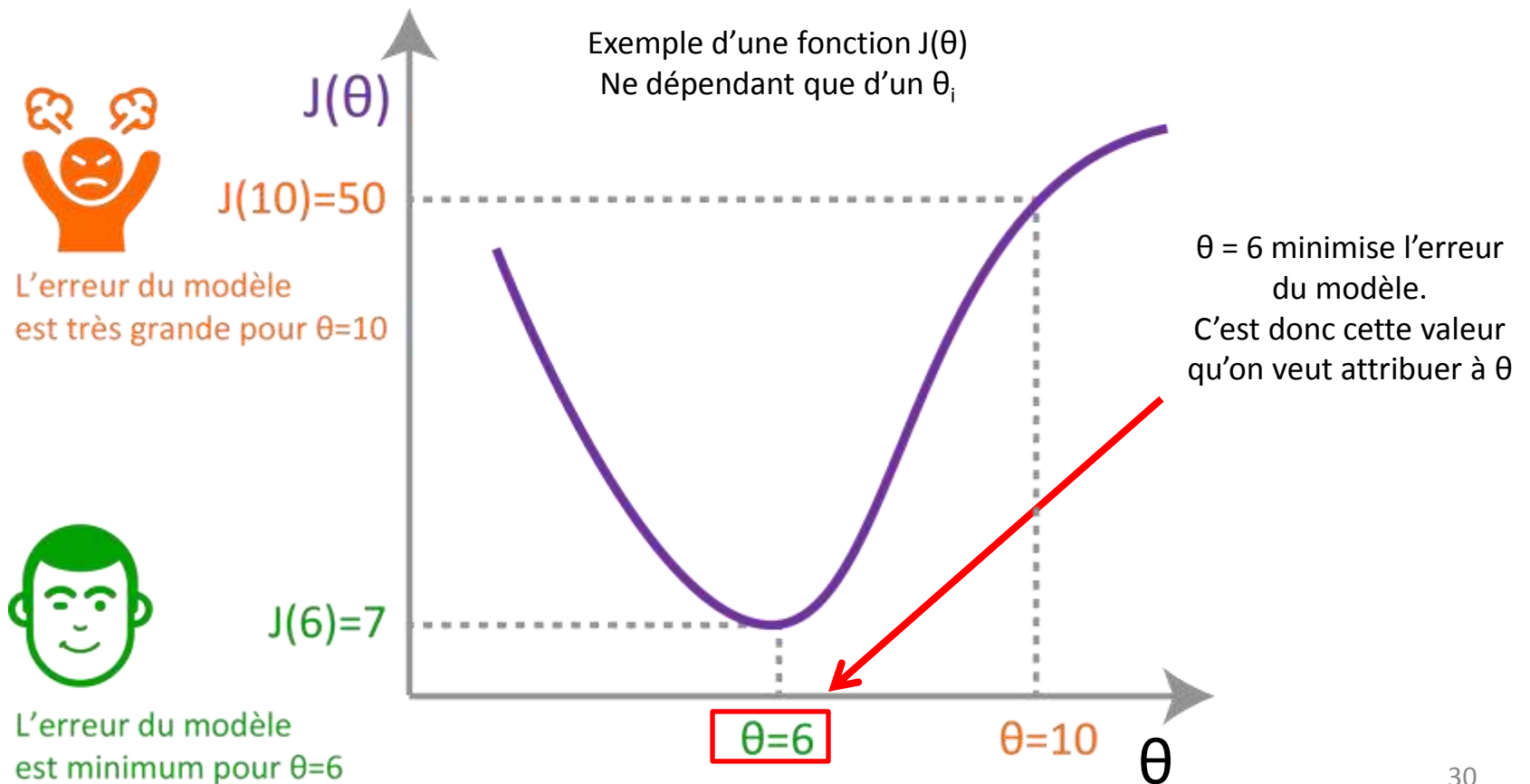
Rappel (si une seule feature x_1)





Comment choisit-on les poids du modèle?

- L'erreur $J(\theta)$ peut être vue comme une fonction qui représente l'erreur du modèle en fonction des valeurs des poids θ_i
- On cherche le poids θ_i tel que l'erreur du modèle, $J(\theta)$, soit minimum





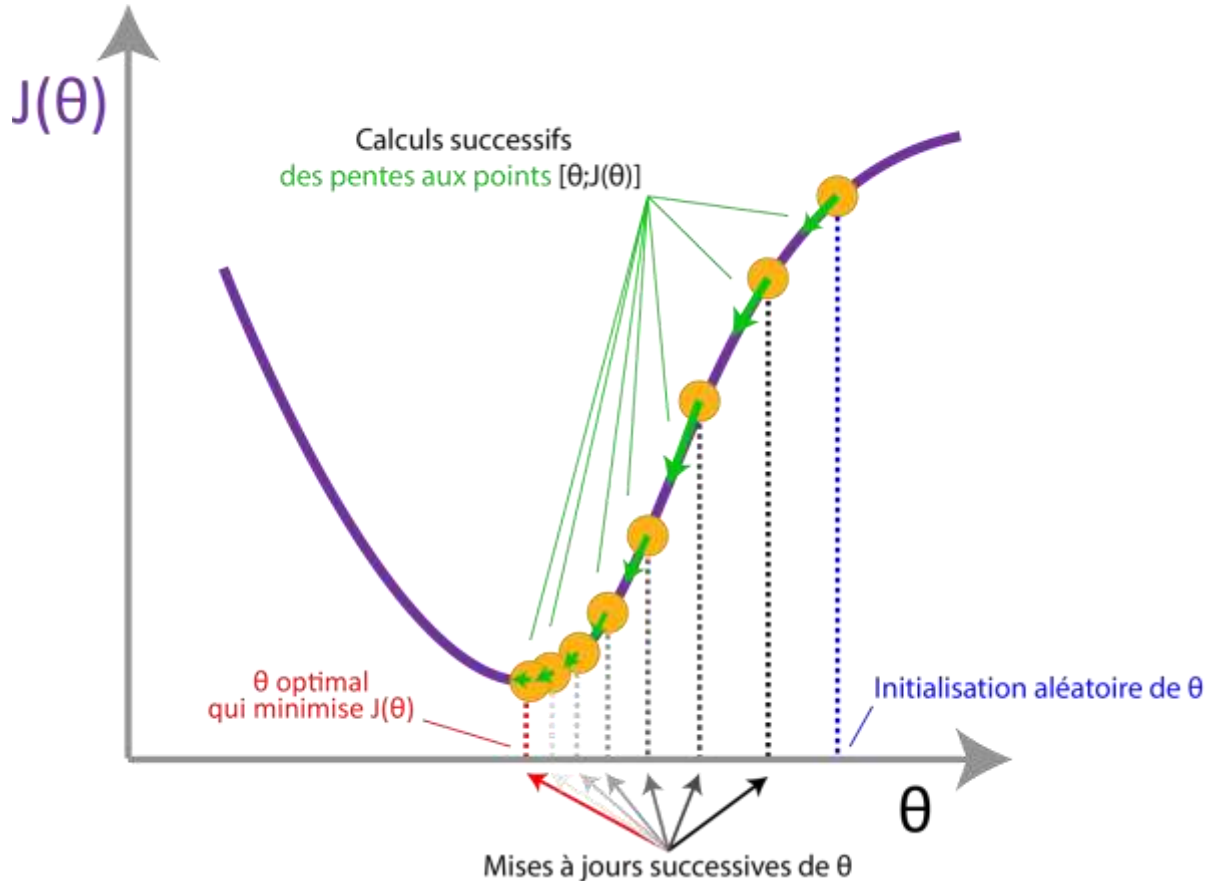
Le Gradient Descent

Comment trouver les θ_i tel que la fonction $J(\theta)$ soit minimum?

- Grâce au Gradient Descent !

Principe du **Gradient Descent** :

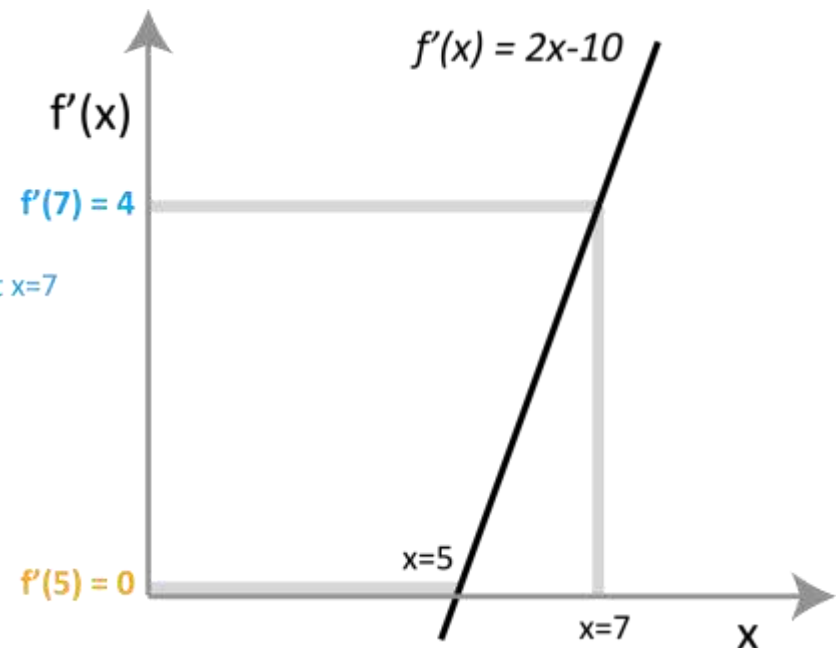
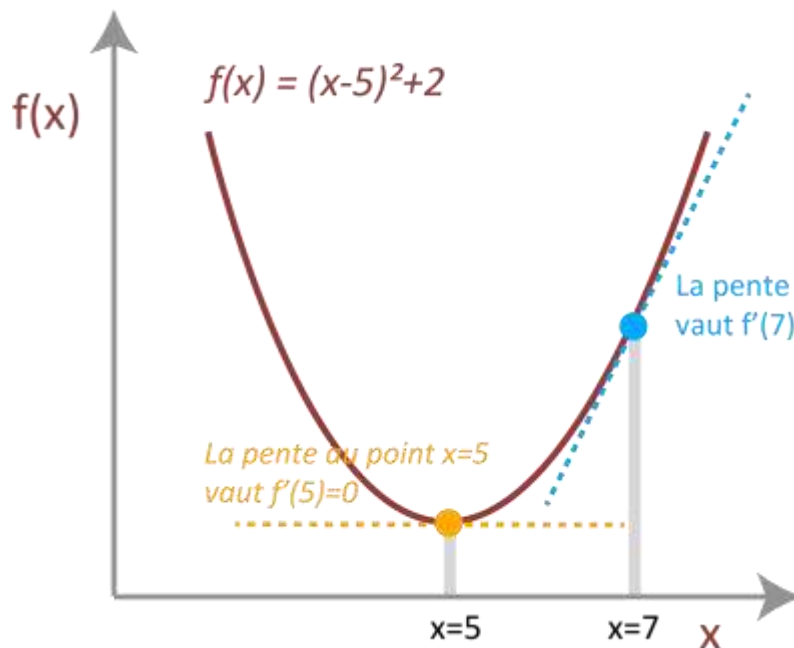
1. On se place aléatoirement sur un point de la fonction
2. On calcule la **pente** du point
3. On descend d'un pas de taille proportionnelle à la pente
4. On répète le processus jusqu'à ce que **la pente soit égale à zéro**





Calculer la pente en un point d'une fonction

- La pente en un point x^i d'une fonction $f(x)$, est donnée par
 - sa dérivée $f'(x^i)$





Gradient descent (ex)

Sur Python

- Exercice ici : Notebook« Exercices_cours_régression_linéaire.ipynb »
- Q1 - Afficher la fonction $J(\theta)=\theta^2$ sur $[-5;5]$
 1. Construire le np.array theta de -5 à 5 par pas de 0,5
 2. Construire le np.array J_theta contenant les valeurs de theta au carré
 3. Afficher les points
- Q2 - Grâce au Gradient Descent, trouver θ^i qui minimise J
 1. Tirer un θ^i de départ au hasard
 2. Calculer la pente au point θ^i
 3. Descendez d'un pas. $\theta^i = \theta^i - 0,1 * \text{pente}$
 4. Répétez l'opération jusqu'à ce que la pente $< 0,01$
 5. Sortez le θ^i

Tips

- $J'(\theta) = 2\theta$
- Choisir « theta » ou le tirer un nombre au hasard entre deux bornes
- Boucle en python
 - ```
for i in ma_liste :
 do something
```
- Élever au carré un numpy array x
  - ```
np.power(x, 2)
```
- Plot des points dont les coordonnées sont stockés dans deux vecteurs x et y
 - ```
import matplotlib.pyplot as plt
plt.plot(x, y, '-o')
plt.show()
```



# Algorithme du Gradient Descent

(Généralisation à n variables)

1. Initialiser aléatoirement les  $\theta_i$
2. *Début de la boucle* - tant que la fonction J décroît, faire :

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\theta_j := \theta_j - \alpha * \frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_j} \text{ avec } \frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Learning rate



## ATTENTION!

Calculer simultanément les nouvelles valeurs de  $\theta_0, \theta_1, \dots, \theta_n$

### Paramètres du GD

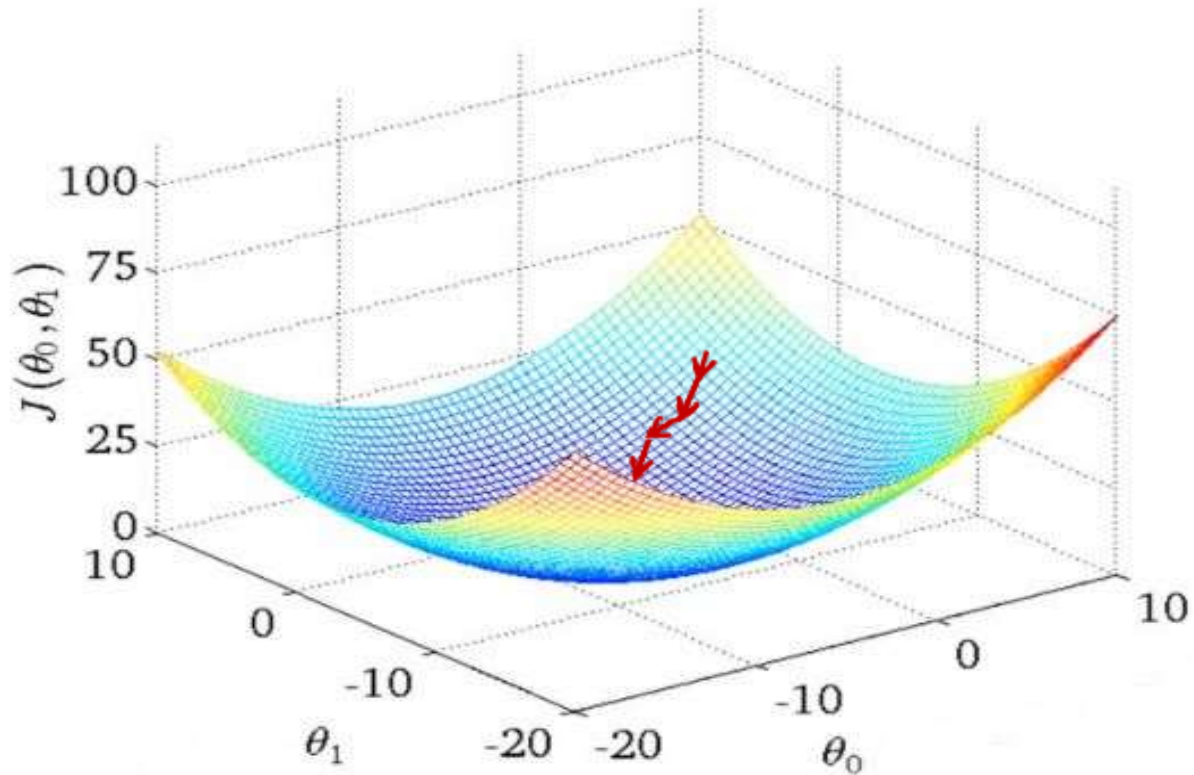
- Le learning rate  $\alpha$
- Nombre maximal d'itérations (*éventuellement*)





# Illustration du Gradient Descent

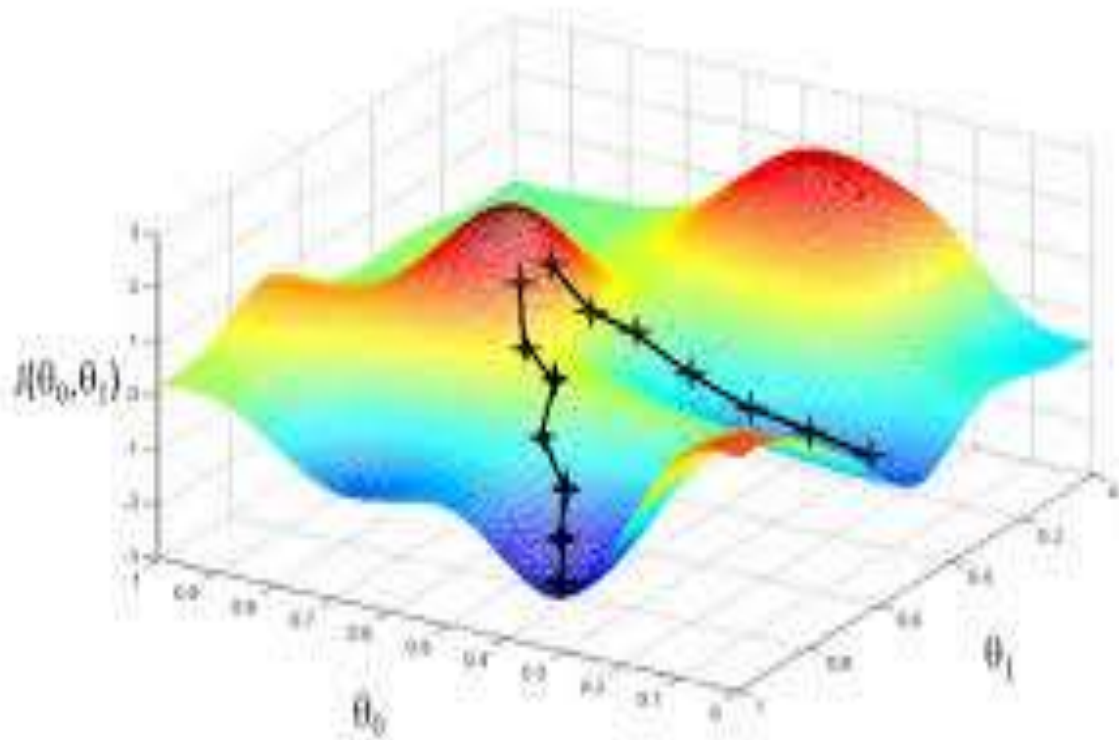
(Régression linéaire, cas avec deux coefficients  $\theta_0$  et  $\theta_1$ )





# Illustration du Gradient Descent

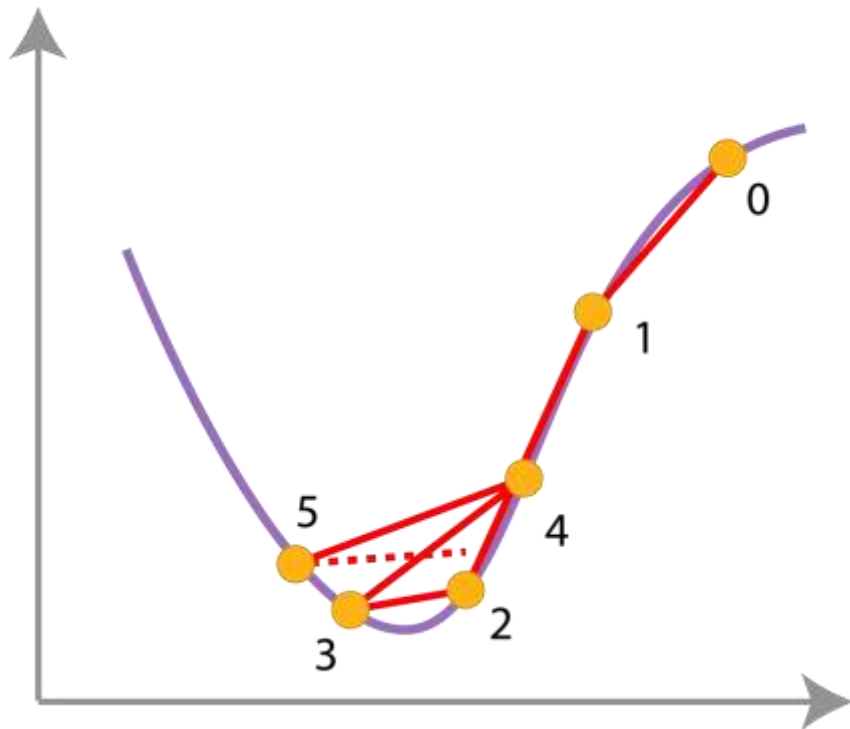
(Neural net, fonction non-convexe, coefficients  $\theta_0$  et  $\theta_1$ )





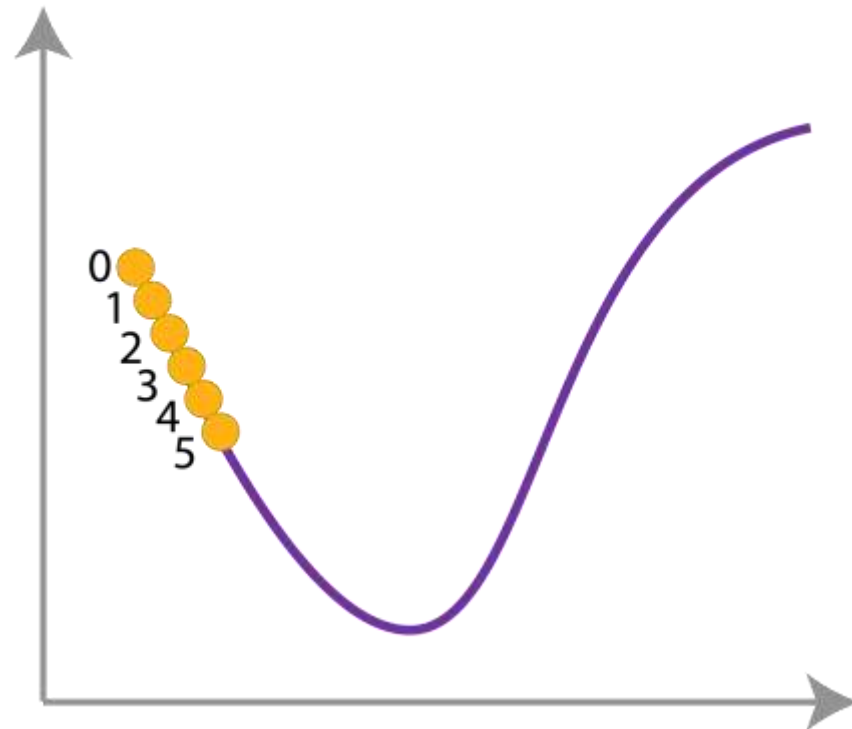
# Définir le learning rate alpha

Learning rate trop élevé



\* Si alpha trop élevé, on peut être balancé sur les différentes parois de la fonction  $J$  et **ne jamais atteindre le theta optimal**

Learning rate trop faible



\* Si alpha trop faible, la convergence vers le theta optimal peut être **extrêmement lente**



*Afin de vérifier que votre code ne contient ni bugs ni learning rate trop élevé, il est recommandé de vérifier que le coût du modèle décroît à chaque itération*



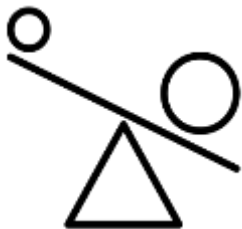
## Alternative pour estimer les poids $\theta_i$ : « équations normales »

Il y a la possibilité de trouver des valeurs exactes des  $\theta_i$

Revient à résoudre le système d'équations linéaires :  $X^T X \theta = X^T y$



$$\theta = (X^T X)^{-1} X^T y \quad \text{avec } X^T X \text{ matrice carrée } n \times n$$



On obtient des valeurs exactes de  $\theta$  si la dimension si  $n \approx 100, 1000$  features

Mais calcul très coûteux pour inverser une matrice avec  $n \approx 10^4, 10^5, 10^6$  features

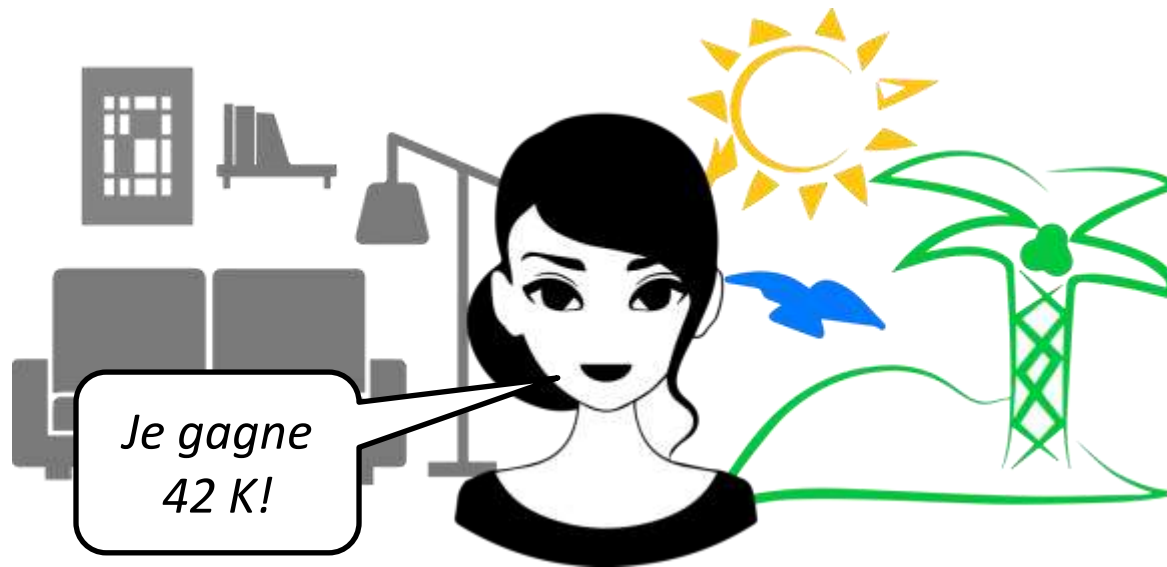
# Régression logistique & Gradient Descent

V. Gigliobianco



# C'est quoi la régression logistique?

- *Un modèle qui estime la probabilité de détenir une caractéristique en fonction de **features diverses***



- Quelle est la probabilité que **Lilou** parte en vacances, sachant **son salaire**?

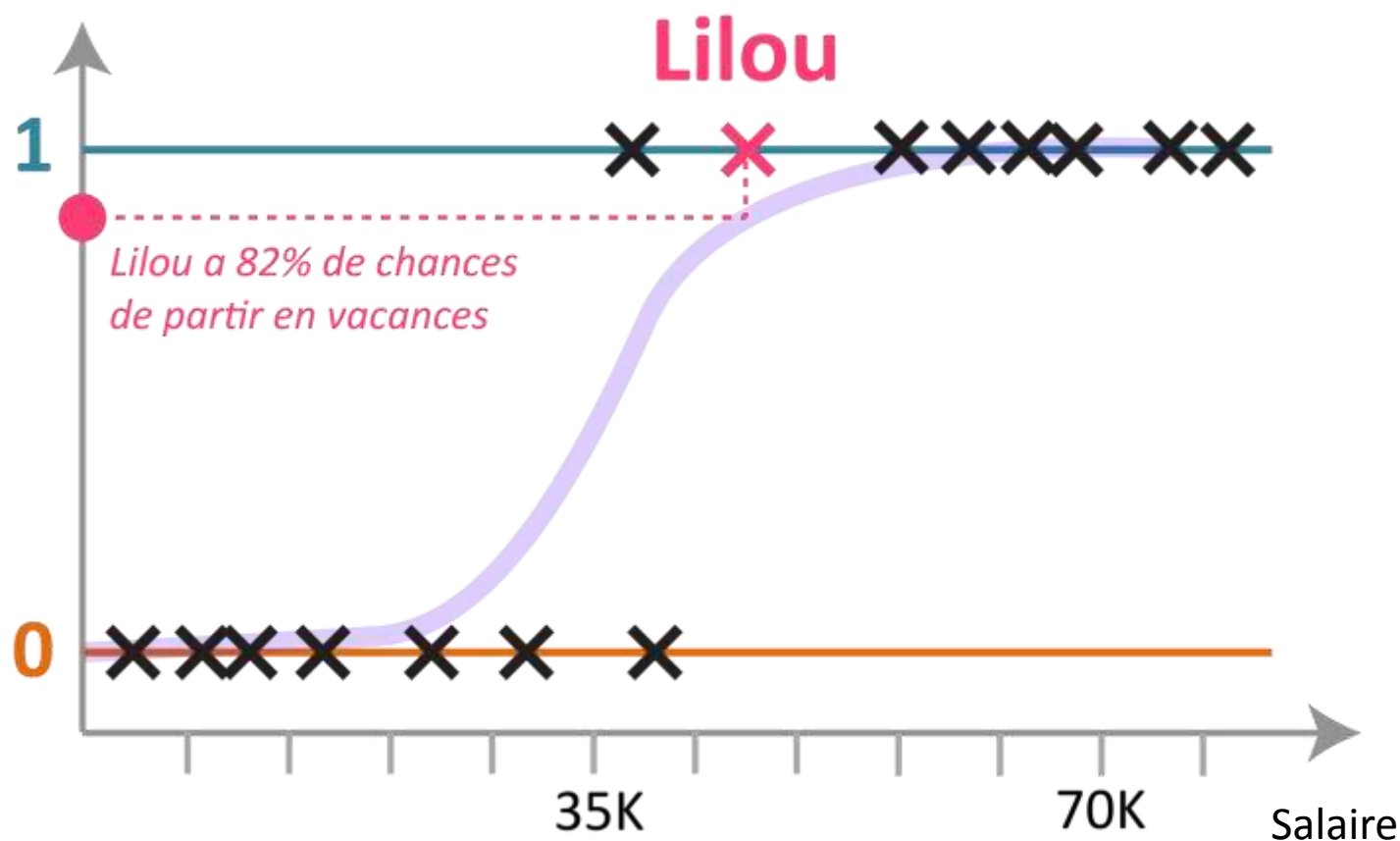




# Principe de la régression logistique

Fiter un nuage de points dont la caractéristique est distribué sur 2 valeurs

- Valeur 1 : *les individus partent en vacances*
- Valeur 0 : *les individus ne partent pas en vacances*





# Le modèle de la régression logistique

Il s'écrit :

- $h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n)$
- $h_{\theta}(x)$  correspond aux probabilités prédites, i.e.  $P(y = 1 \mid \mathbf{X}; \theta_0, \theta_1, \dots, \theta_n)$
- La fonction  $g$  est la fonction sigmoïde
- $x_1, x_2, \dots, x_n$  représentent les  $n$  features du modèle
- $\theta_0, \theta_1, \dots, \theta_n$  représentent les  $n+1$  coefficients du modèle à estimer

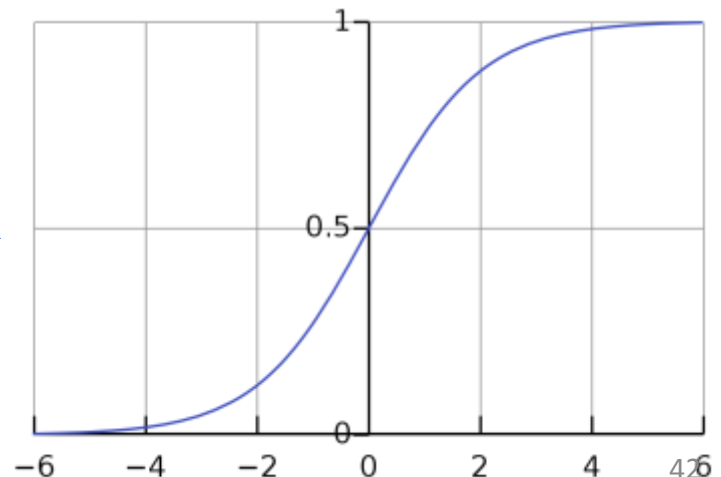
En détaillant, il s'écrit :

- $$h_{\theta}(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n)}}$$

Matriciellement, les prédictions s'écrivent :  $g(\mathbf{X} * \boldsymbol{\theta})$

La fonction sigmoïde

$$f(t) = \frac{1}{1 + e^{-t}}$$



# Remarque

- Régression linéaire
  - $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$
- Régression logistique
  - $h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n)$
  - Avec  $g$  fonction sigmoïde



# Rappel

- Objectif principal du machine learning
  - Estimer les coefficients  $\theta_i$  du modèle qui minimisent l'erreur de celui-ci
- Pour estimer les coefficients  $\theta_i$  du modèle
  - Application du Gradient Descent
    - A. Définir une fonction de coût  $J(\vartheta)$
    - B. Initialiser les  $\vartheta_i$
    - C. Mise à jour des  $\vartheta_i$



# Le Gradient Descent pour la régression logistique

## A. La fonction de coût $J(\theta)$

$$J(\theta_0, \theta_1, \dots, \theta_n) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))$$

## B. Initialiser aléatoirement les $\theta_i$

## C. Mise à jour des $\theta_i$ tant que l'erreur décroît, faire :

$$\theta_j := \theta_j - \alpha * \frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_j}$$

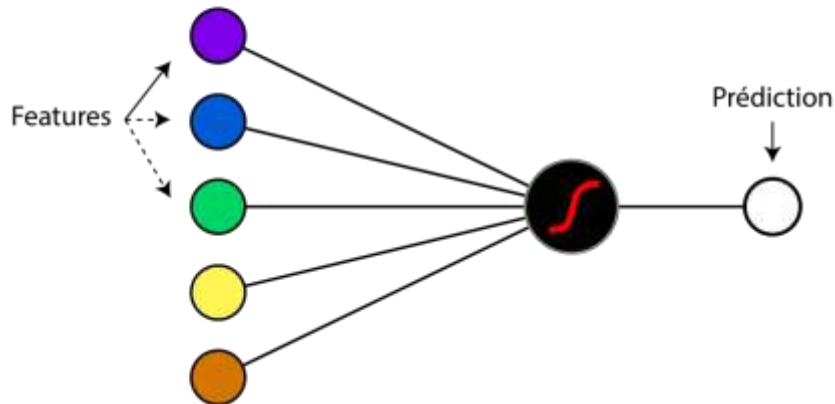
avec  $\frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$

Learning rate

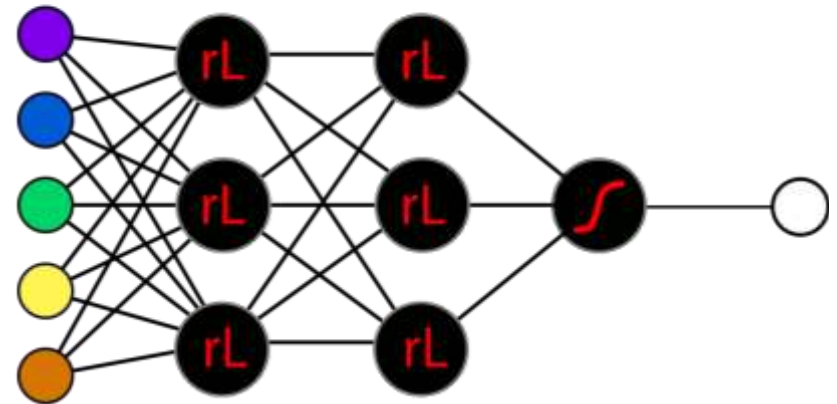
# Aller plus loin



Régression Logistique



Deep Learning  
Réseaux de Neurones



Octroi de crédit

Voiture autonome



Thank you

# Best practices - gradient descent : feature scaling

Si les  $n$  features ont des ordres de grandeurs différents

L'idée : rendre toutes features comparables :

- Enlever la moyenne
- Diviser par l'écart-type (ou par le range ie max – min)



Conséquence: les nouvelles variables sont à valeurs comprises dans  $[-1;1]$

⇒ l'algorithme « gradient descent » va converger plus rapidement !

S'applique aussi bien à la régression linéaire que logistique !

Exemple:

$X_1$  = surface d'une maison (0 – 1000 m<sup>2</sup>)

$X_2$  = nombre de chambres (1-5)



# Aller plus loin



## Gradient descent : best practices - optimisation

Possible d'avoir un GD qui nécessite un nombre d'itérations élevé et un alpha petit:  
Décroissance très lente de  $J$  vers le minimum  
Notamment en régression logistique

Utiliser les algorithmes :

- Conjugate gradient
- BFGS
- LBFGS



Algorithme compliqué



On ne fixe pas  $\alpha$   
Algorithme très performant

## Stochastic gradient descent (scikit-learn)

Si le nombre d'observations  $m$  est très élevé, utiliser SGD et non le gradient descent, car celui-ci est moins performant !



# Rappel

Jusqu'ici, utilisation d'un seul échantillon d'apprentissage !

Afin de minimiser la fonction de coût  $J$

Utilisation du gradient descent pour estimer les paramètres  $\theta_0, \theta_1, \dots, \theta_n$

Régression linéaire

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} [\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2]$$

Régression  
logistique

$$J(\theta_0, \theta_1, \dots, \theta_n) = -\frac{1}{m} [\sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))]$$

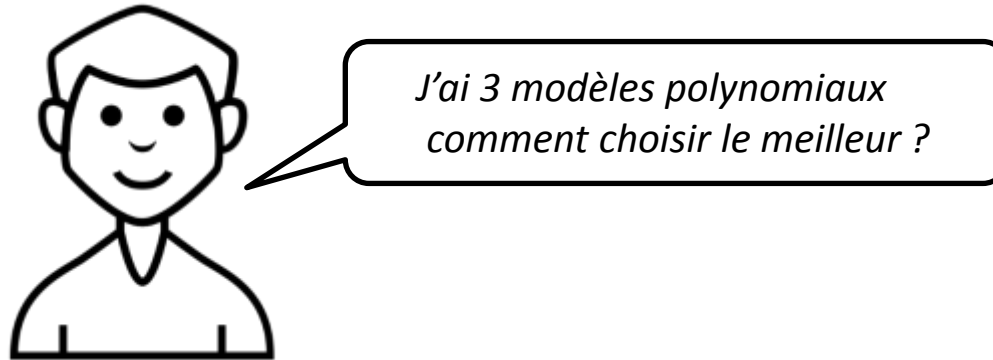
Gradient

$$\theta_j := \theta_j - \alpha * \frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_j}$$

avec  $\frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$

Learning rate

# C'est quoi un bon modèle ?



*J'ai 3 modèles polynomiaux  
comment choisir le meilleur ?*

Comment aider Mike à trouver un bon modèle ?

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

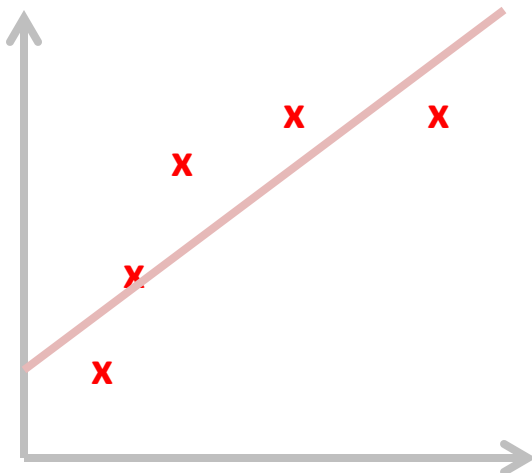
Un bon modèle est capable de bien généraliser, c'est-à-dire, être prédictif pour des données n'ayant pas servi à l'apprentissage du modèle !

# Le sur-apprentissage : c'est quoi ?

Régression linéaire polynomiale basée sur une feature  $X$   
On construit  $p$  features,  $X^1, X^2, \dots, X^p$

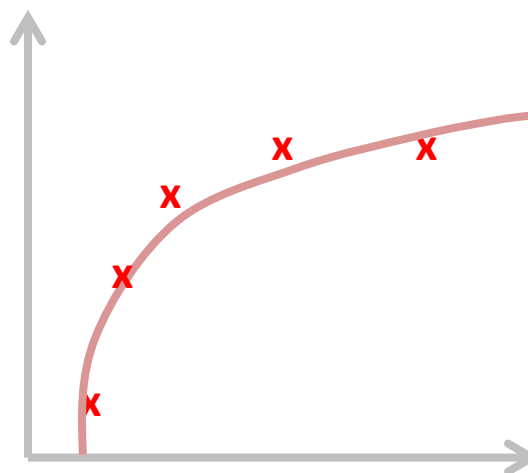


**Un échantillon d'apprentissage est insuffisant !**



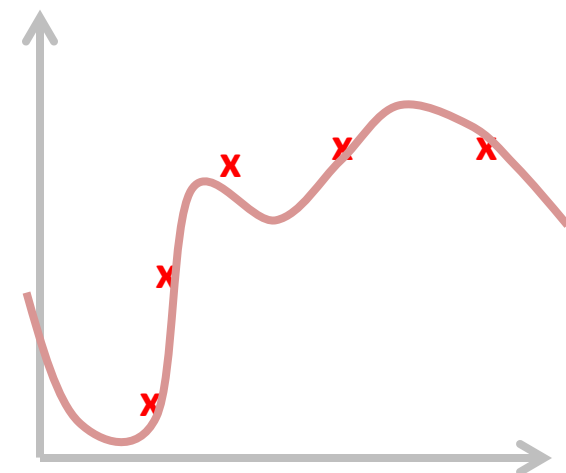
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Sous-apprentissage



$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Ajustement correct



$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Sur-apprentissage

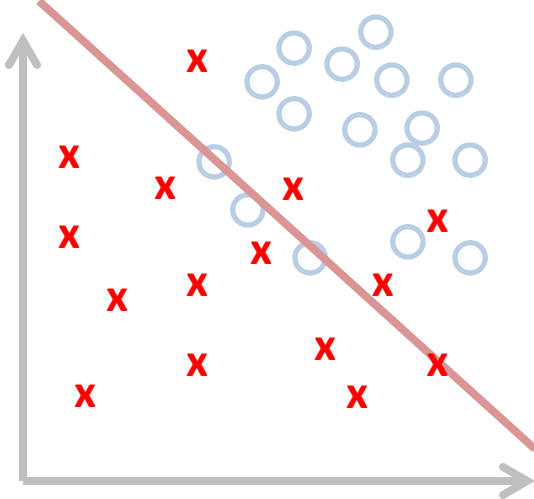
But: trouver un modèle qui sait prédire sur de nouvelles observations!

# Le sur-apprentissage : c'est quoi ?

Régression logistique avec 2 features

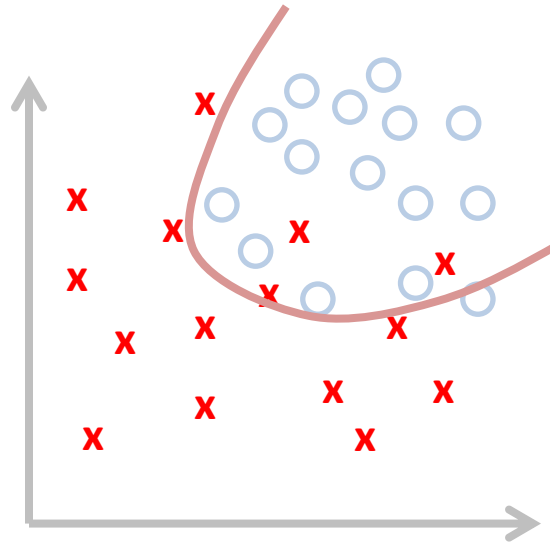


**Un échantillon d'apprentissage est insuffisant !**



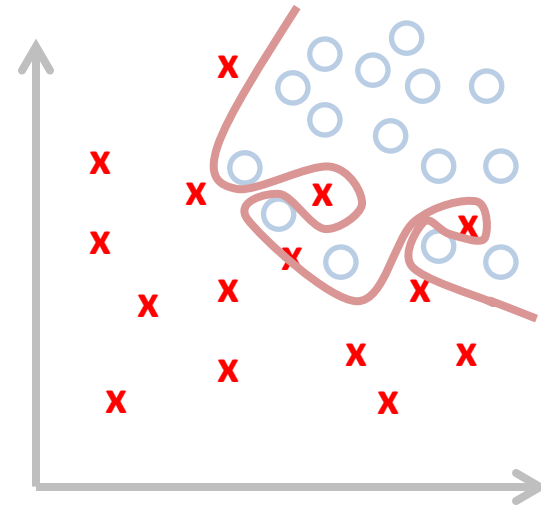
$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Sous-apprentissage



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$

Ajustement correct



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_5 x_1^3 x_2 + \dots)$$

Sur-apprentissage



## Échantillons **apprentissage** - **validation** - **test**

L'échantillon d'apprentissage va servir à **estimer les paramètres**  $\theta_0, \theta_1, \dots, \theta_n$  et calculer **l'erreur du modèle pour l'échantillon d'apprentissage** :

$$J_{\text{apprent}}(\theta_0, \theta_1, \dots, \theta_n)$$

L'échantillon de validation va servir à appliquer le modèle et **choisir le modèle**  
Et évaluer l'erreur du modèle en calculant **l'erreur du modèle pour l'échantillon de validation** :

$$J_{\text{valid}}(\theta_0, \theta_1, \dots, \theta_n)$$

L'échantillon de test va servir à **estimer l'erreur de généralisation**. On pourra aussi calculer **l'erreur du modèle pour l'échantillon de test** :

$$J_{\text{test}}(\theta_0, \theta_1, \dots, \theta_n)$$

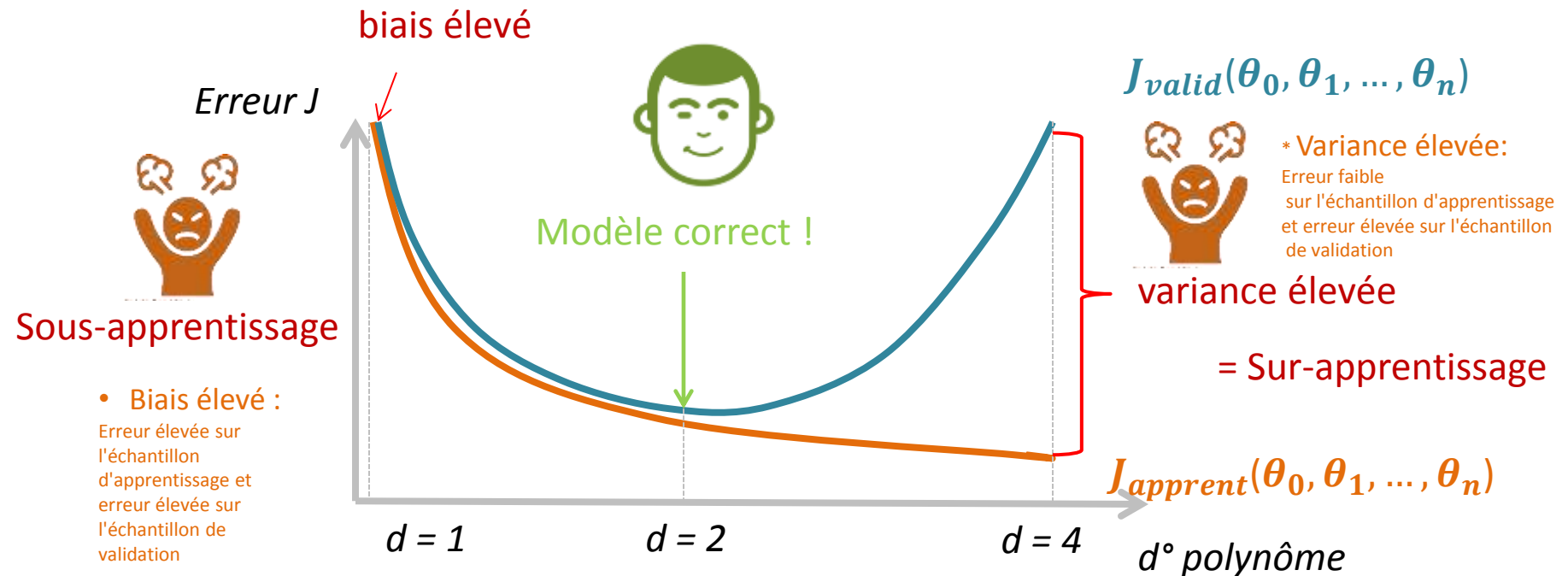
Erreur de  
généralisation  
faible !

$$\Leftrightarrow \begin{cases} J_{\text{apprent}}(\theta_0, \theta_1, \dots, \theta_n) \text{ faible} \\ J_{\text{apprent}}(\theta_0, \theta_1, \dots, \theta_n) \approx J_{\text{valid}}(\theta_0, \theta_1, \dots, \theta_n) \approx J_{\text{test}}(\theta_0, \theta_1, \dots, \theta_n) \end{cases}$$

# Erreur du modèle : best practice n°1

## biais versus variance

En pratique, il faut obtenir une erreur petite sur l'échantillon d'apprentissage et sur l'échantillon de validation



# Régularisation et \*GD : best practice n°2



**Risque de sur-apprentissage plus fort si le nombre de features est élevé !**



Améliorer le modèle grâce à la **RÉGULARISATION** :  
**lutter contre le sur-apprentissage**

**Fonction de coût J à minimiser !**

en utilisant  $\lambda$  paramètre de régularisation

de manière à réduire l'importance des paramètres  $\theta_1, \dots, \theta_n$

Régression linéaire  $J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} [ \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 ]$

Régression logistique  $J(\theta_0, \theta_1, \dots, \theta_n) = -\frac{1}{m} [ \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) + \lambda \sum_{j=1}^n \theta_j^2 ]$



**Rôle de  $\lambda$  : trade-off**

- arriver à ajuster les données d'apprentissage
- maintenir les paramètres  $\theta_1, \dots, \theta_n$  petits

Remarque : pas besoin de régulariser relativement au paramètre  $\theta_0$



# Régularisation et \*GD : best practice n°2

Fonction de coût  $J$  avec régularisation :

Régression linéaire  $J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} [ \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 ]$

Régression logistique  $J(\theta_0, \theta_1, \dots, \theta_n) = -\frac{1}{m} [ \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) + \lambda \sum_{j=1}^n \theta_j^2 ]$

Terme de régularisation

Expressions du gradient :



Régression linéaire

$$\theta_0 := \theta_0 - \alpha * \frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_0}$$

$$\text{avec } \frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

Régression  
logistique

$$\theta_j := \theta_j - \alpha * \frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_j}$$

$$\text{avec } \frac{\partial J(\theta_0, \theta_1, \dots, \theta_n)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j$$

# GD et régularisation : choix de $\lambda$

**Objectif** : fixer le paramètre  $\lambda$  afin d'estimer les coefficients  $\theta_1, \dots, \theta_n$  du modèle qui minimisent l'erreur  $J(\theta_0, \theta_1, \dots, \theta_n)$

$\lambda$  très grand implique que les paramètres  $\theta_1, \dots, \theta_n$  vont être très petits  
favoriser plutôt le sous-apprentissage

$\lambda$  très petit favorise aucune régularisation + hypothèse = on fait du sur-apprentissage



**Conséquence** : faire très attention au choix de  $\lambda$ , afin qu'il ne soit ni trop grand mais aussi ni trop petit

# Régularisation et équations normales

Faire varier  $\lambda$



$$\theta = (X^T X + \lambda I_{regul})^{-1} X^T y$$

avec  $X^T X$  matrice carrée  $n + 1 \times n + 1$   
 $n$  nombre de features

Avantage : la matrice  $X^T X + \lambda I_{reg}$  est toujours inversible !

$I_{regul} =$

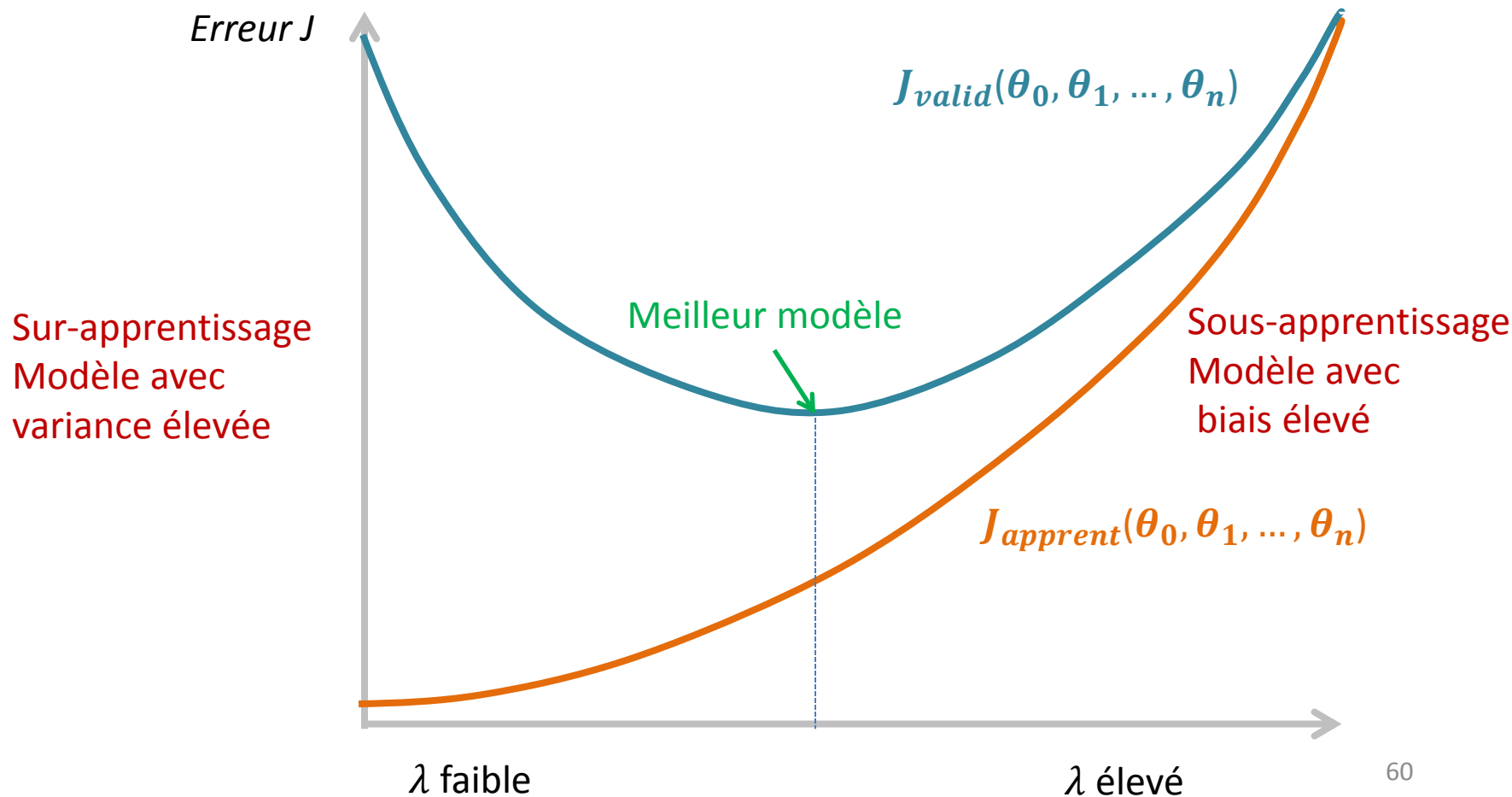
$$\begin{pmatrix} 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & \dots & 0 & 1 \end{pmatrix}$$

$I_{reg}$  matrice carrée  $n+1 \times n+1$

# Régularisation: best practice n°2

## Biais versus variance

Hypothèse :  
notre modèle fait du sur-apprentissage



# Pratique de la régularisation

Au départ, modèle sans régularisation, qui fait du sur-apprentissage !

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

On fait varier  $\lambda$

\*  $\lambda$  très petit  $\Leftrightarrow$  Minimiser J revient à estimer  $\theta_1, \theta_2, \theta_3$  et  $\theta_4$  sans contraintes sur  $\theta_1, \theta_2, \theta_3$  et  $\theta_4 \Leftrightarrow$  estimer  $\theta_1, \theta_2, \theta_3$  et  $\theta_4$  sans régularisation !

\*  $\lambda$  valeur intermédiaire  
 $\Leftrightarrow$  Minimiser J revient à estimer  $\theta_1, \theta_2, \theta_3$  et  $\theta_4$  avec  $\theta_3 \approx 0$  et  $\theta_4 \approx 0$

\*  $\lambda$  très grand  $\Leftrightarrow$  Minimiser J revient à estimer  $\theta_1, \theta_2, \theta_3$  et  $\theta_4$  avec  $\theta_1 \approx 0, \theta_2 \approx 0, \theta_3 \approx 0$  et  $\theta_4 \approx 0$

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Ajustement correct

$$h_{\theta}(x) \approx \theta_0$$

Sous-apprentissage !



# Régularisation et apprentissage - validation - test

Estimer les paramètres  $\theta_0, \theta_1, \dots, \theta_n$  cette fois, pour chaque valeur de  $\lambda$  !  
et évaluer l'erreur du modèle pour l'échantillon d'apprentissage :

$$J_{\text{apprent}}(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Appliquer le modèle sur l'échantillon de validation

Evaluer l'erreur du modèle pour l'échantillon de validation :

$$J_{\text{valid}}(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

afin de **choisir le modèle**

Estimer l'erreur de généralisation. Évaluer l'erreur du modèle pour l'échantillon de test :

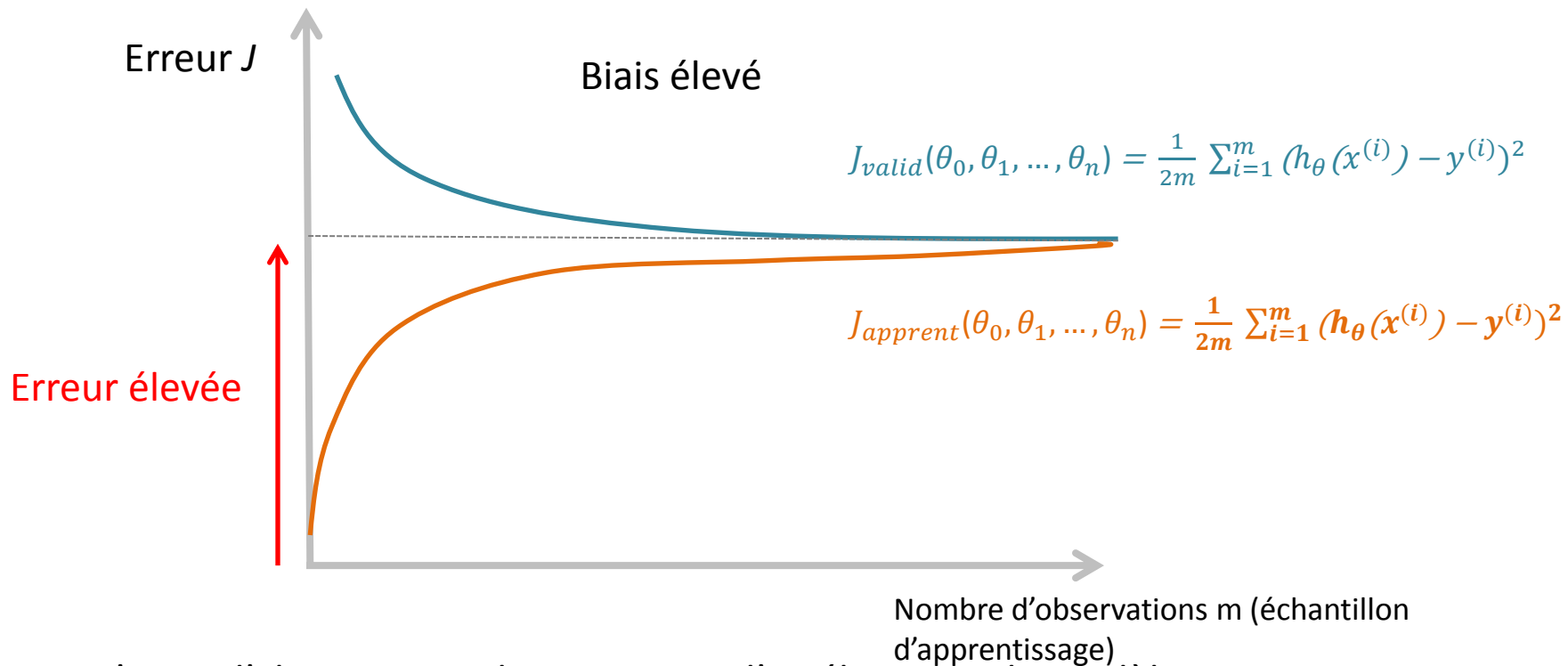
$$J_{\text{test}}(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Erreur de  
généralisation  
faible !

$$\Leftrightarrow \begin{cases} J_{\text{apprent}}(\theta_0, \theta_1, \dots, \theta_n) \text{ faible} \\ J_{\text{apprent}}(\theta_0, \theta_1, \dots, \theta_n) \approx J_{\text{valid}}(\theta_0, \theta_1, \dots, \theta_n) \approx J_{\text{test}}(\theta_0, \theta_1, \dots, \theta_n) \end{cases}$$

# Learning curves : diagnostiquer un biais élevé

Le modèle souffre -t-il d'un biais élevé ?

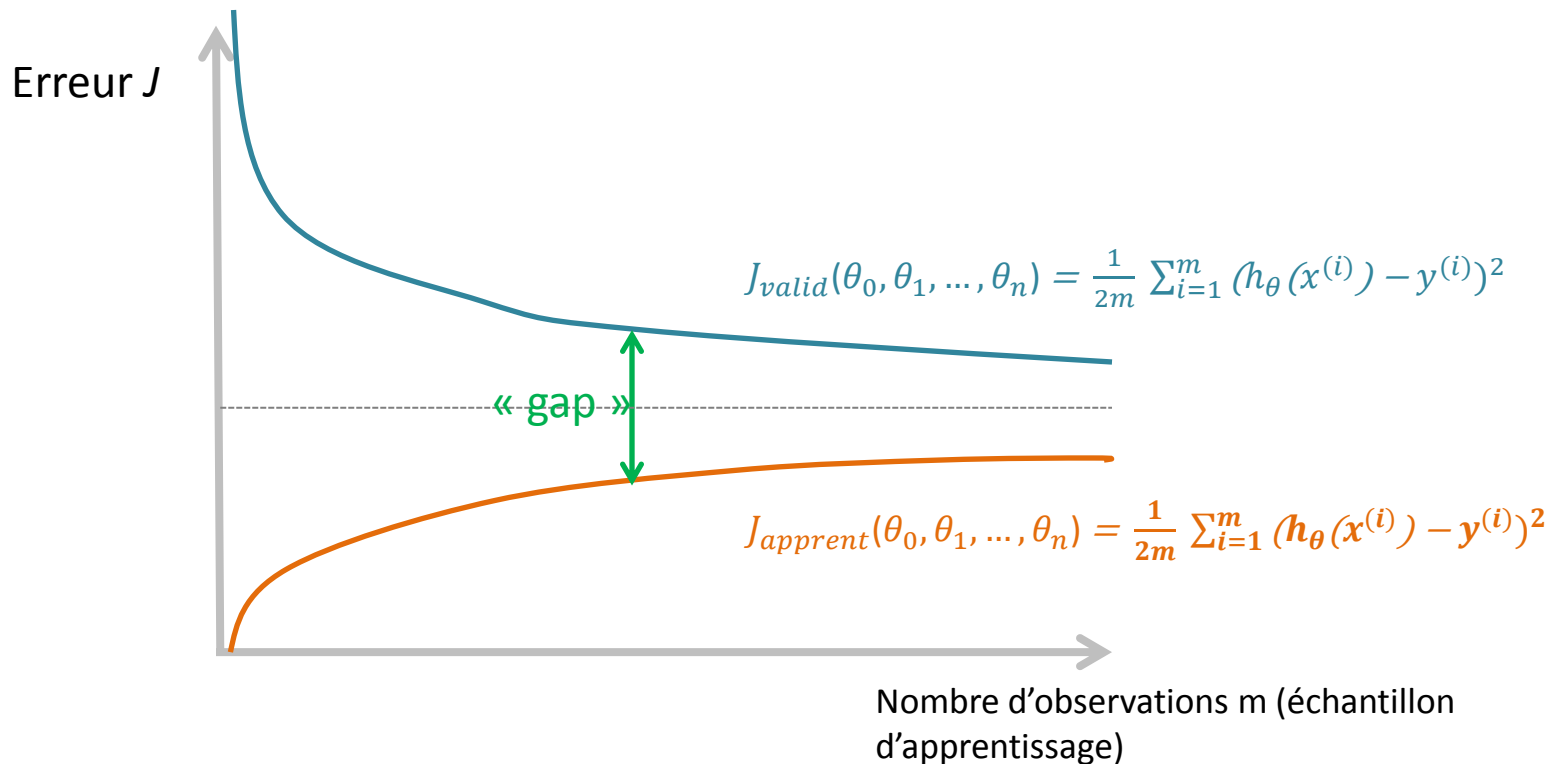


L'ajout d'observations ne permet pas l'amélioration du modèle

\* L'erreur va rester identique, même si le nombre d'observations augmente !

# Learning curves : diagnostiquer une variance élevée

Le modèle souffre -il d'une variance élevée ?



L'ajout d'observations permet l'amélioration du modèle !

\* Le « gap » va diminuer si on a plus d'observations



Thank you