Descente de Gradient

Pascal Poncelet
LIRMM
Pascal.Poncelet@lirmm.fr
http://www.lirmm.fr/~poncelet

Qu’est ce que c’est ?

Non connecté Discussion Contributions Créer un compte Se connecter

Article Discussion Lire Modifier Modifier le code Voir l'historique Rechercher dans Wikipédia

WIKIPÉDIA
L'encyclopédie libre

Accueil
Portails thématiques
Article au hasard
Contact

Contribuer
Débuter sur Wikipédia
Aide
Communauté
Modifications récentes
Faire un don

Outils

Wiki Loves Monuments : photographiez un monument historique, aidez Wikipédia et gagnez !
En apprendre plus

Algorithme du gradient

L'**algorithme du gradient** désigne un algorithme d'optimisation différentiable. Il est par conséquent destiné à minimiser une fonction réelle différentiable définie sur un espace euclidien (par exemple, $\mathbb{R}^n$, l'espace des n-uplets de nombres réels, muni d'un produit scalaire) ou, plus généralement, sur un espace hilbertien. L'algorithme est itératif et procède donc par améliorations successives. Au point courant, un déplacement est effectué dans la direction *opposée* au gradient, de manière à faire décroître la fonction. Le déplacement le long de cette direction est déterminé par la technique numérique connue sous le nom de recherche linéaire. Cette description montre que l'algorithme fait partie de la famille des algorithmes à directions de descente.

Les algorithmes d'optimisation sont généralement écrits pour minimiser une fonction. Si l'on désire maximiser une fonction, il suffira de minimiser son opposée.

Il est important de garder à l'esprit le fait que le gradient, et donc la direction de déplacement, dépend du produit scalaire qui équipe l'espace hilbertien ; l'efficacité

Tout simplement

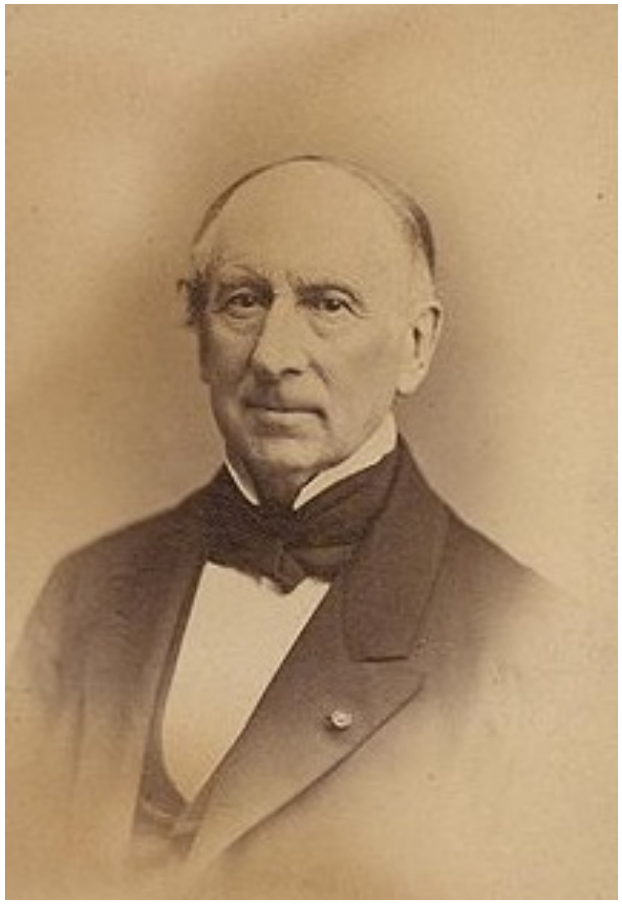
- La descente de gradient (*Gradient Descent*) : algorithme d'optimisation pour chercher le minimum d'une fonction différentiable (dont on peut calculer la dérivée !) définie sur un espace Euclidien (par exemple $\mathbb{R}^n$)

- Algorithme de Newton, Algorithmes de quasi-Newton, Algorithme de Gauss-Newton, promenade aléatoire, recuit simulé, algorithmes génétiques

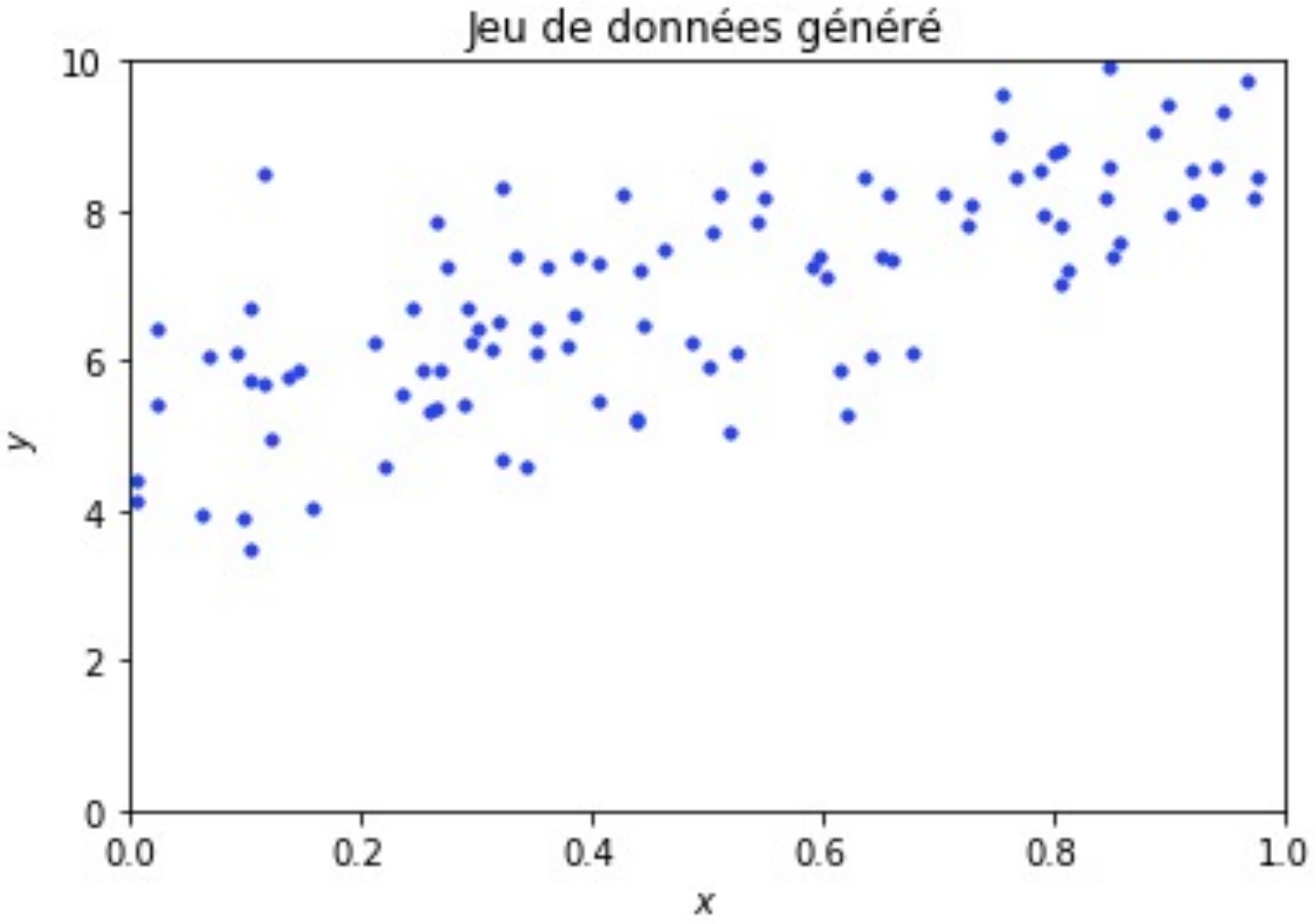
Tout simplement

- Inventé en 1847 par Louis Augustin Cauchy

- Grand intérêt : Deep learning
  - Beaucoup plus en fait car de nombreux algorithmes de ML utilisent la descente de gradient

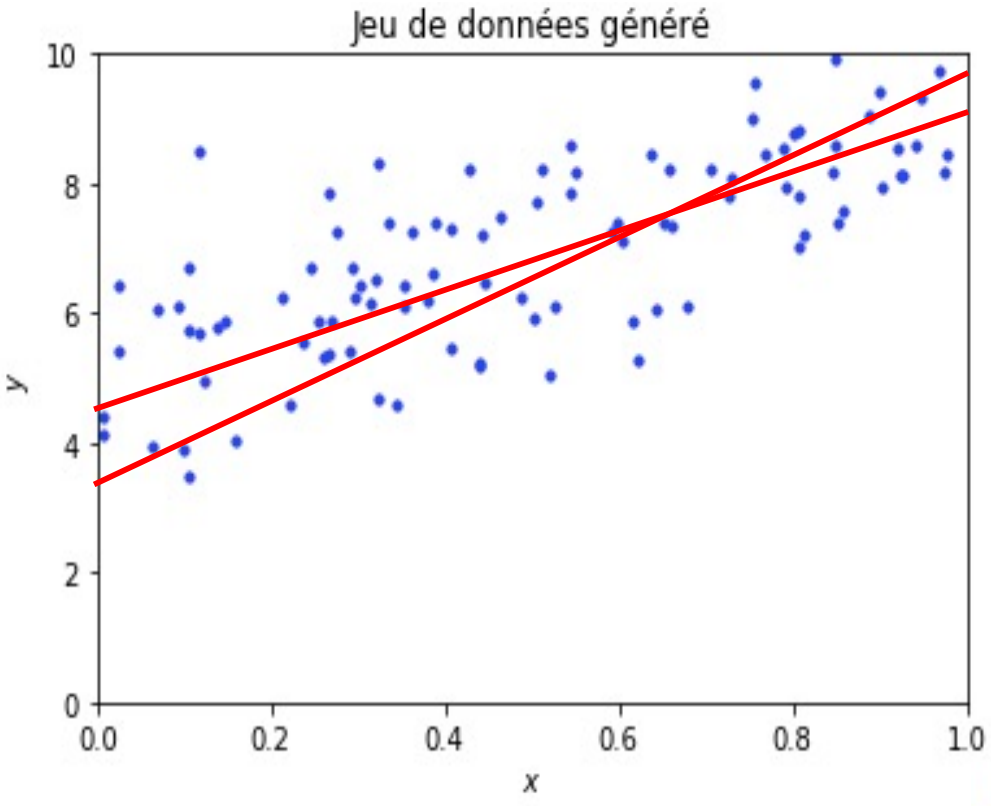


Un problème de régression linéaire



A partir de y=4x+5

Un problème de régression linéaire

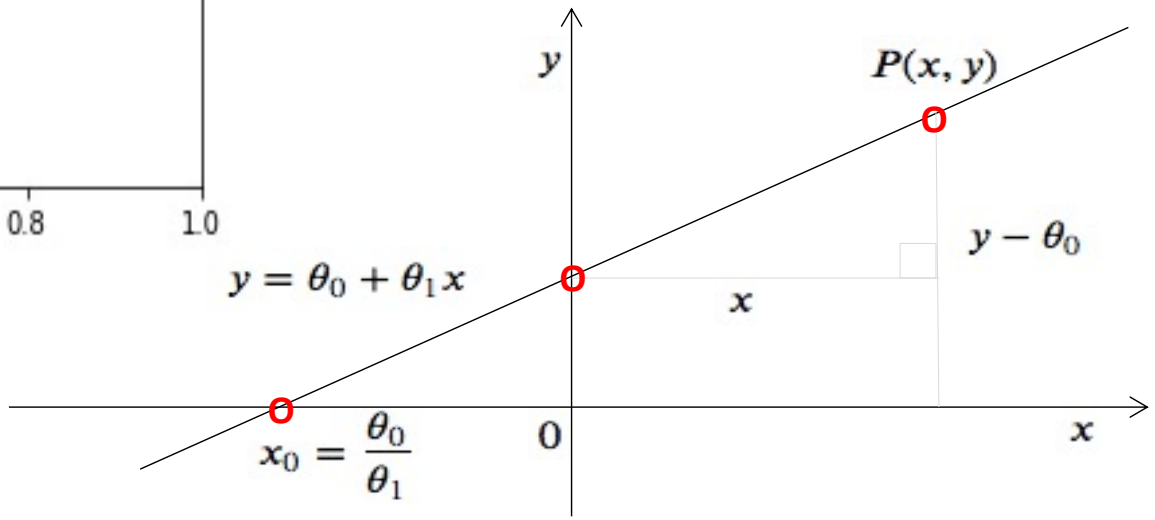


?

Trouver l'hypothèse

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

pour estimer *y* en fonction de *x*



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

```
1  def predict(X, theta0, theta1):
2      return theta0 + theta1*X
3
4  print ('Valeur de X[5] : ',X[5],' - valeur réelle de y[5] : ',y[5])
5  theta0=2
6  theta1=1
7  print ('valeur prédite : ',predict(X[5], theta0, theta1))
```

Avec $\theta_0 = 2$ et $\theta_1 = 1$

Valeurs réelles :

```
Valeur de X[5] :  [0.0253381]  - valeur réelle de y[5] :  [6.41880389]
```

```
valeur prédite :  [2.0253381]
```

Erreur !

Fonction de coût/perte

- Evaluation des erreurs de prédiction
- La fonction de coût (*Cost Function*) calcule l'erreur pour une seule occurrence d’apprentissage
- La fonction de perte (*Loss Function*) : moyenne des fonctions de coûts pour tous les exemples d’apprentissage
- Les deux termes sont utilisés

MSE

- Erreur quadratique moyenne (*Mean square error (MSE)*)

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

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

$$= \frac{1}{m}\sum_{i=1}^{m}((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2$$

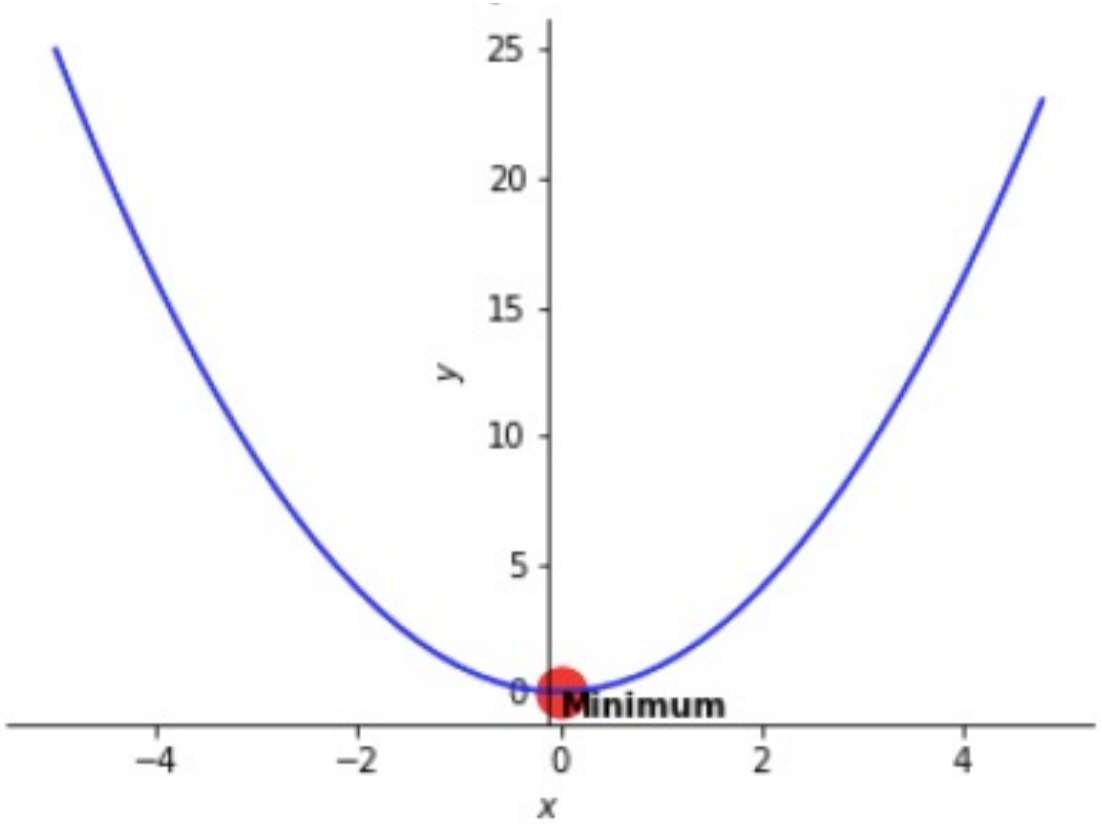
- différences quadratiques préférées aux différences absolues car permettent de trouver une ligne de régression en calculant la dérivée première
- le carré augmente la distance d'erreur ce qui met en évidence les mauvaises prédictions

Comment minimiser *J* ?

En fait :

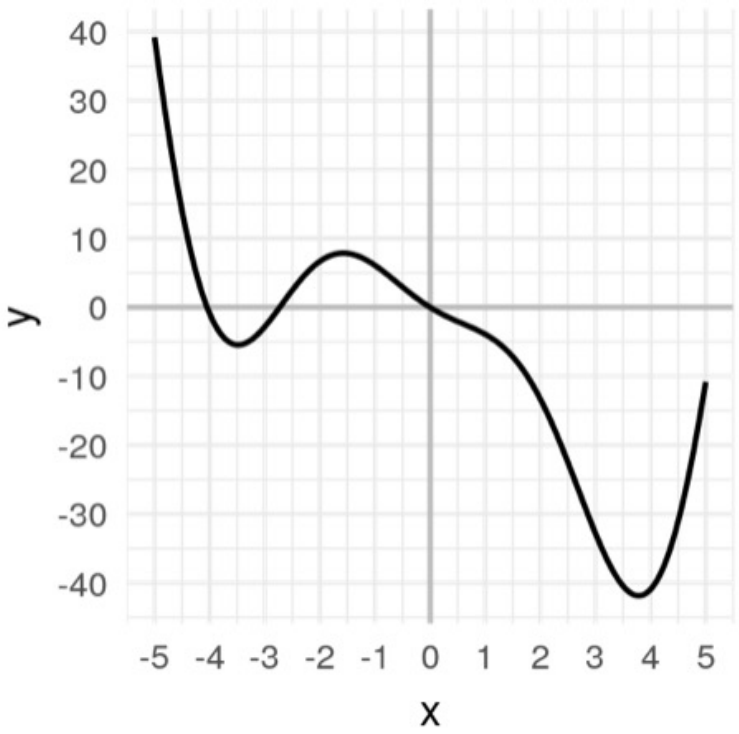
$$\frac{1}{m}\sum_{i=1}^{m}((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2$$

$$y = x^2$$

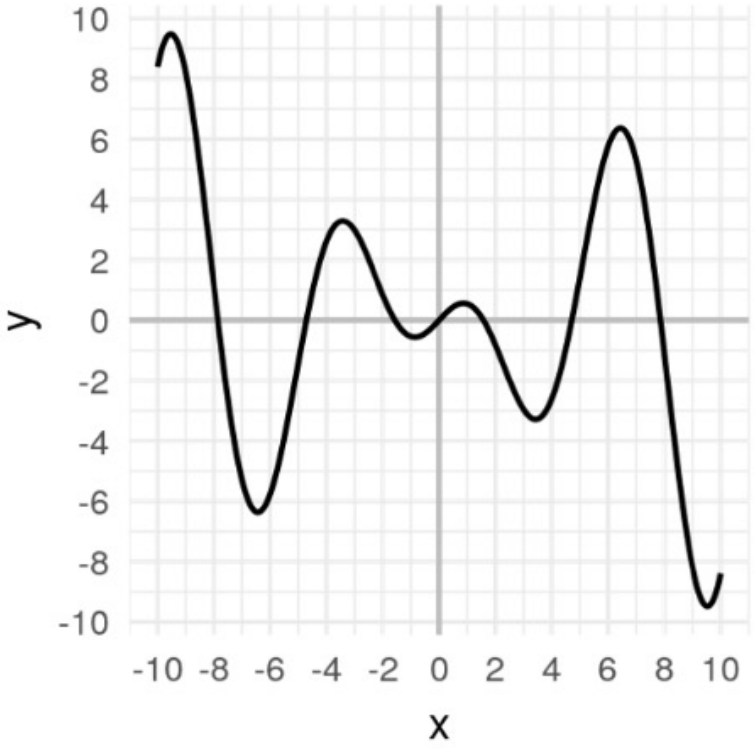


Les fonctions ne sont pas toujours convexes !

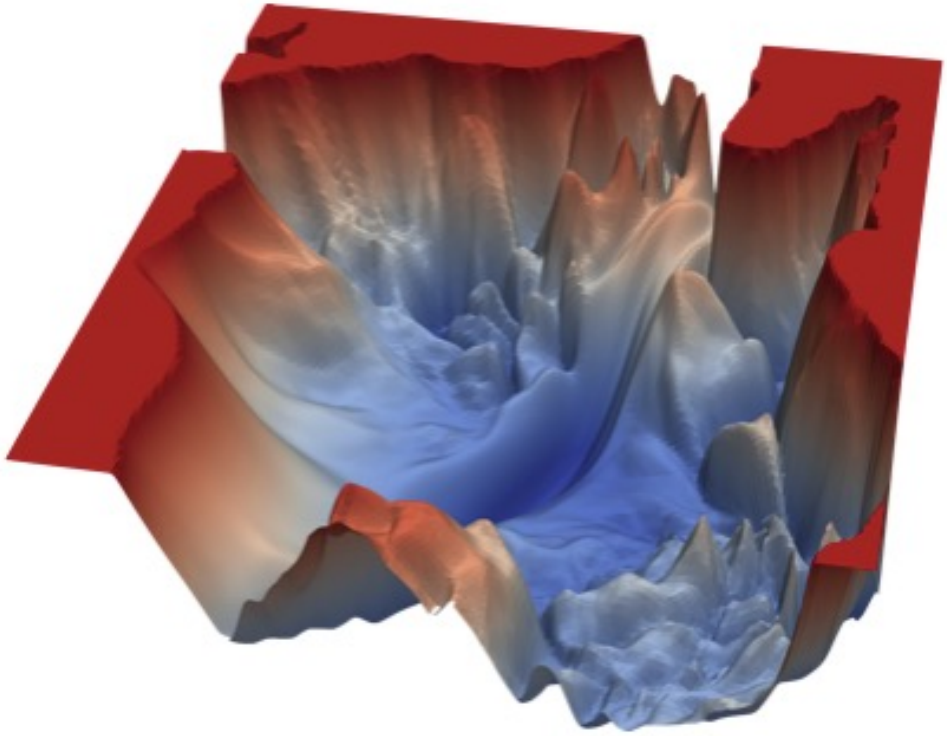
$f(x) = 2x^2 \cos(x) - 5x$

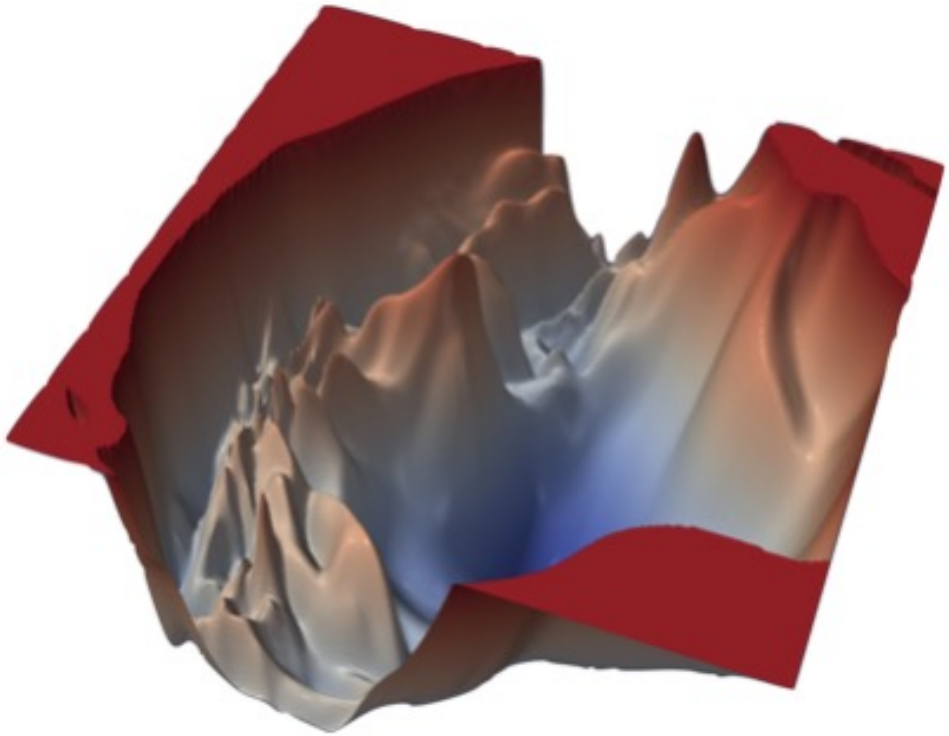


$f(x) = x \cos(x)$



Les fonctions ne sont pas toujours convexes !





Sources : https://www.cs.umd.edu/~tomg/projects/landscapes/

Algorithme de descente de gradient

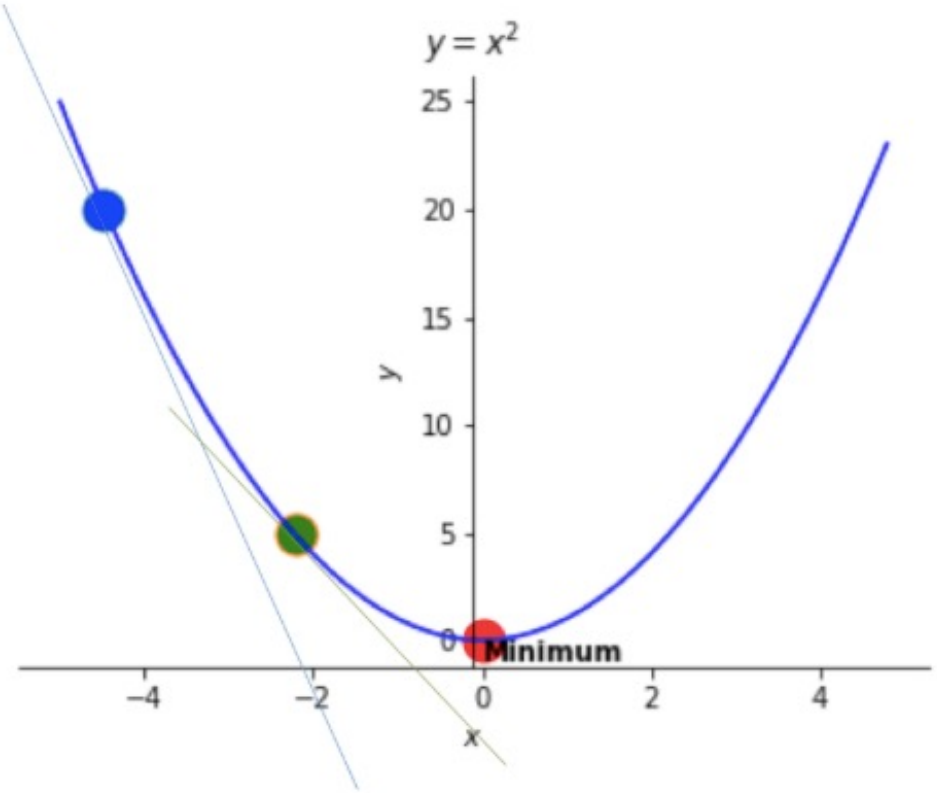
Altitude : $J(\theta_0, \theta_1)$

Algorithme de descente de gradient

- Pour un point donné, la valeur du gradient (**la dérivée**) correspond à l'inclinaison de la pente en ce point
- Pente point vert (x=-2.3) plus faible que point bleu (x=-4.5)

$$f'(2 \times -4.5) = -9$$
$$f'(2 \times -2.3) = -4.6$$

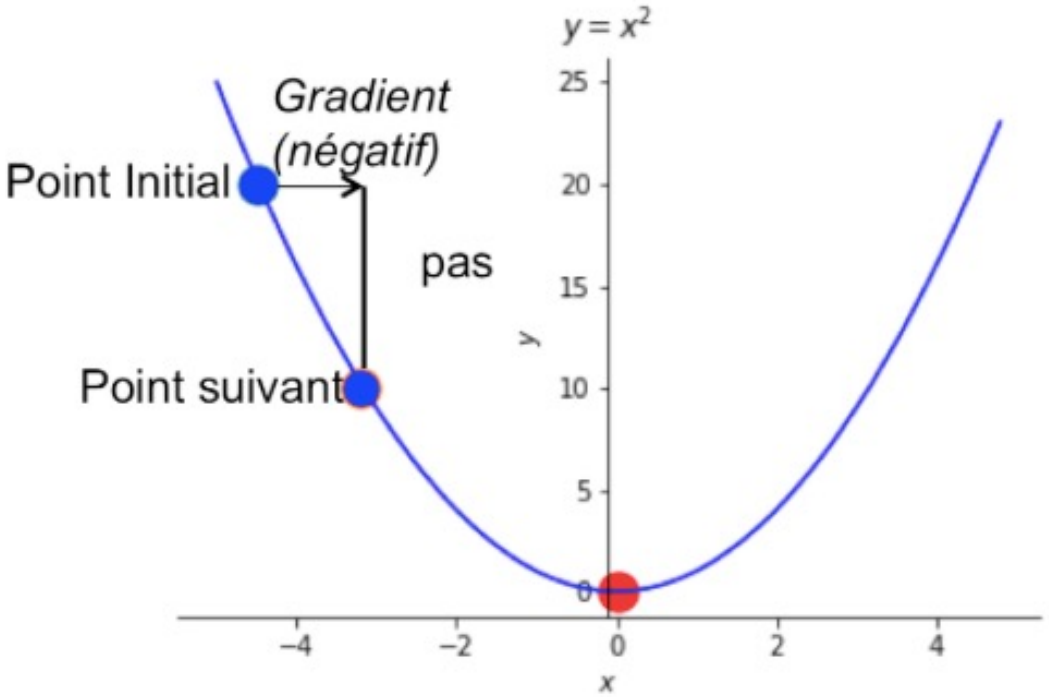
$$f' = 2x$$



Gradient négatif fort -> continuer à descendre dans cette direction

Algorithme de descente de gradient

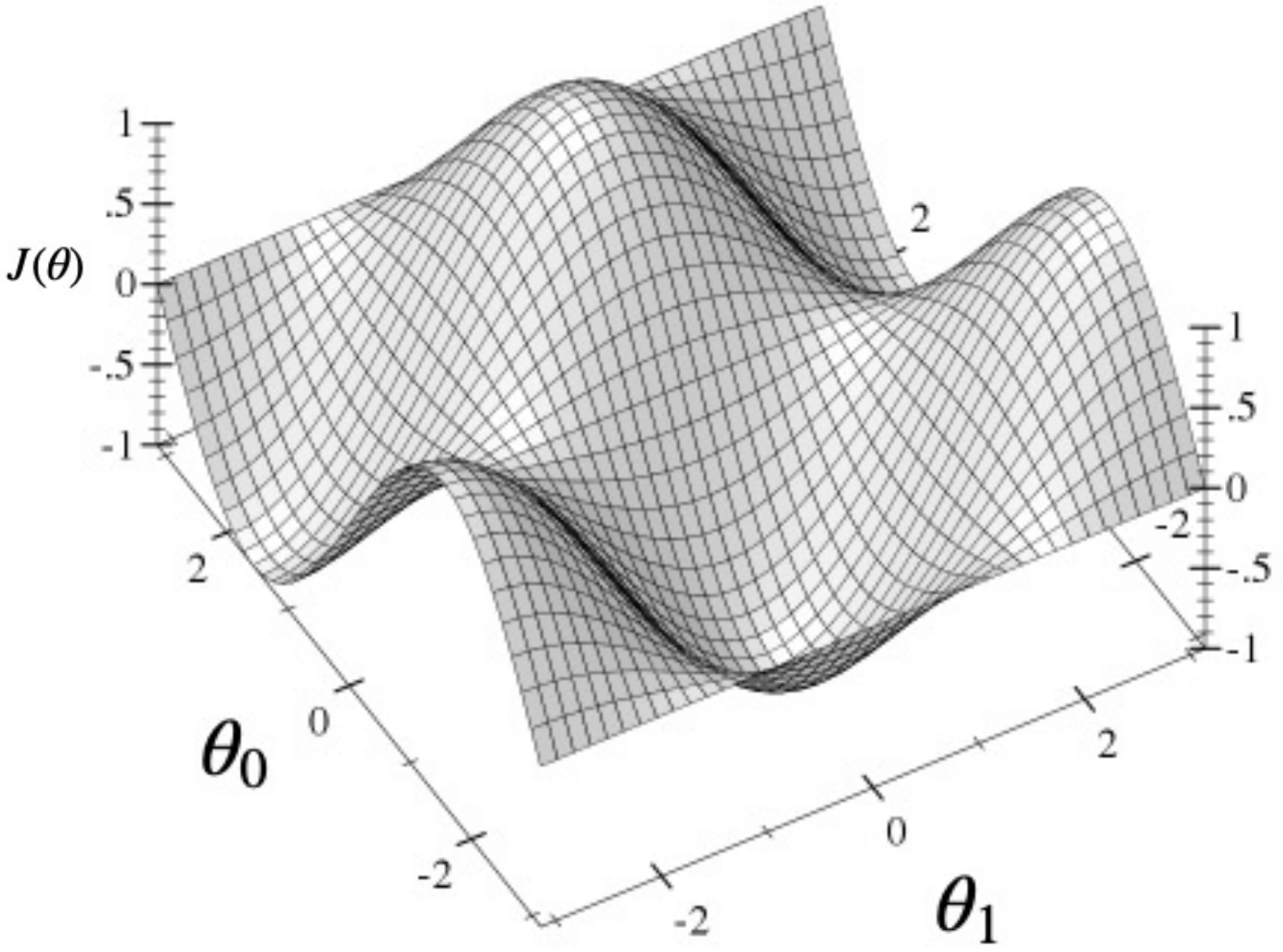
- Pas = taux d'apprentissage (*learning rate*) η
- Plus le pas est petit plus il faut de temps
- Valeurs comprises entre 0.0001 et 1



Tester la valeur du pas

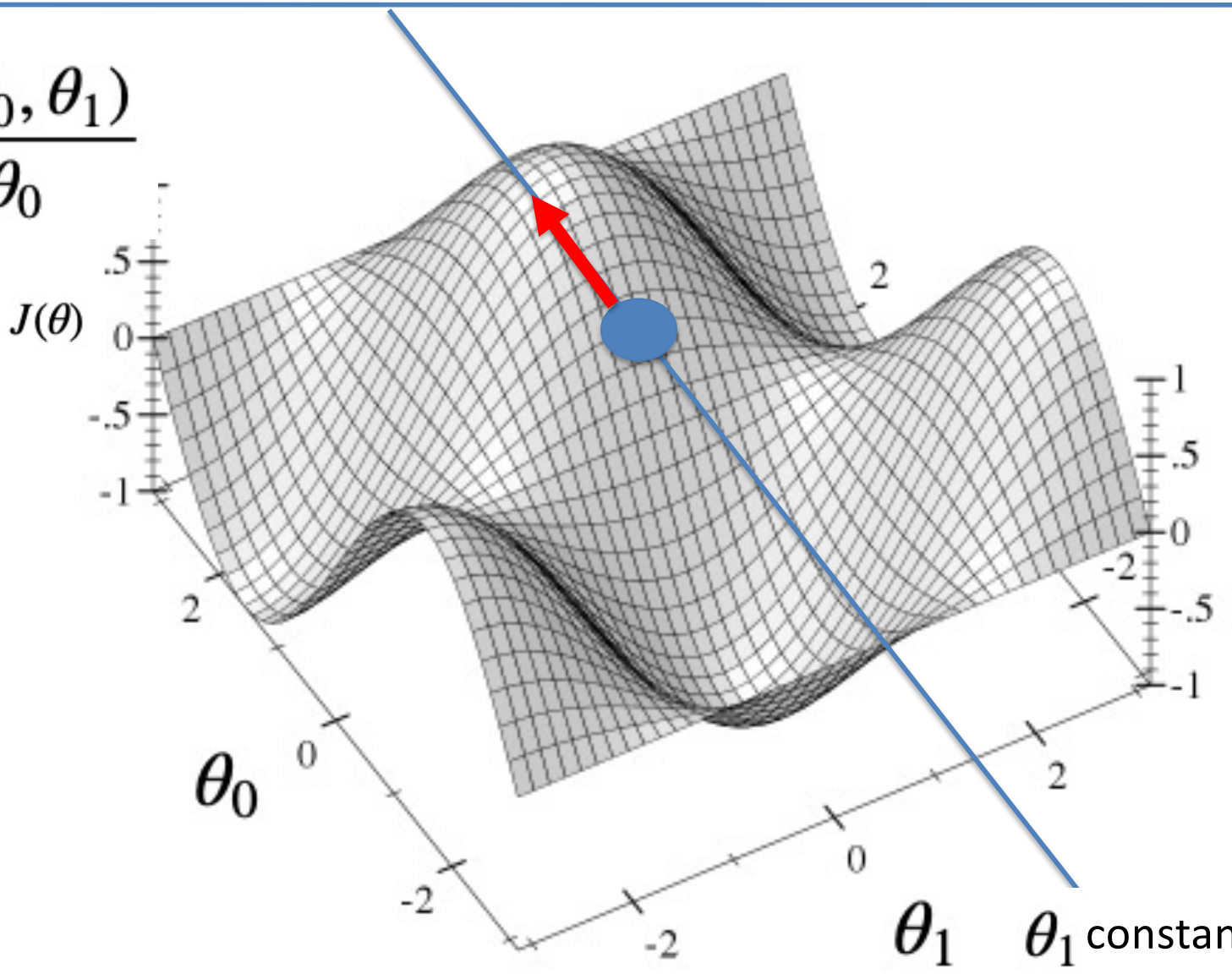
https://developers.google.com/machine-learning/crash-course/fitter/graph.

Rappel sur les dérivées partielles



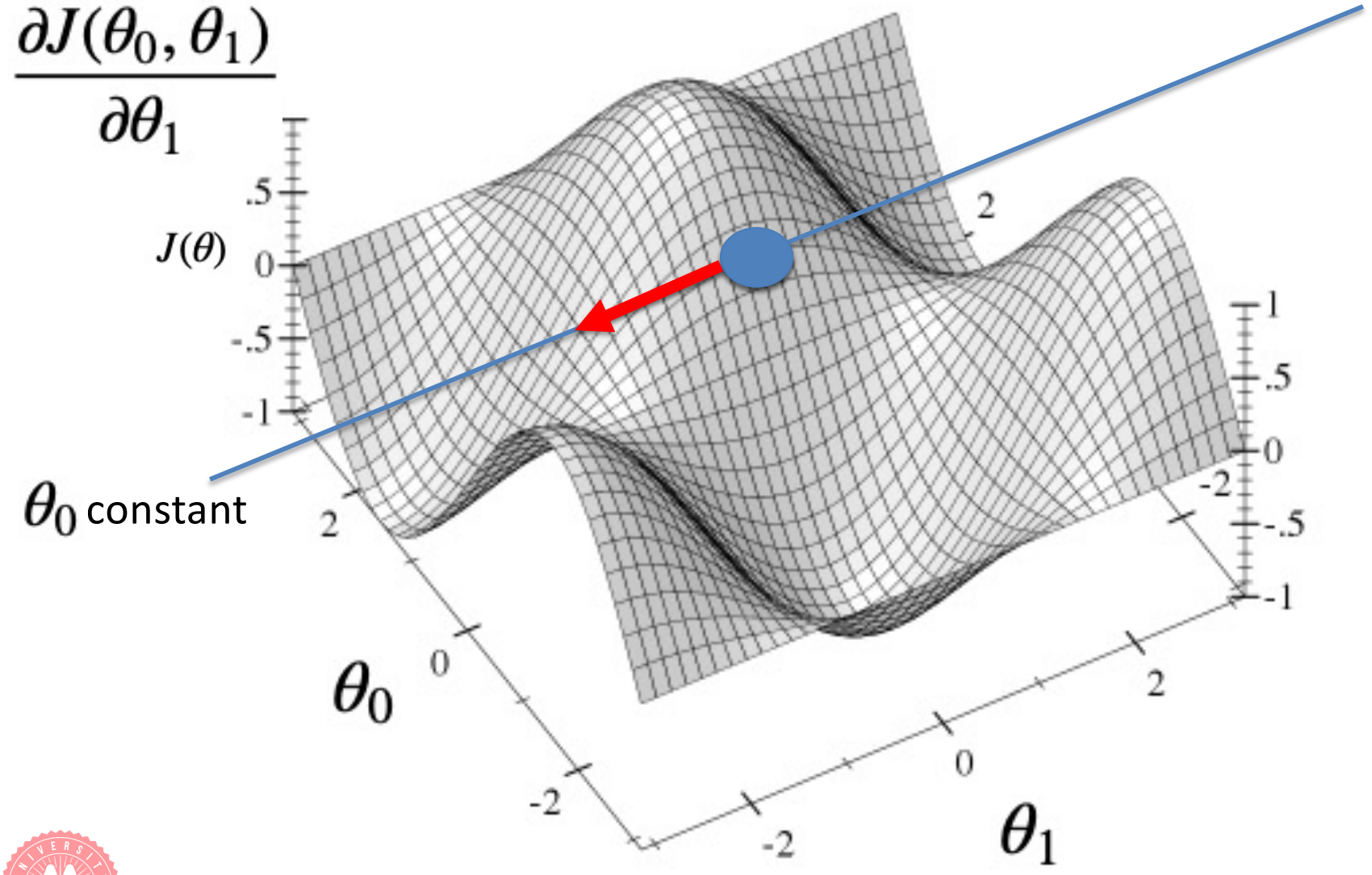
Rappel sur les dérivées partielles

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0}$$



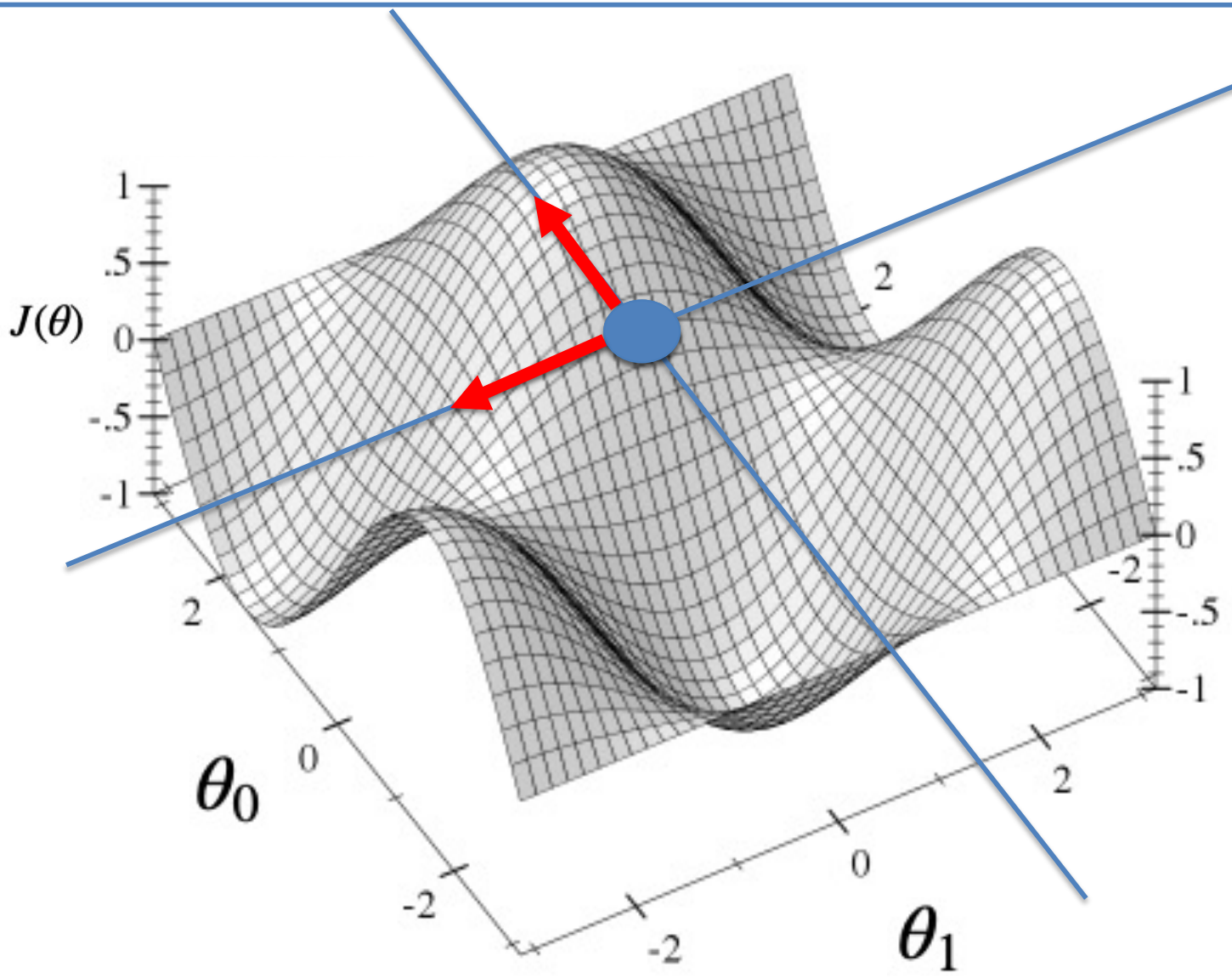
Rappel sur les dérivées partielles

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1}$$

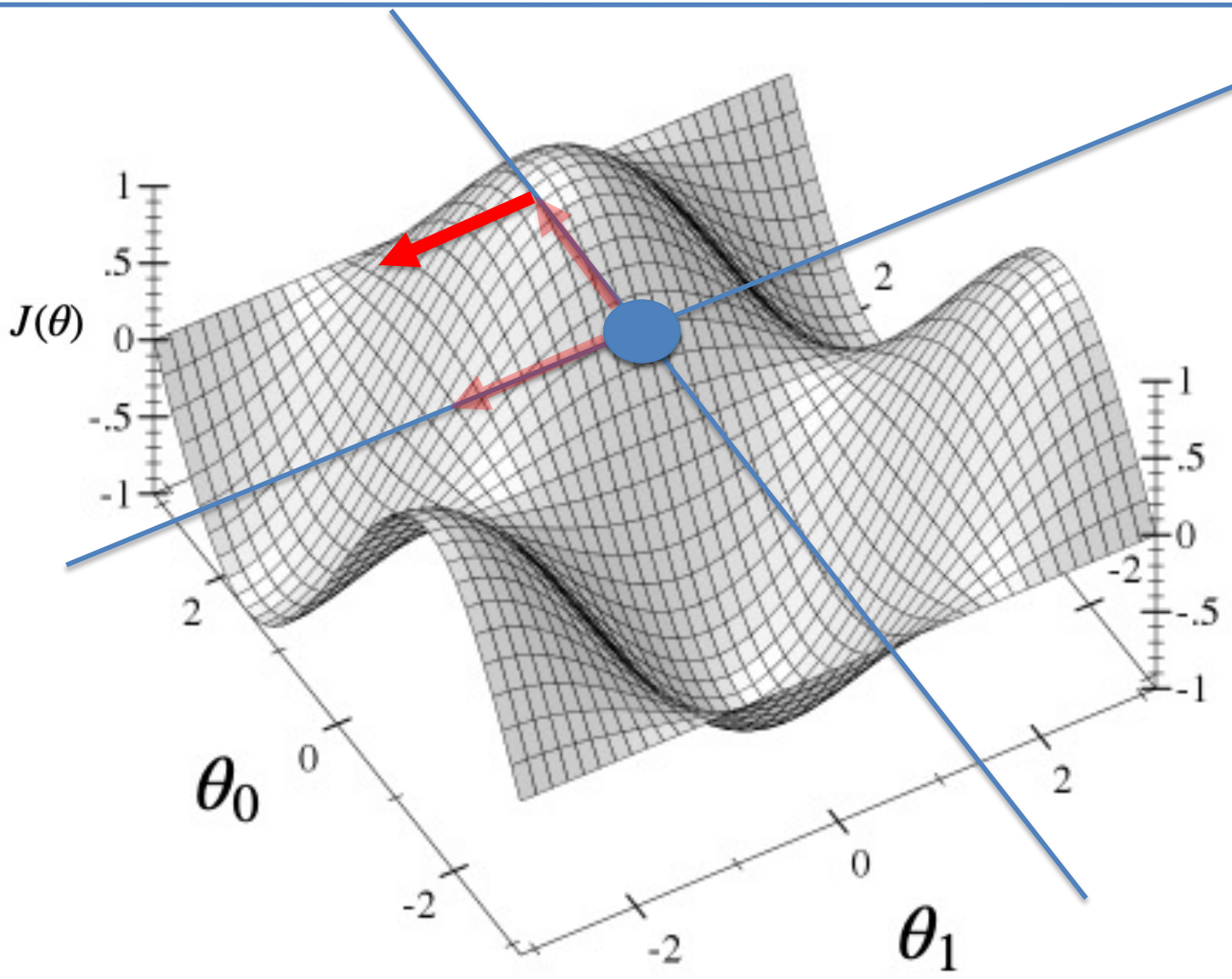


θ_0 constant

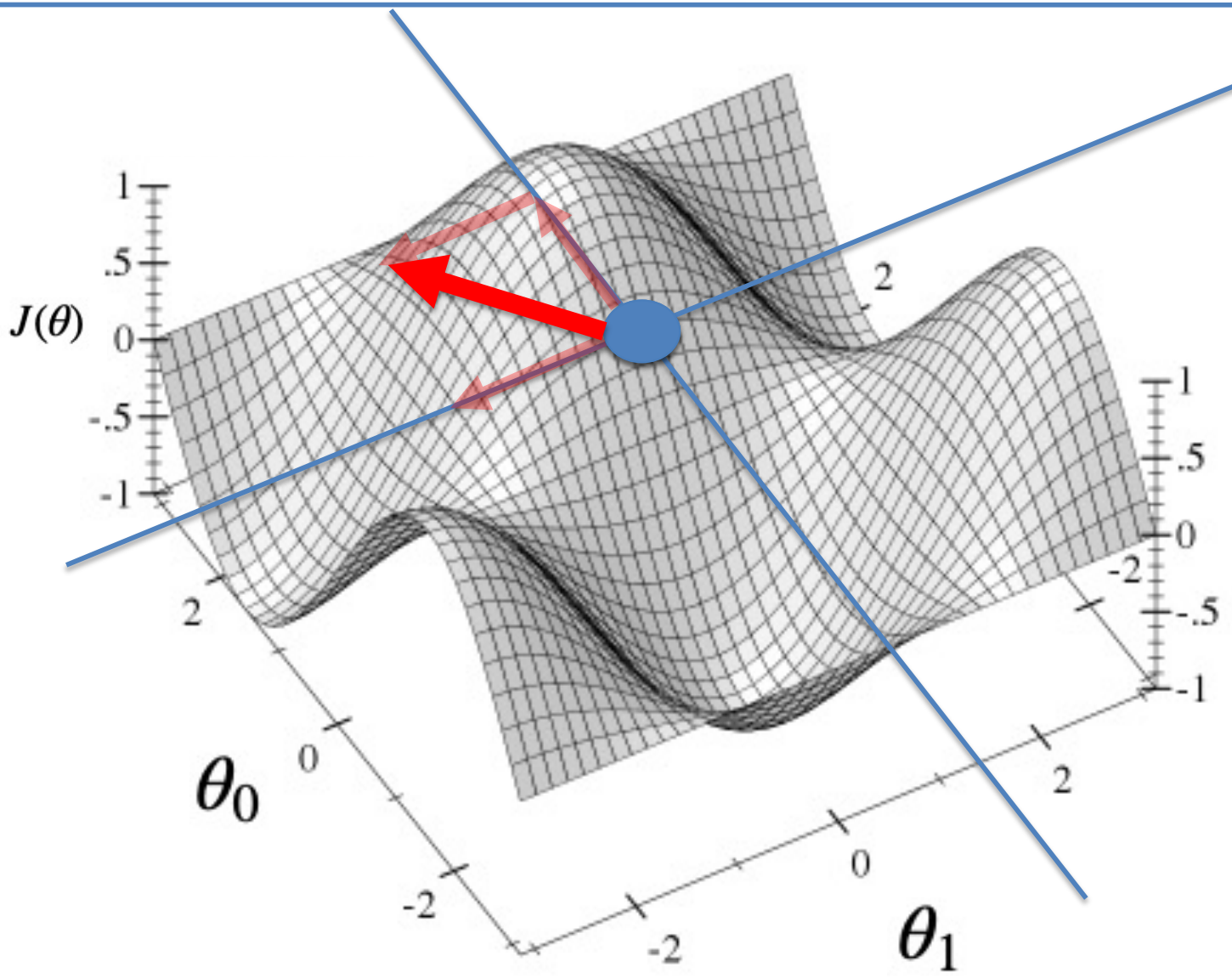
Rappel sur les dérivées partielles



Rappel sur les dérivées partielles



Rappel sur les dérivées partielles



Dérivées partielles

A partir de : $J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i-1}^{m} ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2$

Les dérivées partielles de θ_0 et θ_1 :

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = \frac{-2}{m} \sum_{i=1}^{m} ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})$$

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1} = \frac{-2}{m} \sum_{i=1}^{m} ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}) x^{(i)}$$

L’algotithme

- Initialisation de θ_0 et θ_1 avec des nombres aléatoires
- Répéter jusqu’à convergence {

$$\theta_0 = \theta_0 - \eta \frac{-2}{m} \sum_{i=1}^{m} ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})$$

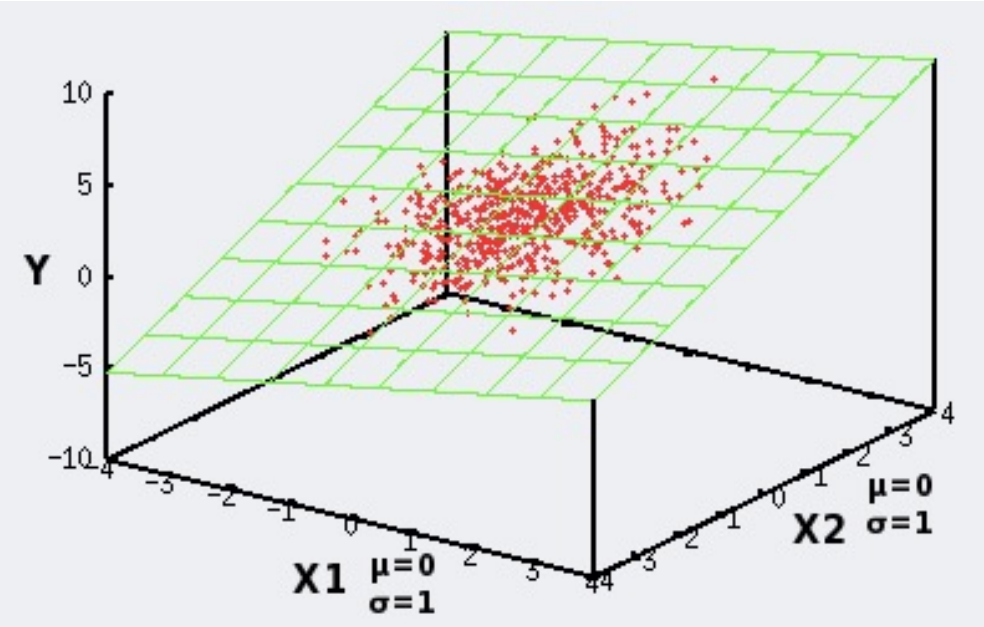
$$\theta_1 = \theta_1 - \eta \frac{-2}{m} \sum_{i=1}^{m} ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}) x^{(i)}$$

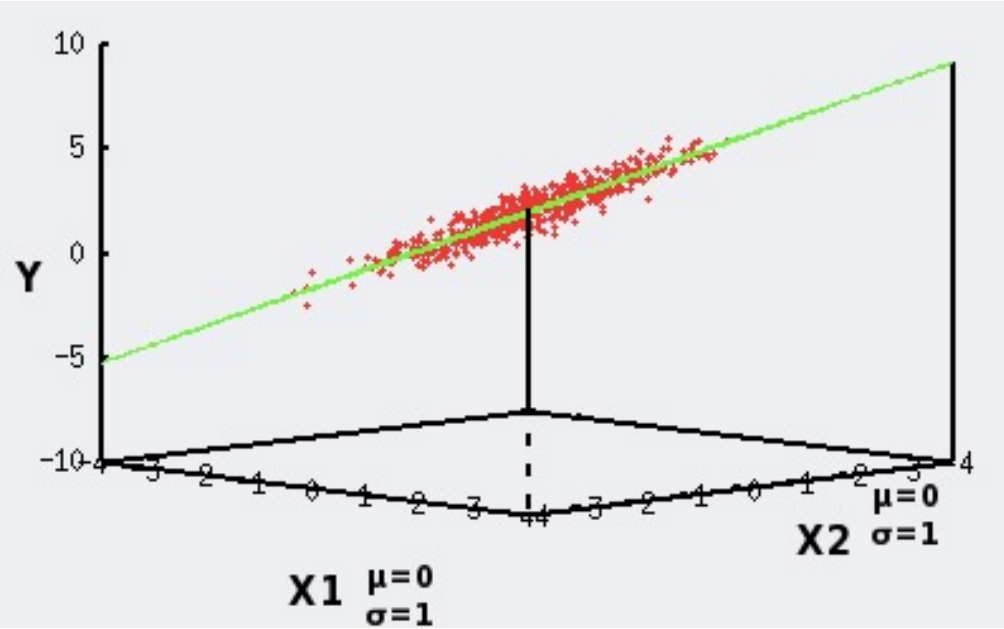
}

Essai

- Notebook régression linéaire

Régression linéaire multiple





Exemple : concentration d'ozone (y)
dépend de la vitesse du vent (x1) et de la température (x2)

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$$

Utilisation de matrices

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$$

$$X = \begin{bmatrix} x_1^1 & x_2^1 & \ldots & x_n^1 \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ x_1^3 & x_2^3 & \ldots & x_n^3 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^m & x_2^m & \ldots & x_n^m \end{bmatrix} (taille : m \times n)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_n \end{bmatrix} (taille : n+1 \times 1)$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{bmatrix} (taille : m \times 1)$$

Utilisation de matrices

Ajout une colonne de 1 à X

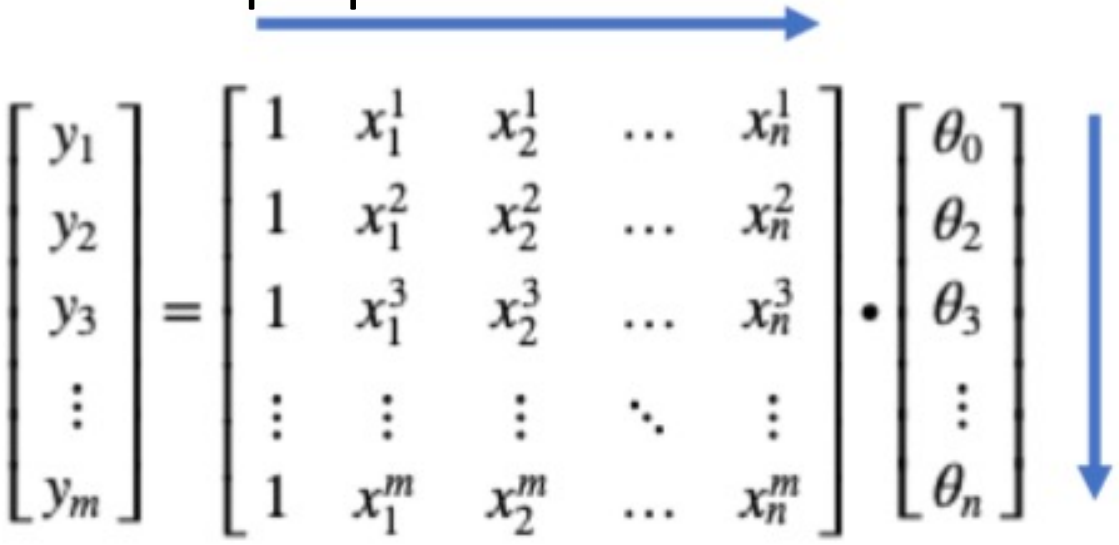
$$X = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \dots & x_n^1 \\ 1 & x_1^2 & x_2^2 & \dots & x_n^2 \\ 1 & x_1^3 & x_2^3 & \dots & x_n^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^m & x_2^m & \dots & x_n^m \end{bmatrix} (taille : m \times n + 1)$$

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$$
$$= \theta^T X$$

Où θ^T correspond à la transposée de la matrice θ

Prédiction

Prédiction = simple produit scalaire de matrice



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \theta_0 & \theta_1 x_1^1 & x_2^1 & \dots & \theta_n x_n^1 \\ \theta_0 & \theta_1 x_1^2 & x_2^2 & \dots & \theta_n x_n^2 \\ \theta_0 & \theta_1 x_1^3 & x_2^3 & \dots & \theta_n x_n^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_0 & \theta_1 x_1^m & x_2^m & \dots & \theta_n x_n^m \end{bmatrix}$$

Les dérivées partielles

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

L’algorithme

- Initialisation de θ_j avec des nombres aléatoires
- Répéter jusqu’à convergence {

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

en mettant θ_j à jour avec j=(1..n)

}

Ce qui donne

$$\theta_0 = \theta_0 - \eta \frac{-2}{m} \sum_{i=1}^{m} (h_\theta x^{(i)}) - y^{(i)}) x_0^{(i)} \; avec x_0^{(i)} = 1$$

$$\theta_1 = \theta_1 - \eta \frac{-2}{m} \sum_{i=1}^{m} (h_\theta x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 = \theta_2 - \eta \frac{-2}{m} \sum_{i=1}^{m} (h_\theta x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$$\vdots$$

L’algorithme simplifié

- Initialisation de θ_j avec des nombres aléatoires
- Répéter jusqu’à convergence {

$$\theta = \theta - \eta \frac{-2}{m}(X \bullet \theta - y)X^T$$

}

Essai

- Notebook Régression linéaire multiple

Optimisations

- Algorithme de descente de gradient « par lot » (batch)
  - Calcul de l'erreur pour tous les exemples d'apprentissage
  - Evaluation de l'ensemble complet d'apprentissage (*1 epoch*) et mise à jour des paramètres

+ facile, efficace en calcul, atteinte d'un état stable

- état stable = optimal ?, ensemble d'apprentissage en mémoire

Stochastic gradient descent

- - tirage aléatoire d'un exemple (batch_size=1) (éventuellement d'un sous ensemble) et descente de gradient pour l'échantillon

+ beaucoup moins de mémoire, converge, plus rapide, aide à sortir des minima locaux (beaucoup de mise à jour)

- convergence plus longue (descente dans des directions différentes), mises à jour fréquentes couteuses en temps de calcul pour 1 exemple, quid des matrices ?

Mini-batch gradient descent

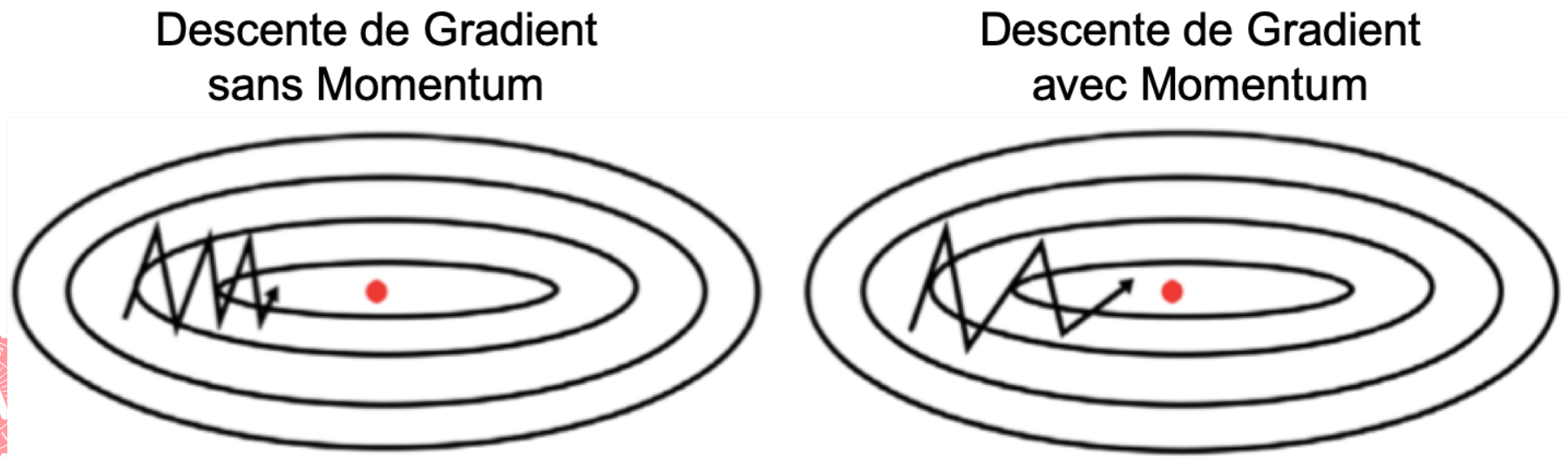
- Mixte de Batch Gradient Descent et Stochastic Gradient Descent

- Ensemble d’apprentissage divisé en plusieurs groupes appelés lots. Chaque lot contient un certain nombre d'échantillons d'apprentissage. A chaque epoch 1 seul lot est traité. Le plus utilisé en deep learning

+ Tient en mémoire, efficace en terme de calcul, matrice, sortie des minima locaux, convergence

Momentum

- Utilisé lorsque la surface s'incline beaucoup plus fortement dans une dimension que dans une autre
- La descente de gradient oscille sur les pentes et ne progresse que doucement vers le minimum local
- Accélérer la descente de gradient dans la direction voulue et d'atténuer les oscillations



Momentum

- Balle qui descend : accumulation de l'élan (*Momentum*) dans la descente et devient de plus en plus rapide jusqu'à ce qu'elle atteigne sa vitesse limite s'il existe une résistance de l'air ->
- quantité de mouvement augmente pour les dimensions dont les gradients pointent dans la même direction et réduit les mises à jour pour les dimensions dont les gradients changent de direction
- Introduction de deux paramètres : la vitesse v que nous essayons d'optimiser et le frottement (β)

$$v = \beta v - \eta \Delta J$$
$$\theta = \theta - v$$

$$v = \beta v - \eta \times \frac{-2}{m} \times (X \bullet \theta - y) X^T$$
$$\theta = \theta - v$$

+ accélération de l'apprentissage

D’autres optimisations

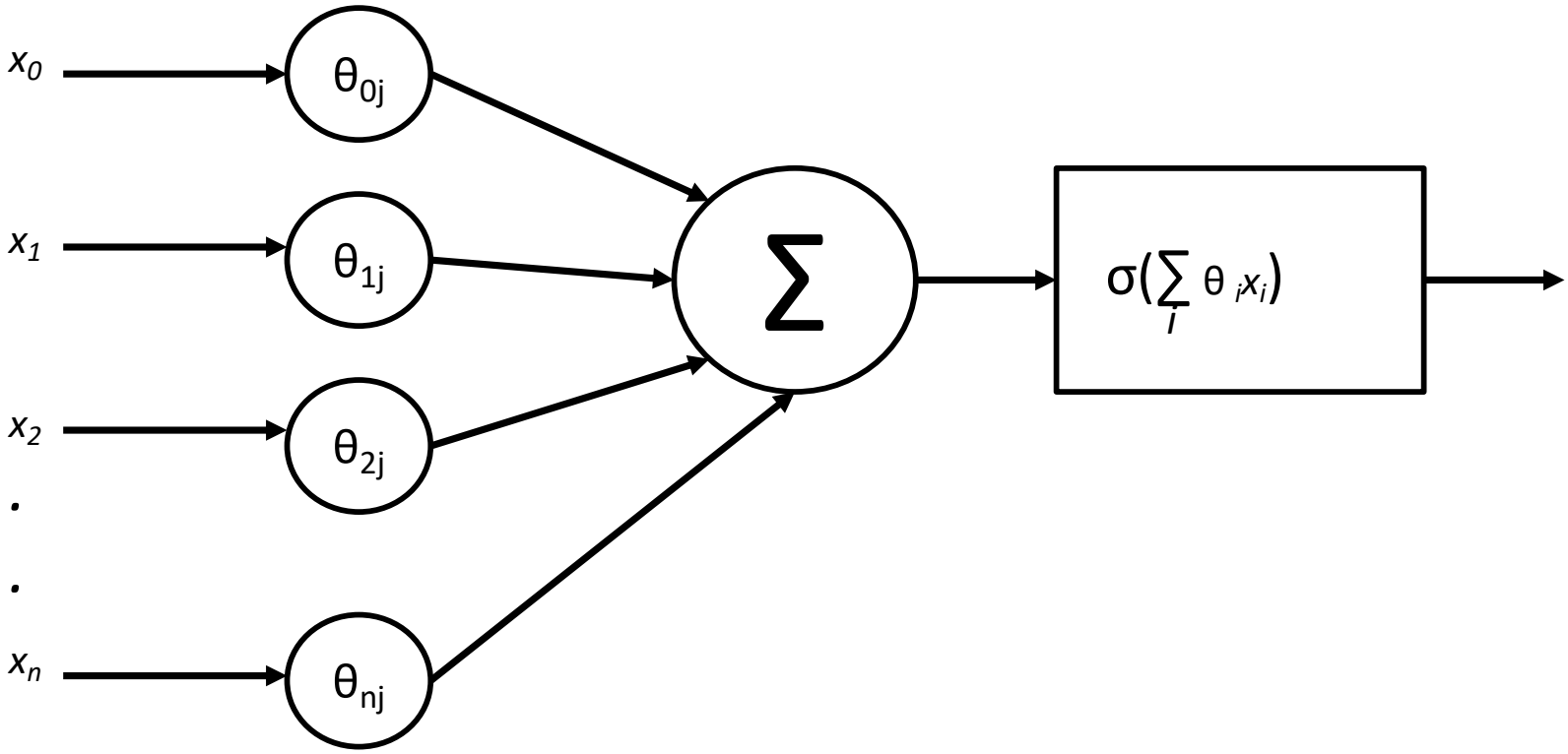
- *Nesterov Momentum* assez proche de Momentum mais qui, au lieu de calculer le gradient à la position actuelle, le calcul à une nouvelle position approximé
- *Adagrad* s'intéresse au learning rate. Précédemment η a toujours été considéré constant. L'objectif d'Adagrad est de s'adapter pour pouvoir accélérer ou ralentir
- *RMSprop* est une extension d'Adagrad pour corriger le fait que ce dernier considère une fonction d'accumulation du gradient qui est croissante monotone. Ceci peut cependant poser des problèmes car le taux d'apprentissage diminue également de façon monotone et l'apprentissage peut s'arrêter car ηdevient trop petit
- *ADAM* cumule les avantages de Momentum et de RMSprop

Régression Logistique

- modèle logit : cas particulier de modèle linéaire généralisé. Particulièrement utilisé comme approche supervisée
- Régression logistique binaire :
  - un ensemble d'entrée X
  - classer dans une catégorie possible 0 ou 1

Longueur des Sépales
Largeur des Sépales
→
Setosa ?
Versicolor ?

Régression Logistique

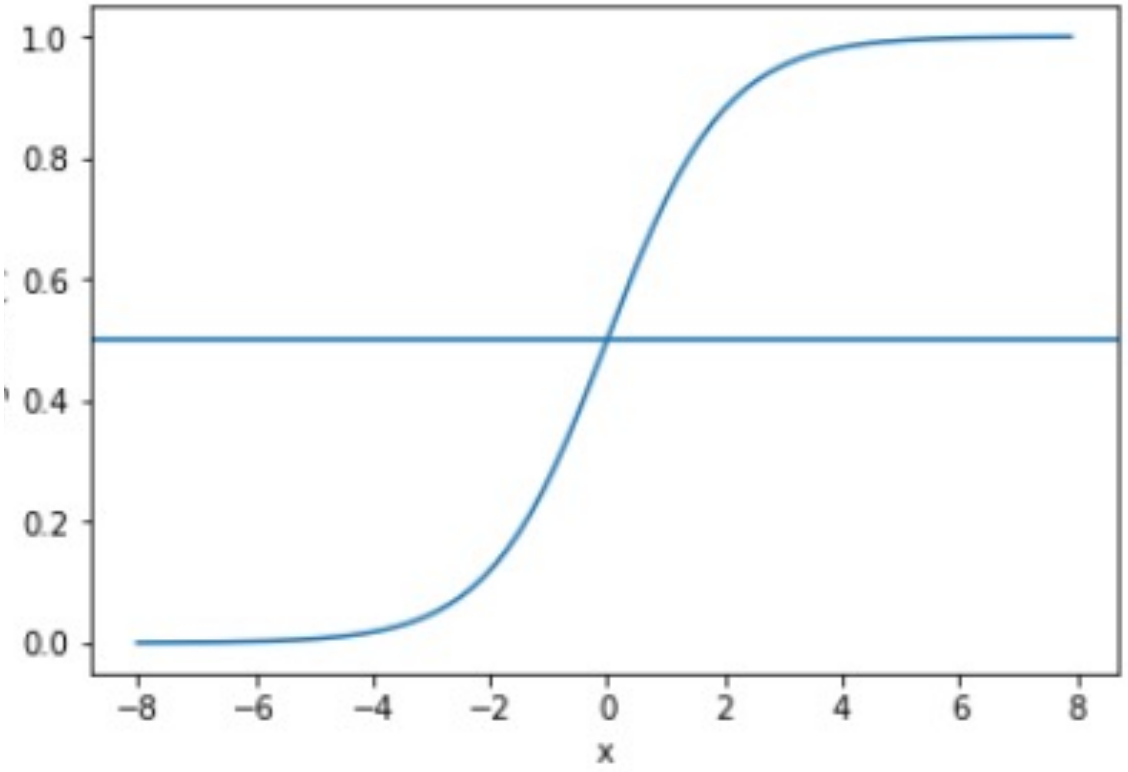


$$h_\theta(x) = \theta^T x \qquad h_\theta(x) = \sigma(\theta^T x)$$

Utilisation de la sigmoid

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

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

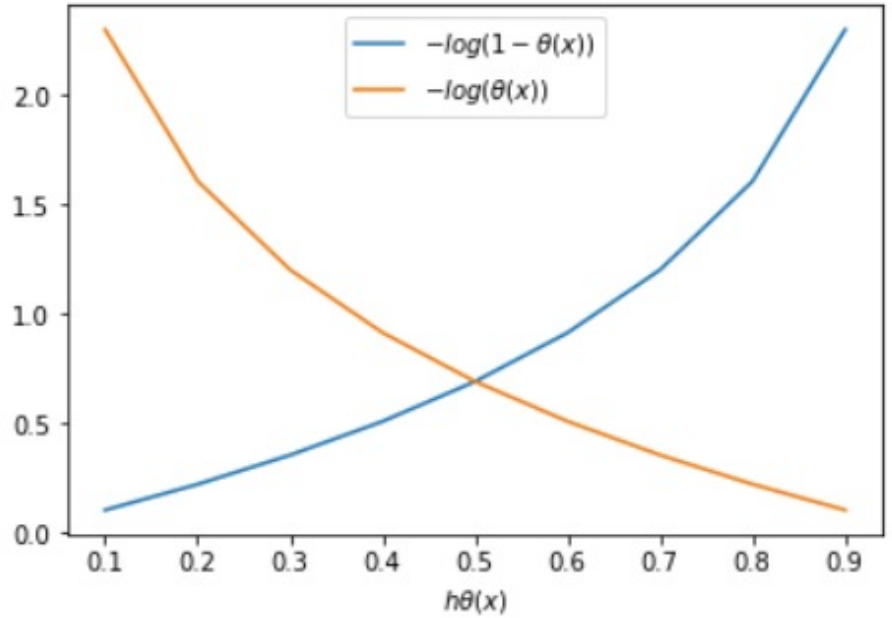


$$h_\theta(x) = \begin{cases} > 0.5, & si\ \theta^T x > 0 \\ < 0.5, & si\ \theta^T x < 0 \end{cases}$$

si la somme pondérée des entrées est supérieure à zéro, la classe prédite est 1 et inversement si elle est inférieure à 0, la classe prédite est 0.

Fonction de coût

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



Si la classe réelle est 1 et que le modèle prédit 0,
il faut fortement le pénaliser et vice-versa

Pour $-log(h_\theta(x))$ *lorsque* $h_\theta(x)$ *est proche de 0,*
le coût est égal à l'infini (le modèle est fortement pénalisé)

La fonction de coût

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

Et la dérivée partielle :

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

Algorithme de descente de gradient :

$$\theta_j = \theta_j - \eta \frac{1}{m} X^T (\sigma(X.\theta) - y)$$

Essai

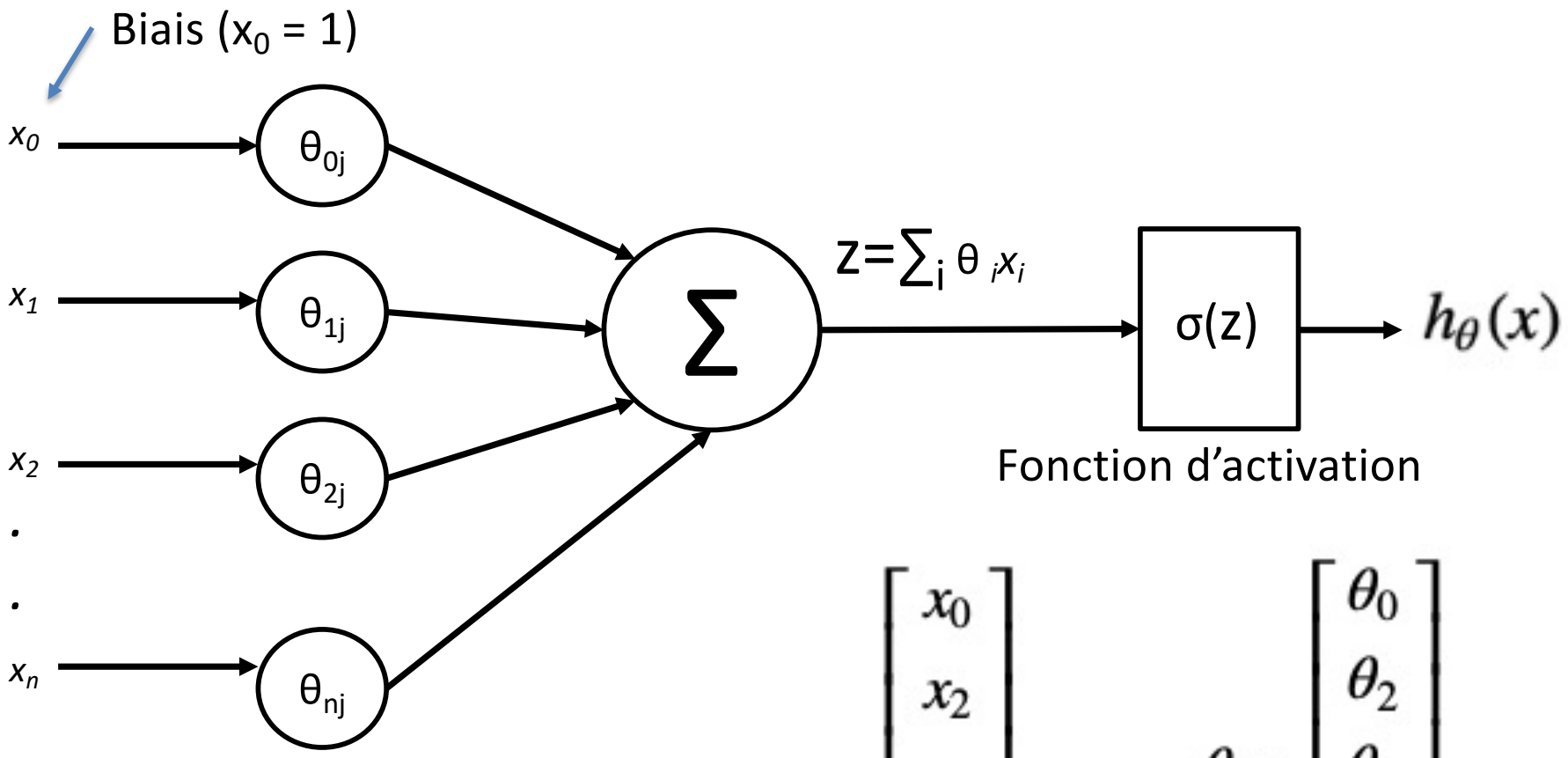
- Notebook sur la Régression logistique

Pour conclure

- Attention il faut toujours normaliser avant !!

- 1) données de tailles très variables
- 2) accélération de la convergence

Vers les réseaux de neurones



$$x = \begin{bmatrix} x_0 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_n \end{bmatrix}$$

- Des questions ?