

**Descente de Gradient**

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

---



---



---



---



---



---



---



---



---

**Qu'est ce que c'est ?**

WIKIPÉDIA L'encyclopédie libre

Accueil Portails thématiques Pages au hasard Contact

Contribuer Débrouse sur Wikipédia Aide Communauté Modifications récentes Faire un don Outils

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

Non connecté Discussion Contribuer Créez un compte Se connecter Article Discussion Lire Modifier Modifier le code Voir l'historique Rechercher dans Wikipedia

Algorithme du gradient

L'algorithme du gradient est 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 vectoriel. 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 suffit 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

2

---



---



---



---



---



---



---



---



---



---

**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

3

---



---



---



---



---



---



---



---



---



---

## 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



4

---



---



---



---

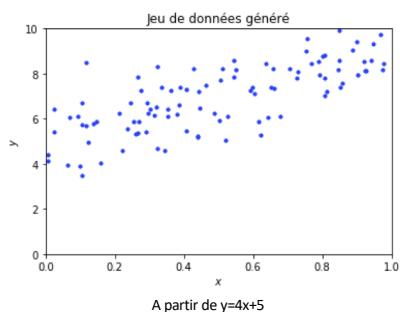


---



---

## Un problème de régression linéaire



5

---



---



---



---

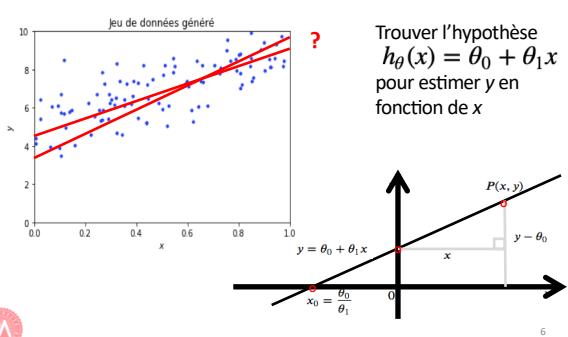


---



---

## Un problème de régression linéaire



6

---



---



---



---



---



---

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

```

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 prédictée : ',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 prédictée : [2.0253381] **Erreur !**

7

## 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

8

## MSE

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

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

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

- differences 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

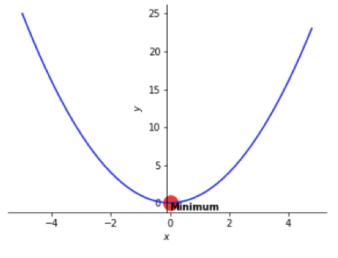
9

## Comment minimiser $J$ ?

En fait :

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

$$y = x^2$$



10

---

---

---

---

---

---

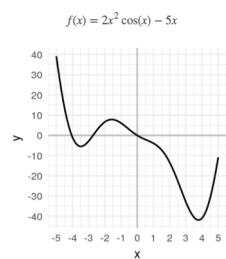
---

---

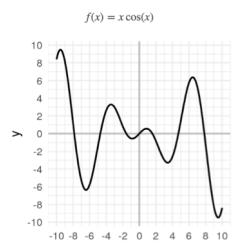
---

---

## Les fonctions ne sont pas toujours convexes !



11




---

---

---

---

---

---

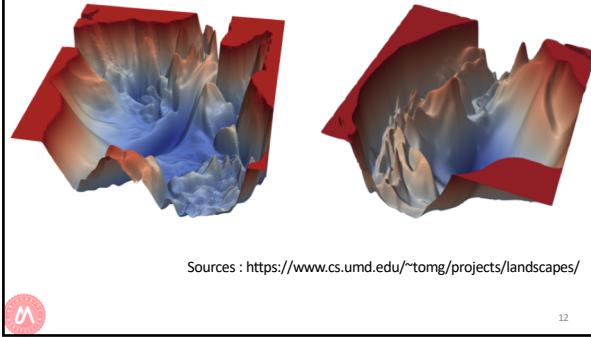
---

---

---

---

## Les fonctions ne sont pas toujours convexes !



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



12

---

---

---

---

---

---

---

---

---

---

## Algorithme de descente de gradient



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



13

---

---

---

---

---

---

---

---

---

---

---

---

## Algorithme de descente de gradient

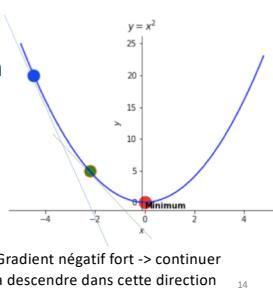
- Pour un point donné, la valeur du gradient (**la dérivée**) correspond à l'inclinaison de la pente en ce point
- Pente point vert ( $x=-2.3$ ) plus faible que point bleu ( $x=-4.5$ )



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

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

$$f' = 2x$$



14

---

---

---

---

---

---

---

---

---

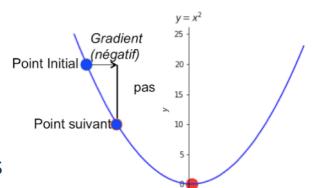
---

---

---

## Algorithme de descente de gradient

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



15

---

---

---

---

---

---

---

---

---

---

---

---

## Tester la valeur du pas

---



---



---



---



---



---



---

[https://developers.google.com/machine-learning/crash-course/fitter/graph.](https://developers.google.com/machine-learning/crash-course/fitter/graph)



16

## Rappel sur les dérivées partielles

---



---



---



---



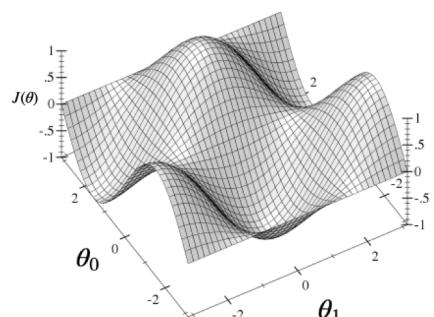
---



---



---



17



## Rappel sur les dérivées partielles

---



---



---



---



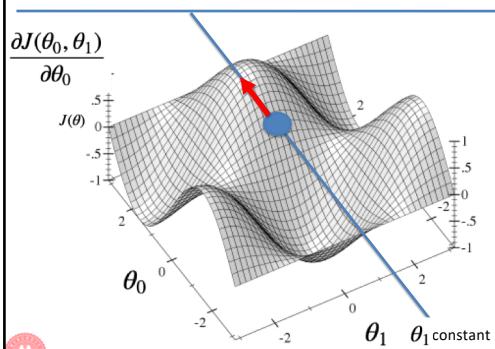
---



---



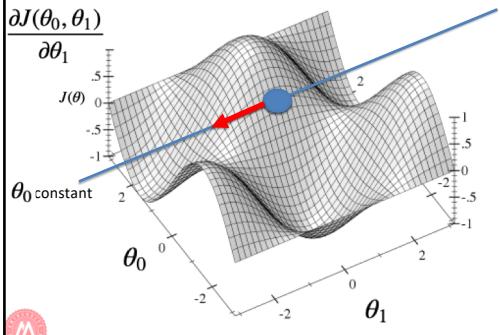
---



18



### Rappel sur les dérivées partielles



19

---

---

---

---

---

---

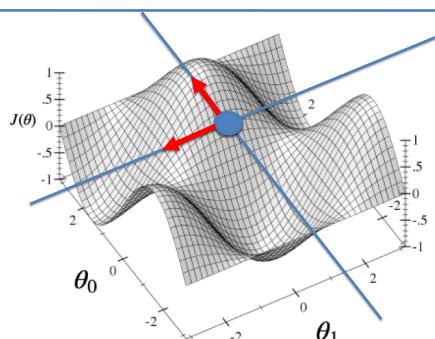
---

---

---

---

### Rappel sur les dérivées partielles



20

---

---

---

---

---

---

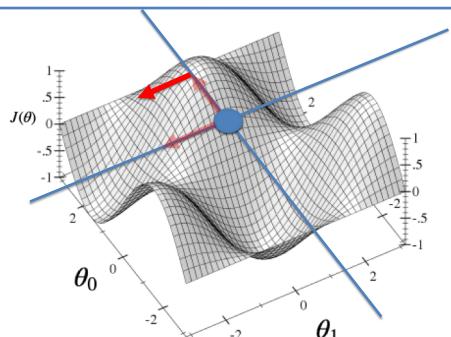
---

---

---

---

### Rappel sur les dérivées partielles



21

---

---

---

---

---

---

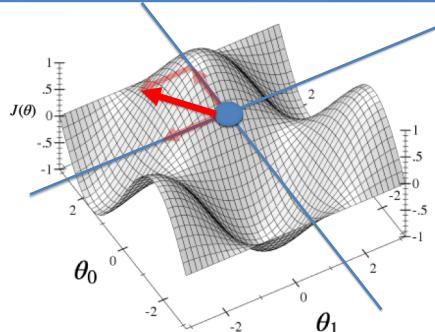
---

---

---

---

## Rappel sur les dérivées partielles



22

## Définition des 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  $\theta_0$  et  $\theta_1$  :

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

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

23

## L'algorithme

- Initialisation de  $\theta_0$  et  $\theta_1$  avec des nombres aléatoires
- Répéter jusqu'à convergence {
 
$$\theta_0 = \theta_0 - \eta \frac{-2}{m} \sum_{i=1}^m ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})$$

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

24

## Essai

- Notebook régression linéaire



25

---

---

---

---

---

---

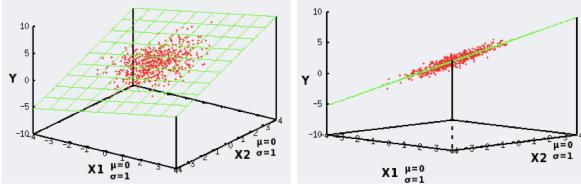
---

---

---

---

## Régression linéaire multiple



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

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



26

---

---

---

---

---

---

---

---

---

---

## Utilisation de matrices

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

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

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_n \end{bmatrix} \quad (\text{taille : } n+1 \times 1)$$

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

27

---

---

---

---

---

---

---

---

---

---

## Utilisation de matrices

Ajout une colonne de 1 à X

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

$$\begin{aligned} h_{\theta}(x) &= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n \\ &= \theta^T X \end{aligned}$$

Où  $\theta^T$  correspond à la transposée de la matrice  $\theta$

28

## Prédiction

Prédiction = simple produit scalaire de matrice

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \dots & x_n^1 \\ 1 & x_1^2 & x_2^2 & \dots & x_n^2 \\ 1 & x_1^3 & x_2^3 & \dots & x_n^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^m & x_2^m & \dots & x_n^m \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_n \end{bmatrix}$$

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

29

## Les dérivées partielles

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

30

## L'algorithme

- Initialisation de  $\theta_j$  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  $\theta_j$  à jour avec  $j=(1..n)$*

}

31

---

---

---

---

---

---

---

---

---

---

## Ce qui donne

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

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

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

:

32

---

---

---

---

---

---

---

---

---

---

## L'algorithme simplifié

- Initialisation de  $\theta_j$  avec des nombres aléatoires
- Répéter jusqu'à convergence {

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

}

33

---

---

---

---

---

---

---

---

---

---

## Essai

- Notebook Régression linéaire multiple



34

---



---



---



---



---



---



---

## 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*



35

---



---



---



---



---



---



---

## 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 coûteuses en temps de calcul pour 1 exemple, quid des matrices ?*



36

---



---



---



---



---



---



---

## 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

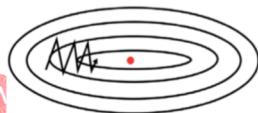


37

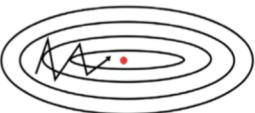
## Momentum

- 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

Descente de Gradient sans Momentum



Descente de Gradient avec Momentum



38

## 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 ( $\beta$ )

$$\begin{aligned} v &= \beta v - \eta \Delta J & v &= \beta v - \eta \times \frac{-2}{m} \times (X \cdot \theta - y) X^T \\ \theta &= \theta - v & \theta &= \theta - v \end{aligned}$$



39

## D'autres optimisations

- Nesterov Momentum assez proche de Momentum mais qui, au lieu de calculer le gradient à la position actuelle, le calcule à une nouvelle position approximée
- Adagrad s'intéresse au learning rate. Précédemment  $\eta$  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  $\eta$  devient trop petit
- ADAM cumule les avantages de Momentum et de RMSprop



40

---

---

---

---

---

---

---

---

## 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 ?

41

---

---

---

---

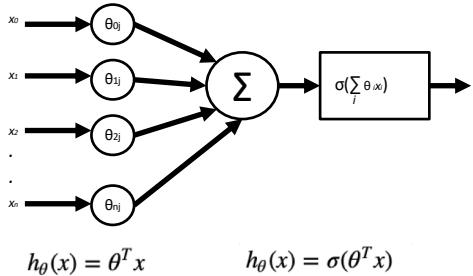
---

---

---

---

## Régression Logistique




---

---

---

---

---

---

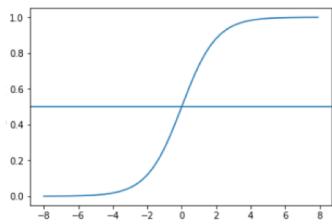
---

---

## Utilisation de la sigmoid

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

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



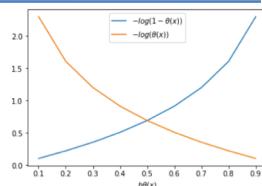
$$h_{\theta}(x) = \begin{cases} > 0.5, & \text{si } \theta^T x > 0 \\ < 0.5, & \text{si } \theta^T x < 0 \end{cases}$$

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

43

## Fonction de coût

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



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)

44

## 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 :

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

45

## Essai

- Notebook sur la Régression logistique

46

---



---



---



---



---



---



---

## Pour conclure

- Attention il faut toujours normaliser avant !!
- 1) données de tailles très variables
- 2) accélération de la convergence

47

---



---



---



---



---

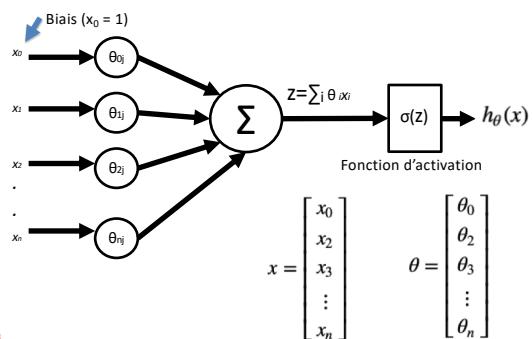


---



---

## Vers les réseaux de neurones




---



---



---



---



---



---



---

- Des questions ?



49

---

---

---

---

---

---