IA Géosciences - Deep learning: Méthode de la descente du gradient

Romain Wenger (Laboratoire Image Ville Environnement)

Table des matières

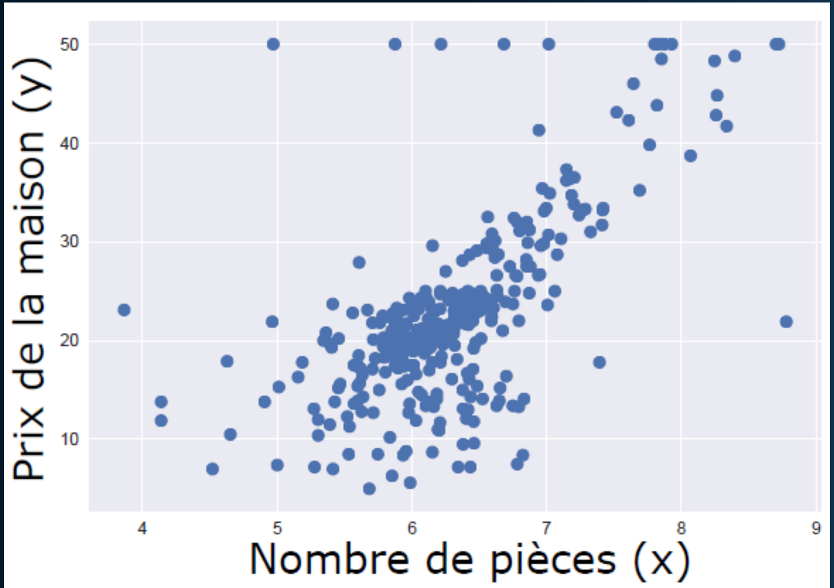
Table des matières

Préliminaires

1. Définir un modèle (e.g. fonction linéaire)
2. Définir un objectif (e.g. fonction de coût)
3. Minimiser la fonction objective
    - Essayer avec la force brute (trop long)
    - Solution optimale (dérivée égale zéro)
    - **Descente du gradient (signe de la dérivée)**

Boston House Prices

Prédire y en fonction de x



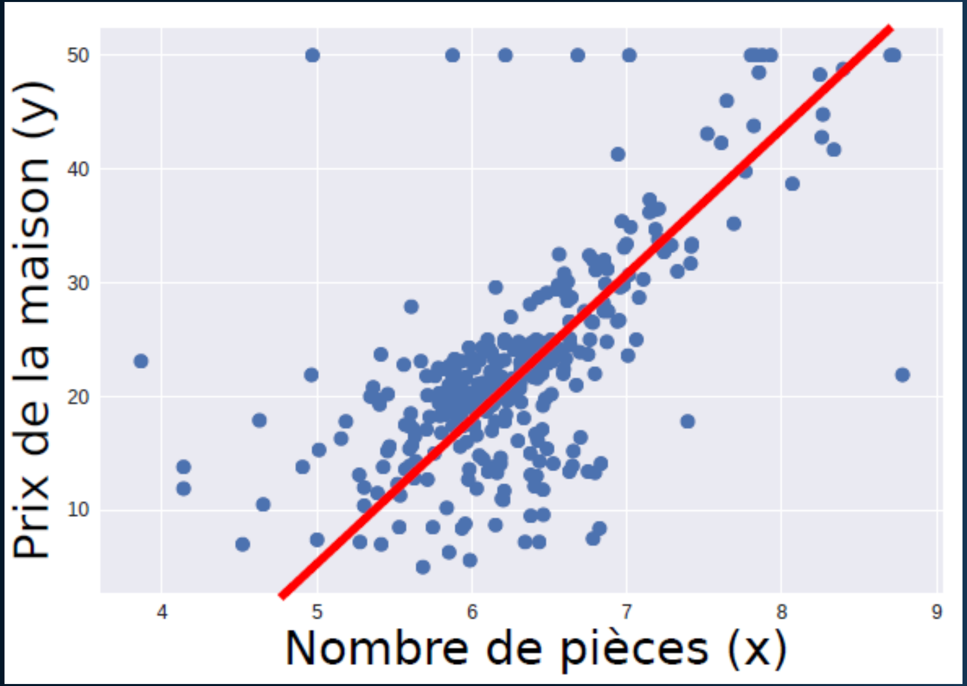
Définir un modèle

Une hypothèse :

$$\hat{y} = h(x)$$

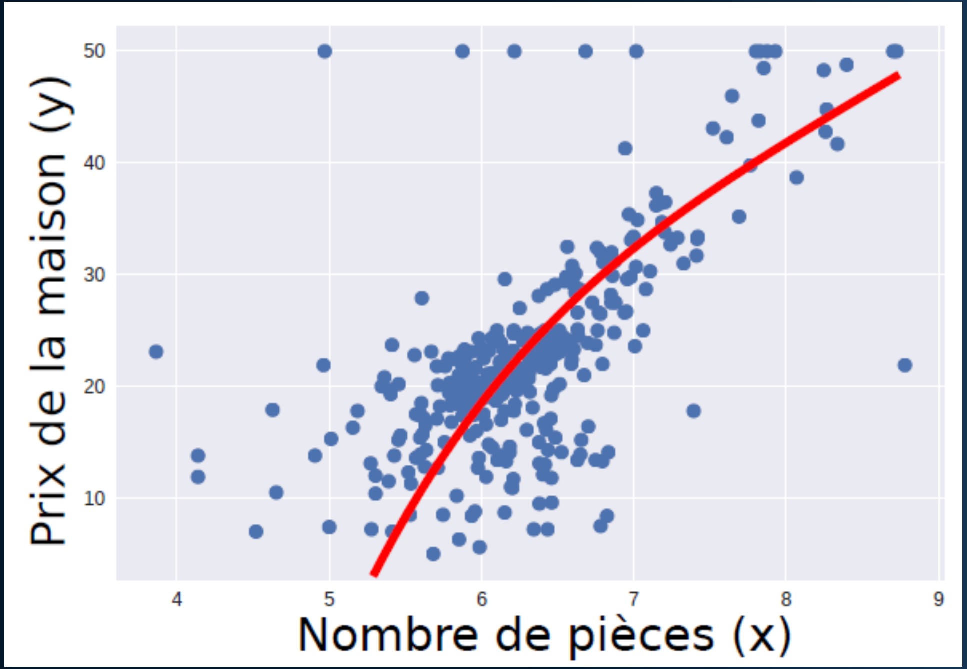
Hypothèse : droite linéaire

$$\hat{y} = h(x) = w * x + b$$



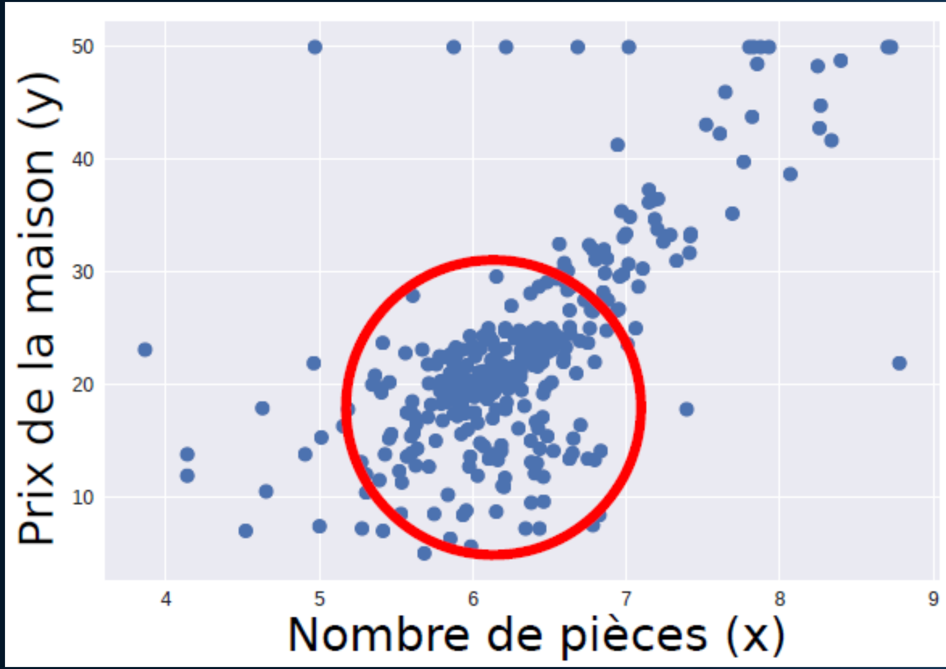
Hypothèse : courbe

$$\hat{y} = h(x) = \; w_1 * x^2 + w_2 * x + b$$



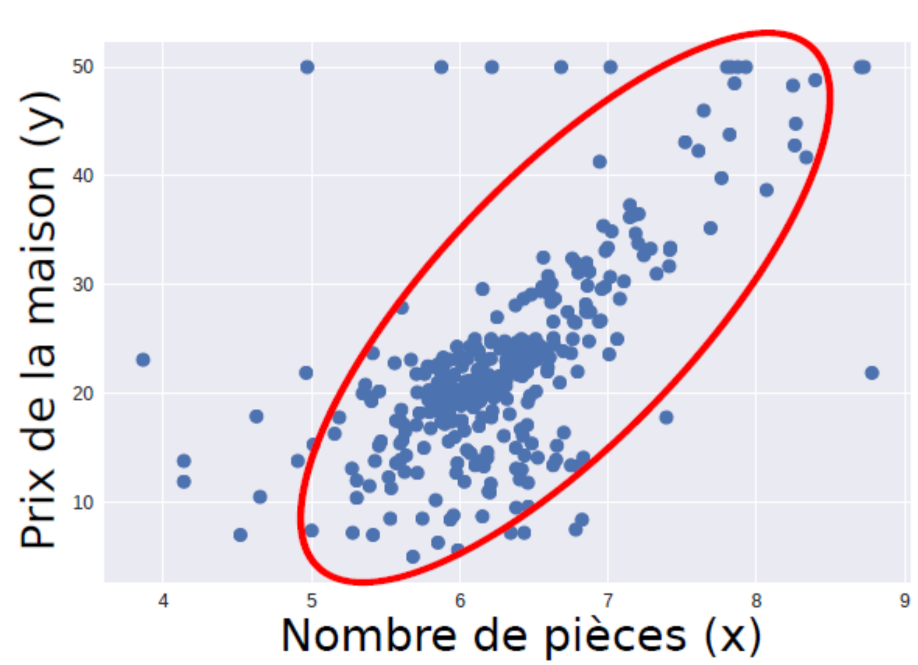
Hypothèse : cercle

$$\hat{y} = h(x) = \pm\sqrt{x^2 - b^2}$$



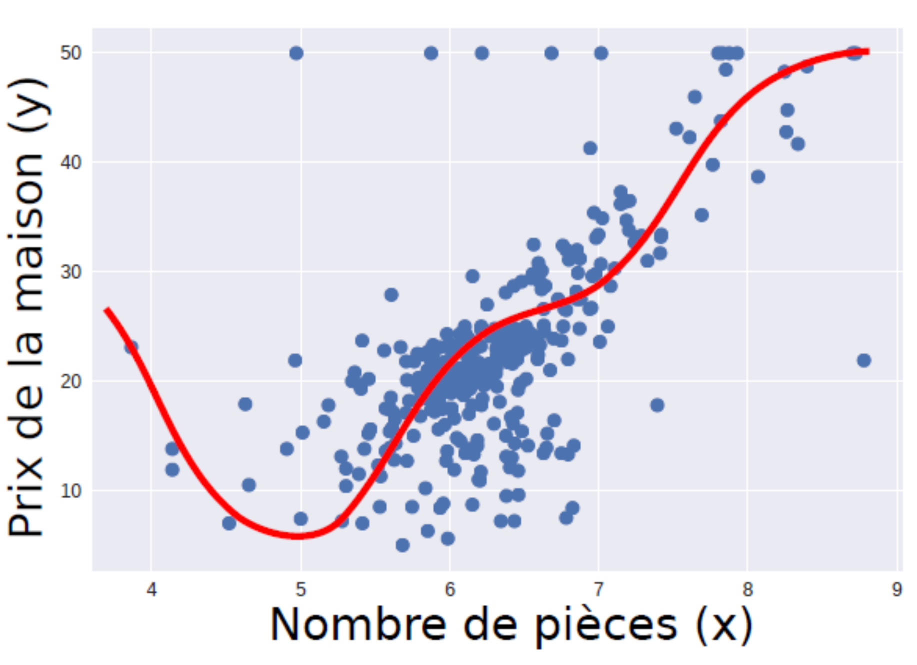
Hypothèse : ellipse

$$\hat{y} = h(x) = \pm\sqrt{1 - w_1 * x^2}$$



Hypothèse : fonction complexe

$$\hat{y} = h(x) = f(x, w_1, w_2, \dots, w_n)$$



Définir un objectif

Fonction de coût :

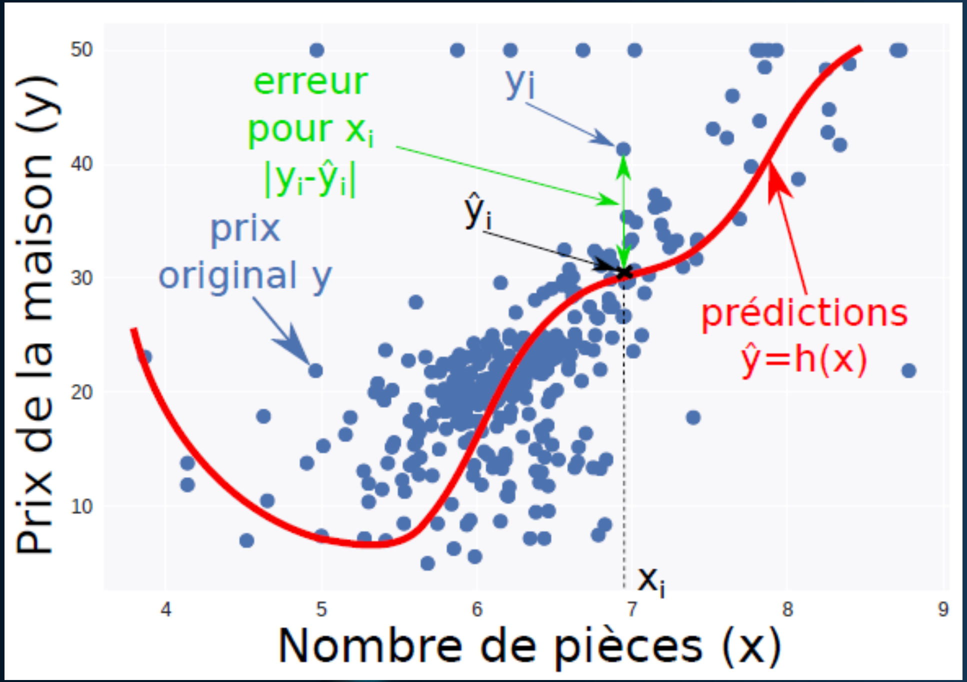
- Fonction composée : elle « englobe » notre hypothèse

$$\boldsymbol{L}(\boldsymbol{h})$$

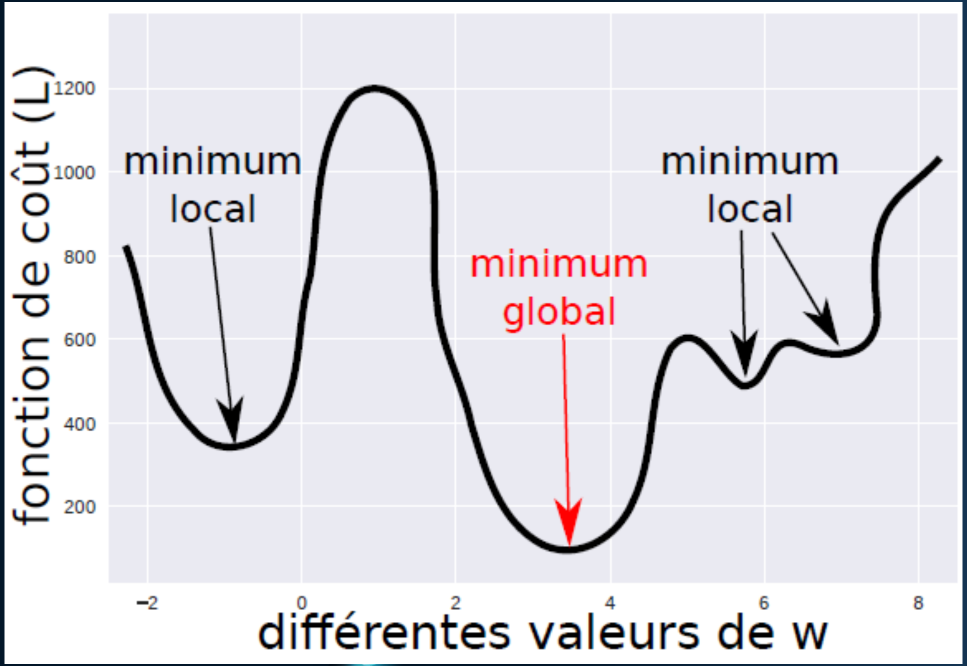
Objectif :

$$\min_{\boldsymbol{h}} \boldsymbol{L}(\boldsymbol{h})$$

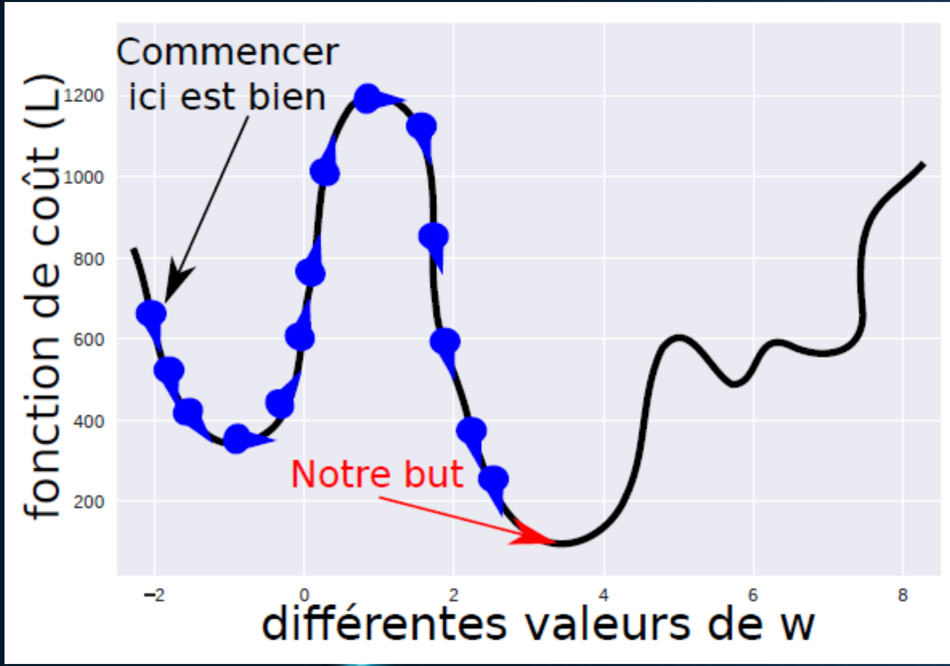
Définition de l'objectif est indépendante du choix du modèle



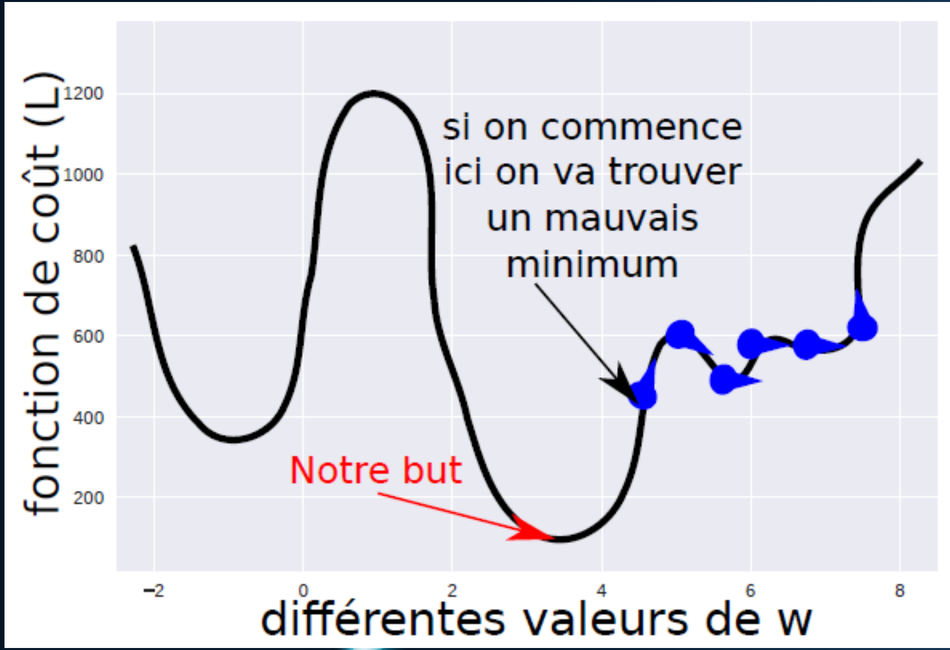
Minimiser une fonction de coût



Méthode par force brute



Méthode par force brute



Méthode par force brute

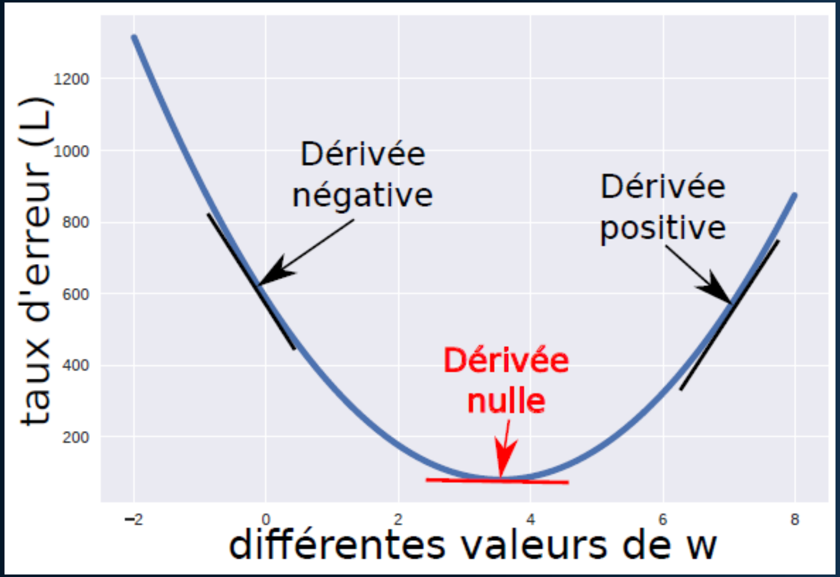
Ne pas appliquer la force brute car :

- Méthode très lente (si on a $w_1, w_2, \dots\ w_{1000}$)
- Difficile de savoir avec quel w commencer
- On peut tomber sur un mauvais minimum

=> On a besoin d'une fonction convexe pour trouver un seul et unique minimum

Dérivée (ou gradient)

Dérivée augmentation partielle (très faible) de w



Solution optimale

But : trouver w pour

$$\frac{dL}{dw} = 0$$

Plusieurs méthodes existent :

- Inverser les équations normales
- Décompositions orthogonales

Limitations :

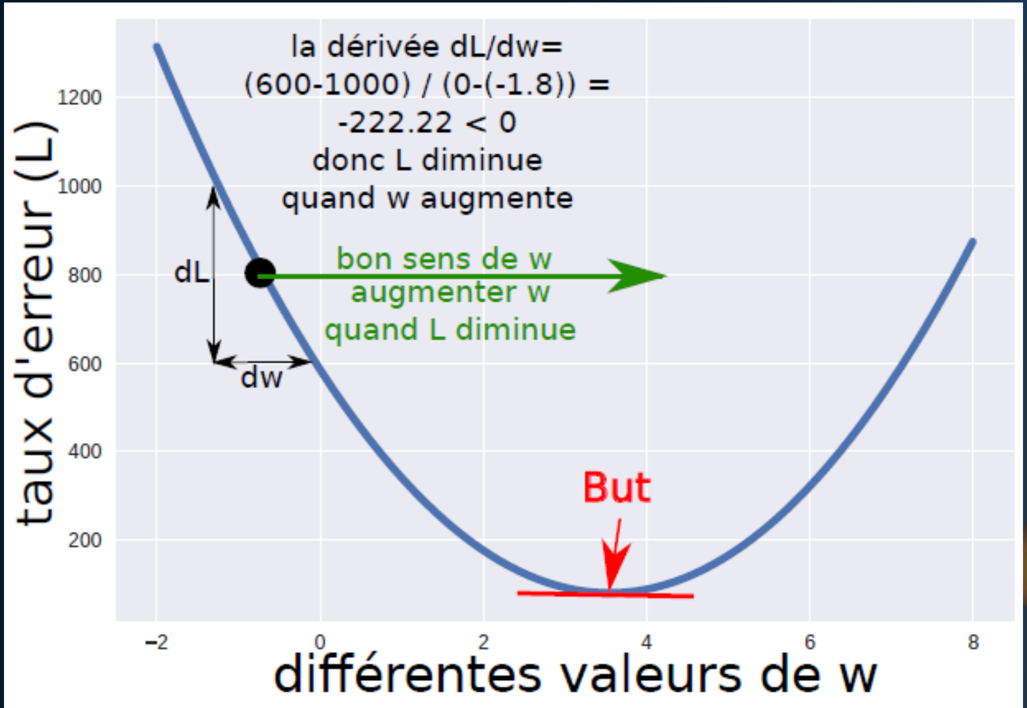
- Méthodes très lentes surtout si le modèle est complexe
- Problème si plusieurs solutions existent pour l'équation ci-dessus

Table des matières

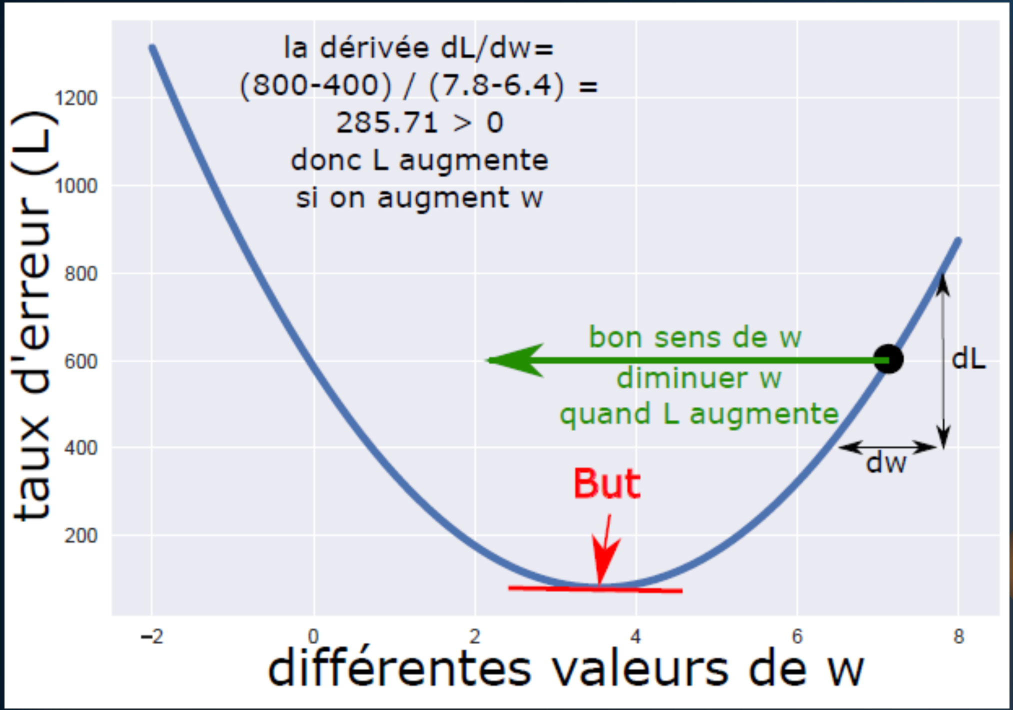
Analogie de la descente du gradient

https://www.kdnuggets.com/2018/06/intuitive-introduction-gradient-descent.html

Le coût diminue



Le coût augmente



Comment varier w ?

Comment varier w ?

- Si w augmente, le taux d'erreur L augmente
  - On est dans une montée
  - Il faut diminuer w
- Si w augmente, le taux d'erreur L diminue
  - On est dans une descente
  - Il faut augmenter w

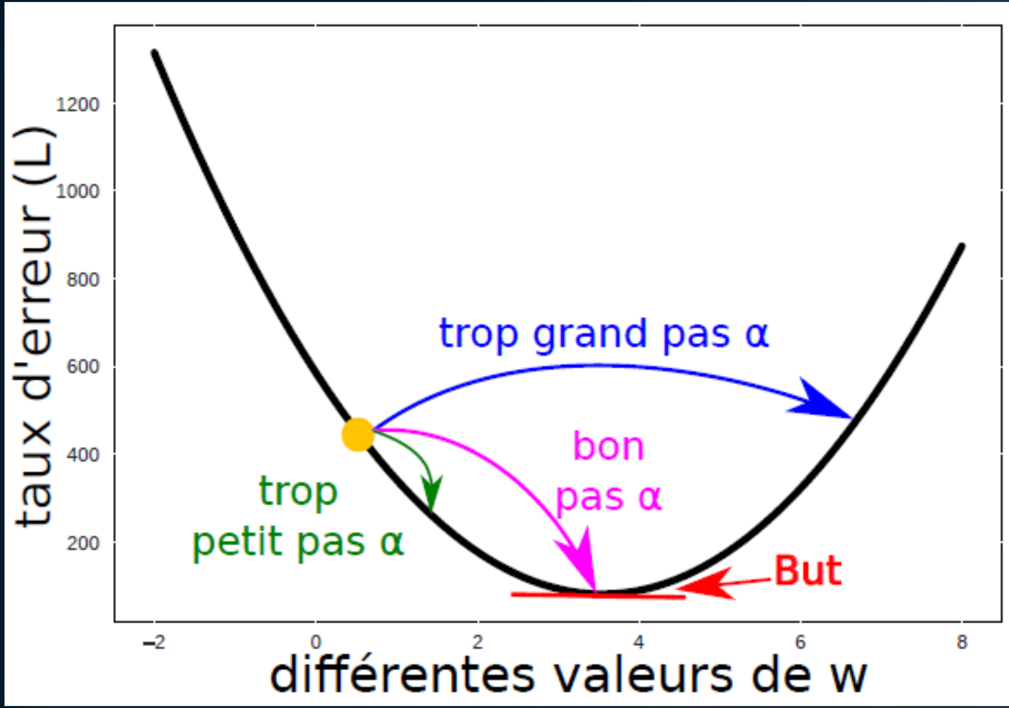
Ok maintenant formalisons tout cela

Comment varier w ?

$$w = w - \alpha \frac{dL}{dw}$$

- w : l'ensemble des paramètres à apprendre
- L : la fonction de coût qu'on veut minimiser
- α : le taux d'apprentissage qui contrôle la variation de w
- $-$: (le signe moins) signifie qu'on varie w dans le sens inverse de la variation de L ⇨ on minimise l'erreur

Pourquoi un taux d'apprentissage α ?



Bon choix de α

Comment choisir α ?

- Un très grand $\boldsymbol{\alpha}$ peut amener à rater l'objectif
- Un très petit $\boldsymbol{\alpha}$ peut amener à une convergence trop lente nécessitant trop d'itérations

⇨ Il faut choisir $\boldsymbol{\alpha}$ pour avoir un équilibre entre le temps de calcul disponible et la précision du minimum qu'on veut atteindre

⇨ Un choix souvent empirique et basé sur les travaux d'autres personnes

Notion de mini-batch

- L'algorithme « *Batch Gradient Descent* » optimise la fonction :

$$\boldsymbol{L(w) = \frac{1}{n}\sum_{i=1}^{n} erreur(y_i, \hat{y}_i)}$$

- Il faut calculer l'erreur sur l'ensemble du jeu d'entraînement ce qui est problématique si n est très grand

⇨ Notion de « mini-batch »

⇨ Calculer L pour $m \lll n$

$$\boldsymbol{L(w) = \frac{1}{m}\sum_{i=s}^{s+m-1} erreur(y_i, \hat{y}_i)\ \forall s\ \epsilon\ \{1, m, 2m, \dots, N-m\}}$$

- Une fois qu'on a passé tous les mini-batchs = une époque (*epoch* en anglais)
- On répète le processus d'entraînement sur plusieurs époques (i itérations)

Pseudo-code pour mini-batch gradient descent

Entrées : X, Y l’ensemble du jeu de données d’entrainement

Sortie : W les paramètres qui modélisent le mieux la relation $X \Rightarrow Y$

Initialisation aléatoire de W

Pour $epoque = 1 \text{ à } 1000$ ***faire***
    Pour *chaque mini-batch* X_m *de taille* m ***faire***
        $predictions = model(X_m, W)$
        $erreurs = fonction_{erreur(Y_m, predictions)}$
        $gradient = \frac{dL(W)}{dW}$
        $W = W - \alpha * gradient$
    fin Pour
fin Pour
Renvoyer W

Variants de la méthode de descente du gradient

Soit m la taille du mini-batch

- Si $m = n \Rightarrow$ *Batch Gradient Descent*
  - Il faut passer sur tout l'ensemble pour mettre w à jour
- Si $m = 1 \Rightarrow$ *Stochastic Gradient Descent*
  - On perd la parallélisation du calcul sur plusieurs exemples
  - Maj des poids pour chaque exemple
- Si $1 < m <\ll n \Rightarrow$ *Minibatch Gradient Descent*
  - On accélère une mise à jour de w en parallélisant sur m exemples
  - Moyenne des poids sur chaque mini-batch

Variants de la méthode de descente du gradient

Batch gradient descent

```
optimizer = torch.optim.Adam(model.parameters(),
                             lr=learning_rate)
for epoch in range(nepochs+1):


    y_pred = model(X)
    loss = F.cross_entropy(y_pred, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step() # adjust weights
```

Mini-batch stochastic gradient descent

```
optimizer = torch.optim.Adam(model.parameters(),
                             lr=learning_rate)
for epoch in range(nepochs+1):
    shuffle X_train, y_train
    for p in range(0, n, batch_size): # one epoch
        batch_X = X_train[p:p+batch_size]
        batch_y = y_train[p:p+batch_size]
        y_pred = model(batch_X)
        loss = F.cross_entropy(y_pred, batch_y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step() # adjust weights
```

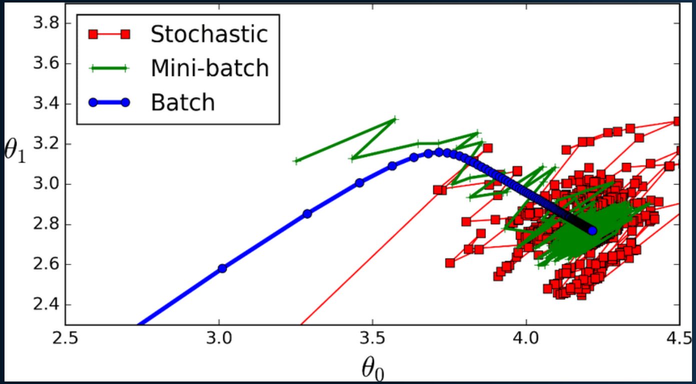


Table des matières

Fonction de coût

Forme générale

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} erreur(y_i, \hat{y}_i)$$

L'erreur quadratique moyenne

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i(w))^2$$

L'erreur quadratique moyenne pour la régression linéaire

$$L(w, b) = \frac{1}{n}\sum_{i=1}^{n} (y_i - (w * x_i \ + b))^2$$

Pour la régression linéaire : deux paramètres à apprendre w et b

Calcul du gradient par rapport à w pour un modèle linéaire

$$\frac{dL(w,b)}{dw} = \frac{d(\frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{y_i} - (\boldsymbol{w} * \boldsymbol{x_i} + \boldsymbol{b})\right)^2)}{dw}$$

$$\frac{dL(w,b)}{dw} = \frac{1}{n}\sum_{i=1}^{n}\frac{d\left(\boldsymbol{y_i} - (\boldsymbol{w} * \boldsymbol{x_i} + \boldsymbol{b})\right)^2}{dw}$$

Or, $\frac{dy_i}{dw} = 0$ car y_i est independant de w :

$$\frac{dL(w,b)}{dw} = \frac{1}{n}\sum_{i=1}^{n}(-2\boldsymbol{x_i}(\boldsymbol{y_i} - (\boldsymbol{w} * \boldsymbol{x} + \boldsymbol{b})))$$

$$\frac{dL(w,b)}{dw} = -\frac{2}{n}\sum_{i=1}^{n}(\boldsymbol{x_i}(\boldsymbol{y_i} - \hat{\boldsymbol{y}}_i))$$

Mise à jour de w :

$$\boxed{w = w - \alpha * (-\frac{2}{n}\sum_{i=1}^{n}(x_i\,(\boldsymbol{y_i} - \hat{\boldsymbol{y}}_i)))}$$

$$\boxed{\boldsymbol{W} = \boldsymbol{W} - \boldsymbol{\alpha} * \boldsymbol{gradient}}$$

Calcul du gradient par rapport à b pour un modèle linéaire

$$\frac{dL(w,b)}{db} = \frac{d(\frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{y_i} - \left(\boldsymbol{w} * \boldsymbol{x_i} + \boldsymbol{b}\right)\right)^{2})}{db}$$

$$\boxed{\boldsymbol{b} = \boldsymbol{b} - \boldsymbol{\alpha} * \boldsymbol{gradient}}$$

$$\frac{dL(w,b)}{db} = \frac{1}{n}\sum_{i=1}^{n}\frac{d\left(\boldsymbol{y_i} - \left(\boldsymbol{w} * \boldsymbol{x_i} + \boldsymbol{b}\right)\right)^{2}}{db}$$

Or, $\frac{dy_i}{db} = 0$ car y_i est independant de b :

$$\frac{dL(w,b)}{db} = \frac{1}{n}\sum_{i=1}^{n}(-\boldsymbol{2}(\boldsymbol{y_i} - (\boldsymbol{w} * \boldsymbol{x} + \boldsymbol{b})))$$

$$\frac{dL(w,b)}{db} = -\frac{2}{n}\sum_{i=1}^{n}(\boldsymbol{y_i} - \hat{\boldsymbol{y}}_i)$$

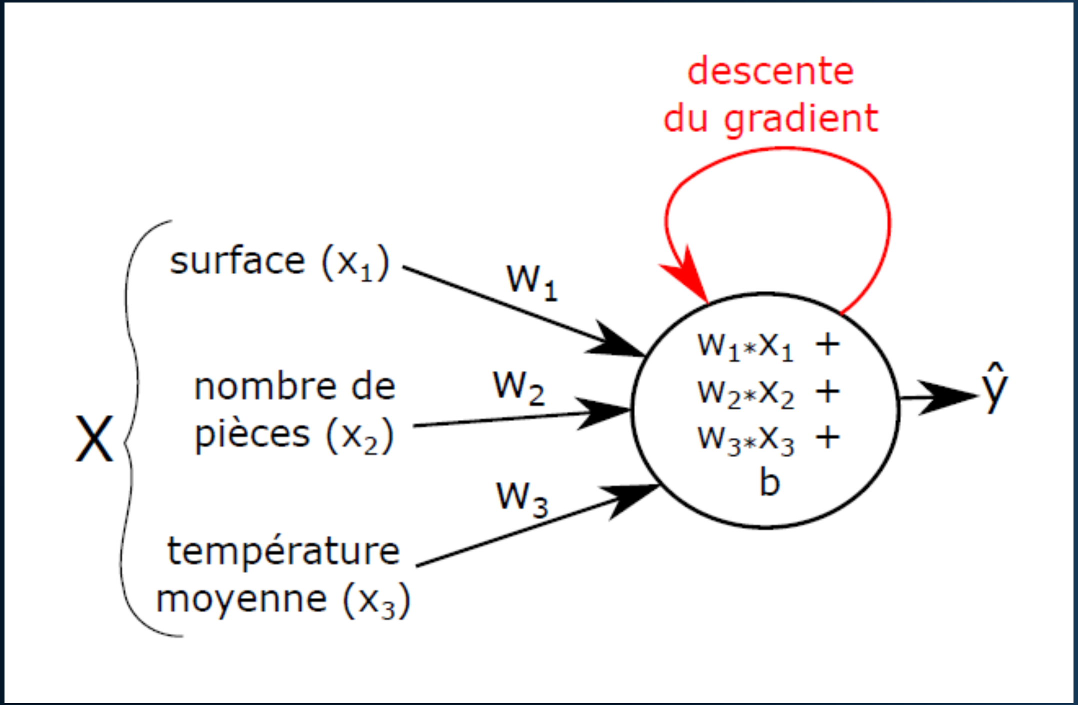
Mise à jour de b :

$$\boxed{\mathrm{b} = b - \alpha * (-\frac{2}{n}\sum_{i=1}^{n}(\boldsymbol{y_i} - \hat{\boldsymbol{y}}_i))}$$

Régression linéaire avec plusieurs variables

- Si l'instance possède plusieurs caractéristiques
- **Exemple** : prédire le prix d'une maison à partir nombre de pièces, population du quartier, le climat de la région, etc.
- Au lieu d'un seul w on aura $W = (w_1, w_2, \ldots, w_r)$ pour r caractéristiques différentes
- On répète le même processus pour chaque $w_i \epsilon W$
- Le modèle devient alors : $\boldsymbol{\hat{y}_i = b + \sum_{j=1}^{r} w_j * x_{i,j}}$
- Avec $x_{i,j}$ la $j^{ième}$ caractéristique du $i^{ième}$ individu de l'ensemble X d'entraînement

Exemple de régression linéaire avec un neurone (Perceptron)



Normalisation des données en entrée

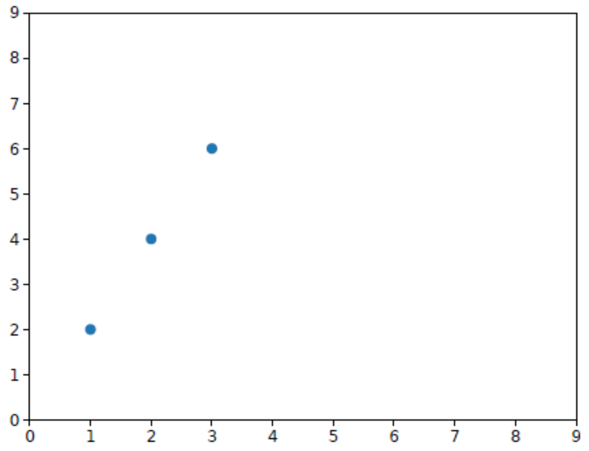
- Consiste à rendre chaque caractéristique entre 0 et 1
- C'est important pour que toutes les caractéristiques aient la même contribution
- Par exemple :
    - La surface d'une maison est entre 0 et 1000 m^2
    - Le nombre de pièce est entre 0 et 10

Sans normalisation ⇨ une petite variation du nombre de pièces n'a pas un grand effet par rapport aux valeurs énormes de la surface ⇨ le modèle aura des difficultés à apprendre

$$\boldsymbol{X_{:,i}} = \frac{\boldsymbol{X_{:,i}} - \mathbf{min}(\boldsymbol{X_{:,i}})}{\mathbf{max}(\boldsymbol{X_{:,i}}) - \mathbf{min}(\boldsymbol{X_{:,i}})} \ \forall \boldsymbol{i} \in [\mathbf{1}, \dots, \boldsymbol{r}]$$

r étant le nombre de caractéristiques dans le jeu de données

Exercice (1/2)



Y	X
2	1
4	2
6	3

Exercice : Trouver w en fixant $b = 0$ et $\alpha = 0.2$ et initialiser $w = 1$

Il faut qu'on arrive à une valeur proche de $w = 2$ (la solution optimale)

Exercice (2/2)

Première époque

$$erreur = \frac{1}{3}[(2 - 1 + 1)^2 + (4 - 1 * 2)^2 + (6 - 1 * 3)^2] \approx 4.67$$

$$gradient = \frac{-2}{3}[1(2 - 1) + 2(4 - 2) + 3(6 - 3)] \approx -9.33$$

$$w = 1 - 0.2(-9.33) \approx 1.87$$

Deuxième époque

$$erreur = \frac{1}{3}[(2 - 1.87 + 1)^2 + (4 - 1.87 * 2)^2 + (6 - 1.87 * 3)^2] \approx 0.08$$

$$gradient = \frac{-2}{3}[1(2 - 1.87) + 2(4 - 3.74) + 3(6 - 5.61)] \approx -1.21$$

$$w = 1.87 - 0.2(-1.21) \approx 2.112$$

On s'approche de $w = 2$ (la solution optimale)

Table des matières

Boston house prices (régression linéaire avec toutes les caractéristiques)

Prix	CRIM	ZN	INDUS	CHAS	NOX	RM	...
18.2	0.7258	0.	8.14	0.	0.538	5.727	...
20.3	0.3494	0.	9.9	0.	0.544	5.972	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

- Chaque maison possède un prix et 13 autres caractéristiques
- Une maison est représentée par le vecteur $x_i = (x_{i,1}, x_{i,2} \dots, x_{i,13})$
- 13 attributs $\Rightarrow$ 13 w_i et 1 b
- Soit un vecteur $W = (w_1, w_2, \dots, w_{13})$

Boston house prices (régression linéaire avec deux caractéristiques)

Prix	CRIM	ZN	INDUS	CHAS	NOX	RM	...
18.2	0.7258	0.	8.14	0.	0.538	5.727	...
20.3	0.3494	0.	9.9	0.	0.544	5.972	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Prix (y)	RM ($X_{:,1}$)	LSTAT ($X_{:,2}$)
18.2	5.727	0.18
20.3	5.972	0.25
⋮	⋮	⋮

Boston house prices (régression linéaire avec deux caractéristiques)

But : prédire le prix en fonction des deux variables

$$prix = f(X) \mid X = [X_{:,1}, X_{:,2}]$$

Modèles à apprendre

$$\hat{y} = W * X + B \Leftrightarrow \hat{y} = w_1 * X_{:,1} + w_2 * X_{:,2} + b$$

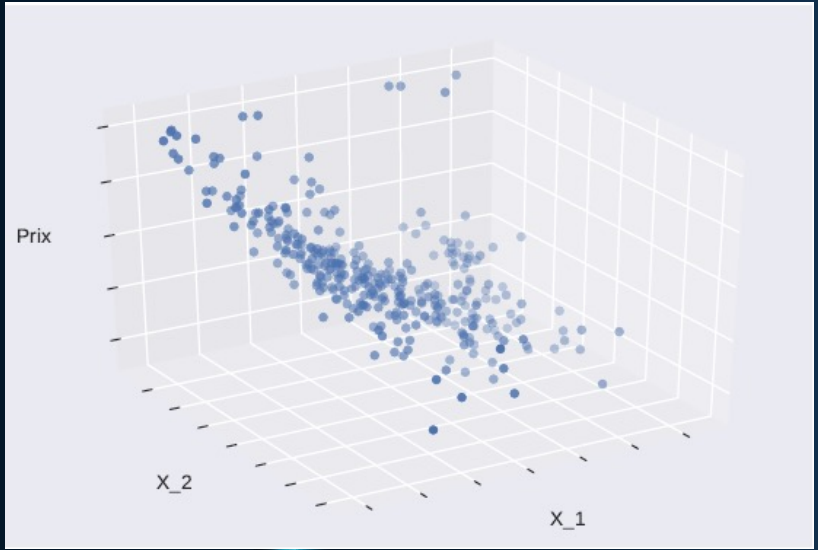
Cette équation correspond à celle d'un plan dans un espace 3D

$$z = w_1 * x += w_2 * y + b$$

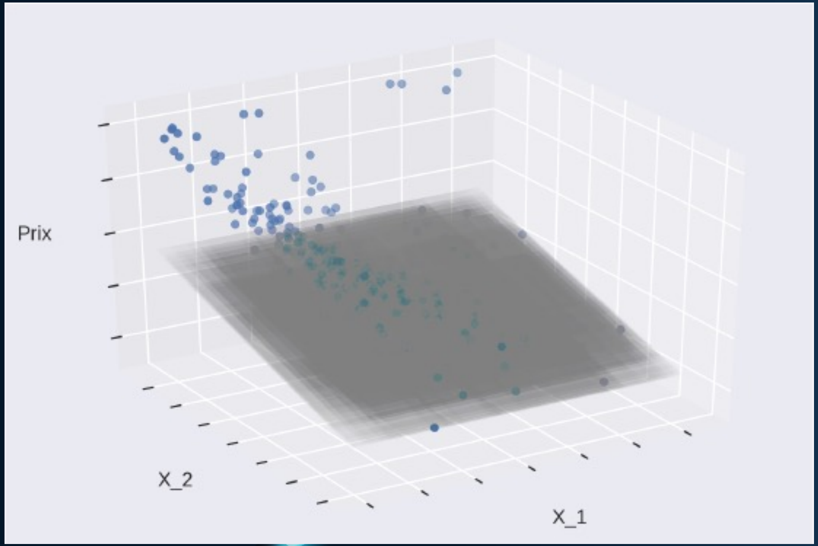
Avec

$$x = X_{:,1}, y = X_{:,2}, z = \hat{y}$$

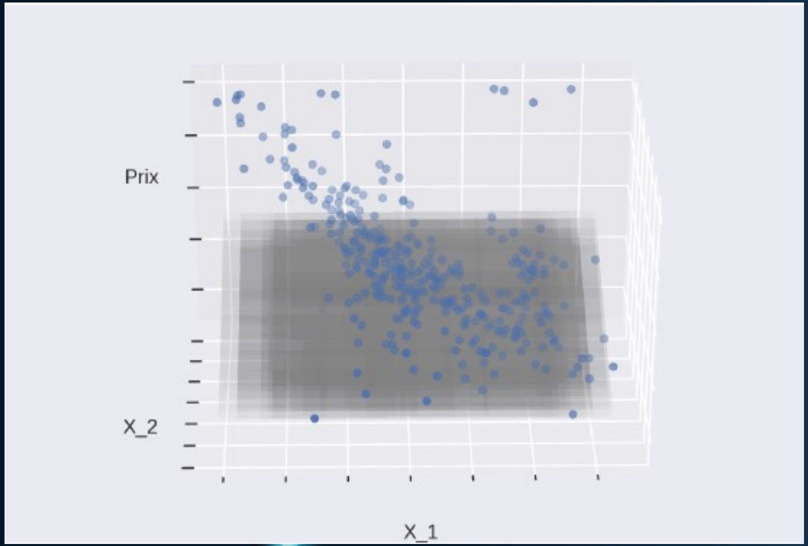
Boston house prices (régression linéaire avec deux caractéristiques)



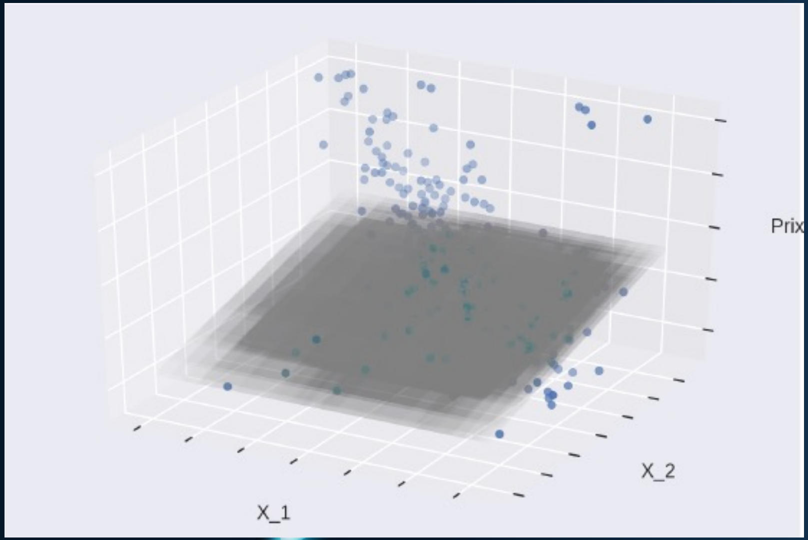
Boston house prices (régression linéaire avec deux caractéristiques)



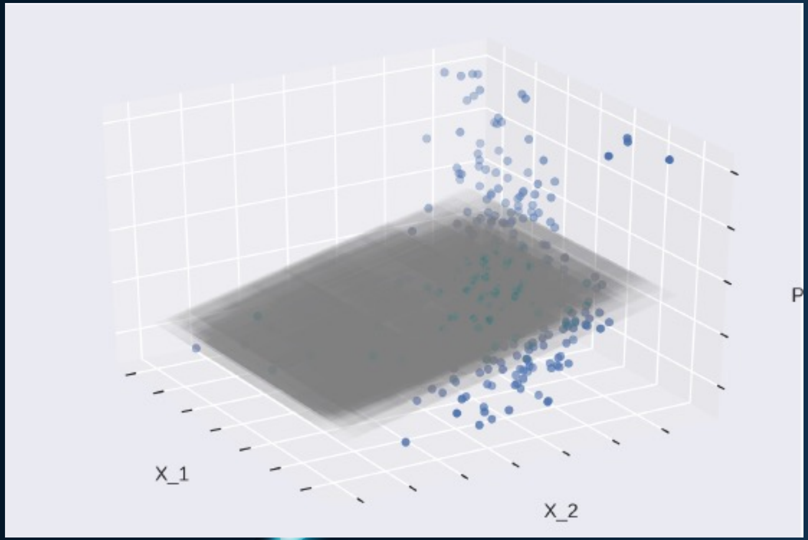
Boston house prices (régression linéaire avec deux caractéristiques)



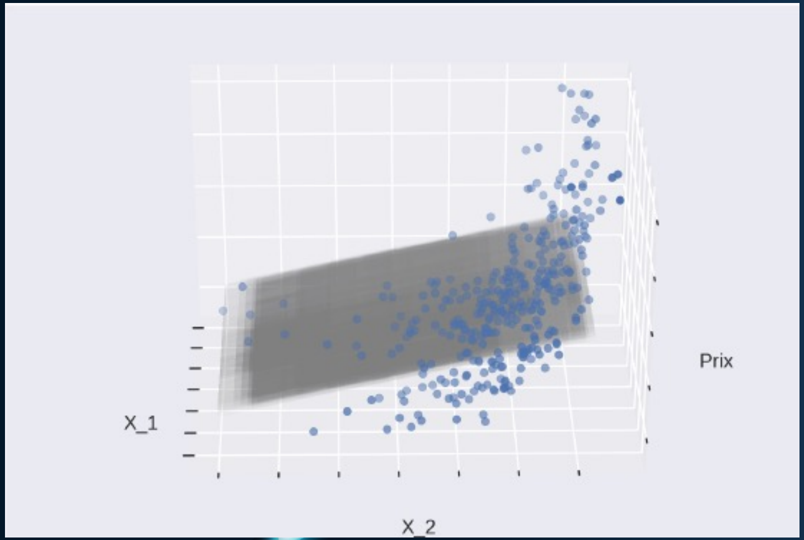
Boston house prices (régression linéaire avec deux caractéristiques)



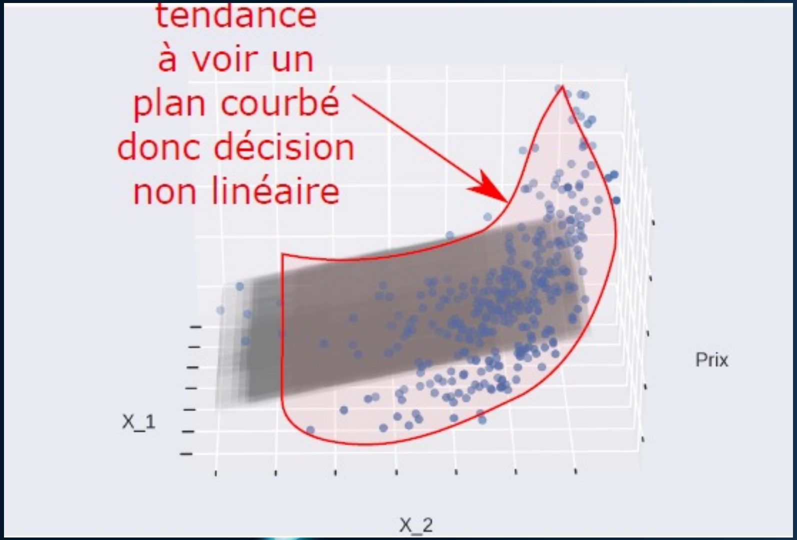
Boston house prices (régression linéaire avec deux caractéristiques)



Boston house prices (régression linéaire avec deux caractéristiques)



Boston house prices (régression linéaire avec deux caractéristiques)



Boston house prices (peut-être prédiction non linéaire ?)

- Non linéaire ⇒une interaction non linéaire entre les caractéristiques
- Par exemple : $x_1 * x_2$ ou x_1^2

$$\hat{y} = \boldsymbol{w_1 * X_{:,1} + w_2 * X_{:,2} + w_3 * X_{:,3} + b}$$

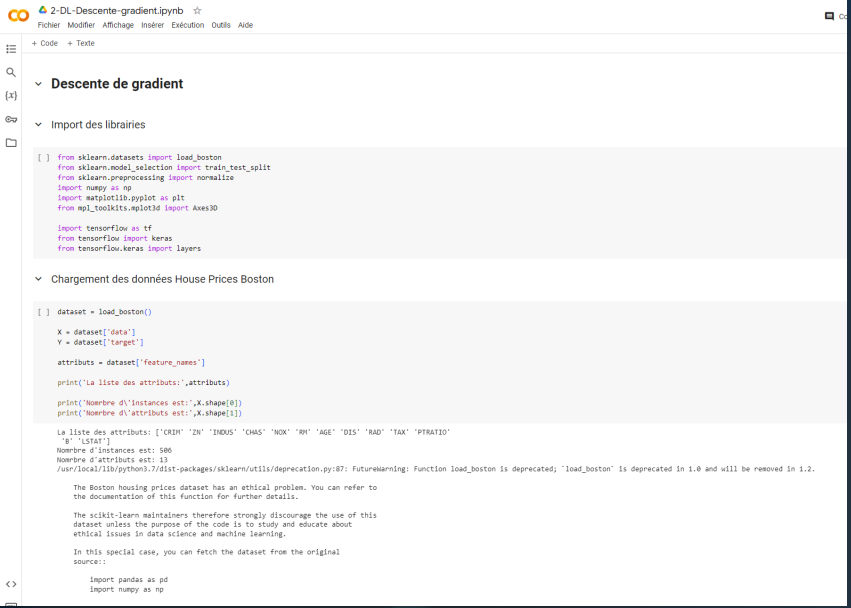
Avec

$$\boldsymbol{X_{:,3} = X_{:,1} * X_{:,2}}$$

Les données deviennent …

Prix (y)	RM ($X_{:,1}$)	LSTAT ($X_{:,2}$)	RM*LSTAT($X_{:,3}$)
18.2	5.727	0.18	5.727*0.18=1.03
20.3	5.972	0.25	5.972*0.25=1.493
⋮	⋮	⋮	⋮

De la théorie vers la pratique



Télécharger le fichier (https://github.com/r-wenger/cours_m1-m2-OTG/blob/main/IA_geosciences_M2/CM/2_DL_Descente_gradient.ipynb) et l'importer sur Google Colab