

II-Algèbre numérique matricielle

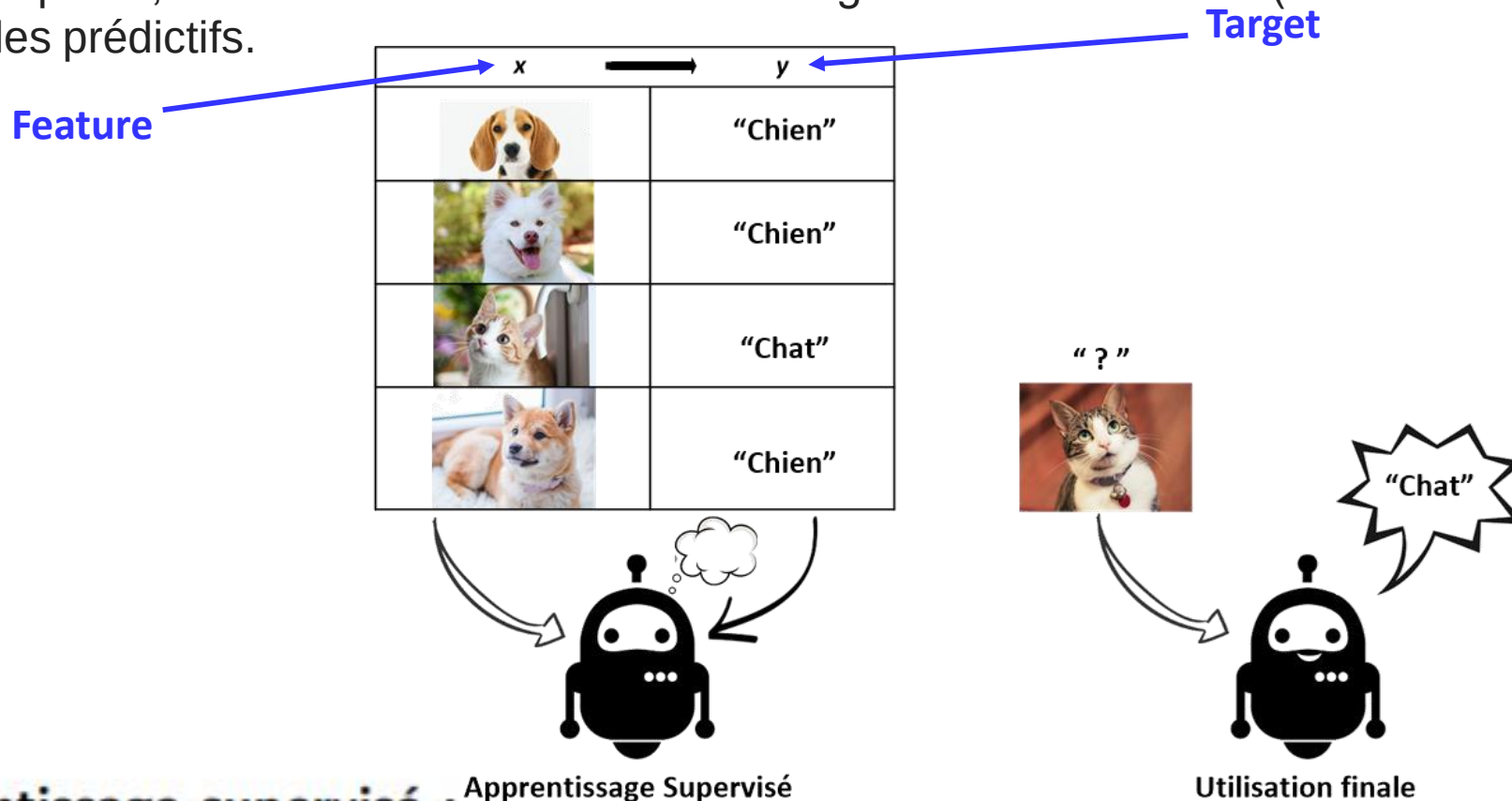
Module : Outils mathématiques pour l'ingénieur



1^{ère} année cycle d'ingénieur

Fadwa EL KIHAL

Dans cette partie, on verra comment fonctionne L'algorithme de Gradient (Gradient Descent Algorithm) pour calculer les modèles prédictifs.



Apprentissage supervisé :

A partir d'un échantillon d'apprentissage

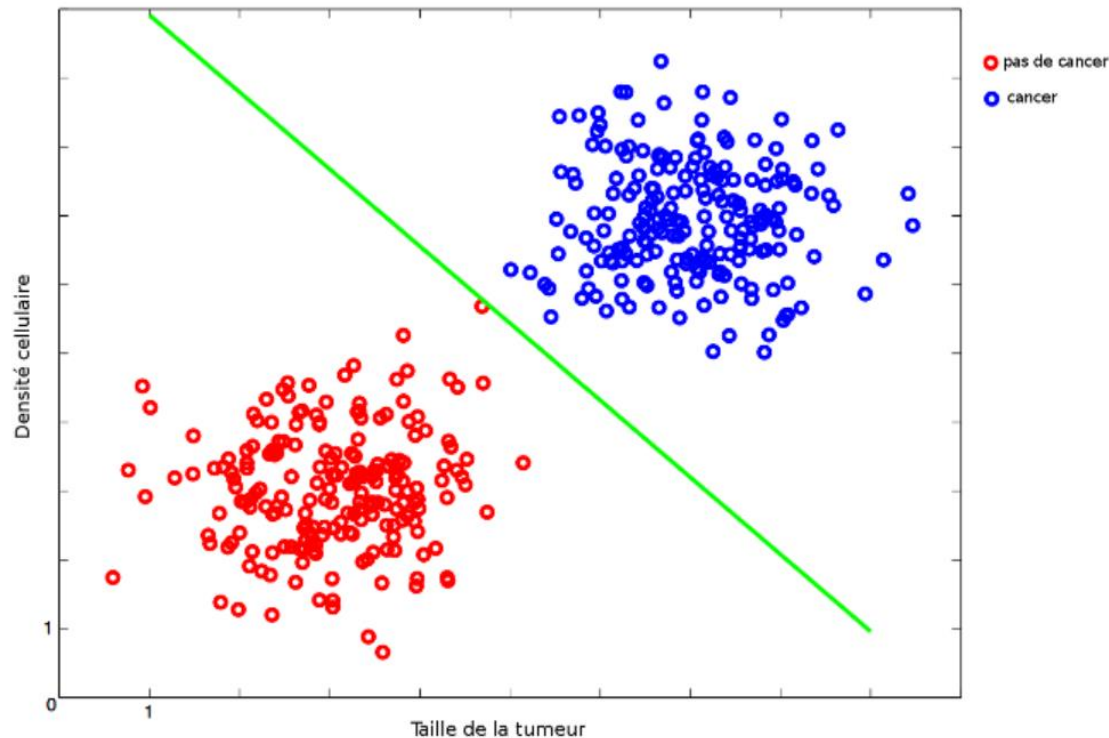
$\mathcal{D}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$, inférer la relation entre x et y .

Synonymes : discrimination, reconnaissance de formes

Voc : x_i = caractéristique = feature = variable explicative

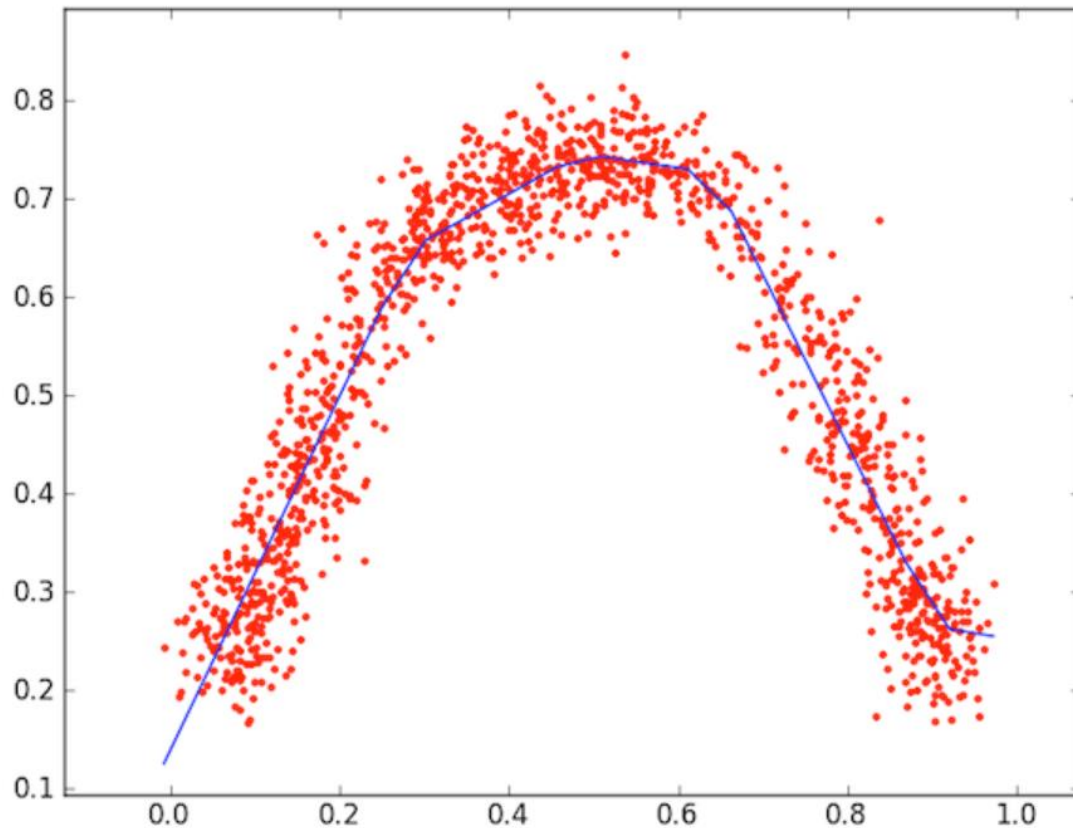
L'apprentissage supervisé (**supervised learning** en anglais) est certainement la forme de machine learning la plus répandue. **Elle consiste à entraîner le programme sur des exemples dont on connaît la catégorie** (c'est à cette connaissance qu'elle doit le nom "supervisé"). Il en existe deux types :

Classification



La classification consiste à déterminer la catégorie d'un objet possédant certaines caractéristiques. Par exemple, prédire si une personne possédant une tumeur est atteinte d'un cancer ou non en fonction de la taille de sa tumeur et de la densité de cellules dans un organe donné. Ainsi, dans le graphique ci-dessus, chaque point représente une personne différente. Elle est placée en fonction de ses caractéristiques.

régression

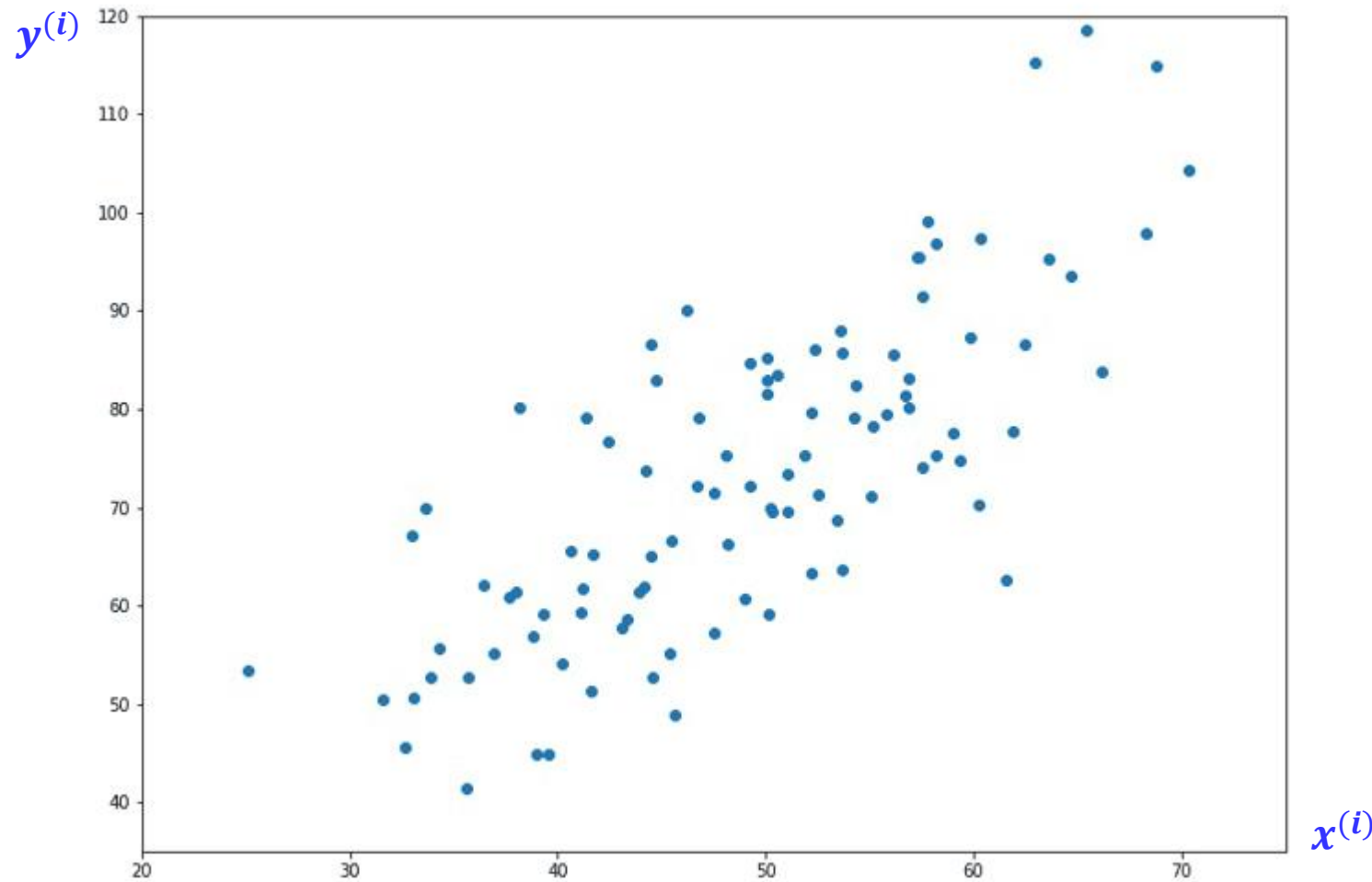


Le deuxième type de problème supervisé est la régression. La seule différence avec la classification est que la target est continue. Il consiste donc à prédire la valeur associée à un sample en fonction de ses features. Dans l'exemple ci-dessus, l'espace des features est à 1 dimension (il n'y a qu'une caractéristique, en abscisse) et l'axe des ordonnées représente la target. Ainsi, la fameuse régression linéaire utilisée en physique depuis votre tendre enfance est en réalité un problème de machine learning.

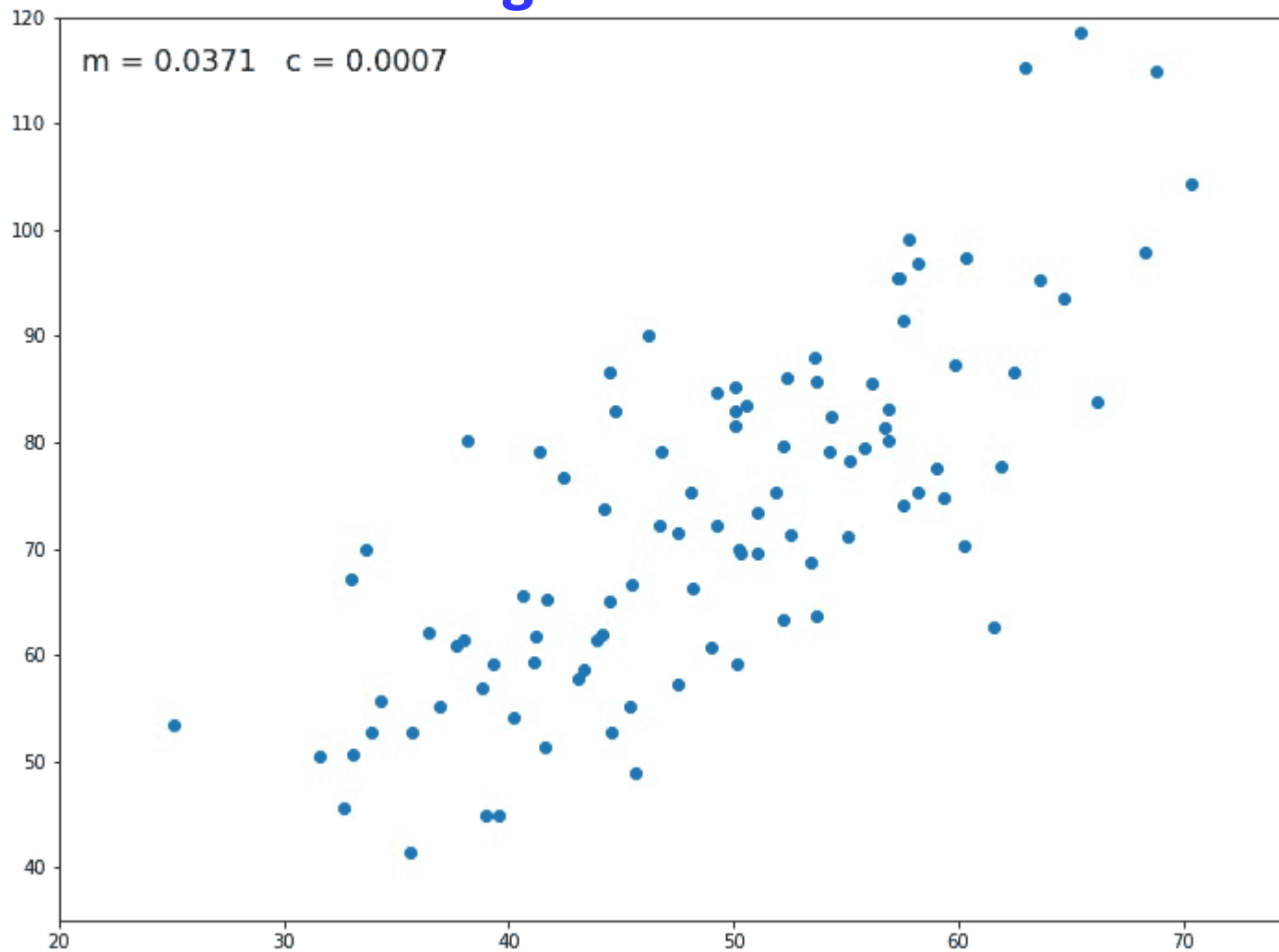
Pour résoudre un problème **d'apprentissage supervisé**, il faut procéder en **4 étapes** :

1. Disposer d'un Dataset (x, y) où x sont les **features** et y est la **target**,
2. Développer un Modèle dont les paramètres sont à trouver,
3. Définir une Fonction Coût qui mesure l'erreur entre le modèle et les valeurs de y du Dataset,
4. Utiliser un Algorithme d'Apprentissage qui cherche le modèle (en réalité les paramètres) qui minimisent la Fonction Coût, c'est-à-dire qui nous donne le modèle aux plus petites erreurs.

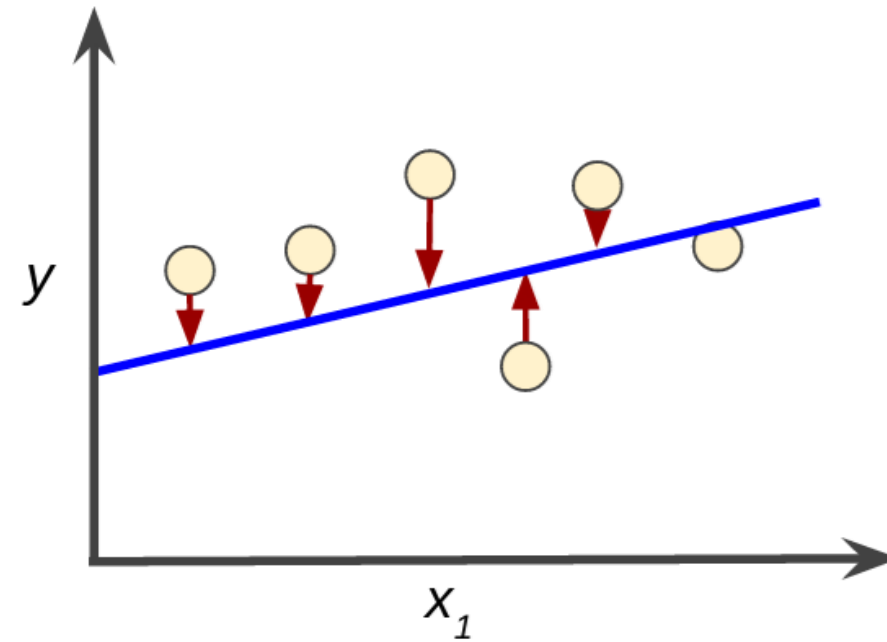
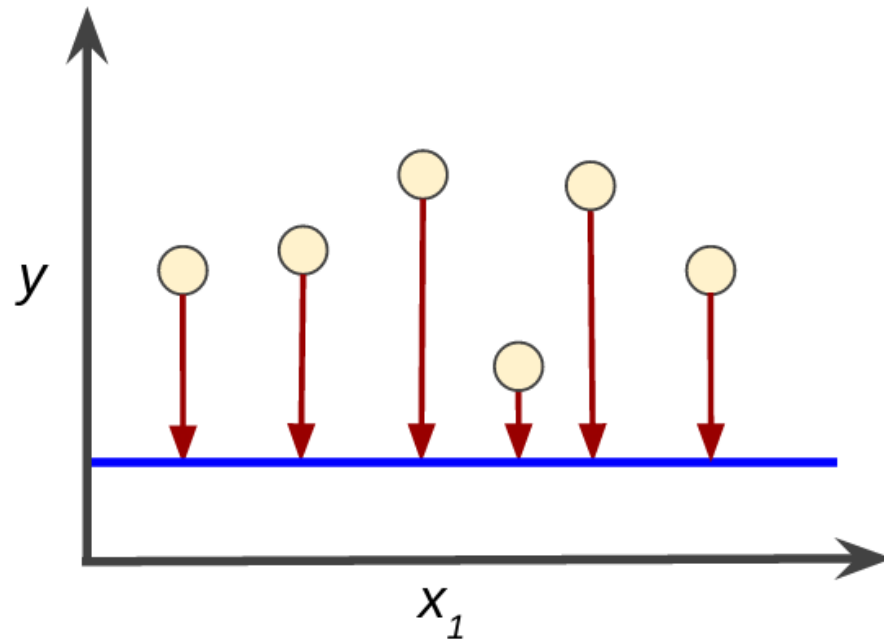
Supposons qu'on a un Dataset avec m exemples et $n = 1$ « feature » variable.



Régression linéaire



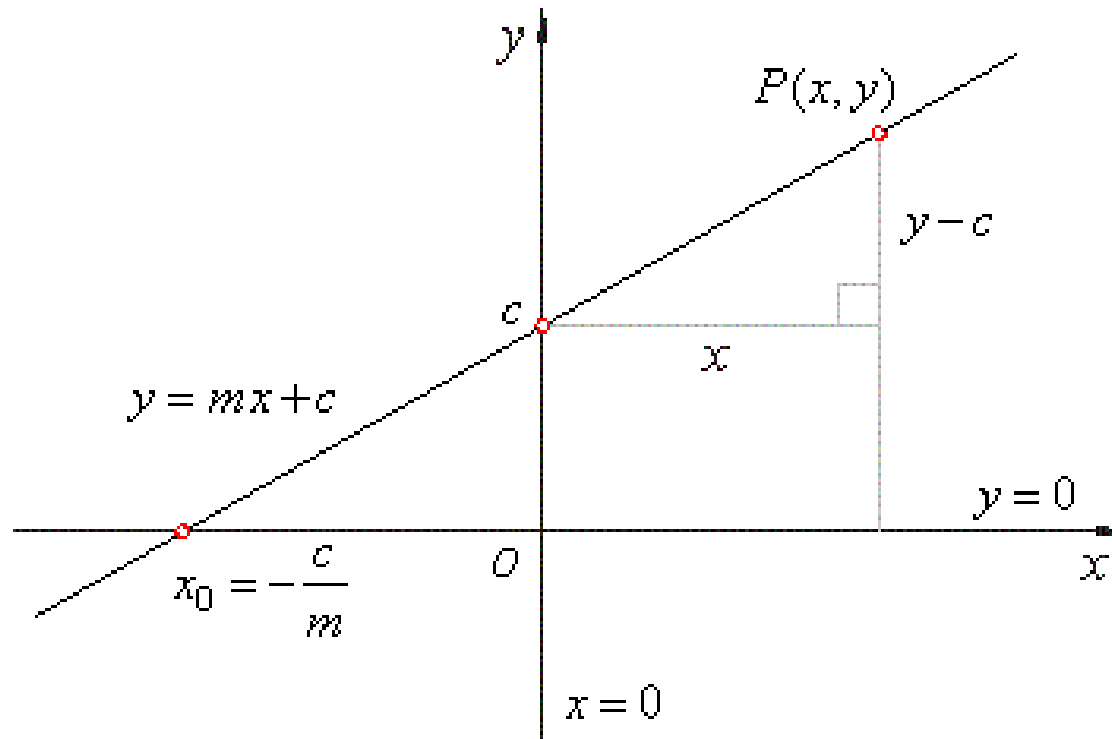
Les valeurs de m et c sont mises à jour à chaque itération pour obtenir la solution optimale.



Perte élevée dans le modèle de gauche ; perte faible dans le modèle de droite.



En statistique, la régression linéaire est une approche linéaire de la modélisation de la relation entre une variable dépendante et une ou plusieurs variables indépendantes. Soit **X** la variable indépendante et **Y** la variable dépendante. Nous allons définir une relation linéaire entre ces deux variables comme suit :



C'est l'équation d'une ligne que vous avez étudiée au lycée. a est la pente de la ligne et b est l'interception y .

Nous allons utiliser cette équation pour entraîner notre modèle avec un ensemble de données donné et prédire la valeur de Y pour toute valeur donnée de X .

Notre défi est de déterminer la valeur de a et b , de telle sorte que la droite correspondante à ces valeurs soit la droite la mieux ajustée qui donne l'erreur minimale.

Fonction coût (fonction de perte)

La perte ou le coût est l'erreur dans notre valeur prédite de a et b . Notre objectif est de minimiser cette erreur pour obtenir la valeur de a et b la plus précise possible.

Nous utiliserons la fonction d'erreur quadratique moyenne pour calculer la perte. Cette fonction comporte trois étapes :

1. Trouver la différence entre la valeur réelle y et la valeur prévue $f(x)$ ($f(x) = ax + b$), pour un x donné.
2. Placez cette différence au carré.
3. Trouvez la moyenne des carrés pour chaque valeur de X .

Ici, $y^{(i)}$ est la valeur réelle et $f(x^{(i)})$ est la valeur prévue.

$$J(a, b) = \frac{1}{2m} \sum_{i=1}^m (f(x^{(i)}) - y^{(i)})^2, m \text{ est la taille du dataset.}$$

On élève donc l'erreur au carré et on trouve la moyenne, d'où le nom **d'erreur quadratique moyenne**. Maintenant que nous avons défini la fonction de perte, entrons dans la partie intéressante - la minimiser et trouver m et c.

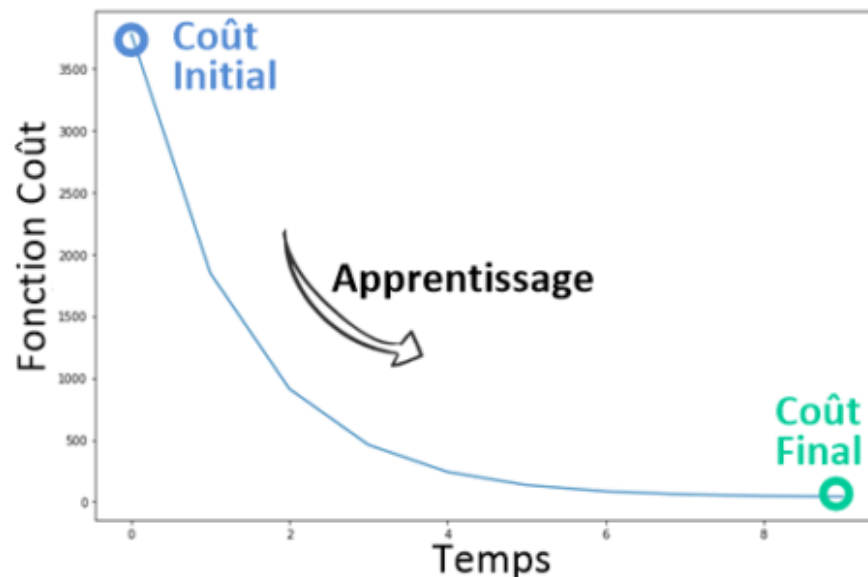
La descente de gradient

La Descente de Gradient, (ou **Gradient Descent** en anglais) est un des algorithmes les plus importants de tout le **Machine Learning** et de tout le **Deep Learning**. il s'agit d'un algorithme d'optimisation extrêmement puissant qui permet d'entraîner les modèles de régression linéaire, régression logistiques ou encore les réseaux de neurones. Si vous vous lancez dans le Machine Learning, il est donc impératif de comprendre en profondeur l'algorithme de la descente de gradient.

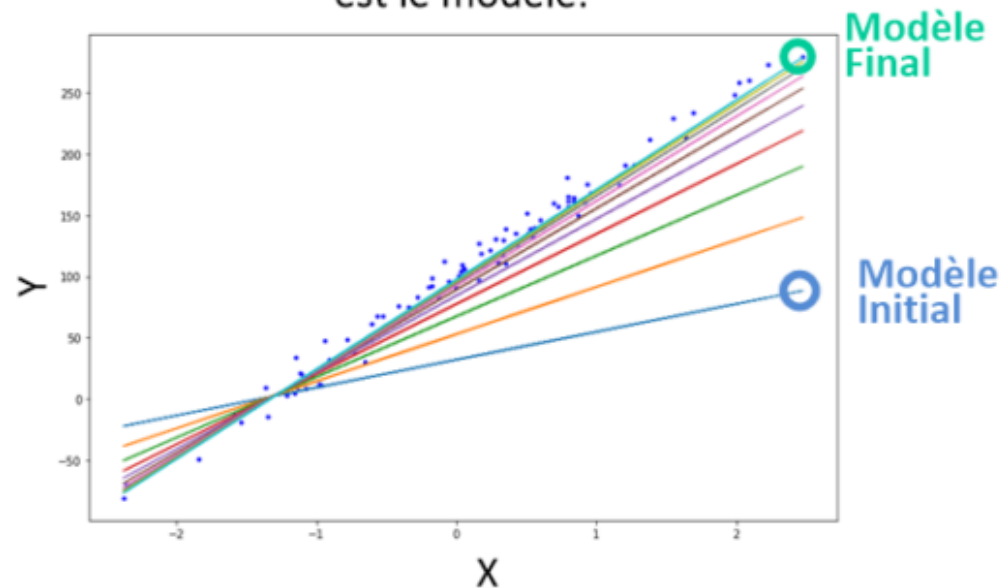
L'algorithme de descente de gradient est un algorithme itératif ayant comme but de trouver les valeurs optimales des paramètres d'une fonction donnée. **Il tente d'ajuster ces paramètres afin de minimiser la sortie d'une fonction de coût face à un certain jeux de données.** Cet algorithme est souvent utilisé en apprentissage machine dans le cadre de régressions non linéaires puisqu'il permet de rapidement trouver une solution approximative à des problèmes très complexes.

L'algorithme de la descente de gradient est une solution pour une régression linéaire simple. Bien qu'il existe une solution analytique optimale à la régression linéaire simple, la simplicité de ce problème en fait un excellent candidat pour démontrer l'application de l'algorithme du gradient.

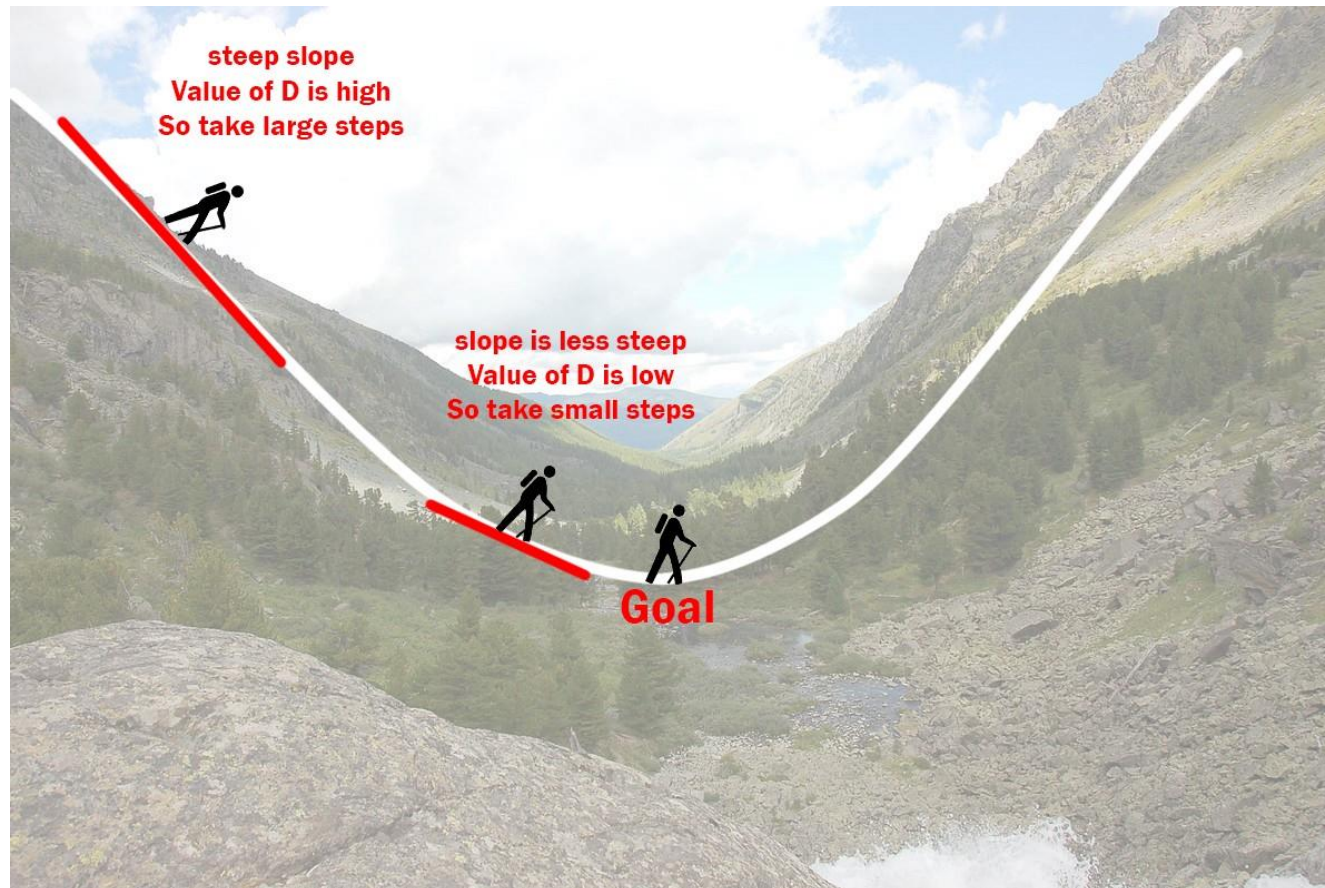
Minimisation de la Fonction Coût



Plus la Fonction Coût est faible, meilleur est le modèle.



Imaginez une vallée et une personne qui n'a aucun sens de l'orientation et qui veut aller au fond de la vallée. Il descend la pente et fait de grands pas quand la pente est raide et de petits pas quand la pente est moins raide. Il décide de sa prochaine position en fonction de sa position actuelle et s'arrête lorsqu'il atteint le fond de la vallée qui était son objectif. Essayons d'appliquer la descente de la pente à **a** et **b** et approchons-la pas à pas :



La descente de gradient est une méthode à direction de descente et recherche linéaire, qui cherche à résoudre le problème sans contrainte

$$\min\{f(x), x \in \mathbb{R}^n\}.$$

Le principe de la méthode est de construire une suite $\{x^{(k)}\}_{k \in \mathbb{N}}$ telle que :

$$x^{k+1} = x^k + \alpha p_k,$$

avec bien sur la direction $p_k = -\nabla f(x^k)$ et $\alpha \in \mathbb{R}_+^*$, le pas α est appelé le taux d'apprentissage (**learning rate** en anglais)

La suite $\{x^{(k)}\}_{k \in \mathbb{N}}$ qui va converger vers le minimum de f .

Appliquons ça à notre problème :

$$\min\{J(a, b), (a, b) \in \mathbb{R}^2\}$$
$$J(a, b) = \frac{1}{2m} \sum_{i=1}^m (f(x^{(i)}) - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^m ((ax^{(i)} + b) - y^{(i)})^2$$

Forme matricielle

Soient $\mathbf{y} = (y^{(1)}, \dots, y^{(m)})^T \in \mathbb{R}^m$ et $X = \begin{bmatrix} x^{(1)} & 1 \\ \vdots & \vdots \\ x^{(m)} & 1 \end{bmatrix} \in M_{m \times 2}(\mathbb{R})$, (En général $X \in M_{m \times (n+1)}(\mathbb{R})$)

$\theta = (a, b)^T$ est de dimension 2, (En général $\theta \in \mathbb{R}^{n+1}$ où n est le nombre de variables (features)).

Le modèle :

$$F = ax^{(i)} + b = \begin{bmatrix} x^{(1)} & 1 \\ \vdots & \vdots \\ x^{(m)} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = X\theta$$


Colonne biais

La fonction coût :

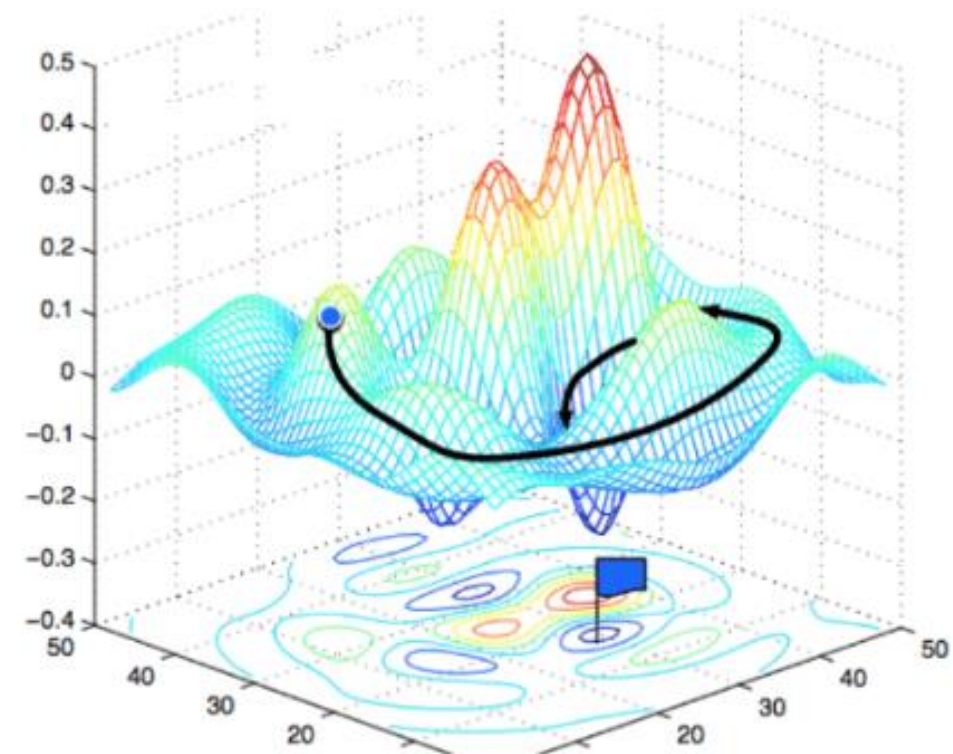
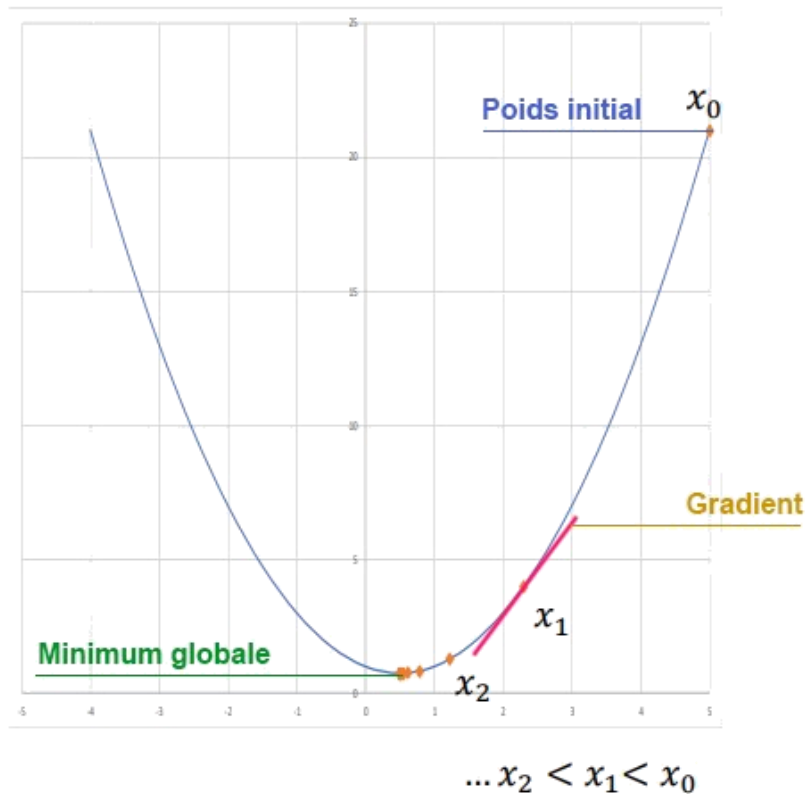
$$J(a, b) = \frac{1}{2m} \sum_{i=1}^m ((ax^{(i)} + b) - y^{(i)})^2 \Leftrightarrow J(\theta) = \frac{1}{2m} (X\theta - y)^2$$

Le gradient :

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

La descente de gradient :

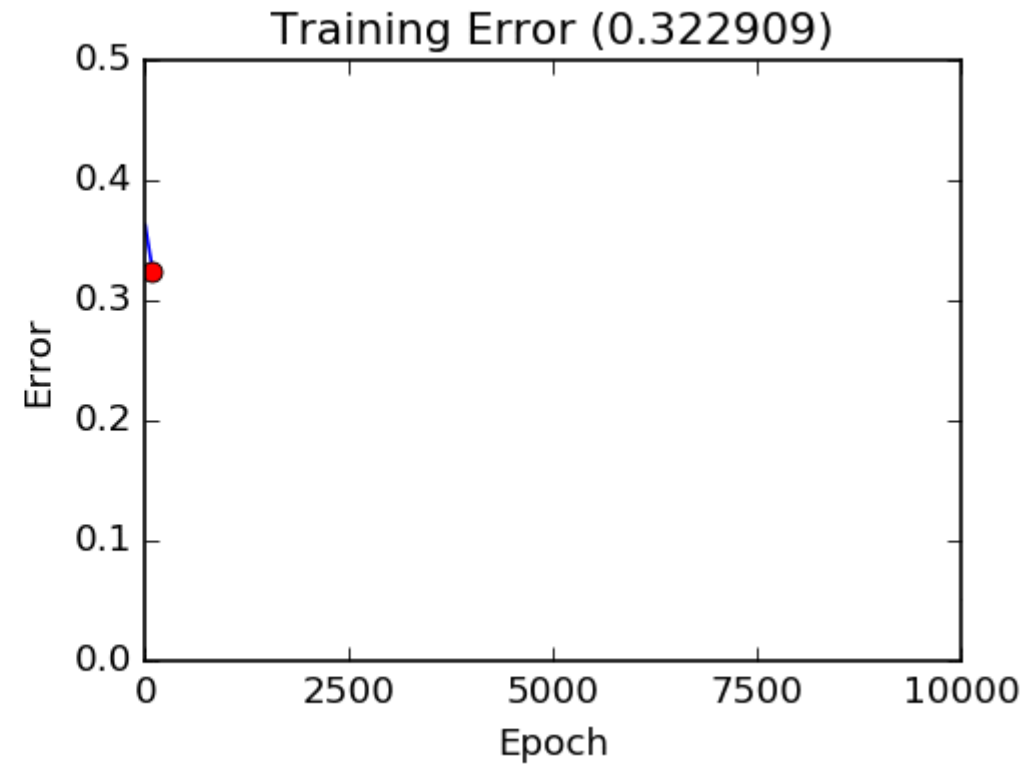
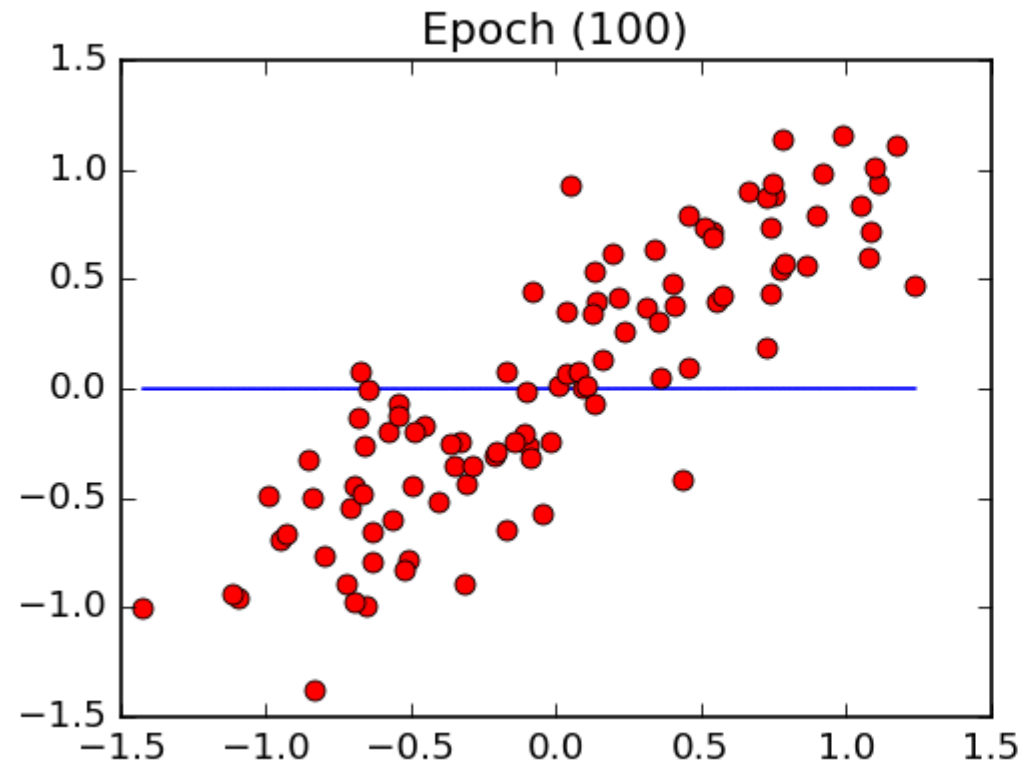
$$\theta^{i+1} = \theta^i - \alpha \nabla J(\theta^i) = \theta^i - \alpha \frac{\partial J(\theta^i)}{\partial \theta^i}$$



Nous répétons ce processus jusqu'à ce que notre fonction de perte soit une très petite valeur ou idéalement 0 (ce qui signifie une erreur de 0 ou une précision de 100 %). Les valeurs de a et b qu'il nous reste maintenant seront les valeurs optimales.

Pour revenir à notre analogie, θ peut être considéré comme la position actuelle de la personne. $\nabla J(\theta)$ est l'équivalent de la pente et α peut être la vitesse à laquelle elle se déplace. Maintenant, la nouvelle valeur de θ sera sa prochaine position. Lorsque la pente est plus raide (gradient élevé), il fait des pas plus longs et lorsqu'elle est moins raide (gradient faible), il fait des pas plus petits. Enfin, il arrive au fond de la vallée, ce qui correspond à notre perte = 0.

Maintenant, avec la valeur optimale de a et b , notre modèle est prêt à faire des prédictions !



Learning rate

Par contre, avec un taux d'apprentissage α fixé, rien ne garantit la convergence de l'algorithme qui peut osciller autour du minimum indéfiniment.

- Plus la valeur α est grande, plus on va avancer vite, mais l'algorithme risque de ne jamais converger.
- Plus la valeur α est petite, plus on va avancer lentement, et donc plus ça va être long de converger.

