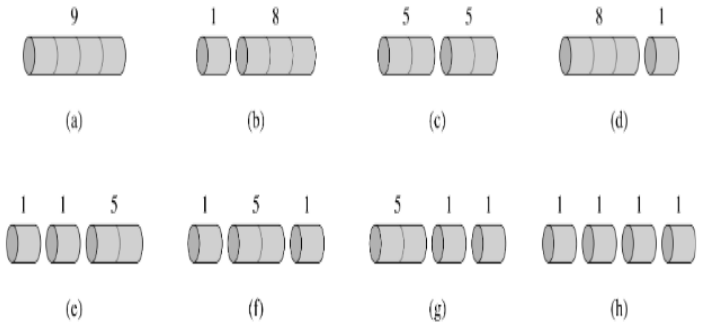
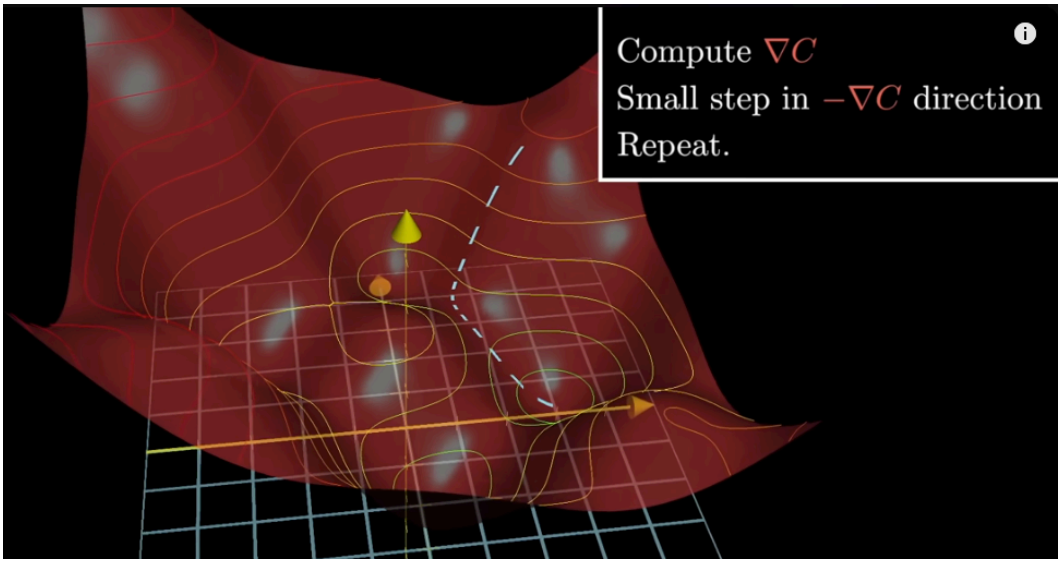
Optimization

Gradient descent

- on a une fonction de loss, mais on ne peut pas trouver son minimum directement
- on dérive, puis on calcule $\text{step\_size} = \text{dérivée} * \text{learning rate}$
- on se déplace dans cette direction

Comme ça, on fait des "baby step" quand on est proche de la solution, puis de grosses steps quand la dérivée est élevée. On s'arrête quand la step size est très proche de zéro.

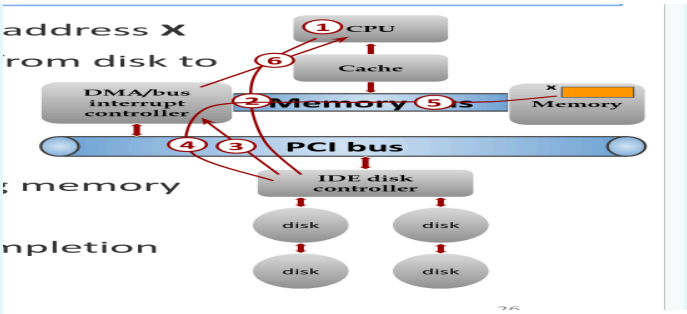


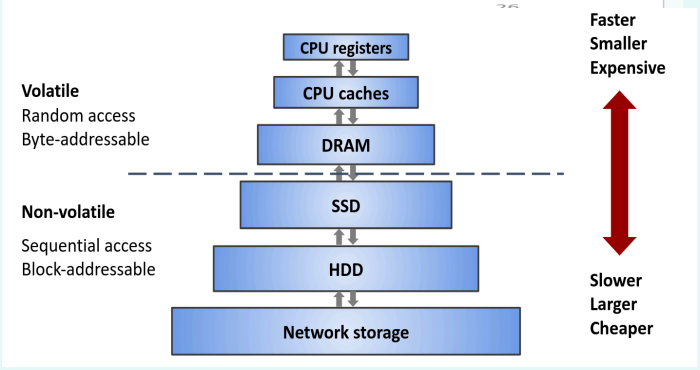


exemple d'une cost function avec 2 weights, on essaye de trouver le minimum
la cost function est donc une fonction qui prend les weights en entrées, la run sur le modele et donne un nombre (l'erreur) pour chaque paire.

Hyperparamètre à bien choisir : le learning rate

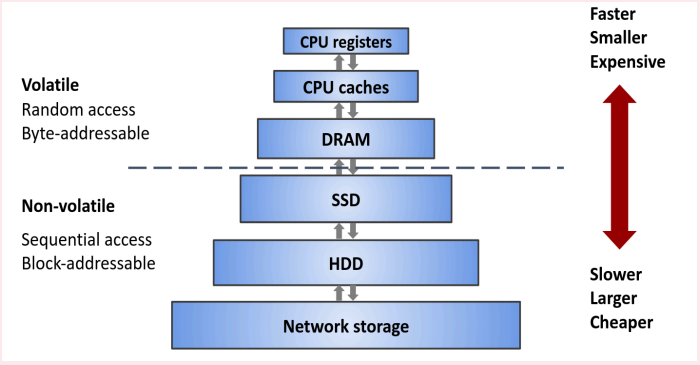
si trop petit, ça va prendre trop de temps, si trop grand, ça va diverger





Pour trouver la bonne learning rate, on utilise Grid search.

La descente de gradient ne donne pas toujours la bonne solution!



Dans certains cas, on ne va jamais converger vers la bonne solution! Heureusement, la fonction de loss pour une linear regression est convexe (comme x^2)! donc on trouvera la bonne solution.

Normaliser les features pour gagner du temps!

where feature 1 has much smaller values than feature 2 (on the right).

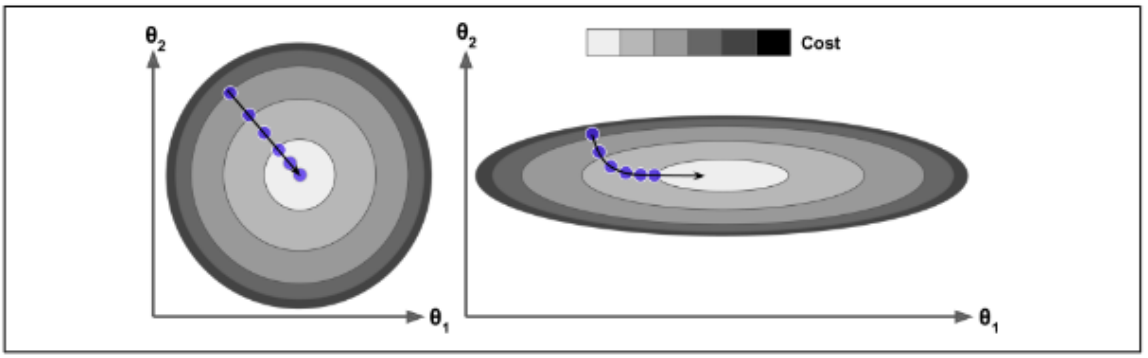


Figure 4-7. Gradient Descent with (left) and without (right) feature scaling

En fonction du scaling des features, on peut arriver plus ou moins vite au résultat (voir l'image).

La fonction n'a pas besoin d'être différentiable partout, juste la ou on veut l'optimiser.

Batch (or full) gradient descent

Comment l'implémenter ?

Equation 4-5. Partial derivatives of the cost function

$$\frac{\partial}{\partial \theta_j}\text{MSE}(\boldsymbol{\theta}) = \frac{2}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

$$y_{\text{pred}}^{(i)} = a_0 + \sum_{j=1}^{d} a_j x_{ij}$$

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}\left(y_{\text{pred}}^{(i)} - y_{\text{true}}^{(i)}\right)^2$$

$$\varepsilon^{(i)} = y_{\text{pred}}^{(i)} - y_{\text{true}}^{(i)}$$

$$\text{MSE}(a_0, \ldots, a_d) = \frac{1}{m}\sum_{i=1}^{m}\left(\varepsilon^{(i)}\right)^2$$

$$\frac{\partial \text{MSE}}{\partial a_k} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial a_k}\left[\left(\varepsilon^{(i)}\right)^2\right]$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left[2\varepsilon^{(i)} \cdot \frac{\partial \varepsilon^{(i)}}{\partial a_k}\right]$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(2\varepsilon^{(i)} \cdot x_{ik}\right)$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(2 \left(y_{\text{pred}}^{(i)} - y_{\text{true}}^{(i)} \right) x_{ik} \right)$$

On calcule la dérivée partielle par rapport a chaque parametre θ. (la formule vient de la dérivée de MSE).

C'est lent ! on doit multiplier une matrice de la taille des samples de training

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} 1 \\ 0.5 \\ -1 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix}$$

Conjugate Gradient Method

⑦ Comment ça marche ?

Le problème :

- on veut minimiser une fonction $f(x)$
- la plupart du temps, f est une fonction quadratique, de la forme $f(x) = \frac{1}{2}x^T Ax - b^T x$, où A est une matrice symétrique.
- minimiser f revient à résoudre le système linéaire $Ax = b$.

L'idée :
On ne peut pas calculer directement A^{-1}, on préfère un procédé itératif notamment quand A est de grande dimension. La conjugate gradient method fait deux choses :

- elle suit la pente, comme le gradient descent, pour trouver le minimum
- elle évite de rebondir inutilement entre les bords de l'ellipse

En pratique :

On définit $g_0 = -\nabla f\left(x_0\right)$ et $g_{k+1} = -\nabla f\left(x_{k+1}\right) + \beta_k g_k$ pour garder en mémoire d'où on vient et ne pas prendre une direction orthogonale.

$$\beta_k \;=\; \frac{\|\nabla f(x_{k+1})\|_2^2}{\|\nabla f(x_k)\|_2^2}, \quad \text{avec } \|\nabla f(x)\|_2^2 \;=\; (\nabla f(x))^T\,(\nabla f(x)).$$

$$\alpha_k \;=\; \arg\min_{\alpha} f\big(x_k + \alpha\, g_k\big) \;=\; \frac{(\nabla f(x_k))^T\,\nabla f(x_k)}{g_k^T\,A\,g_k}.$$

$$x_{k+1} \;=\; x_k \;+\; \alpha_k\, g_k.$$

Preuve:

donne le layout de la preuve sans les détails, je suis nul en analyse

Voici un plan succinct de la preuve, sans entrer dans les calculs détaillés :

1. **Définir la fonction scalaire**
   Poser $\varphi(\alpha) = f(x_k + \alpha\, g_k)$, où $f(x) = \frac{1}{2}x^T Ax - b^T x$.
2. **Développer $\varphi(\alpha)$**
   Écrire $\varphi(\alpha)$ en fonction de α (terme constant, terme linéaire en α, terme quadratique en α).
3. **Calculer la dérivée**
   Poser $\varphi'(\alpha)$ et exprimer cette dérivée en termes de $g_k^T \nabla f(x_k)$ et de $g_k^T A\, g_k$.
4. **Trouver le minimiseur**
   Imposer $\varphi'(\alpha_k) = 0$ pour en déduire

$$\alpha_k = -\frac{g_k^T \nabla f(x_k)}{g_k^T A\, g_k}.$$

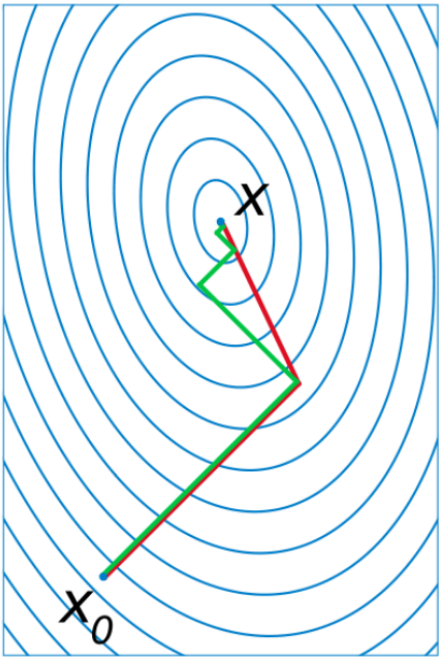
5. **Utiliser la relation entre g_k et le résidu**
   Rappeler que, dans la méthode des gradients conjugués, on a g_k « aligné » avec le résidu $r_k = -\nabla f(x_k)$, d'où

$$-g_k^T \nabla f(x_k) = \|\nabla f(x_k)\|_2^2.$$

6. **Conclure la formule**
   Remplacer dans l'expression obtenue pour α_k afin d'obtenir

$$\alpha_k = \frac{\|\nabla f(x_k)\|_2^2}{g_k^T A\, g_k}.$$

C'est tout : on a juste suivi ces étapes, en passant de la définition de $\varphi(\alpha)$ à la condition de minimisation, puis en exploitant la propriété de conjuguaison qui lie q_k au résidu.



Lagrangian optimization

Comment ça marche ?

On veut trouver le x qui minimise $f(x)$.

On a des contraintes $g_i(x) \leq 0$, pour $i = 1, \ldots, M$.
Et des contraintes $h_i(x) = 0$, pour $i = 1, \ldots, P$.

On calcule le lagrangien :

$$L(x, \lambda, \nu) = f(x) + \sum_{i=1}^{M} \lambda_i g_i(x) + \sum_{i=1}^{P} \nu_i h_i(x)$$

avec $\lambda_i \geq 0$ et $\nu_j \in \mathbb{R}$.

On utilise le Lagrangien pour construire les conditions de Karush-Kuhn-Tucker (KKT):

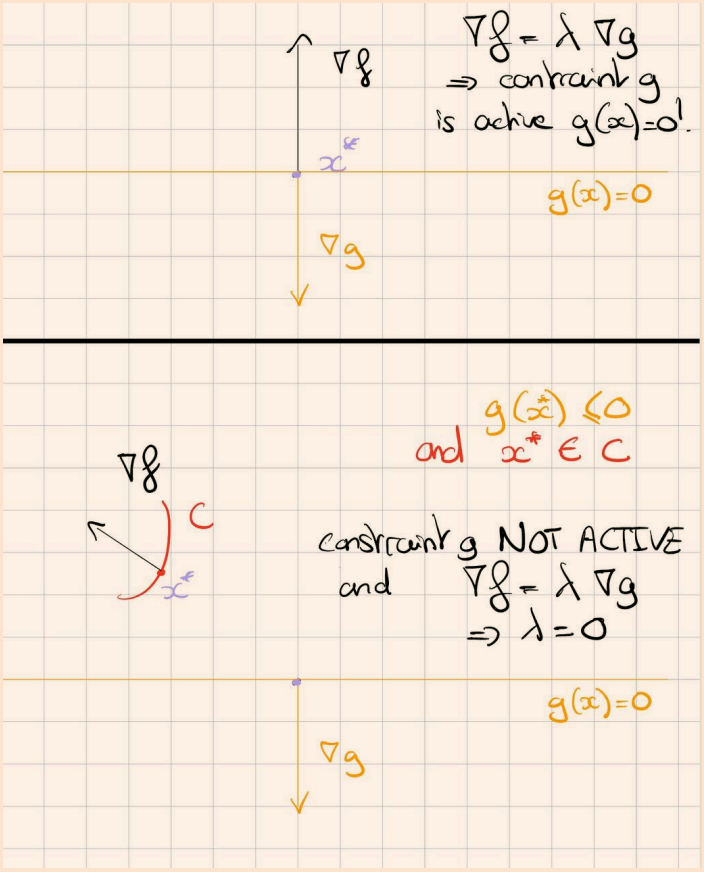
- $\nabla_x L\left(x^*, \lambda^*, \nu^*\right) = 0$ car on veut avoir $\nabla f(x) = -\lambda_1 \nabla g_1 + \ldots$ etc. (comme toujours le gradient de la fonction et de la contrainte doivent être paralleles au min/max -- penser a rejoindre le soleil sous un lac gelé)
- $g_i\left(x^*\right) \leq 0$ (dans tous les cas, c'est notre contrainte de base)
- $h_i\left(x^*\right) = 0$ (pareil)
- $\lambda_i^* \geq 0$ (seulement les λ pas les ν)

  > **Pourquoi ?**
  > en fait cette contrainte vient du fait qu'on veut que le gradient de f et des g soient non seulement paralleles, mais plus précisément **opposés** (par exemple si le gradient de f est vers le haut, on veut minimiser f et donc aller vers le bas, mais le gradient de g est vers le bas, donc si on descend plus on ne satisfait plus la contrainte $g \geq 0$!). Si ce n'était pas le cas, on pourrait trouver un meilleur minimum!

- $\lambda_i^* g_i\left(x^*\right) = 0$

  > **Pourquoi ?**
  > Voir plus haut. Soit on est dans le cas ou effectivement cette contrainte g est active, on la pousse au bout du bout jusqu'a ce que le gradient de f soit égale a quelque chose fois le gradient de g soit la contrainte n'est pas active (car

une autre est plus forte par exemple).



Version duale :

Elle est plus simple à résoudre.

En fait notre objectif c'est de pénaliser notre fonction $f(x)$ convexe quand une contrainte est violée.

On pourrait faire quelque chose comme $P(y) = 0 \text{ si pas violée}, \infty \text{ sinon}$. Le problème c'est que c'est difficile à résoudre.

Sinon on peut faire $P(y) = \max_{u \geq 0} u \cdot y$. Du coup si $y = 0$ ou $y \leq 0$, ce sera zéro, et sinon ce sera l'infini ! (on va toujours prendre le meilleur u).

On a donc $\min_x \max_{u \geq 0} E(x) = f(x) + u \cdot (g(x))$. mais on peut très bien échanger l'ordre des deux contraintes !

$$g(\lambda, \nu) = \inf_{\mathbf{x}} L(\mathbf{x}, \lambda, \nu)$$
$$= \inf_{\mathbf{x}} \left(f(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^{P} \nu_i h_i(\mathbf{x}) \right)$$

- $g(\lambda, \nu)$ is a concave function, i.e. opposite of convex, single maximum also.
- We can find λ and ν by maximizing g:

$$\lambda^*, \nu^* = argmax_{\lambda,\nu}\ g(\lambda, \nu)$$
$$x^* = argmin_{\mathbf{x}}\ L(\mathbf{x}, \lambda^*, \nu^*)$$