Chapitre 14 : Optimisation Numérique

26 février 2026

Plan du chapitre

Le cœur du Machine Learning : L'Optimisation

La quasi-totalité des problèmes d'apprentissage supervisé se ramènent à un problème d'optimisation :

$$\mathbf{w}^* = \underset{\mathbf{w} \in \mathbb{R}^d}{\arg\min}\, J(\mathbf{w})$$

Où $J(\mathbf{w})$ est le risque empirique (souvent régularisé) sur les données.

Sauf dans des cas très simples (ex : Moindres Carrés Ordinaires), il n'existe pas de solution analytique fermée (pas de formule magique). **Il faut utiliser des algorithmes itératifs.**

Le Graal de l'optimisation : La Convexité

Fonction Convexe

Une fonction f est convexe si, pour tout segment reliant deux points de sa courbe, la courbe se situe en dessous du segment :

$$f(\theta \mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y})$$

Pour une fonction C^2, cela équivaut à avoir une matrice Hessienne $\nabla^2 f(\mathbf{x})$ semi-définie positive.

Théorème Fondamental

Pour une fonction convexe, **tout minimum local est un minimum global**. La condition d'optimalité du premier ordre est simplement : $\nabla f(\mathbf{x}^*) = 0$.

Exemples convexes : Régression linéaire, Régression logistique, SVM.

Exemples non-convexes : Réseaux de Neurones.

La Descente de Gradient (Gradient Descent)

Le gradient $\nabla J(\mathbf{w})$ pointe vers la plus forte *montée*.

Pour minimiser J, il faut aller dans la direction opposée : $-\nabla J(\mathbf{w})$.

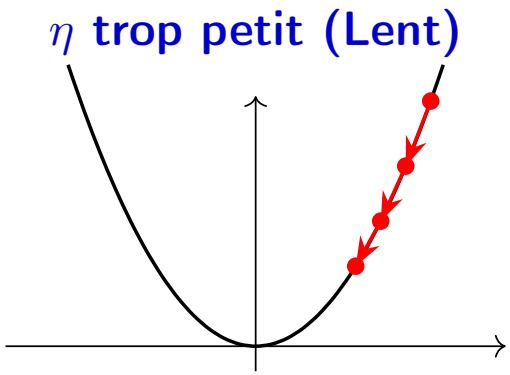
Algorithme (Batch Gradient Descent)

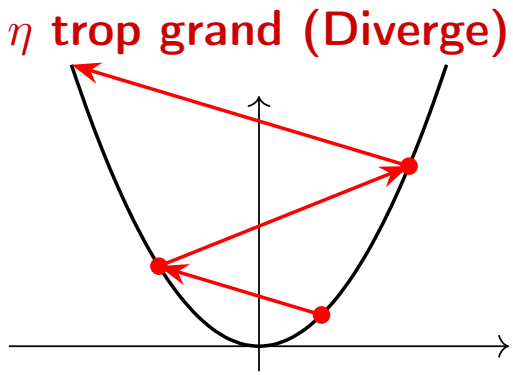
1. Initialiser $\mathbf{w}^{(0)}$ aléatoirement.
2. Répéter jusqu'à convergence :

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla J(\mathbf{w}^{(t)})$$

$\eta > 0$ est le **pas d'apprentissage** (learning rate).

Le rôle critique du Pas d'Apprentissage (η)





Le problème du passage à l'échelle

Le gradient complet est la moyenne des gradients de **tous** les exemples :

$$\nabla J(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \ell(h_{\mathbf{w}}(\mathbf{x}_i), y_i)$$

Si le jeu de données compte 1 million d'exemples ($n = 10^6$), faire **un seul pas** nécessite 1 million de calculs de gradients. C'est beaucoup trop lent.

Descente de Gradient Stochastique (SGD)

On estime le gradient à partir d'**un seul exemple** i tiré au hasard à chaque itération :

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla_{\mathbf{w}} \ell(h_{\mathbf{w}^{(t)}}(\mathbf{x}_i), y_i)$$

Mini-Batch SGD et Bruit

Le SGD pur a une trajectoire très erratique (bruitée) car chaque exemple "tire" dans sa propre direction.

Le compromis parfait : Mini-Batch SGD

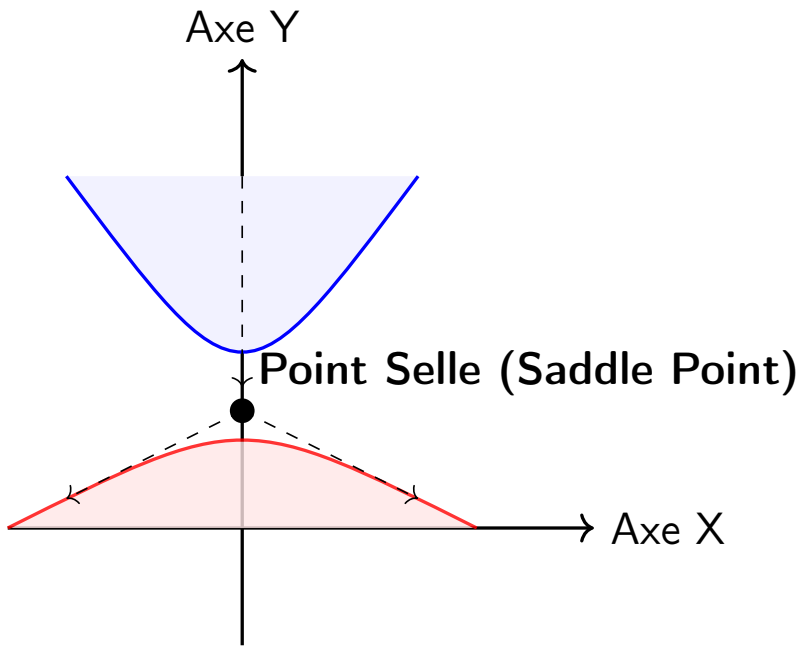
En pratique, on calcule le gradient sur un petit lot d'exemples (*mini-batch*, ex : 32, 64, 256).

- Réduit la variance du gradient par rapport au SGD pur.
- Permet la **vectorisation matérielle** (calculs matriciels massifs parallélisés sur GPU).

Pour garantir la convergence stochastique, le pas η_t doit décroître au cours du temps (Conditions de Robbins-Monro).

La Non-Convexité (Réseaux de Neurones)

L'espace des poids des réseaux de neurones (Deep Learning) est hautement non-convexe. Il foisonne d'obstacles.



En très haute dimension, le principal piège n'est pas le minimum local, mais le **point selle** (gradient nul, minimum dans une direction, mais maximum dans une autre).
Le bruit du SGD aide à s'en échapper.

Les Optimiseurs Modernes : Momentum et RMSProp

Accélérer le SGD et franchir les points selles :

1. L'Élan (Momentum)

On simule une "boule qui roule". On accumule les gradients passés pour garder de la vitesse dans les directions constantes et amortir les oscillations.

$$V_t = \beta V_{t-1} + (1-\beta)\nabla J(\mathbf{w})$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta V_t$$

2. RMSProp (Root Mean Square Propagation)

Ajuste le pas d'apprentissage **pour chaque paramètre individuellement**. Il divise le gradient par la moyenne quadratique des gradients récents. (Avance vite sur les pentes douces, ralentit sur les pentes abruptes).

Adam

L'optimiseur **Adam** (Adaptive Moment Estimation, Kingma & Ba 2014) combine le meilleur des deux mondes : le Momentum et RMSProp.

- Il calcule la moyenne exponentielle glissante du gradient (m_t, Momentum).
- Il calcule la moyenne exponentielle glissante du gradient au carré (v_t, RMSProp).
- Il corrige les biais d'initialisation vers zéro ($\hat{m}_t, \hat{v}_t$).

Mise à jour finale d'Adam

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

C'est aujourd'hui l'optimiseur par défaut (Souvent : $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$) pour quasiment tous les réseaux de neurones.