Algorithmique : la boucle d'entraînement

Ici, aucune simplification : entraîner une IA, c'est exécuter une boucle

À savoir avant de commencer — un document à part

Contrairement à plusieurs documents précédents, celui-ci ne contient aucune approximation à corriger en fin de parcours : le code présenté ici (en pseudocode proche du Python) est très exactement, à la simplification du problème mathématique près (document 0), ce qu'exécute un ordinateur pour entraîner une IA.

Niveau : Première / Terminale — enseignement transversal d'algorithmique et programmation

Introduction : l'entraînement, ce n'est qu'une boucle

Le document 0 a présenté la descente de gradient à travers des calculs faits « à la main », itération après itération. En réalité, ces calculs sont exécutés par un programme informatique, qui répète les mêmes instructions un grand nombre de fois : c'est très exactement une boucle, une des notions centrales de l'algorithmique au programme du lycée.

Ce document traduit, en pseudocode proche du langage Python, exactement ce que tu as déjà calculé à la main dans le document 0.

Partie A — Rappels d'algorithmique

Niveau : Seconde / Première

Rappels de cours

Boucle « pour » (for) : répète un bloc d'instructions un nombre de fois connu à l'avance.

Boucle « tant que » (while) : répète un bloc d'instructions tant qu'une condition reste vraie (le nombre de répétitions n'est pas forcément connu à l'avance).

Affectation (←, ou = en Python) : donne (ou remplace) la valeur d'une variable.

On considère l'algorithme suivant :

Algorithme 1

```
s ← 0
pour i allant de 1 à 5 :
    s ← s + i
retourner s
```

1. Exécute cet algorithme « à la main » (trace), en donnant la valeur de s après chaque tour de boucle. Que retourne l'algorithme au final ?
2. Réécris cet algorithme avec une boucle « tant que » plutôt qu'une boucle « pour », en explicitant la condition d'arrêt et la variable qui doit être mise à jour pour que la boucle se termine.

Partie B — Traduire la descente de gradient en algorithme

Niveau : Première / Terminale

On reprend l'exemple du document 0 : $f(x) = x^2 - 4x + 5$, $f'(x) = 2x - 4$.

Algorithme 2 — Descente de gradient (nombre d'itérations fixé)

```
fonction descente_gradient(x0, alpha, n_iterations) :
    x ← x0
    pour i allant de 1 à n_iterations :
        derivee ← 2x − 4
        x ← x − alpha × derivee
    retourner x
```

3. Exécute « à la main » l'algorithme 2 avec x0=5, alpha=0,1 et n_iterations=3 : donne la valeur de x après chaque tour de boucle. Compare ton résultat avec les calculs du document 0 (partie C) : retrouves-tu exactement les mêmes valeurs ?

4. Modifie l'algorithme 2 pour qu'il conserve, dans une liste nommée historique, toutes les valeurs successives de x (y compris x0), et qu'il retourne cette liste entière plutôt que la seule valeur finale.

Voici la traduction de l'algorithme 2 en Python :

Code Python

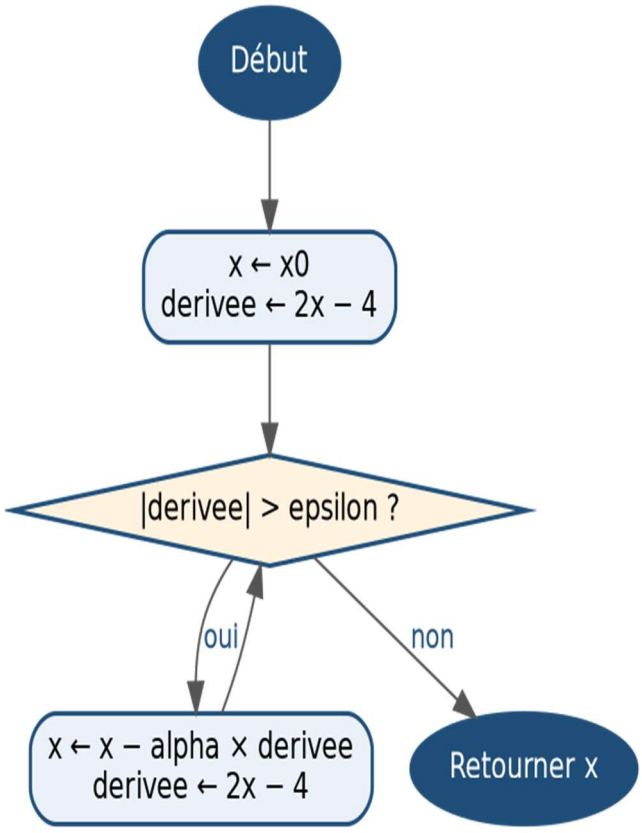
```
def descente_gradient(x0, alpha, n_iterations):
    x = x0
    for i in range(n_iterations):
        derivee = 2*x - 4
        x = x - alpha * derivee
    return x
```

5. Dans ce code, à quoi correspond exactement range(n_iterations) ? Combien de fois le contenu de la boucle est-il exécuté si n_iterations=3 ?

Partie C — S'arrêter dès que c'est suffisant : la boucle « tant que »

Niveau : Terminale

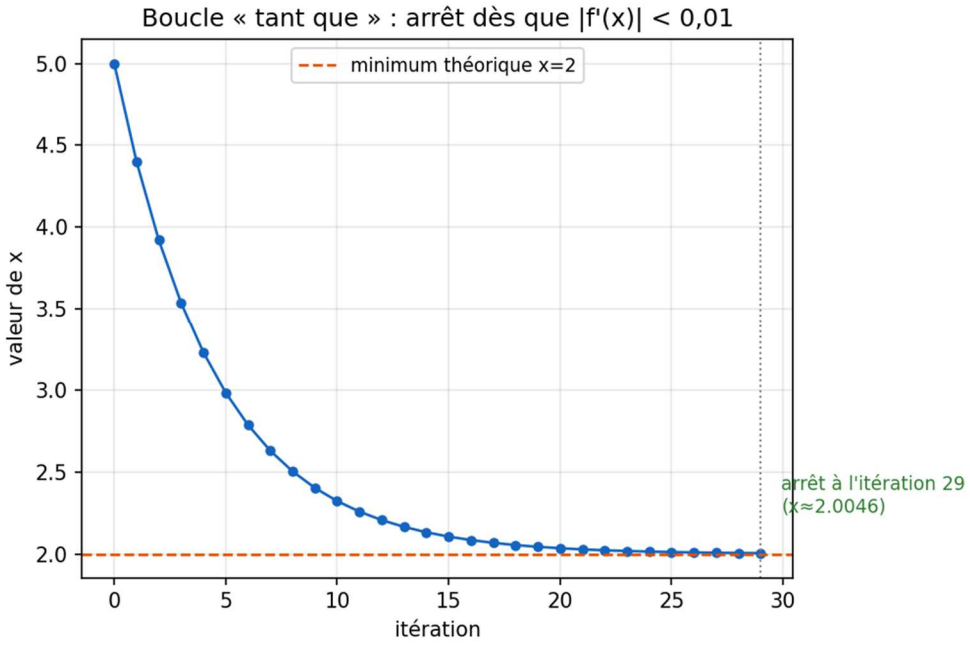
Plutôt que de fixer à l'avance un nombre d'itérations, on peut arrêter la descente de gradient dès que la dérivée devient suffisamment proche de 0 (le critère de convergence utilisé, en pratique, par de nombreux algorithmes d'apprentissage) :



Code Python — arrêt automatique

```
def descente_gradient_arret(x0, alpha, epsilon):
    x = x0
    derivee = 2*x - 4
    n = 0
    while abs(derivee) > epsilon:
        x = x - alpha * derivee
        derivee = 2*x - 4
        n = n + 1
    return x, n
```

On exécute ce code avec x0=5, alpha=0,1 et epsilon=0,01.



6. D'après le graphique, au bout de combien d'itérations la boucle s'arrête-t-elle ?
7. Pourquoi ne peut-on pas, avant d'exécuter ce code, prédire avec certitude et à l'avance le nombre exact d'itérations qu'il va effectuer (contrairement à l'algorithme 2 de la partie B) ? Quelle est la différence essentielle entre une boucle « pour » et une boucle « tant que » à ce sujet ?
8. Que se passerait-il si on choisissait un epsilon beaucoup plus petit, par exemple 0,0000001 ? Le nombre d'itérations augmenterait-il ou diminuerait-il ?
9. Que se passerait-il si, par erreur, on utilisait un pas d'apprentissage alpha trop grand (voir document 0, partie D) dans ce code avec une boucle « tant que » ? La boucle est-elle certaine de s'arrêter un jour ?

Partie D — La vraie boucle d'entraînement : époques et lots de données

Niveau : Terminale

Dans la pratique, une IA ne s'entraîne pas sur un seul point à la fois, ni sur toutes les données d'un coup : elle découpe les données d'entraînement en petits groupes appelés lots (batches), et répète l'ensemble du jeu de données plusieurs fois — chaque passage complet sur toutes les données s'appelle une époque (epoch).

Structure réelle d'une boucle d'entraînement

```
pour epoch allant de 1 à nombre_epoques :
    pour chaque lot dans les données d'entraînement :
        calculer le gradient sur ce lot
        mettre à jour les paramètres (a, b, ...) avec ce gradient
```

On reprend le jeu de données du document 4 (8 élèves, temps de révision et score), en le découpant en lots de 2 élèves à la fois.

10. Combien de lots obtient-on avec ces 8 élèves, à raison de 2 élèves par lot ?
11. Si l'entraînement effectue 3 époques complètes, combien de fois, au total, les paramètres (a, b) sont-ils mis à jour ? Justifie ton calcul à l'aide du principe multiplicatif (document 7).
12. Pourquoi découper les données en lots plutôt que de calculer le gradient sur les 8 élèves à la fois, à chaque étape ? (Indice : pense à un jeu de données de plusieurs millions d'exemples, qui ne tiendrait pas entièrement en mémoire d'un seul coup.)

Partie E — Complexité : compter les opérations d'un algorithme

Niveau : Terminale — lien avec le document 7

On avait vu, dans le document 7, que le mécanisme d'attention nécessite n^2 comparaisons pour un texte de n mots. Cette même idée s'exprime, en algorithmique, par une boucle imbriquée dans une autre boucle :

Code Python — deux façons de parcourir des données

```
# Boucle simple : n opérations
for i in range(n):
    ... une opération ...

# Boucle imbriquée : n × n = n² opérations
for i in range(n):
    for j in range(n):
        ... une opération ...
```

13. Pour n=1000, combien d'opérations exécute la boucle simple ? Combien en exécute la boucle imbriquée ?
14. Retrouve-t-on bien le résultat du document 7 (partie B) ?

Exercices d'entraînement

Exercice 1 — Une autre trace d'exécution

Niveau : Première / Terminale

On reprend l'algorithme 2 de la partie B, avec $g(x)=x^2-6x+10$ (donc $g'(x)=2x-6$), x0=6, alpha=0,2 et n_iterations=3.

15. Exécute « à la main » l'algorithme, en donnant la valeur de x après chaque itération.
16. Compare avec les résultats du document 0 (exercice 1). Retrouves-tu les mêmes valeurs ?

Exercice 2 — Une boucle « tant que » différente

Niveau : Terminale

On exécute le code de la partie C (fonction descente_gradient_arret) avec $g'(x)=2x-6$, x0=6, alpha=0,2 et epsilon=0,05.

17. Sans exécuter le code, prédis si le nombre d'itérations sera plutôt petit (moins de 10) ou plutôt grand (plus de 50). Justifie à partir du pas alpha=0,2, plus grand que celui de la partie C (0,1).
18. Écris les 2 premières itérations de cette boucle (valeurs de x et de la dérivée).

Exercice 3 — Compter les mises à jour

Niveau : Terminale

Un jeu de données contient 10 000 exemples, découpés en lots de 50 exemples. L'entraînement dure 20 époques.

19. Combien de lots y a-t-il par époque ?
20. Combien de mises à jour des paramètres sont effectuées au total sur les 20 époques ?

Corrigé

Partie A

A.1. i=1 : s=0+1=1. i=2 : s=1+2=3. i=3 : s=3+3=6. i=4 : s=6+4=10. i=5 : s=10+5=15. L'algorithme retourne 15.

A.2. s ← 0 ; i ← 1 ; tant que $i \leqslant 5$: s ← s+i ; i ← i+1 ; retourner s. La variable i doit être incrémentée à chaque tour pour que la condition $i \leqslant 5$ finisse par devenir fausse et que la boucle se termine.

Partie B

B.1. i=1 : derivee=2×5−4=6, x=5−0,1×6=4,4. i=2 : derivee=2×4,4−4=4,8, x=4,4−0,1×4,8=3,92. i=3 : derivee=2×3,92−4=3,84, x=3,92−0,1×3,84=3,536. On retrouve exactement les valeurs $x_1=4,4$; $x_2=3,92$; $x_3=3,536$ calculées dans le document 0.

B.2. fonction descente_gradient(x0, alpha, n_iterations) : x ← x0 ; historique ← [x0] ; pour i allant de 1 à n_iterations : derivee ← 2x−4 ; x ← x−alpha×derivee ; ajouter x à historique ; retourner historique.

B.3. range(n_iterations) engendre la séquence 0, 1, 2, ..., n_iterations−1, soit exactement n_iterations valeurs : le contenu de la boucle est donc exécuté n_iterations fois. Pour n_iterations=3, il est exécuté 3 fois.

Partie C

C.1. D'après le graphique, la boucle s'arrête après 29 itérations ($x \approx 2,0046$, avec $|f'(x)| \approx 0,0093 < 0,01$).

C.2. Une boucle « pour » répète un nombre de fois fixé à l'avance (ici n_iterations, connu avant l'exécution) ; une boucle « tant que » répète tant qu'une condition reste vraie, et ce nombre de répétitions dépend du déroulement même du calcul (ici, de la vitesse à laquelle |derivee| diminue), donc on ne peut pas toujours le connaître avant d'exécuter le programme.

C.3. Avec un epsilon beaucoup plus petit, la condition |derivee|>epsilon reste vraie plus longtemps (il faut que la dérivée devienne encore plus proche de 0) : le nombre d'itérations augmenterait.

C.4. Avec un pas alpha trop grand (document 0, partie D), la suite (x) peut osciller de plus en plus fort, voire diverger, sans que |derivee| ne s'approche jamais de 0 : dans ce cas, la condition de la boucle « tant que » ne devient jamais fausse, et la boucle ne s'arrête jamais (boucle infinie). C'est un vrai risque pratique des boucles « tant que » mal maîtrisées.

Partie D

D.1. 8 élèves ÷ 2 par lot = 4 lots.

D.2. Par le principe multiplicatif (document 7) : 3 époques × 4 lots par époque = 12 mises à jour des paramètres au total.

D.3. Un jeu de données de plusieurs millions d'exemples ne tient en général pas entièrement dans la mémoire vive de l'ordinateur (ou du processeur graphique) en une seule fois : découper en lots permet de ne charger et traiter qu'une petite partie des données à chaque étape, tout en mettant à jour les paramètres plus fréquemment (à chaque lot, plutôt qu'une seule fois par époque), ce qui accélère souvent la convergence de la descente de gradient.

Partie E

E.1. Boucle simple : n=1000 opérations. Boucle imbriquée : n^2=1 000 000 opérations.

E.2. Oui : on retrouve exactement les 1 000 000 comparaisons calculées dans le document 7 (partie B) pour n=1000 mots — la boucle imbriquée est très exactement la traduction algorithmique du principe multiplicatif utilisé pour compter les paires.

Exercice 1

1. i=1 : derivee=2×6−6=6, x=6−0,2×6=4,8. i=2 : derivee=2×4,8−6=3,6, x=4,8−0,2×3,6=4,08. i=3 : derivee=2×4,08−6=2,16, x=4,08−0,2×2,16=3,648.

2. Oui, on retrouve exactement x_1=4,8 ; x_2=4,08 ; x_3=3,648, les mêmes valeurs que dans le document 0 (exercice 1).

Exercice 2

1. Avec un pas plus grand (0,2 au lieu de 0,1), chaque itération réduit davantage la dérivée à chaque étape (mais avec un risque d'oscillation si le pas est trop grand) : on peut s'attendre à un nombre d'itérations plutôt petit, car alpha=0,2 reste raisonnable pour cette fonction (pas de divergence).

2. i=1 : derivee=2×6−6=6, x=6−0,2×6=4,8. i=2 : derivee=2×4,8−6=3,6, x=4,8−0,2×3,6=4,08.

Exercice 3

1. 10 000 ÷ 50 = 200 lots par époque.

2. 20 époques × 200 lots = 4 000 mises à jour des paramètres au total.

Mathématiques et informatique du programme mobilisées dans ce document

Algorithmique et programmation (enseignement transversal)

Boucles « pour » et « tant que », conditions d'arrêt, notion de trace d'exécution.

Écriture et lecture de fonctions en pseudocode ou en Python.

Boucles imbriquées et dénombrement du nombre d'opérations (lien avec le document 7).

Terminale (spécialité mathématiques)

Réinvestissement de la descente de gradient et du principe multiplicatif (documents 0 et 9).