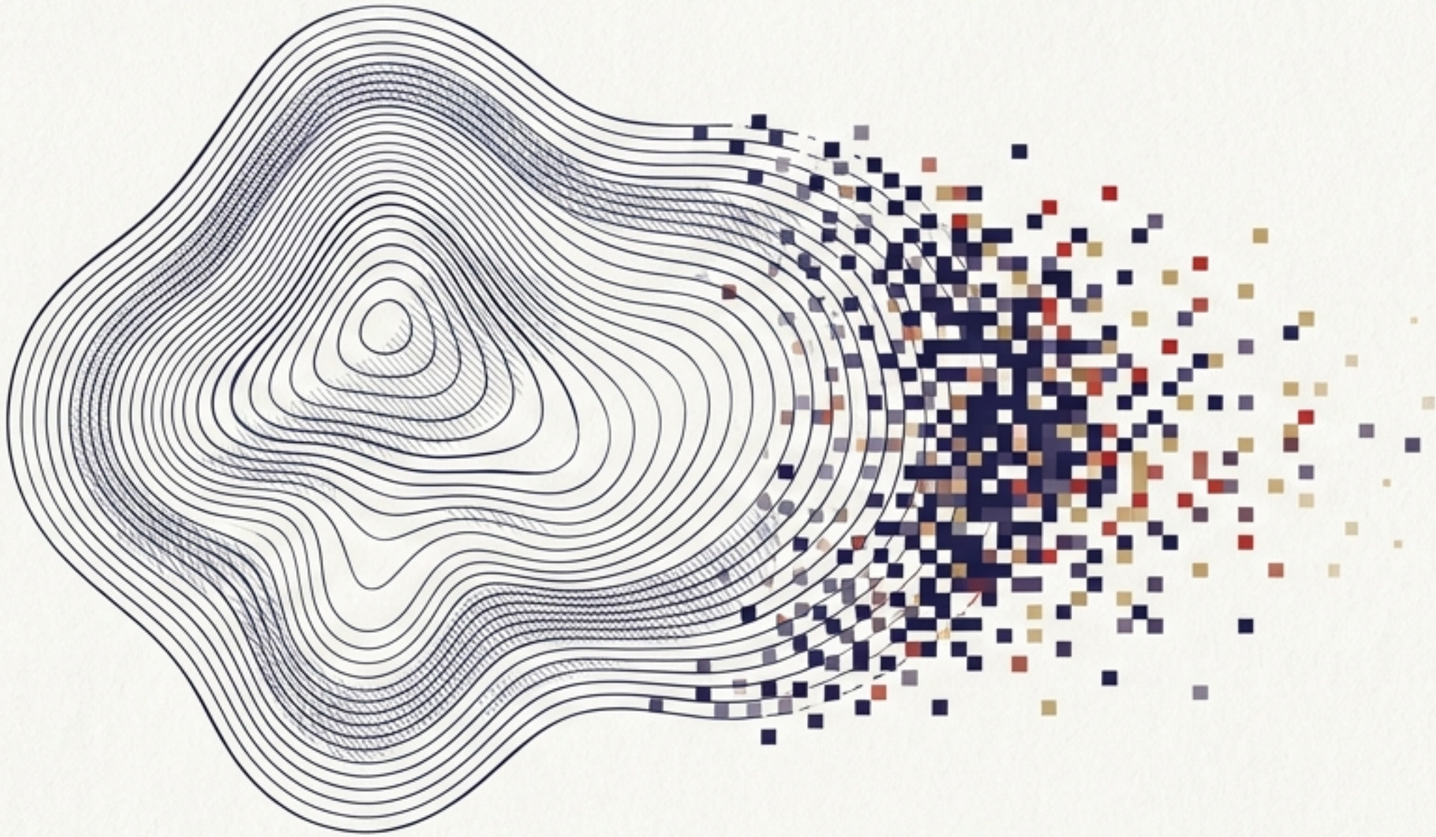


La Descente de Gradient

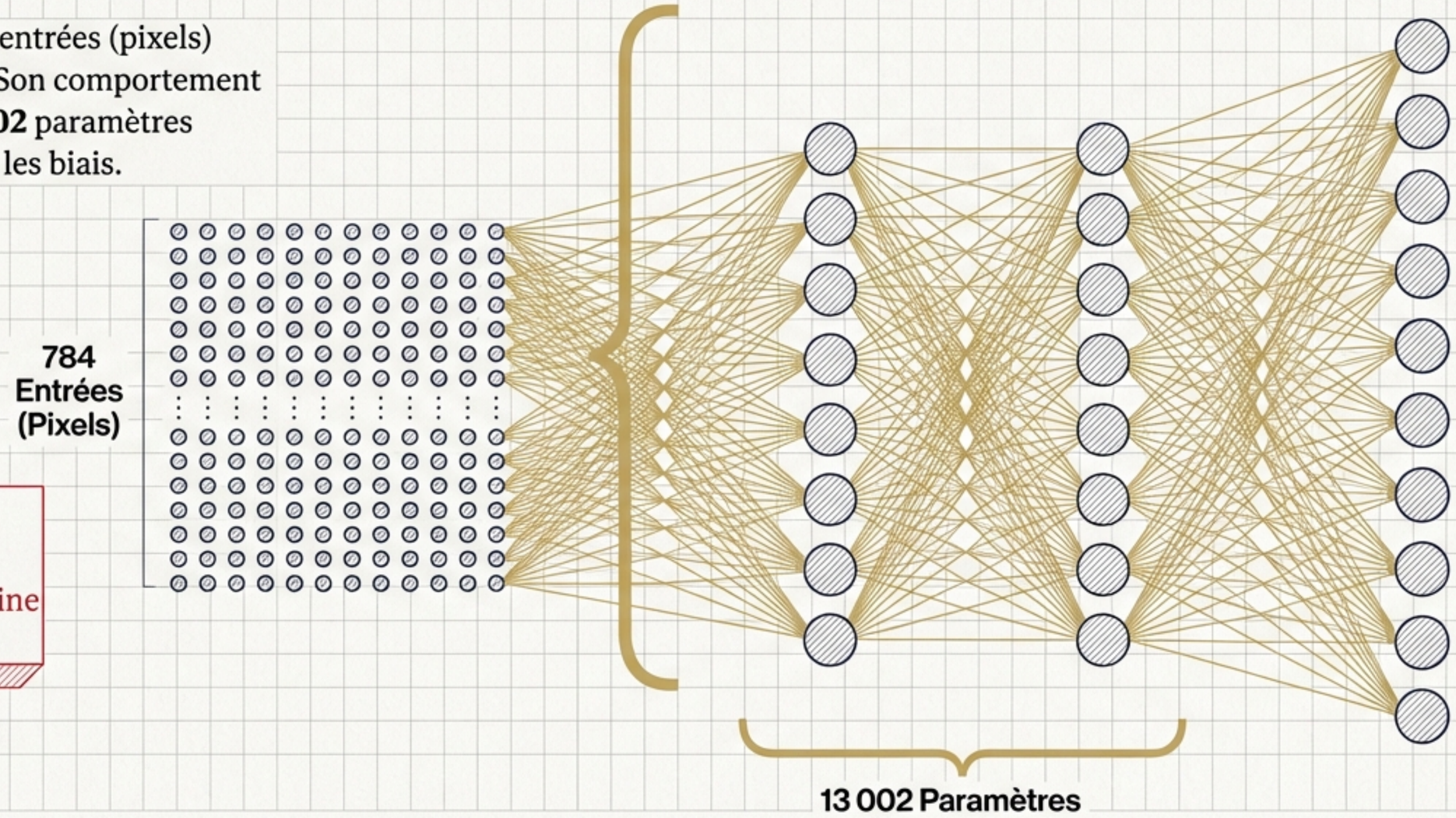
Comment les réseaux de neurones apprennent

Une exploration visuelle de la mécanique derrière l'intelligence artificielle.

Basé sur les travaux de 3Blue1Brown

La structure du réseau

Ce réseau possède 784 entrées (pixels) et 10 sorties (chiffres). Son comportement est déterminé par **13 002** paramètres ajustables : les poids et les biais.

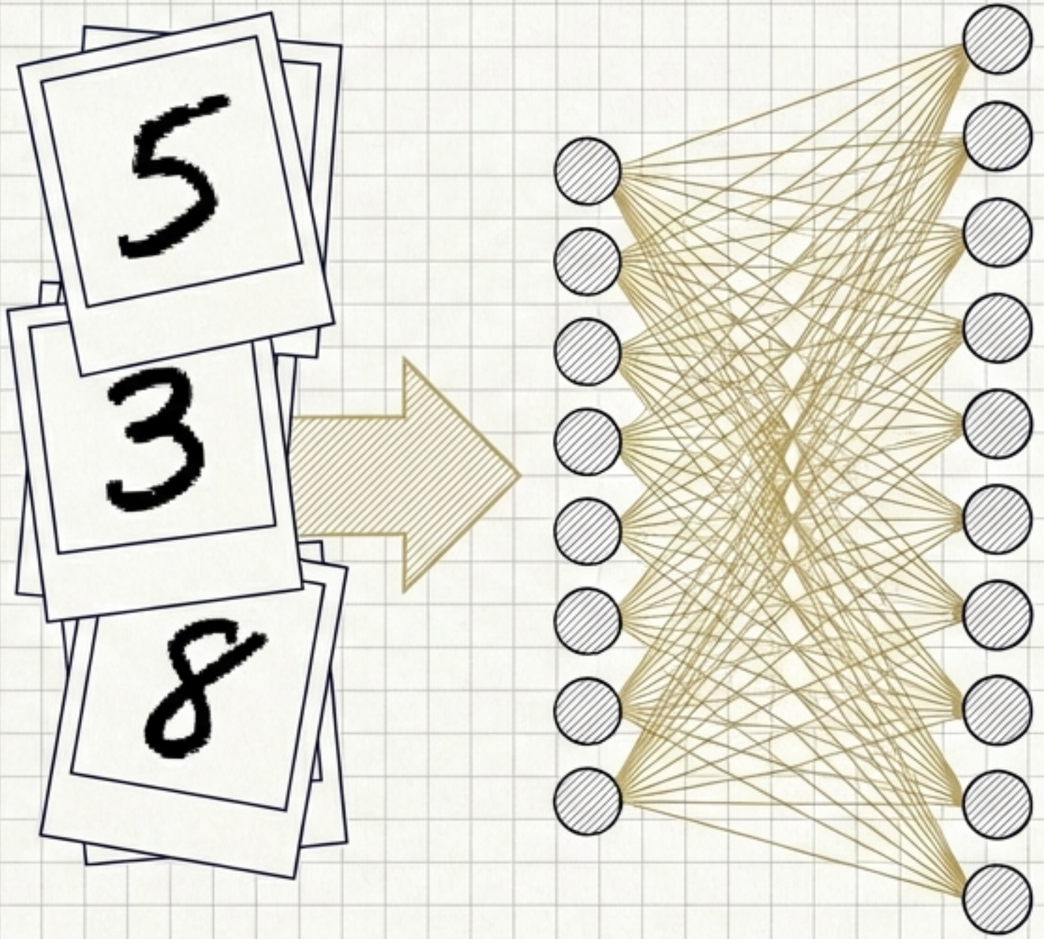


Actuellement, ces paramètres sont aléatoires. La machine est inutile.

Le défi de l’apprentissage

Nous n’écrivons pas d’algorithme pour identifier un “3”. Nous écrivons un algorithme qui *trouve* la bonne configuration des **13 002** paramètres en analysant des milliers d’exemples.

```
if pixel[5] > 0.5 then...
```

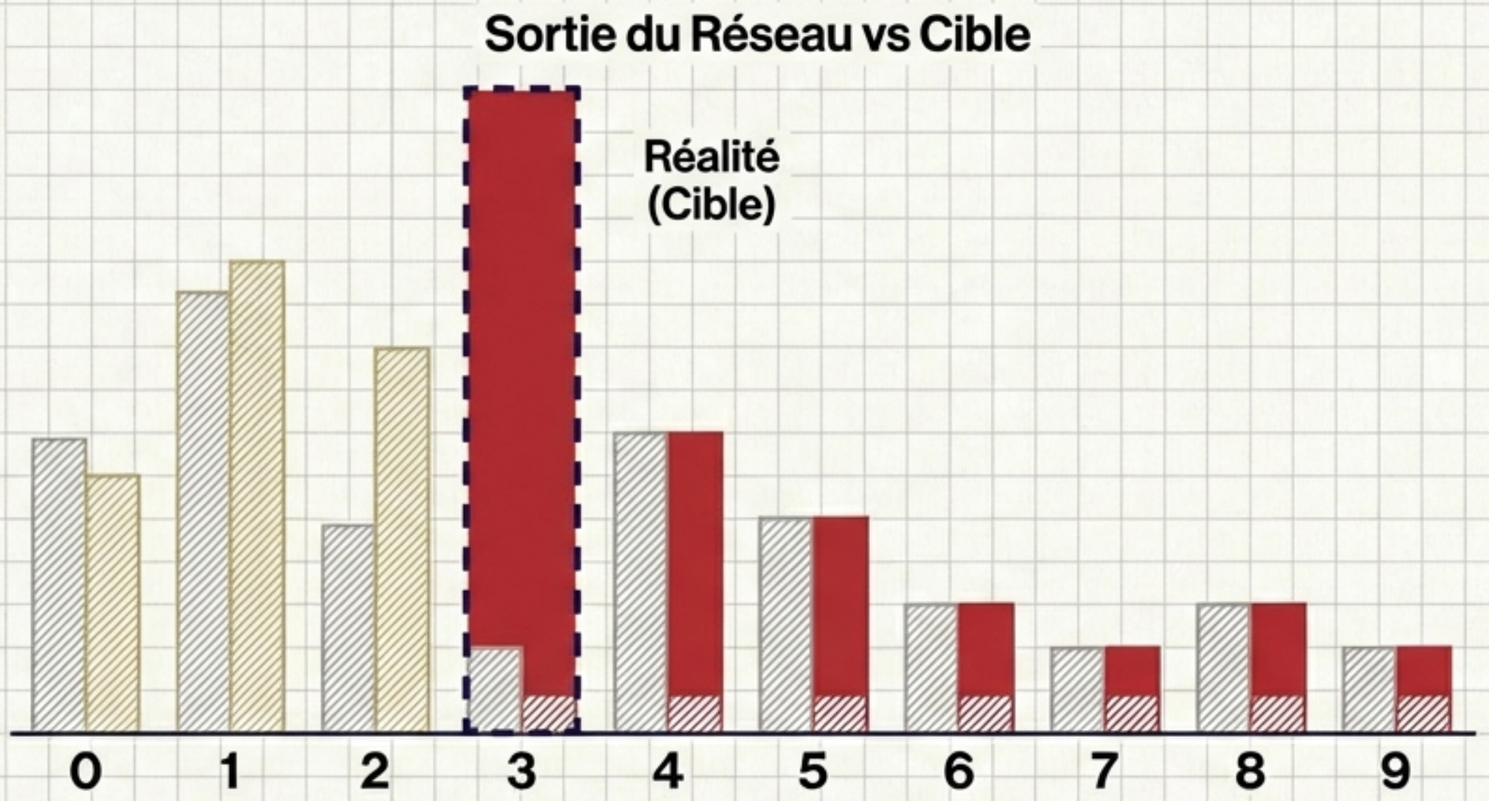


C’est un problème d’optimisation, pas de règles explicites.

Définir l’erreur

Pour s’améliorer, le réseau doit savoir à quel point il se trompe. Nous calculons la différence mathématique entre la prédiction et la bonne réponse.

Entrée (Exemple)



La Fonction de Coût

Le coût total est la somme de ces erreurs individuelles.

La mathématique du coût

Nous faisons la somme des carrés des différences pour chaque neurone de sortie. Le but de l'apprentissage est simple : rendre ce nombre le plus petit possible.

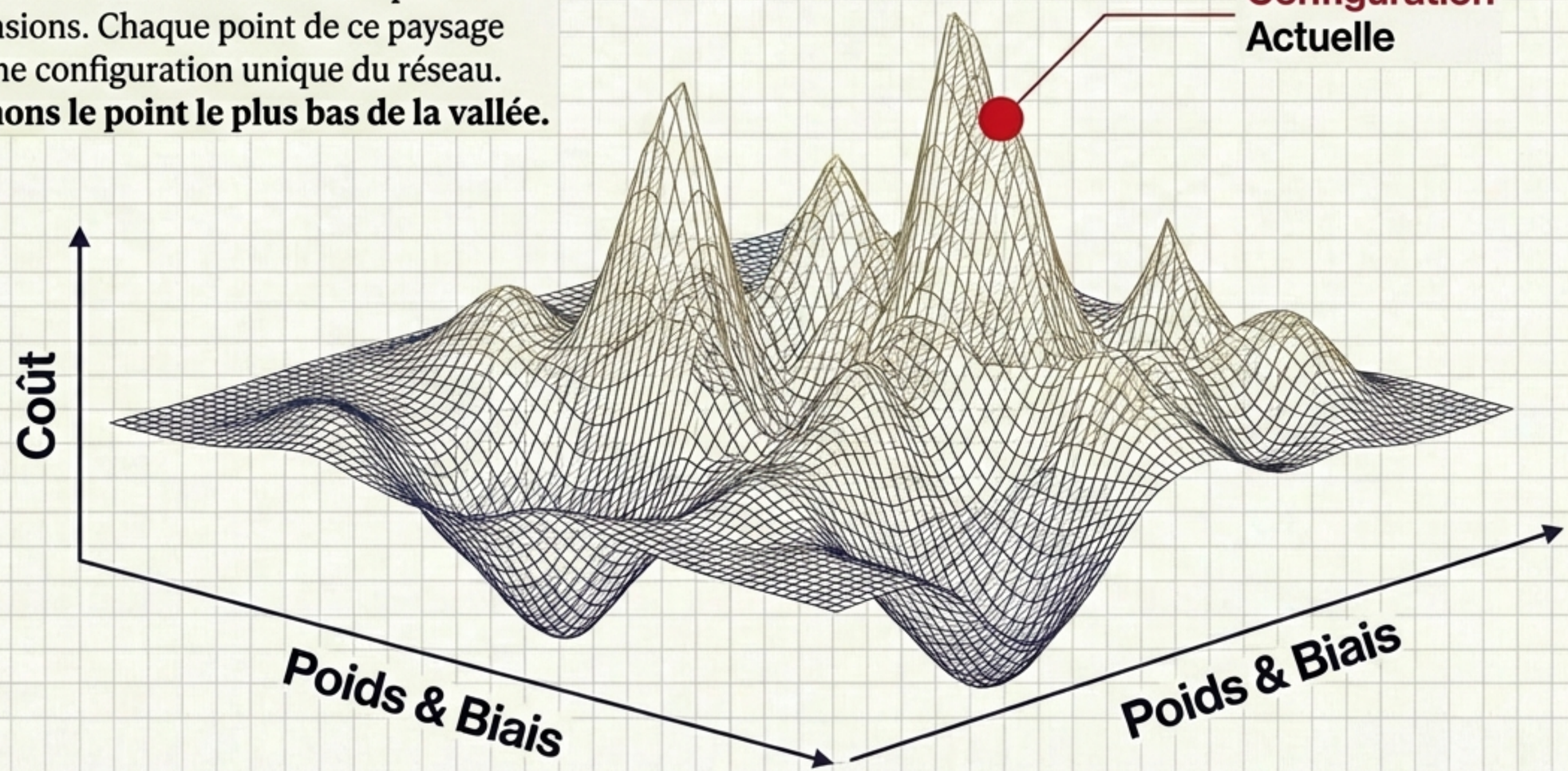
$$C = \sum (y - a)^2$$

Bonne performance

Utter Trash

Le paysage de l’erreur

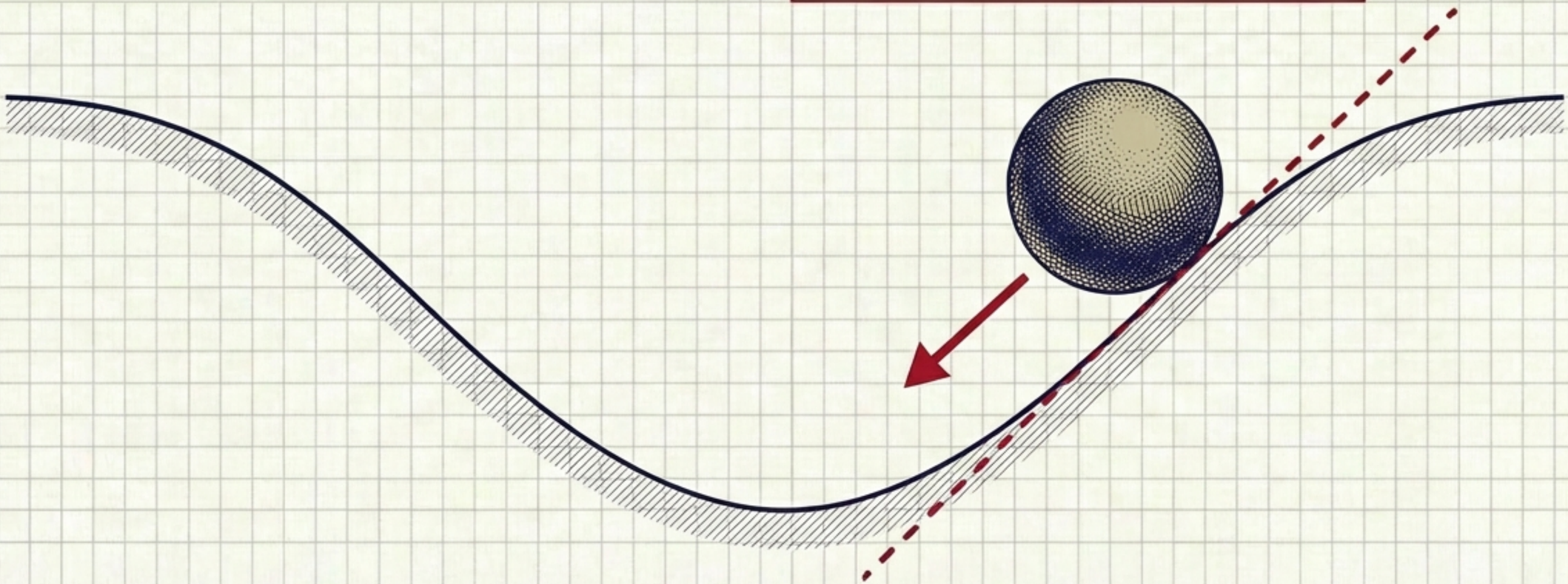
La fonction de coût est une surface complexe en 13 002 dimensions. Chaque point de ce paysage représente une configuration unique du réseau. **Nous cherchons le point le plus bas de la vallée.**



La stratégie de la balle

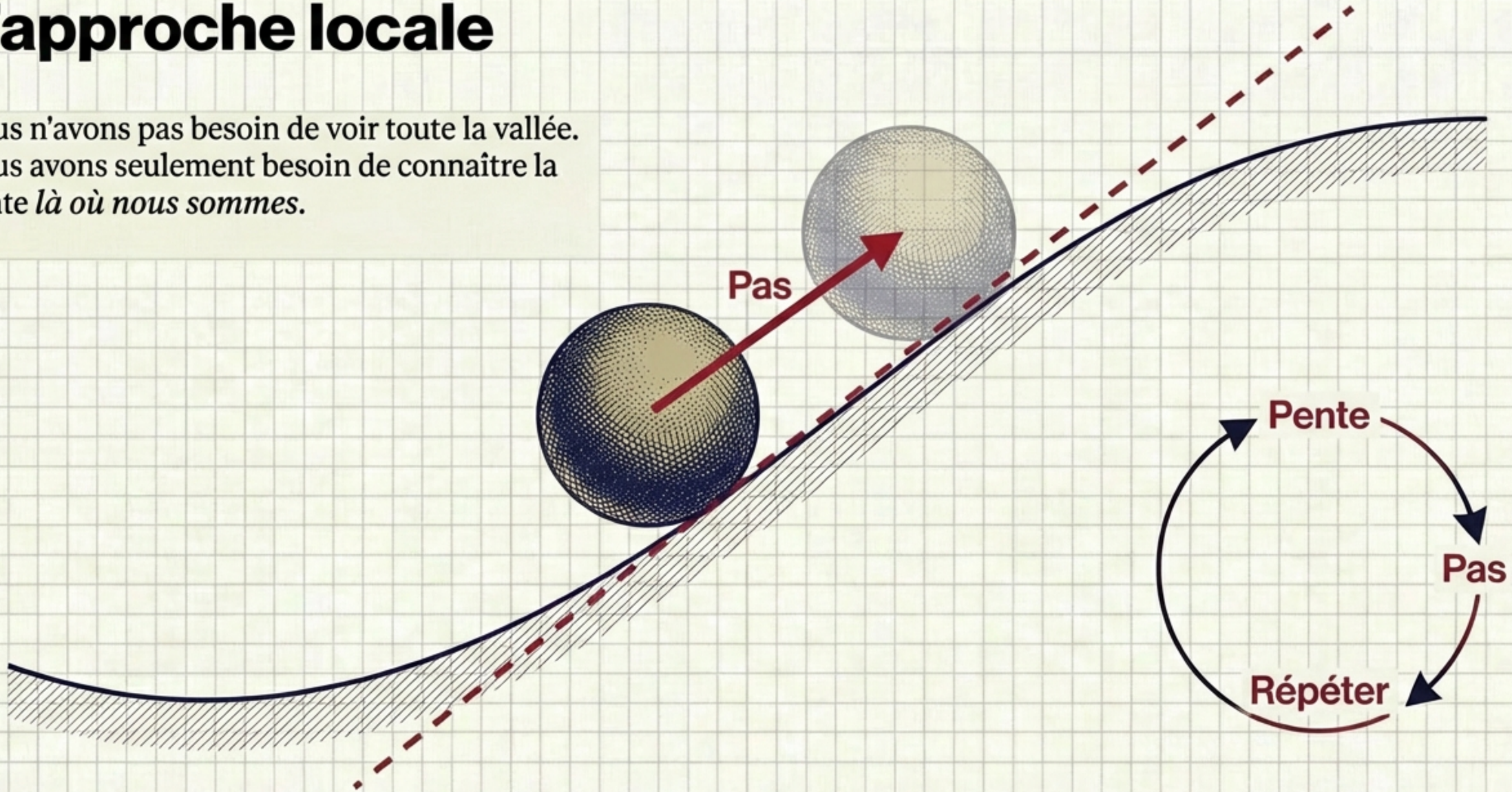
Il est impossible de calculer le minimum directement. À la place, nous agissons comme une balle sur une colline.

Regardez la pente.
Si elle est négative, allez à droite.
Si elle est positive, allez à gauche.



L’approche locale

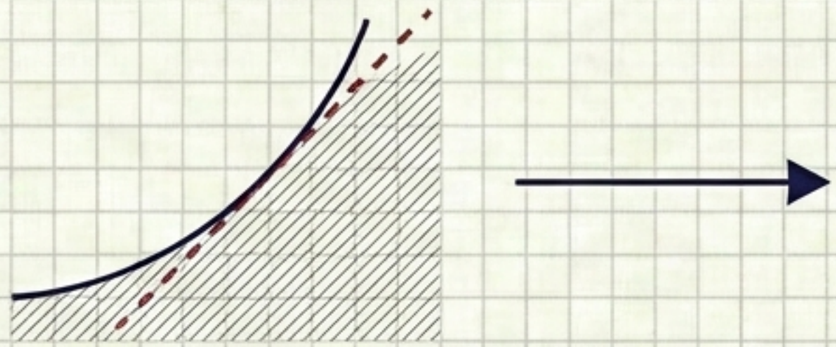
Nous n'avons pas besoin de voir toute la vallée. Nous avons seulement besoin de connaître la pente *là où nous sommes*.

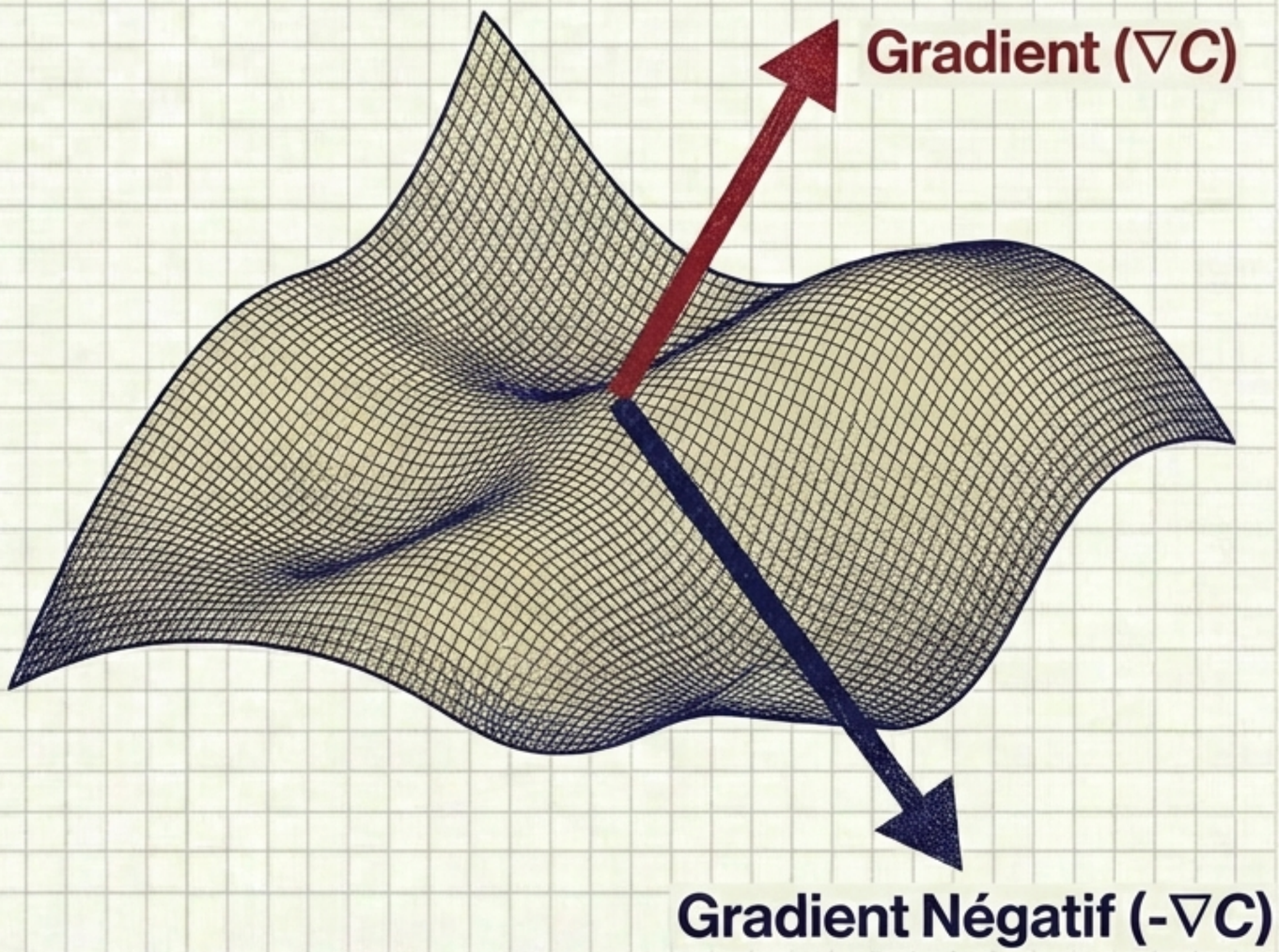


Le Gradient

Dans un espace multidimensionnel, la ‘pente’ est un vecteur appelé le **Gradient** (∇C).

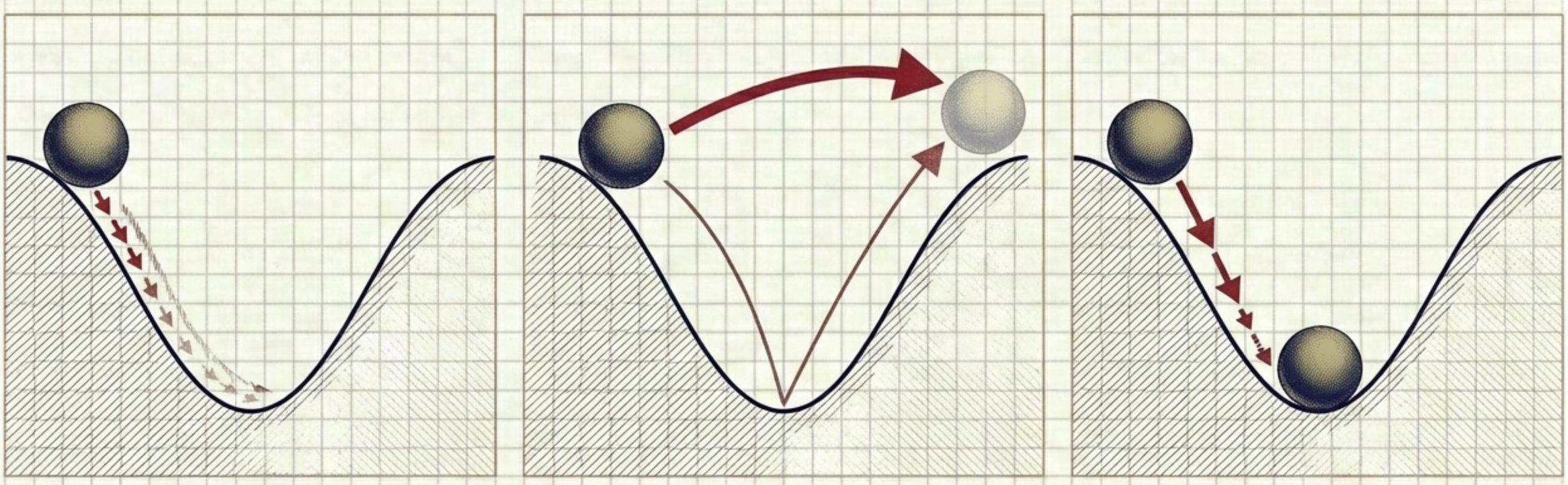
Le gradient indique la direction de la montée la plus raide. Pour apprendre, nous allons dans la direction opposée : le gradient négatif.





Le taux d’apprentissage (η)

La taille du pas est cruciale. C’est un ‘hyperparamètre’ que nous devons régler pour assurer une descente efficace sans dépasser le minimum.



Trop lent

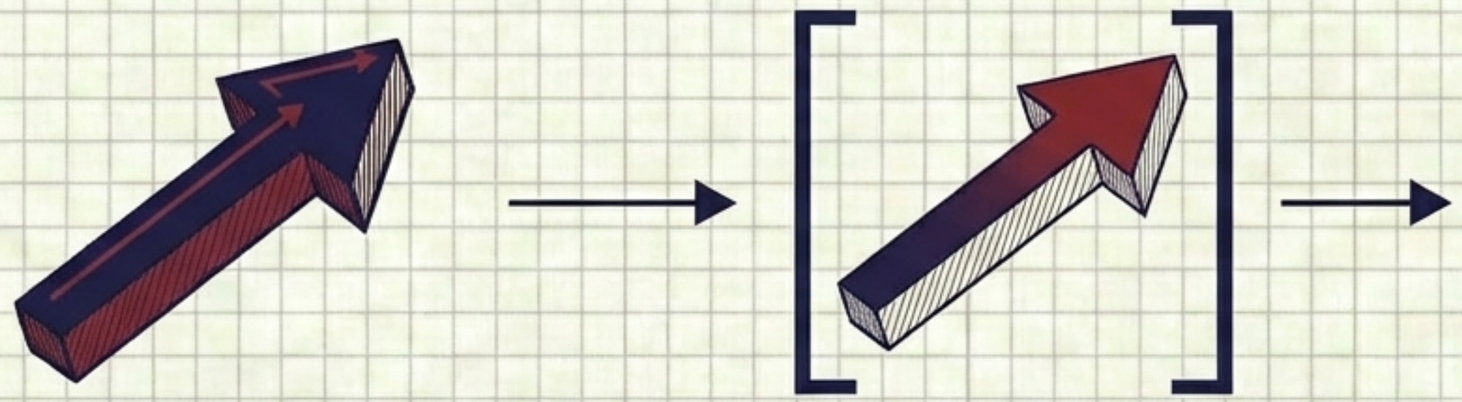
Oscillation

Convergence

Le vecteur gradient

Géométriquement, c’est une direction.
Concrètement, c’est une liste de 13 002 nombres.

Ce vecteur est une ‘To-Do List’. Il indique exactement comment modifier chaque poids et biais pour réduire l’erreur.



13 002 lignes

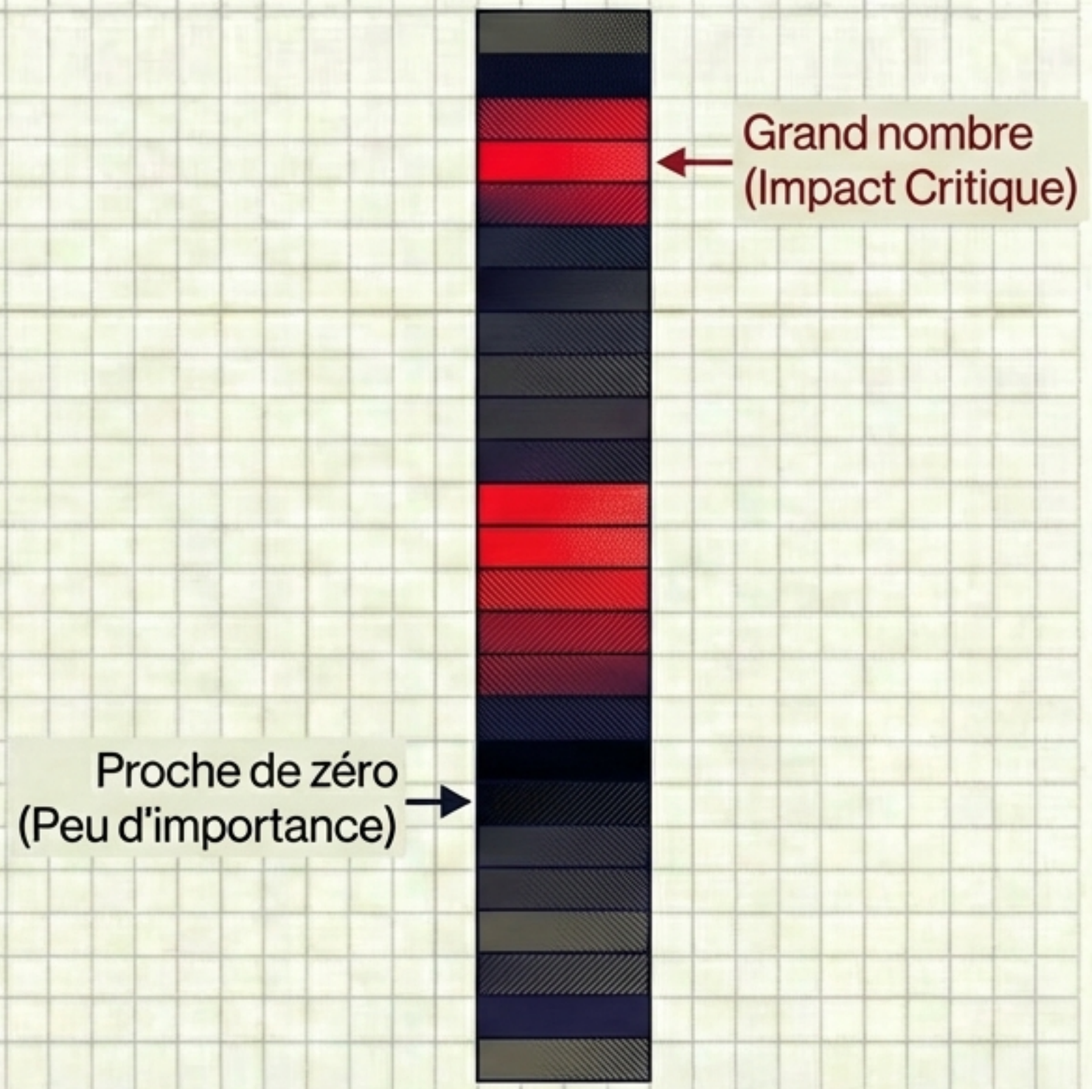
+0.02
−1.5
+0.001
−0.04
+0.1
+0.001
+0.001
+0.001
+0.1
+0.022
−0.02
+0.001
+0.031
−0.03
+0.001
−0.07
+0.1
−0.076
+0.001
+0.031
+0.01
−0.07
+0.001
+0.001
...

Liste de modifications

L’importance relative

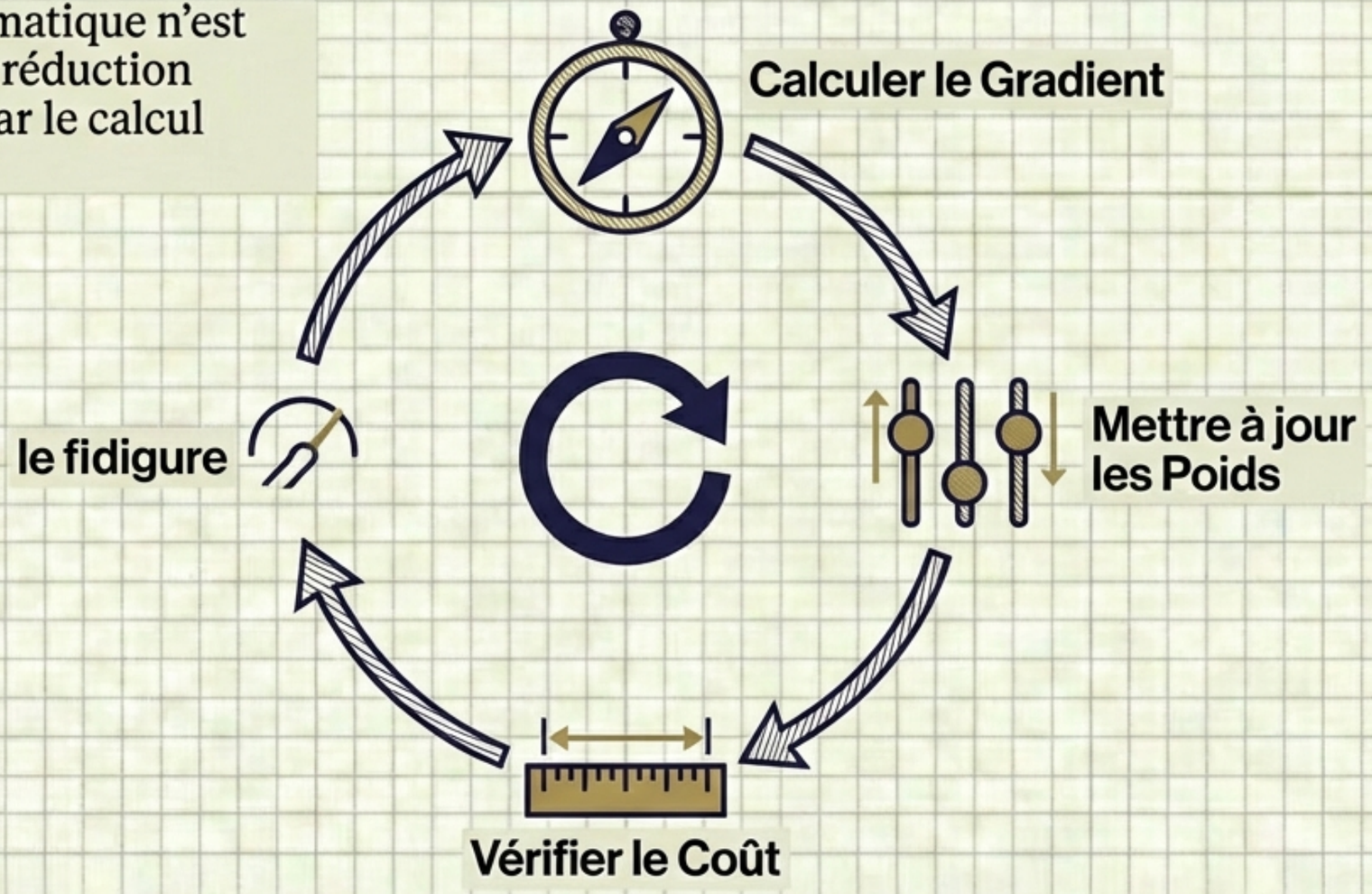
La valeur de chaque nombre dans le gradient indique l’importance de ce poids.

- **Grand nombre :** Changer ce poids aura un impact énorme sur l’erreur.
- **Petit nombre :** Ce poids n’a pas beaucoup d’importance pour l’instant.



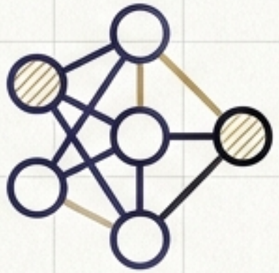
La boucle d’apprentissage

L’apprentissage automatique n’est pas magique. C’est la réduction itérative de l’erreur par le calcul différentiel.

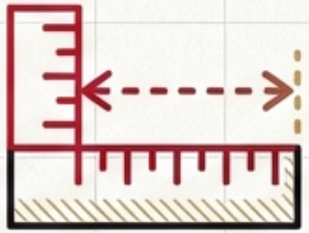


$$\text{nouveau\_poids} = \text{ancien\_poids} - (\eta \times \text{gradient})$$

En résumé

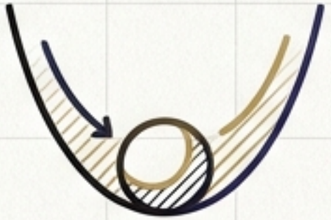


Le Réseau
(La Fonction)



Le Coût
(L'Erreur)

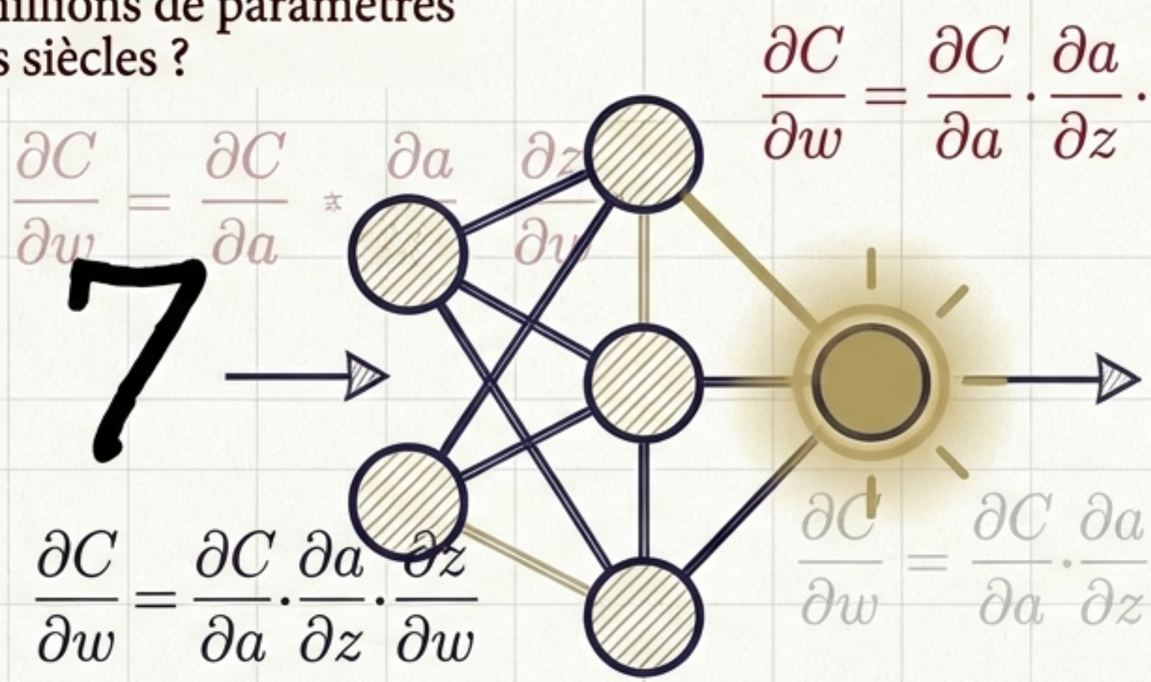
Le Gradient
(La Direction)



La Descente
(L'Optimisation)

Et ensuite ?

Nous savons *quoi* faire (suivre le gradient).
Mais comment calculer ce gradient
efficacement pour des millions de paramètres
sans que cela prenne des siècles ?



Prochain sujet :
La Rétropropagation (Backpropagation)