

Intelligence Artificielle & Données

Réseaux de neurones & Deep Learning

Patrick Félix, Bruno Mery, Grégoire Passault et Pierre Ramet, à partir de notes d'Akka Zemmari

LaBRI, Université de Bordeaux - CNRS

2020 - 2021

Outline

Introduction

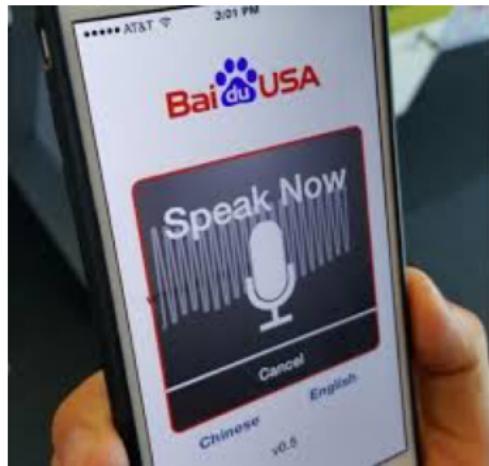
D'abords il y a le neurone

... puis les réseaux de neurones

et c'est quoi alors le deep ?

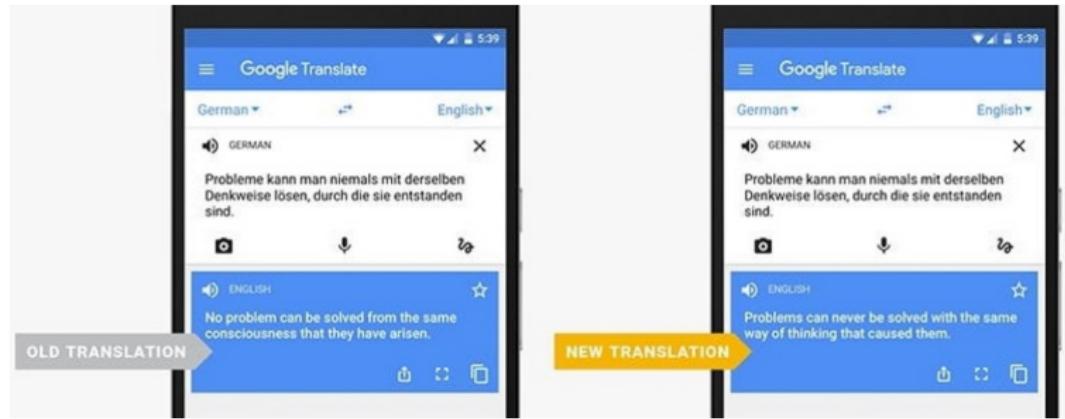
et l'apprentissage dans tout ça ?

Deep Speech

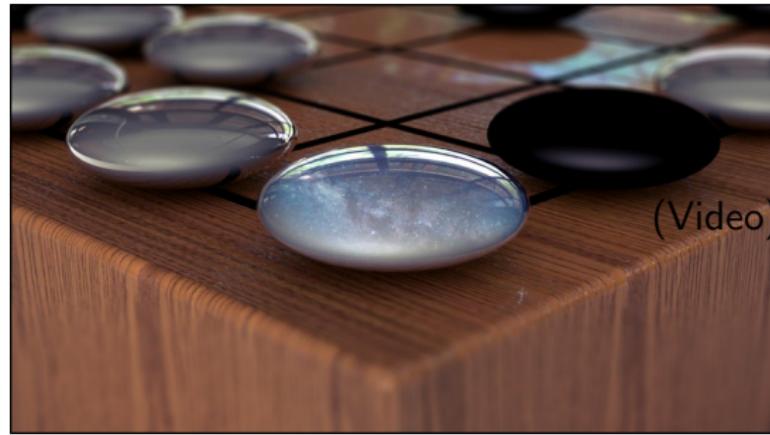


Deep Speech 2 : End-to-End
Speech Recognition in English
and Mandarin

Machine Translation



Alpha Go



Quelques dates et quelques noms

- ▶ 1943 : (McCulloch (neuro-physiologiste) et Pitts (logicien)) proposent les premières notions de neurone formel.
- ▶ 1959 : (Rosenblatt) définit un réseau de neurones avec une couche d'entrée et une sortie.
- ▶ 1960 : (Widrow et Hoff) ADALINE
- ▶ 1969 : (Minsky, Papert) Problème XOR
- ▶ 1986 : (Rumelhart et al.) MLP et backpropagation
- ▶ 1992 : (Vapnik et al.) SVM
- ▶ 1998 : (LeCun et al.) LeNet
- ▶ 2010 : (Hinton et al.) Deep Neural Networks
- ▶ 2012 : (Krizhevsky, Hinton et al.) AlexNet, GPU - 8 couches
- ▶ 2014 : GoogleNet - 22 couches
- ▶ 2015 : Inception (Google) - Deep Dream
- ▶ 2016 : ResidualNet (Microsoft/Facebook) - 152 couches

Ethique et Machine Learning

<http://moralmachine.mit.edu/>

Artificial Intelligence

Expert Systems

Alpha-Beta

Machine Learning

Min-Max
A*

Support Vector Machines

Decision Trees

Random Forests

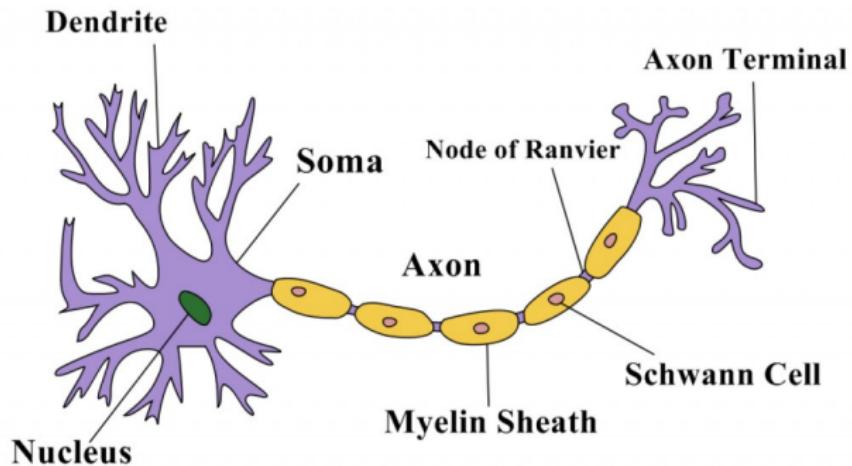
Deep Learning

Turing Award Won by Three Pioneers in Artificial Intelligence



From left, Yann LeCun, Geoffrey Hinton and Yoshua Bengio. The researchers worked on key developments for neural networks, which are reshaping how computer systems are built.

D'abords il y a le neurone



D'abords il y a le neurone

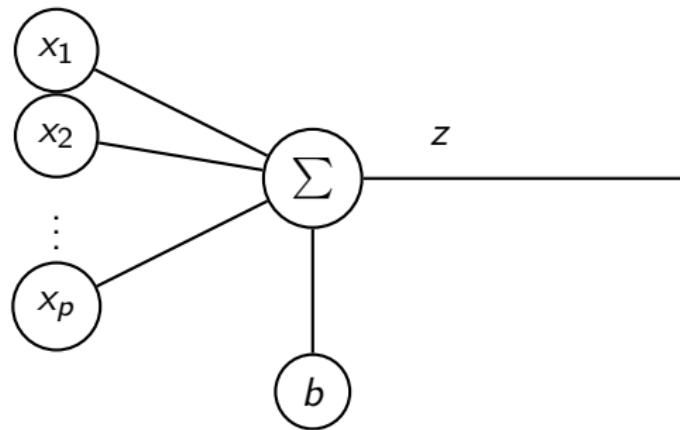


Figure – Un neurone formel.

D'abords il y a le neurone

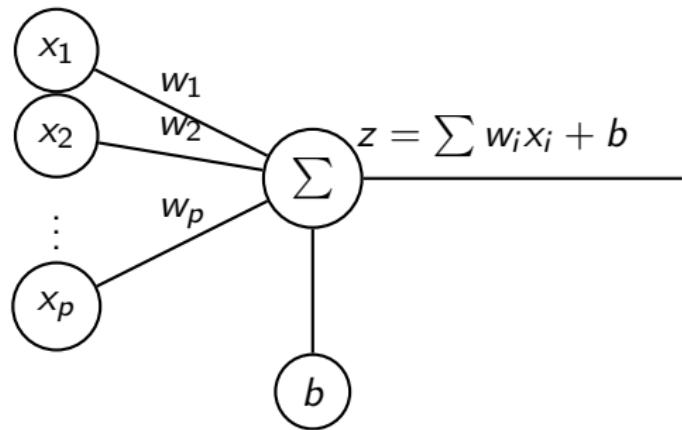


Figure – Un neurone formel.

D'abords il y a le neurone

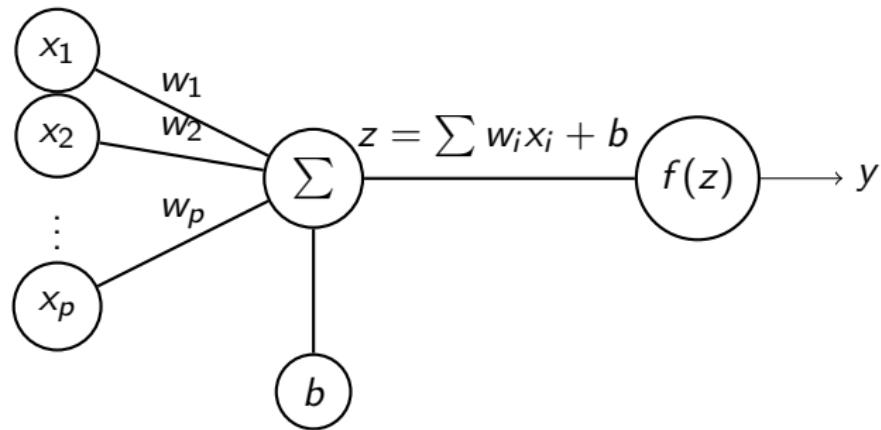


Figure – Un neurone formel.

D'abords il y a le neurone

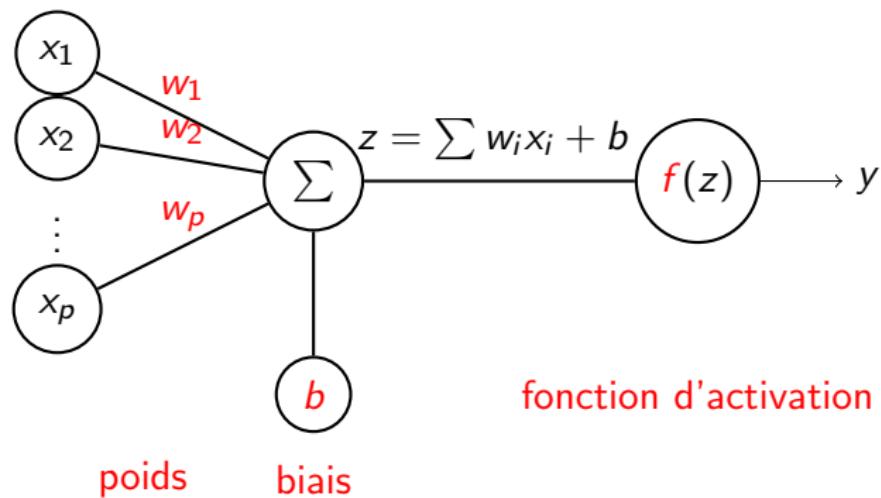
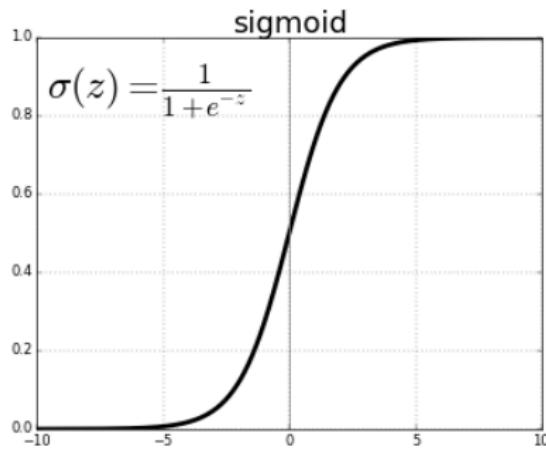


Figure – Un neurone formel.

D'abords il y a le neurone

Exemple de fonction d'activation : sigmoïd

$$f(z) = \frac{1}{1 + e^{-z}}.$$



D'abords il y a le neurone

Autres fonctions d'activation :

- ▶ fonction linéaire : f est la fonction identité,
- ▶ seuil : $f(z) = \mathbb{1}_{[0,\infty[}(z)$,
- ▶ ReLU : $f(z) = \max(0, z)$ (rectified linear unit),
- ▶ radiale : $f(z) = \sqrt{1/2\pi} e^{(-z^2/2)}$,
- ▶ stochastique : $f() = 1$ avec la probabilité $1/(1 + e^{-z/H})$, 0 sinon (H intervient comme une température dans un algorithme de recuit simulé),
- ▶ ...

D'abords il y a le neurone

De manière compacte :

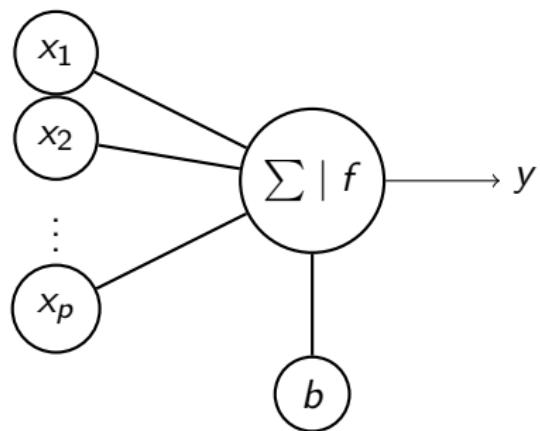


Figure – Un neurone formel.

Réseau de neurones

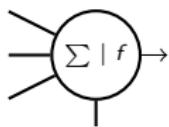


Figure – Un réseau de neurones.

Réseau de neurones

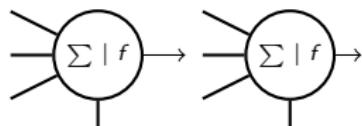


Figure – Un réseau de neurones.

Réseau de neurones

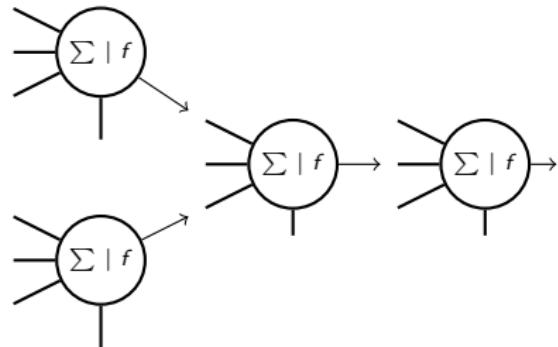


Figure – Un réseau de neurones.

Réseau de neurones

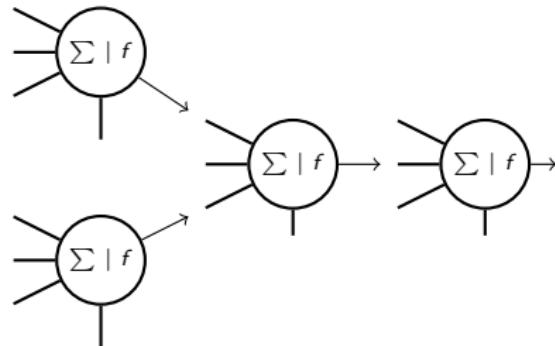


Figure – Un réseau de neurones.

- ▶ Les neurones ont chacun leurs propres poids et biais,

Réseau de neurones

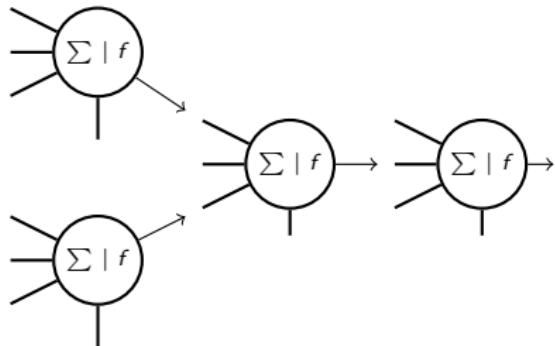
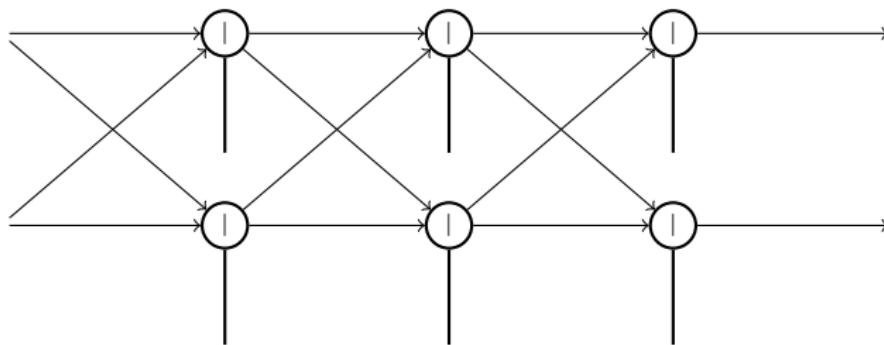


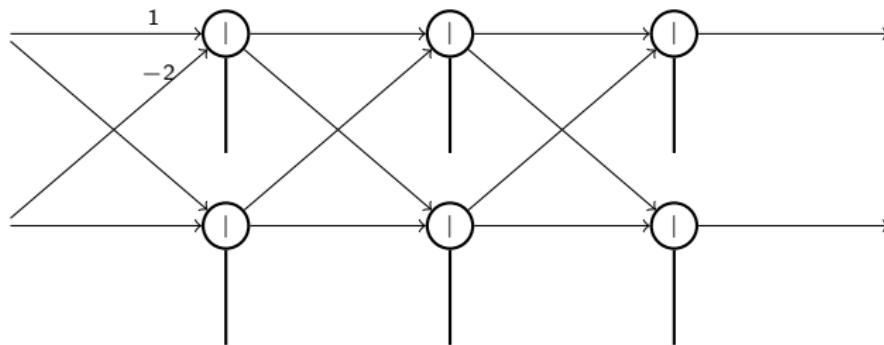
Figure – Un réseau de neurones.

- ▶ Les neurones ont chacun leurs propres poids et biais,
- ▶ Les paramètres du réseau sont alors les différents poids et les différents biais. On note tous ces paramètres θ .

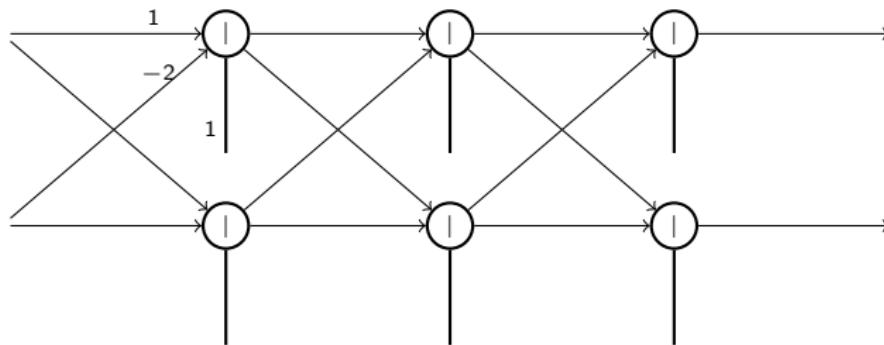
Réseaux de neurones complètement connectés



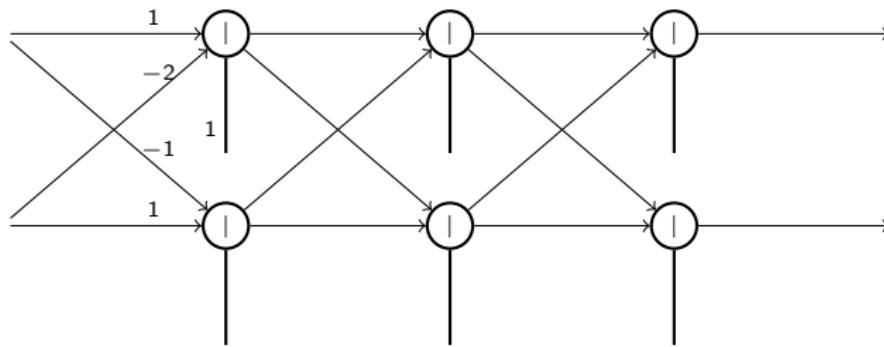
Réseaux de neurones complètement connectés



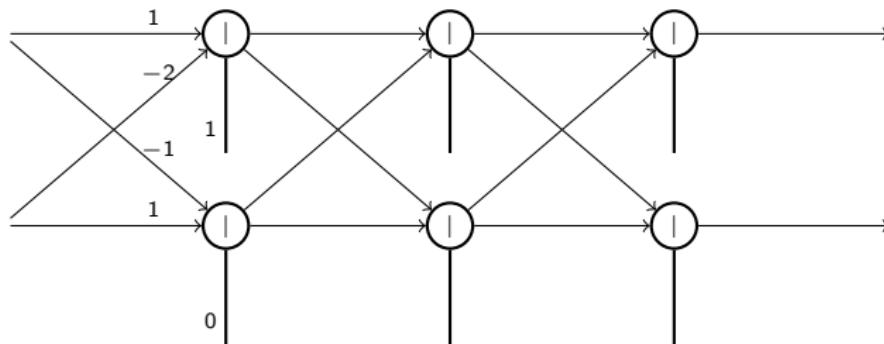
Réseaux de neurones complètement connectés



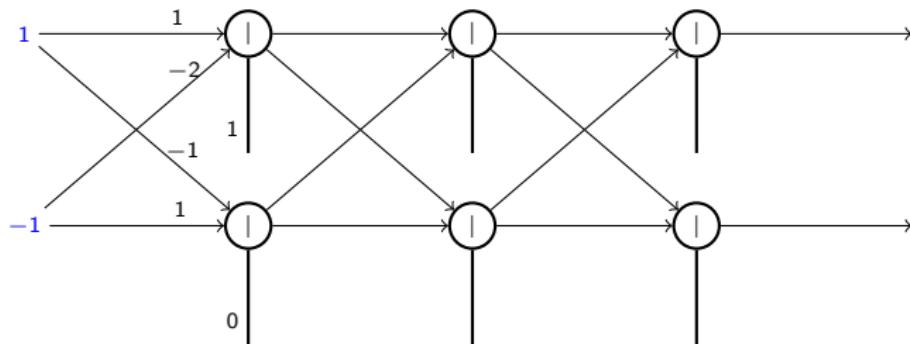
Réseaux de neurones complètement connectés



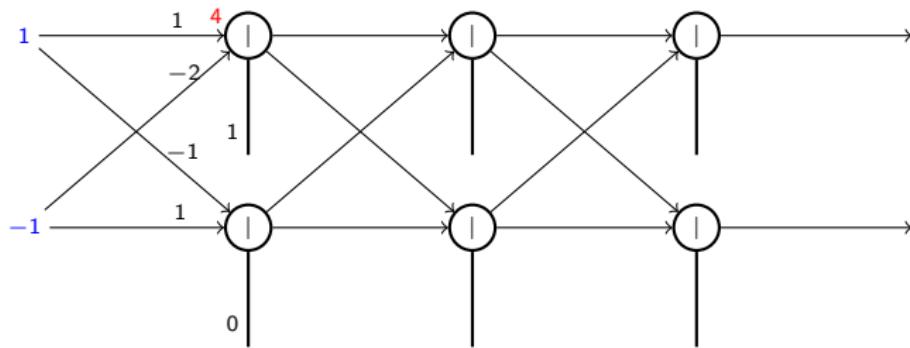
Réseaux de neurones complètement connectés



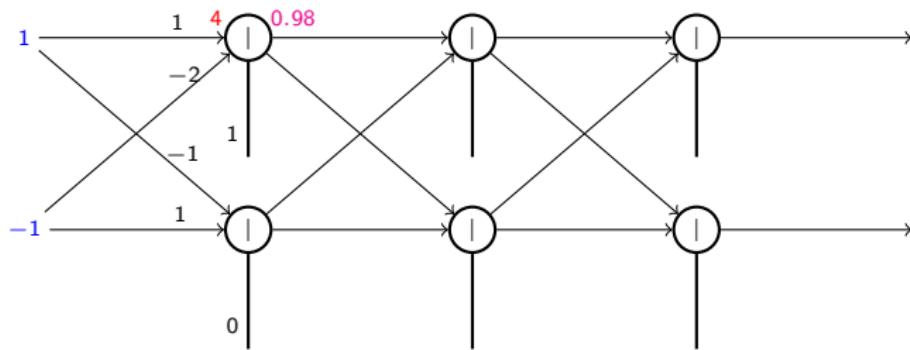
Réseaux de neurones complètement connectés



Réseaux de neurones complètement connectés

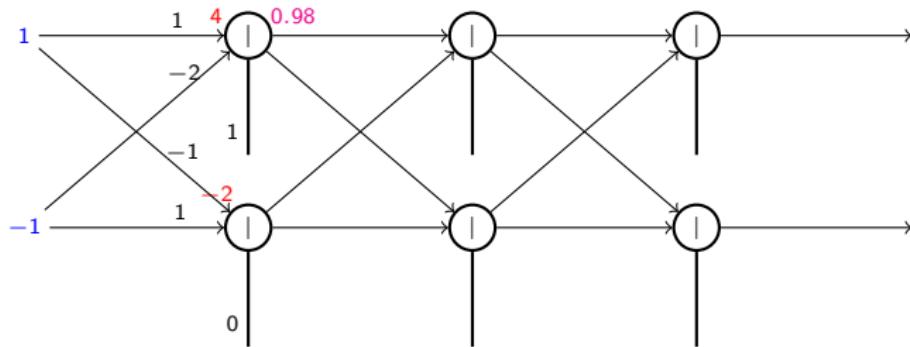


Réseaux de neurones complètement connectés

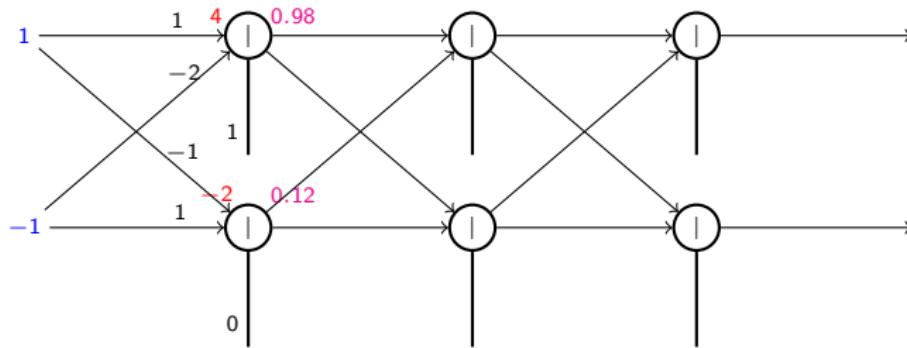


Si f est la fonction sigmoïd : $f(z) = \frac{1}{1+e^{-z}}$

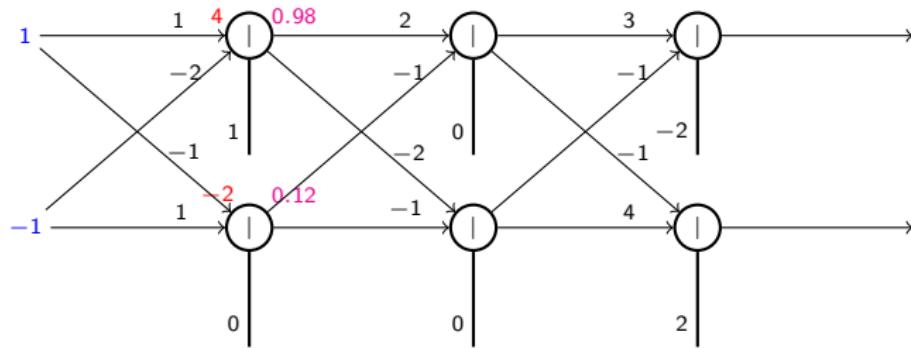
Réseaux de neurones complètement connectés



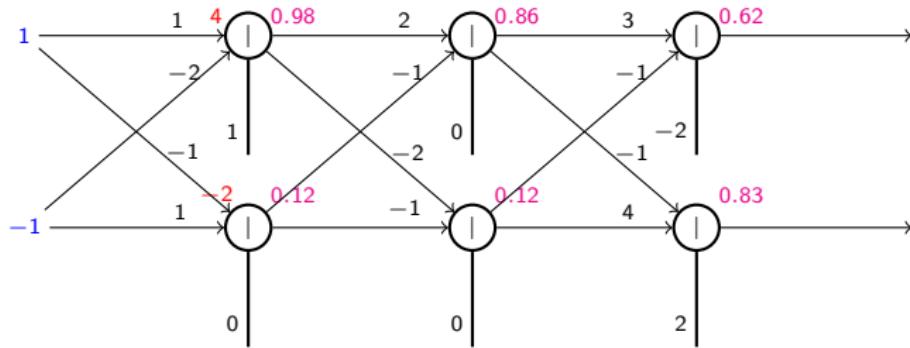
Réseaux de neurones complètement connectés



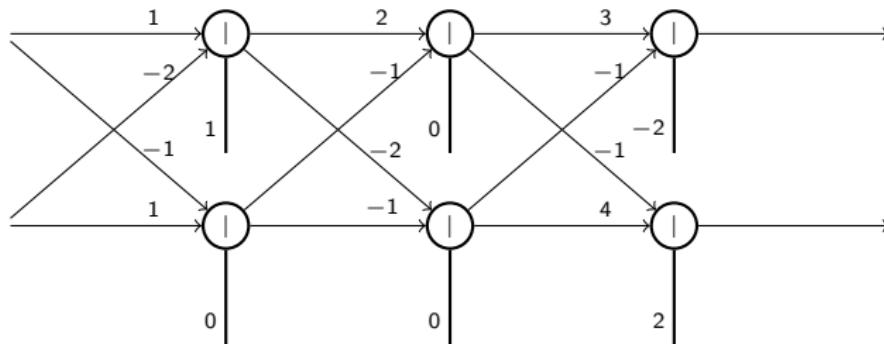
Réseaux de neurones complètement connectés



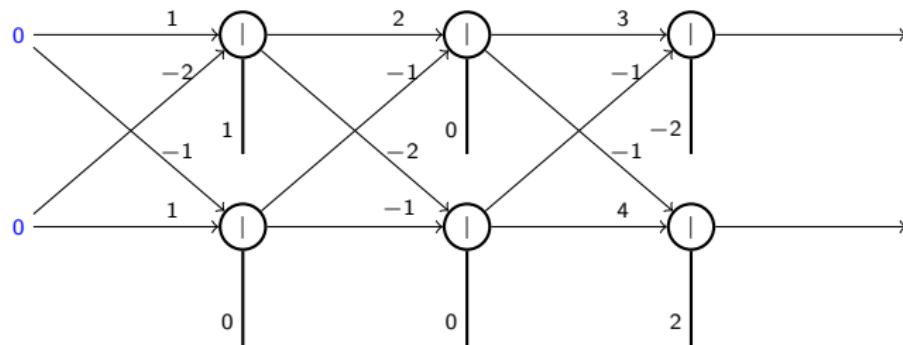
Réseaux de neurones complètement connectés



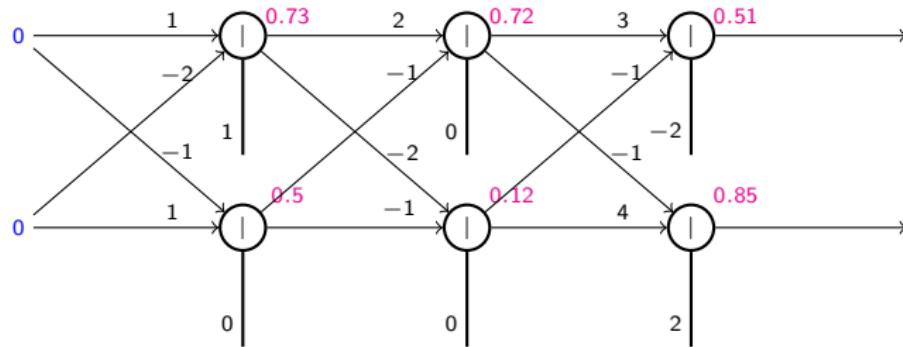
Réseaux de neurones complètement connectés



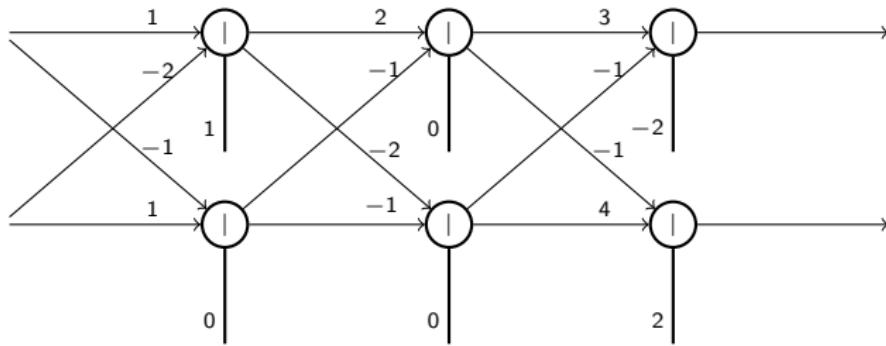
Réseaux de neurones complètement connectés



Réseaux de neurones complètement connectés

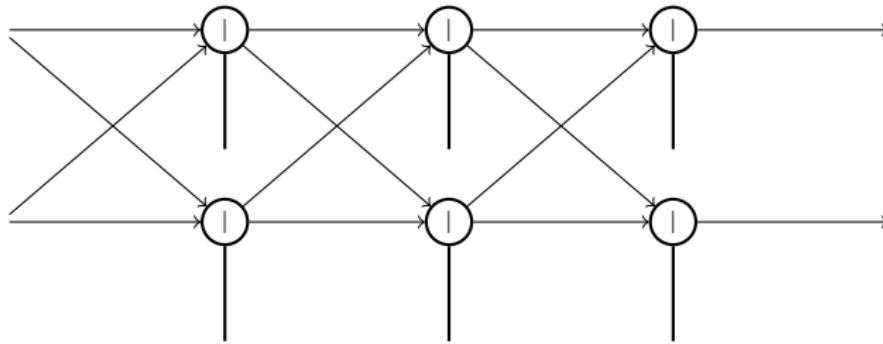


Réseaux de neurones complètement connectés



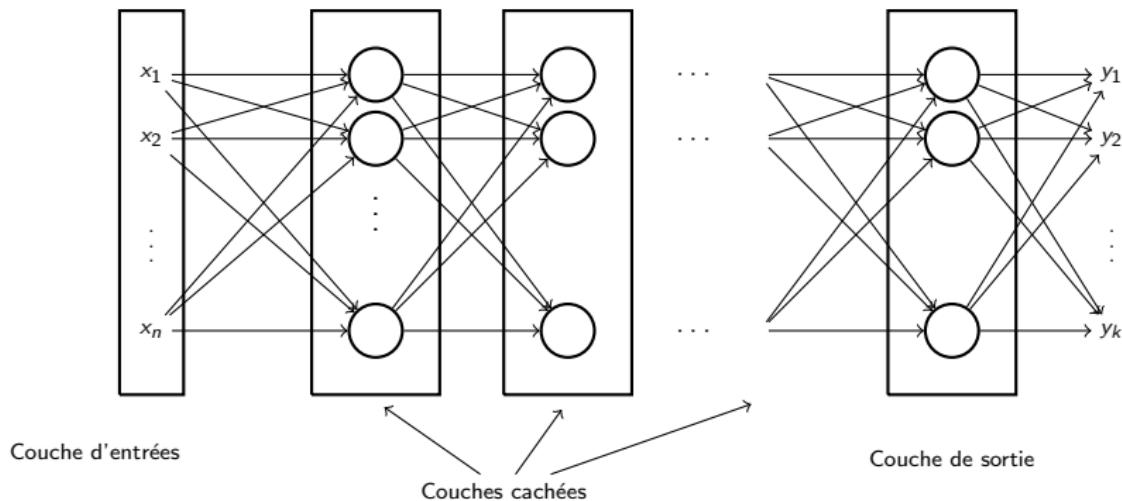
Fixer les paramètres $\theta \rightarrow$ fixer la fonction.

Réseaux de neurones complètement connectés



Donner la structure du réseau → définir un ensemble de fonctions.

Réseaux à plusieurs couches



Apprentissage des paramètres

- ▶ Question :
Comment déterminer les paramètres des neurones
 $w = (w_1, w_2, \dots, w_p)$ et b .
- ▶ Réponse :
Descente du gradient

Deep Learning = (plusieurs) couches cachées.

Exemples de réseaux connus

Deep = Many hidden layers

http://cs231n.stanford.edu/slides/winter1516_lecure8.pdf

8 layers

16.4%



AlexNet (2012)

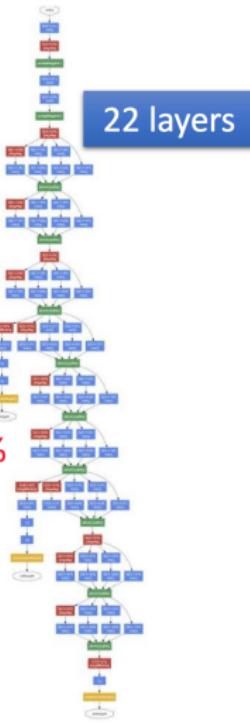
7.3%



VGG (2014)

19 layers

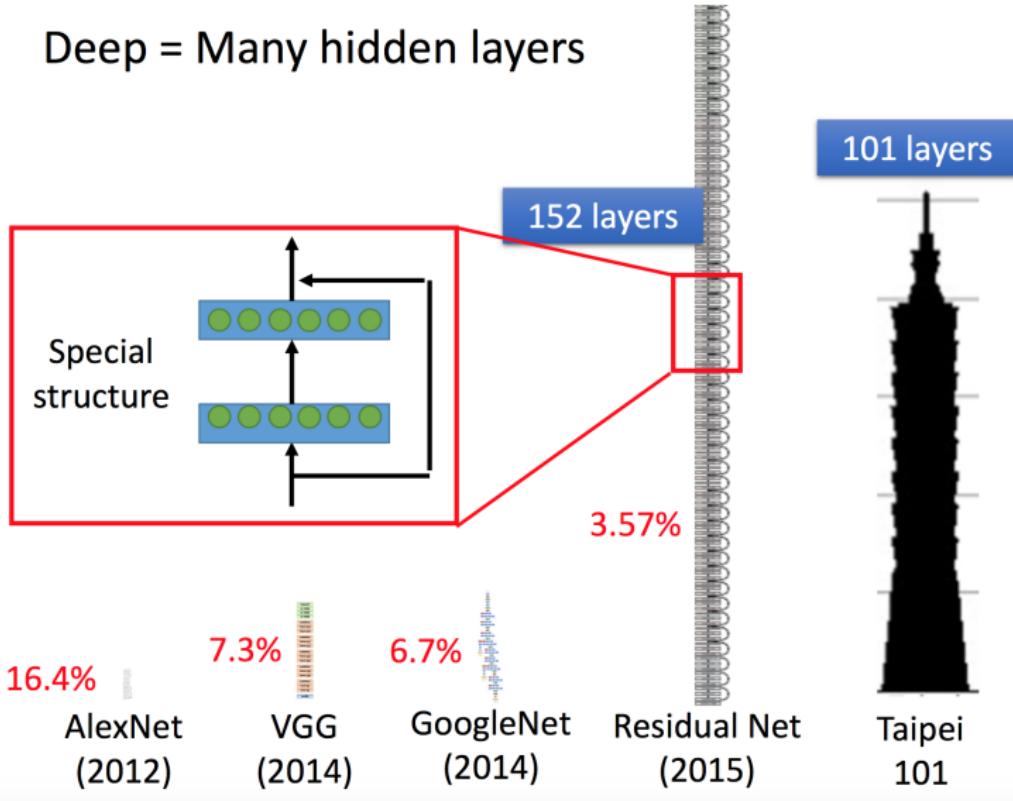
6.7%



GoogleNet (2014)

Exemples de réseaux connus

Deep = Many hidden layers



Les principaux réseaux de neurones

[https://towardsdatascience.com/
illustrated-10-cnn-architectures-95d78ace614d](https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d)

Pourquoi le deep

Un exemple :

- ▶ Apprendre une fonction AND : une seule couche

Pourquoi le deep

Un exemple :

- ▶ Apprendre une fonction AND : une seule couche
- ▶ Apprendre une fonction OR : une seule couche

Pourquoi le deep

Un exemple :

- ▶ Apprendre une fonction AND : une seule couche
 - ▶ Apprendre une fonction OR : une seule couche
 - ▶ Apprendre une fonction XOR : une seule couche ne suffit pas
- ...

Pourquoi le deep

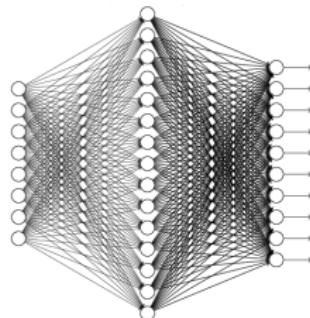
Théorème d'universalité

Toute fonction continue $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ peut être réalisée par un réseau de neurone à une couche cachée.

Pourquoi le deep

Théorème d'universalité

Toute fonction continue $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ peut être réalisée par un réseau de neurone à une couche cachée.

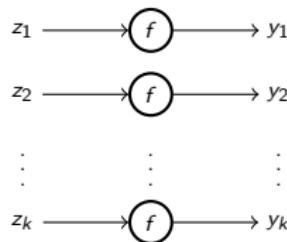


Question : pourquoi des réseaux profonds au lieu de réseaux à une couche ?

Couche de sortie

La couche de Softmax :

Ce que nous avons vu jusque là :

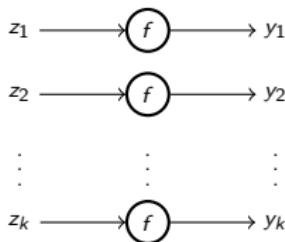


Problème : les sorties y_i prennent des valeurs quelconques et difficiles à interpréter ...

Couche de sortie

La couche de Softmax :

Ce que nous avons vu jusque là :



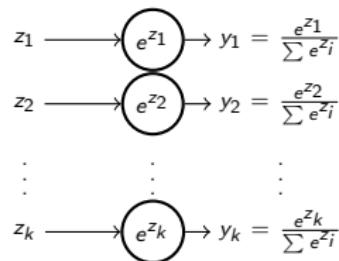
Problème : les sorties y_i prennent des valeurs quelconques et difficiles à interpréter ...

Solution : transformer les sorties en "probabilités".

Couche de sortie

La couche de Softmax :

Solution : ajouter une couche de Softmax :

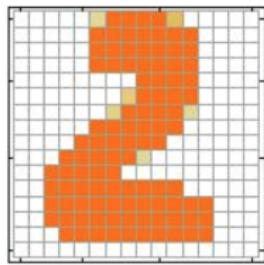


On obtient ainsi une distribution de probabilité :

- ▶ $0 < y_i < 1, \forall i,$
- ▶ $\sum_{i=1}^k y_i = 1.$

Exemple : reconnaissance d'écriture

Input

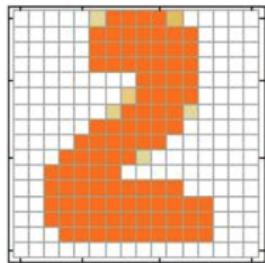


Output



Exemple : reconnaissance d'écriture

Input

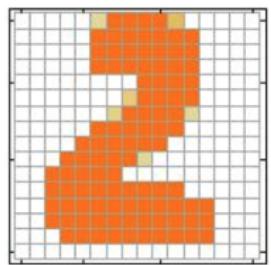


$16 \times 16 = 256$

Output

Exemple : reconnaissance d'écriture

Input



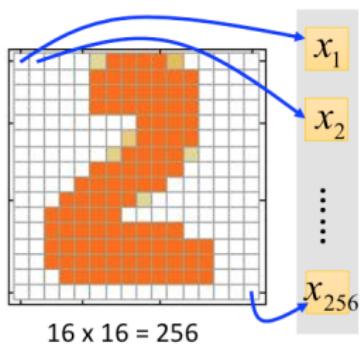
$16 \times 16 = 256$

Output



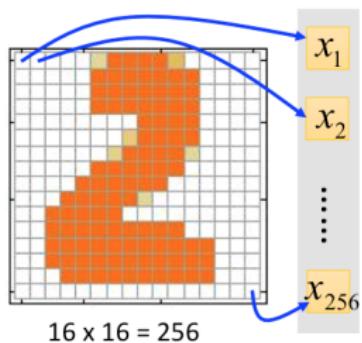
Exemple : reconnaissance d'écriture

Input **Output**



Exemple : reconnaissance d'écriture

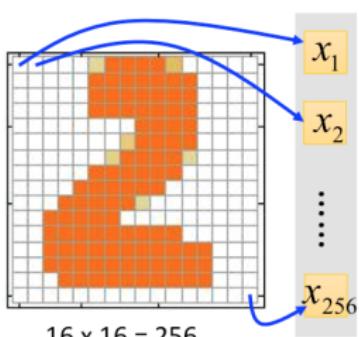
Input **Output**



Ink \rightarrow 1
No ink \rightarrow 0

Exemple : reconnaissance d'écriture

Input



Ink → 1
No ink → 0

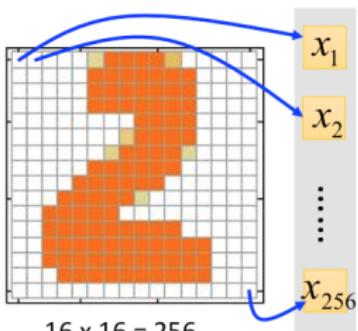
Output

y_1	is 1
y_2	is 2
⋮	⋮
y_{10}	is 0

Each dimension represents the confidence of a digit.

Exemple : reconnaissance d'écriture

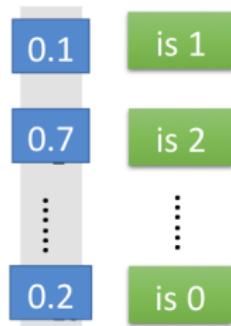
Input



$$16 \times 16 = 256$$

Ink \rightarrow 1
No ink \rightarrow 0

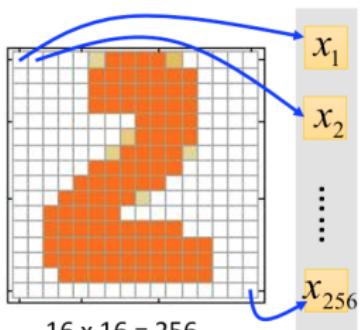
Output



Each dimension represents
the confidence of a digit.

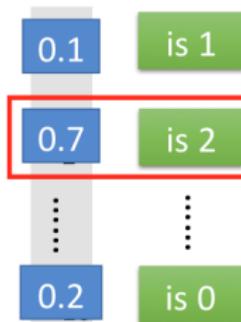
Exemple : reconnaissance d'écriture

Input



Ink → 1
No ink → 0

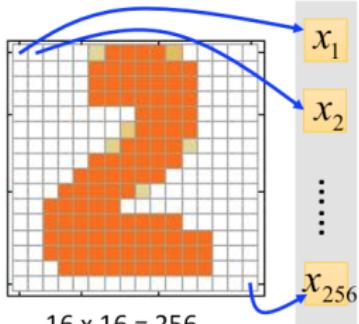
Output



Each dimension represents
the confidence of a digit.

Exemple : reconnaissance d'écriture

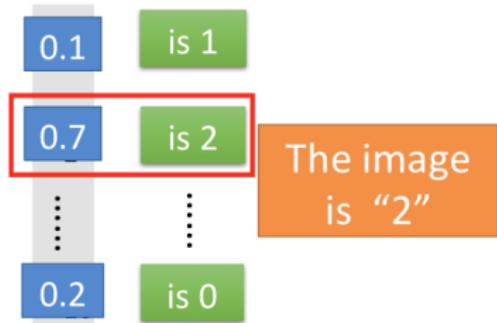
Input



$$16 \times 16 = 256$$

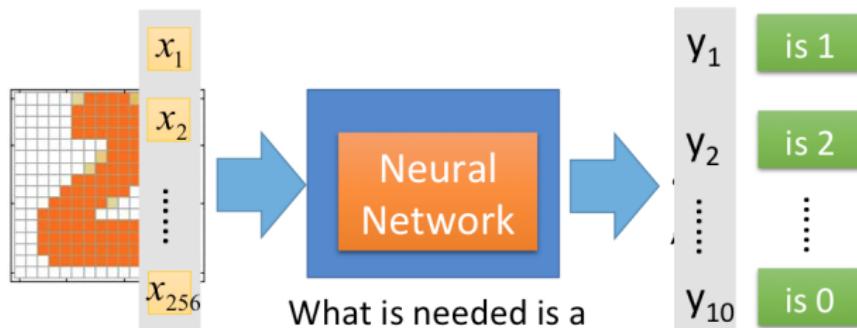
Ink \rightarrow 1
No ink \rightarrow 0

Output



Each dimension represents the confidence of a digit.

Exemple : reconnaissance d'écriture



What is needed is a
function

Input:
256-dim vector

output:
10-dim vector

Questions/Réponses

- ▶ Combien de couches mettre dans le réseaux ?

Questions/Réponses

- ▶ Combien de couches mettre dans le réseaux ?
- ▶ Peut-on concevoir la structure du réseau ?

Questions/Réponses

- ▶ Combien de couches mettre dans le réseaux ?
- ▶ Essayer, apprendre des erreurs et avoir de l'intuition ...
- ▶ Peut-on concevoir la structure du réseau ?

Questions/Réponses

- ▶ Combien de couches mettre dans le réseaux ?
- ▶ Essayer, apprendre des erreurs et avoir de l'intuition ...
- ▶ Peut-on concevoir la structure du réseau ?
- ▶ Réseaux de neurones à convolution ...

Données d'entraînement

Les Images et leurs étiquettes :



"5"



"0"



"4"



"1"



"9"



"2"

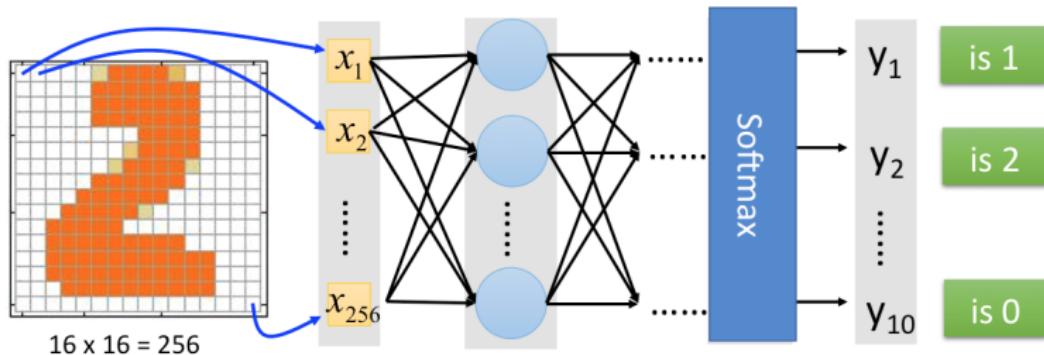


"1"



"3"

Objectif de l'entrainement

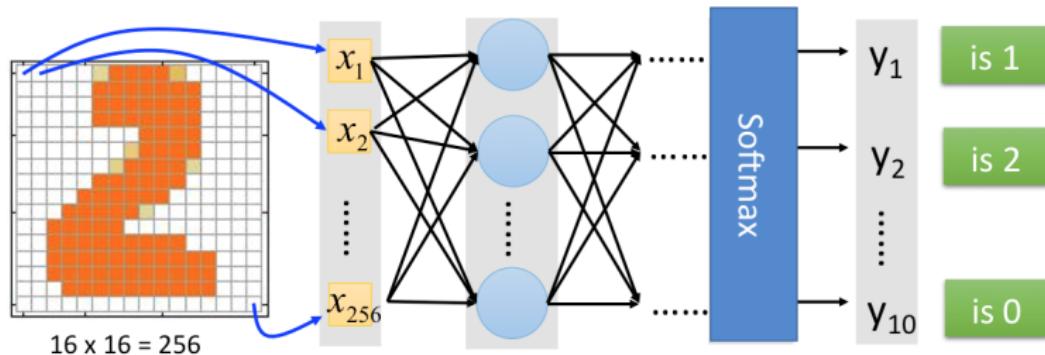


$$16 \times 16 = 256$$

Ink $\rightarrow 1$

No ink $\rightarrow 0$

Objectif de l'entraînement



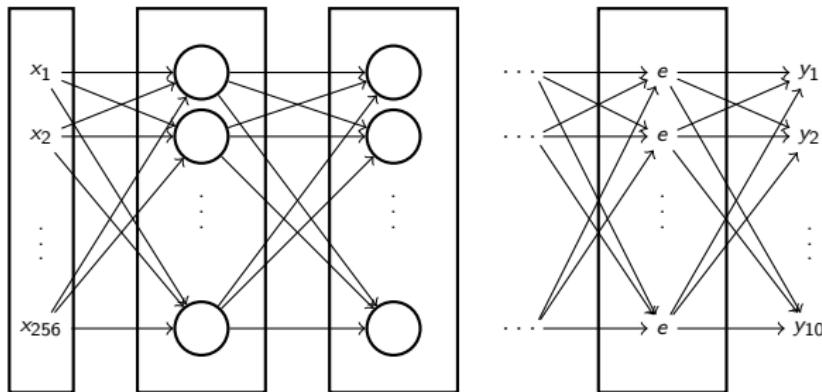
Ink $\rightarrow 1$

No ink $\rightarrow 0$

Input: $\Rightarrow y_1$ a la plus grande valeur,

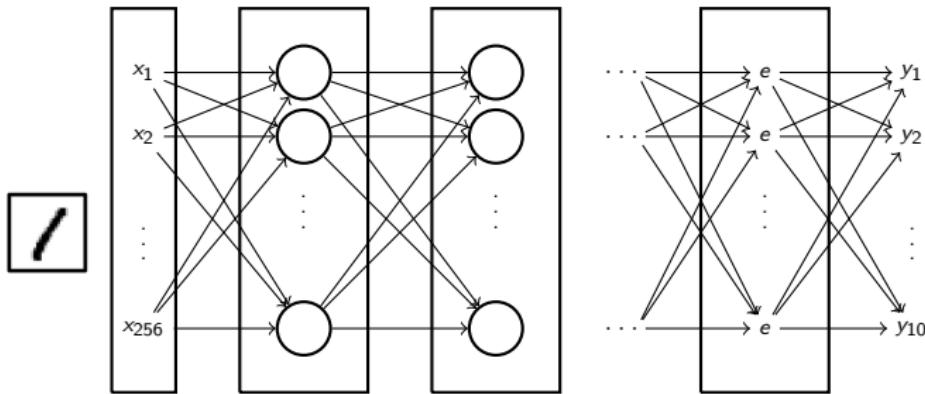
Input: $\Rightarrow y_2$ a la plus grande valeur,

Fonction de perte (Loss Function)



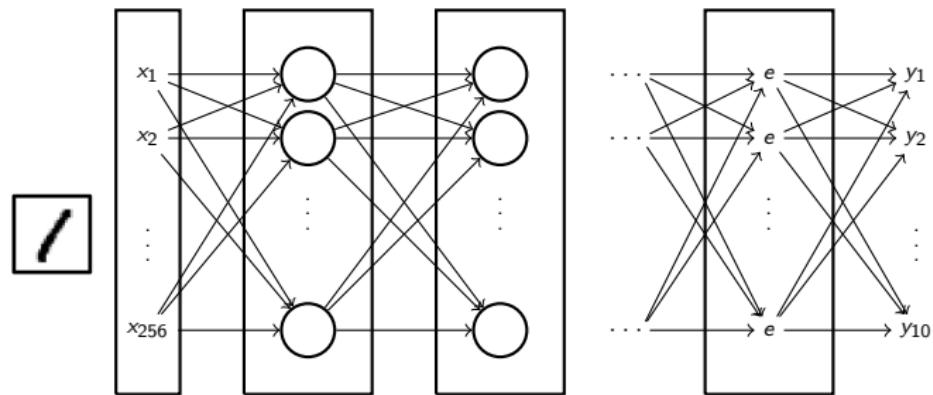
Couche de Softmax

Fonction de perte (Loss Function)



Couche de Softmax

Fonction de perte (Loss Function)

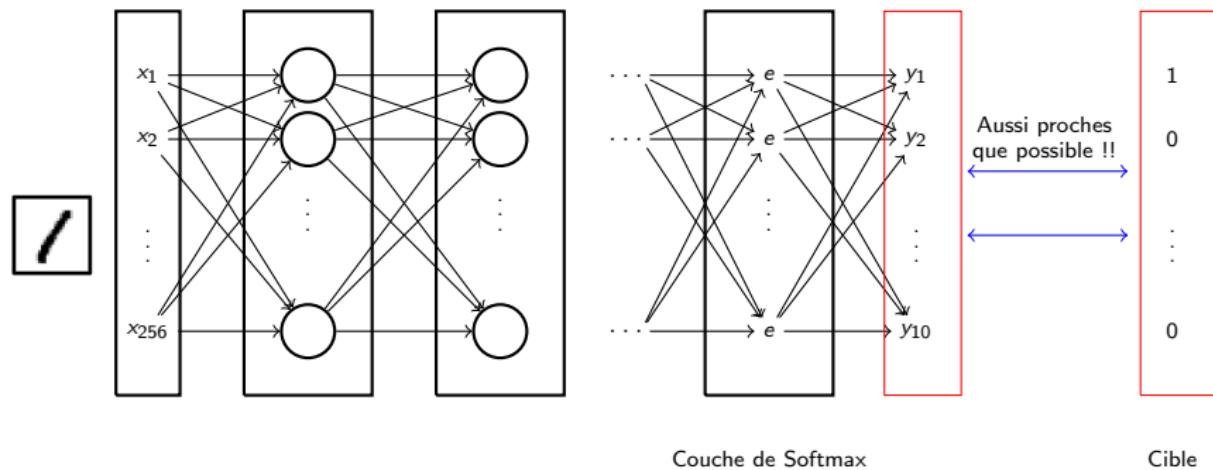


Couche de Softmax

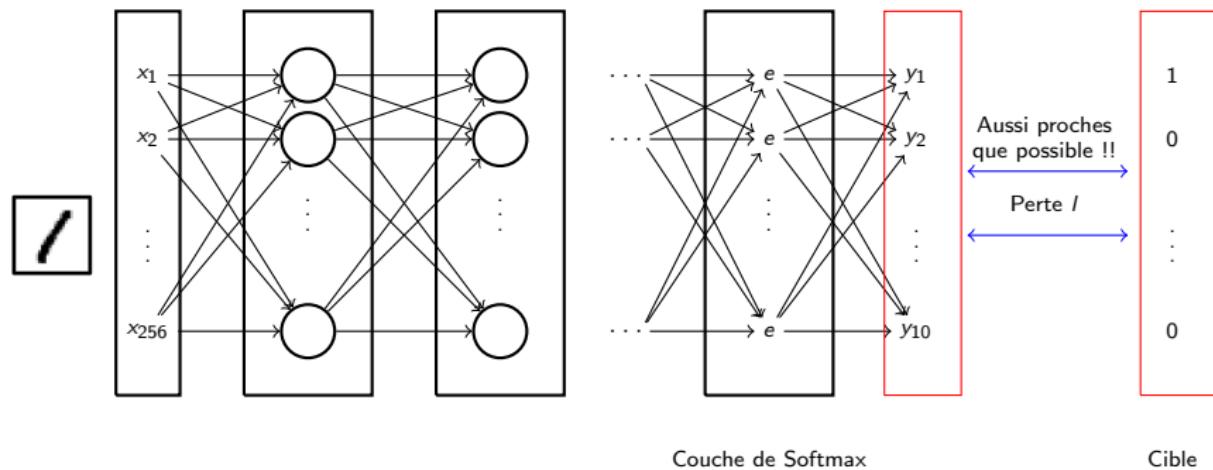
1
0
⋮
0

Cible

Fonction de perte (Loss Function)



Fonction de perte (Loss Function)



Fonction de perte (Loss Function)

- ▶ La fonction de perte peut être :
 - ▶ La somme des carrés des écarts :

$$l = \sum_{i=1}^{10} (y_i - \hat{y}_i)^2.$$

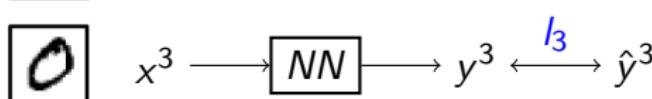
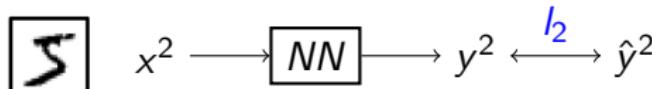
- ▶ La *cross entropy* :

$$l = - \sum_{i=1}^{10} \hat{y}_i \ln(y_i).$$

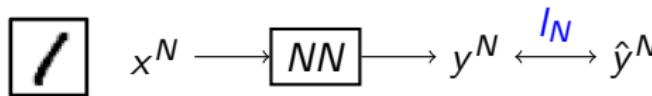
Une bonne fonction (définie par les paramètres que l'on met sur le réseau) doit minimiser la *valeur totale* de la fonction de perte ...

Perte totale

Pour tout l'ensemble de traitement :



⋮ ⋮ ⋮ ⋮ ⋮



Perte totale :

$$L = \sum_{i=1}^N l_i$$

Objectif :
Minimiser cette fonction.

Apprentissage ?

Comment choisir la meilleure fonction de classification ?

Trouver les meilleurs paramètres θ qui minimisent la perte totale L .

- ▶ Solution 1. : Enumérer toutes les possibilités → infaisable !!

Apprentissage ?

Comment choisir la meilleure fonction de classification ?

Trouver les meilleurs paramètres θ qui minimisent la perte totale L .

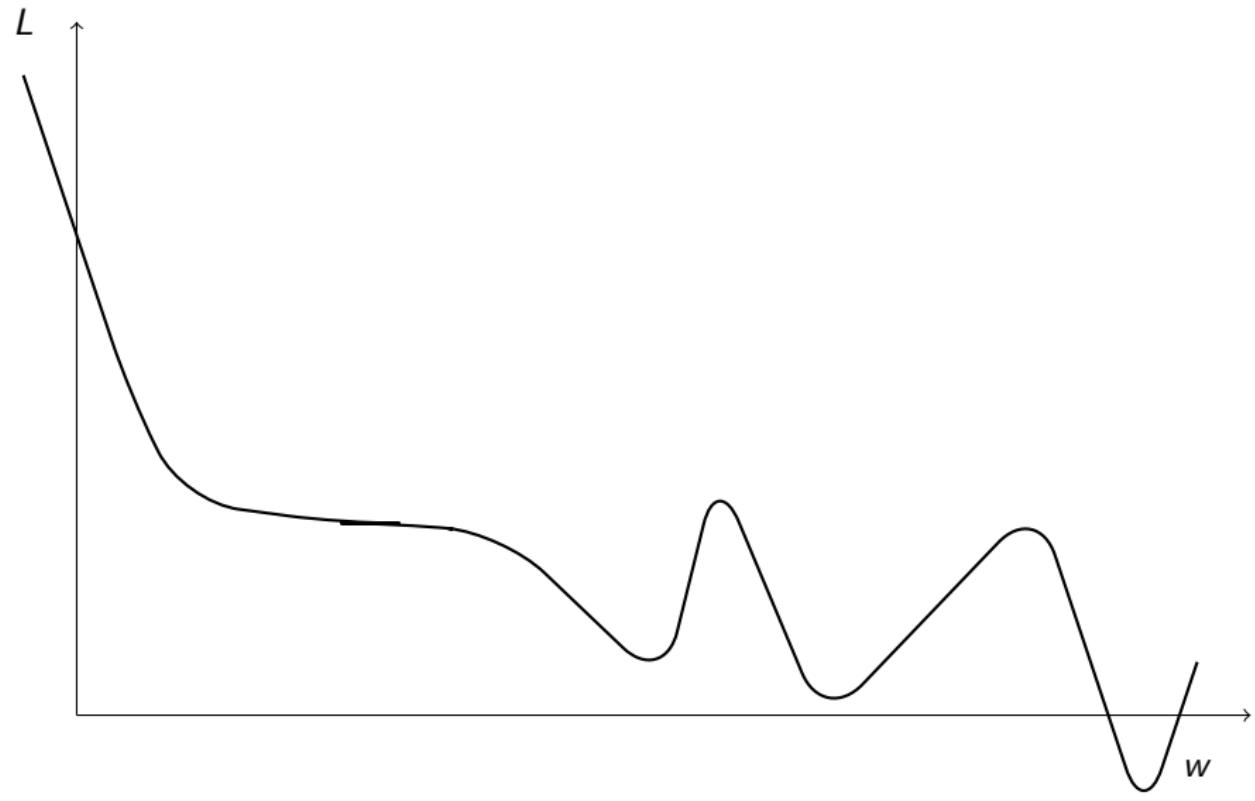
- ▶ Solution 1. : Enumérer toutes les possibilités → infaisable !!
- ▶ Solution 2. : **Descente du Gradient.**

Descente du gradient

Objectif : déterminer les paramètres θ du réseau qui minimisent la perte total L .

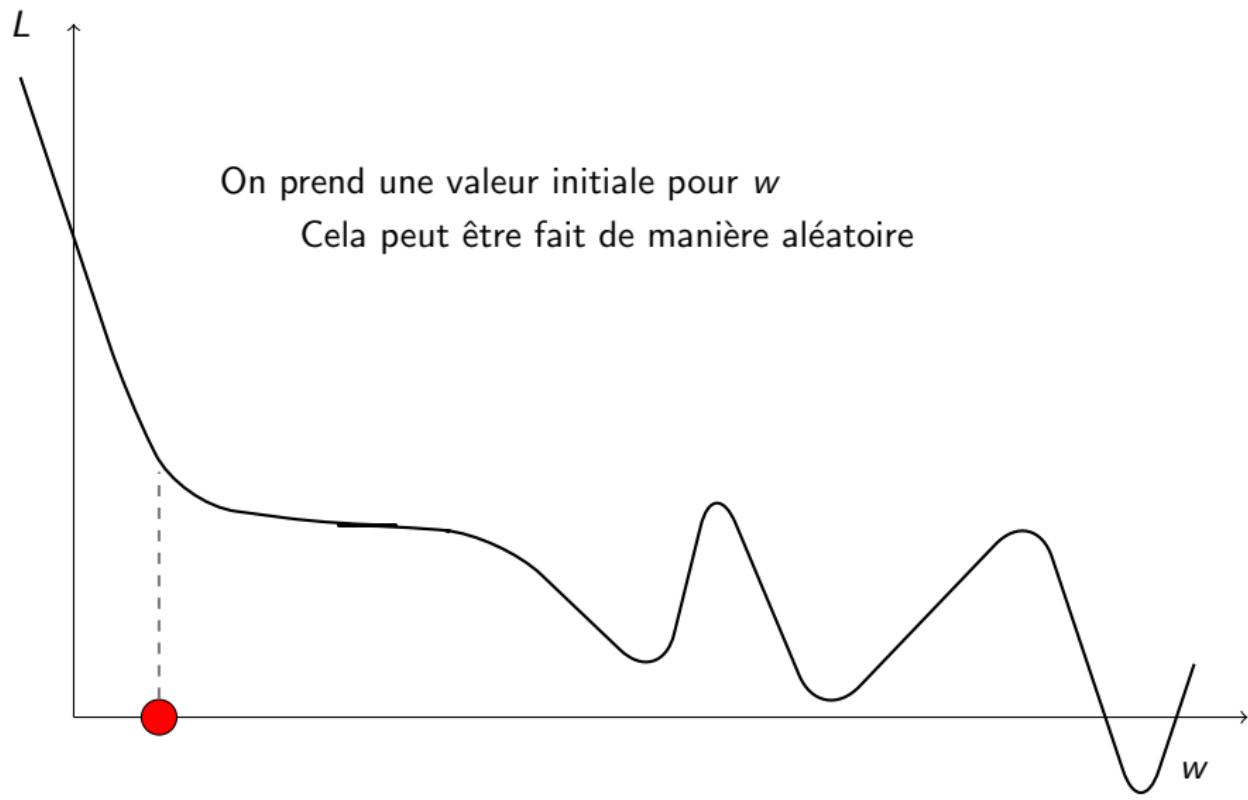
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



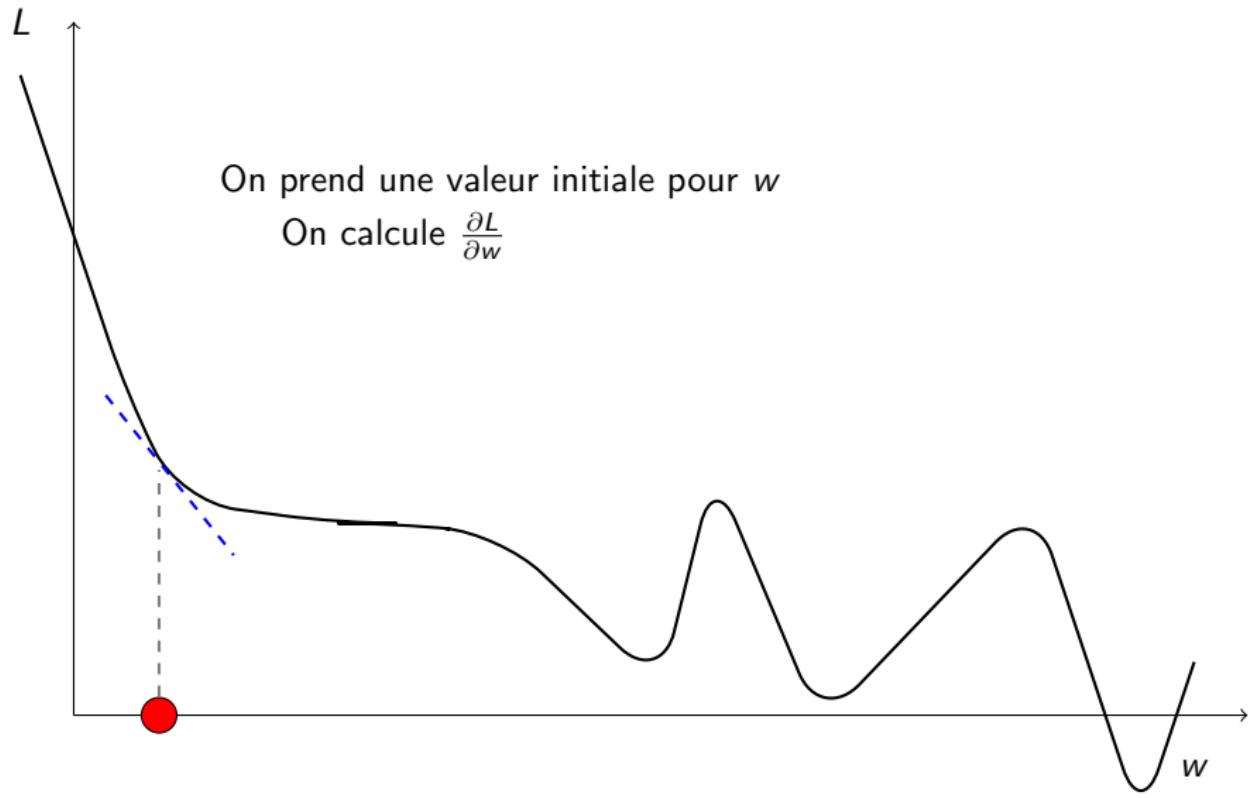
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



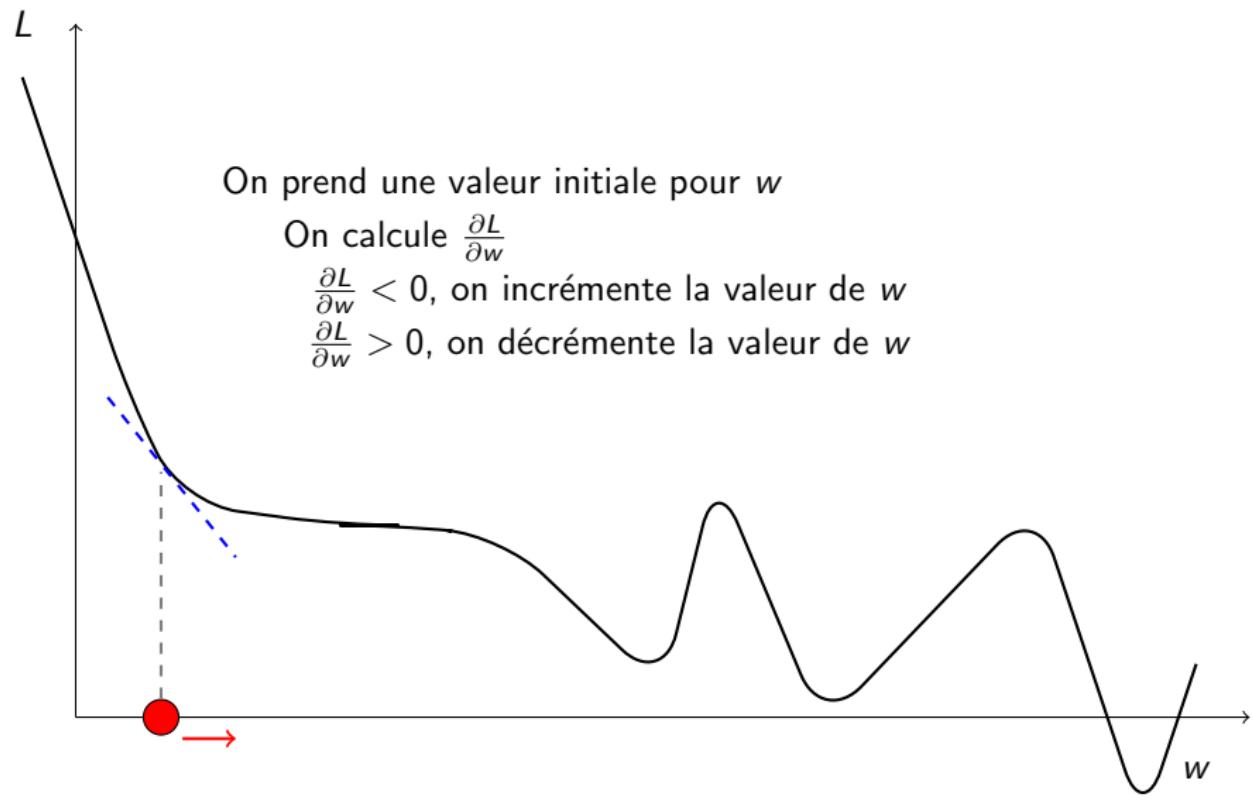
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



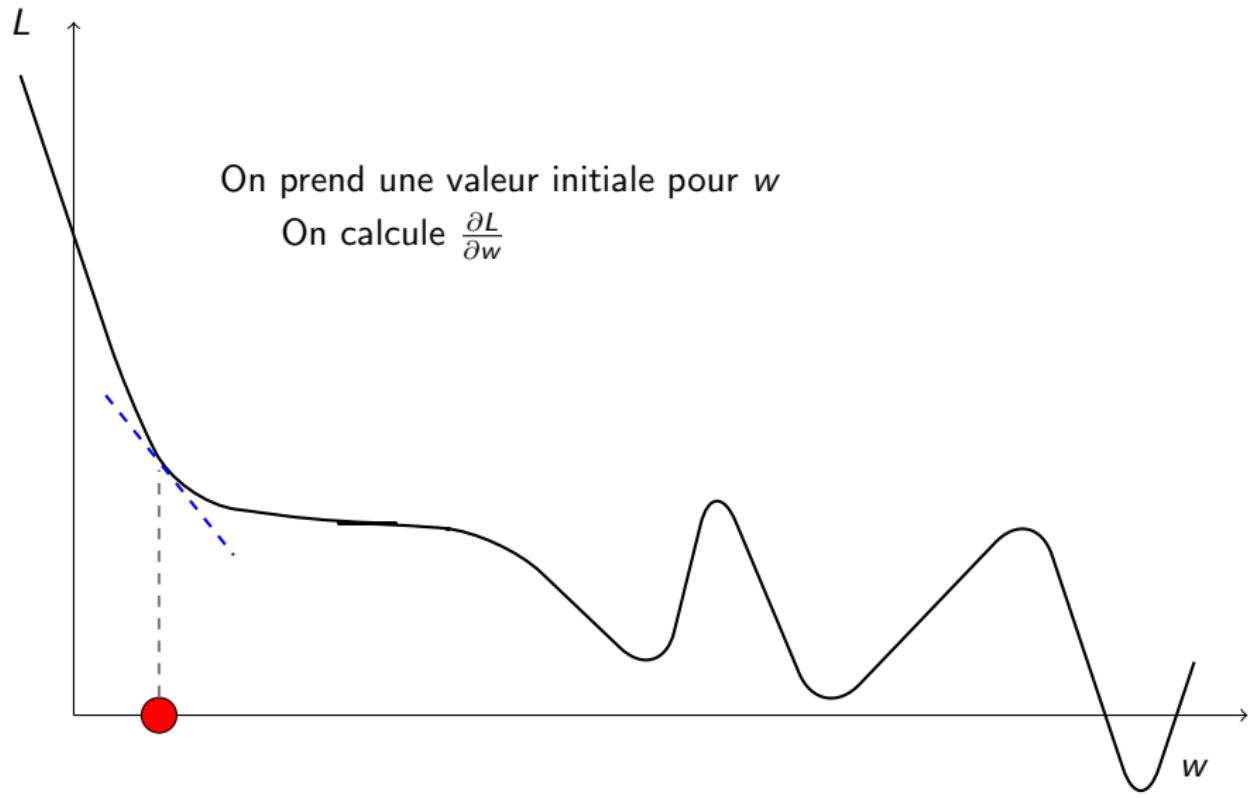
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



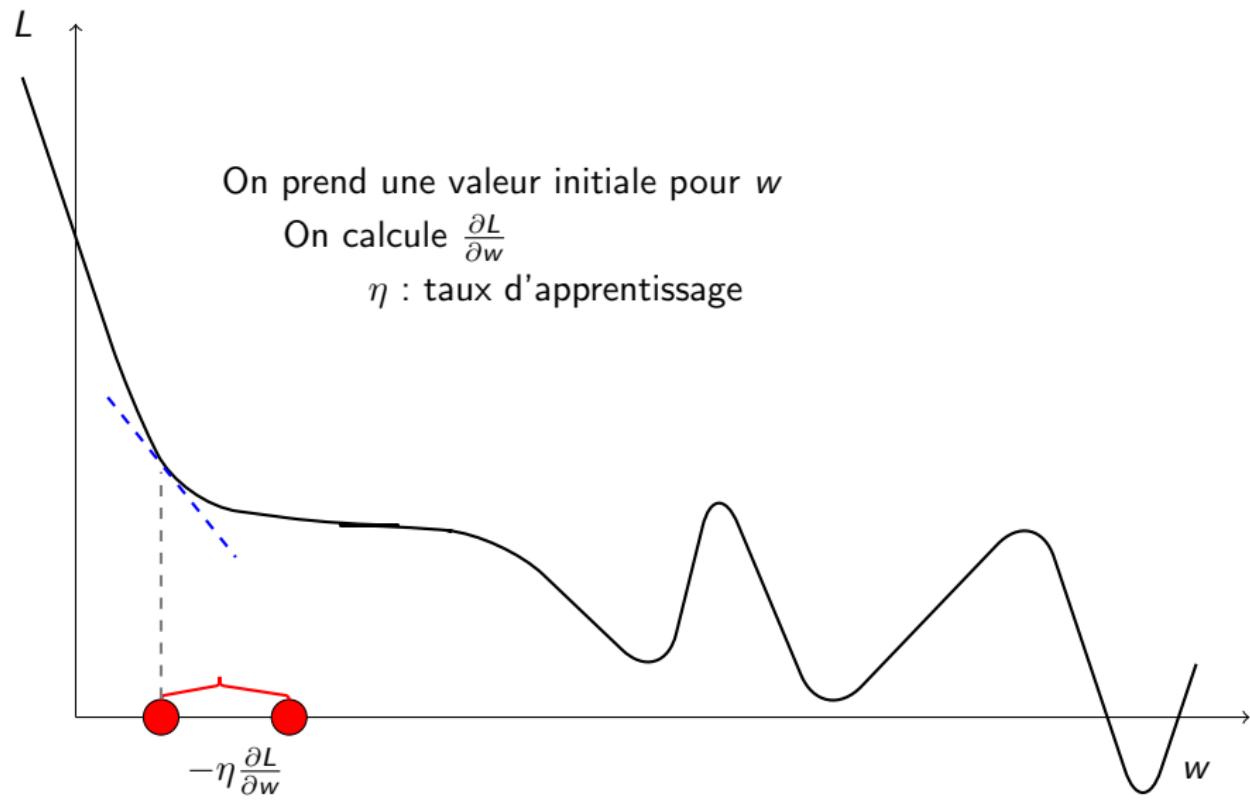
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



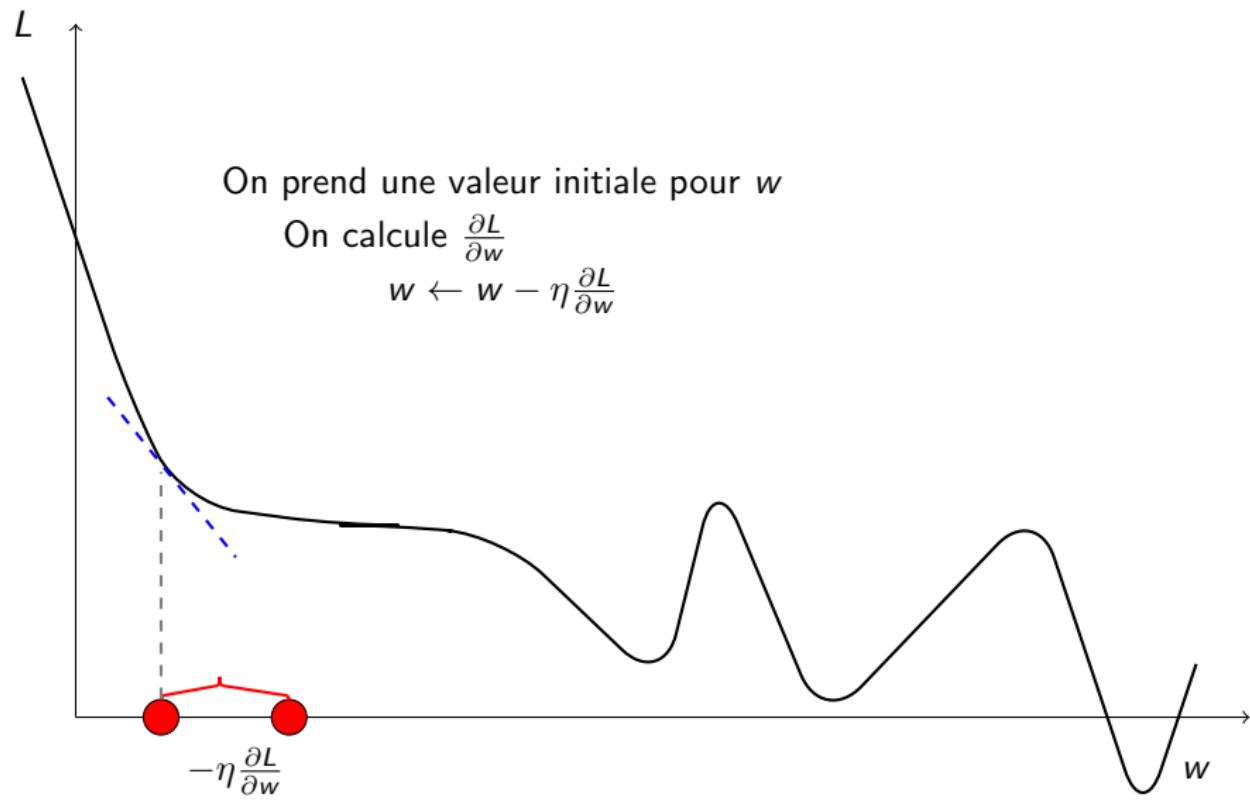
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



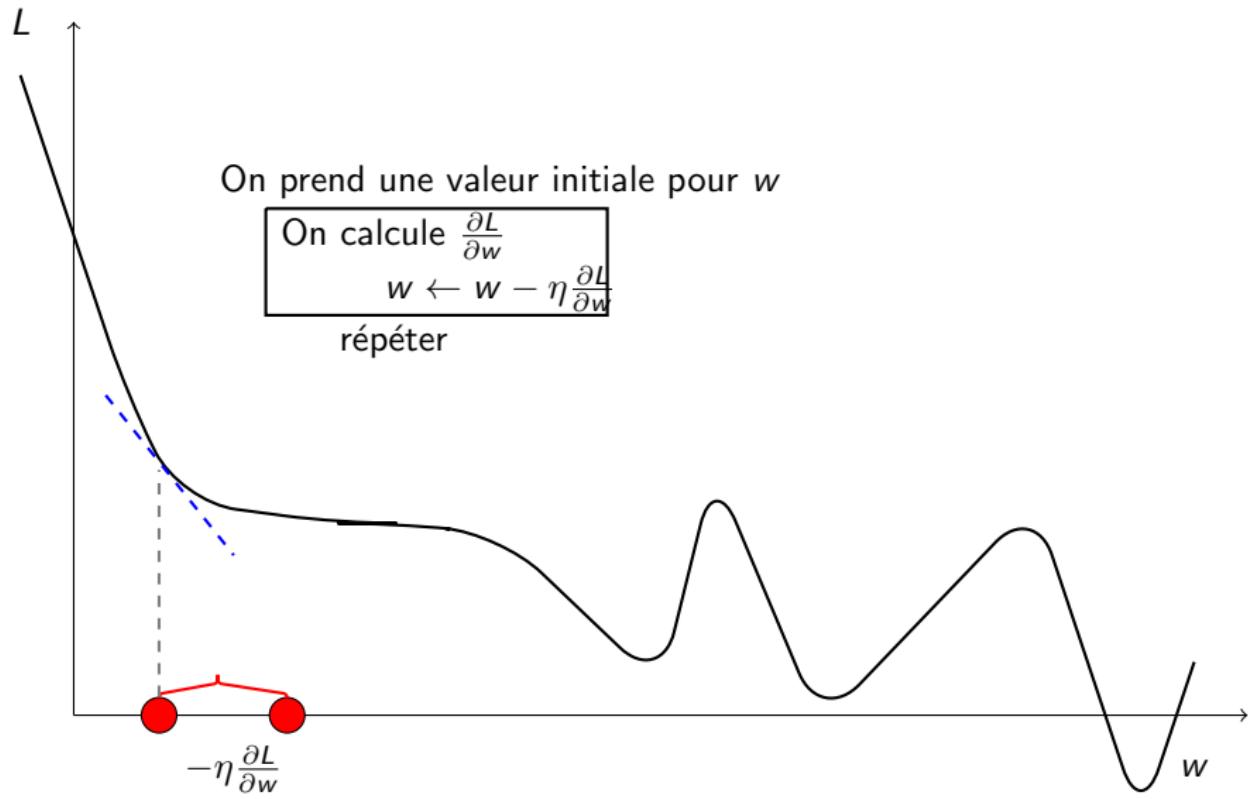
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



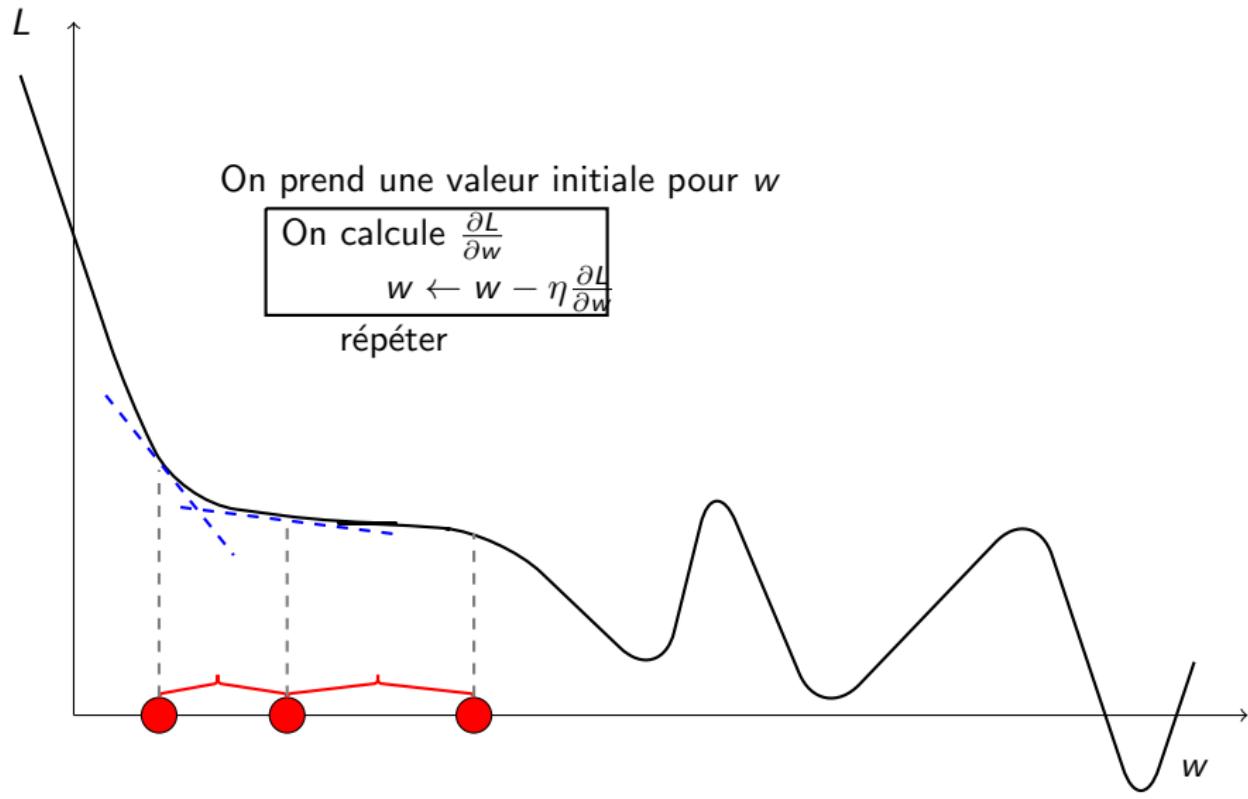
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



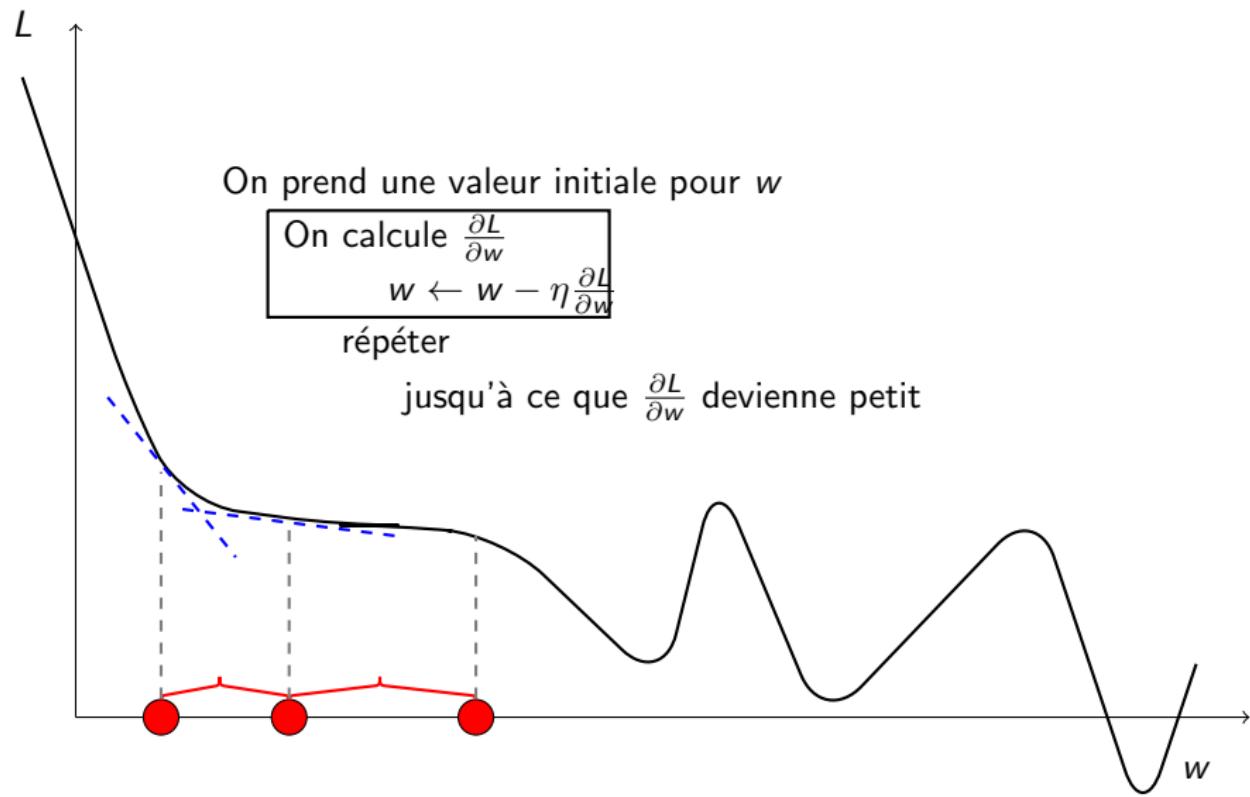
Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .



Descente du gradient

Trouver les paramètres θ du réseau qui minimisent la fonction de perte totale L .

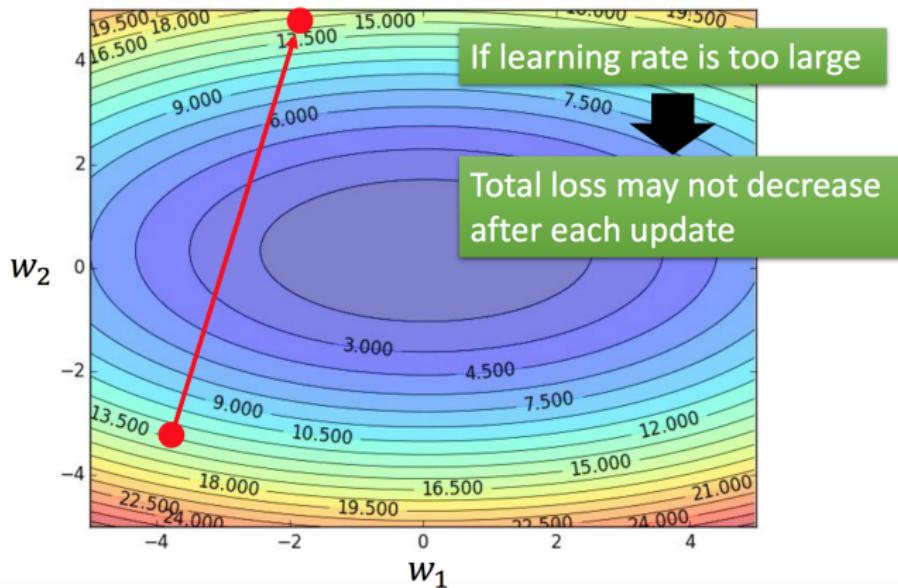


Descente du gradient

de l'importance du choix du taux d'apprentissage η :

Descente du gradient

de l'importance du choix du taux d'apprentissage η :

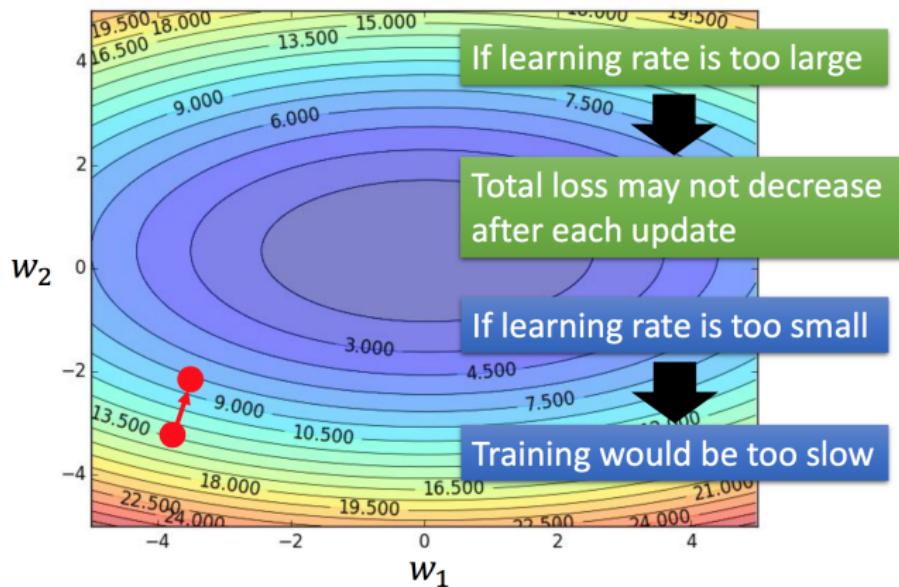


Descente du gradient

de l'importance du choix du taux d'apprentissage η :

Descente du gradient

de l'importance du choix du taux d'apprentissage η :



Descente du gradient

de l'importance du choix du taux d'apprentissage η :

- ▶ Un taux trop petit garantit une convergence vers un minimum mais le temps de convergence peut être trop grand
- ▶ Un taux trop grand peut faire "sauter" le minimum ..

Descente du gradient

de l'importance du choix du taux d'apprentissage η :

Descente du gradient

de l'importance du choix du taux d'apprentissage η :

Une solution simple : réduire le taux par un facteur toutes les K itérations

- ▶ Initialement, on est loin de la destination, on utilise donc un pas assez grand,
- ▶ Après quelques itérations, on est proche de la destination, on réduit donc le pas.
- ▶ En général, on utilise la mise à jour suivante :

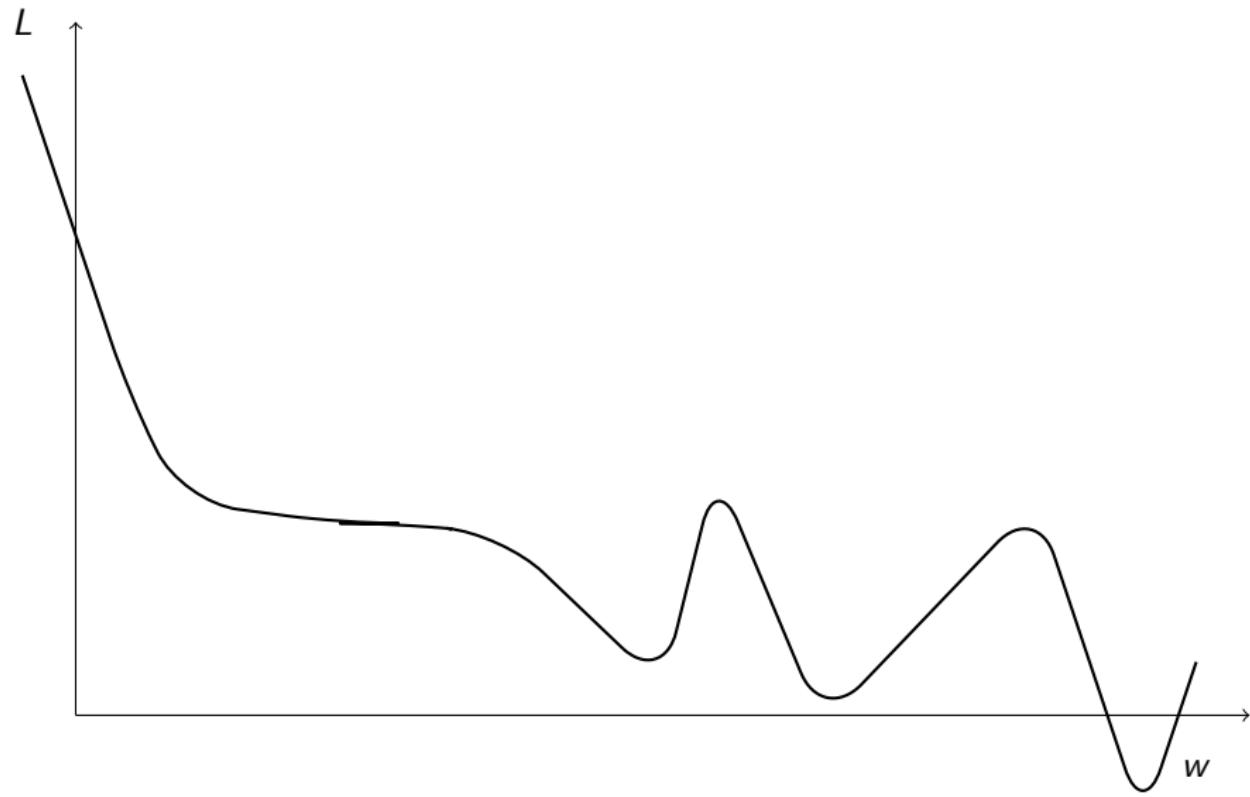
$$\eta^{(t)} = \frac{\eta}{\sqrt{t + 1}}$$

Descente du gradient

- ▶ La descente du gradient ne garantie pas de trouver un minimum global,
- ▶ Néanmoins, c'est LA **méthode d'apprentissage** du deep ...

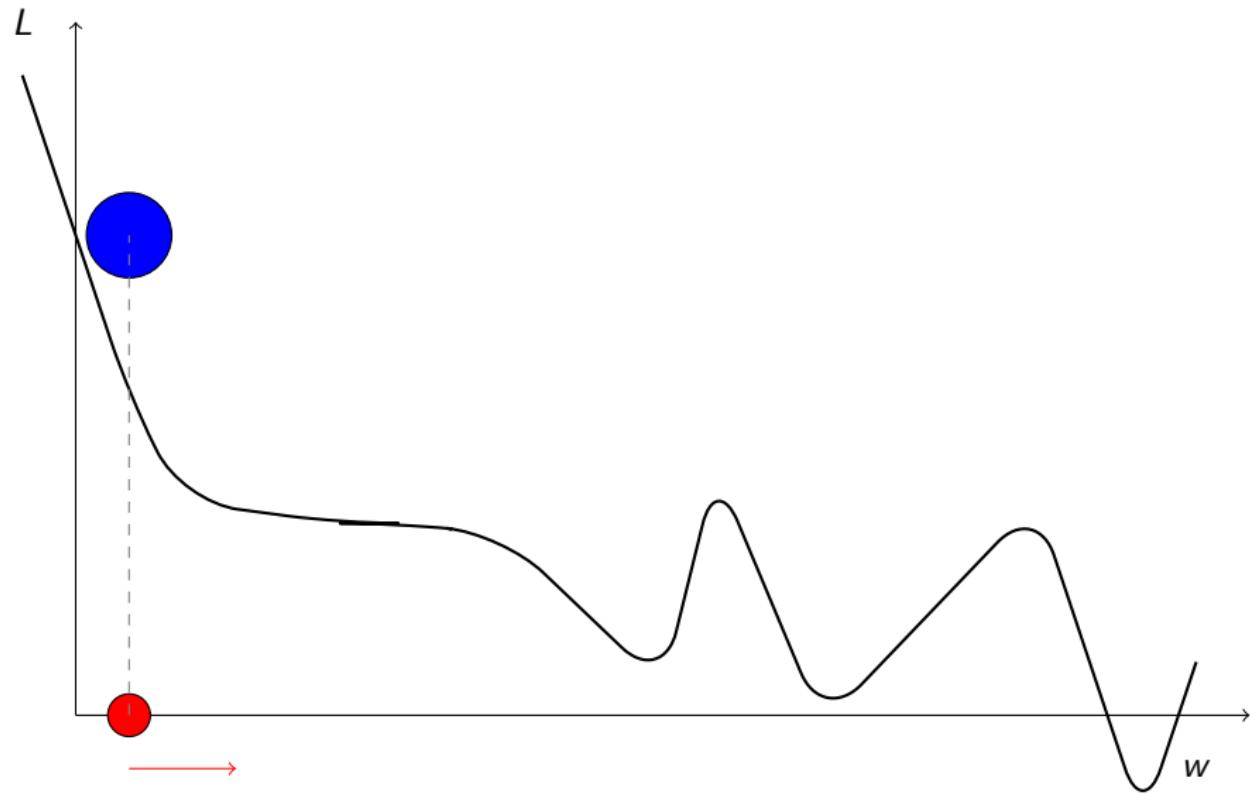
Descente du gradient

Problème : minimum mais local



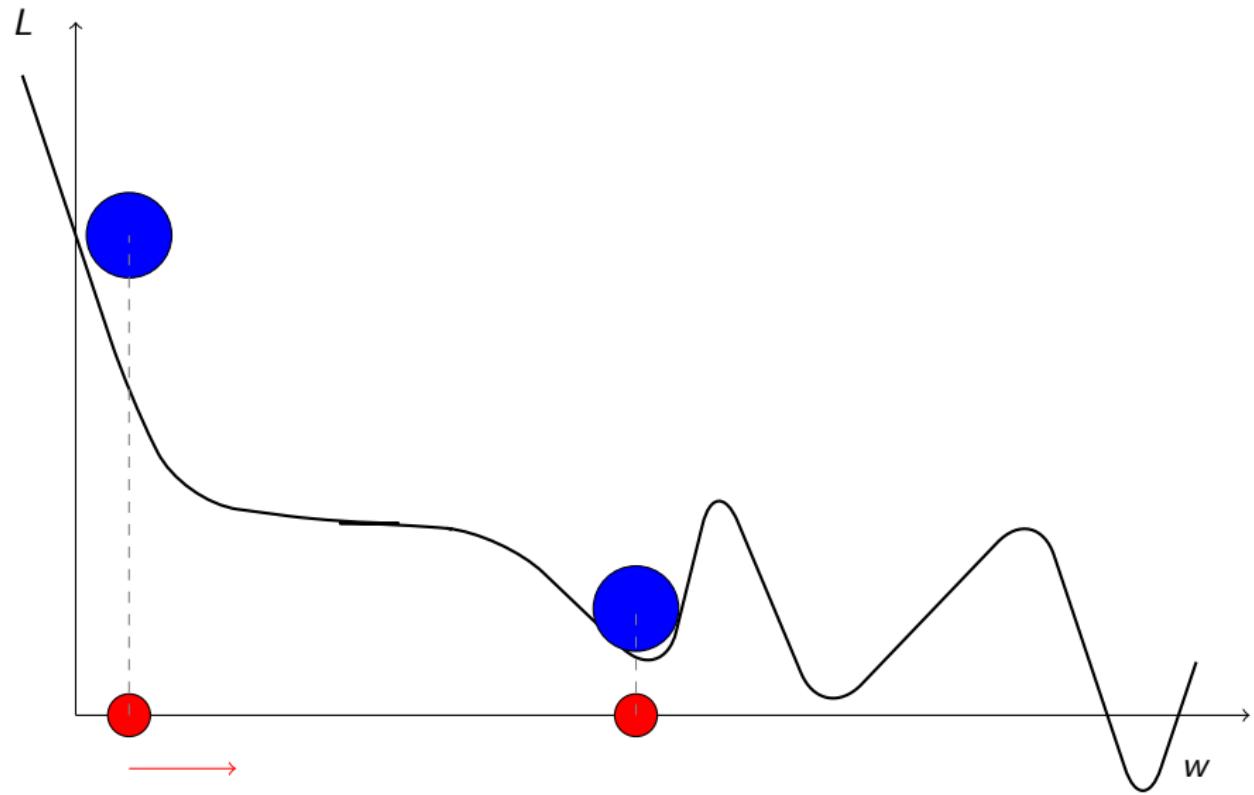
Descente du gradient

Problème : minimum mais local ...



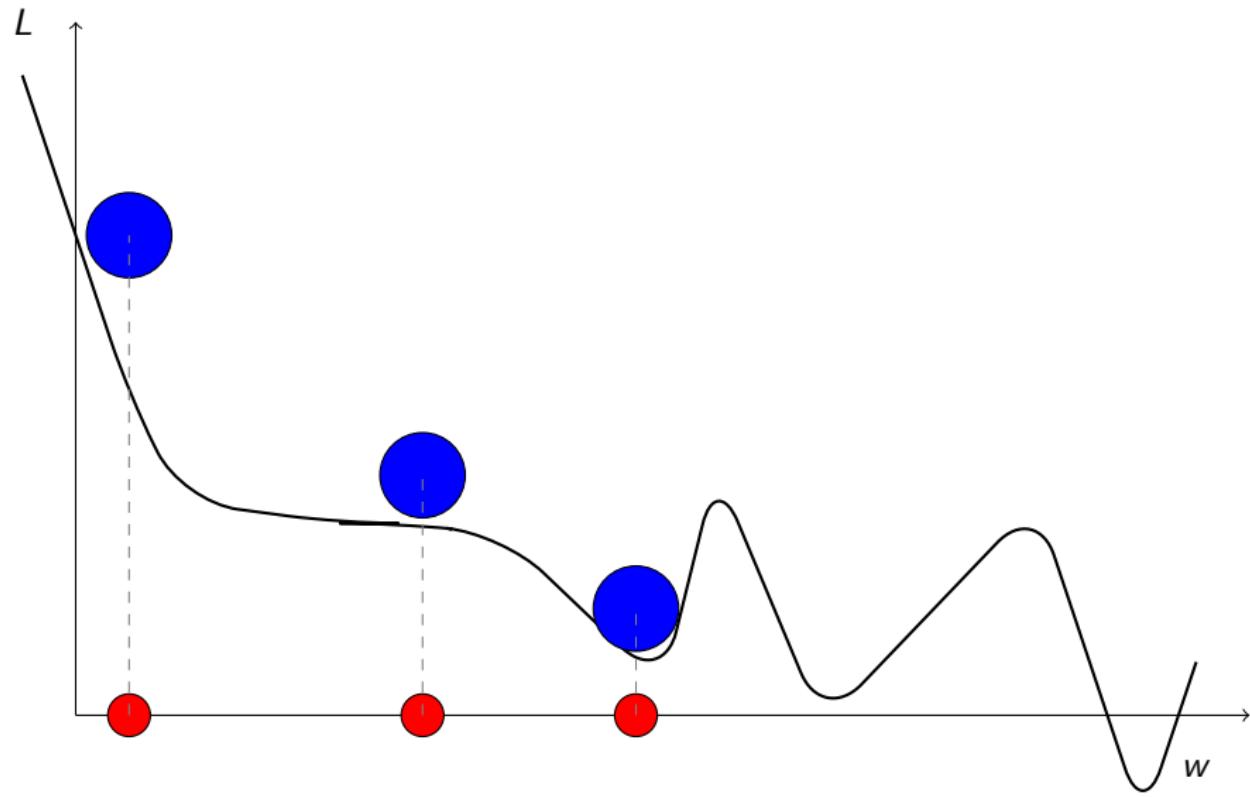
Descente du gradient

Problème : minimum mais local



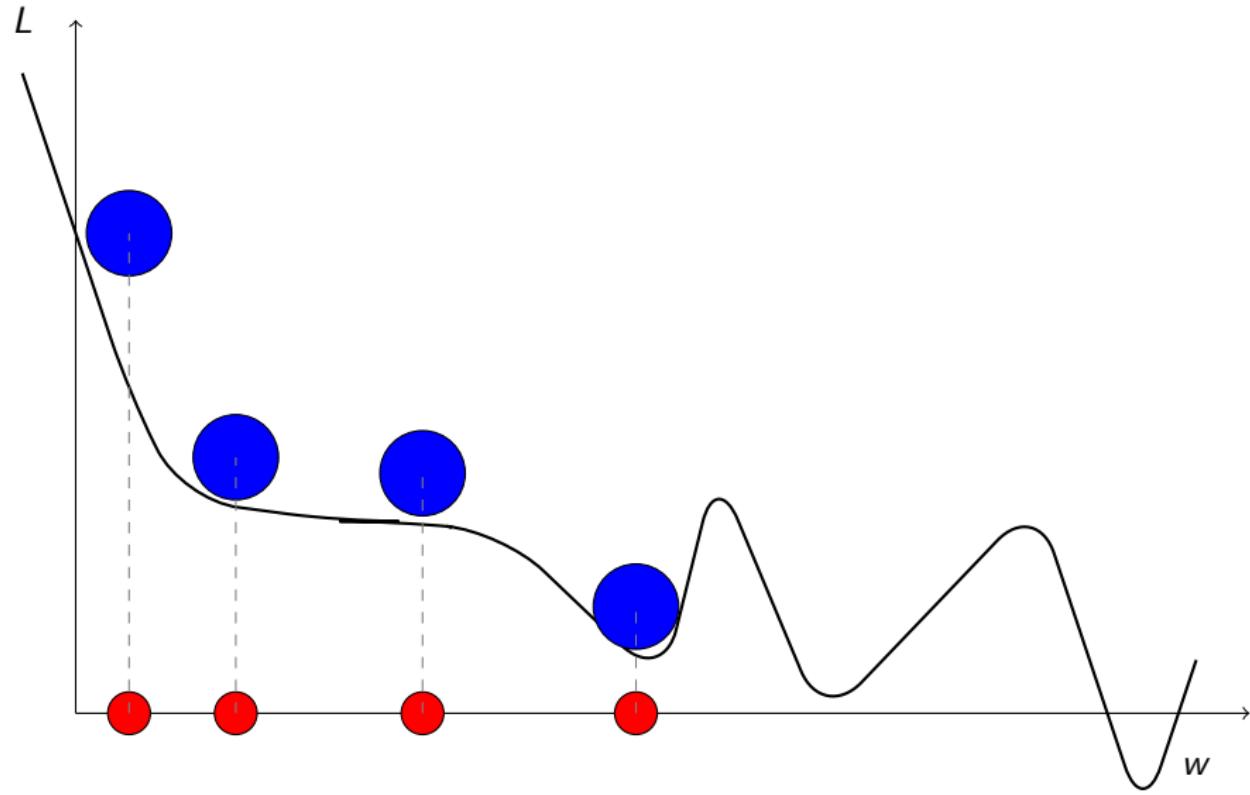
Descente du gradient

Problème : minimum mais local



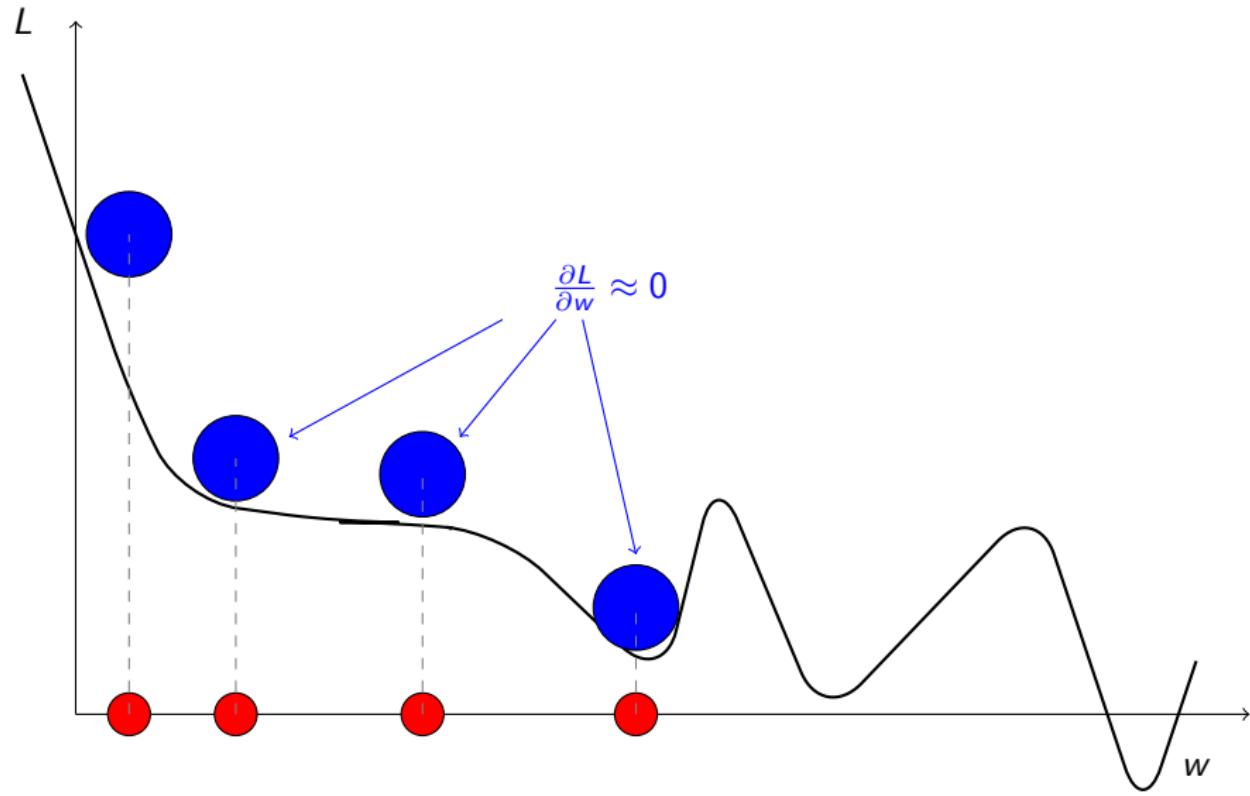
Descente du gradient

Problème : minimum mais local



Descente du gradient

Problème : minimum mais local



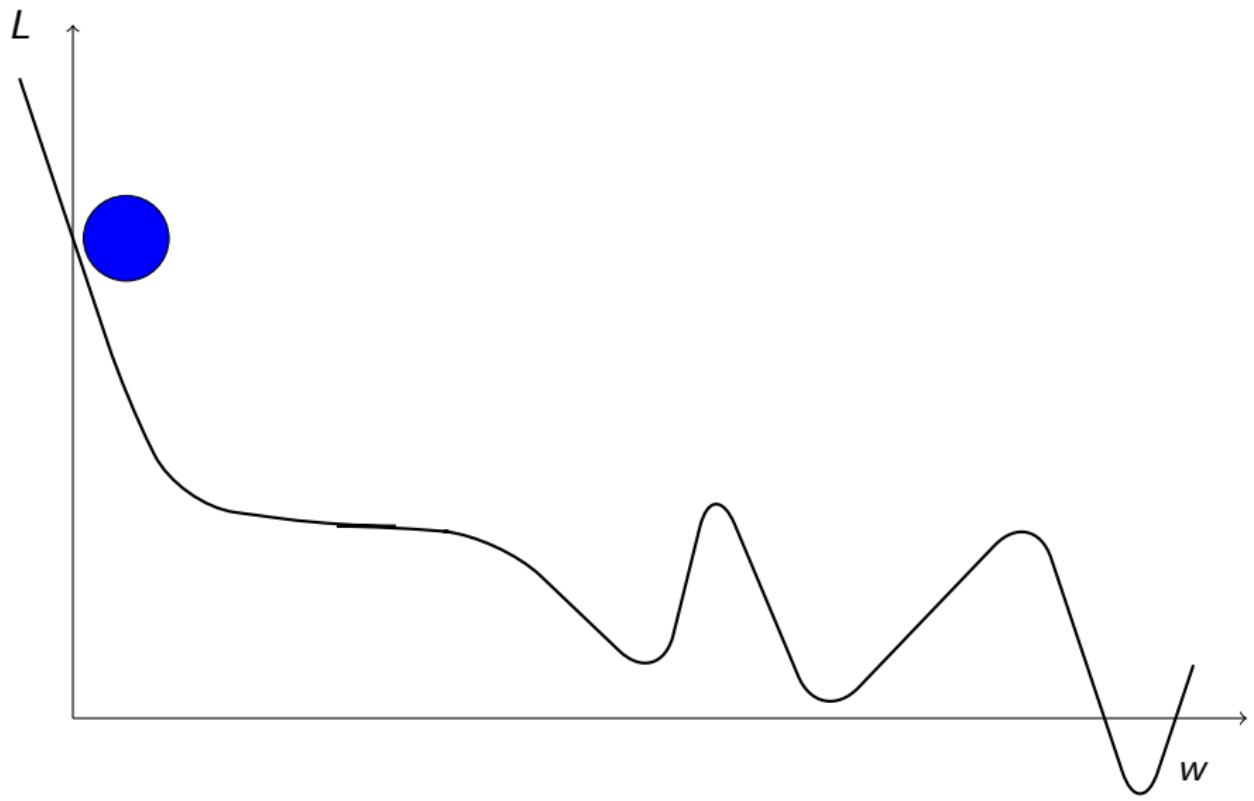
Descente du gradient

Problème : minimum mais local

Idée : s'inspirer de la physique et d'une balle qui dévale une pente

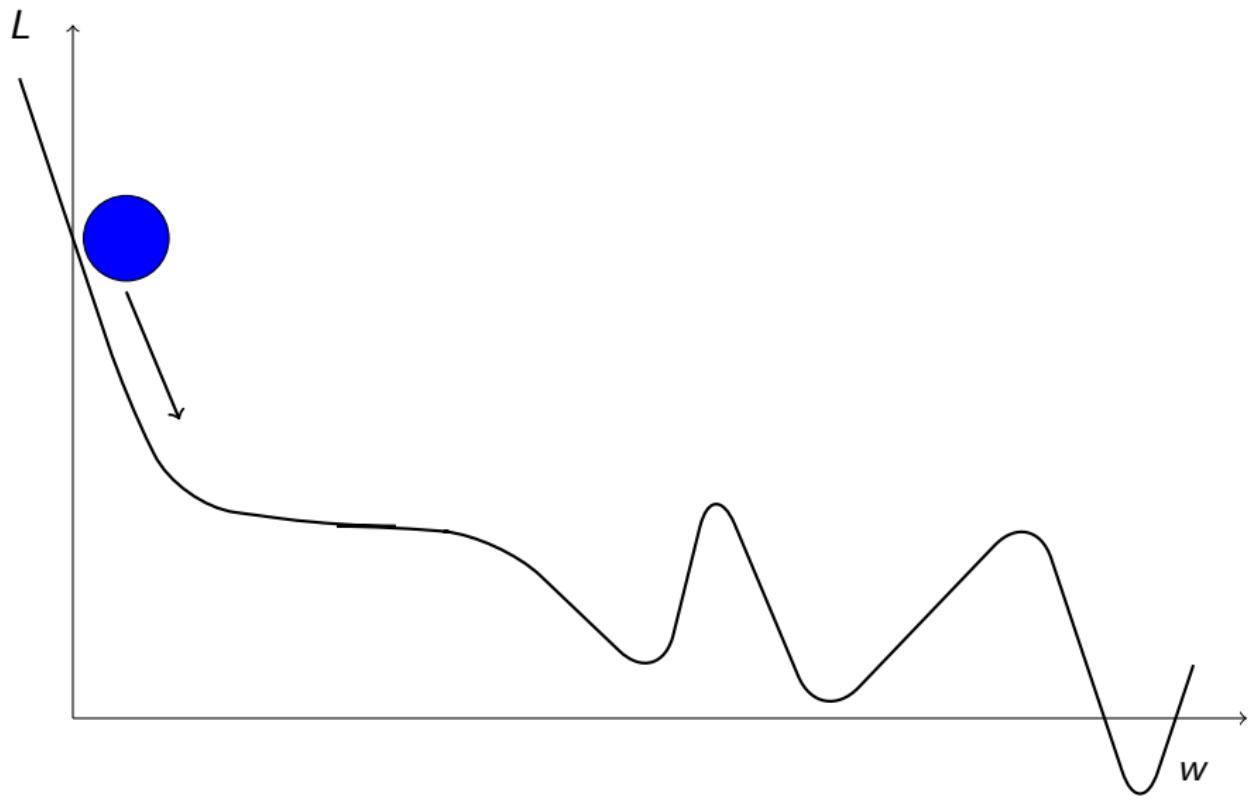
Descente du gradient

Méthode du Momentum :



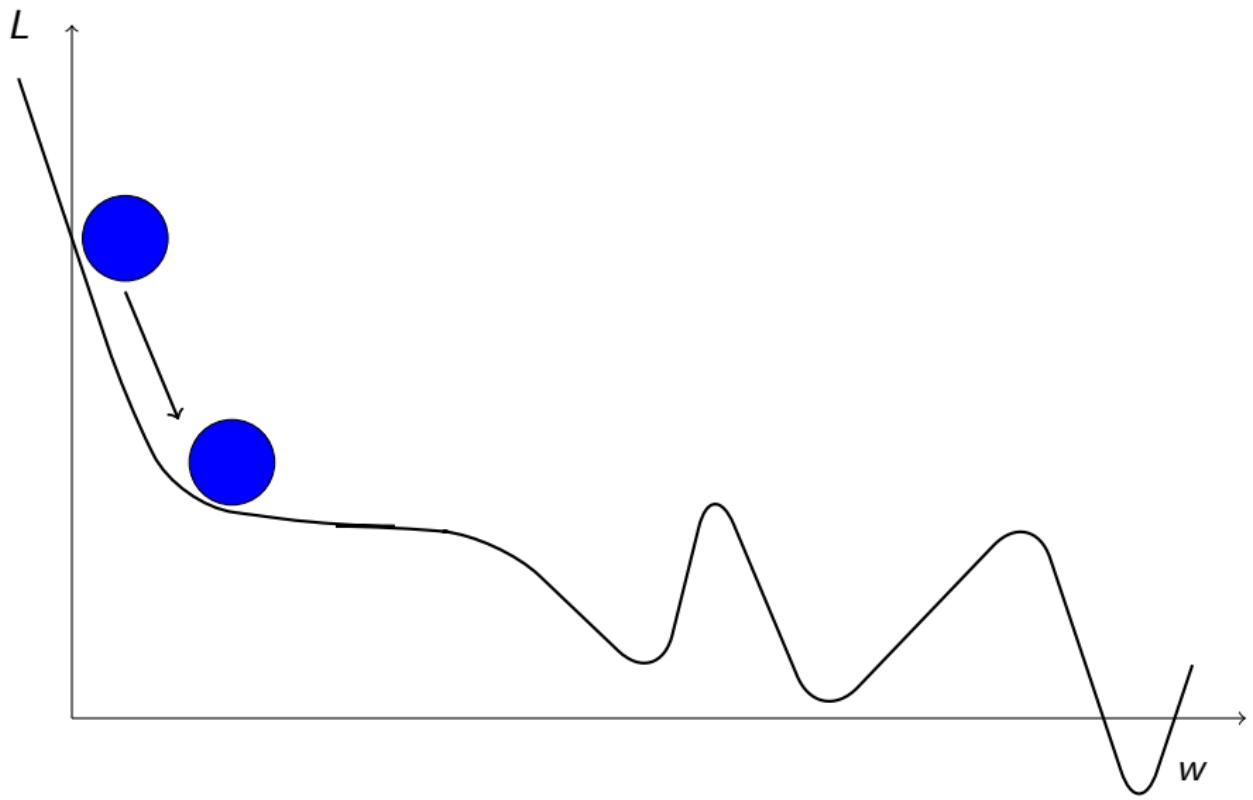
Descente du gradient

Méthode du Momentum :



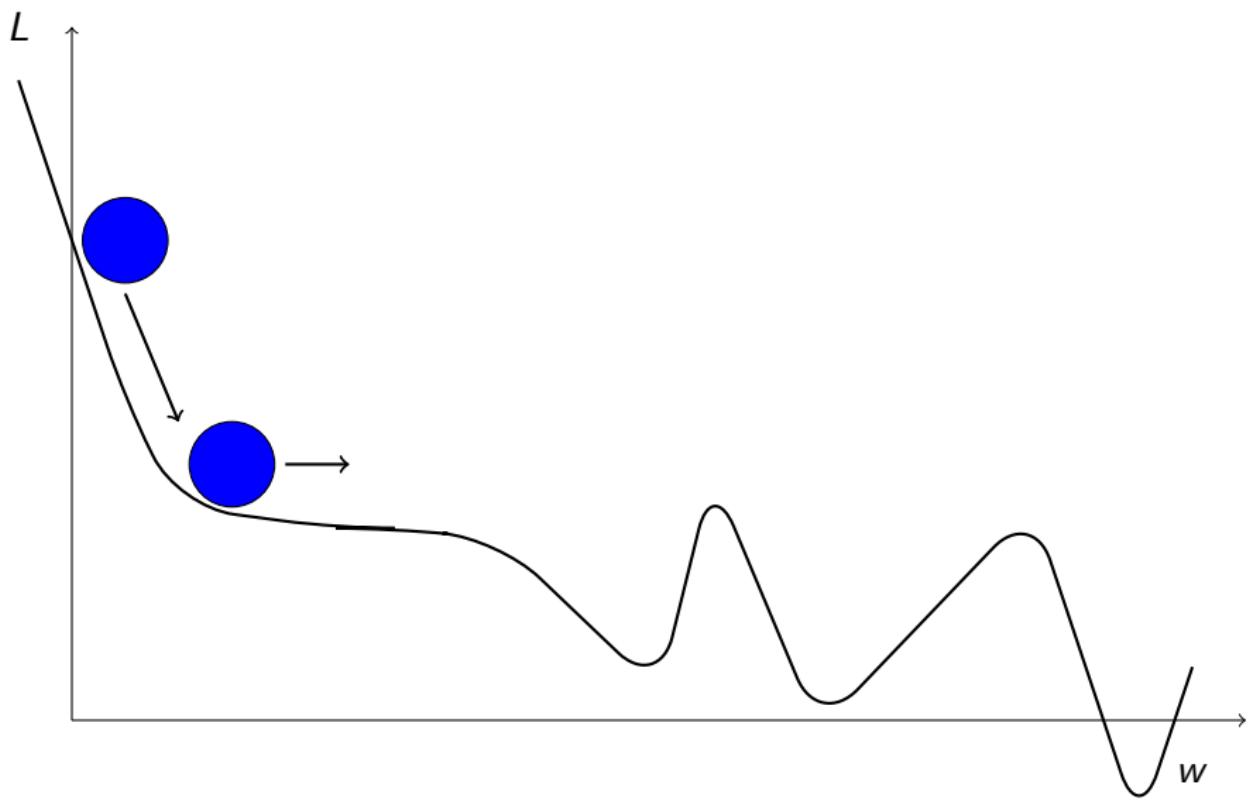
Descente du gradient

Méthode du Momentum :



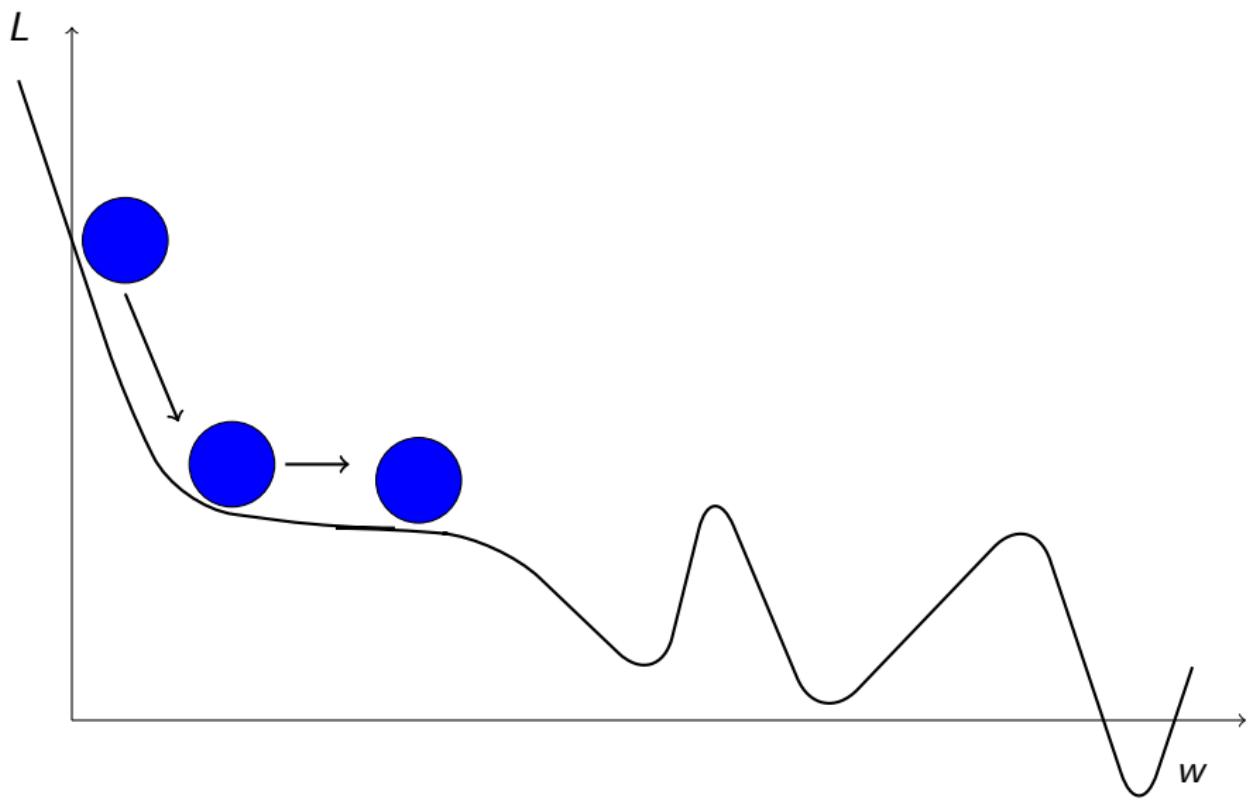
Descente du gradient

Méthode du Momentum :



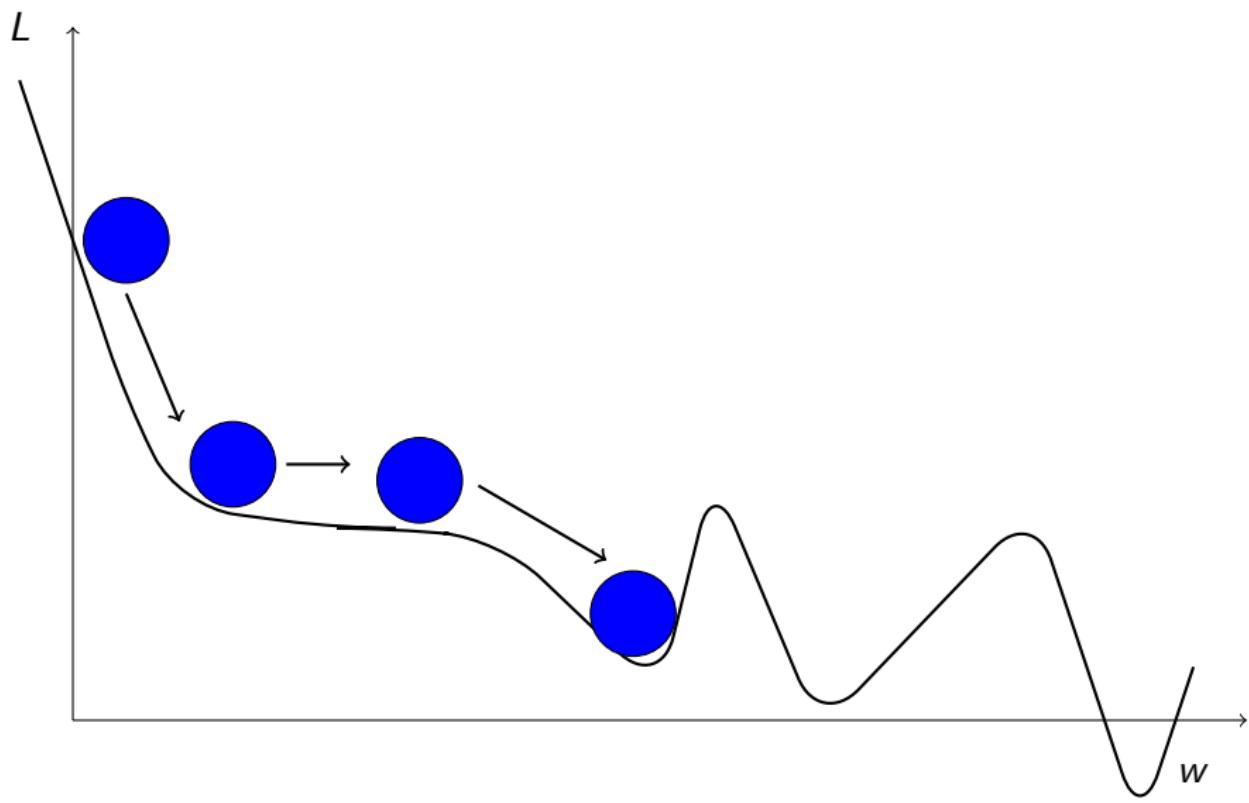
Descente du gradient

Méthode du Momentum :



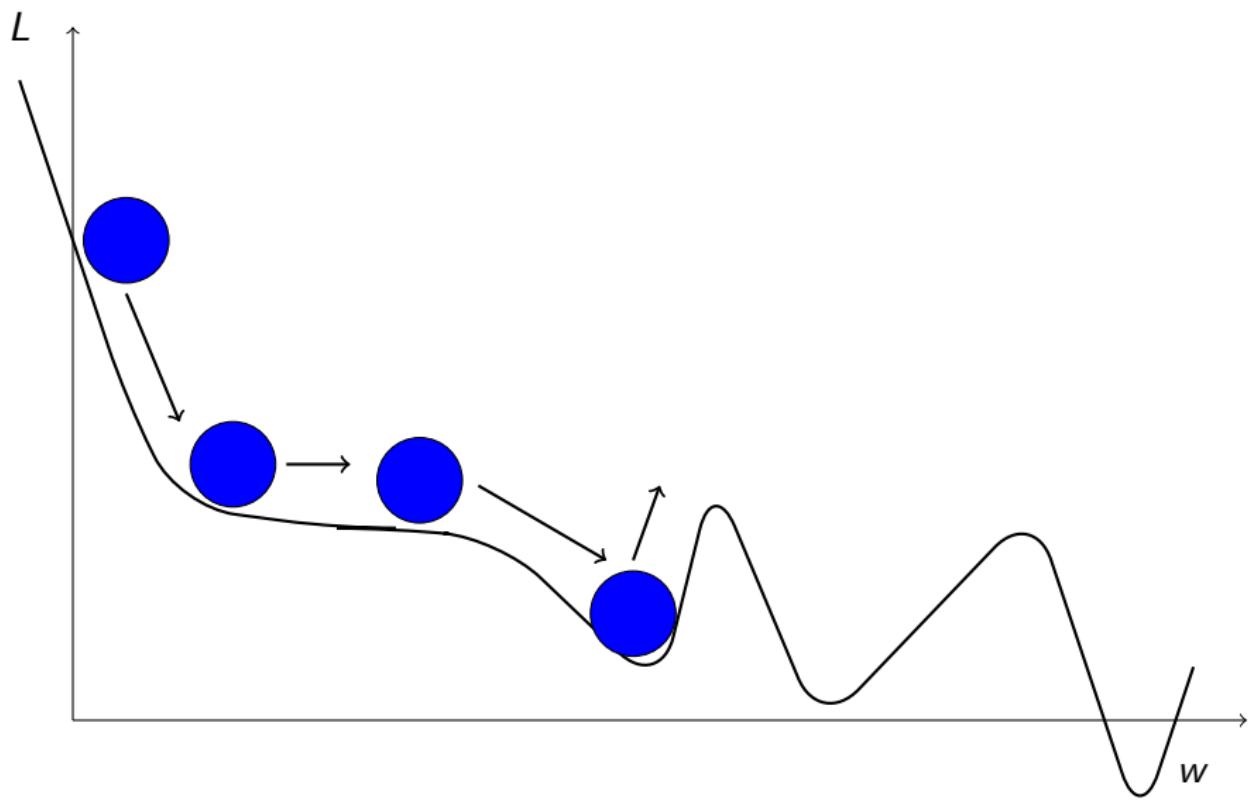
Descente du gradient

Méthode du Momentum :



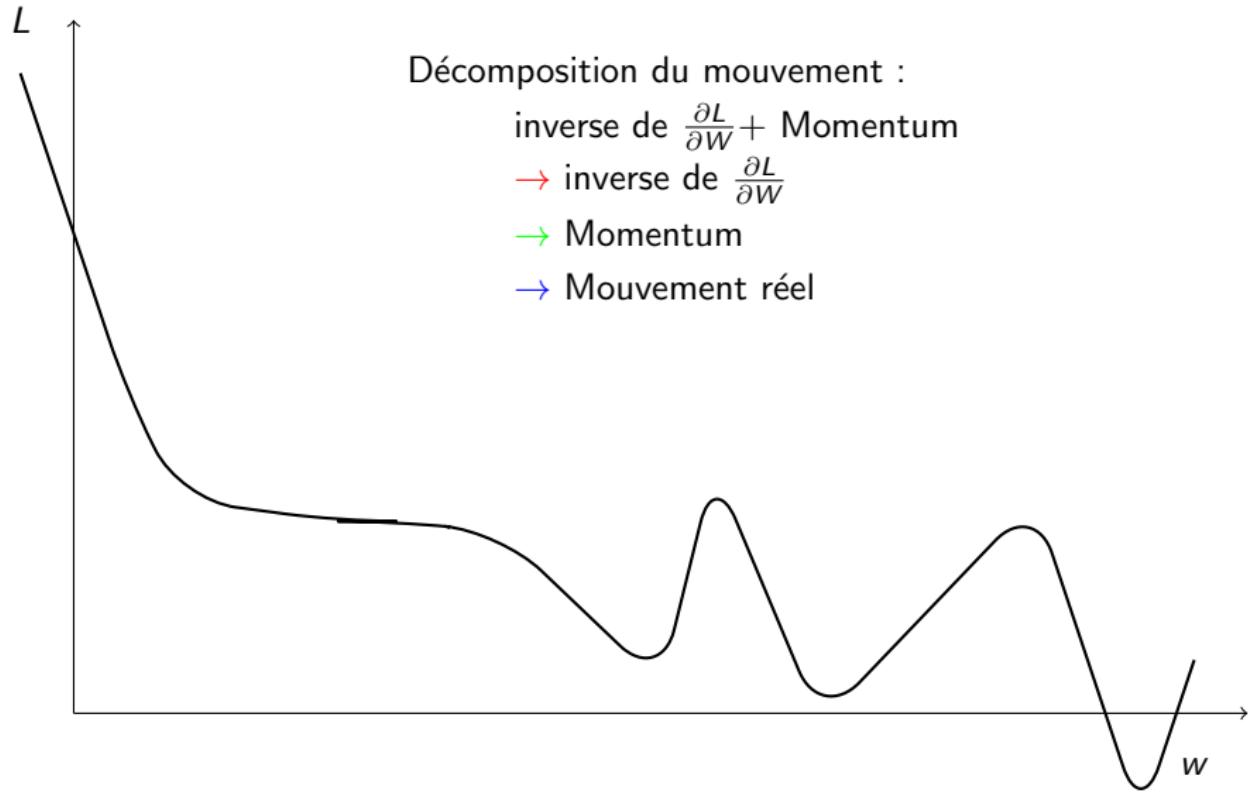
Descente du gradient

Méthode du Momentum :



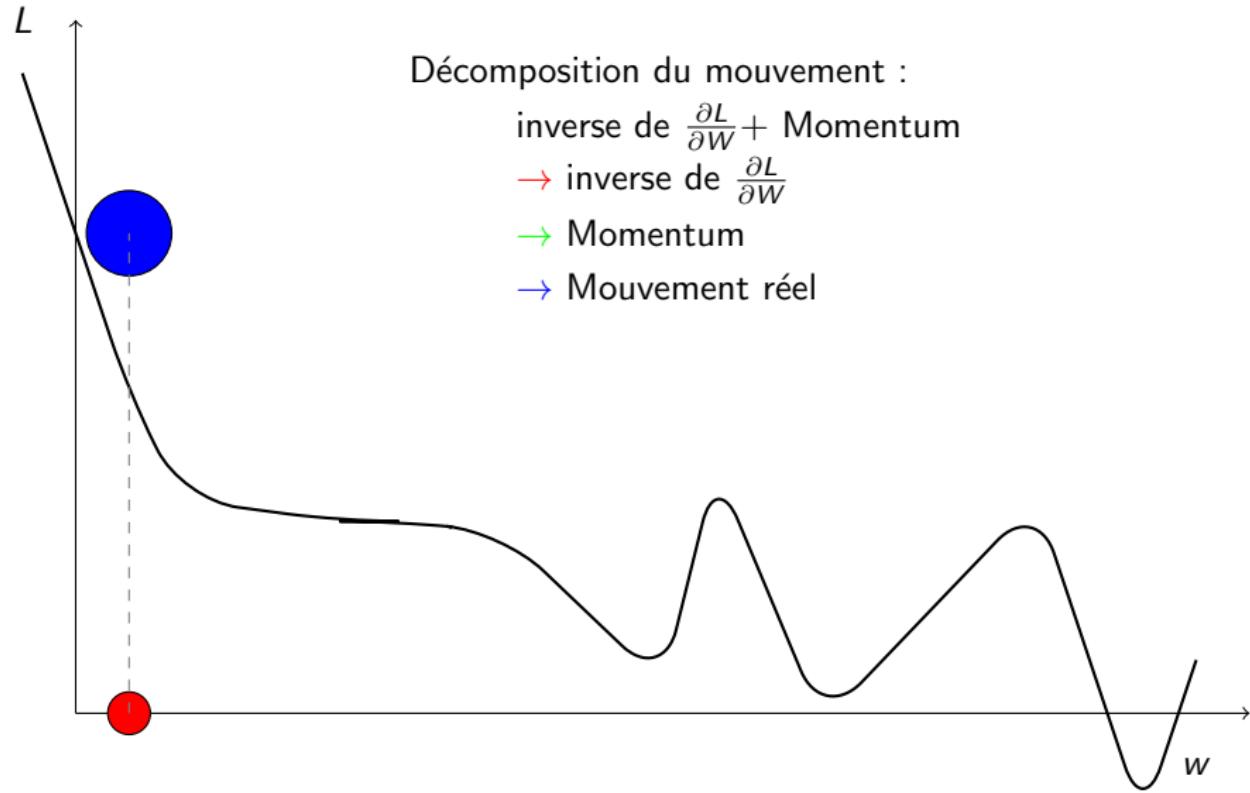
Descente du gradient

Problème : minimum mais local



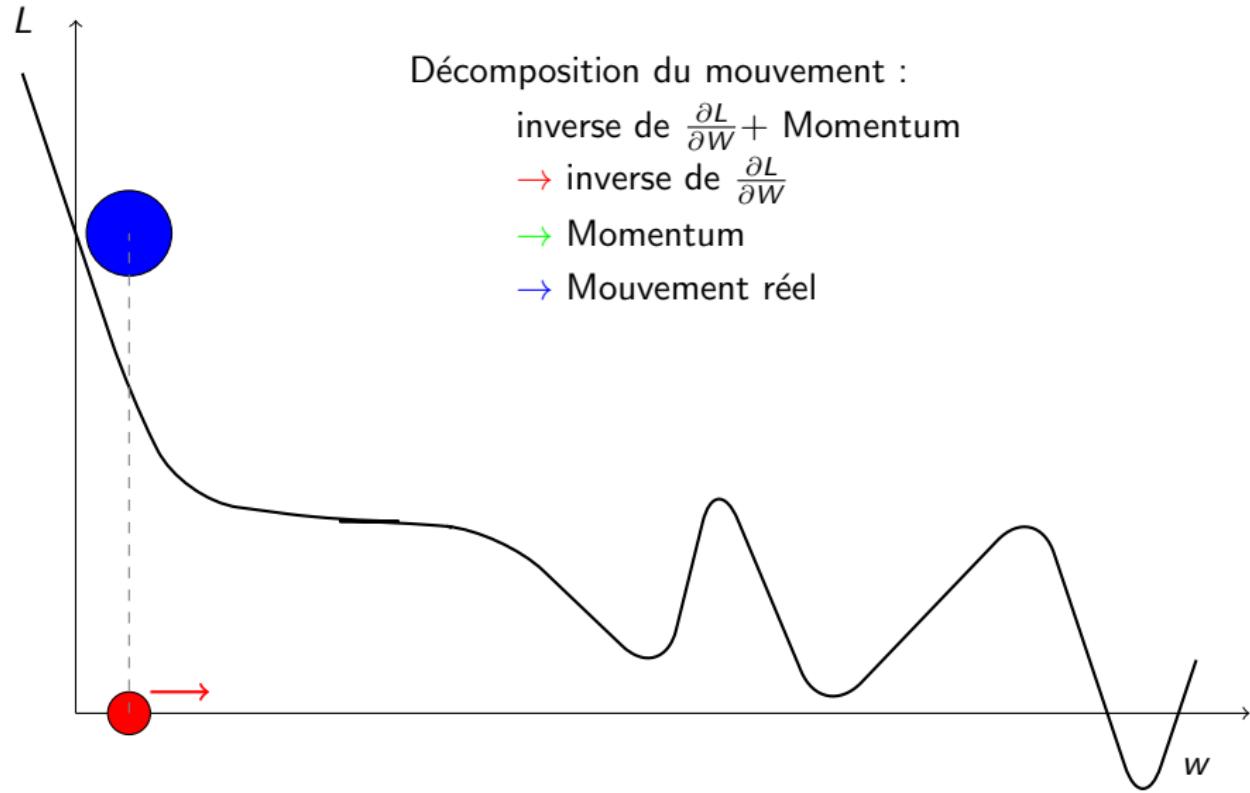
Descente du gradient

Problème : minimum mais local



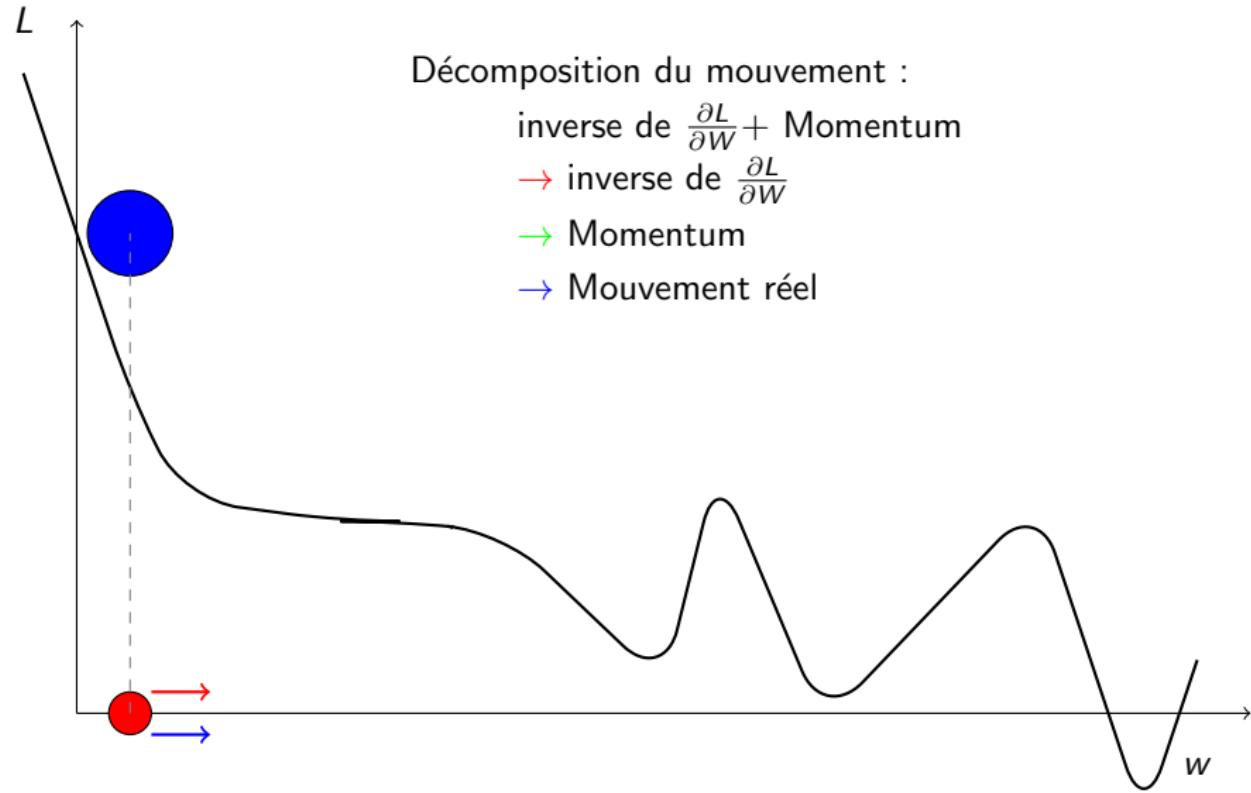
Descente du gradient

Problème : minimum mais local



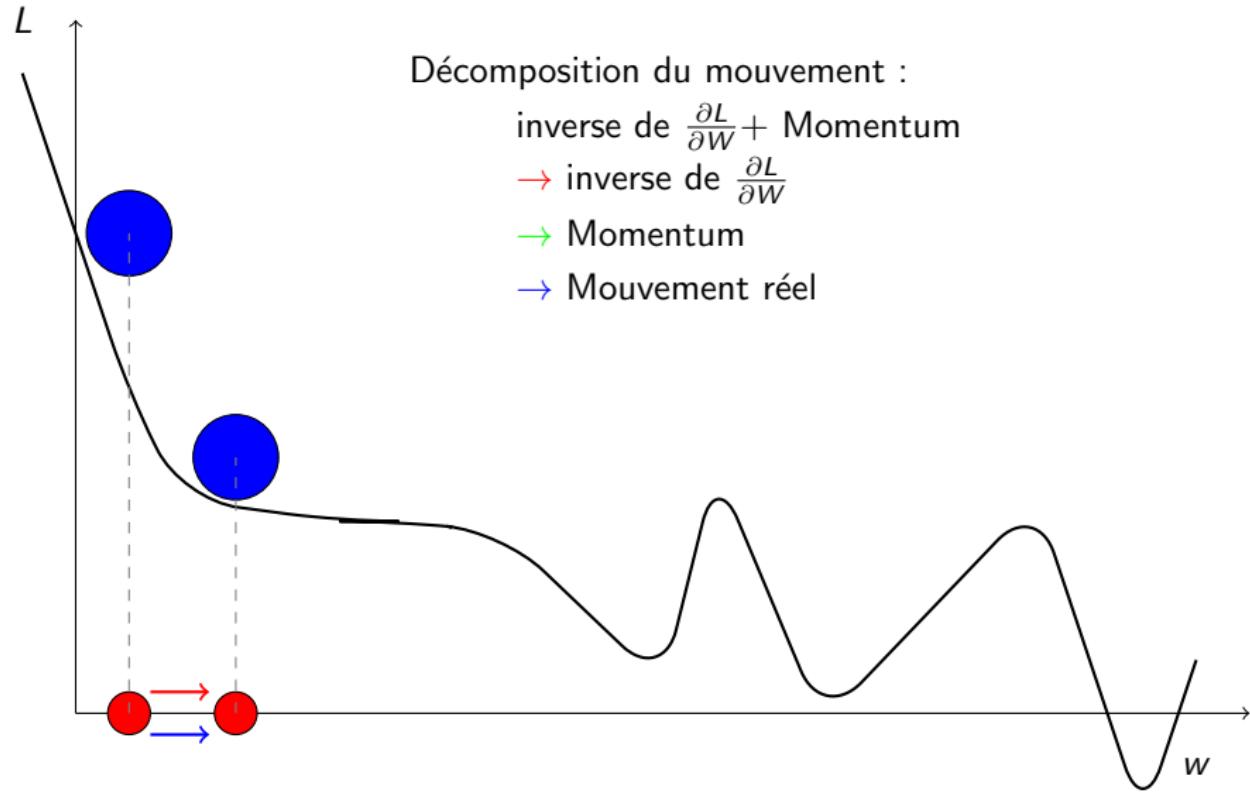
Descente du gradient

Problème : minimum mais local



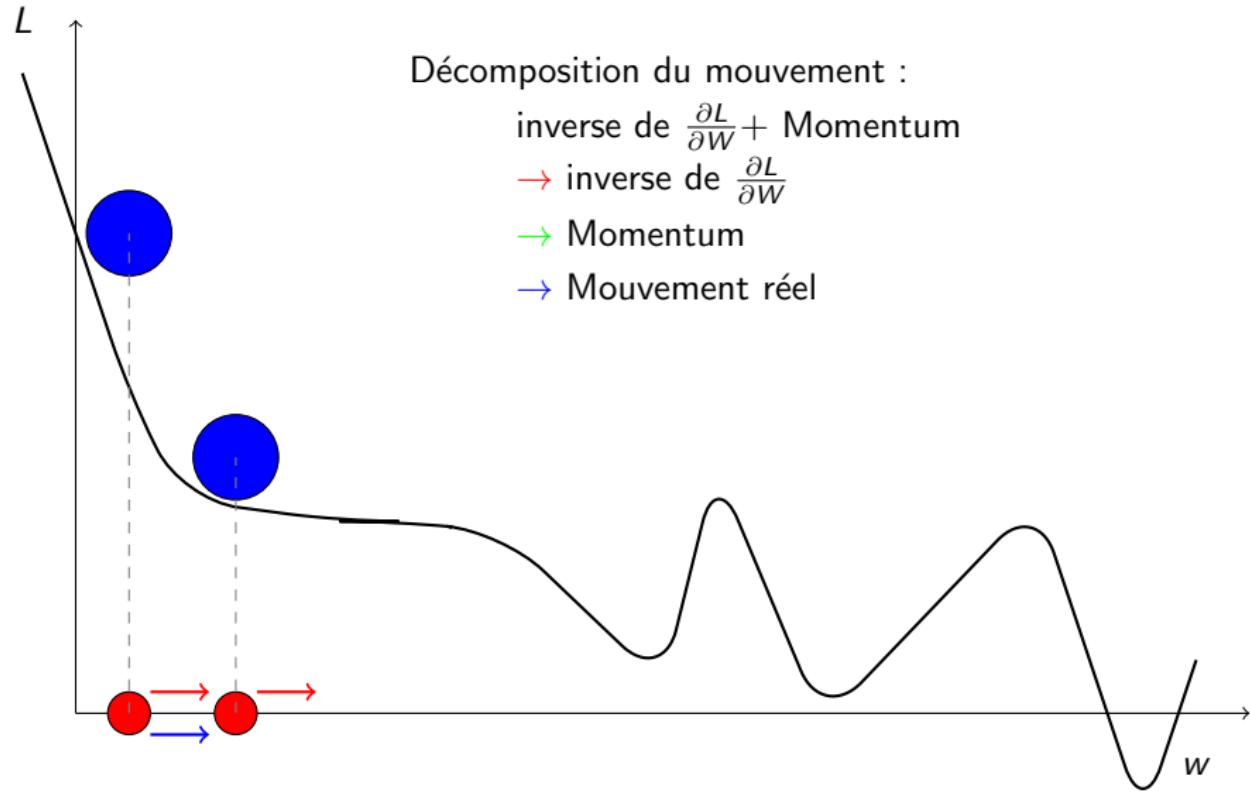
Descente du gradient

Problème : minimum mais local



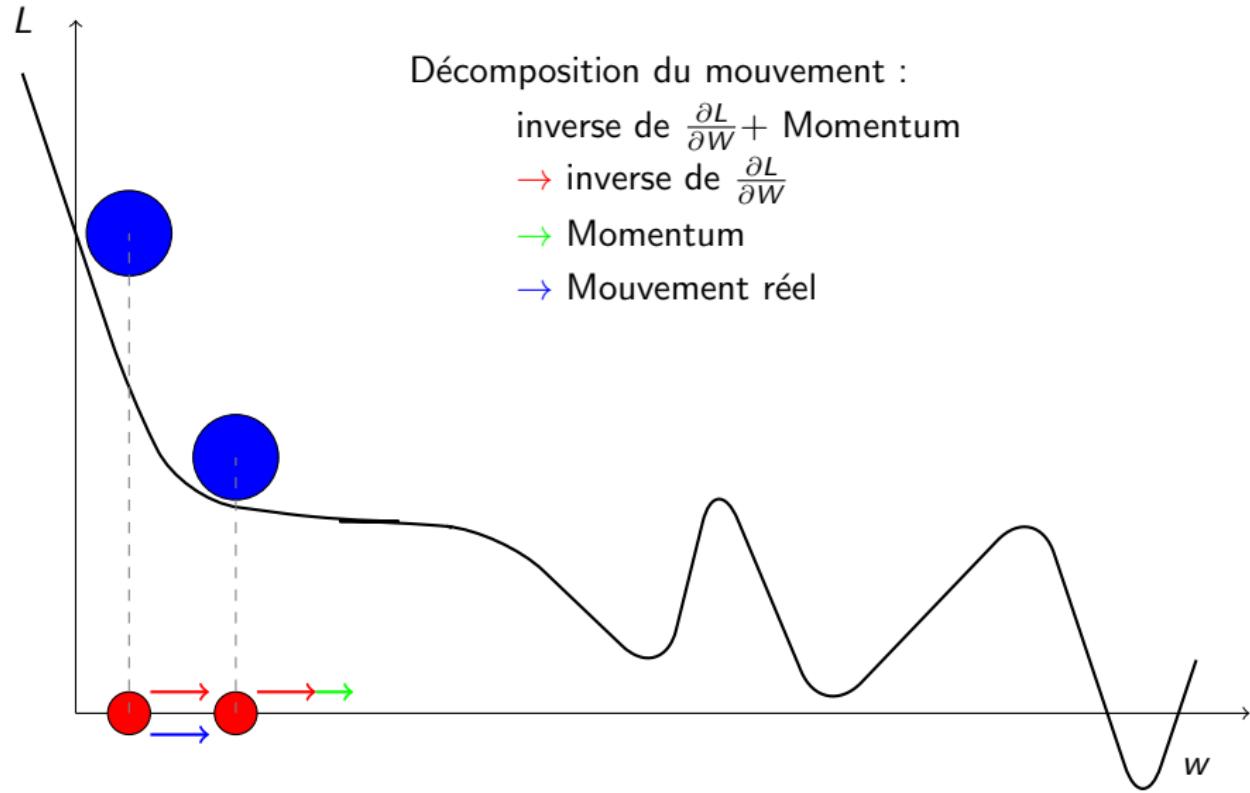
Descente du gradient

Problème : minimum mais local



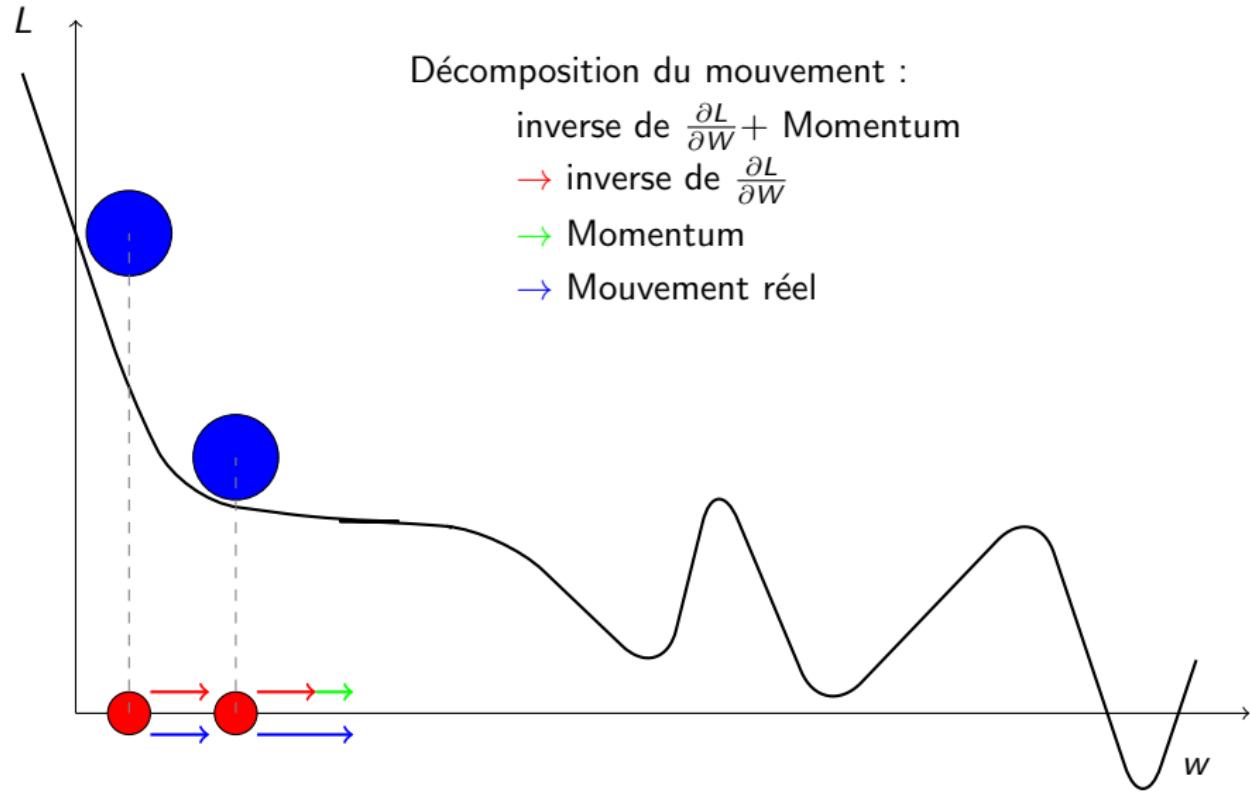
Descente du gradient

Problème : minimum mais local



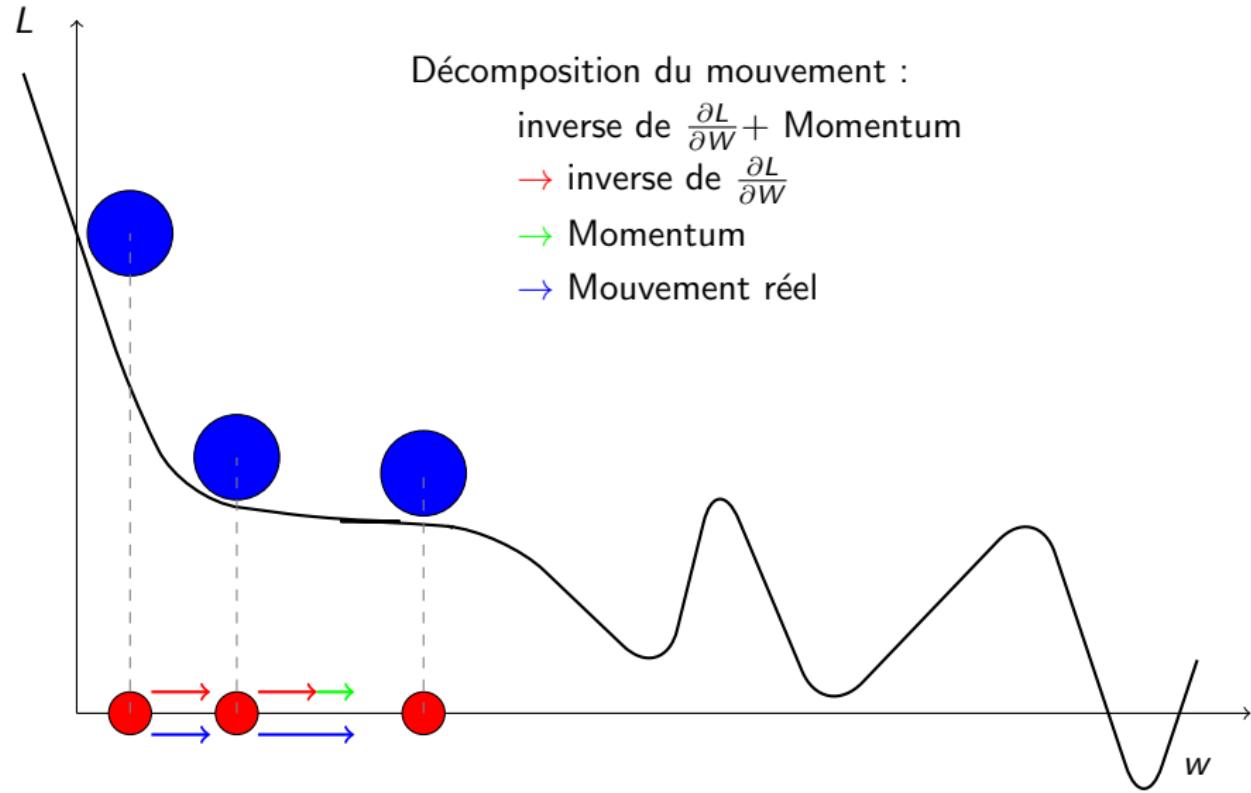
Descente du gradient

Problème : minimum mais local



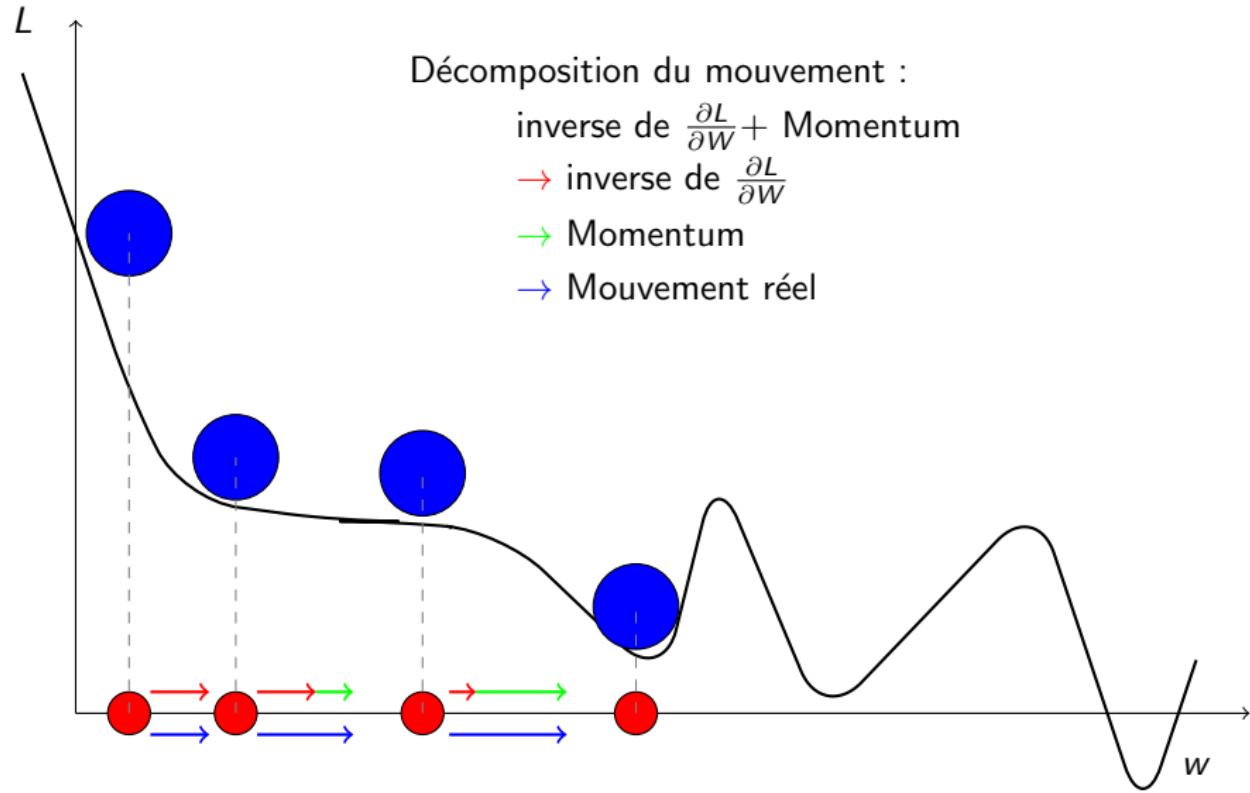
Descente du gradient

Problème : minimum mais local



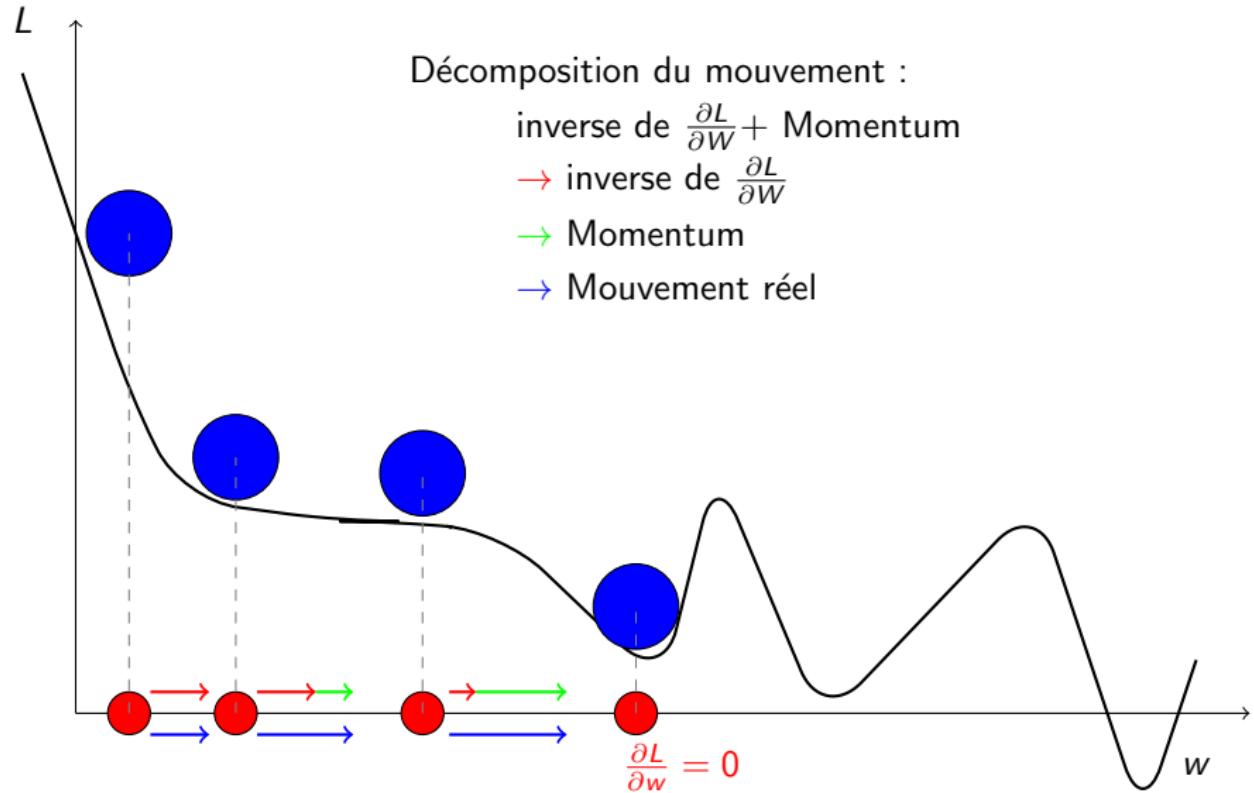
Descente du gradient

Problème : minimum mais local



Descente du gradient

Problème : minimum mais local



Décomposition du mouvement :

inverse de $\frac{\partial L}{\partial w}$ + Momentum

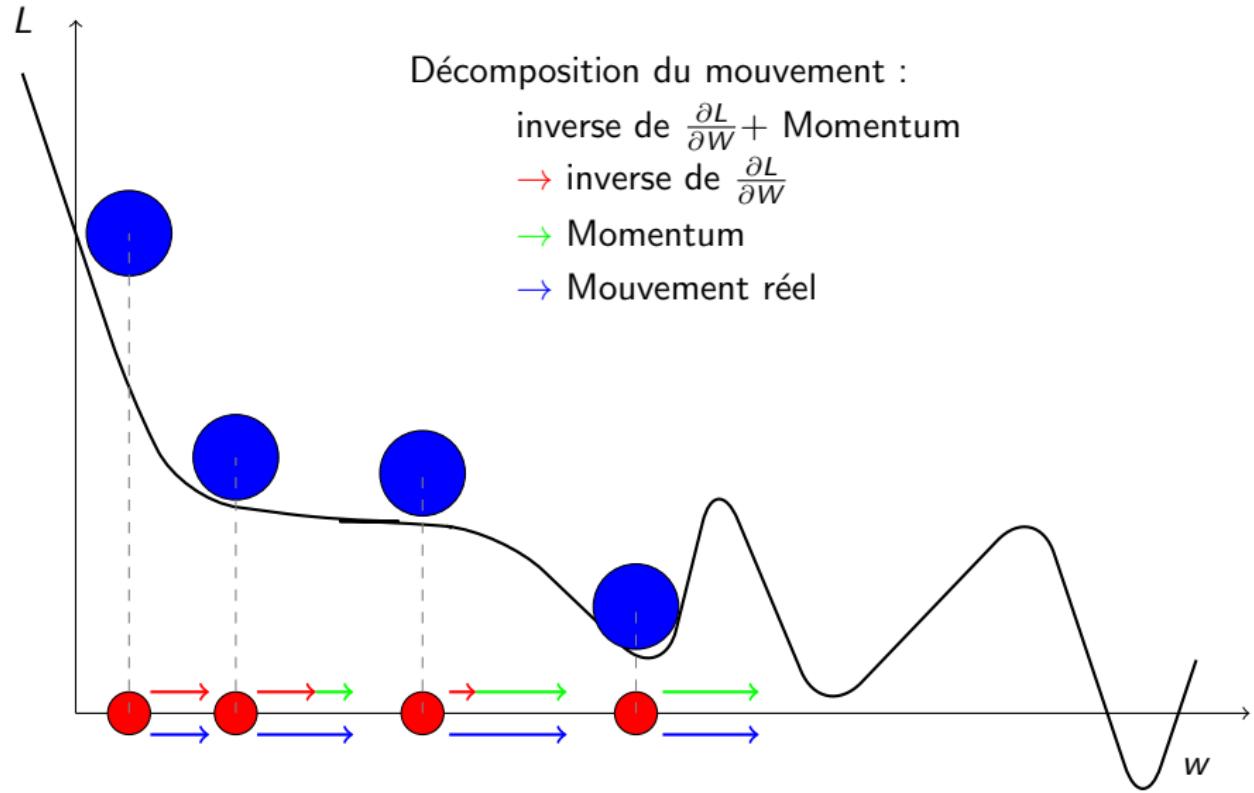
→ inverse de $\frac{\partial L}{\partial w}$

→ Momentum

→ Mouvement réel

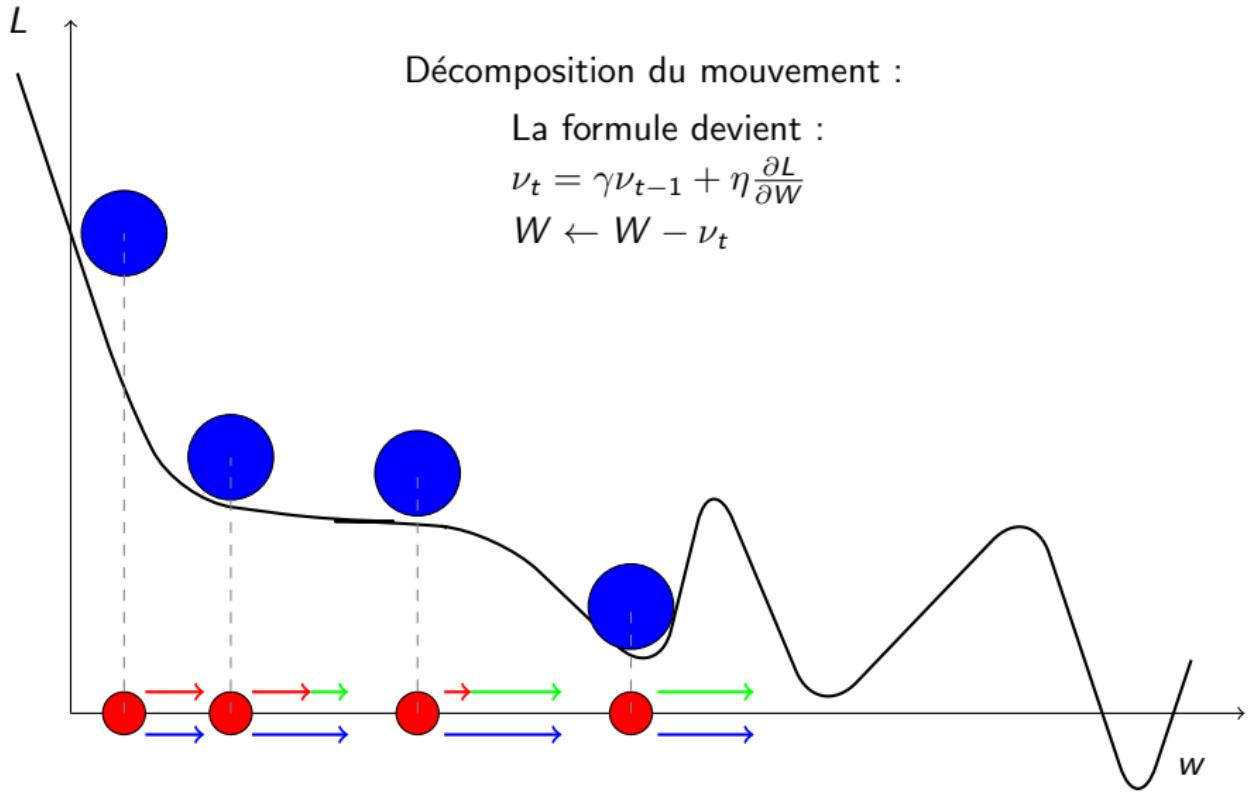
Descente du gradient

Problème : minimum mais local



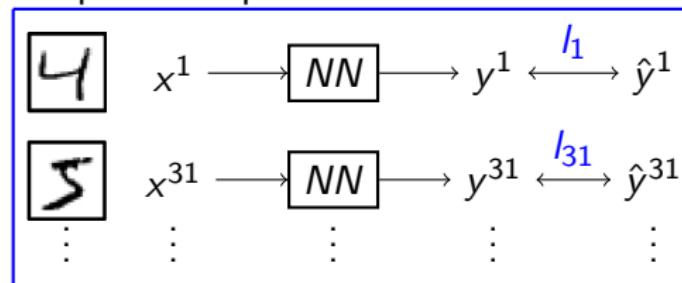
Descente du gradient

Problème : minimum mais local



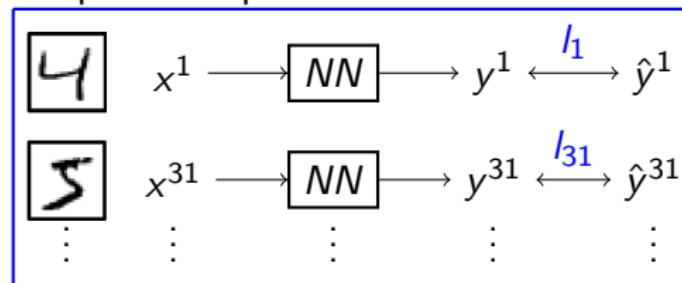
A la recherche de la meilleure fonction

On prend un premier mini-batch :



A la recherche de la meilleure fonction

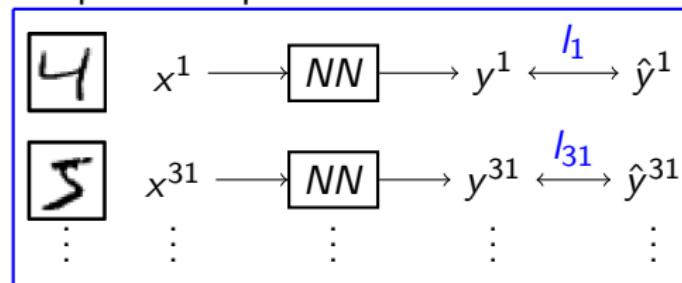
On prend un premier mini-batch :



$$L' = l^1 + l^{31} + \dots$$

A la recherche de la meilleure fonction

On prend un premier mini-batch :

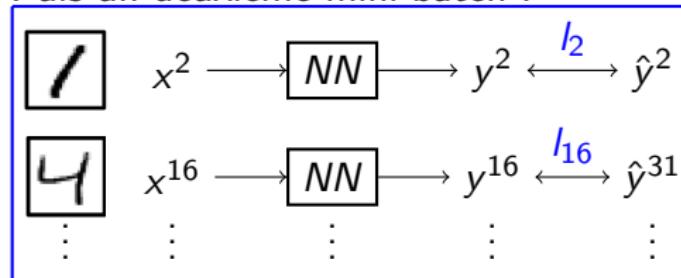


$$L' = l^1 + l^{31} + \dots$$

Mettre à jour les paramètres

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



Et ainsi de suite

...

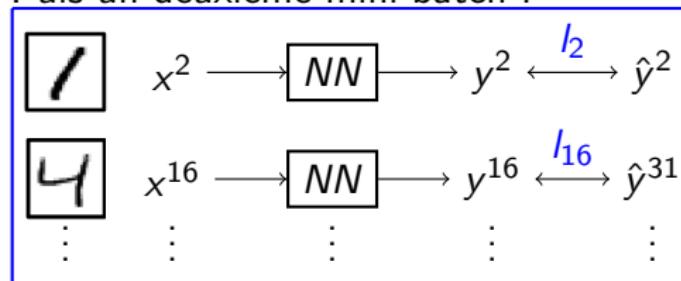
Jusqu'à ce que tous les mini-batches soient lus.

⇒ Celle correspond à un(e) **epoch**.

Et l'on répète le processus un nombre d'epochs ...

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



$$L'' = l^2 + l^{16} + \dots$$

Et ainsi de suite

...

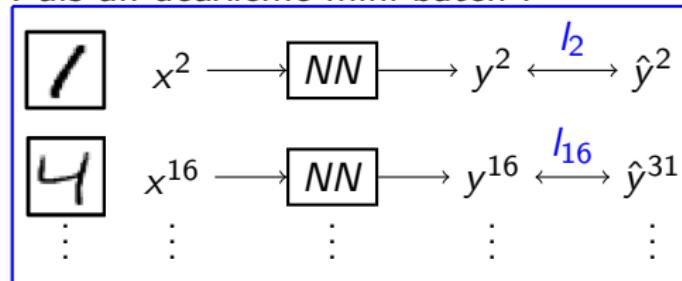
Jusqu'à ce que tous les mini-batches soient lus.

⇒ Celle correspond à un(e) **epoch**.

Et l'on répète le processus un nombre d'epochs ...

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :

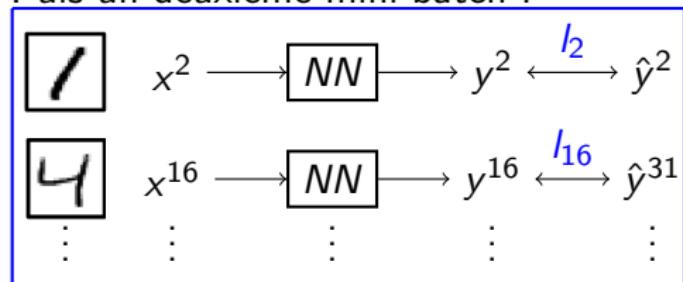


$$L'' = l^2 + l^{16} + \dots$$

Mettre à jour les paramètres

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



$$L'' = l^2 + l^{16} + \dots$$

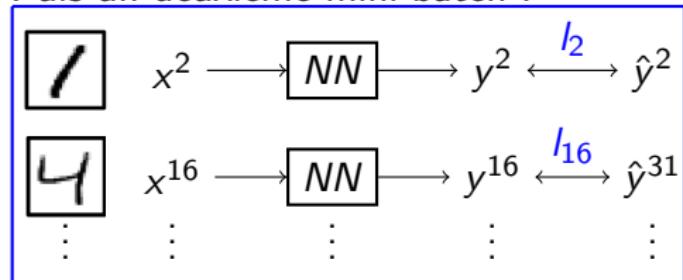
Mettre à jour les paramètres

Et ainsi de suite

...

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



$$L'' = l^2 + l^{16} + \dots$$

Mettre à jour les paramètres

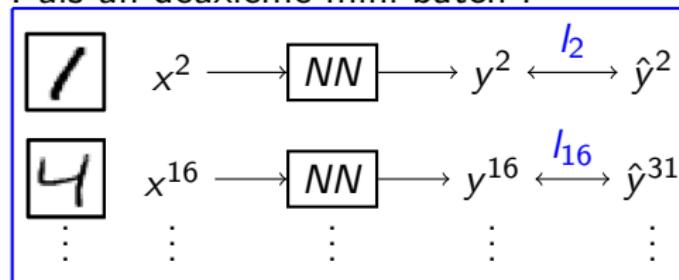
Et ainsi de suite

...

Jusqu'à ce que tous les mini-batchs soient lus.

A la recherche de la meilleure fonction

Puis un deuxième mini-batch :



$$L'' = l^2 + l^{16} + \dots$$

Mettre à jour les paramètres

Et ainsi de suite

...

Jusqu'à ce que tous les mini-batchs soient lus.

⇒ Cela correspond à un(e) **epoch**.

Et l'on répète le processus un nombre d'epochs ...

A la recherche de la meilleure fonction

Dans la majorité des outils de deep learning, ces deux paramètres sont à indiquer :

- ▶ `batch_size` correspond au nombre d'exemples utilisé pour estimer le gradient de la fonction de coût.
- ▶ `epochs` est le nombre d'époques (i.e. passages sur l'ensemble des exemples de la base d'apprentissage) lors de la descente de gradient.

Problème de la disparition du gradient

Intuition :

La mise à jour des paramètres se fait si la dérivée n'est pas nulle.

Problème : avec la sigmoid (et d'autres fonctions) comme fonction d'activation, au bout d'un certain nombre d'itérations, $\frac{\partial L}{\partial W}$ devient nulle ...

Solution :

Utiliser, comme fonction d'activation, la fonction ReLU (Rectified Linear Unit) :

