

Introduction aux réseaux de neurones : La descente de gradient

Matériel de cours rédigé par Pascal Germain, 2019

Out[1]: [voir/cacher le code.](#)

Régression linéaire

Considérons un ensemble d'apprentissage $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$,

avec $\mathbf{x} \in \mathbb{R}^d$ et $y \in \mathbb{R}$.

Considérons un prédicteur $f_{\mathbf{w}}$ linéaire et sans biais,

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x},$$

La minimisation de la *fonction de perte quadratique* revient à résoudre problème d'optimisation des *moindres carrés*:

$$\min_{\mathbf{w}} \left[\frac{1}{n} \sum_{i=1}^n (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2 \right].$$

Nous désirons trouver le minimum de la **fonction objectif** $F(\mathbf{w})$ suivante:

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2.$$

Calculons la **dérivée partielle** de $F(\mathbf{w})$ selon un élément w_k du vecteur \mathbf{w}

$$\begin{aligned} \frac{\partial F(\mathbf{w})}{\partial w_k} &= \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial w_k} (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n 2 (\mathbf{w} \cdot \mathbf{x}_i - y_i) \frac{\partial}{\partial w_k} (\mathbf{w} \cdot \mathbf{x}_i - y_i) \\ &= \frac{1}{n} \sum_{i=1}^n 2 (\mathbf{w} \cdot \mathbf{x}_i - y_i) \frac{\partial}{\partial w_k} (w_k x_{i,k}) \\ &= \frac{1}{n} \sum_{i=1}^n 2 (\mathbf{w} \cdot \mathbf{x}_i - y_i) x_{i,k}. \end{aligned}$$

Le **gradient** de $F(\mathbf{w})$, noté $\nabla F(\mathbf{w})$, est le vecteur de toutes les dérivées partielles au point \mathbf{w} :

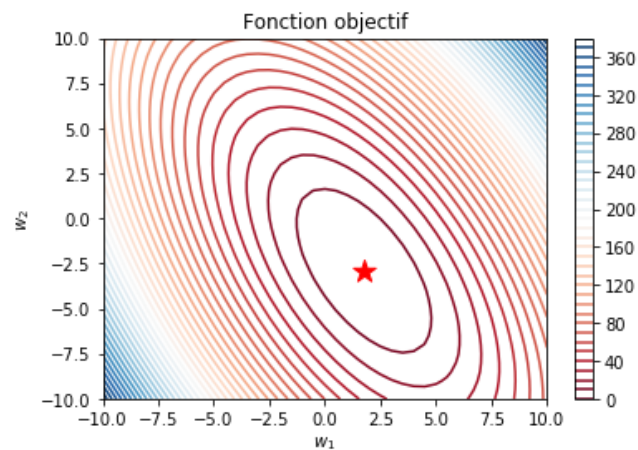
$$\nabla F(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1} \\ \vdots \\ \frac{\partial}{\partial w_d} \end{bmatrix} = \frac{2}{n} \sum_{i=1}^n (\mathbf{w} \cdot \mathbf{x}_i - y_i) \mathbf{x}_i.$$

Le minimum d'une fonction convexe est atteint lorsque $\nabla F(\mathbf{w}) = 0$.

Illustrons la fonction objectif pour un ensemble de données tel que:

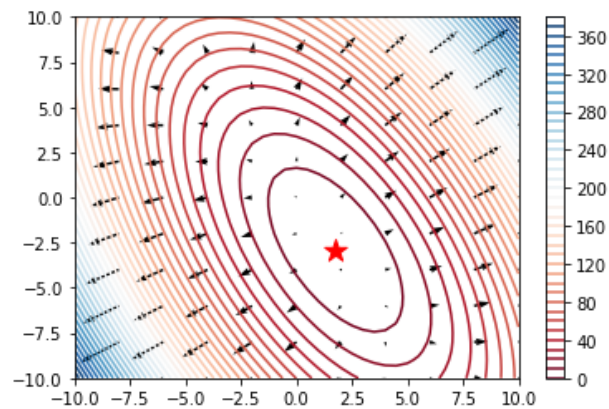
$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 0 & -1 \\ 2 & \frac{1}{2} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} -1 \\ 3 \\ 2 \end{bmatrix}$$

Out[2]: [voir/cacher le code.](#)



Illustrons maintenant les gradients de la fonction objectif.

Out[3]: [voir/cacher le code.](#)



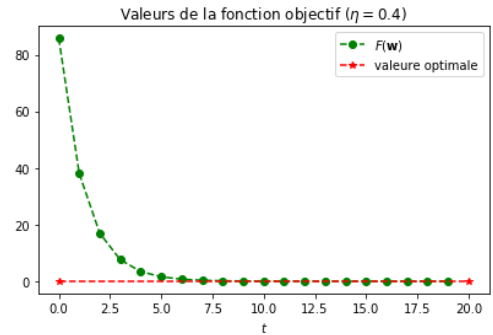
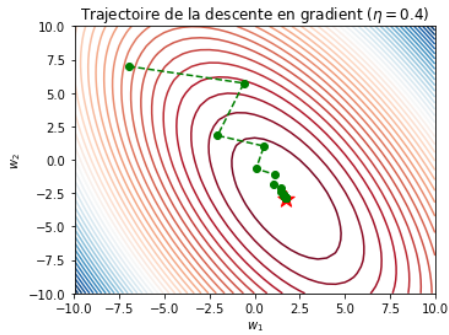
Descente en gradient

Algorithme (pour un *pas de gradient* η et un nombre d'itérations T)

- Initialiser $\mathbf{z}_0 \in \mathbb{R}^d$ aléatoirement
- Pour t de 1 à T :
 - $\mathbf{g}_t = \nabla F(\mathbf{z}_{t-1})$
 - $\mathbf{z}_t = \mathbf{z}_{t-1} - \eta \mathbf{g}_t$
- Retourner \mathbf{z}_T

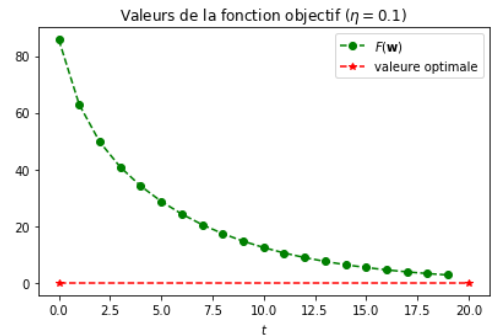
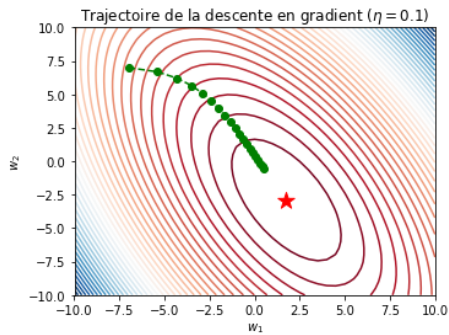
Exemple avec $\eta = 0.4$ et $T = 20$

Out[4]: [voir/cacher le code.](#)



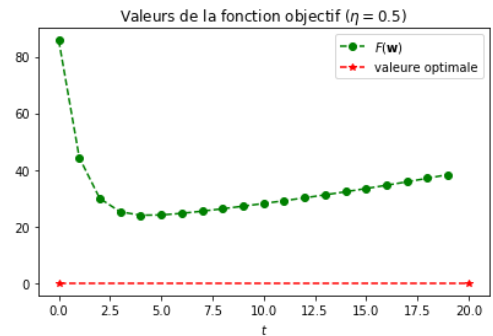
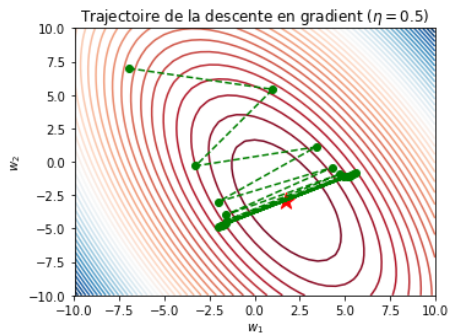
Exemple avec $\eta = 0.1$ et $T = 20$

Out[5]: [voir/cacher le code.](#)



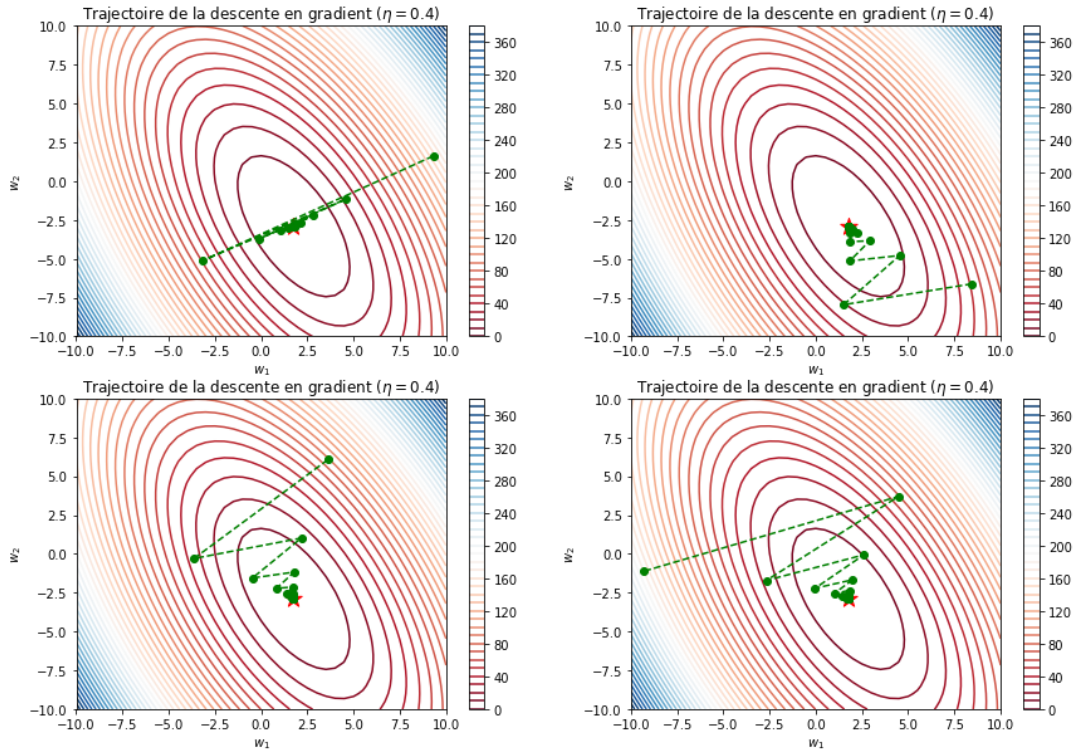
Exemple avec $\eta = 0.5$ et $T = 20$

Out[6]: [voir/cacher le code.](#)

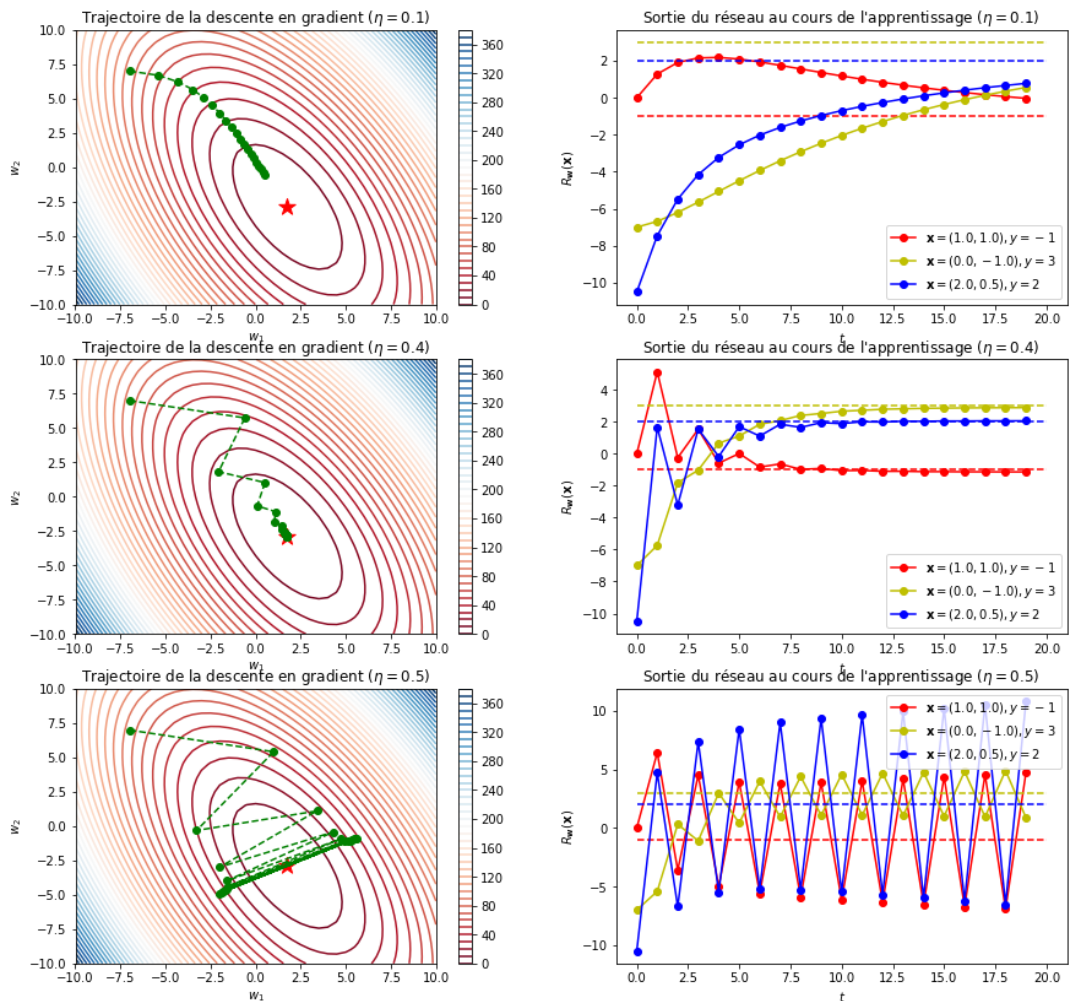


Exemple avec $\eta = 0.4$ et $T = 20$, différentes initialisation aléatoires

Out[7]: [voir/cacher le code.](#)



Out[8]: [voir/cacher le code.](#)



Descente en gradient stochastique

Nous désirons trouver le minimum de la **fonction objectif** $F(\mathbf{w})$ suivante:

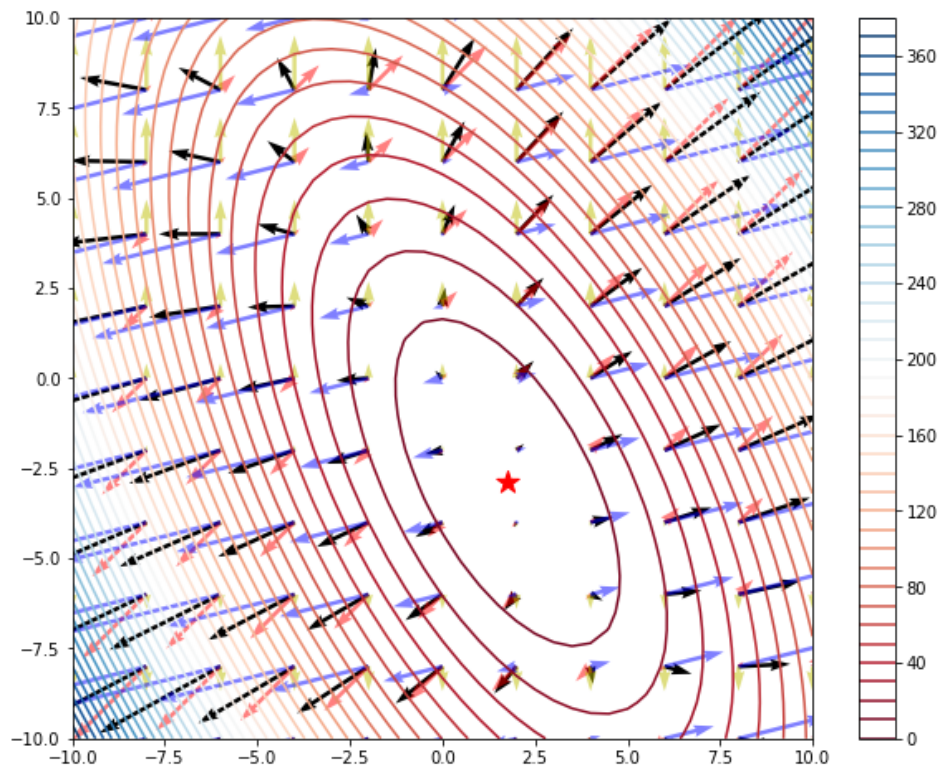
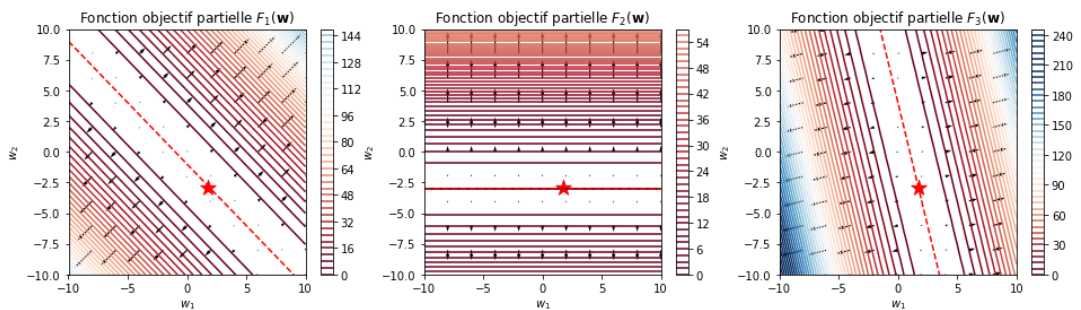
$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2 = \frac{1}{n} \sum_{i=1}^n F_i(\mathbf{w}),$$

avec $F_i(\mathbf{w}) = (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2$ et donc $\nabla F_i(\mathbf{w}) = 2(\mathbf{w} \cdot \mathbf{x}_i - y_i) \mathbf{x}_i$.

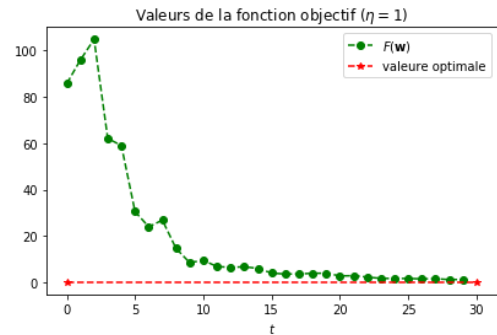
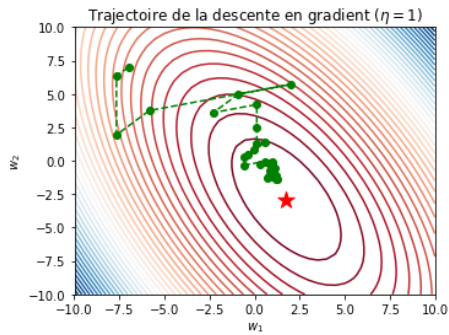
Algorithme (pour un pas de gradient η et un nombre d'itérations T)

- Initialiser $\mathbf{z}_0 \in \mathbb{R}^d$ aléatoirement
- Pour t de 1 à T :
 - Choisir aléatoirement $i \in \{1, \dots, n\}$
 - $\mathbf{g}_t = \nabla F_i(\mathbf{z}_{t-1})$
 - $\mathbf{z}_t = \mathbf{z}_{t-1} - \frac{\eta}{\sqrt{t}} \mathbf{g}_t$
- Retourner \mathbf{z}_T

Out[9]: [voir/cacher le code.](#)



Out[11]: [voir/cacher le code.](#)

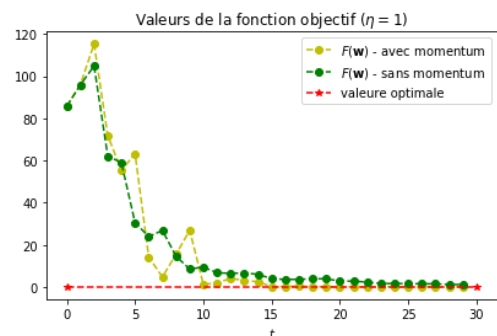
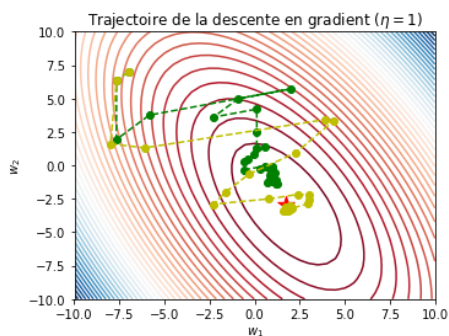


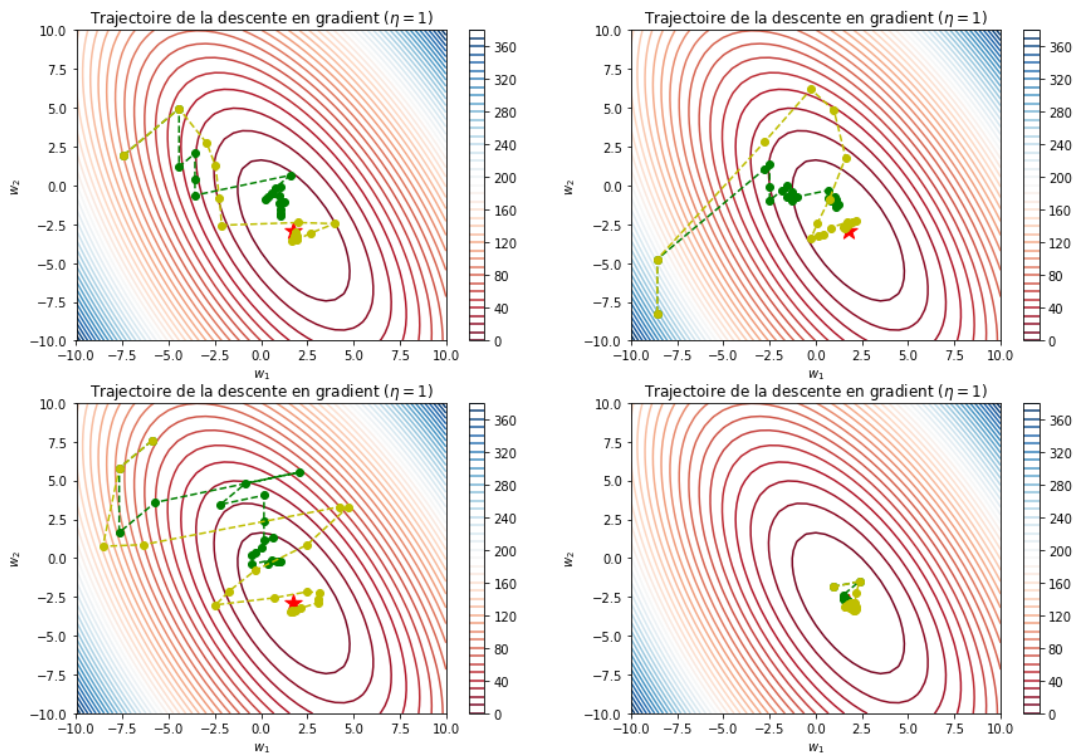
Descente en gradient stochastique avec momentum

Algorithme (pour un *pas de gradient* η , une *vélocité* α et un nombre d'itérations T)

- Initialiser $\mathbf{z}_0 \in \mathbb{R}^d$ aléatoirement
- $\mathbf{v}_0 = 0$
- Pour t de 1 à T :
 - Choisir aléatoirement $i \in \{1, \dots, n\}$
 - $\mathbf{g}_t = \nabla F_i(\mathbf{z}_{t-1})$
 - $\mathbf{v}_t = \alpha \mathbf{v}_{t-1} - \frac{\eta}{\sqrt{t}} \mathbf{g}_t$
 - $\mathbf{z}_t = \mathbf{z}_{t-1} + \mathbf{v}_t$
- Retourner \mathbf{z}_T

Out[12]: [voir/cacher le code.](#)



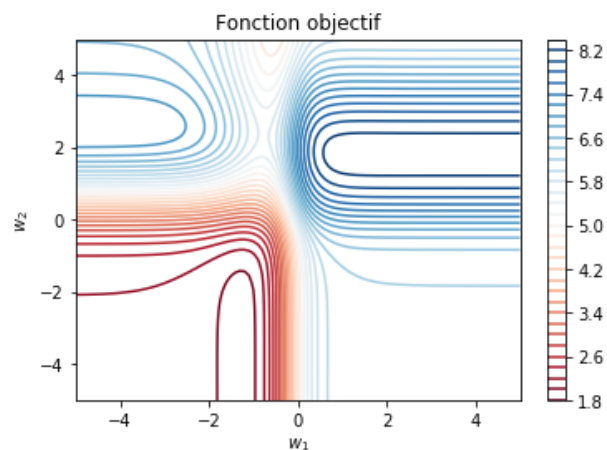


Les réseaux de neurones génèrent des fonctions d'optimisation non-convexes

$$R_{\mathbf{w}}(\mathbf{x}) = \mathbf{v} \cdot \tanh\left(\begin{bmatrix} w_1 & -1 \\ 1 & w_2 \end{bmatrix} \mathbf{x}\right), \quad \text{avec } \mathbf{v} = [-1, 1].$$

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (R_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2.$$

Out[101]: [voir/cacher le code.](#)



Implémentation dans pytorch : Descente en gradient stochastique avec momentum

`torch.optim.SGD?`

Implements stochastic gradient descent (optionally with momentum).

Nesterov momentum is based on the formula from
`On the importance of initialization and momentum in deep learning`__.

Args:

ng
 params (iterable): iterable of parameters to optimize or dicts defini
 parameter groups
 lr (float): learning rate
 momentum (float, optional): momentum factor (default: 0)
0) weight_decay (float, optional): weight decay (L2 penalty) (default:
 dampening (float, optional): dampening for momentum (default: 0)
 nesterov (bool, optional): enables Nesterov momentum (default: False)

Example:

0.9) >>> optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=
>>> optimizer.zero_grad()
>>> loss_fn(model(input), target).backward()
>>> optimizer.step()

__ <http://www.cs.toronto.edu/%7Ehinton/absps/momentum.pdf>

.. note::

The implementation of SGD with Momentum/Nesterov subtly differs from Sutskever et. al. and implementations in some other frameworks.

as Considering the specific case of Momentum, the update can be written

.. math::

$$\begin{aligned} v &= \rho * v + g \\ p &= p - lr * v \end{aligned}$$

where p, g, v and ρ denote the parameters, gradient, velocity, and momentum respectively.

This is in contrast to Sutskever et. al. and other frameworks which employ an update of the form

.. math::

$$\begin{aligned} v &= \rho * v + lr * g \\ p &= p - v \end{aligned}$$

The Nesterov version is analogously modified.

Autre algorithme de descente de gradient

Implémentation dans pytorch : RMSprop

torch.optim.RMSprop?

Implements RMSprop algorithm.

Proposed by G. Hinton in his
`course <http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf>`.

The centered version first appears in `Generating Sequences
With Recurrent Neural Networks <<https://arxiv.org/pdf/1308.0850v5.pdf>>`.

Arguments:

params (iterable): iterable of parameters to optimize or dicts defining
parameter groups
lr (float, optional): learning rate (default: 1e-2)
momentum (float, optional): momentum factor (default: 0)
alpha (float, optional): smoothing constant (default: 0.99)
eps (float, optional): term added to the denominator to improve
numerical stability (default: 1e-8)
centered (bool, optional) : if ``True``, compute the centered RMSProp,
the gradient is normalized by an estimation of its variance
weight_decay (float, optional): weight decay (L2 penalty) (default: 0)

Autre algorithme de descente de gradient

Implémentation dans pytorch : ADAM

`torch.optim.Adam?`

Implements Adam algorithm.

It has been proposed in `Adam: A Method for Stochastic Optimization`.

Arguments:

params (iterable): iterable of parameters to optimize or dicts defining
parameter groups
lr (float, optional): learning rate (default: 1e-3)
betas (Tuple[float, float], optional): coefficients used for computing
running averages of gradient and its square (default: (0.9, 0.99))
eps (float, optional): term added to the denominator to improve
numerical stability (default: 1e-8)
weight_decay (float, optional): weight decay (L2 penalty) (default: 0)
amsgrad (boolean, optional): whether to use the AMSGrad variant of this
algorithm from the paper `On the Convergence of Adam and Beyond`
(default: False)

.. `_Adam`: A Method for Stochastic Optimization:
<https://arxiv.org/abs/1412.6980>
.. `_On the Convergence of Adam and Beyond`:
<https://openreview.net/forum?id=ryQu7f-RZ>