Atelier de Statistique Mathématique

Descente de Gradient Stochastique

N.RAKOTO, Q.RUEL

Master 1 ESA

Janvier 2025

Table des matières

Section 1

Présentation générale de la méthode du SGD

1. Présentation générale de la méthode du SGD

Idée générale de la descente du gradient

- Imaginez une vallée montagneuse. Vous êtes au sommet d'une montagne et vous voulez descendre au point le plus bas de la vallée en faisant le moins d'efforts possible. À chaque étape, vous choisissez la direction qui descend le plus rapidement - c'est l'essence de la descente du gradient,
- Nous sommes des statisticiens, nous allons le faire dans un cadre beaucoup plus théorique, mais l'idée ne change pas

1. Présentation générale de la méthode du SGD

Objectif

Nous souhaitons modéliser au mieux nos données $(y_i, \mathbf{x_i})_{i=1}^n$, *avec* $x_i \in \mathbb{R}^D$, *nos données d'entrées, et* $y_i \in \mathbb{R}$, *la sortie attendue.*

- En Machine Learning, les données d'entrées $(\mathbf{x_i})_{i=1}^n$ représentent les D variables explicatives (ou exogènes) en économétrie,
- La sortie attendue $(y_i)_{i=1}^n$ en ML est la variable expliquée (ou endogène) en économétrie.

1. Présentation générale de la méthode du SGD

Comment allons-t'on parvenir ?

- Nous allons construire une fonction $F : \mathbb{R}^D \to \mathbb{R}$, telle que, lorsque F est évaluée en x_i, on aimerait que ça soit exactement y_i, i.e $F(x_i) \approx y_i$.
- $F(\mathbf{x_i})$ est la sortie produite par le modèle, en économétrie des modèles linéaires, on écrit $y_i = F(\mathbf{x_i}) + \epsilon_i$ où $F(\mathbf{x_i}) = \beta_0 + \sum_{k=1}^{n} \beta_k X_{ik}$
- F dépend donc de plusieurs paramètres $(\beta_0, \ldots, \beta_n)$
- Pour trouver la valeur de ces paramètres, on utilise une méthode d'estimation, par exemple par la méthode des Moindres Carrés Ordinaires (MCO) ou par le maximum de vraisemblance.

1. Présentation générale de la méthode du SGD

En quoi consiste la méthode de descente de gradient stochastique ?

- Pour comprendre l'utilité et l'intuition derrière la SGD (Stochastic Gradient Descent), nous allons d'abord expliquer comment fonctionne la descente de gradient classique.

Objectif

Nous allons minimiser une ***fonction de perte*** *E, appellée aussi erreur totale. Cette fonction de perte mesure à quel point notre modèle est éloigné des données réelles.*

1. Présentation générale de la méthode du SGD

Exemples :

- Cette fonction de perte peut être le gradient de la log-vraisemblance d'un modèle si on souhaite utiliser la méthode du maximum de vraisemblance
- En économétrie, en passant par la MCO, cette fonction de perte serait $\sum_{i=1}^{n} \epsilon_i^2$.
- En d'autres termes,

$$E = \sum_{i=1}^{n} E_i(\beta_0, \ldots, \beta_D) = \sum_{i=1}^{n} \left(y_i - F(x_i)\right)^2$$

1. Présentation générale de la méthode du SGD

Comment trouver les valeurs de ces paramètres ?

- On part d'un vecteur $P_0 = (\beta_1, \ldots, \beta_n)$, tel que P_0 est notre condition initiale pour débuter l'algorithme.
- P_k est le vecteur des paramètres à estimer.
- Ces paramètres définissent une fonction F.

Definition

La descente de gradient classique est définie par :

$$P_k = P_{k-1} - \gamma \nabla E(P_{k-1})$$

1. Présentation générale de la méthode du SGD

Remarques :

- γ est appelé taux d'apprentissage. On prend en général $\gamma = 0.01$. (Nous allons en disctuter plus en détail dans la section 3),
- L'itération s'arrête lorsque les paramètres minimisent globalement (à différencier d'un minimum local) la fonction de perte,
- Dans la vie quotidienne, c'est lorsque nous avons descendu au point le plus bas de la vallée.

1. Présentation générale de la méthode du SGD

Grand problème :

- Notre erreur E dépend de toutes nos données, si on a par exemple 1,000,000 de données (ce qui arrive très souvent en Machine Learning), ça va nous poser des problèmes de mémoire et de longueur de calcul.
- En d'autre terme, l'algorithme trouve les valeurs des paramètres minimisant la fonction de perte des heures,voire même des jours après le début de l'algorithme.

1. Présentation générale de la méthode du SGD

Solution :

- Utiliser la méthode du gradient stochastique
- L'algorithme de la SGD est la même que celle de la descente classique, la seule et **grande** différence réside dans la fonction de perte. En effet :
- L'idée est d'utiliser **une seule donnée** (y_i, x_{ij}) à la fois, à chaque itération au lieu d'utiliser toutes les données. x_{ij} est la *ième* observation de la *jème* variable **x** .

1. Présentation générale de la méthode du SGD

Definition

La descente de gradient stochastique est définie par :

$$P_k = P_{k-1} - \gamma \nabla E_i(P_{k-1})$$

Remarques

- i est choisi de manière aléatoire, d'où le fait que c'est une descente de gradient stochastique
- On évalue donc le gradient qu'à la *ième* observation, et cela à chaque itération
- i change à chaque itération.
- Puisqu'on ne mobilise qu'une seule donnée à la fois, l'algorithme requiert moins de temps et converge plus vite.

Section 2

L'estimateur du maximum de vraisemblance

2. Le maximum de vraisemblance

Definition

La fonction de vraisemblance est définie par :

$$L_n : \Theta * \mathbb{R}^n \to \mathbb{R}^+$$
$$L_n(\theta; x_1, \ldots, x_n) = \prod_{i=1}^{n} f_X(x_i; \theta)$$

Lorsque la variable est discrète, alors $L_n : \Theta * \mathbb{R}^n \to [0,1]$

2. Le maximum de vraisemblance

Definition

L'estimateur du maximum de vraisemblance $\hat{\theta}$ de $\theta \in \Theta$ est une solution du problème de maximisation suivante :

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \ln \mathcal{L}(\theta; x|y)$$

Remarque

- L'estimateur du maximum de vraisemblance $\hat{\theta}$ est une variable aléatoire et l'estimation du maximum de vraisemblance $\hat{\theta}(x)$ correspond à la réalisation de $\hat{\theta}$ sur l'échantillon **x**

2. Le maximum de vraisemblance

Definition

Sous certaines conditions de régularités, l'estimateur du maximum de vraisemblance de θ est la solution du condition de premier ordre (CPO) :

$$\left.\frac{\partial \ln \mathcal{L}(\theta; y|x)}{\partial \theta}\right|_{\hat{\theta}} = \mathbf{0_K}$$

Cette condition est généralement appelée l'équation de la log-vraisemblance.

2. Le maximum de vraisemblance

Definition

Le **gradient** associé à l'échantillon $(y_i, \mathbf{x_i})_{i=1}^n$ correspond à la dérivée première de la log-vraisemblance conditionnel :

$$g_n(\theta; y|x) = \frac{\partial \ln \mathcal{L}(\theta; y|x)}{\partial \theta}$$

Le gradient **évalué au point du maximum de vraisemblance** $\hat{\theta}$ est :

$$g_n(\hat{\theta}; y|x) = \frac{\partial \ln \mathcal{L}(\theta; y|x)}{\partial \theta}\bigg|_{\hat{\theta}} = \mathbf{0_K}$$

2. Le maximum de vraisemblance

Proposition

La condition du second ordre (CSO) du problème de maximisation du vraisemblance : la matrice Hessienne évaluée au point $\hat{\theta}$ doit être définie négative.

$$\left.\frac{\partial^2 \ln \mathcal{L}(\theta; y|x)}{\partial\theta\partial\theta^\top}\right|_{\hat{\theta}} \text{ est définie négative.}$$

2. Le maximum de vraisemblance

Exemple (loi logistique standard)

- En régression logistique, la probabilité de la classe $y_i = 1$ est donnée par le modèle :

$$p_i = P(y_i = 1 \mid x_i) = \frac{1}{1 + e^{-\beta^T x_i}},$$

où $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$ est le vecteur des paramètres, et x_i représente les prédicteurs.

- La log-vraisemblance d'une loi logistique standard est donnée par :

$$\ell(\beta) = \sum_{i=1}^{n} \Big(y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \Big)$$

2. Le maximum de vraisemblance

- Pour maximiser la vraisemblance, il faut résoudre :

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{n} (y_i - p_i) x_i = 0.$$

Cependant, comme p_i dépend de β à travers une fonction exponentielle, ces équations sont **non linéaires** en β, ce qui empêche de trouver une solution analytique.

2. Le maximum de vraisemblance

Solution : Algorithme de Gauss-Newton

Definition

En supposant une valeur initiale β^0, l'algorithme de **Gauss-Newton** est défini par l'itération suivante :

$$\beta^{k+1} = \beta^k + \delta\beta$$

où $\delta\beta$ vérifie les équations normales :

$$\left(\mathbf{J_r^T J_r}\right) \delta\beta = -\mathbf{J_r^T r},$$

2. Le maximum de vraisemblance

Etapes de l'algorithme :

- Initialiser β^0 (valeurs initiales des paramètres).
- À chaque itération k, mettre à jour β^k selon la règle :

$$\beta^{k+1} = \beta^k - \left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}^T\mathbf{r},$$

où :

- $\mathbf{r} = \left(y_1 - p_1, y_2 - p_2, \ldots, y_n - p_n\right)^T$ est le vecteur des résidus,
- $\mathbf{J}$ est la matrice Jacobienne des dérivées partielles de p_i par rapport à β :

$$J_{ij} = \frac{\partial p_i}{\partial \beta_j} = y_i(1 - p_i)x_{ij}.$$

- Répéter jusqu'à la convergence de β i.e jusqu'à ce que $|\beta^{k+1} - \beta^k|$ soit suffisamment petit.

2. Le maximum de vraisemblance

Remarque :

- La Jacobienne $\mathbf{J}$ du modèle logistique est donnée par :

$$\mathbf{J} = \begin{bmatrix} y_1(1-p_1)x_{11} & y_1(1-p_1)x_{12} & \cdots & y_1(1-p_1)x_{1p} \\ y_2(1-p_2)x_{21} & y_2(1-p_2)x_{22} & \cdots & y_2(1-p_2)x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ y_n(1-p_n)x_{n1} & y_n(1-p_n)x_{n2} & \cdots & y_n(1-p_n)x_{np} \end{bmatrix}.$$

Section 3

Application du SGD dans le cadre de l'estimation par maximum de vraisemblance (MLE)

3.Application du SGD dans le cadre MLE

Definition

L'algorithme de la SGD dans le cadre du maximum de vraisemblance est défini à travers l'itération suivante :

$$\theta_n^{\text{SGD}} = \theta_{n-1}^{\text{SGD}} - \gamma \nabla \log f(Y_i; \mathbf{X}_i, \theta_{n-1}^{\text{SGD}})$$

Remarques :

- θ_0 est notre vecteur de paramètre initiale,
- On peut l'initialiser à $(1, \ldots, 1)'$ par exemple.

Exemple (loi logistique standard)

- En reprenant l'équation vue précédemment :

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{n} (y_i - p_i) x_{ij} = 0.$$

- En appliquant l'algorithme de la descente de gradient stochastique on a :

$$\beta_j^n := \beta_j^{n-1} + \gamma \frac{\partial \ell(\beta_j)}{\partial \beta_j}$$

- Comme on ne calcule le gradient qu'à une *ième* observation, i choisi de manière uniforme sur $\{1,2,\ldots,n\}$

$$\beta_j^n := \beta_j^{n-1} + \gamma (y_i - p_i) x_{ij}$$

où γ est le taux d'apprentissage

Exemple suite : loi logistique standard

Remarque :

- La fonction de perte d'une loi logistique standard est définie par :

$$-\frac{1}{n}\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

- Puisque $\forall x \in \mathcal{R}_*^+$, le logarithme népérien est concave, on cherche donc le maximum d'une fonction,
- Comme, le SGD étant un algorithme de minimisation, on a le signe négatif devant.
- Rappel :
  - $\forall x, -max(x) = min(x)$

3. Avantage du SGD dans le cadre du MLE

D'un point de vue théorique

- **Robbins et Monroe (1951)** dans leur papier ont montré que grâce à la théorie des approximations stochastiques θ_n^{SGD} converge vers un point θ_∞ avec

$$\mathbb{E}\left[\nabla \log f(y_n \mid x_n, \boldsymbol{\theta}_\infty)\right] = 0$$

- Or, les propriétés statistiques standard affirment que :

$$\mathbb{E}\left[\nabla \log f(y_n \mid x_n, \boldsymbol{\theta}_*)\right] = 0$$

où θ_* est la vraie valeur du paramètre,

- Ce point est unique sous certaine conditions de régularité (**Lehmann, Castilla, Thm 5.1, p.463**),
- Comme la log-vraisemblance est concave, on respecte ces conditions,
- Donc, $\theta_\infty = \theta_*$, i.e θ_n^{SGD} converge vers la vraie valeur du paramètre que l'on cherche à estimer.

3. Incovénients du SGD dans le cadre du MLE

Mauvaise spécification du taux d'apprentissage

- Malgré les garanties théoriques, le SGD **n'est généralement pas robuste** quant à la mauvaise spécification du taux d'apprentissage.
- En effet, si le paramètre fixé γ tend vers 0, l'itération converge **très lentement** en pratique,
- Tandis que de grandes valeurs du paramètre peuvent provoquer une **divergence numérique**,
- Il faut donc très bien choisir son taux d'apprentissage γ.

Section 4

Application de l'algorithme de la SGD sous Python dans le cadre de la loi logistique standard

Implémentation sous Python

```python
'''
Implémentation Python de l'algorithme du SGD
dans le cadre de l'estimation des paramètres d'une loi
logistique standard (modèle logit)
'''
# Les imports :

import numpy as np
import matplotlib.pyplot as plt

# Fonction sigmoïde :
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Gradient de la fonction logistique :
def log_gradient (x,y,p) :
    return np.dot((y - p),x)  #= <=> (Y-p)X'

# Fonction de perte de la régression logistique (comme dans la présentation)
def fct_de_perte (y,pi) :
    return -np.mean(y*np.log(pi) + (1-y) * np.log(1 - pi))
```

Implémentation sous Python

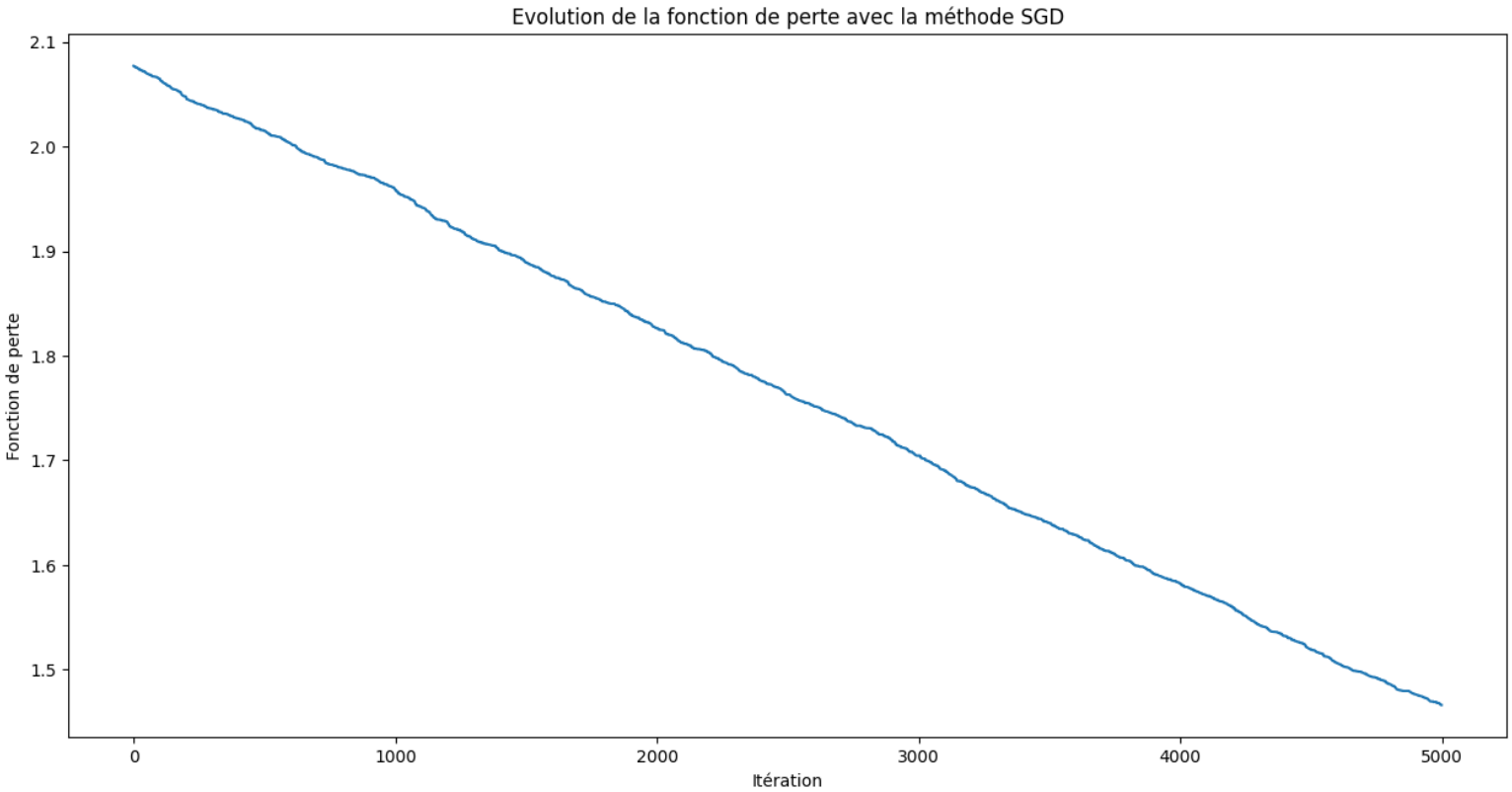
```python
def logistic_sgd (Y,X,learning_rate,nb_iteration) :
    '''Y est de taille n, X de taille n*D (car on suppose
     qu il y a D variables)'''
    n,D = X.shape
    beta = np.ones(D) # Initialisation de notre vecteur de paramètres
    gradient = np.ones((D, 1))  # Initialisation de notre gradient
    cout = list() # Initilisation d'une liste contenant notre fonction
    # de perte pour chaque valeur de y,x et beta.

    for it in range(0,nb_iteration) :
        '''Mélanger les données et itérer sur chaque itération
        Comme i est choisi de manière uniforme sur {1,2,...,n}'''
        i = np.random.uniform(1, n, size=1)
        i_scalar = i.item() # Extraire l'élément du vecteur
        # en un scalaire (c'est un tableau de taille 1)
        i_int = int(i_scalar) # On le converti en entier
        xi = X[i_int] # On prend la ième observation de la variable X
        yi = Y[i_int] # On prend la ième observation de la variable X
        pi = sigmoid(np.dot(xi, beta))
        for j in range(D):
            '''Calcul du gradient'''
            gradient[j] = log_gradient(xi[j], yi,pi)  #le gradient est une
            '''Mise à jour des paramètres : '''       #matrice ligne de taille D
            beta[j] += learning_rate * gradient[j] # Valeur de beta0,...,betaD
        '''Calcul du coût à la fin de chaque itération pour le suivi'''
        pi_all = sigmoid(np.dot(X, beta))
        perte = fct_de_perte(Y, pi_all)
        cout.append(perte)

    return beta, cout
```

Implémentation sous Python

```
55  n = 100
56  np.random.seed(123)
57  # Y prend la valeur 0 ou 1 avec probabilité 1/2
58  Y = np.random.binomial(1,0.5,size = np.array(([n,1])))
59
60  np.random.seed(123)
61  # Simulation des données d'entrées (variables explicatives)
62  X1 = np.random.normal(0,1,size = np.array(([n,1])))
63  X2 = np.random.gamma(1,2,size = np.array(([n,1])))
64  X3 = np.random.exponential(scale=1,size = np.array(([n,1])))
65
66  # Mélange de nos données d'entrées simulées
67  X = np.hstack((X1,X2,X3))
68
69  # ajout d'une colonne pour le calcul de la constante
70  design_matrix = np.hstack((np.ones((n,1)),X))
71
72  nb_iteration = 5000
73  beta,perte= logistic_sgd(Y,X=design_matrix,learning_rate=0.01,
74                           nb_iteration = nb_iteration)
75
76
77  # Affichage des pertes
78  plt.plot(perte)
79  plt.xlabel('Itération')
80  plt.ylabel('Fonction de perte')
81  plt.title('Evolution de la fonction de perte avec la méthode SGD')
82  plt.show()
```

Application sous Python

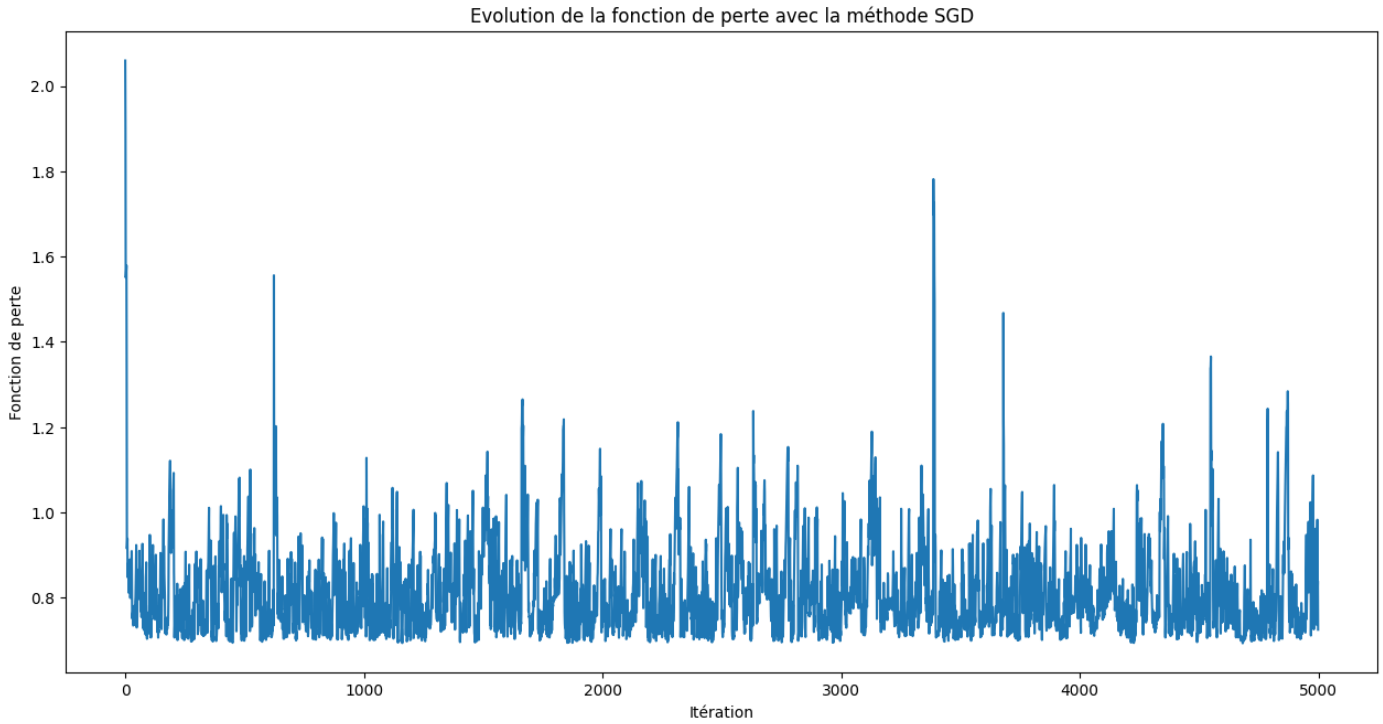
- Si γ est très faible, i.e tend vers 0, on a une convergence très lente :



Avec $\gamma = 0.0001$

Application sous Python

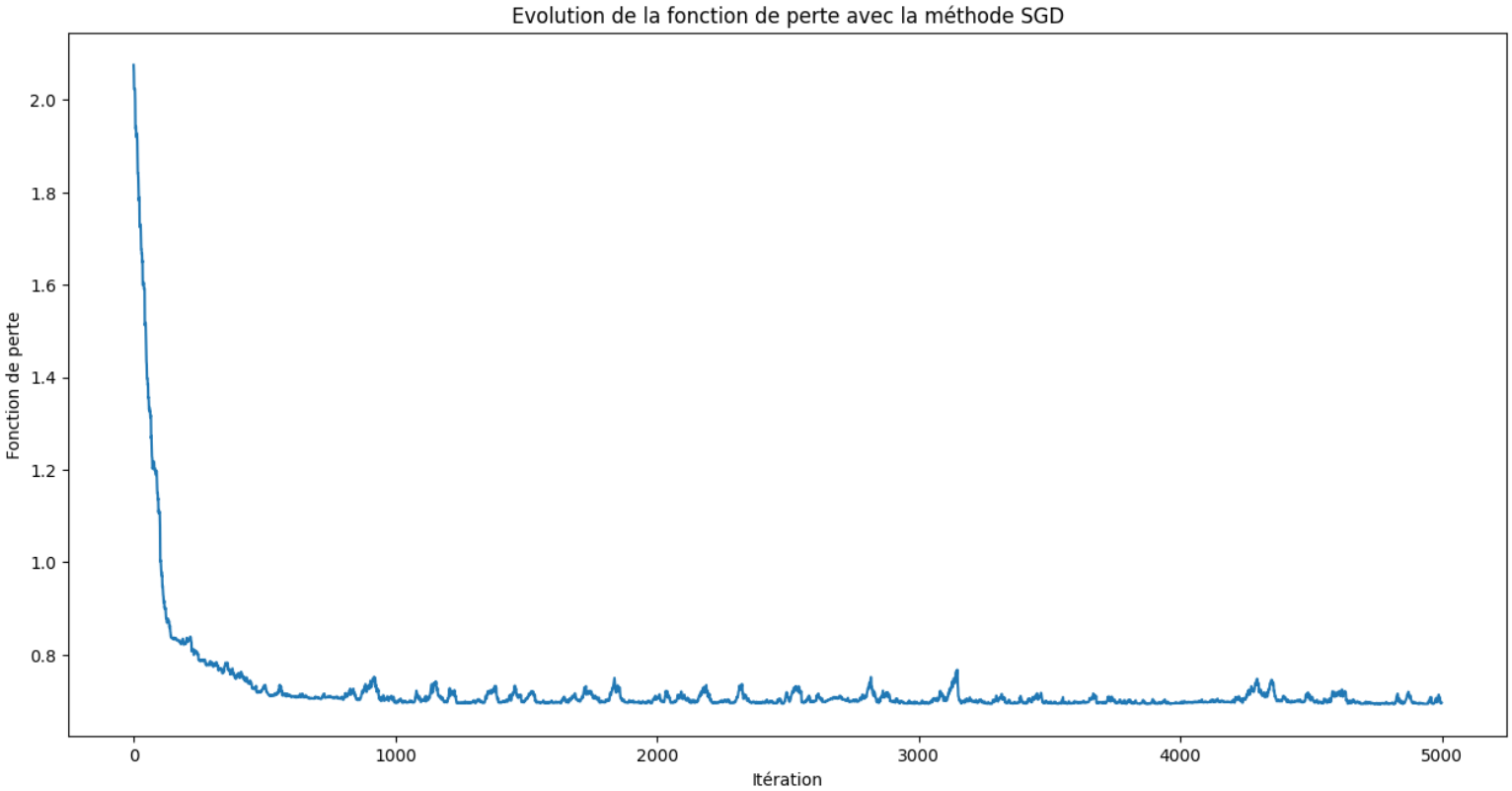
- Si γ est grand, on a une forme très oscillante de la fonction de perte dû aux bruits du gradients (le bruit fait référence à l'aspect aléatoire d'un processus, par exemple un bruit blanc) :



Avec $\gamma = 0.1$

Application sous Python

- Avec la bonne valeur de γ , on a une convergence rapide sans oscillation :



Avec $\gamma = 0.01$