# Projet Tutoré

Geoffrey Laforest & Thomas Delliaux

Avril 2022

## Table des matières



## Introduction

L'intérêt porté au *machine learning* - ou apprentissage automatique - n'a cessé de croître ces dernières décennies, et a été en particulier renouvellé par l'essor du *deep learning* et des réseaux de neurones.

Les algorithmes dits *online* ou stochastiques ont très vite émergé, dès les débuts de l'apprentissage automatique et de l'informatique. C'était une ère qui commandait la simplicité algorithmique et la facilité d'implémentation. Des impératifs auxquels répondaient bien les algorithmes susmentionnés [7].

Un algorithme *online* ou stochastique consiste à répéter l'opération suivante : un exemple est pris au hasard, et les paramètres du modèle d'apprentissage sont mis à jour sur la base de cet exemple uniquement. On peut généralement distinguer deux cas [8].

Ou bien cette phase d'apprentissage se poursuit jusqu'à ce que l'algorithme soit capable de fournir des solutions satisfaisantes sur des données qu'il n'a pas vu lors de son entraînement. C'est d'ailleurs cette étape d'évaluation du modèle sur de nouvelles données qui permet de dire si l'algorithme a appris quelque chose. Le cas échéant, on parle phénomène de généralisation.

L'apprentissage peut également se poursuivre indéfiniment, en même temps que l'algorithme est utilisé pour faire des prédictions, dès lors qu'il reçoit constamment de nouvelles données issues du monde réel et auxquelles on souhaite qu'il s'adapte. Un tel algorithme est dit adaptatif.

Ces algorithmes ont connu et connaissent encore un franc succès, car ils se sont avérés très efficaces pour l'apprentissage en grande dimension. Si la puissance de calcul des ordinateurs n'a cessé d'augmenter, elle n'a pas pour autant pu suivre l'explosion de la production des données [6].

Les réseaux de neurones ont pu tirer avantage de ces évolutions relativement récentes, notamment celle du *Big Data*, terme qui fait référence à ces très grands jeux de données. Nécessitant alors une puissance de calcul parfois considérable, les algorithmes stochastiques se révèlent fort utiles en ce qu'ils requièrent de traiter moins d'exemples, et demandent donc moins de calculs, que des algorithmes d'optimisation plus classiques pour parvenir à une bonne solution. C'est donc cette convergence vers une bonne solution, ainsi que la vitesse à laquelle elle a lieu, qu'on se propose d'étudier ici.

L'étude de la convergence des algorithmes *online* est en effet plus complexe que celle d'autres algorithmes d'optimisation [1], comme la descente de gradient classique. Elle relève de la théorie de l'approximation stochastique.

On présentera d'abord le cadre général de l'apprentissage, basé notamment sur la méthode de la descente de gradient.

On montrera ensuite que, sous de bonnes hypothèses, la méthode de descente de gradient stochastique converge.

Puis nous montrerons que sa vitesse de convergence est au moins aussi bonne, voire meilleure, que pour celle de la descente de gradient classique.

La dernière section sera consacrée à la présentation des résultats expérimentaux.

# 1 Cadre général de l'apprentissage automatique

## 1.1 Fonction de coût

L'objectif d'un système d'apprentissage est généralement de minimiser la fonction $C(w)$, appelée fonction de coût. C'est comme cela qu'on modélise mathématiquement le fait d'apprendre [2], qui autrement peut apparaître comme une expérience subjective se fondant difficilement dans le formalisme mathématique [3].

Cette fonction de coût s'écrit ainsi :

$$C(w) \coloneqq \mathbb{E}_z\big[Q(z,w)\big] \coloneqq \int Q(z,w)dP(z) \tag{1}$$

où $w$ est un vecteur, $Q$ est une fonction dite de perte, et $z$ une observation correspondant à la réalisation d'une variable aléatoire de loi $P$.

Plus précisément, $w$ est le vecteur paramètre. On parle aussi de poids pour désigner les paramètres contenus dans ce vecteur. C'est cette variable que l'on modifie au cours du processus d'apprentissage afin de minimiser $C(w)$. Elle est donc censée s'adapter en fonction des exemples traités par le système, et représente ce que celui-ci apprend.

La fonction de perte $Q(z,w)$ mesure la performance du modèle pour un $w$ donné, relativement à l'évènement $z$. Celui-ci est une réalisation de la variable aléatoire $Z$. Celle-ci suit la distribution $P$, qui est la vraie distribution du phénomène que l'on cherche à prédire avec notre algorithme. Les réalisations de cette variable aléatoire sont supposées indépendantes et identiquement distribuées. Elles constituent les observations issues du monde réel.

La fonction de coût $C(w)$ est alors simplement l'espérance de la fonction de perte $Q(z,w)$ pour une valeur fixée de $w$. On parle aussi de l'espérance du risque.

Cette définition reste très théorique dans la mesure où la distribution $P$ est inconnue par hypothèse.

Un exemple permet de rendre intuitif cette modélisation. Si un modèle de classification binaire doit classer des photos comme étant celles d'un chat ou pas, la fonction de perte prend une valeur positive plus ou moins élevée qui fait office de pénalité. Elle pénalise donc le fait que l'algorithme se trompe. La fonction de coût peut alors être regardée comme la somme des pénalités liée à l'ensemble des erreurs commises par le modèle. Cela signifie que plus la valeur de la fonction de coût est élevée, plus le système s'est trompé. Inversement, plus elle est faible, moins le modèle a commis d'erreur, indiquant ainsi qu'il a appris à reconnaître correctement une photo de chat. D'où le lien entre apprentissage et minimisation de la fonction de coût $C(w)$.

## 1.2 Méthodes de descente de gradient

La fonction de coût ne peut pas être optimisée directement, du fait que la vraie distribution est inconnue. Il est néanmoins possible de calculer une approximation de $C(w)$ en utilisant un *dataset* d'entraînement fini, constitué d'observations $z_1, ..., z_L$, indépendantes et identiquement distribuées.

$$C(w) \approx \hat{C}_L(w) \coloneqq \frac{1}{L} \sum_{n=1}^{L} Q(z_n, w)$$

Il a été montré que minimiser $\hat{C}_L(w)$, qu'on appelle alors le risque empirique, peut fournir une bonne estimation du minimum de $C(w)$ quand le *dataset* d'entraînement est suffisamment grand.

### 1.2.1 Descente de gradient par *batch*

Minimiser le risque empirique $\hat{C}_L(w)$ peut se faire en utilisant un algorithme de descente de gradient par *batch*. C'est l'algorithme de descente de gradient classique. On calcule des estimations successives du paramètre optimal $w_t$ en utilisant la formule suivante, où $\gamma_t$ est un nombre positif :

$$w_{t+1} = w_t - \gamma_t \frac{1}{L} \nabla_w \hat{C}_L(w_t) = w_t - \gamma_t \frac{1}{L} \sum_{i=1}^{L} \nabla_w Q(z_i, w_t)$$

Le comportement de cet algorithme d'optimisation est bien connu. En général, il est possible d'obtenir la convergence vers un minimum local du risque empirique $\hat{C}_L(w)$. Quand celle-ci a lieu, elle est fortement accélérée lorsqu'on a recours à une méthode dite d'ordre deux, où l'on remplace le scalaire $\gamma_t$ par une matrice définie positive bien choisie.

Dans la descente de gradient par *batch*, le terme *batch* désigne l'ensemble des exemples présents dans le *dataset* d'entraînement, et qui sont tous passés en revue par l'algorithme lors de chaque itération avant la mise à jour des paramètres.

Voilà pourquoi cet algorithme implique des calculs fastidieux : il calcule la moyenne des gradients de la fonction de perte $\nabla_w Q(z_n, w)$ à partir de tous les exemples du *dataset*. Des ressources considérables doivent être allouées afin de stocker tous les exemples en mémoire dans l'ordinateur et opérer tous ces calculs, notamment quand le jeu de données est important. Le temps de calcul peut donc être très élevé, voire prohibitif. Ceci amène à relativiser les performances de l'algorithme, et à lui préférer parfois d'autres techniques d'apprentissage.

### 1.2.2 Descente de gradient stochastique

La grande différence entre la descente de gradient par *batch* et celle *online* réside principalement dans le calcul du gradient de leur fonction de coût respective. Dans l'algorithme de descente classique, on fait la moyenne de tous les

gradients calculés à partir de tous les exemples présents dans le *batch*. Dans le cas stochastique, on abandonne cette opération de moyennisation et on estime le gradient de la fonction de coût à partir d'un seul exemple tiré au hasard. À chaque itération donc, l'algorithme stochastique calcule le gradient de la fonction de perte à partir de l'exemple considéré et met ensuite directement à jour les paramètres selon la formule suivante :

$$w_{t+1} = w_t - \gamma_t \nabla_w Q(z_t, w_t) \tag{2}$$

Calculer le gradient grâce à tous les exemples du *dataset* permet de saisir la direction générale dans laquelle il faut aller pour se diriger vers le minimum, puisque chaque exemple a pu apporter sa contribution au calcul de celle-ci. Si l'évaluation du gradient en un exemple pouvait suggérer une direction différente de celle générale qui nous aurait éloigné du minimum local, cette évaluation est contre-balancée par les valeurs du gradient provenant des autres données du *dataset* d'entraînement.

Il en va différemment du gradient stochastique, pour lequel la direction est calculée à chaque fois à partir d'un seul exemple, qui en plus n'est peut-être pas le plus représentatif de l'allure générale de la distribution $P$. Il faut donc espérer que le bruit introduit dans cet algorithme ne perturbe pas son comportement général, et notamment sa convergence vers le minimum.

Enfin, il est commode de décrire la descente de gradient *online* sans référence à un *dataset* d'entraînement. Chaque itération utilise un exemple $z_t$ que l'on regarde comme provenant de la vraie distribution $P$ dont il était question plus haut, et non plus comme issu d'un jeu de données fini. L'algorithme optimise alors directement la fonction de coût $C(w)$ - et non plus le risque empirique $\hat{C}_w$ - lorsqu'il met à jour le gradient après le traitement d'un exemple.

L'avantage de ce choix réside dans le fait qu'il permet d'éviter les discussions habituelles quant aux différences entre optimisation du risque empirique et optimisation de la fonction de coût. Il facilite aussi l'analyse de la vitesse de convergence de l'algorithme stochastique et permet de mieux en souligner l'efficacité, comme on le verra dans la section trois.

### 1.2.3 Cas des fonctions de perte non différentiables

Beaucoup d'algorithmes d'apprentissage très intéressants possèdent une fonction de perte $Q(z, w)$ qui n'est pas différentiable sur un ensemble de points de mesure nulle. L'intuition suggère que le problème est donc mineur dans la mesure où la probabilité d'atteindre un tel point est nulle. Même si cela devait arriver, on peut toujours écarter cet exemple et en regarder un autre.

Formalisons cette intuition. On part de l'algorithme général de descente de gradient stochastique, avec n'importe quel terme de mise à jour $H(z, w)$ qui remplit la condition (1). On fait l'hypothèse que la fonction de coût $C(w)$ est

différentiable quand la fonction de perte $Q(z, w)$ est intégrée contre la distribution de probabilité $P$.

Si l'on définit le terme de mise à jour de la façon suivante, cela reviendra à prendre un nouvel exemple chaque fois que l'on atteint un point non différentiable de la fonction de perte :

$$H(z, w) \coloneqq \begin{cases} \nabla_w Q(z, w) & \text{si } Q(z, w) \text{ est différentiable} \\ 0 & \text{sinon} \end{cases} \tag{3}$$

On suppose que pour chaque valeur du paramètre $w$, la fonction de perte est différentiable partout à l'exception d'un sous-ensemble négligeable d'exemples issus de la variable aléatoire $Z$.

La condition (1) peut alors se réécrire, en utilisant (3),

$$\int H(z, w) dP(z) = \int \nabla_w Q(z, w) dP(z)$$

et ce que l'on aimerait obtenir, afin d'évacuer le problème de la non-différentiabilité en certains points, est l'égalité suivante :

$$\int \nabla_w Q(z, w) dP(z) = \nabla_w \int Q(z, w) dP(z)$$

La théorie de l'intégration de Lebesgue nous offre, à travers le théorème de dérivation sous l'intégrale, une condition nécessaire et suffisante pour échanger l'opérateur d'intégration avec celui de différentiation. Pour chaque paramètre $w$ utilisé par l'algorithme, il suffit de trouver une fonction intégrable $\Phi(z, w)$ et un voisinage $\mathcal{V}(w)$ de $w$ tel que :

$$\forall z, \forall v \in \mathcal{V}(w), \quad |Q(z, v) - Q(z, w)| \leq |w - v| \Phi(z, w) \tag{4}$$

Cette condition assure la pente de $Q(z, w)$ est convenablement bornée. C'est vrai quand $Q(z, w)$ est différentiable et que son gradient est intégrable. C'est évidemment faux quand la fonction $Q(z, w)$ n'est pas continue. Cependant, il faut garder à l'esprit que les points où $Q(z, w)$ n'est pas différentiable (et non continue donc) sont contenus dans un ensemble de mesure nulle, c'est-à-dire que chacun de ces points est de probabilité nulle.

Rappelons que dans la théorie de Lebesgue, on a par convetion

$$0 \times \infty = 0$$

On a alors bien l'égalité suivante :

$$\int H(z, w) dP(z) = \int \nabla_w Q(z, w) dP(z) = \nabla_w \int Q(z, w) dP(z)$$

L'exigence que l'ensemble des points où $Q(z, w)$ n'est pas différentiable soit de mesure nulle assure donc que la condition (4) est suffisante pour les ignorer sans risque.

# 2 Convergence de l'algorithme stochastique

## 2.1 Hypothèses générales

Cette section s'attaque à la convergence de l'algorithme *online* général de descente de gradient appliqué à l'optimisation d'une fonction de coût différentiable $C(w)$ avec les propriétés suivantes :

1. La fonction de coût $C(w)$ a un unique minimum $w^*$
2. $C(w)$ satisfait la condition dite de convexité générale suivante :

$$\forall \epsilon > 0, \quad \inf_{\|w-w^*\|^2>\epsilon} (w - w^*) \cdot \nabla_w C(w) > 0 \tag{5}$$

## 2.2 Descente de gradient continue

La descente de gradient continue est une représentions mathématique de la descente de gradient idéale, c'est-à-dire qu'on suivrait la direction donnée par $-\nabla_w C(w)$ de façon "continue".

Bien qu'abstraite, la convergence de cette descente reste intéressante à étudier. En effet, la preuve de la convergence de la descente de gradient discrète et stochastique suivent les mêmes trois grandes étapes.

Commençons par définir l'équation différentielle suivante :

$$\frac{\partial w}{\partial t} = -\nabla_w C(w) \tag{6}$$

**Etape 1.** Définition d'une fonction de Lyapunov

On commence la preuve en définissant la fonction de Lyapunov

$$h(t) := \|w(t) - w^*\|^2$$

On va montrer que cette fonction converge vers 0 quand $t \longrightarrow \infty$ afin de montrer que $w(t) \underset{t\to\infty}{\longrightarrow} w^*$

**Etape 2.** Convergence de la fonction de Lyapunov

En calculant la dérivée de $h$, on obtient

$$\frac{\partial h}{\partial t} = -2(w - w^*) \cdot \nabla_w C(w).$$

Par hypothèse de convexité générale, on a $\frac{\partial h}{\partial t} \leq 0$. D'où $h$ est une fonction decroissante minorée par zéro. On a donc $h(t) \underset{t\to\infty}{\longrightarrow} l$ où $l \geq 0$

**Etape 3.** Convergence de l'algorithme de descente de gradient continue du fait de la convergence vers zéro de la fonction de Lyapunov.

Comme $h$ est monotone, et qu'elle converge quand $t \to \infty$, sa dérivée tend vers zéro et on a (Cette affirmation semble fausse en générale. Léon Bottou l'a justifié comme ça dans son papier [4], et nous n'avons pas réussi à montrer pourquoi c'est vrai) :

$$-2(w(t) - w^*) \cdot \nabla_w C(w(t)) \underset{t\to\infty}{\longrightarrow} 0$$

**Remarque importante**

Attention, l'argument qui vient d'être donné ci-dessus semble faux en général. Léon Bottou l'a justifié comme ça dans deux de ses papiers [3] [4]. Or, il existe des contre-exemples. Il suffit de penser à une fonction qui forme presque des plateaux et qui par endroits a une pente abrupte. Elle peut très bien être strictement décroissante et monotone, il y a toujours des endroits où sa dérivée ne sera pas zéro et augmentera à nouveau.

Il est possible, comme l'a mentionné notre tuteur, que l'auteur ait pu penser que la preuve pouvait être refaite facilement, mais qu'il manque un élément important pouvant être le lemme de Barbarat.

Ce lemme affirme que si $f(x)$ a une limite finie quand $x \longrightarrow +\infty$, et si $f'$ est uniformément continue (ou $f''$ est bornée), alors $f'(x) \underset{x\to+\infty}{\longrightarrow} 0$, ce qui semble correspondre au résultat que l'on veut ici montrer.

Plus généralement, c'est une situation où le théorème de Lyapunov s'applique, ce qui revient au même. Nous allons le montrer avant de poursuivre la preuve de la convergence de la descente de gradient dans le cas continu.

On est dans le cadre de l'équation différentielle définie plus haut (6).

On pose

$$\begin{aligned} x(t) &\coloneqq w(t) - w^* f(x) = x'(t) \\ f(x) &\coloneqq \frac{\partial x}{\partial t} = \frac{\partial(w(t) - w^*)}{\partial t} = \frac{\partial(w(t)}{\partial t} - \frac{\partial(w^*)}{\partial t} = \frac{\partial(w(t)}{\partial t} \\ &= -\nabla_w C(w(t)) \end{aligned}$$

Soit $t^* \in \mathbb{R}$, tel que $x(t^*) = 0$. On a alors

$$x(t^*) = 0 \iff w(t^*) - w^* = 0 \iff w(t^*) = w^*$$

Étant donné la définition de $f$, on a aussi

$$f(x(t^*)) = \nabla_w C(w(t^*)) = \nabla_w C(w*) = 0 \text{ car } w^* \text{ est le minimum de } C(w).$$

De là, et avec un léger abus de notation, on peut écrire que :

$$h(x) = 0 \iff x = 0$$
$$h(x) > 0 \iff x \neq 0$$

Ces deux équivalences sont facilement vérifiées puisque $h$ est une norme.

On a de plus

$$\begin{aligned}\frac{\partial h(x)}{\partial t} &= \nabla_x h(x) \cdot f(x) \\ &= 2(w - w^*)\frac{\partial w}{\partial t} \\ &= -2(w(t) - w^*) \cdot \nabla_w C(w(t)) \leq 0\end{aligned}$$

On a même que la dernière inégalité est toujours stricte vu les hypothèses, dès lors que $x \neq 0$.

C'est ce qui entraîne, d'après le théorème de stabilité de Lyapunov, que $x$ est non seulement stable, mais même asymptotiquement stable en raison de la dernière inégalité qui est stricte.

Cela entraîne que la trajectoire $x(t)$ tend vers 0, et on a donc le résultat que cherche à démontrer Léon Bottou et qui l'amenait à postuler une affirmation ambigüe sur la dérivée $\frac{\partial h}{\partial t}$. Ce résultat est le suivant :

$$w(t) - w^* \underset{t\to\infty}{\longrightarrow} 0, \text{ i.e } w(t) \longrightarrow w^* \tag{7}$$

**Étape 3 (suite)**

On peut maintenant reprendre la preuve faite par Léon Bottou dans son papier [3].

Supposons que $h$ converge vers une valeur plus grande que zéro. Il existe donc $\epsilon > 0$ tel que pour tout $t \in \mathbb{R}$, $h(t) = \|w(t) - w^*\|^2 > \epsilon$. Or, par hypothèse de convexité générale, pour $t \in \mathbb{R}$, si $\|w(t) - w^*\|^2 > \epsilon$, alors

$$(w(t) - w^*) \cdot \nabla_w C(w(t)) \geq \inf_{\|w-w^*\|^2>\epsilon} (w - w^*) \cdot \nabla_w C(w) > 0$$
$$(w(t) - w^*) \cdot \nabla_w C(w(t)) \geq \inf_{\|w-w^*\|^2>\epsilon} (w - w^*) \cdot \nabla_w C(w) > 0$$

D'où l'on tire que

$$-2(w(t) - w^*) \cdot \nabla_w C(w(t)) \leq -2 \inf_{\|w-w^*\|^2>\epsilon} (w - w^*) \cdot \nabla_w C(w) < 0$$

En passant à la limite quand $t \to \infty$, on obtient $0 < 0$, ce qui est absurde.

On a donc bien $h(t) \underset{t\to\infty}{\longrightarrow} 0$. Finalement on en déduit que $w(t) \underset{t\to\infty}{\longrightarrow} w^*$. La descente de gradient continue converge bien vers le minimum du risque d'espérance.

## 2.3 Descente de gradient discrète

L'analyse de la convergence du gradient discret suit 3 grandes étapes.

**Étape 1.** Définition d'une suite de Lyapunov et critère de convergence d'une suite positive

On définit une suite de Lyapunov, c'est-à-dire une suite de nombres positifs qui représentent l'écart entre la $t$-ième prédiction faite par le modèle et la cible, qui désigne la réponse correcte attendue. Cette suite a la même forme que la fonction de perte de l'erreur quadratique moyenne, et s'écrit

$$h_t = \|(w_t - w^*)\|^2$$

Pour étudier la convergene d'une telle suite, il est alors utile d'en connaître un critère de convergence.

**Critère suffisant de convergence**

Soit $(u_t)_{t\in\mathbb{N}^*}$ une suite. Lorsqu'une suite converge, c'est qu'à partir d'un certain rang, les oscillations de ses termes sont de plus en plus petites autour de sa limite. Une suite convergente est donc une suite dont on peut maîtriser les oscillations. Les oscillations peuvent être vues comme les variations entre les termes de la suite, une variation étant définie par $u_t - u_{t-1}$. Les variations sont positives dès lors que $u_t > u_{t-1}$, et négatives sinon.

Une suite dont les variations diminuent suffisamment vite à partir d'un certain rang est une suite dont on peut contrôler les oscillations. Dans le cas d'une suite positive, si l'on sait que la somme des variations positives est finie, on est certain que la série converge. En effet, tous les termes étant positifs, la somme des variations négatives est nécessairement plus petite ou égale à la somme de celles positives et du premier terme de la suite, sinon on ne pourrait avoir que tous les termes sont positifs. Autrement dit, cette somme est elle aussi finie et on est sûr alors que la suite est bornée. La série, qui est la limite d'une suite croissante majorée, est donc convergente.

Formellement, on définit respectivement la somme des variations positives et celle des variations négatives ainsi :

$$S_t^+ \coloneqq \sum_{i=1}^{t-1}(u_{i+1} - u_i)_+ \text{ avec } (x)_+ \coloneqq \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}$$

$$S_t^- \coloneqq \sum_{i=1}^{t-1}(u_{i+1} - u_i)_- \text{ avec } (x)_- \coloneqq \begin{cases} 0 & \text{si } x \geq 0 \\ x & \text{sinon} \end{cases}$$

On appelle $S_\infty^+$ et $S_\infty^-$ les limites respectives (quand elles existent) de ces deux sommes, et on a alors :

$$\forall\, t \geq 1,\ \ 0 \leq u_t \leq u_1 + S_t^+ + S_t^- \leq u_1 + S_\infty^+ + S_t^- \leq u_1 + S_\infty^+$$

On a donc une borne supérieure puisqu'on a supposé ici que $S_\infty^+$ est finie. Mais également, du fait de la positivité des termes de la suite, une borne inférieure qui nous est donnée par l'inégalité précédente. En effet,

$$-u_1 - S_\infty^+ \leq S_t^- \leq 0$$

$S_t^-$ est donc bornée inférieurement, et étant une suite décroissante, il en va de même pour sa limite.

$(u_t)$ converge donc vers $u_\infty = u_1 + S_\infty^+ + S_\infty^- \geq 0,\ \text{car}\ \forall\, t \geq 1,\ \ u_t \geq 0$

On a donc un critère suffisant pour la convergence d'une suite de nombres positifs, et donc pour la suite de Lyapunov. Il suffit que la série des variations positives soit convergente pour que la suite en question le soit également.

**Étape 2.** Convergence de la suite de Lyapunov

D'après l'expression de $h_t$ et l'algorithme de la descente de gradient discret, une variation pour la suite de Lyapunov s'écrit :

$$\begin{aligned}
h_{t+1} - h_t &= \|w_{t+1} - w^*\|^2 - \|w_t - w^*\|^2 \\
&= \|w_{t+1}^2\| - \|w_t^2\| - 2w^* \cdot (w_{t+1} - w_t) \\
&= \|w_t - \gamma_t \nabla_w C(w_t)\|^2 - \|w_t\|^2 + 2w^* \cdot \gamma_t \nabla_w C(w) \\
&= -2w_t \cdot \gamma_t \nabla_w C(w_t) + \|\gamma_t \nabla_w C(w_t)\|^2 + 2w^* \cdot \gamma_t \nabla_w C(w_t) \\
&= -2(w_t - w^*) \cdot \gamma_t \nabla_w C(w_t) + \gamma_t^2 \|\nabla_w C(w_t)\|^2
\end{aligned}$$

Le critère de convexité générale assure que le premier terme de l'expression obtenue à la dernière ligne est toujours négatif. Contrairement au cas continu, il existe un second terme, dont il faut s'assurer que l'on puisse le contrôler afin que la convergence de l'algorithme de descente de gradient dans le cas discret soit garantie.

Ce second terme, c'est $\gamma_t^2 \|\nabla_w C(w_t)\|^2$, dont la présence s'explique par le fait que $h_{t+1}$ s'exprime en fonction du carré de $w_{t+1}$, qui lui-même s'écrit en fonction de $\nabla_w C(w)$.

Si, pour tout $t$, on contrôle chacun des deux termes de $\gamma_t^2 \|\nabla_w C(w_t)\|^2$, on sera alors certain qu'il y a convergence. D'où les deux conditions suivantes, qui combinées nous offrent la garantie recherchée :

1. $\sum_{i=1}^{\infty} \gamma_i^2 < \infty$
2. $$\|\nabla_w C(w_t)\|^2 \leq A + B \left\|w_t - w*\right\|^2,\ A, B \geq 0 \tag{8}$$

La première condition assure que le pas d'apprentissage décroît suffisamment rapidement, d'où le fait qu'il soit fini. Si ce n'était pas le cas, la valeur des itérés dans l'algorithme de descente de gradient pourrait possiblement être indéfiniment augmentée. Quand bien même le gradient diminuerait lui aussi, la valeur de $\gamma_t^2 \|\nabla_w C(w_t)\|^2$ pourrait être suffisamment élevée pour faire diverger l'algorithme. Il suffit de penser que ce que l'on ajoute à chaque itération soit équivalent à $\frac{1}{n}$ pour bien comprendre le problème. On incrémente alors la valeur de chaque itéré, le dernier itéré pouvant s'écrire comme $u_1$ et la somme des variations observées entre ce premier terme et le dernier itéré. Il est alors facile de voir pourquoi l'algorithme va diverger, puisqu'on a dans ce cas

$$\forall\, t \geq 1,\ \lim_{t\to+\infty} u_t = u_1 + \lim_{t\to+\infty} \sum_{i=1}^{t-1} (u_{i+1} - u_i) \underset{n\to+\infty}{\sim} u_1 + \sum_{n=1}^{\infty} \frac{1}{n} = +\infty$$

La seconde condition assure que le gradient n'augmente pas trop vite quand on s'éloigne un peu du minimum, ce qui pourrait entraîner la divergence de la suite. En effet, la décroissance des pas d'apprentissage pourrait être compensée par une croissance bien plus rapide du gradient. C'est ce que l'on peut observer si cette dernière est exponentielle alors que la décroissance de la suite $(\gamma_t^2)$ n'est que polynomiale. Par croissance comparée, la série des $\gamma_t^2 \|\nabla_w C(w_t)\|^2$ va alors diverger, ce qui signifie que la suite de Lyapunov diverge. L'algorithme de descente de gradient discret ne convergera donc pas.

La preuve se poursuit à l'aide de deux suites qu'il va nous falloir définir.

Posons d'abord

$$\mu_t \coloneqq \prod_{i=1}^{t-1} \frac{1}{1 + \gamma_i^2 B}$$

Montrons que $\mu_t \underset{t\to+\infty}{\longrightarrow} \mu_\infty > 0$

Tout d'abord, remarquons que l'on a

$$\forall\, i \geq 1,\ 0 < \frac{1}{1 + \gamma_i^2 B} \leq 1$$

Cela implique que la limite de $\mu_t$ est toujours inférieure ou égale à 1 en tant qu'elle est limite d'un produit de termes tous inférieurs ou égaux à 1.

Rappelons que

$$\forall\, x \geq 0,\ \ln(1 + x) \leq x$$

On a alors,

$$\forall t \geq 1,\ \ln(\mu_t) = \ln\left(\prod_{i=1}^{t-1} \frac{1}{1+\gamma_i^2 B}\right) = \sum_{i=1}^{t-1} \ln\left(\frac{1}{1+\gamma_i^2 B}\right)$$
$$= \sum_{i=1}^{t-1} \ln(1) - \sum_{i=1}^{t-1} \ln(1+\gamma_i^2 B) \geq -\sum_{i=1}^{t-1} \gamma_i^2 B$$

On a donc

$$\lim_{t\to+\infty} \mu_t \geq \lim_{t\to+\infty} \exp\left(-\sum_{i=1}^{t-1} \gamma_i^2 B\right) = \exp(-cB) > 0, \text{ avec } c \coloneqq \sum_{i=1}^{\infty} \gamma_i^2$$

On a donc bien montré que $\mu_t \underset{t\to+\infty}{\longrightarrow} \mu_\infty > 0$

Définissons enfin la suite $\tilde{h}_t \coloneqq \mu_t h_t$

Si la condition $\|\nabla_w C(w_t)\|^2 \leq A + B\|w_t - w*\|^2,\ A, B \geq 0$, est respectée, on peut alors déduire de l'inégalité précédemment obtenue

$$\begin{aligned} h_{t+1} - h_t &= -2(w_t - w^*) \cdot \gamma_t \nabla_w C(w) + \gamma_t^2 \|\nabla_w C(w_t)\|^2 \\ &\leq -2(w_t - w^*) \cdot \gamma_t \nabla_w C(w) + \gamma_t^2 (A + B\|w_t - w^*\|^2) \\ &\leq -2(w_t - w^*) \cdot \gamma_t \nabla_w C(w) + \gamma_t^2 A + \gamma_t^2 B h_t \\ &\leq \gamma_t^2 A + \gamma_t^2 B h_t \end{aligned}$$

La dernière inégalité vient de ce que le terme $-2(w_t - w^*) \cdot \gamma_t \nabla_w C(w)$ est toujours négatif en raison de l'hypothèse de convexité générale. Enfin, en passant le terme $\gamma_t^2 B h_t$ à gauche, on va obtenir

$$h_{t+1} - h_t - \gamma_t^2 B h_t = h_{t+1} - (1 + \gamma_t^2 B) h_t \leq \gamma_t^2 A$$

En multipliant, dans cette dernière inégalité, le terme de gauche et celui de droite par $\mu_t$, on a

$$\mu_t h_{t+1} - (1 + \gamma_t^2 B)\mu_t h_t \leq \mu_t \gamma_t^2 A$$

Or, $\mu_t h_t = \tilde{h}_t$ et on a donc également

$$\mu_t h_{t+1} = (1 + \gamma_t^2 B)\mu_{t+1} h_{t+1} = (1 + \gamma_t^2 B)\tilde{h}_{t+1}$$

En reportant dans l'inégalité précédente, on obtient

$$
\begin{aligned}
(1+\gamma_t^2 B)(\tilde{h}_{t+1}-\tilde{h}_t) &\leq \mu_t \gamma_t^2 A \\
\tilde{h}_{t+1}-\tilde{h}_t &\leq \frac{\gamma_t^2 A}{1+\gamma_t^2 B} \\
&\leq \gamma_t^2 A
\end{aligned}
$$

La suite $\tilde{h}_t$ est positive car $h_t$ l'est, et comme $\gamma_t^2 A$ est positif, la somme des variations positives de $\tilde{h}_t$ est au plus égale à $\gamma_t^2 A$. Par le critère suffisant de convergence des suites à termes positifs énoncé plus haut, $\tilde{h}_t$ converge. On a montré par ailleurs que $\mu_t$ converge vers $\mu_\infty > 0$. Comme cette dernière limite est positive, et que $\tilde{h}_t = \mu_t h_t$, on en déduit que $h_t$ converge également.

**Étape 3.** Convergence de l'algorithme de descente de gradient discret du fait de la convergence de la suite de Lyapunov

On sait que $h_t$ converge. Ainsi, la relation précédente entraîne la convergence de la série de terme général $(w_t - w^*) \cdot \gamma_t \nabla_w C(w)$. Surtout, la condition de convexité générale assure que ce terme est toujours positif, on a donc une série de termes tous positifs qui est bornée, donc elle converge.

Pour assurer la convergence vers le minimum, il faut rajouter une condition relative au pas d'apprentissage. Celui-ci ne doit pas décroître trop vite, faute de quoi la convergence ayant lieu trop rapidement, elle ne se fait pas nécessairement vers le minimum. On impose donc que :

$$
\sum_{i=1}^{\infty} \gamma_i = \infty \tag{9}
$$

Cela permet de faire bouger le paramètre à des distances arbitraires, ce qui est crucial notamment quand il se trouve dans une zone éloignée du minimum et où le gradient est constant. En effet, dans pareil cas, si le pas était le terme d'une série convergente, on aurait $c \sum_{i=1}^{\infty} \gamma_t < \infty$, où $c$ représente la valeur du gradient qui est constant dans cette zone. On pourrait alors parcourir au maximum une distance $cR$, où $R$ est la valeur de la série. Il serait donc impossible d'atteindre le minimum dès lors qu'il est situé à une distance plus grande que $cR$ de $w_0$, où $w_0$ est le premier terme de la suite de vecteurs paramètres.

Montrons enfin que $h_t$ converge vers zéro. Raisonnons par contradiction et supposons que $h_t = \|w_t - w^*\|^2$ converge vers une quantité strictement plus grande que zéro. On a alors $\|w_t - w^*\| > 0$, ce qui implique par la condition de convexité générale (5) que $(w_t - w^*) \cdot \gamma_t \nabla_w C(w)$ est strictement supérieur à zéro. C'est absurde puisque l'on a montré plus haut que c'est le terme général d'une série convergente et qu'il tend alors vers zéro. Donc nécessairement, $\|w_t - w^*\|$ tend vers zéro, et donc $h_t$ tend vers zéro, ce qui signifie enfin que :

$$
w_t \underset{t \to +\infty}{\longrightarrow} w^*
$$

Comme l'on sait de plus que la croissance du gradient est linéaire en raison de la majoration (8) introduite précédemment, on est donc sûr que :

$$(w_t - w^*) \cdot \gamma_t \nabla_w C(w) \underset{t \to +\infty}{\longrightarrow} 0$$

## 2.4 Descente de gradient stochastique

La preuve de la convergence de la descente de gradient stochastique suit les trois grandes mêmes étapes que les démonstrations précédentes. En raison du côté aléatoire de l'algorithme, on ne pourra prouver une convergence "sûre", mais une convergence presque sûre.

Chaque itération de l'algorithme de descente de gradient stochastique consiste à piocher au hasard un exemple $z_t$ de notre échantillon, qui est vu comme une réalisation d'une variable aléatoire $Z_t$. Les $Z_t$ sont i.i.d et suivent la loi $P$. Notre algorithme est donc le suivant :

On pioche au hasard un exemple $z_t$ dans notre échantillon et on applique la formule de mise à jour suivante, en partant d'un $w_0$ quelconque dans l'espace des poids.

$$w_{t+1} = w_t - \gamma_t H(z_t, w) \tag{10}$$

où $\gamma_t$ est un nombre positif, et où le terme $H(z, w)$ vérifie

$$H(z,w) = \begin{cases} \nabla_w Q(z,w) & \text{si Q est différentiable en w} \\ 0 & \text{sinon} \end{cases}$$

$$\mathbb{E}_Z\big[H(Z,w)\big] = \nabla_w C(w)$$

Dans ce qui suit, on va donc montrer que les itérés $w_t$ de (10) converge p.s vers $w^*$.

**Etape 1.** Définition d'un processus de Lyapunov

La première étape consiste à définir un processus :

$$h_t := \|w_t - w^*\|^2 \tag{11}$$

La définition de $h$ est similaire à celle présente dans la démonstration de la convergence de la descente de gradient par *batch*, mais ici ce n'est pas une suite déterministe. En effet, au vu de la définition de $w_t$, $h_t$ est une variable aléatoire qui dépend de toutes les réalisations des $Z_t$ précédentes.

**Etape 2.** Convergence du processus

Comme dans le cas du gradient par *batch*, à partir des formules (10) et (11), on peut écrire :

$$h_{t+1} - h_t = -2\gamma_t(w_t - w^*) \cdot H(Z_t, w_t) + \gamma_t^2 \|H(Z_t, w_t)\|^2 \tag{12}$$

La preuve de la convergence pour la descente de gradient discrète repose sur le lemme qui nous dit que pour une suite positive, si la somme des variations positives est bornée, alors la suite converge.

Or, ici l'expression de $h_{t+1} - h_t$ fait intervenir des tirages aléatoires $Z_t$. Utiliser le même lemme dans ce cas reviendrait à montrer que l'algorithme converge pour n'importe quel choix d'exemples. Par exemple, imaginons le cas où on tire le même exemple à chaque itération. Sauf dans des cas très particuliers, ce tirage ne va pas faire converger l'algorithme vers $w^*$.

Prendre l'espérance conditionnelle semble être une bonne solution pour retirer la dépendance à l'aléatoire des tirages passés. On va donc conditionner par toute l'information dont on dispose avant le $t$-ième tirage $Z_t$, c'est-à-dire par :

$$\mathcal{P}_t := Z_0, ..., Z_{t-1}, \quad w_0, ...w_t, \quad \gamma_0, ..., \gamma_t$$

**Quasi-Martingales**

Nous allons présenter dans cette section un théorème très proche du lemme utilisé précédemment, qui va nous permettre de conclure quant à la convergence de $h_t$.

Étant donné toutes les informations passées $\mathcal{P}_t$, on voudrait pouvoir trouver un critère qui nous permettent de distinguer les variations positives des négatives. En effet, pour un proccessus $u_t$, connaissant $\mathcal{P}_t$, on ne peut pas conclure sur le signe de $u_{t+1} - u_t$, car $u_{t+1}$ n'est pas entièrement déterminé par $\mathcal{P}_t$. Un moyen de résoudre ce problème est de considérer l'espérance conditionnelle des variations, en posant :

$$\delta_t := \begin{cases} 1 & \text{si } \mathbb{E}\left[u_{t+1} - u_t | \mathcal{P}_t\right] > 0 \\ 0 & \text{sinon} \end{cases}$$

À partir de cette définition, on peut utiliser le théorème de convergence des quasi-martingales qui nous dit que :

$$\left.\begin{array}{l} \forall t, \quad u_t \geq 0 \quad p.s \\ \displaystyle\sum_{t=1}^{\infty} \mathbb{E}\left[\delta_t(u_{t+1} - u_t)\right] < \infty \end{array}\right\} \implies u_t \xrightarrow[t\to\infty]{p.s} u_\infty \geq 0$$

**Convergence dans le cas convexe**

**Etape 2. (suite)**

En passant à l'espérance sachant $\mathcal{P}_t$ dans l'expression (12) on obtient :

$$\begin{aligned}\mathbb{E}\left[h_{t+1} - h_t \mid \mathcal{P}_t\right] &= -2\gamma_t(w_t - w^*)\mathbb{E}_z\left[H(Z_t, w_t)\right] + \gamma_t^2\mathbb{E}_z\left[\|H(Z_t, w_t)\|^2\right] \\ &= -2\gamma(w_t - w*) \cdot \nabla_w C(w_t) + \gamma_t^2\mathbb{E}_z\left[\|H(Z, w_t)\|^2\right]\end{aligned}$$

La première égalité vient du fait que par définition de $P_t$, $w_t$ peut être vu comme une constante dans l'espérance.

De plus, $Z_t$ est indépendante des $Z_0, ..., Z_{t-1}$, d'où l'égalité par propriétés de l'espérance conditionnelle. La deuxième égalité vient juste de la définition de $H$.

Remarquons que le premier terme de cette somme est négatif par l'hypothèse de convexité générale. Comme pour le gradient discret, on va supposer que le pas de descente décroît assez vite et que le second moment de $H(z, w)$ croît au plus de façon linéaire.

$$\sum_{i=1}^{\infty} \gamma_i^2 < \infty \tag{13}$$

$$\mathbb{E}_z\left[H(z, w)^2\right] \leq A + B\left\|w - w^*\right\|^2 \quad A, B \geq 0 \tag{14}$$

En utilsant ces deux hypothèses (13) et (14) on peut transformer l'équation précédente en :

$$\mathbb{E}\left[h_{t+1} - (1 + \gamma_t^2 B)h_t \mid \mathcal{P}_t\right] \leq -2\gamma_t(w_t - w^*) \cdot \nabla_w C(w) + \gamma_t^2 A \tag{15}$$

On définit les mêmes suites $\mu_t$ *et* $h_t^{'}$ que pour le gradient discret :

$$\mu_t \coloneqq \prod_{i=1}^{t-1} \frac{1}{1 + \gamma_i^2 B} \quad et \quad h'_t = \mu_t h_t$$

De façon similaire à la preuve précédente, en multipliant (15) à droite et à gauche par $\mu_t$, on a :

$$\mathbb{E}\left[h'_{t+1} - h'_t \mid \mathcal{P}_t\right] \leq \gamma_t^2 \mu_t A$$

Puisque $\delta_t$ est par définition une variable aléatoire $P_t - mesurable$, on a :

$$\mathbb{E}\left[\delta_t(h'_{t+1} - h'_t)\right] = \mathbb{E}\left[\delta_t\, \mathbb{E}(h'_{t+1} - h'_t \mid P_t)\right] \leq \gamma_t^2 \mu_t A$$

Or, $\sum_{t=1}^{\infty} \gamma_t^2 \mu_t A < +\infty$, d'où $\sum_{t=1}^{\infty} \mathbb{E}\left[\delta_t(h'_{t+1} - h'_t)\right] < +\infty$ , et donc par le théorème de convergence des quasi-martingales, on a que $h'_t$ converge presque sûrement. Or, comme on l'a montré dans la partie sur la descente de gradient discrète, la suite $\mu_t$ converge vers $\mu_\infty > 0$. On a donc que $h_t$ converge presque sûrement vers un $h_\infty \geq 0$.

**Etape 3.**

Maintenant on va utiliser le fait que $h_t$ converge pour montrer la convergence de notre algorithme. Comme la suite $h_t$ converge p.s, (15) implique que :

$$\sum_{i=1}^{\infty} \gamma_i (w_i - w^*) \cdot \nabla_w C(w_i) < +\infty \quad \text{p.s} \tag{16}$$

On va introduire l'hypothèse suivante pour limiter la décroissance de $\gamma_t$, qui joue le même rôle que (9) et permet d'être sûr que ça soit bien $(w_t - w^*) \cdot \nabla_w C(w_t)$ qui entraîne la convergence de la somme. On suppose donc que

$$\sum_{i=1}^{\infty} \gamma_i = +\infty \tag{17}$$

On a donc $(w_t - w^*) \cdot \nabla_w C(w_t) \xrightarrow[t\to\infty]{p.s} 0$. Supposons que $h_t$ converge $p.s$ vers une quantité plus grande que 0 qu'on note l. On a donc $\|w_t - w^*\| \geq \sqrt{l}$ $p.s$. On rentre donc dans le cadre de notre hypothèse de convexité générale et donc que $(w_t - w^*) \cdot \nabla_w C(w_t) > c$ $p.s$ où $c > 0$.
Cela impliquerai donc que $\sum_{i=1}^{\infty} \gamma_i (w_i - w^*) \cdot \nabla_w C(w_i)$ diverge presque sûrement, on a donc une contradiction avec (16). D'où $h_t$ converge vers zéro $p.s$ et on a bien :

$$w_t \xrightarrow[t\to\infty]{p.s} w^*$$

# 3 Vitesses de convergence et complexités algorithmiques

## 3.1 Vitesse de convergence de l'algorithme de gradient par *batch*

Comme pour les descentes de gradient plus classique on peut, sous de bonnes hypothèses, montrer que celle par *batch* converge à vitesse linéaire. De plus, en utilisant une matrice de préconditionnement $\Phi_t$, on peut grandement augmenter la vitesse de convergence de notre algorithme. On peut par exemple prendre $\Phi_t$ une matrice symétrique définie positive qui approxime l'inverse de la hessienne de la fonction de coût.

$$\phi_t \approx H^{-1}(w_t), \quad H(w) = \nabla_w^2 C(w)$$

Cela nous mène à des algorithmes d'optimisation très efficaces, comme l'algorithme de Newton ou en encore l'algorithme du gradient conjugué. Ces algorithmes ont une vitesse de convergence superlinéaire voir quadratique sous de bonnes hypothèses.

## 3.2 Vitesse de convergence pour les algorithmes stochastiques

Tandis que les algorithmes *online* convergent vers la zone générale de l'optimum au moins aussi rapidement que les algorithmes par *batch*, ils semblent ralentir durant la phase finale de convergence[6]. L'estimation bruitée du gradient implique que le vecteur des poids fluctue autour de l'optimum dans un bol dont la taille va dépendre de notre pas d'apprentissage. On peut montrer que la taille du bol décroît au plus en $\frac{1}{t}$. En effet, pour assurer la convergence, on a du supposer que le pas d'apprentissage ne décroît pas trop vite, ce qui est garanti par la condition :

$$\sum_{i=1}^{\infty} \gamma_i = +\infty$$

La descente de gradient stochastique bénéficie quand même de l'utilisation de méthodes du second ordre. En effet, on peut multiplier le gradient en utilisant une matrice symétrique postive $\Phi_t$ qui va approximer l'inverse de la matrice hessienne d'une manière analogue à l'algorithme de Newton. Si les valeurs propres de la matrice $\Phi_t$ sont bornées, alors on peut appliquer les mêmes résultats de convergence qu'on a prouvés précédemment à l'algorithme suivant :

$$w_{t+1} = w_t - \frac{1}{t}\Phi_t \nabla_w Q(z_t, w_t) \tag{18}$$

Pour simplifier dans la suite, on va fixer $\gamma_t = \frac{1}{t}$ qui satisfait bien les conditions (17) et (13). Bien qu'on puisse améliorer la vitesse de convergence avec un préconditionnement, il faut comprendre que les méthodes du second ordre souffrent aussi du bruit qui vient du choix aléatoire des exemples $z_t$. La vitesse de convergence dépend toujours du choix du pas d'apprentissage, elle est donc contrainte encore une fois par la condition (17). C'est une contrainte assez forte car dans le cas de la descente de gradient par *batch*, la même matrice de précondtionnement $\Phi_t$ nous donne une convergence superlinéaire.

À partir de ce qui vient d'être dit au-dessus, on pourrait penser que la descente de gradient stochastique est un mauvais algorithme d'optimisation, et donc un mauvais algorithme d'apprentissage. Pourtant, l'expérience suggère le contraire.

## 3.3 Optimisation *vs* Apprentissage

On a pu voir que notre descente de gradient stochastique converge au plus à vitesse $\frac{1}{t}$. On peut donc se dire que c'est un mauvais algorithme d'un point

de vue de l'optimisation. Mais on peut remarquer que :

- L'algorithme de la descente de gradient par *batch* converge vers un minimum du risque empirique $\hat{C}_L(w)$ qui est défini comme une moyenne des valeurs prises par la fonction de perte sur L exemples de notre échantillon.

- L'algorithme de descente de gradient stochastique, lui, converge vers le minimum du risque d'espérance $C(w)$, qui est défini comme l'espérance de la fonction de perte.

Dans un problème d'apprentissage, c'est intéressant de connaître la vitesse de convergence d'un algorithme vers le minimum du risque d'espérance $C(w)$, car cela donne une idée de l'erreur de généralisation. Mais ici, l'algorithme par *batch* approxime le minimum du risque empirique $\hat{C}_L(w)$, qui est lui-même une approximation du risque d'espérance $C(w)$. On va voir que cette approximation nullifie le fait que la descente de gradient classique puisse converger plus vite vers le minimum que la descente stochastique.

### 3.4 Vitesse de convergence au sens du risque quadratique vers la vraie solution $w^*$

Dans le papier [4] qu'on a étudié, l'auteur a décidé de s'intéresser au risque quadratique pour mesurer la vitesse de convergence des deux algorithmes vers $w^*$ le minimum du risque d'espérance $C$. Dans la suite, on va considérer une suite infinie d'exemples d'entraînement indépendants $(z_1, ..., z_L, ...)$. On pose $w_t^*$ le minimum du risque empirique $\hat{C}_t$ définie sur les $t$ premiers exemples. Un développement de Taylor de $\nabla_w \hat{C}_{L+1}$ au voisinage de $w_t^*$ nous donne :

$$w_{L+1}^* = w_L^* - \frac{1}{t+1}\psi_t \nabla_w Q(z_L, w_L^*) + O\Big(\frac{1}{L^2}\Big) \tag{19}$$

avec

$$\Psi_L \coloneqq \left(\frac{1}{L+1}\sum_{i=1}^{L+1} \nabla_w^2 Q(z_i, w_L^*)\right)^{-1} \underset{t\to\infty}{\longrightarrow} H^{-1}(w_L^*)$$

Les similitudes entre la formule (18) et (19) suggère que les suites $(w_L^*)$ et $(w_t)$ converge à la même vitesse en faisant le bon choix pour la matrice de préconditionnement $\phi_t$. Les analyses théoriques [8] montrent que :

$$\begin{aligned} \mathbb{E}\left[\|w_L^* - w^*\|^2\right] &= \frac{K}{L} + o\Big(\frac{1}{L}\Big) \\ \Phi_t \underset{t\to\infty}{\longrightarrow} H^{-1}(w^*) \implies \mathbb{E}\left[\|w_t - w^*\|^2\right] &= \frac{K}{t} + o\Big(\frac{1}{t}\Big) \end{aligned} \tag{20}$$

où

$$K = trace(H^{-1}(w^*) \cdot \mathbb{E}_Z\Big[(\nabla_w Q(Z, w^*))(\nabla_w Q(Z, w^*))^T\Big] \cdot H^{-1}(w^*))$$

Ce résultat nous dit dans un premier temps que les deux suites convergent vers $w^*$ au sens du risque quadratique en $O\Big(\frac{1}{t}\Big)$, et de plus asymptotiquement on connaît la distance (au sens $L^2$) entre $w^*$ et $w_L^*$(resp. $w_t$) qui est de $\frac{K}{L}$ (resp.$\frac{K}{t}$).

Ce qu'on peut conclure de ces remarques, c'est qu'après $t$ itérations sur de nouveaux exemples, le point $w_t$ atteint par l'aglorithme de gradient stochastique est aussi bon asympotiquement que la solution $w_t^*$ de l'algorithme de descente de gradient par *batch* sur ces mêmes $t$ exemples.

## 3.5 Comparaison des temps de calcul entre l'algorithme de descente de gradient par *batch* et celui stochastique

On a pour l'instant montré que l'algorithme de descente de gradient stochastique fait aussi bien que n'importe quel algorithme de descente par *batch* pour un même nombre d'exemples donné. On va montrer désormais que, à ressources de calcul égales, l'algorithme stochastique peut traiter asymptotiquement plus d'exemples que celui par *batch*.

Chaque itération de l'algorithme par *batch* sur $N$ exemples d'entraînement requière un temps $K_1N + K2$. Les constantes $K1$ et $K2$ représentent respectivement le temps requis pour traîter chaque exemple, et pour adapter les paramètres. Le résultat (20) fournit l'équivalent asymptotique suivant :

$$\mathbb{E}\left[\|w_N^* - w^*\|^2\right] \sim \frac{1}{N}$$

L'algorithme de descente par *batch* doit réaliser suffisamment d'itérations pour approcher l'optimum empirique $w_N^*$ avec au moins la même précision. Un algorithme très performant, avec une convergence quadratique, y arrive après un nombre d'itérations asymptotiquement proportionnel à $loglogN$.

Un algorithme stochastique requière un temps constant $K3$ par exemple traité et mise à jour des paramètres qui s'en suit à chaque fois. Appelons $T$ le nombre d'exemples que l'algorithme stochastique peut ainsi passer en revue en utilisant les mêmes ressources de calcul que l'algorithme par *batch*. On a alors :

$$K_3T \sim (K_1N + K_2)\log\log N \implies T \sim \log\log N$$

Le paramètre $w_T$ de l'algorithme stochastique converge aussi d'après (30). Les développements précédents montrent que l'algorithme stochastique offre une meilleure solution d'un facteur $\sim \log\log N$.

Ce facteur correspond au nombre d'itérations requises par l'algorithme de descente classique pour converger vers l'optimum empirique. Ce nombre augmente lentement en fonction de la précision requise.

En pratique, ce facteur est bien moins significatif que la valeur des constantes $K_1$, $K_2$ et $K_3$ [4]. L'expérience montre en outre que les algorithmes stochastiques sont bien plus faciles à implémenter. En effet, chaque itération d'une descente de gradient par *batch* implique une sommation considérable de tous les exemples à disposition. Cela nécessite d'allouer une part très importante de mémoire. Ces opérations font qu'un algorithme par *batch* sera plus gourmand en ressources qu'un algorithme stochastique. Pour ce dernier, chaque itération n'a besoin que d'un exemple choisi aléatoirement, qu'il n'est pas nécessaire de garder en mémoire ensuite (dans la mémoire vive du moins).

**Pas d'apprentissage optimal pour $K$-Means**

On veut ici donner un exemple de méthode d'ordre deux, très efficace pour accélérer la convergence de l'algorithme, et qui sera mise en oeuvre dans le cadre de l'implémentation informatique.

La dérivée seconde peut être utilisée pour déterminer un pas d'apprentissage très performant pour l'algorithme $K$-Means. L'analyse de la fonction de perte montre que la Hessienne de la fonction de coût est une matrice diagonale dont les coefficients $\lambda(k)$ sont égaux aux probabilités qu'une exemple $x$ soit associé avec le centroïde correspondant $w(k)$.

Ces probabilités peuvent être estimées simplement en comptant combien d'exemples $n(k)$ ont été associés avec chaque centroïde $w(k)$. Chaque itération de l'algorithme stochastique correspondant consiste en le choix d'un exemple $x_t$ pris au hasard, trouver le centroïde le plus proche $w(k)$, et mettre à jour le nombre $n(k)$ et le centroïde $w(k)$ selon l'équation suivante :

$$\left[ \begin{array}{ll} n_{t+1}(k) & = n_t(k) + 1 \\ w_{t+1}(k) & = w_t(k) + \frac{1}{n_{t+1}(k)}(x_t - w_t(k)) \end{array} \right.$$

On a alors une localisation très rapide de la position relative des différents *clusters* dans les données. La convergence finale est ralentit du fait du bruit introduit par le choix aléatoire des exemples. Il existe des preuves expérimentales que la meilleure vitesse est atteinte en utilisant d'abord l'équation ci-dessus et en passant ensuite à un version par *batch* et superlinéaire de $K$-Means.

# 4 Résultats expérimentaux

Les expérimentations ont consisté principalement à illustrer la convergence de la descente de gradient stochastique et comparer sa vitesse avec celle de la

méthode de descente classique. La mise en oeuvre de ces méthodes s'est faite à travers deux algorithmes.

- Un algorithme de *clustering* : $K$-means
- L'algorithme du perceptron, utilisé pour faire de la classification et de la régression, le code ayant été légèrement adapté pour réaliser cette dernière tâche.

### Algorithme du perceptron

#### Cas de la classification

Dans le cas de la classification, l'algorithme du perceptron fonctionne de la façon suivante : on applique un produit scalaire entre les *features* ou caractéristiques de l'exemple $t$ à traiter qu'on appelle $x_t$. On le représente par un vecteur, tout comme le vecteur $w_t$ des paramètres, dont les composantes sont les caractéristiques en question. Une fonction de seuil est appliquée sur le résultat du produit scalaire, dont la valeur déterminera une sortie qui sera 1 ou $-1$, qu'on peut appeler $\hat{y_t}$. On a donc deux valeurs possibles en sortie de l'algorithme, qu'on fait correspondre à deux labels distincts, ce qui permet donc de classer les exemples traités. On désire que la classe d'un exemple prédite, $\hat{y_t}$, corresponde à la véritable classe de celui-ci, qu'on nomme $y_t$.

On opte pour la règle de mise à jour suivante des poids :

$$w_{t+1} = w_t + 2\gamma_t y_t x_t$$

On peut alors regarder cet algorithme comme une descente de gradient stochastique appliquée à la fonction de coût qui suit :

$$Q_{perceptron}(x, w) = (signe(w \cdot x) - y)w \cdot x$$

Cette fonction de perte n'est pas différentiable quand $w \cdot x$ est nul, mais elle remplit les conditions (3) et (4). Elle est bien différentiable partout ailleurs, et il suffit de dénombrer les exemples $x$ dont le produit scalaire avec $w$ est nul, et de les écarter. On peut donc ignorer sans souci les quelques points supposément problématiques. C'est d'ailleurs pourquoi on a pu faire tourner avec succès cet algorithme et converger vers un minimum.

Notre prédiction et la classe que l'on souhaite prédire appartiennent toutes deux à $\{-1, 1\}$, si bien que lorsque la classe d'un exemple est correctement prédite, on a $\hat{(y)} - y = 0$, et le terme de mise à jour de l'algorithme est nul, donc $w_t$ ne change pas.

La fonction de perte du perceptron vaut zéro quand $x_t$ est bien classé, autrement sa valeur est positive et proportionnelle au produit scalaire $w \cdot x$. La fonction de coût atteint son minimum, zéro, quand tous les exemples sont bien

classés. Il existe des résultats théoriques qui assurent la convergence de l'algorithme vers le minimum global avec probabilité 1. Il faut que le problème de classification soit linéaire, i.e séparable par un hyperplan. On est alors assuré que pour un pas suffisamment petit, l'algorithme finira par classer tous les exemples correctement au bout d'un certain temps (un certain nombre d'itérations).

On n'a pas reproduit dans le code toutes les expérimentations que l'on a pu faire, mais on a bien observé qu'en prenant un pas petit, dès lors qu'on avait crée un *dataset* linéairement séparable, on obtenait à chaque fois la convergence. On a même observé, conformément aux affirmations de Léon Bottou, que l'on se dirigeait rapidement vers la zone des minima, mais que la phase de convergence finale pouvait prendre du temps en raison du comportement stochastique de l'algorithme. On remarque que l'on oscille autour du minimum.

Dans le code, l'implémentation est très légèrement différente d'une traduction littérale de cet algorithme, au sens où on l'a modifié de manière à avoir des labels dans $\{0, 1\}$, mais c'est totalement équivalent.

On a testé l'algorithme sur un *dataset* pouvant être linéairement séparable au sens décrit plus haut, et sur un autre qui ne l'était pas.

#### Cas de la régression

Même si l'algorithme du perceptron présenté ci-dessus peut permettre de faire de la régression, il était plus intéressant de le modifier, notamment en changeant la fonction de coût à optimiser, plus adaptée à ce type de tâche.

La fonction de coût à minimiser est alors celle utilisée dans la méthode de regression dite des moindres carrés. C'est aussi celle utilisée dans l'algorithme *Adaline* présenté par Léon Bottou :

$$Q_{Adaline} = (y - w \cdot x)^2$$

La différence avec l'algorithme d'*Adaline* est toutefois qu'ici, $y$ est un réel et pas nécessairement un entier naturel représentant la classe d'un exemple que l'on cherche à prédire. Dans la regression on cherche à approcher au mieux, pour chaque exemple $x_t$, la valeur réelle $y_t$ qui lui est associée. On adapte les poids $w_t$ afin que la valeur $w_t \cdot x_t$ soit aussi proche que possible de $y_t$. L'utilisation du produit scalaire se justifie car on cherche un hyperplan approchant la relation visiblement linéaire qui existe entre les caractéristiques des données et la valeur qu'on cherche à prédire.

L'algorithme du gradient consiste alors à mettre ainsi les poids à jour :

$$w_{t+1} = w_t - \frac{1}{2}(y - w \cdot x)^2$$

C'est une fonction convexe partout différentiable, qui a son minimum global égal à zéro. Elle se prête donc bien à l'optimisation via la méthode de descente de gradient.

Remarquons que, comme pour la règle de mise à jour du perceptron, on est bien dans le cadre de l'optimisation stochastique. On cherche à minimiser une fonction de coût qui servira à évaluer le bon apprentissage ou non de notre modèle, et pour ce faire on met à jour le paramètre $w_t$ en fonction du gradient de la fonction de coût qui nous indique comment on doit modifier les poids.

Ces algorithmes ont été d'abord codés dans une optique stochastique, puis de simples modifications, qu'on peut globalement résumer à faire des moyennisation sur tous les exemples là où c'est nécessaire, ont été apportées afin de pouvoir utiliser une version par *batch* de l'algorithme de descente de gradient.

Il a alors été possible de confirmer les principales assertions faites par Léon Bottou à propos de ces algorithmes. [3]

Si l'algorithme par *batch* converge plus rapidement en termes de nombre d'itérations, à nombre d'exemples égal, celui stochastique se rapproche bien plus rapidement du minimum. Cela est vérifié aussi bien sur des tâches de classification, que le problème soit linéairement séparable ou non, que sur des tâches de régression.

**Algorithme du $K$-means**

On a utilisé l'algorithme de gradient stochastique et celui par *batch* dans le cadre de l'algorithme de *clustering* $K$-means. Le principe de cet algorithme, c'est de ranger nos données en $K$ groupes. Pour cela on va utiliser $K$ points qu'on va appeler des centroïdes, et on va regarder pour chaque point de quels centroïdes il est le plus proche. Par exemple, si le point $x$ est plus proche du centroïde numéro 3, alors je le range dans le groupe 3. On va déplacer les centroïdes à chaque étape de notre algorithme dans le but de minimiser la fonction :

$$J(x_1, ..., x_n) = \sum_{x_1,...,x_n} \frac{1}{2} \min_{j \in 1,...,K} \|x_i - \mu_j\|^2$$

où $x_1, ..., x_n$ sont nos données et $\mu_1, ..., \mu_K$ sont nos centroïdes.

Avec les notations précédentes on a donc :

$$Q_{Kmeans}(x, \mu) = \frac{1}{2} \min_{j \in 1,...,K} \|x_i - \mu_j\|^2$$

où $\mu = (\mu_1, ..., \mu_K)$.

On a finalement :

$$H(x, \mu) = \begin{cases} \nabla_\mu Q_{Kmeans}(x, \mu) = -(x - \mu_j) & \text{si } Q_{Kmeans} \text{ est différentiable} \\ 0 & \text{sinon} \end{cases}$$

où $j = argmin_{i \in 1,...,K} \|x_t - \mu_i\|^2$.

On a d'abord créé un jeu de données factice en créant des nuages de points en dimension 2. On a fait en sorte qu'on puisse distinguer 3 groupes, mais l'algorithme peut être adapté pour $K$ groupes. Pour les centroïdes on tire 3 points au hasard parmis les données. On a imposé une condition pour qu'ils ne soient trop proche sinon cela peut entraîner des problèmes de convergence.

Dans un premier temps on a codé l'algorithme de gradient stochastique qui fait les opérations suivante (on se place à l'étape t) :

1. D'abord on choisit un exemple $x_t$ au hasard dans notre jeu de données.
2. Ensuite on évalue de quels centroïdes il est le plus proche, dans le cas où il est à une même distance de deux centroïdes, on le passe.
3. Une fois qu'on a identifié le centroïde le plus proche on fait l'update suivante :
$$\mu_j = \mu_j + \gamma_t H(x_t, \mu_j) = \mu_j + \gamma_t (x_t - \mu_j)$$
où $\gamma_t$ est notre taux d'apprentissage à l'étape t et $j = argmin_{i \in 1,...,K} \|x_t - \mu_i\|^2$.

Pour le batch gradient on fait :
-Pour les points de $x_1, ..., x_n$. On évalue en chaque point quel est le centroïde le plus proche et on fait l'update suivante :

$$\mu_j = \mu_j + \frac{\gamma_t}{n} \sum_{x \in C_j} H(x, \mu) = \mu_j + \frac{\gamma_t}{n} \sum_{x \in C_j} (x - \mu_j) \quad j = 1, ..., K$$

où $C_j = \left\{ x \in (x_1, ..., x_n) | j = argmin_{i \in 1,...,K} \|x - \mu_i\|^2 \right\}$.
On fait cette update pour chaque centroïde.

On a utilisé ces algorithmes sur un jeu de n données où n = 10,100,500,1500. On fait 1000 itérations pour chaque algorithme. En effet après plusieurs essais on a remarqué que cela ne servait à rien de faire plus car on atteignait le minimum avant.
On observe bien ce que la théorie nous a montré, c'est-à-dire, l'algorithme de gradient stochastique va converger en plus d'itérations vers le minimum que le batch gradient.
On observe aussi que quand n devient assez grand les deux minimises de la même façon notre fonction J. De même, plus on utilise d'exemples pour le batch gradient plus on minimise la fonction J, ce qui encore une fois est cohérent avec la théorie, même si on atteint assez vite des résultats similaire au gradient stochastique avec 500 exemples. On remarqué aussi que moins on utilisait d'exemples pour le batch gradient, plus il fallait mettre un pas d'apprentissage petit afin d'éviter certains problème de convergence, et de ce fait le batch gradient devenait moins rapide que le gradient stochastique.

On aurait voulu comparer le temps d'exécution, enfin de voir si le batch gradient et plus long que le gradient stochastique comme on le pense, mais on le problème vient du fait qu'on ne sait pas si on a codé chaque algorithme de la manière la plus optimisé qui soit. Cela n'est donc pas pertinent d'évaluer ça dans le sens où c'est peut-être juste l'implémentation qu'on en a fait qui fait qu'un algorithme est plus lent que l'autre en temps d'exécution.

**Conclusions sur les expérimentations**

Pour un même nombre d'itérations, l'algorithme par *batch* converge plus rapidement. En revanche, chaque itération de celui-ci implique qu'il traite tous les exemples du *batch*, qui ici correspond à tout le *dataset*, avant de mettre à jour les paramètres.

Si l'on raisonne donc en termes d'exemples traités cette fois-ci, l'algorithme stochastique converge bien plus vite pour un même nombre d'exemples donnés que celui par *batch*. C'est exactement ce qui est dit dans les travaux de Léon Bottou. La raison en est que l'algorithme stochastique se met à jour bien plus fréquemment, puisqu'il adapte ses paramètres après chacune de ses itérations, c'est-à-dire après chaque exemple traité.

Les algorithmes de $K$-means et du perceptron de Rosenblatt (celui pour la classification) ont pu être mis en oeuvre avec succès, alors même que leurs fonctions de coût ne sont pas différentiables partout. Il y a par exemple un problème en zéro pour celle du perceptron. C'est donc la confirmation que des fonctions de coûts non différentiables sur un ensemble de mesure nulle et qui remplissent les conditions (3) et (4) peuvent être utilisées correctement pour faire de la descente de gradient.

Un résultat qui mérite une attention particulière est celui concernant le pas adaptatif, notamment dans le cas de la régression. Si l'on note que l'on se dirige vers le minimum, ce qui est cohérent avec le fait que théoriquement ce pas adaptatif garantit la convergence, cela n'empêche pas qu'elle a également lieu ici avec le pas constant. Surtout, en pratique, le pas devient vite tellement petit que, si d'un point de vue théorique on est assuré de la convergence, ici on s'arrête en chemin en quelque sorte. Le pas décroît vite au bout de plusieurs itérations et on ne modifie plus ou quasiment plus les paramètres. C'est la raison pour laquelle dans les faits, le pas constant est préférable. Il entraîne aussi la convergence et surtout, il atteint concrètement le minimum alors que le pas adaptatif empêche de se rapprocher autant que l'on veut du minimum.

Même si théoriquement on est bien avec un pas qui décroît en $\frac{1}{t}$, pratiquement c'est comme si l'on avait une suite de pas dont la somme est finie. On a d'ailleurs pu vérifier qu'avec un pas qui décroît trop vite et ne respecte pas les exigences de la théorie de l'approximation stochastique (s'il décroît en $\frac{1}{t^2}$ par exemple), alors il est possible de ne jamais converger vers un minimum.

Pour conclure cette partie, il nous semble crucial d'attirer l'attention du lecteur sur la mise en oeuvre concrète de ces algorithmes. Il y a en effet un aller-retour entre théorie et pratique, sûrement particulier aux mathématiques appliquées, qu'il est vital de bien saisir.

Les résultats théoriques nous disent qu'en nombre d'itérations, l'algorithme de descente de gradient par *batch* converge plus rapidement.

Néanmoins, il minimise le risque empirique, alors qu'en pratique, on se soucie du risque réel, autrement dit de la capacité à généraliser, qui est le critère d'un apprentissage réussie.

Revenant à la théorie, il est possible de montrer que l'algorithme stochastique ne converge pas moins vite que celui par *batch* vers ce risque, ils y vont asymptotiquement à la même vitesse. On peut alors montrer que pour un même nombre d'exemples passés en revue par les deux algorithmes, celui stochastique sera asymptotiquement meilleur, raison pour laquelle d'ailleurs on l'utilise sur de grands jeux de données où la limite est moins le nombre d'exemples du dataset que le temps de calcul.

Mais rappelons enfin, comme le mentionne Léon Bottou [4], qu'en pratique cette complexité asymptotique nous intéresse moins que les constantes déterminant le temps de calcul des opérations que doit effectuer chaque algorithme. On en a ici une illustration parfaite.

Regardant beaucoup plus d'exemples que dans le cas du gradient stochastique, on pourrait penser que l'algorithme par *batch* serait beaucoup plus lent à s'éxécuter. Or, il n'en est rien. Après quelques recherches, il apparaît que les explications résident très probablement dans l'implémentation des algorithmes.

Si on a utilisé python pour chacun d'eux, il est possible grâce à Numpy de vectoriser les opérations de calculs de gradients, de somme et de mise à jour des poids. Or, cette vectorisation tire profit de l'architecture de l'ordinateur et accélère énormément les calculs. En outre, l'implémentation Numpy recourt à des langages beaucoup plus rapides, comme le C.

Ainsi, si en théorie, comme il a besoin de plus d'exemples, l'algorithme par *batch* devrait s'éxécuter plus lentement, ces considérations pratiques font qu'il n'en est rien et que l'on semble même observer l'inverse. C'est d'ailleurs la raison pour laquelle on a parfois recourt à la méthode dite de *mini-batch*, qui constitue un mix des deux algorithmes pour tirer profit des avantages de chacune d'elle.

Ces aspects concrets sont à garder en tête quand on désire implémenter un algorithme dont on espère qu'il converge pour un certain temps de calcul donné et incite à toujours garder un peu de recul quand il s'agit de mettre en oeuvre les résultats théoriques.

# Conclusion

On a vu les conditions sous lesquelles l'algorithme de descente de gradient stochastique converge vers le minimum de la fonction de coût, à une vitesse au moins aussi bonne voire meilleure que l'algorithme de descente de gradient classique.

Actuellement, l'explosion des données et la constitution de datasets toujours plus grands justifient l'utilisation d'algorithmes stochatiques [5], qui sont bien meilleurs que les algorithmes par *batch* pour l'apprentissage en grande dimension. Sur de larges jeu de données, les algorithmes par nécessitent une puissance de calcul, et donc un temps d'apprentissage, bien trop importants pour être utilisés en pratique. Il est donc naturel de se tourner vers des techniques de machine learning plus faciles à implémenter et moins gourmandes en calcul. C'est particulièrement le cas des algorithmes stochastiques.

La production de data est entre autres favorisée par le développement des objets connectés, qui génèrent un flux de données quasi continu. Ces derniers participent de l'essor des applications dites en temps réel, qui justement fonctionnent et se mettent à jour continuellement à partir de ce flux.

La masse de données à traiter et la nécessité d'évoluer en fonction des nouvelles informations qui arrivent en temps réel font que les algorithmes stochastiques sont des méthodes d'apprentissage privilégiées. S'ils ont très vite vu le jour avec les débuts de l'informatique en raison de la simplicité de leur implémentation, il est certain qu'ils sont encore promis à un bel avenir.

# Références


[1] Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2.

[2] Léon Bottou. *Une Approche théorique de l'Apprentissage Connexionniste : Applications à la Reconnaissance de la Parole.* PhD thesis, Université de Paris XI, Orsay, France, 1991.

[3] Léon Bottou. On-line learning and stochastic approximations. In *In On-line Learning in Neural Networks*, pages 9–42. Cambridge University Press, 1998.

[4] Léon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004.

[5] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[6] Léon Bottou. Stochastic gradient descent tricks. In *Montavon, G., Orr, G.B., Müller, KR. (eds) Neural Networks : Tricks of the Trade. Lecture Notes in Computer Science, vol 7700*, pages 421—436. Springer, 2012.

[7] Léon Bottou et Noburu Murata. Stochastic approximations and efficient learning. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, Second edition,*. The MIT Press, Cambridge, MA, 2002.

[8] Léon and et Yann Lecun. Large scale online learning. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16 (NIPS 2003)*. MIT Press, Cambridge, MA, 2004.