Outils d'optimisation pour les sciences des données et de la décision

Clément W. Royer

Notes de cours - M2 MIAGE - 2024/2025

- La version la plus récente de ces notes se trouve à l'adresse : **https://www.lamsade.dauphine.fr/∼croyer/ensdocs/ODD/PolyODD.pdf**.
- Pour tout commentaire, merci d'envoyer un mail à `clement.royer@lamsade.dauphine.fr`.
Merci à Sébastien Kerleau pour sa relecture attentive.
- **Historique du document**
  - 2024.10.28 : Suppression du dernier chapitre, correction de typos.
  - 2024.09.20 : Changement de notation.
  - 2024.09.17 : Première version des notes.
- **Objectifs d'apprentissage**
  - Comprendre les caractéristiques des modèles d'optimisation les plus classiques.
  - Connaître les grands principes algorithmiques derrière la résolution de problèmes d'optimisation non linéaire.
  - Appliquer ces principes dans le contexte des sciences des données et de la décision.

Sommaire

1 Introduction à l'optimisation **6**
1.1 Définition et contexte . . . 6
1.1.1 Problème et processus d'optimisation . . . 6
1.2 Logiciels et optimisation . . . 7
1.3 Bases de l'optimisation mathématique . . . 8
1.3.1 Formulation générale et premières définitions . . . 8
1.3.2 Solutions et minima . . . 10
1.3.3 Conditions d'optimalité . . . 10
1.4 Convexité . . . 11
1.4.1 Cas des fonctions convexes . . . 12
1.4.2 Cas des fonctions fortement convexes . . . 13
1.5 Conclusion du chapitre 1 . . . 13

2 Outils d'optimisation sans contraintes **14**
2.1 Descente de gradient . . . 14
2.1.1 Algorithme . . . 14
2.1.2 Choix de la taille de pas . . . 16
2.1.3 Bases théoriques de la descente de gradient . . . 17
2.1.4 Cas convexe et fortement convexe . . . 19
2.2 Optimisation sans gradients . . . 20
2.2.1 Des fonctions aux problèmes non lisses . . . 20
2.2.2 Méthodes de sous-gradient . . . 21
2.3 Accélération . . . 22
2.3.1 Introduction au concept de momentum . . . 22
2.3.2 Méthode du gradient accéléré . . . 22
2.3.3 Méthode Heavy-ball . . . 24
2.4 Conclusion . . . 24

3 Optimisation pour les sciences des données **25**
3.1 Régularisation . . . 25
3.1.1 Problèmes régularisés et premier exemple . . . 25
3.1.2 Régularisation et parcimonie . . . 26
3.1.3 Méthodes proximales . . . 26
3.2 Méthode du gradient stochastique . . . 29
3.2.1 Motivation . . . 29

3.2.2 Algorithme 30
3.2.3 Élements d'analyse 31
3.2.4 Variantes par fournées (batch) 34
3.2.5 Autres variantes basées sur la réduction de variance 35
3.3 Méthodes de gradient stochastique pour l'apprentissage profond 35
3.3.1 Gradient stochastique avec momentum 36
3.3.2 AdaGrad 36
3.3.3 RMSProp 37
3.3.4 Adam 38
3.4 Conclusion 39

Annexe A Bases mathématiques **41**
A.1 Éléments d'algèbre linéaire 41
A.1.1 Algèbre linéaire vectorielle 41
A.1.2 Algèbre linéaire matricielle 43

Avant-propos

Le but de ce cours est de présenter les algorithmes d'optimisation les plus utilisés en sciences des données et de la décision, ainsi que leur illustration sur des problèmes classiques. Il se veut découpé en plusieurs parties suivant le déroulement des séances de cours.

Ce cours n'est pas un cours de mathématiques, mais il repose sur plusieurs concepts élémentaires d'algèbre linéaire et de calcul différentiel. Ceux-ci ne seront pas rappelés en détail en cours, mais sont en revanche détaillés en annexe de ce polycopié. De même, les notations cruciales du polycopié sont décrites ci-après.

Notations

- Les scalaires seront représentés par des lettres minuscules : $a, b, c, \alpha, \beta, \gamma$.
- Les vecteurs seront représentés par des lettres minuscules **en gras** : $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$.
- Les lettres majuscules en gras seront utilisées pour les matrices : $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$.
- Les lettres majuscules cursives seront utilisées pour les ensembles : $\mathcal{A}, \mathcal{B}, \mathcal{C}$.
- La définition d'une nouvelle quantité ou d'un nouvel opérateur sera indiquée par $:=$.
- L'ensemble des entiers naturels sera noté $\mathbb{N}$, l'ensemble des entiers relatifs sera noté $\mathbb{Z}$.
- L'ensemble des réels sera noté $\mathbb{R}$. L'ensemble des réels positifs sera noté $\mathbb{R}_+$ et l'ensemble des réels strictement positifs sera noté $\mathbb{R}_{++}$.
- On notera $\mathbb{R}^n$ l'ensemble des vecteurs à n composantes réelles, et on considèrera toujours que n est un entier supérieur ou égal à 1.
- Un vecteur $\boldsymbol{x} \in \mathbb{R}^n$ sera pensé (par convention) comme un vecteur colonne. On notera $x_i \in \mathbb{R}$ sa i-ème coordonnée dans la base canonique de $\mathbb{R}^n$. On aura ainsi $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$. Étant donné un vecteur (colonne) $\boldsymbol{x} \in \mathbb{R}^n$, le vecteur ligne correspondant sera noté $\boldsymbol{x}^{\mathrm{T}}$. On aura donc $\boldsymbol{x}^{\mathrm{T}} = [x_1 \ \cdots \ x_n]$ et $[\boldsymbol{x}^{\mathrm{T}}]^{\mathrm{T}} = \boldsymbol{x}$.
- Le produit scalaire entre deux vecteurs de $\mathbb{R}^n$ est défini par $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{v}^{\mathrm{T}}\boldsymbol{x} = \sum_{i=1}^{d} x_i v_i$. On définit ensuite la norme d'un vecteur $\boldsymbol{x} \in \mathbb{R}^n$ par $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}$.
- On notera $\mathbb{R}^{m\times n}$ l'ensemble des matrices à m lignes et n colonnes à coefficients réels, où m et n seront des entiers supérieurs ou égaux à 1. Une matrice $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ est dite carrée (dans le cas général, on parlera de matrice rectangulaire).
- On identifiera les vecteurs de $\mathbb{R}^n$ avec les matrices de $\mathbb{R}^{n\times 1}$.
- Étant donnée une matrice $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, on notera $\boldsymbol{A}_{ij}$ le coefficient en ligne i et colonne j de la matrice. La diagonale de $\boldsymbol{A}$ est formée par l'ensembles des coefficients $\boldsymbol{A}_{ii}$ pour $i = 1, \ldots, \min\{m, n\}$. La notation $[\boldsymbol{A}_{ij}]_{\substack{1\leq i\leq m \\ 1\leq j\leq n}}$ sera donc équivalente à $\boldsymbol{A}$. Sans ambigüité sur la taille de la matrice, on notera simplement $[\boldsymbol{A}_{ij}]$.
- Selon les besoins, on utilisera $\boldsymbol{a}_i^{\mathrm{T}}$ pour la i-ème ligne de $\boldsymbol{A}$ ou $\boldsymbol{a}_j$ pour la j-ème colonne de $\boldsymbol{A}$. Selon le cas, on aura donc $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{a}_m^{\mathrm{T}} \end{bmatrix}$ ou $\boldsymbol{A} = [\boldsymbol{a}_1 \ \cdots \ \boldsymbol{a}_n]$.
- Pour une matrice $\boldsymbol{A} = [\boldsymbol{A}_{ij}] \in \mathbb{R}^{m\times n}$, la matrice transposée de $\boldsymbol{A}$, notée $\boldsymbol{A}^{\mathrm{T}}$, est la matrice de $\mathbb{R}^{n\times m}$ telle que
$$\forall i = 1 \ldots m, \ \forall j = 1 \ldots n, \quad \boldsymbol{A}_{ji}^{\mathrm{T}} = \boldsymbol{A}_{ij}.$$
- Pour tout $n \geq 1$, $\mathbf{I}_n$ correspond à la matrice identité $\mathbb{R}^{n\times n}$ (avec 1 sur la diagonale et 0 ailleurs).

Chapitre 1

Introduction à l'optimisation

Le but de ce cours est de proposer une boîte à outils pour étudier et résoudre des problèmes d'optimisation, notamment dans le contexte moderne des sciences des données. En effet, de nombreuses tâches d'apprentissage peuvent en effet être formulées comme des problèmes d'optimisation, qui peuvent être ensuite résolus numériquement par des algorithmes dédiés. D'autres domaines d'application, comme la finance, ont aussi recours à l'optimisation pour modéliser et résoudre des problèmes tels que l'allocation de portefeuille.

Ce chapitre présente les principes de base derrière la formulation et l'étude (à la fois mathématique et algorithmique) d'un problème d'optimisation.

1.1 Définition et contexte

1.1.1 Problème et processus d'optimisation

Avant d'être un concept mathématique, un problème d'optimisation peut être verbalisé : on en donne une décomposition ci-dessous.

Définition 1.1.1 *Un problème d'optimisation mathématique est formé par les trois éléments suivants.*

- *Une* **fonction objectif** *: il s'agit du critère qui permet de quantifier la qualité d'une décision. Selon le cas, une meilleure décision peut signifier une valeur de l'objectif plus faible (auquel cas on aura affaire à un problème de* minimisation*) ou plus élevée (il s'agira alors d'un problème de* maximisation*).*

- *Des* **variables de décision** *: ce sont les différents choix qu'il est possible d'effectuer, dont la valeur influence celle de la fonction objectif. On cherche les meilleures valeurs de ces variables relativement à l'objectif.*

- *Des* **contraintes** *: celles-ci spécifient des conditions qui doivent être satisfaites par les variables de décision pour que les valeurs correspondantes soient acceptables.*

Tout processus d'optimisation passe nécessairement par une phase de **modélisation**, durant laquelle on transforme un problème concret en objet mathématique que l'on peut essayer de résoudre. Pour cela, on se base sur des **algorithmes d'optimisation** qui, bien que pouvant être formulés à la

main, ont généralement vocation à être implémentés sur un ordinateur. Lorsque l'algorithme appliqué au problème renvoie une réponse, celle-ci doit être examinée pour savoir si elle fournit une solution satisfaisante au problème : au besoin, le modèle pourra être modifié, et l'algorithme ré-executé. En pratique, le processus d'optimisation peut donc boucler après une phase d'**interprétation** ou de **discussion** avec des experts du domaine d'application dont le problème est issu.

Remarque 1.1.1 *L'optimisation numérique obéit généralement aux principes suivants :*

- *Il n'y a pas d'algorithme universel : certaines méthodes seront très efficaces sur certains types de problèmes, et peu efficaces sur d'autres. L'étude de la structure du problème facilite le choix d'un algorithme.*
- *Il peut y avoir un fort écart entre la théorie et la pratique : le calcul en précision finie peut induire des erreurs d'arrondis qui conduisent à une performance en-deçà de celle prévue par la théorie mathématique.*
- *La théorie guide la pratique, et vice-versa : pour la plupart des problèmes, il est possible de définir des expressions mathématiques qui permettent de vérifier si l'on a trouvé une solution du problème. Celles-ci sont à la base de nombreux algorithmes que nous étudierons. De manière générale, l'étude théorique d'un problème permet des avancées garanties qui surpassent souvent les heuristiques. Réciproquement, la performance de certaines méthodes a amené les chercheurs en optimisation à en proposer une étude théorique, permettant ainsi d'expliquer le comportement pratique.*

Optimisation en finance L'optimisation est utilisée en finance comme un outil de modélisation et de résolution efficace de problèmes. Dans ce contexte, on retrouve notamment des problèmes de programmation linéaire ou quadratique, dont le fameux problème d'allocation de portefeuille dû à Markowitz. Durant la dernière décennie et suite à la crise dite des *subprimes* de 2007-2008, la règlementation sur les niveaux de risques autorisés dans les milieux financiers a considérablement évolué. Les opérations financières sont aujourd'hui exécutées. En termes d'optimisation, cela a conduit au développement de problèmes sous contraintes de niveau de risque, avec une prise en compte accrue de l'aléatoire.

Optimisation en sciences des données En sciences des données, les problèmes d'optimisation impliquent souvent l'utilisation d'un ensemble massif de données dans la définition de la fonction objectif. On cherche ainsi à construire un modèle ou une prédiction sur la base d'un échantillon, qui n'est pas nécessairement un reflet exact de la distribution sous-jacente des données. Dans un contexte de données potentiellement distribuées, dont le coût d'utilisation peut être drastiquement élevé, les algorithmes d'optimisation ne sont pas tous égaux : de fait, certains algorithmes ayant fait leurs preuves dans des contextes basés sur des modèles s'avèrent peu performants dans ce contexte guidé par les données (ou *data-driven*).

1.2 Logiciels et optimisation

L'optimisation a connu un essor majeur au cours des années 1980 avec le développement des ordinateurs. Les langages de programmation les plus utilisés pour développer des algorithmes d'optimisation

étaient historiquemnet `C/C++/Fortran`, notamment pour développer des codes performants. Le développement de prototypes de recherche se fait lui plutôt via des langages interprétés tels que `Matlab/Octave/Python/Julia`, avec `Python` prenant de plus en plus d'importance de par son utilisation dans les problèmes de sciences des données et apprentissage.

En plus des langages de programmation standard, de nombreux langages ou bibliothèques ont été développés pour modéliser les problèmes d'optimisation. On peut citer notamment `GAMS`, `AMPL`, `CVX`, `Pyomo`, qui ont un champ d'application génériques, ou `MATPOWER` et `PyTorch`, qui sont dédiés à un domaine particulier (respectivement les réseaux d'énergie et l'apprentissage machine).

D'autres outils plus bas niveau ont également été développés, parfois avec un spectre plus large d'applications. Le logiciel MATLAB (propriétaire) dispose ainsi d'une boîte à outils de finance très fournie, qui comporte notamment des algorithmes d'optimisation. On s'intéressera dans ce cours à la bibliothèque `cvxpy`, développée à l'origine à l'université de Stanford (États-Unis), et qui dispose de nombreux outils pour résoudre les problèmes de programmation convexe.

En finance Dans le contexte de la finance, le recours aux tableurs est prépondérant, ce qui a conduit au développement de solveurs en tant qu'outil interne aux tableurs. C'est ainsi que Microsoft Excel, LibreOffice Calc et Google Sheets possèdent des solveurs, notamment de programmation linéaire. Si certains sont disponibles gratuitement (*open source*), le solveur le plus abouti (Microsoft Excel Solver) requiert une licence d'exploitation.

En sciences des données Python est maintenant le langage dominant. Outre la bibliothèque `scipy` standard, les bibliothèques `scikit-learn`, `TensorFlow` et `PyTorch` (ou plus récemment `JAX`) contiennent des algorithmes d'optimisation adaptés aux problèmes de sciences des données.

1.3 Bases de l'optimisation mathématique

1.3.1 Formulation générale et premières définitions

La transformation d'une description formelle d'un problème d'optimisation en objet mathématique conduit à l'écriture suivante :

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimiser}}\, f(\boldsymbol{x}) \quad \text{s.c.} \quad \boldsymbol{x}\in\mathcal{X}, \tag{1.3.1}$$

où minimiser représente ce que l'on cherche à faire (on peut vouloir minimiser ou maximiser), $\boldsymbol{x}\in\mathbb{R}^n$ est un vecteur regroupant les variables de décision du problème, $f:\mathbb{R}^n\to\mathbb{R}$ est la fonction objectif qui mesure la qualité des décisions, et $\mathcal{X}\subset\mathbb{R}^n$ est l'ensemble des points réalisables ou admissibles, qui vérifient les contraintes posées sur les variables de décision. [1]

Définition 1.3.1 *i) Un point $\boldsymbol{x}\in\mathbb{R}^n$ tel que $\boldsymbol{x}\in\mathcal{X}$ est dit* **admissible**, *ou* **réalisable**.

ii) Un point $\boldsymbol{x}\in\mathbb{R}^n$ tel que $\boldsymbol{x}\notin\mathcal{X}$ est dit **irréalisable**.

iii) Si $\mathcal{X}=\emptyset$, alors le problème (1.3.1) est dit irréalisable.

Définition 1.3.2 *i) La valeur minimale (ou le minimum) du problème (1.3.1) est notée :*

$$\min_{\boldsymbol{x}\in\mathbb{R}^n}\left\{f(\boldsymbol{x}) \text{ s.c. } \boldsymbol{x}\in\mathcal{X}\right\}. \tag{1.3.2}$$

[1] L'abbréviation *s.c.* signifie "sous la/les contrainte(s)" ; en anglais, on utilise *s.t.*, pour *subject to*.

Lorsque le problème possède une solution, la valeur minimale est la valeur de f en toute solution, et cette valeur est finie. Lorsque le problème est irréalisable, on pose par convention

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \{f(\boldsymbol{x}) \text{ s.c. } \boldsymbol{x} \in \mathcal{X}\} = +\infty.$$

Enfin, si le problème est réalisable mais qu'il n'existe pas de solution, on aura

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \{f(\boldsymbol{x}) \text{ s.c. } \boldsymbol{x} \in \mathcal{X}\} = -\infty,$$

et on dit alors que le problème est **non borné**.

ii) L'ensemble des solutions du problème (1.3.1) *est noté*

$$\operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} \{f(\boldsymbol{x}) \text{ s.c. } \boldsymbol{x} \in \mathcal{X}\}, \tag{1.3.3}$$

qu'on appelle argument minimal *(ou argmin) du problème. C'est un sous-ensemble de $\mathbb{R}^n$ (et de $\mathcal{X}$) : il s'agit de l'ensemble des points de $\mathcal{X}$ qui donnent la valeur minimale de f. Il est vide si le problème est irréalisable ou réalisable et non borné.*

Remarque 1.3.1 *Le problème* (1.3.1) *n'admet pas forcément de solution (prendre par exemple $d = 1$, $\mathcal{F} = \mathbb{R}$ et $f(w) = w$) : dans ce cas, l'argument minimal est l'ensemble vide, et la valeur minimale est $-\infty$.*

On définit de la même manière que précédemment les problèmes de maximisation ainsi que les opérateurs argmax et $\max$. Dans ce cours, on se concentrera sur les problèmes de minimisation, utilisant en cela la propriété ci-dessous.

Proposition 1.3.1 *Pour toute fonction $f : \mathbb{R}^n \to \mathbb{R}$ et tout ensemble $\mathcal{F} \subset \mathbb{R}^n$, résoudre le problème de maximisation*

$$\operatorname*{maximiser}_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \quad \text{s.c.} \quad \boldsymbol{x} \in \mathcal{F}$$

équivaut à résoudre le problème de minimisation

$$\operatorname*{minimiser}_{\boldsymbol{x}\in\mathbb{R}^n} -f(\boldsymbol{x}) \quad \text{s.c.} \quad \boldsymbol{x} \in \mathcal{F},$$

dans la mesure où

$$\operatorname*{argmax}_{\boldsymbol{x}\in\mathbb{R}^n} \{f(\boldsymbol{x})\text{s.c.}\boldsymbol{x} \in \mathcal{F}\} = \operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{R}^d} \{-f(\boldsymbol{x})\text{s.c.}\boldsymbol{x} \in \mathcal{F}\}$$

et

$$\max_{\boldsymbol{x}\in\mathbb{R}^n} \{f(\boldsymbol{x})\text{s.c.}\boldsymbol{x} \in \mathcal{F}\} = -\min_{\boldsymbol{x}\in\mathbb{R}^n} \{-f(\boldsymbol{x})\text{s.c.}\boldsymbol{x} \in \mathcal{F}\}.$$

Tout problème de maximisation se ramène donc à un problème de minimisation via une reformulation. Nous verrons d'autres exemples de reformulation, qui est une technique très utilisée pour transformer un problème en un autre problème équivalent mais que l'on saura mieux traiter théoriquement et/ou algorithmiquement.

Dans la majorité de ce cours, et notamment dans le reste de ce chapitre, on se concentrera sur des problèmes d'optimisation sans contraintes pour en caractériser plus précisément les solutions.

1.3.2 Solutions et minima

Soit le problème de minimisation sans contraintes

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimiser}}\ f(\boldsymbol{x}), \tag{1.3.4}$$

où $f : \mathbb{R}^n \to \mathbb{R}$.

Une solution du problème d'optimisation (1.3.4) est un vecteur qui conduit à la meilleure valeur possible de l'objectif. Pour un problème de minimisation sans contraintes, cela correspond au concept de minimum global.

Définition 1.3.3 (Minimum global) *Un point $\boldsymbol{x}^* \in \mathbb{R}^n$ est un* **minimum global** *du problème (1.3.4), ou plus simplement de la fonction objectif f, lorsque la propriété suivante est vérifiée :*

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \qquad f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}). \tag{1.3.5}$$

Lorsque l'inégalité est stricte pour $\boldsymbol{x} \neq \boldsymbol{x}^*$, ce minimum est unique.

Déterminer le minimum global d'une fonction est un problème difficile en général : il est donc courant d'utiliser une autre notion (plus faible) de solution, basée sur l'optimalité d'un point relativement à son voisinage.

Définition 1.3.4 (Minimum local) *Un point $\boldsymbol{x}^* \in \mathbb{R}^n$ est un* **minimum local** *du problème (1.3.4), ou plus simplement de la fonction objectif f, lorsque la propriété suivante est vérifiée :*

$$\exists \epsilon > 0,\ \forall \boldsymbol{x} \in \mathbb{R}^n, \qquad \|\boldsymbol{x} - \boldsymbol{x}^*\| \leq \epsilon \Rightarrow f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}). \tag{1.3.6}$$

Tout minimum global est ainsi un minimum local, mais l'inverse n'est pas vraie. De ce point de vue, il est donc plus aisé a priori de déterminer si un point est un minimum local que de déterminer s'il s'agit d'un minimum global. Néanmoins, en se basant sur la seule propriété (1.3.6), vérifier la minimalité locale requiert d'évaluer la fonction une infinité de fois. En pratique, on exploite les propriétés de la fonction à minimiser pour obtenir des conditions facilement vérifiables (à la main ou sur machine) qui caractérisent des minima locaux ou globaux. C'est ce que l'on décrit dans le reste de ce chapitre.

1.3.3 Conditions d'optimalité

Lorsqu'on cherche à minimiser une fonction f dérivable, il est possible de fournir des caractérisations pratiques de minima locaux (voire globaux) à l'aide des dérivées de f, qui fournissent une information locale sur la variation de la fonction. Ces caractérisations sont appelées conditions d'optimalité.

On s'intéresse d'abord aux fonctions dérivables, pour lesquelles on peut fournir une condition d'optimalité basée sur le gradient. Conformément à la suite du cours, nous donnerons le résultat dans le cas particulier des fonctions de classe $\mathcal{C}^1$.

Théorème 1.3.1 (Condition nécessaire d'optimalité à l'ordre 1) *Soient $f \in \mathcal{C}^1(\mathbb{R}^d)$ et $\boldsymbol{x}^* \in \mathbb{R}^d$. Alors*

$$[\boldsymbol{x}^* \textit{minimum local de } f\ ] \Longrightarrow \nabla f(\boldsymbol{x}^*) = 0. \tag{1.3.7}$$

Tout minimum local de f est donc un point en lequel le gradient de f s'annule, mais l'inverse n'est pas forcément vraie ! Comme son nom l'indique, cette condition est nécessaire mais pas suffisante : il peut en effet exister des points en lesquels le gradient s'annule sans qu'il s'agisse de minima. On parle alors de maxima locaux (ex : 0 pour la fonction $x \mapsto -x^2$) ou de points selles (ex : 0 pour $x \mapsto x^3$) : cette dernière catégorie pose notamment des problèmes en apprentissage profond, où les problèmes à résoudre possèdent souvent de nombreux points selles.

Plus globalement, si f est une fonction $\mathcal{C}^1$, un point $\boldsymbol{x}$ tel que $\nabla f(\boldsymbol{x}) = \mathbf{0}$ s'appelle un **point stationnaire d'ordre 1**.

Si l'on suppose que f est deux fois dérivable avec dérivée seconde continue, on peut alors établir des conditions d'optimalité plus fortes que celles à l'ordre un.

Théorème 1.3.2 (Condition nécessaire d'optimalité à l'ordre 2) *Soient $f \in \mathcal{C}^2(\mathbb{R}^d)$ et $\boldsymbol{x}^* \in \mathbb{R}^d$. Alors*

$$[\boldsymbol{x}^* \textit{minimum local de } f\ ] \Longrightarrow \left\{ \begin{array}{lcl} \nabla f(\boldsymbol{x}^*) & = & \mathbf{0}, \\ \nabla^2 f(\boldsymbol{x}^*) & \succeq & \mathbf{0}. \end{array} \right. \tag{1.3.8}$$

La nouvelle propriété fait intervenir la dérivée à l'ordre 2, qui est une matrice : pour un minimum local, la matrice hessienne est nécessairement *semi-définie positive*. [2]

Comme pour l'ordre 1, il est possible qu'un point $\boldsymbol{x}^*$ vérifie $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$ et $\nabla^2 f(\boldsymbol{x}^*) \succeq \mathbf{0}$ sans être un minimum local (voir 0 pour $w \mapsto w^3$ en dimension 1). Un point $\boldsymbol{x}$ que $\nabla f(\boldsymbol{x}) = \mathbf{0}$ et $\nabla^2 f(\boldsymbol{x}) \succeq \mathbf{0}$ s'appelle un **point stationnaire à l'ordre 2** : l'ensemble des points stationnaires d'ordre 2 contient l'ensemble des minima locaux, mais n'y est pas forcément égal.

Contrairement à l'ordre 1, l'ordre 2 permet de définir une version suffisante des conditions d'optimalité, qui permet de reconnaître un minimum local.

Théorème 1.3.3 (Condition suffisante d'optimalité à l'ordre 2) *Soient $f \in \mathcal{C}^2(\mathbb{R}^d)$ et $\boldsymbol{x}^* \in \mathbb{R}^d$. Alors*

$$\left. \begin{array}{lcl} \nabla f(\boldsymbol{x}^*) & = & \mathbf{0}, \\ \nabla^2 f(\boldsymbol{x}^*) & \succ & \mathbf{0}. \end{array} \right\} \Longrightarrow [\boldsymbol{x}^* \textit{minimum local de } f\ ]\,. \tag{1.3.9}$$

Il suffit donc que le gradient en un point soit nul et que la matrice hessienne en ce point soit définie positive [3] pour que ce point soit un minimum local.

1.4 Convexité

La convexité est une notion particulièrement importante en optimisation. Comme on le verra dans ce cours, il est possible de développer des algorithmes très efficaces pour minimiser des fonctions convexes.

[2] Une matrice symétrique $\boldsymbol{H} \in \mathbb{R}^{d \times d}$ est dite semi-définie positive si $\boldsymbol{v}^{\mathrm{T}} \boldsymbol{H} \boldsymbol{v} \geq 0$ pour tout $\boldsymbol{v} \in \mathbb{R}^d$, ce que l'on note $\boldsymbol{H} \succeq \mathbf{0}$.

[3] Une matrice symétrique $\boldsymbol{H} \in \mathbb{R}^{d \times d}$ est dite définie positive si $\boldsymbol{v}^{\mathrm{T}} \boldsymbol{H} \boldsymbol{v} > 0$ pour tout $\boldsymbol{v} \in \mathbb{R}^d$ non nul, ce que l'on note $\boldsymbol{H} \succ \mathbf{0}$.

1.4.1 Cas des fonctions convexes

Nous considérons maintenant une classe de fonctions particulièrement intéressantes en optimisation : les fonctions convexes.

La convexité est une propriété au départ géométrique, qui s'applique à des ensembles. Nous donnons donc la définition d'un ensemble convexe avant celle d'une fonction convexe.

Définition 1.4.1 (Ensemble convexe) *Un ensemble $\mathcal{F} \subseteq \mathbb{R}^n$ est dit* **convexe** *si pour tout couple de points $\mathcal{F}$, le* segment *reliant ces deux points est inclus dans $\mathcal{F}$. Mathématiquement, cela s'écrit :*

$$\forall (\boldsymbol{x}, \boldsymbol{v}) \in \mathcal{F}^2,\ \forall \alpha \in [0,1], \quad \alpha\, \boldsymbol{x} + (1-\alpha)\boldsymbol{v} \in \mathcal{F}. \tag{1.4.1}$$

Il est aussi possible de définir la convexité pour des fonctions.

Définition 1.4.2 (Fonction convexe) *Une fonction $f : \mathbb{R}^n \to \mathbb{R}$ est une* **fonction convexe** *sur un ensemble convexe $\mathcal{F} \subseteq \mathbb{R}^n$ si et seulement si*

$$\forall (\boldsymbol{x}, \boldsymbol{v}) \in \mathcal{F}^2,\ \forall \alpha \in [0,1], \quad f(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{v}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{v}). \tag{1.4.2}$$

Exemple 1.4.1 (Fonctions convexes d'une variable)

1. *La fonction $w \mapsto w^2$ est convexe sur $\mathbb{R}$.*
2. *La fonction $w \mapsto |w|$ est convexe sur $\mathbb{R}$.*

Exemple 1.4.2 (Fonctions convexes en dimension d)

1. *Si f est une norme sur $\mathbb{R}^n$, alors il s'agit d'une fonction convexe sur $\mathbb{R}^n$.*
2. *Une fonction quadratique $\boldsymbol{x} \mapsto \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}$ avec $\boldsymbol{b} \in \mathbb{R}^n$ et $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ semi-définie positive est convexe sur $\mathbb{R}^n$.*

La notion de convexité est préservée par certaines opérations : en particulier, toute somme (et plus généralement toute combinaison linéaire à coefficients positifs) de fonctions convexes est une fonction convexe.

En plus de la définition (1.4.2), on peut caractériser la convexité au moyen des dérivées à l'ordre un et deux lorsque celles-ci existent.

Théorème 1.4.1 *Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction de classe $\mathcal{C}^1$ et $\mathcal{F} \subset \mathbb{R}^n$ un ensemble convexe. Alors, f est convexe sur $\mathcal{F}$ si et seulement si*

$$\forall (\boldsymbol{x}, \boldsymbol{v}) \in \mathcal{F}^2, \quad f(\boldsymbol{v}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{v} - \boldsymbol{x}). \tag{1.4.3}$$

La propriété (1.4.3) joue un rôle fondamental en optimisation convexe.

Théorème 1.4.2 *Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction de classe $\mathcal{C}^2$ et $\mathcal{F} \subset \mathbb{R}^n$ un ensemble convexe. Alors, f est convexe sur $\mathbb{R}^n$ si et seulement si*

$$\forall \boldsymbol{x} \in \mathcal{F}, \quad \nabla^2 f(\boldsymbol{x}) \succeq \boldsymbol{0}, \tag{1.4.4}$$

càd que la matrice hessienne en tout point de $\mathbb{R}^n$ est toujours semi-définie positive.

En termes d'optimisation, les fonctions convexes possèdent la propriété suivante, extrêmement intéressante du point de vue de l'optimisation.

Théorème 1.4.3 *Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction convexe sur $\mathbb{R}^n$. Alors, on a les propriétés suivantes :*

i) Tout minimum local de f est un minimum global.

ii) Si f est de classe $\mathcal{C}^1$, alors tout point $\boldsymbol{x}^*$ tel que $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$ est un minimum global du problème.

1.4.2 Cas des fonctions fortement convexes

On peut définir une propriété encore plus puissante que la convexité, dont on verra qu'elle a un impact majeur sur les résultats d'optimisation. Il s'agit de la convexité forte, définie ci-dessous de trois manières, avec et sans dérivées.

Définition 1.4.3 (Fonction fortement convexe) *Soit $\mathcal{F} \subset \mathbb{R}^n$ un ensemble convexe et $f : \mathbb{R}^n \to \mathbb{R}$. On dit que f est une fonction* **fortement convexe** *de paramètre $\mu > 0$, ou* **μ-fortement convexe** *sur $\mathcal{F}$, si et seulement si*

$$\forall (\boldsymbol{x}, \boldsymbol{v}) \in \mathcal{F}^2,\ \forall \alpha \in [0,1], \quad f(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{v}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{v}) - \mu \frac{\alpha(1-\alpha)}{2} \|\boldsymbol{v} - \boldsymbol{x}\|^2.$$

Théorème 1.4.4 *Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction de classe $\mathcal{C}^1$, $\mathcal{F} \subset \mathbb{R}^n$ un ensemble convexe et $\mu > 0$. La fonction f est μ-fortement convexe sur $\mathcal{F}$ si et seulement si*

$$\forall (\boldsymbol{x}, \boldsymbol{v}) \in \mathcal{F}^2, \quad f(\boldsymbol{v}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}} (\boldsymbol{v} - \boldsymbol{x}) + \frac{\mu}{2} \|\boldsymbol{v} - \boldsymbol{x}\|^2. \tag{1.4.5}$$

Théorème 1.4.5 *Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction de classe $\mathcal{C}^2$, $\mathcal{F} \subset \mathbb{R}^n$ un ensemble convexe et $\mu > 0$. La fonction f est μ-fortement convexe sur $\mathcal{F}$ si et seulement si*

$$\forall \boldsymbol{x} \in \mathcal{F}, \quad \nabla^2 f(\boldsymbol{x}) \succeq \mu \boldsymbol{I}_d.$$

Les fonctions fortement convexes offrent un contexte encore plus favorable pour l'optimisation que les fonctions convexes; la raison en incombe au résultat ci-dessous.

Théorème 1.4.6 *Si $f : \mathbb{R}^n \to \mathbb{R}$ est fortement convexe, elle possède un unique minimum global.*
De plus, si f est de classe $\mathcal{C}^1$, ce minimum est l'unique solution de $\nabla f(\boldsymbol{x}) = \mathbf{0}$.

1.5 Conclusion du chapitre 1

L'optimisation est un outil mathématique qui permet de modéliser de nombreux problèmes en sciences des données et au-delà. L'optimisation comporte toujours une phase de modélisation, qui consiste à formuler mathématiquement le problème en définissant une fonction objectif, des variables de décisions et des contraintes. Cela permet de placer le problème dans une classe, et ainsi de caractériser ce qu'est une solution de ce problème.

Dans un contexte d'optimisation continue, les concepts de dérivées et de convexité sont extrêmement utiles pour caractériser les solutions d'un problème donné.

Chapitre 2

Outils d'optimisation sans contraintes

Dans ce chapitre, on s'intéresse aux algorithmes pour l'optimisation sans contraintes. On commence par la descente de gradient, qui forme la méthode de base pour le développement d'algorithmes plus avancés.

Dans l'ensemble du chapitre, on s'intéresse à des problèmes fondamentalement non linéaires de la forme :

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimiser}}\ f(\boldsymbol{x}). \tag{2.0.1}$$

2.1 Descente de gradient

L'algorithme de descente de gradient est la méthode la plus classique en optimisation dérivable, où l'on suppose que f est de classe $\mathcal{C}^1$. Elle se base sur le principe élémentaire suivant, tiré de la condition d'optimalité (1.3.7) : pour tout point $\boldsymbol{x} \in \mathbb{R}^d$,

1. Soit $\nabla f(\boldsymbol{x}) = 0$, auquel cas $\boldsymbol{x}$ est potentiellement un minimum local (c'en est un si f est convexe, et il est même global);

2. Soit $\nabla f(\boldsymbol{x}) \neq 0$, et on peut alors montrer que la fonction f décroît *localement* dans la direction de l'opposé du gradient $-\nabla f(\boldsymbol{x})$.

2.1.1 Algorithme

L'algorithme de descente de gradient est un processus itératif qui se base sur l'itération suivante :

$$\boldsymbol{x} \ \leftarrow\ \boldsymbol{x} - \alpha\nabla f(\boldsymbol{x}), \tag{2.1.1}$$

où $\alpha > 0$ est un paramètre appelé taille de pas ou longueur de pas[1]. Lorsque $\nabla f(\boldsymbol{x}) = 0$, on remarque que le vecteur $\boldsymbol{x}$ ne change pas lors de la mise à jour : cette propriété est logique puisque dans une telle situation, il est impossible d'utiliser le gradient pour déterminer un meilleur point. En revanche, dès lors que $\nabla f(\boldsymbol{x}) \neq 0$, on s'attend à ce qu'il existe des valeurs de α pour lesquelles une telle mise à jour permette d'obtenir un meilleur point (avec une valeur de fonction objectif plus faible).

[1] On parle en anglais de *stepsize* ou *steplength*.

En utilisant la règle (2.1.1) au sein d'un processus itératif, on peut construire un algorithme dont le but consiste à minimimiser la fonction objectif f : il s'agit de l'algorithme de **descente de gradient**, parfois également appelé *méthode de la plus forte pente*, dont l'énoncé complet est donné par l'algorithme 1.

Algorithme 1: Descente de gradient pour la minimisation d'une fonction f.

Initialisation: Choisir $\boldsymbol{x}_0 \in \mathbb{R}^d$.
Pour $k = 0, 1, ...$

1. Calculer le gradient $\nabla f(\boldsymbol{x}_k)$.
2. Définir une taille de pas $\alpha_k > 0$.
3. Poser $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$.

FinPour

Tel qu'écrit ici, l'algorithme ne possède pas de critère d'arrêt; il s'agit en fait d'un schéma générique, dont il existe de nombreuses variantes qui correspondent à des choix spécifiques de critère d'arrêt, de taille de pas, voire de point initial. Nous passons en revue ces différents aspects ci-dessous.

Critère d'arrêt Dans la pratique, un algorithme est soumis à des contraintes de budget (en termes d'opérations arithmétiques, de temps d'exécution, d'itérations). Des critères d'arrêt de la méthode sont donc nécessaires afin de forcer l'algorithme à terminer si ces limites sont atteintes. Pour l'algorithme 1, il pourrait s'agir de terminer la méthode après avoir effectué $k_{\max}$ itérations, auquel cas $\boldsymbol{x}_{k_{\max}}$ représenterait la meilleur solution obtenue.

Par ailleurs, pour mesurer la capacité d'un algorithme à converger vers une solution du problème, on introduit généralement un critère d'arrêt basé sur les conditions d'optimalité. Par exemple, le critère suivant est très fréquemment utilisé pour les algorithmes basés sur le gradient :

$$\|\nabla f(\boldsymbol{x}_k)\| < \epsilon, \tag{2.1.2}$$

où $\epsilon > 0$ représente une précision donnée, de sorte que ce critère est plus difficile à vérifier lorsque ϵ est très faible.

Enfin, il est toujours possible (et souvent recommandé) d'ajouter des critères d'arrêt de secours, qui permettent de ne pas continuer à faire tourner un algorithme inutilement. Par exemple, lorsque la différence entre deux itérés successifs est du niveau de la précision machine, cela signifie que l'algorithme ne progresse plus, et on peut donc en arrêter l'exécution.

Choix du point initial La performance d'un algorithme peut être grandement améliorée lorsque le point initial est bien choisi. En règle générale, il est cependant difficile de déterminer un tel point : des stratégies de démarrage multiple, qui consistent à effectuer quelques itérations en partant de différents points choisis au hasard, peuvent permettre de déterminer un bon point initial. Mieux encore, dans de nombreuses applications, il existe déjà une valeur de référence ou une idée de ce que pourrait être la solution : dans ce cas, il est souvent avantageux de partir d'un tel point, et de chercher à l'améliorer via le processus d'optimisation.

2.1.2 Choix de la taille de pas

Nous présentons ici les principales techniques de choix de taille de pas; là encore, toute connaissance complémentaire sur le problème peut conduire à de meilleurs choix.

Taille de pas constante L'une des stratégies les plus courantes consiste à utiliser une taille de pas constante pour toutes les itérations de l'algorithme, soit $\alpha_k = \alpha > 0$ pour tout k. Selon le budget disponible, plusieurs valeurs peuvent être testées afin de ne pas se limiter à une seule possibilité. En apprentissage, il est ainsi fréquent de tester une grille de valeurs. Si f vérifie l'hypothèse 2.1.1 (voir ci-dessous), on sait qu'il existe des valeurs constantes qui conduiront à un algorithme convergent. En particulier, le choix

$$\alpha_k = \alpha = \tfrac{1}{L}, \tag{2.1.3}$$

où L représente la constante de Lipschitz pour le gradient, est adapté au problème. Cependant, ce choix nécessite de connaître cette constante de Lipschitz, et une telle information n'est pas forcément aisée à obtenir en pratique.

Taille de pas décroissante Une autre technique classique pour choisir le pas consiste à utiliser une suite de tailles de pas décroissante, de sorte que $\alpha_k \to 0$ lorsque $k \to \infty$. Un tel choix peut permettre d'établir des garanties de convergence; en revanche, il peut aussi conduire à un arrêt prématuré de l'algorithme en pratique, si les tailles de pas diminuent trop rapidement. La vitesse avec laquelle la taille de pas α_k tend vers 0 est un aspect critique de ces stratégies.

Choix adaptatif via une recherche linéaire Les techniques de recherche linéaire sont très populaires en optimisation continue (elles sont moins employées en sciences des données, pour des raisons que nous détaillerons plus loin). Pour une itération k donnée, le but d'une recherche linéaire est de calculer la taille de pas α_k qui conduit à une décroissance de la fonction objectif dans la direction choisie (dans le cas de l'algorithme 1, il s'agit de la direction $-\nabla f(\boldsymbol{x}_k)$). Une recherche linéaire exacte recherche la taille de pas conduisant à la décroissance maximale, ce qui peut être coûteux à effectuer en pratique. On lui préfèrera plutôt des approches *inexactes*, dont la plus répandue est la technique de retour arrière (*backtracking* en anglais) décrite par l'algorithme 2.

Algorithme 2: Recherche linéaire avec retour arrière dans la direction d.

Entrées : $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{d} \in \mathbb{R}^d$, $\alpha_0 \in \mathbb{R}^d$.
Initialisation : Définir $\alpha = \alpha_0$.
Tant que $f(\boldsymbol{x} + \alpha \boldsymbol{d}) \geq f(\boldsymbol{x})$
  $\alpha \to \frac{\alpha}{2}$.
Fin
Sortie : α.

Une telle recherche linéaire peut être utilisée à l'étape 2 de l'algorithme 1 en appelant l'algorithme 2 avec $\boldsymbol{x} = \boldsymbol{x}_k$, $\boldsymbol{d} = -\nabla f(\boldsymbol{x}_k)$ et (par exemple) $\alpha_0 = 1$. Dans ce cas, on cherche généralement une taille de pas vérifiant la condition dite d'Armijo, définie par

$$f(\boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k)) < f(\boldsymbol{x}_k) - c\alpha \|\nabla f(\boldsymbol{x}_k)\|^2, \tag{2.1.4}$$

avec $c \in$ ‘$(0, 1/2)$. Il existe de très nombreuses extensions de ce schéma, qui ont l'avantage de garantir l'obtention d'un meilleur point au sens de la valeur de la fonction objectif. Cependant, ces techniques requièrent des évaluations supplémentaires de la fonction objectif, ce qui représente un coût parfois non négligeable.

2.1.3 Bases théoriques de la descente de gradient

Nous allons maintenant établir des garanties dites de complexité pour l'algorithme 1. Étant donné un critère de convergence dépendant d'une précision ϵ, il s'agira de borner le nombre d'itérations nécessaires pour satisfaire ce critère comme une fonction de la précision ϵ. Nous obtiendrons des résultats différents selon que nous nous intéresserons aux fonctions non convexes, convexes ou fortement convexes. On se concentrera cependant sur les fonctions de la classe définie ci-dessous.

Définition 2.1.1 *Soit $L > 0$. L'ensemble $\mathcal{C}_L^{1,1}(\mathbb{R}^d)$ est l'ensemble des fonctions $f : \mathbb{R}^d \to \mathbb{R}$ telles que $f \in \mathcal{C}^1(\mathbb{R}^d)$ et ∇f est L-lipschitzienne, c'est-à-dire que*

$$\forall (\boldsymbol{v}, \boldsymbol{x}) \in (\mathbb{R}^d)^2, \quad \|\nabla f(\boldsymbol{v}) - \nabla f(\boldsymbol{x})\| \leq L\|\boldsymbol{v} - \boldsymbol{x}\|. \tag{2.1.5}$$

Une propriété importante des fonctions de cette classe est donnée ci-dessous.

Proposition 2.1.1 *Soit $f \in \mathcal{C}_L^{1,1}(\mathbb{R}^d)$, et deux vecteurs $\boldsymbol{v}$ et $\boldsymbol{x}$ dans $\mathbb{R}^d$. Alors,*

$$f(\boldsymbol{v}) \leq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{v} - \boldsymbol{x}) + \frac{L}{2}\|\boldsymbol{v} - \boldsymbol{x}\|^2. \tag{2.1.6}$$

La propriété ci-dessus est fondamentale, car elle permet d'approcher la fonction f par au-dessus via une fonction explicite en $\boldsymbol{v}$.

Dans la suite, nous nous plaçons dans l'hypothèse suivante.

Hypothèse 2.1.1 *La fonction objectif f du problème (2.0.1) est de classe $\mathcal{C}_L^{1,1}(\mathbb{R}^d)$ pour $L > 0$, et f est minorée sur $\mathbb{R}^d$ par $f_{low} \in \mathbb{R}$ (on a donc $f(\boldsymbol{x}) \geq f_{low}$ $\forall \boldsymbol{x} \in \mathbb{R}^d$).*

Grâce à cette hypothèse, on peut déterminer un bon choix de longueur de pas.

Proposition 2.1.2 *On considère la k-ième itération de l'algorithme 1 appliqué à une fonction f vérifiant l'hypothèse 2.1.1. Supposons que $\nabla f(\boldsymbol{x}_k) \neq 0$; alors, si la taille de pas est choisie de sorte que $0 < \alpha_k < \frac{2}{L}$, on a*

$$f(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)) < f(\boldsymbol{x}_k).$$

De plus, si $\alpha_k = \frac{1}{L}$, on obtient :

$$f(\boldsymbol{x}_k - \frac{1}{L}\nabla f(\boldsymbol{x}_k)) < f(\boldsymbol{x}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|^2. \tag{2.1.7}$$

Le résultat de la proposition 2.1.2 permet de garantir que la fonction objectif peut décroître à pas constant.

Nous allons voir comment exploiter ce résultat selon la nature de la fonction objectif.

Cas non convexe En optimisation non convexe, un résultat de complexité correspond à une borne sur le nombre d'itérations requis pour obtenir $\|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$. Pour l'algorithme de descente de gradient, on peut ainsi obtenir le résultat suivant.

Théorème 2.1.1 (Complexité de la descente de gradient pour les fonctions non convexes) *Soit f une fonction non convexe vérifiant l'hypothèse 2.1.1. On suppose que l'algorithme 1 est appliqué à f avec $\alpha_k = \frac{1}{L}$; alors, pour tout $\epsilon > 0$, l'algorithme atteint un itéré $\boldsymbol{x}_k$ tel que $\|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$ en au plus $\mathcal{O}(\epsilon^{-2})$ itérations.*

Démonstration. Soit K un indice tel que pour tout $k = 0, \ldots, K-1$, on ait $\|\nabla f(\boldsymbol{x}_k)\| > \epsilon$. D'après la proposition 2.1.2, on a

$$\forall k = 0, \ldots, K-1, \quad f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}_k)\|^2 < f(\boldsymbol{x}_k) - \frac{1}{2L}\epsilon^2.$$

En sommant cette relation pour toutes les itérations d'indices 0 à $K-1$, on obtient :

$$\sum_{k=0}^{K-1} f(\boldsymbol{x}_{k+1}) \leq \sum_{k=0}^{K-1} f(\boldsymbol{x}_k) - \frac{K}{2L}\epsilon^2,$$

ce qui donne après simplification :

$$f(\boldsymbol{x}_K) \leq f(\boldsymbol{x}_0) - \frac{K}{2L}\epsilon^2.$$

Comme $f(\boldsymbol{x}_K) \geq f_{low}$ d'après l'hypothèse 2.1.1, on aboutit à

$$K \leq 2L(f(\boldsymbol{x}_0) - f_{low})\epsilon^{-2}.$$

Par conséquent, on vient de montrer que le nombre d'itérations pour lesquelles $\|\nabla f(\boldsymbol{x}_k)\| > \epsilon$ est majoré par

$$\lceil 2L(f(\boldsymbol{x}_0) - f_{low})\epsilon^{-2} \rceil = \mathcal{O}(\epsilon^{-2}).$$

□

Remarque 2.1.1 *La notion de borne de complexité est parfois remplacée par celle, équivalente, de vitesse de convergence, qui ne fait pas directement intervenir de précision. Dans le cas présent, par exemple, on dira que l'algorithme de descente de gradient converge en $\mathcal{O}\left(\frac{1}{\sqrt{K}}\right)$, dans le sens où on peut établir (via un raisonnement similaire à celui utilisé pour prouver le théoreme 2.1.1) que*

$$\min_{0 \leq k \leq K-1} \|\nabla f(\boldsymbol{x}_k)\| \leq \frac{C}{\sqrt{K}},$$

où la constante C dépend de $f(\boldsymbol{x}_0)$, f_{low} et L.

On notera que le résultat du théorème 2.1.1 ne garantit pas que la méthode converge vers un minimum local, seulement vers un point stationnaire d'ordre 1 (qui peut donc être un point selle ou un maximum local). En pratique, s'il est facile de trouver des exemples "pathologiques" où l'algorithme de descente de gradient converge vers un point selle ou reste bloqué en un maximum local, on observe plutôt que la méthode converge vers de "bons points". Un résultat récent a permis de préciser ce comportement en présence d'une dérivée seconde.

Théorème 2.1.2 *Sous les hypothèses du théorème 2.1.1, on suppose que f est de classe $\mathcal{C}^2$ et que le point initial $\boldsymbol{x}_0$ est choisi aléatoirement dans $\mathbb{R}^d$. Alors, la méthode de descente de gradient converge presque sûrement vers un point stationnaire d'ordre deux.*

Pour des fonctions de classe $\mathcal{C}^2$, le théorème 2.1.1 garantit donc (avec probabilité 1) que la descente de gradient ne converge pas vers un point en lequel la matrice hessienne n'est pas semi-définie positive.

2.1.4 Cas convexe et fortement convexe

On suppose maintenant (en plus de l'hypothèse 2.1.1) que la fonction objectif est convexe : pour le même algorithme que ci-desssus, on peut alors démontrer que des propriétés plus fortes sont vérifiées à un coût plus faible. Ces résultats illustrent l'intérêt des fonctions convexes en optimisation.

Pour le reste de cette section, on considèrera l'hypothèse suivante.

Hypothèse 2.1.2 *La fonction f est convexe, et admet un minimum $\boldsymbol{x}^* \in \mathbb{R}^d$. On notera la valeur minimale par $f^* := f(\boldsymbol{x}^*) = \min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$.*

Pour une précision $\epsilon > 0$, on va maintenant s'intéresser au nombre d'itérations nécessaires pour atteindre $\boldsymbol{x}_k$ tel que $f(\boldsymbol{x}_k) - f^* \leq \epsilon$. En pratique, f^* n'est pas toujours connue, mais un tel critère a son importance (et peut toujours être estimé en évaluant la valeur de l'objectif).

Théorème 2.1.3 (Complexité de la descente de gradient pour les problèmes convexes) *On considère une fonction f vérifiant les hypothèses 2.1.1 et 2.1.2. On considère l'algorithme 1 appliqué à f avec $\alpha_k = \frac{1}{L}$; alors, pour tout $\epsilon > 0$, l'algorithme obtient $\boldsymbol{x}_k$ tel que $f(\boldsymbol{x}_k) - f^* \leq \epsilon$ en au plus $\mathcal{O}(\epsilon^{-1})$ itérations.*

Remarque 2.1.2 *Par le même raisonnement, on peut montrer que la vitesse de convergence de la descente de gradient dans le cas convexe est $\mathcal{O}\left(\frac{1}{K}\right)$, c'est-à-dire qu'il existe $C > 0$ (dont la valeur dépend de $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|$ et L) telle que*

$$f(\boldsymbol{x}_K) - f^* \leq \frac{C}{K}.$$

Nous nous tournons finalement vers le cas fortement convexe.

Théorème 2.1.4 (Complexité de la descente de gradient pour les fonctions fortement convexes) *Soit f une fonction vérifiant les hypothèse 2.1.1 et 2.1.2, dont on suppose de surcroît qu'elle est μ-fortement convexe de paramètre $\mu \in (0, L]$. On considère l'algorithme 1 appliqué à f avec $\alpha_k = \frac{1}{L}$; alors, pour tout $\epsilon > 0$, la méthode obtient $\boldsymbol{x}_k$ tel que $f(\boldsymbol{x}_k) - f^* \leq \epsilon$ en au plus $\mathcal{O}(\frac{L}{\mu}\ln(\frac{1}{\epsilon}))$ itérations.*

Remarque 2.1.3

- *Par une démonstration similaire, on peut montrer (et on dira donc) que la vitesse de convergence de la descente de gradient dans le cas fortement convexe est en $\mathcal{O}\left((1 - \frac{\mu}{L})^k\right)$;*
- *Dans le cas fortement convexe, on peut également établir ces résultats pour le critère de convergence des itérés $\|\boldsymbol{x}_k - \boldsymbol{x}^*\| \leq \epsilon$: la distance entre l'itéré courant et l'optimum (qui est unique) décroît donc à un taux $\mathcal{O}\left((1 - \frac{\mu}{L})^k\right)$.*

Pour terminer, on notera que les preuves de vitesses de convergence dans le cas convexe (et fortement convexe) sont souvent plus techniques que celles du cas non convexe. Cette technicité est nécessaire pour certifier que les vitesses de convergence de ces algorithmes sont meilleures. Dans ce cours, nous nous intéressons principalement aux résultats, et à ce qu'ils impliquent sur la facilité de résolution de ces problèmes.

2.2 Optimisation sans gradients

2.2.1 Des fonctions aux problèmes non lisses

Il existe des problèmes d'optimisation pour lesquels la fonction objectif n'est pas dérivable. De tels problèmes sont appelés des *problèmes non lisses*. Leurs fonctions objectifs sont également qualifiées de non lisses (par opposition aux fonctions lisses). Pour ce chapitre, on définira les fonctions non lisses comme suit.

Définition 2.2.1 (Fonctions non lisses) *Une fonction $f : \mathbb{R}^d \to \mathbb{R}$ est dite* **non lisse** *si elle n'est pas dérivable partout.*

Exemple 2.2.1 (Exemples de fonctions non lisses)

- $w \mapsto |w|$ *de* $\mathbb{R}$ *dans* $\mathbb{R}$*;*
- $w \mapsto \|w\|_1$ *de* $\mathbb{R}^d$ *dans* $\mathbb{R}$*;*
- *ReLU :* $w \mapsto \max\{w, 0\}$ *de* $\mathbb{R}^d$ *dans* $\mathbb{R}$*.*

Remarque 2.2.1 *Les exemples précédents montrent qu'une fonction non lisse peut être continue. En pratique, on s'intéressera plutôt à ce type de fonctions non lisses.*

Comme les fonctions non lisses ne sont pas différentiables partout, il n'est pas envisageable de résoudre les problèmes non lisses via des méthodes basées sur le gradient. Il existe cependant plusieurs approches possibles pour traiter ces problèmes.

Une première technique consiste à trouver une formulation lisse équivalente à celle du problème non lisse. Le problème $\mathrm{minimiser}_{w\in\mathbb{R}} |w|$ est par exemple équivalent à

$$\underset{w,t^+,t^-\in\mathbb{R}}{\mathrm{minimiser}}\, t^+ + t^- \quad \text{s. c.} \quad w = t^+ - t^-, t^+ \geq 0, t^- \geq 0.$$

Cette reformulation est un programme linéaire, qui fait partie des classes de problèmes classiques, pour lesquelles des algorithmes et codes très efficaces existent.

Lorsque la fonction est non lisse mais lipschitzienne (ce que l'on notera $\mathcal{C}_L^{0,0}$, par analogie avec $\mathcal{C}_L^{1,1}$), on peut en revanche essayer d'appliquer une méthode de gradient, en raison de la propriété suivante.

Théorème 2.2.1 *Soit $f : \mathbb{R}^d \to \mathbb{R}$ une fonction lipschitzienne. Alors, f est dérivable en presque tout point de $\mathbb{R}^d$.*

La fonction ReLU est un exemple classique de fonction lipschitzienne, ce qui fait que les constructions impliquant des fonctions ReLU (comme les réseaux de neurones) ne sont pas dérivables en tout point. En revanche, il est possible de traiter ces problèmes comme dérivables presque partout; cela reste un problème pour certifier qu'un point est un minimum local, car il est fréquent que les fonctions non lisses ne soient pas dérivables en leurs points extrêmaux. La notion de dérivée généralisée, que nous présentons ci-après, est précisément utilisée dans ce but.

2.2.2 Méthodes de sous-gradient

Nous nous concentrons ici sur le cas des fonctions non lisses *convexes*, cas fréquent en pratique, et nous définissons une notion généralisée de gradient.

Définition 2.2.2 (Sous-gradient et sous-différentiel) *Soit $f : \mathbb{R}^d \to \mathbb{R}$ une fonction convexe. Un vecteur $\boldsymbol{g} \in \mathbb{R}^d$ est appelé un sous-gradient de f en $\boldsymbol{x} \in \mathbb{R}^d$ si*

$$\forall \boldsymbol{z} \in \mathbb{R}^n, \qquad f(\boldsymbol{z}) \ \geq \ f(\boldsymbol{x}) + \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}).$$

L'ensemble des sous-gradients de f en $\boldsymbol{x}$ est appelé le sous-différentiel de f en $\boldsymbol{x}$; on le note $\partial f(\boldsymbol{x})$.

Lorsque la fonction f est dérivable en $\boldsymbol{x}$, on a $\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}$, ce qui montre que la notion de sous-différentiel généralise bien celle du gradient.

Les sous-différentiels permettent de caractériser l'optimalité (globale) pour les fonctions convexes, comme le montre le résultat ci-dessous.

Théorème 2.2.2 *Soit $f : \mathbb{R}^d \to \mathbb{R}$ une fonction convexe, et $\boldsymbol{x} \in \mathbb{R}^d$.*

$$\mathbf{0} \in \partial f(\boldsymbol{x}) \quad \Leftrightarrow \quad \boldsymbol{x} \text{ minimum of } f$$

Exemple 2.2.2 *Soit $f : \mathbb{R} \to \mathbb{R}$, $f(w) = |w|$.*

$$\partial f(w) \ = \ \begin{cases} -1 & \text{si } w < 0 \\ 1 & \text{si } w > 0 \\ [-1, 1] & \text{si } w = 0. \end{cases}$$

L'ensemble $[-1, 1]$ contient 0, ce qui prouve que $w^* = 0$ est un minimum (en fait l'unique minimum) de f.

Remarque 2.2.2 *Il est aussi possible de définir des sous-gradients pour les fonctions convexes: cependant, pour de telles fonctions, le sous-différentiel peut être vide en certains points (par exemple en un maximum local), ce qui en limite la portée. Il existe d'autres notions de dérivée généralisée, qui peuvent être utilisées dans un processus d'optimisation.*

En utilisant la notion de sous-gradient, on peut alors définir l'analogue de la descente de gradient pour des fonctions convexes non lisses : c'est le principe décrit dans l'algorithme 3.

Une telle méthode offre un degré de liberté dans le choix du sous-gradient, ce qui peut poser problème. En effet, un sous-gradient peut ne pas être une direction de descente, et au contraire être une direction de montée *pour tout pas possible*. C'est notamment le cas lorsque l'on se trouve en un minimum pour lequel il existe plusieurs sous-gradients : toute direction autre que $\mathbf{0}$ sera nécessairement une direction où la fonction augmentera localement.

Variantes de la méthode du sous-gradient Les variantes de l'algorithme de descente de gradient ont souvent un équivalent dans le cadre des méthodes de sous-gradient, même si leur analyse est parfois plus délicate. L'une des méthodes les plus populaires est l'algorithme du sous-gradient stochastique, utilisé en optimisation stochastique et (par conséquent) en apprentissage de manière générique.

Algorithme 3: Méthode de sous-gradient.

Initialisation : $\boldsymbol{x}_0 \in \mathbb{R}^d$.
Pour $k = 0, 1, ...$

1. Calculer un sous-gradient $\boldsymbol{g}_k \in \partial f(\boldsymbol{x}_k)$.
2. Calculer une taille de pas $\alpha_k > 0$.
3. Poser $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k$.

Fin

2.3 Accélération

2.3.1 Introduction au concept de momentum

Dans la partie précédente, nous avons obtenu des bornes de complexité dites *au pire cas* : celles-ci quantifient la performance d'un algorithme sur une classe de problèmes donnée, mais elles n'indiquent pas la meilleure performance possible pour *une classe d'algorithmes donnée*. On parle ainsi de bornes supérieures, par opposition aux bornes inférieures dont nous allons discuter ici.

Pour l'optimisation d'une fonction $\mathcal{C}_L^{1,1}$ non convexe par une méthode de gradient, on sait que la borne en $\mathcal{O}(\epsilon^{-2})$, que nous avons obtenue dans le théorème 2.1.1 ne peut pas être améliorée sans que l'algorithme utilise plus d'information. En revanche, pour le cas convexe, il existe des algorithmes qui peuvent atteindre une borne de complexité $\mathcal{O}(\epsilon^{-1/2})$, ce qui représente une amélioration notable par rapport à la borne $\mathcal{O}(\epsilon^{-1})$ que nous avons obtenu pour la descente de gradient (cf théorème 2.1.3). Ces méthodes ayant une meilleure complexité reposent de manière plus ou moins explicite sur une technique dite d'**accélération**.

Le principe général de l'accélération (que l'on appelle parfois *momentum*) est d'utiliser de l'information des itérations précédentes (généralement la dernière), ce qui permet potentiellement de profiter du momentum de l'itération précédente pour effectuer un meilleur pas.

2.3.2 Méthode du gradient accéléré

L'utilisation la plus connue des techniques d'accélération est due au mathématicien Yurii Nesterov : l'algorithme associé est d'ailleurs souvent appelé "méthode de Nesterov". Une description générique de cette méthode est donnée par l'algorithme 4.

Comme l'algorithme de descente de gradient vu en section 2.1, l'algorithme du gradient accéléré requiert une seule évaluation de gradient à chaque iération; en revanche, contrairement à l'algorithme de descente de gradient, dans l'algorithme 4, le gradient n'est pas évalué en le point courant $\boldsymbol{x}_k$, mais en une combinaison linéaire de ce point avec le pas précédent $\boldsymbol{x}_k - \boldsymbol{x}_{k-1}$: ce second terme, appelé **terme de momentum**, est à l'origine de la performance améliorée des techniques accélérées.

L'algorithme 4 peut s'écrire d'une autre manière en utilisant deux suites de vecteurs démarrant respectivement à $\boldsymbol{x}_0$ et $\boldsymbol{z}_0 = \boldsymbol{x}_0$; on réécrit alors la mise à jour (2.3.1) comme suit :

$$\left\{ \begin{array}{lcl} \boldsymbol{x}_{k+1} & = & \boldsymbol{z}_k - \alpha_k \nabla f(\boldsymbol{z}_k) \\ \boldsymbol{z}_{k+1} & = & \boldsymbol{x}_{k+1} + \beta_{k+1}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k). \end{array} \right. \tag{2.3.2}$$

Algorithme 4: Méthode du gradient accéléré

Initialisation : $\boldsymbol{x}_0 \in \mathbb{R}^d$, $\boldsymbol{x}_{-1} = \boldsymbol{x}_0$.
Pour $k = 0, 1, ...$

1. Calculer une taille de pas $\alpha_k > 0$ et un paramètre $\beta_k > 0$.
2. Définir le nouveau point comme :

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f\left(\boldsymbol{x}_k + \beta_k(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})\right) + \beta_k(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}). \tag{2.3.1}$$

Fin

Grâce à cette nouvelle formulation, on distingue bien le processus en deux étapes contenu dans la méthode du gradient accéléré : un pas de gradient sur $\boldsymbol{z}_k$ et un pas de type "momentum" sur $\boldsymbol{x}_{k+1}$.

Choix des paramètres En plus du choix de la *taille de pas*, déjà présent dans l'algorithme de descente de gradient, l'algorithme 4 possède un autre paramètre β_k, dit de momentum. Le paramètre α_k peut être choisi selon les mêmes techniques que celles décrites en section 2.1.2 : le choix classique est celui d'une taille de pas constante $\alpha_k = \frac{1}{L}$, qui permet d'obtenir les résultats de complexité attendus.

Le choix de β_k est également crucial dans l'obtention de ces garanties. Ainsi, les valeurs de β_k proposées originellement par Nesterov dépendent de la fonction objectif :

- Si f is μ-fortement convexe, on pose :

$$\beta_k = \beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}} \tag{2.3.3}$$

  pour tout k. Cela requiert de connaître à la fois la constante de Lipschitz pour le gradient et le paramètre de convexité forte.

- Pour une fonction f convexe quelconque, β_k est calculé de manière adaptative en utilisant deux suites :

$$t_{k+1} = \frac{1}{2}(1 + \sqrt{1 + 4t_k^2}), t_0 = 0, \quad \beta_k = \frac{t_k - 1}{t_{k+1}}. \tag{2.3.4}$$

Le résultat suivant résume les résultats de complexité qui peuvent être obtenus pour l'algorithme 4.

Théorème 2.3.1 *Soit l'algorithme 4 appliqué à une fonction f vérifiant les hypothèses 2.1.1 et 2.1.2, avec $\alpha_k = \frac{1}{L}$ et β_k choisi selon la règle (2.3.4). Alors, pour tout $\epsilon > 0$, l'algorithme atteint $\boldsymbol{x}_k$ tel que $f(\boldsymbol{x}_k) - f^* \leq \epsilon$ en au plus*

i) $\mathcal{O}(\epsilon^{-1/2})$ *itérations pour une fonction convexe;*

ii) $\mathcal{O}\left(\sqrt{\frac{L}{\mu}}\ln(\frac{1}{\epsilon})\right)$ *itérations pour une fonction μ-fortement convexe.*

On peut également obtenir des vitesses de convergence pour le gradient accéléré, qui reflètent également cette amélioration. Ainsi, pour des fonctions μ-fortement convexes, on peut démontrer que $f(\boldsymbol{x}_k) - f^* = \mathcal{O}\left(\left(1 - \sqrt{\frac{\mu}{L}}\right)^k\right)$, ce qui est donc plus rapide que la vitesse en $\mathcal{O}\left(\left(1 - \frac{\mu}{L}\right)^k\right)$ de la descente de gradient.

2.3.3 Méthode Heavy-ball

La méthode de la boule lestée (ou *heavy ball*), développée par Boris T. Polyak in 1964, est un précurseur de l'algorithme de Nesterov dédié à la minimisation des fonctions fortement convexes. Sa k-ième itération s'écrit :

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k) + \beta(\boldsymbol{x}_k - \boldsymbol{x}_{k+1}),$$

où la taille de pas et le paramètre de momentum sont typiquement choisis constants en fonction des constantes de Lipschitz et paramètre de convexité forte du problème. La différence entre ce schéma et celui de Nesterov repose sur le point en lequel le gradient est évalué : en ce sens, la méthode de la boule lestée effectue d'abord un pas de gradient, puis un pas de momentum, tandis que l'algorithme du gradient accéléré adopte l'approche inverse. Il est possible de montrer que l'algorithme de la boule lestée possède la même complexité que le gradient accéléré sur des fonctions quadratiques strictement convexes, en revanche le résultat est faux sur des fonctions strictement convexes générales.

2.4 Conclusion

Si l'on considère un problème d'optimisation générique, il n'est en général pas possible d'obtenir une formule explicite pour ses solutions. On utilise alors les propriétés de la fonction à optimiser pour développer des algorithmes itératifs qui peuvent résoudre le problème numériquement. L'algorithme de descente de gradient est l'exemple classique d'une méthode itérative pour l'optimisation différentiable, dont de nombreuses variantes ont été proposées. Ces variantes diffèrent notamment dans leur stratégie de choix de taille de pas. Pour analyser le comportement de la descente de gradient, on se base sur des vitesses de convergence (ou des bornes de complexité) qui sont spécifiques à une classe de fonctions donnée. Ainsi, la vitesse de convergence de la descente de gradient est meilleure sur un problème fortement convexe que sur un problème convexe, et meilleure sur un problème convexe que sur un problème non convexe.

On peut alors se demander quelles sont les bornes de complexité optimales que l'on peut obtenir avec des méthodes basées sur le gradient. Dans le cas non convexe, il n'existe pas de méthode basée sur le gradient avec une meilleure complexité. En revanche, lorsque la fonction objectif est convexe ou fortement convexe, on peut construire des algorithmes dits "accélérés", pour lesquels les meilleures vitesses de convergence possibles sont atteintes. Ces méthodes se basent essentiellement sur le concept de momentum, qui consiste à combiner l'information du gradient avec le déplacement effectué à l'itération précédente : ce paradigme est à la base de nombreux algorithmes d'optimisation très efficaces en pratique/

Les fonctions non lisses (et plus particulièrement non dérivables) sont fréquentes en optimisation, et il peut s'avérer difficile de construire des algorithmes adaptés à leur minimisation. Dans certains cas, on peut avoir recours à des dérivées généralisées telles que les sous-gradients, qui permettent de généraliser en partie les stratégies basées sur les gradients en optimisation lisse.

Chapitre 3

Optimisation pour les sciences des données

Dans ce chapitre, nous allons véritablement prendre en compte la structure des problèmes liés aux sciences des données. En premier lieu, nous nous intéressons aux problèmes sous forme régularisée, qui sont très fréquents dans ce domaine. Nous abordons par la suite la dépendance d'une partie de l'objectif à un échantillon de données, ce qui conduit à la définition des méthodes de gradient stochastique. Nous en décrivons les principes de bases ainsi que certaines variantes plus avancées.

3.1 Régularisation

3.1.1 Problèmes régularisés et premier exemple

Comme décrit dans l'introduction de ces notes, une pratique courante en apprentissage consiste à favoriser les modèles possédant une structure spécifique. En termes d'optimisation, cela se fait à travers la fonction objectif, ce qui donne des problèmes de la forme

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimiser}}\, f(\boldsymbol{x}) \quad + \lambda\Omega(\boldsymbol{x}).$$

où f est une fonction de coût, Ω est une fonction appelée *terme de régularisation* et $\lambda > 0$ est appelé un paramètre de régularisation.

Exemple 3.1.1 (Régularisation ℓ_2) *Un problème avec régularisation ℓ_2, dite régularisation écrêtée ou* ridge *en anglais, est de la forme suivante :*

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimiser}}\, f(\boldsymbol{x}) + \frac{\lambda}{2}\|\boldsymbol{x}\|^2.$$

Le terme de régularisation $\boldsymbol{x} \mapsto \frac{1}{2}\|\boldsymbol{x}\|^2$ possède plusieurs interprétations. Il pénalise les vecteurs $\boldsymbol{x}$ dont les composantes sont larges, et on peut montrer que sa présence est équivalente à imposer une contrainte sur la norme au carré $\|\boldsymbol{x}\|^2$. Par ailleurs, la régularisation écrêtée réduit la variance du problème par rapport aux données définissant f, ce qui est important dans notre contexte. Enfin, lorsque $\lambda > 0$ est suffisamment grand et que la fonction de coût f est minorée, on peut montrer que la fonction objectif est fortement convexe, et donc qu'il existe au plus un minimum global au problème.

Remarque 3.1.1 *Dans le cadre d'un algorithme de type descente de gradient (ou gradient stochastique classique) utilisé en apprentissage, ajouter un terme de régularisation ℓ_2 correspond au concept de* batch normalization.

3.1.2 Régularisation et parcimonie

Lorsque l'on construit un modèle basé sur des données réparties en attributs (*features*) et labels, on peut vouloir un modèle qui explique les données en utilisant peu d'attributs : outre qu'un tel modèle est souvent plus facile à interpréter, il possède l'avantage de sélectionner les attributs les plus significatifs (on parle parfois de *feature selection*). Du point de vue de l'optimisation, si le modèle en question est représenté par un vecteur $\boldsymbol{x} \in \mathbb{R}^d$, on cherche un vecteur qui soit solution du problème d'optimisation mais possède autant de coordonnées nulles que possible (on dira que l'on cherche un vecteur creux, ou parcimonieux).

Il existe des termes de régularisation qui pénalisent les vecteurs avec des composantes non nulles (et non les composantes globalement larges, comme pour la régularisation écrêtée). La fonction norme ℓ_0 est une expression exacte de cette pénalisation[1]. Ainsi, un problème avec régularisation ℓ_0 est de la forme

$$\underset{\boldsymbol{x}}{\text{minimiser}}\, f(\boldsymbol{x}) + \lambda \|\boldsymbol{x}\|_0, \quad \|\boldsymbol{v}\|_0 = |\{i | [\boldsymbol{v}]_i \neq 0\}|.$$

Cependant, cette fonction est non convexe, non lisse et discontinue; elle possède également une nature combinatoire qui la rend complexe à utiliser en optimisation. On lui préfère donc en général la norme ℓ_1, définie par

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{d} |w_i|. \tag{3.1.1}$$

Cette fonction est non lisse, mais elle est continue et convexe; il s'agit également d'une norme, ce qui lui confère de nombreuses propriétés intéressantes en optimisation.

Exemple 3.1.2 *LASSO (Least Absolute Shrinkage and Selection Operator) On se place dans le cadre de la régression linéaire sur des données $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ et $\boldsymbol{y} \in \mathbb{R}^n$. Le problème des moindres carrés linéaires avec régularisation ℓ_1 s'écrit :*

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimiser}}\, \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \lambda \|\boldsymbol{x}\|_1.$$

On peut montrer que la solution de ce problème possède moins d'éléments non nuls que la solution du problème sans terme de régularisation, donnée par la pseudo-inverse.

3.1.3 Méthodes proximales

Nous allons maintenant décrire une classe d'algorithmes dédiée à la résolution de problèmes d'optimisation avec régularisation. Dans ces notes, on s'intéressera plus précisément à la catégorie ci-dessous.

Définition 3.1.1 (Optimisation composite) *Un problème d'optimisation composite est de la forme*

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimiser}}\, f(\boldsymbol{x}) + \lambda \Omega(\boldsymbol{x}),$$

où $f : \mathbb{R}^d \to \mathbb{R}$ est une fonction lisse et $\mathcal{C}^{1,1}$, $\lambda > 0$, et $\Omega : \mathbb{R}^d \to \mathbb{R}$ est une fonction convexe non lisse.

[1] Communément appelée "norme ℓ_0" ainsi même s'il ne s'agit pas d'une norme.

Cette définition couvre nos deux exemples de la partie précédente. Pour les résoudre, nous allons adopter une approche dite proximale.

L'**approche proximale** procède selon un schéma très fréquent en optimisation : un problème donné est remplacé par une suite de sous-problèmes présumés plus faciles à résoudre[2]. Dans le cas des méthodes proximales, on exploite la dérivabilité de f pour obtenir un problème plus simple, tandis que la structure de Ω est incorporée dans les sous-problèmes.

Algorithme 5: Méthode du gradient proximal

Initialisation : $\boldsymbol{x}_0 \in \mathbb{R}^d$.

Pour $k = 0, 1, \ldots$

1. Calculer le gradient de la partie lisse du problème $\nabla f(\boldsymbol{x}_k)$.
2. Définir une taille de pas $\alpha_k > 0$.
3. Calculer le nouvel itéré $\boldsymbol{x}_{k+1}$ tel que

$$\boldsymbol{x}_{k+1} \in \operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{R}^d} \left\{ f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}_k) + \tfrac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|^2 + \lambda\Omega(\boldsymbol{x}) \right\}. \tag{3.1.2}$$

Fin

L'algorithme 5 décrit le déroulement d'une méthode proximale. Le coût de chaque itération est plus élevé que celui des méthodes que nous avons vues jusqu'à présent, car une itération comporte un calcul de gradient puis une résolution d'un sous-problème auxiliaire (3.1.2), que l'on appelle sous-problème proximal.

Remarque 3.1.2 *Si $\Omega \equiv 0$ (Ω est la fonction nulle et il n'y a pas de régularisation), on peut montrer que la solution de (3.1.2) est unique, et donnée par*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k).$$

On reconnaît ainsi l'itération de la descente de gradient décrite par l'algorithme 1.

Les méthodes de gradient proximal peuvent être munies de la plupart des outils qui peuvent être utilisés pour la descente de gradient : différentes tailles de pas, accélération, utilisation de gradients stochastiques, etc. Par ailleurs, il existe de nombreux résultats de complexité pour les méthodes proximales, principalement pour f convexe mais aussi dans le cas non convexe.

Exemple de méthode proximale : ISTA Pour terminer cette partie, nous présentons une instance de l'algorithme 5 très populaire en traitement du signal, qui permet de calculer des représentations parcimonieuses. Cette méthode résout des problèmes lisses régularisés par une norme ℓ_1, donc de la forme

$$\operatorname*{minimiser}_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) + \lambda\|\boldsymbol{x}\|_1$$

[2] Toutes les méthodes vues dans ce cours suivent implicitement cette approche.

avec f de classe $\mathcal{C}^1$. La forme du terme de régularisation permet de caractériser directement la solution du sous-problème (3.1.2). En effet, le sous-problème proximal donné par

$$\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimiser}} \left\{ f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}_k) + \tfrac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{x}_k\|^2 + \lambda\|\boldsymbol{x}\|_1 \right\},$$

possède une unique solution. Pour l'obtenir, on calcule le pas classique de l'algorithme de descente de gradient $\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$, puis on applique une fonction de *seuillage faible* (ou **soft-thresholding**) notée $s_{\alpha_k\lambda}(\bullet)$ à chacune des composantes. Cette fonction est définie par

$$\forall \mu > 0, \forall t \in \mathbb{R}, \qquad s_\mu(t) = \begin{cases} t+\mu & \text{si } t < -\mu \\ t-\mu & \text{si } t > \mu \\ 0 & \text{sinon.} \end{cases}$$

Ainsi, la solution du sous-problème proximal est définie composante à composante en fonction du pas de la descente de gradient. Ce résultat est au coeur de la méthode proximale pour les problèmes avec régularisation ℓ_1, appelée ISTA (pour *Iterative Soft-Thresholding Algorithm*) : une description de cette méthode est donnée par l'algorithm 6.

Algorithme 6: ISTA: Iterative Soft-Thresholding Algorithm.

Initialisation : $\boldsymbol{x}_0 \in \mathbb{R}^d$.
Pour $k = 0, 1, ...$

1. Calculer le gradient de la partie lisse du problème $\nabla f(\boldsymbol{x}_k)$.
2. Définir une taille de pas $\alpha_k > 0$.
3. Calculer le nouvel itéré $\boldsymbol{x}_{k+1}$ composante par composante selon la formule :

$$[\boldsymbol{x}_{k+1}]_i = \begin{cases} [\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k)]_i + \alpha_k\lambda & \text{si } [\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k)]_i < -\alpha_k\lambda \\ [\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k)]_i - \alpha_k\lambda & \text{si } [\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k)]_i > \alpha_k\lambda \\ 0 & \text{si } [\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k)]_i \in [-\alpha_k\lambda, \alpha_k\lambda]. \end{cases} \tag{3.1.3}$$

Fin

La définition de la fonction de "soft-thresholding" force certaines des composantes du nouvel itéré à être nulles, ce qui produira au final une solution plus parcimonieuse que pour un problème non régularisé.

FISTA Les méthodes de gradient proximales peuvent elles aussi être accélérées lorsqu'elles sont appliquées à un problème convexe. Dans le cas particulier de la régularisation ℓ_1, on obtient un algorithme appelé FISTA (pour *Fast ISTA*). Il s'agit aujourd'hui de la variante d'ISTA la plus utilisée.

3.2 Méthode du gradient stochastique

3.2.1 Motivation

On suppose ainsi que l'on dispose d'un échantillon de n exemples sous la forme $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ où $\boldsymbol{x}_i \in \mathbb{R}^d$ et $y_i \in \mathbb{R}$ sont obtenus à partir d'une distribution de données. Comme dans l'exemple de la régression linéaire, on recherche un modèle h tel que $h(\boldsymbol{x}_i) \approx y_i$ pour tout $i = 1, \ldots, n$. On supposera ici qu'un modèle est paramétré par un vecteur $\boldsymbol{x} \in \mathbb{R}^d$ (c'est-à-dire $h(\boldsymbol{x}_i) = h(\boldsymbol{x}; \boldsymbol{x}_i)$), de sorte qu'il suffit de déterminer le vecteur $\boldsymbol{x}$ pour définir le modèle. Afin de quantifier la capacité du modèle à représenter nos données, on définit une fonction de coût (ou de perte) de la forme $\ell : (h, y) \mapsto \ell(h, y)$. Le but de cette fonction est de pénaliser des valeurs (h, y) telles que $h \neq y$. La fonction $(h, y) \mapsto (h - y)^2$, que nous avons déjà rencontrée dans le contexte des moindres carrés, est un exemple d'une fonction de coût pour les modèles linéaires. Avec cette fonction de coût, la perte en un échantillon donné est $\ell(h(\boldsymbol{x}; \boldsymbol{x}_i), y_i)$: on souhaite que notre modèle soit le meilleur relativement à l'ensemble des exemples. On considère ainsi la moyenne des pertes, ce qui conduit au problème d'optimisation suivant.

Définition 3.2.1 (Problème d'optimisation en somme finie) *Soit un jeu de données $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ où $\boldsymbol{x}_i \in \mathbb{R}^d$ et $y_i \in \mathbb{R}$,une classe de modèles $\{h(\boldsymbol{x}; \cdot)\}_{\boldsymbol{x} \in \mathbb{R}^d}$ et une fonction de coût ℓ. On définit le problème d'optimisation en somme finie suivant :*

$$\underset{\boldsymbol{x} \in \mathbb{R}^d}{\text{minimiser}}\, f(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^n \ell(h(\boldsymbol{x}; \boldsymbol{x}_i), y_i) = \frac{1}{n} \sum_{i=1}^n f_i(\boldsymbol{x}). \tag{3.2.1}$$

Supposons que l'on applique l'algorithme 1 de descente de gradient à ce problème, en supposant que tous les f_i sont de classe $\mathcal{C}^1$ (donc que f l'est). La k-ième itération de cet algorithme s'écrira

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) = \boldsymbol{x}_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(\boldsymbol{x}_k). \tag{3.2.2}$$

On voit ainsi que chaque itération de descente de gradient requiert l'accès à **l'ensemble des données** pour effectuer un calcul de gradient. Dans un contexte de données massives, le nombre d'exemples n peut être extrêmement large, et cet algorithme peut être trop coûteux pour être utilisé en pratique.

Remarque 3.2.1 *Dans un contexte stochastique ou "online", les examples peuvent être directement générés de la distribution à la volée. Dans ce contexte, il peut être impossible d'effectuer une moyenne discrète sur les échantillons, et donc d'obtenir un problème sous la forme d'une somme finie. On peut néanmoins formuler le problème en utilisant une espérance mathématique, et ainsi chercher à résoudre :*

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} \mathbb{E}_{(\boldsymbol{x}, y)} \left[f_{(\boldsymbol{x}, y)}(\boldsymbol{x}) \right]. \tag{3.2.3}$$

Ce problème est bien souvent le véritable but de l'apprentissage automatique, mais est souvent remplacé par (3.2.1) pour tenir compte de l'échantillon fini de données à disposition. Par ailleurs, le gradient de cette fonction objectif peut être compliqué voire impossible à calculer, ce qui exclut d'emblée l'utilisation d'algorithmes tels que la descente de gradient. La méthode du gradient stochastique sera au contraire applicable dans ce cas.

3.2.2 Algorithme

L'idée derrière l'algorithme du gradient stochastique est remarquablement simple. Si l'on considère le problème $\text{minimiser}_{\boldsymbol{x}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^n f_i(\boldsymbol{x})$ sous l'hypothèse que chaque fonction f_i est dérivable, chaque itération consiste à choisir un indice i_k au hasard et à faire un pas dans la direction opposée au gradient de la fonction f_{i_k}. L'algorithme 7 décrit ce processus.

Algorithme 7: Méthode du gradient stochastique.

Initialisation: $\boldsymbol{x}_0 \in \mathbb{R}^d$.
for $k = 0, 1, \ldots$ **do**

1. Calculer un pas $\alpha_k > 0$.
2. Tirer un indice $i_k \in \{1, \ldots, n\}$.
3. Calculer le nouvel itéré :

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f_{i_k}(\boldsymbol{x}_k). \tag{3.2.4}$$

end

Le vecteur $\nabla f_{i_k}(\boldsymbol{x}_k)$ s'appelle un **gradient stochastique** au point $\boldsymbol{x}_k$. La propriété principale de l'itération (3.2.4) est qu'elle n'accède qu'à un seul exemple du jeu de données : par conséquent, son coût en termes d'accès aux données est n **fois inférieur à celui d'une itération de descente de gradient** (3.2.2).

Remarque 3.2.2 *En règle générale, et même si le problème d'optimisation s'écrit sous la forme d'une somme finie, l'algorithme 7 peut diverger. À titre d'exemple, considérons le problème*

$$\underset{w\in\mathbb{R}}{\text{minimiser}}\ \frac{1}{2}(f_1(w) + f_2(w))$$

avec $f_1(w) = 2w^2$ et $f_2 = -w^2$. Si $w_0 > 0$ et $i_k = 2$ pour tout k, alors l'algorithme divergera.

Il est aisé de construire des exemples pour lesquels la méthode du gradient stochastique diverge ainsi. En pratique, cependant, pour les problèmes de somme finie issus de l'apprentissage automatique, les données sont suffisamment **corrélées** *pour qu'une mise à jour relativement à un example affecte la prédiction sur d'autres exemples. Cette observation explique en partie le succès des méthodes de gradient stochastique dans ce contexte.*

Remarque 3.2.3 *L'algorithme 7 est souvent désigné par l'acronyme SGD, pour* **Stochastic Gradient Descent**, *ou* descente de gradient stochastique. *Cela étant, il n'est pas possible de garantir que l'algorithme du gradient stochastique est une méthode de descente (qui décroît la fonction objectif à chaque itération). Pour cette raison, dans ce document, nous utiliserons la terminologie (par ailleurs adoptée chez d'autres auteurs) de* **gradient stochastique** *(mais pourrons nous autoriser l'utilisation de l'acronyme SGD, qui se retrouve dans de nombreuses implémentations).*

3.2.3 Élements d'analyse

Dans cette section, on décrit les étapes principales de l'obtention de vitesses de convergence pour l'algorithme du gradient stochastique, sous une version légèrement modifiée de l'hypothèse 2.1.1.

Hypothèse 3.2.1 *La fonction objectif $f = \frac{1}{n}\sum_{i=1}^{n} f_i$ du problème (3.2.1) est de classe $\mathcal{C}_L^{1,1}(\mathbb{R}^d)$ pour $L > 0$, et minorée sur $\mathbb{R}^d$ par $f_{low} \in \mathbb{R}$. Par ailleurs, chaque fonction f_i est de classe $\mathcal{C}^1$.*

L'argument principal de l'analyse de la descente de gradient est le résultat de la proposition 2.1.2, que l'on rappelle ci-dessous :

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \frac{L}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|^2.$$

Il est possible d'en obtenir une version équivalente pour l'algorithme du gradient stochastique, sous des hypothèses sur le tirage aléatoire des indices, et donc des gradients "stochastiques" associés. Ces conditions sont résumées dans l'hypothèse suivante.

Hypothèse 3.2.2 (Hypothèse sur les gradients stochastiques) *À chaque itération de l'algorithme 7 d'indice k, l'indice i_k est tiré tel que :*

i) L'indice i_k est indépendant de $i_0, \ldots, i_{k-1}$;

ii) $\mathbb{E}_{i_k}\left[\nabla f_{i_k}(\boldsymbol{x}_k)\right] = \nabla f(\boldsymbol{x}_k)$;

iii) $\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\boldsymbol{x}_k)\|^2\right] \leq \sigma^2 + \|\nabla f(\boldsymbol{x}_k)\|^2$ avec $\sigma^2 > 0$.

La propriété i) ci-dessus est une hypothèse simplificatrice, qui permet La propriété ii) de l'hypothèse 3.2.2 impose que le gradient stochastique $\nabla f_{i_k}(\boldsymbol{x}_k)$ soit un estimateur sans biais du véritable gradient $\nabla f(\boldsymbol{x}_k)$. La propriété iii) contrôle la variance de la norme de ce gradient stochastique, de sorte à garantir que la variation aléatoire de celui-ci soit contrôlée. Il existe plusieurs manières de générer aléatoirement des indices vérifiant ces propriétés, au premier rang desquelles le tirage uniforme.

Exemple 3.2.1 (Tirage uniforme) *Si à l'itération k, l'indice i_k est tiré uniformément dans $\{1, \ldots, n\}$, alors l'algorithme 7 vérifie l'hypothèse 3.2.2.*

Proposition 3.2.1 *Sous les hypothèses 2.1.1 et 3.2.2, on considère la k-ième itération de l'algorithme 7. Alors, on a*

$$\mathbb{E}_{i_k}\left[f(\boldsymbol{x}_{k+1})\right] - f(\boldsymbol{x}_k) \leq \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}\,\mathbb{E}_{i_k}\left[\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right] + \frac{L}{2}\,\mathbb{E}_{i_k}\left[\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|^2\right]. \tag{3.2.5}$$

Comme le montre l'inégalité ci-dessus, on ne peut garantir la décroissance d'une itération à l'autre qu'**en moyenne** (sous réserve que le membre de droite de (3.2.5) soit négatif). Cependant, une telle propriété suffit pour obtenir des vitesses de convergence (ou des bornes de complexité) pour le gradient stochastique appliqué aux problèmes non convexes, convexes ou fortement convexes. Ces résultats sont fortement dépendants du choix des tailles de pas $\{\alpha_k\}_k$: ce choix est de fait un enjeu majeur en apprentissage, qui correspond à la calibration du taux d'apprentissage (ou *learning rate*).

On présente ci-dessous les différentes possibilités pour un tel choix, et leur impact sur les garanties théoriques, dans le cadre des fonctions fortement convexes. Nous ferons donc l'hypothèse suivante.

Hypothèse 3.2.3 *La fonction objectif f est μ-fortement convexe et possède un unique minimum global $\boldsymbol{x}^*$; on notera $f^* = f(\boldsymbol{x}^*)$.*

On considère d'abord le cas d'une taille de pas constante, pour lequel on peut établir le résultat suivant.

Théorème 3.2.1 (Gradient stochastique à taille de pas constante) *Sous les hypothèses 2.1.1, 3.2.2 et 3.2.3, supposons que l'on applique l'algorithme 7 avec une taille de pas constante*

$$\alpha_k = \alpha \in (0, \tfrac{1}{2\mu}) \forall k.$$

Alors,

$$\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \frac{\alpha L \sigma^2}{2\mu} + (1 - 2\alpha\mu)^k \left[f(\boldsymbol{x}_0) - f^* - \frac{\alpha L \sigma^2}{2\mu}\right]. \tag{3.2.6}$$

Comme on le voit, l'algorithme du gradient stochastique converge en espérance en vitesse linéaire avec une taille de pas constante, tout comme l'algorithme de descente de gradient : cette vitesse signifie que pour garantir $\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \epsilon$, la méthode requiert au plus $\mathcal{O}\left(\ln(1/\epsilon)\right)$ itérations. Cependant, ce résultat n'est pas valide pour tout $\epsilon > 0$, contrairement au cas de la descente de gradient. En effet, le résultat (3.2.6) inclut un terme constant dans son second membre, qui représente un biais de $\frac{\alpha L \sigma^2}{4\mu}$. Cela signifie que le théorème 3.2.1 ne peut garantir la convergence que vers un **voisinage** de l'optimum f^*. Dans le même temps, l'utilisation d'une taille de pas constante permet de profiter de pas prometteurs.

Dans sa version originelle (proposée par Robbins et Monro en 1951), la suite des longueurs de pas devait vérifier

$$\sum_{k=0}^{\infty} \alpha_k = \infty \quad \text{et} \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

Pour vérifier ces hypothèses, il est nécessaire que α_k tende vers 0. On s'intéresse donc maintenant aux variantes du gradient stochastique basées sur une taille de pas décroissante.

Théorème 3.2.2 (Gradient stochastique à taille de pas décroissante) *Sous les hypothèses 2.1.1, 3.2.2 et 3.2.3, on considère l'algorithme 7 appliqué avec une taille de pas décroissante de la forme*

$$\alpha_k = \frac{\beta}{k + \gamma},$$

où $\beta > \frac{1}{\mu}$ et $\gamma > 0$ est choisi de sorte que $\alpha_0 = \frac{\beta}{\gamma} \leq \frac{\mu}{L}$. Dans ce cas, on a

$$\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \frac{\nu}{\gamma + k}, \tag{3.2.7}$$

avec

$$\nu = \max\left\{\frac{\beta^2 L \sigma^2}{2(\beta\mu - 1)}, (\gamma + 1)(f(\boldsymbol{x}_0) - f^*)\right\}.$$

Comme dans le cas de l'algorithme de descente de gradient, le choix d'une taille de pas décroissante peut poser problème, dans la mesure où les pas deviennent nécessairement de plus en plus petits. Par ailleurs, la vitesse de convergence établie par (3.2.7) est sous-linéaire (en $\mathcal{O}(\frac{1}{k})$), ce qui est plus lent que la vitesse linéaire (en $\mathcal{O}((1-t)^k)$) obtenue par une taille de pas constante; en revanche, avec une taille de pas décroissante, la méthode peut satisfaire $\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \epsilon$ pour toute valeur $\epsilon > 0$.

Remarque 3.2.4 (Une approche hybride) *En apprentissage, une stratégie hybride entre la taille de pas constante et la taille de pas décroissante est souvent utilisée. Celle-ci consiste à lancer l'algorithme avec une taille de pas constante égale à α jusqu'à décroître $f(\boldsymbol{x}_k) - f^*$ en dessous d'un certain seuil (typiquement égal à $\frac{\alpha L \sigma^2}{2\mu}$). On choisit ensuite $\alpha' < \alpha$, et on continue l'algorithme avec ce nouveau pas, en visant la précision $\frac{\alpha' L \sigma^2}{2\mu}$. En procédant par réductions successives de α, on peut ainsi atteindre n'importe quelle précision, tout en conservant un pas constant durant plusieurs itérations. On peut montrer que la vitesse de convergence d'un tel processus est sous-linéaire, en ce sens que pour tout $\epsilon > 0$,*

$$\mathbb{E}\left[f(\boldsymbol{x}_k) - f^*\right] \leq \epsilon \quad \textit{after} \quad \mathcal{O}(1/\epsilon) \textit{ itérations.}$$

Pour appliquer cette stratégie de manière adaptive, il est nécessaire de déterminer quand le voisinage est atteint; en pratique, on regarde si l'algorithme stagne, c'est-à-dire ne progresse plus (cela peut être quantifié en termes d'itérés, de norme du gradient stochastique, etc).

Taille de pas dans le cas nonconvexe L'algorithme du gradient stochastique (et ses variantes) sont les méthodes les plus fréquemment utilisées pour l'entraînement des réseaux de neurones : ce problème est fortement non convexe, et l'analyse ci-dessus ne peut donc y être appliquée. Cependant, il est possible de déterminer des vitesses de convergence pour le cas convexe, en étudiant les quantités suivantes :

- $\mathbb{E}\left[\frac{1}{K}\sum_{i=1}^{K}\|\nabla f(\boldsymbol{x}_k)\|^2\right]$ pour les variantes de l'algorithme 7 utilisant une taille de pas constante;
- $\mathbb{E}\left[\frac{1}{\sum_{i=1}^{K}\alpha_k}\sum_{i=1}^{K}\alpha_k\|\nabla f(\boldsymbol{x}_k)\|^2\right]$ pour les variantes utilisant une taille de pas décroissante.

De par l'utilisation d'approximations du gradient, on obtiendra des bornes qui seront moins bonnes que celles du cas déterministe. À titre d'exemple, l'algorithme 7 appliqué avec une longueur de pas constante vérifiera

$$\mathbb{E}\left[\frac{1}{K}\sum_{i=1}^{K}\|\nabla f(\boldsymbol{x}_k)\|^2\right] \leq \epsilon$$

en au plus $\mathcal{O}(\epsilon^{-4})$ itérations, ce qui est une plus mauvaise borne que dans le cas déterministe, en $\mathcal{O}(\epsilon^{-2})$. De plus, ce résultat ne s'appliquera que pour une valeur de ϵ suffisamment large, à cause du biais introduit par l'utilisation de quantités stochastiques.

Remarque 3.2.5 (Utilisation du momentum) *Les variantes du gradient stochastique les plus populaires en pratique (*ADAM, ADAGRAD, RMSPROP*) incorporent généralement un terme de momentum dans l'itération du gradient stochastique. L'idée sous-jacente est que l'ajout de momentum permet d'accumuler les pas dans des bonnes directions (qui sont souvent de descente pour l'algorithme), tandis que les mauvaises directions (correspondant par exemple aux* outliers, *ou données aberrantes) se compenseront au fil du temps. L'analyse de ces méthodes accélérées est cependant beaucoup plus complexe que celle du cas déterministe, et à l'heure actuelle la théorie et la pratique sont encore décorrélées, contrairement au cas déterministe.*

3.2.4 Variantes par fournées (batch)

Comme on l'a vu dans la partie précédente, l'itération de l'algorithme du gradient stochastique se ramène à

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f_{i_k}(\boldsymbol{x}_k),$$

où i_k est un indice tiré au hasard dans $\{1, \ldots, n\}$. Cette méthode utilise ainsi un **seul** exemple pour construire son gradient stochastique, ce qui peut conduire à une variabilité importante entre différentes exécutions de la méthode (rerprésentée en théorie par la variance σ^2 associée au gradient stochastique). A contrario, l'algorithme de descente de gradient utilise tous les exemples et le résultat possède une variance égale à zéro (car il s'agit d'une quantitié déterministe).

Si l'on cherche à améliorer cette variance, il semble donc naturel de considérer des estimateurs stochastiques du gradient basés sur plusieurs exemples à la fois : c'est le principe des méthodes de fournées[3] de gradients stochastiques.

Le principe d'une telle méthode est de tirer un ensemble aléatoire d'indices $S_k \subset \{1, \ldots, n\}$ puis d'effectuer l'itération suivante :

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(\boldsymbol{x}_k) \tag{3.2.8}$$

Lorsque $|S_k| = 1$, on retrouve la formule du gradient stochastique classique. En considérant $|S_k| = n$, on aurait nécessairement $S_k = \{1, \ldots, n\}$, auquel cas l'itération (3.2.8) est équivalente à celle de la descente de gradient.

Plus globalement, on peut identifier deux classes de tailles de fournées :

- $|S_k| \approx n$: une telle variante possède un coût par itération proche de celui de la descente de gradient, et par conséquent est soumise aux mêmes problématiques de coût;
- $|S_k| = n_b << n$, que l'on appelle mini-fournée (ou mini-batching), qui peut être un choix avantageux sur le plan théorique, raisonnable en pratique et permet de réduire la variance. Cette méthode est communément appelée l'algorithme de mini-fournées de gradient stochastique, ou mini-batch SG.

Si l'on suppose que la taille de fournée est constante, c'est-à-dire que $|S_k| = n_b \ \forall k$, on peut alors montrer que, pour la même taille de pas, la variante avec mini-lot requiert n_b fois moins d'itérations que le gradient stochastique. Ces dernières sont n_b fois plus coûteuses, mais l'approche par mini-fournées permet d'exploiter le calcul parallèle, en calculant les n_b gradients stochastiques sur différents processeurs. Par ailleurs, l'estimé de gradient stochastique vérifie la propriété suivante.

Proposition 3.2.2 *Sous les hypothèses 2.1.1 et 3.2.2, la variance du gradient stochastique en "mini-lot" vérifie :*

$$\mathbb{E}_{S_k}\left[\|\tfrac{1}{|S_k|}\sum_{i \in S_k} \nabla f_i(\boldsymbol{x}_k)\|^2\right] \leq \tfrac{\sigma^2}{n_b}.$$

Pour terminer cette partie, nous mentionnons que les techniques par (mini-)fournées restent plus coûteuses (au niveau d'une itération), tout en étant plus sensibles aux redondances dans les données;

[3] Ou *batch*, en anglais.

par ailleurs, la taille de la fournée (qui peut être fixe ou adaptative) introduit un hyperparamètre supplémentaire à choisir. Ces considérations expliquent en partie que l'approche classique du gradient stochastique reste la plus fréquente en pratique.

3.2.5 Autres variantes basées sur la réduction de variance

Comme nous l'avons vu en Section 3.2.3, la théorie du gradient stochastique repose sur l'hypothèse 3.2.2, et plus particulièrement sur un contrôle de la variance de la norme du gradient stochastique (à travers la quantité σ^2). En observant la dépendance en σ des vitesses de convergence telles que (3.2.6), on voit qu'une valeur élevé de σ conduit à de plus mauvaises bornes. Cela se traduit numériquement par le fait qu'une méthode avec une forte variance risque de converger lentement.

Les techniques dites de réduction de variance ont été précisément développées dans le but de diminuer la variance des estimations de gradient utilisées par l'algorithme du gradient stochastique. On peut classer ces techniques en deux groupes, selon qu'elles utilisent plusieurs exemples à chaque itération (comme une approche par fournées) ou qu'elles se basent sur les itérations précédentes. Nous nous concentrerons ici sur les stratégies relevant de la première catégorie.

Les méthodes d'agrégation (aggregation gradient) sont d'autres stratégies de réduction de variance dont les propriétés théoriques (et notamment leurs vitesses de convergence linéaires) ont fait le succès dans la communauté de l'optimisation et de la théorie de l'apprentissage. Elles consistent en substance à effectuer un **pas complet de descente de gradient** à intervalles réguliers durant l'exécution de l'algorithme : cela permet de corriger les composantes du gradient stochastique qui possèderaient une variance trop élevée. En dépit de leur analyse théorique très fouillée, les variantes de ces méthodes ne sont pas encore déployées en pratique, principalement en raison du coût prohibitif que peut représenter un calcul de gradient, même s'il n'est effectué qu'une fois toutes les K itérations (avec $K \gg 1$).

Faire la moyenne des itérés est une autre manière de réduire la variance, moins coûteuse à implémenter que la précédente. L'idée sous-jacente consiste à étudier les propriétés de l'itéré moyen $\frac{1}{K}\sum_{k=0}^{K-1} \boldsymbol{x}_k$: dans certains cas (par exemple lorsque $\alpha = \frac{1}{\mu(k+1)}$ et f est μ-fortement convexe), on peut montrer de bonnes propriétés pour cet itéré, notamment qu'il s'agit d'une solution plus robuste que le dernier itéré obtenu. Cependant, afin de renvoyer cet itéré moyen, il est nécessaire soit de stocker l'historique des itérés (ce qui peut être coûteux), soit de maintenir un itéré moyen, et cela peut générer des erreurs numériques.

3.3 Méthodes de gradient stochastique pour l'apprentissage profond

Dans cette partie, on s'intéresse aux techniques de gradient stochastique utilisées pour entraîner des modèles d'apprentissage profond. On considère toujours un problème de la forme (3.2.1), sous l'hypothèse 3.2.1. Notre but est d'analyser différentes variations du schéma

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{g}_k, \tag{3.3.1}$$

où $\alpha > 0$ est une taille de pas/un taux d'apprentissage (*learning rate*) et $\boldsymbol{g}_k$ est un estimateur stochastique du gradient correspondant à un seul indice (comme le gradient stochastique) ou à un paquet (*batch*) d'indices.

On s'intéressera par la suite à un schéma générique qui couvre toutes les variantes que nous allons étudier. Ce schéma est de la forme :

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{m}_k \oslash \boldsymbol{v}_k, \tag{3.3.2}$$

où $\alpha > 0$, $\boldsymbol{m}_k, \boldsymbol{v}_k \in \mathbb{R}^d$ et $\oslash$ symbolise la division composante à composante :

$$\boldsymbol{m}_k \oslash \boldsymbol{v}_k := \left[\frac{[\boldsymbol{m}_k]_i}{[\boldsymbol{v}_k]_i} \right]_{i=1,\dots,d}.$$

Cette itération généralise (3.3.1) : en effet, en posant $\boldsymbol{m}_k = \boldsymbol{g}_k$ et $\boldsymbol{v}_k = \mathbf{1}_{\mathbb{R}^d}$, on retrouve l'itération 3.3.1. Ce formalisme permet d'exprimer de façon unifiée les méthodes de gradient stochastique les plus populaires en apprentissage profond, et ainsi de mieux les comparer.

3.3.1 Gradient stochastique avec momentum

Dans la lignée des techniques accélérées que nous avons vues au section 2.3, la plupart des implémentations de gradient stochastique considèrent l'addition de momentum au pas basique (3.3.1). Ainsi, une itération de gradient stochastique avec momentum s'écrira

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha(1-\beta)\boldsymbol{g}_k + \alpha\beta\left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\right), \tag{3.3.3}$$

où $\beta \in (0,1)$ est un paramètre constant (choisir $\beta = 0$ reviendrait à faire du gradient stochastique). Il s'agit d'une version de l'algorithme de Polyak pour laquelle la direction de type gradient est combinée avec la direction précédente : on retrouve l'idée selon laquelle l'utilisation du momentum permet d'accumuler l'information de l'itération précédente. En pratique, on observe que l'itération (3.3.3) tend à accumuler les bonnes directions (en termes d'optimisation), tandis que les "mauvaises" directions (et donc les "mauvais" pas) pour l'optimisation ont tendance à se compenser.

L'itération (3.3.3) est un cas particulier de (3.3.2), correspondant à $\boldsymbol{v}_k = \mathbf{1}_{\mathbb{R}^d}$ et $\boldsymbol{m}_k$ défini de manière récursive par $\boldsymbol{m}_{-1} = \mathbf{0}_{\mathbb{R}^d}$ et

$$\boldsymbol{m}_k = (1-\beta)\boldsymbol{g}_k - \beta \boldsymbol{m}_{k-1} \quad \forall k \in \mathbb{N}.$$

où $\beta \in (0,1)$.

La méthode de gradient stochastique avec momentum est implémentée dans les bibliothèques d'apprentissage profond telles que PyTorch. Elle est particulièrement dans l'entraînement de réseaux de neurones profonds sur des problèmes de vision par ordinateur, et est à l'origine de la montée du "deep learning" au début des années 2010.

Remarque 3.3.1 *Les garanties de l'algorithme* (3.3.3) *sont plus difficiles à établir que dans le cas de la descente de gradient accélérée : les approches par momentum ont cependant rencontré un certain succès pratique, même sur des problèmes non convexes tels que l'entraînement de réseaux de neurones.*

3.3.2 AdaGrad

La méthode de gradient adaptatif, ou AdaGrad, a été proposée en 2011 pour répondre à la difficulté du choix de α dans le gradient stochastique tout en évitant d'avoir recours à des procédures adaptatives coûteuses telles que la recherche linéaire. L'approche d'AdaGrad consiste à normaliser

chaque composante du gradient stochastique au moyen d'une accumulation des valeurs de chaque composante au cours des itérations. L'algorithme maintient donc une suite $\{\boldsymbol{r}_k\}_k$ définie par

$$\forall i = 1, \ldots, d, \quad \left\{ \begin{array}{l} [\boldsymbol{r}_{-1}]_i = 0 \\ [\boldsymbol{r}_k]_i = [\boldsymbol{r}_{k-1}]_i + [\boldsymbol{g}_k]_i^2 \quad \forall k \geq 0, \end{array} \right. \tag{3.3.4}$$

L'itération d'AdaGrad s'écrit alors

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{g}_k \oslash \sqrt{\boldsymbol{r}_k}, \tag{3.3.5}$$

où la racine carrée est appliquée à toutes les composantes de $\boldsymbol{r}_k$. On reconnaît ainsi l'itération générique (3.3.2) avec $\boldsymbol{m}_k = \boldsymbol{g}_k$ et $\boldsymbol{v}_k = \sqrt{\boldsymbol{r}_k}$. L'innovation d'AdaGrad tient donc non pas dans l'introduction de momentum, mais dans l'utilisation d'une suite de pas différenciée pour chaque coordonnée, de la forme

$$\left\{ \left[\frac{\alpha}{\sqrt{[\boldsymbol{r}_k]_i}} \right]_{i=1}^{d} \right\}_k .$$

On opère ainsi une normalisation diagonale (*diagonal scaling*) des composantes du gradient stochastique $\boldsymbol{g}_k$, qui permet notamment de traiter le cas de composantes très différentes en amplitude sans avoir à calibrer α très finement. Cependant, de tels pas tendent généralement vers 0 assez rapidement.

Remarque 3.3.2 *En pratique, on remplace $\boldsymbol{r}_k$ par $\boldsymbol{r}_k + \eta \mathbf{1}_{\mathbb{R}^d}$ où $\eta > 0$ est de valeur faible, pour que l'algorithme soit numériquement plus stable.*

La méthode AdaGrad est particulièrement pertinente dans le cas de problèmes à gradients *parcimonieux* (ou *sparse* en anglais), pour lesquels les gradients stochastiques ont tendance à avoir beaucoup de composantes nulles. Dans une telle situation, le calcul de $\boldsymbol{r}_k$ permettra d'ajuster le pas uniquement pour les coordonnées non nulles. Les problèmes issus des systèmes de recommandation possèdent généralement cette propriété, et il s'agit du domaine où AdaGrad est généralement considéré comme l'algorithme référence.

3.3.3 RMSProp

L'algorithme RMSProp (pour *Root Mean Square Propagation*) procède de manière similaire à AdaGrad en jouant sur les composantes du gradient. La méthode repose sur une suite $\{\boldsymbol{r}_k\}_k$ définie par

$$\forall i = 1, \ldots, d, \quad \left\{ \begin{array}{l} [\boldsymbol{r}_{-1}]_i = 0 \\ [\boldsymbol{r}_k]_i = (1-\lambda)[\boldsymbol{r}_{k-1}]_i + \lambda [\boldsymbol{g}_k]_i^2 \quad \forall k \geq 0, \end{array} \right. \tag{3.3.6}$$

avec $\lambda \in (0, 1)$. On voit donc que la méthode peut donner (en fonction de la valeur de λ) plus de poids aux gradients précédents plutôt qu'au gradient de l'itération : cette idée permet aux longueurs de pas de décroître moins rapidement que celles de AdaGrad.

Comme pour AdaGrad, l'itération de RMSProp s'écrit alors sous la forme (3.3.2) avec $\boldsymbol{m}_k = \boldsymbol{g}_k$ et $\boldsymbol{v}_k = \sqrt{\boldsymbol{r}_k}$.

Remarque 3.3.3 *En pratique, et comme pour AdaGrad, on remplacera typiquement $\boldsymbol{r}_k$ par $\boldsymbol{r}_k + \eta \mathbf{1}_{\mathbb{R}^d}$ pour une petite valeur $\eta > 0$.*

La méthode RMSProp a été utilisée avec succès dans le cadre d'entraînements de réseaux de neurones profonds.

3.3.4 Adam

L'algorithme ADAM, proposé en 2013, peut être vu comme combinant le concept de momentum avec celui d'accumulation d'information sur les gradients précédents utilisés par les algorithmes précédents. Son itération correspond au schéma générique (3.3.2) avec

$$\boldsymbol{m}_k = \frac{(1-\beta_1)\sum_{j=0}^{k}\beta_1^{k-j}\boldsymbol{g}_j}{1-\beta_1^{k+1}} \tag{3.3.7}$$

et

$$\boldsymbol{v}_k = \sqrt{\frac{(1-\beta_2)\sum_{j=0}^{k}\beta_2^{k-j}\boldsymbol{g}_j \odot \boldsymbol{g}_j}{1-\beta_2^{k+1}}}. \tag{3.3.8}$$

avec $\beta_1, \beta_2 \in (0,1)$ et $\odot$ le produit composante à composante, dit de Hadamard :

$$\boldsymbol{g}_k \odot \boldsymbol{g}_k = \left[[\boldsymbol{g}_k]_i^2\right]_{i=1}^{d}.$$

Remarque 3.3.4 *En pratique, on utilisera $\boldsymbol{v}_k + \eta \mathbf{1}_{\mathbb{R}^d}$ pour une petite valeur $\eta > 0$ plutôt que $\boldsymbol{v}_k$.*

Les formules ci-dessous correspondent respectivement à une combinaison des directions précédemment employées qui met l'accent sur les directions les plus récentes, et à une normalisation des coordonnées des directions obtenues relativement à une moyenne de ces coordonnées donnant aussi plus d'importance aux dernières itérations. Cet aspect important, qui se justifie de manière statistique, semble être à l'origine du succès d'ADAM, dont la performance impressionnante sur de nombreuses tâches d'entraînement de réseaux de neurones a contribué à son utilisation massive. La méthode ADAM (et sa variante ADAMW, basée sur de la régularisation dont nous parlerons plus loin) sont notamment très efficaces sur des tâches de traitement automatique des langues.

3.4 Conclusion

L'une des structures les plus classiques d'un problème d'optimisation en sciences des données est celle d'un problème régularisé. Le but d'une régularisation est de favoriser les modèles possédant certaines propriétés structurelles : cela est réalisé par l'ajout d'un terme dans la fonction objectif, qui est typiquement indépendant des données. Lorsque la fonction objectif originelle est lisse et que le terme de régularisation est une fonction convexe, un problème d'optimisation avec régularisation peut être traité au moyen d'un algorithme de gradient proximal : ce dernier procède par résolution successive de sous-problèmes impliquant la fonction de régularisation. Dans le cas important de la régularisation ℓ_1, où l'on cherche à obtenir une solution parcimonieuse, l'algorithme de gradient proximal ISTA est la méthode la plus utilisée; pour des régularisations lisses, en revanche, il est possible d'appliquer des techniques d'optimisation lisses. C'est par exemple le cas avec la régularisation ℓ_2, dont le but est de réduire la variance de la solution par rapport à l'objectif.

Les méthodes de gradient stochastique reposent sur de l'information partielle du gradient, et ont donc de moins bonnes propriétés de convergence que l'algorithme de descente de gradient : en ce sens, elles ne sont pas forcément intéressantes pour un problème d'optimisation quelconque. En revanche, lorsque le calcul du gradient implique la manipulation d'un jeu de données très volumineux, les stratégies de gradient stochastique prennent tout leur sens, et se révèlent particulièrement adaptés pour de tels problèmes. Cela est dû au très faible coût de ces méthodes, mais aussi à leur efficacité sur des données possiblement aléatoires ou redondantes : dans un tel contexte, il est payant de privilégier les pas aléatoires et peu coûteux, et on observe ainsi une convergence nettement plus rapide en pratique.

Même si l'algorithme du gradient stochastique est très efficace en général, sa performance peut être sensiblement affectée par des gradients stochastiques à forte variance. Les implémentations de cette méthode cherchent généralement à en réduire la variance, en considérant par exemple des paquets d'exemples simultanément. Les variantes les plus populaires en pratique, et notamment en apprentissage profond, sont basées sur le principe d'accélération, ainsi que sur la normalisation diagonale des directions utilisées. Il est à noter que ces techniques peuvent manquer de justification théorique relativement aux problèmes auxquels elles sont appliquées (typiquement l'entraînement d'un réseau de neurones qui donne lieu à un problème non convexe). Cependant, ces méthodes ont été adoptées par la communauté du fait de leur succès pratique, et leur analyse complète reste un problème ouvert.

Bibliographie

[1] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, Belmont, MA, third edition, 2016.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3:1–122, 2010.

[3] S. J. Wright and B. Recht. *Optimization for Data Analysis*. Cambridge University Press, 2022.

Annexe A

Bases mathématiques

A.1 Éléments d'algèbre linéaire

Les fondements mathématiques de l'optimisation se trouvent dans l'analyse réelle, et en particulier dans le calcul différentiel. Les structures d'algèbre linéaire dans $\mathbb{R}^n$ jouent cependant un rôle prépondérant en optimisation (et en sciences des données). Cette section regroupe les résultats de base qui seront utilisés dans le cours.

Pour approfondir ces notions, on pourra consulter les liens suivants :

- https://www.ceremade.dauphine.fr/∼carlier/polyalgebre.pdf (en français);
- http://vmls-book.stanford.edu/vmls.pdf (Chapitres 1 à 3, en anglais).

A.1.1 Algèbre linéaire vectorielle

On considèrera toujours l'espace des vecteurs $\mathbb{R}^d$ muni de sa structure d'espace vectoriel normé de dimension d. On définit donc les opérations suivantes :

- Pour tous $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, la somme des vecteurs $\boldsymbol{x}$ et $\boldsymbol{y}$ est notée $\boldsymbol{x} + \boldsymbol{y} = [x_i + y_i]_{1 \leq i \leq d}$;
- Pour tout $\lambda \in \mathbb{R}$, on définit $\lambda \boldsymbol{x} \overset{n}{=} \lambda \cdot \boldsymbol{x} = [\lambda x_i]_{1 \leq i \leq d}$. Dans ce contexte, un tel réel λ sera appelé un *scalaire*.
- On notera $\mathbf{0}_d \in \mathbb{R}^d$ le vecteur dont toutes les coordonnées sont égales à 0.

A l'aide de ces opérations, nous pouvons donc construire des **combinaisons linéaires** de vecteurs de $\mathbb{R}^d$ dont le résultat sera un vecteur de $\mathbb{R}^d$, à savoir des vecteurs de la forme $\sum_{i=1}^{p} \lambda_i \boldsymbol{x}_i$, où $\boldsymbol{x}_i \in \mathbb{R}^d$ et $\lambda_i \in \mathbb{R}$ pour tout $i = 1, \ldots, p$.

Pour ce qui est de l'espace des matrices $\mathbb{R}^{n \times d}$, on peut également le munir d'une structure d'espace vectoriel normé de dimension nd. On définit donc les opérations suivantes :

- Pour tous $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n \times d}$, la somme des matrices $\boldsymbol{A}$ et $\boldsymbol{B}$ est notée $\boldsymbol{A} + \boldsymbol{B} = [\boldsymbol{A}_{ij} + \boldsymbol{B}_{ij}]_{\substack{1 \leq i \leq d \\ 1 \leq j \leq d}}$;
- Pour tout scalaire $\lambda \in \mathbb{R}$, on définit $\lambda \boldsymbol{A} \overset{n}{=} \lambda \cdot \boldsymbol{A} = [\lambda \boldsymbol{A}_{ij}]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq d}}$.
- La matrice nulle sera notée $\mathbf{0}$, pour simplifier.

Définition A.1.1 *Un ensemble $\mathcal{S} \subseteq \mathbb{R}^d$ vérifiant les conditions*

1. $\mathbf{0}_d \in \mathcal{S}$;
2. $\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S},\ \boldsymbol{x} + \boldsymbol{y} \in \mathcal{S}$;
3. $\forall \boldsymbol{x} \in \mathcal{S},\ \forall \lambda \in \mathbb{R},\ \lambda \boldsymbol{x} \in \mathcal{S}$.

s'appelle un sous-espace vectoriel *de $\mathbb{R}^n$.*

Définition A.1.2 *Soient $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p$ p vecteurs de $\mathbb{R}^d$. Le sous-espace engendré par les vecteurs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p$, noté $\mathrm{vect}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p)$, est le sous-espace vectoriel*

$$\mathrm{vect}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p) \ := \ \left\{ \boldsymbol{x} = \sum_{i=1}^{p} \alpha_i \boldsymbol{x}_i \middle| \alpha_i \in \mathbb{R}\ \forall i \right\}.$$

On rappelle ensuite les différentes propriétes notables des familles de vecteurs.

Définition A.1.3

- *Une famille libre $\{\boldsymbol{x}_i\}_{i=1}^k$ de vecteurs de $\mathbb{R}^d$ est telle que les vecteurs sont linéairement indépendants : pour tous scalaires $\lambda_1, \ldots, \lambda_k$ tels que $\sum_{i=1}^k \lambda_i \boldsymbol{x}_i = 0$, alors $\lambda_1 = \cdots = \lambda_k = 0$. On a nécessairement $k \leq d$.*
- *Une famille liée est une famille de vecteurs qui n'est pas libre.*
- *Une famille génératrice de vecteurs de $\mathbb{R}^d$ est un ensemble de vecteurs $\{\boldsymbol{x}_i\}$ tel que le sous-espace engendré par ces vecteurs soit égal à $\mathbb{R}^d$.*
- *Une* base *de $\mathbb{R}^d$ est une famille de vecteurs $\{\boldsymbol{x}_i\}_{i=1}^d$ qui est à la fois libre et génératrice. Tout vecteur de $\mathbb{R}^d$ s'écrit alors de manière unique comme combinaison linéaire des $\boldsymbol{x}_i$. Toute base de $\mathbb{R}^d$ comporte exactement d vecteurs.*

Comme la taille d'une base de $\mathbb{R}^d$ est d, on dit que cet espace vectoriel est de dimension d. En conséquence, la dimension d'un sous-espace vectoriel est au plus d.

Exemple A.1.1 *Tout vecteur $\boldsymbol{x}$ de $\mathbb{R}^d$ s'écrit $\boldsymbol{x} = \sum_{i=1}^d x_i \boldsymbol{e}_i$, où $\boldsymbol{e}_i = [0 \cdots 0\ 1\ 0 \cdots 0]^{\mathrm{T}}$ est le i-ème vecteur de la base canonique (le coefficient 1 se trouvant en i-ème position).*

Norme et produit scalaire L'utilisation d'une norme, et du produit scalaire associé, permet de mesurer les distances entre vecteurs, ce qui sera particulièrement utile pour montrer la convergence d'une suite de points générés par un algorithme vers une solution d'un problème d'optimisation.

Définition A.1.4 *La* **norme euclidienne** $\|\cdot\|$ *sur $\mathbb{R}^d$ est définie pour tout vecteur $\boldsymbol{x} \in \mathbb{R}^d$ par*

$$\|\boldsymbol{x}\| \ := \ \sqrt{\sum_{i=1}^{d} x_i^2}.$$

Remarque A.1.1 *Il s'agit bien d'une norme, car elle vérifie les quatres axiomes d'une norme :*

1. $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d,\ \|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$;

2. $\|\boldsymbol{x}\| = 0 \;\Leftrightarrow \boldsymbol{x} = \boldsymbol{0}_d$;
3. $\forall \boldsymbol{x},\; \|\boldsymbol{x}\| \geq 0$;
4. $\forall \boldsymbol{x} \in \mathbb{R}^d, \forall \lambda \in \mathbb{R}, \|\lambda \boldsymbol{x}\| = |\lambda| \|\boldsymbol{x}\|$.

On dira qu'un vecteur $\boldsymbol{x} \in \mathbb{R}^d$ est unitaire si $\|\boldsymbol{x}\| = 1$.

Définition A.1.5 *Pour tous vecteurs $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, le* **produit scalaire** *dérivé de la norme euclidienne est une fonction de $\boldsymbol{x}$ et $\boldsymbol{y}$, notée $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$, et définie par*

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} \;:=\; \sum_{i=1}^{d} x_i y_i.$$

Deux vecteurs $\boldsymbol{x}$ et $\boldsymbol{y}$ tels que $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = 0$ seront dits orthogonaux.

On notera en particulier que $\boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$. Ce produit scalaire définit donc un "produit" entre un vecteur ligne et un vecteur colonne.

Proposition A.1.1 *Soient $\boldsymbol{x}$ et $\boldsymbol{y}$ deux vecteurs de $\mathbb{R}^d$. Alors, on a les propriétés suivantes :*

i) $|\boldsymbol{x} + \boldsymbol{y}\|^2 = \|\boldsymbol{x}\|^2 + 2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + \|\boldsymbol{y}\|^2$;

ii) $\|\boldsymbol{x} - \boldsymbol{y}\|^2 = \|\boldsymbol{x}\|^2 - 2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + \|\boldsymbol{y}\|^2$;

iii) $\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 = \frac{1}{4}\left(\|\boldsymbol{x} + \boldsymbol{y}\|^2 + \|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$;

iv) **Inégalité de Cauchy-Schwarz** *:*

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d, \qquad \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} \leq \|\boldsymbol{x}\| \|\boldsymbol{y}\|.$$

Remarque A.1.2 *La dernière inégalité est un résultat très important, qui s'utilise aussi bien en algèbre linéaire qu'en analyse fonctionnelle. Comme on le verra, elle apparaît également dans la preuve des inégalités de Taylor, qui sont fondamentales en optimisation.*

A.1.2 Algèbre linéaire matricielle

Sur les espaces matriciels, on définit également un produit matriciel entre matrices de dimensions compatibles. Pour toutes matrices $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ et $\boldsymbol{B} \in \mathbb{R}^{d \times m}$, la matrice produit $\boldsymbol{AB}$ est définie comme la matrice $\boldsymbol{C} \in \mathbb{R}^{n \times m}$ telle que

$$\forall i = 1, \ldots, n,\; \forall j = 1, \ldots, m, \quad \boldsymbol{C}_{ij} = \sum_{k=1}^{d} \boldsymbol{A}_{ik}\boldsymbol{B}_{kj}.$$

Par analogie, le produit d'une matrice $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ avec un vecteur $\boldsymbol{x} \in \mathbb{R}^d$ sera le vecteur $\boldsymbol{y} \in \mathbb{R}^n$ donné par

$$\forall i = 1, \ldots, n,\; y_i = \sum_{j=1}^{d} \boldsymbol{A}_{ij} x_j.$$

Remarque A.1.3 *On notera que la notation du produit scalaire sur $\mathbb{R}^d$ correspond au produit de deux matrices de taillle $1 \times d$ et $d \times 1$, dont le résultat est une matrice 1×1, c'est-à-dire un scalaire.*

Lorsque l'on travaille avec des matrices, on s'intéresse généralement aux sous-espaces définis ci-dessous.

Définition A.1.6 (Sous-espaces matriciels) *Soit une matrice $\boldsymbol{A} \in \mathbb{R}^{n \times d}$, on définit les deux sous-espaces suivants :*

- *Le* noyau *de $\boldsymbol{A}$ est le sous-espace vectoriel*

$$\ker(\boldsymbol{A}) \ := \ \left\{ \boldsymbol{x} \in \mathbb{R}^d \ \middle| \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}_n \right\}$$

- *L'*image *de $\boldsymbol{A}$ est le sous-espace vectoriel*

$$\mathrm{Im}(\boldsymbol{A}) \ := \ \left\{ \boldsymbol{y} \in \mathbb{R}^n \ \middle| \ \exists \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \right\}$$

  La dimension de ce sous-espace vectoriel s'appelle le **rang** *de $\boldsymbol{A}$. On la note $\mathrm{rang}(\boldsymbol{A})$ et on a $\mathrm{rang}(\boldsymbol{A}) \leq \min\{n, d\}$.*

Théorème A.1.1 (Théorème du rang) *Pour toute matrice $\boldsymbol{A} \in \mathbb{R}^{n \times d}$, on a*

$$\dim(\ker(\boldsymbol{A})) + \mathrm{rang}(\boldsymbol{A}) = d.$$

Définition A.1.7 (Normes matricielles) *On définit sur $\mathbb{R}^{n \times d}$ la norme d'opérateur $\|\cdot\|$ et la norme de Frobenius $\|\cdot\|_F$ par*

$$\forall \boldsymbol{A} \in \mathbb{R}^{n \times d}, \begin{cases} \|\boldsymbol{A}\| & := \ \max_{\substack{\boldsymbol{x} \in \mathbb{R}^d \\ \boldsymbol{x} \neq \boldsymbol{0}_d}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \max_{\substack{\boldsymbol{x} \in \mathbb{R}^d \\ \|\boldsymbol{x}\| = 1}} \|\boldsymbol{A}\boldsymbol{x}\| \\ \|\boldsymbol{A}\|_F & := \ \sqrt{\sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq d}} \boldsymbol{A}_{ij}^2}. \end{cases}$$

Définition A.1.8 (Matrice symétrique) *Une matrice carrée $\boldsymbol{A} \in \mathbb{R}^{d \times d}$ est dite symétrique si elle vérifie $\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{A}$.*

L'ensemble des matrices symétriques de taille $d \times d$ sera noté $\mathcal{S}^d$.

Définition A.1.9 (Matrice inversible) *Une matrice carrée $\boldsymbol{A} \in \mathbb{R}^{d \times d}$ est dite inversible s'il existe $\boldsymbol{B} \in \mathbb{R}^{d \times d}$ telle que $\boldsymbol{B}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{B} = \boldsymbol{I}_d$ (où l'on rappelle que $\boldsymbol{I}_d$ désigne la matrice identité de $\mathbb{R}^{d \times d}$).*

Si elle existe, une telle matrice $\boldsymbol{B}$ est unique : elle est appelée **l'inverse de A** *et on la note $\boldsymbol{A}^{-1}$.*

Définition A.1.10 (Matrice (semi-)définie positive) *Une matrice carrée $\boldsymbol{A} \in \mathbb{R}^{d \times d}$ symétrique est dite semi-définie positive si*

$$\forall \boldsymbol{x} \in \mathbb{R}^d, \quad \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x} \geq 0,$$

ce que l'on notera $\boldsymbol{A} \succeq 0$.

Une telle matrice est dite définie positive lorsque $\boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x} > 0$ pour tout vecteur $\boldsymbol{x}$ non nul, ce que l'on notera $\boldsymbol{A} \succ 0$.

Définition A.1.11 (Matrice orthogonale) *Une matrice carrée $\boldsymbol{P} \in \mathbb{R}^{d\times d}$ est dite orthogonale si $\boldsymbol{P}^{\mathrm{T}} = \boldsymbol{P}^{-1}$.*

Par extension, on dira que $\boldsymbol{Q} \in \mathbb{R}^{n\times d}$ avec $n \leq d$ est orthogonale si $\boldsymbol{Q}\boldsymbol{Q}^{\mathrm{T}} = \boldsymbol{I}_n$ (les colonnes de $\boldsymbol{Q}$ sont donc orthonormées dans $\mathbb{R}^n$).

On notera que lorsque $\boldsymbol{Q} \in \mathbb{R}^{d\times d}$ est une matrice orthogonale, alors sa transposée $\boldsymbol{Q}^{\mathrm{T}}$ est également orthogonale (ce résultat n'est pas vrai pour une matrice rectangulaire). On utilisera fréquemment la propriété des matrices orthogonales énoncée ci-dessous.

Lemme A.1.1 *Soit une matrice $\boldsymbol{A} \in \mathbb{R}^{n\times d}$ et $\boldsymbol{U} \in \mathbb{R}^{n\times n}$, $\boldsymbol{V} \in \mathbb{R}^{d\times d}$ des matrices orthogonales, respectivement de $\mathbb{R}^{n\times n}$ et $\mathbb{R}^{d\times d}$. On a*

$$\|\boldsymbol{A}\| = \|\boldsymbol{U}\boldsymbol{A}\| = \|\boldsymbol{A}\boldsymbol{V}\| \quad \text{et} \quad \|\boldsymbol{A}\|_F = \|\boldsymbol{U}\boldsymbol{A}\|_F = \|\boldsymbol{A}\boldsymbol{V}\|_F,$$

c'est-à-dire que la multiplication par une matrice orthogonale ne modifie pas la norme d'opérateur.

Par corollaire immédiat du lemme précédent, on note qu'une matrice $\boldsymbol{Q} \in \mathbb{R}^{n\times d}$ orthogonale avec $n \leq d$ vérifie nécessairement $\|\boldsymbol{Q}\| = 1$ et $\|\boldsymbol{Q}\|_F = \sqrt{n}$.

Définition A.1.12 (Valeur propre) *Soit une matrice $\boldsymbol{A} \in \mathbb{R}^{d\times d}$. On dit que $\lambda \in \mathbb{R}$ est une* **valeur propre de $\boldsymbol{A}$** *si*

$$\exists \boldsymbol{v} \in \mathbb{R}^d, \boldsymbol{v} \neq \boldsymbol{0}_d, \quad \boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}.$$

Le vecteur $\boldsymbol{v}$ est appelé un **vecteur propre associé à la valeur propre λ**. *L'ensemble des valeurs propres de $\boldsymbol{A}$ s'appelle le spectre de $\boldsymbol{A}$.*

Le sous-espace engendré par les vecteurs propres associés à la même valeur propre d'une matrice s'appelle un sous-espace propre. Sa dimension correspond à l'ordre de multiplicité de la valeur propre relativement à la matrice.

Proposition A.1.2 *Pour toute matrice $\boldsymbol{A} \in \mathbb{R}^{d\times d}$, on a les propriétés suivantes :*

- *La matrice $\boldsymbol{A}$ possède d valeurs propres complexes mais pas nécessairement réelles.*
- *Si la matrice $\boldsymbol{A}$ est semi-définie positive (respectivement définie positive), alors ses valeurs propres sont réelles positives (respectivement strictement positives).*
- *Le noyau de $\boldsymbol{A}$ est engendré par les vecteurs propres associés à la valeur propre 0.*

Théorème A.1.2 (Théorème spectral) *Toute matrice carrée $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ symétrique admet une décomposition dite* **spectrale** *de la forme :*

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1},$$

où $\boldsymbol{P} \in \mathbb{R}^{d\times d}$ est une matrice orthogonale , dont les colonnes $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_d$ forment une base orthonormée *de vecteurs propres, et $\boldsymbol{\Lambda} \in \mathbb{R}^{d\times d}$ est une matrice diagonale qui contient les d valeurs propres de $\boldsymbol{A}$ $\lambda_1, \ldots, \lambda_d$ sur la diagonale.*

Il est à noter que la décomposition spectrale n'est pas unique. En revanche, l'ensemble des valeurs propres est unique, que l'on prenne en compte les ordres de multiplicité ou non.

Géométriquement parlant, on voit ainsi que, pour tout vecteur $\boldsymbol{x} \in \mathbb{R}^d$ décomposé dans la base des $\boldsymbol{p}_i$ que l'on multiplie par $\boldsymbol{A}$, les composantes de ce vecteur associées aux plus grandes valeurs propres[1] seront augmentées, tandis que celles associées aux valeurs propres de petite magnitude seront réduites (voire annihilées dans le cas d'une valeur propre nulle).

[1] On parle ici de plus grandes valeurs propres en valeur absolue, ou magnitude.