Dauphine | PSL
UNIVERSITÉ PARIS

Optimisation en Apprentissage Automatique

Master IASD

Auteur : Mouhamed SY

Encadrant : Clément Royer

Année universitaire : 2024/2025

Contents

Introduction

L'optimisation est au cœur de l'apprentissage automatique et de la science des données. Qu'il s'agisse de modèles supervisés ou non supervisés, la plupart reposent sur la minimisation d'une fonction de coût afin d'ajuster les paramètres et d'améliorer la capacité de généralisation. Parmi les méthodes les plus utilisées, la descente de gradient et ses variantes jouent un rôle central dans l'entraînement des modèles modernes. Ce projet explore différentes stratégies d'optimisation appliquées à des problèmes emblématiques de l'apprentissage automatique. Nous abordons d'abord la régression linéaire sous forme matricielle, résolue à l'aide de la descente de gradient. Nous étendons ensuite cette analyse au cas de la factorisation matricielle, une méthode essentielle pour la réduction de dimension et la compression d'information. Enfin, nous étudions l'effet de la régularisation, introduite pour renforcer la robustesse des modèles face au bruit et prévenir le surapprentissage.L'objectif de ce travail est double : d'une part, mettre en œuvre concrètement ces méthodes d'optimisation, et d'autre part, analyser leur comportement à travers différentes configurations expérimentales. Chaque partie conjugue rigueur mathématique et validation empirique, afin de mieux comprendre les leviers qui permettent d'optimiser efficacement les performances des modèles.

1 Variables matricielles et optimisation

La régression linéaire est l'un des problèmes les plus fondamentaux en apprentissage automatique et en modélisation statistique. Étant donné un ensemble d'observations, l'objectif est de trouver une relation linéaire entre les variables d'entrée et la sortie correspondante. Dans cette section, nous considérons un problème simple de régression linéaire, où l'on cherche à déterminer un poids scalaire w tel que :

$$Xw = wx \approx y$$

où $X = [x] \in \mathbb{R}^{n\times 1}$ est une matrice contenant un vecteur non nul x, et $y \in \mathbb{R}^n$ est le vecteur cible. Le problème peut être formulé comme la minimisation de la fonction objectif suivante :

$$\min_{w\in\mathbb{R}} \quad \frac{1}{2n}\|wx - y\|^2$$

1.1 Question 1.a : existence de solutions

Nous analysons tout d'abord l'existence et l'unicité des solutions de ce problème d'optimisation. Pour trouver la valeur optimale de w, on dérive la fonction objectif par rapport à w et on annule la dérivée :

$$\frac{d}{dw}\left(\frac{1}{2n}\|wx - y\|^2\right) = \frac{1}{n}x^T(wx - y) = 0$$

En développant cette équation :

$$x^T(wx - y) = 0 \quad \Rightarrow \quad x^T wx - x^T y = 0 \quad \Rightarrow \quad w(x^T x) = x^T y$$

$$\Rightarrow \quad w = \frac{x^T y}{x^T x}$$

Comme x est non nul, on a nécessairement $x^T x \neq 0$, ce qui garantit que la solution est bien définie et unique.
L'ensemble des solutions de ce problème d'optimisation est donc :

$$\left\{w \in \mathbb{R} \mid w = \frac{x^T y}{x^T x}\right\}$$

Cette solution est unique tant que le vecteur x est non nul.

1.2 Question 1.b : surparamétrisation du problème

Dans cette partie, nous nous intéressons à une version surparamétrée du problème de régression linéaire. Initialement, il s'agit de trouver un scalaire $w \in \mathbb{R}$ tel que $wx \approx y$, ce qui revient à minimiser l'erreur quadratique moyenne :

$$f(w) = \frac{1}{2n}\|wx - y\|^2$$

Cependant, cette équation admet une solution exacte uniquement si x et y sont colinéaires. Pour élargir l'espace des solutions, nous reformulons le problème en introduisant une matrice $W \in \mathbb{R}^{n\times n}$, et en minimisant :

$$f(W) = \frac{1}{2n}\|Wx - y\|^2$$

Ce type de reformulation est particulièrement pertinent dans les contextes d'apprentissage profond, où la surparamétrisation joue un rôle clé dans la capacité d'approximation des modèles.
En notant $x \in \mathbb{R}^n$ un vecteur colonne, et $X = x^\top \in \mathbb{R}^{1\times n}$, on peut réécrire la fonction objectif en utilisant la norme de Frobenius :

$$f(W) = \frac{1}{2n}\|XW - y\|_F^2$$

Le gradient de cette fonction par rapport à W est :

$$\nabla f(W) = \frac{1}{n}X^\top(XW - y)$$

La mise à jour de W se fait alors selon la règle classique de descente de gradient :

$$W_{t+1} = W_t - \eta\nabla f(W_t)$$

où η est le taux d'apprentissage.

Implémentation: nous avons généré les vecteurs x et y aléatoirement selon une loi normale standard $\mathcal{N}(0, I_n)$, avec $n = 1000$. La matrice W est initialisée à zéro, et l'optimisation est effectuée sur 200 itérations avec un taux d'apprentissage $\eta = 0.1$. Une classe dédiée à la régression a été implémentée pour encapsuler la fonction objectif et son gradient. La figure 1 montre la convergence de la fonction objectif au cours des itérations. On observe une diminution rapide au début, suivie d'une stabilisation, ce qui confirme l'efficacité de l'algorithme.

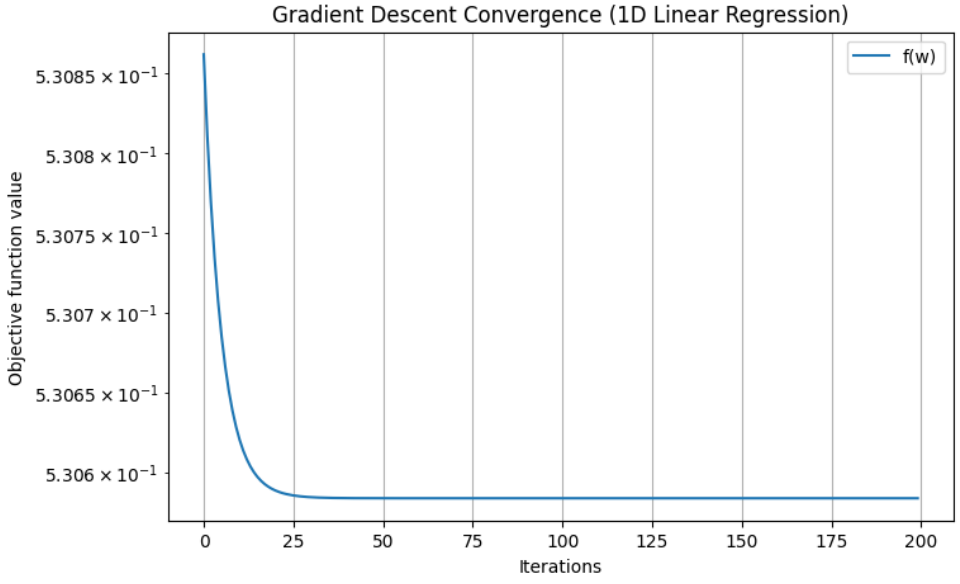

Figure 1: Convergence de la fonction objectif lors de l'optimisation de W.

Les résultats numériques sont résumés dans le tableau ci-dessous :

Itération	$f(w)$	**Gradient**	w
0	5.3086×10^{-1}	-2.3146×10^{-2}	0.002315
50	5.3058×10^{-1}	-1.4574×10^{-4}	0.023877
100	5.3058×10^{-1}	-9.1771×10^{-7}	0.024013
150	5.3058×10^{-1}	-5.7785×10^{-9}	0.024014

Table 1: Évolution de la fonction objectif, du gradient et de la valeur de w au cours de l'optimisation.

Nous avons comparé la solution trouvée par descente de gradient avec la solution analytique :

$$\text{Solution analytique : } w^* = 0{,}024014, \quad \text{Solution GD : } w = 0{,}024014$$

Cette parfaite concordance valide notre implémentation. Le gradient devenant quasi nul montre que l'algorithme converge vers un point stationnaire. Cette expérience illustre l'intérêt de la surparamétrisation : bien que le problème reformulé introduise plus de degrés de liberté, cela ne nuit pas à la convergence. Au contraire, cela peut favoriser une optimisation plus efficace. On peut observer cela dans les réseaux de neurones modernes.

1.3 Question 2.a : justification de la valeur optimale

Nous souhaitons démontrer que la valeur optimale du problème est toujours égale à 0. Par définition, la fonction objectif du problème est :

$$f(W) = \frac{1}{2n}\|Wx - y\|^2$$

S'il existe une matrice W^* telle que $W^*x = y$, alors :

$$\|W^*x - y\|^2 = 0$$

Puisque la norme au carré d'un vecteur nul est égale à 0, il en découle que :

$$f(W^*) = \frac{1}{2n} \times 0 = 0$$

Ainsi, la valeur optimale de la fonction objectif est toujours 0, ce qui signifie que le minimum est atteint dès qu'une solution W^* satisfaisant $Wx = y$ existe.

1.4 Question 2.b : adaptation de la descente de gradient et comparaison des résultats

Nous avons adapté le code de descente de gradient précédemment implémenté pour résoudre le problème surparamétré défini par :

$$\min_{W \in \mathbb{R}^{n \times n}} f(W) = \frac{1}{2n}\|Wx - y\|^2$$

La classe `MatrixRegression` a été développée pour encapsuler ce nouveau problème de régression matricielle. Elle définit la fonction objectif ainsi que le gradient, calculé à l'aide de la formule dérivée du notebook sur la différentiation automatique :

$$\nabla f(W) = \frac{1}{n} X^\top (XW - y)$$

où $X = x^\top \in \mathbb{R}^{1 \times n}$.
L'algorithme de descente de gradient est initialisé avec une matrice nulle $W_0 = 0$, et est exécuté sur 200 itérations avec un taux d'apprentissage constant $\alpha = 0.1$. La mise à jour à chaque itération est de la forme :

$$W_{k+1} = W_k - \alpha \nabla f(W_k)$$

Nous présentons ci-dessous l'évolution de la fonction objectif, de la norme du gradient, et de la valeur de $\|Wx - y\|^2$ à quelques itérations clés :

Itération	$f(W)$	$\|\nabla f(W)\|$
0	5.3086×10^{-1}	$1.0116 \times 10^{+0}$
50	2.1048×10^{-5}	6.3698×10^{-3}
100	8.3449×10^{-10}	4.0109×10^{-5}
150	3.3086×10^{-14}	2.5255×10^{-7}

Table 2: Évolution de la fonction objectif et de la norme du gradient lors de l'optimisation matricielle.

Ces résultats confirment que la fonction objectif décroît rapidement de façon exponentielle, et que le gradient devient très faible, signe d'une convergence effective vers un minimum. Nous comparons les performances des deux approches scalaire et matricielle sur les mêmes données (x, y). Les solutions obtenues sont résumées ci-dessous :

- Solution analytique (scalaire) : $w^* = 0{,}024014$
- Solution par descente de gradient (scalaire) : $w = 0{,}024014$
- Résidu exact pour la solution matricielle : $\|Wx - y\|^2 = 0$

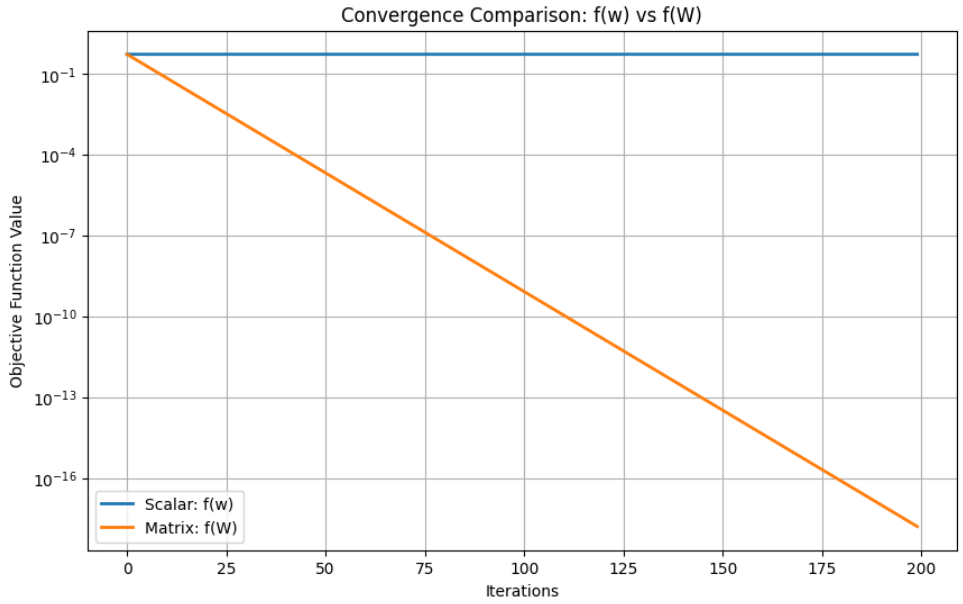

Figure 2: Comparaison de la convergence de la fonction objectif pour la régression scalaire et matricielle (échelle logarithmique sur l'axe des ordonnées).

On peut ainsi remarquer que la régression scalaire permet de trouver une solution optimale unique, mais n'annule pas la fonction objectif à moins que x et y soient colinéaires. En revanche, la formulation matricielle permet d'obtenir une solution exacte (avec résidu nul), ce qui illustre la puissance de la surparamétrisation. Cette capacité d'approximation parfaite des données est une caractéristique clé dans l'apprentissage profond, à condition d'être contrôlée par une régularisation appropriée.

2 Factorisation matricielle

Dans cette partie, on prolonge l'étude précédente en considérant des problèmes de factorisation matricielle, où le modèle est défini à l'aide de deux vecteurs. Comme vu en TD, on cherche à approximer une matrice de données $X \in \mathbb{R}^{n_1 \times n_2}$ par une matrice de rang 1, que l'on peut écrire sous la forme $uv^\top$, avec $u \in \mathbb{R}^{n_1}$ et $v \in \mathbb{R}^{n_2}$. Le problème d'approximation s'écrit alors :

$$\min_{u \in \mathbb{R}^{n_1}, v \in \mathbb{R}^{n_2}} \quad f(u,v) = \frac{1}{2n}\|uv^\top - X\|_F^2 = \frac{1}{2n}\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}([u]_i[v]_j - X_{ij})^2$$

où $n = n_1 \cdot n_2$ et $\|\cdot\|_F$ est la norme de Frobenius.

—

2.1 Question 3.a : Comparaison de la descente de gradient (GD) et de la descente de gradient stochastique (SGD)

Nous avons généré une matrice de rang 1 définie par $X = \bar{x}z^\top \in \mathbb{R}^{10\times 20}$, où les vecteurs $\bar{x} \in \mathbb{R}^{10}$ et $z \in \mathbb{R}^{20}$ sont tirés aléatoirement selon une loi normale $\mathcal{N}(0, I)$. Une classe `MatrixFactorization` a été implémentée pour modéliser le problème, incluant la fonction objectif $f(u, v)$ ainsi que les gradients par rapport à u et v. Deux algorithmes ont été comparés :

- Descente de gradient (GD) : mise à jour à chaque itération du gradient global
- Descente de gradient stochastique (SGD): mise à jour à partir d'un seul échantillon tiré aléatoirement à chaque itération

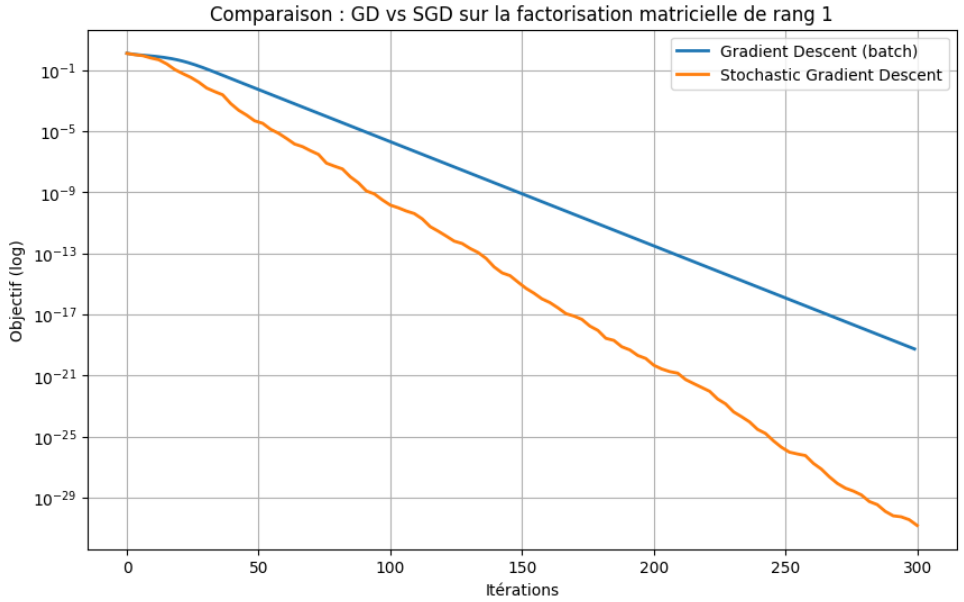

Figure 3: Comparaison entre la descente de gradient classique (batch) et la descente de gradient stochastique (SGD) sur le problème de factorisation matricielle de rang 1.

Comme illustré dans la figure 3, la descente de gradient stochastique converge plus rapidement et de façon plus marquée que la descente de gradient batch. Bien que plus bruitée, la méthode SGD est ici particulièrement efficace pour ce type de problème : elle permet d'approcher très finement la solution optimale avec un coût computationnel par itération réduit.

2.2 Question 3.b : Influence de la taille de batch (mini-batch SGD)

Afin d'analyser l'impact de la taille de batch sur la convergence de l'algorithme SGD, nous avons testé différentes tailles : $\{1, 5, 20, 100, 200\}$. La fonction `mini_batch_sgd` met à jour les vecteurs u et v à partir

d'un sous-ensemble d'éléments de la matrice X, ce qui permet d'interpoler entre la version purement stochastique (batch size = 1) et la version batch complète (batch size = $n_1 \cdot n_2$).

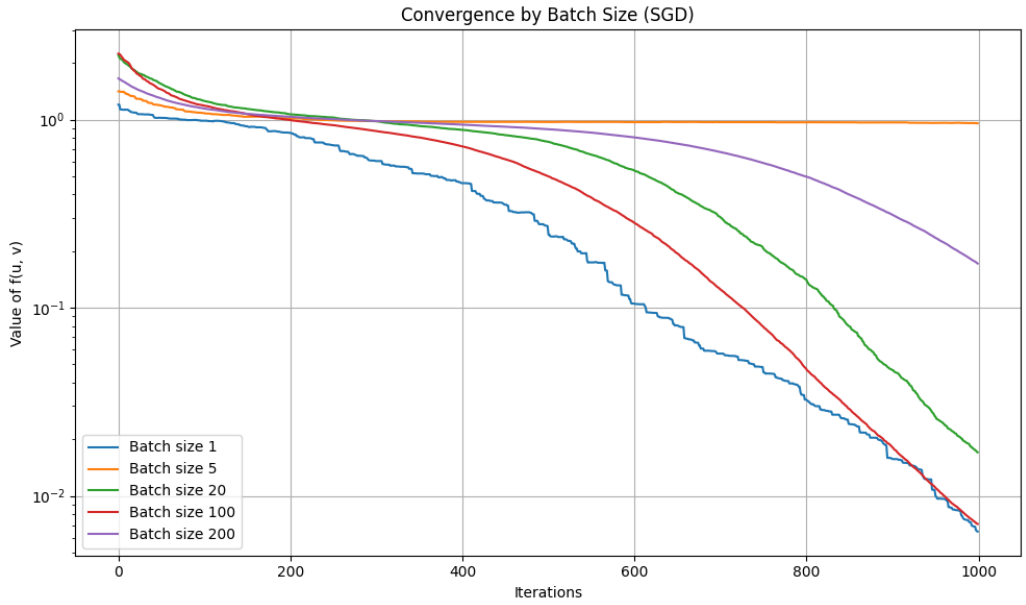

Figure 4: Convergence de la fonction objectif selon différentes tailles de batch en SGD.

Analyse:

- Batch size 1: forte variance des gradients mais excellente vitesse de convergence.
- Batch sizes 5 à 20 : compromis intéressant entre stabilité et rapidité, avec des courbes plus lisses.
- Batch sizes 100 et 200 : convergence plus lente en début d'apprentissage, mais mise à jour plus stable.

La taille du batch joue un rôle central dans l'efficacité de la descente de gradient stochastique. Une taille très petite permet des mises à jour rapides mais instables, tandis qu'une grande taille stabilise l'apprentissage mais ralentit l'optimisation. Dans notre cas, une taille de batch entre 20 et 100 semble offrir le meilleur compromis.

3 Question 4 : Approximation matricielle de rang r

Nous généralisons ici le problème de factorisation de rang 1 en cherchant une approximation matricielle de rang $r \leq \min(n_1, n_2)$, ce qui conduit au problème suivant :

$$\min_{U \in \mathbb{R}^{n_1 \times r}, V \in \mathbb{R}^{n_2 \times r}} \frac{1}{2n} \|UV^\top - X\|_F^2 = \frac{1}{2n} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left([UV^\top]_{ij} - [X]_{ij}\right)^2$$

avec $n = n_1 \cdot n_2$. Ce problème généralise naturellement le cas de rang 1, puisque toute matrice $A \in \mathbb{R}^{n_1 \times n_2}$ de rang $\leq r$ peut s'écrire sous la forme :

$$A = \sum_{i=1}^{r} a_i b_i^\top \quad \text{avec } a_i \in \mathbb{R}^{n_1},\ b_i \in \mathbb{R}^{n_2}$$

—

3.1 i) Adaptation du code de la question 3 pour le rang r

Nous avons modifié l'implémentation précédente pour prendre en charge deux matrices $U \in \mathbb{R}^{n_1 \times r}$ et $V \in \mathbb{R}^{n_2 \times r}$, au lieu des vecteurs u et v. La classe `LowRankFactorization` modélise le nouveau problème. Elle inclut :

- la fonction objectif $f(U, V)$,
- les gradients $\nabla_U f$ et $\nabla_V f$,
- les méthodes d'optimisation par descente de gradient par lots et descente de gradient stochastique (SGD).

Les figures suivantes montrent la convergence des deux méthodes pour une approximation de rang $r = 3$.

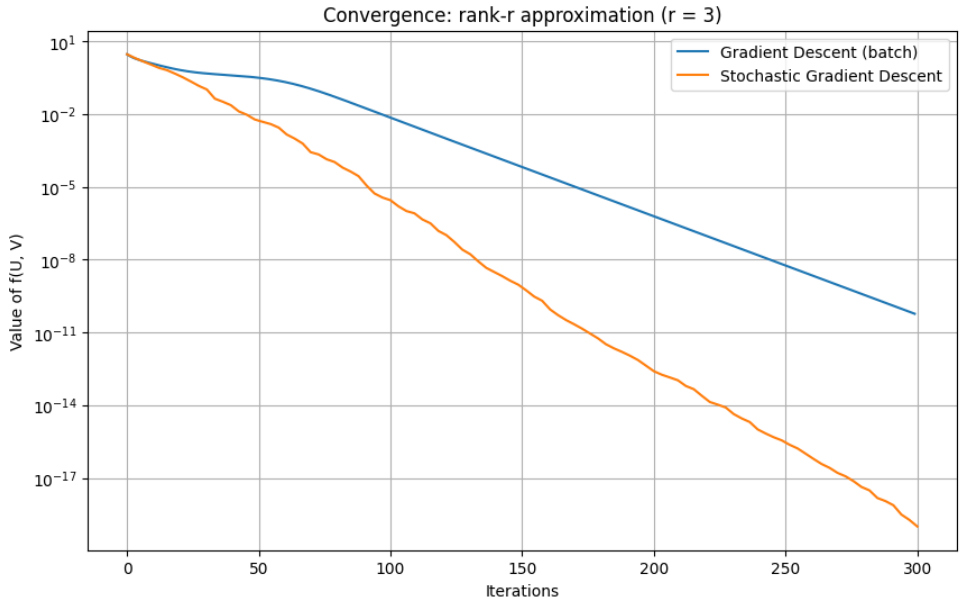

Figure 5: Convergence de la descente de gradient et du gradient stochastique pour une approximation de rang $r = 3$.

On observe une fois de plus que la méthode SGD converge plus rapidement et atteint une erreur bien plus faible que la méthode batch. Le comportement reste similaire à celui observé pour le rang 1.

3.2 ii) Comparaison pour différents cas

Nous testons maintenant l'approche sur deux cas spécifiques avec $(n_1, n_2) = (10, 20)$:

- **Cas 1** : la matrice X est de rang 5, et on fixe $r = 5$
- **Cas 2** : la matrice X est de rang 10, et on fixe $r = 1$

Dans les deux cas, nous générons X à partir de matrices $\bar{U}, \bar{V}$ tirées aléatoirement selon une loi normale standard, et utilisons les algorithmes GD et SGD pour approximer X par une factorisation de rang r.

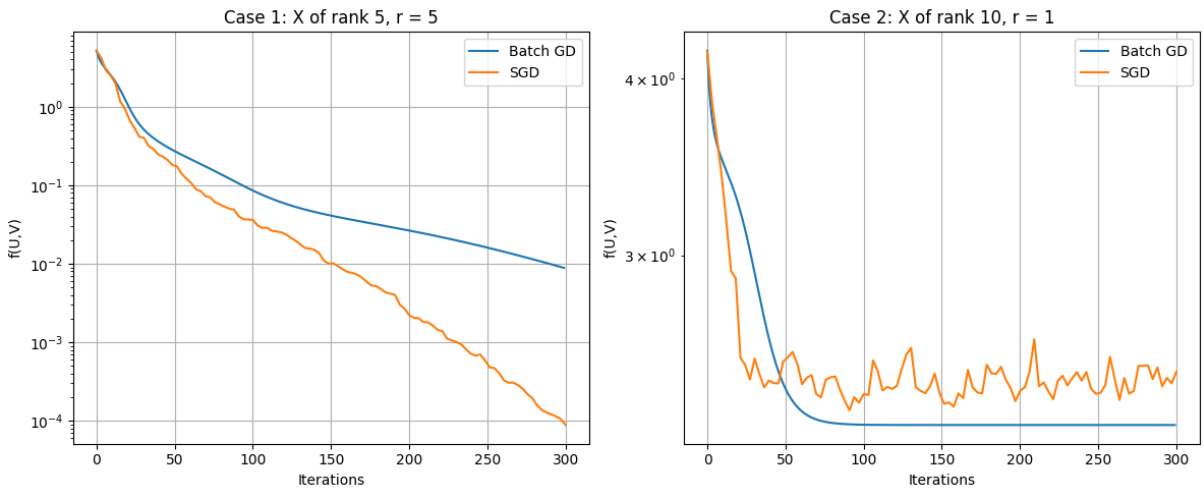

Figure 6: Comparaison des méthodes batch et stochastique sur deux cas : (gauche) $\text{rang}(X) = r = 5$, (droite) $\text{rang}(X) = 10$, $r = 1$.

Interprétation:

- **Cas 1 (rang 5)** : la méthode SGD converge plus vite et atteint une meilleure précision, tandis que GD progresse plus lentement mais reste stable.
- **Cas 2 (rang 10, $r = 1$)** : les deux méthodes atteignent une erreur résiduelle minimale rapidement, mais SGD devient instable par la suite. Cela s'explique par l'impossibilité de bien approximer X avec un rang trop faible.

La méthode stochastique (SGD) surpasse systématiquement la méthode batch en termes de vitesse et de précision, tant que le rang cible r est suffisamment grand. Cependant, en cas de sous-paramétrisation (par exemple $r = 1$ pour une matrice de rang élevé), la convergence est rapide mais l'approximation reste mauvaise. Ce comportement met en évidence l'importance de choisir un rang adapté à la structure de la matrice à approximer.

4 Question 5 : Factorisation matricielle régularisée

Nous considérons ici un cas où $r > 1$ dans le problème (4), mais la matrice cible X est une version bruitée d'une matrice de rang 1. Pour garantir la robustesse de la solution et limiter le sur-apprentissage du bruit, on introduit une régularisation de type ridge (Tikhonov). Le problème devient alors :

$$\min_{U \in \mathbb{R}^{n_1 \times r},\, V \in \mathbb{R}^{n_2 \times r}} \quad \frac{1}{2n}\|UV^\top - X\|_F^2 + \frac{\lambda}{2}\|U\|_F^2 + \frac{\lambda}{2}\|V\|_F^2$$

où $\lambda > 0$ est le paramètre de régularisation, et $n = n_1 \cdot n_2$.

4.1 i) Implémentation de la descente de gradient régularisée

Nous avons adapté la classe `LowRankFactorization` en ajoutant un terme de régularisation à la fonction objectif ainsi qu'à ses gradients. Les algorithmes de descente de gradient (batch) et stochastique (SGD) ont été modifiés pour prendre en compte ce nouveau terme. La matrice $X \in \mathbb{R}^{n_1 \times n_2}$ est générée comme une version bruitée de $xz^\top$, où $x \in \mathbb{R}^{n_1}$, $z \in \mathbb{R}^{n_2}$, et $\varepsilon \in \mathbb{R}^{n_1 \times n_2}$ suit une loi $\mathcal{N}(0, I)$:

$$X = xz^\top + \varepsilon$$

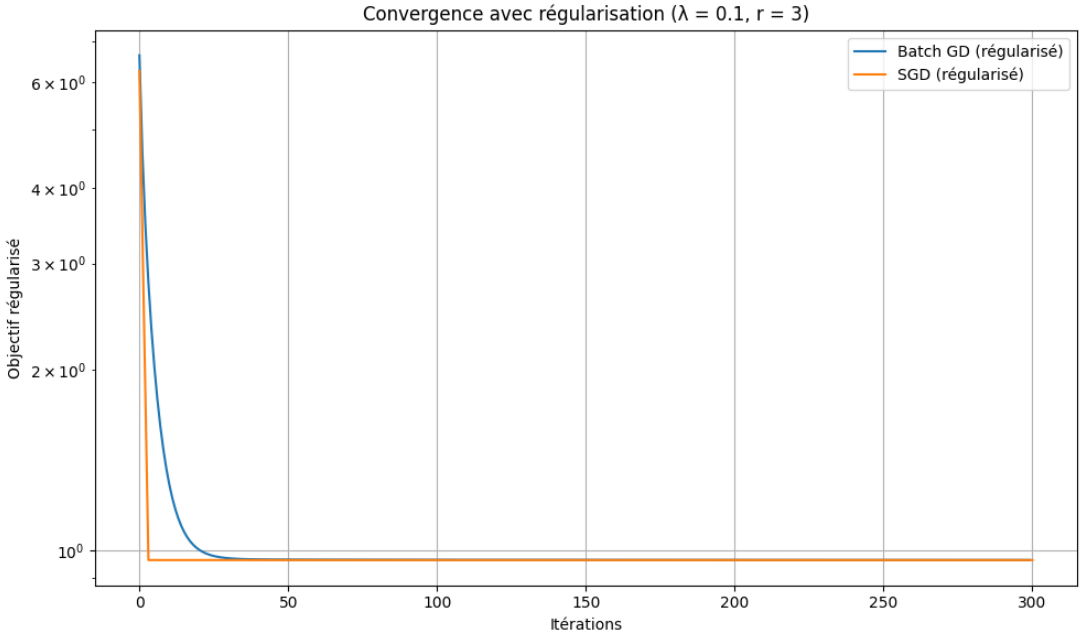

Figure 7: Convergence de la fonction objectif régularisée pour $\lambda = 0.1$ et $r = 3$.

Comme on le voit sur la figure 7, les deux méthodes convergent rapidement vers une solution stable. La régularisation permet d'atteindre une fonction objectif plus lisse et de prévenir le surapprentissage du bruit.

4.2 ii) Étude de l'effet de λ sur le rang effectif

Nous avons testé différentes valeurs du paramètre de régularisation $\lambda \in \{0, 0.001, 0.01, 0.1, 1\}$. Pour chaque cas, nous avons analysé le rang effectif de la matrice $UV^\top$ obtenue à la convergence, via ses valeurs singulières.

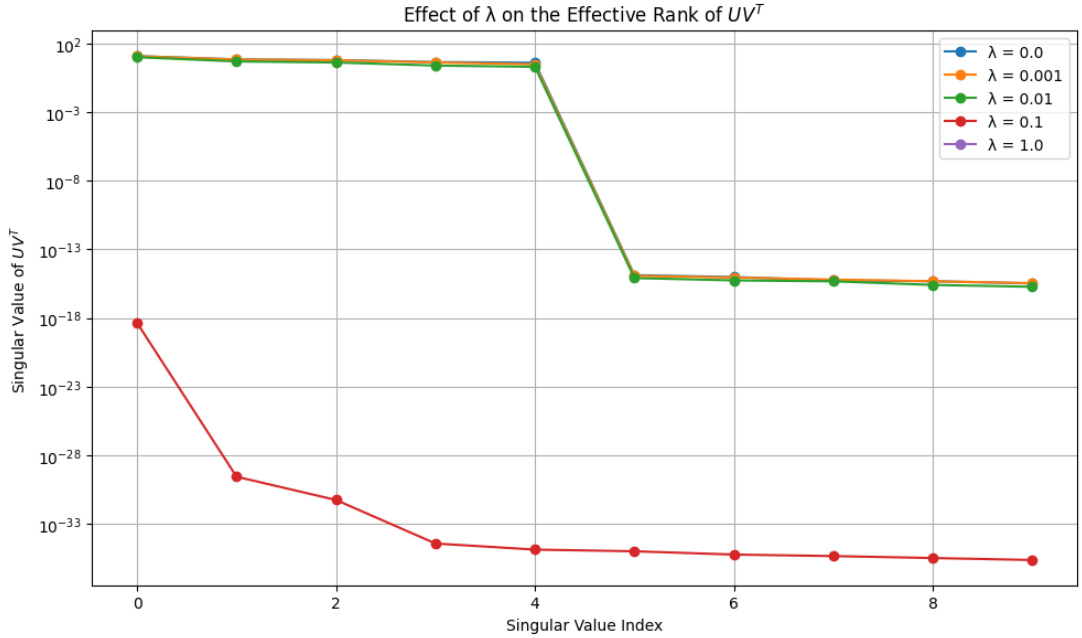

Figure 8: Effet de λ sur le rang effectif de la solution $UV^\top$.

Observations:

- Pour $\lambda = 0$, la solution surapprend le bruit : on observe un rang effectif proche de $r = 5$.
- Pour $\lambda = 0.001$ et $\lambda = 0.01$, les plus petites valeurs singulières persistent, ce qui indique encore une approximation de rang plus élevé.
- À partir de $\lambda = 0.1$, seule une valeur singulière significative subsiste : la régularisation force la solution vers une structure de rang 1.
- Pour $\lambda = 1$, toutes les valeurs sont quasi nulles, ce qui traduit une sous-adaptation.

4.3 iii) Peut-on choisir un λ tel que la solution soit de rang 1 ?

Oui, les résultats montrent qu'il est possible de déterminer un λ (par exemple $\lambda = 0.1$) qui force la solution à être effectivement de rang 1. Ce phénomène est visible via le collapse rapide des valeurs singulières (cf. figure 8). Cependant, la solution obtenue $UV^\top$ ne correspond pas exactement à $xz^\top$, car le bruit ε rend cette correspondance impossible. Néanmoins, la solution capturant la direction dominante est conforme à la composante principale du signal $xz^\top$, ce qui est le but recherché dans une régularisation bien calibrée.

4.4 iv) Les conclusions sont-elles différentes selon que l'on utilise le gradient stochastique ou la descente de gradient ?

D'un point de vue qualitatif, les conclusions principales restent les mêmes que l'on utilise la descente de gradient (GD) ou la descente de gradient stochastique (SGD) : les deux méthodes convergent vers des solutions similaires en présence d'une régularisation bien choisie, comme en témoigne la figure 7. Le rang effectif de la solution finale $UV^\top$ est également comparable, et l'effet de λ sur la structure de la solution reste inchangé.

Cependant, des différences notables apparaissent au niveau de la dynamique d'optimisation :

- **SGD** converge plus rapidement en début d'apprentissage, car les mises à jour sont plus fréquentes et moins coûteuses.
- **GD** suit une trajectoire plus stable, avec une meilleure régularité dans la descente de la fonction objectif.
- En présence de bruit, la régularisation agit de façon similaire pour les deux méthodes, mais GD peut bénéficier d'une meilleure robustesse numérique sur des petits datasets.

La régularisation est un levier essentiel pour obtenir des approximations robustes et éviter le surapprentissage du bruit. Il est possible de forcer une solution de rang 1 à partir d'un problème surparamétré, en contrôlant finement λ. Les conclusions globales sur la qualité de la solution, l'effet de la régularisation et le rang effectif sont inchangées. Les différences entre GD et SGD se situent principalement dans la vitesse de convergence et la stabilité. Le choix de l'algorithme dépendra donc des contraintes pratiques : budget de calcul, taille du dataset, ou encore nécessité d'une convergence rapide.

Conclusion

Ce projet a permis d'approfondir les fondements de l'optimisation en apprentissage automatique, en s'appuyant sur des cas concrets tels que la régression linéaire, la factorisation matricielle et ses variantes régularisées. À travers une approche progressive, nous avons analysé l'effet de la surparamétrisation, comparé différentes méthodes d'optimisation (descente de gradient, version stochastique, mini-batch), et évalué l'impact des paramètres clés sur la convergence et la qualité des solutions. L'étude a notamment mis en évidence l'intérêt de la descente de gradient stochastique pour des problèmes de grande dimension, ainsi que l'importance de la régularisation pour limiter le surapprentissage et obtenir des solutions robustes, même en présence de bruit. De plus, l'ajustement du rang dans les factorisations s'est révélé déterminant pour atteindre un bon compromis entre expressivité du modèle et capacité de généralisation. En combinant analyse théorique et validation expérimentale, ce travail met en lumière le rôle central de l'optimisation dans la conception de modèles performants et fiables, et constitue une base solide pour aborder des problématiques plus avancées en apprentissage automatique.