Optimisation convexe séquentielle

Sorbonne Université

Master M2A : Apprentissage et Algorithmes

Rapport OCO

Auteurs:
Saadaoui Aymane
Afandi Mohammad
Satouri Moïne

Table des matières

1 Algorithmes

1.1 Gradient Descent

On commence par implémenter Gradient Descent sans contrainte. Voici la courbe que nous obtenons :

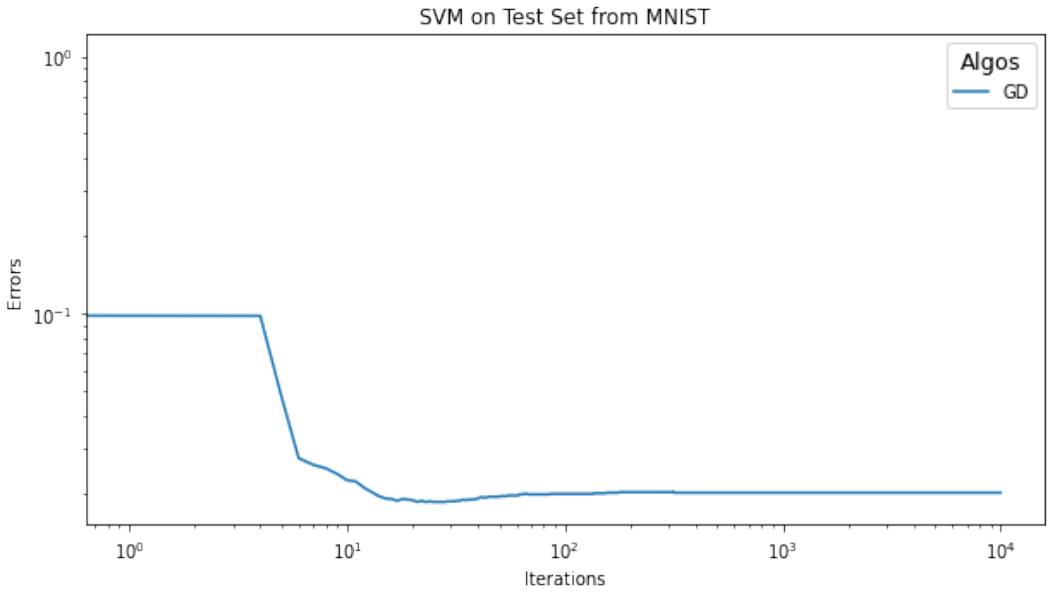

Figure 1: Unconstrained GD

Intéressons-nous désormais à la méthode du gradient projeté sur la boule ℓ^1 :

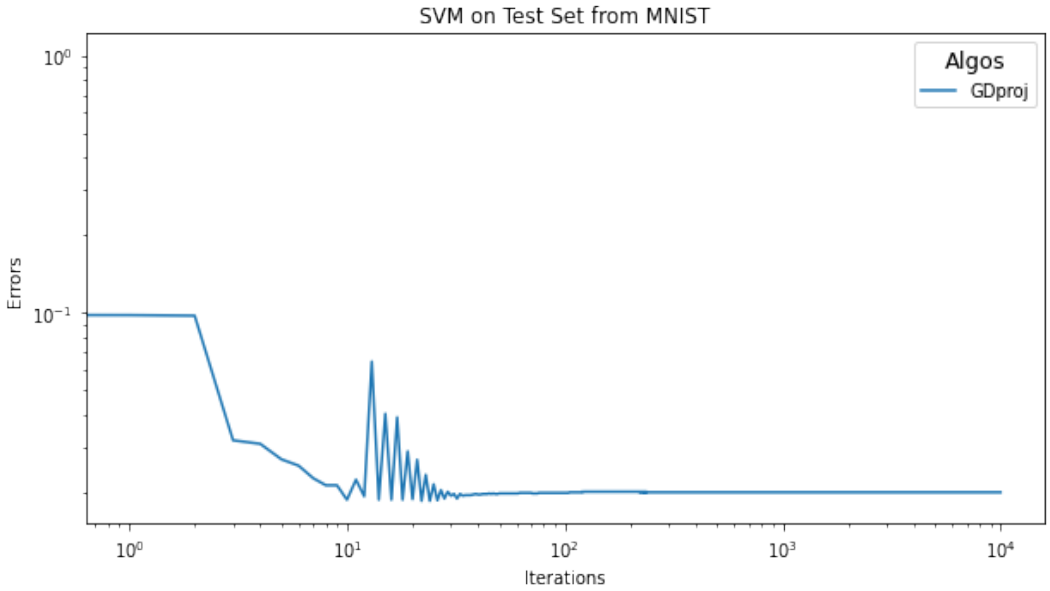

Figure 2: Projected GD

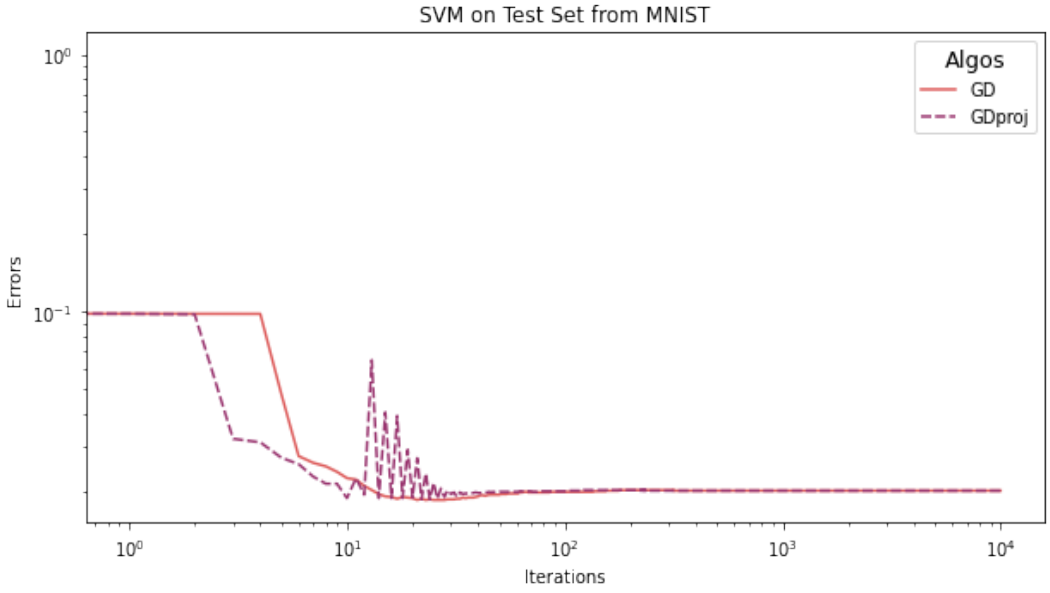

Figure 3: Unconstrained vs Projected GD

On constate que la précision est meilleure pour la version projetée, avec une convergence un peu plus rapide.
On remarque toutefois une détérioration des performances de cet algorithme

après un certain nombre d'itérations. Cela illustre le problème du surapprentissage (overfitting).

En ce qui concerne le choix des paramètres on choisit :

- Paramètre de régularisation ℓ^2 (α dans notre code) : $\frac{1}{3}$
- Rayon de la boule ℓ^1 (*radius* dans notre code) : 100

Le paramètre de régularisation ne doit pas être très grand pour que la solution du problème initial reste proche de celle avec régularisation.

Pour le rayon de la boule on choisit un rayon assez large pour que la projection n'intervienne que si on s'éloigne beaucoup trop.

1.2 Stochastic Gradient Descent

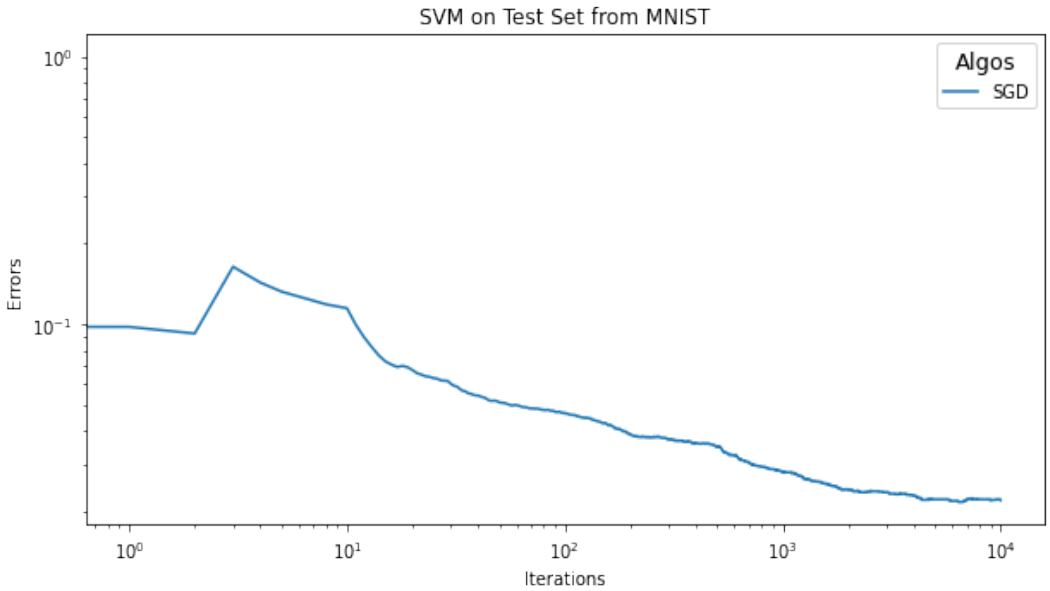

Figure 4: Unconstrained Stochastic GD

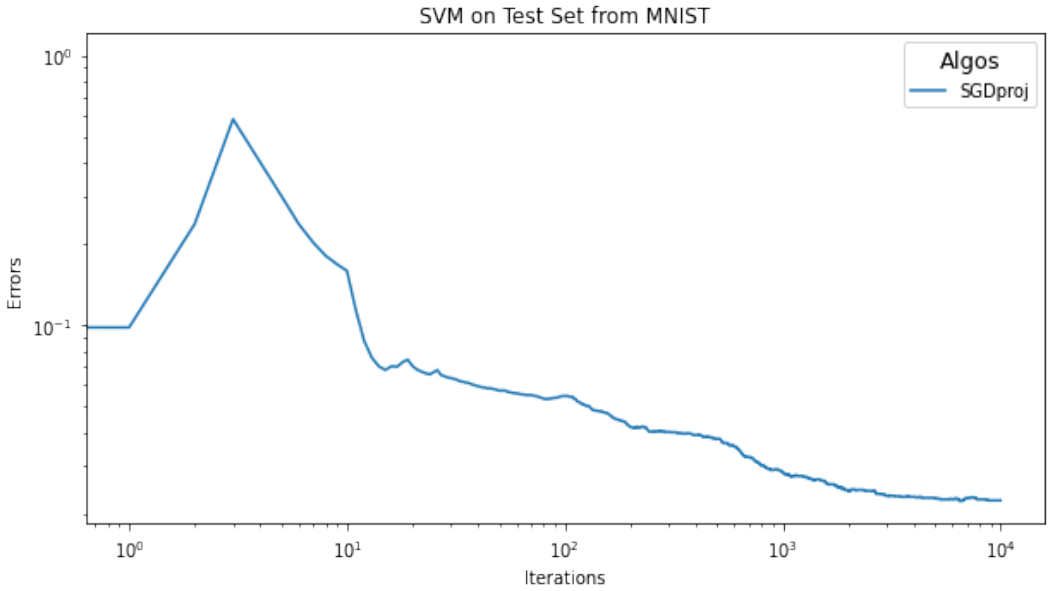

Figure 5: Projected SGD

L'algorithme GD (resp. proj) s'exécute en 12.85 min (resp. 12.79 min) et obtient une erreur finale de 0.0201 (resp. 0.0201) comparé à l'algorithme SGD (resp. proj) qui s'exécute en 1.32 min (resp . 1.41 min) avec une erreur finale de 0.0221 (resp. 0.0224).
La version stochastique de DG s'execute en moyenne 9.7 fois plus rapidement avec des erreurs similaires sur notre test set de MNIST. On préférera la version stochastique dans ce cas de figure. Notons quand même que les algorithmes non stochastique sont beaucoup plus précis et qu'une alternative entre ces deux méthodes serait l'utilisation d'un mini-batch qui stabiliserait la version stochastique.

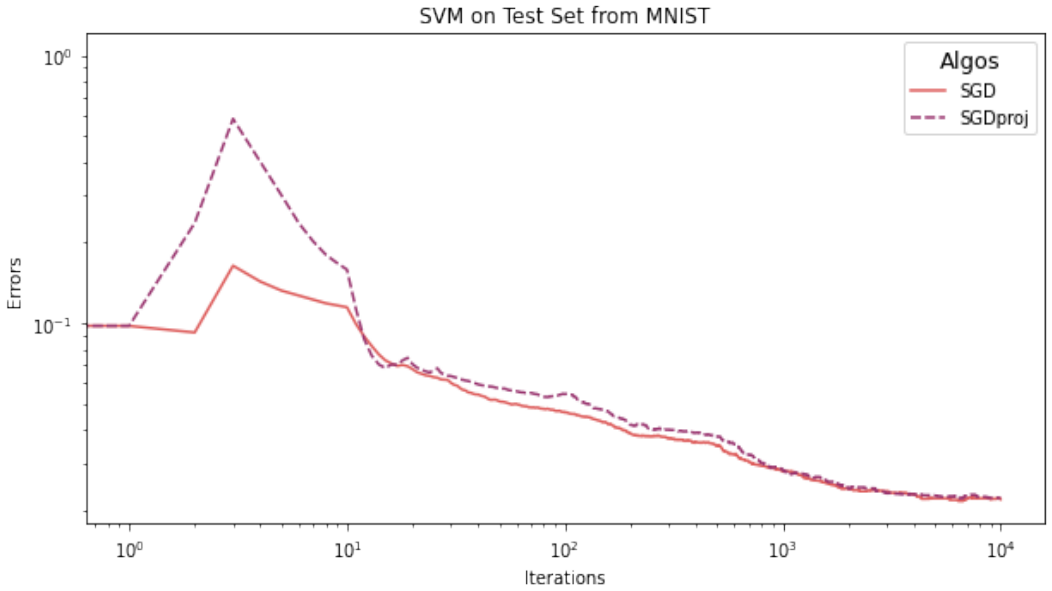

Figure 6: SGD vs Projected SGD

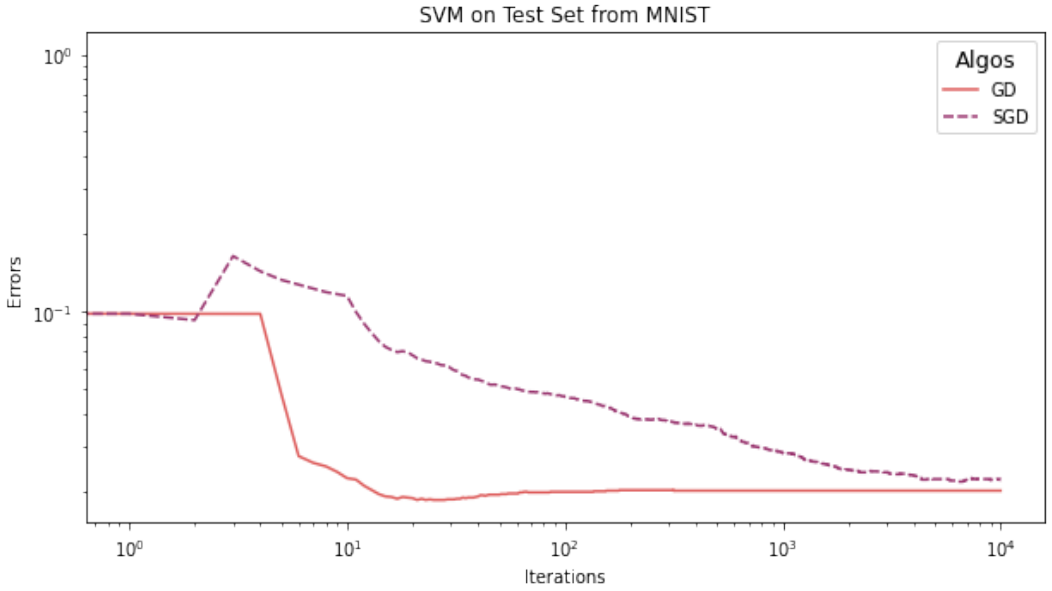

Figure 7: GD vs SGD

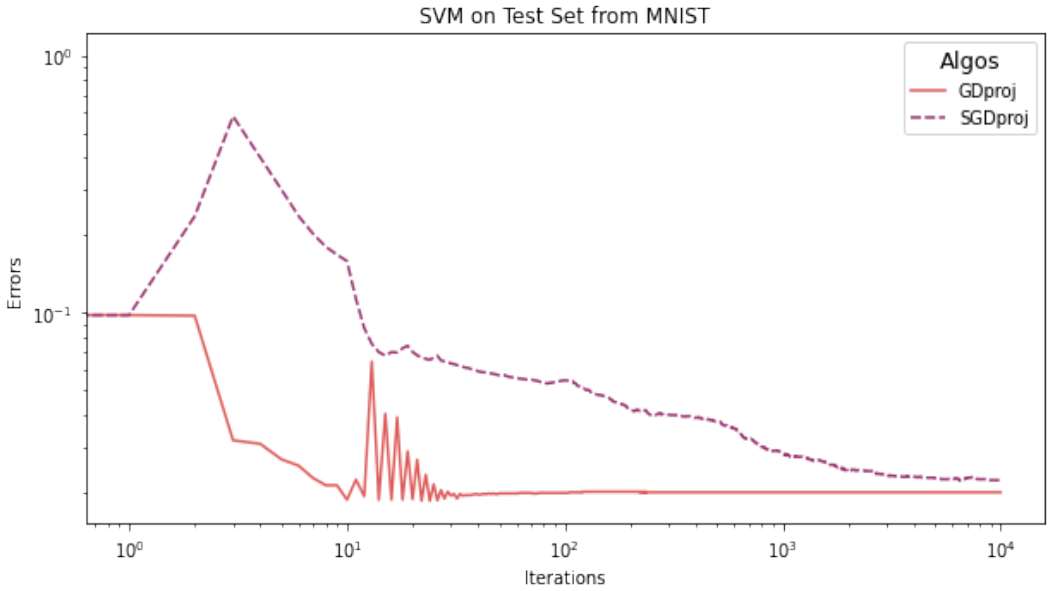

Figure 8: Projected GD vs Projected SGD

1.3 Regularised Follow The Leader

L'algorithme SMD s'exécute en 1.44 min et obtient une erreur finale de 0.0164 comparé à l'algorithme SGD proj qui s'exécute en 1.41 min avec une erreure finale de 0.0224.
Dans ce cas on privilégie l'algorithme SMD qui nous permet d'obtenir une meilleure erreur avec un temps d'exécution similaire. On remarque également que pour les 10 premières itérations SMD fait beaucoup mieux que SGDproj.

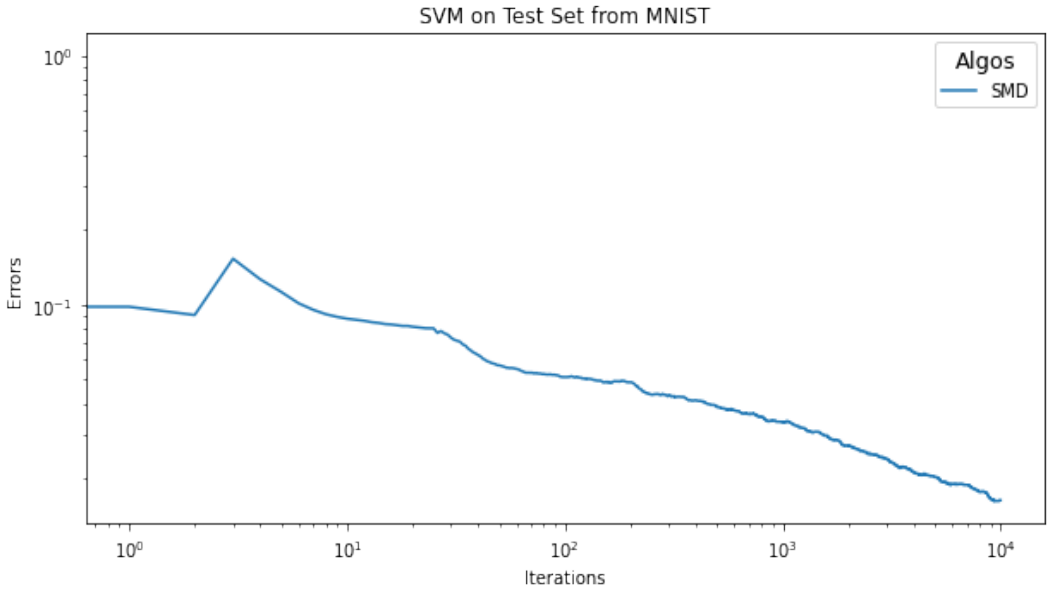

Figure 9: Stochastic Mirror Descent

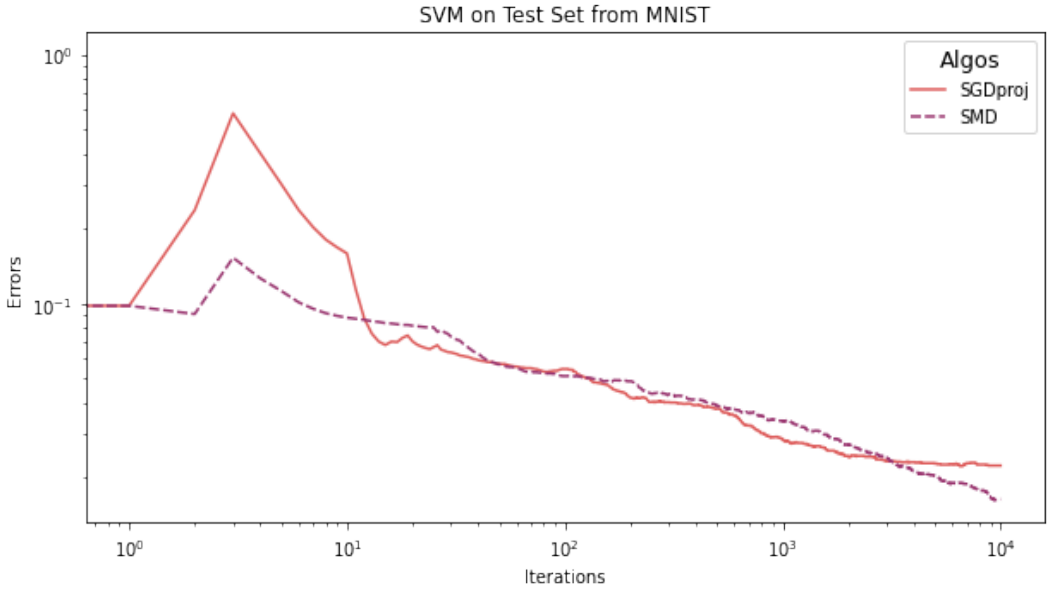

Figure 10: Projected SGD vs SMD

L'algorithme SEG +/- s'exécute en 1.37 min et obtient une erreur finale de 0.0124 comparé à l'algorithme SGD proj qui s'exécute en 1.41 min avec une erreure finale de 0.0224.

Dans ce cas on préfère l'algo SEG+- à SGDproj et même SMD qui nous permet d'obtenir une meilleure erreur avec un temps d'exécution plus rapide.

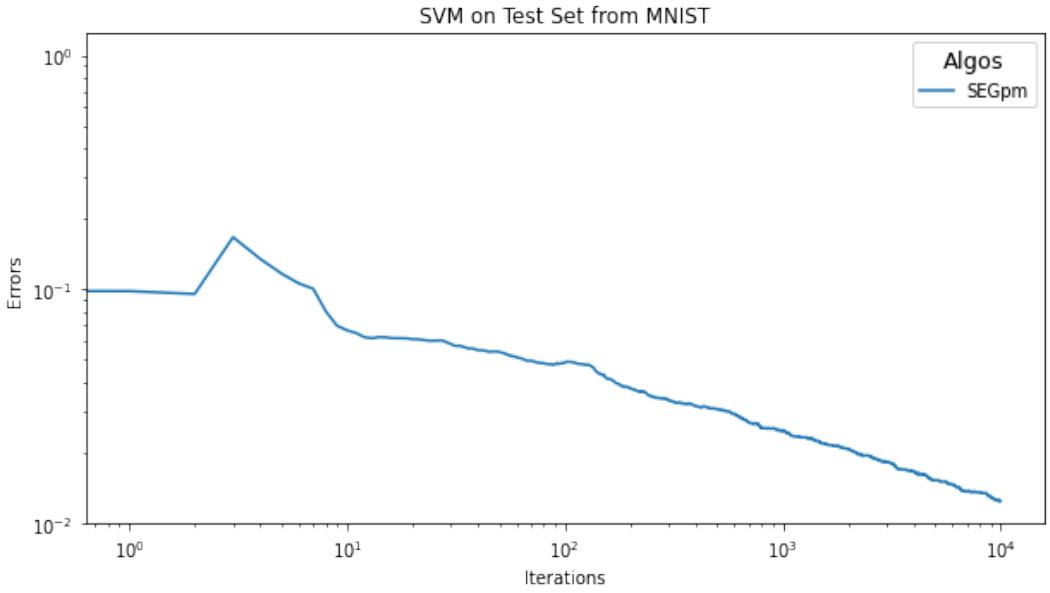

Figure 11: Stochastic Exponentiated Gradient +/-

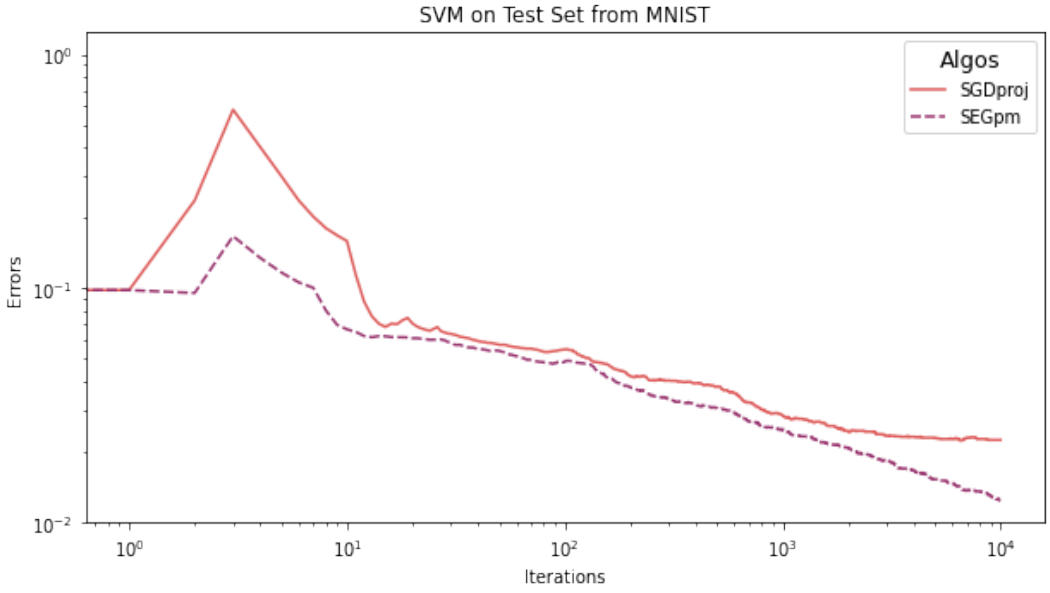

Figure 12: Projected SGD vs SEG +/-

L'algorithme ADAproj s'exécute en 1.8 min et obtient une erreur finale de 0.0276 comparé à l'algorithme SGD proj qui s'exécute en 1.41 min avec une erreure finale de 0.0224.
Dans ce cas on préfère l'algorithme SGDproj en dépit d'une grosse erreur lors des 10 premières itérations.

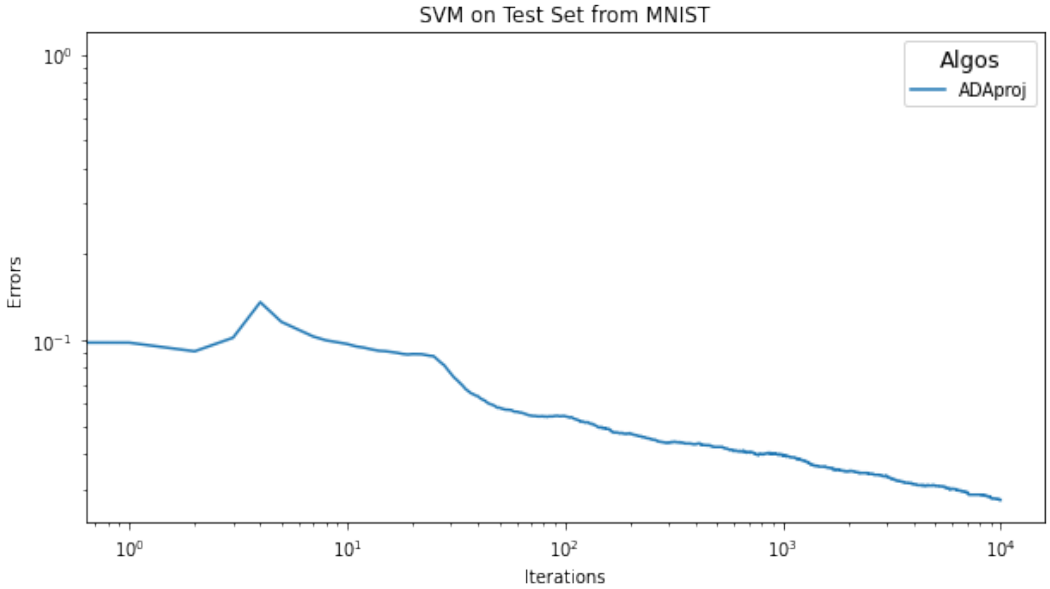

Figure 13: Adaproj

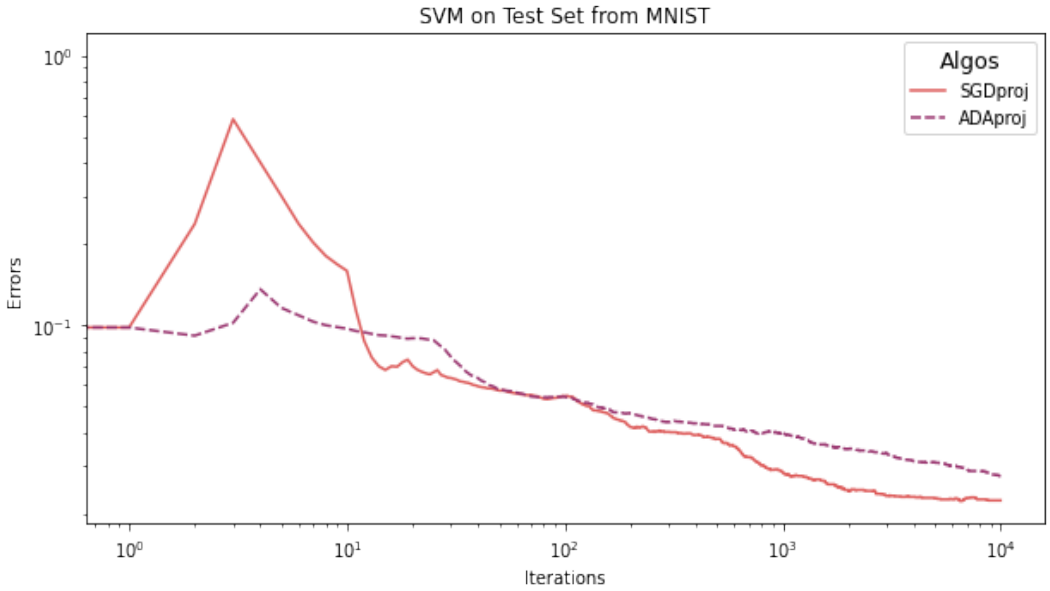

Figure 14: Projected SGD vs Adaproj

1.4 Online Newton Step

L'algorithme ONS s'exécute en 13.98 min (malgré l'utilisation de calcul pour simplifier l'inversion de matrice) qui est similaire au temps d'exécution des méthodes non stochastiques et obtient une erreur finale de 0.0227.
Parmi tout les algorithmes stochastiques vus jusqu'à présent on préfère SGE +/- qui est de loin le meilleur. ONS se compare à SGDproj et le pire de la liste se trouve être ADAproj.

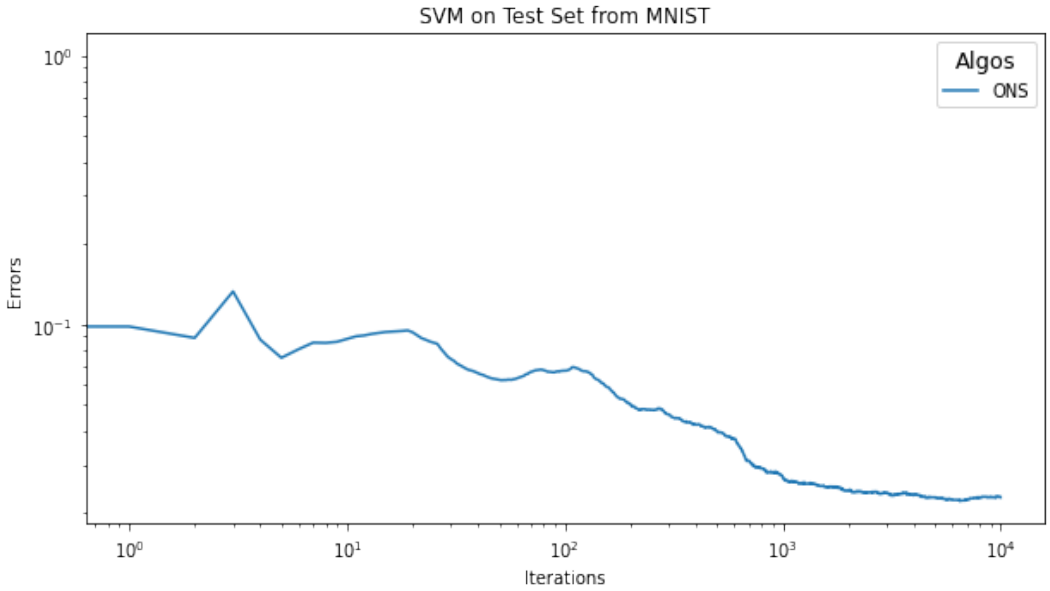

Figure 15: Stochastic Online Newton Step

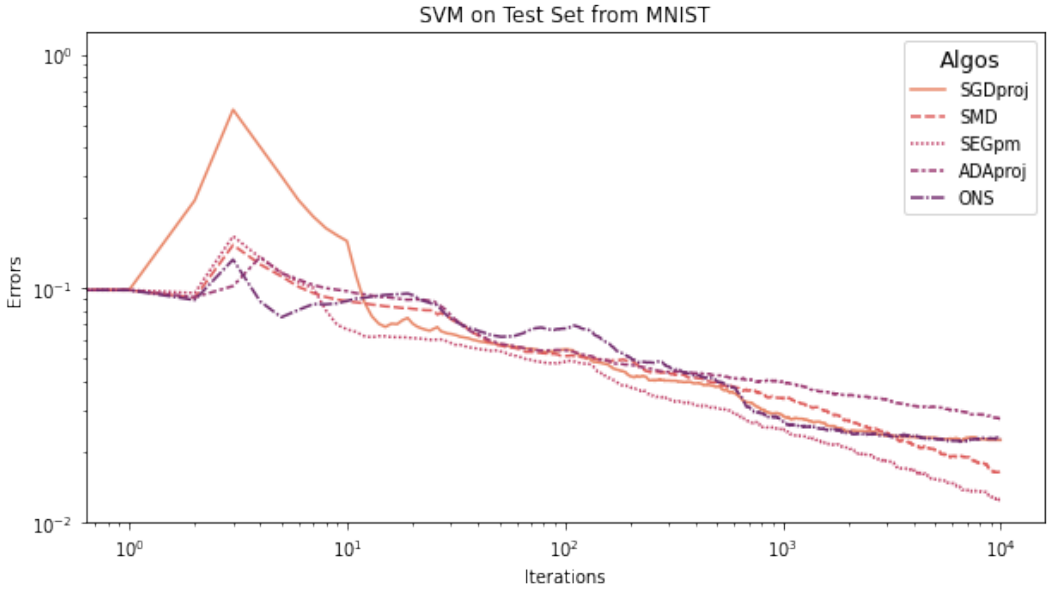

Figure 16: Projected SGD vs SMD vs SEG+/- vs Adaproj vs ONS

1.5 Exploration Methods

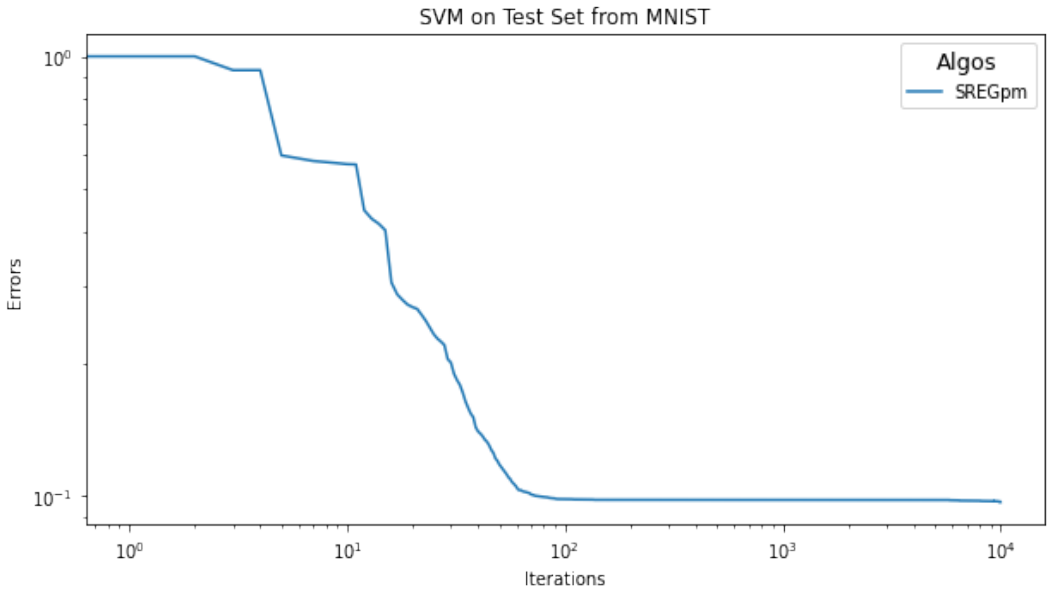

Figure 17: Stochastic Randomized Exponentiated Gradient +/-

Après avoir testé plusieurs valeurs de rayon on n'arrive pas à faire converger cet algorithme.

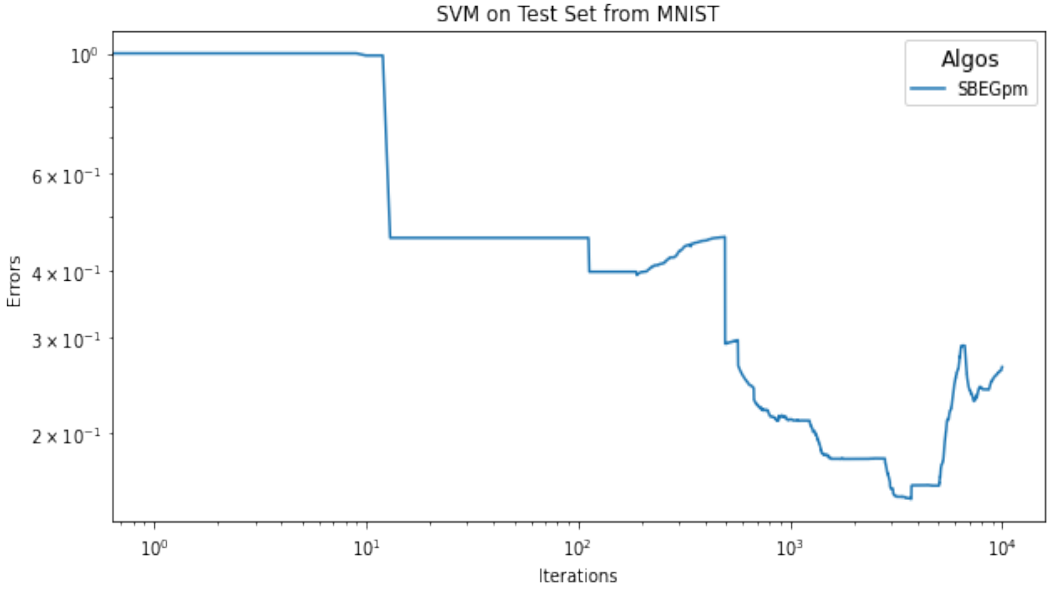

Figure 18: Stochastic Bandit Exponentiated Gradient +/-

Pour l'algorithme Exp2 on utilise $\eta_t = \frac{1}{\sqrt{2dt}}$, $z = 100$ et $\gamma = \frac{1}{8}$. Après plusieurs tests ce sont les paramètres qui semblent permettre une convergence mais on verra que ce n'est pas le cas.
En utilisant $\eta_t = 0.003$ et $\gamma = \frac{1}{8}$ on peut obtenir une meilleure valeur aux alentours de la 1000 ème itération mais on observe par la suite un décrochage.

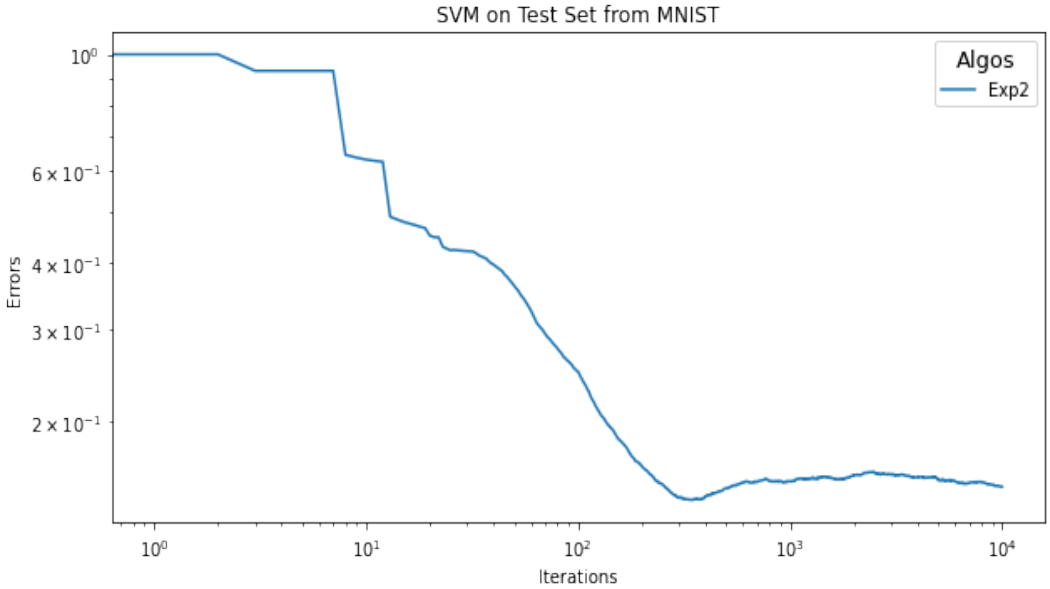

Figure 19: EXP 2

En résumé le seul algorithme exploratoire qui converge est l'algorithme SREG +/-, on dirait qu'on arrive presque à faire converger l'algorithme Exp2 et on n'y arrive pas pour SBEG +/- .

On va étudier les bornes théoriques pour ces algorithmes. Dans notre exemple de MNIST o n a $d = 784$.

Pour l'algorithme Exp2 on devrait observer à partir de $T > 8dlog(2d) = 46146.6$ un regret en $O(\sqrt{T})$. Dans notre cas T vaut au maximum 10 000. On réalise un test avec $T = 100000$. L'algorithme ne converge toujours pas.

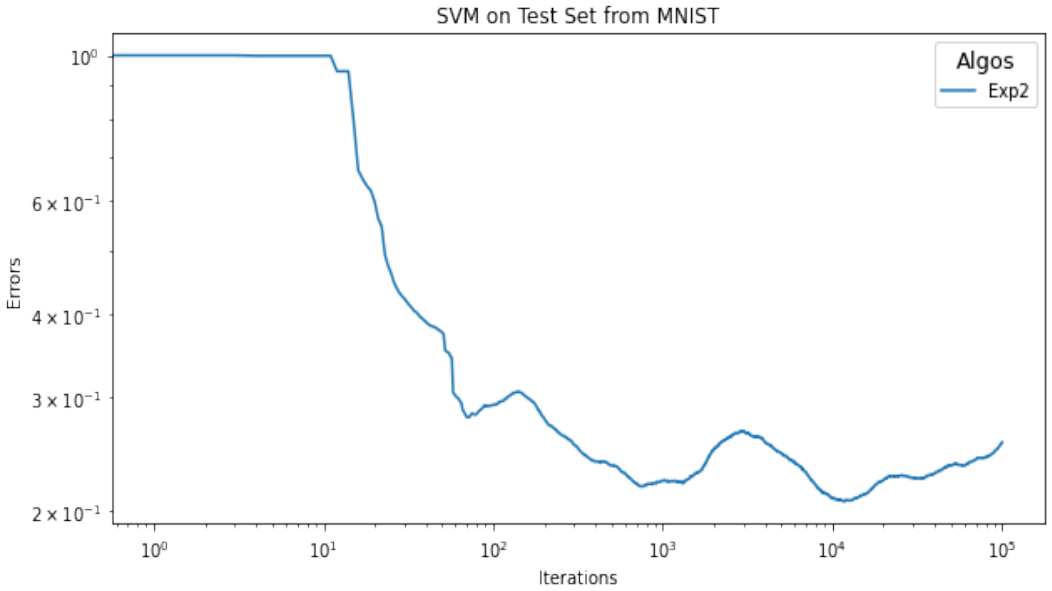

Figure 20: Exp2 avec $T = 100000$

Pour l'algorithme SBEG +/- le regret théorique doit être de l'ordre de $O(\frac{1}{\sqrt{T}})$.

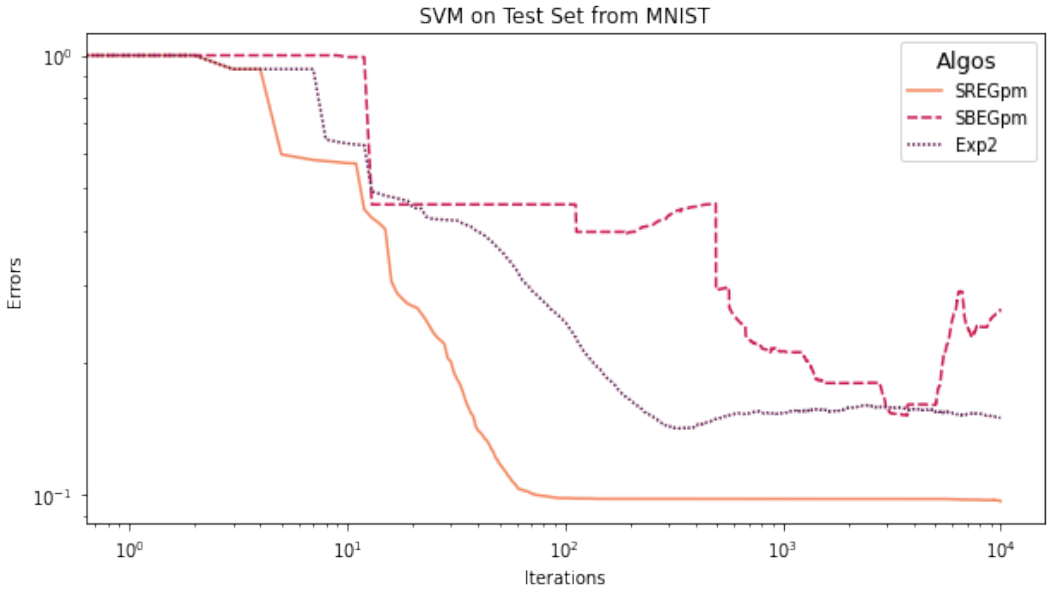

Figure 21: Algorithmes d'exploration

1.6 Synthèse

On présente ici la synthèse de tout les algorithmes vu jusqu'ici.

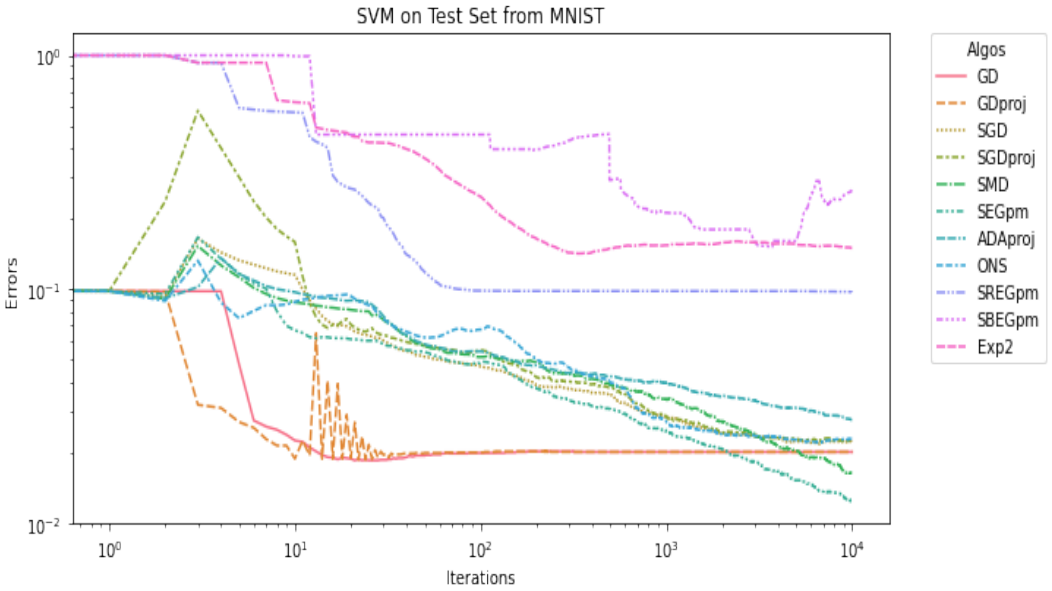

Figure 22: Comparaison des algorithmes

Algorithmes	Temps (min)	Erreur final	Erreur min
GD	12.85	0.0201	0.0185
GDproj	12.79	0.0201	0.0186
SGD	1.32	0.0221	.0218
SGDproj	1.41	0.0224	0.0223
SMD	1.44	0.0164	0.0163
SEGpm	1.37	0.0124	0.0124
Adaproj	1.8	0.0276	0.0276
ONS	13.98	0.0227	0.0220
SREGpm	1.4	0.0970	0.0970
SBEGpm	1.37	0.1517	0.2652
Exp2	1.35	0.1706	0.0943

Table 1: Résultats et temps d'exécution

2 Article : Adaptative Subgradient Methods

Nous avons choisi de traiter l'article *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization* (2011) de Duchi, Hazan et Singer. Cet article introduit l'algorithme AdaGrad (*Adaptive Gradient Algorithm*) que nous avons abordé en cours.

2.1 Motivations

Lorsque l'on se place dans le contexte de l'apprentisage séquentiel, il est courant d'avoir des données de grande dimension et parcimonieuses (*sparse*), c'est-à-dire comportant un grand nombre de valeurs nulles. Pourtant, les méthodes de descente usuelles suivent pour la plupart un schéma ne tenant pas compte de la structure des données. Les algorithmes introduits dans l'article cherchent à tenir compte de la géométrie des données observée au fil des itérations afin d'effectuer un apprentissage plus efficace. De manière informelle, le principe essentiel est d'associer d'une part des taux d'apprentissage très faibles aux attributs apparaissant fréquemment et d'autre part des taux d'apprentissage élevés aux attributs rares. L'idée est de ne pas négliger l'apparition d'un attribut rare au cours de l'apprentissage, car il arrive souvent que de tels attributs soient porteurs de beaucoup d'information.

Cette approche semble particulièrement adaptée à l'ensemble de données MNIST, et Adagrad devrait pouvoir exploiter la parcimonie des pixels correspondant aux chiffres dans les images pour réaliser un apprentissage efficace.

2.2 Configuration & Notations

Dans le cours on supposait que toutes nos fonctions étaient différentiables, ce qui justifiait l'utilisation de la notation ∇f pour le gradient d'une fonction.

Cependant, dans le cas d'une fonction non différentiable, on note $g \in \partial f(x)$ un sous-gradient de f en $x \in X \subseteq \mathbb{R}^d$.

On définit $g_{1:t} := [g_1 \dots g_t]$ matrice des sous-gradients des fonctions f_t évaluées en x_t.

On définit également la matrice $G_t = \sum_{k=1}^{t} g_k g_k^T$ dans $\mathbb{R}^{d \times d}$.

Au lieu de réaliser une simple descente de gradient projeté sans poids on

réalise la descente de gradient projeté avec poids $G_t^{\frac{1}{2}}$:

$$x_{t+1} = \Pi_X^{G_t^{\frac{1}{2}}}(x_t - \eta G_t^{-\frac{1}{2}} g_t)$$

avec $\Pi_X^A(y) = argmin_{x \in X} ||x-y||_A$. Il s'agit de la version dite "Full-Matrice".

Quand d est grand on a un souci de calcul pratique pour le calcul de $G_t^{\frac{1}{2}}$ et de son inverse. Pour contourner ce problème on étudie pluôt la version diagonale

$$x_{t+1} = \Pi_X^{Diag(G_t)^{\frac{1}{2}}}(x_t - \eta Diag(G_t)^{-\frac{1}{2}} g_t)$$

On pose $\phi_t := f_t + \varphi$ avec f_t et φ des fonctions convexes.
On définit le regret de ϕ_t avec x^* point optimal :

$$\begin{aligned} Regret_\phi(T) &= \sum_{t=1}^{T} [\phi_t(x_t) - \phi_t(x^*)] \\ &= \sum_{t=1}^{T} [f_t(x_t) + \varphi(x_t) - f_t(x^*) - \varphi(x^*)] \end{aligned}$$

Le but est de minimiser ce regret afin d'avoir un regret sous-linéaire.

Méthode 1 : Primal-Dual Subgradient

$$x_{t+1} = \arg\min_{x \in X} \left\{ \eta \langle \bar{g}_t, x \rangle + \eta \varphi(x) + \frac{1}{t} \psi_t(x) \right\},$$

où $\bar{g}_t = \frac{1}{t} \sum_{k=1}^{t} g_k$, x_1 minimiseur de φ et ψ_t fortement convexe différentiable qu'on appelle terme proximal.

Méthode 2 : Forward-Backward Splitting

$$x_{t+1} = \arg\min_{x \in X} \{ \eta \langle g_t, x \rangle + \eta \varphi(x) + B_{\psi_t}(x, x_t) \}$$

Avec B_{ψ_t} la divergence de Bregman en ψ_t qui est fortement convexe et différentiable.

On se donne H_t une matrice symétrique telle que $H_t \succeq 0$ et on pose alors $\psi_t(x) = \langle x, H_t x \rangle$.
Le choix de la matrice H_t nous permet d'établir deux algorithmes : AdaGrad Diagonal (matrices diagonales) et AdaGrad Full (matrices entières).

$$H_t = \delta I + Diag(G_t)^{\frac{1}{2}} \;\; et \;\; H_t = \delta I + G_t^{\frac{1}{2}}$$

Par construction de notre matrice H_t on a des algorithmes similaires à ceux de descente de gradient du second ordre.

2.3 Garanties théoriques

Nous nous restreignons à l'étude de l'algorithme diagonal pour des raisons de simplicité et d'explicabilité. Soit $(x_t)_t$ la suite définie par l'algorithme diagonal avec $\delta > max_t||g_t||_\infty$.

1. Primal-dual subgradient update :

$$Regret_\phi(T) \leq \frac{\delta}{\eta}||x^*||_2^2 + ||x^*||_\infty^2 \sum_{i=1}^{d} ||g_{1:T,i}||_2 + \eta \sum_{i=1}^{d} ||g_{1:T,i}||_2.$$

2. FBS update :

$$Regret_\phi(T) \leq \frac{1}{2\eta} \max_{1\leq T} ||x^* - x_t||_\infty^2 \sum_{i=1}^{d} ||g_{1:T,i}||_2 + \eta \sum_{i=1}^{d} ||g_{1:T,i}||_2.$$

Ces bornes sont peu exploitables, c'est pourquoi nous émettons quelques hypothèses simplificatrices :

1. On suppose X compact ;
2. On pose $D_\infty = sup_{x\in X}||x - x^*||_\infty$;
3. On définit la quantité :

$$\gamma_T \triangleq \sum_{i=1}^{d} ||g_{1:T,i}||_2 = \inf_s \left\{ \sum_{i=1}^{T} \langle g_t, diag(s)^{-1} g_t \rangle : \langle 1, s \rangle \leq \sum_{i=1}^{d} ||g_{1:T,i}||_2, s \succeq 0 \right\}.$$

Nous aboutissons alors aux bornes suivantes :

1. Primal-dual subgradient update :

$$Regret_\phi(T) \leq 2||x^*||_\infty \gamma_T + \delta \frac{||x^*||_2^2}{||x^*||_\infty} \leq 2||x^*||_\infty \gamma_T + \delta ||x^*||_1.$$

2. FBS update :

$$Regret_\phi(T) \leq \sqrt{2} D_\infty \sum_{i=1}^{d} ||g_{1:T,i}||_2 = \sqrt{2} D_\infty \gamma_T.$$

2.4 Implémentation

Nous détaillons dans cette partie un peu plus les implémentations pour les algorithmes AdaGrad version Diagonale avec projection et régularisation en reprenant les notations du cours.

η et z correspondent respectivement au learning rate et au rayon de la boule.

1. Projection :

$$\begin{aligned} \nu_{t+1} &= -\eta t D^{-1} \bar{g}_t && (PD) \\ \nu_{t+1} &= x - \eta D^{-1} g_t && (FBS) \\ x_{t+1} &= \underset{x \in B_1(z)}{\arg\min} ||x - \nu_{t+1}||_D \end{aligned}$$

2. Régularisation ℓ^2 :

$$\begin{aligned} u_{t+1} &= \eta t \bar{g}_t && (PD) \\ u_{t+1} &= \eta g_t - D x_t && (FBS) \\ x_{t+1} &= -D^{-1}(u + \alpha(\theta)) \end{aligned}$$

avec $\alpha(\theta)$ solution du problème dual obtenu par minimisation de $\langle u, x\rangle + \frac{1}{2}\langle x, Dx\rangle + \lambda||x||_2$ (Voir le code pour plus de détails).

Rappel :

$$\partial||.||_2(0) = B(0,1)$$

2.5 Résultats

Pour ces implémentations on utilise $\eta_t = \frac{1}{\sqrt{t}}$ au lieu de prendre un learning rate fixe.

PD et FBS font référence aux méthodes Primal-Dual et ForwardBackward Splitting.
ADAproj correspond à l'algorithme où $\varphi = 0$ et on projette sur la boule ℓ^1.
ADA .2 correspond à l'algorithme avec régularisation ℓ^2 avec $\varphi = \lambda||.||_2$

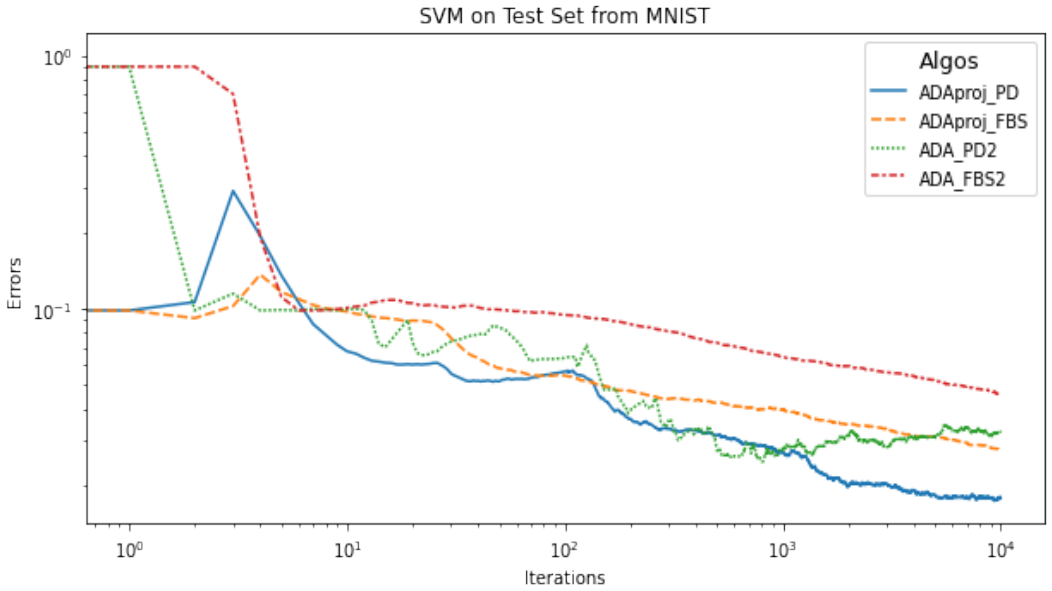

Figure 23: AdaGrad Diagonale

Algorithmes	Temps (min)	Erreur finale	Erreur min
ADAproj PD	2.84	0.0177	0.0173
ADAproj FBS	1.89	0.0276	0.0276
ADA PD2	12.19	0.0324	0.0248
ADA FBS2	10.75	0.0456	0.0456

Table 2: Résultats et temps d'exécution d'AdaGrad

Parmi ces versions d'AdaGrad la meilleure est clairement ADAproj PD avec un temps d'execution rapide qui est simple à mettre en place et avec la plus petite erreur sur le test set de MNIST.

Remarque : ADAproj FBS correspond à l'algorithme du cours ADAproj.

On compare maintenant avec les algorithmes stochastiques vus au début du rapport. On observe que SEG +/- reste le meilleur mais est suivi de très près par notre version d'AdaGrad ADAproj PD. Après la 1000ème itération SEG +/- détrône notre algorithme et on aperçoit également qu'aux environs de la 10 000 ème itération SMD fait un peu mieux que cette version d'AdaGrad.

Le finetuning de l'hyperparamètre η pourrait améliorer sensiblement la convergence et peut-être détroner nos autres algorithmes.

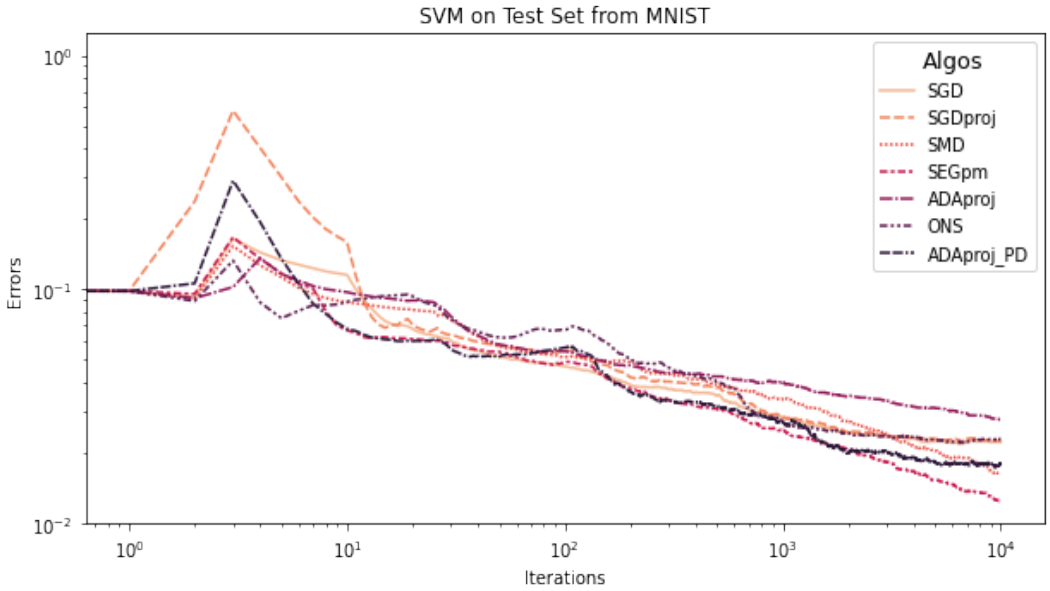

Figure 24: Comparaison d'ADAproj PD et des algos stochastiques

3 Synthèse

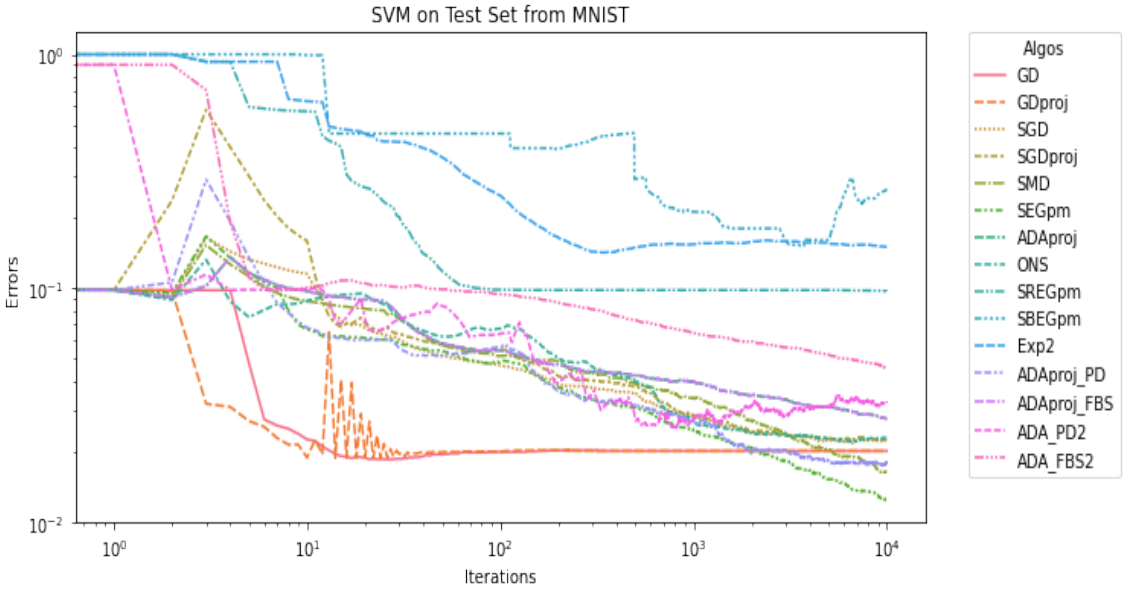

Figure 25: Erreurs de tout nos algorithmes

Algorithmes	Temps (min)	Erreur finale	Erreur min
GD	12.85	0.0201	0.0185
GDproj	12.79	0.0201	0.0186
SGD	1.32	0.0221	.0218
SGDproj	1.41	0.0224	0.0223
SMD	1.44	0.0164	0.0163
SEGpm	1.37	0.0124	0.0124
Adaproj	1.8	0.0276	0.0276
ONS	13.98	0.0227	0.0220
SREGpm	1.4	0.0970	0.0970
SBEGpm	1.37	0.1517	0.2652
Exp2	1.35	0.1706	0.0943
ADAproj PD	2.84	0.0177	0.0173
ADAproj FBS	1.89	0.0276	0.0276
ADA PD2	12.19	0.0324	0.0248
ADA FBS2	10.75	0.0456	0.0456

Table 3: Résultats et temps d'exécution