- Optimisation Convexe Séquentielle - Projet

Paul Liautaud, Nicolas Olivain & Lise Le Boudec
M2A

January 4, 2022

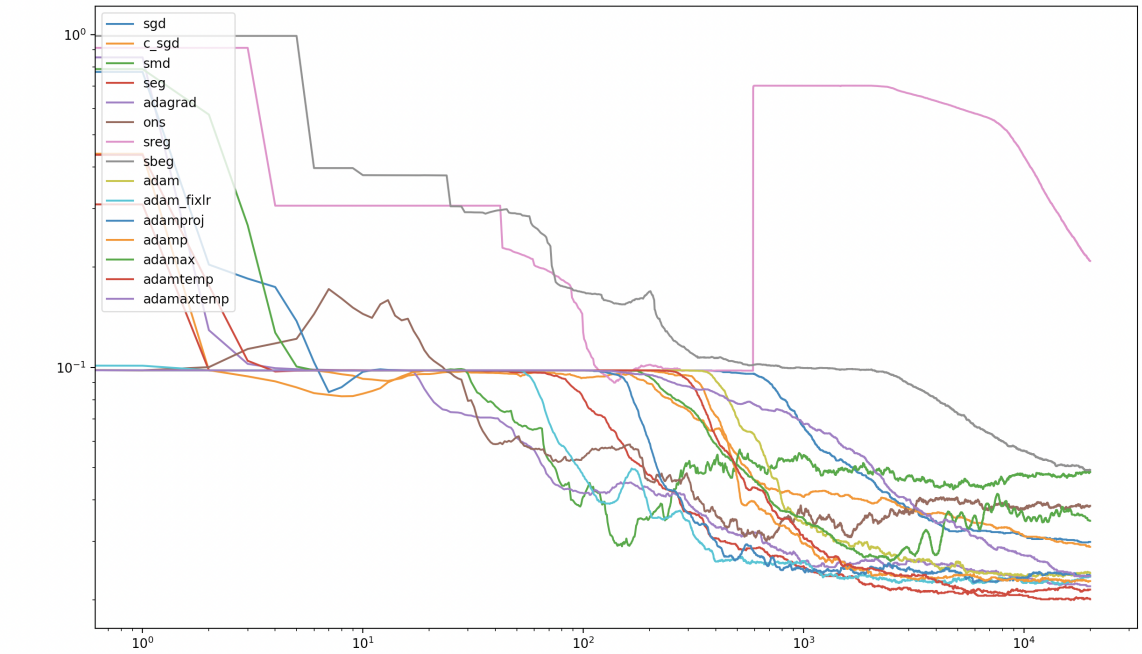

Figure 1: Comparaison des erreurs commises par chacun des algorithmes présentés dans ce projet, évalués sur un ensemble de test `MNIST_test.csv`.[1][2]

Contents

[1] GD et GD projetés ne sont pas présentés sur cette figure, dû au temps de calcul qu'ils nécessitent, mais le sont à la section 1.

[2] Plus d'informations et d'autres courbes pour l'algorithme `SREG` sont présentées à la section 5. En particulier, d'autres hyper-paramètres sont proposés et permettent de faire converger cet algorithme.

Introduction

De nos jours, le monde est confronté à une augmentation sans précédent du volume et de la vitesse des flux de données disponibles. De nombreuses applications doivent passer de méthodes dites "hors ligne" à des méthodes séquentielles capables d'acquérir, d'adapter et de traiter les données à la volée. Dans le même temps, les données deviennent de plus en plus sophistiquées, rendant alors parfois les hypothèses statistiques traditionnelles obsolètes (par exemple des distributions uniformes, indépendantes). La conception d'algorithmes efficaces capables d'apprendre des données au fur et à mesure qu'elles arrivent, avec le moins d'hypothèses possible devient dès lors un défi majeur pour l'apprentissage automatique. Exploiter le potentiel de ces flux de données en temps réel est l'objectif de l'apprentissage en ligne et la motivation de la construction de ces algorithmes.

Ce document présente le projet en lien avec le cours d'*Optimisation Convexe Séquentielle*, [6]. Sauf mention contraire, les résultats et pseudo-codes présentés en sont issus. Les documents relatifs à ce cours sont disponibles à l'adresse : `http://wintenberger.fr/ens.html`.

On se propose alors d'implémenter et d'étudier différents algorithmes d'optimisation en ligne connus dans la littérature mathématique actuelle. Pour ce faire, nous chercherons à résoudre un problème de classification binaire sur le jeu de données `MNIST` (téléchargeable en format `csv` à l'adresse : `https://pjreddie.com/projects/mnist-in-csv/`). `MNIST` est un ensemble d'images sur lesquelles des chiffres ont été écrits à la main. De ce fait, le problème étudié consistera en la détection de 0 contre les autres chiffres. Le jeu de données comprend une partie *d'entraînement* qui comprend 60000 entrées et une partie de *test* comprenant 10000 données. Ce sont sur ces données de test que les performances des algorithmes seront évaluées. Ainsi, les 0 seront encodés par la valeurs 1 et les autres digits par la valeur -1. Ce sont ces labels qui seront utilisés dans le cadre du problème de classification binaire considéré.

Le problème de classification sera résolu en utilisant un SVM linéaire. Notons a_i les données et b_i les labels associés, l'objectif est de résoudre le problème appelé *Hard-Margin Problem* :

$$\min_x \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\mathrm{sgn}(x^T a_i \neq b_i)}.$$

Afin de "convexifier" le problème et de profiter pleinement des propriétés des algorithmes étudiés, la fonction de coût est convexifiée en utlisant une formule relaxée de ce problème appelé *Soft-Margin Problem*. Pour cela, une version convexe de la loss $0-1$ est utilisée (*hinge loss*) et un terme de régularisation est ajouté afin d'obtenir une fonction de coût qui soit fortement convexe. En conclusion, le problème considéré est le suivant :

$$\min_x f(x) = \min_x \frac{1}{n}\sum_{i=1}^{n} \mathrm{hinge}(b_i x^T a_i) + \frac{\lambda}{2}||x||^2,$$

où λ est un hyper paramètre à régler et qui modélise le compromis entre obtenir un classifieur parcimonieux et minimiser la fonction de perte et $hinge(x) = \max(0, 1-x)$.

Les prévisions du modèle sont obtenues en test en calculant

$$\hat{b} = \mathrm{sgn}(\hat{x}^T a).$$

où $\hat{x}$ est l'estimateur obtenu par un algorithme d'optimisation et a est une entrée du jeu de données (en test), sur laquelle la prédiction du label est effectuée.

Dans la plupart des algorithme, la dérivée de la fonction de perte est utilisée :

$$\nabla_x f(x) = -ba\mathbb{1}_{1-bx^T a<0} + \lambda x \tag{1}$$

1 Gradient Descent

Algorithm 1: Gradient Descent

Parameters: Epoch T, step-sizes (η_t).
Initialization: Initial point $x_1 \in \mathcal{K}$.
For each iteration $t = 1, \ldots, T$:
Iteration: Update

$$y_{t+1} = x_t - \eta_t \nabla f(x_t),$$
$$x_{t+1} = \Pi_{\mathcal{K}}(y_{t+1})\,.$$

Return x_{T+1}

Figure 2: Pseudo-code de l'algorithme de descente de gradient

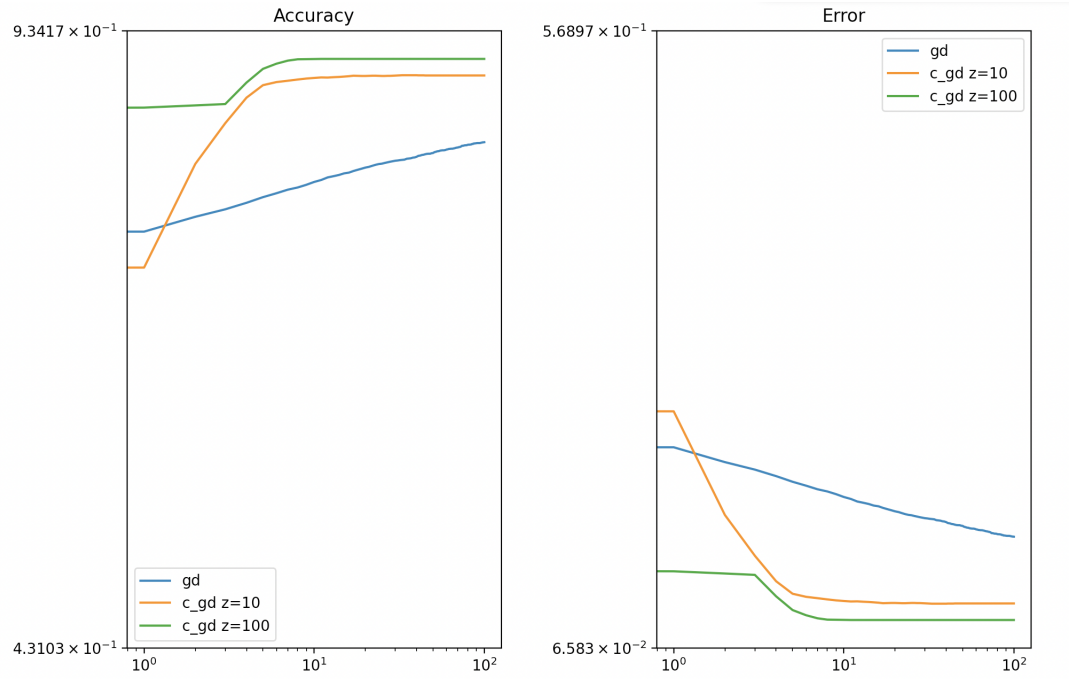

Figure 3: Descente de gradient classique et avec projection sur $\mathcal{B}_{\ell_1}(z = 10)$ et $\mathcal{B}_{\ell_1}(z = 100)$, pour 100 époques. On a considéré un coefficient de régularisation $\lambda = \frac{1}{3}$, qui implique alors un pas d'apprentissage de $\frac{1}{\lambda t}$.

Dans Figure 3, on remarque l'intérêt de la projection sur la boule ℓ_1 à chaque itération. Se ramener à des itérations au sein de la boule convexe permet nettement d'accroître la rapidité d'apprentissage et d'atteindre de meilleures performances via une meilleure convergence. Il est aussi intéressant de noter l'influence du rayon z de l'espace de projection considéré : dans le cadre de la Gradient Descent, pour un pas d'apprentissage de $\frac{1}{\lambda t}$, un rayon $z = 100$ permet une meilleure convergence que pour $z = 10$.

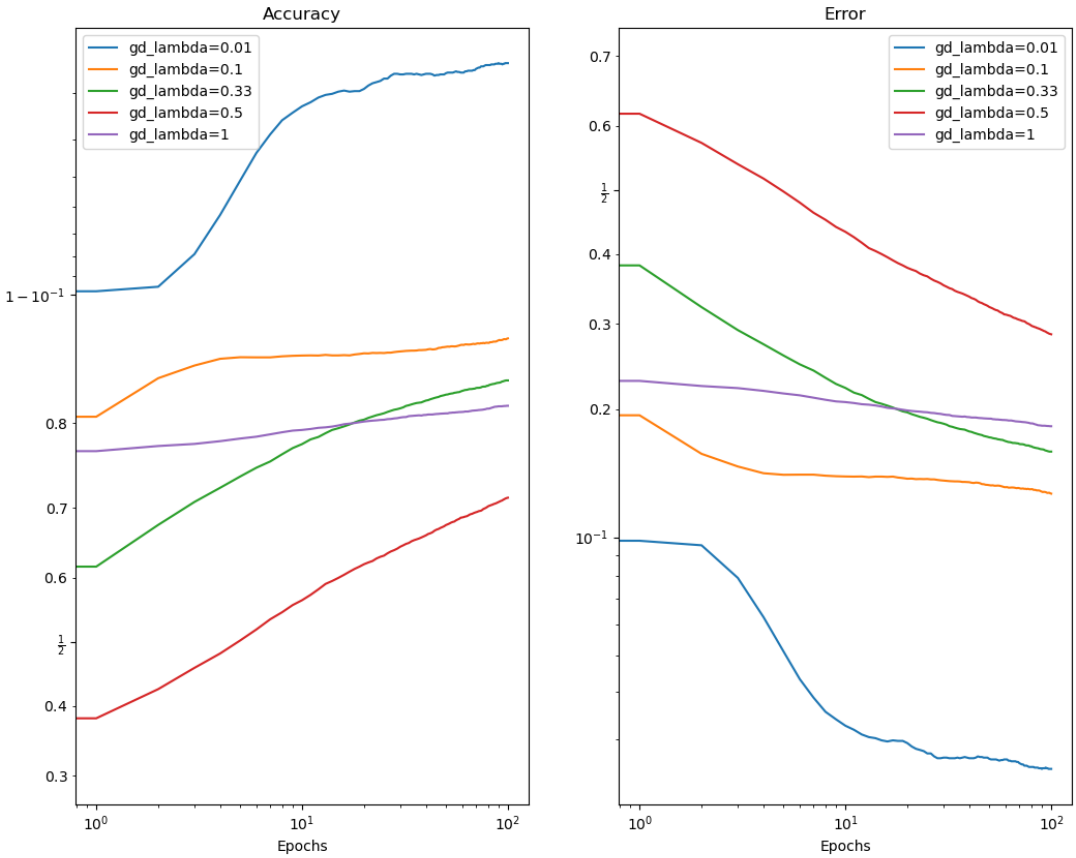

Figure 4: Descente de gradient classique pour différentes valeurs de lambda entraînés sur 100 epochs. On observe que diminuer la régularisation tends à augmenter les performances.

2 Stochastic Gradient Descent

Algorithm 6: SGD for linear SVM.

Parameters: Epoch T, radius $z > 0$, regularization parameter $\lambda > 0$
Initialization: Initial point $x_1 = 0$.
Sample uniformly iid: $(I_t)_{1 \leq t \leq T}$ from $\{1 \leq i \leq n\}$
For each iteration $t = 1, \ldots, T$:
Iteration: Update

$$y_{t+1} = (1 - 1/t)x_t - \frac{\nabla \ell_{a_{I_t}, b_{I_t}}(x_t)}{\lambda t},$$
$$x_{t+1} = \Pi_{B_1(z)}(y_{t+1}).$$

Return: $\overline{x}_{T+1} = \dfrac{1}{T+1} \displaystyle\sum_{t=1}^{T+1} x_t$

Figure 5: Pseudo-code de l'algorithme de descente de gradient stochastique appliqué aux SVM linéaires

Dans Figure 6, on remarque à nouveau dans le cas stochastique l'intérêt de la projection sur la boule ℓ_1 à chaque itération.
Dans tous les cas (projection ou non), en plus de raccourcir le temps d'exécution, l'approche SGD est clairement meilleure dès les premières époques d'entraînement de notre modèle (cf courbes rouge et verte). Dans le cas de la GD projetée et de la SGD projetée, la différence est moins notable, mais la SGD est tout de même plus performante à partir de 10 époques (cf courbe rouge vs orange).

Comparaison en temps/complexité Ici on retrouve bien un coût de calcul en temps (cf. histogramme dans Figure 6) et complexité largement supérieur pour la Gradient Descent que pour la Stochastic Gradient Descent, et pour cause : le calcul du gradient est fait, à chaque itération, sur chaque coordonnée dans un cas, contre une seule dans l'autre cas. La projection sur la boule ℓ_1 implique par ailleurs elle aussi un coût P, qui dépend évidemment de la dimension d de l'espace des données considéré : $P = \mathcal{O}(d \log(d))$.
On se retrouve alors avec les complexités suivantes, à chaque itération $t = 1, \cdots, T$ (avec T nombre d'époques) : $\mathcal{O}(n \times d + P)$ pour la Gradient Descent projetée, contre $\mathcal{O}(d + P)$ dans le cas de la Stochastic Gradient Descent projetée.

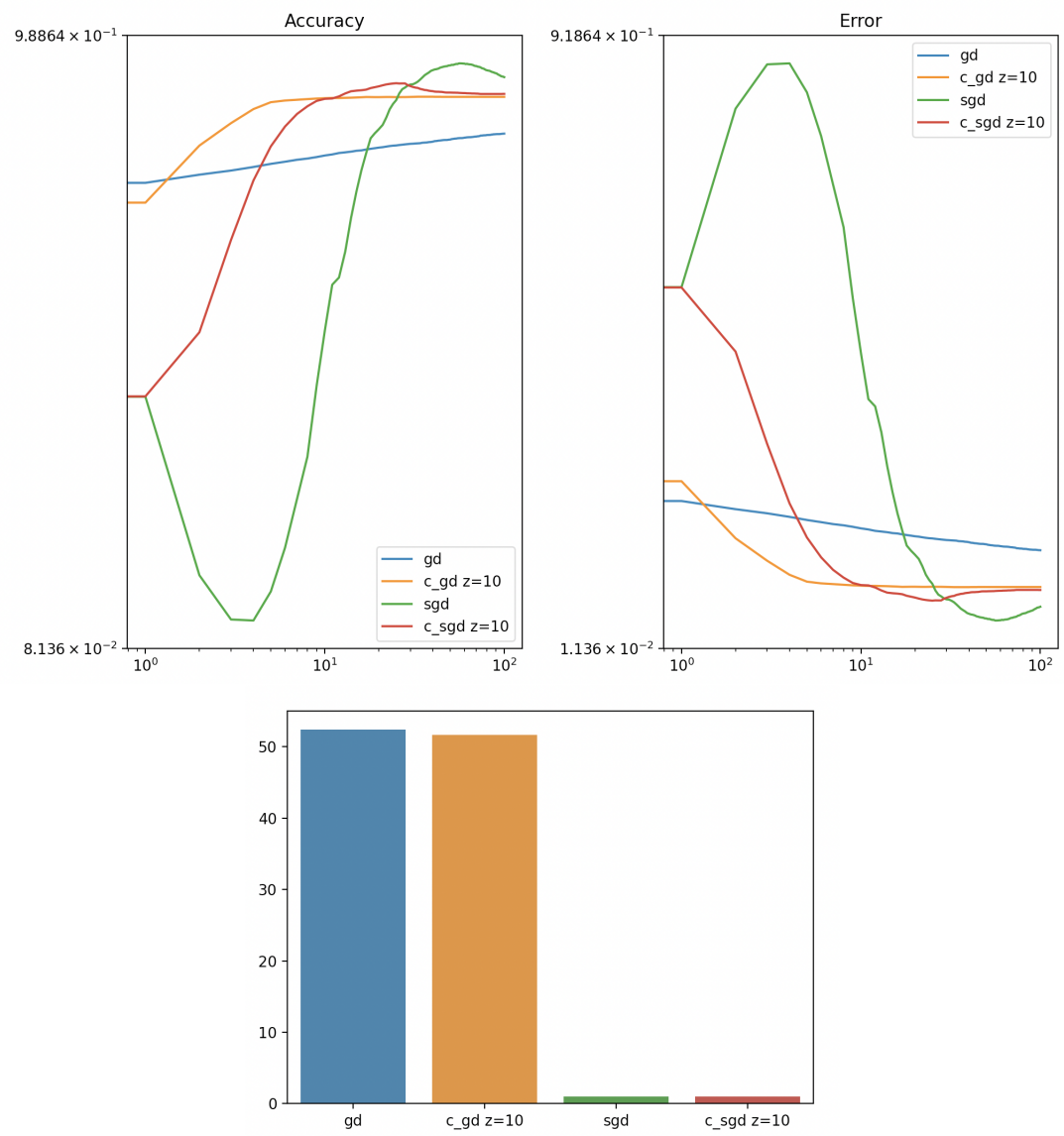

Figure 6: Descente de gradient classique et stochastique pour 2 projections, à savoir $\mathcal{B}_{\ell_1}(z = 10)$ et $\mathcal{B}_{\ell_1}(z = 100)$. Le modèle a été entraîné sur 100 époques.

3 Regularized Follow The Leader

Algorithm 10: SMD for linear SVM

Parameters: Epoch T, radius $z > 0$
Initialization: Initial point $x_1 = y_1 = 0$.
Sample uniformly iid: $(I_t)_{1 \leq t \leq T}$ from $\{1 \leq i \leq n\}$
For each iteration $t = 1, \dots, T$:
Iteration: Update

$$\begin{aligned} \eta_t &= 1/\sqrt{t}\,, \\ y_{t+1} &= y_t - \eta_t \nabla \ell_{a_{I_t}, b_{I_t}}(x_t), \\ x_{t+1} &= \Pi_{B_1(z)}(y_{t+1})\,. \end{aligned}$$

Return: $\overline{x}_{T+1} = \dfrac{1}{T+1} \displaystyle\sum_{t=1}^{T+1} x_t$

Figure 7: Pseudo-code de l'algorithme stochastic mirror descent appliqué aux SVM linéaires

Algorithm 14: SEG+/- for linear SVM

Parameters: Epoch T, radius $z > 0$.
Initialization: Initial point $x_1 = 0$, weights $w = 1/(2d)\,\mathbb{1}$ and $\theta_1 = 0 \in \mathbb{R}^{2d}$.
Sample uniformly iid: $(I_t)_{1 \leq t \leq T}$ from $\{1 \leq i \leq n\}$
For each iteration $t = 1, \dots, T$:
Iteration: Update

$$\begin{aligned} \eta_t &= \sqrt{1/t} \\ \theta_{t+1} &= \theta_t - \eta_t \pm \nabla \ell_{a_{I_t}, b_{I_t}}(x_t)\,, \\ w_{t+1} &= \frac{\exp(\theta_{t+1})}{\sum_{i=1}^{2d} \exp(\theta_{t+1,i})}\,, \\ x_{t+1,i} &= z(w_{t+1,i} - w_{t+1,i+d})\,, \qquad 1 \leq i \leq d\,. \end{aligned}$$

Return: $\overline{x}_{T+1} = \dfrac{1}{T+1} \displaystyle\sum_{t=1}^{T+1} x_t$

Figure 8: Pseudo-code de l'algorithme stochastic exponentiated gradient +/- appliqué aux SVM linéaires

Ici, dans Figure 10, on remarque que pour 1000 époques, une projection avec rayon $z = 100$ dans Adagrad semble bien plus performante (courbe orange contre bleue). Si l'on pousse l'entraînement plus loin, on remarque par contre qu'un rayon plus étroit, $z = 10$ entraîne une meilleure convergence !

Algorithm 16: Adagrad for linear SVM

Parameters: Epoch T, radius $z > 0$.
Initialization: Initial point $x_1 = y_1 = 0$ and $S_0 = 0$ (or $= \delta \mathbb{1}$ small).
Sample uniformly iid: $(I_t)_{1 \leq t \leq T}$ from $\{1 \leq i \leq n\}$
For each iteration $t = 1, \dots, T$:
Iteration: Update

$$
\begin{aligned}
S_t &= S_{t-1} + \nabla \ell_{a_{I_t}, b_{I_t}}(x_t)^2 \\
D_t &= \operatorname{Diag}(\sqrt{S_t}) \\
y_{t+1} &= x_t - D_t^{-1} \nabla \ell_{a_{I_t}, b_{I_t}}(x_t)\,, \\
x_{t+1} &= \arg\min_{x \in B_1(z)} \|x - y_{t+1}\|_{D_t}^2\,, \qquad 1 \leq i \leq d\,.
\end{aligned}
$$

Return: $\overline{x}_{T+1} = \dfrac{1}{T+1} \displaystyle\sum_{t=1}^{T+1} x_t$

Figure 9: Pseudo-code de l'algorithme Adagrad appliqué aux SVM linéaires

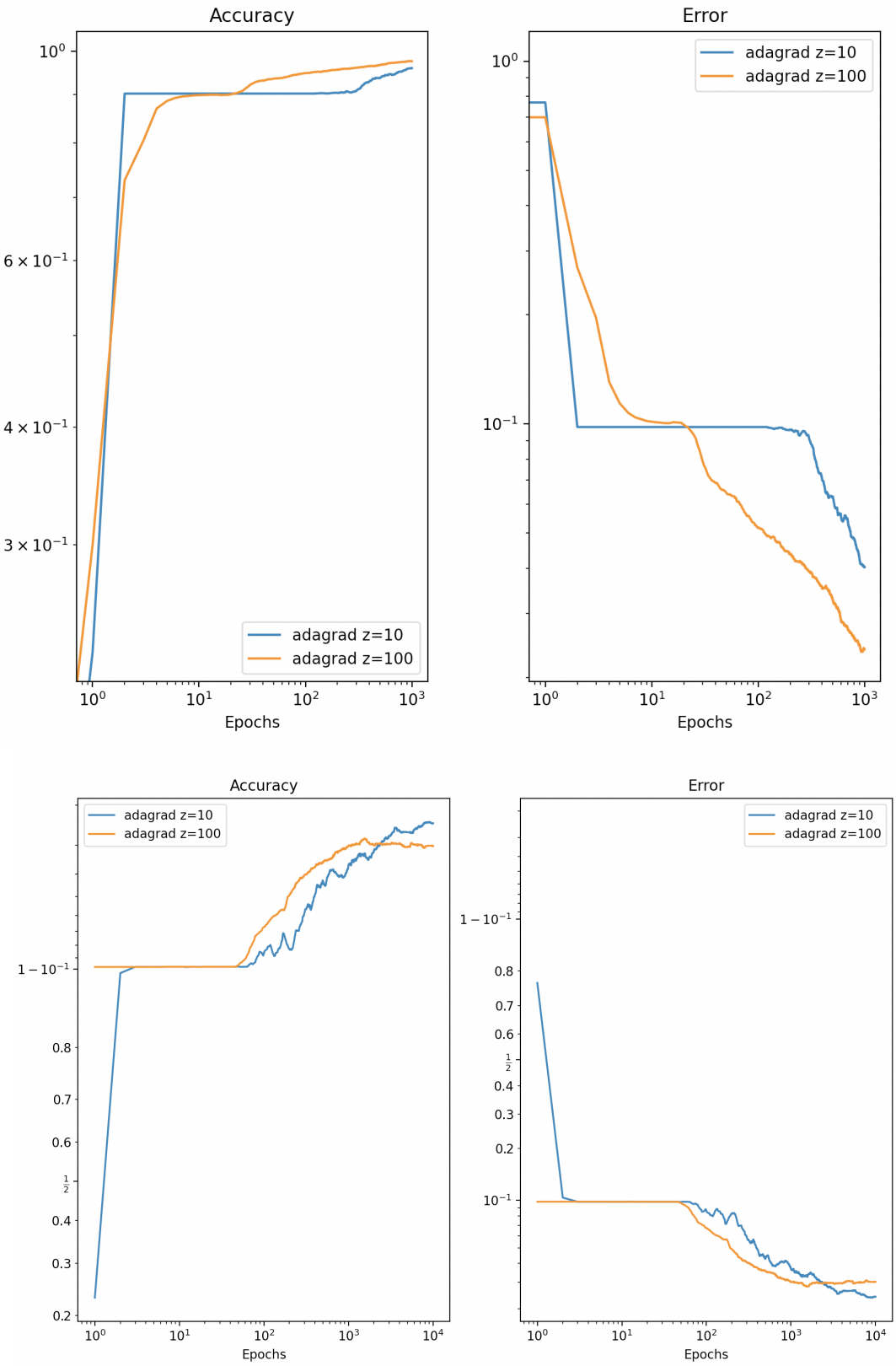

Figure 10: Adagrad avec projections sur $\mathcal{B}_{\ell_1}(z = 10)$ et $\mathcal{B}_{\ell_1}(z = 100)$, pour 1000 et 10000 époques.

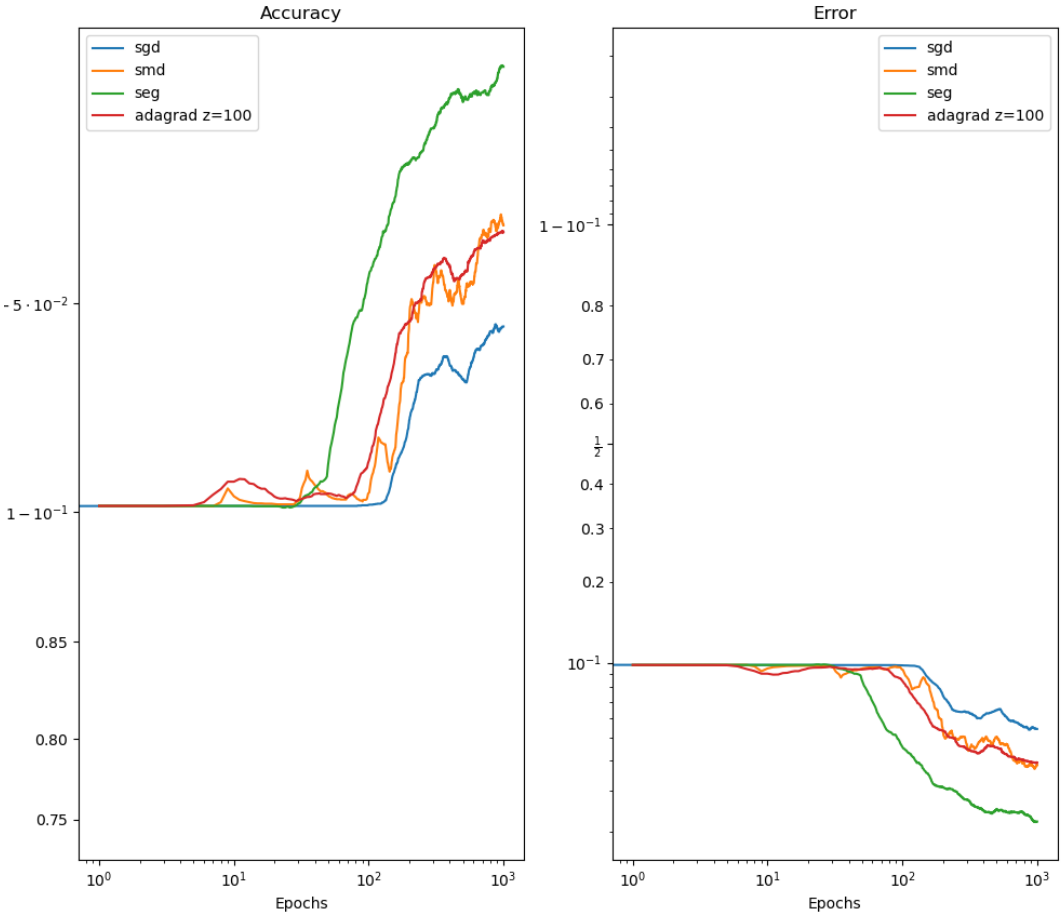

Figure 11: Comparaison des performances de SGD, SMD, SEG et Adagrad sur 1000 epochs, z=100 et learning rate=$1/\lambda * t$.

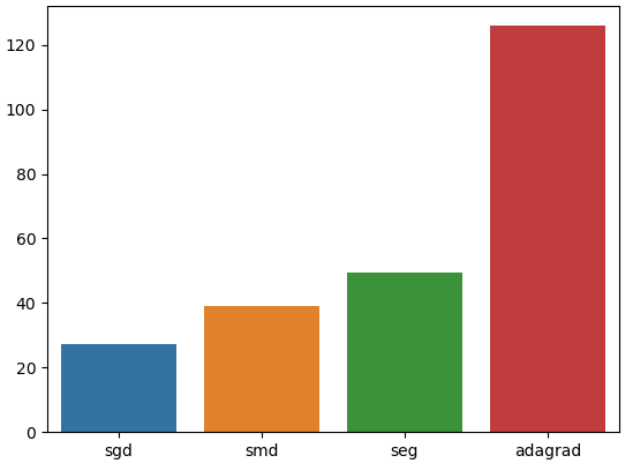

Figure 12: Comparaison des temps d'exécution de SGD, SMD, SEG et Adagrad sur 1000 epochs en secondes.

4 Online Newton Step

Online Newton Step ressemble à une version agile d'`Adagrad` (diminutif de Adaptive gradient), qui a été étudié plus haut. En l'espèce, on rappelle qu'`Adagrad` est une amélioration de la méthode `SGD` vue aussi précédemment, qui détermine maintenant automatiquement un taux d'apprentissage pour chaque paramètre. On réfère en particulier à l'excellent cours [3], qui introduit `ONS`. `Adagrad` était en particulier basé sur une descente de la fonction de régularisation R, où elle était supposée doublement différentiable : $\nabla^2 R = \mathrm{diag}(S)$ pour $S = (s_1, \cdots, s_d)$. Ici, `ONS` y serait semblable en considérant une fonction de régularisation adaptative : $R_t(x) = \frac{\lambda}{2}\|x - x_1\|_{A_t}$, où A_t est définie ci-après.
`ONS` consiste maintenant en une adaptation séquentielle de la méthode de NEWTON-RAPHSON qui, elle, fait ses preuves dans un contexte d'optimisation convexe classique en utilisant le pas itératif suivant :

$$x_{t+1} = x_t = \eta \nabla^2 f(x_t) \nabla f(x_t).$$

Ici `ONS` approche ainsi le

Algorithm 21: ONS for linear SVM

Parameters: Epoch T, radius $z > 0$, regularization parameter $\lambda > 0$ and $\gamma > 0$.
Initialization: Initial point $x_1 = y_1 = 0$, $A_0 = 1/\gamma^2 I_d$ and $A_0^{-1} = \gamma^2 I_d$.
Sample uniformly iid: $(I_t)_{1 \le t \le T}$ from $\{1 \le i \le n\}$
For each iteration $t = 1, \ldots, T$:
Iteration: Update

$$\begin{aligned}
\nabla_t &= \nabla \ell_{a_{I_t}, b_{I_t}}(x_t) + \lambda x_t \\
A_t &= A_{t-1} + \nabla_t \nabla_t^T \\
A_t^{-1} &= A_{t-1}^{-1} - \frac{A_{t-1}^{-1} \nabla_t \nabla_t^T A_{t-1}^{-1}}{1 + \nabla_t^T A_{t-1}^{-1} \nabla_t} \\
y_{t+1} &= x_t - \frac{1}{\gamma} A_t^{-1} \nabla_t \,, \\
x_{t+1} &= \arg\min_{x \in B_1(z)} \|x - y_{t+1}\|_{A_t}^2 \,, \qquad 1 \le i \le d \,.
\end{aligned}$$

Return: $\overline{x}_{T+1} = \dfrac{1}{T+1} \displaystyle\sum_{t=1}^{T+1} x_t$

Figure 13: Pseudo-code de l'algorithme Online Newton Step appliqué aux SVM linéaires

Les figures 14 et 15 comparent les performances et temps d'exécution de `ONS` à `SGD` et à `Adagrad` sur 1000 époques. On observe en terme de temps d'exécution que comme pour `Adagrad`, `ONS` demande plus de temps que `SGD` dû à l'inversion de la matrice A dans à chaque itération. En terme de performance, `ONS` est apparu comme relativement sensible aux hyperparamètres comme `Adagrad`, tout particulièrement au paramètre γ. Son entraînement est parfois chaotique. La version présentée ici, avec $z = 100$ et $\gamma = 1/8$ présente des performances correctes mais inférieures à `Adagrad`.

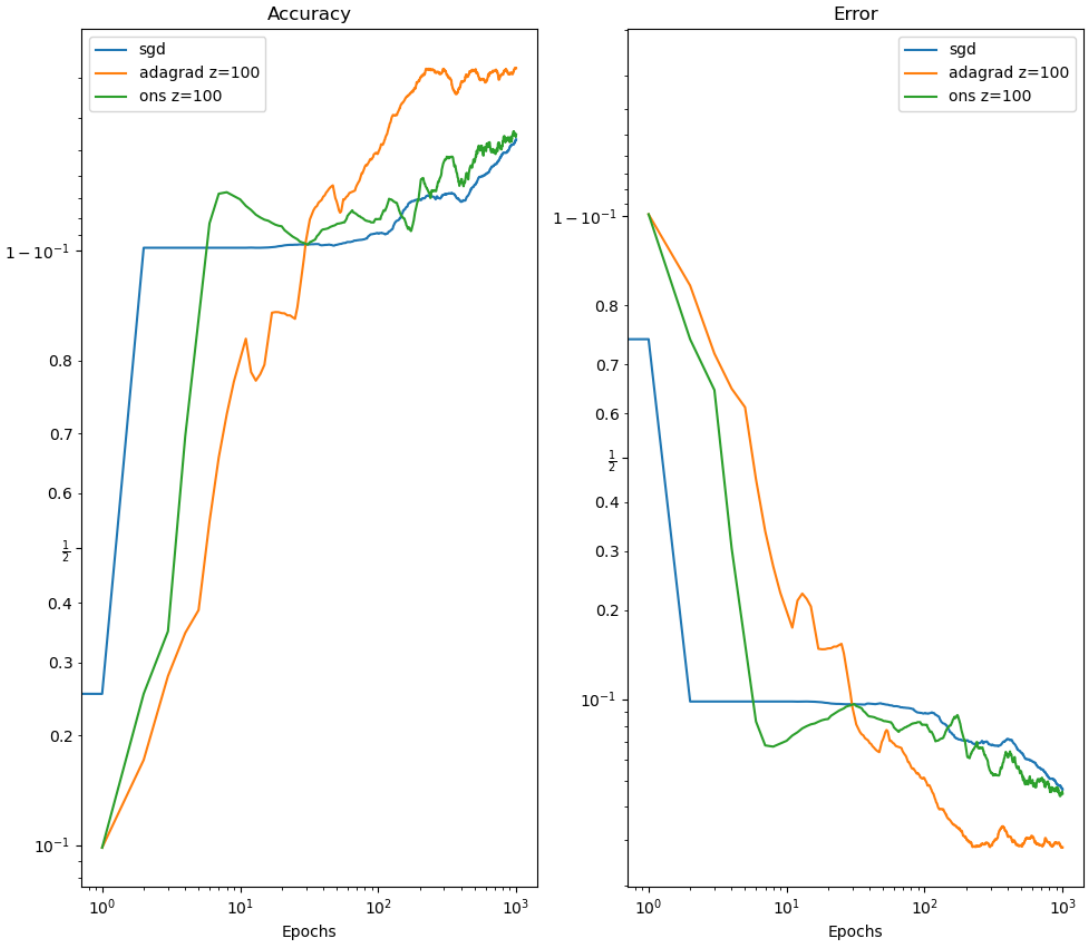

Figure 14: Comparaison des performances de `SGD`, `Adagrad`et `ONS` sur 1000 époques, $z = 100$ et $\gamma = 1/8$

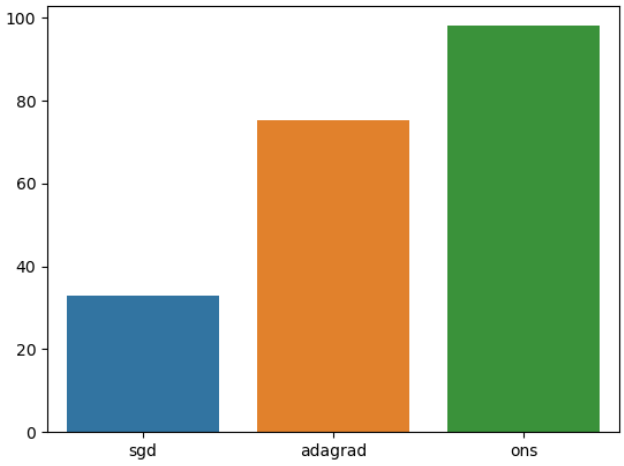

Figure 15: Comparaison des temps d'exécution de `SGD`, `Adagrad`et `ONS` sur 1000 époques, $z = 100$ et $\gamma = 1/8$

5 Exploration Methods

Dans cette section, nous nous intéressons aux performances des algorithmes SREG et SBEG issus de [3]. Le pseudo code de ces algorithmes est détaillé dans les figures 16 et 17 respectivement.

Algorithm 23: SREG+/- for linear SVM

Parameters: Epoch T, radius $z > 0$.
Initialization: Initial point $x_1 = 0$, weights $w_1 = 1/(2d)\,\mathbb{1}$ and $\theta_1 = 0 \in \mathbb{R}^{2d}$.
Sample uniformly iid: $(I_t)_{1\leq t\leq T}$ from $\{1 \leq i \leq n\}$
For each iteration $t = 1, \dots, T$:
Sample a direction: $J_t \in \{1, \dots d\}$ uniformly
Iteration: Update

$$\begin{aligned}
\eta_t &= 1/\sqrt{dt} \\
w_{t,J_t} &\leftarrow \exp(-\eta_t d \nabla \ell_{a_{I_t}, b_{I_t}}(x_t)_{J_t}) w_{t,J_t}\,, \\
w_{t,J_t+d} &\leftarrow \exp(\eta_t d \nabla \ell_{a_{I_t}, b_{I_t}}(x_t)_{J_t}) w_{t,J_t+d}, \\
w_{t+1} &= \frac{w_t}{\sum_{i=1}^{2d} w_{t,i}} \\
x_{t+1,i} &= z(w_{t+1,i} - w_{t+1,i+d})\,, \qquad 1 \leq i \leq d\,.
\end{aligned}$$

Return: $\overline{x}_{T+1} = \dfrac{1}{T+1}\displaystyle\sum_{t=1}^{T+1} x_t$

Figure 16: Pseudo-code de l'algorithme Stochastic Randomized Exponentiated Gradient +/- appliqué aux SVM linéaires

Algorithm 26: SBEG+/- for linear SVM

Parameters: Epoch T, radius $z > 0$.
Initialization: Initial point $x_1 = 0$, weights $w_1 = 1/(2d)\,\mathbb{1}$ and $\theta_1 = 0 \in \mathbb{R}^{2d}$.
Sample uniformly iid: $(I_t)_{1\leq t\leq T}$ from $\{1 \leq i \leq n\}$
For each iteration $t = 1, \dots, T$:
Sample an action: $A_t \in \{\pm e_1, \dots, \pm e_d\} \sim w_t$ that determines a coordinate J_t and a sign $\pm_t \in \{+1, -1\}$
Iteration: Update

$$\begin{aligned}
\eta_t &= 1/\sqrt{dt} \\
w_{t,J_t+d\mathbb{1}(\pm_t=-1)} &\leftarrow \exp(-\pm_t \eta_t \nabla \ell_{a_{I_t}, b_{I_t}}(x_t)_{J_t} / w_{t,J_t+d\mathbb{1}(\pm_t=-1)})\,, \\
w_{t+1} &= \Big(1 - \eta_t\Big) \frac{w_t}{\sum_{i=1}^{2d} w_{t,i}} + \frac{\eta_t}{2d} \\
x_{t+1,i} &= z(w_{t+1,i} - w_{t+1,i+d})\,, \qquad 1 \leq i \leq d\,.
\end{aligned}$$

Return: $\overline{x}_{T+1} = \dfrac{1}{T+1}\displaystyle\sum_{t=1}^{T+1} x_t$

Figure 17: Pseudo-code de l'algorithme Stochastic Bandit Exponentiated Gradient +/- appliqué aux SVM linéaires

5.1 Une première variante

Devant la difficulté à faire converger les algorithmes dans leur version classique, commençons tout d'abord par considérer une approche personnalisée dont les résultats sont présentés en Figure 18 : nous considérons une actualisation des poids $w \in [0; 1]^{2d}$ différence de celle préconisée dans les pseudo-codes.
En effet, une des différences est l'utilisation de la fonction exponentielle, qui sera non plus appliquée seulement à la composante I_t choisie à l'itération t (comme dans 16 et 17) mais plutôt, à chaque époque, sur *chacune* des composantes. En pseudo-code, on fournit alors les versions suivantes en 1 et 2.

Remarque pour `SBEG` **modifié** Dans notre version 2 de `SBEG`, les tirages du signe $\pm_t$ est uniforme dans $\{\pm 1\}$, donc suivant une loi de Bernoulli centrée $\mathcal{B}(1/2)$. Les coordonnées supérieures $\{w_i, 1 \leq i \leq d\}$ et inférieures $\{w_i, d+1 \leq i \leq 2d\}$ sont donc actualisées avec une probabilité égale et constante au cours du

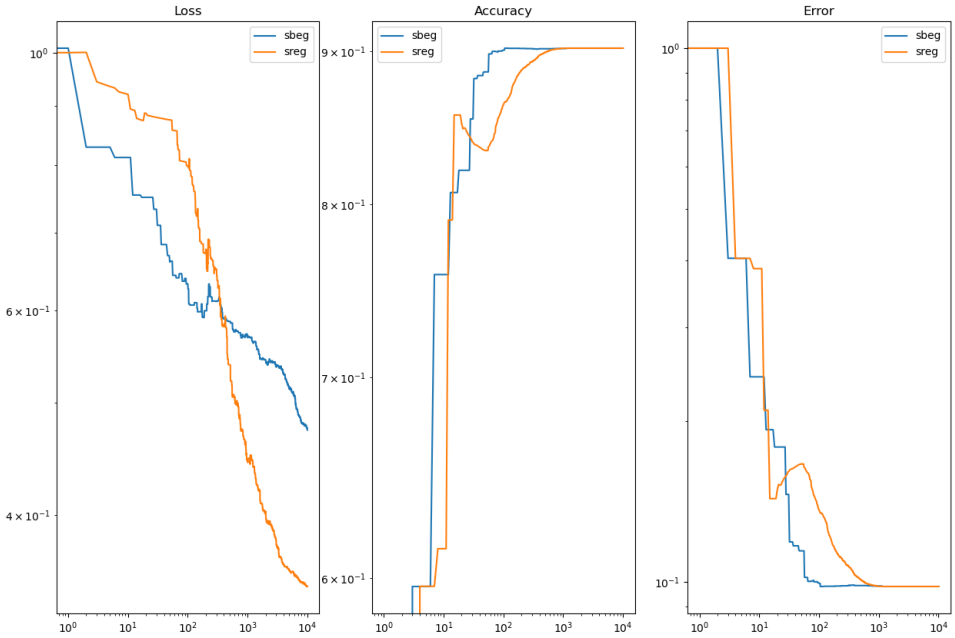

Figure 18: Performances de SREG et SBEG en loss, accuracy et erreur. On considère ici un entraînement sur 10000 époques et un coefficient de régularisation $\lambda = \frac{1}{3}$.

Algorithm 1 SREG modifié - [Le Boudec, Olivain, Liautaud]

Paramètres : Époches T, rayon $z > 0$.
Initialisation : Point initial $x_1 = 0$, poids initial $w_1 = 1/(2d)\mathbb{1}$.
Tirage uniforme et indépendant : $(I_t)_{1 \leq t \leq T}$ dans $\{1, \cdots, n\}$.
Pour chaque itération $t = 1, \cdots, T$:
Tirage d'une direction : $J_t \in \{1, \cdots, d\}$ uniformément.
Itération: Mise à jour :

$$\eta_t = \frac{1}{\sqrt{t}}$$
$$w_{t,J_t} \leftarrow w_{t,J_t} - \eta_t \nabla(\ell_{a_{I_t}, b_{I_t}}(x_t))_{J_t}$$
$$w_{t,J_t+d} \leftarrow w_{t,J_t+d} + \eta_t (\nabla \ell_{a_{I_t}, b_{I_t}}(x_t))_{J_t}$$
$$w_{t+1} = \frac{\exp(w_t)}{\sum_{i=1}^{2d} \exp(w_{t,i})} \quad \text{(SoftMax)}$$
$$x_{t+1,i} = z(w_{t+1,i} - w_{t+1,i+d}), \quad \forall 1 \leq i \leq d.$$

Sortie : $\bar{x}_{T+1} = \frac{1}{T+1} \sum_{t=1}^{T+1} x_t$.

Algorithm 2 SBEG modifié - [Le Boudec, Olivain, Liautaud]

Paramètres : Époches T, rayon $z > 0$.
Initialisation : Point initial $x_1 = 0$, poids initial $w_1 = 1/(2d)\mathbb{1}$.
Tirage uniforme et indépendant : $(I_t)_{1 \leq t \leq T}$ dans $\{1, \cdots, n\}$.
Pour chaque itération $t = 1, \cdots, T$:
Tirage d'une direction : $J_t \in \{1, \cdots, d\}$ uniformément.
Tirage d'un signe : $\pm_t \in \{\pm 1\}$ uniformément.
Itération: mise à jour :

$$\eta_t = \frac{1}{\sqrt{t}}$$
$$w_{t,J_t+d\mathbb{1}\{\pm_t=-1\}} \leftarrow w_{t,J_t} - \pm_t \times \eta_t \nabla(\ell_{a_{I_t},b_{I_t}}(x_t))_{J_t+d\mathbb{1}\{\pm_t=-1\}}$$
$$w_{t+1} = (1-\eta_t)\frac{\exp(w_t)}{\sum_{i=1}^{2d}\exp(w_{t,i})} + \frac{\eta_t}{2d} \quad \text{(SoftMax + Start Entropy with } w_1\text{)}$$
$$x_{t+1,i} = z(w_{t+1,i} - w_{t+1,i+d}), \quad \forall 1 \leq i \leq d.$$

Sortie : $\bar{x}_{T+1} = \frac{1}{T+1}\sum_{t=1}^{T+1} x_t$.

temps.
On fournit les performances de ces premières variantes dans Figure 18, pour une durée d'entraînement de $T = 10000$ époques, $\lambda = 1/3, z = 10$. Les algorithmes convergent tous deux, avec une belle courbe en apprentissage.
Néanmoins, on observe que les 2 algorithmes ont des performances limitées même pour un nombre d'époques élevé (10000). En effet, l'algorithme `SREG`-modifié comme `SBEG`-modifié atteint un plateau en fin d'entraînement, en convergeant vers une erreur d'environ 0.1, sans amélioration postérieure. Cette erreur est notamment assez mauvaise, puisque dans le modèle étudié ici elle correspondrait à un *classifieur aléatoire* : notre dataset est composé de 10% de label 1, c'est à dire que la classe 0 représente 10% des données d'entraînement et de test.
L'algorithme est donc (trop) *explorateur* (random) à chaque itération, et ne semble pas avoir trouvé un équilibre *exploration/exploitation* comme il serait souhaitable d'avoir.

La méthode `Exp2` se propose plutôt d'effectuer un tirage *séquentiel* et *adaptatif* puisqu'au lieu de jouer une action au hasard uniformément comme dans 2, on effectue plutôt un tirage selon la distribution exponentielle moyenne pondérée w_t à l'instant t.

5.2 Implémentation de `SREG` et `SBEG`

Remarque importante sur `SBEG` Dans le bloc d'itération, lors de la phase d'actualisation des poids , on considérera plutôt la mise à jour suivante :

$$w_{t,J_t+d\mathbb{1}\{\pm_t=-1\}} \leftarrow \exp(-\pm_t \eta_t \nabla l_{a_{I_t},b_{I_t}}(x_t)_{J_t}) w_{t-1,J_t+d\mathbb{1}\{\pm_t=-1\}}. \tag{2}$$

La division itérative par les poids, au sein de l'exponentielle, comme suggérée par le pseudo-code en 17 ne permet pas une bonne stabilité et encore moins une convergence convenable. En effet, cela conduit à une explosion des poids, et, par conséquent, à forcer la sortie des w du simplexe Λ très rapidement. La version implémentée sera alors fidèle à l'algorithme `Exp2` (voir [6]), en utilisant (2). Cette version conduit à de très bonnes performances, comme en témoigne la suite.
Il convient également de citer l'excellente référence [1], où la section 2.1 "Expanded Exponential Weights" (`EXP2`) explicite très bien le principe de l'optimisation combinatoire en ligne adapté dans le cadres des MAB (Multi Armed Bandits).

En jouant avec les hyperparamètres (et notamment le coefficient radius z), on observe une assez bonne convergence en terme d'erreur de prédiction (i.e. faible) pour `SREG`. Ceci est illustré dans la Figure 19, où en particulier le cas $z = 100$ atteint une belle performance, pour $\lambda = 1/3$ en coefficient de régularisation. Pour `SBEG`, il en est de même et on obtient les résultats en Figure 20.

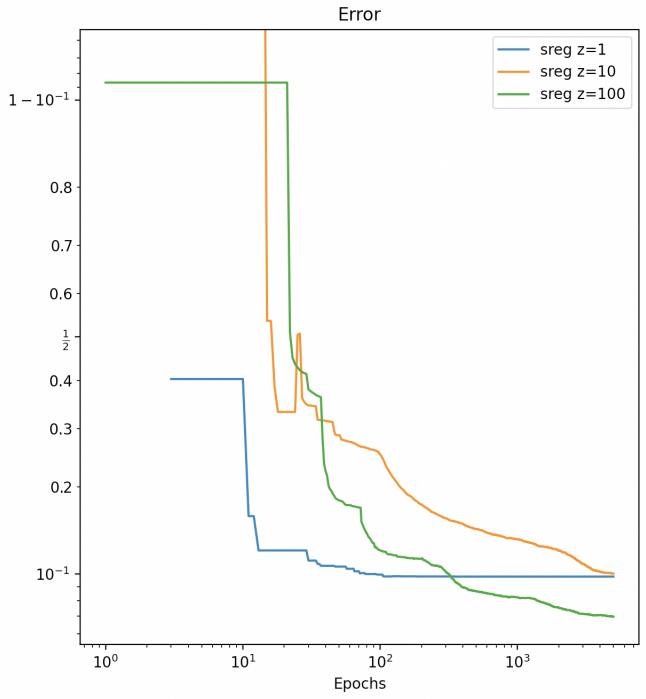

Figure 19: Erreur de SREG pour différents $z = 1, 10, 100$ au cours de l'entraînement sur 10000 époques.

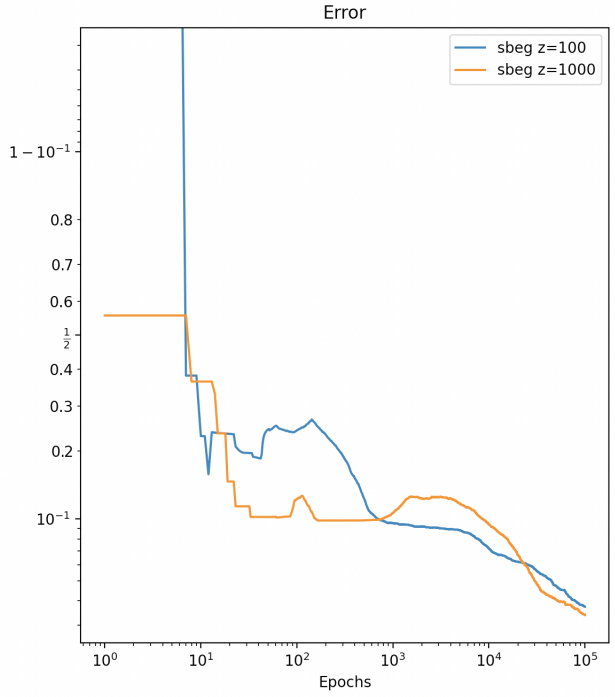

Figure 20: Erreur de SBEG pour différents $z = 100, 1000$ au cours de l'entraînement sur 10000 époques.

5.3 Comparaison de SREG et SBEG

La Figure 21 compare les temps d'exécutions entre SREG et SBEG.

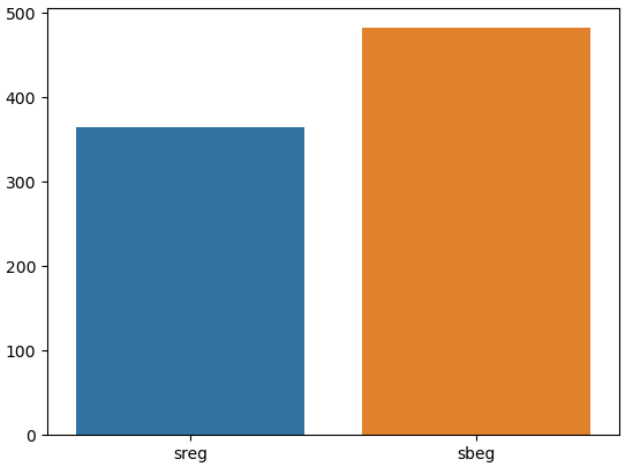

Figure 21: Temps d'exécution de SREG et SBEG pour 10000 époques d'entraînement.

Dans la Figure 22 on se propose alors de comparer les performances de SREG et SBEG sur $T = 50000$ époques, en utilisant les meilleurs hyperparamètres pour chacun. On a vu (Figures 19 et 17) que SREG était performant pour $z = 100$ alors que SBEG lui semblait meilleur pour $z = 1000$. Dans les 2 cas on considère un coefficient de régularisation $\lambda = 1/5$.

Conclusion Théoriquement, via SREG on gagne un facteur de $\sqrt{d}$ dans la borne supérieure de $\mathbb{E}[h_T^R]$, par rapport à celle de SBEG. En effet, ceci est bien visible sur les courbes et se comprend parfaitement : alors que l'*exploitation* et l'*exploration* dans SREG sont totalement indépendantes (tirage aléatoire des directions/bras à chaque étape), l'*exploration* (i.e. l'aléa) devient adaptative dans le contexte OCS de SBEG (les probabilités de tirages de chaque bras sont actualisées itérativement).

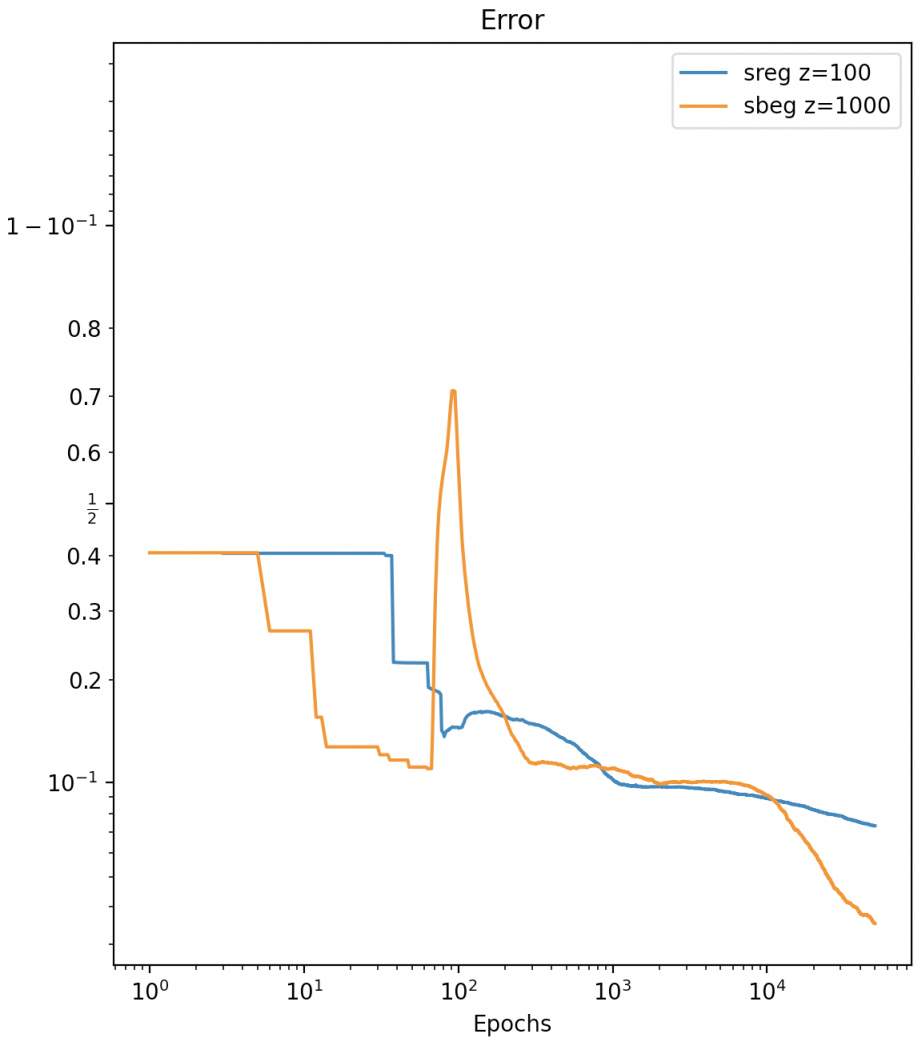

Figure 22: Erreurs de SREG et SBEG pour 10000 époques d'entraînement.

6 Adams

Cette section propose d'étudier l'algorithme `Adam` et les différentes variantes et améliorations proposées dans [5]. Lidée de l'algorithme `Adam`, pour *Adaptative moment estimation*, est tirée des algorithmes `Adagrad` [4] et `RMSprop` [2]. Les auteurs souhaitent combiner les avantages de ces deux méthodes qui proposent déjà de très bons résultats. `Adam` combine ainsi de nombreux avantages : les mises a jours des paramètres sont plus robustes car invariantes aux mises à l'échelle du gradient, les pas sont bornés par l'hyperparamètre noté α dans les pseudo-codes 23 et 24 et η_t dans le pseudo code 25, il ne nécessite pas d'objectif stationnaire (particularité de `RMSprop`), fonctionne avec des gradient parcimonieux (propriété d'`Adagrad`), et propose un pas adaptatif.

Finalement, les auteurs avancent une borne sur le regret en $\mathcal{O}(\sqrt{T})$ ce qui est comparable aux meilleurs bornes existantes dans l'état de l'art des algorithmes d'optimisation.

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 23: Pseudo-code de l'algorithme `Adam` (issu de [5])

Algorithm 2: *AdaMax*, a variant of Adam based on the infinity norm. See section 7.1 for details. Good default settings for the tested machine learning problems are $\alpha = 0.002$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. With β_1^t we denote β_1 to the power t. Here, $(\alpha/(1 - \beta_1^t))$ is the learning rate with the bias-correction term for the first moment. All operations on vectors are element-wise.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $u_0 \leftarrow 0$ (Initialize the exponentially weighted infinity norm)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $u_t \leftarrow \max(\beta_2 \cdot u_{t-1}, |g_t|)$ (Update the exponentially weighted infinity norm)
    $\theta_t \leftarrow \theta_{t-1} - (\alpha/(1 - \beta_1^t)) \cdot m_t / u_t$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 24: Pseudo-code de l'algorithme `AdaMax` (issu de [5])

Notre étude va se porter sur `Adam` et ses différentes variantes. Ainsi, les algorithmes suivants on été implémentés :

- `Adam` avec un pas fixe
- `Adam` avec un pas évolutif $a_t = \alpha \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$ comme proposé dans [5].
- `Adam` avec projection sur la boule ℓ_1 (voir pseudo-code 25)
- `Adam` en considérant des normes L_p, pour $1 \leq p < \infty$, c'est à dire que l'étape de mise à jour des moments v_t dans 23 devient $v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p)|g_t|^p$ et la mise à jour des paramètres dépend

Algorithm 19: Adam for linear SVM, Kingma and Ba (2014)

Parameters: Epoch T, radius $z > 0$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.
Initialization: Initial point $x_1 = y_1 = 0$ and $S_0 = 0$ (or $= \delta \mathbb{1}$ small).
Sample uniformly iid: $(I_t)_{1 \leq t \leq T}$ from $\{1 \leq i \leq n\}$
For each iteration $t = 1, \ldots, T$:
Iteration: Update

$$
\begin{aligned}
\eta_t &= 1/\sqrt{t}\,, \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla \ell_{a_{I_t}, b_{I_t}}(x_t) \\
S_t &= \beta_2 S_{t-1} + (1-\beta_2)\nabla \ell_{a_{I_t}, b_{I_t}}(x_t)^2 \\
D_t &= \mathrm{Diag}(\sqrt{S_t}) \\
y_{t+1} &= x_t - \eta_t D_t^{-1} m_t\,, \\
x_{t+1} &= \arg\min_{x \in B_1(z)} \|x - y_{t+1}\|_{D_t}^2\,, \qquad 1 \leq i \leq d\,.
\end{aligned}
$$

Return: $\overline{x}_{T+1} = \dfrac{1}{T+1}\displaystyle\sum_{t=1}^{T+1} x_t$

Figure 25: Pseudo-code de l'algorithme `Adam` appliqué aux SVM linéaires

désormais de la racine p-ième des v_t, c'est-à-dire $\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\epsilon + \sqrt[p]{\hat{v}_t}}$

- `AdaMax` (cas particulier de la norme L_∞) : voir pseudo-code figure 24.
- `Adam` utilisant un moyenne temporelle supplémentaire c'est-à-dire en posant $\bar{\theta}_t = \beta_2 \bar{\theta}_{t-1} + (1-\beta_2)\theta_t$ avec $\bar{\theta}_0 = 0$ et $\hat{\theta}_t = \frac{\bar{\theta}_t}{(1-\beta_2^t)}$
- `AdaMax` utilisant une moyenne temporelle supplémentaire (même que pour `Adam`)

En utilisant les hyperparamètres suivant :

- learning rate = 0.003
- number of epoch = 10000
- $\lambda = \frac{1}{3}$
- ℓ_1-ball radius = 100

On obtient les courbes de précision sur les données d'entraînement suivantes :

Notons dans un premier temps, les très bonnes performances obtenues par toutes les versions étudiées. Ensuite, on remarque que les meilleures précisions sont obtenues avec les versions de `AdaMax`. `Adam` avec pas d'apprentissage fixe propose également de bonnes performances en termes de vitesse de convergence, puisqu'entre les itération 100 et 1000 il converge bien plus vite que les autres algorithmes. En revanche, l'accuracy finale est très légèrement plus élevée que les versions `AdaMax`. Finalement, les moyennage temporels ajoutés sur `Adam` et `AdaMax` ne permettent pas, dans notre cas d'améliorer significativement les résultats que ce soit en terme de vitesse de convergence, mais également de précision finale.

Une rapide comparaison avec les autres algorithmes 27 présentés dans les sections précédentes, permet encore une fois de mettre en évidence les performances intéressantes d'`Adam`.

On retrouve également, les mêmes formes de courbes que celles présentées dans [5], c'est-à-dire qu'`Adam` convergence vers un meilleur score que les autres algorithmes d'optimisation existants (notamment la version projetée).

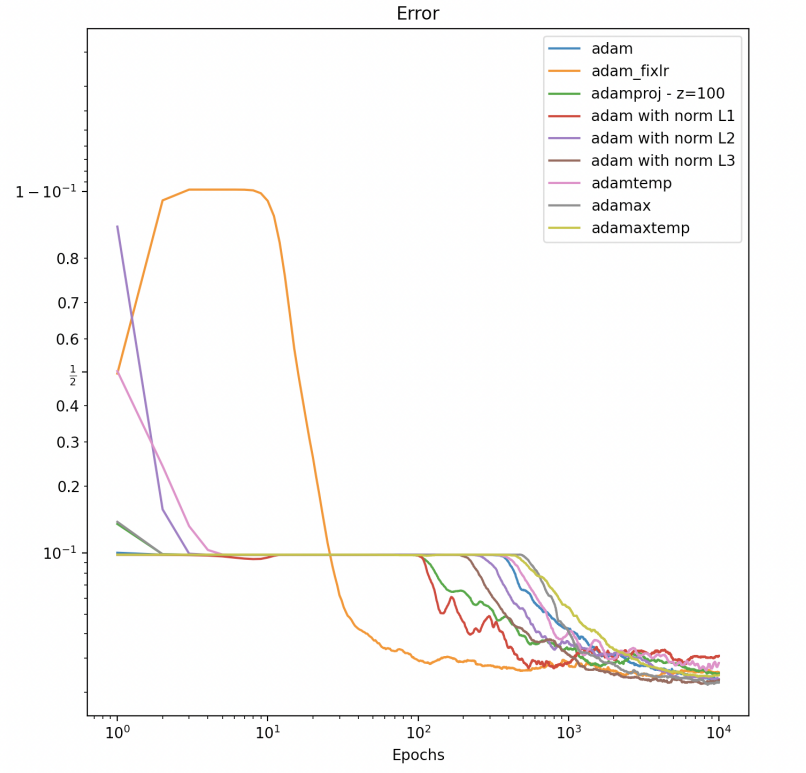

Figure 26: Accuracy pour les variantes d'`Adam`

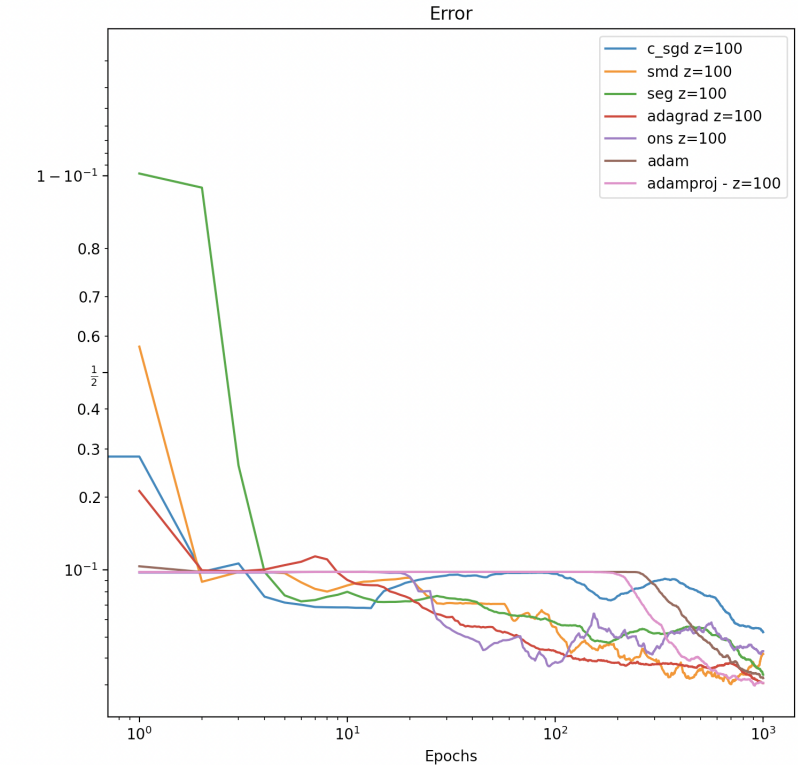

Figure 27: Accuracy pour `Adam`, `Adam` proj, `SGD`, `Adagrad`, `SMD`, `SEG` et `ONS` (test effectué sur 1000 itérations)

7 Temps d'exécution

Cette section compare le temps d'exécution des différents algorithmes présentés dans ce rapport. La figure 28 présente un histogramme comparant les algorithmes sur 10000 epochs d'entraînement en secondes. Les résultats sont attendus et reflètent bien la complexité théorique des candidats.

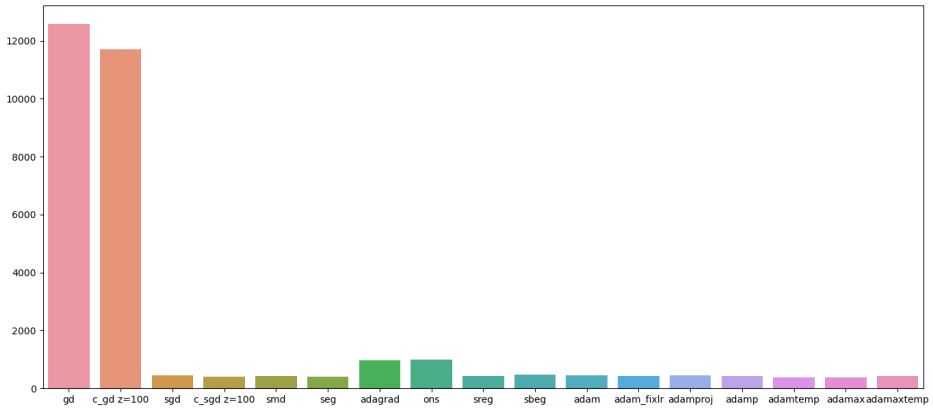

Figure 28: Comparaison du temps d'exécution des différents algorithmes sur 10000 epochs en secondes

Tout d'abord on observe que la descente de gradient (`GD`) et la descente de gradient projetée (`c-GD`) requièrent significativement plus de temps pour un nombre d'itérations identique. Cela est dû à une complexité théorique plus grande car toutes les données sont parcourues à chaque itération, contrairement à par exemple son équivalent stochastique (`SGD`) qui se contente d'un échantillon aléatoire. On passe donc d'une complexité de $\mathcal{O}(nd)$ à $\mathcal{O}(d)$ pour chaque itération, avec n la taille du dataset et d le nombre de dimension. Les algorithme `ONS` et `Adagrad` demandent eux aussi un temps de calcul plus important, bien que sans commune mesure avec la descente de gradient. Cela s'explique par l'inversion d'une matrice à chaque itération, ce qui est une opération de complexité non négligeable qui se traduit donc dans le temps d'exécution. Les autres algorithmes présentes tous des performances comparables à nombre d'itérations identique.

Conclusion

Ce projet a proposé d'implémenter différents algorithmes de l'état de l'art en optimisation convexe en ligne et de les appliquer plus particulièrement à un problème de classification binaires à l'aide d'un classifieur SVM linéaire. De nombreux algorithmes ainsi que leurs variantes ont ainsi été étudiés sous différents points de vues, afin d'en comprendre les mécanismes internes et l'influence de leurs différents hyperparamètres. Les algorithmes étudiés dans ce rapport sont :

- Descente de gradient et descente de gradient avec projection sur la boule ℓ_1
- Descente de gradient stochastique et descente de gradient stochastique avec projection sur la boule ℓ_1
- Stochastic Mirror Descent, Stochastic Exponentiated Gradient$+/-$ et `Adagrad` avec projection diagonale
- Online Newton Step
- Stochastic Randomized Exponentiated Gradient$+/-$ et Stochastic Bandit Exponentiated Gradient +/-
- `Adam`, `Adam` avec projection sur la boule ℓ_1, `Adam` en considérant des normes L^p, pour $1 \leq p < \infty$, `AdaMax` (norme L^∞), ainsi que les variantes `Adam` et `AdaMax` utilisant une moyenne temporelle supplémentaire.

Tous ces algorithmes ont été comparés selon différents critères parmi lesquels leur **vitesse de convergence**, leur **temps d'exécution** ou encore la **précision** atteinte sur les données de test.

References

[1] Jean-Yves Audibert, Sébastien Bubeck, and Gábor Lugosi. Regret in online combinatorial optimization. *Mathematics of Operations Research*, 39(1):31–45, 2014.

[2] Kevin Swersky Geoffrey Hinton, Nitish Srivastava. Neural networks for machine learning, 2012.

[3] Elad Hazan. Introduction to online convex optimization. *arXiv preprint arXiv:1909.05207*, 2019.

[4] Yoram Singer John Duchi, Elad Hazan. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[6] Olivier Winteberger. Online convex optimization, 2021.