Adam : A method for stochastic Optimization

Maxime Darrin

23 mars 2020

Table des matières

1 Introduction

Adam est une méthode originellement proposée par en 2015 par D.P Kingma et J. Lei Ba à la conférence ICLR. C'est une méthode d'optimisation du premier ordre introduisant une méthode de conservation du moment d'inertie (*momentum* en anglais) couplée à un *learning rate* adaptatif.

Il reprend le principe de l'*adaptative learning rate* utilisé par *Adadelta[5]* et *RMSProp[4]*, c'est à dire qu'il conserve une moyenne temporelle (l'importance du passé diminue exponentiellement avec le temps) du carré des gradients précédemment calculés de sorte à conserver une notion de variance des-dits gradients.

A cela, il ajoute une conservation de l'inertie. Il estime la moyenne des gradients (simples cette fois-ci) précédemment calculés, de même que précédemment en accordant plus d'importance aux observations récentes qu'aux passées en faisant diminuer exponentiellement avec le temps l'importance du passé. L'idée est de conserver une notion de moyenne de la pente courante et de continuer à aller "un peu" dans les directions prises dans le passé.

Ces deux propriétés simulent en fait la trajectoire qu'une boule qui roulerait (avec de la friction) sur la surface d'erreur, aurait.

Dans un premier temps nous commencerons par donner les grandes lignes et justifications de la règle de mise à jour proposée par *Adam*, ensuite nous présenterons l'algorithme et son implémentation en python. Dans un second temps nous détaillerons les garanties théoriques offertes par cet algorithme en terme de regret avant de mettre *Adam* à l'épreuve sur différents problèmes en le comparant aux algorithmes similaires habituellement utilisés.

2 Principes et justifications

2.1 *Adaptavie learning rate*

Tout d'abord, on rappelle la méthode *RMS Prop* proposée par Geoff Hinton[4]. On maintient une estimation de la moyenne des carrés des coordonnées des gradients, c'est à dire de la variance non centrée. Et on l'utilise pour adapter le *learning rate*. Dans la suite on notera g_t le vecteur des gradient calculé au temps t et l'application des carrés se fait coordonnées à coordonnées.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

FIGURE 1 – Mise à jour pour *RMSProp*

Adam reprend ce principe de manière plus générale : pour $\beta_2 \in [0, 1]$, on utilise :

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

comme estimateur de la variance non centrée des gradients. Cependant cet estimateur est biaisé et il est nécessaire de de le redresser.

On suppose tout d'abord que pour tout tout t, $g_t \sim p(g_t)$, des distributions de probabilité inconnues. L'objectif est d'exprimer $\mathbb{E}[v_t]$ fonction de $\mathbb{E}[g_t^2]$ pour en extraire le biais – et donc savoir comment le corriger.

Tout d'abord, on note que $v_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2$.

On peut alors passer à l'espérance et obtenir

$$\begin{aligned}
\mathbb{E}[v_t] &= \mathbb{E}\left[(1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2\right] \\
&= (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \mathbb{E}[g_i^2] && \text{Par linéarité de l'espérance} \\
&= (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} (\mathbb{E}[g_t^2] + \zeta_i) && \text{où } \mathbb{E}g_i^2 = \mathbb{E}[g_t^2] + \zeta_i \\
&= (1 - \beta_2) \sum_{i=1}^{t} (\beta_2^{t-i} \mathbb{E}[g_t^2] + \beta_2^{t-i} \zeta_i) \\
&= (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \mathbb{E}[g_t^2] + (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \zeta_i \\
&= \mathbb{E}\left[g_t^2\right] \cdot (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} + \zeta \\
&= \mathbb{E}\left[g_t^2\right] \cdot \left(1 - \beta_2^t\right) + \zeta
\end{aligned}$$

Le terme $\zeta = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \zeta_i$ est rendu petit en compensant les écarts grandissants entre le dernier gradient et les gradients plus lointains par le poids exponentiellement décroissant qui leurs sont attribués.

Ainsi on note qu'il suffit de diviser notre estimateur par $1 - \beta_2^t$ pour le redresser. On obtient alors l'estimateur proposé pour *Adam* qui est non biaisé.

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

2.2 Inertie

En plus de l'adaptation du *learning rate Adam* conserve de l'inertie dans sa descente de gradient, pour ce faire maintien une estimation de la moyenne des prédédents gradients, pour β_1 : $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

De la même manière que pour la variance, cet estimateur est biaisé et on veut le redresser.

$$\begin{aligned}
\mathbb{E}[m_t] &= \mathbb{E}\left[(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\cdot g_i\right] \\
&= (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\mathbb{E}[g_i] && \text{Par linéarité de l'espérance} \\
&= (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}(\mathbb{E}[g_t]+\zeta_i) && \text{où } \mathbb{E}g_i = \mathbb{E}[g_t] + \zeta_i \\
&= (1-\beta_1)\sum_{i=1}^{t}(\beta_1^{t-i}\mathbb{E}[g_t]+\beta_1^{t-i}\zeta_i) \\
&= (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\mathbb{E}[g_t] + (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\zeta_i \\
&= \mathbb{E}[g_t]\cdot(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i} + \zeta \\
&= \mathbb{E}[g_t]\cdot\left(1-\beta_1^t\right) + \zeta
\end{aligned}$$

De la même façon que pour la variance ζ est négligeable quitte à choisir une bonne valeur de β_2.

On peut ainsi corriger ce second estimateur de la même manière :

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

2.3 Couplage des deux grandeurs

En couplant les estimations de l'inertie et de la variance on obtient une règle de mise à jour qui adapte le *learning rate* et qui simule l'inertie de descente :

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \quad & \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ v_t &= \beta_2 b_{t-1} + (1-\beta_2) g_t^2 \quad & \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \end{aligned}$$

On peut alors analyser la règle de mise à jour. On a $\Delta_{t+1} = \theta_{t+1} - \theta_t = -\frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$. Ainsi, la taille d'un pas de la descente de gradient est de l'ordre de $\frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$, c'est à dire la moyenne sur le carré de la variance non centrée qui est ici le ratio signal bruit. Et on voit donc, que lorsque ce ratio est faible, c'est à dire que le bruit est élevé – et donc l'incertitude sur la direction à suivre, on fait des pas plus petits ce qui correspond intuitivement à ce que l'on voudrait faire. En effet, cette incertitude est en général d'autant plus grande qu'on se rapproche d'un minimum (local ou non). Au contraire, lorsque ce ratio est élevé, on peut se permettre de faire de plus grands pas sans risques.

3 Algorithme ADAM

3.1 Pseudo code

Algorithm 1 Adam

Require : η stepsize
Require : $\beta_1, \beta_2, \epsilon \in [0, 1]$
Require : $f(\theta)$ loss to minimize
  Initialize θ_0
  Initialize m_0, v_0 to zeros vectors
  $t \leftarrow 0$
  while θ_n has not converged **do**
    $g_t \leftarrow \nabla_{\theta_t} f(\theta_t)$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 b_{t-1} + (1 - \beta_2) g_t^2$
    $\hat{m}_t \leftarrow \dfrac{m_t}{1 - \beta_1^t}$
    $\hat{v}_t \leftarrow \dfrac{v_t}{1 - \beta_2^t}$
    $\theta_{t+1} \leftarrow \theta_t - \dfrac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$
    $t \leftarrow t + 1$
  end while

3.2 Implémentation python

```python
# Computing adam mean and variance
m = beta1 * prev_m + (1-beta1)*grad
v = beta2 * prev_v + (1-beta2)*np.power(grad, 2)

# We correct them to avoid biais to 0
mchap = m / (1-np.power(beta1, t))
vchap = v / (1-np.power(beta2, t))

# update weights
w = w - (eta / (np.sqrt(vchap)+eps)) * mchap
```

FIGURE 2 – Pas de descente de Adam en python

3.3 Garanties théoriques

Les auteurs de *Adam* proposent un résultat de convergence avec une borne sur le regret atteint par l'algorithme. Nous ne donnerons pas ici la preuve en détails car elle est particulièrement longue mais elle est disponible en annexe du papier original[2], nous nous contenterons de l'énoncer et d'en dériver le comportement asymptotique du regret moyen.

On se place dans le cadre décrit par [6]. Considérons que nous disposons de $f_1 \cdots f_T$ fonctions de coûts, convexes. A chaque pas de temps on optimise θ_t de sorte à minimiser $f_t(\theta_t)$ mais aussi tous les autres f_t, l'objectif étant qu'à la suite de ces optimisations séquentielles on dispose d'un θ qui minimise efficacement toutes ces fonctions. On va comparer notre résultat à la solution optimale. On définit ainsi le regret accumulé sur la séquence :

$$R(T) = \sum_{t=1}^{T} \left[f_t\left(\theta_t\right) - f_t\left(\theta^*\right) \right]$$

Avec $\theta^* = \arg\min_\theta \sum_{t=1}^{T} f_t(\theta)$, c'est à dire les paramètres minimisant globalement les f_t.

Ce processus correspond à la procédure d'entraînement par *batch* dans laquelle on entraîne le modèle séquentiellement par minimisation de fonctions de coûts définies par le *batch* courant.

Dans le cadre précédent, on suppose que toutes les f_t sont lipschitziennes pour la norme euclidienne et la norme infinie, de constantes respectivement D et D_∞. On suppose de plus que la distance entre deux θ consécutifs issus de *adam* est bornée pour les deux normes précédentes par D et D_∞ respectivement. Plus clairement, on demande que les pas générés par *adam* soient de longueurs plus petites que les constante de Lipschitz des fonctions de coûts. Par ailleurs, on suppose que β_1, β_2 ont été choisi de sorte que $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$ On pose alors $\alpha_t = \frac{\alpha}{\sqrt{t}}$ et $\beta_{1,t} = \beta_1 \lambda^{t-1}, \lambda \in (0,1)$.

On note aussi, $g_{1:T,i} = (g_{1,i} \cdots g_{T,i})$, $\gamma = \frac{\beta_1^2}{\sqrt{\beta_2}}$.

On peut alors borner le regret atteint par *Adam* :

$$R(T) \leqslant \frac{D^2}{2\alpha\left(1-\beta_1\right)} \sum_{i=1}^{d} \sqrt{T\widehat{v}_{T,i}} + \frac{\alpha\left(1+\beta_1\right) G_\infty}{\left(1-\beta_1\right)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2 + \sum_{i=1}^{d} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\alpha\left(1-\beta_1\right)(1-\lambda)^2}$$

Les auteurs notent en particulier que dans le cas où les gradients sont très creux – par exemple lorsque les données en entrées le sont – le terme $\sum_{i=1}^{d} \left\| g_{1:T,i} \right\|_2$ peut être

particulièrement petit et mener à des résultats de convergences plus forts dans certains cas.

Néanmoins, dans le cas général on peut obtenir un regret moyen asymptotique de l'ordre $\mathcal{O}(\frac{1}{\sqrt{T}})$. Il suffit d'utiliser dans l'expression précédente le fait que $\sum_{i=1}^{d} \|g_{1:T,i}\|_2 \leqslant dG_\infty\sqrt{T}$, car les gradients sont bornés par hypothèses.

En effet on a :

$$\begin{aligned}
R(T) &\leqslant \frac{D^2}{2\alpha\,(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\widehat{v}_{T,i}} + \frac{\alpha\,(1+\beta_1)\,G_\infty}{(1-\beta_1)\,\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{d} \|g_{1:T,i}\|_2 + \sum_{i=1}^{d} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\alpha\,(1-\beta_1)\,(1-\lambda)^2} \\
&\leqslant \frac{D^2}{2\alpha\,(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\widehat{v}_{T,i}} + \frac{\alpha\,(1+\beta_1)\,G_\infty}{(1-\beta_1)\,\sqrt{1-\beta_2}(1-\gamma)^2} dG_\infty\sqrt{T} + \sum_{i=1}^{d} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\alpha\,(1-\beta_1)\,(1-\lambda)^2} \\
&\underset{T\longrightarrow\infty}{\sim} \frac{D^2}{2\alpha\,(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\widehat{v}_{T,i}} + \frac{\alpha\,(1+\beta_1)\,G_\infty}{(1-\beta_1)\,\sqrt{1-\beta_2}(1-\gamma)^2} dG_\infty\sqrt{T} \\
&\underset{T\longrightarrow\infty}{=} \mathcal{O}(\sqrt{T})
\end{aligned}$$

D'où $\frac{R(T)}{T} \underset{T\longrightarrow\infty}{=} \mathcal{O}(\frac{1}{\sqrt{T}})$

Ainsi *Adam* atteint le même ordre de grandeur de regret moyen aussi bon que les meilleurs proposés dans la littérature au moment de sa publication[2] tout en montrant des qualités empiriques supérieures.

4 Résultats empiriques

4.1 Comportement sur des problèmes jouets

On commence par tester l'algorithme présenté pour optimiser des fonctions jouets et on compare les résultats obtenus avec une descente de gradient usuelle. Le problème présenté ici consiste à minimiser la fonction $f_{y_1,y_2} = ||x - y_1|| * (||x - y_2|| + 2)$, avec $y_1 = (-10, 0)$ et $y_2 = (10, 0)$. Cette fonction assure la présence de deux minimums locaux en y_1 et en y_2 avec le minimum global en y_1 meilleur que celui en y_2. On va utiliser cette dissymétrie pour comparer les différents algorithmes et leur capacité à trouver le minimum global.

Dans les exemples ci-dessous, on observe qu'*Adam* est plus efficace que les autres algorithmes pour trouver le minimum global comme on peut le voir il rejoint toujours le meilleur minimum dans ces exemples. Même placé proche de y_2, il arrive à s'extirper de la pente la plus "proche" pour atteindre y_1 dans plus de cas que les autres algorithmes [1].

1. Différentes vidéos des différentes descentes sont disponibles `https://github.com/icannos/online-optimization/tree/master/gdsvm/exports`

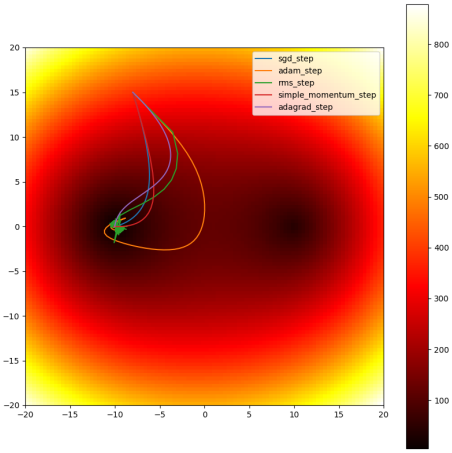


FIGURE 3 – Comparaison entre adam et différents algorithmes usuels

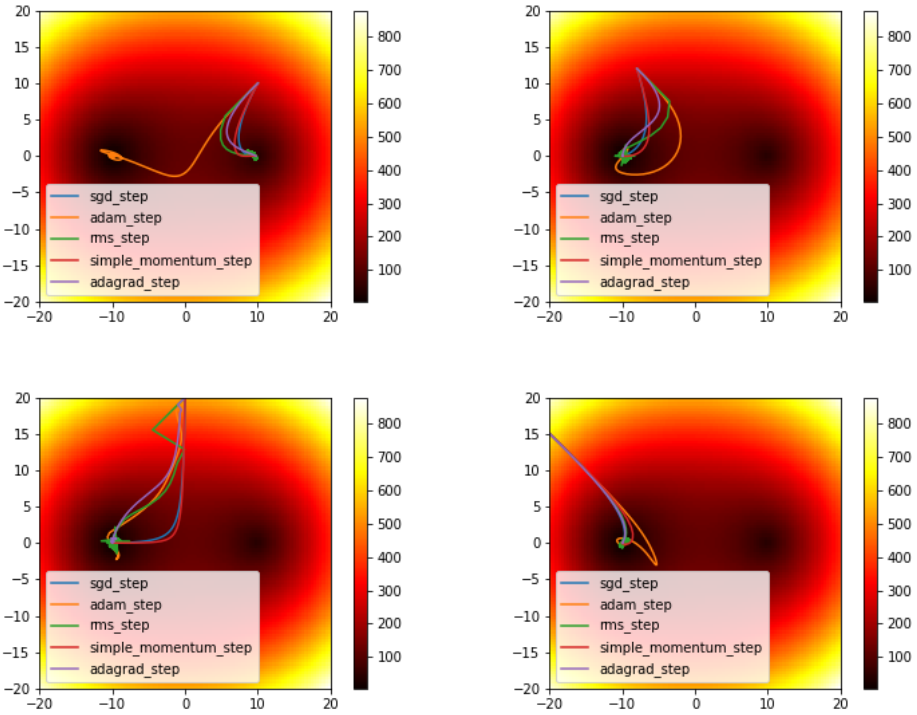


FIGURE 4 – Différentes trajectoires avec différentes initialisations

4.2 Résultats sur MNIST

4.2.1 Protocole expérimental

On utilise le jeu de données d'entraînement complet pour entraîner par batch la SVM du cours à partir de fonction donnant le gradient de la fonction de coût que l'on souhaite minimiser données en cours.

On effectue un nombre d'epoch fixé et pour chaque epoch on fait une passe sur l'ensemble du jeu de données scindé en batch. Pour chaque batch (et donc pas de descente) on sauvegarde l'erreur courante et la précision courante sur le batch en cours et sur l'ensemble du jeu de test pour constituer les graphiques qui suivent.

4.2.2 Aperçu global et premières comparaisons

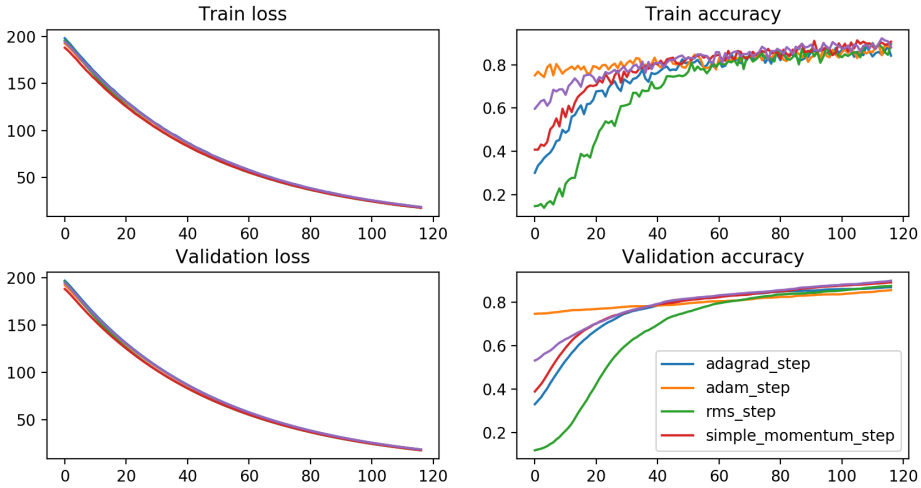


FIGURE 5 – Comparaison globale d'Adam sur MNIST

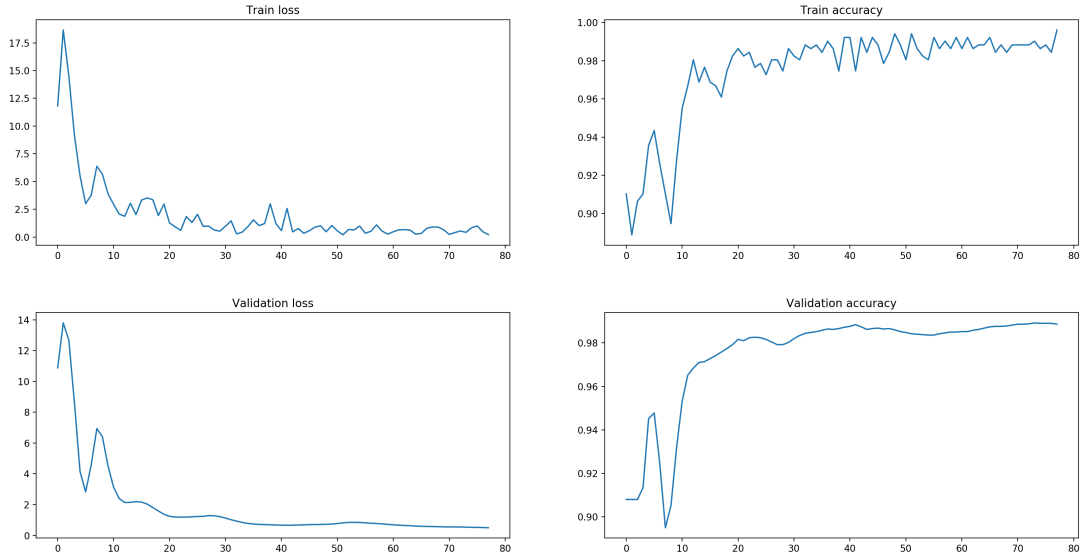


FIGURE 6 – Erreur et précision sur MNIST pour le jeu d'entraînement et le jeu de validation de Adam

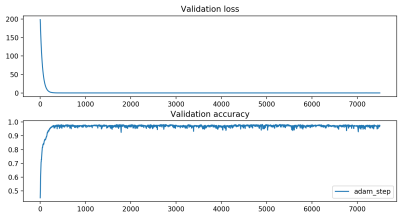
4.2.3 Efficacité de Adam selon la taille des batch

L'une des motivations avancées pour le développement d'*Adam* par ses auteurs était la résistance au bruit induit par l'entraînement par batch. On peut donc comparer la résilience d'*Adam* face à différentes tailles de batch.

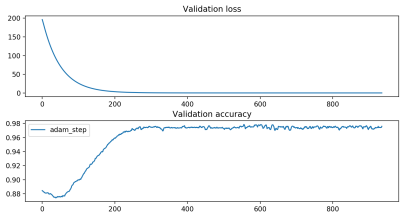
Tout d'abord on peut observer l'influence de la taille des batch sur Adam seul [2].

On présente d'abord ici globalement les effets de la taille des batchs sur tous les algorithmes, puis plus précisément l'effet de cette taille sur les très petits batchs.

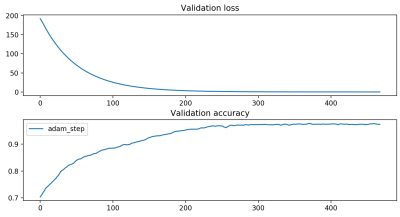
2. Tous les graphes sont disponibles en grande taille dans le dossier gdsvm/exports/.



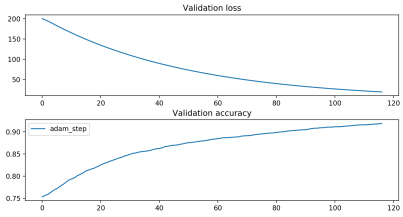
(a) batch_size = 8



(b) batch_size = 64

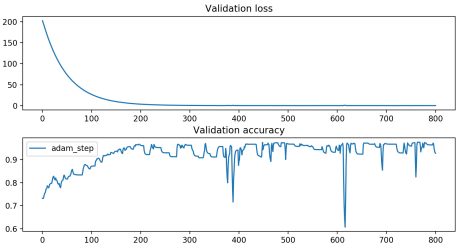


(c) batch_size = 128

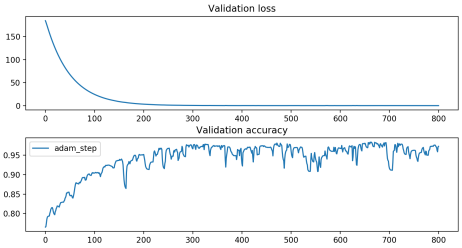


(d) batch_size = 512

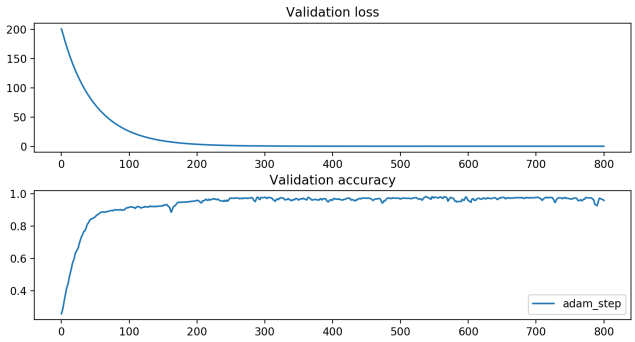
FIGURE 7 – Influence de la taille des batch sur Adam



(a) batch_size = 1



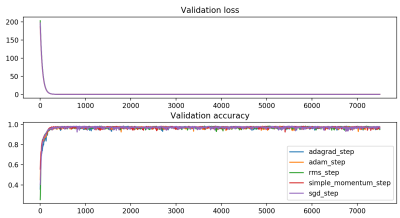
(b) batch_size = 2



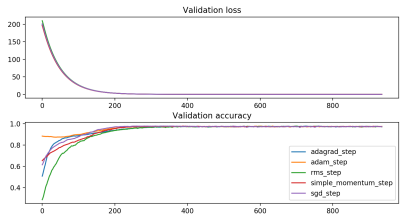
(c) batch_size = 5

FIGURE 8 – Influence de la taille des batch sur Adam

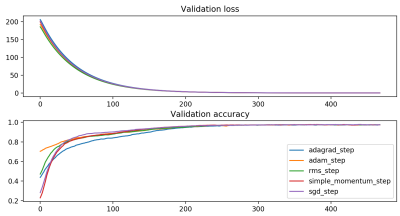
Et comparer les résultats avec les autres algorithmes usuels.



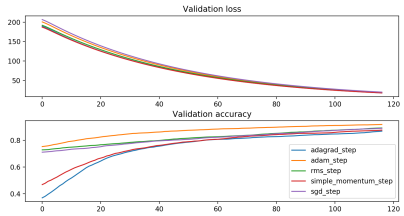
(a) batch_size = 8



(b) batch_size = 64

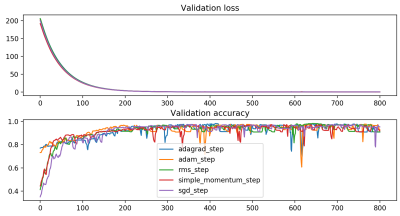


(c) batch_size = 128

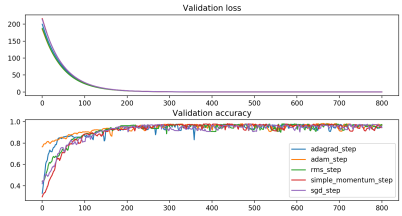


(d) batch_size = 512

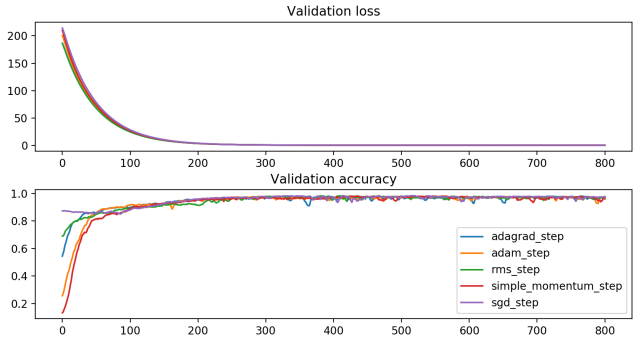
FIGURE 9 – Influence de la taille des batch, comparaison avec les autres algorithmes



(a) batch_size = 1



(b) batch_size = 2



(c) batch_size = 5

FIGURE 10 – Influence de la taille des batch, comparaison avec les autres algorithmes

On peut noter la vitesse de convergence bien plus grande de *Adam* sur les grands batch que pour les autres algorithmes.

4.3 Limites d'Adam

Il est apparu que *Adam* avait régulièrement tendance à moins bien permettre la généralisation lors de son utilisation pour l'apprentissage automatique en traitement de l'image notamment. En particulier, une SGD simple semble régulièrement plus efficace. En fait, il semble que ce sont des valeurs extrêmes de la taille du pas, apparaissant en fin d'entrainement qui produisent ces irrégularités. Une solution proposée en 2019 avec *Adabound*[3] consiste à clipper la taille du pas dans une boîte qui se rétrécit avec le temps pour converger vers la valeur optimale pour SGD(M) que l'on obtient par exemple par *grid search*. En fait, cet algorithme se comporte comme *Adam* en début d'optimisation (la boîte de clippage étant grande, elle ne change pas ou pratiquement pas le pas proposé par *Adam*) puis plus l'entraînement progresse plus le clippage garantit une valeur de pas constante en se comportant comme SGD. J'ai effectué les modifications nécessaires sur *Adam* pour implémenter cette amélioration et effectuer quelques comparaisons sur notre exemple.

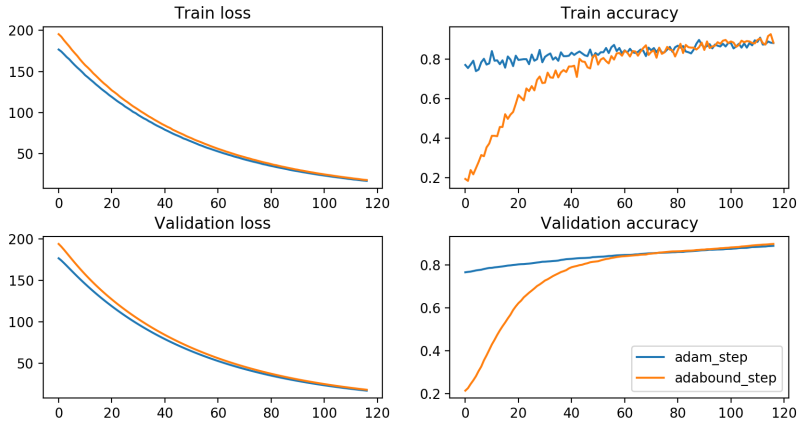


FIGURE 11 – Comparaison entre *Adam* et *Adabound* sur mnist avec les paramètres par défaut

5 Conclusion

S'il semble qu'aujourd'hui *Adam* soit l'algorithme d'optimisation le plus utilisé dans la littérature de machine learning [1], il connait des limites notamment en terme de généra-

lisation sur certains problèmes en reconnaissance d'images, notamment face à SGD, par exemples et des améliorations ou alternatives sont proposées[3] comme *Adabound* publié en 2019.

6 Références

[1] Vitaly Bushaev. Adam latest trends in deep learning optimization, 2018.

[2] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2014.

[3] Liangchen Lua and al. Adaptive gradient methods with dynamic bound of learning rate. 2019.

[4] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[5] Matthew D. Zeiler. Adadelta : An adaptive learning rate method, 2012.

[6] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML03, page 928935. AAAI Press, 2003.