STATISTIQUE
SCIENCE DES DONNÉES
UNIVERSITÉ DE MONTPELLIER

Apprentissage Statistique, 2023

ADAM : a method for stochastic optimization
Diederik P. Kingma & Jimmy Lei Ba
Lecture d'article

LECTURE D'ARTICLE

Adam : Adaptative Moment Estimation

Anne BERNARD[1] and Alexandre CAPEL[1]

[1]Université de Montpellier

Abstract

Suite à la lecture de l'article *Adam: a method for stochastic optimization*, nous avons rédigé ce document permettant de bien comprendre les enjeux de cette méthode, pourquoi elle était nécessaire et son utilisation. Nous avons également réalisé un script en python permettant de voir comment l'utiliser grâce à la librairie *pytorch* disponible sur le lien suivant : Adam-Method

Introduction

Lors d'une descente de gradient, il est parfois difficile de trouver le bon taux d'apprentissage. En effet, la plupart du temps ce taux est fixe et s'il est trop petit cela peut créer des difficultés à trouver le minimum puisque la méthode n'avancera pas assez et elle peut s'arrêter avant de l'atteindre. S'il est trop grand nous avancerons vite mais il est possible de tourner autour du point sans l'atteindre car le pas est trop grand. Il faut donc trouver le pas parfait.

C'est ce que nous permet la méthode *Adam* [1] en étant adaptative. Le taux d'apprentissage est modifié à chaque étape. Cette méthode nous permet donc d'optimiser les descentes de gradients stochastiques. De plus, un autre avantage est l'utilisation de sous-ensemble de l'échantillon pour calculer les gradients ce qui nous permet de faire les calculs sur moins de données, c'est donc plus rapide.

Nous souhaitons minimiser une fonction dite de perte que l'on note $f(\theta)$. Souvent cette fonction objective est composée de la somme de sous-fonctions évaluées en différents sous-ensembles de données. Elles sont donc stochastiques car nous choisissons les sous-ensembles de manières aléatoires.

Cette méthode nécessite seulement les gradients de premier ordre et peu de mémoire en théorie. La méthode implémentée dans pytorch stocke les paramètres mais elle est tout de même bien optimisée. Elle combine les avantages de deux méthodes : **AdaGrad** et **RMSProp** que l'on verra ensuite.

Notations

1. Pour x, y des vecteurs, x^a, $\frac{x}{y}$ et $x + b$ sont des opérations terme à terme.
2. Θ désigne un ouvert de $\mathbb{R}^d$, avec $d \in \mathbb{N}^*$.
3. $\nabla_\theta f$ correspond au gradient de la fonction f par rapport à θ.

Cadre théorique

On se place dans le modèle d'échantillonnage de l'espace probabilisé $(\mathcal{X}, \mathcal{A}, \mathbb{P})$ et soit une fonction $f : \mathcal{X}^n \times \Theta \to \mathbb{R}$ telle que :

- l'application $\theta \mapsto f(x, \theta)$ soit différentiable pour tout $x \in \mathcal{X}^n$.
- l'application $x \mapsto f(x, \theta)$ soit $\mathcal{A}^{\otimes n}$-mesurable pour tout $\theta \in \Theta$.

L'objectif de l'algorithme *Adam*, est de trouver le paramètre θ minimisant cette fonction f, que l'on appellera fonction objective, en utilisant un échantillon issu de notre modèle statistique. La nature aléatoire du problème nous oblige alors à minimiser non pas directement la fonction mais son espérance c'est-à-dire :

$$\theta^* \in \arg\min_{\theta \in \Theta} \mathbb{E}[f(X, \theta)]$$

Pour ce faire, nous allons utiliser une variante de la descente de gradient.

Description de la méthode

On dispose de $(x_1, \dots, x_T)$, T réalisations de notre modèle d'échantillonnage (c'est à dire T échantillons de taille n). A partir de ces réalisations, nous allons être en mesure de produire une suite d'estimateurs nous permettant d'essayer d'approcher θ^*.

Soit $\beta_1, \beta_2 \in [0, 1[$, $\alpha > 0$, on définit alors :

- la suite $g_t(\theta) = \nabla_\theta f(x_t, \theta)$, correspondant aux réalisations du gradient de f en θ.

- la suite $m_t(\theta) = \beta_1 m_{t-1} + (1-\beta_1) g_t(\theta)$ correspondant à la réalisation d'un estimateur de l'espérance du gradient (avec $m_0(\theta) = 0$).

- la suite $v_t(\theta) = \beta_2 v_{t-1} + (1-\beta_2) g_t(\theta)^2$ correspondant à la réalisation d'un estimateur du moment d'ordre 2 du gradient (avec $v_0(\theta) = 0$).

Grâce à ces suites d'estimateurs paramétriques, nous allons pouvoir définir notre suite θ_t de manière itérative simultanément avec m_t et v_t de la manière suivante :

$$(\theta_{t+1}, m_{t+1}, v_{t+1}) = H_{t+1}(\theta_t, m_t, v_t)$$

avec la fonction $H_t : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d$ définie par :

$$H_t(\theta, m, v) = \begin{pmatrix} \theta - \alpha \frac{\hat{m}_t(\theta)}{\sqrt{\hat{v}_t(\theta)} + \epsilon} \\ \beta_1 m + (1-\beta_1) g_t(\theta) \\ \beta_2 v + (1-\beta_2) g_t(\theta)^2 \end{pmatrix}$$

où $\epsilon > 0$, $\hat{m}_t(\theta) = \frac{1}{1-\beta_1^t}(\beta_1 m + (1-\beta_1) g_t(\theta))$
et $\hat{v}_t(\theta) = \frac{1}{1-\beta_2^t}(\beta_2 v + (1-\beta_2) g_t(\theta)^2)$.

L'algorithme

Voici comment retranscrire la partie précédente de manière algorithmique :

Algorithme 1 : ADAM

Entrées : $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$
initialisation : θ_0, $m_0 = 0$, $v_0 = 0$, $t = 0$;
tant que θ_t *ne converge pas* **faire**
  t = t+1 ;
  $g_t = \nabla_\theta f_t(\theta_{t-1})$;
  $m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$;
  $v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$;
  $\hat{m}_t = m_t / (1-\beta_1^t)$;
  $\hat{v}_t = v_t / (1-\beta_2^t)$;
  $\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
fin

Figure 1. Pseudo algorithme de *Adam*

L'algorithme a été implémenté sur Pytorch : ce sera celle-ci que l'on utilisera dans nos applications.

Interprétation de la méthode

Adam apporte une amélioration à deux méthodes existante : **RMSProp** et **AdaGrad**. La première fonctionne très bien dans le cadre de gradients clairsemés (*sparse gradient*) et la seconde dans le cadre d'environnement non-stationnaire. La procédure est conçue pour récupérer ces deux avantages. Nous allons voir dans cette section les intérêts à utiliser non pas les gradients de la fonction objective mais une combinaison de ceux-ci.

D'un point de vue géométrique, $g_t(\theta)$ représente une réalisation de la direction du gradient en θ. Les flèches sont plus longues lorsque la pente est plus raide. m_t peut être vu comme une estimation de la direction moyenne des gradients précédents. Cela capture la tendance générale des gradients. Quant à v_t, on peut le voir comme l'estimation de la "vitesse" des gradients précédents. Cela capture la magnitude des gradients ainsi que leur tendance à augmenter ou diminuer. Ces estimations sont cependant biaisées pour des valeurs de β_i proche de 1 : d'où l'introduction d'une correction effectuée par les $\hat{m}_t$ et $\hat{v}_t$.

Les intérêts d'utiliser une moyenne des gradients sont les suivants:

- Les gradients peuvent contenir du bruit (à cause de l'échantillon), utiliser la moyenne de ces derniers permet de prendre la tendance générale et de lisser les oscillations causées par le bruit.

- m_t est utilisée pour ajuster le taux d'apprentissage. Celui-ci varie en fonction de la stabilité du gradient.

- C'est un élément utile lorsque l'on est dans un environnement d'optimisation non-stationnaire, c'est-à-dire lorsque les caractéristiques de la fonction perte évolue avec le temps.

Notons Δ_t le pas utilisé (en $\epsilon = 0$) pour calculer θ_t : $\Delta_t = \alpha \cdot \hat{m}_t / \sqrt{\hat{v}_t}$. Ce rapport $\hat{m}_t / \sqrt{\hat{v}_t}$ est appelé, *SNR, (signal-to-noise ratio)* dans l'article [1]. Plus le *SNR* est petit, plus Δ_t est proche de 0. Ce qui signifie que nous sommes plus proche d'un optimum.

Une application avec MNIST

Regardons maintenant une application directe de cette méthode en la comparant avec les méthodes **RMSProp** et **AdaGrad**. Pour cela, on considère le jeu de donnée MNIST et on se place dans un premier temps dans le cadre de régression multinomiale à 10 classes, puis avec un réseau de neurone simple.

Régression multinomiale

MNIST est un jeu de données contenant des images de taille 28×28 contenant des chiffres écrits à la main. Les labels associés correspondent au chiffres écrits dans l'image (c'est à dire de 0 à 9).

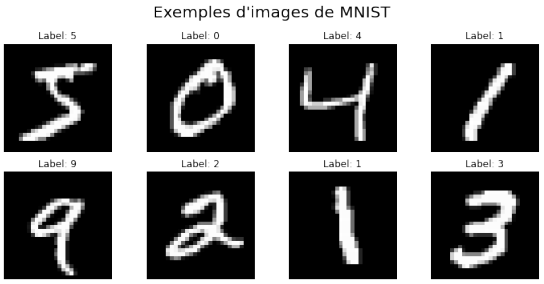

Figure 2. Images du dataset MNIST.

Dans ce contexte, notre objectif est de maximiser la vraisemblance f, correspondant à la probabilité qu'une image appartienne à une certaine classe, en des paramètres linéaires aux données.

Pour revenir dans le cadre de notre problème, la fonction objective choisie sera la log-vraisemblance négative (pour pouvoir minimiser). Nous allons séparer aléatoirement notre jeu de données en des sous échantillons de taille 128, pour pouvoir simuler la réalisation de plusieurs échantillons. Nous avons utilisé les paramètres de la fonction *Adam* de Pytorch par défaut, correspondant aux valeurs par défaut de l'algorithme présenté plus haut.

On obtient alors les courbes suivantes :

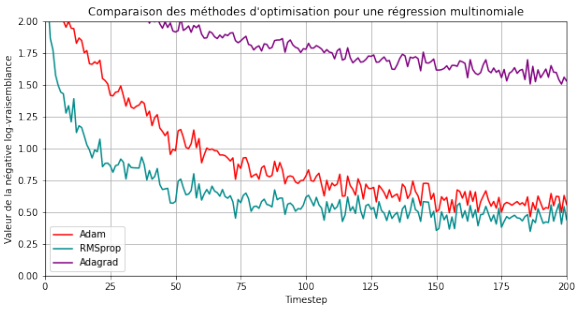

Figure 3. Comparaison des méthodes avec un pas d'apprentissage $\alpha = 0.001$ dans le cadre d'une régression multinomiale.

Nous pouvons voir dans cet exemple que *Adam* fonctionne bien, on arrive à observer une convergence de la fonction objective bien que **RMSProp** semble meilleur ici. Cependant il reste meilleur que **AdaGrad**. De plus, *Adam* et **RMSProp** finisse par converger au même point.

Réseau de neurone

On considère maintenant un réseau de neurone jouet constitué de deux couches cachés et d'un neurone de sortie linéaire. La fonction objective que l'on choisit dans cet exemple est la Cross Entropy Loss, les paramètres à optimiser étant ceux présents à chaque neurone de notre réseau.

On reconstruit alors aléatoirement des sous échantillons de notre jeu de donnée MNIST, et voici les nouvelles courbes obtenues:

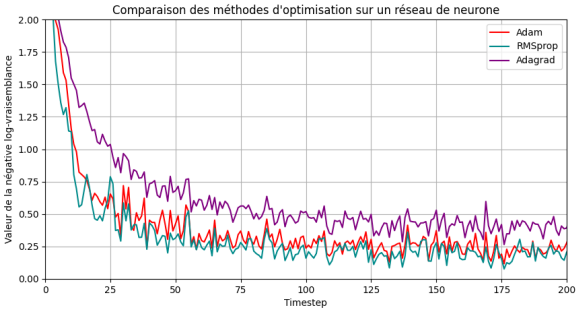

Figure 4. Comparaison des méthodes avec un pas d'apprentissage $\alpha = 0.001$ dans le cadre de l'optimisation d'un réseau de neurone.

Les résultats obtenus sont encore plus concluant. *Adam* converge vite comme **RMSProp** (**AdaGrad** est moins bon mais converge quand même). Dans cet exemple *Adam* et **RMSProp** sont identiques en terme de performance. La méthode présentée dans l'article réalise donc bien un compromis positif entre les deux méthodes.

Il est possible de trouver des jeux de données ainsi que des fonctions objectives qui représentent encore mieux ce compromis.

Convergence

Les résultats théorique sur la méthode *Adam* ne sont pas légions. Nous n'avons par exemple pas de garantie théorique sur la convergence de notre suite d'estimateurs θ_t.

Dans [1], il est montré que nous avons une convergence pour le critère du regret défini par :

$$R(T) = \sum_{t=1}^{T} (f(X_t, \theta_t) - f(X_t, \tilde{\theta}_T))$$

où $\tilde{\theta}_T = \arg\min_{\theta \in \Theta} \sum_{t=1}^{T} f(X_t, \theta)$.

Sous certaines hypothèses de régularité sur la fonction objective f et sur les paramètres de la méthode, on peut montrer que :

$$\frac{R(T)}{T} = O\left(\frac{1}{\sqrt{T}}\right)$$

Cette propriété est souhaitable pour un algorithme d'optimisation, car cela signifie que notre algorithme va s'améliorer lorsque l'on va ajouter davantage d'information et donc se rapprocher d'une décision "optimale".

Conclusion et perspectives

Adam est une méthode d'optimisation du gradient stochastique basée sur le calcul des estimations des moments du premier et du second ordre des gradients.

Cette méthode, bien que beaucoup utilisée en pratique pour ces performances qui ne sont plus à prouver, n'a pourtant que très peu de résultat théorique. Enfin, on peut aussi rappeler l'origine de cette méthode combinant à elle seule les avantages de ses deux parents : **RMSProp** et **AdaGrad**.

Adam utilise une "norme" L^2 pour normaliser le gradient, on peut étendre la méthode pour une norme L^p. De plus, lorsque $p \to \infty$ on obtient un algorithme simple et relativement stable. Cette variante existe, il s'agit de l'algorithme *AdaMax* décrit dans l'article [1].

On aurait voulu observer les performances de *Adam* dans les particuliers où les gradients sont clairsemés (instance SparseAdam dans Pytorch). De plus, il aurait pu être intéressant de voir l'évolution des résultats lorsque l'on modifie les paramètres β_i.

References

1. Kingma DP, Ba J. Adam: A method for stochastic optimization 2015;.