Revue des optimiseurs

Batch gradient descent Le point crucial dans une descente de gradient est d'approximer le (vrai)-gradient à suivre pour converger vers une solution optimale. Dans le *batch gradient descent* on décide d'approximer ce gradient en utilisant à chaque pas l'ensemble des données à notre disposition. On peut donc supposer que ce gradient est plutôt bien approximé et évite le surapprentissage. On va dans la direction qui améliore en moyenne le modèle sur tout le jeu de données.

SGD Dans le cas de *Stochastic gradient descent* on utilise pour chaque pas uniquement un point du dataset pour estimer le gradient et on suit cette direction. On améliore le modèle uniquement dans la direction qui améliore le résultat sur ce point ci.

Mini batch gradient descent Cette méthode est un compromis des deux précédentes. On estime le gradient à chaque pas sur un sous ensemble du *batch* complet et on effectue un pas dans la direction qui améliore le modèle en moyenne sur ce sous ensemble. *SGD* est un cas particulier de *minibatch gd* avec un *minibatch* de taille 1.

Adagrad *Adadelta* consiste à mettre à jour le taux d'apprentissage de chaque paramètre de sorte à le réduire pour les paramètres qui ont déjà été beaucoup entraînés. Pour cela il accumule les carrés des gradients déjà appliqués à chaque paramètre et divise le taux d'apprentissage de chaque paramètre par un terme dépendant de cette grandeur. (En pratique on prend la racine de cette grandeur plus un ϵ pour éviter des divisions par zero).

Un problème de cette méthode c'est que l'accumulation des carrés des gradients mène à un taux d'apprentissage potentiellement très faibles et donc à un point où l'algorithme n'apprend plus les pas étant trop petits.

Adadelta Cette algorithme cherche en particulier à corriger le problème soulevé par *Adagrad* en évitant que le taux d'apprentissage ne décroisse trop. Pour cela, au lieu de prendre l'accumulation du carré des gradients on prend leur moyenne pondérée par le temps : les gradients récents ont un poids plus fort que les gradients lointains. On obtient en fait à cette occasion the *RMS error criterion*.

Adadelta se passe ainsi de taux d'apprentissage comme hyperparamètres, n'utilisant que *RMS* dans sa règle de mise à jour.

RMSprop *RMSprop* est en fait très similaire à *Adadelta* et a été conçu dans le même but de résoudre le problème de la trop forte décroissance du taux d'apprentissage d'*adagrad*.

Il fixe la pondération entre les nouveaux gradients et l'ancienne moyenne (à savoir 0.1 et 0.9) et utilise le même critère de mise à jour basé sur RMS.

NAG *NAG* est un algorithme de descente de gradient sans mise à jour adaptative du taux d'apprentissage mais avec un terme d'inertie (*momentum*).

L'idée derrière cette méthode est de ne pas seulement estimer l'inertie mais aussi d'estimer là où on va potentiellement aller en utilisant les informations passées. Pour cela on calcule l'évolution des termes selon la mise à précédente.

Adam *Adam* suit les mêmes principes que les algorithmes précédents (*Adagrad* et *Adadelta*) mais en y ajoutant un terme d'inertie (*momentum*) (*NAG*). Il adapte le taux d'apprentissage membre à membre comme les méthodes précédentes.

Le terme d'inertie est la moyenne des gradients passés (avec une pondération décroissant exponentiellement fonction du temps) avec le gradient courant. Si l'on descendait très vite avant, on continue de descendre rapidement même si le gradient a diminué à l'instant courant mais avec de la friction. Cela permet de sortir des minimums locaux en cuvettes. Et garanti en somme que si l'on reste dans le minimum local c'est que c'est une cuvette avec des rebords hauts.