**Introduction**

En pratique la plupart des problèmes traités par l'Homme ne le sont que de façon approchée, la difficulté étant de trouver les valeurs exactes relatives à la résolution d'un problème précis. Il est donc question de définir une fonction de coût permettant d’évaluer les erreurs accumulées lors d’une expérience. L’optimisation aura donc pour but la minimisation de cette fonction. De ce fait plusieurs approches ont vu le jour dont la descente de gradient qui par la suite va subir de multiples améliorations donnant naissance à des variantes encore plus performantes telle Adadelta. On peut dès lors s’interroger sur l'apport de cette méthode par rapport à la descente de gradient standard. Nous nous étalerons donc sur la présentation de Adadelta de son origine à son application concrète.

1. L’intuition : pourquoi Adadelta

La descente de gradient standard est limité par de nombreux facteurs qui affectent grandement ses performances. On peut citer en autre le choix de la position de départ ,le choix du pas ,les conditions d’arrêt …

Comme exemple concret nous pouvons visualiser ci- dessous l'impact du choix du pas dans le déroulement de l’algorithme de descente de gradient

(Mettre les images )

On remarque donc que pour les pas trop grands on risque de ne jamais atteindre l’optimum car on oscille de par et d’autres de celui-ci ,et avec les petits trop petits, les risques de non convergence sont d’autant élevés et coûteux.

*Comment donc effectuer cette descente de gradient sans courir le risque du mauvais choix du pas .* Nous aboutissons à Adadelta qui est une extension de la descente de gradient et dont la description a été faite en 2012 par Matthew zeiler dans son article intitulé : **« ADADELTA : An Adaptive Learning Rate Method »**

Adadelta est conçu pour accélérer le processus d’optimisation, par exemple diminuer le nombre d’évaluations de fonctions nécessaires pour atteindre les optima, ou pour améliorer la capacité de l’algorithme d’optimisation, par exemple aboutir à un meilleur résultat final.

Il est mieux compris comme une extension des algorithmes AdaGrad et RMSProp. Adagrad commence d’abord par calculer une taille de pas (taux d'apprentissage) pour chaque paramètre de la fonction objectif à chaque mise à jour. La taille de pas est calculée en additionnant d'abord les dérivées partielles du paramètre vu jusqu'à présent pendant la recherche, puis en divisant l'hyperparamètre de taille de pas initial par la racine carrée de la somme des dérivées partielles au carré. Ensuite RMSprop quant à lui utilise une moyenne décroissante ou une moyenne mobile des dérivées partielles au lieu de la somme dans le calcul de la taille du pas pour chaque paramètre. Ceci est réalisé en ajoutant un nouvel hyperparamètre « rho » qui agit comme une quantité de mouvement pour les dérivées partielles.

Adadelta sera donc une extension supplémentaire de RMSProp conçue pour améliorer la convergence de l’algorithme et supprimer le besoin d’un taux d’apprentissage initial spécifié manuellement. Adadelta est donc une méthode de taux d’apprentissage adataptif .

1. Étapes de la méthode et fondements mathématiques

* La moyenne mobile décroissante de la dérivée partielle au carré est calculée pour chaque paramètre.

*Remarque* : Après avoir dérivé indépendamment la mise à jour RMSProp, les auteurs ont remarqué que les unités dans les équations de mise à jour pour la descente de gradient, la quantité de mouvement et Adagrad ne correspondent pas. Pour résoudre ce problème, ils utilisent une moyenne décroissante exponentielle des mises à jour carrées

* la taille de pas personnalisée est calculée comme la racine carrée de la moyenne mobile décroissante de la variation du delta divisée par la racine carrée de la moyenne mobile décroissante des dérivées partielles au carré comme indiqué par la formule :

cust\_step\_size(t+1) = (ep + sqrt(delta(t))) / (ep + sqrt(s(t))) Où

cust\_step\_size(t+1) est la taille de pas personnalisée pour un paramètre pour une mise à jour donnée, ep est un hyperparamètre qui est ajouté au numérateur et au dénominateur pour éviter une erreur de division par zéro, delta(t) est la moyenne mobile décroissante de la variation au carré du paramètre (calculée lors de la dernière itération), et s(t) est la moyenne mobile décroissante de la dérivée partielle au carré (calculée lors de l’itération actuelle).

NB :L' hyperparamètre ep est défini sur une petite valeur telle que 1e-3 ou 1e-8. En plus d'éviter une erreur de division par zéro, cela aide également à la première étape de l'algorithme lorsque le changement au carré de la moyenne mobile décroissante et le gradient au carré de la moyenne mobile décroissante sont nuls.

* Ensuite, la modification du paramètre est calculée comme la taille de pas personnalisée multipliée par la dérivée partielle

Change(t+1) = cust\_step\_size(t+1) \* f’(x(t))

* Ensuite, la moyenne décroissante de la variation au carré du paramètre est mise à jour.

Delta(t+1) = (delta(t) \* rho) + (change(t+1)^2 \* (1.0-rho)) Où delta(t+1) est la moyenne décroissante du changement de la variable à utiliser dans la prochaine itération, change(t+1) a été calculé à l’étape précédente et rho est un hyperparamètre qui agit comme l’élan et a une valeur comme 0,9.

* Enfin, la nouvelle valeur de la variable est calculée à l’aide de la modification.

X(t+1) = x(t) – changement(t+1)

Ce processus est ensuite répété pour chaque variable de la fonction objectif, puis l’ensemble du processus est répété pour naviguer dans l’espace de recherche pour un nombre fixe d’itérations d’algorithme.