EXPOSE INFO4248 :

OPTIMISATION DE LA DESCENTE DE GRADIENT AVEC ADADELTA

MEMBRES :

MELCHIZEDECK EFFA'A OZIAS 18T2564

DJONTHU NGENELO LOIC CHAREL 17S2017

EKANGA CELESTIN YVES 18T2530

MAFRE MEIDIE SANOU SONIA 18T2775

Sous la supervision du **Dr. PAULIN MELATAGIA**

SOMMAIRE

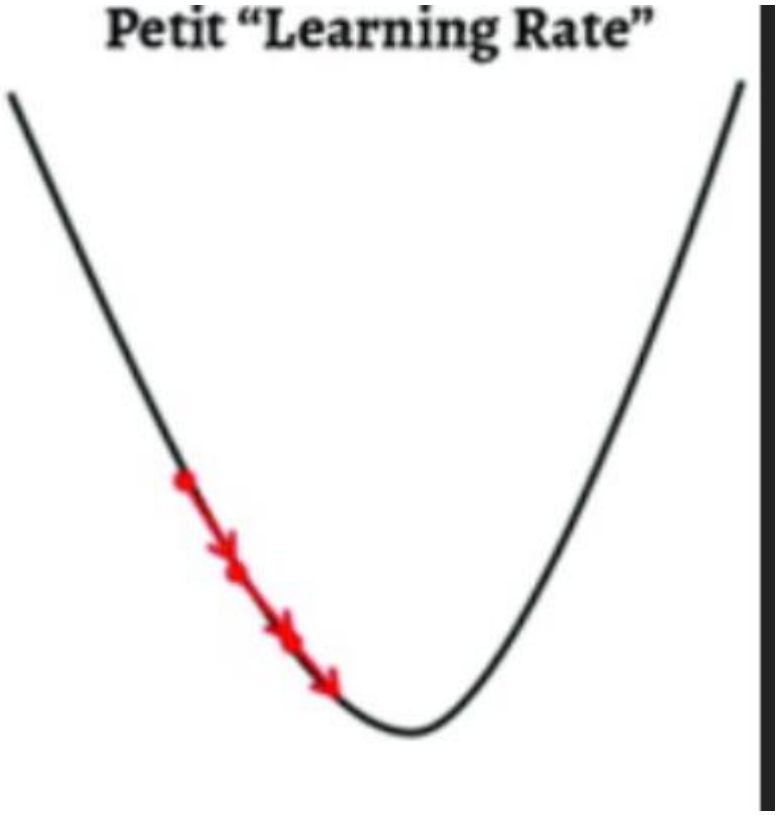
Introduction

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.

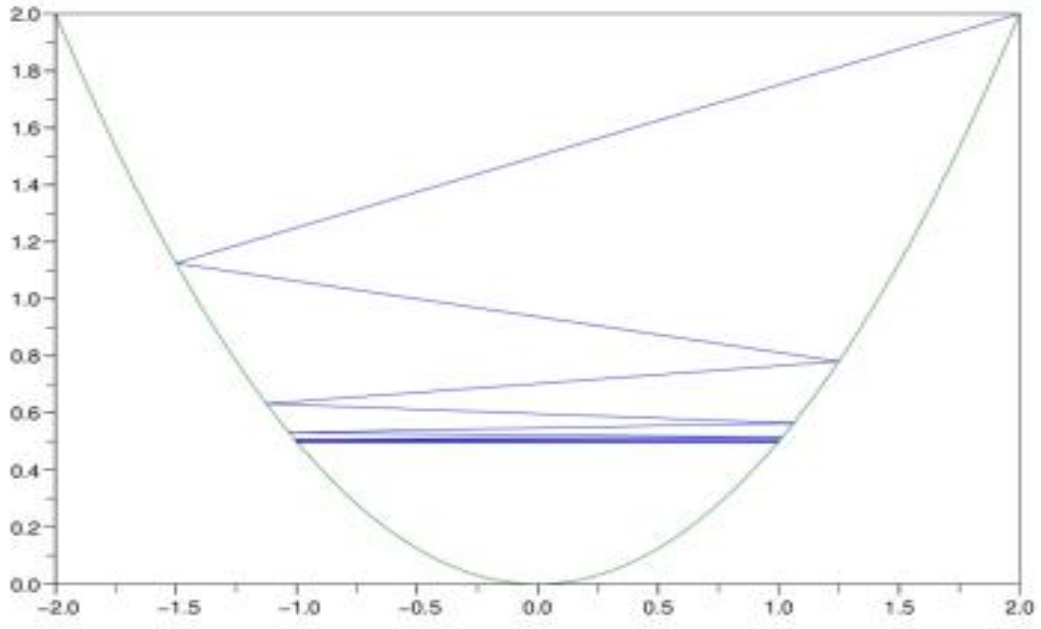
1. L'intuition : pourquoi Adadelta

La descente de gradient standard est limitée 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



Exemple : cas de très petit pas



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 .

2. 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.

3. Implémentation et codes

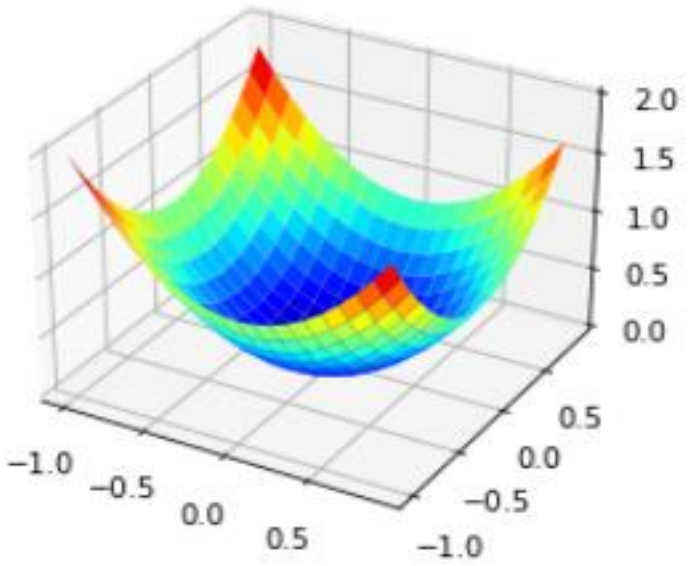
Les lignes de code qui vont suivre implémente la méthode Adadelta en se basant sur le langage python.

```
#representation de la fonction objective x**2 + y**2

from math import sqrt
from numpy import asarray
from numpy import arange
from numpy.random import rand
from numpy.random import seed
from numpy import meshgrid
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
import math

#definition de la fonction objective
def Objective(x, y):
    return x**2.0 + y**2.0

min, max = -1.0, 1.0
xaxis = arange(min, max, 0.1)
yaxis = arange(min, max, 0.1)
x,y = meshgrid(xaxis, yaxis)
results = Objective(x, y)
figure = pyplot.figure()
axis = figure.gca(projection = '3d')
axis.plot_surface(x, y, results, cmap = 'jet')
pyplot.show()
```



- Pour commencer présentons les résultats pour la descente de gradient stochastique simple

```
from math import sqrt
from numpy import asarray
from numpy import arange
from numpy.random import rand
from numpy.random import seed
from numpy import meshgrid
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
import math

#definition de la fonction objective
def Objective(x, y):
    x= float(x)
    y= float(y)
    return x**2.0 + y**2.0

#definition du gradient
def Gradient(x, y):
    x= float(x)
    y= float(y)
    return (x*2)+(y*2)

def normeGradient(x,y):
    x= float(x)
    y= float(y)
    ng=math.sqrt(math.pow(x*2, 2) + math.pow(2*y, 2))
    return ng

#descente de gradient stochastique simple
def GradientDescent(x, y, alpha, precision):
    x= float(x)
    y= float(y)
    alpha= float(alpha)
    precision= float(precision)
    nbreiteration = 0

    while(normeGradient(x, y)> precision):
        x = x - (alpha*(Gradient(x,0)))
        y = y - (alpha*(Gradient(0, y)))
        nbreiteration += 1
    return (x, y), nbreiteration
```

```
#test Descente Gradient
import os
a = input("entrer la coordonnée x ")
b = input("entrer la coordonnée y ")
c = input("entrer le pas ")
d = input("entrer la précision ")
print(" on obtient : ", GradientDescent(a, b, c, d))
```

```
entrer la coordonnée x 4
entrer la coordonnée y 2
entrer le pas 0.25
entrer la précision 0.001
 on obtient :  ((0.000244140625, 0.0001220703125), 14)
```

- Résultats obtenus avec Adadelta

```
#utilisation de la methode AdaDelta
#**********************************
def AdaDelta(x, y, rho, precision):
    x= float(x)
    y= float(y)
    precision= float(precision)
    nbreiteration = 0
    while normeGradient(x, y)>precision:
        #initialisation de la moyenne décroissante des gradients au carrée
        sq_grad_avg_x = 0
        sq_grad_avg_y = 0
        sq_para_avg_x = 0
        sq_para_avg_y = 0

        sgx = Gradient(x, 0)**2.0 #calul du carré du gradient et mise a jour de la moyenne mobile decroissante des gradients au
        sgy = Gradient(0, y)**2.0

        sq_grad_avg_x = (sq_grad_avg_x * rho) + (sgx *(1.0 -rho))
        sq_grad_avg_y = (sq_grad_avg_y * rho) + (sgy *(1.0 -rho))


        alpha_x= (precision + sqrt(sq_para_avg_x)) / (precision + sqrt(sq_grad_avg_x)) # calcul de la taille du pas adaptatif
        alpha_y= (precision + sqrt(sq_para_avg_y)) / (precision + sqrt(sq_grad_avg_y))

        change_x = alpha_x * Gradient(x, 0) # pour modifier la variable
        change_y = alpha_y * Gradient(0, y)

        sq_para_avg_x = (sq_para_avg_x * rho) + (change_x**2.0 *(1.0 - rho))
        sq_para_avg_y = (sq_para_avg_y * rho) + (change_y**2.0 *(1.0 - rho))

        value_x = x - change_x
        value_y = y - change_y

        x = value_x
        y = value_y

        solution_eval = Objective(x, y)
        print(x, y, solution_eval)
        nbreiteration +=1
    return[x, y, solution_eval], nbreiteration
```



```
def AdaDeltaN(x, y, rho, precision, nbreiteration):
    x= float(x)
    y= float(y)
    precision= float(precision)
    for i in range( nbreiteration):
        #initialisation de la moyenne décroissante des gradients au carrée
        sq_grad_avg_x = 0
        sq_grad_avg_y = 0
        sq_para_avg_x = 0
        sq_para_avg_y = 0

        sgx = Gradient(x, 0)**2.0 #calul du carré du gradient et mise a jour de la moyenne mobile decroissante des gradients au
        sgy = Gradient(0, y)**2.0

        sq_grad_avg_x = (sq_grad_avg_x * rho) + (sgx *(1.0 -rho))
        sq_grad_avg_y = (sq_grad_avg_y * rho) + (sgy *(1.0 -rho))


        alpha_x= (precision + sqrt(sq_para_avg_x)) / (precision + sqrt(sq_grad_avg_x)) # calcul de la taille du pas adaptatif
        alpha_y= (precision + sqrt(sq_para_avg_y)) / (precision + sqrt(sq_grad_avg_y))

        change_x = alpha_x * Gradient(x, 0) # pour modifier la variable
        change_y = alpha_y * Gradient(0, y)

        sq_para_avg_x = (sq_para_avg_x * rho) + (change_x**2.0 *(1.0 - rho))
        sq_para_avg_y = (sq_para_avg_y * rho) + (change_y**2.0 *(1.0 - rho))

        value_x = x - change_x
        value_y = y - change_y

        x = value_x
        y = value_y

        solution_eval = Objective(x, y)
        print(x, y, solution_eval)
    return[x, y, solution_eval]
```

```python
#test
import os
a = input("entrer la coordonnée x ")
b = input("entrer la coordonnée y ")
d = input("entrer la précision ")
rho = 0.99
choix = int(input("choix de la condition d'arret\n 1-Norme du gradient\n 2-nombre Diteration
if choix==1:
    print(" on obtient : ", AdaDelta(a, b, rho, d))
else:
    nbreiter = int(input("entrer le nombre diterations"))
    print(" on obtient : ", AdaDeltaN(a, b, rho, d, nbreiter))
```

```
entrer la coordonnée x 4
entrer la coordonnée y 2
entrer la précision 0.001
choix de la condition d'arret
 1-Norme du gradient
 2-nombre Diteration
1
3.990012484394507 1.9900249376558603 19.880398878116235
3.980024999999951 1.9800499999992207 19.761197003121524
3.97003754697268 1.9700751882816316 19.64239437185576
3.9600501254702185 1.9601005037735666 19.523990981130083
3.95006273565128 1.9501259477648047 19.40598682772645
3.940075377675779 1.9401515215648237 19.28838190839723
3.930088051704844 1.930177226503203 19.171176219864773
3.920100757900828 1.9202030639300365 19.054369758820947
3.910113496427323 1.9102290352163578 18.937962521926725
3.900126267449171 1.9002551417545737 18.821954505811696
3.8901390711324773 1.890281384958911 18.70634570707363
3.880151907644624 1.8803077662658751 18.59113612227798
```

```
0.12706247628538514 1.3040093608158307e-05 0.0161448730498181
0.11744108496671829 -1.2972252921415606e-05 0.01379240860643929
0.1078494446281309 1.2905115368137067e-05 0.011631502873138275
0.09829251282545134 -1.2838670063934376e-05 0.009661418242372965
0.08877657501575252 1.2772906348060204e-05 0.00788128043467467
0.07930975720961705 -1.2707813777690737e-05 0.006290037750136935
0.06990280837777173 1.2643382122382762e-05 0.004886402778954585
0.06057034009097299 -1.2579601358699188e-05 0.003668766256982500 8
0.05133287998218951 1.2516461664997006e-05 0.0026350647239276846
0.04222046108963170 6 -1.2453953416372003e-05 0.0017825674897220605
0.03327932410888712 1.2392067179754722e-05 0.0011075135667076845
0.02458551227087695 -1.2330793709152505e-05 0.0006044475656699146
0.01627552863909845 2 1.2270123941032448e-05 0.0002648929830380554 4
0.00862564650682156 5 -1.2210048989840505e-05 7.44019267459394e-05
0.00229519715713147 4 1.2150560143652044e-05 5.2680776262162055e-06
-0.00085097848102490 83 -1.2091648859949396e-05 7.243105831396126e-07
0.00060344232018754 69 1.2033306761522060e-05 3.642874342649467e-07
-0.00047347127275428 19 -1.1975525632485447e-05 2.243184593377339e-07
 on obtient :  ([-0.0004734712727542819, -1.1975525632485447e-05, 2.243184593377339e-07], 406)
```

CONCLUSION

Au terme de cette analyse ou il était question de discuter sur l'apport de Adadelta par rapport à la descente de gradient standard, nous pouvons affirmer sans réserve que reforme de la descente de gradient de gradient s'est inspirée des limites des anciennes pour combler l'une des principales difficultés qu'est le choix du pas. En outre peut on affirmer toute fois que version de la descente de gradient est la meilleure ?