BAPTISTE
BRINON
L3 Informatique

27/03/2020

PYTHON ET MACHINE LEARNING

Université d'Angers, année scolaire 2019/2020

Introduction :

le rapport suivant est le résultat du travail fourni chez moi. J'ai abandonné les résultats de ce que j'avais fait à l'université pour maintenir une cohérence dans les résultats. Ainsi, tout à été fait sur mon ordinateur portable, donc j'ai dû limiter à 4000 le nb d'epochs pour que la durée de chaque test soit correcte et l'affichage se réinitialise toutes les 10 epochs. Le pseudo utilisé pour le concours en ligne est **bbrinon**.

Méthodologie :

Pour garder des résultats comparables, une seed pour le random à été définie.

I – reseau de neurones

les tests se sont faits en 3 parties :

- Tests et selection d'un optimizer
- Tests et selection d'un eta
- Recherche d'une stucture de reseau de neurones appropriée

1) Tests et selection d'un optimizer

Pour pouvoir tester les differents optimizers, on doit fixer une configuration par défaut. Utilisation de la configuration suivante:

4000 epochs, 0.50 eta (car nombre d'epochs faible), reseau->51,400,200,200,50

Résultats :

SGD:[10:46<00:00, 6.19it/s, error_test=0.342, error_train=0.285, iter=3990, loss=0.559]

ASGD: [10:45<00:00, 6.19it/s, error_test=0.295, error_train=0.365, iter=3990, loss=0.586]

Adadelta:[09:56<00:00, 6.71it/s, error_test=0.253, error_train=0.252, iter=3990, loss=0.495]
meilleur actuel (~75% réussite)

Adagrad: [10:36<00:00, 6.29it/s, error_test=0.48, error_train=0.48, iter=3990, loss=0.692]

Adam: [09:25<00:00, 7.08it/s, error_test=0.505, error_train=0.474, iter=3990, loss=0.703]

AdamW:[17:10<00:00, 3.88it/s, error_test=0.488, error_train=0.523, iter=3990, loss=0.729]
AdamW est supposé être le meilleur, mais est l'un des pire. Probablement cas de surapprentissage dû à l'eta élevé

1.5) recherche de la cause du résultat de AdamW. Verification de l'eta

on fixe l'eta à 0.10 plutot que 0.50 :

ancien pire : AdamW: [21:07<00:00, 3.16it/s, error_test=0.491, error_train=0.476, iter=3990, loss=0.692]

ancien meilleur : Adadelta: [09:27<00:00, 7.05it/s, error_test=0.261, error_train=0.256, iter=3990, loss=0.512]

eta à 0.80:

AdamW: [17:23<00:00, 3.83it/s, error_test=0.476, error_train=0.518, iter=3990, loss=0.693]
Adadelta:[10:19<00:00, 6.46it/s, error_test=0.285, error_train=0.265, iter=3990, loss=0.512]

eta à 0.02, pour être sûr qu'il faut diminuer celui-ci:

AdamW: [14:01<00:00, 4.75it/s, error_test=0.269, error_train=0.104, iter=3990, loss=0.217]
Adadelta: [10:00<00:00, 6.66it/s, error_test=0.28, error_train=0.3, iter=3990, loss=0.588]

Adadelta semble avoir des résultats plus ou moin constants. Aprés recherche sur internet, il s'agirait du fait qu'Adadelta adapte lui-même l'eta, et que celui qu'on fixe n'est que la valeure initiale

test de l'optimizer ASGD à 0.02, pour verifier qu'AdamW est bien le meilleur :
[10:32<00:00, 6.32it/s, error_test=0.301, error_train=0.279, iter=3990, loss=0.565]

2) Tests et selection d'un eta

selection de AdamW. Changement des eta:

0.0100 : [12:37<00:00, 5.28it/s, error_test=0.231, error_train=0.108, iter=3990, loss=0.245]

0.0200 : [14:01<00:00, 4.75it/s, error_test=0.269, error_train=0.104, iter=3990, loss=0.217] (rappel)

0.0300 : [12:33<00:00, 5.31it/s, error_test=0.238, error_train=0.166, iter=3990, loss=0.366]

0.0050 : [12:00<00:00, 5.55it/s, error_test=0.25, error_train=0.0627, iter=3990, loss=0.148]

0.0010 : [09:11<00:00, 7.26it/s, error_test=0.254, error_train=0.00543, iter=3990, loss=0.0253]
Precision en ligne: 0.6816367265

0.0400 : [17:57<00:00, 3.71it/s, error_test=0.319, error_train=0.251, iter=3990, loss=0.443]

0.0250 : [12:44<00:00, 5.23it/s, error_test=0.228, error_train=0.105, iter=3990, loss=0.257]

0.0350 : [15:35<00:00, 4.28it/s, error_test=0.22, error_train=0.137, iter=3990, loss=0.307]
Precision en ligne: 0.6457085828343314

0.0005 : [10:33<00:00, 6.31it/s, error_test=0.259, error_train=0.00838, iter=3990, loss=0.0352]

0.0015 : [09:28<00:00, 7.04it/s, error_test=0.258, error_train=0.0126, iter=3990, loss=0.0399]

on conserve l'eta 0.0010, car il a les meilleurs résultats en ligne, bien qu'ayant un error_test plus faible que 0.0350, il a une loss bien plus faible.

3) Recherche d'une structure de reseau de neurones appropriée

à ce moment du projet, il y a eu constat qu'un oubli avait été fait (layer 2 et layer 3 pas reliés, correction et poursuite des tests)

Dû à la grande quantité de résultats, ceux-ci sont disponibles en annexes (RN). Seuls les plus importants seront présentés ici.

On à tout d'abord essayer différentes quantités de couches. Il se trouve que pour un même nombre de neurones, il y a de meilleurs résultats pour un grand nombre de couches qu'un nombre faible.

51,100,20 [01:04<00:00, 61.63it/s, error_test=0.238, error_train=0.0324, iter=3990, loss=0.108]
precision : 0.7095808383233533

51,50,50,20 [01:01<00:00, 64.86it/s, error_test=0.233, error_train=0.0833, iter=3990, loss=0.22]
Precision : 0.7325349301397206

La derniere couche contient 20 neurones, car il s'agit du nombre de colonnes du jeu de données. Ainsi, avec 20 neurones sur la dernière couche on a de meilleurs résultats.

Ensuite, on à changer la taille du reseau de neurones pour trouver le meilleur nombre de couches possible

51,20,20,20,20,20,20,20,20,20,20,20 [02:05<00:00, 31.87it/s, error_test=0.214, error_train=0.132, iter=3990, loss=0.338] ***Precision : 0.7345309381237525***

51,20,20,20,20,20,20,20,20,20,20 [01:07<00:00, 59.19it/s, error_test=0.208, error_train=0.131, iter=3990, loss=0.324]

51,20,20,20,20,20,20,20,20,20 [01:13<00:00, 54.68it/s, error_test=0.209, error_train=0.131, iter=3990, loss=0.33]

51,20,20,20,20,20,20,20,20 [00:58<00:00, 68.06it/s, error_test=0.221, error_train=0.119, iter=3990, loss=0.314]

51,20,20,20,20,20,20,20 [01:08<00:00, 58.65it/s, error_test=0.224, error_train=0.119, iter=3990, loss=0.32]

51,20,20,20,20,20,20 [00:42<00:00, 93.20it/s, error_test=0.215, error_train=0.128, iter=3990, loss=0.326]

Ici, j'ai fait le constat que même avec la seed, les résultats diffèrent d'une fois sur l'autre. Donc il est difficile de faire des tests « précis ».

on conserve tout de même la structure avec de meilleurs résultats :
51,20,20,20,20,20,20,20,20,20 (une couche de + diminue les résultats)

J'ai alors, durant mes recherches sur le net, vu que parfois on remplace le dernier sigmoid par un softmax. Je l'ai donc testé :

ancien résultat: [01:07<00:00, 59.19it/s, error_test=0.208, error_train=0.131, iter=3990, loss=0.324]
nouveau résultat: [01:11<00:00, 56.08it/s, error_test=0.475, error_train=0.483, iter=3990, loss=13.3]

Au final, on a de meilleurs résultats en conservant sigmoid

Ensuite, il à fallu déterminer la quantité de neurones de chaque couche. Les résultats s'améliorant en diminuant celle-ci plutôt qu'en l'augmentant, j'ai chercher à m'approcher le plus possible du surapprentissage sans pour autant en souffrir.

51,25,20,10,10,10,10,10,10,20 [01:17<00:00, 51.62it/s, error_test=0.205, error_train=0.128, iter=3990, loss=0.34] ***Precision : 0.7375249500998005***

51,10,10,10,10,10,10,10,10,20 [00:45<00:00, 87.24it/s, error_test=0.214, error_train=0.18, iter=3990, loss=0.393]

51,10,10,10,10,10,10,10,2,20 [00:42<00:00, 95.06it/s, error_test=0.2, error_train=0.163, iter=3990, loss=0.384] ***Precision : 0.7684630738522954***

51,256,128,64,32,16,8,4,2,20 [06:10<00:00, 10.81it/s, error_test=0.222, error_train=0.0962, iter=3990, loss=0.292]

51,102,128,64,32,16,8,4,2,20 [03:30<00:00, 19.03it/s, error_test=0.209, error_train=0.109, iter=3990, loss=0.319]
résultats cohérents

51,20,10,2,2,2,2,2,2,20 [00:50<00:00, 79.56it/s, error_test=0.215, error_train=0.132, iter=3990, loss=0.349]
(surapprentissage probable. montée du taux d'echecs sur la fin)

51,2,2,2,2,2,2,2,2,20 [00:36<00:00, 108.82it/s, error_test=0.475, error_train=0.483, iter=3990, loss=0.693]
effectivement, surapprentissage confirmé

51,8,8,8,8,8,8,4,2,20 [00:38<00:00, 103.55it/s, error_test=0.196, error_train=0.183, iter=3990, loss=0.416]

51,4,4,4,4,4,4,4,2,20 [00:38<00:00, 103.18it/s, error_test=0.191, error_train=0.186, iter=3990, loss=0.42]

51,4,4,4,4,4,4,2,2,20 [00:37<00:00, 105.84it/s, error_test=0.196, error_train=0.201, iter=3990, loss=0.433]

au final, le meilleur que j'ai pu obtenir est un error_test de 0.191 avec 51,4,4,4,4,4,4,4,2,20 et une Precision de 0.76 sur le test en ligne.

Ses résultats sont comparable à 51,10,10,10,10,10,10,10,2,20 qui a 0.77 de precision, malgrès l'error_test de 0.2 uniquement. Voilà qui conclu la partie sur les RN

II – regression logistique

Ici, seul l'eta était à modifier pour voir lequel apportait de meilleurs résultats. Voici ce que ça à donné :

eta 0.0010: [00:07<00:00, 514.64it/s, error_test=0.321, error_train=0.322, iter=3900, loss=6.07]

eta 0.0100: [00:09<00:00, 429.26it/s, error_test=0.212, error_train=0.21, iter=3900, loss=0.486]

eta 0.1000: [00:07<00:00, 527.49it/s, error_test=0.242, error_train=0.217, iter=3900, loss=4.36]

eta 0.0500: [00:07<00:00, 503.56it/s, error_test=0.215, error_train=0.273, iter=3900, loss=5.23]

eta 0.0250: [00:14<00:00, 282.35it/s, error_test=0.304, error_train=0.299, iter=3900, loss=2.93]

meilleur résultat à un eta de 0.01, avec un error_test de 0.212, qui est inférieur à celui du reseau de neurones.

II – K plus proches voisins

Malheureusement, par manque de temps je n'ai pas pu terminer Kvoisins. En vérité, il y a une erreur à la compilation que je n'ai pas encore pu résoudre, et je ne peux pas me permettre de passer plus de temps dessus. Le code est tout de même fourni, bien que non fonctionnel.

De ce que j'ai compris du problème, il s'agit d'un soucis de dimension, car le 2nd argument fourni est à 2 dimensions alors qu'il en attendait un à 1 dimension. J'ai essayé de faire un parcours de matrice en créant une boucle j parcourant cet argument, et retournant ainsi chaque ligne, fournissant un paramètre en 1D pour l'appel de fonction. Mais je me suis vite rendu compte que le temps d'execution serait beaucoup trop long, et ait donc conclu qu'il ne s'agissait pas de la bonne méthode.

Par soucis de temps, je n'ai pas non plus été essayer une des autres méthodes de skicitlearn.

ANNEXES

Annexe RN

essai avec quantité faible (3) de neurones de grandes valeurs
51,400,200 [09:00<00:00, 7.40it/s, error_test=0.258, error_train=0.00682, iter=3990, loss=0.0337]
essai avec quantité faible (3) de neurones de petites valeurs
51,40,20 [00:30<00:00, 131.73it/s, error_test=0.23, error_train=0.0987, iter=3990, loss=0.248]
51,100,40 [01:46<00:00, 37.71it/s, error_test=0.239, error_train=0.0386, iter=3990, loss=0.119]
essai avec quantité faible (3) de neurones de valeurs moyennes
51,200,100 [03:00<00:00, 22.15it/s, error_test=0.257, error_train=0.0226, iter=3990, loss=0.0697]

on extrapole le meilleur cas :
51,600,200 [12:15<00:00, 5.43it/s, error_test=0.257, error_train=0.00791, iter=3990, loss=0.0353] ***test precision : 0.6876247504990021***

essai avec quantité grande (8) de neurones de petites valeurs décroissantes puis croissantes
51,80,70,60,50,40,30,20 [02:36<00:00, 25.49it/s, error_test=0.223, error_train=0.0817, iter=3990, loss=0.268] ***test precision : 0.7425149700598803***
51,20,30,40,50,60,70,80 [02:13<00:00, 29.91it/s, error_test=0.238, error_train=0.137, iter=3990, loss=0.315]
51,50,40,35,30,20,10 [01:12<00:00, 55.27it/s, error_test=0.229, error_train=0.0738, iter=3990, loss=0.254] ***test : 0.686626746506986***
essai avec quantité grande (8) de neurones de grandes valeurs
51,400,350,300,250,200,150,100 [23:04<00:00, 2.89it/s, error_test=0.268, error_train=0.0377, iter=3990, loss=0.132] ***test precision : 0.719560878243513***
51,300,200,200,100,100,50,20 [09:58<00:00, 6.68it/s, error_test=0.24, error_train=0.0734, iter=3990, loss=0.243] ***test precision : 0.7544910179640718***

meilleurs resultats sur les groupes finissant par des valeurs faibles. Tentatives supplémentaires
51,400,200,200,100,50,20,2 [12:49<00:00, 5.20it/s, error_test=0.483, error_train=0.479, iter=3990, loss=0.692]
51,400,200,200,100,50,40,20 [11:07<00:00, 5.99it/s, error_test=0.242, error_train=0.0651, iter=3990, loss=0.219] ***test precision : 0.7584830339321358***
51,400,200,100,50,40,20 [10:10<00:00, 6.56it/s, error_test=0.234, error_train=0.0575, iter=3990, loss=0.2] ***test precision : 0.7415169660678642***

quelques tests:
51,100,50 [01:23<00:00, 47.91it/s, error_test=0.246, error_train=0.0385, iter=3990, loss=0.113] ***precision : 0.666666666666666***
51,100,20 [01:04<00:00, 61.63it/s, error_test=0.238, error_train=0.0324, iter=3990, loss=0.108] ***precision : 0.7095808383233533***
51,100,30 [01:15<00:00, 53.17it/s, error_test=0.238, error_train=0.0347, iter=3990, loss=0.109] ***precision : 0.6786427145708582***

confirmation de l'efficacité de la fin par 20 -> logique, vu qu'il s'agit du nombre de colonnes du jeu de données

51,50,50,20 [01:01<00:00, 64.86it/s, error_test=0.233, error_train=0.0833, iter=3990, loss=0.22] Precision : 0.7325349301397206
pour une même quantité de neurones, il vaut mieux avoir plusieurs couches ?
51,20,20,20,20,20,20 [01:08<00:00, 58.65it/s, error_test=0.224, error_train=0.119, iter=3990, loss=0.32] Precision : 0.7315369261477046
51,50,50,50,50,50,20 [01:52<00:00, 35.53it/s, error_test=0.246, error_train=0.0828, iter=3990, loss=0.206] Precision : 0.7375249500998005
quand valeurs élevées moin de precision, tentative avec valeurs faibles

51,100,5,2,3,12,20 [01:26<00:00, 46.09it/s, error_test=0.265, error_train=0.0259, iter=3990, loss=0.102]
Precision : 0.5728542914171657 sans doutes car surapprentissage lors du passage de 100 neurones à 5

tentative de rajouter des couches tout en augmentant le nombre de neurones (7 couches -> 11 couches)
51,20,20,20,20,20,20,20,20,20,20,20 [02:05<00:00, 31.87it/s, error_test=0.214, error_train=0.132, iter=3990, loss=0.338] ***Precision : 0.7345309381237525***
diminution 1 par 1 de la quantité de couches de neurones pour trouver la " bonne taille "
51,20,20,20,20,20,20,20,20,20,20 [01:07<00:00, 59.19it/s, error_test=0.208, error_train=0.131, iter=3990, loss=0.324]
51,20,20,20,20,20,20,20,20,20 [01:13<00:00, 54.68it/s, error_test=0.209, error_train=0.131, iter=3990, loss=0.33]
51,20,20,20,20,20,20,20 [00:58<00:00, 68.06it/s, error_test=0.221, error_train=0.119, iter=3990, loss=0.314]
51,20,20,20,20,20,20 [01:08<00:00, 58.65it/s, error_test=0.224, error_train=0.119, iter=3990, loss=0.32]
51,20,20,20,20,20 [00:42<00:00, 93.20it/s, error_test=0.215, error_train=0.128, iter=3990, loss=0.326]

constat que même avec la seed, les résultats diffèrent d'une fois sur l'autre.

on conserve tout de même celui avec de meilleurs résultats : 51,20,20,20,20,20,20,20,20,20 (une couche de + diminue les résultats)
dernier neurone softmax plutot que sigmoid (vu sur internet), test :
ancien résultat: [01:07<00:00, 59.19it/s, error_test=0.208, error_train=0.131, iter=3990, loss=0.324]
nouveau résultat: [01:11<00:00, 56.08it/s, error_test=0.475, error_train=0.483, iter=3990, loss=13.3]
donc softmax parait inapproprié à la situation.

on augmente le nombre de neurones de chaque couche
51,40,20,20,20,20,20,20,20,20 [01:17<00:00, 51.43it/s, error_test=0.214, error_train=0.136, iter=3990, loss=0.377]
51,80,40,20,20,20,20,20,20,20 [01:52<00:00, 35.44it/s, error_test=0.22, error_train=0.134, iter=3990, loss=0.383]
augmentation plus rapide
51,200,100,200,100,100,80,51,20,20 [08:32<00:00, 7.81it/s, error_test=0.238, error_train=0.0675, iter=3990, loss=0.193] Precision : 0.652694610778443 . ***Precision moin bonne malgrès une meilleure loss et error_train***

diminution de la quantité de neurones
51,25,20,10,10,10,10,10,10,20 [01:17<00:00, 51.62it/s, error_test=0.205, error_train=0.128, iter=3990, loss=0.34] ***Precision : 0.7375249500998005***
51,10,10,10,10,10,10,10,10,20 [00:45<00:00, 87.24it/s, error_test=0.214, error_train=0.18, iter=3990, loss=0.393]
51,10,10,10,10,10,10,10,2,20 [00:42<00:00, 95.06it/s, error_test=0.2, error_train=0.163, iter=3990, loss=0.384] ***Precision : 0.7684630738522954***
51,256,128,64,32,16,8,4,2,20 [06:10<00:00, 10.81it/s, error_test=0.222, error_train=0.0962, iter=3990, loss=0.292]
51,102,128,64,32,16,8,4,2,20 [03:30<00:00, 19.03it/s, error_test=0.209, error_train=0.109, iter=3990, loss=0.319]
résultats cohérents
51,20,10,2,2,2,2,2,2,20 [00:50<00:00, 79.56it/s, error_test=0.215, error_train=0.132, iter=3990, loss=0.349]
(surapprentissage probable. montée du taux d'echecs sur la fin)
51,2,2,2,2,2,2,2,2,20 [00:36<00:00, 108.82it/s, error_test=0.475, error_train=0.483, iter=3990, loss=0.693]
effectivement, surapprentissage confirmé
51,8,8,8,8,8,8,4,2,20 [00:38<00:00, 103.55it/s, error_test=0.196, error_train=0.183, iter=3990, loss=0.416]
51,4,4,4,4,4,4,4,2,20 [00:38<00:00, 103.18it/s, error_test=0.191, error_train=0.186, iter=3990, loss=0.42]
51,4,4,4,4,4,4,2,2,20 [00:37<00:00, 105.84it/s, error_test=0.196, error_train=0.201, iter=3990, loss=0.433]