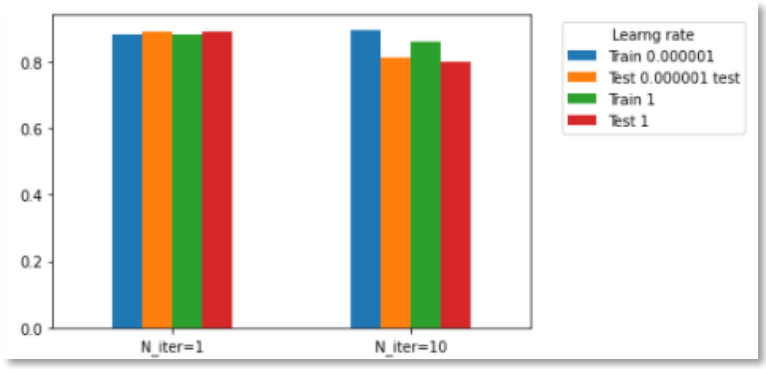
TP 2 Réseaux de neurones : Edouard Nadaud – Pierric Loedel – 41UFA

1)

a) Pour commencer ce tp nous avons importé plus split le dataset en 70/30, comme lors du tp1. Puis nous affichons les données test et train.

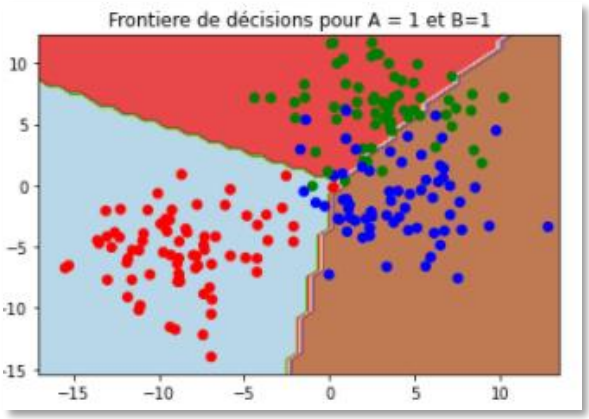
b) Nous avons ensuite défini le réseau de neurones classifieur mono-couche.
Nous possédons 300 données, 2 features et 3 classes. Chaque perceptron recevra en entrée 2 features (poids) et aura pour seuil : -1 ou 1. Comme nous avons 3 couches nous aurons besoin de 3 perceptrons soit 3*2 entrées donc 6 poids au total et 3 seuils (-1 ou 1) pour les 3 perceptrons.

c) Vous pourrez voir en annexe les manipulations réalisées, notamment avec la fonction score. Les résultats obtenus grâce aux données précédemment déterminées sont tels que :



Et ce selon les valeurs de train et test données en consigne.

A titre d'exemple, pour les valeurs A=1 et B=1, nous trouvons, grâce notamment à la fonction score, les frontières de décisions suivantes :



Les résultats précédents varient-ils lorsque l'on relance plusieurs fois le même apprentissage ? Pourquoi ?

Ici les résultats ne vont pas varier car nous avons mis un random states, si on relance à chaque fois le mélange sera toujours le meme et la classification du model aussi car le random state a aussi été initialisé pour la création du model. Si l'on enlève le randomstate alors les résultats varieront car le mélange sera différent à chaque lancement du split. Conséquences, les entrainements ne seront pas les mêmes ni le test et les performances varieront.
De plus, lors de la descente de gradient un point au hasard comme départ est pris, les performances dépendent donc de ce point, de sa distance avec le minimum ainsi que de notre nombre d'itérations rentrée en hyperparametre.

Que doit-on envisager pour améliorer les performances ? Justifier votre réponse.

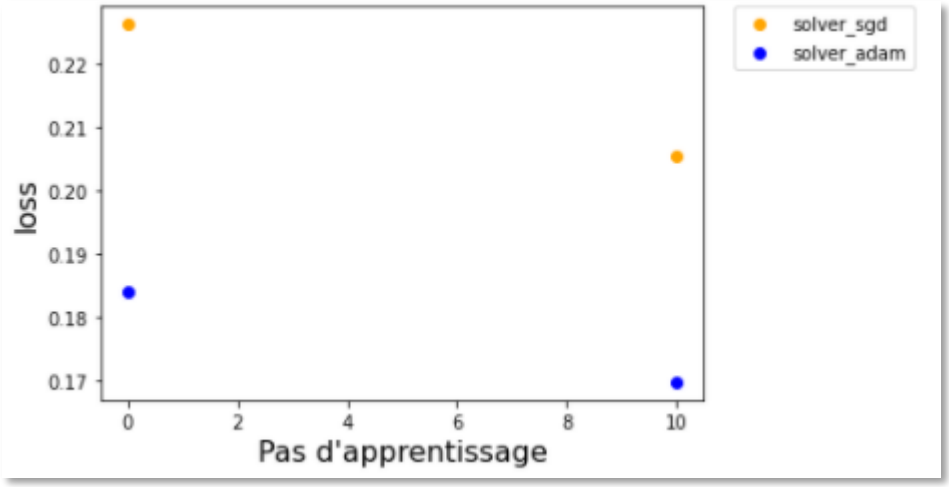
Pour améliorer les performances, on peut réduire le pas et augmenter le nombre d'itérations. Cela nous permettrait d'être au plus précis dans notre descente de gradient mais demandera beaucoup plus de temps de calcul.
Néanmoins les potentiomètre seront réglés au plus précis. On peut aussi juste augmenter le nombre d'itérations avec le même pas que précédemment cela demandera un peu moins de calcul . Nous irons plus loin dans la descente de gradient. Ou Bien augmenter le nombre de neurones et de couches mais cela augmente considérablement le nombre de potentiomètre et donc de réglages nécessaires (plus de temps de calcul).
Sur un petit dataset, cela ne dérange pas car nous avons peu de feature et donc relativement peu de poids et seuil à ajuster,mais si nous avions des millions de features, cela deviendrait compliqué.

2)

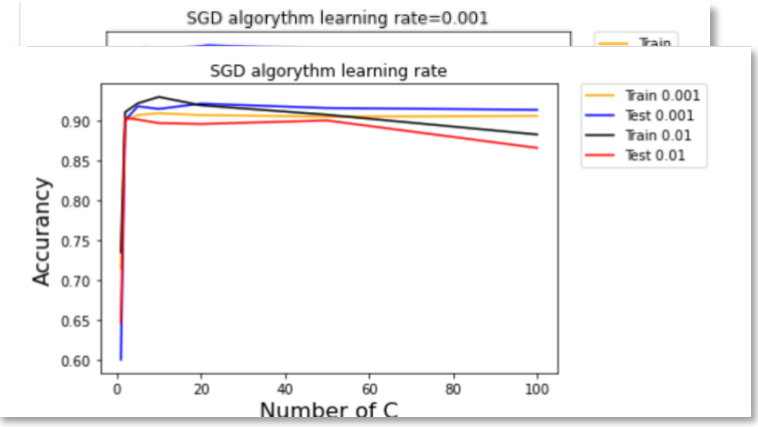
a) On commence par définir un réseau de neurones classifieur multi-couches.
b) Après calculs visibles en annexe, la valeur du coût de fin d'apprentissage et du nombre d'itérations effectuées sont les suivantes :

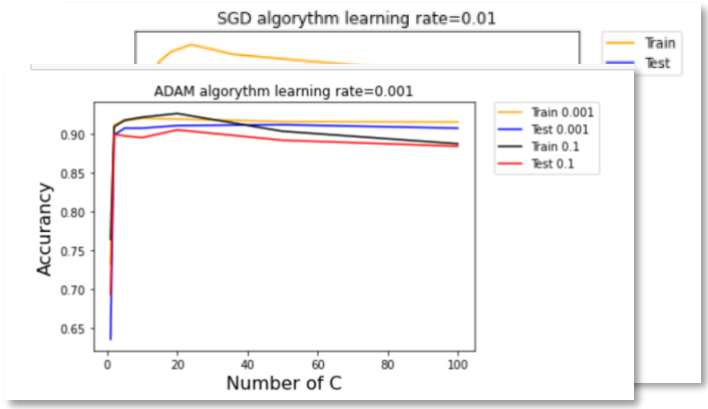
```
La valeur du coût en fin d'apprentisage est de 0.22611778744555958
Le nombre d'itération éfféctué est de : 100
Le score de train est de  : 0.9095238095238095
Le score de test est de  : 0.9111111111111111
```

Nous multiplions le pas d'apprentissage par 10 puis nous changeons le solver en adam au lieu de sgd. Les résultats obtenus sur le graph ci-après nous montrent simplement que le solver sgd nous renvoie des résultats de loss plus haut donc une classification moins éfficace qu'avec le solver adam. Les meilleurs paramètres à utiliser sont donc ceux pour lesquels nous avons le moins de loss, soit le point de solveur adam ayant pas d'apprentissage = 10 et loss = 0,17 sur le graph. Cependant, il ne faut pas oublier que la précision amenée va avec la lenteur d'exécution.

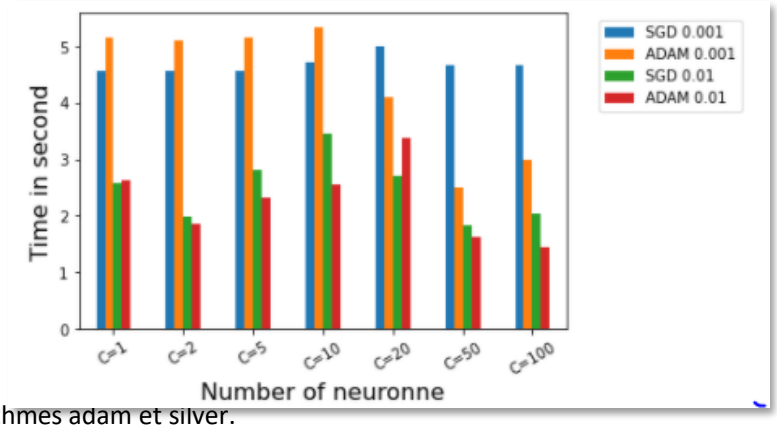


c) Pour trouver le nombre optimal de neurones cachés C*, nous ne remontrerons pas les lignes de commandes ici car celles-ci prendraient trop de place, mais elles sont visibles en annexe. Quant aux résultats, nous obtenons :



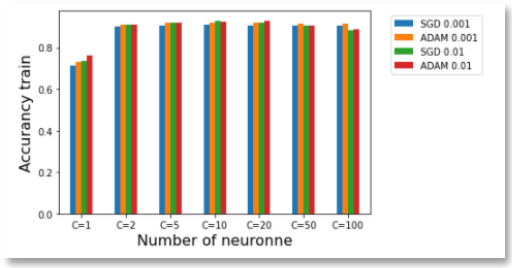


Ensuite, nous avons donc calculé la moyenne des temps (en secondes) par rapport au nombre de neurones, C =1, 2, 5, 10, 50 et 100 :
En termes de temps de calcul, nous pouvons observer que jusqu'a 10 neurones dans la couche cachée sgd avec un learning rate de 0.001 est plus performant que l'algo adam. Néanmoins en termes de temps de calcul il est préférable de prendre l'algorithme adam ave un learning rate de 0.01. Il est logique que plus le pas d'apprentissage est grand moins le temps de calcul est long, car on arrive plus rapidement au minimum de la courbe surtout pour un jeu de données aussi faible avec peu de feature.

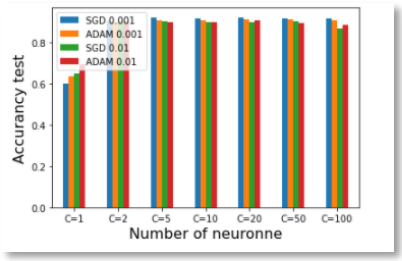


Nous avons par la suite calculé la précision des algorithmes adam et silver.
Les résultats obtenus sont ceux selon les valeurs SGD 0.001, ADAM 0.001, SGD 0.01, ADAM 0.01 et également selon la valeur de C.

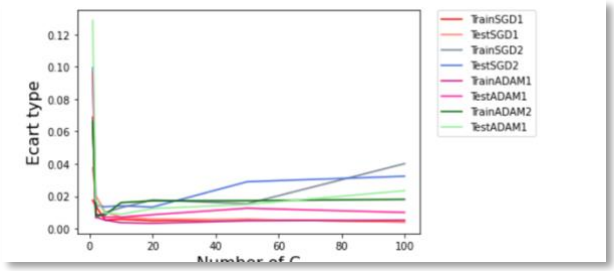
En train :



En test :



Calcul des ecarts type



Pour SGD1, on constate que l'écart type tend vers quasi exactement la même chose entre train et test, avec un écart type très faible et tendant vers 0,005.

Pour le SGD2, l'écart type connaît un plus grand écart entre test et train. De plus, globalement, l'écart type va tendre vers 0,04, ce qui est plus haut que ce que nous avons pu obtenir avec le SGD1, et donc moins rentable.

Pour ADAM1, l'écart type est plutôt stable et varie à environ 0,01. Cette valeur est toujours plus haute que celle obtenue avec SGD1 et donc moins rentable.

Enfin pour ADAM2, nous obtenons des écarts types proches entre test et train, cependant la moyenne des deux se situe à 0,02, soit un écart encore trop élevé.

Avec le graph global, on remarque que les courbes les plus intéressantes et à écart type faible sont celles de train et test SGD1.
On sait qu'afin d'obtenir la meilleure solution possible, il faut minimiser au plus l'écart type. Alors on comprend ici que la meilleure solution est apportée par le SGD1.
Au vu du temps, de l'ecart type est des performances la valeur optimal de C est de 2 pour le model sgd avec un learning rate =0.01. Et de 10 pour les trois autres models car ce nombre de neuronnes a le meilleur compris entre temps de calcul, perfomance et ecart type. En effet en maching learning nous essayons de minimiser l'ecart type pour clusteriser au mieux possible les variables.

Malheureusement, au vu du temps de calcul et de la précision apportée en plus, le gain n'est absolument pas suffisant different pour utiliser un classifieur multicouches, pour ce petit dataset avec peu de features.
L'avantage d'un réseau de neurones et de ne prendre que les features et non chaque donnée pour tout recalculer comme un knn. Cela a l'avantage de prendre moins de temps calcul sur un dataset de millions de données, neanmoins plus il y a de features plus il y a de poids a ajusté et donc de calcul et de temps . l'ideal est de couplé une PCA qui réduira notre nombre de dimension a un réseau de neuronne pour accelerer le processus. Pour un petit dataset, il est mieux e priviliéger un reseau monocouche plus rapide et avec un gain assez elevé que un réseau multicouche. Le rapport biais (erreur) sur variance (résilience du model) reste assez corecte (notre algorythme n'est pas sur ou sous entrainée). Cependant, le jour où nous aurons beaucoup (trop) de données à classer, cela ne sera pas une solution aussi rentable et il faudrait priviliégé un model MLP. En effet notre cout pour un réseau de neuronne une seul couche était de 0.1 en train et 0.2 en test soit environs entre 18 et 20 données mal classé sur notre train de 210 donnée et notre test de 90 données. En utilisant notre réseau MLP notre cout minimun était de 0.07 en train et 0.1 soit environ 15 donnée et 9 données en test de mal classée. La différence n'etant pas significative sur un petit dataset comaré au temps de calcul et aux ressources demandé pour un MLP nous priviliegerons le reseau mono couche. Néanmoins si on multiplie la taille du dataset par 1000 en supoosant avoir les meme resultats de cout nous arrivons a une mauvaise classification de 18000 et 20000 pour notre réseau moncouche. Et pour notre MLP nous avons 9000 et 15000 ; La différence commence a devenir significative meme si les datasets restent pourtant le temps de calcul n'augmenterais pas enormement car un reseau de neurone se base sur les features et non le nombre de données c'est pourquoi pour de plus grand dataset nous priviliegeront un MLP.
Les résultats d'écarts types que nous avons actuellement seraient faibles et presque inexistants pour un petit nombre de données, par exemple là ou 0,04 d'écart type reviendrait à 4%, il y aurait un petit écart pour 100 données, mais un écart énorme pour 1000 fois plus de données. Nous utiliserons donc un réseau d'1 seule couche.

Out[140]:

	C=1	C=2	C=5	C=10	C=20	C=50	C=100
score train	0.970952	0.991905	0.991429	0.991905	0.991905	0.991429	0.991905
poid	5.000000	10.000000	25.000000	50.000000	100.000000	250.000000	500.000000
ecart type test	0.161230	0.075719	0.049889	0.057778	0.051652	0.040307	0.040000
ecart type train	0.099751	0.041905	0.034352	0.032381	0.032381	0.033350	0.033350
score test	0.955556	0.990000	0.984444	0.986667	0.985556	0.990000	0.990000
biais du neuronne	8.000000	10.000000	16.000000	26.000000	46.000000	106.000000	206.000000
valeurs libre	13.000000	20.000000	41.000000	76.000000	146.000000	356.000000	706.000000