Rapport Réseaux de neurones

Badr GHAZLANE et Clément FERNANDES

25/01/2017

Introduction

Le but de ce TP est de comparer les performances d'un Deep Belief Network (DBN) pré-entraîné et un autre non pré-entraîné. Pour cela nous allons créer deux DBN, le premier sera pré-entrainer en considérant chacune des couches intermédiare comme un RBM (Restricted Boltzmann Machines) et en utilisant donc sur chacune des couches l'algorithme de contrastive divergence avec nos données. Puis nous utiliserons l'algoritmhe de retro-propagation du gradient avec nous données et nos labels pour entraîné de manière supervisé ce DBN. Le deuxiéme DBN sera lui juste entraîné de manière supervisé par l'algorithme de rétro-propagation du gradient.

Nos données sont des matrice 20*16 qui représentes des lettres (de A à Z) et des chiffres (de 1 à 9) écrits à la main.

I) Le RBM

- Pour nous familiariser avec le pré-entrainement nous allons tout d'abord pré-entrainer un RBM simple avec l'algorithme de contrastive divergence (Voir Script1). Pour vérifier que le pré-entrainement est un succés nous regénérons des données nouvelles par ce RBM (grâce à sa propriété de modèles génératifs).

Fig.1 : "A" appris puis générés par DBN

Fig.2 : "A" appris puis générés par DBN

Fig.3 : "A" appris puis générés par DBN

- On remarque que l'algorithme nous génére des lettres, notre modèle est bien pré-entraîné. Cependant si

on rajoute plus de lettre dans le pré-apprentissage les résultats sont moins bon.

Fig.4 : "C" appris puis générés par DBN

Fig.5 : "D" appris puis générés par DBN

Fig.6 : B" appris puis générés par DBN

II) Le DNN

- Nous allons maintenant pré-entrainer notre DBN sans considérer la couche de sortie. Pour cela nous allons supposer que ce DBN se comporte comme un empilement de RBM et nous allons les pré-entrainer chacun à la suite. (voir Script2). Pour vérifier que le pré-entrainement est un succés nous regénérons des données nouvelles par ce DBN

Fig.7 : "A" appris puis générés par DBN

Fig.8 : "A" appris puis générés par DBN

Fig.9 : "A" appris puis générés par DBN

- On remarque que l'algorithme nous génére des lettres, notre modèle est bien pré-entraîné. Par ailleurs on constate que les lettres sont plus lisibles que celles générées par notre RBM et que l'algorithme est capable d'apprendre et de regénérer un plus grand nombre de caractères à la fois.

Fig.10 : "D" appris puis générés par DBN

Fig.11 : "A" appris puis générés par DBN

Fig.12 : "C" appris puis générés par DBN

III) Comparaison

- Nous avons bien pré-entraîné notre DBN, maintenant nous allons rajouter la couche de sortie à celui-ci, puis nous allons le comparer avec un autre DBN non-pré-entrainé sur la reconnaissance des caractères écrits à la main. (Script3)
- Nous allons ensuite comparer les taux d'erreurs des deux DBN en faisant varier le nombre de couches de neurones utilisées (script4), le nombre de données utilisées (script6) et le nombre de neurones par couche (script5)

IV) Importation des données:

- Nous laissons un exemple de code qui permet d'importer les données.

```
Vect1 <- read.csv("~/Downloads/Vect/tauxErr_c=[2 3 4 5]_n=200_d=3000.csv",header=F)
Vecte1 <- read.csv("~/Downloads/Vect/tauxErr_pre_c=[2 3 4 5]_n=200_d=3000.csv",header=F)
```

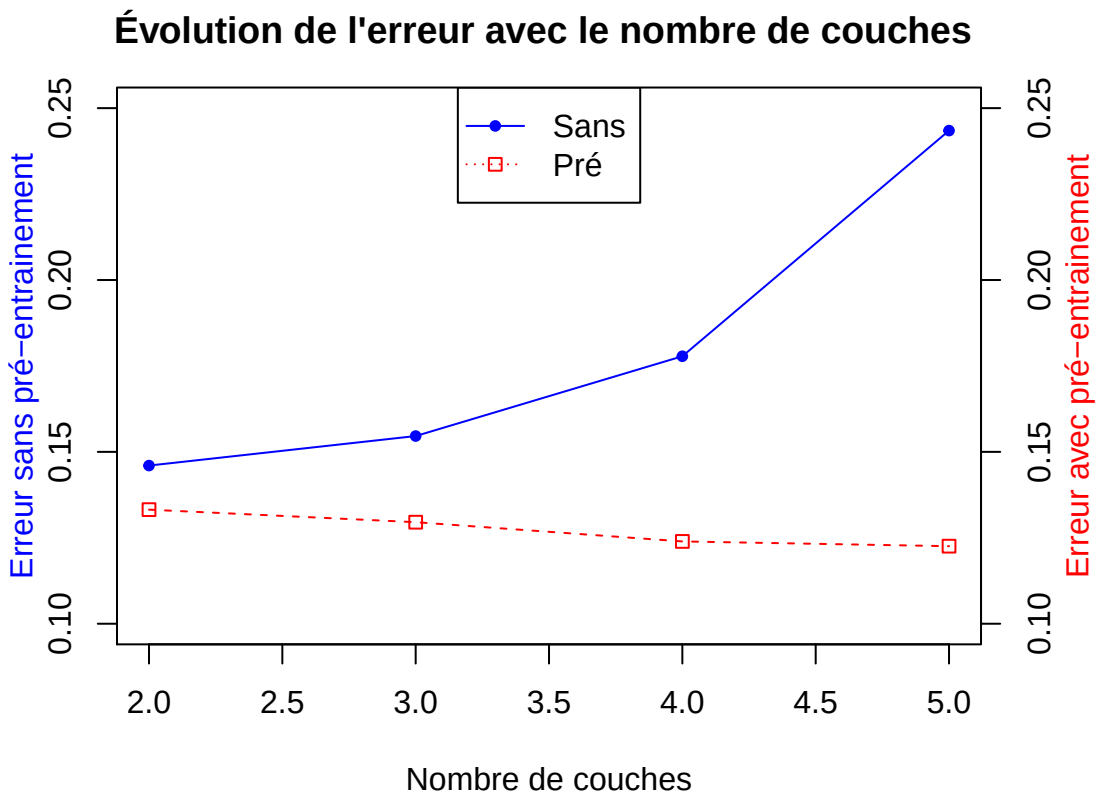
V) Graphes

Nous laissons un exemple de code permettant de générer le graphe.

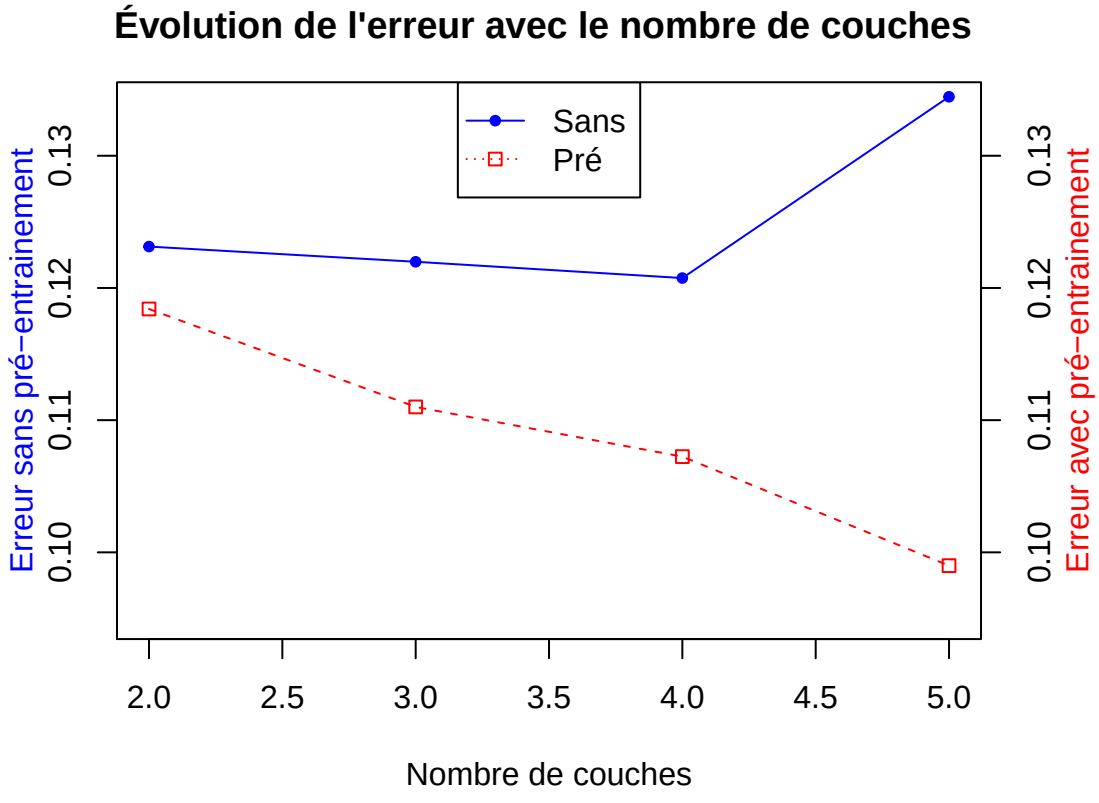
V.A) Taux d'erreurs en fonction du nombre de couches

V.A.1 Nombre de neurones par couches=200 et 3000 données

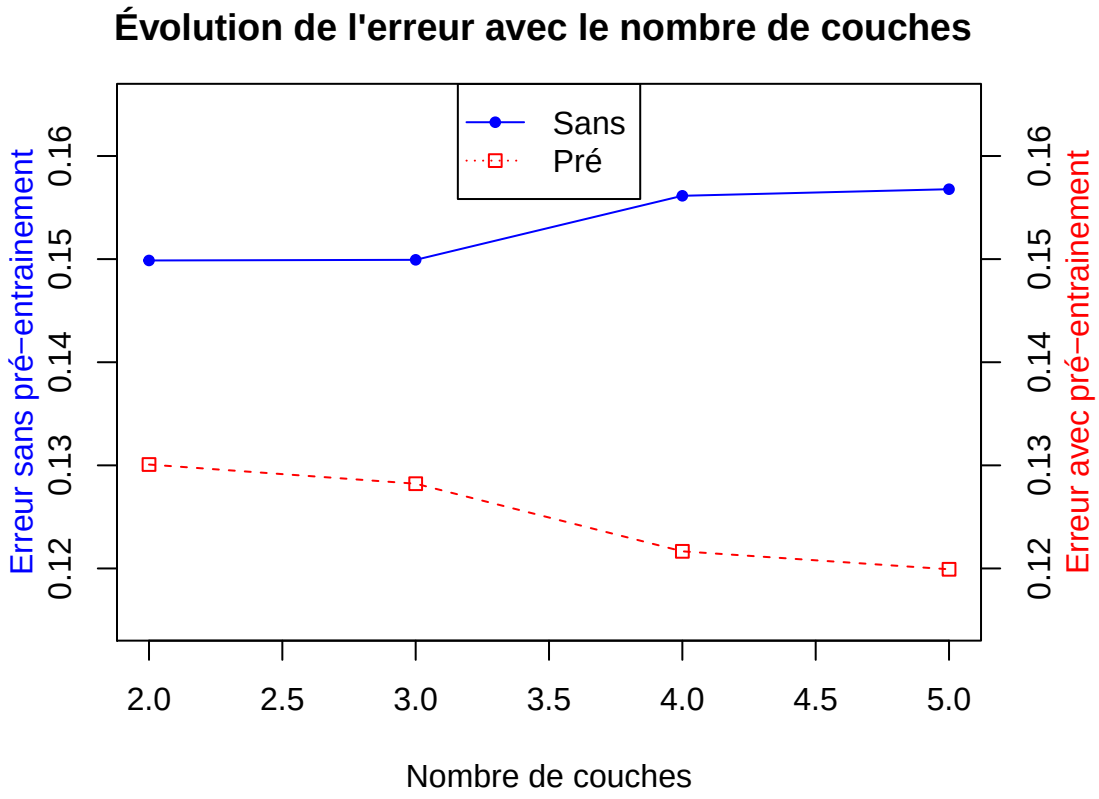
```
par(mar = c(5,5,3,5))

a=c(0.10,0.25)
#premiere courbe
 plot(2:5,Vect1, type="o",ylab=NA,ylim = a

              ,xlab="Nombre de couches"
              , main="Évolution de l'erreur avec le nombre de couches "
              ,cex=1
              ,lwd=1
              ,lty=1
              ,pch=20,col="blue")
mtext("Erreur sans pré-entrainement", 2,side=2, col="blue")


#deuxieme courbe
par(new = T)

     plot(2:5,Vecte1, axes=F, xlab=NA, ylab=NA, type="o",col="red", pch=22, lty=2,ylim = a)
axis(side = 4)
mtext("Erreur avec pré-entrainement", 2,side=4, col="red")


#legende
legend("top",
       legend=c("Sans", "Pré"),
       lty=c(1,3), pch=c(20, 22), col=c( "blue","red"))
```



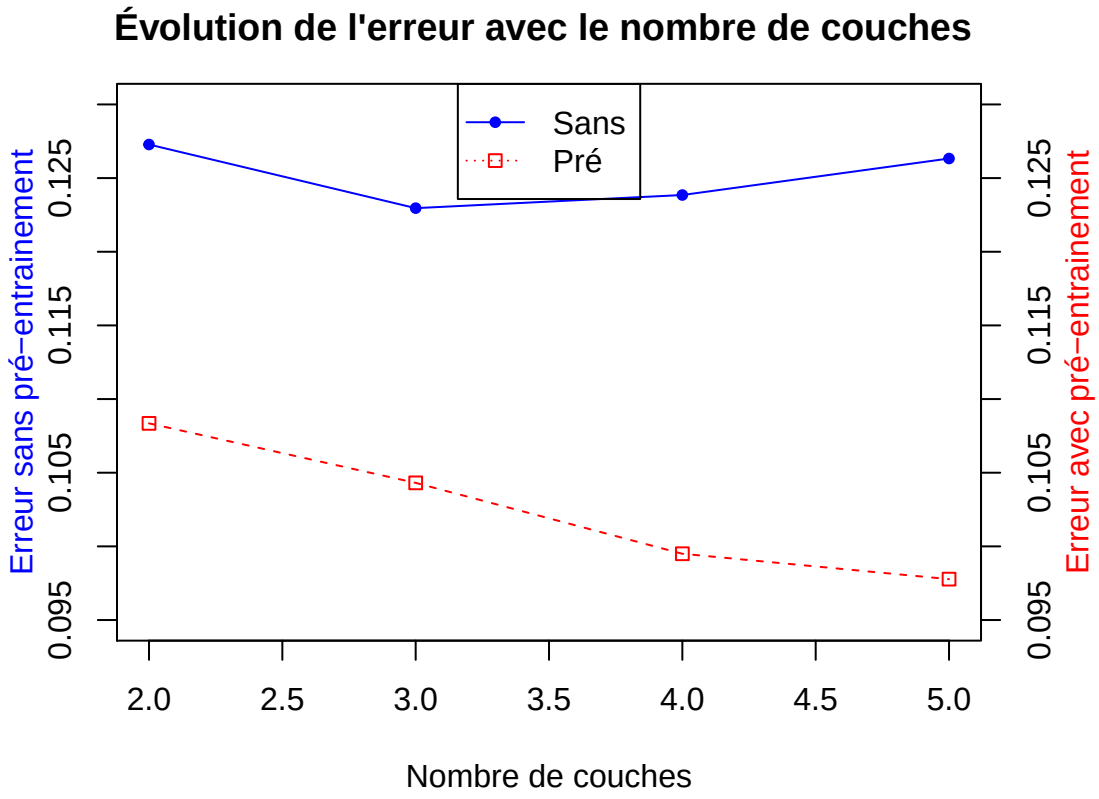
V.A.2) Nombre de neurones par couches=200 et 10000 données



V.A.3) Nombre de neurones par couches=500 et 3000 données



V.A.4) Nombre de neurones par couches=500 et 10000 données

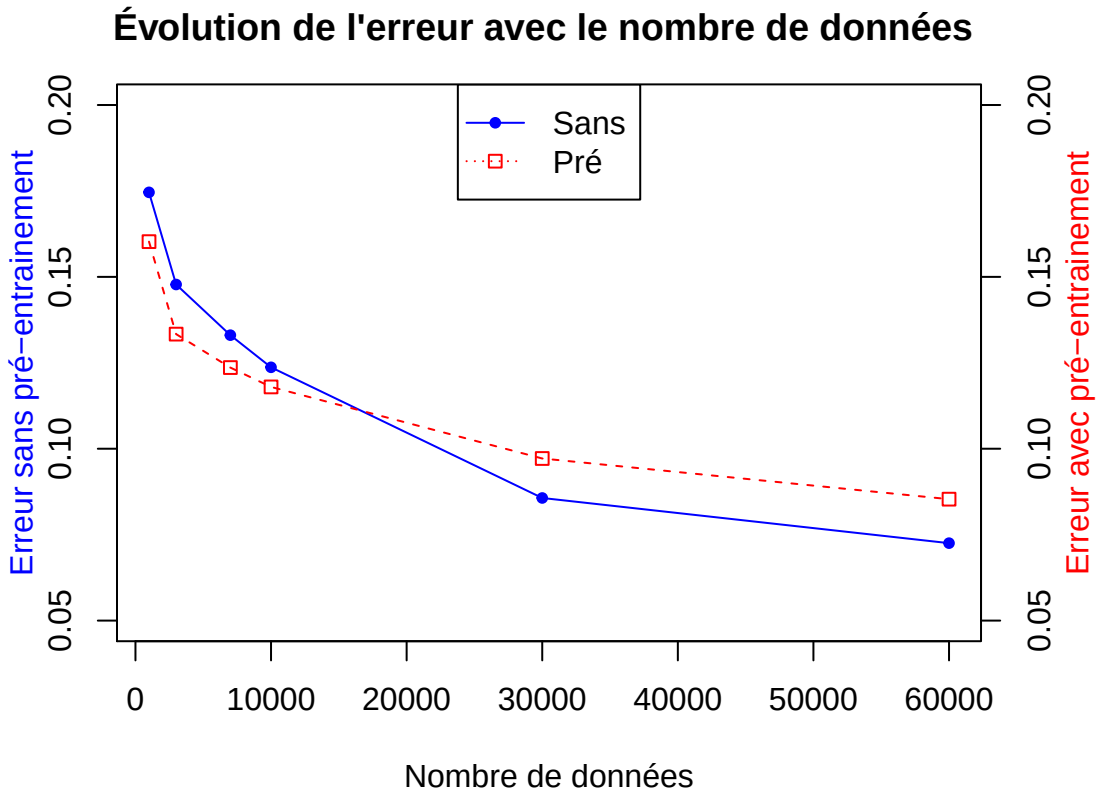


Analyse

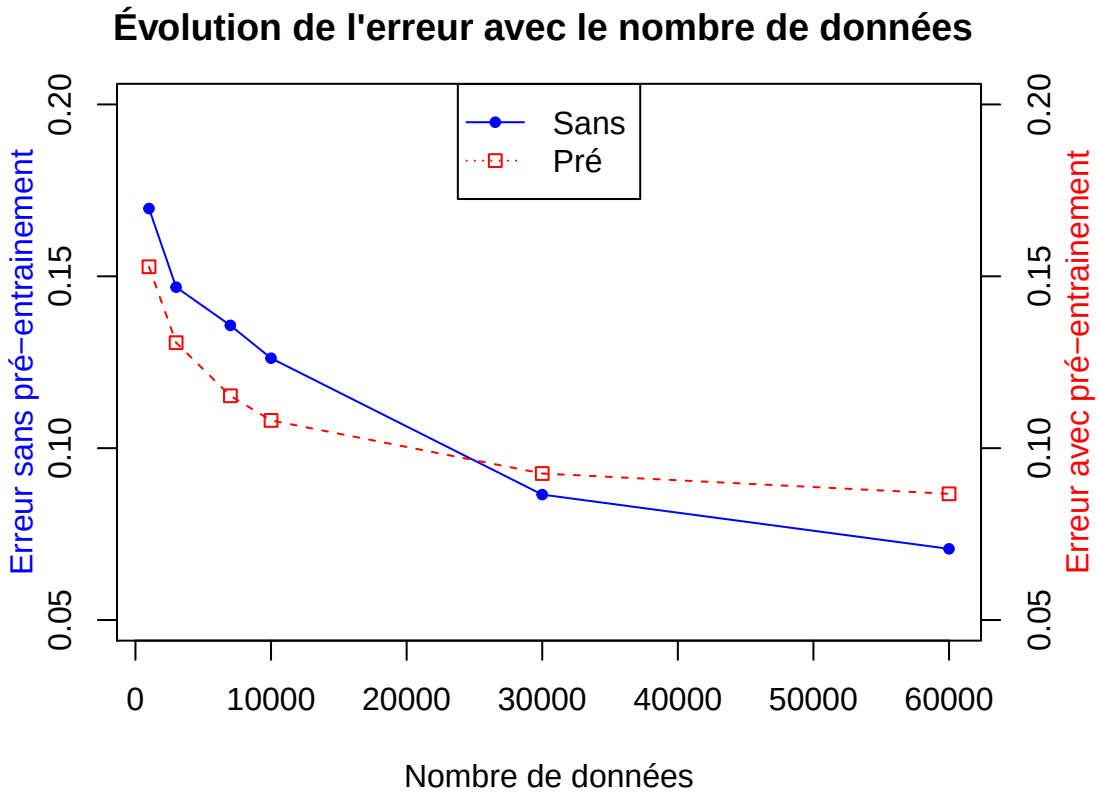
On remarque que pour le réseau pré-entrainé, le taux d'erreur va décroitre en fonction du nombre de couche et cela, peu importe le nombre de données choisies et le nombre de neurones par couches. Il est par ailleurs meilleur que le réseau non pré-entrainé. Ce dernier quant à lui a une variation différente en fonction du nombre de couche. On peut en effet remarquer que pour un nombre de données faible (ici 3000) le taux d'erreur augmente en fonction du nombre de couche mais cette augmentation est ralentie si on a plus de neurones par couches. Pour un nombre de données plus important le taux d'erreur commence par décroître en fonction du nombre de couche puis il recommence a croître passé un certain stade. Ainsi pour un réseau pré-entrainé, augmenter le nombre de couche va réduire le taux d'erreur alors qu'il semble que pour un réseau non pré-entrainé il faut ajuster le nombre de couche en fonction du nombre de données: un nombre de couche élevé ne réduira le taux d'erreur que si il y a suffisamment de donnée d'apprentissage, si ce n'est pas le cas le taux d'erreur augmentera.

V.B) Taux d'erreurs en fonction du nombre de données d'entrainement

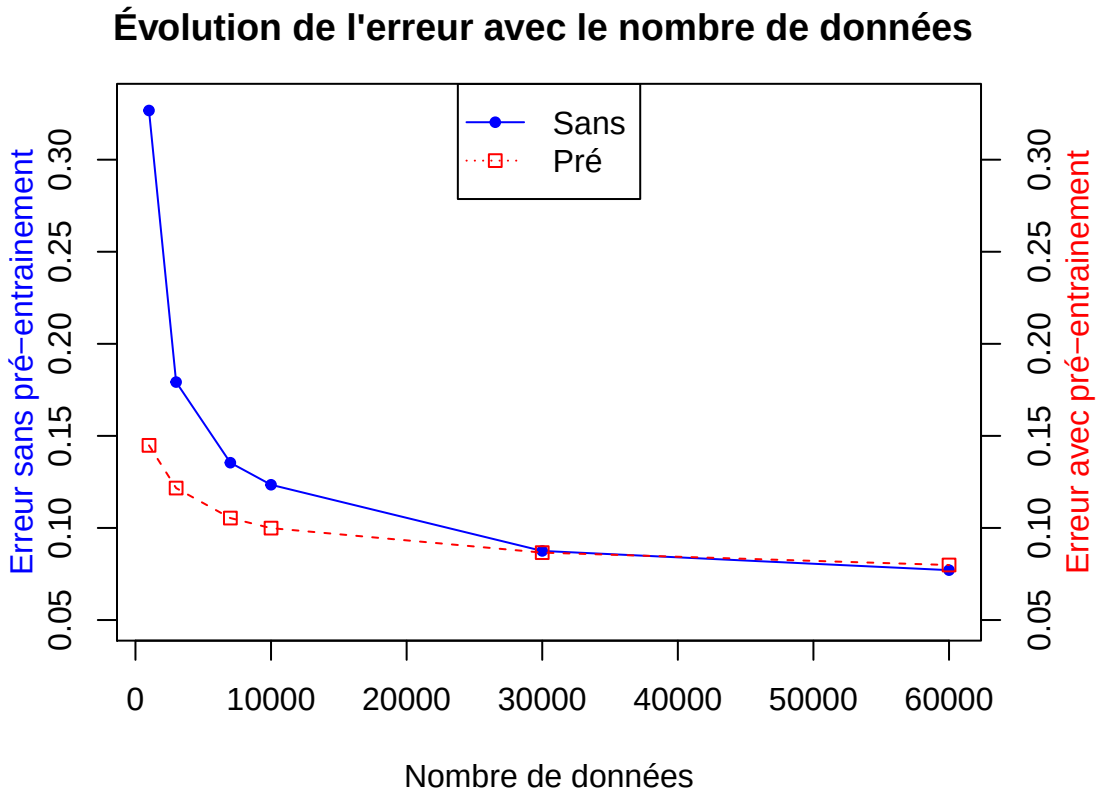
V.B.1) Nombre de neurones par couches 200 et nombre de couches 2



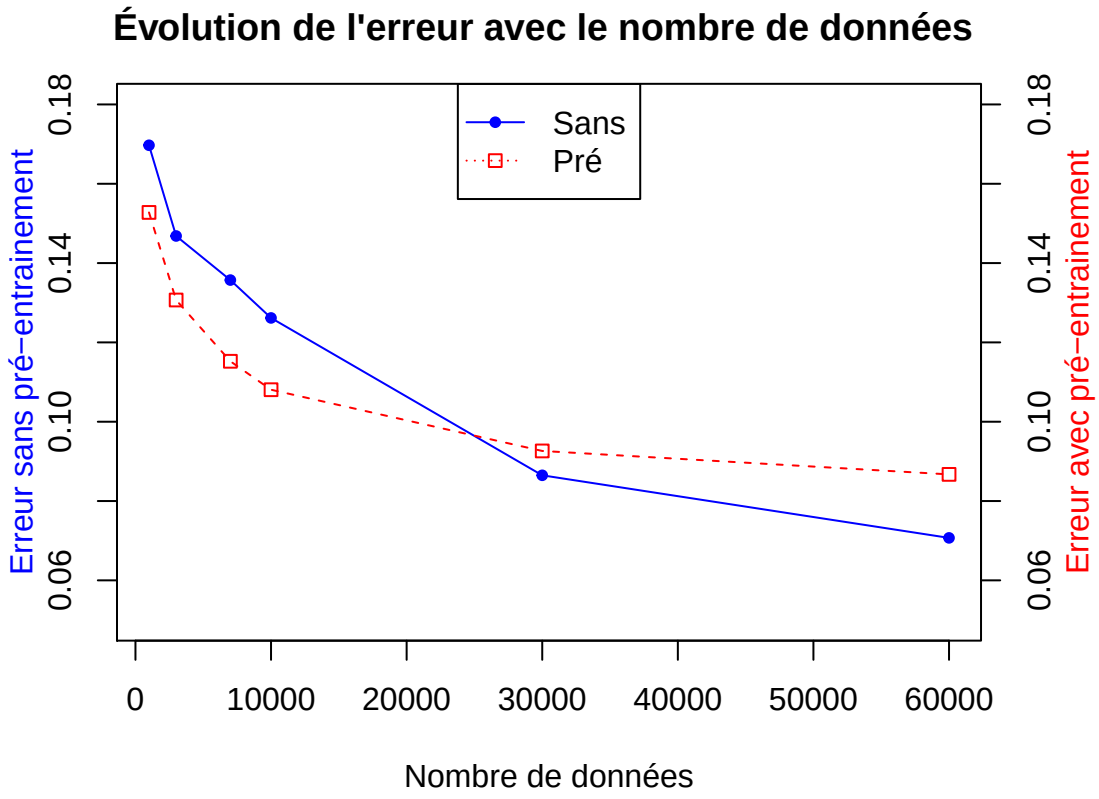
V.B.2) Nombre de neurones par couches 200 et nombre de couches 4



V.B.3) Nombre de neurones par couches 500 et nombre de couches 2



V.B.4) Nombre de neurones par couches 500 et nombre de couches 4

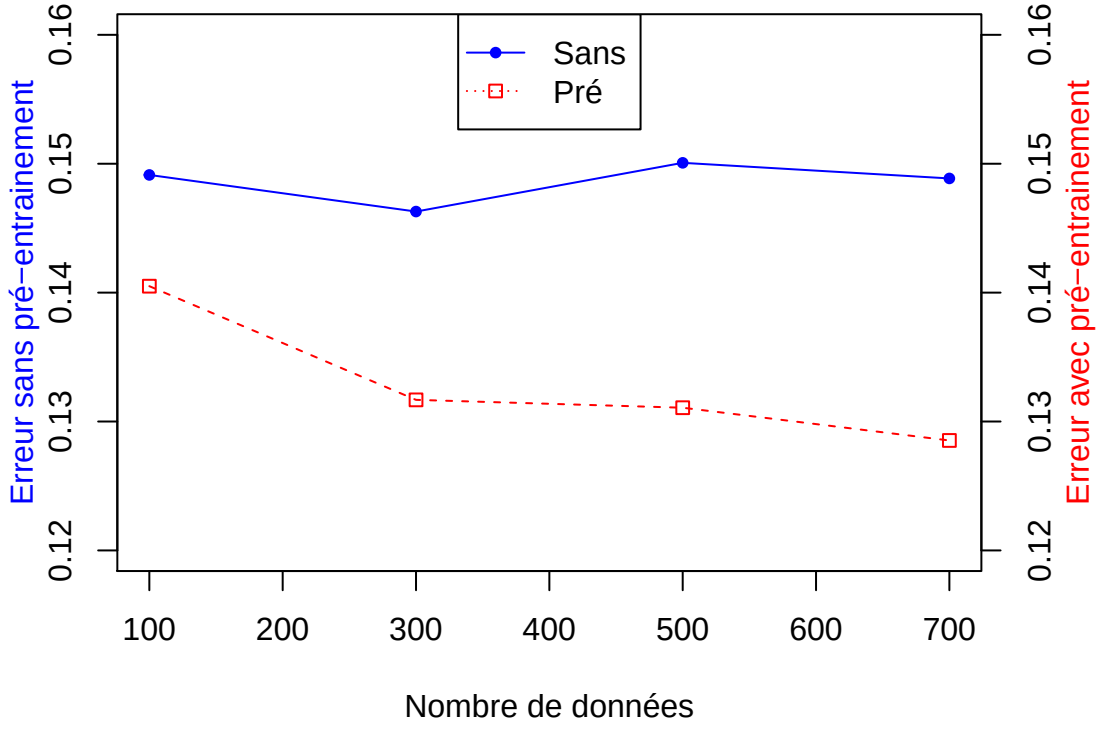


Bilan

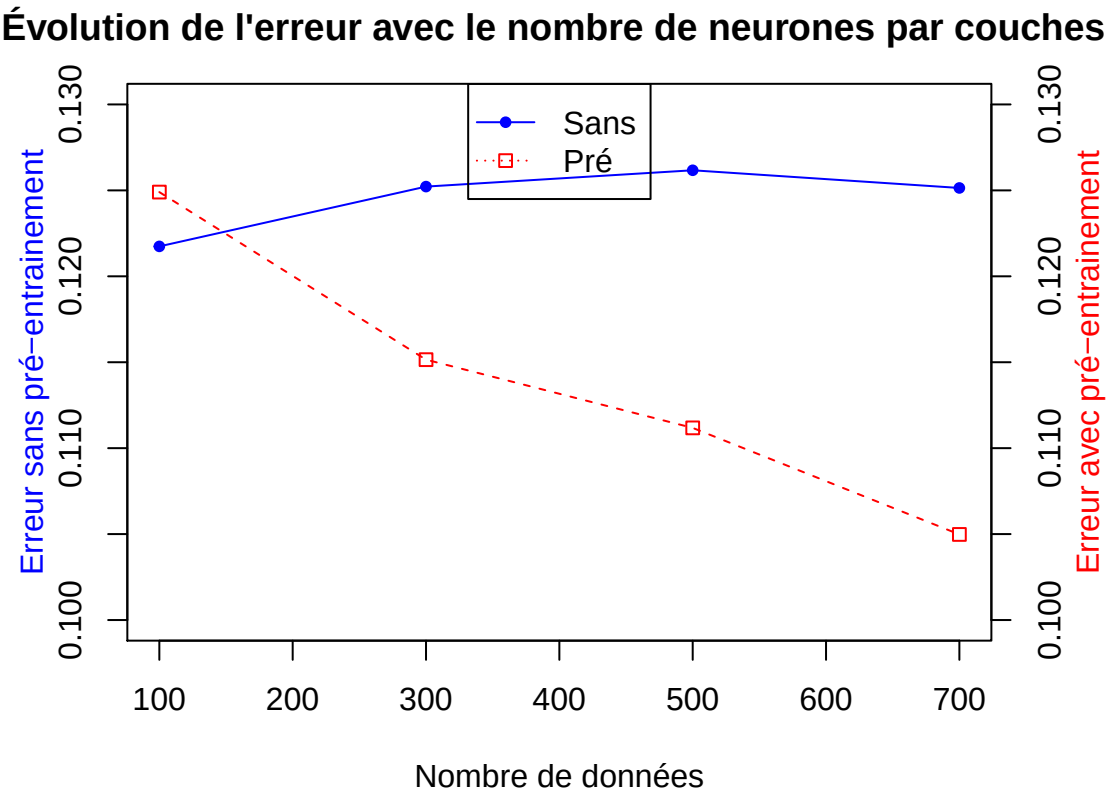
On remarque que le réseaux pré-entrainé est plus performant avec un nombre de données d'apprentissage plus faible, mais si on augmente le nombre de donnée la décroissance du taux d'erreur va ralentir et le réseau non pré-entrainé sera plus performant. Ce résultat est paradoxal, puisque le reséau pré-entrainé est censé etre plus précis. En effet, le pré-apprentissage permet de situer une zone de l'espace des paramètres du résau de neurones. Mais, l'explication plausible est qu'avec une très grande quantité de donnée, et un espace des paramètres déjà défini, on a un phénomène qui peut s'apparenter à du sur-apprentissage. Mais comme l'erreur de test continue de décroitre, le terme sur-apprentissage n'est pas le plus adapté.

V.C) Taux d'erreurs en fonction du nombre de neurones par couches

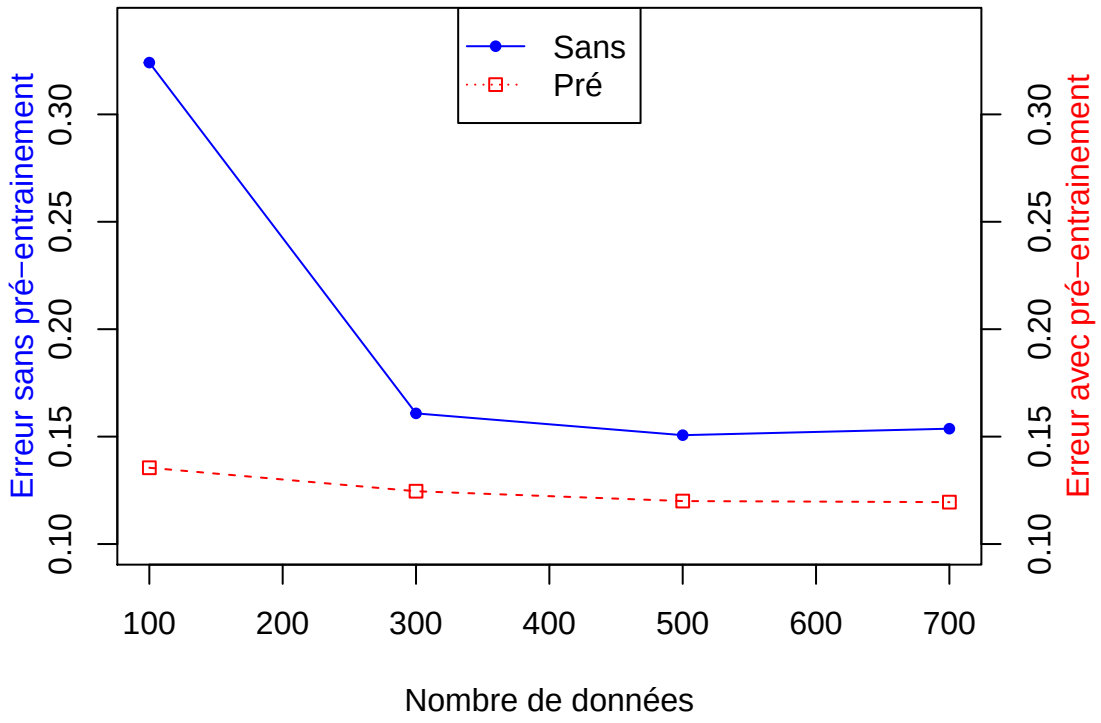
V.C.1) Nombre de couches 2 et nombre de données 3000



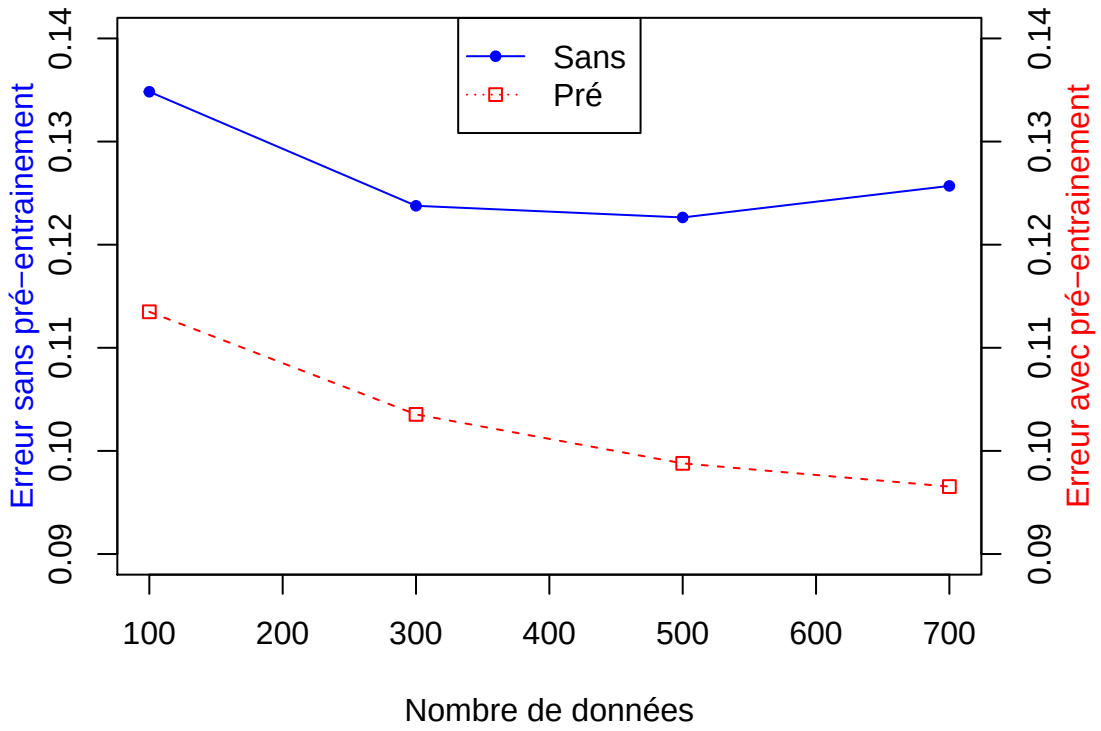
V.C.2) Nombre de couches 2 et nombre de données 10000



V.C.2) Nombre de couches 4 et nombre de données 3000



V.C.4) Nombre de couches 4 et nombre de données 10000



Pour le réseau pré-entrainé, le taux d'erreur décroit en fonction du nombre de neurones par couche. On remarque quand même que la decroissance est plus prononcé pour un nombre de données plus important alors que la décroissance est atténuée par le nombre de couches. Pour le réseau pré-entrainé on remarque que pour un nombre de couche faible le nombre de neurones par couche ne fait pas trop varier le taux d'erreur. Cependant si on augmente le nombre de couche, le nombre de neurones par couche fait diminuer le taux d'erreur du réseau non pré-entrainé jusqu'à un certain palier, une fois ce palier atteint le taux d'erreur ne diminue plus. Le réseau pré-entrainé est toujours meilleur que le réseau non pré-entrainé avec l'augmentation du nombre de neurones par couche.

Conclusion

Le réseau pré-entraîné est beaucoup plus efficace dans le cas de données moins importantes. En effet, il donne de meilleurs résultats de taux d'erreurs et on peut facilement baisser ce dernier en augmentant le nombre de couches et le nombre de neurones par couche. Le réseau non pré-entraîné sera beaucoup moins efficace dans le cas d'un nombre de données faible d'autant plus qu'augmenter le nombre de couche ou le nombre de neurones par couche ne va pas faire baisser le taux d'erreur mais risque au contraire de l'augmenter. Pour un nombre de données plus élevé, le réseau non pré-entrainé commence a devenir plus efficace, en effet pour un nombre de données élevé le nombre de couche et le nombre de neurones par couche vont faire diminuer son taux d'erreur. De plus il semble que le taux d'erreur de réseau pré-entrainé décroît plus lentement en fonction du nombre de données que celui du réseau non pré-entrainé. Il existe alors un nombre de données limite dépendant du nombre de couches et du nombre de neurones par couche, pour lequel le réseau pré-entrainé sera moins performant que le réseau non pré-entrainé.

Plus précisément, le pré-entraînement non-supervisé initialise l'apprentissage supervisé dans une bonne région de l'espace des paramètres, à partir de laquelle une descente locale trouve une meilleure solution que celles

trouvées à partir d'initialisation aléatoire. Le pré-entraînement non-supervisé permet donc d'optimiser, en favorisant les poids des couches inférieures qui sont cohérentes avec un bon modèle de la distribution des entrées: en d'autres termes, pour bien généraliser il faut que les couches cachées soient bien entrainées à capter des features pertinents des entrées.

Annexe

Pour vérifier nos résultats:

- Le **script1** correspond au pré-entrainement et à la génération de nouvelle donnée avec un RBM
- Le **script2** correspond au pré-entrainement et à la génération de nouvelle donnée avec un DBN
- Le **script3** est le script ou on compare pour la première fois les taux d'erreurs des deux DBN.
- Le **script4** permet d'obtenir les valeurs du taux d'erreur en fonction du nombre de couche de neurones et ce pour deux nombres de neurones par couches et deux tailles de sous ensembles de donnée différents.
- Le **script5** permet d'obtenir les valeurs du taux d'erreur en fonction du nombre de neurones par couches et ce pour deux nombres de couchess et deux tailles de sous ensemble de donnée différents.
- Le **script6** permet d'obtenir les valeurs du taux d'erreur en fonction du nombre de données utilisées pour l'entrainements et ce pour deux nombres de couches et deux nombres de neuronne par couches différents.