TP DNN

Enzo Cabaret

February 7, 2021

1 Introduction

On etudie dans ce devoir le pouvoir de reconstruction des rbm et dbn ainsi que les capacités de reconnaissance de caractères des dnn pré-entrainés ou non

2 Pseudos Codes principaux

On introduit les algorithmes principaux d'entrainement utilisé, CD-1 pour les Rbm et descente de gradient pour notre dnn.

Pour actualiser les poids du rbm avec une couche v visible de nv neurones, une couche h cachée de nh neurones et des paramètres de poids W,a,b. On utilise la méthode de contrastive divergence 1 pour une donnée x on effectue la procédure suivante :

Algorithm 1: CD-1

calculer $P(h = 1 \mid v = x)$
tirer h_{gibbs} selon la distribution $P(h \mid v = x)$
tirer v_{gibbs} selon la distribution $P(v \mid h = h_{gibbs})$
calculer $P(h = 1 \mid v = v_{gibbs})$
$dW \leftarrow x \cdot P(h = 1 \mid v = x)^T + v_{gibbs} \cdot P(h = 1 \mid v = v_{gibbs})^T$
$db \leftarrow x - v_{gibbs}$
$da \leftarrow P(h = 1 \mid v = x) - P(h = 1 \mid v = v_{gibbs})$

Pour actualiser les poids du dnn par descente de gradient on applique l'algorithme suivant pour une donnée x de label y qui génère une sortie x_{sortie}

Algorithm 2: Descente de gradient

$\delta = x_{sortie} - y$
for rbm dans liste rbm inversée **do**
  $dW_{rbm} \leftarrow \text{sortie}_{\text{rbm}} \cdot \delta^T$
  $db_{rbm} \leftarrow \delta$
  if rbm n'est pas le premier element de list rbm **then**
    $\delta \leftarrow sortie_{rbm} \odot (1 - sortie_{rbm}) \odot (\delta \cdot W_{rbm}^T)$
  end if
end for

Ne pas oublier avant d'actualiser les poids de multiplier par le taux d'apprentissage.

3 Travaux préliminaires

On présente ici les résultats de régénération de données produite par un rbm à 50 neurones cachées ainsi que un dbn à trois couches cachée de 50 neurones. Le rbm et le dbn ont été entraînés sur des 0,8,5 manuscrits de la base binary alpha digit. Le choix spécifique de ces données repose sur leur similarité lors du passage à l'écrit.

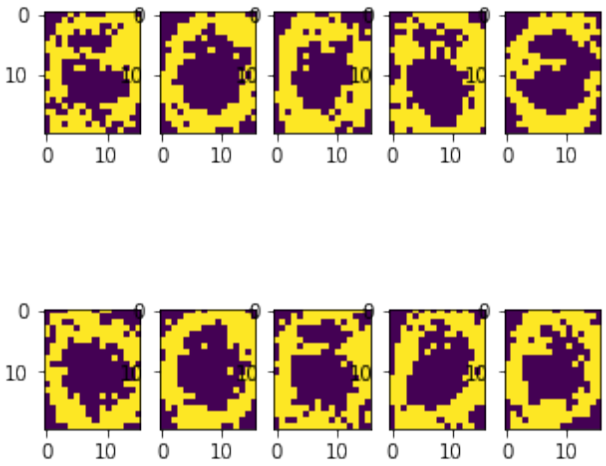

Figure 1: Données regénérées par notre rbm

sur la figure 1 le rbm génère facilement les 0 difficilement les 8 et jamais les 5 on va donc tenter la même opération avec le dbn.

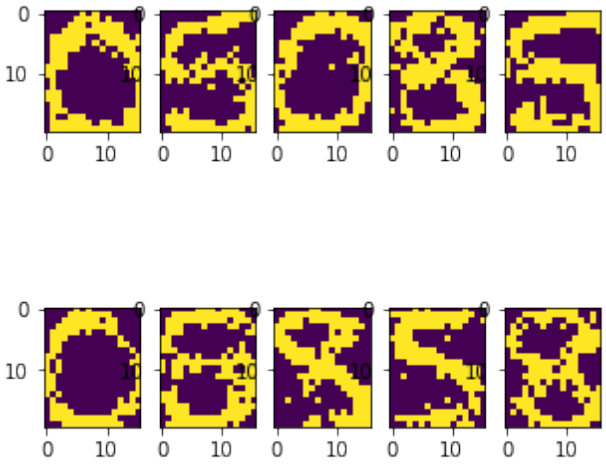

Figure 2: Données regénérées par notre dbn

Sur la figure 2 on voit clairement l'apport des couches cachées le dbn est capable de reconstituer facilement les 0,5,8.

4 Etude à réaliser

4.1 Etude de l'impact du nombre de couches :

Dans cette première partie de l'étude nous fixons tous les paramètres, sauf le nombre de couches que nous faisons varier, et comparons le comportement des dnn en fonction du nombre de couches choisies et du préentrainement ou non. le nombre de neurones par couches est de 200.

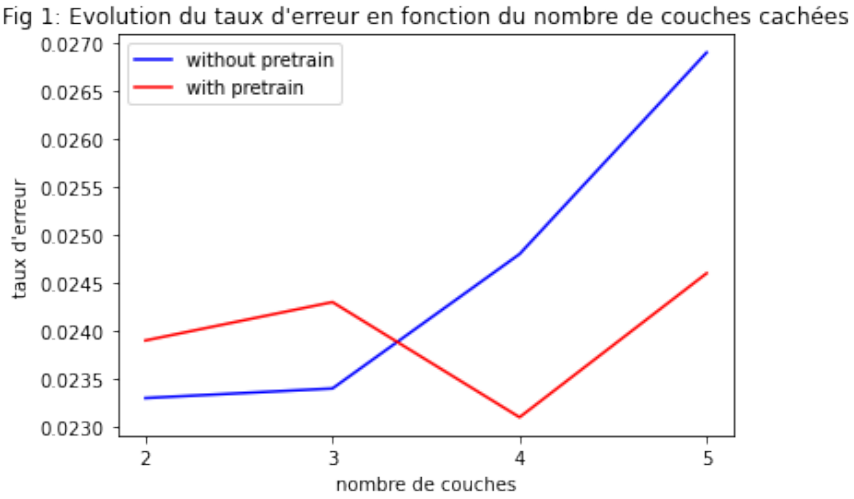

Figure 3: Évolution du taux d'erreur en fonction du nombre de couches

La figure ci dessus met en évidence que le pré-training est contre productif pour des réseaux peu profonds mais plus performant pour des réseaux profonds, notons ici que ce qui nous interesse est la différence de performances entre le réseau pré entrainé et celui non pré entrainé cad que l'on ne peut en rien interpréter la perte de performance lors de l'ajout de la 5ième couche cachée par exemple. Dans l'idéal il aurait fallu effectuer plusieurs simulations pour contourner ce problème... On déduit de cette première partie de l'étude que le réseau préentrainé permet d'obtenir de meilleurs résultats sur les réseaux plus profonds.

4.2 Étude de l'impact du nombre de neurones

Dans cette deuxieme partie de l'étude nous fixons tous les paramètres, sauf le nombre de neurones par couches que nous faisons varier, et comparons le comportement des dnn en fonction du nombre du nombre de neurones choisi et du préentrainement ou non. la profondeur du réseau est de 2 couches.

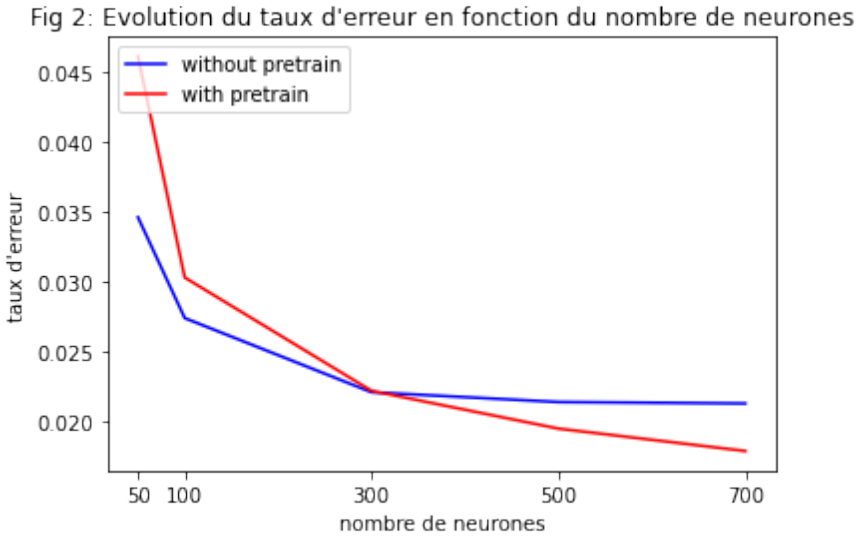

Figure 4: Evolution du taux d'erreur en fonction du nombre de neurones par couche

Cette partie nous montre que le réseau préentrainé est moins performant pour un réseau moins dense mais obtient de meilleures performances pour un réseau plus dense. On obtient d'ailleurs de très bons résultats avec la dernière configuration avec moins de 1.9% d'erreur.

4.3 Étude de l'impact du nombre de données utilisées

Dans cette troisième partie de l'étude nous fixons tous les paramètres, sauf le nombre de donnée utilisées pour l'entrainement que nous faisons varier, et comparons le comportement des dnn en fonction du nombre du nombre de données choisi et du préentrainement ou non.

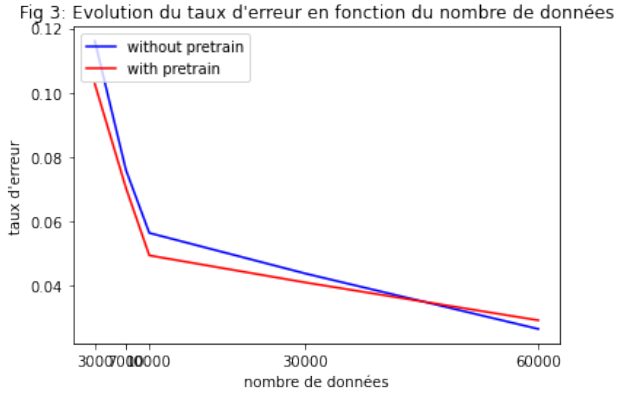

Figure 5: Évolution du taux d'erreur en fonction du nombre de données

Ici comme on est sur un réseau peu profond et peu dense on s'attendrait à avoir de meilleures performances sur le dnn non préentrainé toutefois on remarque que lorsque l'on dispose de moins de données le dnn préentrainé donne de meilleurs résultats le préentrainement permettrait donc d'avoir de meilleurs résultats lorsque l'on manque de données.

4.4 Meilleure configuration

On utilise un réseau pré entrainé à 4 couches cachées de 700 unités pour lequel on obtient un score de 1.7 % d'erreur

5 Conclusion

On conclut que le préentrainement permet de d'entrainer efficacement des réseaux plus profonds et plus denses. Toutefois les réseaux peu profonds et peu denses préentrainés performent moins bien que les réseaux non préentrainés, cela peut s'expliquer par le fait qu'un petit nombre de neurones ou de couche ne permet pas de stocker assez d'information sur la structure des données et donc il se peut que l'information stockée ne soit pas l'information significative pour la classification. Dans ces cas là le pré entraînement est contre productif et induit le réseau en erreur. Au contraire lorsque le réseau est profond et dense plus d'information peut être stockée grâce au préentrainement dont l'information significative pour la classification. Bien évidemment pour plus de rigueur toutes les expériences menées ici devraient être reproduites plusieurs fois ce que je n'ai pas pu faire car les calculs prennent beaucoup de temps.