Deep Learning II
Rendu de projet
TP DNN

Eric PATARIN*, Lucas SELINI†
M2 Data Science

March 31, 2024

1 Introduction

Ce document détaille nos résultats expérimentaux obtenu à l'aide de modèles génératifs sur deux jeux de données : Binary AlphaDigit et MNIST.

*eric.patarin@ensta-paris.fr
†lucas.selini@ensta-paris.fr

2 Etude sur Binary AlphaDigit

Dans cette partie, nous allons analyser la qualité des images générées par notre RBM et par notre DBN, en fonction des différents hyperparamètres. Nous allons commenter les résultats obtenus en générant des images. Notre appréciation servira de label ; puisqu'il s'agit ici d'un entraînement non supervisé.

2.1 RBM

Les performances de l'algorithme du modèle RBM sont fortement influencées par les différents hyperparamètres :

- le nombre d'unités cachées q.
- le nombre d'itérations lors de l'apprentissage.
- le learning rate.
- le nombre de caractères différents dans la base de données

Nous travaillons sur une seule lettre avec les paramètres suivants (dans principal_RBM_alpha) :

learning rate	batch size	nb iter
0.01	5	100

2.1.1 Influence de la learning rate

Nous utilisons des learning rates à valeurs dans [0.0001,0.001,0.01,0.05,0.1,0.3,0.8]. Les images sont dans l'ordre croissant.

La qualité du résultat est logiquement en cloche, avec un pic de qualité pour un learning rate d'environ 0.1.

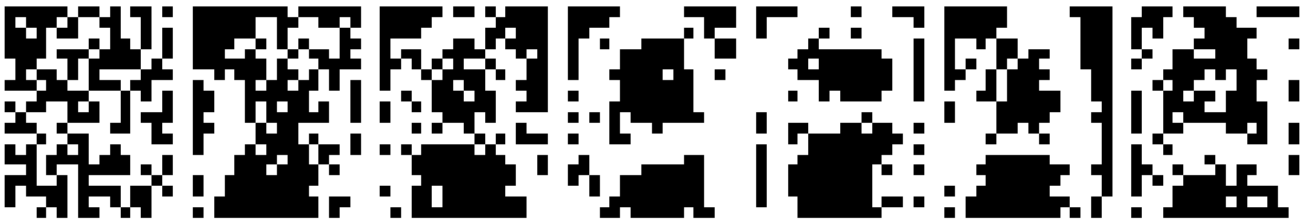

Figure 1: Output généré par la RBM pour différentes learning rates

2.1.2 Influence du nombre d'unitées cachées q

Nous générons des images pour q dans [1,5,10,20,40,50,75,100,200].

Le résultat obtenu est cohérent. La qualité du résultat est croissante avec q puis stagne à partir d'un certain seuil.

Figure 2: Output généré par la RBM pour différentes valeurs de q

2.1.3 Influence du nombre de la taille de l'alphabet en entrée

Plus on augmente la quantité de lettres en entrée de la RBM plus la qualité de la génération se dégrade. En voici un exemple obtenu avec les paramètres suivants :

learning rate	batch size	nb iter	q
0.01	5	200	130

Figure 3: Les dernières générations ne correspondent à l'alphabet appris.

Cette figure est reproductible en exécutant principal_RBM_alpha_data_loop.py avec des alphabets prenant leurs valeurs dans [[10], [10,11], [10,11,12], [10,11,12,13], [10 à 17], [10 à 35]].

2.2 DBN

Comme le RBM, les performances de l'algorithme du modèle DBN sont fortement influencées par les différents hyperparamètres :

- le nombre d'unités cachées q.
- le nombre d'itérations lors de l'apprentissage.
- la learning rate.
- le nombre de caractères différents dans la base de données
- le nombre de couches cachées

2.2.1 Influence du nombre d'unitées cachées q.

Nous travaillons sur une seule lettre avec les paramètres suivants (dans principal_DBN_alpha) :

learning rate	batch size	nb iter
0.0075	5	5

Nous utilisons des q à valeurs dans la liste [1,5,10,20,40,50,75,100,200]. Nous obtenons le résultat suivant :

Figure 4: Output généré pour un DBN([p,q,q] à q variable.

Il est assez surprenant de constater que l'image obtenue même pour des valeurs de q très faibles est bonne, et se dégrade en augementant la valeur de q.

2.2.2 Influence du nombre de couches cachées

Nous présentons les résultats obtenus par le DBN pour des nombres de couches cachées différents. La liste utilisée est la suivante : [[p,q],[p,q,q],[p,q,q,q],[p,q,q,q,q],[p,q,q,q,q,q,p],[p,q,q,q,q,q,q,q,q,q,q,q]]. Et ce pour $q = 50$ et $p = 320$.

Figure 5: Output généré par le DBN pour différentes configurations de couches cachées.

2.2.3 Influence de la learning rate

Voici les images générées par le DBN pour des learning rates valant respectivement 0.0001, 0.001, 0.01, 0.05, 0.1, 0.3, 0.8.

Figure 6: Output généré par le DBN pour différentes learning rates.

On constate qu'il n'existe pas de relation strictement monotone entre qualité de l'output et learning rate. L'équilibre se situe, tous paramètres égaux par ailleurs, au milieu des valeurs considérées.

2.2.4 Influence du nombre de la taille de l'alphabet en entrée

Voici les images générées par le DBN avec des alphabets prenant leurs valeurs dans [[10], [10,11], [10,11,12], [10,11,12,13], [10 à 17], [10 à 35]] (avec un q = 10)

Figure 7: Output généré par le DBN différents alphabets.

Nous constatons que le DBN fourni des résultats de bonnes qualités pour plusieurs lettres dans son alphabet (plus que le RBM), mais au bout d'un certain nombre de lettres il ne parvient plus à fournir des résultats satisfaisants.

3 Etude sur MNIST : DNN

Nous allons maintenant nous focaliser sur la base de données MNIST afin d'évaluer notre réseau de neurones profond (DNN).

Afin d'évaluer l'avantage du pré apprentissage non supervisé sur les capacités de classification de notre DNN, nous allons comparer deux réseaux : un premier que nous allons pré entrainer en le considérant comme un empilement de RBM (apprentissage non supervisé sur la base d'entraînement) et un deuxième que nous initialisons aléatoirement. Nous allons ensuite appliquer la rétropropagation aux deux réseaux sur la base d'entraînement (apprentissage supervisé) et comparer leur taux de de mauvaises classifications sur les données d'entraînement et sur des données de test.

3.1 Informations sur l'analyse

Nous avons réalisé trois analyses qui mènent à trois figures :

- Figure 1 : 4 courbes exprimant le taux d'erreur sur les données train et sur les données test des 2 réseaux en fonction du nombre de couches cachées (de 0 à 5)
- Figure 2 : 4 courbes exprimant le taux d'erreur sur les données train et sur les données test des 2 réseaux en fonction du nombre de neurones dans les couches cachées (100, 300, 500, 700 et 900)
- Figure 3 : 4 courbes exprimant le taux d'erreur sur les données train et sur les données test des 2 réseaux en fonction du nombre de données train (100, 250, 500, 750, 1000, 3000, 7000, 10000, 15000, 20000, 25000, 30000, 40000, 45000, 50000, 55000 et 60000)

Dans chacune des analyses, nous avons utilisé comme hyperparamètres :

- learning rate = 0.075
- nombre de couches cachées (hors figure 1) = 2
- nombre de neurones dans les couches cachées (hors girugre 2) = 200
- nombre de données d'entrainement (hors figure 3) = 48 000 (car nous avons constaté un fort overfitting si nous prenons tout le dataset d'entraînement)
- nombre d'itérations de rétropropagations = 50 (overfitting trop important sur 200)
- batch size = 125
- nombre d'iterations de descentes de gradients pour les RBM = 100

3.2 Code

Nos codes se décomposent comme ceci :

- Dans le dossier utils, un fichier dnn.py qui contient la class DNN et qui définit les méthodes d'initialisation, pré entraînement, rétropropagation et obtention du taux de mauvaises classifications
- le fichier principal_DNN_MNIST.py qui permet d'exécuter simplement le code sur un réseau que l'on pré entraîne et avec les hyperparamètres qui meènnt à la meilleure classification
- les fichiers principal_DNN_MNIST_Fig1.py, principal_DNN_MNIST_Fig2.py et principal_DNN_MNIST_Fig3.py qui permettent d'exécuter le code qui fournit les différentes figures (loop sur les différents hyperparamètres et plot)

3.3 Figures et analyse

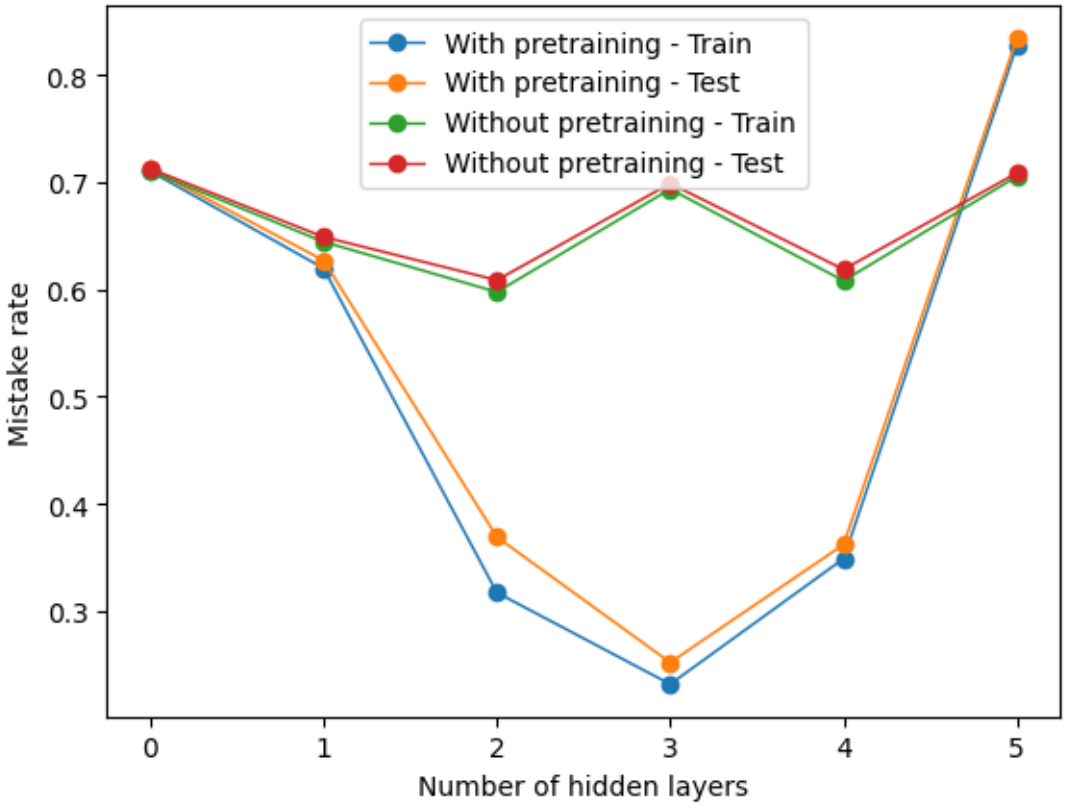

Figure 8: Analyse de l'impact du nombre de couches cachées

Les résultats sont toujours meilleurs avec pré entraînement que sans. Ceci nous confirme la grande efficacité du processus de Pre-Training.

Sur la figure 1, nous constatons une forte importance du nombre de couche cachées : entre 2 et 4, les résultats sont bien améliorés avec le pré-entraînement, mais moins de 2 ou plus de 4 couches cachées et les résultats avec pré-entraînement deviennent similaire à ceux sans pré-entraînement.

Sur la figure 2, nous constatons une assez faible dépendance de la précision de notre modèle au nombre de neurones que nous mettons dans les couches cachées.

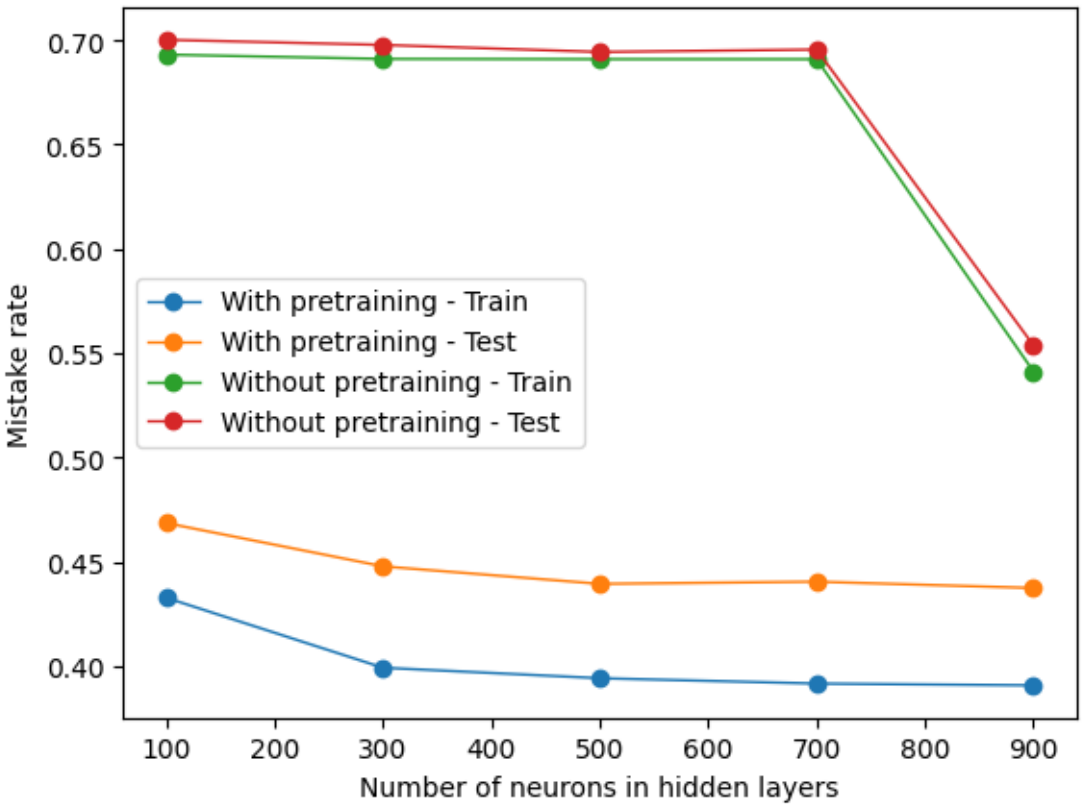

Figure 9: Analyse de l'impact du nombre de neurones dans les couches cachées

Sur la figure 3, nous constatons une très grande importance au nombre de données d'entraînement que nous donnons à notre modèle. En effet, le modèle est très mauvais avec tout le dataset d'entraînement (60 000 données), ce qui nous fait penser qu'il y a un fort overfitting dans ce cas là. De manière plus surepreprenante, nous n'avons pas une courbe convexe comme nous pourrions nous y attendre : nous avons de meilleurs résultats avec 10 000 données, tandis qu'avec 7 000 ou 15 000 ceux-ci sont moins bons. Ceci est surement dû à une assez grande volatilité des résultats vis à vis de la répartition des lettres et de la "qualité" du dataset d'entraînement qui peut varier d'un batch à un autre.

3.4 Meilleure configuration

A partir de ces figures, nous avons trouvé que la configuration qui permet d'obtenir le meilleur taux de classification possible est avec 3 couches cachées de 300 neurones, entraîné sur 30 000 données d'entraînement et avec 60 epochs de rétropropagation (overfitting trop important sur 200 epochs).

Nous obtenons pour cette configuration un taux de mauvaise classification de 23% sur le set d'entraînement et de 24.5% sur le set de test.

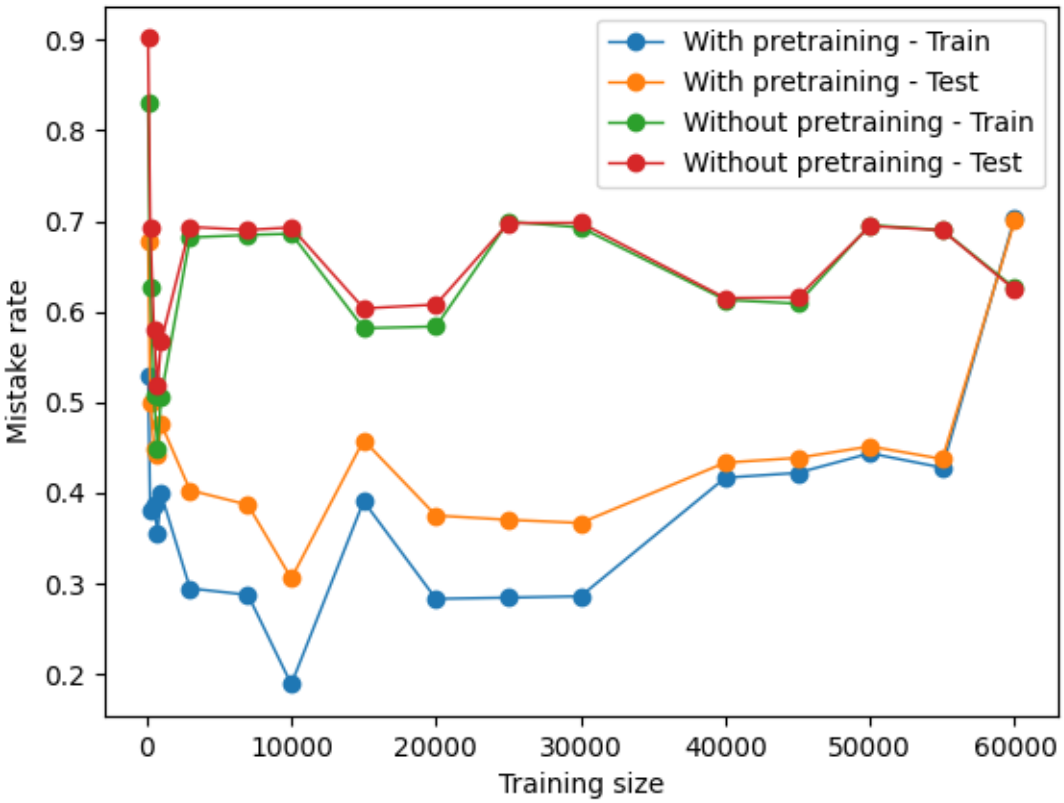

Figure 10: Analyse de l'impact du nombre de données d'entraînement

3.5 Conclusion

Au fil de nos analyses, nous constatons qu'il est difficile de trouver les meilleurs hyperparamètres. L'overfitting semble survenir très rapidement et en fonction d'un hyperparamètre, survenir pour un certain autre hyperparamètre différent à chaque fois.

De plus, l'entraînement d'un réseau prenant beaucoup de temps (parfois plusieurs heures), il nous est impossible d'essayer toutes les configurations possibles.