Projet Deep Learning II

Martin BORDES

Mars 2024

1 Étude sur Alpha Binary Digits

La première étape de ce projet était de construire un RBM puis un DNN capables de générer des données. Dans cette partie, nous allons vérifier que nos modèles sont effectivement capables de générer des données cohérentes et étudier les sorties de ces réseaux en fonction du nombre de couches, du nombre de neurones et du nombre de caractères à apprendre. Pour cela, nous utiliserons la base Binary AlphaDigits, ne contenant que 39 images par caractère et 36 caractères (10 chiffres et 26 lettres). Pour générer les images tout au long de cette partie, le learning rate est de 0.01, une taille de batch de 10, 1000 epochs pour l'entraînement et 200 itérations de Gibbs sampling. La première couche de chaque réseau doit contenir 320 neurones car les images de la base sont au format 20x16.

1.1 Variation du nombre de couches

Tout d'abord, nous pouvons étudier l'impact du nombre de couches sur sur la génération d'images tout en gardant un nombre total de neurones constant (640 ici pour tous les réseaux). On étudie ainsi l'arbitrage entre ajouter plus de couches et plus de neurones par couche. Pour générer les images de la figure 1, les réseaux sont entraînés sur les chiffres 4 à 9.

(a) Réseau 320-320 (b) Réseau 320-170-100-50 (c) Réseau 320-120-80-60-40-20

Figure 1: Images générées avec différents nombres de couches

On peut voir que le DBN à une seule couche (en réalité un RBM) génère des chiifres assez propres mais semble ne générer presque que des 8. Il s'est avéré que les réseux courts avaient tendance à sur-représenter les chiffres arrondis. Ensuite, lorsqu'on augmente le nombre de couches, le réseau arrive à générer des chiffres plus variés, d'abord de façon assez confuse puis de meilleur qualité avec un nombre de couches encore plus élevée.

1.2 Variation du nombre de neurones

Ensuite, on peut regarder l'impact du nombre de neurones par couche. Pour cela, on s'intéressera à la fois au RBM et au DBN. Encore une fois, les différents réseaux sont entraînés sur les chiffres 4 à 9.

1.2.1 RBM

(a) RBM à 50 neurones latents (b) RBM à 150 neurones latents (c) RBM à 300 neurones latents

Figure 2: Images générées par un RBM avec différents nombres de neurones

Pour le RBM, la figure 2 nous montre que l'augmentation du nombre de neurones permet une meilleure qualité de génération, ce qui est logique puisqu'on augmente la capacité du réseau à apprendre. On retrouve cependant la tendance du RBM à ne pas généraliser à tous les chiffres et à sur-représenter les caractères ronds.

1.2.2 DBN

(a) Réseau 320-50-30-10 (b) Réseau 320-150-100-50 (c) Réseau 320-320-320-320

Figure 3: Images générées par un DBN avec différents nombres de neurones

Pour le DBN, on retrouve bien une meilleure généralisation que pour le RBM (même si elle n'est pas encore parfaite) et également une restitution plus cohérente avec un nombre de neurones plus élevé, comme on peut le voir sur la figure 3

1.3 Variation du nombre de caractères à apprendre

Nous pouvons maintenant nous intéresser à l'impact du nombre de caractère de la base d'entraînement sur la génération d'images. Ici, nous n'utiliserons qu'un seul type de structure, un DBN 320-300-250-200-150-100. Comme nous le verrons dans la seconde sous-partie, ajouter le caractère 1 à la base d'entraînement a un impact énorme, c'est pour cela que nous commencerons par ne jamais l'inclure dans la première sous-partie.

1.3.1 Sans inclure le 1

Ici, nous étudions la performance du même modèle entraîné sur 1, 5, 9 et 35 caractères différents. La figure 4 nous montre les résultats. On observe que le réseau n'a aucun mal avec un seul caractère puis cela devient de plus en plus compliqué jusqu'à ne plus arriver à générer une seule images cohérentes avec 35 caractères. Notre DBN est donc fortement

(a) Chiffre 9

(b) Chiffres 5 à 9

(c) Chiffres 0 et 2 à 9

(d) 36 caractères d'AlphaDigit sauf le 1

Figure 4: Images générées par un DBN avec différents nombres de caractères à apprendre

limité par le nombre de caractères à apprendre et aurait sûrement besoin de plus d'exemples de chaque caractère pour mieux performer.

1.3.2 En incluant le 1

Il y a un chiffre qui dérègle totalement l'apprentissage du réseau lorsqu'on l'ajoute aux données d'entraînement le 1. Nous pouvons en effet voir sur la figure 5 que les sorties sont beaucoup moins cohérentes que lorsque le chiffre 1 était absent.

Figure 5: Images générées par un DBN apprenant les chiffres 1 et 5 à 9

Afin de comprendre cela, nous pouvons regarder à quoi ressemblent les exemples du chiffre 1 dans la base de données. La figure 6 nous en montre 16 (sur 39) et l'on peut observer qu'ils sont assez incohérents, prenant parfois une très grand epartie de l'image, ce qui ne peut pas mener à une génération qualitative.

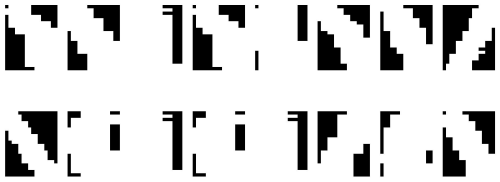


Figure 6: Exemples de 1 dans la base d'entraînement

2 Étude sur MNIST

Après avoir vérifié que notre implémentation du DBN est bien capable de générer des images cohérentes sur la base Binary AlphaDigits, nous allons maintenant utiliser la base MNIST afin de construire un classifieur. Un DBN peut être pré-entraîner et servir d'initialisation à un DNN (un réseau de neurones classique) afin d'accélérer sa convergence. Nous allons donc comparer les performances de classifieurs initialisés aléatoirement et de classifieurs pré-entraînés à l'aide d'un DBN. Pour tous les entraînements que nous effectuerons dans cette partie, nous utiliserons un learning rate de 0.001, une taille de batch de 10, 5 epochs pour la phase de pré-entraînement du DBN et 10 epochs pour la phase de rétropropagation du gradient. La première couche de chaque réseau doit contenir 784 neurones car les images sont au format 28x28. La dernière couche, de classification, doit elle contenir 10 neurones car il y a 10 classes possibles, les chiffres.

2.1 Exemples de sorties

Tout d'abord, nous pouvons vérifier que notre implémentation du DNN fonctionne correctement. Ici, nous utilisons un réseau assez simple 784-200-200-10. On commence par effectuer la phase rétropropagation sur un réseau dont les poids ont été initialisés aléatoirement. Le taux d'erreur est de 3.05% sur la base d'entraînement et de 3.86% sur la base de test, ce qui est déjà assez satisfaisant. Les figures 7 et 8 montrent des exemples des images de test associées à leurs probabilités de sortie, la deuxième figure ne montrant que des images mal classifiés.

On effectue alors la même rétropropagation, mais pour un réseau pré-entraîné. Le taux d'erreur est de 1.77% sur la base d'entraînement et de 2.43% sur la base de test. Le pré-entraînement a bien permis d'augmenter la performance du modèle. En réalité, il permet de converger beaucoup plus rapidement, il a besoin de moins d'epochs pour arriver à la même performance sur la base d'entraînement, ce qui permet d'obtenir une performance maximale sur le test meilleure afin d'atteindre le point de sur-apprentissage. Les figures 9 et 10 montrent des sorties équivalentes à ce qu'on a vu pour le réseau précédent. On peut notamment observer sur les deux réseaux que les exemples mal classifiés sont généralement particuliers, certains étant même difficiles à classifier pour l'homme.

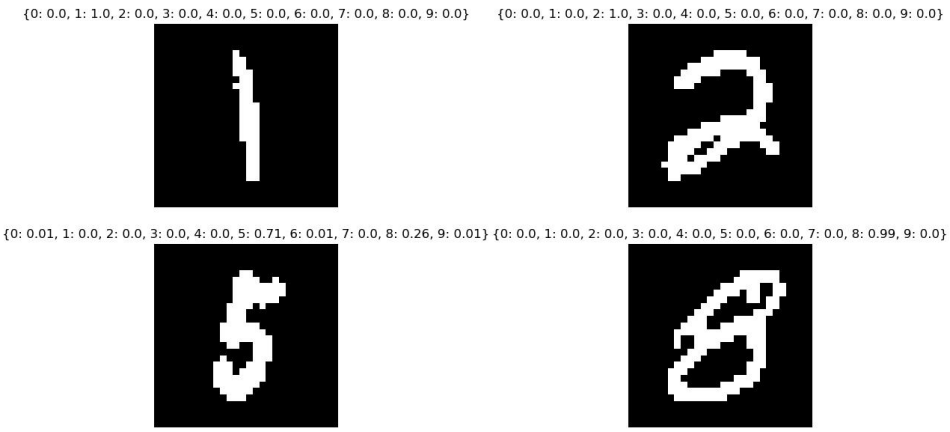


Figure 7: Exemples de sorties d'un DBN initialisé aléatoirement

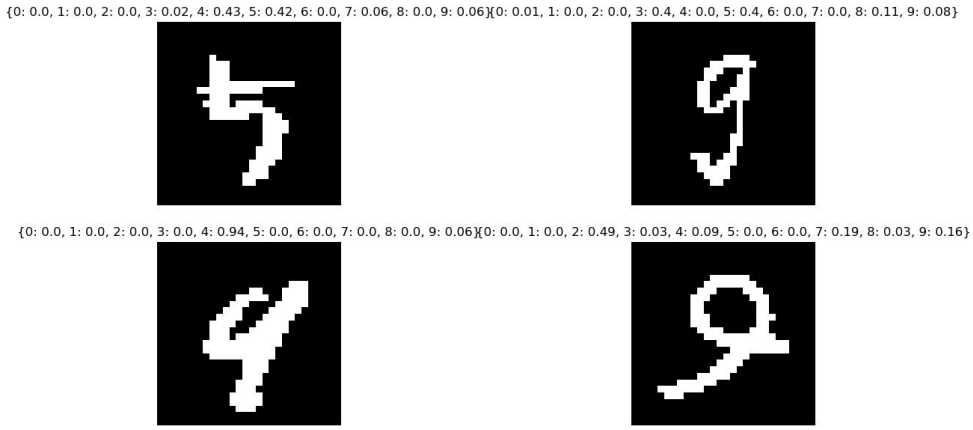


Figure 8: Exemples de sorties mal classifiées d'un DBN initialisé aléatoirement

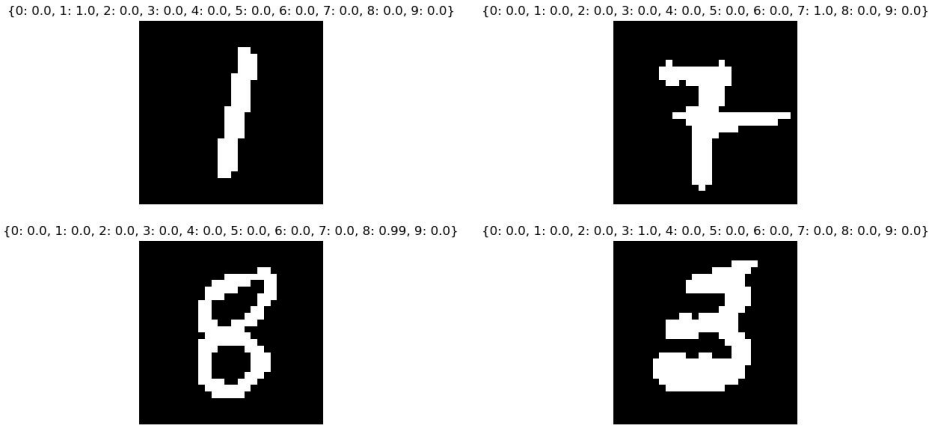


Figure 9: Exemples de sorties d'un DBN initialisé pré-entraîné

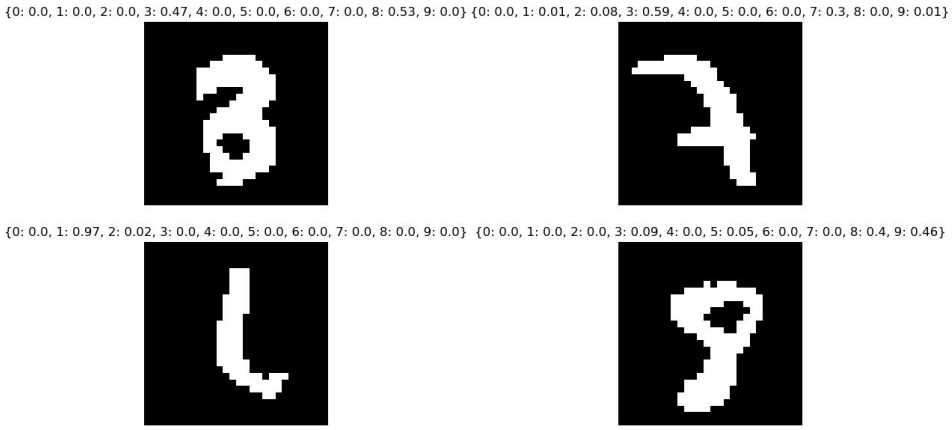
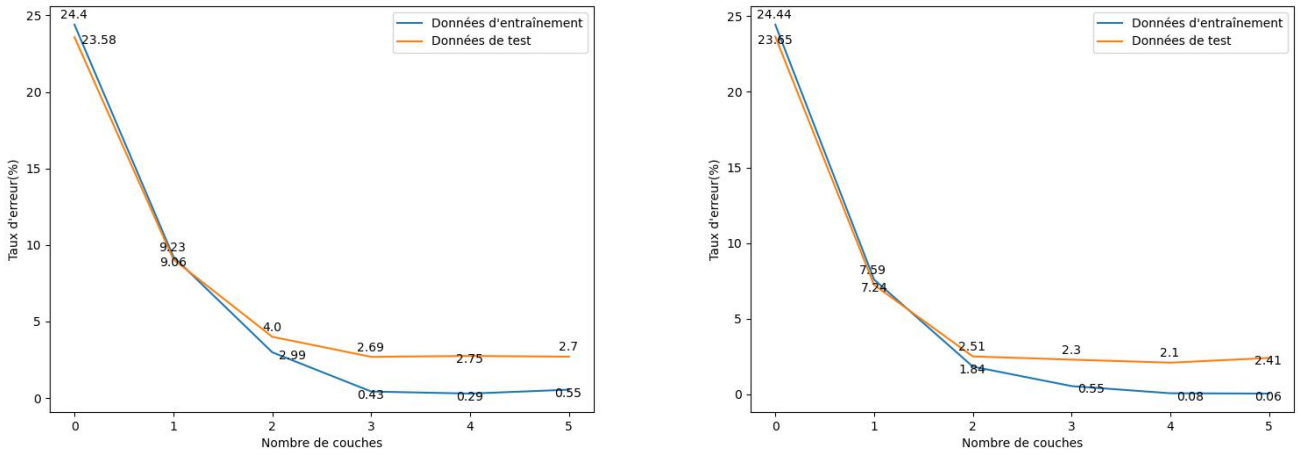


Figure 10: Exemples de sorties mal classifiées d'un DBN pré-entraîné

2.2 Variation du nombre de couches

Nous pouvons maintenant nous intéresser à l'impact de différentes variables d'ajustements sur la performance de nos deux types de modèles. Le nombre de couches en est une première. Ici, nous ne compterons pas dans le terme nombre de couches les premières et dernières couches. Chaque couche rajouté contiendra 200 neurones chacune. Ainsi, un réseau à 3 couches aura pour structure 784-200-200-200-10. Nous comparerons des réseaux allant de 0 à 5 couches. La figure 11 représente l'évolution de la performance pour des réseaux initialisés aléatoirement ou pré-entraînés. Pour les deux réseaux, la performance augmente avec le nombre de couches jusqu'à 3 ou 4 couches. Le réseau pré-entraîné est meilleur pour chaque nombre de couches, sauf pour 0 qui ne consiste en réalité qu'à une simple couche de classification sans DBN, le pré-entraînement étant donc impossible.



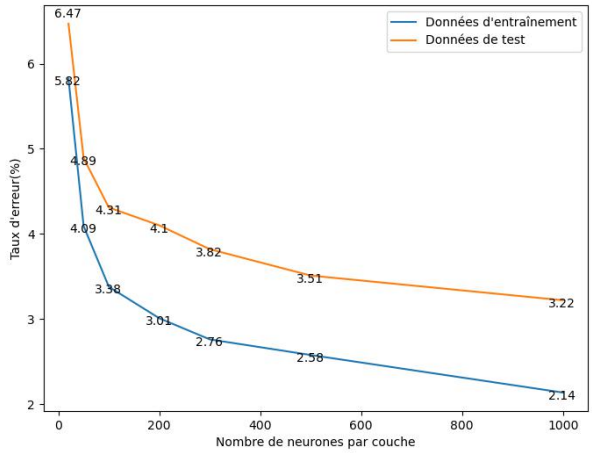
(a) Réseau initialisé aléatoirement

(b) Réseau pré-entraîné

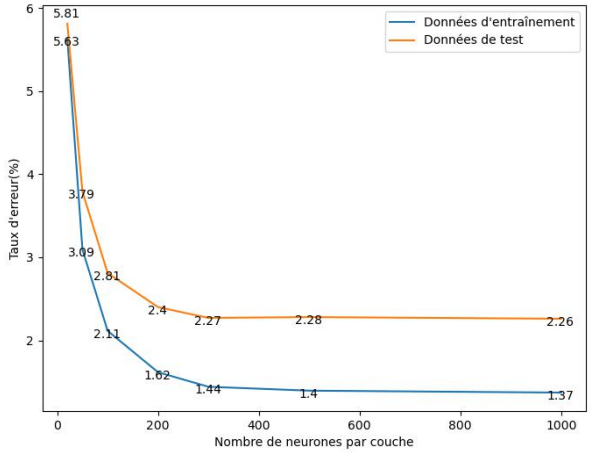
Figure 11: Évolution des taux d'erreur sur les bases d'entraînement et de test en fonction du nombre de couches pour un DBN pré-entraîné ou non

2.3 Variation du nombre de neurones

Ensuite, nous pouvons regarder l'impact du nombre de neurones par couche. Ici, tous les réseaux considérés contiendront 2 couches cachés. Un réseau 200 neurones aura alors pour structure 784-200-200-10. La figure 12 nous montre des choses très intéressantes. Non seulement le modèle pré-entraîné performe mieux pour chaque nombre de neurones, mais il atteint également son plateau beaucoup plus rapidement. Il semble l'avoir atteint à partir de 300 neurones par couche alors que le réseau initialisé aléatoirement ne l'a toujours pas atteint au bout de 1000 neurones. Pré-entraîner le réseau permet donc d'avoir besoin d'une structure plus légère et ainsi plus rapide à entraîner tout en étant moins sujette au sur-apprentissage.



(a) Réseau initialisé aléatoirement

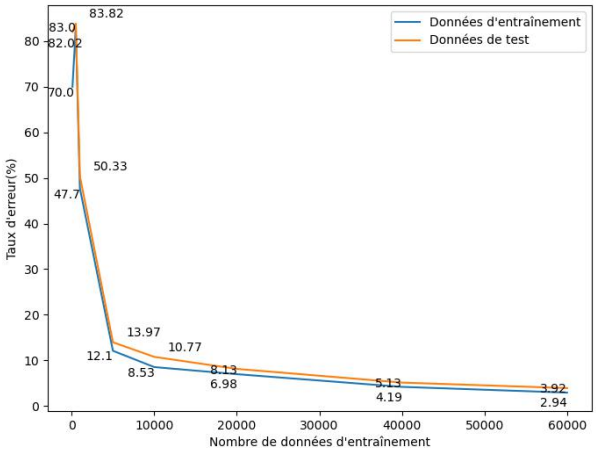


(b) Réseau pré-entraîné

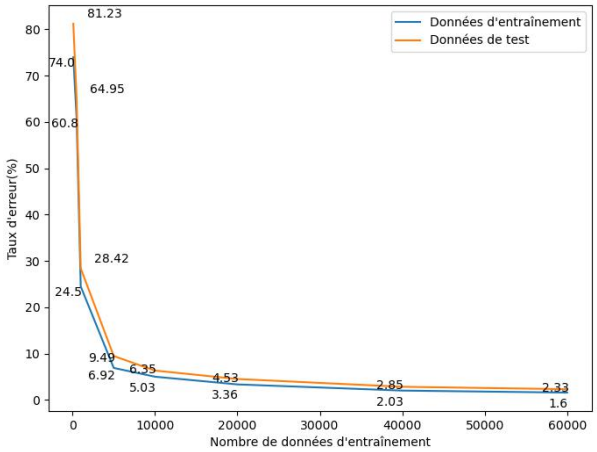
Figure 12: Évolution des taux d'erreur sur les bases d'entraînement et de test en fonction du nombre de neurones par couche pour un DBN pré-entraîné ou non

2.4 Variation du nombre de données d'entraînement

Enfin, nous pouvons étudier l'importance du nombre de données d'entraînement. Ici, tous les réseaux sont de la forme 784-200-200-10. La figure 13 nous donne également des informations intéressantes. Encore une fois, le réseau pré-entraîné performe mieux pour tous les nombres de données d'entraînement. Il semble également atteindre son plateau (même si la performance ne cesse d'augmenter avec le nombre de données) plus rapidement une nouvelle fois. Un réseau pré-entraîner pourrait donc nécessiter moins de données, une caractéristiques très importante dans de nombreux domaines où les données d'entraînement ne sont pas faciles à obtenir en grande quantité, comme par exemple la santé.



(a) Réseau initialisé aléatoirement



(b) Réseau pré-entraîné

Figure 13: Évolution des taux d'erreur sur les bases d'entraînement et de test en fonction du nombre de données d'entraînement pour un DBN pré-entraîné ou non

2.5 Le meilleur résultat possible

Afin d'obtenir le meilleur taux de classification possible, nous allons utiliser toutes les informations que nous avons pu obtenir lors de notre analyse. Nous avons vu qu'utiliser un réseau pré-entraîner avait de nombreux bénéfices, nous allons donc évidemment utiliser un tel modèle. Nous avons vu que le meilleur résultat était obtenu pour 4 couches. De plus, la performance atteignait un plateau à partir 400 neurones par couche environ. Enfin, il est toujours utile d'utiliser le plus de données d'entraînement possible ici. Le réseau utilisé aura lors une structure 784-400-400-400-400-10 et sera entraîné sur toute la base d'entraînement. Les hyperparamètres ayant mené au meilleur taux de classification sont les suivants: un learning rate de 0.001, une taille de batch de 5, 5 epochs de pré-entraînement et 8 epochs de rétropropagation. Finalement, le taux d'erreur est de 0.00% sur la base d'entraînement et de 1.65% sur la base de test, soit un taux de classification de 98.35%

3 Code

Le code rendu contient 4 scripts et 2 notebooks. Le script *utils.py* contient une classe et une fonction qui permettent de lire les données. Les scripts *principal_RBM_alpha.py*, *principal_DBN_alpha.py* et *principal_DNN_MNIST.py* contiennent respectivement les classes RBM, DBN et DNN comme définies dans les consignes.

Ensuite, les notebooks *etude_binary.ipynb* et *etude_mnist.ipynb* contiennent le code qui a mené aux différents graphiques qui ont mené à l'analyse dans le rapport, respectivement sur AlphaDigit et sur MNIST.

Comme demandé les données ne sont pas incluses. Afin que les notebooks tournent sans avoir à modifier les chemins d'accès, il est nécessaire d'avoir le fichier *binaryalphadigs.mat* (récupéré ici : AlphaDigits) au même niveau que les notebooks et les 4 fichiers contenant les données MNIST (récupéré ici : MNIST) dans un dossier *MNIST*.