Restricted Boltzmann Machine appliqué à la génération d'images

Emilien VIMONT

27 Février 2022

1 Introduction

On implémente un RBM dans le but d'apprendre et de générer des chiffres et des lettres écrits à la main. On étudiera alors l'efficacité du RBM pour la génération de contenu pour la base de données AlphaDigits et mettrons en évidence des paramètres 'optimales' pour cette base de données. Le RBM sera ensuite appliqué à la base de données MNIST. Finalement, nous comparerons les résultats obtenus pour la génération de chiffres et de lettres à ceux d'un GAN.

2 Données

2.1 AlphaDigits

On travaille premièrement sur la base de données AlphaDigits, composée de chiffres et de lettres manuscrits. La base de données est composée d'images de taille 20x16 représentant les chiffres de 0 à 9 et les lettres de A à Z en binaire. Cette base de données contient 39 échantillons pour chaque classe, on a donc 1404 données. Voir Figure 1

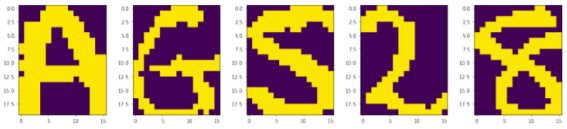


Figure 1: Exemples de chiffres et de lettres de la base Alphadigits.

2.2 MNIST

La base de données MNIST, une extension de la base de données NIST, est une collection de données peu complexes de chiffres manuscrits utilisée pour entraîner et tester divers algorithmes d'apprentissage automatique supervisés. La base de données contient 70 000 images de taille 28x28 en noir et blanc représentant les chiffres de zéro à neuf. Les données sont divisées en deux sous-ensembles, 60 000 images appartenant à l'ensemble d'apprentissage et 10 000 images appartenant à l'ensemble de test. On transformera les données en données binaires pour le TP. Voir Figure 4.

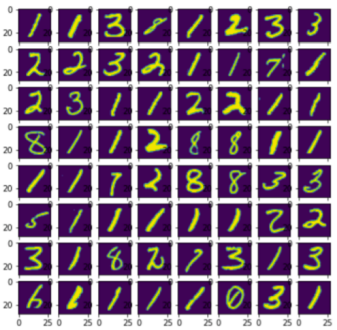
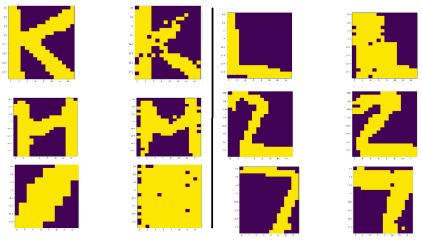


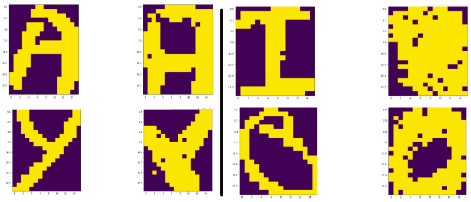
Figure 2: Exemples de chiffres de la base MNIST.

3 RBM appliqué à AlphaDigits

On entraîne le RBM (Restricted Boltzmann Machine) sur chaque caractère de la base de données AlphaDigits. On génère ensuite des caractères similaires à ceux appris dans la base de données. La Figure 3a fournit une comparaison entre les images réelles de la base de données et les images générées par le RBM avec une nombre d'unités de 100, pour 100 epochs, un learning rate de 0.1 et 10 itérations de l'échantillonneur de Gibbs. On ajoute également une fonction mini-batch permettant de travailler avec des batch dont la taille est fixée à 6 pour les figures obtenues. Les résultats affichés sont valables pour des caractères spécifiquement choisis (voir Figure 3a et 3b).



(a) Comparaison entre les caractères K,L,M,2,1 et 7 de la base de données AlphaDigits et ces mêmes caractères générés par le RBM.



(b) Comparaison entre les caractères A,I,Y et 0 de la base de données AlphaDigits et ces mêmes caractères générés par le RBM.

Figure 3: Comparaison entre les caractères de la base de données AlphaDigits et les caractères générés par le RBM.

La fonction de perte quadratique atteint 0.02 pour le centième epoch du RBM. Les images sont globalement bien reconstituées. La figure 3 illustre différents cas rencontrées. Les lettres et chiffres de formes assez simples et arrondies telles que le O, le Q ou le P par exemple sont souvent très bien retranscrits bien que le rendu puisse paraître assez bruité. Le bruit est également présent mais tout aussi limité pour l'ensemble des chiffres de 0 à 9 à l'exception du chiffre 1. Ce bruit s'explique par les différences d'écriture que l'on peut trouver au sein de la base de données pour un même caractère. On constate alors (pour le cas de la Figure 3) que les chiffres 7 ou 2 sont globalement écrits de la même façon, ce qui coïncide avec les données d'entraînement. La lettre L de son côté est sujette à plus de variabilité quant à son écriture comme en témoigne la Figure 3, le schéma global est tout de même discernable. De la même façon, les lettres plus complexes comme le K ou le Y sont également bien représentées.

Le chiffre 1 et la lettre I ne sont pas très bien retranscrits. Cela s'explique par les différents styles d'écriture qu'on retrouve pour ces deux caractères au sein de la base de données. Par exemple, la Figure 4 montre que le chiffre 1 est représenté de deux façons très différentes dans le jeu de données d'entraînement. En effet, le chiffre 1 est écrit dans les deux cas en diagonale mais celle-ci ne va pas dans le même sens. Parfois, le 1 peut également prendre quasiment l'intégralité des pixels. Le RBM

génère alors un caractère maximisant la similarité avec les données d'entraînement (en terme de pixels). Cette représentation biaise ainsi l'algorithme. Pour correspondre aux deux manières d'écrire le chiffre, le RBM renvoie un chiffre qui prend en compte les deux manières d'écrire le chiffre 1, ce dernier prend alors la forme d'une tâche.

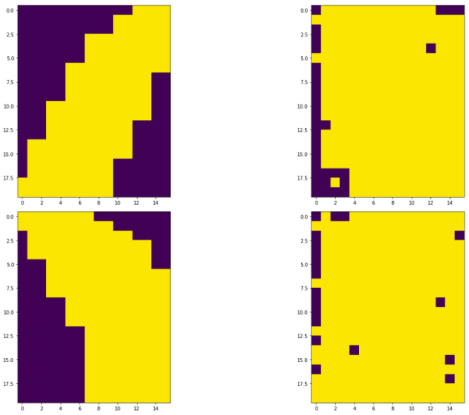


Figure 4: Différentes façons d'écrire le chiffre 1.

3.1 Etude selon le learning rate

On cherche maintenant à optimiser le choix du learning rate. Pour cela, on se limite à l'étude du caractère K (choix arbitraire qu'on pourrait remettre en question mais on estime ici que le learning rate optimal pour cette lettre ne differera que de peu par rapport à celui des autres lettres), pour 100 époques, avec un nombre d'unités cachées fixé à 100 et 10 itérations de Gibbs. La figure 5 fournit les résultats obtenus pour la fonction de perte pour l'epoch final du modèle pour un RBM entraîné avec des learning rates différents.

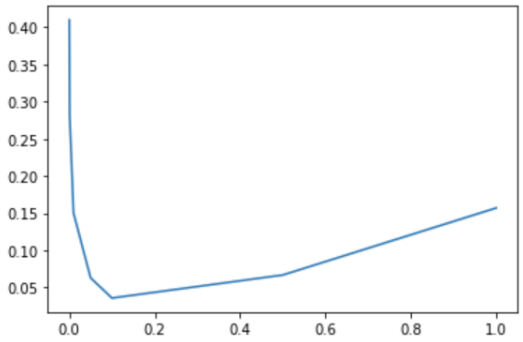


Figure 5: Evolution de la valeur finale de la fonction de perte pour différentes valeurs du learning rate.

On en conclut que, dans le cadre de notre étude, la valeur de 0.1 semble optimale pour le learning rate.

3.2 Etude selon le nombre d'unités cachées

De la même façon que dans la section précédente, on fixe le nombre d'epochs à 100, la lettre étudiée à K, le nombre d'itérations de Gibbs à 10 et cette fois-ci le learning rate à 0.1. La Figure 6 présente les variations de la fonction de perte finale obtenue pour différents nombres d'unités cachées.

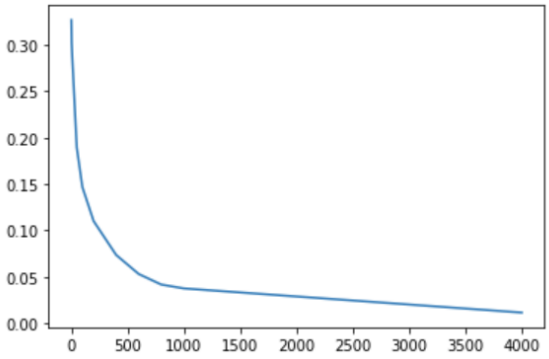


Figure 6: Evolution de la valeur finale de la fonction de perte pour différentes valeurs pour le nombre d'unités cachées.

Il apparaît qu'une augmentation du nombre d'unités cachées ne fait qu'améliorer les performances du modèle. L'amélioration des performances devient minime à partir d'une valeur de 1000 ce qui pourrait donc être qualifié la valeur optimale (le temps de calcul augmente légèrement lorsqu'on augmente le nombre d'unités).

3.3 Fonction de perte selon le caractère étudié

Le tableau 1 fournit la valeur de la perte quadratique finale obtenue pour chacun des caractères de la base de données. La configuration de l'étude étant de 100 épochs, un learning rate de 0.1, un nombre d'unités cachées de 100 et un nombre d'itérations de Gibbs de 100.

Caractère	**Perte quadratique après l'entraînement pour la base AlphaDigits**	**Perte quadratique après l'entraînement pour la base MNIST**
0	0.02	1e-3
1	0.03	1e-4
2	0.015	2e-3
3	0.017	8e-4
4	0.017	1e-3
5	0.016	4e-4
6	0.018	2e-3
7	0.018	8e-4
8	0.022	8e-4
9	0.017	5e-4

Table 1: Perte finale après entraînement pour chaque caractère

Il semble donc que l'algorithme n'ait aucune peine à converger quelque soit le chiffre étudié.

4 RBM appliqué au MNIST

La configuration utilisée pour le RBM dans le cas du MNIST est légèrement différente de celle utilisée dans le cas de AlphaDigits pour la section précédente. On choisit d'abord de ne pas travailler avec l'ensemble des données d'entraînement correspondant à chaque chiffre (~ 5000) mais avec 75 images afin de minimiser le temps de calcul, ce nombre est faible mais permet déjà d'obtenir des résultats assez satisfaisants. On choisit une configuration de 100 epochs, un learning rate de 0.1, un nombre d'unités cachées de 800 et un nombre d'itérations de Gibbs de 100 (pour un nombre d'unités cachées de 100, les images générées n'étaient pas satisfaisantes ce qui correspond aux résultats obtenus pour la Figure 6, la loss passe un seuil à partir d'un nombre d'unités cachés de 500). Le tableau 1 montre que le RBM converge également sans difficulté pour tous les chiffres de la base de données. La figure 7 montre les images générées par le RBM pour chaque catégorie.

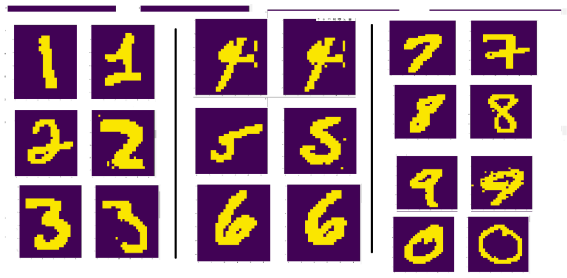


Figure 7: Images générées par le RBM pour MNIST (colonne de droite), comparées à une image de la base correspondant au label généré.

On constate que les chiffres générés sont assez représentatifs de chacune des catégories et que le modèle génère bien les différents types d'écriture pour chaque chiffre.
On observe cependant, pour certaines configurations, que des chiffres 9 "refermés" peuvent être générés. Cette situation apparaît lorsque peu de données d'entraînement sont utilisées et qu'un neuf fermé est présent dans la base de données. (voir Figure 8)

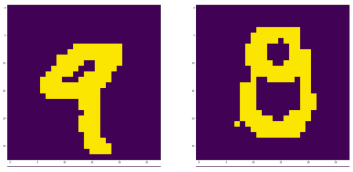


Figure 8: Génération d'un 9 fermé.

Le chiffre 1 semble aussi bien mieux traité par le MNIST car les pixels ont à peu près tous la même position alors que ce n'est pas le cas dans AlphaDigits. Le tableau 1 montre que le RBM appliqué au MNIST entraîne une fonction de perte plus faible, cela est sans doute dû à une variance plus faible dans la façon d'écrire un chiffre dans la base MNIST. Cette variabilité plus faible peut aussi être dû à la résolution de l'image qui est plus élevée pour MNIST (28x28 contre 20x16, plus de détails sont visibles sur un chiffre MNIST que sur un chiffre AlphaDigits, ce qui implique nécéssairement que le chiffre sera mieux retranscrit).

On note aussi qu'à contrario, un volume trop important de données déteriore la qualité des images générées car le RBM apprend des structures très particulières d'écriture qui ne sont pas nécéssaires à son apprentissage. Ce phénomène est observé pour un entraînement avec 2000 données d'entraînement pour un chiffre du MINST. (voir Figure 9)

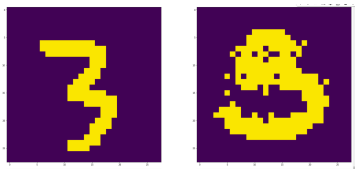


Figure 9: Génération d'un 3 pour 2000 données d'entraînement à droite et 75 données à gauche.

Finalement, meilleure est la qualité de l'image et plus les chiffres écrits dans la base de données se ressemblent, meilleurs seront les résultats. L'algorithme sera cependant bien plus long pour effectuer les 100 epochs (le RBM était déjà assez long à tourner pour une dizaine d'epochs et une image 100x100). L'augmentation du nombre d'images a également permis d'obtenir de meilleurs résultats pour certains chiffres.

5 GAN appliqué au MNIST

5.1 Introduction

Le principe des GAN est le suivant : deux réseaux, un générateur et un discriminateur, sont en compétition l'un avec l'autre. Le générateur crée des données "fausses" qu'il transmet au discriminateur. Le discriminateur voit également des données d'entraînement réelles et prédit si les données qu'il reçoit sont réelles ou fausses.

Le générateur est formé pour tromper le discriminateur, il veut produire des données qui ressemblent le plus possible à des données d'entraînement réelles. Le discriminateur est un classificateur entraîné à déterminer quelles données sont réelles et lesquelles sont fausses. En fin de compte, le générateur apprend à produire des données qui sont indiscernables des données réelles pour le discriminateur.

5.2 Chiffres générés

Les chiffres générés sont visibles sur la Figure 10. On constate que les chiffres ont davantage l'air d'avoir été écrit par un humain mais on constate cependant que les zones clés pouvant amener à reconnaître immédiatement le chiffre sont assez bruitées. Cela amène à des confusions entre certains chiffres tels que le 8 avec le 9 par exemple.

Figure 10: Génération des chiffres MNIST par un GAN.