MAT5016

Merlin Dumeur et Max Cohen

Février 2019

1 Apprentissage non supervisé

Dans cette partie, on utilisera exclusivement la base de donnée `binaryAlphaDigits`.

1.1 RBM

1.1.1 Training

On construit un RBM avec 16 neurones sur la couche cachée, que l'on entraîne pendant 30 époques (c'est à dire jusqu'à la convergence) sur notre base de donnée. Afin d'accélérer le training, on utilisera une descente de gradient par mini-batch de 32 exemples. Le taux d'apprentissage est fixé expérimentalement à 10^{-2}. On visualise le training à travers la fonction de coût (Mean Squared Error) qui décroît selon les époques (Figure 1).

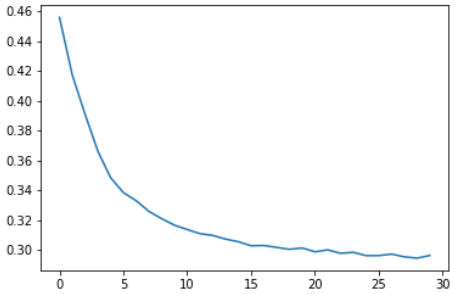

Figure 1: Coût en fonction des époques

1.1.2 Échantillonneur de Gibbs

Afin de générer des images suivant la distribution de la base de donnée, on utilise un échantillonneur de Gibbs, qu'on initialise avec un bruit blanc. On remarque que l'échantillonneur converge après environ 5 itérations (Figure 2)

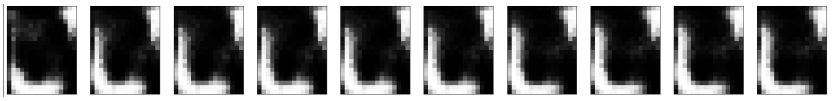

Figure 2: Variation de l'image générée selon les itérations de l'échantillonneur

1.1.3 Images générées

On fait tourner notre générateur plusieurs fois pour avoir une idée des images créées. Le nombre d'itérations est fixé à 10 pour assurer la convergence de l'échantillonneur. On ne distingue pas des caractère de la base de training, ce qui était prévisible car il y en a 36 pour seulement 16 neurones cachés. Cependant, on reconnaît facilement des formes familières (Figure 3)

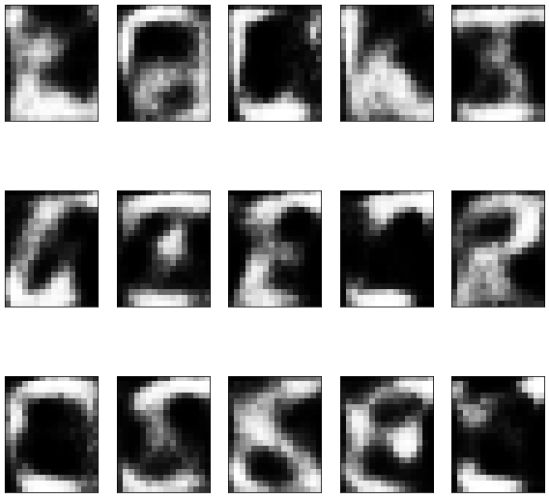

Figure 3: Images générées par le RBM

1.2 DBN

1.2.1 Training

On construit un DBN avec 64 neurones sur la première couche, 32 sur la seconde et 16 neurones sur la dernière couche cachée. On remarque qu'on a donc autant de neurones sur la dernière couche cachée que pour le RBM, environ moitité moins que le nombre de caractères dans la base de donnée.

On entraîne le réseau pendant 200 époques (c'est à dire jusqu'à la convergence) sur notre base de donnée. On utilise encore une descente de gradient par mini-batch de 32 exemples. Le taux d'apprentissage est fixé expérimentalement à 10^{-2}. On visualise le training à travers la fonction de coût (Mean Squared

Error), où chaque couleur représente une couche, et qui décroît selon les époques (Figure 4).

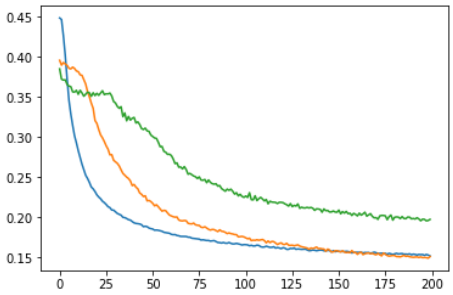

Figure 4: Coût en fonction des époques

1.2.2 Échantillonneur de Gibbs

On effectue la même expérience que précédemment, mais avec plus d'itérations. En effet il faut maintenant une trentaine d'itérations avant que l'échantillonneur ne converge. La Figure 5 présente la même image, en faisant augmenter le nombre d'itérations de 1 à 100 par pas de 10.

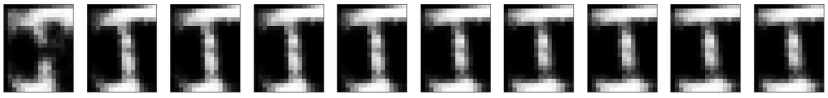

Figure 5: Variation de l'image générée selon les itérations de l'échantillonneur

1.2.3 Images générées

On fait tourner notre générateur plusieurs fois pour avoir une idée des images créées. Le nombre d'itérations est fixé à 50 pour assurer la convergence de l'échantillonneur. Cette fois-ci, on distingue des caractère de la base de training, ce qui n'était pas la cas avec le RBM (Figure 6).

2 Apprentissage supervisé

Dans cette partie, on travaille exclusivement sur la base de donnée `mnist`.

2.1 Réseau

On construit un réseau à 3 couches, avec 64 neurones dans la première, 32 dans la seconde et 10 dans la dernière, ce qui correspond aux dix caractères de la base de donnée. Pour l'implémention, la classe `DNN` hérite de la classe

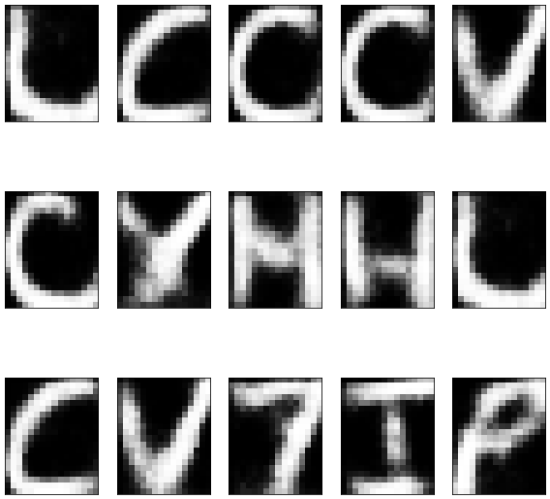

Figure 6: Images générées par le DBN

`DBN`, et beaucoup de fonctions restent inchangées. Cela permet en particulier de réutiliser la fonction de pré-training.

2.2 Training et analyse

2.2.1 Sans pré-train

On entraîne directement l'algorithme de manière supervisée. Sur 10 époques, on utilise une descente de gradient par mini-batch de 16 exemples, avec un taux d'apprentissage de 3. L'algorithme atteint 90% d'accuracy sur la base de test (Figure 7).

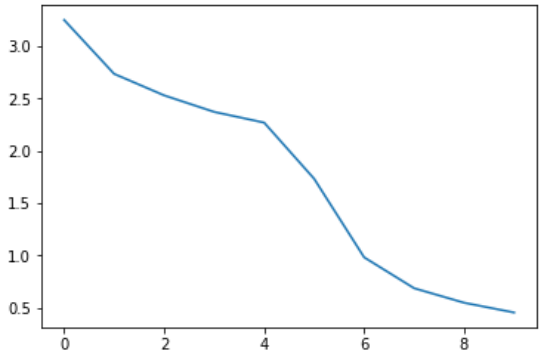

Figure 7: Évolution du coût (initialisation aléatoire)

2.2.2 Avec pré-train

On commence cette fois par un pré-training non supervisé. La fonction est donc similaire que pour un `DBN`, sauf que l'on exclut la dernière couche, ce qui explique qu'il n'y ait que deux courbes sur le graphe (Figure 8).

L'entraînement est effectué de la même manière que précédemment, si ce n'est le taux d'apprentissage fixé à 0.03.

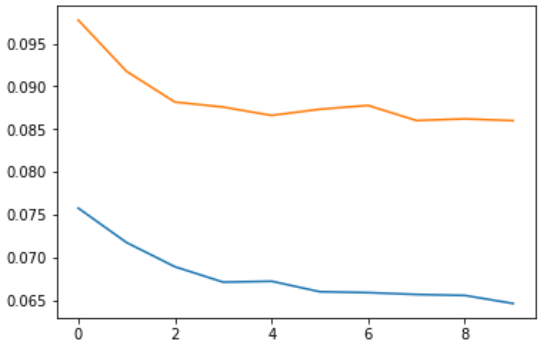

Figure 8: Évolution du coût du DBN (pré entraînement)

On exclut pour l'instant de générer des données comme dans la partie précédente puisque les poids de la dernière couche sont encore aléatoires.

On effectue ensuite un training supervisé, en gardant les même paramètres que pour la partie précédente. L'accuracy est très élevée dès la première époque, et on converge en 10 époques à 94% (Figure 9).

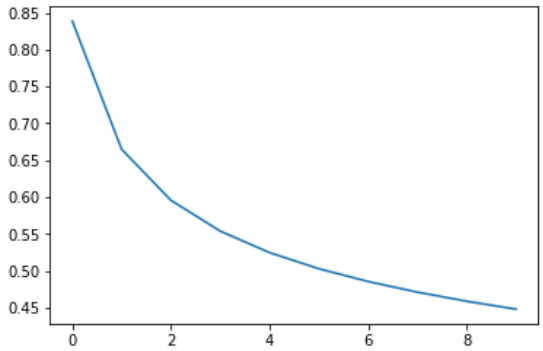

Figure 9: Évolution du coût du DNN (pré entraîné)

2.2.3 binaryAlphadigits

La différence entre un modèle initialisé aléatoirement et un modèle pré entraîné n'est pas flagrante sur `mnist`, c'est pourquoi nous avons aussi effectué des train-

ing sur `binaryAlphaDigits`. Cette fois, la différence est importante. Sans pré entraînement, le réseau met beaucoup d'époques à converger, et dépasse difficilement les 60% d'accuracy. Après un entraînement non supervisé, on converge très rapidement, et on atteint des accuracy beaucoup plus élevées.

2.3 Images générées

Les résultats de l'échantillonneur après un entraînement supervisé ne sont pas particulièrement intéressants, en pratique on retrouve toujours approximativement la même image (Figure 10).

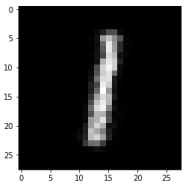

Figure 10: Image générée après un entraînement supervisé

2.4 Résultats

Le classifier permet bien de classifier les donnée de la base de donnée, comme on le voit sur la Figure 11.

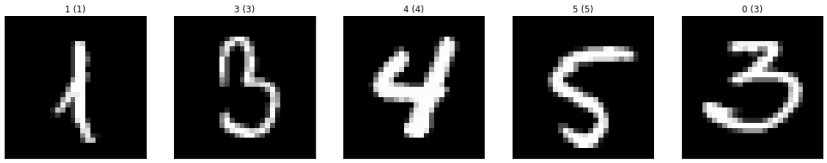

Figure 11: Résultats du classifier sur la base de test