Rapport de projet Deep Learning II

Cadmos Kahalé-Abdou, João Luís Reis Freitas, et Nicolas Oulianov

Institut Polytechnique de Paris

May 29, 2021

1 Consignes

Pour l'implémentation, nous avons décidé d'utiliser le langage Python. Le code est disponible à l'adresse suivante : https://github.com/oulianov/dl2-rbm

2 Données

Nous utilisons les images du dataset Binary AlphaDigits (http://www.cs.nyu.edu/~roweis/data.html) et MNIST (http://yann.lecun.com/exdb/mnist/).

3 Fonctions élémentaires

Travaillant en Python, nous avons décidé d'utiliser une architecture orientée objet plutôt que fonctionnelle. Yohan Petetin nous a confirmé en cours que c'était tout à fait possible.

Nous créons donc des objets RBM, DBN, Layer, et DBN. L'idée est qu'un DBN est une interface pour gérer une liste de RBM, et un DBN une interface pour gérer une liste de Layer (couches d'un réseau de neurones).

Nos conventions de nommage étant légèrement différentes des instructions, nous faisons ici la liste des équivalences entre nom des fonctions dans les consignes, et méthodes de classe dans notre code.

Nom de la fonction dans les consignes	**Fonction ou méthode équivalente dans notre code**
lire_alpha_digit	lire_alpha_digit
init_rbm	RBM.__init__
entree_sortie_RBM	RBM.entree_sortie
sortie_entree_RBM	RBM.sortie_entree
train_RBM	RBM.train
generer_image_RBM	RBM.generer_image
init_DNN	DNN.__init__
pretrain_DNN	DBN.train ; DNN.init_DNN_with_DBN
generer_image_DBN	DBN.generer_image
calcul_softmax	calcul_softmax
entree_sortie_reseau	DBN.entree_sortie_reseau
retropropagation	DBN.train
test_DNN	DBN.error_rate

4 Travail préliminaire

Nous entraînons d'abord un RBM et un DBN sur un extrait des données AlphaDigits.

4.1 Images générées par un RBM

Nous créons d'abord un RBM avec 100 unités cachées. Nous l'entraînons sur les données AlphaDigits correspondant à la lettre F pendant 200 epochs, une taille de batch de 32, et un learning rate de 0.05. Nous générons 4 images avec 20 itérations du sampleur de Gibbs.

Voilà les images générées :

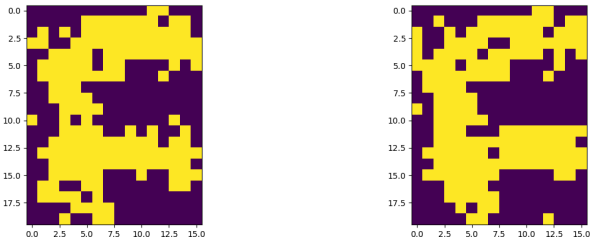

Figure 1: Images générées par un RBM entraîné sur la lettre F

Le RBM est bien capable de générer une image ressemblant à la lettre F. Toutefois, elle est très bruitée. On observe de nombreux artifacts qui rendent les traits de la lettre imprécis. Dans ce qui devrait être un trait plein, on observe des trous.

4.2 Images générées par un DBN

Nous créons ensuite un DBN avec 3 couches cachées de 100 unités. Nous l'entraînons sur les données AlphaDigits correspondant à la lettre A pendant 100 epochs d'entraînement par RBM, une taille de batch de 32 et un learning rate de 0.1. Nous générons 4 images avec 40 itérations du sampleur de Gibbs.

Voilà les images générées :

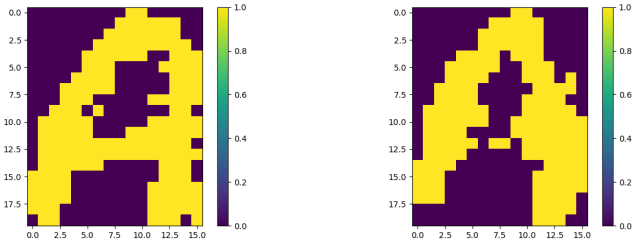

Figure 2: Images générées par un DBN entraîné sur la lettre A

Le DBN parvient bien à générer une image ressemblant à la lettre A. La génération est bien moins bruitée que pour le RBM. Il y a moins d'artifacts détachés des traits de la lettre. Toutefois, on observe que sur l'image de droite la barre horizontale du A ne relie pas totalement les deux traits verticaux. Sans doute avec une architecture plus grande ou un sampleur de Gibbs mieux paramétré on pourrait avoir de plus beaux résultats.

A noter que nous avons essayé d'entraîner le DBN sur le dataset entier : toutes les lettres simultanément. Mais nous n'avons pas réussi à obtenir des résultats de génération satisfaisant : le DBN générait seulement une grosse boule jaune au milieu.

5 Etude à réaliser (MNIST)

Dans cette partie, nous travaillons sur le dataset MNIST. La tâche consiste à prédire le chiffre représenté sur une image du dataset. Ces images sont considérées comme 784 pixels qui sont soit noirs (0), soit blancs (1) suite à un seuillage à 0,5. Il n'y a pas de niveau de gris.

Nous comparerons les taux d'erreurs de deux réseaux de neurones fully connected initialisés différemment. Dans les schémas suivant, nous nous référerons aux intiialisations grâce aux noms suivants :

- **Random init :** Initialisation aléatoire. Chaque poids du DNN est initialisé grâce à une loi normale de moyenne 0 et d'écart type 1.
- **DBN Pretrain :** Préentraînement non-supervisé d'un DBN. Chaque poids du DNN est initialisé grâce aux poids de ce DBN. Précisément, on initialise uniquement les couches cachées : la dernière couche, la couche de sortie, est intialisée aléatoirement, de même que ci-dessus.

Les paramètres liés au réseau et à l'apprentissage sont, sauf exception, les suivants :

Architecture du réseau	1 couche d'entrée de 784 unités, 2 couches cachées de 200 unités, 1 couche de sortie de 10 unités
Nombre d'epochs entraînenemnt DNN	20 itérations de descente de gradient
Nombre d'epochs pré-entraînement DBN	20 itérations par RBM du DBN
Taille de batch	128 samples
Learning rate	0.1
Nombre de données d'apprentissage	10000
Nombre de données de test	40000

Nous utilisons la sigmoïde comme fonction d'activation pour les couches cachées intermédiaires.

En raison de la durée d'entraînement, nous entraînons chaque modèle une seule fois. Le taux d'erreur rapporté est donc sujet à une variance peut-être élevée.

5.1 Influence du nombre de couches

Nous étudions l'influence du nombre de couches. Dans cette partie, nous entraînons des modèles ayant entre 2 et 5 couches cachées de 200 unités.

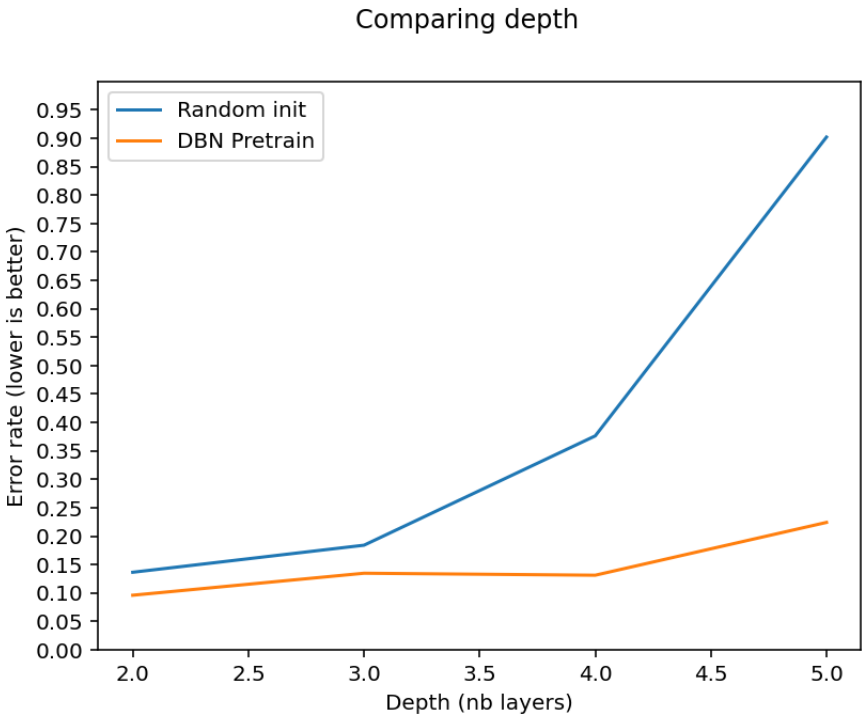

Figure 3: Taux d'erreur en fonction du nombre de couches

Le préentraînement DBN a un taux d'erreur bien plus faible et bien plus stable avec l'augmentation du nombre de couches que l'initialisation aléatoire, même avec un réseau très profond.

En effet, les réseaux de neurones profonds sont durs à entraîner, notamment à cause du problème du vanishing gradient. Le réseau de neurone avec initialisation aléatoire illustre ce problème.

Le préentraînement DBN mitige cet effet, mais y reste sensible. Peut-être avec plus d'itérations, on aurait de meilleurs résultats pour les réseaux de neurones les plus profonds.

5.2 Influence du nombre de neurones par couche

Nous étudions l'influence du nombre de neurones par couche. Dans cette partie, nous entraînons des modèles avec 2 couches cachées, ayant entre 100 et 700 unités.

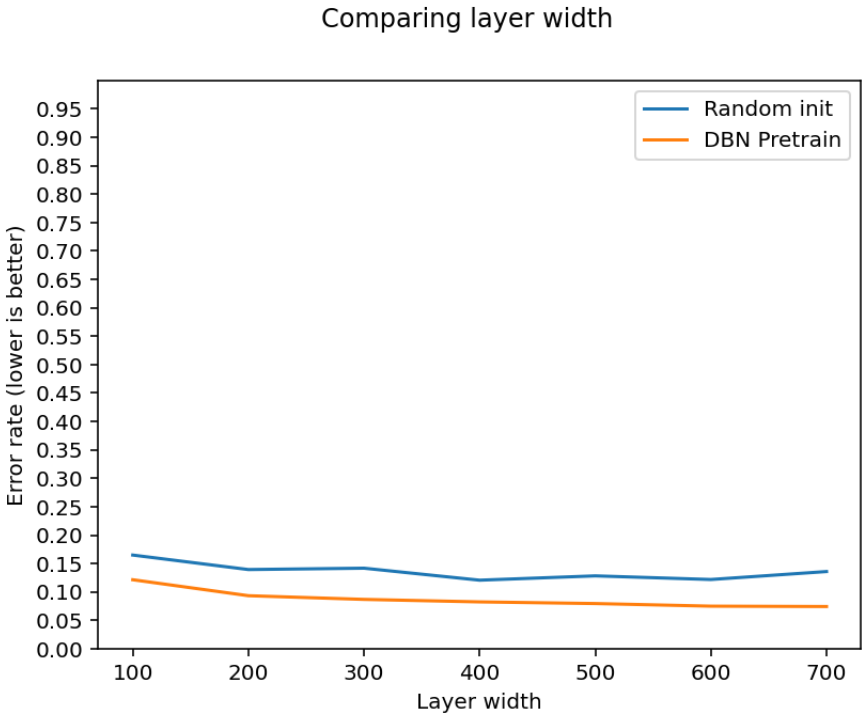

Figure 4: Taux d'erreur en fonction du nombre de neurones par couche

Pour le DBN Pretrain, augmenter le nombre de neurones par couche améliore les performances. Ce n'est pas le cas pour le Random Init. Ce n'est pas normal. On sait que les réseaux de neurones sont des approximateurs universels pour peu qu'il y ait assez d'unités sur leur couche caché, donc augmenter le nombre d'unités devrait améliorer les performances du modèle.

C'est sans doute un signe d'underfitting, c'est-à-dire que nous n'entraînons pas sur suffisamment d'epochs ou avec de bons hyperparamètres les réseaux de neurones. Nous n'avons malheureusement pas pu faire plus d'essais en raison du temps de calcul. En effet, générer ce graphique a pris plus d'une heure et demi.

Cependant, les performances du préentraînement DBN sont meilleures. On peut penser que c'est parce que le préentraînement DBN ajoute 30 epochs d'entraînement : le réseau préentraîné a un avantage déloyal. Peut-être que l'écart s'effacerait si on entraînait le modèle avec initialisation aléatoire pendant 40 epochs au lieu de 20.

5.3 Influence du nombre de données d'entraînement

Nous étudions l'influence du nombre de données d'entraînement. Dans cette partie, nous entraînons des modèles sur un dataset ayant entre 100 et 60000 données d'entraînement.

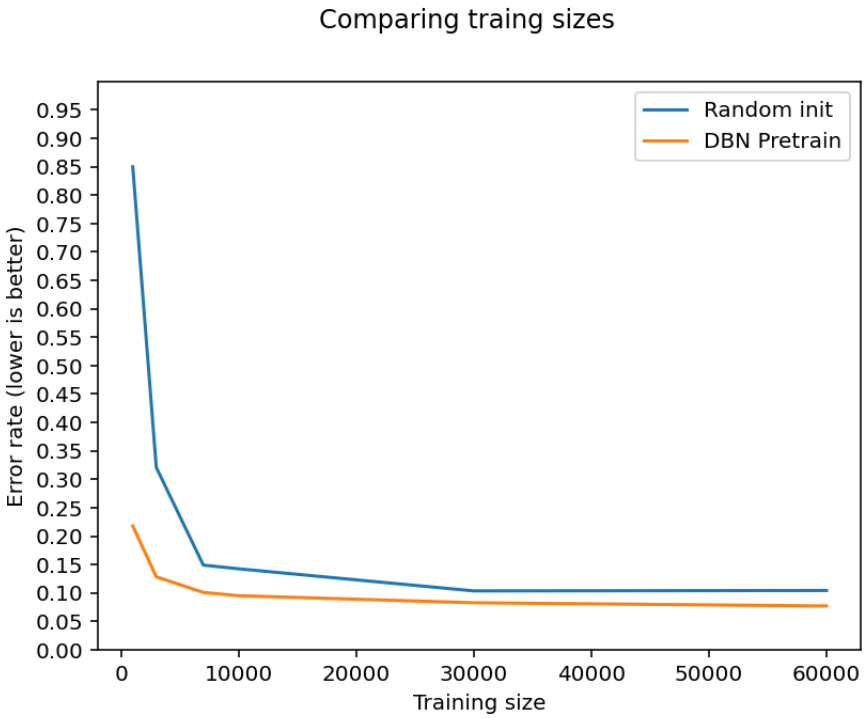

Figure 5: Taux d'erreur en fonction du nombre de données d'entraînement

Augmenter la taille du dataset d'entraînement améliore les performances de tous les réseaux de neurones. Le préentraînement DBN est meilleur que l'initialisation aléatoire, mais l'écart s'estompe avec l'augmentation du nombre de données d'entraînement.

Contrairement aux indications, nous n'avons hélas pas réussi à obtenir de situation où l'initialisation aléatoire était meilleure que le préentraînement DBN. C'est peut-être dû à notre choix d'hyperparamètres, ou au fait que nous n'ayons essayé qu'une seule fois en raison du temps de calcul.

Il est possible également que l'optimisation se coince sur un minima local car nous avons implémenté la version la plus simple de la descente de gradient. En effet, même sur l'ensemble de l'entraînement, les valeurs de la loss cross-entropy restaient coincées aux alentours de 0,3.

5.4 Meilleure architecture

On choisit d'entraîner un réseau de neurones avec 3 couches cachées de 700 unités, étant donné que les architectures profondes avec beaucoup d'unités ont eu les taux d'erreur les plus bas.

On initialise les poids grâce à un préentraînement DBN sur 10000 samples et 20 epochs. Le learning rate choisi est 0.1, et la taille de batch est 128. Puis on entraîne le DNN sur les 60000 données d'entraînement, pendant 10 epochs. Le learning rate est également de 0.1 mais la taille de batch est 256.

Le score final est de 6,9% d'erreur.

Nous aurions pu faire un réseau encore plus profond et encore plus large. Cependant, nous n'avions pas la puissance de calcul nécessaire à le faire tourner suffisamment rapidement.