Projet Deep Learning II
Institut Polytechnique de Paris

Yohan Petetin

23/01/2023

1 Consignes

L'objectif du TP est de réaliser, dans un langage de votre choix (R, Python, Matlab,...), un réseau de neurones profond pré-entraîné ou non pour la classification de chiffres manuscrits. Il vous est demandé de comparer les performances, en terme de taux de bonnes classifications, d'un réseau pré-entrainé et d'un réseau initialisé aléatoirement, en fonction du nombre de données d'apprentissage, du nombre de couches du réseau et enfin du nombre de neurones par couches.

Pour ce travail, vous devez rendre avant le XXX, 23h59mn59s (yohan.petetin@telecom-sudparis.eu):

- un compte rendu en .pdf contenant vos noms-prénoms, "noms-TP-DNN.pdf" détaillant l'analyse de vos résultats et comment exécuter votre code. Il est inutile de décrire chaque fonction que vous avez implémenté. L'étude à mener est spécifiée au Paragraphe 5;
- vos programmes dans une archive nom-codes.zip; cette archive ne contiendra pas les données.

Les sections suivantes vous guideront sur le travail à réaliser.

2 Données

On considère deux jeux de données. La première base de données contient des images de petite dimension et servira à tester rapidement vos différents programmes pour l'entrainement *non supervisé*. La seconde base de données MNIST est une base couramment utilisée en machine learning et sert de benchmark. C'est sur cette base que l'apport d'un pré-entrainement non supervisé sera évalué.

Les données : elles seront représentées par une matrice dont une ligne représentera une donnée et une colonne une composante de la donnée.

Les labels : ils seront également représentés par une matrice de 0 et de 1 dont la ligne représentera le numéro de la donnée et la colonne la classe correspondante à la donnée. Ces labels ne seront donnés que pour la base de données du MNIST.

- Récupérer les images de la base de données Binary AlphaDigits à partir de l'adresse suivante : `http://www.cs.nyu.edu/~roweis/data.html` et la base de données MNIST `http://yann.lecun.com/exdb/mnist/`;

- Pour la base Binary AlphaDigits, on écrira une fonction `lire_alpha_digit` permettant de récupérer les données sous forme matricielle (en ligne les données, en colonne les pixels) et qui prend en argument les caractères que l'on souhaite "apprendre". Pour cette base de données, on ne s'intéressera pas aux labels dans la mesure où elle ne servira qu'à vérifier les programmes associés à l'étape d'estimation non supervisée. Pour la base de données du MNIST, on pourra récupérer une fonction déjà écrite mais on veillera à transformer les images en niveau de gris en noir et blanc.

3 Fonctions élémentaires

Un RBM pourra être représenté par un objet/une structure contenant un champ W (matrice de poids reliant les variables visibles aux variables cachées), un champ a (biais des unités d'entrée) et un champ b (biais des unités de sortie)

Un réseau de neurones (DNN) et un Deep Belief Network (DBN) pourront être représentés par une liste de RBM, la taille de cette liste étant égale au nombre de couches cachés du réseau (+ couche de de classfication dans le cas du DNN). Chaque élement de cette liste coincidera donc à un RBM et contiendra un champ W (matrice de poids reliant 2 couches consécutives), un champ a (biais des unités d'entrée qui coincident avec les paramètres variationnels estimés, sauf pour la première couche) et un champ b.

Voici une liste de fonctions à implémenter qui vous permettra de construire/d'apprendre votre réseau de neurones.

3.1 Construction d'un RBM et test sur Binary AlphaDigits

On complètera au fur et à mesure un script `principal_RBM_alpha` permettant d'apprendre les caractères de la base Binary AlphaDigits de votre choix via un RBM et de générer des caractères similaires à ceux appris. La construction de ce programme nécessite les fonctions suivantes:

- Écrire une fonction `lire_alpha_digit` permettant de récupérer les données sous forme matricielle (en ligne les données, en colonne les pixels) et qui prend en argument les caractères (ou leur indice $0, \cdots, 35$) que l'on souhaite "apprendre".

- Écrire une fonction `init_RBM` permettant de construire et d'initialiser les poids et les biais d'un RBM. Cette fonction retournera une structure RBM avec des poids et biais initialisés. On initialisera les biais à 0 tandis que les poids seront initialisés aléatoirement suivant une loi normale centrée, de variance égale à 0.01.

- Écrire une fonction `entree_sortie_RBM` qui prend en argument une structure RBM et des données d'entrée et qui retourne la valeur des unités de sortie calculées à partir de la fonction sigmoïde.

- Écrire une fonction `sortie_entree_RBM` qui prend en argument un RBM, des données de sortie et qui retourne la valeur des unités d'entrée à partir de la fonction sigmoïde.

- Écrire une fonction `train_RBM` permettant d'apprendre de manière non supervisée un RBM par l'algorithme Contrastive-Divergence-1. Cette fonction retournera un structure RBM et prendra en argument une structure RBM, le nombre d'itérations de la descente de gradient (epochs), le learning rate, la taille du mini-batch, des données d'entrées... À la fin de chaque itération du gradient, on affichera l'erreur quadratique entre les données d'entrées et les données reconstruites à partir de l'unité cachée afin de mesurer le pouvoir de reconstruction du RBM.

- Écrire une fonction `generer_image_RBM` permettant de générer des échantillons suivant un RBM. Cette fonction retournera et affichera les images générées et prendra en argument une structure de type RBM, le nombre d'itérations à utiliser dans l'échantillonneur de Gibbs et le nombre d'images à générer.

3.2 Construction d'un DBN et test sur Binary AlphaDigits

On complètera un script `principal_DBN_alpha` permettant d'apprendre les caractères de la base Binary AlphaDigits de votre choix via un DBN et de générer des caractères similaires à ceux appris. La construction de ce programme nécessite les fonctions suivantes:

- Écrire une fonction `init_DBN` permettant de construire et d'initialiser aléatoirement les poids et les biais d'un DBN. Cette fonction retournera un DBN, prendra en argument la taille du réseau et pourra utiliser de manière itérative la fonction `init_RBM`.

- Écrire une fonction `train_DBN` permettant d'apprendre de manière non supervisée un DBN (Greedy layer wise procedure). Cette fonction retournera un DBN pré-entrainé et prendra en argument un DBN, le nombre d'itérations de la descente de gradient, le learning rate, la taille du mini-batch, des données d'entrées. On rappelle que le pré-entrainement d'un DBN peut être vu comme l'entrainement successif de RBM. Cette fonction utilisera donc `train_RBM` ainsi que `entree_sortie_RBM`.

- Écrire une fonction `generer_image_DBN` permettant de générer des échantillons suivant un DBN. Cette fonction retournera et affichera les images générées et prendra en argument un DNN pré-entrainé, le nombre d'itérations à utiliser dans l'échantillonneur de Gibbs et le nombre d'images à générer.

3.3 Construction d'un DNN et test sur MNIST

On rappelle qu'un DNN peut être vu, informatiquement parlant, comme une liste de RBM. On complètera au fur et à mesure un programme principal `principal_DNN_MNIST` permettant d'apprendre un réseau de neurones profonds pré-entrainé par via un DBN.

- Écrire une fonction `init_DNN` permettant de construire et d'initialiser aléatoirement les poids et les biais d'un DNN. Cette fonction retournera un DBN, prendra en argument la taille du réseau et utilisera la fonction `init_DBN` (on rapelle qu'un DNN est un DBN avec une couche de classification supplémentaire).

- Écrire une fonction `pretrain_DNN` permettant de préentrainer les couches cachés (hors couche classification) du DNN. Cette fonction retournera un DNN pré-entrainé et prendra en argument un DNN, le nombre d'itérations de la descente de gradient, le learning rate, la taille du mini-batch, des données d'entrées et utilisera la fonction `train_DBN`.

- Écrire une fonction `calcul_softmax` qui prend en argument un RBM, des données d'entrée et qui retourne des probabilités sur les unités de sortie à partir de la fonction softmax. Cette fonction vise à implémenter la couche de classification du DNN.

- Écrire une fonction `entree_sortie_reseau` qui prend en argument un DNN, des données en entrée du réseau et qui retourne dans une liste les sorties *sur chaque couche du réseau* (couche d'entrée inclue) ainsi que les probabilités sur les unités de sortie. Cette fonction pourra utiliser les fonctions `entree_sortie_RBM` et `calcul_softmax`.

- Écrire la fonction `retropropagation` permettant d'estimer les poids/biais du réseau à partir de données labellisées. Cette fonction retournera un DNN et prendra en argument un DNN, le nombre d'itérations de la descente de gradient, le learning rate, la taille du mini-batch, des données d'entrée, leur label,... On pensera à calculer à la fin de chaque epoch, après la mise à jour des paramètres, la valeur de l'entropie croisée que l'on cherche à minimiser afin de s'assurer que l'on minimise bien cette entropie.

- Écrire une fonction `test_DNN` permettant de tester les performances du réseau appris. Cette fonction prendre en argument un DNN appris, un jeu de données test, et les vrais labels associés. Elle commencera par estimer le label associé à chaque donnée test (on pourra utiliser `entree_sortie_reseau`) puis comparera ces labels estimés aux vrais labels. Elle retournera enfin le taux d'erreur.

4 Etude sur Binary AlphaDigit

On commencera à travailler exclusivement sur la base de données Binary AlphaDigit afin de vérifier la cohérence de vos programmes `train_RBM` et `train_DBN`. Voici un plan pour vous guider dans la construction de votre programme principal.

- Spécifier les paramètres liés au réseau et à l'apprentissage : taille du réseau (vecteur contenant le nombre de neurones), nombre d'itérations (ex : 100) pour les descentes de gradient , learning rate (ex: 0.1), taille des mini-batch, le nombre de données d'apprentissage, ...

- Charger les données;

- Commencer par entraîner de manière non supervisée un RBM et vérifier que le RBM entraîné est capable de générer des données dont la structure est semblable aux données d'apprentissage;

- Pré-entraîner de manière non supervisée le DBN (sans considérer la couche de sortie) et vérifier que le DBN entraîné est capable de générer des données dont la structure est semblable aux données d'apprentissage;

Un aperçu des images régénérées pour chaque modèle, en fonction du nombre de neurones, de couches et du nombre de caractères à apprendre sera présenté dans le rapport. On veillera à étudier avec soin les limites de ces modèles génératifs par rapport au nombres de caractères qu'il est possible d'apprendre.

5 Étude à réaliser sur MNIST

5.1 Programme

Lorsque vos algorithmes sont plus ou moins capables de re-générer les caractères appris de la base Binary AlphaDigit, on se focalisera sur l'analyse finale qui s'effectuera sur la base MNIST. qu'on pensera à binariser (0 pour les pixels dont la valeur est inférieur à 127, 1 pour ceux dont la valeur est supérieur à 127). Voici un plan pour vous guider dans la construction de votre programme principal:

- Spécifier les paramètres liés au réseau et à l'apprentissage : taille du réseau (vecteur contenant le nombre de neurones), nombre d'itérations pour les descentes de gradient (100 pour les RBM, 200 pour l'algorithme de rétro-propagation du gradient), learning rate (ex : 0.1), taille des mini-batch, le nombre de données d'apprentissage, ...
- Charger les données;
- Initialisation aléatoire du DNN;
- Si pré-apprentissage, pré-entraîner de manière non supervisée le DNN;
- Entraîner de manière supervisé le DNN préalablement pré-entrainé via l'algorithme de rétro-propagation du gradient.
- Avec le réseau appris, observer les probabilités de sortie de quelques images de la base d'apprentissage.

5.2 Analyse

Voici enfin un plan pour vous aider à comparer les performances d'un réseau pré-entrainé avec un réseau initialisé aléatoirement. À partir des données 'train' de la base MNIST :

1. initialiser deux réseaux identiques;
2. pré-apprendre un des deux réseau en le considérant comme un empilement de RBM (apprentissage non supervisé);
3. apprendre le réseau pré-appris préalablement avec l'algorithme de rétro-propagation;
4. apprendre le second réseau qui a été initialisé aléatoirement avec l'algorithme de rétro-propagation;
5. Calculer les taux de mauvaises classifications avec le réseau 1 (pré-entraîné + entraîné) et le réseau 2 (entraîné) à partir du jeu 'train' et du jeu 'test'

3 figures suffisent à analyser vos résultats :

- Fig 1 : 2 courbes exprimant le taux d'erreur des 2 réseaux en fonction du nombre de couches (par exemple 2 couches de 200, puis 3 couches de 200, ... puis 5 couches de 200). On utilisera toutes les données d'apprentissage et test;
- Fig 2 : 2 courbes exprimant le taux d'erreur des 2 réseaux en fonction du nombre de neurones par couches (par exemple 2 couches de 100, puis 2 couches de 300, ...puis 2 couches de 700,...). on utilisera toutes les données d'apprentissage et test;
- Fig 3 : 2 courbes exprimant le taux d'erreur des 2 réseaux en fonction du nombre de données train (par exemple on fixe 2 couches de 200 puis on utilise 1000 données train, 3000, 7000, 10000, 30000, 60000).

On cherchera enfin une configuration permettant d'obtenir le meilleur taux de classification possible.

6 Bonus : Auto-encodeurs variationnels

Comparer visuellement des images générées par i) un RBM; ii) un DBN; iii) un VAE avec, à peu près, le même nombre de paramètres, lorsque ces modèles ont été entrainés sur la base de données MNIST.