

Intelligence Bio Inspirée

Réseaux de neurones

Mathieu Lefort

15 et 16 décembre 2016


Vous rendrez un compte-rendu de TP incluant votre code et vos réponses aux questions à la fin de chacune des 2 séances de TPs et la version finale pour le 03/01/17. Pensez à indiquer votre nom dans le fichier déposé sur Spiral.

1 Objectif

L'objectif de ce projet est d'implémenter l'algorithme de perceptron multi-couches vu en cours, de comprendre son fonctionnement et ses limites.

2 Données

Vous avez à votre disposition 3 jeux de données dérivés de la base MNIST qui contient des images de chiffres

manuscripts (images de taille 28*28) comme par exemple : . Chaque base de données est structurée comme suit : ((tableau_image_apprentissage, tableau_label_apprentissage), (tableau_image_test, tableau_label_test)). Les images sont stockées sous la forme d'un vecteur de 784 (28*28) valeurs et les labels sont sous la forme d'un codage tabulaire. Par exemple si le chiffre représenté sur l'image est un 5, son label sera : [0, 0, 0, 0, 0, 1, 0, 0, 0, 0].

Les 3 bases de données sont :

- `mnist.pkl.gz` qui contient tous les chiffres
- `mnist0-4.pkl.gz` qui contient uniquement les chiffres de 0 à 4
- `mnist5-9.pkl.gz` qui contient uniquement les chiffres de 5 à 9

3 Algorithme

L'algorithme du perceptron multi-couches, qui pour rappel correspond à la descente de gradient stochastique de l'erreur quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (t - y)^2$ pour le réseau de neurone, est le suivant :

Pour chaque entrée choisie au hasard dans la base :

1. Calculer la sortie $y_i^l = \frac{1}{1 + e^{-\sum_j w_{ji}^l y_j^{l-1}}}$ de chaque neurone i de chaque couche l du réseau par propagation couche par couche de l'activité¹
2. Pour chaque neurone i de la couche n de sortie, calculer l'erreur : $\delta_i^n = y_i^n(1 - y_i^n)(t_i - y_i^n)$
3. Rétro-propager couche par couche l'erreur à travers chaque neurone i de chaque couche l du réseau $\delta_i^l = y_i^l(1 - y_i^l) \sum_j \delta_j^{l+1} w_{ij}$
4. Modifier chaque poids $\Delta w_{ij}^l = \eta \delta_j^l y_i^{l-1}$

1. Pour la première couche la sortie est fonction de l'entrée courante \mathbf{x} : $y_i^l = \frac{1}{1 + e^{-\sum_j w_{ji}^l x_j}}$

4 Partie 1 : Codage de l'algorithme

Question : Implémenter l'algorithme du perceptron multi-couches en python. Un squelette de code ainsi que des tutoriels vous sont fournis sur Spiral. Vous pouvez lancer un programme python par la commande `python fichier.py` ou lancer un interpréteur directement avec la commande `python` pour tester des morceaux de votre code.

Question : Testez votre algorithme sur la base `mnist.pkl.gz`. Trouver un paramétrage (nombre de couches, nombre de neurones par couche, nombre d'itérations, pas d'apprentissage) qui offre de bonnes performances en un temps raisonnable².

5 Partie 2 : Étude de l'algorithme

Question : En utilisant le paramétrage trouvé dans la question précédente, entraînez un réseau différent avec chacune des bases d'apprentissage proposées. Pour chaque réseau testez le sur les 3 bases de tests (vous aurez donc 9 résultats). Discutez des résultats obtenus.

Question : Reprenez le réseau appris sur la base `mnist0-4.pkl.gz` de la question précédente. Étudiez l'évolution des performances de ce réseau sur les bases `mnist0-4.pkl.gz` et `mnist5-9.pkl.gz` lors que vous apprenez maintenant au réseau les chiffres de 5 à 9.

6 Partie 3 : Pour aller plus loin (optionnel)

Si vous avez le temps et l'envie vous pouvez faire les questions suivantes.

Question : Regardez l'évolution de la performance d'un réseau au fur et à mesure de son apprentissage. Pour l'arrêter lorsqu'il est le plus performant, en pratique on utilise une méthode dite d'*early stopping*. Il s'agit de retirer une partie des données de la base d'apprentissage et de les utiliser pour valider l'évolution de l'apprentissage du réseau. Les poids qui auront eu les meilleures performances sur ces données de validation seront considérés comme les poids finaux du réseau qui seront alors testés sur la base de test. Mettez en place ce mécanisme en utilisant les bases suffixées par `_validation` contenant les données d'apprentissage puis de validation puis de test.

Question : Remplacer la fonction d'activation sigmoïde d'un neurone par un softplus d'équation : $\ln(1+e^x)$. Il faut alors recalculer les équations de rétro-propagation du gradient. Pour information la dérivée de la fonction softplus est la fonction sigmoïde.

2. Comme point de départ, avec une couche cachée de 10 neurones, un pas d'apprentissage de 0.1 et 100000 itérations vous devriez avoir une erreur d'environ 10%