TP : Réseau de neurones sous matlab

Clément Chatelain

December 12, 2012

Introduction

Ce TP a pour but de vous faire développer un réseau de neurones à une couche cachée et d'apprendre à contrôler son apprentissage.

1 Les données

On se propose d'utiliser des données de bases étiquetées connues disponible sur le site de l'UCI. Afin de gagner du temps, certaines d'entre elles sont disponibles au format matlab à l'adresse suivante :

http://clement.chatelain.free.fr/rdn/

Le fichier *XXX.mat* peut être chargé à l'aide de la commande *load XXX.mat*, vous obtenez alors une matrice *nb sample* lignes et *nb carac+1* colonnes, où la dernière colonne est la classe associée à l'exemple.

2 Travail à réaliser

Pour commencer, transformer le fichier .mat en matrices X et Y comme définies dans le cours. Centré et réduire la matrice X serait une bonne idée. Comme pour tout problème d'apprentissage, il convient de séparer la base en Apprentissage, Validation et Test. La base de validation sert à régler correctement les paramètres du réseau et à contrôler le surapprentissage, alors que la base de test n'est utilisée qu'une seule fois, à l'aide des meilleurs paramètres trouvés sur la base de validation.

Il faut ensuite construire le réseau de neurones. Dans le cadre de ce TP, on se limitera à un réseau de type MLP, avec une couche d'entrée (les données), une couche cachée, et une couche de sortie de type softmax.

2.1 Première implémentation du réseau

Voici une proposition de marche à suivre :

- Construire un réseau à E entrées, J neurones en couche cachée, et 10 sorties. Définir les vecteurs de poid w_{ji} et w_{kj} et les initialiser au hasard.

Remarque : on prendra des valeurs au hasard entre -1 et 1 pour éviter les problèmes de saturation. Définir également une fonction d'activation non linéaire φ_1 : tanh ou sigmoïde par exemple. Ne pas oublier de gérer les biais.

- Implémenter une fonction *compute* permettant de propager les entrées vers la sortie, et deux fonctions *errorSample* et *errorDataset* qui calculent l'erreur du réseau sur un exemple, et sur l'ensemble de la base.
- Coder une fonction *backprop* permettant de rétropropager l'erreur à travers le réseau
- Mettre en œuvre l'algo de rétropropagation du gradient afin d'itérer l'apprentissage sur la base. Visualiser l'erreur sur la base d'apprentissage en fonction des itérations.
- Trouver la meilleure combinaison de paramètres sur la base de validation, et donner les résultats finaux sur la base de test.

2.2 Améliorations possibles

Voici plusieurs propositions d'améliorations :

- **Apprentissage batch, et/ou Apprentissage stochastique**
- **Line search pour trouver une bonne valeur de pas**

2.3 Alternatives possibles

Si vous avez déjà codé un MLP, voici plusieurs alternatives :

- **Réseau RBF** : les fonctions d'activation sont des gaussiennes à la place de fonctions monotones croissantes. Le réseau est ainsi modélisant; il pert en capacités de discrimination, mais permet le rejet de distance. L'apprentissage se fait en deux temps : K-means pour régler les gaussiennes, puis un réglage des poids de la deuxième couche par moindres carrés.
- **Réseau à deux couches cachés / à nombreuses couches cachées**
- **Méthode du gradient conjugué pour obtenir le pas optimal**

3 Conclusions

Quel est votre score sur la base de test ? Avec quelle jeu d'hyperparamètre ? Comment faire mieux ?