Dossier de TIPE

Camille Chanial, Tristan Du Castel, Tristan Stérin

05/06/2014

Résumé

C'est soin taii

Table des matières

Introduction : position du problème

1 La justification mathématiques

2 L'algorithme de rétropropagation

2.1 Réaliser l'apprentissage : comment minimiser l'erreur, quelle erreur ?

On a donc vu que les réseaux de neurones présentent un modèle pertinent dans le cadre de la régression non linéaire.
Il faut désormais mettre en place l'algorithme qui va permettre de trouver, d'apprendre, la combinaison de poids optimales afin de réaliser l'approximation désirée.
On se donne une base d'apprentissage :

$$B = \{(p_1, q_1), \ldots, (p_n, q_n)\} \quad avec : \quad (p_i, q_i) \in A^a \times B^b$$

Où A et B sont les espaces de départ et d'arrivée et a et b leur dimension respective.
Si on note $f : A \to B$ la fonction du réseau (variable au cours de l'apprentissage), notre but est d'avoir un algorithme tel qu'en un nombre fini d'itérations :

$$\forall i, \quad f(p_i) = q_i$$

On va procéder à cet apprentissage par cycle de présentation de tous les couples de la base au réseau.
Lors d'un cycle, on présente chaque couple au réseau, pour le ième couple on calcule le vecteur erreur instantanée $\vec{e} = q_i - f(p_i)$.
On se donne comme critère de "bon apprentissage" de notre réseau, comme indice de performance l'erreur quadratique moyenne :

$$F(p_i) = E[\,{}^t\vec{e} * \vec{e}]$$

L'espérance en question n'étant pas calculable a priori, on l'approxime l'indice de performance par l'erreur instantanée :

$$\hat{F}(p_i) =^t \vec{e} * \vec{e}$$

Ainsi notre but est de miniser la fonction $\hat{F}$. On utilise pour cela la méthode de descente de gradient telle que :

$$\Delta\omega_{i,j}^k = -\frac{\partial \hat{F}}{\partial \omega_{i,j}^k}$$

Avec $\omega_{i,j}^k$ le $j^{ième}$ poids du $i^{ième}$ neurone de la $k^{ième}$ couche, η le taux d'apprentissage.
Le sujet de cet exposé n'étant pas la descente de gradient, on introduira très briévement à l'oral les notions permettant de comprendre la

suite.
Calculer les dérivées partielles de $\hat{F}$ n'est pas un problème simple.
On expliquera à l'oral les principales étapes de ce calcul.
On donne ici uniquement l'expression final qui jsutifie l'algorithme proposé par la suite.

On introduit une quantité intermédiraire appelée sensibilité, associée à chaque couche, définie vectoriellement par :

$$s^k = \frac{\partial \hat{F}}{\partial n^k}$$

s^k est la sensibilité de la couche k, et n^k le vecteur des niveaux d'activation de la $k^{ième}$ couche.
On a : $\Delta W^k = -\eta s^k * {}^t a^{k-1}$
Avec W^k la matrice des poids de la $k^{ième}$ couche et a^k le vecteur sortie de la $k^{ième}$ couche.
Les sensibilités se calculent par récurrence (on ne donne pas la formule ici, on en parlera cependant à l'oral) d'où le nom de *back propagation* : l'information doit se transmettre de la dernière couche vers la première pour calculer les sensibilités et non pas juste de la première vers la dernière comme lorsqu'on calcule la sortie associée à une entrée.

2.2 L'algorithme d'entraînement

L'algorithme d'entraînement est donc le suivant :

1. Initialiser tous les poids du réseau à des valeurs aléatoires.
2. Pour chaque association (p_i, q_i) dans la base d'apprentissage :
   — Propager les entrées p_i vers l'avant
   — Rétropropager les sensibilités
   — Mettre à jour les poids
3. Si le critère d'arrêt est atteint, stop
4. Recommencer à l'étape 2

On appelle perceptron multicouche un réseau de neurones capable de réaliser un apprentissage à travers des algorithmes comparables (parfois plus subtils) à celui de *back-propagation* proposé ici.

De très nombreuses questions pratiques se posent alors : quel critère d'arrêt prendre ? Comment initialiser le réseau ? Comment adapter les entrées aux fonctions de transfert, ici principalement sigmoïde ?
Ces questions sont traitées dans la partie suivante.

3 L'application pratique : aspects techniques et résultats

3.1 Considérations pratiques quant à l'apprentissage

3.1.1 Entraîner le réseau

3.1.2 Quelques mots d'implémentation

3.2 Application : le problème des vélibs

3.2.1 Le data-mining et la base de données

3.2.2 Définition du perceptron utilisé

3.3 Analyse des résultats

3.3.1 Résultats

3.3.2 Limites et potentialités