# 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, $\eta$ 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.

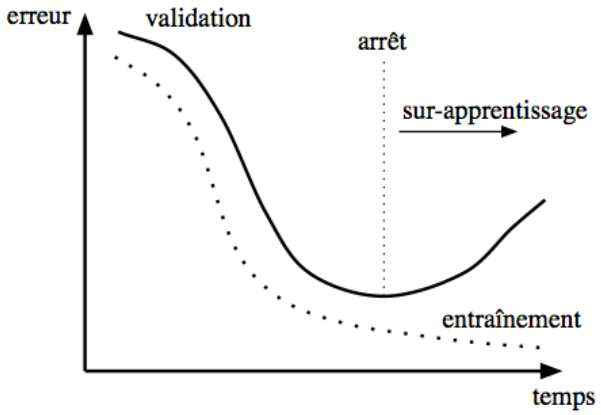
# 3 L'application pratique : aspects techniques et résultats

## 3.1 Considérations pratiques quant à l'apprentissage

### 3.1.1 Entraîner le réseau

Pour réaliser l'apprentissage pratique on doit tout d'abord normaliser les données d'entrée et de sortie.
Les contraintes de normalisation sont données par les fonctions de transfert utilisées. Ici des sigmoides $\mathbb{R} \to [0; 1]$
En pratique la convergence vers les asymptotes étant très rapide on normalise l'entrée dans $[-1, 1]$
Une autre considération pratique dont on doit tenir compte lorsqu'on entraîne un PMC concerne le phénomène de saturation des neurones où, sous certaines conditions, les neurones peuvent contrairement à toute fin pratique cesser d'apprendre tellement leur convergence devient lente. Pour éviter cela on normalise la sortie dans $[0.05, 0.95]$. On pourra détailler à l'oral.

Il faut de plus se donner un critère d'arrêt satisfaisant : quand est ce que l'on juge que le réseau à suffisament appris ?
Il faut pour cela considérer le phénomène de sur-apprentissage. Il peut arriver que le réseau est si bien appris un échantillon d'entrée qu'il devienne sensible au bruit spécifique à cet échantillon et qu'il perde son aptitude à prédire d'autres sorties.
Pour éviter cela, on se donne deux échantillons : un échantillon d'apprentissage et un échantillon de validation. À la fin de chaque cycle d'apprentissage on présente au réseau les données de validation. Le critère d'arrêt consiste alors à stopper l'apprentissage lorsque l'indice de performance calculé sur les données de validation cesse de s'améliorer pendant plusieurs périodes d'entraînement. Il s'agit d'une technique dite de *cross-validation*.



### 3.1.2 Quelques mots d'implémentation

La programmation orientée objet a été très pratique dans l'implémentation du réseau.
Trois classes ont été utilisées : Neurone, Couche, Reseau. On comprend le lien entre elles, un objet Neurone est élémentaire (contient notamment ses poids), une couche contient des neurones et un réseau des couches.

Python a d'abord été utilisé pour sa simplicité. Toutefois, l'apprentissage requérant un temps de calcul très important (voir résultats), une implémentation en *c++* a du être réalisée.

## 3.2 Application : le problème des vélibs

### 3.2.1 Définition du perceptron utilisé

Nous souhaitions d'abord modéliser l'ensemble de la consommation de vélib parisienne mais nous avons vu nos ambitions à la baisse, il y'a plus de 900 stations de velib. En terme de calcul c'est gigantesque.
Nous proposons ici la modélisation de la consommation par rapport à une seule station au comportement rationnel (modélisable) : il s'agit d'une station en face de Jussieu donc au comportement rythmé, dans les jours de semaine par la vie des étudiants.
Ainsi on se limite aux jours de la semaine sans les week end. L'entrée de notre réseau est un quadruplet qui contient dans l'ordre : l'heure de la journée, un booleen qui indique si l'on est à l'heure ou à la demi heure, la pluviométrie correspondante à cette heure et la température. Éléments qui nous paraissent intuitivement être significatif dans la consommation de vélib.

L'approximation universelle ne recquiert qu'une seule couche cachée. On prend donc une seule couche cachée avec, arbitrairement 5 neurones. En sortie on souhaite modéliser le nombre de vélos présent et le nombre de stand libres (la somme doit être constante). Donc deux neurones sur la couche de sortie.

### 3.2.2 Le data-mining et la base de données

Un autre problème pratique est le fait de pouvoir avoir des données d'entraînement. Il se trouve que les données des velibs sont en open data, on peut avoir à tout instant le nombre de vélos restants dans chaque station de paris.
Il a fallu en parallèle relever la météo associée à ces requêtes.
Tout cela a été fait en python (urllib) en utilisant les api de jcd et d'un organisme libre de météo.

Les données ont été stockées dans la base de données suivant le format suivant :

Le programme du perceptron communique directement par la base de donnée via sqlite (module sqlite3 python, équivalent en cpp).

## 3.3 Analyse des résultats

Partie complétée pour mardi.

### 3.3.1 Résultats

### 3.3.2 Limites et potentialités