Dossier de TIPE
Velib' et Réseaux de neurones

Camille Chanial, Tristan Du Castel, Tristan Stérin

05/06/2014

Table des matières

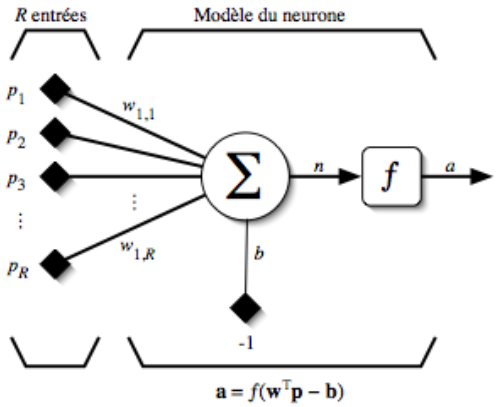
Introduction : position du problème

Comment modéliser l'évolution du nombre de Velib' disponibles à une station donnée ? Plus largement, comment modéliser la dynamique de répartition des Velib' ? Un tel problème ne saurait suivre une loi entièrement déterministe. Il s'agit donc de mettre en place un système capable de prédire le nombre de Velib' disponibles à une borne en fonction de divers paramètres explicatifs en se basant uniquement sur des données empiriques et donc d'effectuer une régression non linéaire qui relie les paramètres explicatifs d'entrée au résultat souhaité. Une telle modélisation s'appelle une régression non linéaire *boîte noire*. Les réseaux de neurones artificiels, que l'on nommera simplement dans la suite réseaux de neurones, apparaissent comme un outil pertinent pour la résolution de ce type de problème. Ils constituent un puissant outil de prédiction ayant déjà fait ses preuves dans de nombreux domaines (finance, distribution d'électricité...).
Dans un premier temps, nous avons décidé de nous concentrer sur une unique station et d'étudier l'influence de l'heure de la journée et des conditions météorologiques sur le nombre de Velib' disponible dans cette station. La première partie de cet exposé est consacrée à la définition des réseaux de neurones, à la position du formalisme mathématique et à la mise en évidence de la pertinence des réseaux de neurones pour la résolution du problème. Ensuite, nous décrivons le fonctionnement détaillé du type de réseau de neurones que nous avons utilisé : le *perceptron multicouche*. La troisième partie décrit la démarche suivie, du codage d'un réseau de neurone à l'interprétation des résultats en passant par la récupération des données utiles.

1 Les réseaux de neurones : fonctionnement et intérêt

1.1 Le neurone formel

Le neurone formel est l'élément de base d'un réseau de neurones. Il est schématisé ci-dessous.



Le neurone formel effectue une combinaison linéaire de ses entrées dont les scalaires sont appelés les poids à laquelle il retranche le biais b. À ce résultat, noté n il applique une fonction f appelée fonction d'activation. Si l'on note $p = (p_1, \ldots, p_R)$ le vecteur des entrées du neurone et $w = (w_1, \ldots, w_r)$ le vecteurs des poids, la sortie a du neurone s'écrit :

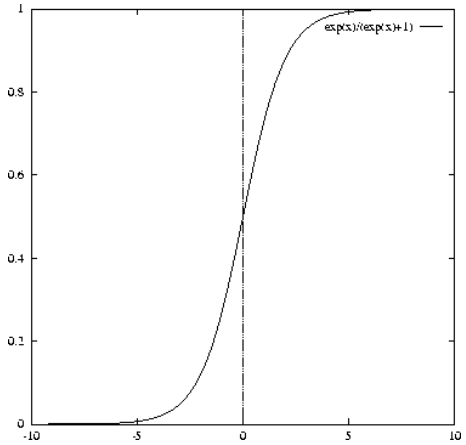
$$a = f({}^t wp - b)$$

Une fonction d'activation f est une fonction continue sur R telle que :

$$\begin{cases} \lim\limits_{x \to -\infty} f(x) = 0 \\ \lim\limits_{x \to +\infty} f(x) = 1 \end{cases}$$

Les fonctions d'activation usuelles sont la fonction seuil et la fonction sigmoïde. Dans toute la suite nous allons prendre comme fonction d'activation la fonction sigmoïde définie par :

$$\forall x \in \mathbb{R}, f(x) = \frac{1}{1 - \exp(-x)}$$

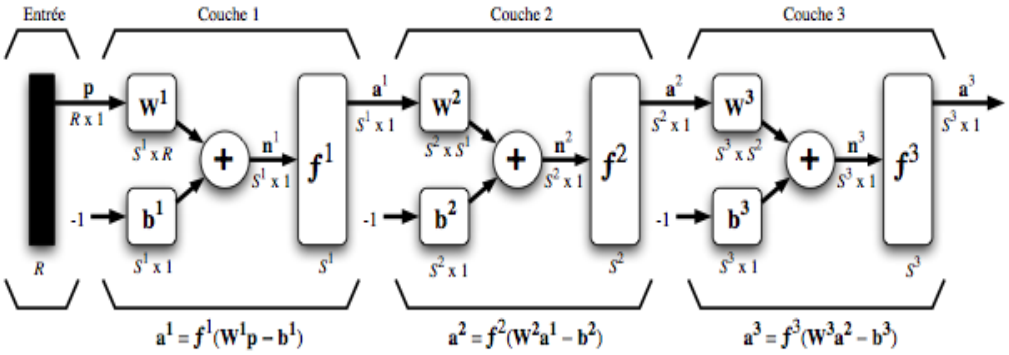


Une analogie avec les réseaux de neurones humains montre que les poids représentent l'efficacité d'une connexion synaptique et mesure donc l'influence d'une entrée du neurone sur sa sortie. Le biais correspond au seuil d'activation d'un neurone : si la combinaison linéaire ${}^t wp$ est plus petite que b, l'argument de la fonction d'activation sera négatif. Dans le cas extrême de la fonction d'activation seuil, la sortie sera alors nulle.

1.2 L'architecture d'un réseau

Construire un réseau de neurones, c'est agencer les différents neurones formels et définir les connections entre eux. L'architecture d'un réseau de neurones présente trois caractéristiques principales :

- le nombre de couches de neurones : une couche est un ensemble de neurones tous connectés aux mêmes entrées. On appelle couche de sortie la dernière couche et couches cachées celles qui la précèdent.
- le nombre de neurones par couche
- les connections entre les neurones



1.3 Apprentissage

L'une des particularités des réseaux de neurones est de s'adapter aux stimuli qu'il reçoit en entrée et de modifier son comportement pour correspondre au comportement désiré.
Le principe d'un apprentissage et de soumettre au neurones un ensemble d'entrées et de sorties désirées appelées base d'apprentissage et de le modifier afin que les sorties délivrées par le réseau correspondent le mieux possible (notion volontairement floue qui sera explicitée dans la deuxième partie) aux sorties désirées.
L'apprentissage se fait uniquement en modifiant les poids entre les différents neurones.

1.4 Intérêt des réseaux de neurones : l'approximation universelle

Dans le cadre de notre problématique, les réseaux de neurones constituent un outil puissant. En effet, ils sont capables, sous certaines conditions, d'approximer n'importe quelle fonction. Ainsi, sans connaître la relation déterministe entre les paramètres que nous considérons explicatifs (*entrées*) et le nombre de vélos à une borne (*sortie*), le réseau de neurones est capable de modéliser le lien entre les entrées et les sorties, opérant ainsi à une régression non linéaire. C'est la propriété d'approximation universelle.

Théorème 1 (Approximation universelle) *Toute fonction bornée suffisamment régulière peut être approchée uniformément avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini possédant tous la même fonction d'activation.*

Prouver ce théorème revient à prouver que toute fonction pouvant être approximée avec une précision ϵ par des fonctions en escalier peut être approximée par une fonction d'activation (typiquement une sigmoïde) avec une précision de 4ϵ. Le détail de cette preuve essentiellement technique sera fait à l'oral.

Il reste désormais à savoir comment parvenir à approximer la fonction recherchée, et donc à savoir comment procéder à l'apprentissage. C'est l'objet de la deuxième partie.

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 finale qui justifie l'algorithme proposé par la suite.

On introduit une quantité intermédiaire 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 (PMC) 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 d'activation, ici principalement sigmoïde ?
Ces questions sont traitées dans la partie suivante.

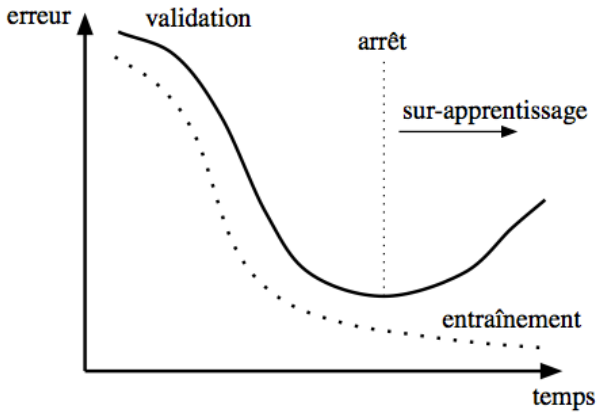
3 L'application pratique : aspects techniques et résultats

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 d'activation utilisées. Ici des sigmoïdes $\mathbb{R} \to [0; 1]$
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 ait 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 Vélib'. 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 significatifs 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 présents et le nombre de stands 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 Vélib' 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 JCDecaux 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