MASTER : Traitement de l'Information et Exploitation des Données
Fouad Badran, Cécile Mallet, Awa Niang, Charles Sorror, Sylvie Thiria

# TPB01 : Régression par Perceptron Multicouches

*Pour la réalisation de ce TP, on utilise le module* **triedpy**. Il sera donc nécessaire de s'assurer qu'il pourra bien être importé par les fonctions fournies.

## I - Les Objectifs

Le but de ce TP est de mettre en œuvre une méthodologie constructive pour déterminer une fonction de régression à l'aide de perceptrons multicouches. Ce TP comporte 2 Parties :

- Un PMC réalisant une régression à partir de données brutes simulées pour approximer l'espérance.
- Une approximation de la variance de données brutes également effectuée par une régression à l'aide d'un PMC.

==========

*Le rapport de TP devra être synthétique. Il doit montrer la démarche suivie, et ne faire apparaître que les résultats nécessaires. Il s'agit de quantifier les résultats tout en rédigeant un rapport qui les analyse et les commente. Les paramètres utilisés devront être indiqués, Les graphiques des expériences doivent être insérés dans le rapport. Les résultats présentés devront être analysés et commentés.*

## II - Les Données

Le T.P se fera sur données simulées. La manière dont on génèrera les données doit permettre d'illustrer la méthode et les résultats auxquels on peut parvenir.

Dans le fichier **TPB01_methodes.py**, on trouve 2 fonctions **schioler** et **silverman** mises à votre disposition. Elles permettent de simuler des données bruitées autour des deux fonctions « schioler » **(schiolerClean)** et « silverman » **(silverClean)**. La fonction générée par **schioler** est plus simple que celle générée à l'aide de **silverman**.

# III - Eléments pour le déroulement du TP

## Autres programmes mis à disposition :

**LittleDemo.py** est un petit script simple qui montre une utilisation basique de différentes fonctions issues du sous-module **trieddeep** du module **triedpy**. Autrement dit le fichier **trieddeep.py** contenu dans le répertoire **triedpy** contient des fonctions permettant l'apprentissage et l'évaluation de réseaux de neurone. On présente succinctement celles utilisées ci-dessous :

**pmcinit** : Permet de définir une architecture de perceptron multicouches et d'initialiser les poids. Les nombres d'entrées et de sorties sont fixés en fonction de la dimension des formes d'apprentissage qui sont utilisées, les biais sont ajoutés automatiquement. Les poids initiaux sont stockés dans les matrices W1 (poids entre couche d'entrée et couche cachée) et W2 (poids entre couche cachée et couche de sortie)

**pmctrain** : Permet d'effectuer l'apprentissage pour un ensemble d'apprentissage (X,Y) donné en utilisant une version de l'algorithme de rétropropagation du gradient. Il conviendra de consulter ce code afin de prendre connaissance des paramètres qu'il utilise.
L'algorithme s'arrête selon 3 critères possibles :

- le nombre maximum d'itération (**nbitemax**) a été effectué.
- les poids ne changent presque plus (variabilité<seuil).
- Si des données de validation sont passées, le programme s'arrête après un certain nombre d'itérations au-delà du minimum de l'erreur en validation (**ntfpamin*tfp**).

Lorsque des données de validation sont passées, les poids (W1, W2) retournés sont ceux du minimum de la validation, et les poids obtenus en fin de procédure sont stockés dans le fichier **pmcendwei.npy**, sinon ils sont retournés (sans être stockés).
Ce programme comporte pas mal de paramètre dont une grande partie ont des valeurs par défaut. Ainsi, pour ces différents paramètres, il est possible de garder la valeur existante ou de la changer. Ici, nous vous proposons les valeurs de paramètres suivants : **F1='tah', F2='lin', nbitemax=10000, dfreq=10, tfp=10, ntfpamin=200**. Pour les autres paramètres, on se satisfera des valeurs par défaut.

**pmcout** : Permet de calculer les valeurs de la régression effectuée par le perceptron multicouches [W1,W2] aux points X.

**errors** : Est utilisés pour calculer les erreurs.

## IV - 1ère Partie du TP : Régression : Approximation de l'espérance

### Travail à réaliser

On effectuera cette 1ère partie de TP uniquement en utilisant la procédure **schioler** définie de la façon suivante :

- Sur les abscisses les points suivent une distribution uniforme sur ]-2, 2[.
- $y = f(x) + delta$ où :

$$f(x)\begin{cases} = sin(\pi x) & sur\ ]-1, 1[ \\ = 0 & sur\ ]-2,-1] \cup [1, 2[ \end{cases}$$

et *delta* est un bruit qui suit une distribution normale $\mathcal{N}(0, \sigma^2)$ avec l'écart type $\sigma=0{,}2$.

1°) Simuler un ensemble de 1000 points à partir duquel vous devrez constituer un ensemble d'Apprentissage en choisissant un point sur deux, et un ensemble de Validation en prenant les 500 autres points restant. Prévoir par ailleurs un autre ensemble de 500 points pour le Test.
Présenter 2 figures :
- La première avec, dans un même repère, les données d'apprentissage et de validation, dans des couleurs différentes.
- La seconde pour y représenter les données de Test.

2°) Trouver une architecture de PMC optimale en faisant croître le nombre **m** de neurones sur la couche cachée. On pourra partir de **m=1** et se limiter par exemple à un maximum de **9** neurones en progressant par pas de **2**. On déterminera visuellement cette architecture en observant les courbes de régression obtenues pour chacune d'elle. Pour cela, vous devrez tracer les régressions obtenues pour chacune des valeurs de **m**. Vous pouvez faire une seule figure qui devra contenir les données d'apprentissage, les courbes de régression obtenues ainsi que celle de la vraie fonction (chacune dans des couleurs différentes).
Indication : Pour tracer les courbes vous aurez besoin de définir suffisamment de points d'abscisses répartis régulièrement dans l'intervalle de définition de la fonction.

3°) Calculer pour chaque ensemble (apprentissage, validation et test) la RMS ( $\sqrt{\frac{1}{N}\sum_{i=1:N}\left(Y_{i\,calculé} - Y_{i\,désiré}\right)^2}$ ) (fonction à utiliser : **errors**). Présenter sur une figure (dans un seul repère) ces erreurs en fonction du nombre **m** de cellules cachées. Mentionner également ces erreurs en clair dans un tableau. Vérifier si l'erreur en généralisation est minimale pour l'architecture que vous aurez choisie à la question 2° précédente.

4°) Générer de nouveaux ensembles d'apprentissage et validation en prenant un point sur n avec n = 3, 4, 5, 10, 30, 100. (en gardant toujours le même l'ensemble de Test que précédemment.)
Déterminer à chaque fois l'architecture optimale. Présenter les mêmes éléments (figures et tableau) que ceux des questions 2° et 3°. Que remarquez-vous ?

5°) On va utiliser des ensembles d'apprentissage et de validation dont l'échantillonnage est irrégulier c'est-à-dire dont certains intervalles de la fonction ne sont pas représentés :

- 1er cas : [-0.75 -0.25] U [1 1.25 ]
- 2ème cas : [-0.75 -0.25] U [1 2.00 ]

(fonction à utiliser : **where** du module **numpy**)

On continuera d'utiliser par contre le même ensemble Test que précédemment.
Pour chaque cas :
- Présenter une figure avec les données d'apprentissage et de validation.
- Effectuer l'apprentissage (en faisant varier m comme au 2°). Présenter les mêmes éléments (figures et tableau) que ceux des questions précédentes.

Pour l'ensemble de Test on calculera de plus l'erreur intervalle par intervalle.
Que remarquez-vous ?

Note : Pour toutes les figures qui seront présentées n'oublier pas de les habiller des éléments que vous jugerez nécessaires à leur compréhension (titre, légende, ...).

## V - 2ème Partie du TP : Régression : Approximation de la Variance

Cette partie du TP propose de déterminer des intervalles de confiance en estimant la variance associée à chaque observation. Les données brutes utilisées devront être simulées à partir de la procédure **silverman**.

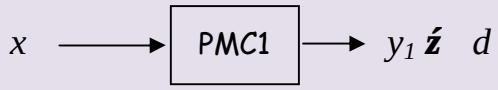
- Sur les abscisses les points suivent une distribution uniforme sur $]-0.25,\ 1[$ .
- $y = f(x) + delta(x)$ où :

$$f(x) \begin{cases} = sin(2\pi(1-x)^2) & \text{sur } ]0,\ 1[ \\ = 0 & \text{sur } ]-0.25,\ 0] \end{cases}$$

$delta(x)$ est un bruit qui suit un distribution normale $\mathcal{N}(0, sigma(x)^2)$ avec la variance $sigma(x)^2$ $\begin{cases} = 0.0025 & si\ x <= 0.05 \\ = x^2 & si\ x > 0.05 \end{cases}$

**Principe :** La méthode proposée ici pour l'estimation de la variance est très simple.

♣) Un premier PMC (PMC1) est optimisé pour modéliser la fonction $y=f(x)$ à partir d'une base d'apprentissage $\{(x_i,d_i)\ i=1,N\}$. Il s'agit d'une simple régression à l'instar du script **LittleDemo**.



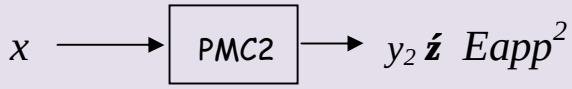

Après apprentissage, la sortie $y$ de ce premier réseau donne une estimation de l'espérance de $d/x$ : $y \approx E[d/x]$. L'erreur (noté $Eapp$) commise par ce 1er réseau est :

$$Eapp = (d/x \text{ ï } y) \approx (d/x \text{ ï } E[d/x])$$

♣) Pour chaque valeur $x$, un second réseau (PMC2) va permettre d'approximer la variance des données autour de la moyenne. Cette variance s'écrit :

$$Var(d/x) = E[(d/x \text{ ï } E[d/x])^2] \approx E[Eapp^2/x].$$

On voit donc que les sorties de ce second réseau devront approximer l'erreur quadratique du 1er réseau (qui représente la variance empirique sous l'hypothèse que le PMC1 est un bon estimateur de la moyenne conditionnelle) :




Remarque : Cette méthode d'estimation de la variance est sous-optimale car la moyenne estimée par PMC1 ne tiens pas compte de la variance. Il existe des architectures spécifiques et des méthodes d'apprentissage complète (comme celle vue en cours) qui permettent un apprentissage simultané de $E(d/x)$et $Var(d/x)$ qui sont alors deux sorties d'un même réseau pour lequel la fonction de coût est définie de manière ad hoc (maximum de vraisemblance).

## Travail à faire :

Dans un 1er temps, on utilisera un ensemble d'Apprentissage de taille Napp=1000 données (simulées avec la fonction **silverman**). Il conviendra de paramétrer un nombre d'itérations d'apprentissages suffisant.

1. - Déterminer PMC1 (5 neurones cachés devraient suffire). Visualiser ensuite dans le même repère :
   - l'ensemble des données
   - la fonction de régression $y_1$ estimé par PMC1 que l'on pourra comparer à
   - la courbe théorique (c'est-à-dire la courbe de la vraie fonction sans ajout de bruit).

2. - Identifier la ligne de code de **silverman** qui simule la fonction de la variance du bruit.
   (Remarque : Avec la fonction **randn** du module **numpy.random**, pour avoir des données N(0, $\sigma^2$) (de variance $\sigma^2$)
   il faut multiplier **randn** par l'écart type $\sigma$).

3. - Déterminer PMC2 en choisissant un nombre de neurones cachés approprié. Utiliser une fonction de sortie exponentielle (**'exp'**) pour s'assurer que les valeurs des sorties sont positives.
Comme pour la question 1, on vous demande de présenter dans le même repère :
   - les données d'apprentissage de la variance empirique que vous avez utilisées,
   - la fonction de régression $y_2$ estimé par PMC2 (en sortie) que l'on pourra comparer à la courbe théorique correspondant à la variance théorique du bruit identifiée à la question 2.

4. - On représentera sur un même graphique:
   - Les mêmes éléments qu'à la question 1 plus :
   - Les courbes d'encadrement à plus ou moins deux écarts types
     - avec la variance estimée par PMC2 et qui correspondent à :

$$y_1 \pm 2\sqrt{y_2} \quad \text{et à} \quad y_1 + 2\sqrt{y_2}$$

     - mais également avec la variance théorique pour permettre la comparaison.

   Vérifier si plus de 96,4% des données au moins se situent dans cet intervalle.

Dans un 2ème temps :

5. – On s'interroge sur la validité de la procédure dans le cas où l'on disposerait de moins de données. On recommence donc ces expériences avec Napp=50 puis Napp=200. Que peut-on en dire ?