CHAPITRE 8

Les mélanges d'experts et l'algorithme EM

8.1 Introduction

Quand on essaie de prévoir le comportement futur d'un système réel, on peut mettre en évidence le problème de l'apprentissage par coeur du modèle. Des solutions peuvent être apportées à ce problème en utilisant des **mélanges d'experts**. Cette architecture, appelée aussi **modèle modulaire** est composée de deux types de réseau, un contrôleur et des experts dont nous détaillerons le fonctionnement dans la suite du chapitre. Pour faire fonctionner ce modèle, nous avons besoin d'une fonction de coût et d'un algorithme, l'algorithme EM (Expectation-Maximisation).

8.2 La théorie du modèle modulaire

L'idée de ce type d'architecture est de partitionner une tâche en plusieurs tâches indépendantes et plus simples. On ne sait pas quelle segmentation des motifs d'entrée est à considérer. C'est pourquoi les sous-problèmes et le partitionnement sont traités de façon simultanée. D'où l'utilisation de deux types de modules : les **experts** qui résolvent un sous-problème linéaire ou non et le **contrôleur** qui donne une partition de l'espace d'entrée en régions correspondant aux différents experts. Les experts rentrent en compétition lors de l'apprentissage des exemples d'entrées et le contrôleur sert de médiateur à cette compétition.

La théorie du modèle modulaire est basée sur la notion de probabilité conditionnelle. On va maintenant détailler les équations et l'algorithme d'optimisation [Man95, MW95a, MW95b].

On considère un couple entrée-sortie (x et y) et un mélange d'experts composé d'un contrôleur et de K experts. La figure 8.1 représente une telle architecture.

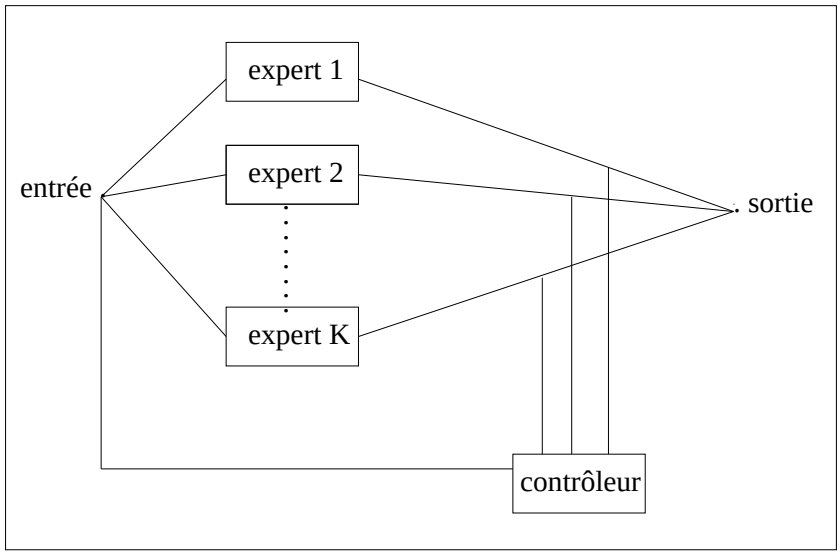

FIG. 8.1 – Schéma d'un modèle modulaire composé de K experts.

La sortie *Y* est une variable aléatoire à valeurs dans $\boldsymbol{R}$, conditionnée par une autre variable aléatoire *I* à valeurs dans {1, 2, ..., *K*}. La probabilité $P_x(I = j), j = 1, 2, ..., K$, qui dépend de l'entrée *x* (probabilité que *I = j* sachant *x*) donne la loi de *Y* et est calculée par le contrôleur.

La sortie *Y* s'écrit comme suit :

$$Y = f_j(x) + \eta_j \quad \text{si} \quad I = j$$

où $f_j(x)$ constitue la sortie du $j^{\text{ème}}$ expert et où η_j est une variable aléatoire centrée. $f_j(x)$ est alors l'espérance de *Y* si *I = j* sachant *x*.
La loi de *Y* peut s'écrire de la façon suivante, en utilisant la formule des probabilités conditionnelles :

$$P_x(Y = y) = \sum_{j=1}^{K} P_x(I = j) P_x(Y = y \mid I = j).$$

L'espérance de *Y* est :

$$E_x[y] = \sum_{j=1}^{K} P_x(I = j) f_j(x)$$

où $f_j(x)$ constitue la sortie du $j^{\text{ème}}$ expert.

L'objectif est de construire une architecture qui permette de modéliser le contrôleur et les experts. Le contrôleur est modélisé par une fonction qui dépend de paramètres θ_g (par exemple, θ_g représente les poids entre les neurones d'entrée et les neurones cachés ainsi que ceux entre les neurones cachés et les neurones de sortie si le contrôleur est un réseau du type perceptron). Cette fonction approxime la fonction : $x \mapsto (P_x(I = j))_{j=1,2,...,K}$. Le vecteur sortie du contrôleur est noté $(g_j(x, \theta_g))_{j=1,2,...,K}$.
De la même façon, on modélise l'expert *j* par une fonction qui dépend d'un paramètre θ_j, pour *j* = 1, 2, ..., K.
La sortie *Y* s'écrit alors :

$$Y = f_j(x, \theta_j) + \epsilon_j \quad \text{si} \quad I = j$$

où $f_j(x, \theta_j)$ est la sortie du $j^{\text{ème}}$ expert et ϵ_j un bruit centré de variance σ_j^2.
On fait l'hypothèse que les bruits $(\epsilon_j)_{j=1,2,...,K}$ sont gaussiens ($\epsilon_j \sim N(0, \sigma_j^2)$).

Le modèle est maintenant mis en place ; il ne reste plus qu'à estimer l'ensemble des paramètres $\Theta = (\theta_g, \theta_1, \theta_2, ..., \theta_K, \sigma_1^2, \sigma_2^2, ..., \sigma_K^2)$ en utilisant la méthode du maximum de vraisemblance.

Remarque : Soit H = $\{x_1, x_2, ..., x_n\}$ un ensemble de *n* échantillons indépendants.
Soit Θ un vecteur composé des paramètres à estimer.
La fonction suivante, qui dépend de Θ, est appelée **vraisemblance** [DH73] de Θ par rapport à l'ensemble des échantillons :

$$P(H \mid \Theta) = \prod_{k=1}^{n} P(x_k \mid \Theta).$$

Le maximum de vraisemblance est par définition la valeur de Θ qui maximise $P(H \mid \Theta)$. Intuitivement, cela correspond à la valeur de Θ qui s'accorde le mieux avec les échantillons observés.

L'expression de la probabilité $P_x(Y = y \mid I = j)$ est la suivante :

$$P_x(Y = y \mid I = j)dy = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(y - f_j(x, \theta_j))^2}{2\sigma_j^2}\right) dy.$$

La loi de *Y* est alors :

$$P_x(Y = y)dy = \sum_{j=1}^{K} g_j(x, \theta_g) \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(y - f_j(x, \theta_j))^2}{2\sigma_j^2}\right) dy.$$

La prévision de *y*, notée $\widehat{y}(x)$ est :

$$\widehat{y}(x) = E_x[y] = \sum_{j=1}^{K} g_j(x, \theta_g) f_j(x, \theta_j).$$

On suppose maintenant que l'on dispose d'un échantillon de *N* couples d'entrée-sortie $(x^{(m)}, y^{(m)})$ pour $m = 1, 2, ..., N$. A chaque échantillon, on associe une variable aléatoire $I^{(m)}$ à valeurs dans $\{1, 2, ..., K\}$.
On pose $X = (x^{(m)})_{m=1,2,...,N}$ et $Y = (y^{(m)})_{m=1,2,...,N}$.

On peut alors calculer la vraisemblance :

$$\begin{aligned} L_X(Y, \Theta) &= \prod_{m=1}^{N} P_{x^{(m)}}\left(y^{(m)}\right) \\ &= \prod_{m=1}^{N} \sum_{j=1}^{K} g_j\left(x^{(m)}, \theta_g\right) P_{x^{(m)}}\left(Y = y^{(m)} \mid I^{(m)} = j\right) \\ &= \prod_{m=1}^{N} \sum_{j=1}^{K} \left[g_j(x^{(m)}, \theta_g) \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{\left(y^{(m)} - f_j(x^{(m)}, \theta_j)\right)^2}{2\sigma_j^2}\right) dy \right]. \end{aligned} \tag{8.1}$$

Maximiser la fonction précédente revient à minimiser la fonction de coût suivante, appelée log vraisemblance :

$$\begin{aligned} C(Y,\Theta,X) &= -\ln L_X(Y,\Theta) \\ &= \sum_{m=1}^{N} -\ln\left[\sum_{j=1}^{K} g_j(x^{(m)},\theta_g)\frac{1}{\sqrt{2\pi\sigma_j^2}}\exp\left(-\frac{\left(y^{(m)}-f_j(x^{(m)},\theta_j)\right)^2}{2\sigma_j^2}\right)\right] - \ln(dy)^N. \end{aligned} \tag{8.2}$$

Dans le paragraphe suivant, nous allons décrire comment en pratique nous pouvons construire une telle structure.

8.3 L'architecture du modèle modulaire

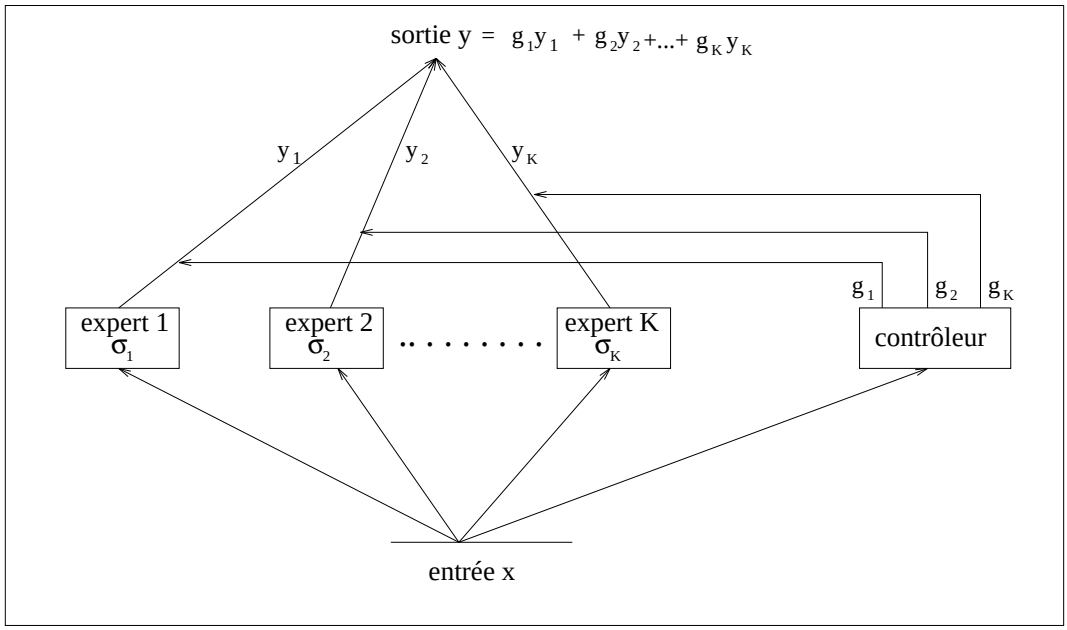

FIG. 8.2 – Architecture d'un modèle modulaire comprenant K experts.

Comme dit précédemment, l'architecture est composée de deux types de modules (figure 8.2). On peut donner à ces modules le même vecteur d'entrée ou partager l'information suivant les connaissances a priori du modèle.

Le rôle de chaque expert est de résoudre un problème dans une région de l'espace des entrées. C'est pendant l'apprentissage que le contrôleur définit ces régions et engendre des sorties $\left(g_j(x^{(m)},\theta_g)\right)_{1\leq j\leq K}$ positives et de somme égale à 1. Les paramètres du contrôleur sont modifiés au cours de l'apprentissage en fonction de la valeur de sortie des experts. On parle d'**apprentissage supervisé** pour les experts qui apprennent une valeur de sortie connue et d'**apprentissage non supervisé** pour le contrôleur quand il détermine le partitionnement (qui est inconnu).

Le principe de fonctionnement du contrôleur est le suivant : il est muni de *K* sorties. La $j^{\text{ème}}$ sortie représente la probabilité que le $j^{\text{ème}}$ expert soit activé connaissant l'entrée. Soit $(s_j)_{j=1,2,\ldots,K}$

le vecteur de sorties, calculé de manière classique (si le contrôleur est du type perceptron multi-couches). Pour que la somme des probabilités de sortie soit égale à 1, on utilise la transformation suivante, par la fonction "softmax" :

$$g_j(x^{(m)}, \theta_g) = \frac{\exp(s_j)}{\sum_{k=1}^{K} \exp(s_k)}, \forall j = 1, 2, ..., K.$$

On peut ainsi calculer la sortie totale du modèle :

$$\widehat{y}(x^{(m)}) = \sum_{j=1}^{K} g_j(x^{(m)}, \theta_g) f_j(x^{(m)}, \theta_j)$$

$$\widehat{y}(x^{(m)}) = \sum_{j=1}^{K} \frac{\exp(s_j)}{\sum_{k=1}^{K} \exp(s_k)} f_j(x^{(m)}, \theta_j).$$

Il reste maintenant à déterminer les mécanismes de modification des paramètres. Pour cela, nous allons utiliser l'algorithme EM.

8.4 L'algorithme EM (Expectation-Maximization)

La fonction de coût définie précédemment est difficile à minimiser. C'est pourquoi nous allons introduire des variables aléatoires dites cachées, représentant des partitions inconnues. Cette technique est appelée **algorithme EM, Expectation-Maximisation** et permet de simplifier la fonction de coût $C(Y, \Theta, X)$.

On fait les mêmes hypothèses que dans le paragraphe 4.2 : échantillon de *N* couples $\left(x^{(m)}, y^{(m)}\right)$ pour $m = 1, 2, ..., N$, *N* variables aléatoires $\left(I^{(m)}\right)_{m=1,2,...,N}$ dont la loi dépend de $x^{(m)}$. On construit *K* variables aléatoires cachées de somme égale à 1, $\left(J_j^{(m)}\right)_{j=1,2,...,K}$ telles que :

$$J_j^{(m)} = \begin{cases} 1 & \text{si} \quad I^{(m)} = j \\ 0 & \text{sinon} \end{cases}$$

Nous introduisons alors une nouvelle fonction dont l'opposé du logarithme népérien sera minimisé en passant par une étape d'estimation des variables cachées.

Si on pose $Y_{\text{cachées}} = \left(J_j^{(m)}\right)_{1 \leq j \leq K, 1 \leq m \leq N}$, la fonction que nous venons d'évoquer est la suivante :

$$L_2(Y, Y_{\text{cachées}}, \Theta, X) = \prod_{m=1}^{N} \prod_{j=1}^{K} \left[g_j(x^{(m)}, \theta_g) P_{x^{(m)}} \left(y^{(m)} \mid I^{(m)} = j \right) \right]^{J_j^{(m)}}.$$

Remarque : La distribution des variables cachées a été choisie de manière à s'accorder avec la distribution des données complètes $(Y, Y_{\text{cachées}})$. En effet :

$$\begin{aligned} \int P_X(Y, Y_{\text{cachées}})\, dY_{\text{cachées}} &= \int \prod_{m=1}^{N} \prod_{j=1}^{K} \left[g_j(x^{(m)}, \theta_g) P_{x^{(m)}} \left(y^{(m)} \mid I^{(m)} = j \right) \right]^{J_j^{(m)}} d\left(J_k^{(m)}\right)_{k=1,2,...,K} \\ &= \sum_{k=1}^{K} \prod_{m=1}^{N} \left[g_k(x^{(m)}, \theta_g) P_{x^{(m)}} \left(y^{(m)} \mid I^{(m)} = k \right) \right] \\ &= P_X(Y). \end{aligned} \tag{8.3}$$

La fonction $L_2(Y, Y_{\text{cachées}}, \Theta, X)$ ne peut pas être utilisée car on ne connaît pas la valeur des variables cachées. Le principe de l'algorithme EM est de remplacer ces variables par leurs espérances $h_j^{(m)} = h_j(x^{(m)}, y^{(m)}, \theta_g, \theta_1, ..., \theta_K, \sigma_1^2, ..., \sigma_K^2)$ qui sont calculées lors de l'**étape E** de l'algorithme :

$$\begin{aligned} h_j(x^{(m)}, y^{(m)}, \theta_g, \theta_1, ..., \theta_K, \sigma_1^2, ..., \sigma_K^2) &= E_{x^{(m)}}\left[J_j^{(m)} \mid y^{(m)}\right] = P_{x^{(m)}}\left(J_j^{(m)} = 1 \mid y^{(m)}\right) \\ &= P_{x^{(m)}}\left(I^{(m)} = j \mid y^{(m)}\right) \\ &= \frac{P_{x^{(m)}}\left(y^{(m)}, I^{(m)} = j\right)}{P_{x^{(m)}}\left(y^{(m)}\right)} \\ &= \frac{P_{x^{(m)}}\left(I^{(m)} = j\right) P_{x^{(m)}}\left(y^{(m)} \mid I^{(m)} = j\right)}{P_{x^{(m)}}\left(y^{(m)}\right)} \\ &= \frac{g_j(x^{(m)}, \theta_g) P_{x^{(m)}}\left(y^{(m)} \mid I^{(m)} = j\right)}{\sum_{k=1}^{K} g_k(x^{(m)}, \theta_g) P_{x^{(m)}}\left(y^{(m)} \mid I^{(m)} = k\right)}. \end{aligned} \tag{8.4}$$

En supposant que chaque expert a une distribution gaussienne, on peut écrire $h_j^{(m)}$ comme suit :

$$h_j^{(m)} = \frac{g_j(x^{(m)}, \theta_g) \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{\left(y^{(m)} - f_j(x^{(m)}, \theta_j)\right)^2}{2\sigma_j^2}\right)}{\sum_{k=1}^{K} g_k(x^{(m)}, \theta_g) \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{\left(y^{(m)} - f_k(x^{(m)}, \theta_k)\right)^2}{2\sigma_k^2}\right)}.$$

En posant $H = \left(h_j^{(m)}\right)_{1 \leq j \leq K, 1 \leq m \leq N}$, la fonction à minimiser est la suivante :

$$\begin{aligned} C_{EM}(Y, H, \Theta \mid X) &= -\ln(L_2(Y, Y_{\text{cachées}}, \Theta, X)) \\ &= -\sum_{m=1}^{N}\sum_{j=1}^{K} h_j^{(m)} \ln\left[g_j(x^{(m)}, \theta_g) P_{x^{(m)}}\left(y^{(m)} \mid I^{(m)} = j\right)\right] \\ &= -\sum_{m=1}^{N}\sum_{j=1}^{K} h_j^{(m)} \ln\left[g_j(x^{(m)}, \theta_g) \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{\left(y^{(m)} - f_j(x^{(m)}, \theta_j)\right)^2}{2\sigma_j^2}\right)\right] \\ &= -\sum_{m=1}^{N}\sum_{j=1}^{K} h_j^{(m)} \left[\ln\left(g_j(x^{(m)}, \theta_g)\right) - \frac{\left(y^{(m)} - f_j(x^{(m)}, \theta_j)\right)^2}{2\sigma_j^2} - \frac{1}{2}\ln\left(2\pi\sigma_j^2\right)\right]. \end{aligned} \tag{8.5}$$

L'**étape M** consiste à ajuster le paramètre $\Theta = (\theta_g, \theta_1, \theta_2, ..., \theta_K, \sigma_1^2, \sigma_2^2, ..., \sigma_K^2)$ de manière à minimiser la fonction précédente.

Remarque : Les deux fonctions introduites $C(Y, \Theta, X)$ et $C_{EM}(Y, H, \Theta \mid X)$ ne sont pas identiques ; cependant, il a été démontré qu'elles admettent toutes les deux un minimum local au même point (cf Annexe B).

Une itération de l'algorithme EM est constituée de l'étape E suivie de l'étape M.

Les nouvelles variances des experts se calculent en dérivant la fonction C_{EM} par rapport à σ_j^2 :

$$\frac{\partial C_{EM}}{\partial \sigma_j^2} = -\sum_{m=1}^{N} h_j^{(m)} \left[-\frac{1}{2\sigma_j^2} + \frac{1}{2(\sigma_j^2)^2} \left(y^{(m)} - f_j(x^{(m)}, \theta_j) \right)^2 \right]$$

$$\frac{\partial C_{EM}}{\partial \sigma_j^2} = 0 \Longrightarrow \sigma_j^2 = \frac{\sum_{m=1}^{N} h_j^{(m)} \left(y^{(m)} - f_j(x^{(m)}, \theta_j) \right)^2}{\sum_{m=1}^{N} h_j^{(m)}}.$$

Nous ne pouvons pas calculer explicitement les valeurs de θ_j tels que $\partial C_{EM}/\partial \theta_j = 0$. Nous allons donc utiliser des méthodes d'optimisation itératives, basées sur le gradient.

Dans la méthode de descente du gradient, nous avons besoin de connaître $\partial C_{EM}/\partial \theta_j$ pour l'expert $j = 1, 2, ..., K$ et $\partial C_{EM}/\partial \theta_g$ pour le contrôleur. Nous allons par conséquent calculer les gradients pour le motif m.

En ce qui concerne l'expert *j*, on a :

$$C_{EM}^{(m),j} = -h_j^{(m)} \left[\ln \left(g_j(x^{(m)}, \theta_g) \right) - \frac{\left(y^{(m)} - f_j(x^{(m)}, \theta_j) \right)^2}{2\sigma_j^2} - \frac{1}{2} \ln \left(2\pi\sigma_j^2 \right) \right].$$

Ainsi :

$$\frac{\partial C_{EM}^{(m),j}}{\partial \theta_j} = \frac{\partial C_{EM}^{(m),j}}{\partial f_j(x^{(m)}, \theta_j)} \frac{\partial f_j(x^{(m)}, \theta_j)}{\partial \theta_j}$$

avec

$$\frac{\partial C_{EM}^{(m),j}}{\partial f_j(x^{(m)}, \theta_j)} = -\frac{h_j^{(m)}}{\sigma_j^2} \left(y^{(m)} - f_j(x^{(m)}, \theta_j) \right).$$

On procède de la même manière pour le contrôleur. On obtient :

$$\frac{\partial C_{EM}^{(m)}}{\partial \theta_g} = \sum_{j=1}^{K} \frac{\partial C_{EM}^{(m)}}{\partial s_j} \frac{\partial s_j}{\partial \theta_g}$$

où

$$\frac{\partial C_{EM}^{(m)}}{\partial s_j} = -\left(h_j^{(m)} - g_j(x^{(m)}, \theta_g) \right).$$

En effet :

$$C_{EM}^{(m)} = -\sum_{j=1}^{K} h_j^{(m)} \left[\ln \left(g_j(x^{(m)}, \theta_g) \right) - \frac{\left(y^{(m)} - f_j(x^{(m)}, \theta_j) \right)^2}{2\sigma_j^2} - \frac{1}{2} \ln \left(2\pi\sigma_j^2 \right) \right].$$

On a ainsi :

$$\begin{aligned}
\frac{\partial C_{EM}^{(m)}}{\partial s_j} &= -\frac{\partial}{\partial s_j}\left(\sum_{k\neq j} h_k^{(m)} \ln\left(g_k(x^{(m)},\theta_g)\right) + h_j^{(m)} \ln\left(g_j(x^{(m)},\theta_g)\right)\right) \\
&= -\sum_{k\neq j} h_k^{(m)} \frac{\partial}{\partial s_j}\left(\ln \frac{exp(s_k)}{\sum_{i=1}^K exp(s_i)}\right) - h_j^{(m)} \frac{\partial}{\partial s_j}\left(\ln \frac{exp(s_j)}{\sum_{i=1}^K exp(s_i)}\right) \\
\frac{\partial C_{EM}^{(m)}}{\partial s_j} &= -\sum_{k\neq j} h_k^{(m)} \left(-\frac{1}{\sum_{i=1}^K exp(s_i)}\right) exp(s_j) - h_j^{(m)} \frac{1}{g_j(x^{(m)},\theta_g)} \frac{exp(s_j)\left(\sum_{i=1}^K exp(s_i)\right) - (exp(s_j)}{\left(\sum_{i=1}^K exp(s_i)\right)^2} \\
&= \sum_{k\neq j} h_k^{(m)} g_j(x^{(m)},\theta_g) - h_j^{(m)}\left(1 - g_j(x^{(m)},\theta_g)\right) \\
&= -\left(h_j^{(m)} - g_j(x^{(m)},\theta_g)\right).
\end{aligned} \tag{8.}$$

Il faut insister sur le fait que jusqu'à présent, nous sommes restés dans un contexte très général en ce qui concerne les deux types de modules.

Nous allons à partir de maintenant seulement, nous placer dans le cas où les experts et le contrôleur sont des réseaux du type perceptron multicouches. θ_j, $j = 1, 2, .., K$ représentent ainsi les poids de l'expert *j* et θ_g ceux du contrôleur. $\partial f_j(x^{(m)},\theta_j)/\partial\theta_j$ et $\partial s_j/\partial\theta_g$ se calculent alors de manière classique. C'est ce que nous détaillerons dans la suite.

Nous prendrons les mêmes notations que dans le chapitre 3, à savoir : a_i est l'entrée totale du neurone *i*, s_i est sa sortie, w_{ij} est la connexion entre le neurone *i* et le neurone *j* de la couche précédente et *f* est la fonction de transfert considérée.

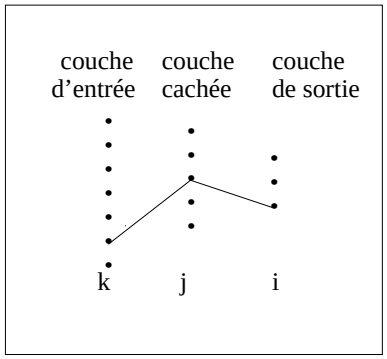

FIG. 8.3 – Indices utilisés pour le calcul des nouveaux poids.

Nous commencerons par calculer $\partial s_i/\partial w_{ij}$ (w_{ij} est le poids entre le neurone *i* de la couche de sortie et le neurone *j* de la couche cachée). On a :

$$\begin{aligned}
s_i &= f(a_i) \\
s_i &= f\left(\sum_{n\in\text{cachés}} w_{in} s_n\right).
\end{aligned} \tag{8.7}$$

Donc :

$$\frac{\partial s_i}{\partial w_{ij}} = s_j f'(a_i).$$

Calculons maintenant $\partial s_i / \partial w_{jk}$ où w_{jk} est le poids entre le neurone *j* de la couche cachée et le neurone *k* de la couche d'entrée. Comme précédemment, on sait que :

$$\begin{aligned} s_i &= f(a_i) \\ s_i &= f\left(\sum_{n \in \text{cachés}} w_{in} s_n\right) \\ s_i &= f\left(\sum_{n \in \text{cachés}} w_{in} f\left(\sum_{l \in \text{entrée}} w_{nl} e_l\right)\right) \end{aligned} \tag{8.8}$$

où e_l est l'entrée du neurone d'indice *l*. On en déduit donc :

$$\frac{\partial s_i}{\partial w_{jk}} = e_k f'(a_j) w_{ij} f'(a_i).$$

Une fois ces dérivées partielles connues, nous pouvons modifier les poids selon l'algorithme de rétropropagation du gradient, comme il est défini dans le premier chapitre.

Remarque : Des preuves de convergence de cet algorithme ont été établies [HSSW97, JX95] (cf Annexe B).

Donnons maintenant un exemple d'application de cet algorithme.

8.5 Application à une série simulée

Nous allons dans ce paragraphe tester l'algorithme EM sur une série créée artificiellement.

Notre expérience consiste en la prédiction de la série à un pas de temps, i.e connaissant la valeur de la série à l'instant t, on cherche à estimer sa valeur à l'instant $(t+1)$.

Le premier des deux processus est un processus déterministe, appelé série de McKay-Glass. Il est généré par une équation simple, mais évolue de façon chaotique.

Le deuxième processus est issu de la composition d'un processus autorégressif de degré 1 avec ajout d'un bruit gaussien centré de variance 0.1 et d'une tangente hyperbolique.

On décide que l'on passe d'un processus à l'autre grâce à un commutateur *c* qui passe de 0 à 1 ou de 1 à 0 avec une probabilité de 0.02 suivant une loi uniforme (i.e., le temps moyen de changement est environ de 50 pas).

La série étudiée est la suivante :

$$x_t = \begin{cases} (1-b)x_{t-1} + \frac{a x_{t-\tau}}{1+(x_{t-\tau})^{10}} & \text{si } c = 0 \text{ (processus 1)} \\ tanh(-1.2 x_{t-1} + \epsilon_t) & \text{si } c = 1 \text{ (processus 2)} \end{cases}$$

où ϵ_t est un bruit blanc gaussien centré de variance 0.1.

On choisira : $a = 0.2$, $b = 0.1$ et $\tau = 307$.

La série que l'on cherchera à apprendre est celle représentée sur la figure 8.4.

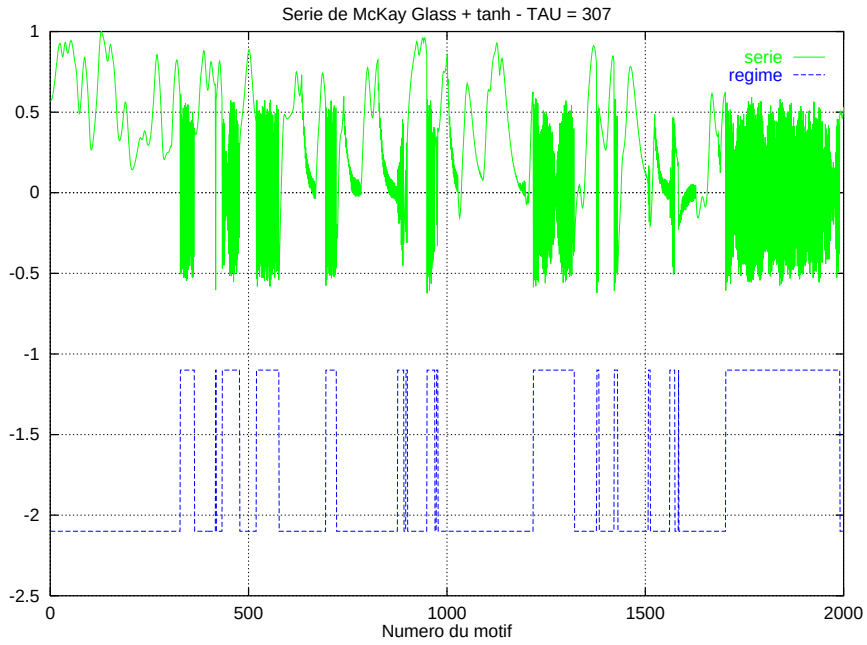

FIG. 8.4 – Série et régime associé utilisée comme base de données.

Quand on essaie d'apprendre chaque série séparément avec un perceptron, on constate qu'avec 5 entrées et 3 neurones cachés, la série peut être apprise avec une erreur moyenne de l'ordre de 0.03. Par contre quand on apprend la série entière avec le perceptron, même si on augmente considérablement le nombre de neurones cachés, le résultat obtenu est nettement moins bon ; néanmoins la série est bien apprise. On décide donc d'utiliser l'architecture modulaire neuronal.

L'architecture est constituée de trois réseaux de neurones du type perceptron multicouche : 2 experts et 1 contrôleur. Chaque expert possède 3 neurones d'entrée et 3 neurones cachés ; le contrôleur possède 5 neurones d'entrée et 10 neurones cachés.

Nous allons maintenant présenter les résultats obtenus : évolution du coût quadratique, évolution des sorties du contrôleur, évolution des variances.

L'évolution du coût quadratique est représentée sur la figure 8.5.

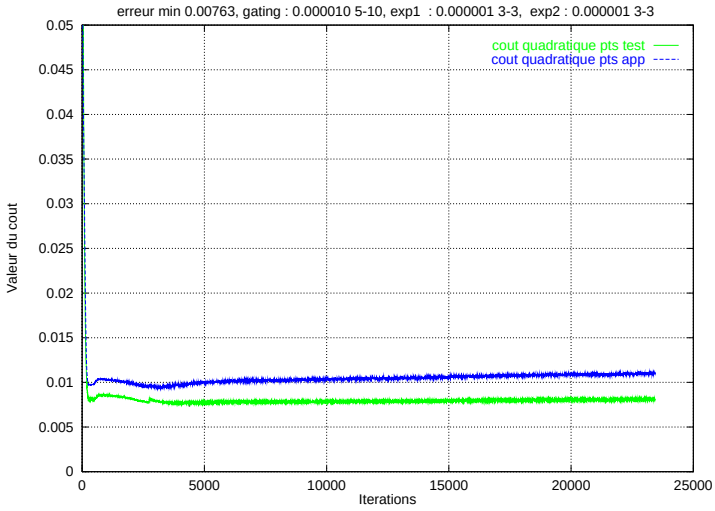

FIG. 8.5 – Evolution du coût quadratique pendant l'apprentissage.

Le coût quadratique obtenu en fin d'apprentissage est du même ordre que celui obtenu en utilisant le perceptron multicouches. Donc cette architecture n'apporte qu'une très faible amélioration au niveau de la prévision. On constate par ailleurs qu'il n'y a pratiquement aucun sur-apprentissage, autrement dit, l'erreur de généralisation n'augmente pas.

L'architecture utilisée permet de déterminer le comportement des experts grâce aux valeurs des sorties du contrôleur. C'est ce qui est représenté sur la courbe 8.6.

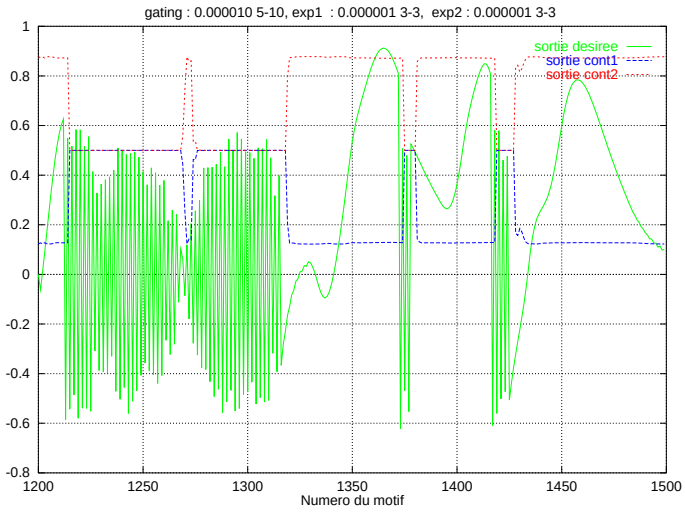

FIG. 8.6 – Répartition des sorties du contrôleur en fonction des points de la base de test.

Quand on introduit l'architecture modulaire avec deux experts, on constate que le contrôleur a bien séparé l'espace des entrées : l'un des experts apprend quasiment tout seul le processus 1 alors que le processus 2 est appris par les deux experts en même temps. Cependant, il est difficile d'interpréter les sorties du contrôleur car le réseau apprend une distribution et non un comportement. Pour pouvoir interpréter correctement les sorties du contrôleur, il faudrait être capable de donner précisément le rôle de chaque module.

Pour terminer le paragraphe, nous allons regarder l'évolution des variances de chaque expert.

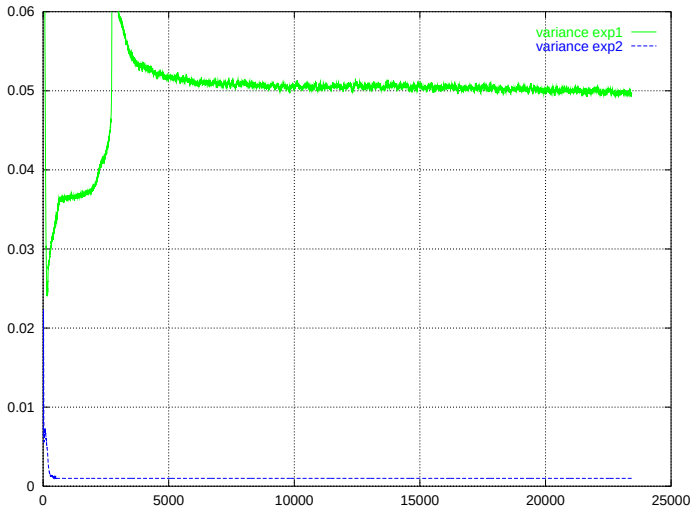

FIG. 8.7 – Evolution des variances de chaque expert durant l'apprentissage.

On peut noter grâce à la courbe précédente qu'il y a une phase de transition due à l'apprentissage. Cela se traduit par des variations importantes des variances et montre le lien étroit entre apprentissage et réglage des variances. Les variances ont bien convergé en fin d'apprentissage et leurs valeurs asymptotiques sont différentes.

Remarque : En utilisant un système à trois experts , les résultats obtenus pour les sorties du contrôleur sont moins significatifs.

8.6 Comparaison avec un algorithme qui utilise une fonction de coût quadratique

Dans ce paragraphe, on reprend les mêmes notations que précédemment, mais on va utiliser l'algorithme de rétropropagation du gradient comme il est défini dans le premier chapitre, i.e avec la fonction de coût quadratique.

La fonction de coût utilisée est la suivante :

$$C = \frac{1}{2} \sum_{m=1}^{N} \left(y^{(m)} - \sum_{i=1}^{K} g_i(x^{(m)}, \theta_g) f_i(x^{(m)}, \theta_i) \right)^2 .$$

Pour appliquer l'algorithme de descente du gradient, il faut calculer $\partial C / \partial \theta_j$ pour l'expert j, $j = 1, 2, ..., K$ et $\partial C / \partial \theta_g$ pour le contrôleur. Calculons ces dérivées partielles pour le motif m.

On pose :

$$C^{(m)} = \frac{1}{2}\left(y^{(m)} - \sum_{i=1}^{K} g_i(x^{(m)}, \theta_g) f_i(x^{(m)}, \theta_i)\right)^2 .$$

En ce qui concerne l'expert j :

$$\frac{\partial C^{(m)}}{\partial \theta_j} = \frac{\partial C^{(m)}}{\partial f_j(x^{(m)}, \theta_j)} \frac{\partial f_j(x^{(m)}, \theta_j)}{\partial \theta_j}$$

avec :

$$\frac{\partial C^{(m)}}{\partial f_j(x^{(m)}, \theta_j)} = g_j(x^{(m)}, \theta_g)\left(y^{(m)} - \sum_{i=1}^{K} g_i(x^{(m)}, \theta_g) f_i(x^{(m)}, \theta_i)\right).$$

Pour le contrôleur :

$$\begin{aligned} \frac{\partial C^{(m)}}{\partial \theta_g} &= \sum_{j=1}^{K} \frac{\partial C^{(m)}}{\partial g_j(x^{(m)}, \theta_g)} \frac{\partial g_j(x^{(m)}, \theta_g)}{\partial \theta_g} \\ &= \sum_{j=1}^{K} f_j(x^{(m)}, \theta_j)\left(y^{(m)} - \sum_{i=1}^{K} g_i(x^{(m)}, \theta_g) f_i(x^{(m)}, \theta_i)\right) \frac{\partial g_j(x^{(m)}, \theta_g)}{\partial s_j} \frac{\partial s_j}{\partial \theta_g} \end{aligned} \tag{8.9}$$

avec :

$$\frac{\partial g_j(x^{(m)}, \theta_g)}{\partial s_j} = \left(g_j(x^{(m)}, \theta_g) - g_j(x^{(m)}, \theta_g)^2\right).$$

Les dérivées partielles $\partial f_j(x^{(m)}, \theta_j)/\partial \theta_j$ et $\partial s_j / \partial \theta_g$ ont été calculées à la fin du paragraphe 4.4.

On applique ensuite la procédure de rétropropagation du gradient.

Maintenant, on reprend la série précédente et on applique l'algorithme qui vient d'être donné avec la même architecture : un contrôleur à 5 entrées et 10 neurones cachés, 2 experts avec chacun 3 entrées et 3 neurones cachés.

La série est aussi bien apprise qu'avec l'algorithme EM, mais les résultats sur les sorties du contrôleur sont moins significatifs. Ceux-ci sont représentés sur la courbe 8.8.

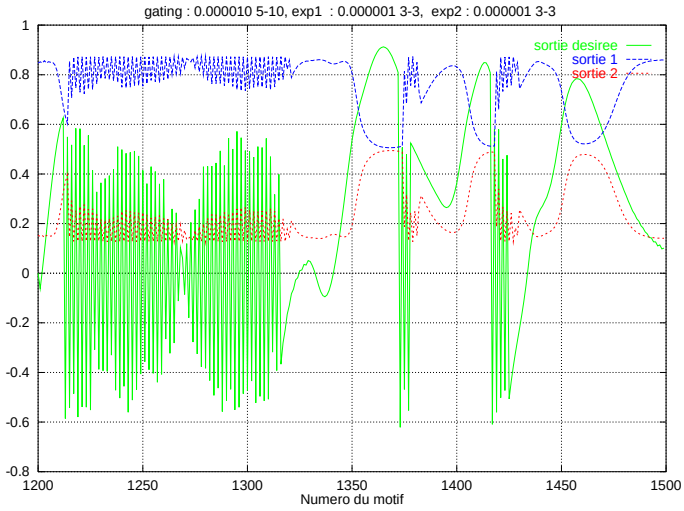

FIG. 8.8 – Répartition des sorties du contrôleur en fonction des entrées.

Le résultat obtenu sur les sorties du contrôleur montre qu'un expert apprend quasiment seul les points de la base de test représentés ici. Cependant, comme dit précédemment, l'interprétation des sorties est délicate : les réseaux apprennent une distribution de probabilité et non un comportement.

Cette étude nous permet de conclure que l'algorithme EM, contrairement à l'algorithme de rétropropagation avec une fonction de coût quadratique essaie de prévoir quel expert va être utilisé pour apprendre un motif donné.

8.7 Convergence de l'algorithme EM

Nous allons donner dans cette annexe la démonstration de la convergence de l'algorithme EM dans le cas général vers un extremum local. En effet, EM ne garantit pas la convergence vers un extremum global.

L'énoncé du problème est le suivant : soit μ une mesure et f une fonction que l'on suppose positive. On introduit alors la fonction g telle que :

$$g(\theta) = \int f(x,\theta)\,\mu(dx)$$

où θ est une variable qui appartient à l'ensemble Θ.
On suppose que $g(\theta)$ est non nulle pour tout $\theta \in \Theta$. Le but est de maximiser g par rapport à θ.

Pour se faire , nous allons utiliser l'algorithme EM qui est déterministe, itératif et défini de la manière suivante. Connaissant $\theta^{(i)}$, on cherche à déterminer $\theta^{(i+1)}$ tel que :

$$\theta^{(i+1)} = T\left(\theta^{(i)}\right) \quad \text{et} \quad g\left(\theta^{(i+1)}\right) \geq g\left(\theta^{(i)}\right).$$

On pose :

$$p(x,\theta) = \frac{f(x,\theta)}{g(\theta)}.$$

On introduit la fonction $Q(\theta, \theta')$ définie par :

$$Q\left(\theta, \theta'\right) = \int \left(\log f(x, \theta)\right) p\left(x, \theta'\right) \, \mu(dx)$$

La première étape de l'algorithme, appelée étape E, est l'étape d'estimation de la fonction *Q*. La deuxième étape, étape M, est la suivante :

$$\theta^{(i+1)} = \quad \text{Arg}\max_{\theta} Q\left(\theta, \theta^{(i)}\right).$$

Une itération de l'algorithme EM est composée de l'étape E, suivie de l'étape M.

On va maintenant montrer que cet algorithme converge vers les maximas locaux de *g*. La démonstration se décompose de la manière suivante : on montre d'abord que *g* est une fonction croissante des $\theta^{(i)}$, puis qu'un extremum local de *Q* en est un pour *g*.

Commençons par la monotonie de *g*.

$$\begin{aligned} Q\left(\theta, \theta'\right) &= \int \left(\log f(x, \theta)\right) p\left(x, \theta'\right) \, \mu(dx) && (8.10) \\ Q\left(\theta, \theta'\right) &= \log\left(g(\theta)\right) + \int \left(\log p(x, \theta)\right) p\left(x, \theta'\right) \, \mu(dx) \\ Q\left(\theta, \theta'\right) - Q\left(\theta', \theta'\right) &= \log\left(g\left(\theta\right)\right) - \log\left(g\left(\theta'\right)\right) + \int \left(\log \frac{p(x, \theta)}{p\left(x, \theta'\right)}\right) p(x, \theta') \, \mu(dx) && (8.11) \end{aligned}$$

Le deuxième terme de la somme est l'opposé de la divergence de Kullback-Leibler. Elle est notée $KL(\theta \mid\mid \theta')$.

Nous allons maintenant appliquer l'inégalité de Jensen à la fonction $\log$. On considère la variable aléatoire $Y = p(x, \theta)/p\left(x, \theta'\right)$, distribuée selon $p(x, \theta)$.

De l'inégalité de Jensen, on déduit que :

$$\log\left(E\left[Y\right]\right) \geq E\left[\log Y\right].$$

Or :

$$\begin{aligned} \log\left(E\left[Y\right]\right) &= \log\left(\int Y p\left(x, \theta'\right) \, \mu(dx)\right) && (8.12) \\ &= \log\left(\int p(x, \theta) \, \mu(dx)\right) = 0. \end{aligned}$$

Et :

$$E[\log Y] = KL(\theta \mid\mid \theta').$$

Ce qui implique que :

$$KL(\theta \mid\mid \theta') \leq 0.$$

De plus, on cherche à maximiser $Q(\theta, \theta')$. Par conséquent :

$$Q\left(\theta, \theta'\right) - Q\left(\theta', \theta'\right) \geq 0 \quad \Longrightarrow \quad \log\left(g(\theta)\right) - \log\left(g(\theta')\right) \geq 0$$

Donc la fonction g est monotone et on a prouvé le premier point.

Abordons maintenant le deuxième point de la démonstration.
Soit θ' un extremum local de $Q(.,\theta')$.
Comme :

$$Q(\theta,\theta') = \log(g(\theta)) + \int (\log p(x,\theta))\, p(x,\theta')\, \mu(dx),$$

on en déduit que (en supposant que les conditions pour dériver sous le signe intégrale soient vérifiées) :

$$\begin{aligned} \left.\frac{\partial Q(\theta,\theta')}{\partial\theta}\right|_{\theta=\theta'} &= \left.\frac{\partial(\log g(\theta))}{\partial\theta}\right|_{\theta=\theta'} + \int \left(\left.\frac{\partial(\log p(x,\theta))}{\partial\theta}\right|_{\theta=\theta'}\right) p(x,\theta')\,\mu(dx) \\ &= 0. \end{aligned} \tag{8.13}$$

Nous allons maintenant utiliser la formule de Fisher pour montrer que :

$$\left.\frac{\partial(\log g(\theta))}{\partial\theta}\right|_{\theta=\theta'} = 0.$$

En utilisant la définition de *g*, on a en dérivant sous le signe intégrale :

$$\begin{aligned} \frac{\partial g(\theta)}{\partial\theta} &= \int \frac{\partial f(x,\theta)}{\partial\theta}\,\mu(dx) \\ \Rightarrow \frac{\partial(\log g(\theta))}{\partial\theta} g(\theta) &= \int \frac{\partial(\log f(x,\theta))}{\partial\theta} f(x,\theta)\,\mu(dx) \end{aligned} \tag{8.14}$$

d'après la formule de Fisher appliquée aux fonctions f et g.

On en déduit donc :

$$\frac{\partial(\log g(\theta))}{\partial\theta} = \int \frac{\partial(\log f(x,\theta))}{\partial\theta} p(x,\theta)\,\mu(dx).$$

Or :

$$\int \frac{\partial(\log f(x,\theta))}{\partial\theta} p(x,\theta)\,\mu(dx) = \frac{\partial(\log g(\theta))}{\partial\theta} + \int \frac{\partial(\log p(x,\theta))}{\partial\theta} p(x,\theta)\,\mu(dx).$$

Donc :

$$\int \frac{\partial(\log p(x,\theta))}{\partial\theta} p(x,\theta)\,\mu(dx) = 0.$$

Et par conséquent d'après l'équation (B.4) :

$$\left.\frac{\partial(\log g(\theta))}{\partial\theta}\right|_{\theta=\theta'} = 0.$$

Le deuxième point de la démonstration est donc prouvé.

On a ainsi montré la convergence de l'algorithme EM.

8.8 Jensen et Fisher

8.8.1 Inégalité de Jensen

- *Propriété*

  Soit φ une fonction convexe et *X* une variable aléatoire.
  Alors, on a l'inégalité suivante :

$$\varphi\left(E\left[X\right]\right) \leq E\left[\varphi(X)\right]$$

  pour autant que ces espérances existent et soient finies.
- *Preuve*

  Nous allons écrire le développement en série de Taylor de φ au second ordre :

$$\exists \epsilon \in]x, y[\quad \text{tel que} \quad \varphi(x) = \varphi(y) + \varphi'(y)(x-y) + \frac{\varphi''(\epsilon)(x-y)^2}{2}.$$

  Or φ est une fonction convexe.
  Donc sa dérivée seconde est positive en tout point et on a l'inégalité suivante :

$$\varphi(y) + \varphi'(y)(x-y) \leq \varphi(x).$$

  On prend $x = X$ et $y = E[X]$. On obtient alors :

$$\varphi\left(E[X]\right) + \varphi'\left(E[X]\right)\left(X - E[X]\right) \leq \varphi(X).$$

  Il suffit pour finir de prendre l'espérance de chaque membre de l'inégalité :

$$E\left[\varphi\left(E[X]\right)\right] + E\left[\varphi'\left(E[X]\right)\left(X - E[X]\right)\right] \leq E\left[\varphi(X)\right].$$

  Or :

$$E\left[\varphi\left(E[X]\right)\right] + E\left[\varphi'\left(E[X]\right)\left(X - E[X]\right)\right] = \varphi\left(E[X]\right).$$

  Par conséquent :

$$\varphi\left(E[X]\right) \leq E\left[\varphi(X)\right].$$

  L'inégalité est ainsi montrée.

8.8.2 Formule de Fisher

Soit *f* une fonction continue et dérivable sur son ensemble de définition D_f.

On a l'égalité suivante bien connue :

$$\forall x \in D_f \quad \frac{\partial\left(\log f(x)\right)}{\partial x} = \frac{f'(x)}{f(x)}.$$

La formule de Fisher consiste à écrire :

$$\forall x \in D_f \quad f'(x) = f(x)\frac{\partial\left(\log f(x)\right)}{\partial x}.$$

Bibliographie

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene analysis*. Wiley-Interscience, 1973.

[HSSW97] D.P. Helmbold, R.E. Schapire, Y. Singer, and M.K Warmuth. A comparison of new and old algorithms for a mixture estimation problem. *Machine Learning*, vol 27, 1997.

[JX95] M.I. Jordan and L. XU. Convergence results for the em approach to mixtures of experts architectures. *Neural Networks*, vol 8 :1409–1431, 1995.

[Man95] M. Mangeas. *Propriétés statistiques des modèles paramétriques non-linéaires de prévision de séries temporelles*. PhD thesis, Université Paris I, Panthéon-Sorbonne, UFR de Mathématiques et Informatique, 1995.

[MW95a] M. Mangeas and A.S. Weigend. Avoiding overfitting by locally matching the noise level of the data. *World Congress on Neural Networks (WCNNN'95)*, vol II :1–9, 1995.

[MW95b] M. Mangeas and A.S. Weigend. First experiments using a mixture of nonlinear experts for time series prediction. *World Congress on Neural Networks (WCNNN'95)*, vol II :1–9, 1995.