Annexe I

APPRENTISSAGE DES RÉSEAUX DE NEURONES

Position du problème.

Nous considérons ici le problème pratique de l'apprentissage d'un réseau de neurones bouclé décrit par la représentation d'état générale (forme canonique, chapitre 1 §I.2) :

$$\begin{cases} S(k+1) = \varphi_{RN}\left(S(k),\, I(k);\, C\right) \\ Y(k) = \psi_{RN}\left(S(k),\, I(k);\, C\right) \end{cases}$$

où nous notons $I(k) \in \mathbb{R}^{N_I}$ le vecteur des entrées externes du réseau à l'instant k, $S(k) \in \mathbb{R}^{N_S}$ le vecteur des variables d'état du réseau à l'instant k, $S(k+1) \in \mathbb{R}^{N_S}$ le vecteur des variables d'état du réseau à l'instant k+1, $Y(k) \in \mathbb{R}^{N_Y}$ le vecteur des sorties du réseau à l'instant k, et C les coefficients du réseau. φ_{RN} (. , . ; C) et ψ_{RN} (. , . ; C) représentent les fonctions réalisées par le réseau de neurones de la forme canonique interconnectés avec les coefficients C.

La tâche du réseau est définie (chapitre 1 §III.2) :

- par des séquences d'apprentissage constituées d'une séquence appliquée aux entrées externes {I(k)}, et une séquence de valeurs désirées correspondantes {D(k)} pour les sorties du réseau ;
- par une fonction de coût, définie à l'itération i sur une fenêtre fixe englobant toute la longueur N de la séquence d'apprentissage :

$$J(C, i) = \frac{1}{N} \sum_{k=1}^{N} E^i(k)^T W\, E^i(k) = \frac{1}{N} \sum_{k=1}^{N} \left(D(k) - Y^i(k)\right)^T W \left(D(k) - Y^i(k)\right)$$

où C représente les coefficients du réseau C(i-1) disponibles à l'itération i, $E^i(k)$ le vecteur des erreurs à l'instant k et à l'itération i, W une matrice définie positive (souvent diagonale), D(k) le vecteur des sorties désirées à l'instant k, et $Y^i(k)$ le vecteur des sorties du réseau à l'instant k et à l'itération i.

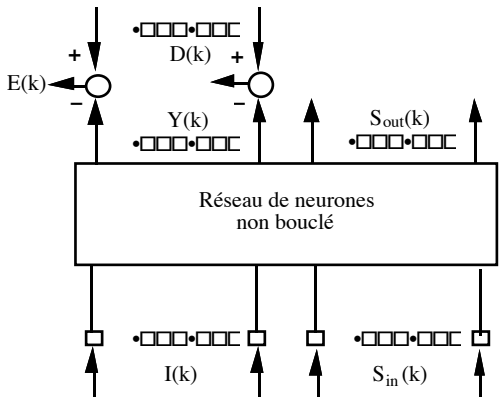

Figure 1.
Copie n°k utilisée pour l'apprentissage.

Notations.

Si l'on développe " spatialement " le comportement temporel du réseau [NER92b], on obtient un grand réseau non bouclé, dans lequel le réseau non bouclé de la forme canonique intervient N fois en cascade avec les coefficients disponibles à l'itération i. Les multiples interventions de ce même réseau sont appelées " *copies* ". Seules les valeurs des entrées et des sorties des neurones sont différentes d'une copie à l'autre. La copie numéro k est représentée sur la figure 1 (en omettant les indices i se référant à l'itération i).

Les entrées de la copie k sont :

- les entrées externes à l'instant k, I(k) $\mathbb{R}^{N_I}$ (séquence d'apprentissage),
- les variables d'état d'entrée à l'instant k, $S^{in}(k) \in \mathbb{R}^{N_S}$, avec $S^{in}(k)=S^{out}(k-1)$.

Les sorties de la copie k sont :

- les sorties à l'instant k, $Y(k) \in \mathbb{R}^{N_Y}$, qui sont les activités de N_Y neurones de sortie,
- les variables d'état de sortie à l'instant k, $S^{out}(k) \in \mathbb{R}^{N_S}$, qui sont les activités de N_S neurones d'état.

$D(k) \in \mathbb{R}^{N_Y}$ est le vecteur des sorties désirées à l'instant k (séquence d'apprentissage).

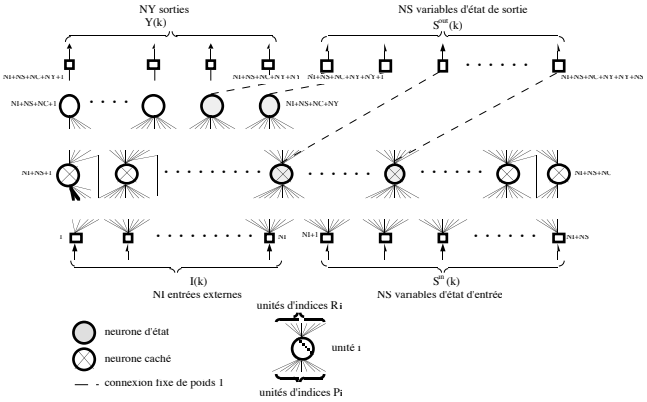

Figure 2.
Détail de la copie n°k utilisée pour l'apprentissage.

Dans, la suite, les valeurs des sorties de la copie sont les valeurs des activités de N_Y " unités " supplémentaires connectées aux N_Y neurones de sortie avec un coefficient 1. Les valeurs des variables d'état de sortie sont les valeurs des activités de N_S " unités " supplémentaires connectées aux N_S neurones d'état avec un coefficient 1 (un neurone d'état est un neurone caché ou un neurone de sortie). La fonction d'activation d'une unité d'entrée d'état, de sortie d'état ou de sortie, est l'identité. Chaque copie est ainsi composée de $N_I+N_S+N_C+N_Y+N_Y+N_S$ unités :

- N_I unités d'entrées externes (numérotées de 1 à N_I) ;
- N_S unités d'entrées d'état (numérotées de N_I+1 à N_I+N_S) ;
- N_C neurones cachés (ordonnés de N_I+N_S+1 à $N_I+N_S+N_C$) ;
- N_Y neurones de sortie (ordonnés de $N_I+N_S+N_C+1$ à $N_I+N_S+N_C+N_Y$) ;
- N_Y unités de sorties (numérotées de $N_I+N_S+N_C+N_Y+1$ à $N_I+N_S+N_C+N_Y+N_Y$) ;
- N_S unités de sorties d'état (numérotées de $N_I+N_S+N_C+N_Y+N_Y+1$ à $N_I+N_S+N_C+N_Y+N_Y+N_S$).

Il est commode de définir la connectivité du réseau par les P_i, ensemble des indices des unités *p*ropageant leur activité vers l'unité i, et les R_i, ensemble des unités *r*ecevant l'activité du neurone i.

En particulier, l'ensemble R_i d'une unité de sortie d'état est constitué de l'indice du neurone d'état auquel elle correspond, et l'ensemble R_i d'une unité de sortie est constitué de l'indice du neurone de sortie auquel elle correspond. De même, l'ensemble P_i d'un neurone de sortie est constitué de l'indice de l'unité de sortie à laquelle il correspond, et, si c'est aussi un neurone d'état, de l'indice de l'unité de sortie d'état à laquelle il correspond.

Exemple.

À titre illustratif, la forme canonique associée à un prédicteur entrée-sortie bouclé de la forme :

$$y(k+1) = \varphi_{RN}\left(y(k), \ldots, y(k-n+1), u(k), \ldots, u(k-m+1); C\right)$$

est représentée sur la figure 3, avec les mêmes notations que celles de la figure 2.

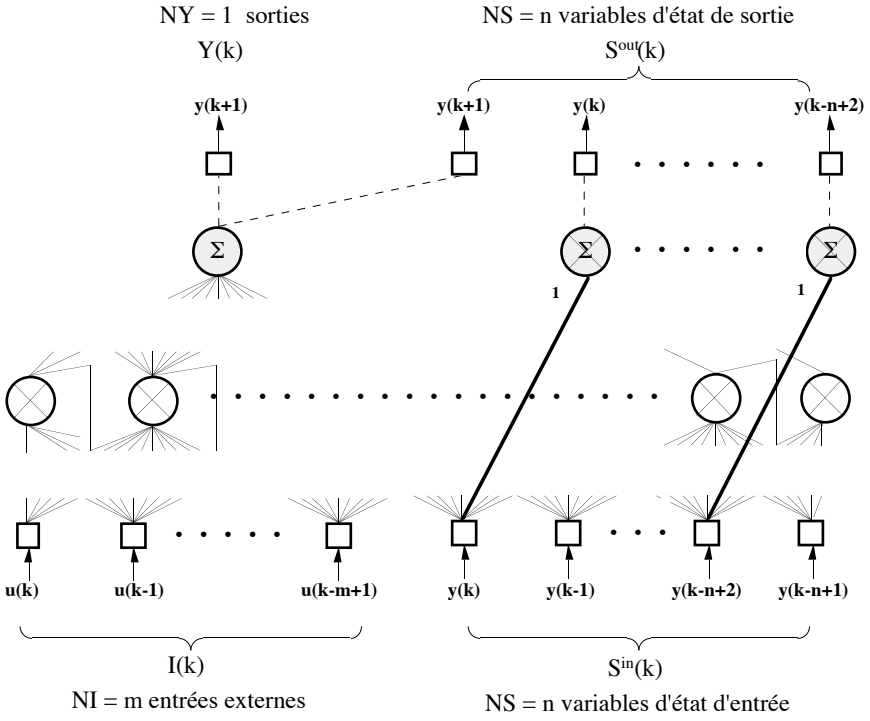

Figure 3.
Copie n°k utilisée pour l'apprentissage d'un prédicteur entrée-sortie (prédiction à 1 pas).

Le neurone de sortie est linéaire, ainsi que les neurones d'état. Ils ne possèdent qu'une connexion de poids 1 fixé sur les variables d'état d'entrée (valeurs passées de la sortie) auxquelles ils correspondent. Dans la suite, nous omettons ces neurones sur les schémas (voir les notation du chapitre 2 §I.1, le chapitre 3, et les notations du chapitre 5).

I. CALCUL DE LA FONCTION DE COÛT ET DE SON GRADIENT.

Calcul de la fonction de coût par propagation.

La propagation consiste à calculer les valeurs des potentiels $\{v_i(k)\}$ et des activités $\{z_i(k)\}$ de toutes les unités du réseau ainsi que les erreurs de sortie $\{e_i(k)\}$ à chaque instant k, donc pour chaque

copie (k est le numéro de la copie), de l'instant 1 à l'instant N. On peut ainsi calculer la fonction de coût définie pour une itération, dont nous rappelons l'expression en omettant l'indice de l'itération :

$$J(C) = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{NY} w_{ii}\, E_i^2(k)$$

Calcul du gradient de la fonction de coût par rétro-propagation.

La rétro-propagation consiste à calculer à chaque itération de l'algorithme les valeurs $\{\partial J/\partial v_i(k)\}$ des dérivées partielles de la fonction de coût par rapport au potentiel de chaque unité du réseau à tout instant k, de la dernière unité de chaque copie à la première, de l'instant N à l'instant 1, d'où la dénomination de rétro-propagation[1]. Ces dérivées permettent ensuite le calcul des dérivées de la fonction de coût par rapport aux coefficients. En effet, écrivons la différentielle de J(C) :

$$dJ(C) = \sum_{i,\,j} \sum_{k=1}^{N} \frac{\partial J}{\partial c_{ij}(k)}\; dc_{ij}(k) \quad \text{avec } dc_{ij}(k) = dc_{ij}\ \forall k, \text{ soit } \quad dJ(C) = \sum_{i,\,j} \sum_{k=1}^{N} \frac{\partial J}{\partial c_{ij}(k)}\; dc_{ij}$$

où $c_{ij}(k)$ représente le coefficient c_{ij} dans la copie numéro k. $c_{ij}(k)$ n'est qu'une écriture, puisque les coefficients sont les mêmes pour chaque copie de la fenêtre de la fonction de coût, donc pour chaque instant k. La composante du gradient de J relative au coefficient c_{ij} s'écrit :

$$\left(\frac{\partial J}{\partial c_{ij}}\right)_{\substack{\forall p,q \neq i,j \\ c_{pq} constant}} = \sum_{k=1}^{N} \left(\frac{\partial J}{\partial c_{ij}(k)}\right)_{\substack{\forall p,q \neq i,j \\ \forall r \neq m \\ c_{pq}^{r} constant}}$$

La rétro-propagation consiste à poser le calcul de la manière suivante :

$$\frac{\partial J}{\partial c_{ij}(k)} = \frac{\partial J}{\partial v_i(k)} \frac{\partial v_i(k)}{\partial c_{ij}(k)} = \frac{\partial J}{\partial v_i(k)}\, z_j(k)$$

où P_i est l'ensemble des unités propageant leur activité au neurone i. Les valeurs des activités $\{z_i(k)\}$ ont toutes été calculées lors de la propagation. La connaissance des $\{\partial J/\partial v_i(k)\}$ et des $\{z_i(k)\}$ permet donc le calcul du gradient.

---

[1] Le calcul par rétro-propagation suppose implicitement la nullité des dérivées de la fonction de coût par rapport aux entrées d'état de la première copie. Pour leur imposer des valeurs non nulles, il faut calculer le gradient par une méthode directe [NER92a]. Or imposer des valeurs non nulles n'est justifié que pour un apprentissage récursif, ce qui ne sera pas le cas dans ce mémoire, où nous traitons l'apprentissage de systèmes non adaptatifs. Nous utilisons donc exclusivement le calcul par rétropropagation, beaucoup plus économe que le calcul direct en nombre d'opérations.

Si le réseau n'est pas bouclé (N_S=0), chaque vecteur d'erreur E(k) est calculé indépendamment des autres puisque les copies ne se transmettent aucune valeur. De même, la contribution de chaque erreur au gradient est calculée de façon indépendante. La contribution de la copie k à la valeur du gradient dépend des entrées externes I(k) et des valeurs désirées D(k) (N rétro-propagations).

Si le réseau est bouclé, les valeurs des variables d'état sont transmises d'une copie à la suivante :

$$S_i^{in}(k) = S_i^{out}(k\text{-}1)$$

La contribution de chaque erreur au gradient ne peut plus être calculée de façon indépendante. En effet, on a :

$$\frac{\partial J}{\partial S_i^{out}(k)} = \frac{\partial J}{\partial S_i^{in}(k+1)}$$

On effectue dans ce cas une seule rétro-propagation.

I.1. Calcul de la fonction de coût, ou propagation.

* Entrées externes : *par définition*, pour i=1 à N_I

$$z_i(k) = v_i(k) = I_i(k)$$

* Variables d'état d'entrée : *par définition*, pour i=1 à N_S

$$k = 1 : \quad z_{NI+i}(1) = v_{NI+i}(1) = S_i^{in}(1) = \text{valeur imposée}$$

$$k > 1 : \quad z_{NI+i}(k) = v_{NI+i}(k) = S_i^{in}(k) = S_i^{out}(k\text{-}1)$$

La valeur imposée est choisie en fonction du problème particulier considéré, cf chapitres 3 et 5.

* Neurones cachés, neurones de sortie, unités de sortie, unités de sorties d'état : *calcul*, pour i variant de N_I+N_S+1 à $N_I+N_S+N_C+N_Y+N_Y+N_S$

$$z_i(k) = f_i\left(v_i(k)\right) \text{ où } \quad v_i(k) = \sum_{j \in P_i} c_{ij}\, z_j$$

où P_i est l'ensemble des unités propageant leur activité au neurone i.

* Sorties : *par définition,* pour i=1 à N_Y

$$Y_i(k) = z_{NI+NS+NC+NY+i}(k)$$

* Variables d'état de sortie : *par définition*, pour i=1 à N_S

$$S_i^{out}(k) = z_{NI+NS+NC+NY+NY+i}(k)$$

* Erreurs : *calcul*, pour i=1 à N_Y de

$$E_i(k) = D_i(k) - Y_i(k)$$

I.2. CALCUL DU GRADIENT DE LA FONCTION DE COÛT, OU RÉTRO-PROPAGATION.

* Variables d'état de sortie : *par définition*, pour i=1 à N_S

$$k = N : \quad \frac{\partial J}{\partial v_{NI+NS+NC+NY+NY+i}(N)} = \frac{\partial J}{\partial S_i^{out}(N)} = 0$$

$$k < N : \quad \frac{\partial J}{\partial v_{NI+NS+NC+NY+NY+i}(k)} = \frac{\partial J}{\partial S_i^{out}(k)} = \frac{\partial J}{\partial S_i^{in}(k+1)}$$

* Sorties : *calcul*, pour i=1 à N_Y

$$\frac{\partial J}{\partial v_{NI+NS+NC+NY+i}(k)} = - \frac{2}{N} w_{ii} e_i(k)$$

* Neurones de sortie, neurones cachés et unités d'entrées d'état : *calcul*, pour i décroissant de $N_I+N_S+N_C+N_Y$ à N_I+1

$$\frac{\partial J}{\partial v_i(k)} = f_i^{'}\left(v_i(k)\right) \sum_{h \in R_i} c_{hi} \frac{\partial J}{\partial v_h(k)}$$

où R_i est l'ensemble des unités recevant l'activité du neurone i.

* Variables d'état d'entrée : *par définition*, pour i=1 à N_S

$$\frac{\partial J}{\partial S_i^{in}(k)} = \frac{\partial J}{\partial v_{NI+i}(k)}$$

L'algorithme de calcul de la fonction de coût et de son gradient est dit " dirigé " si le modèle est non bouclé (N_S=0), " semi-dirigé " sinon. Ceci signifie, dans le premier cas, que le réseau n'a pas d'état, et qu'il est donc entièrement " dirigé " par les entrées externes, et, dans le second cas, que l'état du réseau ne lui est imposé qu'au début de la fenêtre de la fonction d'apprentissage (pour la première copie)[2]. Les notions de dirigé et de semi-dirigé prendront tout leur sens dans les chapitres consacrés à l'apprentissage de prédicteurs et de correcteurs.

I.3. RÉSUMÉ DES CALCULS.

La figure 4 résume les calculs à effectuer pour chaque copie k, k variant de 1 à N, horizon sur lequel la fonction de coût est définie. La propagation, ou calcul de la fonction de coût par le réseau de propagation RP, doit être effectuée dans tous les blocs avant la rétro-propagation, ou calcul du gradient de la fonction de coût par le réseau RRP, sauf si le réseau n'est pas bouclé (les flèches grisées disparaissent).

---

[2] Dans le cas récursif, un troisième algorithme, dit " non-dirigé ", est également utilisé [NER92a].

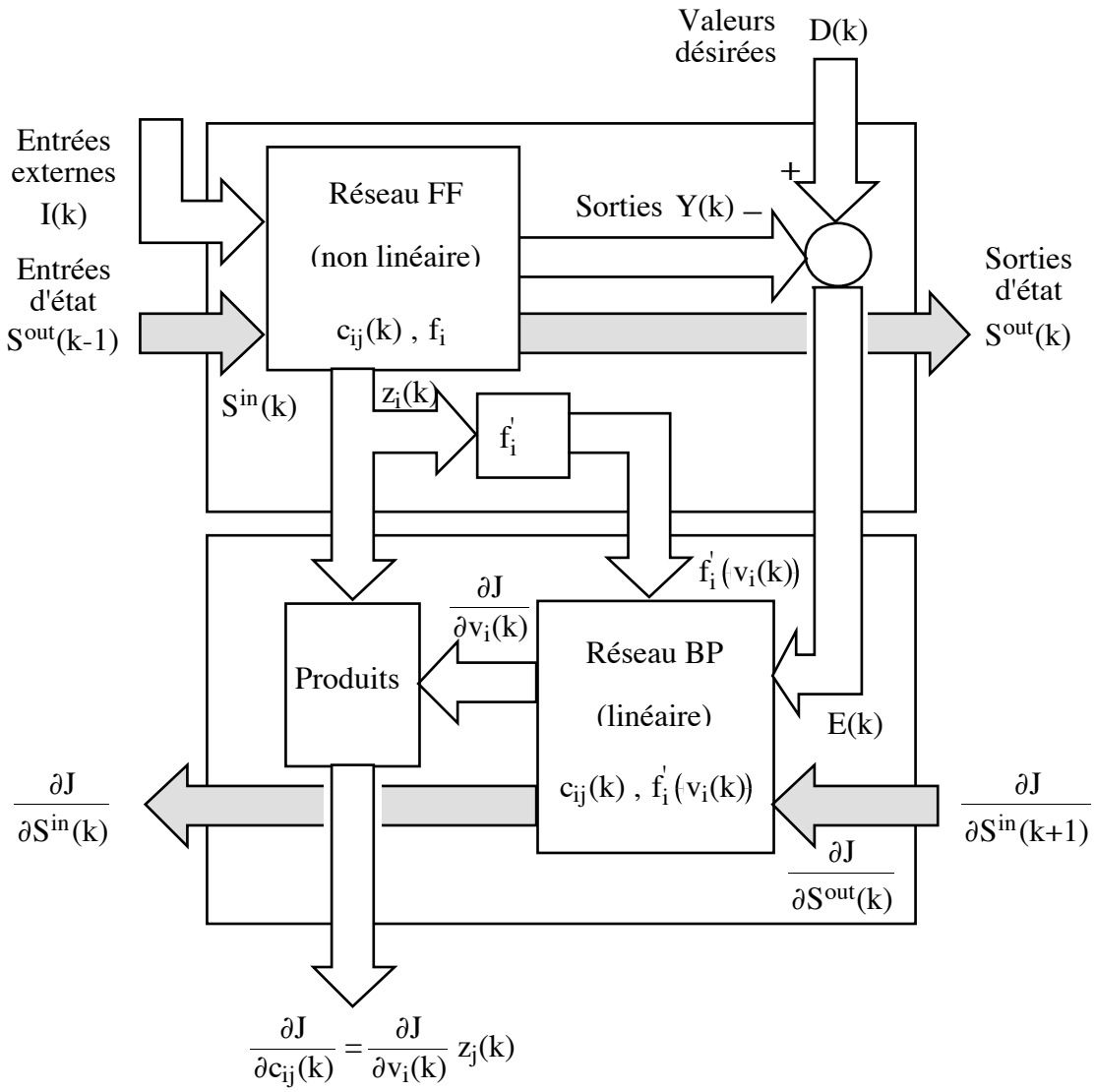

Figure 4.
Calculs correspondants à la copie k.

II. MODIFICATION DES COEFFICIENTS.

Dans le cas général, le calcul de la modification des coefficients (dont la valeur n'est pas fixée[3]) à l'itération i s'écrit :

$$\Delta C = + \mu D$$

où D est une direction de descente,

$\mu>0$ est le pas de descente.

[3] Dans le cas de l'apprentissage d'un correcteur en partculier, le système d'apprentissage utilise un modèle de simulation, dont les coefficients ne doivent pas être modifiés.

II.1. MÉTHODE DE GRADIENT À PAS CONSTANT.

La méthode la plus couramment utilisée est celle où D est l'opposé du gradient et μ est constant. Les modifications des coefficients sont données par l'expression :

$$\Delta c_{ij} = - \mu \frac{\partial J}{\partial c_{ij}} = - \mu \sum_{k=1}^{N} \frac{\partial J}{\partial c_{ij}}(k) = - \mu \sum_{k=1}^{N} \frac{\partial J}{\partial v_i(k)} z_j(k)$$

Cette méthode présente l'inconvénient d'avoir une vitesse de convergence très lente lorsque l'on s'approche du minimum. Nous utilisons les méthodes suivantes qui permettent d'accélérer la convergence de façon appréciable.

II.2. MÉTHODES À PAS VARIABLE.

Quelle que soit la direction de descente utilisée, il est possible d'asservir le pas μ de telle sorte que la fonction de coût diminue à chaque modification des coefficients. Nous avons utilisé deux méthodes de minimisation unidimensionnelle " économiques ", les méthodes de Nash [NAS90] et de Wolfe et Powell [MIN83]. Ces méthodes permettent d'obtenir un pas convenable avec un nombre très limité d'évaluations de la fonction de coût et du gradient J. Notons C_i les coefficients, D_i la direction de descente, et μ_i le pas de descente à l'itération i. Soit la fonction g définie par : $g\ (\mu_i) = J\left(C_i + \mu_i D_i\right)$. Sa dérivée s'écrit : $g'(\mu_i) = D_i^T \nabla J\left(C_i + \mu_i D_i\right)$. g est donc la fonction dont on cherche le minimum en fonction du pas μ_i.

a) Règle de Nash.

Le principe de cette méthode est que le pas de descente ne doit pas être choisi trop grand sinon l'algorithme risque d'avoir un comportement oscillatoire. Le pas de descente μ_i doit être choisi de façon telle que :

$$g\left(\mu_i\right) \leq g\left(0\right) + m_1\ \mu_i\ g'\ \left(0\right)$$

soit :

$$J\left(C_i + \mu_i D_i\right) \leq J\left(C_i\right) + m_1\ \mu_i\ D_i^T \nabla J\left(C_i\right)$$

Cette condition assure la propriété de descente. m_1, le facteur de tolérance, est un nombre petit devant 1 (par exemple 10^{-3}). Si la condition ci-dessus n'est pas satisfaite, le pas μ_i est multiplié par un facteur de réduction (Nash propose 0,2). En pratique, avec les valeurs précédentes, on se limite à une vingtaine d'évaluations de J, après quoi les modifications effectuées sont de l'ordre de grandeur des erreurs d'arrondis.

b) Règle de Wolfe et Powell.

(a) Le pas de descente ne doit pas être choisi trop grand sinon l'algorithme risque d'avoir un comportement oscillatoire.

(b) Le pas de descente ne doit pas être choisi trop petit pour permettre une convergence rapide de l'algorithme.

μ_i doit être choisi de façon telle que :

(1) $g(\mu_i) \leq g(0) + m_1\, \mu_i\, g'(0)$ avec $m_1 \in]0,1[$

Cette première règle assure la propriété de descente.

(2) $g'(\mu_i) \geq m_2\, g'(0)$ avec $m_2 \in]m_1,1[$

La seconde règle permet de s'assurer que le pas n'est pas non plus choisi trop petit, comme le montre la figure 5. Les deux règles s'écrivent, en fonction du critère et de son gradient :

(1) $J(C_i + \mu_i D_i) \leq J(C_i) + m_1\, \mu_i\, D_i^T \nabla J(C_i)$ avec $m_1 \in]0,1[$

(2) $D_i^T \nabla J(C_i + \mu_i D_i) \geq m_2\, D_i^T \nabla J(C_i)$ avec $m_2 \in]m_1,1[$

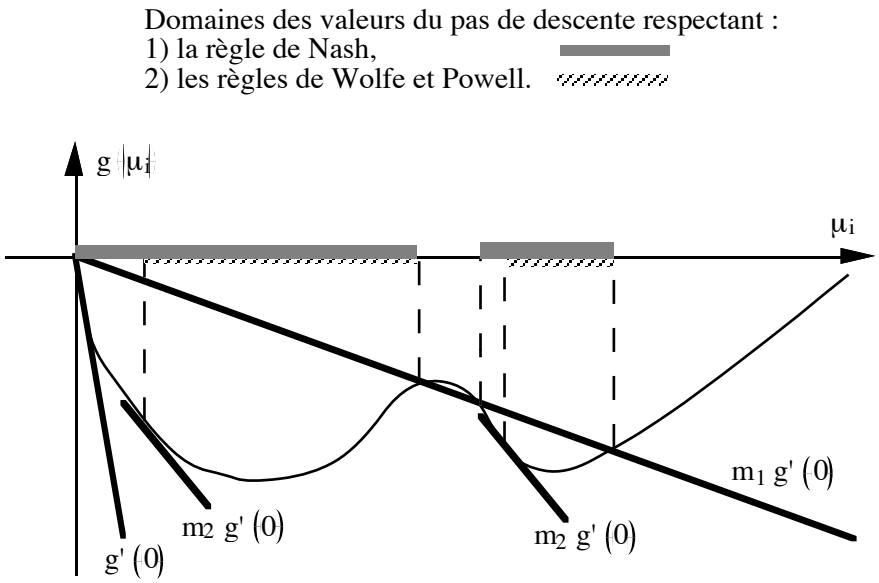

Figure 5.
Optimisation unidimensionnelle : règles de Nash et de de Wolfe et Powell (coupe dans la direction de descente D_i).

Pratiquement, la procédure est la suivante :

Initialiser les valeurs : $\mu=1$; $\mu_{min}=\mu^0_{min}$, ; $\mu_{max}=\mu^0_{max}$.

a) Calculer $J(C_i + \mu D_i)$.

b) Si la condition (1) est vérifiée, aller en c); sinon $\mu_{max}=\mu$ (μ est trop grand) et aller en d).

c) Si la condition (2) est vérifiée $\mu_i=\mu$ est accepté ; sinon $\mu_{min}=\mu$ (μ est trop petit) et aller en d).

d) Modifier le pas μ : $\mu = \frac{\mu_{min} + \mu_{max}}{2}$ et aller en a).

On prend par exemple : $m_1 = 0{,}1$; $m_2 = 0{,}7$; $\mu_{min}=0$; $\mu_{max}=10$ (ce choix n'est pas très critique).

La mise en œuvre de cette méthode est donc un peu plus lourde que la précédente.

II.3. MÉTHODE QUASI-NEWTONIENNE À PAS VARIABLE.

Nous avons également employé une méthode utilisant une autre direction de descente que le gradient, une méthode quasi-newtonienne (ou méthode à métrique variable) utilisant l'algorithme de Broyden, Fletcher, Goldfarb et Shanno (BFGS) dont la vitesse de convergence est beaucoup plus grande que celle du gradient [MIN83].

Méthode de Newton.

La méthode de Newton consiste à remplacer la fonction de coût par son approximation quadratique au voisinage du point courant (la quadrique osculatrice) :

$$q(C) = J(C_i) + \nabla J^T(C_i)(C - C_i) + \frac{1}{2}(C - C_i)^T \nabla^2 J(C_i)(C - C_i)$$

et à choisir les coefficients C_{i+1} de manière à minimiser q(C). Le minimum existe si le Hessien $\nabla^2 J(C_i)$ est défini positif. Ce qui conduit à la formule itérative :

$$C_{i+1} = C_i - \left[\nabla^2 J(C_i)\right]^{-1} \nabla J(C_i)$$

La méthode de Newton demande donc de calculer puis d'inverser le Hessien à chaque itération. Notons que le pas de descente est fixé (μ=1). Appliquée à une fonction J des coefficients C quadratique strictement convexe, cette méthode converge en une seule itération, mais dans le cas général d'une fonction non-linéaire quelconque, elle ne possède pas la propriété de convergence globale. Si les coefficients de départ sont trop éloignés du minimum, la méthode peut ne pas converger.

Méthode de quasi-Newton.

Le principe des méthodes quasi-newtoniennes consiste en une généralisation de la formule itérative de Newton :

$$C_{i+1} = C_i + \mu_i D_i \text{ avec } D_i = -H_i \nabla J(C_i)$$

où H_i est une matrice définie positive donnant la direction de descente à partir du gradient $\nabla J(C_i)$, et μ_i est choisi tel que $J(C_i + \mu_i D_i) < J(C_i)$, donc par exemple par les méthodes de Nash ou de Wolfe et Powell. La matrice H_i est une approximation de l'inverse du Hessien. En effet, elle est modifiée à chaque itération de telle manière que, pour une fonction quadratique, elle converge vers l'inverse du Hessien $\left[\nabla^2 J(C_i)\right]^{-1}$. Différentes formules du type :

$$H_{i+1} = H_i + \Delta_i$$

ont été proposées. L'une d'elle est la formule BFGS suivante :

$$H_{i+1} = H_i + \left[1 + \left(\frac{\gamma_i^T H_i \gamma_i}{\Delta C_i^T \gamma_i}\right)\right] \frac{\Delta C_i^T \Delta C_i}{\Delta C_i^T \gamma_i} - \frac{\Delta C_i \gamma_i^T H_i + H_i \gamma_i \Delta C_i^T}{\Delta C_i^T \gamma_i}$$

avec $\gamma_i = \nabla J(C_{i+1}) - \nabla J(C_i)$ et $\Delta C_i = C_{i+1} - C_i$.

Si la matrice H_{i+1} ainsi calculée n'est pas définie positive, elle est réinitialisée à la matrice identité. On repart alors avec $D_i = -\nabla J(C_i)$, c'est-à-dire dans la direction opposée à celle du gradient. Un des principaux attraits de la méthode BFGS est qu'elle est relativement insensible au choix du pas. Comme nous l'avons précisé ci-dessus, il suffit que μ_i soit tel que $J(C_i + \mu_i D_i) < J(C_i)$. On peut donc se contenter de la méthode de Nash, la plus simple des méthodes présentées.