# MercureO2, Version2

M. Boughanem et S. Marcel

# Table des matières

# Introduction

MercureO2 est un système de recherche d'informations orienté objet basé sur l'approche connexionniste. Nous présentons dans ce rapport une nouvelle version de MercureO2. Les changements importants par rapport à la version précédente résident principalement dans la partie fonctionnelle. De nouvelles fonctionnalités ont été rajoutées. Par contre tout ce qui concerne les opérations de génération et construction d'une base d'informations restent inchangées. Il faut bien comprendre que dans toutes les versions de Mercure, ou MercureO2 nous distinguons deux modules : la premier est le module de construction d'une base d'informations, c'est ce que l'on appelle *la génération dans MercureO2*, et le deuxième module s'occupe de de l'utilisation des informations de la base, c'est ce que l'on appelle *Moteur de MercureO2*. Les changements que nous avons apportés concernent le deuxième module. Cette version permet de mieux prendre en compte les notions de reinjection de pertinence et d'apprentissage à long terme.

L'architecture que nous proposons dans MercureO2 est très proche de l'architecture multi-couches des réseaux de neurones. Ceci nous a permis d'adapter sans aucun problème les fonctionnalités des réseaux de neurones pour répondre aux principales fonctions d'un SRI à savoir représentation des besoins d'un utilisateur, évaluation de ces besoins, sélection d'informations répondant à l'utilisateur, reformulation automatique de requêtes et éventuellement restructuration des informations de la base d'informations pour l'évolution du SRI.

Ce rapport est divisé en deux parties. la première est consacrée à la description fonctionnelle de MercureO2. Nous présenterons dans cette partie l'architecture du modèle ainsi que ces fonctionnalités à savoir, l'évaluation de requêtes, la modification automatique de requêtes ainsi que la fonction d'apprentissage.
Nous présenterons dans la deuxième partie du rapport le modèle conceptuel et interne de MercureO2. Nous décrirons dans cette partie les différents objets utilisés dans le moteur de MercureO2.

# Chapitre 1

# MercureO2 : le modèle fonctionnel

## 1.1 Le modèle

MercureO2 est un système de recherche d'informations orienté objet basé sur l'approche connexionnsite. L'élément de base de MercureO2 est l'objet neurone.

Du point de vue fonctionnelle, MercureO2 peut être modélisé par un réseau connexionniste multi-couches (voir figure 1.1). On distingue :

1. une entrée, représentant la requête de l'utilisateur,
2. une couche de neurones termes représentant la couche d'entrée du réseau,
3. une couche de neurones documents représentant la couche de sortie,
4. des couches cachées entre la couche terme et la couche documents,
5. une sortie représentant les résultats de recherche obtenus,
6. des associations entres les neurones.

### 1.1.1 La couche des neurones documents

Elle représente l'ensemble des neurones documents. Le neurone document représente l'image en mémoire d'un descripteur d'un document. Le descripteur d'un document est une liste de termes décrivant le contenu de ce document. Cette liste est obtenue grâce *à* une procèdure d'indexation automatique, qui est exécutée lors de la construction de la base. Elle consiste d'une façon générale *à* :

1. Extraire les mots isolés du document.
2. Utiliser une liste de mots vide pour éliminer les mots non significatifs.

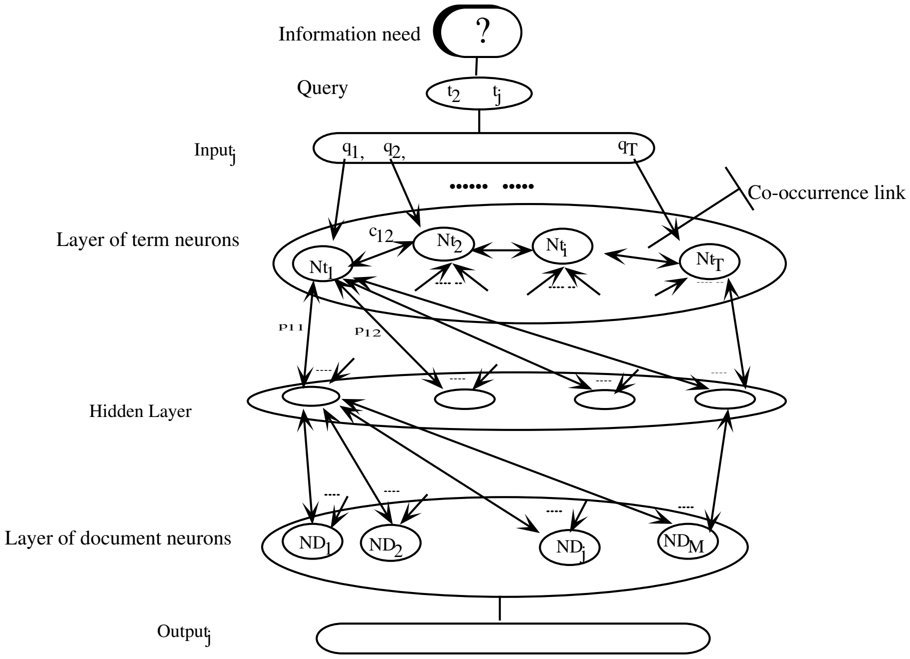


FIG. 1.1 - *Architecture fonctionnelle de MercureO2*

3. Chercher la racine du mot. Dans cas deux méthodes peuvent être utilisées, si le texte est en français, la méthode utilisée est la troncature à sept caractères [17]. Si le texte est anglais, nous utilisons une méthode, appelé *stemmer* développée par Porter [16], permettant de trouver le radical d'un mot.

### 1.1.2 La couche des neurones termes

Le neurone terme représente un terme d'indexation pendant la phase de recherche. L'identificateur du neurone est l'adresse du terme dans le fichier inverse. Les termes d'indexation, c'est *à* dire les termes extraits des documents, constituent un espace vectoriel que l'on peut noter $\overrightarrow{T}$. Ainsi, tout document de la base peut être représenté dans $\overrightarrow{T}$ sous forme vectoriel. Cette représentation est utilisée dans le système SMART [7].

Exemple : le document $D_j(a_{1j}, a_{2j}, ..., a_{Tj})$

Où :
T : est le nombre de termes d'indexation reconnu dans la base,
$a_{1j}$ : est égal *à* 1 ou 0 selon que le terme $t_i$ appartient au document $D_j$ ou non.

### 1.1.3 L'entrée du réseau

Elle représente le besoin en information d'un utilisateur. Ce besoin est présenté au système sous forme d'une question en langage naturel ou simplement une liste de mots clés. Cette question est analysée par la procédure d'indexation utlisée pour les documents. Ceci va conduire à la construction d'une liste de mots significatifs qui va représenter la requête de l'utilisateur. A chaque terme est ensuite associé un poids donné par :

$$q_{ij} = \frac{freq_{ij}}{max_j(freq_{ij})}$$

L'entrée du réseau est alors représentée par un vecteur ayant la forme suivante :

$$Input_j = (q_{1j}, q_{2j}, ..., q_{Tj})$$

Où :

$q_{ij}$ représente le poids du terme dans la requête.
$freq_{ij}$ : fréquence d'apparition du terme i dans la requête j.
T : le nombre de termes d'indexation.

Noter que l'indice j utilisé pour les requêtes n'est pas nécessaire, il est utilisé ici juste pour désigner une requête spécifique.

### 1.1.4 La sortie du réseau

La sortie représente les résultats obtenus pour l'entrée présentée. Elle est composée des valeurs de sorties des neurones documents. Elle représente les documents restitués en réponse à la requête initiale.

### 1.1.5 les couches cachées

MercureO2 prévoit l'utilisation des couches cachées, par analogie aux réseaux de neurones multi-couches. Leur utilisation serait intéressante lors de la phase d'apprentissage du réseau. L'avantage de ces couches est discuté dans de nombreux articles et ouvrages [14]. Le nombre de ces couches, variant de 0 à $k > 0$, et le nombre de leurs neurones sont des paramètres importants dans les réseaux multi-couches. Il faut noter que dans la vesrion actuelle de MercureO2 ces couches ne sont pas utilisées, ( le paramètre, nombre de couches cachées est initialisé à 0). Ces couches ne sont pas représentées dans la suite de ce rapport

### 1.1.6 Les liens

Les neurones sont associés entre eux par des liens bidirectionnels pondérés. Cette pondération représente le degré d'importance entre deux neurones. Plus la valeur du poids du lien est grande plus l'interaction entre ces neurones est importance.
Ces liens bidirectionnels vont permettre l'envoi et le retour des signaux entre les neurones termes et/ou les neurones documents. Ce processus de va et vient permet d'effectuer des évaluations et des reformulations de requêtes par retour de pertinence.
On distingue deux types de liens:

- lien inter-couches,
- lien intra-couhes.

### Lien inter-couches

Les neurones d'une couche sont reliés sont tous reliés aux neurones de la couche suivante. Les poids de ces liens pourraient être déterminés d'une manière aléatoire. Dans notre cas, en l'abscence de couche cachées, nous avons juste des liens entre les neurones termes et les neurones documents, ces liens représentent des liens d'indexation. Le poids de ce lien détermine le degré significatif du terme dans le document. Il a été montré dans [8], que la performance d'un système de recherche d'informations dépend beaucoup du choix de ce poids. Dans notre cas nous avons utilisé la formulation habituelle :

$$poids = IDf * tf$$

Cette formulation est une combinaison entre la fréquence d'apparition d'un terme dans le document (tf) et la frequence inverse d'apparition du terme dans la collection de document(IDF) [7] [8].

Le poids du lien entre un neurone terme et un neurone document que nous avons utilisé dans MercureO2 est donné par :

$$p_{ij} = \frac{freq_{ij}}{\sqrt{\sum_{k=1}^{M}(freq_{kj}^2.log(\frac{M}{M_k})^2)}} * \log(\frac{M}{M_i})$$

Où
$freq_{ij}$ : est la fréquence d'apparition du terme i dans le document j,
M : est le nombre de documents contenus dans la collection,
$M_i$ : est le nombre de documents contenant le terme i.

Les termes n'apparaissant pas dans un document ont des poids nuls.

### Lien intra-couches

Les liens intra-couche que nous avons défini, concernent seulement les neurones de la couche termes. Le lien entre deux neurones termes représente une relation de cooccurrence [1], [5] . Ce lien est déterminé *à* partir de la distribution des termes dans les documents. Nous nous sommes basé pour cela sur l'hypothèse suivante :
*si deux termes apparaissent souvent ensemble dans des documents, ces deux termes pourraient être sémantiquement liés.*

Pour évaluer le lien entre deux termes, nous avons tenu compte des poids des liens d'indexation des deux termes dans tous les documents et du nombre de documents contenant respectivement les deux termes en même temps et chacun des termes séparément. Différentes formulations ont été testées, celle que nous avons retenue est :

$$c_{ij} = \alpha.\frac{\sum_{k=1}^{M}(p_{ik} * p_{jk})}{\sqrt{\sum_{k=1}^{M} p_{ik}^2 + \sum_{k=1}^{M} p_{ik}^2 - \sum_{k=1}^{M}(p_{ik} * p_{jk})}}$$

Où :
$p_{ik}$ : poids du lien entre le terme i et le document k,

M : nombre de documents.

Les valeurs de cooccurrence sont rangées dans une matrice symétrique $c_{ij} = c_{ji}$

## 1.2 Evaluation des requêtes

L'évaluation d'une requête va permettre la sélection des documents qui répondent à la question de l'utilisateur. Le principe est entièremment basé sur un processus d'activation propagation de signaux défini dans les résaux de neurones multi-couches. Brièvemment, ceci revient à présenter une ***entrée*** au réseau, les neurones de la premiére couche calculent une entrée, une valeur d'activation puis transmettent ces signaux d'activation (ou de sorties) vers les neurones des autres couches ou vers la sortie du réseau. Les valeurs de sorties de la couche terminale (couche documents dans notre cas) représentent la ***sortie obtenue***.

Dans MercureO2 cela se traduirait comme suit :

1. Aprés une succession de traitement, la requête de l'utilisateur est transformée en une entrée, notée $Input_j = (q_{1j}, q_{2j}, ..., q_{Tj})$. Cette entrée est présentée au réseau, c'est à dire à la couche de termes. Chaque neurone de la couche calcule une valeur d'entrée, à l'état initial, elle est égale à :

   $$In(Nt_i) = q_{ij}$$

   puis une valeur d'activation ( signal de sortie):

   $$Out(Nt_i) = g(In(Nt_i))$$

2. Ces signaux sont transmis vers les neurones de la couche suivante, donc vers la couche documents. Les neurones de cette couche calculent :

   $$In(ND_i) = \sum_{j=1}^{T} Out(Nt_j) * w_{ij}$$

   puis,

   $$Out(ND_i) = g(In(ND_i))$$

La sortie du réseau, notée *Output*, est représentée par l'ensemble des valeurs de sortie, soit :

$$Output_j(Out(ND_1), Out(ND_2), ..., Out(ND_M))$$

Ces valeurs de sortie calculées par les neurones documents vont permettre de trier les documents correspondants. Normalement plus une valeur de sortie d'un neurone document est importante plus le document correspondant est susceptible d'être pertinent à la requête.

Il faut noter que le neurone dans MercureO2 est représenté par un objet, il n' y a pas de distinction entre un neurone document et un neurone terme ou un neurone des couches cachées. Les fonctions que nous avons utilisées pour calculer respectivement l'entrée et l'activation(sortie) d'un neurone sont :

> On considère MercureO2 comme un réseau à M couches, numérotées de 1 à M. La couche m=1 et la couche d'entrées, les neurones termes et la couche m=M est celle des neurones documents, les couches de 2 à M-1 sont les couches cachées.

la fonction d'entrée : $In(N_i^{m+1}) = \sum_{j=1} Out(N_j) * w_{ij}m$
la fonction de sortie : $Out(N_i^{m+1}) = g(In(N_i^{m+1}))$

Où :
$N_i^m$ : neurone i de la couche m,
$w_{ij}^m$ : le poids du lien entre les neurones $i$ de la couche m et $j$ e la couche m+1.

Pour résumer brièvemment ce processus, nous voyons que la valeur d'entrée permet de mesurer l'importance du neurone dans la phase d'évaluation de requête. Plus une valeur d'entrée d'un neurone est grande plus le neurone est important. L'utilisation d'une fonction de sortie (activation) permet de ramener les valeurs d'entrées dans un intervalle qu'on peut facilement contrôler pour effectuer par exemple des seuillages. Plusieurs fonctions de sortie peuvent etre utilisées, nous avons dans notre cas utilisé la sigmoide :

$$g(x) = \frac{e^x - 1}{e^x + 1}$$

## 1.3 Modification automatique de requêtes

La modification automatique de requêtes est devenue un processus nécessaire dans le domaine de la recherche d'informations. Cette modification peut intervenir de deux façons :

- modification directe : dans ce cas la requête de l'utilisateur est modifiée avant de sélectionner les documents,

- modification indirecte : la requête est modifieé en tenant compte d'une liste de documents déjà jugés par l'utilisateur, c'est le processus de reinjection de pertinence, communémment appelé *Relevance Feedback*.

Nous allons voir comment sont réalisés ces deux processus dans notre modèle

### 1.3.1 Modification directe

La modification directe consiste à ajouter de nouveaux termes à la requête initiale. Cette modification est très simple à réaliser grâce à l'architecture réseau que nous avons adoptée. On présente une entrée au réseau, les neurones activés dans la couche termes vont trasmettre leurs activations vers les autres neurones termes à travers les liens de co-occurrence. Les neurones recevant des signaux vont calculer leur valeurs d'entrées et d'activations. Les neurones activés à cette étape peuvent à leur tour envoyés des signaux vers les autres neurones neurones termes etc. (voir figure 1.3). Ce processus peut être réitéré tant qu'il y a de nouveaux termes. Mais nous avons montré dans [12] qu'au delà d'une itération les termes rajoutés à la requêtes ne sont pas significatifs et peuvent engendrer du bruit.

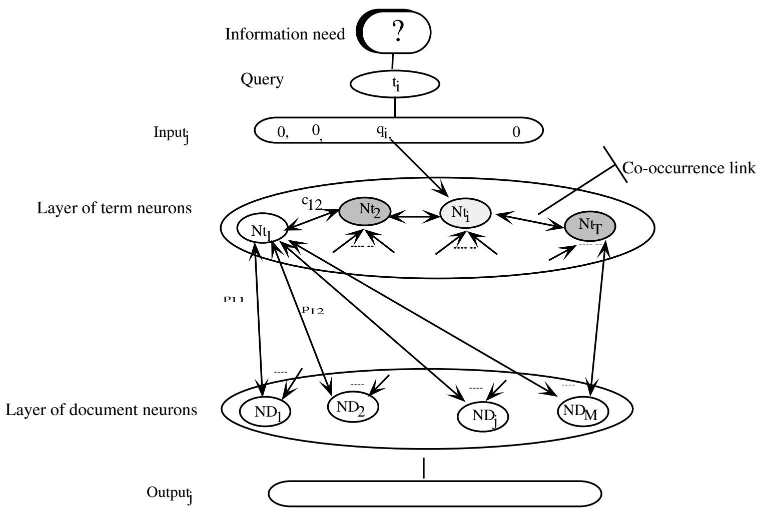


FIG. 1.2 - *Modification directe de requêtes*

Nous voyons bien que grâce à l'architecture de MercureO2, la modification automatique de requêtes est très facile à réaliser et nous permet surtout de mieux tenir compte des poids des associations entre les termes et le nombre de ces associations. En effet comme les associations sont pondérées, un neurone $Nt_i$ reçevant plus de signaux qu'un neurone $Nt_j$ aura normalement une valeur d'entrée plus importante et par la même une valeur de sortie plus importante. Ceci va permettre d'une part de mieux prendre en compte cette association en situant un terme (représenté par son neurone) par rapport *à* l'ensemble des termes de la requête, et d'autre part de mieux contrôler le nombre de termes *à* rajouter *à* une requête. Ce dernier point est évidemment très important dans les bases volumineuses ou le nombre d'associations entre les termes peut être très grand.

### 1.3.2 Modification basée sur la reinjection de pertinence

Cette modification est basée sur les jugements de l'utilisateur de la pertinence des documents restitués en sortie du réseau.

La(es) technique(s) que nous utilisons est largement inspirée de l'algorithme de retropropagation du gradient dans les réseaux de neurones multi-couches. En effet, les documents jugés par l'utilisateur, pertinents ou non pertinents, seront utilisés comme un exemple de ***sortie désirée*** par l'utilisateur. Des signaux seront transmis de la sortie désirée vers l'entrée, une nouvelle requête sera alors construite.

Plusieurs stratégies sont envisageables. Ces stratégies dépendent du type des signaux à retropropager, donc de la sortie désirée, du type d'entrée, et d'autres paramètres. Nous allons présenter deux de ces stratégies :

– retropropagation de la pertinence

– retropropagation du gradient.

**Retropropagation de la pertinence**

La sortie désirée par l'utilisateur est présentée à la couche documents, les signaux d'entrées et de sortie seront calculés par les neurones documents, puis ils sont retropropagés vers les neurones termes qui à leur tour vont calculer leurs signaux. Une nouvelle entrée sera alors calculée à partir des signaux calculés par la couche d'entrée, puis sera présentée au réseau.

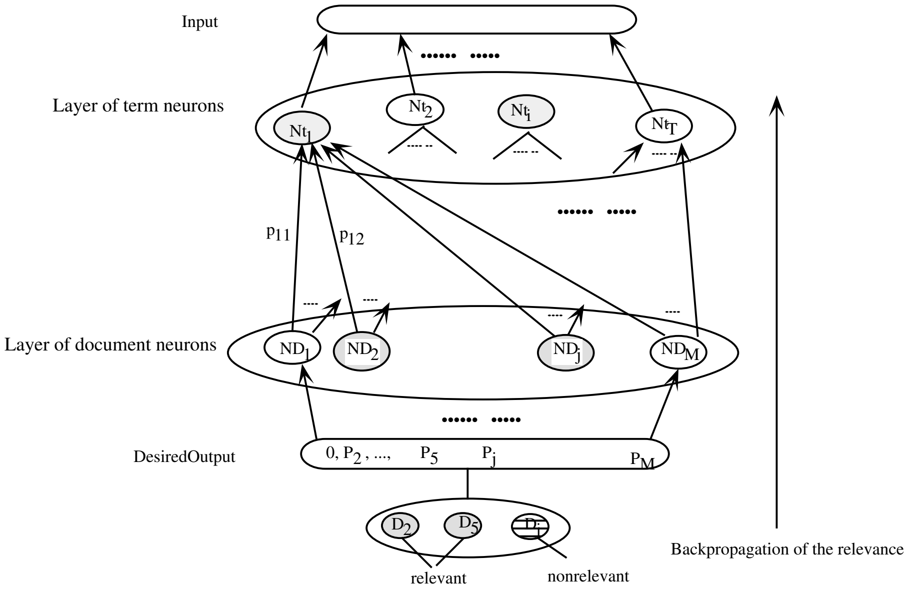


FIG. 1.3 - *Modification directe de requêtes*

L'élément important dans cette stratégie est le vecteur de la sortie désirée. Pour construire ce vecteur nous avons défini la notion de *valeur de pertinence* d'un document, nous détaillerons ce point à la fin du paragraphe. La sortie désirée est representée comme suit :

$$DesiredOutput = (rel_1, ..., rel_i, , ..., rel_M)$$

Où:

M : le nombre de neurones documents,
$rel_i$ : est la valeur de pertinence du document i.

Ce processus de va et vient, propagation et retropropagation, peut être réalisé plusieurs fois pour une même entrée. De ce fait, deux cas seront considérés, selon qu'on tienne compte de ou non l'activation des neurones à l'ètape précédente, sachant qu'une propagation est étape et une retroproagation est une autre étape .

**a- on ne tient pas compte des activations précédentes des neurones**
Les étapes de cette stratégie peuvent être décrite comme suit :

1. construire la sortie désirée : $DesiredOutput = (rel_1, ..., rel_i, , ..., rel_M)$,
2. présenter la sortie désirée à la couche sortie du réseau (couche documents), les neurones documents vont calculer une valeur d'entrée :

$$In(ND_i) = rel_i$$

Puis, un signal de sortie :

$$Out(ND_i) = g(In(ND_i))$$

3. ce signal est transmis vers la couche de neurones termes. Chaque neurone va calculer une entrée,

$$In(Nt_i) = \sum_{j=1}^{M} (p_{ij} * Out(ND_j))$$

puis un signal de sortie,

$$Out(Nt_i) = g(In(Nt_i))$$

4. les signaux ainsi calculés seront transmis l'entrée. Si l'on considère $Input_j$ comme étant l'entrée présentée, la nouvelle entrée sera calculée comme suit :

$$NewInput_j = Input_j + \alpha * Out(Nt)$$

5. cette nouvelle requête sera présentée au réseau, de nouveaux documents seront donc sélectionnés. Cette stratégie peut être répétée plusieurs fois.

Où :
Nt : représente l'ensemble des neurones de la couche termes, $(Nt_1, Nt_2, ..., Nt_T)$,
g : la fonction sigmoide,
$p_{ij}$ : poids entre neurone termes $Nt_i$ et neurone documents $ND_j$,
$rel_i$ : valeur de pertinence du document i.

**b-on tient compte des activations précédentes des neurones**

Dans ce cas le signal de sortie d'un neurone à l'étape $t$ donnée est calculée en fonction des signaux qu'il reçoit et son signal de sortie précédent. Ainsi, nous aurons :

$$Out^{t+1}(N_i) = Out^t(N_i) + g(In(N_i))$$

Sachant quà t=0, qui correspond à la première propagation, $Out^0(N_i) = g(In(N_i))$
Pour ce qui des étapes de déroulement d'une evaluation de requête et de retropropagation de la pertinence nous retrouvons les mêmes étapes que précédemment.

La notion de valeur de pertinence que nous avons défini va permettre de quantifier le signal qui sera transmis par les documents pertinents et par les non pertinents. Ainsi la formule que nous proposons est :

si le document est pertinent

$$rel_i = Coef\_Pert * log_2(\frac{NbDocVus}{NbPert} + \beta)$$

si le document est non pertinent

$$rel_i = Coef\_NPert * log_2(\frac{NbDocVus}{NbNPert} + \beta)$$

Où:
Coef_Pert, Coef_NPert : représentent respectivement les coefficients de pertinence et de non pertinence des documents,
NbPert, NbNPert : représentent respectivement le nombre de documents pertinents et non pertinents jugés, par l'utilisateur,
NbDocVus : nombre de documents jugés par l'utilisateur.

Nous constatons que dans cette, nous tenons compte aussi bien des documents pertinents que des non pertinents. De plus, comme notre but est de rapprocher nos recherches vers les documents pertinents et en même temps les éloigner des documents non pertinents, la valeur de pertinence des documents pertinents sera positive et celle des non pertinents sera négative. Ceci est assuré par les coefficients, Coef_Pert et Coef_NPert qui doivent obligatoirement vérifier les conditions suivantes : $Coef\_Pert > 0$ et $Coef\_NPert < 0$

### Retropropagation du gradient

Cette stratégie est une application directe de l'algortithme de retropopagation du gradient. Comme dans la stratégie précédente la sortie désirée présentée au réseau est composée des documents jugés pertinents ou non pertinents, par contre dans ce cas, une erreur comise entre la sortie obtenue et la sortie désirée, sera calculée puis sera retropropagée vers les couches supérieures. Une nouvelle erreur est calculée au niveau de chaque couche par chaque neurone, puis retropropagée vers les couches supérieures, ceci jusqu'à la couche d'entrée. Une nouvelle entrée va être alors calculée en fonction de l'erreur de la couche d'entrée et de l'ancienne entrée.

D'une manière algorithmique nous aurons :

*Hypothèse : le processus d'évaluation est terminé, l'utilisateur a jugé les documents restitués en réponse à la requête représentée par* $Input_j$ *et la sortie obtenue par* $Output_j(Out(ND_1), Out(ND_2), ..., Out(ND_M))$

1. construire la sortie désirée : $DesiredOutput = (P_1, ..., P_i, , ..., P_M)$
   où $P_i = 1$ si document pertinent 0 si non pertinent.

2. calculer l'erreur entre la sortie obtenue et la sortie désirée

$$\delta_j^D = g'(In(ND_j)) * (P_j - Out(ND_j))$$

3. retropropager l'erreur vers les autres couches, ici vers la couche termes (ceci reste valable si nous avons utilisé les couches cachées). Chaque $Nt_i$ calcule son erreur à partir des erreurs qu'il a reçu :

$$\delta_i^t = g'(In(Nt_i)) * \sum(p_{ij} * \delta_j^D)$$

4. calculer la nouvelle entrée

$$NewInput_j = Input_j + \alpha * \delta_j^t$$

## 1.4 Apprentissage à long terme

L'apprentissage à long terme permet à MercureO2 d'évoluer dans le temps. Cette évolution va entrainer la modification des représentations des documents et des liens entre les différents éléments de MercureO2. L'apprentissage que nous proposons permet la modification des liens inter-couches.
*Pourquoi cette modification ?* Nous avons discuté et montré dans nos travaux précédents [11] que les liens entre les termes et les documents ne sont pas toujours valides, car ils sont établis d'une manière statistique en se basant principalement sur la présence ou l'absence d'un terme dans un document. Le but de l'apprentissage est d'améliorer les représentations des documents en modifiant les poids des liens termes documents.

L'apprentissage pourrait être effectué soit par un professeur, par exemple l'administrateur de la base, ou par expérience acquise au bout d'un certain nombre d'utilisation.

L'apprentissage que nous proposons est effectué par un professeur et il basé sur l'algorithme de retropropagation du gradient. Dans ce cas, nous disposons d'un ensemble d'exemple à apprendre. Chaque exemple est constitué d'une entrée et de sa sortie désirée. Nous retrouvons dans cette stratégie, les différentes étapes de propagation et retropropagation définies précédemment. Il faut noter que l'apprentissage que nous proposons est effectué avant toute utilisation du système MercureO2 à ne pas confondre avec ce que l'on a vu précédemment et qui concerne l'utilisation du système MercureO2 par un utilisateur occasionnel.

La stratégie d'apprentissage est présentée comme suit :

*Les exemples à apprendre sont repésentés comme suit : ($E^k$, $DesiredOutput^k$). On considère le réseau MercureO2 avec son architecture globale, c'est à dire avec les couches cachées. Ainsi, le réseau contient M couches sachant que la couche m=1 est la couche d'entrée, et m=M est la couche de sortie.*

1. présenter l'entrée $E^k(E_1^k, E_2^k, ..., E_T^k)$ à la couche d'entrée (m=1). Chaque neurone calcule :
$$In(N_i^1) = E_i^k$$
$$Out(N_i^1) = g(In(N_i^1))$$
2. propager le signal de sortie vers les neurones de la couche suivante. pour chaque neurone de la couche, calculer :
$$In(N_i^m) = \sum_{j=1} w_{ij}^m * Out(N_i^{m-1})$$
$$Out(N_i^m) = g(In(N_i^m))$$
Jusqu'à la couche de sortie, m=M.
3. calculer l'erreur entre la sortie obtenue et la sortie désirée
$$\delta_i^M = g'(In(N_i^M)) * (DesiredOutput_i^k - Out(N_i^M))$$
4. retropropager l'erreur vers les autres couches, Chaque $N_i^m$ calcule son erreur à partir des erreurs qu'il a reçu:
$$\delta_i^m = g'(In(N_i^m)) * \sum_{j=1}(w_{ji}^{m+1} * \delta_j^{m+1})$$
calculer ces erreurs pour tous les neurones et sur toutes les couches, m=M-1,M-2,...,1.
5. modifier les poids des liens inter-couches entre les neurones.
$$w_{ij}^{new} = w_{ij}^{old} + \alpha * \Delta w_{ij}^{old}$$
$$\Delta w_{ij}^m = \eta * \delta_i^{m+1} * Out(N_i^m)$$
6. refaire le processus en présentant un nouvel exemple.

Où :
$N_i^m$: le neurone i de la couche m,
$\delta_i^m$ : erreur commise sur le neurone i de la couche m,
$w_{ij}^m$ : lien entre le neurone $N_i$ de la couche m-1 et le neurone $N_j$ de la couche m,
$\eta$: coefficient utilisé pour contrôler la vitesse d'apprentissage.

La question que nous allons nous poser concerne la forme des exemples qui sont présentés au réseau. En d'autres termes, à quoi vont correspondre les entrées et les sorties désirées. Plusieurs formes sont envisageables, nous pouvons citer :

1. l'entrée est un document, c'est à dire la liste des mots du document, et la sortie désirée sera le document lui même. Nous aurons donc $E^k = (p_{1k}, ..., p_{Tk})$ et la sortie désirée, $DesiredOutput^k = (0, ..., 1, 0, .., 0)$ à la position k du vecteur sortie désirée nous avons 1. Dans ce cas on présente à l'entrée du réseau un document et on veut avoir en sortie ce même document.

2. on construit un ensemble de requêtes, on désigne pour chacune de ces requêtes ses documents pertinents. Dans ce cas l'entrée du réseau est une requête, la sortie désirée sont les documents pertinents ou non non pertinents qui sont associés à la requête. Le représentation de l'entrée et la sortie désirée est faite de la même façon que ce que l'on a vu dans le paragraphe précédent.

# Chapitre 2

# MercureO2: le modèle formel, conceptuel et interne

## 2.1 Le modèle formel

MercureO2 utilise un modèle de réseau de neurones à couches. On peut donc représenter formellement un tel réseau [18].
On définit un réseau de neurones par le triplet (I, H, O) avec

$I = (NbI, \{..., C_q^I, ...\})$ structure des couches d'entrées
$H = (NbH, \{..., C_q^H, ...\})$ structure des couches cachées
$O = (NbO, \{..., C_q^O, ...\})$ structure des couches de sorties
NbI, NbH, NbO : respectivement nombre de couches d'entrées, cachées et de sorties
$C_q^X$ : couche de type X d'indice q avec $q \in [1..NBX]$ et $X \in [I, H, O]$

Une couche sera représentée formellement comme suit :

$$C^k = [N, \beta, \beta', \theta, \eta, \epsilon, \phi, \gamma, NBIn, (..., In_i^k, ..), NbOut, (..., Out_j^k, ...), \Pi_{[N,N]}]$$

Où :

$C^k$: couche numéro k
N : nombre de neurones de la couche
$\beta$ : seuil d'activation des neurones
$\beta'$ : seuil de rétroactivation des neurones
$\theta$: vitesse de propagation de l'information dans les neurones
$\eta$ : vitesse d'apprentissage
$\epsilon$ : seuil de désapprentissage
$\phi$ : seuil de disparition des connexions
$\gamma$ : seuil de rétropropagation

Tous les paramètres permettent d'initialiser les neurones, mais certains de ces paramètres pourront être modifiés durant les apprentissages.

NbIn : nombre de couches en entrée
$In_i^k = (C^{k"}, \Pi_{[N,C_{k'}::N]})$ : i-ème couche d'entrée avec $k > k'$

NbOut : nombre de couches en sortie
$Out_i^k = (C^{k'}, \Pi_{[N,C_{k''}::N]})$ : j-ème couche de sortie avec $k < k''$
$\Pi_{[n,m]}$: matrice n*m de connexions

Nous noterons :: l'opérateur d'accessibilité à nos structures de couches et de réseaux.

La vitesse de propagation intervient dans les fonctions d'activation et de rétroactivation pour freiner ou accélérer le signal dans les neurones. Les seuils d'activation et de rétroactivation, eux, interviennent dans les fonctions de sorties.

Ce modèle formel met en évidence les différents paramètres nécessaire aux processus tels que :

- la propagation du signal,
- la rétropropagation de l'erreur,
- l'apprentissage.

Ce modèle n'a pas la prétention d'être exhaustif en la matière, mais il nous suffira amplement pour concevoir le moteur de MercureO2.

A travers ce modèle, il se dessine déjà l'architecture interne du moteur de MercureO2. Cependant, il convient tout d'abord d'élaborer un modèle plus conceptuel basé sur la notion d'objets.

## 2.2 Le modèle conceptuel

En effet, la notion d'objet s'adapte parfaitement aux réseaux de neurones. Ainsi, nous pouvons représenter un modèle conceptuel sous la forme d'un graphe orienté, où les noeuds sont des objets et les arcs sont des liens d'héritage ou des liens de composition.

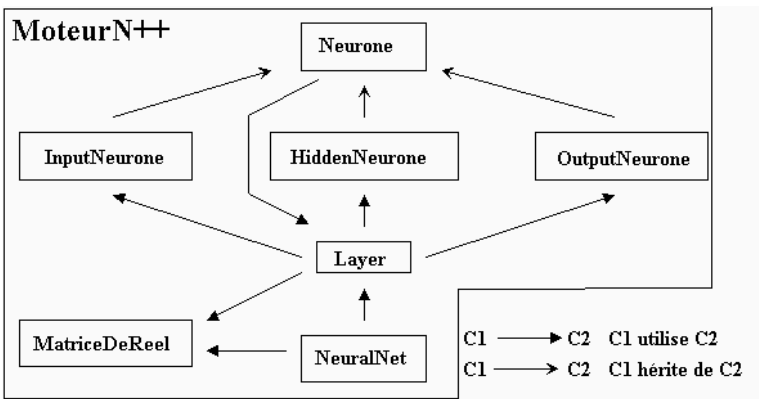


FIG. 2.1 - *Modèle conceptuel du moteur de MercureO2*

On distingue l'objet principal de type réseau de neurones (*NeuralNet*). Il utilise, par des liens de composition, l'objet de type couche de neurones (*Layer*) et l'objet de type matrice (*MatriceDeReel*). A son tour, l'objet Layer est composé des objets de type neurone d'entrée, neurone caché et neurone de sortie (respectivement *InputNeurone, HiddenNeurone* et *OutputNeurone*). Ces derniers héritent des propriétés de l'objet de base de type Neurone.

## 2.3 Les objets

Ce paragraphe va permettre de détailler la composition des objets du modèle conceptuel. Ceci va nous permettre d'aboutir à la description du modèle interne du moteur de MercureO2.

### 2.3.1 Objet neurone

L'objet Neurone est l'élément de base de MercureO2. Il comprend la majorité des paramètres du modèle formel, ainsi que les méthodes de :

- réception du signal,
- traitement du signal,
- propagation du signal,
- réception de l'erreur,
- traitement de l'erreur,
- rétropropagation de l'erreur,
- modification des connexions.

Un neurone connaît également la couche à laquelle il appartient, ce qui lui permettra d'accéder aux autres neurones.

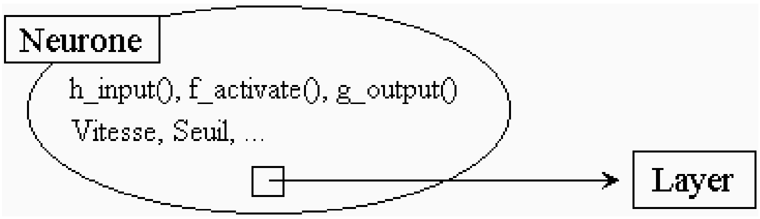


FIG. 2.2 - *Modèle de l'objet Neurone*

Cet objet n'est utilisé par les autres objets du modèle qu'indirectement et ce par l'intermédiaire des objets InputNeurone, HiddenNeurone et OutputNeurone. On ne détaillera pas plus ces derniers objets. En effet, ils héritent des propriétés de l'objet Neurone, et ils n'assurent simplement que la cohérence dans le typage des objets.

### 2.3.2 Objet couche de neurones

L'objet Layer modélise une couche de neurones d. Elle est constituée :

- d'un ensemble de références sur ses neurones,
- d'un ensemble de références sur les couches inférieures,
- d'un ensemble de références sur les couches supérieures.

Cet objet associe également à chaque référence de couche, la référence sur une matrice d'inter-connexions. La matrice des intra-connexions est référencée si elle existe.

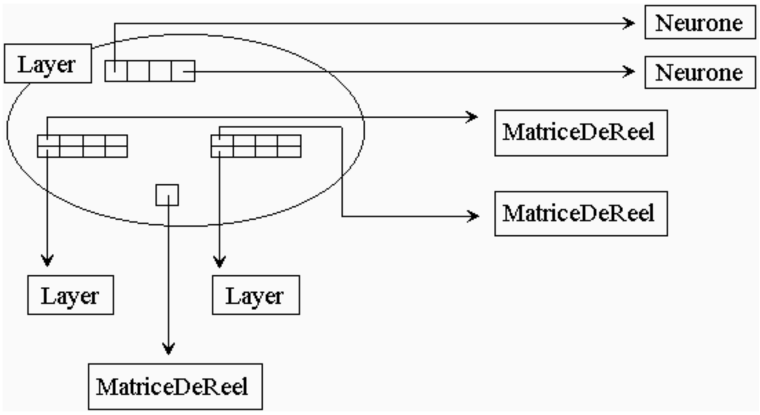


FIG. 2.3 - *Modèle de l'objet Layer*

### 2.3.3 Objet matrice

L'objet MatriceDeReel assure le stockage des poids des connexions entre les neurones. Il permet également la manipulation de vecteurs pour activer le réseau en entrée et récupérer les résultats en sortie. Dans le cas de MercureO2, les matrices stockeront les poids d'indexation et les poids de cooccurrence.

Cet objet permet la représentation de matrices de grandes tailles notamment grâce à une technique de matrice creuse et à un compromis mémoire statique - mémoire dynamique - fichier. Ainsi, cet objet offre un accès rapide à une grande quantité de données, ce qui est primordial pour garantir des temps de réponse raisonnables dans nos réseaux de neurones.

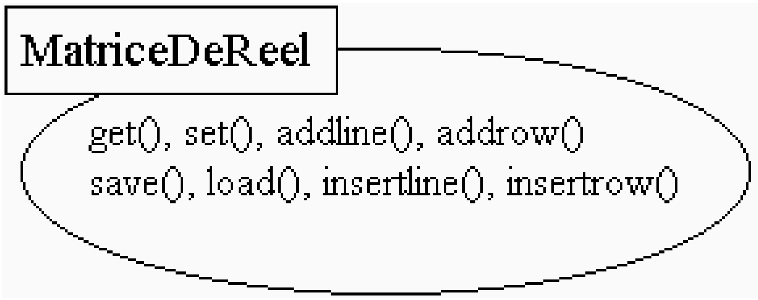


FIG. 2.4 - *Modèle de l'objet MatriceDeReel*

De nombreuses primitives donnent à cet objet une grande robustesse de contrôle sur l'exécution, ainsi qu'une grande simplicité d'utilisation.

### 2.3.4 Objet réseau

L'objet réseau (NeuralNet) assure l'encapsulation des couches de neurones. Il référence la ou les couches d'entrée, de sortie et éventuellement cachée. Il comprend également des méthodes

- de construction du réseau,
- de propagation,
- de rétropropagation.

L'objet NeuralNet détient également la liste des références de toutes les matrices afin de pouvoir les détruire.

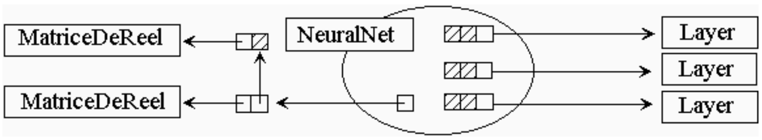


FIG. 2.5 - *Modéle de l'objet réseau*

## 2.4 Le modèle interne

Le modèle interne s'obtient en regroupant les différents objets voir figure 2.6

Un neurone i peut connaître tous les neurones j avec qui il est connecté de plusieurs manières.

• si il y a des intra-connexions, le neurone accède à sa couche et donc à la matrice des intra-connexions M avec les neurones j,
• si il y a des couches inférieures et/ou supérieures, le neurone accède à sa couche, puis à chaque couche inférieure et/ou supérieure et donc aux matrices des inter-connexions M associées à chacune de ces couches.

On détermine que :

- si $M[i,j] = 0$ alors il n'y a pas de connexion avec le neurone j,
- si $M[i,j] \neq 0$ alors il y a une connexion avec le neurone j et son poids est M[i,j].

Ainsi, les signaux peuvent circuler dans deux sens dans le réseau.

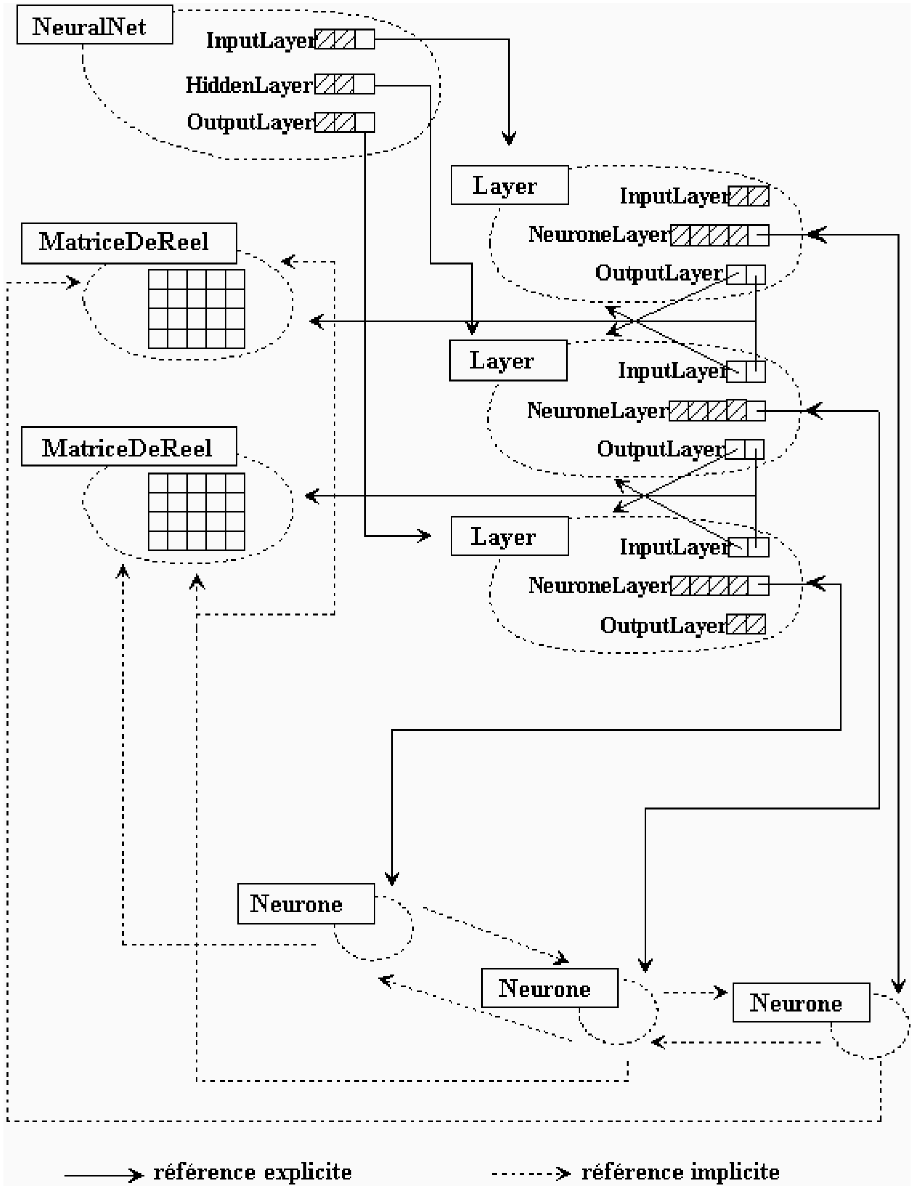


Fig. 2.6 - *Modèle interne de MercureO2*

# Conclusion

Nous avons présenté dans ce rapport la nouvelle version de MercureO2. Les changements que nous avons apporté par rapport à la version précédente résident principalemt dans le moteur de MercureO2. Ce moteur s'occupe comme nous l'avons déjà souligné de la phase d'utilisation de MercureO2, c'est à dire, interrogation, reformulation de requêtes soit directement ou par reinjection de pertinence et l'apprentissage. La partie concernant la génération d'une base d'informations effectuée dans le module génération de MercureO2 n'a pas été modifié.

Nous remarquons que l'architecture que nous avons adoptée dans cette version est identique à celle utilisée dans le réseau multi-couches. Ceci nous a permis d'adapter sans aucun problème un certain nombre de fonctions d'un réseau multicouches pour effectuer avec succés des opérations de recherche et de sélection d'informations définies dans les SRI.
Ainsi, nous pouvons classer ces opérations en trois catégories:

1. Evaluation de requêtes : cette opération est effectuée par propagation de signaux de la couche de termes vers la couche de documents, l'entrée du réseau correpond à la requête de l'utilisateur et sa sortie les documents restitués en réponse à cette requête.

2. Modification de requête : nous avons défini deux type de modification. Une modification directe réalisée grâce à la propagation de signaux entre les neurones termes à travers des liens définis entre ces neurones. Une modification basée sur la reinjection de pertinence, cette opération est effcetuée grâce à la procèdure de retropropagation. Nous avons développé plusieurs stratégies inspirées de l'algorithme de retroproagation du gradient.

3. Apprentissage : la fonction d'apprentissage va permettre au réseau d'évoluer dans le temps. Pour effectuée cette fonction nous avons appliqué l'algorithme de retroproagation du gradient au réseau multi-couhes MercureO2.

Le système MercureO2 que nous avons présenté dans ce rapport est opérationnel sur une station SUN sous unix. MercureO2 a été utilisé dans TREC'96 [13] pour effectuer les tâches Adhoc et Routing.

# Annexe A

# Déclaration des objets de MercureO2

## A.1 Objets neurone

```
class Neurone
{
friend InputNeurone;
friend HiddenNeurone;
friend OutputNeurone;
friend NeuralNet;
public:
// Constructeur
Neurone(LayerNeuronType, ulong, double, double, double, double, Layer *);
// Type du neurone
// Identificateur
// Seuils
// Vitesse d'apprentissage
// Vitesse de propagation
// Adresse de la couche

// Destructeur
~Neurone();

// Reception en entree de l'activation d'un neurone
virtual Booleen ReceptionInput(double, double, Booleen, uint) { return FAUX; };
// Activation recue
// Poids de la connexion

// Reception d'une erreur retro-propagee
virtual Booleen ReceptionErreur(double, double, uint, Layer *, Booleen)
  { return FAUX; };
// Erreur recue
```

```
// Poids de la connexion
// Identificateur du neurone qui renvoie l'erreur
// Adresse de la couche du neurone qui envoie l'erreur
// Effectuer le retro-seuillage

// Activation des neurones en sortie
Booleen ActiverOutput(uint);

// Retro-propager l'erreur
Booleen Retropropager(Booleen);

// Afficher les composantes du neurone
void Afficher();

// Election du neurone
void SetFirst() { bIsFirst=VRAI; };

// Initialisation de l'etat du neurone
void Init();

// Initialisation de l'etat du neurone
void InitFeedback();

protected:
// Nombre d'activations attendues
uint GetNbrInput();

// Nombre de neurones a activer
uint GetNbrOutput();

// Apprentissage par la regle du delta
double DeltaRule(double, double);
// Erreur retro-propagee
// Poids d'une connexion

// Fonction d'entree
double h_input(double, double);
// Activation
// Poids

// Fonction d'activation
double f_activate(double);

// Fonction de sortie
double g_output(double);
```

```
// Fonction derivee de la fonction f
double f_derivee(double);

// Fonction derivee de la fonction g
double g_derivee(double);

// Identificateur
ulong Identificateur;

// Etat d'election
Booleen bIsFirst;

// Seuil d'activation
double BetaSeuil;

// Seuil de retro-activation
double BetaRetro;

// Vitesse d'apprentissage et de propagation
double EtaSpeed, TetaSpeed;

// Valeur d'entree
double I;
double OldI;

// Valeur d'activation
double A;

// Valeur de sortie
double O;
double OldO;

// Valeur de l'erreur locale
double E;
double OldE;

// Nombre d'activations recues
uint NbrActivationsRecues;

// Nombre d'erreurs recues
uint NbrErreursRecues;

// Adresse de la couche du neurone
Layer *HostLayer;

// Type du neurone
```

```
LayerNeuronType NeuroneType;
};
```

## A.2 Objet couche de neurones

```
class Layer
{
friend class NeuralNet;
friend class Neurone;
friend class InputNeurone;
friend class HiddenNeurone;
friend class OutputNeurone;
public:
// Constructeur
Layer(LayerNeuronType, uint, MatriceDeReel *, uint, uint, uint);
// Type de la couche
// Numero de la couche
// Matrice des intra-connexions
// Nombre de neurones de la couche
// Nombre de couches In
// Nombre de couches Out

// Destructeur
~Layer();

// Fixer les parametres
Booleen SetParam(double, double, double, double);
// Seuils
// Vitesse d'apprentissage
// Vitesse de propagation

// Obtenir le nombre de neurones de la couche
uint GetNbrNeurone();
//Verifie si tous les neuronesde la couche ont calcules leurs activations
Booleen ActivationDsCoucheEntreeFini();

// Verifie si tous les neurones de la couche sortie on calcules leurs sorties
Booleen ActivationDsCoucheSortieFini();

// Fixer les inter-connexions
Booleen SetInterConnexions(LayerNeuronType, Layer *, MatriceDeReel *);
// Type d'interconnexion
// Adresse de la couche a interconnecter
// Matrice des inter-connexions
```

```
// Recupere le nombre d'input d'un neurone
uint GetNbrNeuronesActifs() {return NbNeuronesActifs;};
// Identificateur du neurone

// Recupere d'activation a recevoir a partir de la couche d'entree
uint GetNbrActivationA_Recevoir();

// Recupere le nombre d'input d'un neurone
uint GetNbrInputNeurone(ulong);
// Identificateur du neurone

// Recupere le nombre d'output d'un neurone
uint GetNbrOutputNeurone(ulong);
// Identificateur du neurone

// Afficher la couche
void Afficher();

// Sauvegarder les caracteristiques de la couche
Booleen Save(FILE *);

// Initialisation des neurones de la couche
Booleen InitLayer();

// Initialisation des neurones de la couche pour la reftropropagation
Booleen InitLayerForFeedback();
private:
// Numero de la couche
uint IdLayer;

// Etat de la couche
Booleen LayerState;

// Caracteristique de la couche
SDefLayer sLay;

// Type de la couche
LayerNeuronType LayerType;

struct SInterLayer
{
Layer *LayerConnect;
MatriceDeReel *InterConnexions;
};
```

```
// Nombre de couches In et de couches Out
uint NumLayerIn, NumLayerOut;

// Vecteur des couches d'entrees et de sorties
SInterLayer *InterInputConnexions, *InterOutputConnexions;

struct SNeurone { Neurone *AdresseNeurone; };

// Vecteur des neurones de la couche
SNeurone *NeuroneLayer;

// Nombre de Neurones Actifs de la couche
uint NbNeuronesActifs;
//
Booleen ActivationDsCoucheFini;
};
```

## A.3 Objet réseau

```
class NeuralNet
{
public:
// Constructeur 1
NeuralNet(char *, uint=1, uint=1, uint=1);
// Nom du fichier de sauvegarde
// Nombre de couches d'entrees
// Nombre de couches cachees
// Nombre de couches de sorties

// Destructeur
~NeuralNet();

// Fixe les parametres d'une couche : ETAPE 1
Booleen SetLayer(LayerNeuronType, uint, SDefLayer);
// Type de la couche
// Numero de la couche
// Parametres de la couche

// Fixe les connexions inter-couches : ETAPE 2
Booleen SetInterLayerConnexions(LayerNeuronType, uint, LayerNeuronType, uint,
MatriceDeReel *);
// Type de la couche In
// Numero de la couche In
// Type de la couche Out
```

```
// Numero de la couche Out
// Matrice des connexions inter-couche In-Out

// Propagation
Booleen Propagation(MatriceDeReel &, MatriceDeReel &, uint=0, Booleen=FAUX);
// Matrice [i,j] d'activation des neurones j des couches i d'entrees
// Matrice [i,j] des etats des neurones j des couches i de sorties
// Nbr de cycles pour la propagation intra-couche
// si VRAI alors Auto-Apprentissage sinon rien

// Retro-propagation
Booleen RetroPropagation(MatriceDeReel &, MatriceDeReel &, Booleen=FAUX);
// Matrice [i,j] des etats des neurones j desires aux couches i de sortie
// Matrice [i,j] des etats des neurones j retropropages aux couches i d'entree
// Effectuer le retro-seuillage

// Afficher le reseau de neurones
void Afficher();

// Calculer le taux de rappel
double TauxRappel(MatriceDeReel, MatriceDeReel);
// Sortie obtenue
// Sortie desiree

// Analyser l'etat du reseau
Booleen State() { return NetState; };

// Renommer le reseau de neurones
void Rename(char *);
// Nouveau nom

// Sauvegarde le reseau
Booleen Save();

// Initialise le reseau
Booleen InitNet();

// Initialise le reseau pour la retropropagation
Booleen InitNetForFeedback();
private:
// Initialisation
void Init();

// Charge le reseau
Booleen Load();
```

```
// Teste si le reseau existe
Booleen Existe();

// Etat du reseau
Booleen NetState;

struct SLayer
{
Layer *AdresseCouche;
};

// Recuperer l'adresse d'une couche et sa taille
SLayer* SelectLayer(LayerNeuronType, uint&);
// Type de couche
// Nombre de couches de ce type

// Nombre de couches
uint NbrInputLayer, NbrHiddenLayer, NbrOutputLayer;

// Vecteur des adresses des couches
SLayer *InputLayer, *HiddenLayer, *OutputLayer;

// Nombre d'inter-connexions
uint NbrInterConnexions;

struct SListMatriceDeConnexions
{
Layer *LayerIn, *LayerOut;
MatriceDeReel *AdresseMatrice;
SListMatriceDeConnexions *Suivant;
};

// Liste des matrices de connexions
SListMatriceDeConnexions *ListeMatriceDeConnexions;

// Nom du fichier
char *NomRN;
};
```

## A.4 Objet matrice de réels

```
class MatriceDeReel
{
public:
```

```
// Constructeur 1
MatriceDeReel(ulong, ulong, double =0.0);
// Taille maximale de la matrice
// Valeur de l'initialisation

// Constructeur 2
MatriceDeReel(FILE *);
// Fichier

// Constructeur 3
MatriceDeReel(GestionBaseDocumentaire *, Booleen);
// Adresse de l'objet base documentaire
// Etat, VRAI si matrice des liens d'indexation, FAUX si cooccurrents

// Destructeur
~MatriceDeReel();

// Modificateur
Booleen Set(ulong, ulong, double);
// indices
// nouvelle valeur

// Accesseur
 Booleen Get(ulong, ulong, double &);
// indices
// valeur recuperee

// Accesseur
 Booleen GetColonne(ulong, ulong, double &);
// indices
// valeur recuperee
// Accesseur non nul
 Booleen GetNonNulEnLigne(ulong , ulong &, double &, Booleen &);
// indices
// valeur recuperee
// booleen indiquant la fin de la liste

// Accesseur non nul
 Booleen GetNonNulEnColonne(ulong &, ulong , double &, Booleen &);
// indices
// valeur recuperee
// booleen indiquant la fin de la liste


//Liste Accesseur
Booleen GetListe(ulong, double *);
```

```
// indices
// liste indexations recuperee

// Affichage de la matrice
Booleen Afficher(char *);
// message

// Affichage de la matrice avec une plus grande precision
Booleen AfficherPrecision(char *);
// message

// Affichage de la matrice
Booleen AfficherNonNul(char *);
// message

// Affichage de la matrice avec une plus grande precision
Booleen AfficherPrecisionNonNul(char *);
// message

// Recuperation de la taille de la matrice
ulong GetNbrLignes() { return i_max; };
ulong GetNbrColonnes() { return j_max; };

// Recuperation du nombre de lignes vides d'une colonne
ulong GetNbrLignesVides(ulong, double=0.0);
// indice de colonne
// valeur vide

// Recuperation du nombre de colonnes vides d'une ligne
ulong GetNbrColonnesVides(ulong, double=0.0);
// indice de ligne
// valeur vide

// Recuperation du nombre de lignes pleines d'une colonne
ulong GetNbrLignesPleines(ulong, double=0.0);
// indice de colonne
// valeur vide

// Recuperation du nombre de colonnes pleines d'une ligne
ulong GetNbrColonnesPleines(ulong, double=0.0);
// indice de colonne
// valeur vide

// Sauvegarde de la matrice
Booleen Save(FILE *);
```

```
Booleen Existe();

private:
// Liberation de la memoire
void Free();

// Tailles de la matrice
ulong i_max, j_max;

// Adresse de la matrice
double *AdresseMatrice;

// Adresse de la base documentaire
GestionBaseDocumentaire *AdresseBaseDoc;

// les colonnes et les lignes d'une matrice creuse
LienRecord *LesLignes;
LienRecord *LesColonnes;

// Booleen d'etat de la matrice (liens d'indexation ou cooccurrence)
Booleen bLienIndex;
};
#endif
```

# Bibliographie


[1] D., J. Harper, & C., V. Van Rijsbergen, *An evaluation of feedback in document retrieval using co-occurrence data.* Journal of Documentation, 34(4), 189-216 1978.

[2] S., E Roberston, & K. Sparck Jones *Relevance weighting of search term* Journal Of American Society for Information Science, 27(3), 129-146, 1976 .

[3] J. Rocchio, *Relevance Feedback in Information Retrieval.* in The Smart System experiments in Automatic Document Processing, G. Salton, editor, Prentice-Hall, Inc., Englewood Cliffs, NJ, 313-323. 1971

[4] D. Harmann *Relevance Feedback Revisited.* Paper presented at the 15th Annual Internationnal ACM SIGIR Conference on Research and Development in Information Retrieval Copenhague, Denmark, 1992

[5] Y. Qui, & H., P. Frei *Concept based query expansion..* ACM/SIGIR'93 Pittsburg, 1993.

[6] G. Salton, *Improving Retrieval Performance by Relevance Feedback.* Technical report 1988.

[7] G. Salton, *The SMART Retrieval System.* Englewood Cliffs, N. J.: Prentice-Hall, Inc, 1970.

[8] G. Salton, & C. Buckley, *Term weighting approaches in Automatic Text Retrieval.* Technical report 1988.

[9] G. Salton & M., J. McGill, *Introduction to Modern Information Retrieval* McGRAW-HILL International Book Company (USA). 1983.

[10] M. D Gordon, *The necessity for adaptation in modified boolean document retrieval systems.* Information Processing and Managment, 24(3):339-347, 1988.

[11] M. Boughanem, *Systeme de recherche d'informations : d'un modele classique a un modele connexionniste* These de Doctorat de l'universite Paul Sabatier de Toulouse Decembre 1992

[12] M. Boughanem, *MercureO2 : Système de reherche d'informations orienté objet basé sur l'approche connexonniste* Rapport de recherche MSI95-05 décembre 1995.

[13] D. Harman, *Overview on first TREC conference.* ACM SIGIR Conference on research and developpement in Information Retrieval, Pitssburg 1993.

[14] D. Rumelhart & G., E. Hinton & R. Williams, *Learning internal representations by error propagation.* In Parrallel Distributed Proceedings, Vol.1 D. Rumelhart, J. McClelland eds MIT Press.

[15] D. O Hebb *Learning internal representations by error propagation.* In Parrallel Distributed Proceedings, Vol.1 D. Rumelhart, J. McClelland eds MIT Press.

[16] M., F. Porter, *An algorithm for suffix stripping.* program 14(3) 10-37 1980.

[17] C. Soule-dupuy, *Les systèmes de recherche d'informations : Le système Videotex INFODIAB : Mécanismes d'indexaton et d'interrogation. .* Thése de Doctorat de l'université Paul Sabatier de Toulouse Fevrier 1990.

[18] S. Marcel, *Un moteur de type réseau de neurones pour la révision des connaissances dans un système tuteur intelligent.* DEA de l'Université Paul Sabatier, Toulouse III, 1996