3.1.4 Matrice d'adjacence :

Plusieurs représentations sont possibles pour les graphes. Elles ne sont pas équivalentes, et le choix d'une représentation adaptée au problème à résoudre permet d'obtenir une meilleure efficacité des algorithmes utilisés. On distingue ainsi la représentation par matrice d'adjacence, par matrice d'incidence sommets-arcs (ou sommets-arêtes), et par liste d'adjacence. Nous utilisons la matrice d'adjacence, car cette représentation permet de calculer simplement certaines caractéristiques des graphes, telles que la distance entre les sommets. La matrice d'adjacence d'un graphe G = {S, A} est la matrice M(G) dont les coefficients $m_{i,j}$ sont définis par :

$$m_{i,j} = \begin{cases} 1 \text{ si } (s_i, s_j) \in A, c'est - à - dire \text{ si les noeuds } s_i \text{ et } s_j \text{ sont agjacents} \\ 0 \text{ si } (s_i, s_j) \notin A \end{cases} \quad (19)$$

La Figure 3.2 représente un graphe ainsi que la matrice d'adjacence qui lui correspond.

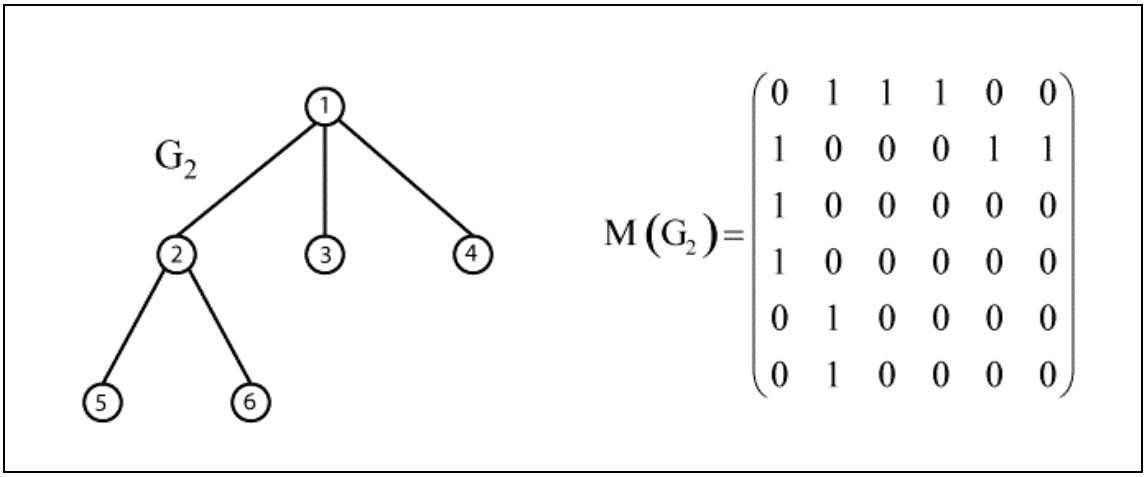

Fig.3.2 : Représentation d'un graphe par sa matrice d'adjacence

Si *n* est le nombre de nœuds du graphe, cette matrice est de taille (*n*, *n*), et chaque ligne ainsi que chaque colonne correspond à un nœud particulier. Puisque la matrice dépend de la numérotation des nœuds du graphe, la représentation n'est pas unique (il existe *n*! possibilités). Il est cependant possible de passer d'une représentation à une autre par permutation de lignes et de colonnes.

La matrice d'adjacence permet d'énumérer les chemins du graphe, et d'en déduire la connexité, la cyclicité ainsi que les distances entre les nœuds. On utilise pour cela le produit

matriciel. Par exemple, la matrice $\mathbf{M}^2$ a pour coefficients :

$$(\mathbf{M}^2)_{i,j} = \sum_{k=1}^{n} m_{i,k}, m_{k,j}$$

Chaque composant de la somme est donc non nul ssi il existe une arête de s_i à s_k et de s_k à s_j, c'est-à-dire s'il existe un chemin de longueur 2 reliant s_i à s_j et passant par s_k. Par extension, on peut démontrer que si **M** est la matrice d'adjacence d'un graphe G dont les sommets sont numérotés de 1 à *n*, le nombre de chemins de longueur exactement l allant de s_i à s_j est le coefficient (*i, j*) de la matrice M^l.
Un algorithme permet alors de calculer la distance entre deux nœuds s_i et s_j :

$$L_{i,j} = min\,(\boldsymbol{M}^l)_{i,j} \neq 0 \qquad (20)$$

On en déduit le diamètre du graphe :

$$D = \max_{(s_i s_j)\in S^2} L_{i,j} \qquad (21)$$

Il est par ailleurs possible de déterminer la connexité d'un graphe, c'est-à-dire le nombre de composantes connexes, et d'en déduire le nombre de cycles indépendants que comporte le graphe :

$$N_{cycles} = N_{arêtes} - N_{noeuds} + connexité \qquad (22)$$

Cette représentation des graphes permet ainsi d'accéder aisément aux principales caractéristiques d'un graphe, ainsi que d'effectuer des opérations telles que la suppression d'une arête ou d'un cycle.

3.2 Apprentissage à partir de graphes : RAAMs et LRAAMs

Nous allons maintenant expliquer comment il est possible d'établir une relation entre un ensemble de données structurées, représentées par des graphes, et un ensemble de nombres réels, réalisant ainsi l'association structures-données réelles évoquée. Les premiers essais de modélisation de données structurées remontent aux mémoires associatives dynamiques. Ces essais ont par la suite conduit aux Mémoires Auto-Associatives Récursives (RAAMs), qui permettent de représenter de façon compacte les arbres, et aux RAAMs étiquetés (LRAAMs).

3.2.1 Les Mémoires Auto-Associatives Récursives

Afin de montrer la capacité des réseaux de neurones à manipuler des données structurées, Pollack a proposé un modèle, fondé sur l'idée d'une mémoire associative entre une structure et un vecteur de taille fixe [53]. Les RAAMs (pour *Recursive AutoAssociative Memory*) et leurs dérivées sont des modèles établis à partir de réseaux de neurones, qui apprennent un codage d'une structure en un vecteur, puis sont à même de décoder la structure d'origine à partir de ce vecteur avec une perte d'information minimale.

L'élément de base des RAAMs est un codeur-décodeur constitué par un réseau de neurones possédant autant de variables que de sorties ($N_e = N_s$), et dont le nombre de neurones cachés est inférieur au nombre de variables ($N_c < N_e$). Un exemple d'un tel codeur-décodeur est représenté sur la Figure 3.3. À l'issue de l'apprentissage, le système réalise une auto-association, car les sorties du modèle sont identiques aux variables : la base d'apprentissage est du type ($\{\mathbf{x}_k, \mathbf{x}_k\}$, $1 \leq k \leq n$).

Lorsque le problème admet une solution, les neurones cachés réalisent un codage du vecteur de variables **x** sous forme compacte **f**(**x**) (car $N_c < N_e$), où **f**(**x**) désigne le vecteur des sorties des neurones cachés. Ce vecteur **f**(**x**) peut ensuite être décodé, afin de retrouver en sorties les données d'entrée.

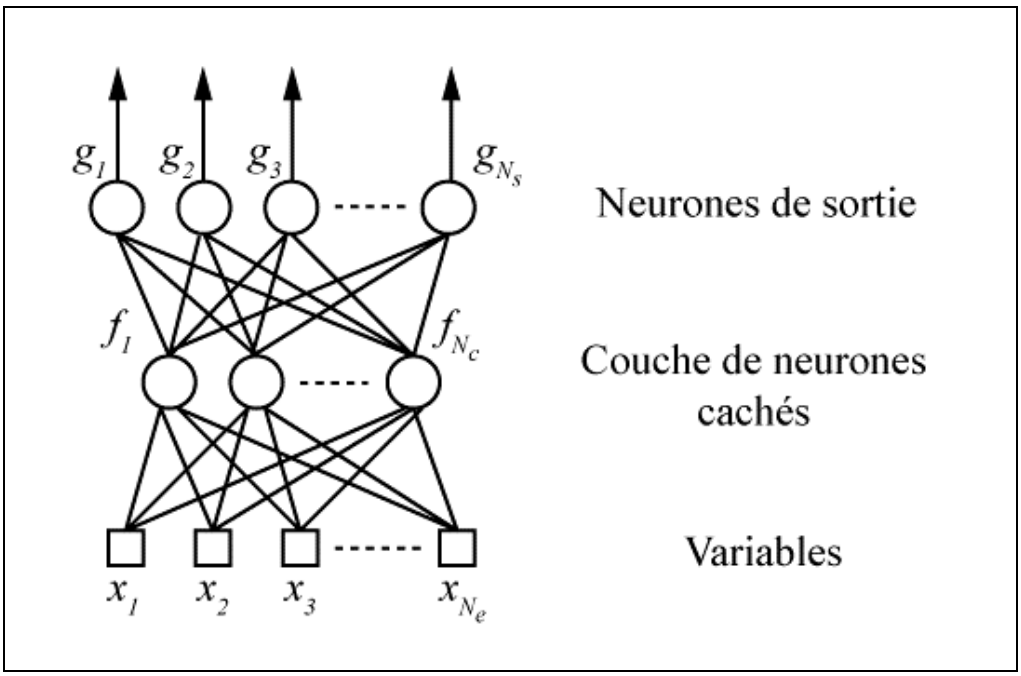

Fig.3.3 : Codeur-décodeur constitué de neurones formels

Cette unité codeur-décodeur peut être utilisée pour effectuer le codage d'un arbre, de différentes manières. Dans l'article [54], le codage s'effectue de la façon suivante : les feuilles de l'arbre, c'est-à-dire les noeuds sans enfant, sont tout d'abord codées, puis leurs parents et les noeuds suivants, de façon récursive, jusqu'au noeud racine. Le vecteur obtenu en sortie du codeur correspondant au noeud racine est alors une représentation compacte de l'ensemble de l'arbre. Le décodage s'effectue de façon symétrique et de manière récursive, fournissant successivement le décodage du noeud racine jusqu'aux feuilles de l'arbre. Il est alors possible de réaliser un apprentissage de l'ensemble codeur-décodeur obtenu, de façon à retrouver en sortie un vecteur égal au vecteur d'entrées. Considérons par exemple l'arbre binaire ((**A**,**B**),(**C**,**D**)) de la Figure 3.4 où **A**, **B**, **C** et **D** sont des vecteurs de taille identique.

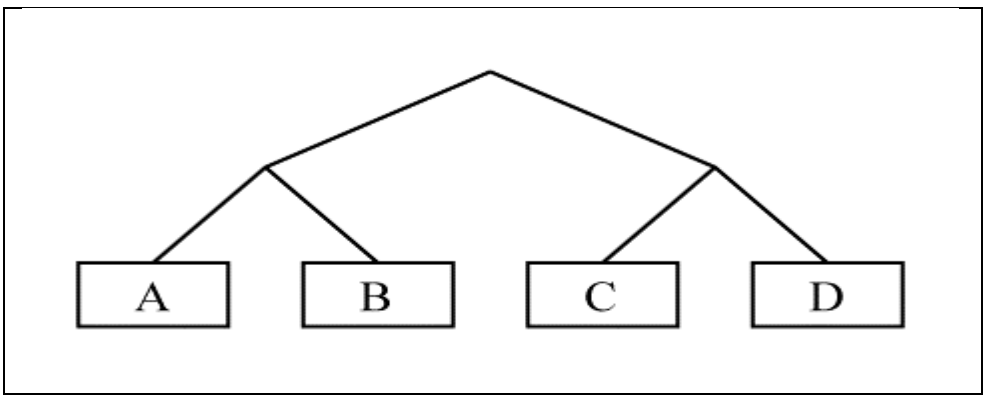

Fig.3.4 : Arbre binaire

A et **B** sont tout d'abord codés en une représentation R_1, **C** et **D** en une représentation R_2, puis R_1 et R_2 sont codés à leur tour en R_3, qui est une forme compacte de l'arbre. Pour retrouver l'information initiale, il faut alors décoder R_3 en (R'_1 , R'_2), à leur tour respectivement décodés en (**A'**,**B'**) et (**C'**,**D'**). L'apprentissage est séquentiel : il se fait tout d'abord sur le codeur – décodeur pour auto-associer (**A**,**B**) à (**A**,**B**), puis sur la paire (**C**,**D**), et enfin sur la paire (R_1,R_2). Ce type de codage pose alors le problème de la « cible mobile» : les représentations des vecteurs non-terminaux (dans le cas de l'exemple évoqué, R_1 et R_2), donc une partie de la base d'apprentissage, changent au cours de l'apprentissage. La convergence de celui-ci n'est alors plus garantie.

Pour nous affranchir de ce problème, nous proposons une méthode de codage légèrement différente. Le principe de base consiste à créer, à partir de plusieurs codeurs-décodeurs tels que ceux précédemment décrits, un modèle dont la structure est isomorphe à celle des arbres à coder, à partir de réseaux de neurones à poids partagés. Ce modèle effectue alors, avec le même jeu de paramètres, le codage des paires (**A**,**B**), (**C**,**D**) et (R_1,R_2).

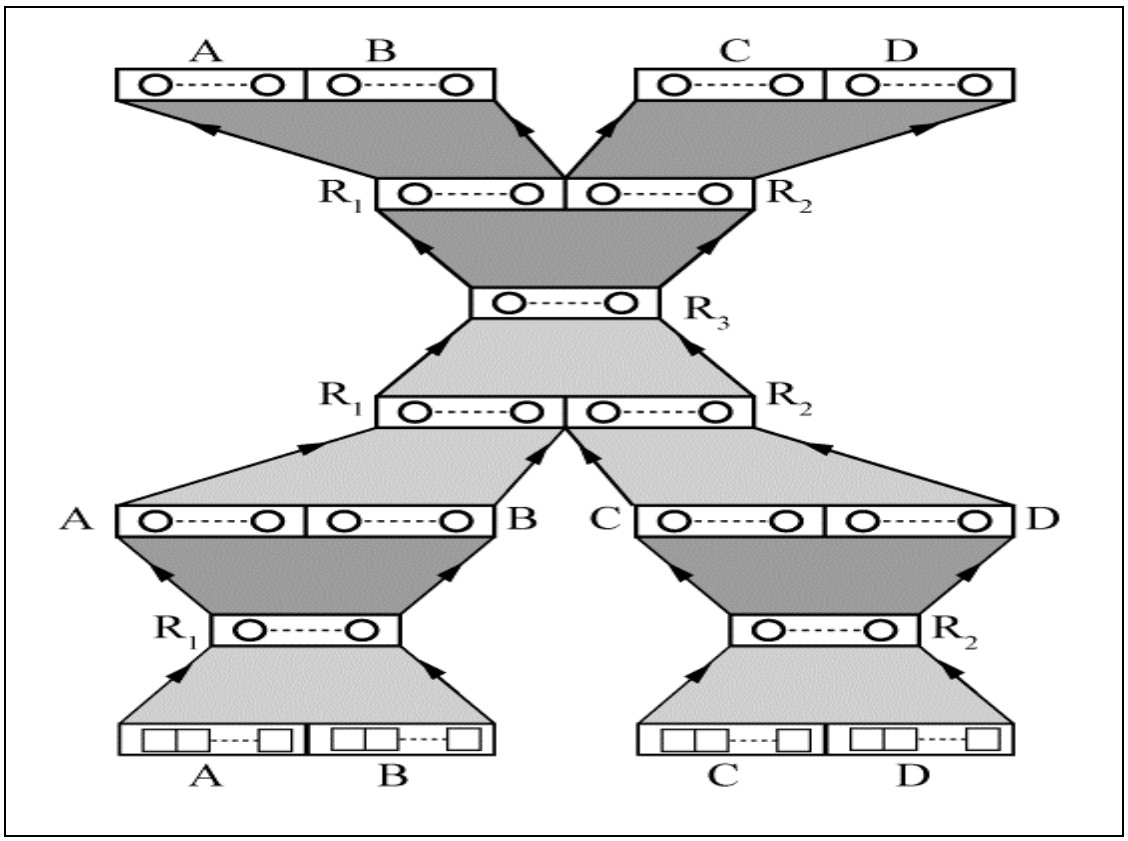

Fig.3.5: Modèle associé au graphe de la Fig.3.4

La Figure 3.5 représente l'ensemble « codeurs-décodeurs » correspondant au modèle associé à l'arbre de la Figure 3.4. Les connexions ne sont pas détaillées, mais représentées par des

zones grisées : les zones grisées de même teinte correspondent à des connexions de même vecteur de paramètres. Sur ce schéma, les carrés n'effectuent aucun calcul. On peut remarquer que la structure de ce réseau est effectivement identique à celle de l'arbre qui est codé.
Les zones grisées peuvent représenter non pas une seule couche de paramètres, mais deux couches de paramètres et une couche de neurones cachés. Supposons que les zones grisées représentent une seule couche de paramètres. Le nombre de neurones cachés doit alors être égal au nombre de neurones nécessaires pour coder une feuille de l'arbre. Or, la complexité de la relation établie entre les entrées et les sorties, qui est liée au nombre de neurones cachés, n'est en général pas liée au nombre de neurones nécessaire au codage des feuilles. Il est possible de se libérer de cette contrainte en intercalant une couche cachée supplémentaire dans le codeur et/ou le décodeur. Dans ce cas, les zones grisées ne représentent plus une seule couche de paramètres, mais deux, ainsi qu'une couche cachée. Les zones grisées de même teinte correspondent alors à des paramètres et à des neurones identiques.

Les RAAMs permettent également de coder des séquences (elles sont alors appelés SRAAMs, pour Sequential RAAMs). Considérons par exemple la séquence (x_1,x_2 ,x_3) représentée sur la Figure 3.6. Le codage est récursif : si l'on note R_x la représentation codée d'un vecteur x, les éléments d'entrée des codeurs successifs sont (0 x_1), (R_{x_1} x_2) et ($R_{x_1x_2}$ x_3).

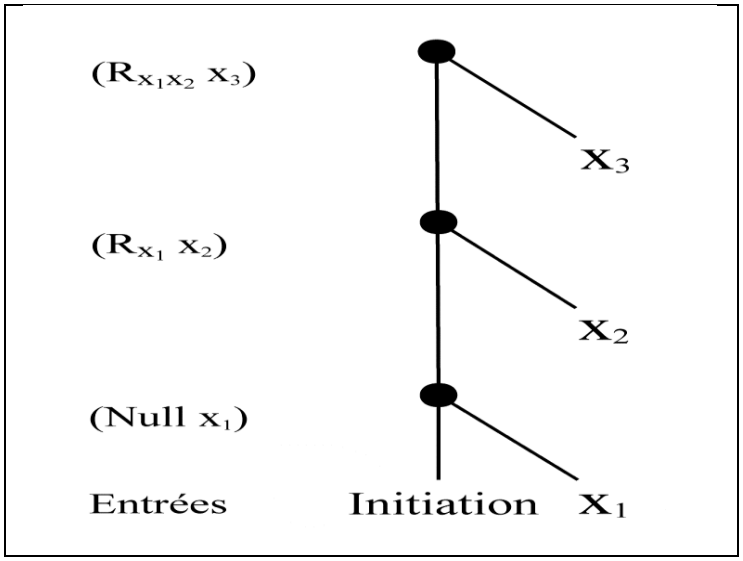

Fig.3.6 : Codage de la séquence (x_1, x_2, x_3) par une SRAAM

La couche d'entrée est ainsi composée de deux sous-éléments, l'un contenant le vecteur de variables correspondant au noeud à coder (c'est-à-dire x_1, x_2, x_3), l'autre au vecteur nul, puis,

au fur et à mesure du codage, à un pointeur vers le noeud enfant, qui est la représentation de la sous-séquence déjà codée (R_{x_1} ou $R_{x_1x_2}$).

3.2.2 Les Mémoires Récursives Auto-Associatives Étiquetées

Les Mémoires Récursives Auto-Associatives Étiquetées (LRAAMs, pour Labeling RAAMs) [54] sont des RAAMs qui permettent de tenir compte des étiquettes d'un graphe. Le principe des LRAAMs consiste à considérer deux types d'entrées pour chaque noeud : une partie des entrées correspond aux pointeurs vers les enfants du noeud dans le graphe, comme précédemment, et une seconde partie correspond aux étiquettes des noeuds.

À chaque noeud du graphe est ainsi associée une entrée, qui comporte un vecteur de variables (qui peuvent être codées de façon binaire) associé à ses étiquettes, et un ensemble de pointeurs vers ses enfants dans le graphe, qui sont des nombres réels, et dont le nombre est égal à son degré sortant. Ces données constituent donc les entrées des codeurs associés à chaque noeud. Les SRAAMs sont en ce sens des LRAAMs, pour lesquelles il n'y a qu'un seul pointeur.

Ceci soulève une nouvelle fois le problème de la cible mobile, car les pointeurs des nœuds non-terminaux sont calculés lors de l'apprentissage, donc varient. Ce problème peut être évité en utilisant la même idée que celle présentée dans le cadre des RAAMs, c'est-à-dire en associant à l'arbre étiqueté un modèle dont la structure est isomorphe à celle de l'arbre. À chaque noeud sont associés un codeur et un décodeur, et les réseaux de neurones constituant les codeurs-décodeurs des différents noeuds sont à poids partagés : ces réseaux partagent le même jeu de paramètres.

3.3 Les graph machines

Nous avons présenté, dans la section précédente, des techniques qui permettent de manipuler des données structurées par apprentissage artificiel.

Nous allons maintenant montrer comment il est possible, sur cette base, d'établir une relation entre des données structurées et des vecteurs de réels, de même que les outils de modélisation traditionnels établissent une relation entre des vecteurs de variables et les vecteurs de données à modéliser.

3.3.1 Modélisation à partir de graphes acycliques

Nous avons rappelé comment les RAAMs et les LRAAMs permettaient de trouver une représentation compacte d'un arbre, sous forme vectorielle, en sortie du codeur. Cette représentation vectorielle de l'arbre peut alors être exploitée, non plus par un décodeur pour retrouver l'information d'origine, mais comme vecteur d'entrée d'un classifieur, par exemple un réseau de neurones [55]. De plus, l'apprentissage de la représentation et celui du classifieur peuvent être réalisés simultanément : l'apprentissage du codeur est effectué dans le contexte de la « cible mobile » tandis que celui du classifieur se fait de manière conventionnelle. Nous pouvons cependant éviter le problème de la cible mobile de la manière décrite dans la section 3.2.1 : à chaque graphe est associé un réseau de structure identique, et les codeurs correspondant à chacun des noeuds sont à poids partagés.

3.3.2 Structure mathématique des graph machines

La première étape, qui constitue l'idée principale sous-jacente aux *graph machines* [56], consiste à construire, pour chacun des graphes, une fonction mathématique réalisée en combinant des fonctions élémentaires, selon une combinaison décrite par le graphe qui lui est associé.

Considérons un ensemble de graphes acycliques $G = \{G_i\}$. Pour chaque graphe G_i, on construit une fonction g^i de la façon suivante : à chaque nœud de G_i est associée la fonction paramétrée dite « fonction de nœud » f_θ, où θ est le vecteur des paramètres, qui est identique pour toutes les fonctions. La fonction associée au nœud racine peut être différente des fonctions relatives aux autres nœuds : nous la noterons F_Θ. Les fonctions f_θ sont alors composées de façon à refléter la structure du graphe : si s_j et s_k sont deux sommets du graphe G_i , tels qu'un arc part de s_j et arrive en s_k, alors le résultat de la fonction associée au nœud s_j est argument de celle associée au nœud s_k.

La fonction de nœud d'un nœud s_k est ainsi de la forme :

$$f_\theta(z_k) = f_\theta(z_0, v_k, x_k). \qquad (23)$$

Les arguments de la fonction sont de plusieurs types :

- z_0 est une constante égale à 1.

- v_k est un vecteur dont les composantes sont égales aux valeurs prises par les fonctions associées aux nœuds enfants du nœud s_k. Puisque la fonction f_θ est la même pour tous les nœuds, ce vecteur doit avoir un nombre fixe de composantes, que nous définissons ainsi :

  $M_i = \text{argmax}_k\, d_k^+$, où d_k^+ est le degré sortant du nœud s_k.

  Pour un nœud s_k tel que $d_k^+ < M_i$, les composantes superflues de v_k sont égales à 0.

- x_k est un vecteur optionnel qui apporte de l'information sur le nœud : ce sont les étiquettes du nœud.

z_k est ainsi un vecteur de taille D_i tel que :

$$D_i = 1 + M_i + |x_k| \qquad (24)$$

Sa première composante z0 vaut 1, les composantes 2 à $d_k^+ + 1$ sont les valeurs prises par les fonctions f_θ des nœuds enfants. Si $d_k^+ < M_i$, c'est-à-dire si le nombre de nœuds enfants du nœud s_k est inférieur au maximum M_i possible, les composantes $d_k^+ + 2$ à $M_i + 1$ sont nulles. Enfin, les composantes $M_i + 1$ à D_i sont celles du vecteur x_k.

La fonction paramétrée, appelée *graph machine*, relative au graphe G_i, est finalement de la forme :

$$g^i_{\theta,\Theta} = F_\Theta\,(z_r\,) \qquad (25)$$

Où $\mathbf{z}_r$ est le vecteur des arguments de la fonction associée au nœud racine.

Lorsque de telles fonctions sont construites pour l'ensemble des graphes $G = \{Gi\}$, les fonctions de nœud f_θ sont identiques pour tous les nœuds d'un même graphe, mais aussi pour tous les graphes. Il est donc nécessaire de s'assurer que les vecteurs $\mathbf{z}$ sont de dimension D fixée quel que soit le graphe considéré, c'est-à-dire $1+M_i + |\mathbf{x}| = D$ pour tout i.

Il faut donc :

- Que le vecteur $\mathbf{x}$ fournisse les mêmes types d'informations pour tous les graphes (et soit ainsi de taille fixe) ;
- Que le vecteur $\mathbf{v}$ soit de dimension M fixe. Il suffit de choisir $M = \max_i M_i$.

3.3.3 Les étiquettes

Le vecteur **x** fournit des informations sur chacun des noeuds du graphe : il correspond aux étiquettes des LRAAMs. Ces informations peuvent être codées de différentes façons. Si la propriété caractérisée par une étiquette est quantitative (si elle mesure par exemple la taille des régions d'une image), il est pertinent d'affecter une seule entrée à cette étiquette, c'est-à-dire une valeur du vecteur **x**, qui varie selon le noeud considéré et la valeur de l'étiquette associée. Au contraire, si cette propriété n'agit pas de façon quantitative sur le problème modélisé, et que l'ensemble des valeurs prises est borné et fini (par exemple les couleurs possibles des régions d'une image), on a affaire à une variable catégorielle : un codage « un parmi *n* » est mieux adapté [56].

3.4 L'apprentissage des graph machines

3.4.1 Propriété d'approximation universelle

Les réseaux de neurones sont des approximateurs universels, ce qui signifie que toute fonction bornée, suffisamment régulière, peut être approchée dans un domaine fini de l'espace de ses variables, avec une précision arbitraire, par un réseau de neurones possédant une couche de neurones cachés et un neurone de sortie linéaire.

Il a été montré que cette propriété peut être étendue aux réseaux de neurones récursifs [57-59]. Elle est également valable dans le cas des *graph machines*. Celles-ci se composent en effet de deux parties : un codeur, qui fournit une représentation compacte d'un graphe sous une forme vectorielle, et une fonction (par exemple un réseau de neurones) permettant d'effectuer une classification ou une régression à partir de cette représentation vectorielle. Cette fonction possède la propriété d'approximation universelle ; la capacité d'approximation des *graph machines* dépend donc de celle du codeur.

Considérons une application F(G) de l'ensemble des arbres *n*-aires, dont les étiquettes sont des réels, vers l'ensemble des réels. Soit $\varepsilon > 0$ une précision arbitrairement choisie et $\delta > 0$ une confiance donnée. Il est alors possible de trouver une machine *M* qui possède la propriété suivante :

$$P\left[G||M(G) - F(G) > \varepsilon|\right] < \delta \qquad (26)$$

Ce résultat, prouvé dans [57-59], signifie qu'il est possible de trouver une *graph machine M* capable d'approcher la fonction *F*, pour tout arbre *n*-aire, avec une précision arbitraire.
Les *graph machines* sont donc, en théorie, bien adaptées pour établir une relation entre des données structurées et des sorties réelles. Nous allons maintenant montrer comment leur apprentissage est réalisé.

3.4.2 Utilisation des algorithmes traditionnels

Nous avons présenté dans le chapitre 2 l'apprentissage traditionnel, pour lequel la base d'exemples permettant d'estimer les paramètres du modèle g_θ est un ensemble de N couples entrées / sortie $(\{\mathbf{x}^i, y^i\}, i = 1,...,N)$. Le modèle est le même pour tous les exemples, et, lors de l'apprentissage, la fonction de coût minimisée est :

$$J(\theta) = \sum_{i=1}^{N} (y^i - g(x^i, \theta))^2 \qquad (27)$$

Lors de l'apprentissage des *graph machines*, la base d'apprentissage est constituée de N couples structure – sortie $(\{G_i, y^i\}, i = 1,\ldots,N)$. Il n'y a plus un modèle unique pour tous les exemples : à chaque exemple correspond une fonction particulière $g^i_{\theta,\Theta}$, composée à partir des fonctions paramétrées F_Θ (pour le nœud racine) et f_θ (pour les autres nœuds), de façon à refléter la structure de l'exemple $_i$. Rappelons que les fonctions F_Θ et f_θ sont les *mêmes* pour tous les exemples.
Il est alors possible de définir une fonction de coût similaire à la fonction de coût des moindres carrés traditionnelle. Cette fonction mesure les écarts entre les observations et les valeurs prédites par le modèle, et peut comporter des termes de régularisation :

$$J(\theta,\Theta) = \sum_{i=1}^{N} (y^i - g^i_{\theta,\Theta})^2 + \lambda_1 \|\Theta\| + \lambda_2 \|\theta\| \qquad (28)$$

où λ_1 et λ_2 sont des constantes de régularisation correctement choisies. La régularisation vise à limiter l'amplitude des paramètres, pour éviter un surajustement du modèle.

La minimisation de la fonction de coût s'effectue de la même manière que lors d'un apprentissage classique, en modifiant les paramètres de façon itérative en fonction de son gradient.

La méthode des poids partagés permet le calcul de celui-ci. Sa *k*-ième composante, c'est-à-dire la dérivée de la fonction de coût par rapport à la composante *k* du vecteur **θ**, est :

$$\frac{\partial J(\theta,\Theta)}{\partial \theta_k} = \sum_{i=1}^{N} \frac{\partial J^i}{\partial \theta_k} \quad (29)$$

où J^i est la contribution de l'exemple i à la fonction de coût.

Le terme $\frac{\partial J^i}{\partial \theta_k}$ peut être calculé grâce à la technique des poids partagés. Le jeu de paramètres **θ** est en effet le même pour tous les nœuds, donc si le graphe G_i comporte n_i nœuds, le nombre d'occurrences du paramètre θ_k dans le graphe G_i est également n_i. Ce terme peut alors s'exprimer comme la somme des contributions de chacun des nœuds :

$$\frac{\partial j^i}{\partial \theta_k} = \sum_{j=1}^{n_i} \frac{\partial J^i}{\partial \theta_k^j} \quad (30)$$

où θ_k^j désigne le paramètre θ_k qui correspond au nœud *j*.

Le gradient s'exprime finalement ainsi :

$$\frac{\partial J(\theta,\Theta)}{\partial \theta_k} = \sum_{i=1}^{N} \sum_{j=1}^{n_i} \frac{\partial J^i}{\theta_k^j} \quad (31)$$

Lorsque les fonctions F_Θ et f_θ sont des réseaux de neurones, ce gradient peut être calculé par rétropropagation, de la manière usuelle. Dans d'autres cas, il est possible d'avoir recours à des méthodes numériques pour ce calcul.

Les algorithmes traditionnels de descente du gradient, tels que Levenberg-Marquardt, BFGS ou le gradient conjugué, peuvent alors être mis en œuvre pour minimiser la fonction de coût.

3.4.3 Sélection de modèle

Les outils que nous avons passés en revue au paragraphe 2.2.2 du chapitre 2 permettent de sélectionner les modèles présentant les meilleures performances de généralisation. Si la validation croisée et le leave-one-out réel peuvent être mis en œuvre de la même manière qu'en modélisation traditionnelle, l'utilisation des leviers et du leave-one-out virtuel n'est pas aussi immédiate : les expressions de la fonction de coût et du gradient ne sont en effet pas les mêmes, et le calcul des leviers n'est pas applicable tel quel. Cette méthode peut cependant être étendue aux *graph machines*.

3.5 Modélisation à partir de graphes cycliques

Nous avons choisi une approche qui consiste à transformer les graphes cycliques en graphes acycliques tout en tenant compte de la présence des cycles du graphe initial. Cette méthode nous permet ainsi de modéliser indifféremment des structures cycliques ou acycliques.

3.5.1 Transformation de graphes quelconques en arborescences

Pour que la structure d'un graphe connexe quelconque puisse être exploitée par la méthode présentée, il est nécessaire de transformer celui-ci en arborescence, donc de rendre ce graphe acyclique et de choisir un nœud racine, ce qui oriente implicitement l'arborescence obtenue. Nous avons décidé d'effectuer cette transformation en appliquant un algorithme dont sa première étape consiste à sélectionner, parmi les nœuds du graphe, celui qui sera le nœud racine de l'arborescence, grâce à un premier algorithme qui numérote les nœuds de façon canonique et unique, selon des critères tels que leurs distances aux autres nœuds du graphe et leur degré. Le nœud choisi comme racine du futur arbre est celui qui porte le numéro 1 ; il est un nœud central du graphe. D'autres critères peuvent être pris en considération, en fonction du type de structures étudiées, et des informations fournies par les étiquettes des nœuds. Par exemple, lorsque les graphes représentent des images, et les nœuds des régions de ces images, il peut être nécessaire d'ajouter un critère pour tenir compte de la taille de ces régions, fournie par les étiquettes, lors de la numérotation canonique des nœuds. Les nœuds des graphes

associés à des molécules chimiques sont numérotés suivant les critères détaillés dans [60]. Dans une seconde étape, les graphes sont transformés en graphes acycliques si nécessaire. Il faut pour cela supprimer une arête pour chaque cycle du graphe, tout en veillant à ce que celui-ci reste connexe. Un second algorithme effectue le choix de ces arêtes à supprimer, en sélectionnant les plus distantes du nœud racine. L'arborescence alors obtenue est orientée du nœud central vers les extrémités du graphe.

Une particularité importante des graph machines est la façon de gérer les cycles : bien que certaines arêtes soient supprimées pour rendre les graphes acycliques, l'information correspondant à leur présence ne l'est pas. Elle est en effet implicitement conservée, par l'intermédiaire des étiquettes, qui fournissent le degré, dans le graphe initial, des deux nœuds reliés par l'arête supprimée. Considérons ainsi les graphes de **la Figure 3.7.**

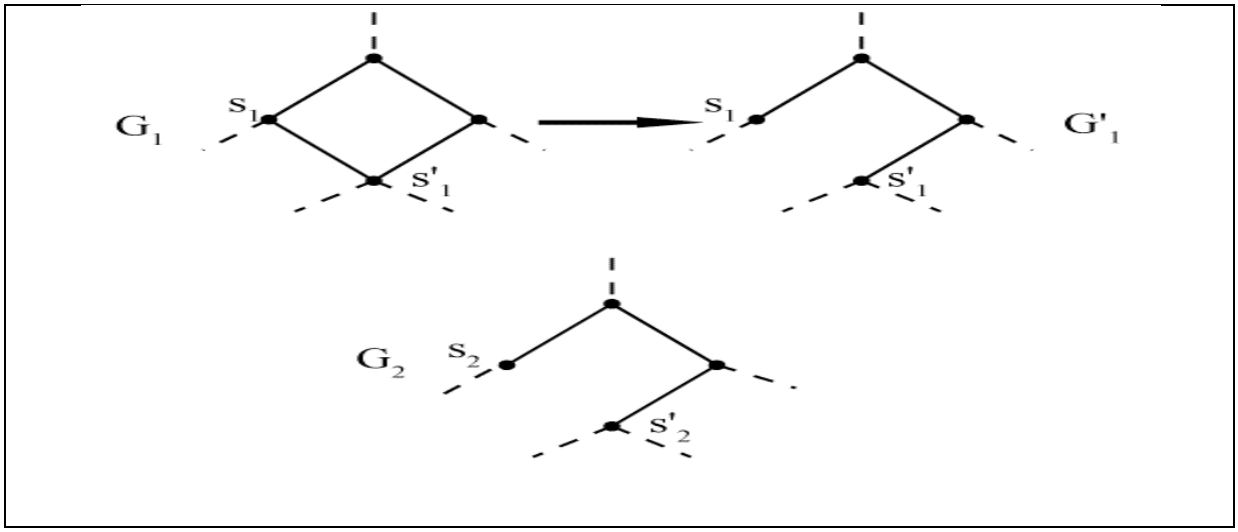

Fig.3.7 : Graphes cycliques et acycliques différenciés par leurs degrés

Le graphe G1 est un graphe cyclique, qui après ouverture devient le graphe G'_1. Le graphe G_2 est acyclique, et semble identique à G'_1 , mais il diffère de celui-ci par ses étiquettes : tandis que les nœuds s_1 et s'_1 sont respectivement de degrés d_1 et d'_1, les nœuds s_2 et s'_2 sont de degrés d_1 -1 et d'_1 -1. Ainsi, un graphe acyclique après suppression d'un ou plusieurs cycles n'est pas identique à un graphe acyclique de même structure, mais n'ayant pas subi de transformation, grâce à l'information contenue dans le degré des nœuds, et portée par les étiquettes.

La Figure 3.8 illustre comment le graphe G_3, qui comporte quatre cycles, est transformé en arborescence. Les numéros affectés aux nœuds lors de la numérotation canonique figurent près des nœuds du graphe G_3. Le nœud 1 est le centre du graphe : c'est le nœud pour lequel la

distance maximale aux autres nœuds est la plus faible, et de plus fort degré. Il est alors choisi comme nœud racine. Le graphe G_3 comporte 4 cycles : il faut donc supprimer 4 arêtes. Le cycle formé par les nœuds 1, 2 et 4 constitue un exemple simple : l'arête 2-4 étant la plus éloignée du nœud racine, c'est celle qui est supprimée. Les autres cycles sont ouverts de la même manière, et l'arborescence obtenue, orientée du nœud racine aux feuilles, est le graphe G'_3 . Les noeuds initialement reliés dans le graphe G_3 conservent leur degré, malgré la suppression des arêtes : ces degrés sont indiqués sur le graphe G'_3 par les chiffres en gras et en italique à gauche des nœuds.

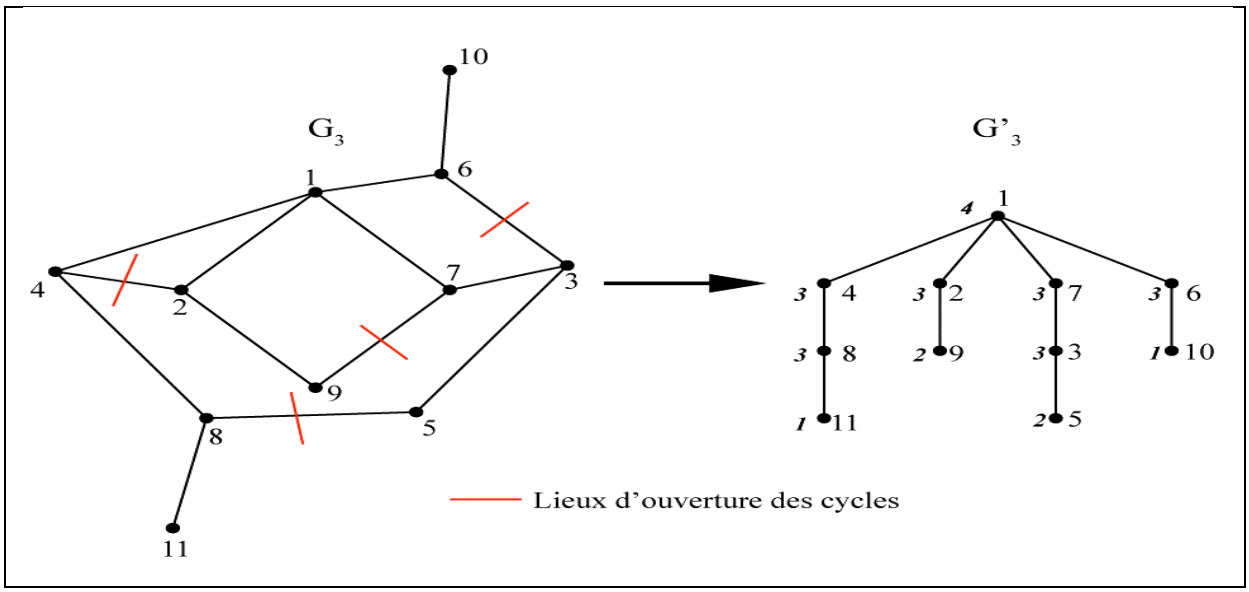

Fig.3.8 : Transformation d'un graphe comportant plusieurs cycles en graphe acyclique

Les graphes cycliques peuvent alors être codés par les *graph machines* de la même manière que les graphes acycliques.

3.5.2 Méthode alternative de modélisation à partir de graphes cycliques

Il est également possible de modéliser les graphes cycliques sans les rendre acyclique. La construction des *graph machines* associées nécessiterait, de la même façon que pour les graphes acycliques, de choisir un nœud racine et d'orienter le graphe. Considérons par exemple les graphes de la Figure 3.9, où G_4 est la structure d'origine et G'_4 le graphe cyclique orienté correspondant, dont le nœud 1 est la racine.

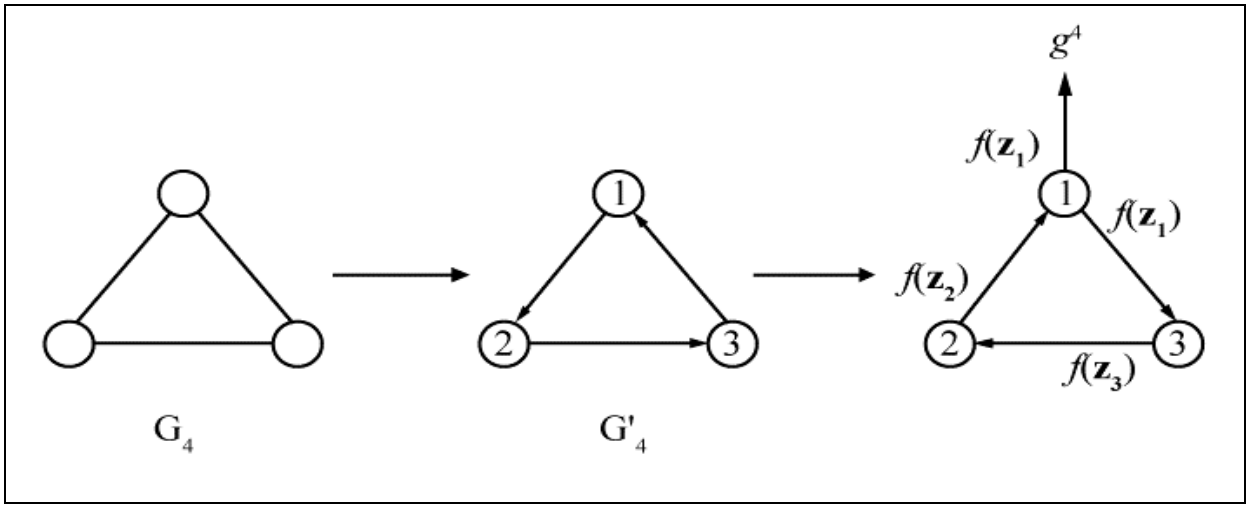

Fig.3.9 : Alternative pour modéliser un graphe cyclique - graphe cyclique simple

La méthode utilisée dans le cas de graphes acycliques, et appliquée au graphe G'_4 , donne alors la relation :

$$g^4 = \underline{f(z_1)} = f(f(z_2), x_1) = f(f(f(z_3), x_2), x_1) = f\left(f\left(f\left(\underline{f(z_1)}, x_3\right), x_2\right), x_1\right) \quad (32)$$

Cette relation ne peut être vérifiée que pour une certaine forme de fonction f donnée, qu'il est possible de déterminer en résolvant l'équation récursive (53).

Un exemple de structure plus complexe est présenté sur la Figure 3.10.

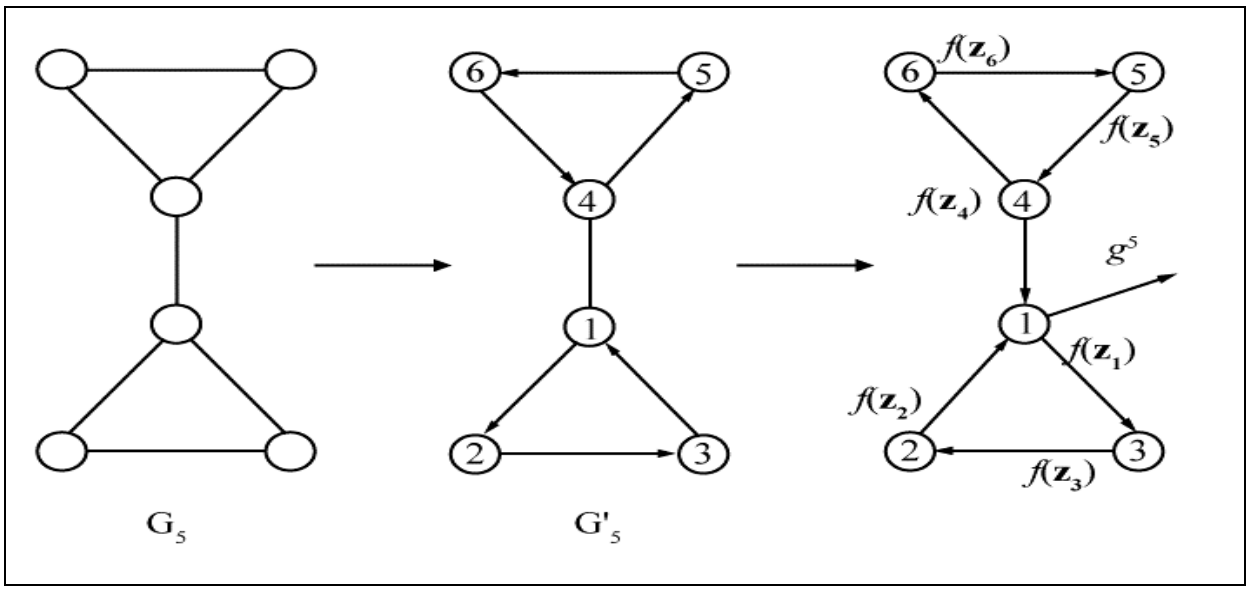

Fig.3.10 : Alternative pour modéliser un graphe comportant deux cycles

Nous voyons que dans ce cas, *f* doit satisfaire deux équations récursives différentes. Il n'est donc plus possible d'utiliser une fonction identique pour tous les nœuds.

L'étude d'une base constituée de *N* structures nécessite ainsi la résolution de *N* équations récursives, et conduit à *N* différentes fonctions de nœud racine F^i , $i \in [1, \mathrm{N}]$. Les inconvénients de cette méthode sont donc multiples :

- La résolution des équations récursives, qui peuvent devenir très complexes pour de grandes structures, augmente le temps de calcul.
- Nous ne pouvons plus utiliser les poids partagés et ne disposons pas d'un outil de calcul nous permettant, par apprentissage, de déterminer les paramètres de différentes fonctions *f*.
- La fonction de nœud n'étant plus unique, nous ne pouvons plus utiliser les leviers ni tous les outils qui s'y rapportent.

Cette approche semble donc moins efficace que celle retenue pour la modélisation des graphes cycliques.

3.6 Exemple de prédiction d'une propriété moléculaire par les graph machines (coefficient de partage eau/octanol)

Le transport, le passage à travers les membranes, la bioaccumulation ou encore l'activité pharmacologique d'une molécule peuvent être conditionnés par son partage entre une phase lipidique et une phase aqueuse, c'est-à-dire son caractère hydrophile. Celui-ci peut être quantifié par le coefficient de partage eau-octanol, noté logP, qui mesure la solubilité différentielle d'un soluté dans ces deux solvants non miscibles :

$$LogP = log\left(\frac{C_{octanol}}{C_{H_2O}}\right) \quad (33)$$

$C_{octanol}$ et C_{H2O} sont les concentrations du soluté dans l'octanol et l'eau. Le logP est ainsi utilisé dans de nombreux modèles en tant que descripteur, pour la prédiction d'effets toxiques ou biologiques ou d'interactions ligand-récepteur. Cette propriété physico-chimique peut être mesurée, mais ces mesures sont généralement longues et coûteuses. Par conséquent,

différentes méthodes de prédiction du logP ont été mises au point, et il existe un nombre important de logiciels de prédiction de cette propriété. Ceux-ci s'appuient aussi bien sur des méthodes de contribution de groupes (ACD/LogP [61], KowWin [62, 63], cLogP [64]…) que sur des régressions multilinéaires à partir de descripteurs (VLogP [65]) ou sur des réseaux de neurones (AUTOLogP [66]).

Il a été réalisé un modèle prédictif du coefficient de partage eau-octanol, par apprentissage, à l'aide d'une base de 1050 composés appartenant à diverses familles de molécules. L'erreur expérimentale sur les valeurs du logP est estimée entre 0,2 et 0,3. La modélisation et la sélection du modèle sont réalisées sur un ensemble d'apprentissage de 875 exemples, choisis de façon aléatoire. Le coût d'apprentissage et le score de leave-one-out virtuel, donnés dans le Tableau 3, sont quasiment identiques pour le modèle choisi : celui-ci ne devrait donc pas être surajusté aux données d'apprentissage.

La qualité du modèle obtenu est alors évaluée sur une base de test de 175 exemples. Les résultats d'apprentissage (EQMA) et de test (EQMT) sont comparés dans le Tableau 3 aux résultats obtenus par une méthode de régression par des réseaux de neurones, à partir de descripteurs [67]. Nous y indiquons également le nombre d'exemples dans chacune des bases d'apprentissage (N_{app}) et de test (N_{test}) qui sont également indiqué dans le tableau.

Modèle	**GM**	**Réseaux de neurones [79]**
N_{app} / N_{test}	875 / 175	980 / 105
EQMA	0,29	0,41
LOO virtuel	0,30	-
EQMT	**0,30**	0,53

Tableau 3 : Comparaison des performances de modélisation du logP des graph machines et de réseaux de neurones

Les prédictions sur la base de test sont également représentées sur la Figure 3.11.

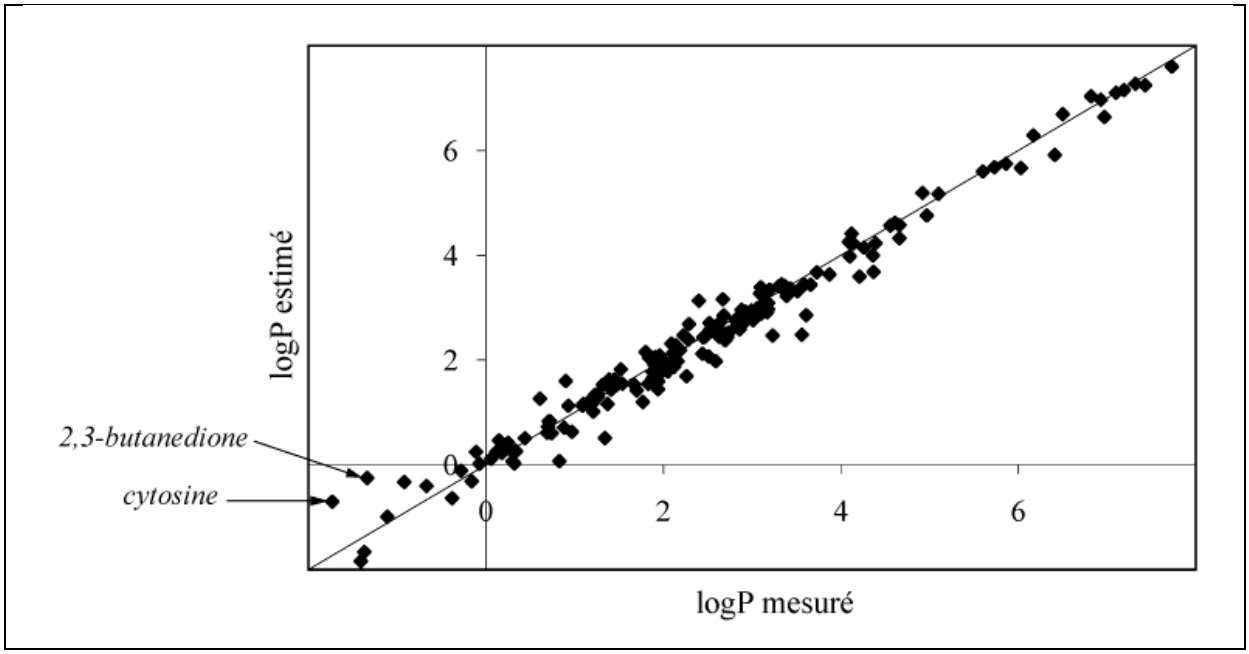

Fig.3.11 : Prédiction du coefficient de partage eau-octanol sur la base de test

Le coefficient de partage eau-octanol de ces molécules est bien prédit : les erreurs commises par le modèle sur les bases d'apprentissage et de test sont proches de l'erreur expérimentale. De plus, l'erreur de prédiction sur la base de test est quasiment identique au coût d'apprentissage, ce qui suggère à nouveau que le modèle n'est pas surajusté. Les valeurs du Log les moins bien prédites sont deux valeurs très faibles, qui se situent à la périphérie du domaine d'apprentissage [57].

3.7 Conclusion

Dans de nombreux problèmes de modélisation, les données se présentent sous la forme de structures, il est plus pertinent de tirer directement profit de la structure des données, par l'intermédiaire d'un modèle capable d'établir de façon directe une association entre ces structures et un vecteur de sorties, sachant que les données structurées se représentent aisément sous la forme de graphes.

Dans ce chapitre nous avons commencé par la présentation des graphes en tant qu'objets mathématiques. Nous avons introduit ensuite les mémoires récursives auto-associatives, premiers modèles à réaliser un codage de données structurées par apprentissage artificiel, ainsi que les *graph machines* qui permettent d'établir une relation entre des données structurées et des vecteurs de réels. Enfin un exemple de modélisations de propriétés physico-chimiques de molécule a été présenté.