L'interprétation d'un réseau de neurones en estimation du coût de logiciels

Ali Idri
ENSIAS, BP. 713, Agdal, Rabat
Université Mohamed V-Souissi
E-mail : idri@ensias.um5souissi.ac.ma

Alain Abran
Ecole de Technologie Supérieure
Montréal, Canada
Email : aabran@ele.etsmtl.ca

Samir Mbarki
BP. 133, Faculté des sciences
Université Ibn Tofail, Kénitra
E-mail : s.mbarki@caramail.com

Résumé : Dans la littérature**,** l'utilisation des réseaux de neurones en estimation des coûts de logiciels s'est limitée au développement et à l'analyse de modèles permettant de prédire l'effort de logiciels à une étape assez précoce dans leur cycle de développement et de maintenance. Cependant, , il y a encore une réticence marquée vis-à-vis de leur adoption comme une technique acceptable dans le domaine, principalement à cause de la qualification de 'boîte noire' de la modélisation par les réseaux de neurones. Pour remédier à cette critique, nous étudions ici la possibilité de donner une interprétation compréhensible à un modèle d'estimation du coût basé sur un réseau de neurones multicouche. La proposition principale de cette étude est de transformer un tel réseau de neurones en un système à base de règles floues (si-alors); l'hypothèse de cette proposition est que si les règles floues obtenues sont facilement interprétables, le réseau de neurones le serait aussi de même. Cette proposition est ensuite évaluée en utilisant les données empiriques du modèle COCOMO'81.

Abstract : The use of the neural networks to estimate software development effort has been viewed with skepticism and sometimes with hostility by the majority of the cost estimation community. Although, neural networks have shown their strengths in solving complex problems, their shortcoming of being 'black boxes' has prevented them to be accepted as a common practice for cost estimation. In this paper, we study the interpretation of cost estimation models based on a Back-propagation three multi-layer Perceptron network. Our idea consists in the use of a method that maps this neural network to a fuzzy rule-based system. Consequently, if the obtained fuzzy rules are easily interpreted in cost estimation, the neural network will be the same. Our experiment is made on COCOMO'81 dataset.

1. Introduction

Les modèles d'estimation en génie logiciel ont pour objectif de prédire des attributs de projets logiciels tels que l'effort de développement, la fiabilité logicielle et la productivité des programmeurs. La prédiction de ces attributs logiciels à une étape assez précoce dans le cycle de développement et/ou de maintenance permet de contrôler les différents aspects d'un projet logiciel. Parmi ces modèles d'estimation, ceux permettant de prédire l'effort de développement de logiciels ont fait l'objet de plusieurs travaux de recherche. L'estimation de l'effort est généralement basée sur une fonction mathématique de la forme : $Effort = a \times taille^{b}$ ou sur d'autres techniques telles que les réseaux de neurones artificiels, le raisonnement par analogie et les arbres de régression. Dans cet article, nous nous intéressons aux modèles d'estimation de l'effort utilisant les réseaux de neurones artificiels. La modélisation par les réseaux de neurones est inspirée de la structure biologique du cerveau humain. Un réseau de neurones est caractérisé par son architecture, son algorithme d'apprentissage et ses fonctions d'activation. En général, pour l'estimation de l'effort de développement d'un logiciel l'architecture, l'algorithme d'apprentissage et la fonction d'activation les plus utilisés sont respectivement le 'feed-forward' Perceptron multicouche, l'algorithme de rétro-propagation et la fonction Sigmoïde [5,13,16,17,18,19,20,21]. La plupart de ces travaux se sont intéressés à l'évaluation de la précision des estimations de l'approche neuronique en comparaison avec celles des autres techniques. Le tableau 1 résume les résultats obtenus par ces études.

Etude	Algorithme d'apprentissage	Base de données	Nombre de projets	Résultats
Venkatachalam	Rétro-Propagation	COCOMO	63	Satisfaisants
Wittig &Finnie	Rétro-Propagation	Desharnais/ ASMA	81 136	MMRE=17%
Jorgenson	Rétro-Propagation	Jorgenson	109	MMRE=100%
Serluca	Rétro-Propagation	Mermaid-2	28	MMRE=76%
Samson et al.	Rétro-Propagation	COCOMO	63	MMRE=428%
Sarinivasan & Fisher	Rétro-Propagation	Kemerer & COCOMO	78	MMRE=70%
Hughes	Rétro-Propagation	Hughes	33	MMRE=55%

Tableau 1: Sommaire des études en estimation du coût utilisant les réseaux de neurones [17]

L'utilisation des réseaux de neurones pour estimer le coût d'un logiciel a deux avantages principaux. Premièrement, ils permettent l'apprentissage automatique à partir des situations et des résultats précédents. L'apprentissage est très important pour les modèles d'estimation du coût car l'industrie de logiciels est en évolution croissante et continue. Deuxièmement, ils peuvent modéliser les relations complexes existantes entre la variable dépendante (coût ou effort) et les variables indépendantes du modèle (facteurs de coût). Cependant, l'approche neuronique a trois limitations qui l'empêchent d'être acceptée comme une pratique usuelle en estimation du coût :

- L'approche neuronique est considérée comme étant une approche 'boîte noire'. Par conséquent, il est difficile d'expliquer et d'interpréter son processus.
- Les réseaux de neurones ont été utilisés spécifiquement pour les problèmes de classification et de catégorisation, alors qu'en estimation du coût, nous traitons un problème de généralisation et non pas de classification.
- Il n'existe aucune démarche standard pour le choix des différents paramètres de la topologie d'un réseau de neurones (nombre de couches, nombre d'unités de traitement par couche, valeurs initiales des poids de connexions, etc.).

Dans ce travail, nous étudions la première limitation de l'approche neuronique. La majorité des chercheurs en estimation du coût refusent d'utiliser les réseaux de neurones à cause de leur caractère opaque, dit 'boîte noire'. En effet, les réseaux de neurones ne fournissent aucune interprétation ou explication à leur processus de raisonnement. Par conséquent, il est très difficile d'évaluer leur fiabilité. Ainsi, il est très important de fournir une interprétation compréhensible de la connaissance contenue dans l'architecture et les synapses d'un réseau de neurones afin de remédier à cette limitation. Notre proposition consiste à utiliser une méthode qui transforme un réseau de neurones en un système à base de règles floue (FRBS) équivalent. Ainsi, si les règles floues 'si-alors' obtenues sont facilement interprétables alors le réseau de neurones correspondant l'est aussi. Cette approche est ensuite testée de façon empirique en utilisant les données du modèle COCOMO'81.

Cet article est composé de six sections. Dans la section 2, nous présentons notre modèle d'estimation du coût utilisant un réseau de neurones 'feed-forward' de trois couches. Dans la section 3, nous discutons les résultats obtenus de l'application de ce modèle sur les projets logiciels de la base de données COCOMO'81. Dans la section 4, nous présentons la méthode de Benitez et al. que nous utilisons pour extraire les règles floues à partir de notre réseau de neurones. Dans la section 5, nous appliquons la méthode de Benitez sur notre réseau de neurones et nous discutons de l'interprétation des règles floues obtenues de cette application en estimation du coût de logiciels.

2. Application des réseaux de neurones à l'estimation de l'effort de développement de logiciels

Dans la littérature, plusieurs modèles de réseaux de neurones ont été développés [13]. Ils peuvent être classifiés en deux catégories principales: les réseaux de neurones 'feed-forward' où il n'y a aucune boucle dans l'architecture du réseau et les réseaux récurrents où une ou plusieurs boucles récursives apparaissent dans l'architecture du réseau. Le Perceptron multicouche utilisant l'algorithme d'apprentissage de rétro-propagation est souvent le plus adopté en estimation du coût de logiciels. Dans ce modèle, les neurones sont organisés en couches et chaque couche ne peut avoir des

connexions que vers la couche suivante. La figure 1 illustre un exemple d'un tel réseau pour l'estimation de l'effort de développement de logiciels. Le réseau produit un résultat (effort) en propageant ses entrées initiales (facteurs du coût ou attributs du projet logiciel) à travers les différents neurones du réseau jusqu'à la sortie. Chaque neurone d'une couche calcule sa sortie en appliquant sa fonction d'activation à ses entrées. Souvent, la fonction d'activation d'un neurone est la fonction Sigmoïde définie par :

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

Notre expérimentation consiste à estimer l'effort de développement de logiciels en utilisant un réseau de neurones. Dans cette expérimentation, nous utilisons la base de données COCOMO'81. Cette base de données contient 63 projets logiciels [2,3,4]. Chaque projet est décrit par 17 attributs : la taille du logiciel mesurée en KISL (Kilo Instruction Source Livrée), le mode de projet évalué par trois qualifications (organique, semi-détaché ou intégré), et les autres quinze attributs sont mesurés sur une échelle composée de six valeurs linguistiques : *'très bas', 'bas', 'nominal', 'élevé', 'très élevé' et 'extra élevé'*. Parmi ces 17 attributs, nous avons retenu la taille et 12 autres attributs dont les valeurs linguistiques sont représentées par des ensembles flous [6]. Les autres attributs ne seront pas considérés dans notre étude car leurs descriptions s'avèrent insuffisantes. Les ensembles flous associés à chaque attribut seront utilisés dans l'interprétation des règles floues 'si-alors' obtenues à partir de notre réseau de neurones.

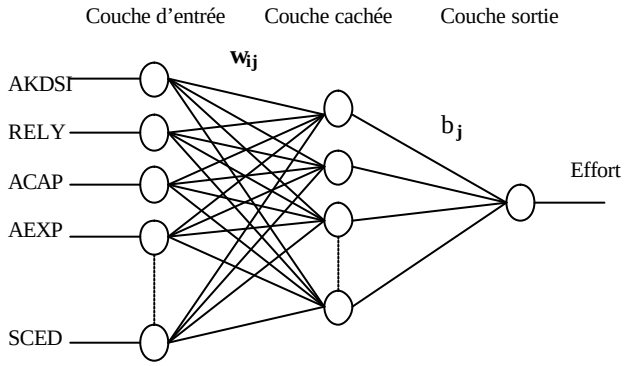

Figure 1. Architecture d'un réseau de neurone à 3 couches pour l'estimation de l'effort

La configuration d'un réseau de neurones pour l'estimation de l'effort de développement de logiciels nécessite le choix d'une architecture adéquate, un algorithme d'apprentissage et des fonctions d'activation. Dans notre cas, la méthode que nous utiliserons, pour générer les règles floues 'si-alors' à partir du réseau de neurones, exige que le réseau soit formé de trois couches et que les fonctions d'activation soient la fonction Sigmoïde pour la couche cachée et la fonction Identité pour la couche de sortie. Notre réseau de neurones a 13 entrées (facteurs du COCOMO'81) et une seule sortie (effort). Toutes les entrées ainsi que la sortie du réseau sont numériques. Toutes les entrées sont normalisées pour améliorer la performance du processus d'apprentissage du réseau [14]. L'apprentissage du réseau est réalisée en présentant les données plusieurs fois au réseau ; elle utilise l'algorithme de rétro-propagation avec un taux d'apprentissage de 3.10^{-2} et une erreur maximale de l'ordre de 10^{-5}. Finalement, les poids des différentes connexions sont initialisés avec des valeurs aléatoires très petites. D'après l'architecture de notre réseau, l'effort (sortie) est calculé par :

$$Effort = \sum_{j=1}^{h} z_j b_j \quad avec \quad z_j = f(\sum_{i=1}^{n} w_{ij} x_i)$$

où f est la fonction Sigmoïde, w_{ij} sont les poids des connexions entre la couche des entrées et la couche cachée, et les b_j sont les poids des connexions entre la couche cachée et la couche de sortie.

3. Discussion des résultats

Dans cette section, nous présentons et nous discutons les résultats obtenus de l'application de notre réseau de neurones aux projets logiciels de la base de données COCOMO'81. Nous avons développé un logiciel prototype en langage C qui implante notre réseau de neurones afin de faciliter et d'accélérer les calculs. La précision du modèle est évaluée par l'erreur relative (MRE), définie par:

$$MRE = \left| \frac{Effort_{estimé} - Effort_{réel}}{Effort_{réel}} \right|$$

Cette erreur relative est calculée pour chaque projet de la base de données COCOMO'81. En plus de l'erreur relative, nous utilisons un autre critère d'évaluation, souvent utilisé en littérature, Pred , défini par :

$$Pred(p) = \frac{k}{N}$$

où N est le nombre total de projets logiciels et k est le nombre de projets ayant une erreur relative inférieure ou égale à p. Le plus souvent, on choisit p égal à 0,25. D'autres critères issus du MRE sont utilisés : min des MREs, max des MREs, médiane des MREs et la moyenne des MREs (MMRE).
Nous avons effectué plusieurs essais avec des données empiriques pour choisir le nombre des unités cachées. Les différents essais utilisent la totalité des projets logiciels de la base de données COCOMO'81 pour l'apprentissage et les tests du réseau. Une précision acceptable a été obtenue avec 13 unités cachées et 3.10^5 itérations d'apprentissage (Tableau 2).

MRE%	Réseau de neurones avec 13 unités cachées
Max	16,67
Moy	1,50
Min	0,00

Tableau 2. Les valeurs des MREs obtenues par un réseau de neurones à 13 unités cachées

Cependant, quelques études ont illustré qu'un réseau de neurones fournit des estimations moins satisfaisantes quand les données d'apprentissage sont différentes de celles de tests [18,19]. Ceci peut être du en particulier aux trois raisons suivantes:

- Le nouveau projet est totalement différent des projets utilisés dans la phase d'apprentissage.
- Le nombre de projets utilisés dans le processus d'apprentissage est insuffisant.
- Il n'existe pas de relation entre les facteurs du coût considérés (entrées du réseau) et l'effort (sortie du réseau).

Pour confirmer l'observation citée ci-dessus, nous avons réalisé deux essais avec notre réseau de neurones. Dans le premier, nous avons aléatoirement supprimé 23 projets de la base de données COCOMO'81; ces 23 projets seront utilisés pour le test du réseau. Les 40 autres projets seront utilisés pour l'apprentissage. Dans le deuxième essai, nous avons supprimé de la base COCOMO'81 un seul projet qui sera utilisé pour le test du réseau; les 62 projets restant seront utilisés pour l'apprentissage. Le projet supprimé dans le deuxième essai est l'un des 23 projets supprimés dans l'essai 1. Le deuxième essai est répété 23 fois avec, à chaque fois, un projet différent. Le tableau 3 montre les résultats obtenus de ces deux expériences. Les différents critères dévaluation des estimations (MRE, Pred(0,25), etc.) montrent clairement que la précision augmente en fonction du nombre de projets utilisés dans l'apprentissage.

Nombre de projets : Apprentissage/Tests	**Min MRE %**	**Max MRE %**	**Moy MRE %**	**Méd. MRE %**	**Pred(0,25) %**
40/23	2,6	1188,89	203,66	92,53	13,04
62/1	2,84	432,5	84,35	53,67	34,78

Tableau 3 : Précision de prédiction du réseau de neurones en fonction du nombre de projets (apprentissage/test)

Après cette évaluation empirique de notre réseau de neurones, l'objectif est d'expliquer son fonctionnement en donnant une interprétation compréhensible à la connaissance contenue dans son architecture. Pour ceci, nous utilisons la méthode de Benitez qui transforme un réseau de neurones, comme celui utilisé dans ce travail, en un système à base de règles floues 'si-alors' [1]. Dans la section suivante, nous présentons cette méthode.

4. Equivalence entre les réseaux de neurones et les systèmes de règles floues

Depuis sa fondation par Zadeh en 1965 [22], la logique floue n'a cessé de faire l'objet de plusieurs travaux de recherches. Les systèmes à base de règles floues 'si-alors' est certainement l'une des contributions majeures de la logique floue pour la résolution de problèmes complexes. Un système à base de règles floues ('Fuzzy Rule-based system' - FRBS) est constitué principalement d'un ensemble de règles floues 'si-alors' représentant la connaissance dans le domaine. Une règle floue est une expression 'si-alors' dont la prémisse et la conséquence consistent en des propositions floues. Un exemple de règles floues en estimation du coût est : *'Si la compétence des analystes est élevée Alors l'effort de développement est faible'*. L'avantage principale des règles floues par rapport aux règles classiques est qu'elles sont facilement compréhensibles. En effet, les règles floues utilisent des valeurs linguistiques tandis que les règles classiques utilisent des valeurs numériques dans leurs prémisses et leurs conséquences. Par conséquent, plusieurs travaux de recherches ont été entrepris pour établir une équivalence entre les réseaux de neurones et les systèmes à base de règles floues [1,11]. Ces travaux ont pour objectif de représenter la connaissance encodée dans le réseau de neurones par un langage compréhensible tel que celui des règles floues 'si-alors'. Benitez et al. ont développé une méthode qui transforme un réseau de neurones, comme celui présenté dans la section précédente, en un système à base de règles floues de type Sugeno [1]. Dans ce qui suit, nous présenterons cette méthode.
Considérons un réseau de neurones de trois couches, avec la fonction Sigmoïde pour les unités cachées et la fonction Identité pour l'unité de sortie. Ce réseau de trois couches est équivalent à un système de règles floues dont les règles R_k sont associées aux paires de ses unités (cachée, sortie).

$$R_j \ : \ Si \ \sum_{i=1}^{n} x_i \, w_{ij} \ est \ A \quad Alors \ y = b_j \qquad j = 1, \cdots, h$$

où x_i sont les entrées, y_k est la sortie, w_{ij} sont les poids entre la couche d'entré et la couche cachée, b_j sont les poids entre la couche cachée et la couche de sortie, et A est un ensemble flou dont la fonction d'appartenance est la fonction Sigmoïde. Le nombre de règles floues R_j , h, est égal aux nombre des unités de la couche cachée. Afin de rendre les règles floues R_j facilement interprétables, Benitez et al. ont démontré que chacune d'elles pourra être exprimée par :

$$R_j \ : \ Si \ \ x_1 \ \ est \ \ A_j^1 \ * \ x_2 \ \ est \ \ A_j^2 \ * \cdots * \ x_n \ \ est \ \ A_j^n \quad Alors \ \ y = b_j$$

où A_j^i sont des ensembles flous obtenus de A et w_{ij}. Leurs fonctions d'appartenances sont définies par $m_{A_j^i}(x) = m_A(x w_{ij})$ et * est l'opérateur 'i-or' défini par :

$$i\text{–}or(a_1, \cdots, a_n) = \frac{a_1 \cdots a_n}{(1-a_1)\cdots(1-a_n) + a_1 \cdots a_n}$$

Selon Benitez et al. la proposition floue $x \ est \ A_j^i$ est interprétée par '*x est approximativement plus grand que 2,2/w_{ij}*' si w_{ij} est positive ou '*x n'est pas approximativement plus grand que 2,2/-w_{ij}*' si w_{ij} est négative.

5. Validation et interprétation des règles floues

Dans cette section, nous appliquons la méthode développée par Benitez et al. au réseau de neurones présenté et discuté dans les sections 2 et 3. Le réseau de neurones considéré a 13 unités cachées et utilise tous les projets de la base de données COCOMO'81 dans le processus d'apprentissage. Par conséquent, la base de règles obtenue contient 13 règles floues. La prémisse de chaque règle floue est

composée de 13 propositions floues; chacune est associée à un facteur de coût (entrées du réseau). La conséquence de chaque règle est une valeur numérique (positive ou négative). Ces 13 règles floues représentent la connaissance encodée par les poids synaptiques du réseau. L'objectif est de fournir une interprétation compréhensible de ces règles floues en utilisant nos travaux de recherches dans le domaine d'estimation du coût de logiciels [6,7,8,9,10,11]. Pour fins de discussion et d'illustration dans cet article, notre discussion se limitera, à titre d'exemple, à deux règles floues (Tableau 4).

En analysant ces deux règles, nous remarquons que la sortie de la première règle, $R1$, est positive (6049,77) alors que celle de la deuxième, $R2$, est négative (-2979,21). Ces deux valeurs représentent les poids synaptiques entre la couche cachée et la couche de sortie. Ainsi, l'interprétation naturelle que nous pouvons donner aux sorties de ces deux règles est qu'elles représentent des contributions partielles à l'effort total de développement. Elles ont la même signification que les multiplicateurs d'effort du modèle COCOMO'81. Elles peuvent augmenter (valeur positive) ou diminuer (valeur négative) l'effort total. La seule différence entre les deux est que les sorties des règles floues sont des valeurs positives ou négatives, car la fonction coût du système flou utilise l'opérateur Sommation, alors que les multiplicateurs d'effort dans COCOMO'81 sont supérieurs à 1 (augmentent l'effort) ou inférieurs à 1 (diminuent l'effort) car la fonction coût du modèle COCOMO'81 utilise l'opérateur Multiplicateur.

R1	PREMISSE		
I-OR	DATA	***est*** **~>**[1]	117,57
	VIRTmin	***n'est pas*** **~>**	30,14
	TIME	***n'est pas*** **~>**	156,28
	STORE	***n'est pas*** **~>**	155,72
	VIRTmaj	***n'est pas*** **~>**	448,88
	TURN	***n'est pas*** **~>**	25,76
	ACAP	***n'est pas*** **~>**	203,72
	AEXP	***n'est pas*** **~>**	400,37
	PCAP	***n'est pas*** **~>**	331,69
	VEXP	***n'est pas*** **~>**	1958,86
	LEXP	***n'est pas*** **~>**	921,91
	SCED	***n'est pas*** **~>**	794,68
	AKDSI	***n'est pas*** **~>**	132,27
	CONSEQUENCE		
	Y	=	6049,77

R2	PREMISSE		
I-OR	DATA	***est*** **~>**	118,46
	VIRTmin	***n'est pas*** **~>**	18,14
	TIME	***est*** **~>**	3716,39
	STORE	***n'est pas*** **~>**	573,41
	VIRTmaj	***est*** **~>**	1232,53
	TURN	***n'est pas*** **~>**	46,33
	ACAP	***n'est pas*** **~>**	393,35
	AEXP	***n'est pas*** **~>**	272,94
	PCAP	***n'est pas*** **~>**	483,23
	VEXP	***est*** **~>**	1357,35
	LEXP	***est*** **~>**	985,36
	SCED	***n'est pas*** **~>**	171,20
	AKDSI	***est*** **~>**	176,79
	CONSEQUENCE		
	Y	=	-2979,21

Tableau 4. Exemples de deux règles floues

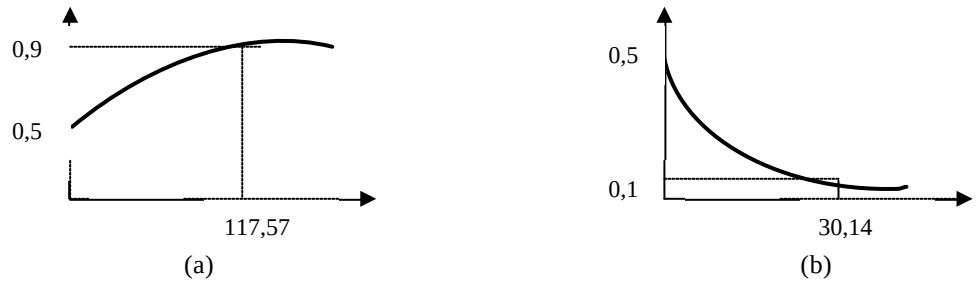

Figure 2. (a) Ensemble flou associé à la qualification 'approximativement grand que 117,57'. (b) Ensemble flou associé à la qualification 'n'est pas approximativement grand que 30,14'

La prémisse de chaque règle floue est composée de 13 propositions floues combinées par l'opérateur 'i-or'. Chaque proposition est traduite par l'expression '*x est approximativement plus grand que v*' ou '*x n'est pas approximativement plus grand que v*'. La valeur linguistique *'approximativement plus grand que v*' est représentée par un ensemble flou dont la fonction d'appartenance est celle de l'équation 1. La valeur *v* est celle dont le degré d'appartenance est égal à 0,9 dans le cas

[1] Approximativement plus grand que

'*approximativement plus grand que v*' ou 0,1 dans le cas *'n'est pas approximativement plus grand que v'*. Dans la littérature des réseaux de neurones, les valeurs 0,1 et 0,9 sont utilisées pour indiquer respectivement l'absence et la présence totale de l'activation des neurones. Dans notre application, les entrées du réseau ne peuvent avoir que des valeurs positives; par conséquent, nous utilisons seulement la partie positive des domaines des ensembles flous correspondant aux deux valeurs linguistiques '*est approximativement plus grand que v*' et *'n'est pas approximativement plus grand que v*'. La figure 2 montre deux exemples qui illustrent les ensembles flous des deux premières propositions de la règle $R1$.

L'interprétation de chaque proposition floue dépend de la signification du facteur coût qui lui est associée. Par exemple, la proposition *'Data est approximativement plus grand que 117,57*' utilise le facteur Data évalué dans COCOMO'81 par le ratio suivant (Figure 3):

$$\frac{D}{P} = \frac{Taille\ de\ la\ base\ de\ donnees}{Taille\ du\ programme\ en\ ISL}$$

l'attribut Data représente l'effet de la taille de la base de données sur l'effort de développement de logiciels. Ainsi, plus la taille de la base de données est grande plus elle influence positivement l'effort total. En utilisant notre fuzzification de l'attribut Data, nous remarquons que la valeur 117,57 appartient à la valeur linguistique 'élevé' ou 'très élevé' [6].
Par conséquent, la proposition floue *'Data est approximativement plus grand que 117,57'* peut être considérée comme équivalente à *'Data est élevé ou très élevé'*. Cette dernière a l'avantage d'être facilement comprise en estimation du coût (Figure 3).
Cependant, dans plusieurs cas la valeur *v* n'appartient pas à l'intervalle des valeurs permises pour un attribut particulier. Par exemple, dans la règle R1, la proposition *'Lexp n'est pas approximativement plus grand que 921,91'* utilise la valeur 921,91 ; cette valeur n'appartient pas à l'intervalle des valeurs possibles de l'attribut Lexp, Figure 4. En effet, l'attribut Lexp représente l'expérience des programmeurs et il est mesuré par le nombre de mois d'expérience. Dans le modèle COCOMO'81, la plus grande valeur possible est 36 mois. En utilisant notre fuzzification de l'attribut Lexp, la proposition *'Lexp n'est pas approximativement plus grand que 921,91'* serait équivalente à '*Lexp est Q(very low)'* où *Q* est un modificateur linguistique tel que 'plus que'.

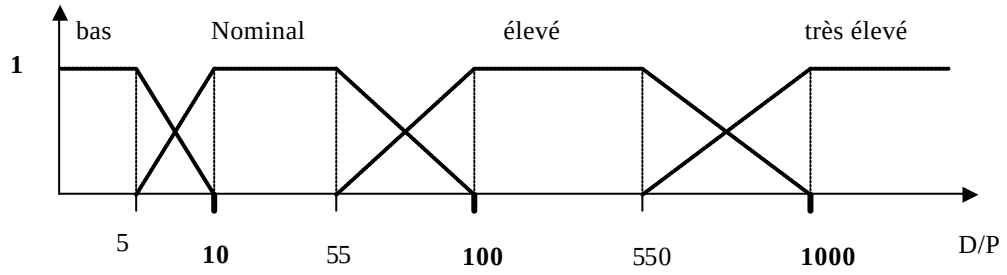

Figure 3: Fonctions d'appartenance aux ensembles flous associés au facteur DATA [6]

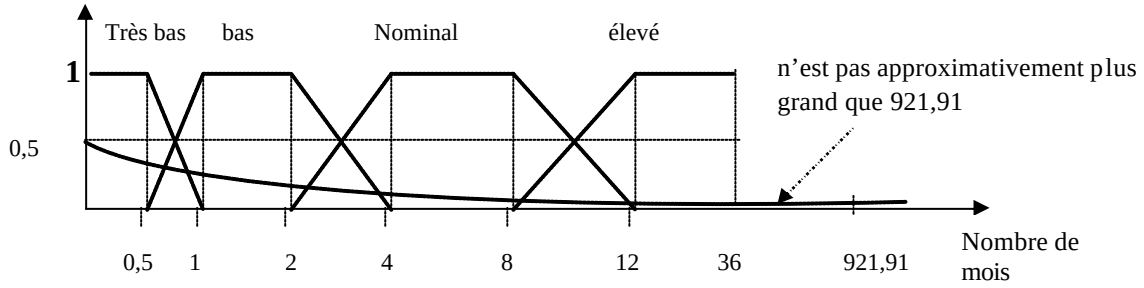

Figure 4: Exemple où la valeur utilisée pour le facteur Lexp n'est pas dans l'intervalle permis

Jusqu'à maintenant, nous avons proposé une interprétation des prémisses et des conséquences des règles floues. Il reste à expliquer la signification de l'opérateur 'i-or'. Selon Benitez et al., l'opérateur

'i-or' a une interprétation naturelle et peut être utilisé dans l'évaluation de plusieurs situations du monde réel telles que l'évaluation des articles scientifiques et l'évaluation de la qualité du jeu de deux joueurs de tennis. En analysant les 13 règles floues, il nous semble que l'opérateur 'i-or' n'est pas approprié pour évaluer l'effet des prémisses sur l'effort. En effet, l'influence d'une prémisse sur l'effort est évaluée en considérant toutes les propositions floues composant la prémisse. Chaque proposition floue est associée à un facteur du coût. En estimation du coût, l'effet d'un facteur sur l'effort dépend de son type et de son importance. En plus l'opérateur 'i-or' ne modélise pas convenablement les relations complexes existantes entre les facteurs du coût. Par exemple, supposons que la valeur de vérité d'une proposition est égale à 1 et que toutes les autres propositions de la prémisse ont des valeurs de vérité au voisinage de 0; la combinaison de ces propositions par 'i-or' donne une valeur de vérité égale à 1! Ceci est en contradiction avec notre intuition, spécifiquement, si la proposition qui a une valeur de vérité égale à 1 est associée au facteur le moins significatif pour l'effort.

6. Conclusion et perspectives

Dans cet article, nous avons étudié l'interprétation d'un réseau de neurones en estimation du coût de développement de logiciels. Pour ceci, nous avons développé un modèle basé sur un réseau de neurones de trois couches utilisant la fonction sigmoïde pour les neurones de la couche cachée et la fonction Identité pour ceux de la couche de sortie. Nous avons utilisé les projets de la base de données COCOMO'81 pour l'apprentissage et le test de notre modèle d'estimation. La précision des estimations obtenues est satisfaisante. Pour interpréter le réseau, nous avons utilisé la méthode de Benitez pour transformer notre réseau de neurones en un système à base de règles floues (si-alors). Les règles floues obtenues expriment la connaissance encodée dans le réseau par les poids synaptiques. L'interprétation de chaque règle floue est donnée par l'interprétation de sa prémisse et de sa conséquence. Notre étude a montré que nous pouvons fournir une interprétation compréhensible aux propositions floues composant la prémisse ainsi qu'à la conséquence de chaque règle floue. Cependant, la combinaison des différentes propositions floues par l'opérateur 'i-or' semble inapproprié en estimation du coût. En plus, l'opérateur 'i-or' ne peut être facilement expliqué dans plusieurs situations où il génère des sorties contradictoires à notre intuition. Par conséquent, nous envisageons d'utiliser d'autres méthodes d'extraction de règles floues à partir d'un réseau de neurones. Ces règles floues pourraient impliquer des opérateurs facilement interprétables tels que l'opérateur 'et' ou 'ou'.

Bibliographie

[1] J. M. Benitez, J.L. Castro, I. Requena, 'Are Artificial Neural Networks Black Boxes?', IEEE Transaction on Neural Networks, Vol. 8, NO. 5, September, 1997, pp. 1156-1164

[2] B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

[3] B.W. Boehm, and *al.*, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", *Annals of Software Engineering on Software Process and Product Measurement*, Amsterdam, 1995.

[4] D.S. Chulani, "Incorporating Bayesian Analysis to Improve the Accuracy of COCOMO II and Its Quality Model Extension", Ph.D. Qualifying Exam Report, USC, February, 1998.

[5] R.T Hughes, 'An Evaluation of Machine Learning Techniques for Software Effort Estimation', University of Brighton, 1996

[6] A. Idri, L. Kjiri, and A. Abran, "COCOMO Cost Model Using Fuzzy Logic", *7th International Conference on Fuzzy Theory & Technology*, Atlantic City, NJ, February, 2000. pp. 219-223

[7] A. Idri, and A. Abran, "Towards A Fuzzy Logic Based Measures For Software Project Similarity", *Sixth Maghrebian Conference on Computer Sciences, Fes*, Morroco, November, 2000. pp. 9-18

[8] A. Idri, and A. Abran, "A Fuzzy Logic Based Measures For Software Project Similarity: Validation and Possible Improvements", 7th International Symposium on Software Metrics, *IEEE computer society*, 4-6 April, England, 2001. pp. 85-96

[9] A. Idri, and A. Abran, "Evaluating Software Project Similarity by using Linguistic Quantifier Guided Aggregations", 9th IFSA World Congress/20th NAFIPS International Conference, 25-28 July, Vancouver, 2001. pp. 416-421

[10] A. Idri, A. Abran, T. M. Khoshgoftaar, "Fuzzy Analogy: A new Approach for Software Cost Estimation", 11th International Workshop in Software Measurements, 28-29 August, Montreal, 2001, pp. 93-101

[11] A. Idri, T. M. Khoshgoftaar, A. Abran, , "Estimating Software Project Effort by Analogy based on Linguistic values", To be presented in 8th IEEE International Software Metrics Symposium", 4-7 Ottawa, Canada, 2002

[12] J. S. Jang, C. T. Sun, 'Functional equivalence between radial basis function networks and fuzzy inference systems', IEEE Transaction on Neural Networks, Vol. 4, 1992, pp. 156-158
[13] M Jorgersen, 'Experience with Accuracy of Software Maintence Task Effort Prediction Models', IEEE Transaction on Software Engineering, Vol. 21(8), 1995, pp. 674-681
[14] A. Lapedes, Farber R., 'Nonlinear signal prediction using neural networks', Prediction and System modeling', Los Alamos National Laboratory, Tech. Report, LA-UR-87-2662, 1987
[15] R. P. Lippman, , 'An Introduction to computing with neural nets', IEEE ASSP Mag, vol. 4, pp.4-22, 1987
[16] B. Samson, Ellison D., Dugard P, 'Software Cost Estimation using an Albus Perceptron', 8th International COCOMO Estimation meeting, Pittsburgh, 1993
[17] C. Schofield, 'Non-Algorithmic Effort Estimation Techniques', Tech. Report TR98-01, March, 1998
[18] C. Serluca, 'An Investigation into Software Effort Estimation using a Back-propagation Neural Network, M.Sc. Thesis, Bournemouth University, 1995
[19] K. Srinivasan, Fisher D, 'Machine Learning Approaches to Estimating Softawre Development Effort', IEEE Transaction on Software Engineering, Vol. 21, No. 2, February, 1995, pp. 126-136
[20] A. R. Verkatachalam, 'Software Cost Estimation using Artificial Neural Networks, International Joint Conference on Neural Networks, Nogoya, IEEE, 1993
[21] G. Wittig, G. Finnie, 'Estimating Software Development Effort with connectionist Models', Information and Software Technologie, vol. 39, 1997, pp. 469-476
[22] L.A. Zadeh, "Fuzzy Set", Information and Control, Vol. 8, 1965, pp. 338-353