Validating and Understanding Software Cost Estimation Models based on Neural Networks

Ali Idri
Department of Software Engineering
ENSIAS, Mohamed V University
Rabat, Morocco
E-mail : idri@ensias.ma

Alain Abran
École de Technologie Supérieure
1180 Notre-Dame Ouest,
Montreal, Canada H3C 1K3
E-mail : aabran@ele.etsmtl.ca

Samir Mbarki
Department of Mathematics
Ibn Tofail University
Kenitra, Morocco
E-mail: Mbarki@univ-ibntofail.ac.ma

Abstract

Software development effort estimation with the aid of artificial neural networks (ANN) attracted considerable research interest at the beginning of the nineties. However, the lack of a natural interpretation of their estimation process has prevented them from being accepted as common practice in cost estimation. Indeed, they have generally been viewed with skepticism by a majority of the software cost estimation community. In this paper, we investigate the use and the interpretation of the Radial Basis Function Networks (RBFN) in the software cost estimation field. We first apply the RBFN to estimating the costs of software projects, and then study the interpretation of cost estimation models based on an RBFN using a method which maps this neural network to a fuzzy rule-based system, taking the view that, if the fuzzy rules obtained are easily interpreted, then the RBFN will also be easy to interpret. Our case study is based on the COCOMO'81 dataset.

1. Introduction

Estimating software development effort remains a complex problem, and one which continues to attract considerable research attention. Improving the accuracy of the estimation models available to project managers would facilitate more effective control of time and budgets during software development. In order to make accurate estimates and avoid gross misestimations, several cost estimation techniques have been developed. These techniques may be grouped into two major categories: (1) parametric models, which are derived from the statistical or numerical analysis of historical projects data [2,3], and (2) non-parametric models, which are based on a set of artificial intelligence techniques such as artificial neural networks, analogy-based reasoning, regression trees, genetic algorithms and rule-based induction [6,12,20,22,24,25].

Although experience has shown that there does not exist a 'best' prediction technique outperforming all the others in every situation [4,12,16,21], non-parametric models still have two significant advantages over parametric ones: first, the ability to model the complex set of relationships between the dependent variable required to predict (cost, effort) and the independent variables (cost drivers) collected earlier in the lifecycle; second, the ability to learn from historical projects data, especially for artificial neural networks. In this paper, we are concerned with cost estimation models based on artificial neural networks.

An artificial neural network is characterized by its architecture, its learning algorithm and its activation functions [7,8,17]. In general, for software cost estimation modeling, the most commonly adopted architecture, the learning algorithm and the activation function are the feedforward multi-layer Perceptron, the Backpropagation algorithm and the Sigmoid function respectively [22,23,25]. Most of these referenced studies have focused more on the accuracy of the approach relative to that of other cost estimation techniques.

In this work, we are concerned with Radial Basis Function Network (RBFN) models, which differ from the Backpropagation feedforwad Perceptron in the following ways: 1) usually, an RBFN architecture is composed of three layers, as opposed to a Perceptron which may have more; 2) RBFN models use a hybrid learning scheme that combines supervised and unsupervised learning algorithms, whereas the Backpropagation is a supervised learning algorithm; and 3) the activation function of the middle-layer neurons of an RBFN is usually the Gaussian function, whereas that of the hidden Perceptron neurons can be Sigmoid, Gaussian or other types of functions. However, even though there is variety among neural network architectures, and most of them have shown their strengths in solving complex problems, many researchers in various fields are hesitant to use them because of their common shortcoming of being 'black boxes' models.

To overcome this limitation, we have studied the use of FRBSs to provide a natural interpretation of cost estimation models based on a Backpropagation three-layer feedforward Perceptron [11]. What we have proposed comprised essentially the use of the Benitez's method to extract the if-then fuzzy rules from this network [1]. These fuzzy rules express the information encoded in the architecture of the network, and the interpretation of each fuzzy rule has been determined by analyzing its premise and its output. While our case study has shown that we can explain the meaning of the output and the propositions composing the premise of

each fuzzy rule, the entire fuzzy rule cannot be easily interpreted because it uses the 'i-or' operator. In this paper, we explore another mapping method, that is, the Jang and Sun method, to extract if-then fuzzy rules from artificial neural networks. The use of this method requires that the architecture of the network be an RBFN [14]. The main advantage of Jang and Sun method is that it uses a more understandable operator, that is, the 'and' logical conjunction, to combine the propositions composing the premise of each rule. This case study is based on the COCOMO'81 historical dataset.

This paper is organized as follows: In Section 2, we briefly describe the RBFN architecture most often used. Section 3 shows how an RBFN can be used to estimate software development effort. We also present the architecture of the RBFN that will be used in our case study. In Section 4, we discuss the results obtained when the RBFN is used to estimate software development effort. In Section 5, we briefly outline the principle of the Jang and Sun method that will be used to extract the if-then fuzzy rules from our network. In Section 6, we apply the Jang and Sun method to our RBFN and we discuss the interpretation of the obtained fuzzy rules in software cost estimation. A conclusion and an overview of future work conclude this paper.

2. Radial basis function networks for software cost estimation

Based on biological receptive fields, Moody and Darken proposed a network architecture called the Radial Basis Function Network (RBFN) which employs local receptive fields to perform function mappings [15]. It can be used for a wide range of applications, primarily because it can approximate any regular function [19]. An RBFN is a three-layer feedforward network consisting of one input layer, one middle-layer and an output layer. Figure 1 illustrates a possible RBFN architecture configured for software development effort. The RBFN generates output (effort) by propagating the initial inputs (cost drivers) through the middle-layer to the final output layer. Each input neuron corresponds to a component of an input vector. The middle layer contains M neurons, plus, eventually, one bias neuron. Each input neuron is fully connected to the middle-layer neurons. The activation function of each middle neuron is usually the Gaussian function:

$$f(x) = e^{\left(-\frac{\|x-c_i\|^2}{\sigma_i^2}\right)} \qquad \text{(Eq. 1)}$$

where c_i and σ_i are the center and the width of the i^{th} middle neuron respectively. $\|.\|$ denotes the Euclidean distance. Hence, each i^{th} hidden neuron has its own *receptive field* in the input space, a region centered on c_i with size proportional to σ_i. The Gaussian function decreases rapidly if the width σ_i is small, and slowly if it is large. The output layer consists of one output neuron that computes the software development effort as a linear weighted sum of the outputs of the middle layer:

$$Effort = \sum_{j=1}^{M} y_j \beta_j \qquad with \quad y_j = e^{\left(-\frac{\|x-c_j\|^2}{\sigma_j^2}\right)} \qquad \text{(Eq. 1)}$$

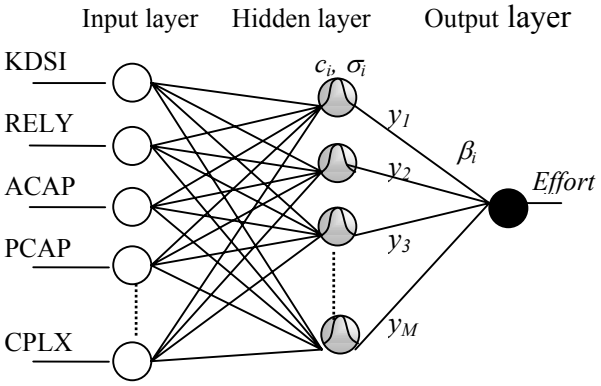

Figure 1: A Radial Basis Function Network architecture for software development effort

3. Experiment design

This section describes the experiment design of an RBFN on the COCOMO'81 historical projects. We present and discuss how we set the different parameters of the RBFN architecture according to the characteristics of the COCOMO'81 dataset, especially the number of input neurons, number of hidden neurons, centers c_i, widths σ_i and weights β_j.

Before an RBFN is ready to make estimates for new software projects, it is trained by a set of combinations of inputs (cost drivers) and outputs (development efforts) of historical software projects. Our case study consists of estimating the software development effort by using an RBFN on the COCOMO'81 dataset. This dataset contains 63 software projects [2]. Each project is described by 13 attributes: the software size measured in KDSI (Kilo Delivered Source Instructions) and the remaining 12 attributes are measured on a scale composed of six linguistic values: 'very low', 'low', 'nominal', 'high', 'very high' and 'extra high'.

As mentioned earlier, the use of an RBFN to estimate software development effort requires the determination of the middle-layer parameters and the weights β_j. The simplest and most general method for deciding on the middle-layer neurons is to create a neuron for each training software project. Consequently, the middle layer of our RBFN will have 63 neurons. This approach does not take into account the existing similarities among the 63 historical software projects. As we have shown in various studies with the COCOMO'81 dataset, there are some similarities among certain software projects of the COCOMO'81 dataset [10,12,13]. As a consequence, we prefer to first cluster the 63 projects into a reasonable number of groups containing similar software projects. We most often use the APC-III clustering algorithm developed by Hwang and Bang [9] to do this. APC-III is

a one-pass clustering algorithm, and has a constant radius R_0 defined as follows:

$$R_0 = \alpha \frac{1}{N} \sum_{i=1}^{N} \min_{i \neq j} (\|P_i - P_j\|) \qquad \text{(Eq. 3)}$$

where N is the number of historical software projects and α is a predetermined constant. R_0 expresses the radius of each cluster and consequently it controls the number of the clusters provided. APC-III generates many clusters if R_0 is small and few clusters if it is large. According to Hwang and Bang, the APC-III is quite efficient at constructing the middle layer of an RBFN, since it can finish clustering by going through all the training patterns only once; this is not true with K-means and SOFM, which are multi-pass and time-consuming clustering algorithms.

Concerning the weights β_j, we may set each β_j to the associated effort of the center of the j^{th} neuron. However, this technique is not optimal and does not take into account the overlapping that may exist between receptive fields of the hidden layer. Thus, we use the Delta rule to derive the values of β_j.

Finally, our RBFN has 13 inputs (COCOMO cost drivers) and one output (development effort). All the inputs as well as the output of the network are numeric. The RBFN is trained by iterating through the training data many times. The Delta rule uses a learning rate, and maximum error equals 0.03 and 10^{-5} respectively.

4. Overview of the empirical results

The following section presents and discusses the results obtained when applying the RBFN to the COCOMO'81 dataset. We have conducted several experiments with the APC-III algorithm to decide on the number of hidden units. These experiments use the full COCOMO'81 dataset for training. Table 1 shows the classifications obtained according to different values of α (Eq. 3).

Table 1: Number of clusters according to α for the COCOMO'81 dataset

α	Number of clusters
0.4|0.45|0.46|0.47|0.49	62|59|60|58|55
0.5|0.52|0.6|0.62	54|52|50|48
0.66|0.70|0.75	47|45|43
0.78|0.80	41|37

In analyzing the results of Table 1, we noticed that the number of clusters is monotonous decreasing according to α. This is due to the fact that the radius R_0 is monotonous increasing according to α. Choosing the 'best' classification of those in Table 1 to determine the number of hidden neurons and their centers is a complex task. For software cost estimation, we suggest that the 'best' classification is the one that satisfies the following two criteria:

- it improves the accuracy of the RBFN;
- it provides coherent clusters, i.e. the software projects of a given cluster have satisfactory degrees of similarity.

Thus, we have conducted several experiments with an RBFN, each time using one of the classifications of the table 1. These experiments use the full COCOMO'81 dataset for training and testing. The weights β_j are calculated using the Delta learning rule. The RBFN converges quickly, with fewer than 12000 iterations of learning. The accuracy of the estimates generated by the RBFN is evaluated by means of the Mean Magnitude of Relative Error 'MMRE' and the Prediction level 'Pred', defined as follows:

$$MMRE = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{Effort_{actual,i} - Effort_{estimated,i}}{Effort_{actual,i}} \right| \times 100 \; ;$$

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

where N is the total number of observations, k is the number of observations with an MRE less than or equal to p. A common value for p is 0.25. The MMRE and Pred are calculated for each classification of the table 1. Figure 2 shows the MMRE and Pred as functions of the classification (α). We can notice that the accuracy of the RBFN is better when α is lower than 0.49 (MMRE≤30 and Pred(25) ≥70). When α is higher than 0.49, the MMRE is high, although the values of Pred(25) are acceptable. Indeed, the MMRE is very sensitive to variations in the estimated values, particularly when α increases, in which case the classification obtained may be less coherent, i.e. some clusters are composed of software projects that are not sufficiently similar. For those projects, the RBFN may generate inaccurate estimates.

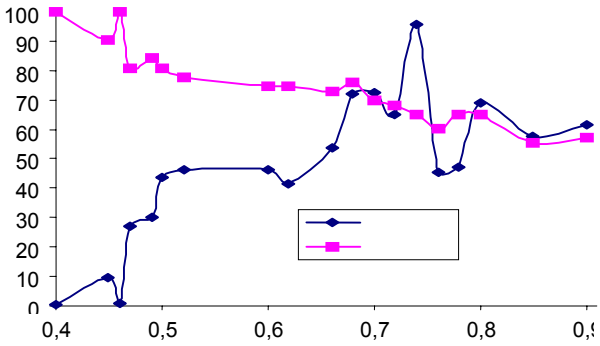

Figure 2 : Relationship between the accuracy of the RBFN (MMRE and Pred) and the used classification (α).

In addition to the number of hidden neurons (number of clusters generated by the APC-III algorithm), the accuracy of the RBFN depends on the widths σ_i used by the hidden neurons (receptive fields). In the previous experiments (Figure 2), we had set the values of σ_i to 30. However, in the literature, the widths σ_i were usually determined to cover the input space as uniformly as

possible [8,9]. Covering the historical software project space uniformly implies that the RBFN will be able to generate an estimate for a new project even though it is not similar to any historical project. In such a situation, we prefer that the RBFN does not provide any estimate than one it may easily lead to wrong managerial decisions and project failure.

As a consequence, we have adopted a simple strategy based primarily on two guidelines. First, we assign one value to all widths σ_i. Second, this value depends on the radius R_0. Indeed, R_0 is used in the APC-III algorithm to delimit the area of each cluster. This looks like a reasonable way to set the widths σ_i to a value closer to R_0.

As we noticed earlier, in addition to the accuracy of the RBFN, the aim is to explain its process by mapping the RBFN to a fuzzy rule-based system. The above empirical experiments allow us to choose which RBFN architecture that will be used in the mapping process. The method developed by Jang and Sun will be used to extract the if-then fuzzy rules from the chosen RBFN architecture[14]. This method is presented below.

5. Equivalence between Radial Basis Function networks and fuzzy rule-based systems: Background

Since its foundation by Zadeh in 1965 [26], fuzzy logic has been the subject of many investigations. One of its main contributions to solving complex problems is undoubtedly the Fuzzy Rule-Based Systems [27]. Essentially, an FRBS is based on a set of if-then fuzzy rules. A fuzzy rule is an if-then statement where the premise and the consequence consist of fuzzy propositions, whereas in a classical production rule the premise and the consequence are crisp. An example of a fuzzy rule in cost estimation may be 'if the competence of the analysts is high then the effort is low'. The main advantage of fuzzy rules over classical rules is that they are easier for humans to understand and may be easily interpreted. This is because fuzzy rules, unlike classical rules, use linguistic values instead of numerical data in their premises and consequences. As a result, some researchers have investigated the equivalence between neural networks and FRBSs [1,5,14,18], their objective being to translate the knowledge embedded in the neural network into a more understandable language, that is, the if-then fuzzy rules. Furthermore, establishing equivalence between neural networks and FBRSs can be used to apply advances and new developments of one model to the other and vice versa.

Jang and Sun have developed a method which proves the functional equivalence between an RBFN, such as the one used in the previous sections, and a fuzzy rule-based system which uses Takagi-Sugeno rules [14]. Hence, we use their method to provide a natural interpretation of cost estimation models based on the RBFN. Jang and Sun fixed five conditions to the establishment of a functional equivalence between an RBFN and a FBRS using Takagi-Sugeno rules.

Under these conditions, the RBFN is equivalent to an FBRS which uses a set of fuzzy rules R_j associated with all hidden neurons:

$$R_j: \quad if\ x_1\ is\ A_j^1\ and\ x_2\ is\ A_j^2\ and..and\ x_n is\ A_j^n\ then\ y_j = \beta_j$$

where x_i are the inputs of the RBFN, β_j are the weights of the hidden units to the output unit, and A_j^i is a fuzzy set with a Gaussian membership function defined by:

$$\mu_{A_j^i}(x) = e^{-\frac{(x - c(A_j^i))}{2\sigma^2}} \qquad \text{(Eq. 4)}$$

where $c(A_j^i)$ is the i[th] component of the j[th] center of the classification used by the RBFN.

6. Validation and interpretation of the fuzzy rules

In this section, we apply the method developed by Jang and Sun on the RBFN presented and discussed in Sections 2, 3 and 4. The RBFN that we consider adopts the classification associated to α=0.45 because of the accuracy of its estimates (Figure 2). Hence, the RBFN has 59 hidden units and therefore the equivalent FBRS has 59 fuzzy if-then rules. Each fuzzy rule contains 13 fuzzy propositions in its premise and one numerical output. These 59 fuzzy rules express the knowledge encoded into the synaptic weights of our RBFN. The objective is to give a more comprehensible interpretation of these fuzzy rules in software cost estimation. Each fuzzy rule will interpreted by interpreting its premise, its output and its implication. For simplicity, we discuss only three fuzzy rules (Table 2).

The premise of each j[th] fuzzy rule is composed of 13 fuzzy propositions combined by the 'and' operator. Each is associated with one cost driver (inputs of the RBFN). For instance, the first proposition of the rule R_1 *'DATA is approximately equal to 7.74'* concerns the cost driver DATA, which represents, with the other three cost drivers, the effect of the size and complexity of the database on the software development effort. Each i[th] proposition is expressed as *'x_i is A_j^i'* where A_j^i is a fuzzy set with the membership function of Equation 4. It can be understood as *'x_i is approximately equal to $c(A_j^i)$'*, where $c(A_j^i)$ is the i[th] element of the j[th] center. The linguistic qualification *'approximately equal to'* is represented here by a fuzzy set with a Gaussian membership function of Equation 4.

The output of each fuzzy rule is a numerical value. By analyzing the rules of Table 2, we notice that the output of rule R_1 is positive (72.44), whereas that of rule R_{15} is negative (-30.11). These two values are the synapse weights of the hidden units representing the clusters C_1 and C_{15} to the output unit. Consequently, the natural

interpretation that we can give to the outputs of the 59 fuzzy rules is that they can be considered as partial contributions to the total effort. They have the same role as the effort multipliers in the COCOMO'81 model. They can increase (positive value) or decrease (negative value) the total development effort. The only difference between them is that the outputs of the fuzzy rules are positive or negative. This is because the cost function of the FRBS uses the summation operator, whereas in the COCOMO'81 model, due to its cost function that uses the multiplication operator, the effort multipliers are greater than 1 (increase the effort) or less than 1 (decrease the effort).

Up to now, we have been suggesting a natural interpretation of the premises and outputs of the obtained fuzzy rules. It still remains to consider the meaning of each fuzzy rule taken as a whole, in other words, the interpretation of the implication contained in each fuzzy rule. According to our understanding of the premises and the outputs, each fuzzy rule R_j can be written as follows:

R_j : **if** x_1(P) is approximately equal to $c(A_j^1)$ and
x_2(P)is approximately equal to $c(A_j^2)$ and
…
x_M(P)is approximately equal to $c(A_j^M)$
then effort (P)= β_j

where x_i(P) are the values of the cost drivers of the new project P and $c(A_j^i)$ are those of project C_j, that is, the center of the j^{th} cluster. If we reduce the width σ of the hidden neurons, the values β_j, derived from the Delta learning rule, converge to the real efforts of C_js, where C_js are the centers of clusters. Indeed, in this case, receptive fields (hidden neurons) do not overlap and consequently each new project is classified into only one cluster. Thus, each fuzzy rule can be interpreted as follows:

P is similar to C_j implies that effort(P) is similar to effort(C_j)

This affirmation is the basis of cost estimation models based on reasoning by analogy [12,20]. The main advantage of using reasoning by analogy to estimate software development effort is that we can understand its process and easily explain it to users. This is because humans are familiar with this kind of reasoning, as they use it every day of their lives

7. Conclusion and Future work

In this paper, we have studied one of the most important limitations of neural networks, which is the difficulty of understanding why a neural network makes a particular decision. Our study is intended for application to the cost estimation field, and the neural network that we have used to predict the software development effort is the Radial Basis Function network. We used the entire COCOMO'81 dataset to train and test the RBFN. We have found that the accuracy of the RBFN depends essentially on the parameters of the middle layer, especially the number of hidden neurons and the values of the widths σ_i. The number of hidden neurons is determined by one of the existing clustering algorithms; in particular, we have adopted the APC-III algorithm because it is a one-pass clustering technique and therefore it makes the estimation process faster than if we had used an elaborate but time-consuming clustering algorithm. In addition to the number of hidden units, the accuracy of the RBFN depends on the widths σ_i. We have shown that choosing one value closer to the radius R_0 for all the σ_i improves the accuracy of the RBFN.

.

Table 2 : Three examples of the obtained fuzzy rules

If :	R_1		R_{14}		R_{15}	
DATA	is approximately equal to	7.74 and	is approximately equal to	7.786 and	is approximately equal to	8.472 and
VIRTmi	is approximately equal to	6.471 and	is approximately equal to	11.481 and	is approximately equal to	0.410 and
TIME	is approximately equal to	79.999 and	is approximately equal to	64.407 and	is approximately equal to	56.506 and
STORE	is approximately equal to	94.14 and	is approximately equal to	71.238 and	is approximately equal to	82.974 and
VIRTma	is approximately equal to	36.053 and	is approximately equal to	106.918 and	is approximately equal to	4.299 and
TURN	is approximately equal to	3.50and	is approximately equal to	1.049 and	is approximately equal to	2.536 and
ACAP	is approximately equal to	70.50 and	is approximately equal to	55.564 and	is approximately equal to	78.149 and
AEXP	is approximately equal to	30.632 and	is approximately equal to	16.414 and	is approximately equal to	33.263 and
PCAP	is approximately equal to	82.105 and	is approximately equal to	59.636 and	is approximately equal to	82.304 and
VEXP	is approximately equal to	3.213 and	is approximately equal to	4.846 and	is approximately equal to	3.952 and
LEXP	is approximately equal to	1.93 and	is approximately equal to	6.844 and	is approximately equal to	3.547 and
SCED	is approximately equal to	12.338 and	is approximately equal to	90.231 and	is approximately equal to	84.720 and
KDSI	is approximately equal to	3.0 and	is approximately equal to	15.275 and	is approximately equal to	5.3 and
Then	Y=72.44		Y=203.37		Y=-30.11	

After evaluating the accuracy of the RBFN, we applied the Jang and Sun method to extract the if-then fuzzy rules from the most accurate RBFN architecture. These fuzzy rules express the information encoded in the architecture of the network. The interpretation of each fuzzy rule is determined by analyzing its premise, its output and its implication. Our case study shows that we can explain the meaning of the output and the propositions composing the premise of each fuzzy rule. Indeed, each proposition of a premise can be expressed

as a statement like *'x is approximately equal to v'* where the linguistic value *'approximately equal to'* is represented by a fuzzy set with a Gaussian membership function. After providing a natural interpretation of the premise and output of each fuzzy rule, we have suggested an explanation of the entire fuzzy rule. Each fuzzy rule can be interpreted as an analogy affirmation *'similar software projects have similar costs',* which is easy for users to understand.

8. Bibliography

[1] J. M. Benitez, J.L. Castro and I. Requena. "Are Artificial Neural Networks Black Boxes?" IEEE Transactions 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 et 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] L. Briand, T. Langley and I. Wieczorek. "Using the European Space Agency data set: A replicated assessment and comparison of common software cost modeling." In Proc 22th IEEE International Conference on Software Engineering, Limerick, Ireland, 2000, pp. 377-386.

[5] J. J. Buckley, Y. Hayashi and E. Czogala. "On the Equivalence of Neural Nets and Fuzzy Expert Systems." Fuzzy Sets and Systems, no. 53, 1993, pp. 129-134.

[6] C. J. Burgess and M. Lefley. "Can Genetic Programming Improve Software Effort Estimation?" Information and Software Technology, vol. 43, 2001, pp. 863-873.

[7] S. Hayken. Neural Networks: A Guide to Intelligent Systems." Addison-Wesley, 1994.

[8] J. Hertz, A. Krogh and R. G. Palmer. Introduction to the Theory of Neural Computation. Addisson-Wesley, 1991.

[9] Y-S Hwang and S-Y Bang. "An Efficient Method to Construct a Radial Basis Function Network Classifier." Neural Networks, vol. 10, no. 8, 1997, pp. 1495-1503.

[10] A. Idri, and A. Abran, "A Fuzzy Logic Based Measure 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

[11] A. Idri, T. M. Khoshgoftaar and A. Abran. "Can Neural Networks be easily Interpreted in Software Cost Estimation." FUZZ-IEEE, Hawaii, 2002, pp. 1162-1166.

[12] A. Idri., A. Abran and T. M. Khoshgoftaar. "Estimating Software Project Effort by Analogy based on Linguistic values." 8th IEEE International Software Metrics Symposium," 4-7 April, Ottawa, Canada, 2002, pp. 21-30.

[13] A. Idri, T. M. Khoshgoftaar and A. Abran. "Investigating Soft Computing in Case-based Reasoning for Software Cost Estimation." International Journal of Engineering Intelligent Systems, vol. 10, no. 3, 2002, pp. 147-157.

[14] J. S. Jang and C. T. Sun. "Functional equivalence between radial basis function networks and fuzzy inference systems." IEEE Trans. on Neural Networks, vol. 4, 1992, pp. 156-158

[15] J. Moody and C. Darken. Fast Learning in Networks of Locally-Tuned Processing Units. Neural Computation, vol. 1, 1989, pp. 281-294.

[16] I. Myrtveit and E. Stensrud, "A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models." IEEE Transactions on Software Engineering, 25, 4, July/August, 1999, pp. 510-525

[17] M. Negnevitsky. Artificial Intelligence: A Guide to Intelligent Systems. Addison-Wesley, 2001.

[18] H-X. Li and C. L. P. Chen. "The Equivalence between Fuzzy Logic Systems and Feedforward Neural Networks." IEEE Tran. on Neural Networks, vol. 11, no. 2, 2000, pp. 356-365.

[19] J. Park and I. W. Sandberg. "Approximation and Radial Basis Function Networks." Neural Computation, vol. 5, 1993, pp. 305-316.

[20] M. Shepperd and C. Schofield. "Estimating Software Project Effort Using Analogies." Transactions on Software Engineering, vol. 23, no. 12, 1997, pp. 736-747.

[21] S. Shepperd and G. Kadoda, "Using simulation to evaluate prediction systems." 7th International Symposium on Software Metrics, IEEE Computer Society, 4-6 April, UK, 2001, pp. 349-358

[22] K. Srinivasan, and D. Fisher "Machine Learning Approaches to Estimating Software Development Effort." IEEE Transactions on Software Engineering, vol. 21, no. 2. February, 1995, pp. 126-136.

[23] A. R. Verkatachalam. "Software Cost Estimation using Artificial Neural Networks." International Joint Conference on Neural Networks, Nogoya, IEEE 1993.

[24] S. Vicinanza, and M.J. Prietolla, "Case-Based Reasoning in Software Effort Estimation." Proceedings of the 11th Int. Conf. on Information Systems, 1990.

[25] G. Wittig and G. Finnie. "Estimating Software Development Effort with Connectionist Models." Information and Software Technology, vol. 39, 1997, pp. 469-476.

[26] L.A. Zadeh. "Fuzzy Set." Information and Control, vol. 8, 1965, pp. 338-353.

[27] H. J. Zimmermann. Fuzzy Set Theory and its Applications. 2nd ed, Boston, MA, Kluwe, 1991.