An Experiment on the Design of Radial Basis Function Neural Networks for Software Cost Estimation

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

This paper is concerned with the use of Radial Basis Function (RBF) neural networks for software cost estimation. The study is devoted to the design of these networks, especially their middle layer composed of receptive fields, using two clustering techniques: the C-means and the APC-III algorithms. A comparison between an RBFN using C-means and an RBFN using APC-III, in terms of estimates accuracy, is hence presented. This 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 [4,9,18,19,21,22]. In this paper, we are concerned with cost estimation models based on artificial neural networks, especially Radial Basis Function Neural Networks.

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 [16]. 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:

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

The use of an RBFN to estimate software development effort requires the determination of the middle-layer parameters (receptive fields) and the weights β_j. The choice of the receptive fields, especially their distribution in the input space is often critical to the successful performance of an RBF network [17]. In general, there are two primary sources of this knowledge:

- The use of some clustering techniques to analyze and find clusters in the training data. The results of this grouping are used to establish prototypes of the receptive fields.
- The use of existing empirical domain knowledge to form the set of receptive fields. Along this line emerge some fuzzy models; in particular, some interesting equivalence between RBF networks and Fuzzy Rule-Based systems [12].

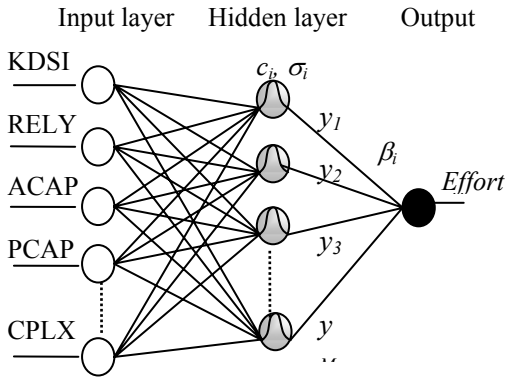

Figure 1: An example of Radial Basis Function Network architecture for software development effort

The aim of this study is to discuss the preprocessing phase of the design of RBF neural network in software cost estimation as it becomes accomplished through data clustering techniques. Especially, we use two clustering techniques: 1) the APC-III algorithm developed by Hwang and Bang [7], and 2) the best-known clustering method that is the C-means algorithm.

This paper is composed of six sections. In Section 2, we briefly describe how we can use the two clustering algorithms: APC-III and C-means in the design of an RBF neural network. Section 3 presents the experiment design of an RBFN construction based on APC-III or C-means in software cost estimation. In Section 4, we discuss the results obtained, in terms of estimates accuracy, when the RBFN is used to estimate the software development effort. A conclusion and an overview of future work conclude this paper.

2. Clustering techniques for RBF networks

Generally speaking, clustering is one method to find most similar groups from given data, which means that data belonging to one cluster are the most similar; and data belonging to different clusters are the most dissimilar. In the literature, researchers have proposed many solutions for this issue based on different theories, and many surveys focused on special types of clustering algorithm have been presented [1,5,6,13,14].

Clustering techniques have been successfully used in many application domains, including biology, medicine, economics, and patterns recognition. These techniques can be grouped into major categorises: Hierarchical or Partitional [6]. In this paper, we focus only on the partitional clustering algorithm since it is used more frequently than other clustering algorithms in pattern recognition fields. Generally, partitional clustering algorithms suppose that the data set can be well represented by finite prototypes. Partitional clustering is also called objective function-based clustering algorithm.

Clustering has been often exercised as a preprocessing phase used in the design of the RBF neural networks. The primary aim of this algorithm is to set up an initial distribution of the receptive fields (hidden neurons) across the space of the input variables. In particular, this implies a location of the modal values of these fields (e.g., the modes of the Gaussian functions).

In an earlier work, we have used the APC-III clustering algorithm to determine the receptive fields of an RBF network for software cost estimation [11]. 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. The outline of APC-III algorithm can be stated as follows:

1- Initially, the number of clusters is set at 1; the center of this cluster C_1 is the first software project in the dataset, say P_1

2- Iterate from i=2 to N (N is the number of historical software projects)

a. For j=1 to c (c is the number of clusters)
- Compute d_{ij} (d_{ij} is the Euclidean distance of P_i and c_j, c_j is the center of cluster C_j)
- If $d_{ij}<R_0$ then
  - Include P_i into C_j and adjust the center of C_j
  - Exit from the loop

b. If P_i is not included in any clusters then
- Create a new cluster that has P_i as a center

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. However, the APC-III proves to be dependent upon order of data presentation.

This paper suggests the use of the best-known clustering method that is the C-means algorithm to determine the receptive fields of an RBF network for software cost estimation. C-means clustering algorithm has many successfully applications in fields such as patter recognition and data compression. It is a multi-pass and time-consuming clustering algorithm.

The C-means algorithm partitions a collection of N vectors into c clusters C_i, i=1,..,c. The aim is to find cluster centers (centroids) by minimizing a dissimilarity (or distance) function which is given in Equation 4.

$$J = \sum_{i=1}^{c} \sum_{x_k \in C_i} d(x_k, c_i) \qquad \text{(Eq. 4)}$$

c_i is the center of cluster C_i;
$d(x_k,c_i)$ is the distance between i^{th} center (c_i) and k^{th} data point;

For simplicity, the Euclidian distance is used as dissimilarity measure and overall dissimilarity function is expressed as in Equation 5.

$$J = \sum_{i=1}^{c} \sum_{x_k \in C_i} \left\| x_k - c_i \right\|^2 \qquad \text{(Eq. 5)}$$

The outline of C-means algorithm can be stated as follows:

1- Define the number of the desired clusters, c.
2- Initialize the centers c_i,i=1,..c. This is typically achieved by randomly selecting c points from among all of the data points.
3- Compute the Euclidean distance between x_j and c_i, j=1..N and i=1..c
4- Assign each x_j to the most closer cluster Ci
5- Recalculate the centers c_i
6- Compute the objective function J given in equation 5. Stop if its improvement over previous iteration is below a threshold.
7- Iterate from step 3

The performance of the algorithm depends on the initial positions of centers. So the algorithm gives no guarantee for an optimum solution.

3. Experiment Design and Data Description

This section describes the experiment design of an RBFN on an artificial COCOMO'81 dataset. The original COCOMO'81 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'. These 12 attributes are related to the software development environment such as the experience of the personnel involved in the software project, the method used in the development and the time and storage constraints imposed on the software. Because the original COCOMO'81 dataset contains only 63 historical software projects and in order to have a robust empirical study, we have artificially generated, from the original COCOMO'81 dataset, three other datasets each one contains 63 software projects (see [9] for more details). The union of the four datasets constitutes the artificial COCOMO'81 dataset that is used in this study.

As mentioned earlier, the use of an RBFN to estimate software development effort requires the determination of its architecture parameters 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.

The number of input neurons is the same as the number of attributes (cost drivers) describing the historical software projects in the COCOMO'81 dataset. Hence, the number of input neurons is equal to 13. The number of hidden neurons is determined by the number of clusters (c) provided by the APC-III or the C-means algorithms described in Section 2. Concerning the widths σ_i, they were usually determined in the literature to cover the input space as uniformly as possible [7]. 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. In our earlier work [11], we have adopted a simple strategy based primarily on assigning one value to all σ_i; this value depends on the radius R_0 used in the APC-III algorithm. Because here we investigate two clustering techniques, we adopt one of the following formulas to determine σ_i:

$$\sigma_i = \begin{cases} \max\limits_{x_j \in C_i} d(x_j, c_i) & if\ card(C_i) > 1 \\ \max\limits_{k/card(C_k)>1} \sigma_k & if\ card(C_i) = 1 \end{cases} \qquad \text{(Eq. 6)}$$

$$\sigma_i = \begin{cases} \max\limits_{x_j \in C_i} d(x_j, c_i) & if\ card(C_i) > 1 \\ \min\limits_{k/card(C_k)>1} \sigma_k & if\ card(C_i) = 1 \end{cases} \qquad \text{(Eq. 7)}$$

Where $card(C_i)$ is the cardinality of cluster C_i

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.

4. Overview of the empirical results

The following section presents and discusses the results obtained when applying the RBFN to the artificial COCOMO'81 dataset. The calculations were made using two software prototypes developed with C language under a Microsoft Windows PC environment. The first software prototype implements the APC-III and the C-means clustering algorithms, providing both the clusters and their centers from the COCOMO'81. The second software prototype implements a cost estimation model

based on an RBFN architecture in which the middle-layer parameters are determined by means of the first software prototype

We have conducted several experiments with both the C-means and the APC-III algorithms to decide on the number of hidden units. These experiments use the full artificial COCOMO'81 dataset for training. Choosing the 'best' classification 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 provides coherent clusters, i.e. the software projects of a given cluster have satisfactory degrees of similarity;
- it improves the accuracy of the RBFN.

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 25.

4. 1 RBFN with C-means algorithm

To measure the coherence of clusters in the case of C-means algorithm, we use one of the two criteria: the objective function J given in Equation 5 or the Dunn's validity index defined by the following formula:

$$D_1 = \min_{1\le i\le c}\left(\min_{i+1\le j\le c-1}\left(\frac{d(C_i,C_j)}{\max_{1\le k\le c}(d(C_k))}\right)\right) \qquad \text{(Eq. 8)}$$

where $d(C_i,C_j)$ *is* the distance between clusters C_i, and C_j (intercluster distance);

$$d(C_i,C_j) = \min_{x_i\in C_i, x_j\in C_j} d(x_i,x_j)$$

and $d(C_k)$ is the intracluster distance of cluster C_k

$$d(C_k) = \max_{x_i,x_j\in C_k} d(x_j,x_k)$$

The Dunn's index (D_1) expresses the idea of identifying clusters that are compact and well separated. The main goal of the measure (D_1) is to maximise the intercluster distances and minimise the intracluster distances. Therefore, the number of cluster that maximise D_1 is taken as the optimal number of the clusters.

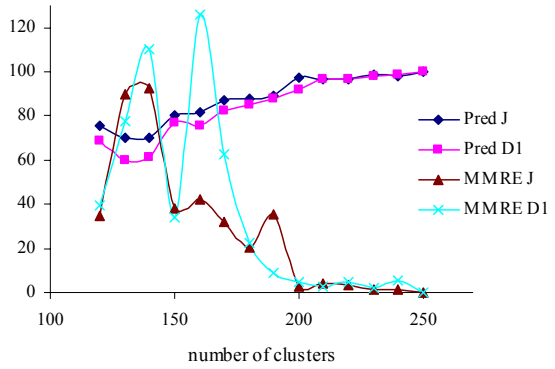

Figure 2: Relationship between the accuracy of RBFN (MMRE and Pred) and the used classification of the C-means (J and D1).

We have conducted several experiments with an RBFN, each time using one of the classifications generated by the C-means algorithm. These experiments use the full COCOMO'81 dataset for training and testing. For each number of clusters (c), the RBFN uses the two C-means classifications that respectively minimise J or maximise D1. For instance, when fixing c to 120, the two choosing classifications are respectivelly those for which J is equal to 1064,22 and D_1 is equal to 8,11. Figure 2 shows the relationship between the accuracy of the RBFN, measured in terms of Pred and MMRE, and the used classifications (number of clusters) minimizing the objective function J or maximizing the D_1 index. We can notice that the accuracy of the RBFN when using the C-means classification that minimizes J (Pred_J and MMRE_J) is better than that when using the classification maximizing D_1 (Pred_D_1 and MMRE_D_1). In figure 2, we only show the results of experiments when the number of clusters is higher than 120 because the evaluated accuracy of the RBFN is acceptable (the common values used in the literature are Pred(25)>= 70 and MMRE<=30). Also, the obtained classifications for c lower than 120 are, in general, 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.

4.2 RBFN with the APC-III algorithm

The classifications generated by the APC-III algorithm depend on the number α that defines the radius R_0. Table 1 shows the classifications obtained according to different values of α (Eq. 3).

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. For each value of α, we varied the presentation sequence of the 252 software projects. Indeed, the classification provided by the APC-III

depends on this presentation sequence because it influences the determination of the centers; we retained for each the classification maximizing D_1 index.

α	Number of clusters
0.4|0.5|0.6|0.7|0.8	251|244|234|216/200
0.9|1.0|1.02|1.04	189|170|162|161
1.06|1.08|1.1	155|151|150

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

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 MMRE and the Pred indicators. Figure 3 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 1,04 (MMRE=29,81 and Pred(25)=73,81). When α is higher than 1,04, the MMRE and pred(25) become not acceptable.

To conclude the Section 4, we compare the accuracy of the RBFN using the C-means algorithm with that of the RBFN when using the APC-III algorithm (Figure 4). We notice that the RBFN with C-means performs better than the RBFN with APC-III. Indeed, an acceptable accuracy of the RBFN-C-Means is still achieved until the number of clusters is equal to 120; by contrast, it was acceptable only until the number of clusters is equal to 150 in the case of the RBFN-APC-III..

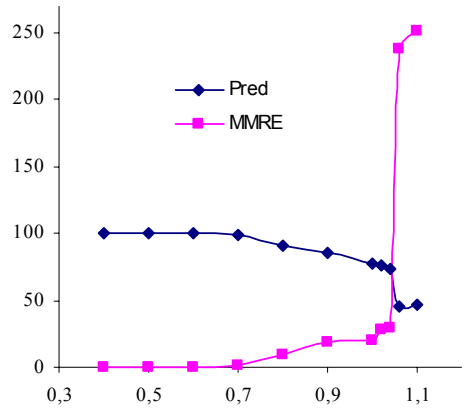

Figure 3: Relationship between the accuracy of RBFN (MMRE and Pred) and the used classification of the APC-III (α and D1).

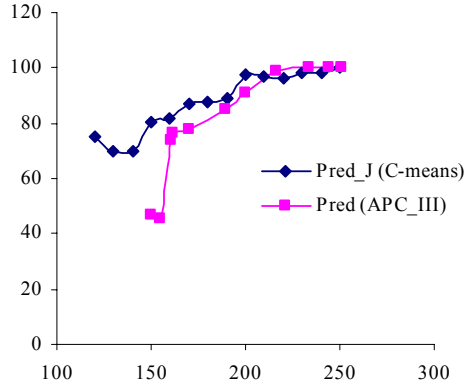

Figure 4: Comparing the accuracy of RBFN using C-means (Pred_J) and RBFN using APC-III (Pred).

5. Conclusion and Future Work

In this paper, we have empirically studied the use of two clustering techniques when designing RBF neural networks for software cost estimation. The two used clustering algorithms are the well-known C-means and the APC-III. This study is based on an artificial COCOMO'81 dataset that contains 252 software projects. We used the entire COCOMO'81 dataset to train and test the designed RBFN. We have found that the RBFN designed with the C-means algorithm performs better, in terms of cost estimates accuracy, than the RBFN designed with the APC-III algorithm. To confirm this affirmation, we are looking currently in applying an RBFN construction based C-means on other historical software projects datasets.

6. Bibliography

[1] P. Berkhin, Survey Of Clustering Data Mining Techniques, 2002, http://citeseer.nj.nec.com/berkhin02survey.html

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

[5] R. O. Duda and P. E. Hart, "Pattern Classification and Scheme Analysis", New York: Wiley, 1973.

[6] Margaret H. Dunham,"Datamining: Introduction and Advanced Topics", Prentice-Hall, 2003

[7] 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.

[8] 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.
[9] 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.
[10] 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.
[11] A. Idri, A Abran, S. Mbarki, 'Validating and Understanding Software Cost Estimation Models based on Neural Networks', International Conference on Information and Communication Technologies: from Theory to Applications (ICTTA), 19-23 April, Damascus, Syria, 2004, pp. 433-438
[12] 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.
[13] Jian Yu, " General C-Means Clustering Model", IEEE Transactions on Patter Analysis and Machine Intelligence, Vol. 27, No. 8, August, 2005.
out such research.
[14] L. Kaufman and P.J. Rousseeuw, "Finding Groups in Data: An Introduction to Cluster Analysis". New York: John Wiley & Sons, Inc., 1990.
[15] J. Moody and C. Darken. "Fast Learning in Networks of Locally-Tuned Processing Units". Neural Computation, vol. 1, 1989, pp. 281-294.
[16] J. Park and I. W. Sandberg. "Approximation and Radial Basis Function Networks." Neural Computation, vol. 5, 1993, pp. 305-316.
[17] W. Pedrycs, "Conditionnal Fuzzy Clustering in the Design of Radial Basis Function Neural Networks", IEEE Transaction on Neural Networks, Vol. 9, No. 4, July, 1998.
[18] M. Shepperd and C. Schofield. "Estimating Software Project Effort Using Analogies." Transactions on Software Engineering, vol. 23, no. 12, 1997, pp. 736-747.
[19] 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.
[20] A. R. Verkatachalam. "Software Cost Estimation using Artificial Neural Networks." International Joint Conference on Neural Networks, Nogoya, IEEE 1993.
[21] S. Vicinanza, and M.J. Prietolla, "Case-Based Reasoning in Software Effort Estimation." Proceedings of the 11th Int. Conf. on Information Systems, 1990.
[22] G. Wittig and G. Finnie. "Estimating Software Development Effort with Connectionist Models." Information and Software Technology, vol. 39, 1997, pp. 469-476.