Proposing an Enhanced Artificial Neural Network Prediction Model to Improve the Accuracy in Software Effort Estimation

Iman Attarzadeh, Amin Mehranzadeh, Ali Barati
Department of Computer Engineering
Dezful Branch, Islamic Azad University
Dezful, Iran
{attarzadeh, mehranzadeh ,abarati}@iaud.ac.ir

Abstract—**Software companies develop different software in parallel, which is a very complex task. Project managers have to manage different software development processes based on different time, cost, and number of staff, sequentially. Software time, cost, and number of staff estimates are the critical tasks for project managers in software companies. Estimation of these parameters at early stage of software project planning is one the challenging issued in software project management, for the last decade. Software cost and time estimation supports the project planning and tracking, and it controls the expenses of software development. Software effort estimation refers to the estimates of the likely amount of cost, schedule, and manpower required to develop a software. Accurate effort estimate at the early phase of software development can help project managers to efficiently control project progress and improve the project success rate. This paper proposes a novel artificial neural network (ANN) prediction model incorporates Constructive Cost Model (COCOMO), ANN-COCOMO II, to provide more accurate software estimates at the early phase of software development. This model uses the advantages of artificial neural networks such as learning ability and good interpretability, while maintaining the merits of the COCOMO model. The ANN is utilised to calibrate the software attributes using past project data, in order to produce accurate software estimates. The proposed model is evaluated using 156 sets of project data from two data sets, COCOMO I and NASA93. The analysis of the obtained results shows 8.36% improvement in estimation accuracy in the ANN-COCOMO II model, when compared with the original COCOMO II.**

Keywords- Software engineering, software project management, software cost estimation models, COCOMO model, soft computing techniques, and artificial neural networks.

I. INTRODUCTION

Project cost and time estimation is of the challenging issues in software project management. Enhancing the precision of project time and cost prediction in the existing software development estimation models would facilitate the time and budget management of software projects during the development process. Software effort estimation models helps project managers to estimate project time, cost, and manpower required for the software development. Using reliable and precise software effort estimation models is still an ongoing challenge for software project managers. In order to provide precise estimates and decrease the estimation error, several cost estimation models have been proposed and developed. Among these models, Constructive Cost Model (COCOMO) is the most commonly used model in the software companies because of its capabilities and features for estimating the software effort in person-month (PM) at different development stages. This paper proposed a precise artificial neural network estimation model incorporates COCOMO model to overcome the vagueness and uncertainly of the input parameters in the COCOMO model. This paper is organised as follows. Section II, describes the characteristics and weakness of existing effort estimation models, particularly COCOMO II model, and advantages of artificial neural networks that used in the proposed model. Sections III, discusses the related works to the software effort estimation. Sections IV and V present the new artificial neural network estimation model incorporates COCOMO II model, and the appropriate training algorithm. Section VI, discusses the used data sets and analysis of the results. Finally, section VII concludes this paper.

II. SOFTWARE EFFORT ESTIMATION MODELS AND ARTIFICIAL NEURAL NETWORKS

A. Software Development Effort Estimation Models

Software project managers and programmers always are interested to estimate the project cost and time at the early phase of software development process. One of the common methods is comparing the existing tasks with similar tasks that have already been developed. Although, using different factors to make estimation, nature of software estimation and developing software in different environment caused that the software engineers could not propose a fix and universe estimation model. Software cost and time estimation supports planning and tracking of development process. Effective controlling of software costs and time during development process is of high importance [1]. Understanding the value of cost and time estimation in small software companies can help them to become a mature software company. The precise and reliable software time and cost estimation is an ongoing challenge in software project management, as it can help project managers to make good decision on software cost and strategic planning. Software effort prediction methods can be categorised into algorithmic and non-algorithmic methods. Algorithmic methods use regression techniques, the statistical analysis of historical data (past projects). Software Life Cycle Management (SLIM) [1] and Constructive Cost Model (COCOMO) [2] are two common estimation methods in software engineering. Non-algorithmic methods are based on heuristic approaches such as Expert Judgment, Price-to-Win and machine learning approaches. Machine learning approach uses a set of computer-based techniques that represent some of the real facts of human mind [3]. The important techniques in machine learning are regression trees, rule induction, and variety of soft computing techniques such as fuzzy systems, genetic algorithms, artificial neural networks, Bayesian networks and evolutionary computation. Soft decision- making, trainable algorithm, using flexible rules and reputation are a part of advantages of soft computing methods.

B. The Constructive Cost Model (COCOMO)

The Constructive Cost Model (COCOMO) is the well-known software effort estimation model based on regression techniques. The COCOMO model was proposed by Barry Boehm in 1981[2] and it is one of the most cited, best known, widely used and the most plausible of all proposed effort estimation methods. The COCOMO model uses to calculate the amount of effort then based on the calculated effort it makes time, cost and number of staff estimates for software projects. COCOMO 81 was the first and stable model on that time. Nowadays, the big problem with using COCOMO 81 is that it matches software parameters and the development environment of the 1990's. Therefore, in 1997 the new version of it, COCOMO II, was proposed and developed to solve most of the COCOMO 81 problems. The Post-Architecture Level of COCOMO II uses 17 cost drivers that they presents project attributes, programmer abilities, developments tools, and etc. [2]. The cost drivers and scale factors for COCOMO II are rated on a scale from Very Low to Extra High in the same way as in COCOMO 81. COCOMO II post architecture level is shown in "(1)".

$$\text{Effort} = A \times [\text{Size}]^B \times \prod_{i=1}^{17} \text{Effort Multiplier}_i \quad (1)$$

$$\text{where } B = 1.01 + 0.01 \times \sum_{j=1}^{5} \text{Scale Factor}_j$$

In "(1)":

A: Multiplicative Constant

Size: Size of the software measures in terms of KSLOC (thousands of Source Lines of Code, Function Points or Object Points).

The scale factors (SF) are based on a significant source of exponential variation on a project's effort or productivity variation.

C. Artificial Neural Networks

Artificial neural network (ANN) is a basic knowledge of how the human brain works. Artificial neural networks are simplified mathematical techniques of human brain, a collection of neurons or process elements with internal connection, and they function as parallel distributed computing networks. However, in contrast to existing computers, which are programmed by a programmer to perform sequential of commands, codes, most artificial neural networks must be trained. They can learn from previous project information, experiences and details to provide new data, rules, and experiences based on the inferring from learnt data. That is one of the basic advantages of using ANN. The main idea in ANN is to produce intelligent systems capable of sophisticated computations. It is similar to the biological neurons in human brain structures. In fact, each neuron is like a mathematical function with some inputs, a mathematical formula, and outputs. Artificial neural networks include some active and hidden layers. Each layer has interconnected nodes, neurons, where each node generates a non-linear function of its input [3]. The most widely used training techniques in the artificial neural network for estimation is known as back-propagation and feed-forward techniques. Neural network has been used in the software architecture, modeling, reliability as well as software risk management [4]. Fig. 1 shows a simple artificial neural network.

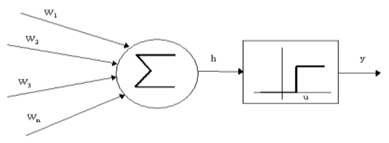

Figure 1. A simple of artificial neural networks

Each artificial neuron produce the weighted sum of its M inputs, x_j, where j = 1,2,……m, and generate an output of 1 if this result is above the defined threshold u. Otherwise, an output of 0 generates. The obtained formula is shown in "(2)".

$$y = \theta\left(\sum_{j=1}^{m} W_j X_j - u\right) \qquad (2)$$

In "(2)", θ is a unit step function at 0 and w_j is the synapse weight associated with the j^{th} input. U is considered as another weight i.e. $w_0 = -u$ attached to the neuron with a constant input of $x_0 = 1$. Positive weights model excitatory synapses, while negative weights model inhibitory ones. The activation function in Figure 1 is known as a step function however, there are a number of functions that can be utilised such as Sigmoid, Gaussian, and Linear. The Sigmoid function is the most widely used in artificial neural networks. Architectures of ANN are divided into two categories:

- Feed-forward architecture, where no loops in the network path occur
- Feedback architecture: that have recursive loops

Combined architecture is possible architecture for ANN. Between the different ANN architectures combined model, the feed-forward back-propagation, is the most widely used model [5, 6].

III. Related Work

Software engineers from different places of the world share their knowledge to improve the accuracy of software effort estimation models to overcome the problem. Many software effort estimation models have been proposed and developed over the last decades. Besides, among of soft computing approaches artificial neural networks, have good learning ability; uses to model complex nonlinear algorithms and relationships; provides more reliability and flexibility to integrate previous experiences and expert knowledge into the model. Using artificial neural networks in software development effort estimation can provide accurate results in algorithmic estimation models. The accurate effort, time and cost, estimation is one of the wishes of software project managers. Using back propagation learning algorithm on a multilayer perceptron is one good sample of soft computing techniques. It proposes by Witting and Finnie [7] to estimate software development effort. In another research, Karunanithi [8] suggests using artificial neural networks techniques such as the feed forward and Jordon networks with expert experiments to estimate software flexibility and reliability. That is another application of ANN in software engineering. Samson [9] uses another soft computing technique, an Albus multiplayer perceptron, to estimate software development time and cost. The COCOMO dataset includes 63 projects data that commonly uses for the evaluation of the developed systems as well as experimental data. A different artificial neural network with a back propagation learning algorithm proposes by Tadayon [10], however, in that research it is not clear how the applied dataset was divided to train and validate of the proposed system. Khoshgoftaar and Jingzhou [11, 12] present a case study to consider the real time approach to estimate the usability of each module from some metrics such as source lines of code. The researchers try to apply and use the advantages of artificial neural networks to propose an accurate, reliable and flexible software estimation model. ANN has been successfully used to several problems in software engineering. Those models can be used as estimation models because they are model based techniques and capable to compute complex functions [13, 14].

IV. The Proposed COCOMO Model Incorporates Artificial Neural Networks

The proposed artificial neural network model is established to accommodate the COCOMO II Post-architecture model, which was described in the previous section in "(1)". The COCOMO II includes five scale factors (SF) and seventeen effort multipliers (EM). Each of scale factors and effort multipliers show the critical attributes of project such as product attributes, hardware attributes, and personnel attributes. Also, they differ for different projects. The input parameters of COCOMO model in "(1)" are 25 and include size, scale factors and effort multipliers, and the system output is effort estimation in PM (person-month). Therefore, the new artificial neural network model has 23 input nodes in the input layer that corresponds to all size, SFs and EMs as well as two bias values. Besides, to establish the COCOMO II post-architecture model, some pre-processing of data for input layer, one specific ANN hidden layer and a sigmoid activation function are considered. The established ANN model is not a fully connected network, but indicated hidden layers nodes that take into account the contribution of scale factors and effort multipliers separately as shown in Fig. 2 and 3.

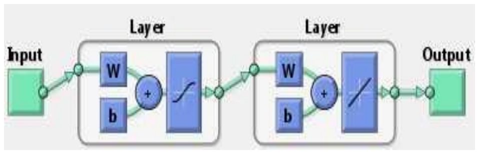

Figure 2. Structure of proposed artificial neural network

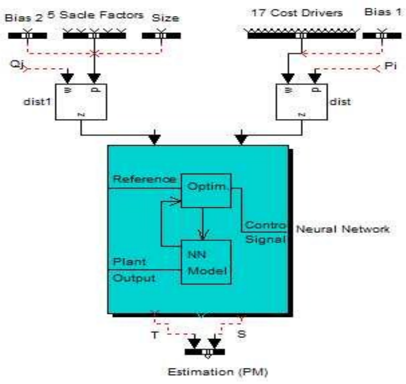

Figure 3. The proposed artificial neural network based on COCOMO II

In the proposed ANN model, all effort multipliers values, EM_i, used in COCOMO model are pre-processed to $log(EM_i)$ and the size of the product, in KSLOC, is not considered as one of the input parameters to the network but as a co-factor for the initial weights, initialisation parameter, for scale factors (SF). The sigmoid activation function in the hidden layer is defined by $f(x) = \frac{1}{1+e^{-x}}$. The related weights of input nodes connected to the hidden layer are defined by P_i for Bias1 and each input $log(EM_i)$ for $1 \leq i \leq 17$. Besides, the related weights with each scale factor, SF_j, from input nodes to the hidden layer are q_j + log(size) for $1 \leq j \leq 5$ and the bias defined by Bias2. The parameters 'W' and 'b' in Figure 2, show the related weights to the arcs from the hidden layer nodes to the output layer nodes. The weight parameters 'W 'and 'b', are relevant to the values of the hidden layer nodes. The output nodes have the specific identity function. One of the contributions of this research compared to other related works is the addition of Log(Input Size) to the weight Q_j of scale factors in system input, which adjusts the weights Q_j . Another important difference in this model compared to other works is the training and biasing approach of artificial neural network. Customisation of the COCOMO formula is done by adjusting the initial values of weights 'W' and 'b' to the offset of the values of the nodes in the ANN hidden layers. The back-propagation algorithm is used as the training method in the proposed model. The data sets that used for system training will discuss at the following section. However, if there is no appropriate data set for system training, the weights and biases parameters in the model results the estimation using random generated input/output. The proposed model output, the effort, would be derived from COCOMO, in "(1)", by considering the initial values of Bias1 as Log(A) and Bias2 as 1.01. The ANN weights are initialised as $p_i = 1$ for $1 \le i \le 17$ and $q_j = 1$ for $1 \le j \le 5$. For propagating the input parameters the values of nodes in the hidden layer are considered as follow:

$$f\left(p_0\, Bias1 + \sum_{1}^{17} p_i * log(EM_i) \right) = sigmoid \left(Bias1 + \sum_{1}^{17} p_i * log(EM_i) \right) = \frac{A * \prod_{i=1}^{17} EM_i}{1 + A * \prod_{i=1}^{17} EM_i} = \alpha \qquad (3)$$

$$f((q_0 + log(size)) * Bias2 + \sum_{j=1}^{5} \left(q_j + log(size)\right)\left(SF_j\right) = sigmoid\ (log(size) * (\mathrm{Bias2} + \sum_{j=1}^{5} \mathrm{SF_j}) = \frac{\mathrm{Size}^{1.01 + \sum_{j=1}^{5} \mathrm{SF_j}}}{1 + \mathrm{Size}^{1.01 + \sum_{j=1}^{5} \mathrm{SF_j}}} = \beta \qquad (4)$$

Then initialisation of weights 'W' and 'b' as follow:

$$\mathrm{W} = \frac{\beta}{2\,(1-\alpha)(1-\beta)} \quad \text{and} \quad \mathrm{b} = \frac{\alpha}{2\,(1-\alpha)(1-\beta)} \qquad (5)$$

The ANN output is calculated as:

$$\mathrm{PM} = \mathrm{W} * \alpha + \mathrm{b} * \beta = \frac{\alpha\beta}{(1-\alpha)(1-\beta)} = \mathrm{A}.\mathrm{Size}^{1.01 + \sum_{j=1}^{5} \mathrm{SF_j}} * \prod_{i=1}^{17} \mathrm{EM_i} \qquad (6)$$

Note that the initial values of S and T are adjusted so that the nodes in the hidden layer have same contribution to the output node PM. The following section describes the procedure for training the network.

V. Training Algorithm

The training algorithm is done by iteration of forward and backward techniques until the terminating conditions are satisfied. The terminating conditions are when all changes in weights are less than or equal a specific threshold or a specific number of iterations have been done. The training algorithm is as follow steps:

- Selecting a training sample and propagate the input parameters across the ANN to compute the system output.
- Error detection in system output, and determining the amount of error gradient in all the other layers.
- Determining the amount of changes for the ANN weights and updating the ANN weights.
- Repeating the steps until the ANN error is sufficiently small, less than or equals a specific threshold, after an epoch is complete.

The rate of learning approach that affects the changes in weights corresponding to effort multipliers and scale factors have been subscripted with their related layers nodes.

VI. Results and Discussion

The network training and evaluation of the ANN-COCOMO II were carried out using two different data sets:

- Data set #1—the original COCOMO I data set, which includes 63 sets of project data [1].
- Data set #2—NASA'93 data set, which includes 93 sets of project data in small, medium, and large-scale project size [1].

A. Datasets Description

These two data sets include a number of past projects data, gathered from software companies, and NASA93 software projects. Boehm [1] was the first researcher that established and validated an appropriate software development cost estimation model based on a data set—the COCOMO I data set. These two data sets are available in the public domain [1] and were used in the network training and evaluation of the ANN-COCOMO II. Table I shows samples of the two data sets (Data set #1 and Data set #2).

TABLE I. Samples of the Project Data

No.	Mode	Size	Effort
1	1.1200	51.2500	246.5900
2	1.2000	12.5500	58.2800
3	1.0500	81.5200	550.4000
…	...	…	…
97	1.2000	56.5300	354.7300
98	1.0500	16.0400	67.1400
100	1.1200	54.1700	262.3800

B. *Evaluation Method*

The evaluation of the ANN-COCOMO II was done by using the most widely-used evaluation methods: Mean Magnitude of Relative Error (MMRE), and Prediction at level L (PRED (L)). The Magnitude of Relative Error (MRE) is defined as follows:

$$\mathrm{MRE_i} = \frac{|\text{Actual Effort}_{\,i} - \text{Estimated Effort}_{i}|}{\text{Actual Effort}_{\,i}} \quad \text{i: project number} \quad (7)$$

The MRE was calculated for each software project, i, based on "(7)". The mean of MRE over multiple projects (for i=1 to N) can be achieved through the Mean MRE (MMRE) equation as follows:

$$\mathrm{MMRE} = \frac{1}{N}\sum_{i}^{N} \mathrm{MRE_i} \qquad (8)$$

A complementary evaluation condition is prediction at level L, PRED (L) = k/N, where k is the number of project data where MRE is less than or equals to L, and N is the total number of projects. The PRED (L) method shows the probability of a project having a relative error of less than or equal to level L. For example, PRED (25%) gives the percentage of projects which have been estimated based on the MRE – less than or equal to 25%.

The two data sets, Data set #1 and Data set #2, were applied to the ANN-COCOMO II and the original COCOMO II, respectively. For each project data in the data sets, the estimated effort, MRE, and PRED (25%) were calculated. Finally, the MMRE for each data set was calculated to avoid any sensitivity to each data set. The comparison of the results obtained from the original COCOMO II, and the ANN-COCOMO II, is shown in Table II.

TABLE II. COMPARISON OF THE RESULTS OBTAINED FROM THE ORIGINAL COCOMO II AND THE ANN-COCOMO II

Data set	**Model**	**Evaluation**	
		MRE	PRED (25%)
Data set #1	COCOMO II	0.542561832	45%
	ANN-COCOMO II	0.487257017	52%
Data set #2	COCOMO II	0.462579313	30%
	ANN-COCOMO II	0.428617366	39%
Mean	COCOMO II	0.502570573	37.5%
	ANN-COCOMO II	0.457937192	45.5%

Table 2 shows the results of when the two different data sets were applied on the ANN-COCOMO II and the original COCOMO II, respectively. The last column of Table 3 shows the mean of MMRE and PRED (25%) for the three data sets (156 sets of project data). The results show that the mean of MMRE and PRED (25%) are 0.457937192 and 45.5% for the ANN-COCOMO II and 0.502570573 and 37.5% for the original COCOMO II. The analysis of the results shows that:

- A software development cost estimation model with a smaller MMRE value gives better estimates than a model with a bigger MMRE value. The ANN-COCOMO II produced the MMRE=0.457937192, which is less than the MMRE=0.502570573, in the original COCOMO II. It implies that the estimation accuracy of ANN-COCOMO II is much better than the COCOMO.
- A software development cost estimation model with a bigger PRED (25%) value gives better estimation accuracy than a model with a smaller value. The ANN-COCOMO II produced the PRED (25%) = 45.5% which is bigger than the PRED (25%) = 37.5% in the original COCOMO II. It implies that the estimation accuracy of ANN-COCOMO II is much better than the original COCOMO II.
- Based on the results obtained, the ANN-COCOMO II produces more accurate results than the original COCOMO II. Also, it shows improvement in estimation accuracy of the ANN-COCOMO II.

Table III shows the comparison of the improvement in estimation accuracy between the ANN-COCOMO II and the original COCOMO II.

TABLE III. IMPROVEMENT IN ESTIMATION ACCURACY OF THE ANN-COCOMO II AND THE ORIGINAL COCOMO II

Model	**Evaluation**	
		MMRE
ANN-COCOMO II vs. COCOMO II	COCOMO II	0.502570573
	ANN-COCOMO II	0.457937192
	Improvement %	8.36%

Table III shows the percentage of the improvement in estimation accuracy of the ANN-COCOMO II in terms of MMRE when compared with the original COCOMO II. The ANN-COCOMO II shows 8.36% improvement in the MMRE. The results imply that the ANN-COCOMO II produces more accurate software estimates than the original COCOMO II. Also, the ANN-COCOMO II has better performance due to the high granularity demanded from the results.

VII. CONCLUSION

Accurate and reliable software project estimates such as time, cost, and manpower in the early phase of software development, is one of the crucial objectives in software project management. Software project attributes are often vague and uncertain as they are estimated or calculated by human judgement, and also there is difference in the software development environment. A software development cost estimation model incorporating artificial neural networks can alleviate the problems associated with the vagueness and uncertainty of the software attributes. This approach would be a worthy attempt in software development cost estimation because it can reduce the effect of inaccurate software attributes on the software estimates through data calibration approach. This paper described the research on the incorporation of an adaptive artificial neural network architecture in the COCOMO II (ANN-COCOMO II) to handle the uncertain and imprecise software attributes. The results showed that using artificial neural networks for calibration of the COCOMO II software attributes can lead to more accurate software estimates. The ANN-COCOMO II showed an 8.36% improvement in estimation accuracy in MMRE when compared with the original COCOMO II. The application of other soft computing techniques such as fuzzy logic, evolutionary computation, and neuro-fuzzy systems can be explored for other software development cost estimation models in the future.

REFERENCES

[1] B. Boehm, Software Engineering Economics, Prentice-Hall: Englewood Cliffs, NJ, 1981.

[2] B. Boehm, C. Abts, and S. Chulani, “Software Development Cost Estimation Approaches – A Survey,” University of Southern California Center for Software Engineering, Technical Reports, USC-CSE-2000-505, 2000.

[3] L. H. Putnam, “A General Empirical Solution to the Macro Software Sizing and Estimating Problem,” IEEE Transactions on Software Engineering, vol. 4, no. 4, 1978, pp. 345 – 361.

[4] K. Srinivasan and D. Fisher, “Machine Learning Approaches to Estimating Software Development Effort,” IEEE Transactions on Software Engineering, vol. 21, no. 2, 2008, pp. 117-128.

[5] K. Molokken and M. Jorgensen, ”A review of software surveys on software effort estimation,” IEEE International Symposium on Empirical Software Engineering (ISESE’03), 2003, pp. 223 – 230.

[6] S. Huang and N. Chiu, ”Applying fuzzy neural network to estimate software development effort,” Applied Intelligence Journal, vol. 30, no. 2, 2010, pp. 73-83.

[7] G. Witting and G. Finnie, “Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort,” Journal of Information Systems, vol. 1, no.2, 1994, pp. 87-94.

[8] N. Karunanitthi, D. Whitely, Y. K. Malaiya, “Using Neural Networks in Reliability Prediction,” IEEE Software Engineering, vol. 9, no.4, 2011, pp. 53-59.

[9] B. Samson, “Software cost estimation using an Albus perceptron,” Journal of Information and Software, vol. 7, no. 3, 2011, pp. 55-60.

[10] N. Tadion, ”Neural Network Approach for Software Cost Estimation,” International Conference on Information Technology: Coding and Computing (ITCC), 2005, pp. 128-134.

[11] T. M. Khoshgoftar, E. B. Allen, and Z. Xu, “Predicting testability of program modules using a neural network,” The 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 2010, pp. 57-62.

[12] L. Jingzhou and R. Guenther, “Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+,” Empirical Software Engineering Journal, vol. 13, no. 1, 2008, pp. 63–96.

[13] H. Liu and L. Yu, “Toward Integrating Feature Selection Algorithms for Classification and Clustering,” IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 4, 2007, pp. 491-502.

[14] N. H. Chiu and S. J. Huang, “The adjusted analogy-based software effort estimation based on similarity distances,” Journal of Systems and Software, vol. 25, no. 2, 2009, pp. 628–640.