# Proposing a Novel Artificial Neural Network Prediction Model to Improve the Precision of Software Effort Estimation

Iman Attarzadeh and Siew Hock Ow

Department of Software Engineering
Faculty of Computer Science & Information Technology
University of Malaya, 50603 Kuala Lumpur, Malaysia
attarzadeh@siswa.um.edu.my, show@um.edu.my

**Abstract.** Nowadays, software companies have to mange different software development processes based on different time, cost, and number of staff sequentially, which is a very complex task and supports project planning and tracking. Software time, cost and manpower estimation for separate projects is one of the critical and crucial tasks for project managers. Accurate software estimation at an early stage of project planning is counted as a great challenge in software project management, in the last decade, as it allows considering project financial, controlling, and strategic planning. Software effort estimation refers to the estimations of the likely amount of cost, schedule, and manpower required to develop software. This paper proposes a novel artificial neural network prediction model incorporating Constructive Cost Model (COCOMO). The new model uses the desirable features of artificial neural networks such as learning ability, while maintaining the merits of the COCOMO model. This model deals efficiently with uncertainty of software metrics to improve the accuracy of estimates. The experimental results show that using the proposed model improves the accuracy of the estimates, 8.36% improvement, when the obtained result compared to the COCOMO model.

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

## 1 Introduction

Accurate and reliable software cost and time estimation is counted as one of the great problems and ongoing challenges in software engineering, in the last decades. Enhancing the precision of estimates would facilitate more effective time and budgets controlling for project managers during the development process. In order to make precise software estimates, several algorithmic and non-algorithmic cost estimation models have been proposed and developed. The Constructive Cost Model (COCOMO) is the most popular model in software companies due to its capabilities and characteristics to estimate software effort in person-month (PM) during

J. Suzuki and T. Nakano (Eds.): BIONETICS 2010, LNICST 87, pp. 334–342, 2012.

development process. This paper proposes a precise artificial neural network estimation model incorporating COCOMO model to overcome the vagueness and uncertainty of software estimates.

### 1.1 Software Effort Estimation Models

Software project managers and developers, always, interested to estimate the total software budget and schedule at the early stages of development process. These estimates can help them to make good decisions on project management and strategic planning. Software effort prediction approaches can be categorised into algorithmic and non-algorithmic methods. Algorithmic methods use regression techniques with statistical analysis of historical data. Software Life Cycle Management (SLIM) [1] and Constructive Cost Model (COCOMO) [2] are two common algorithmic estimation methods. Non-algorithmic methods are based on heuristic approaches such as Expert Judgment, Price-to-Win, and machine learning approaches [3]. 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 for project effort, time, cost, and manpower estimations. The COCOMO II model includes three sub-models for software estimations as follows: Application Composition Level, Early Design Level, and Post-Architecture Level. The Post-Architecture Level of COCOMO II includes 17 project cost factors, 5 scale factors, and software size, which presents project attributes and characteristics [2]. The COCOMO II formula shown in equation "1" as follows:

$$\text{Effort} = \text{A} \times [\text{Size}]^{\text{B}} \times \prod_{\text{i}=1}^{17} \text{Effort Multiplier}_{\text{i}} \tag{1}$$

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

In the equation"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)

### 1.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are simplified mathematical techniques of human brain. They are collection of neurons, Process Elements (PEs), with internal connection and their function is based on distributed computing networks. They can learn from previous project information and experiences to provide new data, rules, and experiences based on inference of learnt data. 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. Each ANN includes a specific architecture, layers, and nodes. Each node has a mathematical function, inputs, and outputs, which generates a non-linear function of its input [3, 4]. Using ANN starts by generating network architecture then selecting a proper leaning technique for train, test, and validate the network based on a data set. The most widely used ANN training techniques are feed-forward and recurrent techniques. Figure 1 shows the functionality of a node in ANN.

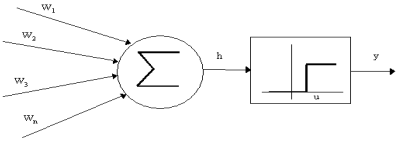


**Fig. 1.** The functionality of a node in ANN

Each node 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 equation "2".

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

In the equation "2", the 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. w0 = -u attached to the neuron with a constant input of x0 = 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 [5, 6].

## 2 Related Works

Software researchers attempt to improve software effort estimation models to overcome the uncertainty of results. Many software effort estimation models have been proposed and developed over the last decades. Using capabilities of artificial neural networks in software effort estimation, especially learning from historical project, can be as an alternative to achieve acceptable results. Using back propagation learning algorithm on a multilayer perceptron is one good application of soft computing techniques, which proposed by Witting and Finnie [7] to estimate software development effort. In another research, Karunanithi [8] suggested using artificial neural network techniques such as the feed-forward and Jordon-network with expert experiments to estimate software flexibility and reliability.

Samson [9] utilised another soft computing technique, Albus multiplayer perceptron, to estimate software development time and cost. He used the COCOMO data set, which includes 63 projects information. A different artificial neural network

with back propagation learning algorithm proposed by Tadayon [10], however, it is not clear how the applied dataset was divided to train and validate of his proposed system. Khoshgoftaar and Jingzhou [11, 12] considered a real time approach to estimate the usability of each software metrics such as source lines of code. Researchers still attempt to apply and use the advantages of artificial neural networks to propose an accurate, reliable and flexible software estimation model. The ANN has been successfully used for solving several problems in software engineering [13, 14].

## 3 The Proposed COCOMO Model Incorporating Artificial Neural Networks

In this research a new architecture of ANN proposes to accommodate the COCOMO II Post-architecture model. The COCOMO II includes five scale factors (SF), seventeen effort multipliers (EM), and software size. The proposed model has 23 inputs, which include software size, scale factors, effort multipliers and the system output is effort estimation in PM (person-months). Therefore, the proposed artificial neural network architecture includes 23 input nodes in the input layer, which corresponds to all SFs, EMs, and software size parameters. Applying the proposed ANN architecture on the COCOMO II post-architecture model needs to data pre-processing in the input layer using the Sigmoid Activation function. The proposed ANN architecture for the COCOMO model shown in Figure 2 and 3 and follows:

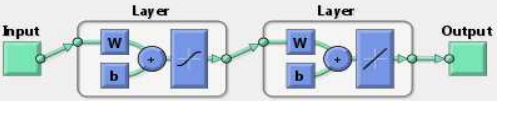


**Fig. 2.** Architecture of proposed artificial neural network

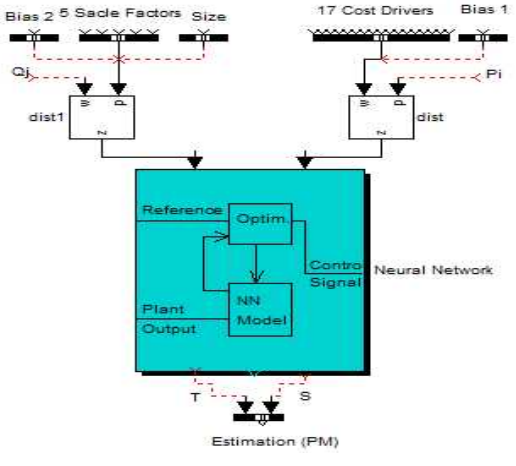


**Fig. 3.** The proposed artificial neural network based on COCOMO II

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 equation, '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 $1 \leq i \leq 17$ and $q_j = 1$ for $1 \leq j \leq 5$ . For propagating the input parameters the values of nodes in the hidden layer are considered as follow:

$$f(\ p_0\ Bias1 + \sum_1^{17} p_i * log(EM_i)) = sigmoid\ (Bias1 + \sum_1^{17} p_i * log(EM_i)) = \frac{A * \prod_{i=1}^{17} EM_i}{1 + A * \prod_{i=1}^{17} EM_i} = \alpha \qquad (3)$$

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

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:

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

# 4 Training Algorithm

The training algorithm is done by iteration of forward and backward techniques until the terminating conditions are satisfied, for instance the changes in weights are less than or equal to a basic threshold or a basic number of iterations have been done.
The training algorithm includes 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.

# 5 Results and Discussion

Experiments were done by using two different data sets: the original COCOMO data set, which includes 63 projects information and an artificial data set, which includes 100 projects data. The artificial data set inferred from the existing COCOMO dataset based on another ANN model using previous training algorithm.

## 5.1 Data Sets Description

The COCOMO data set, Data set #1, was the first attempt to evaluate the proposed ANN-COCOMO model, which is includes 63 historical projects and it is a public data set [1]. This data set retrieved form 63 historical projects from software companies and industries. The second data set created based on the COCOMO data set by proposing a new ANN architecture, Data set #2, which is trained by the COOCMO data set and characteristics of training algorithm explained in the previous section. Table 1 shows the structure of second data set.

**Table 1.** The artificial data set includes 100 projects information

| 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 |

## 5.2 Evaluation Method

The proposed ANN-COCOMO model evaluation and validation is done by using the most widely accepted evaluation methods: Mean Magnitude of Relative Error

(MMRE) and Pred (L). The Pred(L) method means probability of a project having a relative error of less than or equal to L ( for instance, Pred(25%)). The Magnitude of Relative Error (MRE) is defined as follows:

$$\mathrm{MRE_i} = \frac{|\text{Actual Effort }_i - \text{Predicted Effort}_i|}{\text{Actual Effort }_i} \tag{7}$$

The value of MRE is calculated for each observation i whose effort is estimated. The aggregation of MRE over multiple observations (for i=1 to N) can be achieved through the Mean MRE (MMRE) as follows:

$$\mathrm{MMRE} = \frac{1}{N} \sum_i^N \mathrm{MRE_i} \tag{8}$$

The Magnitude of Error Relative to the estimation (MER) is another measure similar to MRE. Intuitively, MRE seems preferable to MER since it measures the relative error to the estimate. MRE uses predicted effort as shows in equation "7". The MMRE is used to the mean MER in equation "8". However, the MMRE and MMER are sensitive to individual estimations with excessively large MREs or MERs. Therefore, the aggregate measure less sensitive to extreme values is also considered, namely the median of MRE and MER values for the N observations (MdMRE and MdMER respectively). A complementary condition is the prediction at level l, Pred(l) = k/N, where k is the number of observations where MRE (or MER) is less than or equal to l, and N is the total number of observations. Thus, Pred(25%) gives the percentage of projects which were predicted with a MRE (or MER) less or equal than 0.25.

The proposed ANN-COCOMO model and the original COCOMO model are used for model evaluation. The two data sets, Data set #1 and Data set #2, separately applied to the new artificial neural network effort estimation model and original COCOMO model. For each project in the data set the estimated effort, MRE, and Pred (25%) calculated by applying on the proposed ANN model and the original COCOMO model. Finally, the MMRE for each data set is calculated to avoid any sensitivity to the calculated results. The results comparison of COCOMO data set, Date Set #1, and artificial dataset, Data Set #2, shown in Tables 2 and 3.

**Table 2.** Results comparison of the ANN proposed model and COCOMO model

| Data set | Model | Evaluation | |
|---|---|---|---|
| | | MRE | Pred (25%) |
| Data set #1 | COCOMO II | 0.542561832 | 45% |
| | Proposed Model | 0.487257017 | 52% |
| Data set #2 | COCOMO II | 0.462579313 | 30% |
| | Proposed Model | 0.428617366 | 39% |
| MMRE | COCOMO II | 0.502570573 | 37.5% |
| | Proposed Model | 0.457937192 | 45.5% |

The results in the Table 2 shows: the values of MMRE in the proposed ANN model for Data set #1 and #2 are 0.487257017 and 0.428617366 and for the Pred(25%) are 52% and 39%. The aggregation of Data sets, #1 and #2 shows that the

MMRE is 0.457937192 and Pred(25%) equals 45.5%. The analysis of the results in Table 2 indicates:

- The proposed ANN effort estimation model has the MMRE, MMRE_ANN = 0.457937192, less than COCOMO model, MMRE_COCOMO = 0.502570573. Obviously, it means the accuracy of proposed ANN model is better than COCOMO model. Because, if the value of MMRE is closed to zero, it means the amount of error, difference of actual and estimated effort, is very low. In other words, the accuracy of estimation is high.
- In case of Pred(25%), the value of Pred(25%) for proposed ANN estimation model is 45.5% and for COCOMO model is 37.5%. Pred method presents the number of projects with the MRE lass than 25%. Therefore, if the value of Pred(25%) is closed to 100%, it means the estimated value for the project is closed to the actual amount. So, the accuracy of estimation is high.

According to the analysis of the final results, the proposed ANN effort estimation model shows better accuracy than the COCOMO model. Table 3, compares the percentage of the accuracy improvement of ANN estimation model in compare to the COCOMO model.

**Table 3.** The accuracy of the estimation models

| Model | Evaluation | |
|---|---|---|
| | | MMRE |
| Proposed Model vs. COCOMO II | COCOMO II | 0.502570573 |
| | Proposed Model | 0.457937192 |
| | Improvement % | 8.36% |

The analysis of the final result indicates that the percentage of the accuracy improvement in the proposed artificial neural network model is 8.36%. In summary, the analysis of the experimental results shows that the proposed artificial neural network effort estimation model generates more accurate estimates than the COCOMO model and the system output provides a better performance due to the high granularity demanded from the results.

## 6 Conclusion

One on the crucial and challenging issues in software project management is accurate and reliable estimation of the required effort at the early stages of software development process. Software attributes essentially have properties of vagueness and uncertainty when they are measured by human judgment and they differ from software development environments. A software effort estimation incorporating artificial neural networks can overcome the vagueness and uncertainty of software attributes, which used in effort estimation. This approach can be a worthy attempt in the software project

management. This research work presented a new artificial neural network architecture to handle uncertainty and imprecision in software effort estimation.

This approach shows that by applying artificial neural network on the algorithmic effort estimation models, accurate estimates are achievable. The analysis of the results demonstrated that using artificial neural network approach for the software cost estimation is an applicable approach to address and overcome the vagueness and uncertainty of software attributes. Furthermore, the proposed artificial neural network estimation model presented better estimation accuracy when compared to the COCOMO model. The utilisation of soft computing approaches such as artificial neural networks, fuzzy logic, and neuro-fuzzy systems for other software engineering approaches can be considerable in the future.

## References

[1] Boehm, B.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)
[2] Boehm, B., Abts, C., Chulani, S.: Software Development Cost Estimation Approaches – A Survey. University of Southern California Center for Software Engineering, Technical Reports, USC-CSE-2000-505 (2000)
[3] Putnam, L.H.: A General Empirical Solution to the Macro Software Sizing and Estimating Problem. IEEE Transactions on Software Engineering 4(4), 345–361 (1978)
[4] Srinivasan, K., Fisher, D.: Machine Learning Approaches to Estimating Software Development Effort. IEEE Transactions on Software Engineering 21(2) (1995)
[5] Molokken, K., Jorgensen, M.: A review of software surveys on software effort estimation. In: IEEE International Symposium on Empirical Software Engineering, ISESE, pp. 223–230 (October 2003)
[6] Huang, S., Chiu, N.: Applying fuzzy neural network to estimate software development effort. Applied Intelligence Journal 30(2), 73–83 (2009)
[7] Witting, G., Finnie, G.: Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort. Journal of Information Systems 1(2), 87–94 (1994)
[8] Karunanitthi, N., Whitely, D., Malaiya, Y.K.: Using Neural Networks in Reliability Prediction. IEEE Software Engineering 9(4), 53–59 (1992)
[9] Samson, B.: Software cost estimation using an Albus perceptron. Journal of Information and Software, 55–60 (1997)
[10] Tadion, N.: Neural Network Approach for Software Cost Estimation. In: International Conference on Information Technology: Coding and Computing, ITCC, pp. 116–123 (2005)
[11] Khoshgoftar, T.M., Allen, E.B., Xu, Z.: Predicting testability of program modules using a neural network. In: 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, pp. 57–62 (2000)
[12] Jingzhou, L., Guenther, R.: Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+. Empirical Software Engineering Journal 13(1), 63–96 (2008)
[13] Liu, H., Yu, L.: Toward Integrating Feature Selection Algorithms for Classification and Clustering. IEEE Transactions on Knowledge and Data Engineering 17(4), 491–502 (2005)
[14] Chiu, N.H., Huang, S.J.: The adjusted analogy-based software effort estimation based on similarity distances. Journal of Systems and Software 25, 628–640 (2007)