Proposing An Effective Artificial Neural Network Architecture to Improve the Precision of Software Cost Estimation Model

Iman Attarzadeh
Department of Computer Engineering
Islamic Azad University, Dezful Branch, Dezful, Iran
attarzadeh@iaud.ac.ir

Siew Hock Ow
Department of Software Engineering
University of Malaya, 50603 Kuala Lumpur, Malaysia
show@um.edu.my

Received 20 September 2010
Revised 5 March 2013
Accepted 18 January 2014

Software companies have to manage different software projects based on different time, cost, and manpower requirement, which is a very complex task in software project management. Accurate software estimates at the early phase of software development is one of the crucial objectives and a great challenge in software project management, in the last decades. Since software development attributes are vague and uncertain at the early phase of development, software estimates tend to a certain degree of estimation error. A software development cost estimation model incorporates soft computing techniques provides a solution to fit the vagueness and uncertainty of software attributes. In this paper, an adaptive artificial neural network (ANN) architecture for Constructive Cost Model (COCOMO) is proposed in order to produce accurate software estimates. The ANN is utilized to determine the importance of calibration of the software attributes using past project data in order to produce accurate software estimates. Software project data from the COCOMO I and NASA'93 data sets were used in the evaluation of the proposed model. The result shows an improvement in estimation accuracy of 8.36% of the ANN-COCOMO II when compared with the original COCOMO II.

Keywords: Software engineering; software project management; software cost estimation models; COCOMO II; soft computing techniques; artificial neural networks.

1. Introduction

Inaccurate and unreliable software development cost and time estimates, is counted as one of the main problems and ongoing challenges in software engineering,
especially in software project management, in the last decades. The source of this
inaccuracy comes from the software attributes, which are vague and uncertain at
the early phase of software development process. In order to make such software
estimates, several algorithmic and non-algorithmic software development cost es-
timation models have been proposed and developed such as Constructive Cost
Model (COCOMO) [1], Expert Judgment [2], and Software Lifecycle Management
(SLIM) [3] models. The COCOMO is one of the most popular software develop-
ment cost estimation models in software companies because of its good char-
acteristics and capabilities such as the use of clear definition of software project
attributes. Improving the precision of software estimates can provide efficient
control to software development schedule and budget, which is an essential task to
software project managers during the development process. In software develop-
ment cost estimation process, the first step is to estimate or calculate the software
effort required for developing a software system, which is obtained from software
attributes at the early phase of development, followed by estimating software de-
development time, cost, and manpower requirement based on the software effort. The
software attributes are often vague, uncertain, and not calibrated at the early
phase of software development. Therefore, it is difficult to make accurate software
estimates based on such vague and not calibrated software attributes. Software
project attributes usually estimate or calculated based on human judgment, which
differ from one data analyst to another. Recently, software researchers have
attempted to explore the application of soft computing techniques such as artificial
neural networks, fuzzy logic, genetic algorithm, and analogy-based estimation, to
software development cost estimation models to improve the accuracy of software
estimates [4]. However, the accuracy of software estimates are still far from satis-
factory and no convenient model is currently available for the calibration of soft-
ware project attributes. Since, the vagueness and uncertainty of software attributes
cannot be avoided, a software development cost estimation model incorporates an
adaptive artificial neural network architecture is able to efficiently calibrate the
software attributes, using past software project data, in order to produce accurate
software time and cost estimates. This paper aims to examine the potential benefits
of applying artificial neural network to the software development cost estimation
model to reduce the impact of vague and uncertainty of software attributes on the
software estimates.

This research paper begins with a brief background of software development cost
estimation models, characteristics of the COCOMO II and artificial neural networks,
in Sec. 1. Related work to the software development cost estimation model and
artificial neural networks are discussed in Sec. 2. Problem statement of this research
is explained in Sec. 3. Development process of the proposed software development
cost estimation model incorporates artificial neural network is presented in Sec. 4.
Section 5 explains the ANN learning algorithm, which is used to train the proposed
model. The description of data sets, evaluation methods, and analysis of the results
are discussed in Sec. 6. Section 7 summarises this research and outlines further research in software development cost estimation.

1.1. Software development cost estimation models

Software project managers and developers, always, interested to have accurate estimates of software development schedule, budget, and manpower requirement at the early phase of software development process. These estimates can help them to make intelligent decisions on software development process and strategic planning. To date, software researchers have been proposed and developed several software development cost estimation models, which can be classified into the algorithmic and non-algorithmic models. Algorithmic models were established based on regression techniques and statistical analysis of past project data. Software Life Cycle Management (SLIM) [1] and Constructive Cost Model (COCOMO) [2] are widely-used algorithmic software development cost estimation models in software companies. Non-algorithmic models were established based on heuristic approaches and experts’ knowledge. Expert Judgment, Price-to-Win, and Top-Down models belong to this category. The COCOMO was proposed by Barry Boehm in 1981 [2] and it is one of the most cited, best known, widely-used software development cost estimation models. The COCOMO uses to estimate software project effort, followed by software development time, cost, and manpower estimates. The latest version of this model, COCOMO II, includes three sub-levels for software estimates as follows:

- Application Composition Level — uses for software development with modern GUI-builder tools.
- Early Design Level — getting rough estimates of software time and cost before determination of software architecture. It uses small sets of new software attributes, called effort multipliers, and it has different equations when compared with the earlier version. The software size estimation in this level is based on unadjusted function point or thousand source lines of code.
- Post-Architecture Level — it is the most detailed level and is used when the overall architecture for the software project has been designed. It involves the actual development process and maintenance of software.

This research focuses on the Post-Architecture Level of the COCOMO II, as it is the most detailed level and widely-used model in software companies when compared with the other sub-levels. The COCOMO II includes different software attributes: 17 Effort Multipliers (EMs), 5 Scale Factors (SFs), and 1 Software Size (SS), which are used in the Post Architecture Model of the COCOMO II to estimate software effort, followed by the estimation of software time, cost, and manpower based on estimated effort, in Person Month (PM), [2] as shown in Eq. (1). The estimated effort represents how much human activities need to accomplish the project tasks, which includes
The estimation of software development time, cost, and manpower requirement in the COCOMO II is based on the following steps:

- Step 1: Estimation or calculation of software project size based on source line of code (SLOC), Function Point (FP), or Object Point techniques.
- Step 2: Estimation or calculation of effort multipliers and scale factor based on the software attributes.
- Step 3: Applying the values obtained from Steps 1 and 2 to the Eq. (1) to estimate the software effort, followed by estimating software cost, time, and manpower requirement.

In the COCOMO II, the inaccuracy of software size, effort multipliers, and scale factors can impact the estimated effort, followed by software time, cost, and manpower estimates. The use of an adaptive artificial neural network to calibrate these software attributes based on past project data, can improve the accuracy of software attributes, which resulted in improving the accuracy of software estimates.

### 1.2. Artificial neural networks

Artificial Neural Networks (ANNs) are simplified mathematical techniques of human brain. They are collection of neurons, Process Elements (PEs), with internal connections 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 such learnt data. The main idea in the ANN is to produce intelligent systems capable of sophisticated computations. It is similar to the biological neurons in human brain structure. In fact, each neuron or network node, is a mathematical function with some input connections, mathematical formula(s), and output connections. Each ANN includes a unique architecture, layers, and nodes. Each node generates a non-linear function of its inputs [5]. Using ANN starts by proposing a network architecture then selecting a proper learning technique.
for network training, test, and validation based on a data set. The most widely used ANN architectures can be grouped into two categories:

- Feed-forward architecture — where no loops in the network path occur.
- Recurrent architecture — that have recursive loops.

Also, the combination of the ANN architectures can be an alternative to use variety of capabilities of ANN such as learning, precision, and adaptation. Among different ANN architectures the feed-forward Backpropagation is the most widely-used architecture [6]. Figure 1 shows the structure of a simple node in an ANN.

Each node produces the weighted sum of its $m$ inputs, $W_j$, where $j = 1, 2, \ldots, n$ and generates an output of 1 if this result is above the defined threshold $u$. Otherwise, an output of 0 is generated. The node activation function is shown in Eq. (2).

$$y = \theta \left( \sum_{j=1}^{n} W_j x_j - u \right)$$ (2)

In Eq. (2), $\theta$ is the unit step function at 0 and $W_j$ is the synapse weight associated with the $j$th input. The $u$ parameter is considered as another weight, i.e. $w_0 = -u$ attached to the node with a constant input of $x_0 = 1$. Positive weights model excitatory synapses, while negative weights model inhibitory ones. The activation function in Fig. 1 is known as a step function. However, there are several functions that can be applied such as Sigmoid, Gaussian, and Linear functions [5, 6]. The simple ANN with nodes, connections, layers (input, hidden, and output) is shown in Fig. 2.

Using the capabilities of ANN such as learning from past project data, precision, and adaption in the software development cost estimation model can improve the accuracy of software project estimates.

Understanding and calculation of algorithmic techniques based on historical data are different due to inherent complex relationship between the related attributes. Attributes and relationships used to estimate software development effort could

Fig. 1. Structure of a simple node in an ANN.
change overtime and differ for software development environment and hence may create problems to software managers in committing resources and controlling costs. The inherent uncertainty in software attribute measurement has significant impact on estimation accuracy because these attributes are measured based on human judgment and are often vague and imprecise. In fact, software estimates involves identification and quantification of several predictive attributes that have significant impact on effort prediction. Thus, measuring these attributes should be performed in a consistent and robust way in order to avoid imprecision and vagueness. Since, the vagueness and uncertainty of software attributes cannot be avoided, a software development cost estimation model incorporates an adaptive artificial neural network architecture is able to efficiently calibrate the software attributes, using past software project data, in order to produce accurate software time and cost estimates. This paper aims to examine the potential benefits of applying artificial neural network to the software development cost estimation model to reduce the impact of vague and uncertainty of software attributes on the software estimates.

2. Related Work

In the last few decades, many software development cost estimation models have been proposed to improve the accuracy of software estimates. Software researchers have also attempted to improve existing software development cost estimation models to overcome the inaccuracy of software estimates [4]. These include the use of soft computing techniques such as case-based reasoning, fuzzy logic, and artificial neural network [5, 6]. Artificial neural networks have the capability to learn from past project data and make calibration, which can lead to more acceptable and accurate software project estimates. Witting and Finnie [7] attempted to capture the significant attributes of the software development environment to enable improved accuracy in forecasting of development effort using backpropagation artificial neural networks. The data for the study was gathered from commercial 4GL software development projects. They claimed that the neural network model predictions were

![Simple artificial neural network with layers and their connections.](image-url)

**Fig. 2.** Simple artificial neural network with layers and their connections.
reasonably accurate in comparison with other published results, indicating the potential of the use of this approach. In another research, Karunanithi [8] suggested the use of artificial neural network techniques such as Feed-forward and Jordon-network with expert knowledge to estimate software flexibility and reliability. The study shows, using the failure history, the neural-network model automatically develops its own internal model of the failure process and predicts future failures. Because it adjusts model complexity to match the complexity of the failure history, it can be more accurate than some commonly used analytic models. The results with actual testing and debugging data which suggest that neural-network models are better at endpoint predictions than analytic models. Srinivasan [9] applied two methods of machine learning to build estimators of software development effort from historical data. The experiments indicated that those techniques are competitive with traditional estimators on the used dataset, but also illustrated that those methods were sensitive to the data on which they are trained. Also, the results showed that applying ANN on software development cost estimation can be a feasible approach and it showed better results than the regression trees approach. Samson [10] used the Albus multiplayer perceptron technique to estimate software development time and cost. He claimed that the use of soft computing techniques can overcome the inaccuracy of software attributes, thus, improving the accuracy of software estimates. Tadion [11] proposed an artificial neural network model with Backpropagation learning algorithm for software development cost estimation. However, there was no report on the model validation and evaluation.

Witting and Finnie [17] applied a simple ANN model to the software cost estimation models. They used the Australian Software Metrics Association (ASMA) data set for network training, and were able to estimate software development effort to within 25% of the actual effort in more than 75% of the projects in the test data set, and with a MRE of less than 0.25. However, they did not compare the results with the results obtained from other software cost estimation models. Their model produced more accurate cost estimates when compared with the use of the original models.

Shukla [18] proposed a genetic algorithm for software cost estimation model. He used the information of COCOMO I and its data set; and 15 sets of project data from the Kemerer data set. The standard deviation (μ) of the experimental study indicated that the genetic algorithm produced μ = 5.5 when compared with Backpropagation-trained Neural Network (BPNN) with μ = 29.7, and Quickpropagation-trained Neural Network (QPNN) with μ = 24.3. These results showed that the accuracy of estimation can be improved using a genetic algorithm because of its quick network propagation and training capabilities. However, a large data set is needed to train such an ANN.

Mair et al. [19] studied the use of ANN approaches and their estimation capability on software development cost estimation models. They evaluated the performance of a few selected models, which included the regression, ANN, and rule-induction cost estimation models. The experimental results showed that the estimation error with
the mean absolute percentage error (MAPE) ranged from 38% to 100% for the regression model; 21% to 66% for the ANN model; and 86% to 140% for the rule-induction model. They found that ANN produced more accurate estimation when compared with other machine-learning approaches. However, they reported difficulties with the network configuration, analysis, and interpretation of the ANN-based model (called ANN black box).

Heiat [20] compared the estimation performance of the Radial Basis Function (RBF) neural networks using regression analysis. He indicated that when a combined 3GL and 4GL data set was used, the artificial neural network contributed to improve performance over conventional regression analysis in terms of mean absolute percentage error (MAPE). The MAPE results for RBF neural network using FP was 47.6%, and for RBF neural network using SLOC was 31.96%, which was better than using the regression model — 70.86% for FP and 40.37% for SLOC.

Kalichanin and Lopez [21] examined the effort estimation for 209 small-scale software projects using two techniques: statistical regression and neural network. They established two sample groups for the software projects. The sample groups had 132 projects and 77 projects, respectively, and were used for verifying and validating the two techniques. The software projects were undertaken by 40 and 24 programmers, respectively. They estimated the software effort for each project using Artificial Neural Network and Multiple Linear Regression (MLR). The study showed that project using ANN produced better results — MMRE = 0.22, and MLR MMER = 0.24. Also, they suggested that a feed-forward artificial neural network could be a good option for small-scale software project estimations. They did not make any comparison with the popular software cost estimation models such as COCOMO, and therefore, the performance and usability of their model is unknown.

Many researchers have reported on their success in applying artificial neural network approach to develop an accurate software development cost estimation model [12, 13]. This motivated us to explore and examine the application of an adaptive artificial neural network model in software development cost estimation. It is generally known that software attributes have properties of vagueness and uncertainty [14, 15]. A software development cost estimation model based on artificial neural network is able to handle the vagueness and uncertainty of software attributes. Hence, this research is aimed at determining the effectiveness of applying a new adaptive artificial neural network architecture on the software development cost estimation model (COCOMO) to overcome the shortcomings of existing models, and to improve estimation accuracy.

3. Problem Statement

Modeling, understanding, calculation, and analysis of software development cost estimation models based on past software project data are important tasks in
software project management because of the inherent complex relationships between the different software attributes, and weak control on the categorical data, and reasoning-based capabilities. Besides, the different software attributes and relationships used in software estimates can change over time, and the software development environment is also different. The source of the vagueness and uncertainty of software attributes are due to software project attributes. Thus, inaccuracy of the software attributes can affect the accuracy of software estimates. Artificial neural network technique has the capability to learn from previous data, able to adapt, and to produce more precise results. In order to address and overcome this inaccuracy in software estimates, a software development cost estimation model that incorporates an adaptive artificial neural network model is considered.

4. Proposed Artificial Neural Network Architecture for the COCOMO II

In this paper, an adaptive ANN architecture is proposed to accommodate the COCOMO II Post-architecture model. The COCOMO II includes 17 effort multipliers (EMs), 5 scale factors (SF), and 1 software size (SS). Therefore, the proposed ANN-COCOMO II has 23 inputs and 1 system output, estimated effort in PM (person-months), which is used for software development time, cost, and manpower estimation. Furthermore, to control and decrease the effect of error of the three input data categories (effort multipliers, scale factors, and software size), three bias factors were defined corresponding to each input data category. The summary of input variables of the ANN-COCOMO II is presented in Table 1.

Based on the list of input variables, 26 input nodes were defined in the input layer of the ANN-COCOMO II corresponding to the EMs, SFs, software size, and the three bias factors. To apply the proposed ANN architecture to the COCOMO II Post-Architecture model, data pre-processing using the Sigmoid Activation function in the input layer is necessary to calibrate the network input variables. The proposed architectures for the ANN and the ANN-COCOMO II are shown in Figs. 3 and 4, respectively.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Number of inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Size</td>
<td>1</td>
</tr>
<tr>
<td>Bias factor for software size</td>
<td>1</td>
</tr>
<tr>
<td>Effort Multipliers (EMs)</td>
<td>17</td>
</tr>
<tr>
<td>Bias factor for EMs</td>
<td>1</td>
</tr>
<tr>
<td>Scale Factors (SFs)</td>
<td>1</td>
</tr>
<tr>
<td>Bias factor for SFs</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 1. List of input variables of the ANN-COCOMO II.
The proposed ANN architecture is established based on feed-forward Back-propagation architecture which includes:

1. **Input layer** — for the EMs, SFs, Software Size, and Bias factors
   - where: \( x(t) \): Input data (EMs, SFs, Software Size, ...), \( y(t) \): input from other layers

2. **Hidden layer** — initialising network weights and connections
   - where: \( w \): Weight, \( b \): Bias

3. **Output layer** — effort estimation
   - where: \( y(t) \): output

---

**Fig. 3.** Proposed ANN architecture.

---

**Fig. 4.** The proposed artificial neural network architecture for the COCOMO II (ANN-COCOMO II).
In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

One of the novelties of the proposed ANN-COCOMO II is to use separate Bias factors for each category of network input data, instead of using only one bias factor for all the network input data. This idea controls the impact of inaccuracy of each category of input data on the output. Use of only one bias factor for all network input data cannot control and overcome the error produced by each category of input data.

Based on Table 1 and Fig. 3, the proposed ANN-COCOMO II architecture includes 26 nodes in the input layer, 20 nodes in the hidden layer, and one node in the output layer.

In the ANN-COCOMO II, the effort multiplier values, $EM_i$, were pre-processed to $\log(EM_i)$. The Sigmoid activation function in the hidden layer was defined by $f(x) = \frac{1}{1+e^{-x}}$. The corresponding weights to input nodes connected to the hidden layer were defined by $P_i$ for Bias1 and each input $\log(EM_i)$ for $1 \leq i \leq 17$. Besides, the corresponding weights to each scale factor, $SF_j$, from input nodes to the hidden layer are $q_j + \log(size)$ for $1 \leq j \leq 5$ and it was defined by Bias2. The parameters “W” and “b” in Fig. 5 show the corresponding weights to the arcs from the hidden layer nodes to the output layer nodes, and they are also relevant to the values of the hidden layer nodes. The network output nodes have the specific identity function.

Another contribution when compared with other related work, is the addition of Log(Input Size) to the weight $Q_j$ of the scale factors in the network input, which adjusts the weights $Q_j$. Customization of the COCOMO II formula was 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 Backpropagation was used as the learning algorithm in the ANN-COCOMO II. The data sets used for the network training are discussed in the following section. However, if no appropriate data set is available for network training, the weights and biases parameters in the ANN-COCOMO II are initialized randomly. The output of ANN-COCOMO II, estimated effort, was calculated using COCOMO II formula, in Eq. (1), by setting the initial values of Bias1 to Log(A) and Bias2 to 1.01. The ANN-COCOMO II weights were initialized as $p_i = 1$ for $1 \leq i \leq 17$ and $q_j = 1$ for $1 \leq j \leq 5$. To propagate the values of the input parameters into the network, the weights of nodes in the hidden layer were calculated using the following formulas:

$$f\left(p_0 \text{Bias1} + \sum_{i=1}^{17} p_i \cdot \log(EM_i)\right) = \text{sigmoid}\left(\text{Bias1} + \sum_{i=1}^{17} p_i \cdot \log(EM_i)\right)$$

$$= \frac{A \cdot \prod_{i=1}^{17} EM_i}{1 + A \cdot \prod_{i=1}^{17} EM_i} = \alpha$$

$$f((q_0 + \log(size)) \cdot \text{Bias2} + \sum_{j=1}^{5} (q_j + \log(size)) \cdot (SF_j))$$
\[
\text{sigmoid}(\log(\text{size}) \ast \left(\text{Bias2} + \sum_{j=1}^{5} \text{SF}_j\right)) = \frac{\text{Size}^{1.01+\sum_{j=1}^{5} \text{SF}_j}}{1 + \text{Size}^{1.01+\sum_{j=1}^{5} \text{SF}_j}} = \beta
\]

Then initialization of weights “W” and “b” are based on:

\[
W = \frac{\beta}{2(1-\alpha)(1-\beta)} \quad \text{and} \quad b = \frac{\alpha}{2(1-\alpha)(1-\beta)}
\]

The ANN-COCOMO II output was calculated based on:

\[
\text{PM} = W \ast \alpha + b \ast \beta = \frac{\alpha \beta}{(1-\alpha)(1-\beta)} = A \cdot \text{Size}^{1.01+\sum_{j=1}^{5} \text{SF}_j} \cdot \prod_{i=1}^{17} \text{EM}_i
\]

5. Training Algorithm

The training of the ANN-COCOMO II was done through iteration of forward and backward techniques until the terminating conditions were satisfied. For example, the value of changes in the weights are less than or equal to a basic threshold or a basic number of iterations have been carried out. The training algorithm includes following steps:

- Selecting a training sample and propagating the input parameters across the ANN-COCOMO II to calculate the network output.
- Detecting error in the network output, and determining the value of error gradient in the other network layers.
- Determining the value of changes for the ANN-COCOMO II weights and updating the network weights.
- Repeating the steps until the ANN error is sufficiently small, less than or equal to a specific threshold after an epoch is complete.

The ANN-COCOMO II training process is shown in Fig. 5.

![Fig. 5. The ANN training process.](image-url)
The ANN-COCOMO II was calibrated based on the comparison of differences between the output and the target values, until the network output value matches the target value. The output value represents the estimated effort, which was produced by the ANN-COCOMO II and the target value represents the actual effort corresponding to each set of project data, which was retrieved from the data sets (the data sets characteristics are explained in the following section). The differences between estimated effort (output) and actual effort (target) were used for the adjustment of the weights in the network. The Backpropagation algorithm in the ANN-COCOMO II includes four steps:

1. **Step 1** — compute how fast the error changes as the activity of an output unit is changed. This error derivative is the difference between the actual and the desired activity.
2. **Step 2** — compute how fast the error changes as the total input received by an output unit is changed. This quantity is the answer from Step 1 multiplied by the rate at which the output of a unit changes as its total input is changed.
3. **Step 3** — compute how fast the error changes, as a weight on the connection into an output unit is changed. This quantity is the answer from Step 2 multiplied by the activity level of the unit from which the connection emanates.
4. **Step 4** — compute how fast the error changes, as the activity of a unit in the previous layer is changed. This crucial step allows Backpropagation to be applied to multilayer networks. When the activity of a unit in the previous layer changes, it affects the activities of all the output units to which it is connected. Thus, to compute the overall effect on the error, all these separate effects are added on to the output units. But each effect is simple to calculate. It is the answer in Step 2 multiplied by the weight on the connection to that output unit.

By using Steps 2 and 4, the output of one layer can be transferred as the input values for the previous layer(s). This procedure can be repeated to produce the inputs for many as desired of the previous layers. Once the inputs of a layer are computed, Steps 2 and 3 are used to compute the output on its incoming connections.

Also, the network training conditions were applied as follows:

- Using Data set #1 and Data set #2: 156 sets of project data (the data sets characteristics are explained in the following section)
- 70% of total projects, 110 projects, were used for model training
- 15% of total projects, 23 projects, were used for model validation
- 15% of total projects, 23 projects, were used for model testing.

In the network training of the ANN-COCOMO II, 156 sets of project data were used. The iteration value was set to 1000-epoch, and the threshold value for network training error was set to 0.001. After network training, the weights of nodes in the ANN-COCOMO II were initialized and calibrated to the proper values.
6. Results and Discussion

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

- Data set #1 — the original COCOMO I data set, which includes 63 sets of project data [16].
- Data set #2 — NASA’93 data set, which includes 93 sets of project data in small, medium, and large-scale project size [16].

6.1. Data set description

These two data sets include a number of past projects data, gathered from software companies, the industry, and NASA software projects [16]. Boehm [1] was the first researcher to establish and validate an appropriate algorithmic 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 [16] and were used in the network training and evaluation of the ANN-COCOMO II. Table 2 shows samples of the two data sets (Data set #1 and Data set #2).

6.2. Evaluation methods

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:

\[
MRE_i = \frac{|Actual\ Effort_i - Estimated\ Effort_i|}{Actual\ Effort_i} \quad i: \text{project number} \quad (7)
\]

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

\[
MMRE = \frac{1}{N} \sum_{i=1}^{N} MRE_i \quad (8)
\]

<table>
<thead>
<tr>
<th>No.</th>
<th>Mode</th>
<th>Size</th>
<th>Actual effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1200</td>
<td>51.2500</td>
<td>246.5900</td>
</tr>
<tr>
<td>2</td>
<td>1.2100</td>
<td>12.5500</td>
<td>58.2800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>1.0200</td>
<td>56.5300</td>
<td>354.7300</td>
</tr>
<tr>
<td>155</td>
<td>1.0150</td>
<td>16.0400</td>
<td>67.1400</td>
</tr>
<tr>
<td>156</td>
<td>1.1000</td>
<td>54.1700</td>
<td>262.3800</td>
</tr>
</tbody>
</table>
A complementary evaluation condition is prediction at level \( L \), \( \text{PRED} (L) = k/N \), 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 \( \text{PRED} (L) \) method shows the probability of a project having a relative error of less than or equal to level \( L \). For example, \( \text{PRED} (25\%) \) 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 \( \text{PRED} (25\%) \) 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 3.

Table 3 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 \( \text{PRED} (25\%) \) for the two data sets (156 sets of project data). The results show that the mean of MMRE and \( \text{PRED} (25\%) \) 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.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Model</th>
<th>MRE</th>
<th>\text{PRED} (25%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set #1</td>
<td>COCOMO II</td>
<td>0.542561832</td>
<td>45%</td>
</tr>
<tr>
<td></td>
<td>ANN-COCOMO II</td>
<td>0.487257017</td>
<td>52%</td>
</tr>
<tr>
<td>Data set #2</td>
<td>COCOMO II</td>
<td>0.462579313</td>
<td>30%</td>
</tr>
<tr>
<td></td>
<td>ANN-COCOMO II</td>
<td>0.428617366</td>
<td>39%</td>
</tr>
<tr>
<td>Mean</td>
<td>COCOMO II</td>
<td>0.502570573</td>
<td>37.5%</td>
</tr>
<tr>
<td></td>
<td>ANN-COCOMO II</td>
<td>0.457937192</td>
<td>45.5%</td>
</tr>
</tbody>
</table>

Table 4. Comparison of the improvement in estimation accuracy of the ANN-COCOMO II and the original COCOMO II.

<table>
<thead>
<tr>
<th>Model</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN-COCOMO II vs. COCOMO II</td>
<td>MMRE 0.457937192</td>
</tr>
<tr>
<td>COCOMO II vs. ANN-COCOMO II</td>
<td>MMRE 0.502570573</td>
</tr>
<tr>
<td>Improvement %</td>
<td>8.36%</td>
</tr>
</tbody>
</table>
Table 5. Comparison of the proposed model with other software development cost estimation models.

<table>
<thead>
<tr>
<th>Researcher</th>
<th>Model name</th>
<th>Soft computing technique(s) applied</th>
<th>Data set(s) used</th>
<th>Research findings/remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Witting and Finnie</td>
<td>Artificial Neural Network Model For Software Cost Estimation</td>
<td>ANN</td>
<td>COCOMO I</td>
<td>Estimated software effort within 25% of the actual effort in more than 75% of the projects.</td>
</tr>
<tr>
<td>Shukla</td>
<td>Genetic Algorithm for COCOMO I</td>
<td>Genetic Algorithm</td>
<td>COCOMO I</td>
<td>GA model: Standard deviation = 5.5</td>
</tr>
<tr>
<td>Mair et al.</td>
<td>Artificial Neural Network Model For Software Cost Estimation</td>
<td>ANN</td>
<td>COCOMO I</td>
<td>ANN: MAPE = 21% to 66%</td>
</tr>
<tr>
<td>Kalichanin and Lopez</td>
<td>ANN-Statistical Regression (i.e. applied on the COCOMO II)</td>
<td>Multiple Linear Regression (MLR) and ANN</td>
<td>COCOMO I</td>
<td>ANN: MMRE = 0.22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MLR: MMER = 0.24</td>
</tr>
<tr>
<td>The Proposed Model</td>
<td>ANN-COCOMO II</td>
<td>ANN-Backpropagation multilayer networks</td>
<td>COCOMO I</td>
<td>MMRE: 0.502570573 PRED: 37.5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>COCOMO II</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NASA'93</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ANN-COCOMO II</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I. Attarzadeh & S. H. Ow
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 \[ \text{PRED} (25\%) = 45.5\% \] 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 4 shows the comparison of the improvement in estimation accuracy between the ANN-COCOMO II and the original COCOMO II.

Table 4 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.

The results indicate that the ANN-COCOMO II performed better than the original COCOMO II, and has great potential of improving the estimation accuracy of software development effort, schedule, cost, and manpower requirement. Compared to the COCOMO II, ANN-COCOMO II also shows that its estimation accuracy is more stable and accurate, based on the used data sets. The obtained results also support the success in applying artificial neural network approach to develop an accurate software development cost estimation model as it mentioned above in [7, 8, et al.].

The proposed model is compared with other software development cost estimation models in which ANN is also applied. The results of the comparison are shown in Table 5.

### 7. 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 development cost estimation.
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.

8. Future Work

This research is one of the few researches that focus on improving the accuracy of
software development effort estimation model by applying artificial neural network
in COCOMO II. The soft computing techniques were primarily used to handle the
vague and imprecise software attributes such as software size, software complexity,
and system analyst’s capability, which are important issues to consider when making
any estimation in software development. The findings from this research should
provide the motivation for further research on the advantages of using soft com-
puting techniques in software development cost estimation models. In this context,
future studies should consider the following:

- Design a model and a system that can give an accurate estimate of the software
  size for each software project. This would certainly address a problematic and
  challenging issue in software development estimation.
- Make improvement to the existing ANN training algorithms to produce more
  accurate software estimates.
- Study the use of soft computing techniques such artificial neural network and
  genetic algorithm in other software development cost estimation models such as
  the SLIM model, and the Expert Judgment model.

Acknowledgments

This research was funded by the University of Malaya under the University of
Malaya Research Grant (UMRG), Account Number: RG034/10ICT.

References

2. B. Boehm, C. Abts and S. Chulani, Software development cost estimation approaches —
   A survey, University of Southern California Center for Software Engineering, Technical
3. L. H. Putnam, A general empirical solution to the macro software sizing and estimating
4. K. Srinivasan and D. Fisher, Machine learning approaches to estimating software de-


