Improving the Accuracy of COCOMO II Effort Estimation Based on Neural Network with Hyperbolic Tangent Activation Function

Sarah Abdulkarem Alshalif, Noraini Ibrahim and Waddah Waheeb
Software Engineering Research Group, Faculty of Computer Science and Software Engineering, Universiti Tun Hussein Onn Malaysia, 86400 Parit Raja, Batu Pahat, Johor, Malaysia.
gi140034@siswa.uthm.edu.my

Abstract— **Constructive Cost Model II (COCOMO II) is one of the best-known software cost estimation model. The estimation of the effort in COCOMO II depends on several attributes that categorized by software size (SS), scale factors (SFs) and effort multipliers (EMs). However, provide accurate estimation is still unsatisfactory in software management. Neural Network (NN) is one of several approaches developed to improve the accuracy of COCOMO II. From the literature, they found that the learning using sigmoid function has always mismatched and ill behaved. Thus, this research proposes Hyperbolic Tangent activation function (Tanh) to use in the hidden layer of the NN. Two different architectures of NN with COCOMO (the basic COCOMO-NN and the modified COCOMO-NN) are used. Back-propagation learning algorithm is applied to adjust the COCOMO II effort estimation parameters. NASA93 dataset is used in the experiments. Magnitude of Relative Error (MRE) and Mean Magnitude of Relative Error (MMRE) are used as evaluation criteria. This research attempts to compare the performance of Tanh activation function with several activation functions, namely Uni-polar sigmoid, Bi-polar sigmoid, Gaussian and Softsign activation functions. The experiment results indicate that the Tanh with the modified COCOMO-NN architecture produce better result comparing to other activation functions.**

Index Terms—**Activation Functions; Constructive Cost Model II; Effort Estimation; Neural Network.**

I. INTRODUCTION

Software cost estimation processes is the most crucial and challenge task in the management of software project and in the software engineering area [1]. Software project manager and developers are interested to accurately estimate the effort and cost at the early stage of software development. Many models of software cost estimation have been developed and improved which can be categorized to algorithmic and non-algorithmic models. Algorithmic models, also known as conventional method, provide mathematical and experimental equations to compute software cost based on statistical analysis of the past projects data and use a mathematical formula to estimate project cost based on the project size and other factors such as number of software engineers. Constructive cost model (COCOMO) [2] and Software Lifecycle Management (SLIM) [3] are some examples of the algorithms models. Non-algorithmic models or non-parametric models established based on heuristic approaches and experts' knowledge, where Expert Judgement and Top-Down models belong to this category [4]. The most popular algorithmic cost estimation model is Boehm's Constructive Cost Model (COCOMO I and II) [3]. COCOMO II is used to estimate project effort, followed by software development time, cost and manpower estimation. COCOMO II model consists of three sublevels or models which are Application-Composition model, Early Design model and Post-Architecture model. Application-Composition model is appropriate for the applications that developed rapidly using interoperable components that based on GUI builders and is based on new object point's estimation. While Early Design model is utilized in the early phases of a software project and can be used in Application Generator, System Integration, or Infrastructure Development Sector. It size can be measures by Unadjusted Function Points (UFP). Post-Architecture model is the most detailed model comparing to others and used after the overall architecture of the projects has been designed. Either function points or Lines of Code (LOC) can be used to determine the size of software project [4].

Due to the restriction of the algorithmic models, the use of the non-algorithmic models has been discovered based on soft computing techniques for cost estimation involved fuzzy logic (FL), evolutionary computation (EC), and neural networks (NN) [5]. The architecture of the neural network and its parameters involved the number of layers, the number of nodes, transfer function, weights, biases and the learning rate. These factors influence the network performance. This research investigates the role of different activation functions to improve the accuracy of the effort estimation.

Feed forward multilayer perceptron with backpropagation learning algorithm is generally used for estimating software effort [6]. This paper considered the features of COCOMO II based on NN using Hyperbolic Tangent function and compares the result with other activation functions using NASA93 projects dataset. This research also explores the expectation from [7] to proves that Tanh improve the COCOMO II.

This article is ordered as following. For section I, is the introduction. In section II, it shows the literature review. Section III presents the research methodology and discusses the two-different architecture of COCOMO-NN that used in this research. In Section IV, the methodology is evaluated as well as the results are presented and discussed. Section V presents the conclusion and summarizes the future research directions.

II. Literature Review

For decades, many models for cost estimation have been developed and used in software management and many techniques have been used to improve these models to accurately estimate the effort. Neural network is the most used technique due to its learning capability and its ability to model complex problems [8].

Alshalif, Ibrahim and Herawan [7] proposed Tanh to improve COCOMO II by providing initial results. The research is extended by this paper by comparing the impact between Tanh and other activation functions to two different architectures of NN with COCOMO.

Reddy and Raju [9] used neural network to improve COCOMO model with 17 Effort Multiplier and 5 Scale Factors. The proposed method used multi-layer feed forward neural network and used the modified COCOMO-NN. The network was trained with back propagation learning algorithm. Linear activation function was used in all the layers of the network. They compared the estimated effort with the actual effort. COCOMO81 dataset was used in the experiments and the result indicates that the estimation accuracy has been improved by using this model.

Kaushik et al. [10] used intermediate COCOMO with 15 Effort multipliers. They integrated COCOMO with Neural Network using basic COCOMO-NN. Perceptron learning algorithm used for training. For the input layer, they used identity activation function and for the hidden and output layer the threshold activation function was used. As well as, the COCOMO81 datasets with 63 projects was used in the experiments. The Mean Magnitude of Relative Error (MMRE) for the proposed model was less than the MMRE for the COCOMO model.

Kaushik et al. [11] used the approach of Reddy and Raju [9] with different activation functions. They used sigmoid activation function in the hidden and output layers and used modified COCOMO-NN. Three datasets were used, namely COCOMO81 with 63 projects, COCOMO NASA 2 with 93 projects and COCOMO II SDR with 12 projects. Comparing with Reddy model [9], this model performs better improvement in term of Mean Magnitude of Relative Error (MMRE).

In [12], the authors used multi-layer feed forward neural network with modified COCOMO-NN to accommodate COCOMO. The network was trained with backpropagation learning algorithm and used identity function for all the layers. The attributes used with this model were 15 EM and 5 SF and 2 biases. COCOMO81 with 63 historical projects were used in this work. This model improves the accuracy comparing to the COCOMO model.

Mukherjee and Malu [13] used multi-layer feed forward neural networks with basic COCOMO-NN. They used back-propagation learning algorithm for the training. In this model, they used sigmoid activation function in the hidden layer and linear one in the output layer. This model shows better result comparing to COCOMO model and comparing to the model developed by Kaushik et al. [10].

Kumar and Bhatia [14] used multi-layer feed forward neural networks with basic COCOMO-NN. The inputs to the network were the 15 EM and software size in KLOC. Back-propagation learning algorithm with tangent function in the hidden layer and linear function in the output layer were chosen. This model shows improvement in the accuracy using the NASA I that consists of 60 projects.

Sarno et al. [15] compared the two-different architecture of COCOMO-NN consist of modified and basic COCOMO-NN. In this comparison, they used back-propagation learning algorithm with sigmoid function at the hidden layer and linear at the output layer. They used 17 inputs for the basic COCOMO-NN and 23 for the modified COCOMO-NN with COCOMO81 and NASA93 datasets, respectively. This model shows that the modified COCOMO-NN performs better than the basic one.

In [16], the authors used fuzzy logic with the modified multi-layer feed forward neural networks, and back-propagation learning algorithm with sigmoid function at the hidden layer and linear at the output layer. Nasa93 dataset was used for training and testing.

Rijwani and Jain [17] used multi-layer feed forward neural networks with basic COCOMO-NN to achieve more accuracy in software effort estimation. The proposed NN used 23 inputs and a hidden layer with tangent function. COCOMO81 dataset was used in the experiments and in this research, it noted that the model with neural network improve the estimation accuracy significantly.

III. Research Methodology

Through a detailed literature review, we observed that there are several factors that affect the network performance due to the architecture of the network and their parameter settings. These factors consist of the number of layers, how many nodes in each layer, the transfer function in each node, learning algorithm parameters and the weights which define the connectivity between nodes. From these factors, there is no standard rule to define the ideal parameter settings but even small parameter changes can cause large differences in the results of almost all networks [11]. The aim of this work is to compare the Tanh activation function with other activation functions in the two different neural network architectures. Feed-forward neural network with backpropagation learning algorithm will be used for both architectures. Four main phases to perform the back-propagation training algorithm. First, the initialization of the weights and biases. Next, feed forward. Then, back propagation of errors. Finally, updating the weights and biases.

A. Activation functions

The activation functions also called as transfer functions due to their task of transforming the neuron activation level into output signal. This research is demonstrated the use of different activation functions in neural network to estimate the effort. Moreover, COCOMO II associated with Neural Network (NN) helps to accurately estimate the software effort. Several activation functions can be used with neural network such as Tanh, Uni-polar sigmoid, Bi-polar sigmoid, Gaussian and softsign activation functions. The formulas of these functions [18] are as follows:

Tanh: $$f(x)=\frac{1-e^{-2x}}{1+e^{-2x}} \quad (1)$$

Uni-polar sigmoid: $$f(x)=\frac{1}{1+e^{-x}} \quad (2)$$

Bi-polar sigmoid: $$f(x)=\frac{1-e^{-x}}{1+e^{-x}} \quad (3)$$

Gaussian: $$f(x) = e^{-x^2} \quad (4)$$

Softsign: $$f(x) = \frac{x}{1+|x|} \quad (5)$$

B. Basic COCOMO neural network architecture

This architecture of the network is based on the used dataset which is NASA93 dataset. This dataset contains 93 projects and 24 attributes represented as 17 Effort Multipliers, 5 Scale Factors, one Software Size and the Actual Effort. In this architecture one hidden layer between input and output layer will be used. There is no specific value for number of nodes in the hidden layer. Therefore, four nodes will be used in the hidden layer. The steps needed to implement this model are:

Step 1 : Initialize the inputs as $x_i = \ln(input_i)$.

Step 2 : Initialize the weights w_i , wh_i and biases b_i .

Step 3 : Set the learning rate α(0 < α ≤ 1).

Step 4 : Test stopping condition for false. Repeat the steps 5-13.

Step 5 : Perform steps 6-12 for each training pair

Step 6 : Compute the hidden unit H_i sums its weighted input signals to calculate net input given by:

$$H_n = b_n + \sum x_i * w_i \quad (6)$$

where: n=1 to 4
i=1 to 23

Step 7 : Activate the hidden layer by applying the activation function over H_i and send the output signal from the hidden unit to the input of output layer units.

Step 8 : Compute the output unit, E_{est} calculates the net input given by:

$$E_{est} = b2 + H_1 * wh_1 + \cdots + H_n * wh_n \quad (7)$$

where: n=1 to 4

Step 9 : Calculate the error correction term δ as:

$$\delta = E_{act} - E_{est} \quad (8)$$

where E_{est} is the actual effort from the dataset *ln(E$_{act}$)* and E_{est} is the estimated effort from step 8. The hidden error is calculated as:

$$\delta_{hn} = \delta * wh_n * \text{DevOAF} \quad (9)$$

where: n=1 to 4
DevOAF = the derivative of the activation function that will be used.

Step 10 : Update the weights between hidden and the output layer as:

$$wh(new) = wh(old) + \alpha * \delta * H_i \quad (10)$$

$$b2(new) = b2_i(old) + \alpha * \delta \quad (11)$$

Step 11 : Update the weights and bias between input and hidden layers as:

$$w_i(new) = wei_i(old) + \alpha * \delta_h * x_i \quad (12)$$

$$b_n(new) = b_n(old) + \alpha * \delta_n \quad (13)$$

Step 12 : Count the test data using new weights

Step 13 Check for the stopping condition. If the error between estimated and actual effort in the test data is smaller than a specific tolerance or the number of iteration overrides a specific number, stop: else continue.

Using this training process, iteration forward and backward proceed until the terminating condition is satisfied. The learning rate used this symbol α that will be used in the above formula and it is constant to determine the network learning speed. The bigger value for the learning rate, the faster it will learn. But, sometimes bigger value could make the learning process over fitting. Figure 1. Shows the architecture of Basic COCOMO-NN.

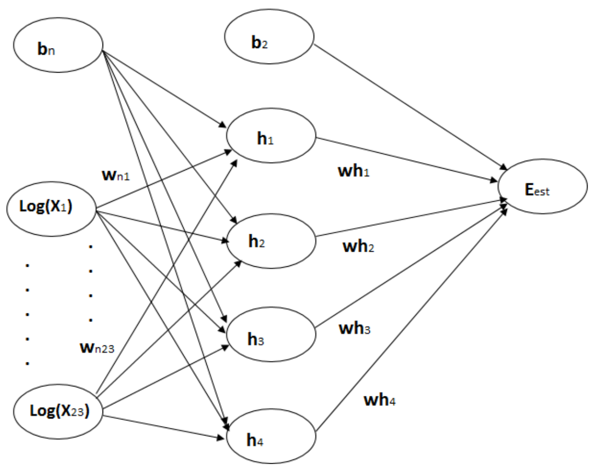

Figure 1: Basic COCOMO-NN Architecture

C. Modified COCOMO neural network architecture

The input in this architecture is from NASA93 dataset containing 24 inputs but the major difference as compared to the basic COCOMO-NN is that the separation of the inputs into two parts. Part one deals with the 17 Effort Multipliers which represent the upper section and the second part deals with the 5 Scale Factors that represent the lower section. Moreover, software size is not used as an input in the input layer, but it is applied as a constant in scale factors weight. This model also has several steps to use the architecture. They are:

Step 1 : Initialize the inputs as x_i and y_j.

Step 2 : Initialize the weights $wei_i = weh = wsh = 1$ and $wsi_j = 0$. Initialize the bias $b1$ and $b2$.

Step 3 : Set the learning rate α(0 < α ≤ 1).

Step 4 : Test stopping condition for false. Repeat the steps 5-13.

Step 5 : Perform steps 6-12 for each training pair.

Step 6 : Each hidden unit H_{EM} and H_{SF} sums its weighted input signals to calculate net input given by:

$$H_{EM} = b1 + \sum x_i * wei_i \quad (14)$$

for i= 1 to 17.

$$H_{SF} = b2 + \sum y_i * (wsi_i + \ln(size)) \quad (15)$$

for i= 1 to 5.

Step 7 : Activate the hidden layer by applying activation function over H_{EM} and H_{SF} , then send the output signal from the hidden unit to the input of output layer units.

Step 8 : Compute the output unit E_{est} , calculates the net input given by:

$$E_{est} = H_{EM} + weh + H_{SF} * wsh \tag{16}$$

Step 9 : Calculate the error correction term (δ) as:

$$\delta = E_{act} - E_{est} \tag{17}$$

Where E_{act} is the actual effort from the dataset *ln(E_{act})* and E_{est} is the estimated effort from step 8.

Step 10 : Update the weights between hidden and the output layer as:

$$weh(new) = weh(old) + \alpha * \delta * H_{EM} \tag{18}$$

$$wsh(new) = wsh(old) + \alpha * \delta * H_{SF} \tag{19}$$

The hidden error is calculated as:

$$\delta_{EM} = \delta * weh * DevOAF \tag{20}$$

$$\delta_{SF} = \delta * wsh * DevOAF \tag{21}$$

Where DevOAF is the derivative of the activation function that will be used.

Step 11 : Update the weights and bias between input and hidden layers as:

$$wei_i(new) = wei_i(old) + \alpha * \delta_{EM} * x_i \tag{22}$$

for i=1 to 17.

$$wsi_i(new) = wsi_i(old) + \alpha * \delta_{SF} * y_i \tag{23}$$

for i=1 to 5.

$$b1(new) = b1(old) + \alpha * \delta_{EM} \tag{24}$$

$$b2(new) = b2(old) + \alpha * \delta_{SF} \tag{25}$$

Step 12 : Count the test data using new weights.

Step 13 : Check for the stopping condition. If the error between estimated and actual effort in the test data is smaller than a specific tolerance or the number of iteration overrides a specific number, stop: else continue.

Using this training approach, iteration forward and backward conducted until the terminating condition is satisfied. Figure 2 shows the modified COCOMO-NN architecture.

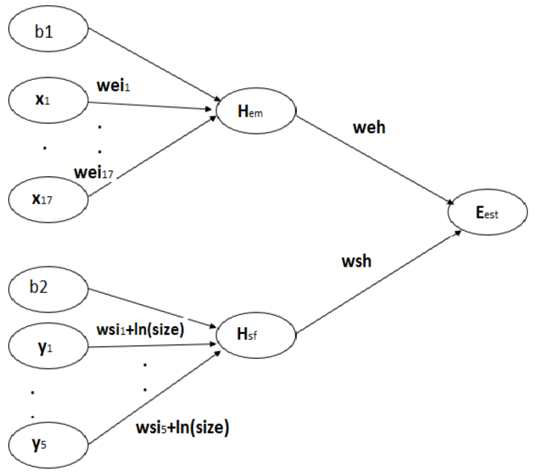

Figure 2: Modified COCOMO-NN Architecture

IV. EXPERIMENTAL RESULTS

Integration of the COCOMO II with soft computing technique can minimize and cope with the ambiguity and uncertainty of the software attributes. This paper aims to integrate the neural network that use hyperbolic tangent activation function in its hidden layer with the COCOMO II to treat these ambiguities and uncertainty. This section will show and compare the result of the proposed model using hyperbolic tangent function with other functions for different architectures of COCOMO-NN using NASA93 datasets and compare the results based on the Magnitude of Relative Error (MRE) and Mean Magnitude of Relative Error (MMRE). The equation of calculating MRE and MMRE are:

$$\text{MRE} = \frac{|\text{ActualEffort} - \text{EstimatedEffort}|}{\text{ActualEffort}} \text{ x } 100 \tag{26}$$

$$\text{MMRE} = \frac{1}{\text{N}} \sum_{\text{i}}^{\text{N}} \text{MRE}_i \tag{27}$$

From these equations, the smaller value of MRE and MMRE is the closer to actual effort. Table 1 presents the result and comparison using Basic COCOMO-NN on NASA93 dataset by comparing the performance of different activation functions, namely Uni-polar, Bi-polar sigmoid, Gaussian and softsign with Tanh. The result shows that MMRE for the Uni-polar sigmoid is 28.5116, Bi-polar sigmoid is 29.8051, Gaussian is 24.5334, softsign is 26.4988 and Tanh is 28.3494. It can be observed that the Gaussian function performs better estimation comparing to other activation functions in the Basic COCOMO-NN architecture.

Figure 3 shows the graphical view of MRE and MMRE it shows that Gaussian function has the lower value of MMRE with basic COCOMO-NN model which is 24.5334 and it indicates that the Gaussian function is suitable to use in the basic architecture of COCOMO-NN.

Table 1
MRE and MMRE for NASA93 dataset with the Basic COCOMO-NN.

Basic COCOMO-NN						
No	Project-Id	Uni-polar sigmoid	Bi-polar sigmoid	Gaussian	softsign	Tanh
1	75	17.7625	20.4278	9.5649	13.6148	17.4282
2	76	20.6300	23.2024	12.7183	16.6270	20.3074
3	77	25.8022	28.2070	18.4061	22.0601	25.5007
4	78	14.9948	11.2678	26.4576	20.7945	15.4622
5	79	12.5220	15.3572	3.8021	8.1100	12.1664
6	80	19.7499	22.3508	11.7505	15.7025	19.4238
7	81	27.0147	29.3802	19.7395	23.3337	26.7180
8	82	14.7901	17.5517	6.2963	10.4925	14.4437
9	83	17.7410	20.4070	9.5413	13.5923	17.4067
10	84	13.2443	16.0561	4.5964	8.8688	12.8917
11	85	36.9044	38.9494	30.6150	33.7222	36.6480
12	86	29.6715	31.9508	22.6610	26.1244	29.3856
13	87	28.9678	31.2699	21.8872	25.3853	28.6790
14	88	30.4366	32.6912	23.5024	26.9282	30.1538
15	89	18.7401	21.3737	10.6400	14.6417	18.4098
16	90	41.6338	43.5255	35.8158	38.6901	41.3966
17	91	14.7901	17.5517	6.2963	10.4925	14.4437
18	92	111.7053	104.8439	132.8082	122.3826	112.5657
19	93	44.6200	39.9329	59.0359	51.9139	45.2078
MMRE		28.5116	29.8051	24.5334	26.4988	28.3494

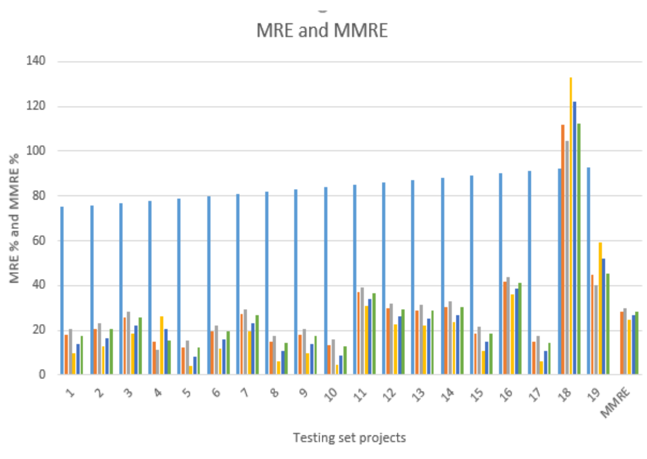

Figure 3: MRE and MMRE for Basic COCOMO-NN

For the Modified COCOMO-NN the results show in Table 2 the comparison is done on NASA93 dataset as well. Here, there is a decrement in the relative error using the Tanh function. The Mean Magnitude of Relative Error (MMRE) for the entire testing set is 18.2699, 11.5390, 29.3847, 29.7889 and 9.8948 for Uni-polar sigmoid, Bi-polar sigmoid, Gaussian, softsign and Tanh, respectively. This clearly shows that the COCOMO II model using neural network with Tanh function provide better cost estimation comparing to the estimation done using other functions in the modified COCOMO-NN architecture. As well as, Modified COCOMO-NN architecture provide better estimation as the Basic architecture when comparing table 1 with Table 2.

Figure 4 shows the graphical representation of MRE and MMRE values for the five activation functions. MRE values were plotted for each project in the testing set and MMRE shows that Tanh provides the lower values which is 9.8948 so it provides better estimation comparing to other activation functions.

Table 2
MRE and MMRE for NASA93 dataset with the modified COCOMO-NN.

Modified COCOMO-NN						
No.	Project-Id	Uni-polar sigmoid	Bi-polar sigmoid	Gaussian	softsign	Tanh
1	6	44.9670	32.8479	69.1336	37.4301	26.3220
2	16	24.9599	18.0946	42.1190	14.5541	13.0196
3	26	7.5987	7.2539	2.8914	17.065	5.0737
4	31	14.0153	5.8354	32.2389	7.4512	0.9552
5	43	14.5330	6.7451	29.3132	43.0227	8.1455
6	55	18.2050	3.8868	28.1683	42.1008	6.2391
7	61	5.3815	16.056	16.3778	32.5969	9.1636
8	63	16.2227	8.1656	29.5367	43.2029	8.5635
9	76	18.5462	4.9653	14.6828	30.6764	11.5707
MMRE		18.2699	11.5390	29.3847	29.7889	9.8948

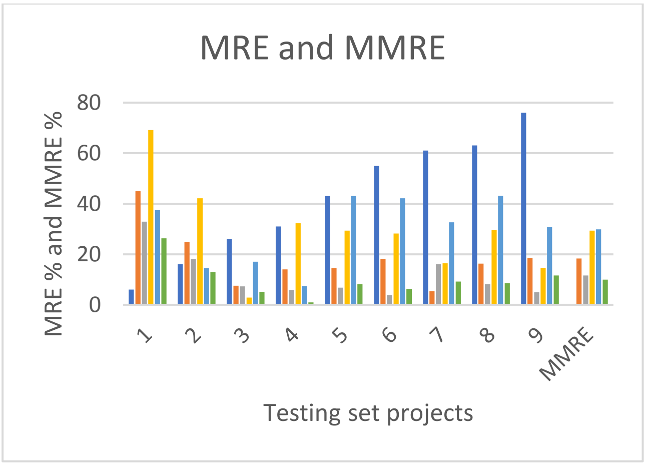

Figure 4: MRE and MMRE for modified COCOMO-NN

V. Conclusion

In this work, two NN-COCOMO II with different activation functions, namely Tanh, Uni-polar sigmoid, Bi-polar sigmoid, Gaussian and softsign activation functions models were constructed and implemented for effort estimation. The performance of the modified NN_COCOMO II with Tanh produced the more accurate effort in the MMRE measuring criteria. Moreover, it should be noted that the comparison of the effort was done with all the projects in the dataset. However, the MRE and MMRE comparison was done in the testing set only to evaluate the model. This method can be improved further by hybridizing this model with evolutionary algorithms such as genetic algorithm and particle swarm.

Acknowledgments

This paper is funded by Fundamental Research Grant Scheme (FRGS) vote number 1610 from Ministry of Higher Education.

References

[1] C. Jones, "Software cost estimation in 2002," *The Journal of Defense Software Engineering*, vol. 15, pp. 4-8, Jun. 2002.
[2] B. W. Boehm, *Software engineering economics*. Englewood Cliffs, NJ: Prentice-hall, 1981, pp. 9-29.

[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, pp. 345-361, Jul. 1978.

[4] B. W. Boehm, R. Madachy, and B. Steece, *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall, 2000, pp. 12-31.

[5] A. Kaushik, A. Chauhan, D. Mittal, and S. Gupta, "COCOMO estimates using neural networks," *International Journal of Intelligent Systems and Applications*, vol. 4, pp. 22-28, Aug. 2012.

[6] O. Tailor, A. Kumar, and M. P. Rijwani, "A new high performance neural network model for software effort estimation," *International Journal of Innovative Science, Engineering & Technology*, vol. 1, pp. 400-405, May 2014.

[7] S. A. Alshalif, N. Ibrahim, T. Herawan, "Artificial neural network with hyperbolic tangent activation function to improve the accuracy of COCOMO II model," in *The Second International Conference on Soft Computing and Data Mining (SCDM)*, Springer, 2016, pp. 81-90.

[8] C. Lopez-Martin, and A. Abran, "Neural networks for predicting the duration of new software projects," *Journal of Systems and Software*, vol. 101, pp. 127-135, Mar. 2015.

[9] C. S. Reddy, and K. Raju, "A concise neural network model for estimating software effort," *International Journal of Recent Trends in Engineering*, vol. 1, pp. 188-193, May. 2009.

[10] A. Kaushik, A. Chauhan, D. Mittal, and S. Gupta, "COCOMO estimates using neural networks," *International Journal of Intelligent Systems and Applications*, vol. 4, pp. 22-28, Aug. 2012.

[11] A. Kaushik, A. K. Soni, and R. Soni, "A simple neural network approach to software cost estimation," *Global Journal of Computer Science and Technology*, vol. 13, pp. 23-30, Dec. 2013.

[12] M. Madheswaran, and D. Sivakumar, "Enhancement of prediction accuracy in COCOMO model for software project using neural network," in *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, IEEE, 2014, pp. 1-5.

[13] S. Mukherjee, and R. K. Malu, "Optimization of project effort estimate using neural network," in *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, 2014, pp. 406-410.

[14] G. Kumar, and P. K. Bhatia, "Automation of software cost estimation using neural network technique," *International Journal of Computer Applications*, vol. 98, pp. 11-17, Jan. 2014.

[15] R. Sarno, and J. Sidabutar, "Comparison of different Neural Network architectures for software cost estimation," in *2015 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*, 2015, pp. 68-73.

[16] R. Sarno., and J. Sidabutar, "Improving the accuracy of COCOMO's effort estimation based on neural networks and fuzzy logic model," in *International Conference on Information & Communication Technology and Systems (ICTS)*, 2015, pp. 197-202.

[17] P. Rijwani, and S. Jain, "Enhanced software effort estimation using multi layered feed forward artificial neural network technique," *Procedia Computer Science*, vol. 89, pp. 307-312, Dec. 2016.

[18] B. Karlik, and A. V. Olgac, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems,* vol. 1, pp. 111-122, Feb. 2011.