Comparison of Different Neural Network Architectures for Software Cost Estimation

Riyanarto Sarno[1], Johannes Sidabutar[2], and Sarwosri[3]
Department of Informatics Engineering
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
[1]riyanarto@if.its.ac.id, [2]jo.chris93@gmail.com, [3]sri@its-sby.edu

***Abstract*— This research will observe the use of Artificial Neural Networks (ANN) for estimating software cost. Constructive Cost Model (COCOMO) is the most famous estimating model for software cost, which will be used in this research. The model estimates software cost by calculating several variables which are created by expert with some equations. Furthermore, ANN helps to estimate COCOMO effort accurately. This research offers multilayer feed-forward neural network to adjust COCOMO effort estimation parameters. Also, an algorithm such as Back-propagation is applied to improve the architecture by comparing actual effort with estimated effort and updating the network. However, there are several types of neural network architecture. This research tries to compare several types of architecture by testing each architecture model to dataset. This paper concerns with two different architectures. The difference of this two architecture is basic architecture only uses effort multipliers as input layer while modified architecture divides input layer into two categories such as effort multipliers and scale factors. The result is the proposed model increases the accuracy and each model has different result.**

***Keywords*—** software cost estimation, COCOMO, neural networks, artificial neural network, cost driver, effort estimation, back propagation, feed forward.

I. INTRODUCTION

Estimating application cost is the major issue in developing an application. Accuracy of estimated effort is crucial to manage the software development resources. By overestimating or underestimating software effort can impact directly to software cost and resources. Recently, a quick configuration of software is critical to manage the development of software [3]. This issue has motivated significant research in recent decades. There are some techniques to overcome this problem such as artificial neural network, analogy method [1], linear regression [2], mathematical function, and other methods.

This research is concerned about developing model to estimate application cost based on Constructive Cost Model (COCOMO). COCOMO was published by B. Boehm [4] in 1981. He is one of the pioneering in estimating software effort. COCOMO depends on several variables which are called cost drivers. This cost drivers consist of Effort Multiplier (EM), Scale Factors (SF), and Line of Code (LOC). All of this cost drivers was made by expert. The model was taken from 63 projects at TRW Aerospace. These data are waterfall model and have range in size from 2000 to 100000 lines of code.

Improving COCOMO effort estimation accuracy is important to make it closer to actual effort. Non algorithmic methods increase accuracy of COCOMO effort estimation. Several researches have been proved that Neural networks is the best method to handle this problem. Artificial neural networks (ANN) have been developed from biological neuron system. A simple ANN architecture consists of two layer: input and output layer. While multilayers ANN are developed with more than two layer. The layers between input and output are called hidden layer. Like human neurons, each layer of ANN has artificial neurons or nodes. This nodes transmit input signal to other nodes use a weighted connection. This paper proposed an artificial neural network with feed-forward algorithm and back-propagation learning method. But, there is no absolute architecture to handle this problem. This research tries to build some architectures and compares the accuracy of each architecture.

In order to build accurate architecture, this paper investigates some interesting papers which are concerned about effort estimation techniques. A. Idri, T.Khoshgoftaaar, and A. Abran [9] have been proved that neural network can improve accuracy of COCOMO effort estimation. They used feed forward architecture with back propagation learning algorithm to propose their model. Reddy and Raju [10] tried to improved accuracy of COCOMO by investigating the use of neural network. Back-propagation is implemented as a learning algorithm to develop the network by measuring the difference between estimation and actual effort repeatedly. The result is that they can improve the accuracy significantly.

This paper is arranged as follows: In Section 2, it shows literature review which describes any information that can help readers to understand the research. It consists of software cost estimation, COCOMO, and artificial neural network description. In Section 3, it describes proposed model that we use in this research. In Section 4, the result is presented uses all the proposed model. In Section 5, it discusses a conclusion of all the research.

II. LITERATURE REVIEW

A. Software Cost Estimation

Software cost estimation could be described as the close judgment of the costs for a software project. The accuracy of estimated cost is critical in developing software. During early

stage of development, estimated effort can help to manage the plan and budget of a project. Since there is a limit in project staffs or funds, all of the development process must be maintained efficiently and effectively. However, the most major problem to estimate software cost is the uncertainty in software development. So, there are some techniques and procedures to handle this issue. Both algorithmic method and non-algorithmic method can help to estimate software cost. Algorithmic method usually use linear regression techniques by collecting data from past projects. Non-algorithmic method tries to construct rules that fit to the data. These include artificial neural network, fuzzy and genetic algorithm. Effort is a unit to measure cost estimation. It describes the number of time for each staff to develop the software. Commonly, the more efforts are used, the more expensive cost will be.

B. COCOMO Model

The Constructive Cost Model (COCOMO) is an algorithm which is observed by B. Boehm to estimate software cost. The model was generated from real project data and used basic regression formula with some parameters. In 1981, COCOMO 81 was introduced as the early COCOMO model. The model consists of 15 different cost drivers. Every cost drivers has range from Very Low to Extra High based on each contribution. There are three different modes which depend on work situation, scope, and project thresholds. The conditions are:

- Organic: is applied for tiny staff and usually in software house.
- Embedded: has strong threshold of hardware, software, regulations and operational procedures.
- Semi-detached: is a condition between organic and embedded and has almost 300,000 lines of code.

There are two mathematical expressions that are applied to calculate effort and schedule time. They are:

$$PM = a * (KDSI)^b * EAF \quad (1)$$

$$TDEV = c * (PM)^d \quad (2)$$

Where:

- PM is effort in person-month
- EAF is the effort adjustment factor
- TDEV is the schedule time
- KDSI is the number of lines of code (in thousands)
- a, b, c, and d are all constants based on the mode which we are using (shown in Table 1)

TABLE 1.
List of Constants Based on Mode

Model	A	b	C	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

In July of 1994, B. Boehm started COCOMO II project to develop model that would be helpful to estimate software cost for the later era [4]. There are several major goals of COCOMO II. They are [5]:

- Developing model based on projects of the 1990's and 2000's.
- Developing model for improving model repeatedly.
- Providing framework and method to evaluate the use of software improvements.

There are 17 effort multipliers and 5 scale factors that have been applied to COCOMO II model. Table 2 shows group of COCOMO II effort multiplier [6].

TABLE 2.
List of COCOMO II's Cost Drivers

Category	Effort Multiplier
Product Attributes	RELY - Required software reliability
	DATA - Database size
	CPLX- Product complexity
	RUSE – Developed for Reusability
	DOCU – Documentation match to life-cycle needs
Computer Attributes	TIME – Execution time constraint
	STOR – Main storage constraint
	PVOL – Platform volatility
Personnel Attributes	ACAP – Analyst capability
	PCAP – Programmer capability
	PCON – Personnel continuity
	APEX – Application experience
	PLEX – Platform experience
	LTEX – Language and tool experience
Project Attributes	TOOL – Use of software tools
	SITE – Multisite development
	SCED – Required development schedule

There are two main equation. They are:

$$PM = A * (Size)^E * \prod_{i=1}^{n} EM_i \quad (3)$$

$$TDEV = C * (PM)^F \quad (4)$$

Where A = 2.94; C = 3.67; E = Scale Factors

C. Artificial Neural Network

Artificial neural network (ANN) was extracted from the complex connection of biological neurons and derived the learning and reasoning properties of human brains. Like people, artificial neural networks discover by example. Artificial neural network has to be constructed so that a set of inputs generates

the expecting set of output. ANN is formed of at least two layer which are input and output. Multilayer artificial neural network uses hidden layer as a mediator between input and output (Fig. 1.).

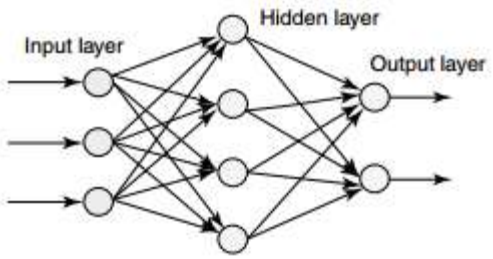

Figure 1. Multilayer artificial neural network.

Each layer contains neuron or nodes. Each node connects with other nodes with a weight. The most usual architecture for neural network is feed forward network with back-propagation learning algorithm [7, 8]. The training algorithm of back propagation involves four stages:

1. Initialization of weights
2. Feed forward
3. Back Propagation of errors
4. Updation of the weights and biases.

III. PROPOSED WORK

The efficiency of neural network is based on its architecture and their parameters approach [11]. Some parameters that improve neural networks are amount of layers, amount of nodes for every layer, activation function, and also learning algorithm which will update the network. There is no clear rule to build neural network architecture and determines each parameter. The aim of this work is to compare between two different neural network architecture which also use different parameter. Both of the architecture use feed-forward neural network with back-propagation learning algorithm and all weights that are used for initial value is set as 1.

A. Basic COCOMO Neural Network

This architecture uses COCOMO 81 parameters as its input. The neural network has 16 parameters (15 Effort Multipliers and 1 KDSI) and 2 biases. There is a hidden layer between input and output layer. There is no specific value for number of nodes. In this research we use 5 nodes in hidden layer. Learning rate (η) is a constant of how fast the network will learn. The bigger number of learning rate, the faster it will learn. But, sometimes bigger number of learning rate could make the learning process over fitting. Fig. 2. shows the proposed architecture that will be applied in this research. There are some steps to implement this model. They are

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

Step 2 : Initialize the weights, biases and number of nodes in hidden layer.
$w_i=wh_i=1$; $bias_i=1$

Step 3 : Set learning rate η ($0 < \eta \leq 1$)

Step 4 : Test stopping condition for false,
Repeat the steps 5 to 13

Step 5 : For each training data,
Repeat the steps 6 to 12

Step 6 : Compute the hidden layers
$Hidden_j = b_1 + \sum X_i * w_{ij}$ for i=1 to 16; j = 1 to n

Step 7 : Activate the hidden layers
$Hidden_i = \frac{1}{1+e^{-Hidden_i}}$ for i = 1 to n

Step 8 : Compute the output layer
$e = bias_3+Hidden_1*wh_1+...+Hidden_n*wh_n$

Step 9 : Count error
$err = \ln(\text{Actual Effort}) - e$

Step 10 : Count Δw and hidden layer errors
$\Delta wh_i = \eta * err * hidden_i$ for i = 1 to n
$hidden_i\_err = err * wh_i * hidden_i * (1-hidden_i)$ for i = 1 to n
$\Delta w_i = hidden_i\_err * \eta * X_i$ for i = 1 to n
$\Delta wb_1 = hidden_i\_err * \eta$
$\Delta wb2 = \eta * err$

Step 11 : Update the weights
$wh_i = wh_i + \Delta wh_i$ for i = 1 to n
$wij = wij + \Delta w_j$ for i = 1 to 16; j = 1 to n
$bias_i = bias_i + \Delta wb_i$ for i = 1 to 2

Step 12 : Count the test data using new weights

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

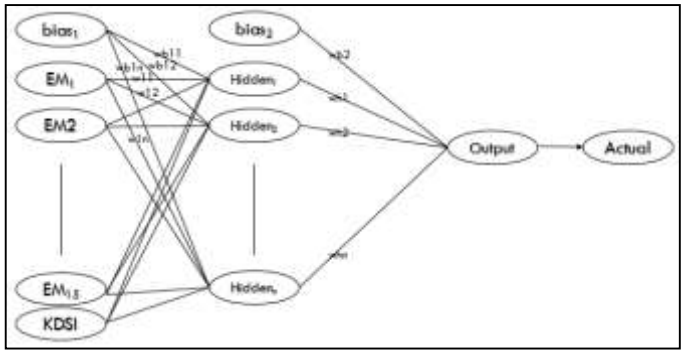

Figure 2. Basic COCOMO Neural Network Architecture

B. Modified COCOMO Neural Network

The major change for this architecture is the use of Scale Factors which are derived from COCOMO II. Number of nodes in hidden layer also differs than basic architecture. In this proposed model, there are only two nodes of hidden layer. One is connected with Effort Multipliers input while the other is linked with Scale Factors input. Moreover, number of lines of code (KLOC) is not used as one of nodes in input layer, whereas it is applied as a constant in scale factors weight. Fig. 3. presents the architecture of modified neural network. This model also has several steps to use the architecture. They are:

Step 1 : Initialize the inputs as ln(input)

Step 2 : Initialize the weights and biases
$w_i=wb_i=wh_i=1$; $v_i=0$; $bias_i=1$

Step 3 : Set learning rate η ($0 < \eta \leq 1$)
Step 4 : Test stopping condition for false,
Repeat the steps 5 to 13
Step 5 : For each training data,
Repeat the steps 6 to 12
Step 6 : Compute the hidden layers
$Hidden_1 = b_1 + \sum X_i * w_i$ for i=1 to 17
$Hidden_2 = b_2 + \sum Y_i * (v_i + \ln(size))$ for i=1 to 5
Step 7 : Activate the hidden layers
$Hidden_i = \frac{1}{1+e^{-Hidden_i}}$ for i = i to 2

Step 8 : Compute the output layer
$e = bias_3 + Hidden_1 * wh_1 + Hidden_2 * wh_2$
Step 9 : Count error
$err = \ln(\text{Actual Effort}) - e$
Step 10 : Count Δw, hidden layer errors and Δv
$\Delta wh_i = \eta * err * hidden_i$ for i = 1 to 2
$hidden_i\_err = err * wh_i * hidden_i * (1-hidden_i)$ for i = 1 to 2
$\Delta w_i = hidden_1\_err * \eta * X_i$ for i = 1 to 17
$\Delta v_i = hidden_2\_err * \eta * Y_i$ for i = 1 to 5
$\Delta wb_i = hidden_i\_err * \eta$ for i = 1 to 2
$\Delta wb_3 = \eta * err$
Step 11 : Update the weights
$wh_i = wh_i + \Delta wh_i$ for i = 1 to 2
$wi = wi + \Delta w_i$ for i = 1 to 17
$vi = vi + \Delta v_i$ for i = 1 to 5
$wb_i = wb_i + \Delta wb_i$ for i = 1 to 3
Step 12 : Count the test data using new weights
Step 13 : Test stopping condition
If the error between estimated effort and actual effort in test data is smaller than a specific tolerance or the number of iteration exceeds a specific number, stop : else continue.

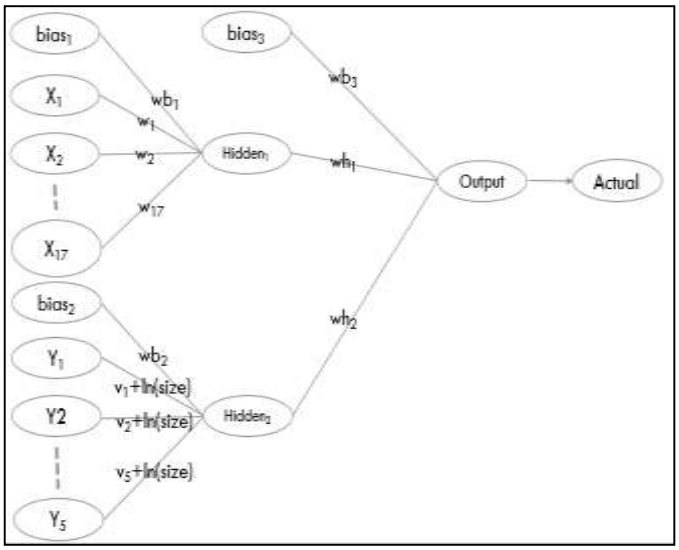

Figure 3. Modified COCOMO Neural Network Architecture

IV. EXPERIMENTAL RESULT

This sections displays the outcome which is obtained by applying two different model on datasets. This research is applied by using COCOMO 81 and NASA 93 datasets. COCOMO 81 has of 63 data while NASA has 93 data. Each data is composed of at least 17 variables which are 15 Effort Multipliers, 1 KLOC, and 1 Actual Effort. Each dataset is divided into two category such as 80% of training data and 20 % of testing data.

The accuracy of estimated effort is calculated by using the most popular method such as Magnitude Relative Error (MRE) and Mean Magnitude Relative Error (MMRE) [12], which are described as in "(5,6)"

$$MRE = \frac{|Actual\ Effort - Estimated\ Effort|}{Actual\ Effort} * 100 \quad (5)$$

$$MMRE = \frac{1}{N}\sum_{x=1}^{n} MRE_i \quad (6)$$

Table 3 presents the result and comparison on NASA 93 dataset by using COCOMO and both of proposed model. For example, Project ID 10 has 69% of MRE by applying COCOMO Model, neural network improves the accuracy dramatically which scores 4.22% of MRE for Basic Neural Network and 3.77 of MRE for Modified Neural Network. The MMRE for NASA 93 datasets is 45.74% for COCOMO, 21.06% for basic neural network and 18.27% for modified neural network.

Table 4 presents the result and comparison on COCOMO 81 dataset. As NASA 93 dataset, both of the proposed model could improve the accuracy significantly. The MMRE for COCOMO 81 datasets is 37.1% for COCOMO, 22.61% for basic neural network and 21.22% for modified neural network. Overall, modified neural network has better accuracy than basic neural network, but few projects show different result. This deviation is caused by lack of dataset and diversity of each data points in dataset.

TABLE 3.
COMPARISON OF EFFORT ESTIMATION RESULTS IN MRE ON NASA 93 DATASET

Project ID	MRE(%) using COCOMO Model	MRE(%) using Basic Neural Network	MRE(%) using Modified Neural Network
4	44.57	8.92	6.15
9	61.45	2.42	21.18
10	69	4.22	3.77
14	24.13	24.27	51.32
20	26.14	5.03	43.58
27	37.94	2.88	0
30	41.76	5.9	2.94
37	25.47	0.66	8.73
39	34.48	8.82	6.79
43	57.64	43.17	32.87
44	31.65	39.35	14.34
55	58.11	27.72	13.9
59	74.18	48	45.16
63	19.28	20.64	7.14
64	61.26	19.42	8.78
70	30.63	33.72	21.5
73	26.90	25.77	24.52
79	63.37	27.93	4.22
90	81.25	51.37	30.26
MMRE(%)	**45.74**	**21.06**	**18.27**

TABLE 4.
COMPARISON OF EFFORT ESTIMATION RESULTS IN MRE ON COCOMO 81 DATASET

Project ID	MRE(%) using COCOMO Model	MRE(%) using Basic Neural Network	MRE(%) using Modified Neural Network
2	50.87	33.94	35.68
6	51.92	7.09	0.02
15	40.32	7.86	1.91
18	71.25	45.31	36.74
25	58.46	20.55	16.28
26	57.67	24.36	28.01
29	18.64	2.85	40.6
39	8.5	5.95	21.38
40	0.75	82.03	47.49
42	73.75	10.30	2.13
45	26.65	12.98	18.32
52	17.11	14.05	1.86
62	6.41	26.6	25.49
MMRE(%)	**37.1**	**22.61**	**21.22**

V. CONCLUSION

Estimating software cost is a major for both the software industrial and academic researchs. An accurate estimated cost is needed in the early stage of application development in order to manage budget and resources. We have implemented a model to estimate application cost based on COCOMO and apply neural network to increase the accuracy. The neural network that we have applied is multilayer feed-forward neural network with back-propagation algorithm. The back-propagation has been used to find out the error of estimated effort and update the network.

This research has proved that neural network could increase the accuracy of COCOMO for both COCOMO 81 dataset and NASA 93 dataset. This research also proves that modified neural network architecture has better MMRE rather than basic neural network. This research could be continued by connecting this model with fuzzy logic to minimize the uncertainty of software cost estimation.

VI. REFERENCES

[1] R. Sarno, J. L. Buliali and S. Maimunah, "Pengembangan metode analogy untuk estimasi biaya rancang bangun perangkat lunak," *MAKARA Journal of Technology Series,* vol. 6, no. 2, pp. 46 - 55, 2002.

[2] V. Anandhi and R. M. Chezian, "Regression techniques in software effort estimation using cocomo dataset," *Intelligent Computing Applications (ICICA),* pp. 353 - 357, 2014.

[3] R. Sarno, C. Djeni, I. Mukhlash and D. Sunaryono, "Developing a workflow management system for enterprise resource planning," *Journal of Theoretical and Applied Information Technology,* vol. 72, no. 3, pp. 412-421, 2015.

[4] B. Boehm, Software Engineering Economics, New Jersey, USA: Prentice-Hall, Englewood Cliffs, 1994.

[5] B. Boehm, B. Clark, E. Horowitz, R. Madachy, R. Shelby and C. Westland, "Cost models for future software life cycle processes: COCOMO 2.0," *Annals of Software Engineering,* 1995.

[6] B. Boehm, Software Cost Estimation with COCOMO II, New Jersey, USA: Prentice Hall, 2000.

[7] K. Molokken and M. Jorgensen, "A review of software surveys on software effort estimation," *Proceedings of IEEE International Symposium on Empirical Software Engineering ISESE,* pp. 223-230, 2003.

[8] S. Huang and N. Chiu, "Applying fuzzy neural network to estimate software development effort," *Proceedings of Applied Intelligence Journal,* pp. 73 - 83, 2009.

[9] A. Idri, T. Khoshgoftaaar and A. Abran, "Can neural networks be easily interpreted in software cost estimation," *World Congress on Computational Intellegence,* 2002.

[10] C. Reddy and K. Raju, "A concise neural network model for estimating software effort," *International Journal of Recent Trends in Engineering,* vol. 1, no. 1, pp. 188 - 193, 2009.

[11] A. Kaushik, A. Soni and R. Soni, "A simple neural network approach to software cost estimation," *Global Journal of Computer Science and Technology,* vol. 13, no. 1, pp. 23 - 30, 2013.

[12] L. Briand, K. Emam, D. Surmann and I. Wieczorek, "An assessment and comparison of common software cost estimation modeling techniques," *ISERN,* 1998.