IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 3, May 2014.

www.ijiset.com

ISSN 2348 – 7968

A NEW HIGH PERFORMANCE NEURAL NETWORK MODEL FOR SOFTWARE EFFORT ESTIMATION

OMPRAKASH TAILOR1,**AMIT KUMAR**2 & **Mrs. POONAM RIJWANI**3

Student M.Tech, Department of Computer Science and Engineering, Pratap University, Jaipur, India1&2

Assistant Professor & HOD Department of Computer Science and Engineering, Pratap University, Jaipur, India3

Abstract: In this research, it is concerned with concerned with constructing software effort estimation models based on artificial neural network. The model is designed accordingly to improve the performance of the network that suits to the COCOMO model. Recent year the software industry is growing rapidly and people pay more attention on how to keep high efficiency in the process of software development and management. In the process of software development , time, cost, manpower are all critical factor. At the stage of software project planning, project manager will evaluate these parameter to get an efficient software develop process. Software effort evaluate is an important aspect which includes amount of cost, schedule, and manpower requirement. In this paper, it is proposed to use multilayer feed forward neural network to accommodate the model and its parameter to estimate software development effort. The network is trained with back propagation learning algorithm by iteratively processing a set of training samples and comparing the network's prediction with actual effort. COCOMO dataset is used to train and to test the network and it was observed that proposed neural network model improve the estimation accuracy of the model. The compared with that of the OCOCMO model. The aim of this study is to enhance the estimation accuracy of COCOMO model , so that the estimated effort is more close to the actual effort.

KEYWORDS: COCOMO, back propagation, , feed forward neural networks, software effort estimation, Artificial Neural Network, software cost estimation.

I.INTRODUCTION

Estimating software development method remains a complex problem and one which continues to attract considerable research attention. Accurate cost estimation of a software development effort is critical for good management decision making. The precision and reliability of the effort estimation is very important for software industry because both overestimate and underestimate of the software effort are harmful to software companies. The improvement of the precision and reliability in the software development process will promote the efficiency of development significantly. An important objective of software engineering community has been to develop useful model that are accurately estimating software effort. Many project manager tend to use reliable and precise software estimation model to evaluate among these model constructive cost model (COCOCMO) the effort estimation approach commonly acknowledged as the most accurate and easy to use which is based on model proposed by Barry Bohem in 1981. In 2001 COCOMO II was proposed which has a significant improvement and modification and COCOMO 81 making it more suitable for estimating the model software development project. The post architecture level of COCOMO II include 17 cost drivers which present software reliability, development tool , and programming ability, etc. Recent year artificial neural network combined with fuzzy system, genetic algorithm evolutionary computation and will have and now deep development in practical application. Besides, artificial neural network apply widely in

ISSN 2348 – 7968

software effort estimation. Artificial neural network can model complex non premier relationship and approx any major able function. The most commonly adopted architecture for estimating software effort is feed forward multilayer perceptron with back propagation learning algorithm and sigmoidal activation function. In this paper, it is concerned with constructive cost estimation model based on ANN. Particularly multi-layer feed forward neural network. The network model is designed accordingly to accommodate the COCOCMO model and its parameter and back propagation learning algorithm is used to train the network by iteratively processing a set of training samples and comparing the network prediction with the actual. The difference in estimation is propagated to the input for adjusting the coefficient. The process consist of repeatedly feeding input and output data from empirical observation ,propagating the error value and adjusting the connection weights until the error values fall below a user specified tolerance level.

II. LITERATURE SURVEY

Over the last decades there are many software cost estimation models have been introduced. Jorgensen provides a detailed review of different studies on the software development effort. Neural networks are good at modeling complex nonlinear relation and it have learning ability too. It provides more flexibility to integrate expert knowledge into the model. These valuable efforts also shown a promising research direction in software cost estimation and the approaches on neural network are very far from nature. There are many researchers who have applied the neural approach to estimate software development effort or many different modelsof neural network have been proposed . They may be classified in two major categories . first one is feed forward network where no loops in the network path occur. The other one is feed back network that have recursive loop. The feed forward multilayer perceptron with back propagation remaing algorithm are the most commonly used in the cost estimation field. The other study by samson et.al (5) used an albus multilayer perceptron to predict software effort. They use Bohem's COCOMO data set rather than srinivasan and fisher [7] report the use of a neural network outperformed other techniques.karunanithi et. al [9] produced very accurate result about primary work in the use of neural network in estimating software cost, but the large set back in the work was due to the fact the accuracy of the result relied heaving on the size of the training set. The most popular and widely used software estimation model is COCOMO,which integrates valuable expert knowledges[3]. To understanding the advertising in applying neural network, Nasser Tadayon [11] has propsed a dynamic neural network that will initally use COCOMO II model. Still COCOCMO has some limitation like COCOMO can not effectively deal with imprecise and uncertain informaton, and calibration of COCOMO is one of the very important task which need to be done in order to get accurate estimation. So, for the better predictive accuracy there is always scope revelent to developing various estimation models.

III. PROPOSED WORK

A. problem Formulation

In a wide sense the neural network itself is a model because topology and transfer function of the nodes are usually problems. For various application many network architecture have been developed. The performance of a neural network totally depends upon its architecture and parameters setting. There are so many parameters leads the architecture of the neural network including the number of layers, the number of nodes in each layer, and the transfer function in each node and the weight which determine the connectivity between nodes. There is no clearly mentioned theory which allows for the calculation of the ideal parameters setting and as rule even slight parameters changes can cause major variation in the behavior of almost all networks.

The techniques which uses the neural network for predicting software cost estimation is back propagation trained multilayer feed forward networks with sigmoidal activation function. There are some

ISSN 2348 – 7968

limitation that prevents it from accepted as common practice in cost estimation slow convergence is the one major drawback of the back propagation learning. The sigmoid activation function used in its hidden and output layer units, it will causes slow convergence in back propagation. B.E. Segee examined the behavior of the conventional back propagation network. He found that the sigmoidal function is very ill behaved and always mismatched for the function to be learned. This is why the network with sigmoidal function will learn more slowly and will be more sensitive to the loss of parameters than networks using more suitable activation functions.

Ratherthan, inappropriate selection of network patterns and learning rules many cause real difficulties in network performance and training. The problem at hand decides the number of layers and number of nodes in the layers and learning algorithm as well. However the guiding criterion have to select the minimum nodes which would not impair the network performance so that memory demand for storing weight can be kept minimal. So the number of layers and nodes should be minimized to amplify the performance. A proper selection of tuning parameters such as momentum factor, learning and designing of stable network. Hence to overcome the above limitation and to improve the performance of the network that suits to the COCOMO Model , in this research,it is proposed an architecture that suits to the COCOMO Model, using multi layer feed forward neural network with back propagation learning algorithm and an identity function.

B.Tuning the COCOMO

Although many algorithmic models are usually not very accurate, they reflect a lot of useful expert knowledge in this field. In order to make the best use of this kind of expert knowledge, it is worthwhile to integrate the algorithmic model ininitial to the present neural network model. Calibrating the parameters of algorithmic model is usually is very challenging task. In the neural network model, parameters of the algorithmic model are adjusted through learning . The initial defination of the COCOMO II had the Following Form:

$$Effort = A * [Size]^{1.01 + \sum_{i=1}^{5} SF_i} * \prod_{i} EM_i \qquad (1)$$

where, A is the multiplicative constant, Size of the software project measured in terms of KSLOC, SF is the Scale Factors and EM is the Effort Multipliers. The COCOMO Model shown in "(1)," is a non linear model. To solve this problem we transform the non-linear model of "(1)," into a linear model using the logarithms to the base e, is shown as follows in "(2)".

$$\ln(PM) = \ln A + \ln(EM_1) + \ln(EM_2) + \ldots + \ln(EM_{17}) + [1.01 + SF_1 + \ldots + SF_5] * \ln(Size) \qquad (2)$$

The above equation has the form of the model below: Part 1

$$O_{est} = \overbrace{[b_1 + u_1 * I_1 + u_2 * I_2 + \ldots u_{17} * I_{17}]}^{} + \underbrace{[b_2 + I_{18} + \ldots + I_{22}] * [v_i + \ln(size)]}_{P} \qquad (3)$$

art 2 where

$O_{est} = \ln(PM)$;

$I_1 = \ln(EM_1)$;……; $I_{17} = \ln(EM_{17})$; $I_{18} = SF_1$; ….. ; $I_{22} = SF_5$;

ISSN 2348 – 7968

b_1 and b_2 are the biases and the coefficients u_i, v_i are the additional terms introduced in the model, which are to be estimated as follows. Initially we assume that the values of the coefficients are to be 1 for u_i and 0 for v_i to estimate the output O_{est}. Then this estimated effort is compared with that of the actual observed effort in the log space. The difference is the error in the estimation, which should be minimized. All the model's parameters are assumed to be responsible for the output error [5]. The difference in this estimation is propagated to the inputs for adjusting the coefficients, so that this knowledge can be incorporated in to the model in the form of coefficients. Thus these coefficients of the model can be estimated by the proposed neural network as discussed below. Note that part 1 of "(3)," deals with effort multipliers, which represents the upper section of the neural network under study and part 2 deals with scale factors that represents the lower section.

C.Architecture of NN model

This paper proposes a new architecture with multi layer feed forward neural network and is constructed to accommodate the COCOMO II model, to overcome the limitations which were identified in the previous section. The proposed model improves the performance of the network and the accuracy of the estimation in predicting the effort. The use of the proposed neural network to estimate ln(PM) (Person months in logs) in the "(2)," requires 22 input nodes in the input layer that corresponds to all EM, SF and it has two bias values. The proposed network consists of two hidden layers nodes H_{EM} and H_{SF} that take into account the contribution of Effort Multipliers and Scale Factors separately as shown in "Fig. 1."

All the inputs are normalized to speed up the training process of the network, as it is already proved that the neural networks work better if inputs and outputs lie between 0 and 1 [20]. However, in order to structure the network to accomplish the COCOMO II model with multi layer feed forward neural network, some pre-processing of data for the input layer is considered.

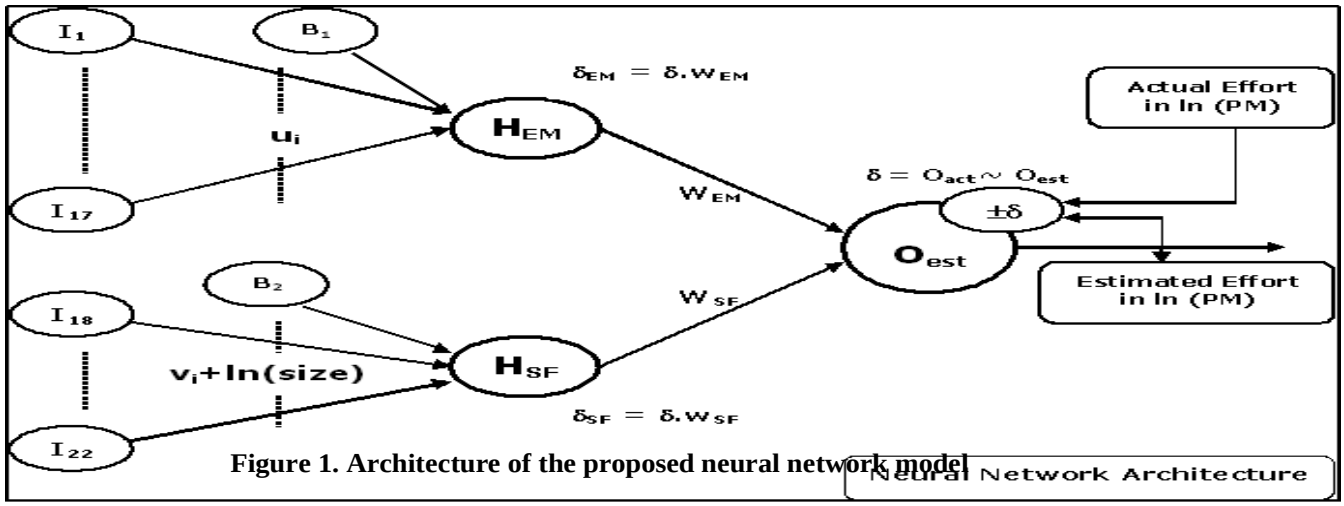

Figure 1. Architecture of the proposed neural network model

Note that in the above network, all the input values I_1 to I_{22} are pre-processed according to the model defined in
"(3)". The size of the product in KSLOC is not considered as one of the inputs to the network but considered as a cofactor for the weights for the scale factors. To overcome the draw backs associated with sigmoidal functions and to accomplish the effort in ln person months, identity function is used at the input, hidden and output units. The weights associated with each input nodes connected to the hidden layers are denoted by u_i for each input ln (EM_i) for 1 £ i £ 17 and for B_1. On the other hand, the weights associated with each SF_i input nodes to the hidden layer are v_i + ln(size) for 18 £ i £ 22 and the bias denoted by B_2. The weights associated with each hidden layer nodes connected to the output

layer are denoted by w_{EM} and w_{SF}. The process of establishing a neural network involves, initializing the network and the training the network. Our study focuses on these two steps. The operational diagram of our proposed network model is shown in "Fig. 1."

D. Initializing and Training the Network with Learning Algorithm

The proposed network will be initialized by giving initial value of b_1 as log (A) and b_2 as 1.01. The biasing is done to neutralize the network from extreme inputs, so that the network can tolerate extreme inputs. The remaining weights in the network are initialized to small values as u_i =1 for i = 1 to 17, v_i = 0 for i = 18 to 22. In order to accommodate COCOMO II formula the initial values of weights are set as w_{EM} = w_{SF} = 1. Propagating the inputs forward, we get the response of each node at the hidden layer and the value of ln(PM) at node in the output layer i.e. O_{est}. The following algorithm describes the procedure for getting response of nodes in the hidden and output layers. Here we used identity transfer function for all the input, hidden and output layers. Using the above network architecture, with the set of known initial weights of the network, we initially predict the software effort in ln person months. After each project is finished, the actual value of PM is calculated and compared against the predicted value in the natural logarithmic space.

Before using any adjustment to the weights of the network, we consider the expert judgment's input and train the network accordingly. This can be achieved by allowing estimators or experts to impose the weight change distribution for training the network. This way, the estimator's knowledge is stored in the network to help with customizing the training of the system. The back propagation procedure is used to train the network by iteratively processing a set of training samples and comparing the network's prediction with the actual. For each training sample, the weights are modified so as to minimize the error between the network predicted value and actual. The algorithmic description for training the above network and for calculating new set of weights is depicted in the following steps.

Step 1 : Initialize the weights
Set learning rate □□(0 < □□£ 1)

Step 2 : Test stopping condition for false, Repeat the steps 3 to 8.

Step 3 : For each training pair, s, Repeat the steps 4 to 7.

Step 4 : Set activations of input units $I_i = s_i$; i = 1 to 22

Step 5 : Compute the response of each unit $H_{EM}= b_1+\square\square I_i * u_i$ for i= 1 to 17
$H_{SF} = b_2+\square\square I_i * (v_i + \ln(size))$ for i= 18 to 22 $O_{est} = H_{EM} * w_{EM} + H_{SF} * w_{SF}$

Step 6 : Update the weights between hidden and output layer.
w_{EM} (new) = w_{EM} (old) + □□* □□* H_{EM}

w_{SF} (new) = w_{SF} (old) + □□* □□* H_{SF}
The error is calculated as □□= (O_{act} – O_{est})

Step 7 : Update the weights and bias between input and hidden layers.
u_i (new) = u_i (old) + □*□$_{EM}$* I_i for i = 1 to 17 v_i (new) = v_i (old) + □*□$_{SF}$* I_i for i = 18 to 22 b_1 (new) = b_1 (old) + □□* □$_{EM}$
b_2 (new) = b_2 (old) + □□* □$_{SF}$ The error is calculated as
□$_{EM}$ = □□* w_{EM} ; □$_{SF}$ = □□* w_{SF} ;

Step 8 : Test stopping condition;
If the error is smaller than a specific tolerance or the number of iteration exceeds a specific value, stop : else continue.

Using this approach, we iterate forward and backward until the terminating condition is satisfied. This terminating condition is when the error is smaller than a specific tolerance level or a specific number of iterations have been done. When this condition is met the network is stopped and these weights will be used as a knowledge source for estimating effort for the next epoch. The variable □□used in the above formula is the learning rate, a constant, typically having a value between 0 and 1.

The learning rate can be increased or decreased by the expert judgment indicating their opinion of the input effect. In other words, the error should have more effect on the expert's indication that a certain input had more contribution to the error propagation or vice

versa. For each project, the expert estimator can identify the importance of the input value to the error in the estimation. If none selected by the expert, the changes in the weights are as specified by the learning algorithm. Note that the estimated effort is recalculated after the size of a project is known to determine the error since we are training the network to adjust to the factors and not the error caused by the size estimation. The network should also be trained according to correct inputs. For example, if during estimation, CPLX (Product Complexity) is set as low but after end of project, the management realizes that it was nominal or even high, then the system should not consider this as a network error and before training the system, the better values of cost factors should be used to identify the estimated cost.

IV. CONCLUSION

The back propogation algoritham is used to train the network. We have used full COCOMO dataset to train and to test the network and it is observed that our neural network model provide significantly better cost estimation than the estimation done using cocomo model . Another great advantage of this work is that we could put together expert knowledge, project data and the traditional algorithmic model into one general frame work that can have a wide range of applicability in software cost estimation.

REFERENCES

[1] Anthony Senyard et al., "Software engineering methods for neural Networks," IEEE proceddings of the Tenth Asia-Pacific Software engineering conference,(ASPEC 03) 2003.

[2] Bohem B.,Clark B., Horowitz E. Madachy R. " Cost Model for future Software Life Cycle Processes: COCOMO 2.0," Annals of Software Engineering 1995.

[3] Bohem , B.W., " Software Engineering Economics " Prentic –Hall , Englewood Cliffs, NJ, USA 1994.

[4] Iman Attarzadeh, Amin Mehranzadeh, Ali Barati " Proposing an Enhance Artificial Neural Network Prediction Model to Improve the accuracy in Software Effort Estimation," in computational Intellegience communication system and network conference , pp. 167-172, 2012.

[5] Samson , B.,Ellison, D., Dugard, P., " Software cost Estimation using an Albus perceptron (CMAC),"Journal of Information and Software Technology, Volume 39(1), 55-60,1997.

[6] Jovan popovic and Dragan Bojic, " A comparative Evaluation of Effort Estimation Method in the Software Life Cycle ", Com SIS Vol. 9, No. 1, January 2012.

[7] Srinivasan, k., Fisher, D., "Machine Learning approches to Estimating Software development Effort," IEEE Transcation on Software Engineering , Volume 21(2), 126-137,1995

[8] Rathi, Kamalraj. R, Karthik. S, " Survey on Effective Software Effort Estimation Technique , International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) ISSN :2278-1323, Volume 1, Issue 8, october 2012.

[9] Karunanithi, N., et al., " Usine Neural Network in reliability prediction ," IEEE Software,pp. 53-59,1992.

[10] P.k. Suri Pallavi Ranjan , " Comparative Analysis of Software Effort Estimation Techniques " International Journal of Computer Application (0975-8887) Volume 48- No. 21, June 2012.

[11] Tadayon, N., " Neural network approach for software cost estimation," International Conference om Information Technology : Coding and Computing (ITCC 2005).