International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-4 Number-1 Issue-14 March-2014

Implementation of Prediction Model for Object Oriented Software Development Effort Estimation using One Hidden Layer Neural Network

Chandra Shekhar Yadav[1], Raghuraj Singh[2]

Abstract

The prediction model for object-oriented software development effort estimation using one hidden layer neural network has been implemented in this paper. This prediction model has been empirically validated on PROMISE software engineering repository dataset. Accurate prediction of software development effort and schedule is still a challenging job in software industry. This prediction model has been implemented through programming in MATLAB using one hidden layer feed forward neural network(OHFNN) and results obtained from this program are compared with existing algorithms like traingda and traingdm of NNTool. By a large number of simulation work OHFNN 16-19-1 is found optimal structure for this prediction model. OHFNN 16-19-1 means 16 neurons in input layer, 19 neurons in hidden layer and 1 in output layer. Training of the neural network has been done by using back propagation with a gradient descent method. Performance of predictor is better in terms of accuracy than existing well established constructive cost estimation model (COCOMO). In this network, convergence is obtained by minimizing the root mean square error of the input patterns and optimal weight vector is determined to predict the software development effort.

Keywords

Effort Estimation, Artificial Neural Network (ANN), One Hidden Layer Feed Forward Neural Network (OHFNN), Back propagation learning with gradient descent

1. Introduction

There are two types of model for prediction of software development effort i.e. model based and model free. Generally model based prediction model is not well suited for the estimation of software development effort due to complex mapping between project characteristics and development effort. In this situation, model free prediction using soft computing is well suited for estimating the development effort of the software project. The most important model based Constructive Cost Estimation Model (COCOMO) has been represented in following equation [1].

$$Effort = a * (KLOC)^b \quad (1)$$

Here, *a*, and *b* are domain specific parameters. For predicting the software development effort, parameters *a*, and *b* have been adjusted on the past data set of various projects. Five scale factors have been used to generalize and replace the effects of the development model in COCOMO II. There are fifteen parameters which affect the effort of software development. These parameters are analyst capability$(acap)$, programmer's capability $(pcab)$, application experience $(aexp)$, modern programming practices $(modp)$, use of software tools $(tool)$, virtual memory experience $(vexp)$, language experience $(lexp)$, schedule constraint $(sced)$, main memory constraint $(stor)$, database size $(data)$, time constraint for CPU $(time)$, turn- around time $(turn)$, machine volatility $(virt)$, process complexity $(cplx)$and required software reliability $(rely)$.

$$Effort = a * (KLOC)^b * c \quad (2)$$

KLOC is estimated directly or computed from a function point analysis and *c* is the product of fifteen effort multipliers.

$$Effort = a * (KLOC)^b * (EM1 * EM2 * \ldots * EM15) \quad (3)$$

Khoshgaftaar et al. (1992, 1994, and 1997) used artificial neural network (ANN) model as a tool for predicting development faults and software quality of a large telecommunications system that have been classified as fault prone or not fault prone. It has been found that ANN model gives better predictive accuracy. In this analysis, raw data has been transformed into variables that are not correlated to each other [2, 3, and 4]. K.K. Shukla (2000) proposed

Manuscript received on February 15, 2014.
Chandra Shekhar Yadav, Computer Science and Engineering, Noida Institute of Engineering & Technology, Greater Noida, India.
Raghuraj Singh, Computer Science and Engineering, Harcourt Butler Technological Institute, Kanpur, India.

the neuro-genetic approach for prediction of software development effort and compared it with both a regression-tree based approach as well as back propagation-trained NN approach [5].
The Remaining part of the paper is organized as follows. In section 2 related works has been explained. In section 3, mathematical model of OHFNN approach to effort prediction has been represented. Section 4 presents result and discussion. Section 7 gives the conclusion drawn from results and future scope of the research work.

2. Related Work

Till now more than three hundred research papers have been published related to software development effort or cost estimation. Various estimation approaches have been used in these papers like regression, expert judgment, work break-down, function points, classification and regression trees, artificial neural network, genetic algorithm, particle swarm optimization, and neuro-fuzzy etc. Colin J. Burgess et al. (2001) used genetic programming for improving the software effort estimation and evaluated potential of genetic programming (GP) in software development effort estimation against existing approaches and found machine learning method can estimate development effort accurately comparison to using only human expertise. Eung Sup Jun et al. (2001)used quasi-optimal case selective neural network model for estimating software development effort and found neural network is more suitable for predictor and estimator than existing statistical models [6, 7]. N. Gupta et al. (2005) used pattern mapping technique of artificial neural network for estimating the reliability of software with execution time model [8].Gyimothy et al. (2005) empirically validated chidamber and kamerer metrics on various open source software for predicting the fault. The regression (linear and logistic regression) and machine learning methods have been applied for prediction model [9].Liang Tian et al. (2005) modified Levenberg-Marquardt algorithm with Bayesian regularization for improving the ability to predict the software cumulative failure time [10].Parameters of COCOMO model have been tuned to predict software development effort by using GA, e.g. Alaa F. Sheta (2006) [11]. K.K. Agarwal et al. (2006) used artificial neural network for predicting the maintainability of object oriented system [12]. S. Kanmani et al. (2007) used neural network model for predicting the fault during the development of object oriented system [13].Yogeshet al. (2008) predicted testing effort using ANN method for NASA projects and established relationship between OO metrics and testing effort [14].JieXu et al. (2008) validated object oriented metrics for fault prediction [15]. The network has been trained with back propagation learning algorithm and resilient back propagation algorithm for comparing network prediction with the actual effort e.g. Ch. Satyananda Reddy (2009, 2010) [16, 17].Yuming Zhou et al. (2010) analyzed object-oriented design metrics set for predicting high and low severity faults [18]. Alaa F. Sheta et al. (2010) proposed GA effort estimation model to predict software development with the help of line of code and methodology. In this model only two independent parameters have been considered. In the proposed model all the sixteen parameters proposed in COCOMO II model are used to predict the software development effort [19]. Ruchika Malhotra et al. (2011) used machine learning techniques for predicting fault in object oriented software [20]. Anil Kumar et al. (2012) proposed model using particle swarm optimization (PSO) for tuning the parameters of basic COCOMO model to predict the software development effort accurately. In this model only KLOC parameter has been considered [22].Manisha et al. (2013) explored multilayer feed forward neural network as bidirectional associative memory (BAM) for function approximation. In future this network can be used for faster convergence [23]. Performance Analysis of Software Effort has been done using neural networks, e.g. E. Praynlin (2013) [24]. Somesh Kumar et al. (2013) used feed forward neural network with a descent gradient of distributed error and the genetic algorithm (GA) for recognizing the handwritten 'SWARS' of Hindi curve script [25]. C.S. Yadav et al. (2014) modified the COCOMO81 model for better prediction of development effort using GA. This modified model has been validated on PROMISE project data set [26].

3. OHFNN Approach to Effort Prediction

To estimate the mapping function between input vector and desired output vector, OHFNN with gradient descent back propagation learning method has been used in this model. Let us consider input vector say $X_k^T = (x_1, x_2, \ldots x_n)$ where n=16 and output vector say $D_k^T = (d_1; d_2, \ldots d_p)$. The Neural Network can be trained by using the input and output vector mapping. Flow chart of back propagation

training is shown in figure 1. *P* is set of *Q* training vector pairs;

$$P = \{X_k, D_k\}_{k=1}^{Q} \quad (4)$$

$$X_k \in R^n, D_k \in R^p, where\ n = 16, p = 1\ and\ Q = 40$$

Here net_k generates an output signal vector $f(Y_k)$, Y_k is vector of activations of output layer neuron.

Error at k^{th}training pair (X_k, D_k) is as follows

$$E_k = D_k - f(Y_k) \quad (5)$$

$$Where\ E_k = (e_1^k, \dots e_p^k)^T$$

$$= \left(d_1^k - f(y_1^k), \dots, d_p^k - f(y_p^k)\right)^T \quad (6)$$

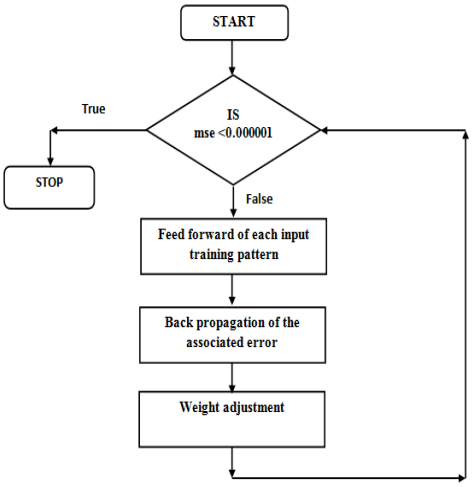

Figure 1: Flow chart of back propagation training

Squared error is sum of squares of each individual output error e_j^k, i.e.

$$\xi_k = \frac{1}{2}\sum_{j=1}^{p}(d_j^k - f(y_j^k))^2 = \frac{1}{2}E_k^T E_k \quad (7)$$

The mean square error (mse), is computed over the entire training set P

$$mse = \frac{1}{Q}\sum_{k=1}^{Q}\xi_k \quad (8)$$

Hidden to output layer weights are updated as follows

$$w_{hj}^{k+1} = w_{hj}^k + \Delta w_{hj}^k \quad (9)$$

Input to hidden layer weights are updated as follows

$$w_{ih}^{k+1} = w_{ih}^k + \Delta w_{ih}^k \quad (10)$$

Where Δw_{hj}^k and Δw_{ih}^k are weight changes computed in previous step

$$f(x_i^k) = x_i^k, i = 1, \dots, n \quad (11)$$

$$f(x_0^k) = 1, \forall k \quad (12)$$

Here x_i^k is the i^{th} component of the input vectorX_k and

$f(x_0^k)$ *is bias value at input layer*

For the hidden layer

$$Z_h^k = \sum_{i=0}^{n} w_{ih}^k\ f(x_i^k), h = 1, \dots, q \quad (13)$$

$$f(Z_h^k) = \frac{1}{1 + e^{-z_h^k}}, h = 1, \dots, q \quad (14)$$

Where $w_{0h}^k$is the bias value at hidden layer, and$f(z_0^k)$ is the hidden layer bias neuron signal.

For the output layer

$$y_j^k = \sum_{h=0}^{q} w_{hj}^k f(z_h^k), j = 1, \dots, p \quad (15)$$

$$f(y_j^k) = \frac{1}{1 + e^{-y_j^k}}, j = 1, \dots, p \quad (16)$$

Here w_{0j}^k are the biases of output neurons.

$$\Delta w_{ij} = -\eta \frac{\partial \xi_k}{\partial w_{ij}^k} \quad (17)$$

Here η is learning rate and w_{ij} is weight from node *i* to node *j*. The activation of each node *j*, for pattern *X* is as follows

$$net_{X_j} = \sum_i w_{ij}\ f_X \quad (18)$$

The output from each node *j* is as follows

$$f_X = f\left(net_{X_j}\right) \quad (19)$$

Weights are adjusted due to change in error by the following equation

$$\Delta w_{ij} = \eta\left\{k f_x(1 - f_x)\sum_k \delta_k' w_{jk}\right\} f_x \quad (20)$$

Now the weights have been updated in output and hidden layer by the following equations

$$\Delta w_{hj}^{k+1} = \Delta w_{hj}^k + \eta\delta_X^k\ f(z_h^k),$$

$$\Delta w_{ih}^{k+1} = \Delta w_{ih}^k + \eta\delta_X^k\ x_i^k \quad (21)$$

We can introduce the momentum into back propagation with the help of following equations

$$\Delta w_{hj}^k = \eta\delta_j^k f(z_h^k) + \alpha\Delta w_{hj}^{k-1}\ and$$

$$\Delta w_{ih}^k = \eta\delta_h^k x_i^k + \alpha\Delta w_{ih}^{k-1} \quad (22)$$

All the fifteen parameters which have linguistic values affect the effort are shown in Table 1.

Table 1: Various parameters which affect the effort estimation with addition to KLOC

S. No	Name	Meaning
1	$acap$	Analyst capability $\{N,H,VH\}$
2	$pcap$	Programmer's capability $\{N,H,VH\}$
3	$aexp$	Application experience $\{N,H,VH\}$
4	$modp$	Modern Programming practices $\{L,N,H,VH\}$
5	$tool$	Use of software tool $\{VL,L,N,H,VH\}$
6	$vexp$	Virtual machine experience$\{L,N,H\}$
7	$lexp$	Language experience$\{VL,L,N,H\}$
8	$sced$	Schedule constraint $\{L,N,H\}$
9	$stor$	Main memory constraint $\{N,H,VH,EH\}$
10	$data$	Database size $\{L,N,H,VH\}$
11	$time$	Time constraint for CPU $\{N,H,VH,EH\}$
12	$turn$	Turn- around time $\{L,N,H\}$
13	$virt$	Machine volatility $\{L,N,H\}$
14	$cplx$	Process complexity $\{L,N,H,VH,EH\}$
15	$rely$	Required software reliability $\{L,N,H,VH\}$

$VL \rightarrow Very\ Low, L \rightarrow Low, N \rightarrow Nominal,$
$H \rightarrow High, VH \rightarrow Very\ High,$
$EH \rightarrow Extremaly\ High$

Suppose we assign 1 to *VL*, 2 to *L*, 3 to *N*, 4 to *H*, 5 to *VH* and 6 to *EH*. After normalizing the value we get the following:

$rely = \{0.2722, 0.4082, 0.5443, 0.6804\}$
$data = \{0.2722, 0.4082, 0.5443, 0.6804\}$
$cplx = \{0.2108, 0.3162, 0.4216, 0.5270, 0.6325\}$
$time = \{0.3235, 0.4313, 0.5392, 0.6470\}$
$stor = \{0.3235, 0.4313, 0.5392, 0.6470\}$
$virt = \{0.3714, 0.5571, 0.7428\}$
$turn = \{0.3714, 0.5571, 0.7428\}$
$acap = \{0.4243, 0.5657, 0.7071\}$
$aexp = \{0.4243, 0.5657, 0.7071\}$
$pcap = \{0.4243, 0.5657, 0.7071\}$
$vexp = \{0.3714, 0.5571, 0.7428\}$
$lexp = \{0.1826, 0.3651, 0.5477, 0.7303\}$
$modp = \{0.2722, 0.4082, 0.5443, 0.6804\}$
$tool = \{0.1348, 0.2697, 0.4045, 0.5394, 0.6742\}$
$sced = \{0.3714, 0.5571, 0.7428\}$

In normalization we use formula $n = \left(\sqrt{1/sum(m^2)}\right).m$

Suppose vector $a = [2,3,5,8,9]$ then normalized vector $b = normr(a)$. If we want to obtain vector *a* from vector *b*, then we use the following formula:

$a = b./e;\ where\ e = b(1,1)/a(1,1)$

Here $a(1,1)$ is the first element of vector *a*,$b(1,1)$ is the first element of vector b.

OHFNN comprises of many processing units which have been organized in various layers. Every processing unit in a layer is connected with all the processing units of the previous layer. All the connections in this network are not equal. Each connection has a different strength or weight. In this network, data has been given at input layer and signal passes through hidden layer reaches at output layer. OHFNN consists of one input layer, one output layer and one hidden layer. The computation has been done by hidden layer. Each processing unit in hidden layer performs additional operation based on input from previous layer. The result obtained from hidden layer is transformed by sigmoid function [21].

4. Result and Discussion

First, OHFNN with one neuron is used in this prediction model with two bias values $b_1$at hidden layer and b_2 at output layer as shown in figure 2. The parameters used in this architecture OHFNN 16-1-1 are shown in table 5. After training the network on 40 patterns, weights of various connections from input to hidden layer and from hidden to output layer are shown in table 3. This network reduces sum of squared error to 0.0125 in 10 million epochs. Though this network is taking exponential time for the execution but good thing is that network converges. After that three neurons have been taken at hidden layer with two bias values $b_1$at hidden layer andb_2 at output layer as shown in figure 3. After training the network on same data set, this network reduces the sum of squared error to 1.3589e-005 in 70, 63978 epochs. The weights of various connections of this network from input to hidden layer and from hidden to output layer are shown in Table 4. This network also takes large time to converge but less than the previous network. Both the networks predict the software development effort precisely. From this experiment, it is clear that as the neurons at hidden layer increases, network is taking less time to converge. But it is not always true because in further experiment it is shown that OHFNN 16-19-1 takes less time to converge than OHFNN 16-23-1. The basic data processing unit of neural network is called the neuron. The parameters used in OHFNN 16-3-1

are shown in Table 6. From various simulation runs OHFNN 16-19-1 is optimal structure for convergence in less time. Root mean square errors for OHFNN 16-19-1 at different learning rates are represented in Table 7. Using this model software development efforts of 19 projects have been predicted as shown in Table 2. In this architecture only three neurons have been taken at hidden layer. How many neurons should be taken at hidden layer is not decided by any formulae in neural network. By hit and trial method number of neurons at hidden layer can be decided. Here a program is written in MATLAB to implement this prediction model in which a variable is taken for number of neurons at hidden layer. By varying the number of neurons at hidden layer, sum of mean squared error can be reduced up to tolerance range in minimum number of epochs. In this simulation work first neural network has been trained on 40 patterns and then validated on another data set [27]. In this model gradient descent with momentum training has been used.The proposed algorithm shows better results in terms of predicted accuracy than the existing algorithms of NN Tool as shown in table 8. But this model takes large epochs for convergence to obtain the accuracy. Here our main objective is to predict software development effort accurately. Therefore, the large number of epochs can be compromised for better prediction because the training is one time process. When the network is trained correctly with the training pattern set; software development effort is predicted precisely using this network. Comparison between actual efforts with predicted efforts of 19 object oriented software projects using existing traingda learning algorithm of NNTool is shown in the Figure 4. By varying the number of neurons at hidden layer of neural network architecture, the optimal neural architecture of OHFNN is 16-19-1 for traingdm and traingda training methods of NNTool. Best validation performance of OHFNN 16-19-1 with traingda is 0.0088132 at epoch 2173 and the best validation performance of OHFNN 16-19-1 with traingdm is .012839 at epoch 1, 00,900. During the analysis of this work it has been found that development effort of some projects is not predicted precisely. From table 7 best root mean square error is 0.00149074 for network architecture OHFNN 16-19-1 at learning rate 1.01 and at momentum 0.7 in one million epochs.

5. Conclusion and Future Scope

In this research work, by a large number of simulation work some of the above suitable architecture of OHFNN has been given to predict the development effort accurately. Performance index of prediction model depends not only on the architecture of network and learning algorithm for training but also on historical data sets of various projects. In this study, OHFNN 19-16-1 has been fixed with both the training algorithms for having common platform in the comparison of the performance. In future other neural network like Radial Basis Function (RBF) can be used for the prediction model. With the help of wide range data sets on various diverse projects, prediction model can be trained and used to predict the software development effort. Software development cost and time are also dependent on development effort. So by predicting development effort accurately requirements of software cannot be overestimated or underestimated. Correlation coefficient between actual and predicted effort using proposed algorithm is 0.999966 and correlation coefficient between actual and predicted effort using NNTool is 0.980257. So, it can be said that proposed algorithm of prediction model is better than algorithm of NNTool in terms of accuracy. But in terms of number of epochs and time algorithms of NNTool is better than proposed algorithm. In this model, training of network is very slow. Training may be fast by using faster algorithm. The fast algorithm uses heuristic and optimization techniques for increasing the speed of network training. Conjugate gradient, Quasi-Newton and Levenberg-Marquardt algorithms of MATLAB NN tool may be used to increase the performance of network. These fast algorithm uses optimization techniques for increasing the performance. Multilayer feed forward network as bidirectional associative memory architecture can also be used to train this prediction model of software development effort. Two hidden layers feed forward network may be used to train the network of prediction model. For adjusting the weights of connection for better results Neuro GA or Neuro PSO may be applied. Both GA and PSO are optimization techniques.

References

[1] B.W. Boem, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[2] Taghi M. Khoshgoftaar, Abhijit S. Pandya, and Hemant B. More, “A Neural Network Approach

for Predicting Software Development Faults", 0-8186-2975-4/92 IEEE, 1992.

[3] Taghi M. Khoshgoftaar, Robert M. Szabo, "Improving Neural Network Predictions of Software Quality Using Principal Components Analysis", IEEE World congress on computational Intelligence, 1994 IEEE international conference volume:5, pp. 3295-3300 DOI:10.1109/ICNN.1994.374764.

[4] Taghi M. Khoshgoftaar, Edward B. Allen et.al, "Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunication System", IEEE Transactions on Neural Networks VOL. 8, NO. 4, July 1997.

[5] K.K. Shukla, "Neuro-genetic prediction of software development effort", Information and Software Technology 42(2000) 701-713 ELSEVIER.

[6] Colin J. Burgess, Martin Leffley, "Can genetic programming improve software effort estimation? A comparative evaluation", Information and Software Technology 43(2001) 863-873 ELSEVIER.

[7] Eung Sup Jun, Jae Kyu Lee, "Quasi-optimal case-selective neural network model for software effort estimation", Expert Systems with Applications 21(2001) 1-14 PERGAMON.

[8] Nidhi Gupta, Manu Pratap Singh, "Estimation of software reliability with execution time model using the pattern mapping technique of artificial neural network", Computers & Operations Research 32(2005) 187-199, ELSEVIER.

[9] Tibor, Gyimothy, Rudolf Ferenc, and IstvanSiket, "Empirical Validation of Object- Oriented Metrics on Open Source Software for Fault Prediction", IEEE Transactions on Software Engineering, VOL. 31, NO. 10 October 2005 pp 897-910.

[10] Liang Tian, AfzelNoore, "Evolutionary neural network modeling for software cumulative failure time prediction", Reliability Engineering & System Safety 87(2005) 45-51 ELSEVIER.

[11] Alaa F. Sheta, "Estimation of the COCOMO Model Parameters using Genetic Algorithms for NASA Software Projects", Journal of Computer Science 2(2): 118-123, ISSN 1549-3636, 2006.

[12] K.K. Agarwal, Yogesh Singh, ArvinderKaur, and RuchikaMalhotra, "Application of Artificial Neural Network for Predicting Maintainability using Object Oriented Metrics", World Academy of Science Engineering and Technology 22 2006.

[13] S. Kanmani, V. RhymendUthariaraj, V. Sankaranarayanan, P. Thambidurai, "Object-oriented software fault prediction using neural networks", Information and Software Technology 49(2007) 483-492 ELSEVIER.

[14] Yogesh Singh, ArvinderKaur, RuchikaMalhotra, "Predicting Testing Effort using Artificial Neural Network", Proceedings of the World Congress on Engineering and Computer Science 2008.

[15] JieXu, Danny Ho and Luiz Fernando Capretz, "An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction", Journal of Computer Science 4(7): 571-577, 2008.

[16] Ch. Satyananda Reddy, KVSVN Raju, "A Concise Neural Network Model for Estimating Software Effort", International Journal of Recent Trends in Engineering, Issue.1, Vol. 1, May 2009 pp. 183-193.

[17] Ch. Satyananda Reddy and KVSVN Raju, "An Optimal Neural Network Model for Software Effort Estimation", Int. J. of Software Engineering, IJSE Vol.3 No.1 January 2010 pp 63-78.

[18] Yuming Zhou, BaowenXu, Hareton Leung, "On the ability of complexity metrics to predict fault-prone classes in object-oriented systems", The Journal of Systems and Software 83(2010) 660-674, ELSEVIER.

[19] Alaa F. Sheta, Alaa Al-Afeef, "A GP Effort Estimation Model utilizing Line of Code and Methodology for NASA software projects" 978-1-4244-8136-1 IEEE Transaction, 2010 pp. 284-289.

[20] RuchikaMalhotra, Yogesh Singh, "On the Applicability of the Machine Learning Techniques for Object Oriented Software Fault Prediction" Software Engineering: An International Journal (SEIJ), VOL. l, NO. 1, September 2011.

[21] SaurabhShrivastava, Manu Pratap Singh, "Performance evaluation of feed-forward neural network with soft computing techniques for hand written English alphabets", Applied Soft Computing 11(2011) 1156-1182.

[22] Anil Kumar, C. S. Yadav et.al, "Parameter Tuning of COCOMO Model for Software Effort Estimation using PSO" ISBN 978-93-81583-34-0 pp. 99-105 ICIAICT 2012.

[23] Manisha Singh, Somesh Kumar, "Exploring Optimal Architecture of Multi-layered Feed-forward (MLFNN) as Bidirectional Associative Memory (BAM) for Function Approximation", IJCA (0975-8887), Vol. 76, No. 16, August 2013.

[24] E. Praynlin, P. Latha, "Performance Analysis of Software Effort Estimation Models Using Neural Networks", I.J. Information Technology and Computer Science, 2013, 09,101-107.

[25] Somesh Kumar, Manu Pratap Singh et al ,"Hybrid evolutionary techniques in feed forward neural network with distributed error for classification of handwritten Hindi (SWARS)", Connection Science, 2013 Taylor & Francis pp. 197-215.

[26] Chandra Shekhar, Raghuraj Singh, "Tuning of COCOMO81 Model Parameters for estimating software development effort using GA for

PROMISE Project Data Set", International Journal of Computer Applications (0975-8887) vol. 90, No. 1 Foundation of computer science, New York, USA,2014 pp. 37-43.

[27] Data Website: http://promise.site.uottawa.ca/SERepository/datasets/cocomo81.arff.

Mr. Chandra Shekhar Yadav is an Associate Professor in Computer Science & Engineering Department at Noida Institute of Engineering & Technology (NIET), Greater Noida. He has about 15 years of experience in teaching. He received Master of Computer Application degree from Institute of Engineering & Technology (IET), Lucknow in year 1998, M. Tech (Computer Science & Engineering) from JSSATE, Noida in year 2007. Currently he is pursuing Ph.D. (Computer Science & Engineering) from U.P. Technical University, Lucknow. He has supervised 06 M. Tech. theses.

Dr. Raghuraj Singh is Professor and Head in Computer Science & Engineering Department at Harcourt Butler Technological Institute (HBTI), Kanpur. He has about 24 years of experience in teaching & research at various other Institutions like Birla Technical Training Institute, Pilani and C. R. State College of Engineering, Murthal, Distt.-Sonepat. He received B. Tech. degree from H.B.T.I., Kanpur, M.S. (Software Systems) from BITS, Pilani and Ph.D. degree from U. P. Technical University, Lucknow. His area of research is software Engineering and Human Computer Interface. He has published more than 80 research papers in various National and International journals & conferences. He has supervised 07 Ph.D. and 20 M. Tech. theses.

Table 2: Actual and predicted software development effort of 19 projects

S. No.	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	I16	A_Eff	P_Eff
1	VH	H	H	VH	VH	N	N	VH	VH	VH	N	H	H	H	L	227	1181	1181
2	H	L	H	N	N	L	L	N	N	N	N	H	H	N	L	14	60	51.73565
3	N	N	H	H	N	N	N	N	N	N	N	N	N	N	N	16	114	113.5806
4	N	N	H	N	N	N	N	N	N	N	N	N	N	N	N	13	60	60.06093
5	N	N	H	N	N	N	N	N	N	N	N	N	N	N	N	8	42	44.5997
6	N	N	N	N	N	N	N	N	H	H	N	N	N	N	N	10	48	51.73565
7	H	L	H	N	N	L	L	N	N	N	N	H	H	N	L	24.6	117.6	99.90332
8	H	L	H	N	N	L	L	N	N	N	N	H	H	N	L	2.2	8.4	19.0292
9	H	L	H	N	N	L	L	N	N	N	N	H	H	N	L	3.5	10.8	21.40785
10	N	L	H	N	N	L	L	H	VH	H	N	H	N	N	N	15	48	44.5997
11	N	L	H	N	N	L	L	H	H	H	N	H	N	N	N	31.5	60	59.46626
12	N	L	H	N	N	L	L	H	VH	H	N	H	N	N	N	6	24	20.81319
13	N	L	H	N	N	L	L	H	VH	VH	N	H	N	N	N	20	72	66.00755
14	H	L	H	N	N	L	L	N	N	H	N	N	H	VL	N	302	2400	2400.058
15	H	N	H	H	H	L	H	N	H	N	N	N	L	VH	N	219	2120	2119.972
16	H	N	H	H	H	L	H	N	H	N	N	N	L	VH	N	50	370	368.6908
17	N	N	H	N	H	N	N	H	H	N	N	H	H	N	H	47.5	252	252.137
18	H	H	N	N	N	L	L	N	H	H	N	H	N	N	N	79	400	400.208
19	N	H	L	N	N	H	N	H	H	N	N	N	H	H	N	282	1368	1367.724

Table 3: Weights of various connections from input to hidden layer (Wih) and from hidden to output layer (Whj) for OHFNN 16-1-1

$$w_{ih} = \begin{bmatrix} 4.8219 \\ 3.5145 \\ 2.0018 \\ -7.9542 \\ -2.6366 \\ 2.1341 \\ -1.3342 \\ -2.6753 \\ 2.9029 \\ -3.0343 \\ 1.2645 \\ -3.2814 \\ 0.4911 \\ -2.0013 \\ 0.2822 \\ 1.9156 \\ -10.4672 \end{bmatrix}_{17\times1} \qquad w_{hj} = \begin{bmatrix} 0.3888 \\ -6.5480 \end{bmatrix}_{2\times1}$$

Table 4: Weights of various connections from input to hidden layer (Wih) and from hidden to output layer (Whj) OHFNN 16-3-1

$$W_{ih} = \begin{vmatrix} -1.0015 & 2.3724 & -2.8761 \\ -1.8439 & 1.7511 & 1.9971 \\ 2.5393 & -4.4116 & 2.2543 \\ -4.1587 & -0.5479 & -4.5701 \\ -4.0960 & 1.8103 & 2.4022 \\ 2.0439 & -0.3764 & 2.8698 \\ 0.0178 & 2.6123 & 1.3908 \\ 3.6324 & -2.6101 & 1.4602 \\ -1.3092 & 6.2803 & -3.7826 \\ -0.4264 & -3.4456 & -3.9910 \\ -1.0202 & -0.0110 & -1.3624 \\ 4.9126 & -6.6467 & 0.4777 \\ -0.9289 & 0.1218 & -1.2316 \\ -0.7409 & -3.3314 & -0.2065 \\ 0.2200 & 1.1672 & 1.5450 \\ -1.3565 & -0.2027 & 2.7929 \\ 2.5582 & -36.3012 & 8.8004 \end{vmatrix}_{17\times3} \qquad W_{hj} = \begin{vmatrix} -2.2769 \\ -6.4795 \\ -18.0189 \\ 6.8314 \end{vmatrix}_{4\times1}$$

Table 5: Parameters used in OHFNN 16-1-1 Architecture

Parameters	**Number/Values**
Input Neurons	16
Neurons in hidden layer(h1) q1	01
Number of patterns Q	40
Learning rate, η	0.98
Momentum,α	0.7
Tolerance, τ	0.00001
Learning Algorithm	Back Propagation with gradient descent

Table 6: Parameters used in OHFNN 16-3-1 Architecture

Parameters	**Number/Values**
Input Neurons	16
Neurons in hidden layer(h1) q1	03
Number of patterns Q	40
Learning rate, η	0.98
Momentum, α	0.7
Tolerance, τ	0.00001
Learning Algorithm	Gradient descent back Propagation

Table 7: Root mean square errors for OHFFN 16-19-1 at different learning rates

epochs	Root Mean Square errors Momentum α=0.7						
	η= 0.01	η= 0.8	η=0.7	η=0.5	η=1.05	η=1.01	η=0.98
10 thousand	0.01054032	0.00753939	0.00799794	0.00847195	0.00746453	0.00736268	0.00748933
50 thousand	0.00615286	0.00457790	0.00503273	0.00519174	0.00484754	0.00448162	0.00489588
100 thousand	0.00498955	0.00409193	0.00452595	0.00461027	0.00440810	0.00398843	0.00443357
200 thousand	0.00449541	0.00342913	0.00417068	0.00424649	0.00403550	0.00327033	0.00385286
300 thousand	0.00429477	0.00289301	0.00389827	0.00402509	0.00359866	0.00271478	0.00317870
400 thousand	0.00415403	0.00247100	0.00351732	0.00376302	0.00300512	0.00230791	0.00260461
500 thousand	0.00402252	0.00216147	0.00305622	0.00342543	0.00241248	0.00203263	0.00221567
1 million	0.00308021	0.00150979	0.00165208	0.00182862	0.00159311	0.00149074	0.00151314

Table 8: Comparison between Actual Effort and Predicted Effort

S. No.	I16	Actual Effort	Predicted Effort using proposed Algorithm	Predicted Effort using NNTool
1	227	1181	1181	1184.568
2	14	60	51.73565	77.30614
3	16	114	113.5806	111.7966
4	13	60	60.06093	68.38662
5	8	42	44.5997	56.49295
6	10	48	51.73565	46.91889
7	24.6	117.6	99.90332	121.3112
8	2.2	8.4	19.0292	30.32779
9	3.5	10.8	21.40785	35.67976
10	15	48	44.5997	48.76234
11	31.5	60	59.46626	74.33283
12	6	24	20.81319	41.03172
13	20	72	66.00755	83.2527
14	302	2400	2400.058	1691.815
15	219	2120	2119.972	2043.856
16	50	370	368.6908	431.7251
17	47.5	252	252.137	60.06093
18	79	400	400.208	411.5066
19	282	1368	1367.724	1375.455

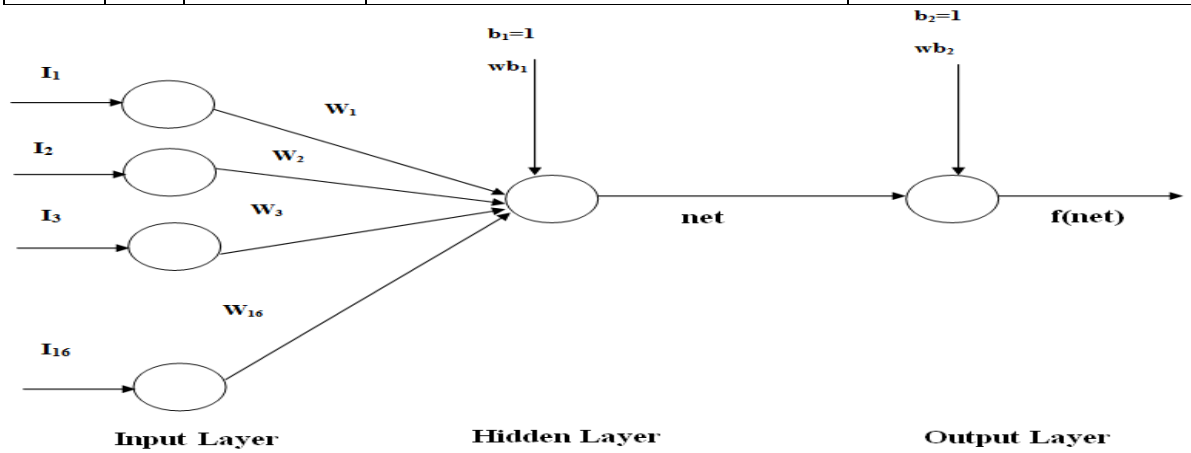

Figure 2: OHFNN 16-1-1with one neuron

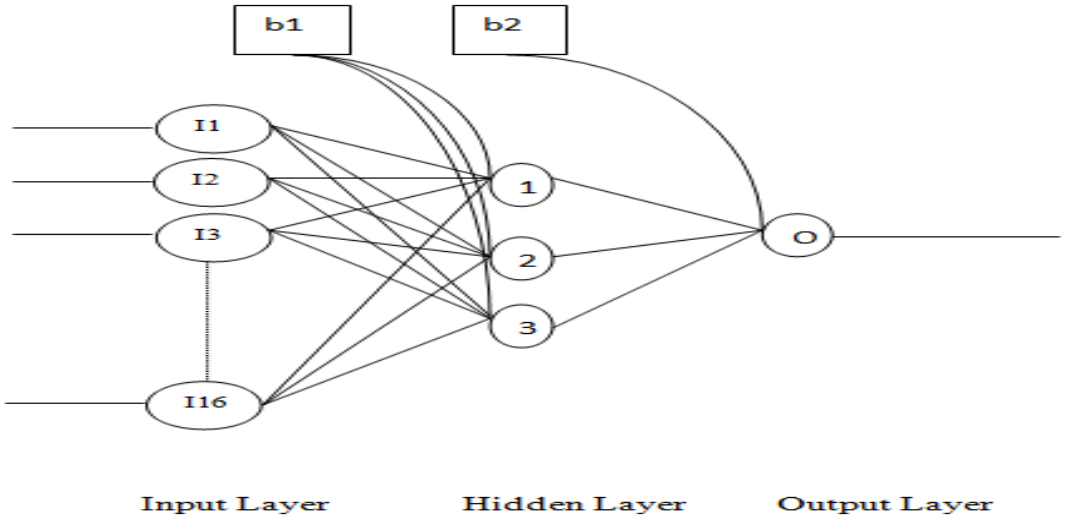

Figure 3: OHFNN 16-3-1 with three neurons

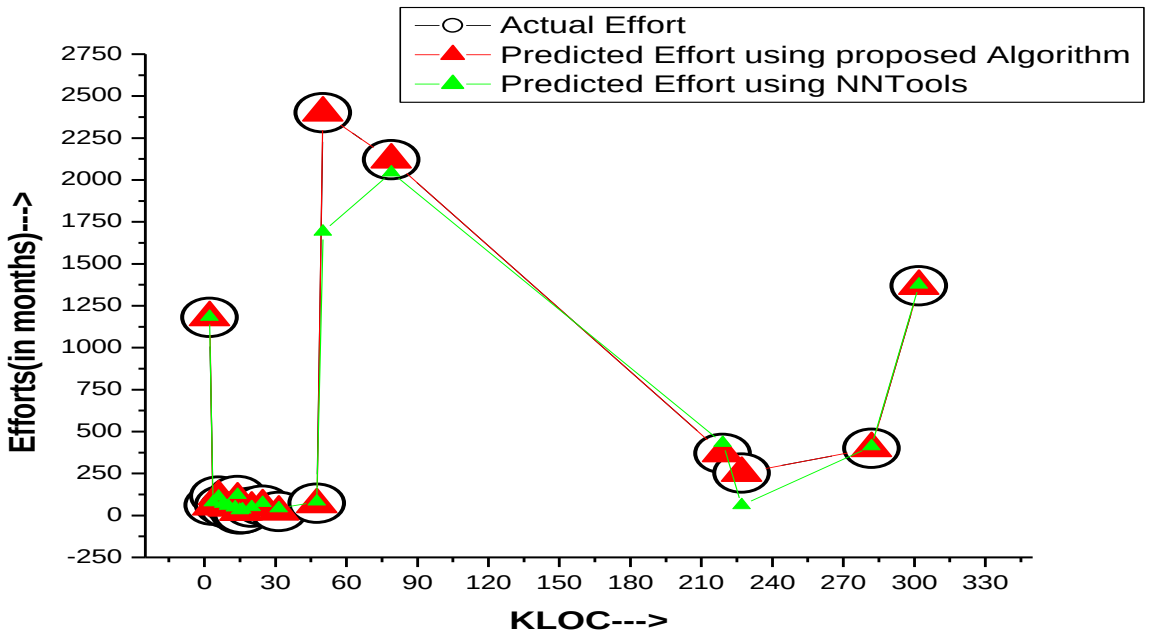

Figure 4: Comparison between Actual effort and Predicted effort using proposed algorithm and traingda algorithm of NNtool