

A Concise Neural Network Model for Estimating Software Effort

Ch. Satyananda Reddy, KSVSN Raju
DENSE Research Group

Department of Computer Science and Systems Engineering,
College of Engineering, Andhra University, Visakhapatnam.
satyanandau@yahoo.com, kvsvn.raju@gmail.com

Abstract -- In this research, it is concerned with constructing software effort estimation model based on artificial neural networks. The model is designed accordingly to improve the performance of the network that suits to the COCOMO model. In this paper, it is proposed to use multi layer feed forward neural network to accommodate the model and its parameters 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 the actual effort. COCOMO dataset is used to train and to test the network and it was observed that proposed neural network model improves the estimation accuracy of the model. The test results from the trained neural network are compared with that of the COCOMO model. The preliminary results obtained suggest that the proposed architecture can be replicated for accurately forecasting the software development effort. 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.

Index Terms -- artificial neural networks, back propagation, COCOMO, feed forward neural networks, software cost estimation, software effort estimation.

I. INTRODUCTION

Estimating software development effort 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 overestimates and underestimates of the software effort are harmful to software companies. Thus, from an organizational perspective, an early and accurate cost estimate will reduce the possibility of organizational conflict during the later stages [25]. Predicting software development effort with high precision is still a great challenge for project managers. An important objective of the software engineering community has been to develop useful models that are accurately estimating the software effort.

Among the software cost estimation techniques, COCOMO (Constructive Cost Model) is the mostly used algorithmic cost modeling technique because of its simplicity for estimating the effort in person-months for a project at different stages. COCOMO uses the mathematical formulae to predict project cost estimation. Many researchers have [4, 6, 9, 16] explored the possibility of using neural networks for estimating the effort. Artificial neural networks can model complex non-linear relationships and approximate any measurable function. They are particularly useful in problems where there is a complex relationship between an input and output.

The most commonly adopted architecture for estimating software effort is feed forward multilayer perceptron with back propagation learning algorithm and the sigmoid activation function. One of the major drawbacks in back propagation learning is its slow convergence. The main reason for slow convergence is the sigmoid activation function used in its hidden and output layers. The network with sigmoidal function will be more sensitive to the loss of parameters. Besides that inappropriate selection of network patterns and learning rules may cause serious difficulties in network performance and training. So the number of layers and nodes should be minimized to amplify the performance. Hence to overcome the above limitations and to improve the performance of the network that suits to the COCOMO Model, in this research, it is proposed concise neural network architecture with linear activation function.

In this paper, it is concerned with constructing cost estimation model based on artificial neural networks, particularly multi layer feed forward neural networks. The network model is designed accordingly to accommodate the COCOMO model and its parameters, and back propagation learning algorithm is used to train the network by iteratively processing a set of training samples and comparing the network's prediction with the actual. The difference in the estimation is propagated to the input for adjusting the coefficients. This process consists of repeatedly feeding input and output data from empirical observations, propagating the error values, and adjusting the connection weights until the error values fall below a user-specified tolerance level.

The rest of the paper is organized as follows: Section 2 presents the review of the work done in the application of neural network techniques to software effort estimation. Section 3 describes the architecture of the proposed neural network model. It also discusses the learning algorithm used in training the network. Section 4 presents the experimental methodology and results. Finally, Section 5 summarizes our work and gives conclusions and further directions.

II. RELEVANT WORK

There are many software cost estimation models have been developed over the last decades. A recent study by Jorgensen [10] provides a detailed review of different studies on the software development effort. Neural networks have learning ability and are good at modeling complex nonlinear relationships; provides more flexibility to integrate expert knowledge into the model. Although these efforts showed a promising research direction in software cost estimation, the approaches based on neural network are far from mature. Many researchers have applied the neural networks approach to estimate software development effort [7, 8, 11, 15, 17, 19, 21, and 24]. Many different models of neural networks have been proposed [11]. They may be grouped in two major categories. First one is feed forward networks where no loops in the network path occur. Another one is feedback networks that have recursive loops. The feed forward multilayer perceptron with back propagation learning algorithm are the most commonly used in the cost estimation field. Another study by Samson et al. [15] uses an albus multilayer perceptron in order to predict software effort. They use Boehm's COCOMO dataset. Srinivasan and Fisher [21] report the use of a neural network with a back propagation learning algorithm. They found that the neural network outperformed other techniques.

Some primary work in the use of neural network in estimating software cost by Karunanithi et al. [12] produced very accurate results, but the major set back in their work was due to the fact the accuracy of the result relied heavily on the size of the training set. COCOMO is arguably the most popular and widely used software estimation model, which integrates valuable expert knowledge [3]. Understanding the adversity in applying neural networks, Nasser Tadayon [22] has proposed a dynamic neural network that will initially use COCOMO II Model. COCOMO, however, has some limitations. It cannot effectively deal with imprecise and uncertain information, and calibration of COCOMO is one of the most important tasks that need to be done in order to get accurate estimations. So, there is always scope for developing effort estimation models with better predictive accuracy [13].

III. PROPOSED NEURAL NETWORK MODEL

A. Problem-Formulation

In a broad sense, the neural network itself is a model because the topology and transfer functions of the nodes are usually formulated to match the current problem. Many network architectures have been developed for various applications. The performance of a neural network depends on its architecture and their parameter settings. There are many parameters governing 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, learning algorithm parameters and the weights which determine the connectivity between nodes. There is no clearly defined theory which allows for the calculation of the ideal parameter settings and as a rule even slight parameter changes can cause major variations in the behavior of almost all networks [1].

So far, the technique in the use of the neural network for predicting software cost estimation is back propagation trained multilayered feed forward networks with sigmoidal activation function. But, there are some limitations that prevent it from being accepted as common practice in cost estimation. One of the major drawbacks of the back propagation learning is its slow convergence. The main reason for slow convergence in back propagation is the sigmoid activation function used in its hidden and output layer units. B.E. Segee [18] investigated the behavior of the conventional back propagation networks, he found that the sigmoidal function is very ill behaved and always mismatch 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.

Furthermore, inappropriate selection of network patterns and learning rules may cause serious difficulties in network performance and training. The problem at hand decides the number of layers and number of nodes in the layers and the learning algorithm as well. However, the guiding criterion is to select the minimum nodes which would not impair the network performance so that memory demand for storing the weights 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 coefficient are required for efficient learning and designing of stable network. Hence to overcome the above limitations 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 into the present neural network model. Calibrating the parameters of algorithmic models is usually is very challenging task. In the current neural network model, parameters of the algorithmic model are adjusted through learning. The initial definition of the COCOMO II model and its results are provided in [2]. The initial COCOMO II model had the following form:

$$Effort = A \times [Size]^{1.01 + \sum_{i=1}^5 SF_i} \times \prod_{i=1}^{17} EM_i \quad (1)$$

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)”.

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

The above equation has the form of the model below:

$$O_{est} = \underbrace{[b_1 + u_1 * I_1 + u_2 * I_2 + \dots + u_{17} * I_{17}] + [b_2 + I_{18} + \dots + I_{22}] * [v_1 + \ln(size)]}_{\text{Part 2}} \quad (3)$$

where

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

$$I_1 = \ln(EM_1) ; \dots ; I_{17} = \ln(EM_{17});$$

$$I_{18} = SF_1 ; \dots ; I_{22} = SF_5;$$

b₁ and b₂ 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 the 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.

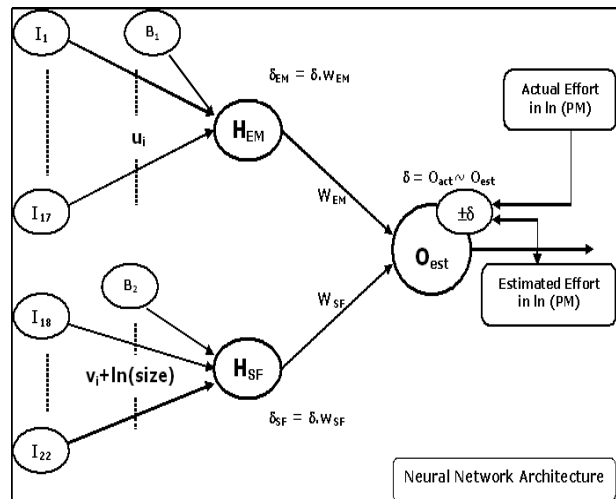


Figure 1. Architecture of the proposed neural network model

Note that in the above network, all the input values I₁ to I₂₂ 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 B₁. 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 B₂. 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 $\ln(\text{PM})$ 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 η ($0 < \eta \leq 1$)
- 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
 $H_{EM} = b_1 + \sum I_i * u_i$ for $i = 1$ to 17
 $H_{SF} = b_2 + \sum I_i * (v_i + \ln(\text{size}))$ for $i = 18$ to 22
 $O_{\text{est}} = H_{EM} * w_{EM} + H_{SF} * w_{SF}$
- Step 6 : Update the weights between hidden and output layer.
 $w_{EM}(\text{new}) = w_{EM}(\text{old}) + \eta * \delta * H_{EM}$

$$w_{SF}(\text{new}) = w_{SF}(\text{old}) + \eta * \delta * H_{SF}$$

The error is calculated as $\delta = (O_{\text{act}} - O_{\text{est}})$

- Step 7 : Update the weights and bias between input and hidden layers.

$$u_i(\text{new}) = u_i(\text{old}) + \eta * \delta_{EM} * I_i \text{ for } i = 1 \text{ to } 17$$

$$v_i(\text{new}) = v_i(\text{old}) + \eta * \delta_{SF} * I_i \text{ for } i = 18 \text{ to } 22$$

$$b_1(\text{new}) = b_1(\text{old}) + \eta * \delta_{EM}$$

$$b_2(\text{new}) = b_2(\text{old}) + \eta * \delta_{SF}$$

The error is calculated as

$$\delta_{EM} = \delta * w_{EM} ; \delta_{SF} = \delta * w_{SF} ;$$

- 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 η 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. EVALUATION CRITERIA AND RESULTS

This section presents the methodology used in conducting experiments and discusses the results obtain when applying our proposed neural network to the COCOMO dataset. The analysis undertaken in this study and the dataset used in this work are from COCOMO database, a dataset publicly available which consists of 63 projects [3]. To get a more realistic estimate of the accuracy of prediction we followed the similar procedure as in [14]. Based on this process, we have divided the entire dataset into two sets, training set and validation set in the ratio of 80%: 20%. Training set consists of 50

projects selected randomly and validation set consists of remaining 13 projects. All the data is normalized to fit in the natural logarithmic space. The calculations were made using a software prototype developed in java language.

The proposed neural network is implemented with 22 input nodes, two hidden layers and one output node and the network is trained with training data and tested with the validation data. The evaluation consists in comparing the accuracy of the estimated effort with the actual effort. The following evaluation criterion is used to assess and compare the performance of the present neural network model. A common criterion for the evaluation of cost estimation model is the magnitude of relative error (MRE), and mean magnitude of relative error (MMRE) [23]. MRE is defined as in “(4)”.

$$MRE = \frac{|ActualEffort - EstimatedEffort|}{ActualEffort} \times 100 \quad (4)$$

and Mean Magnitude of Relative Error (MMRE) computes the average of MRE over N projects. Generally, the acceptable target value for MMRE is 25%. It indicates that the MRE of each project for the established estimation models should be less than 25% on the average. In evaluating the trained network model presented in this study, the MRE values are calculated using “(4),” for each project in the validation set and compared with that of the COCOMO Model. For example, the relative error calculated for Project ID 61 for COCOMO and the proposed model is 13.1 and 12.5 respectively. In the case of Project ID 47 it is 16.4 and 10.06. The mean magnitude of relative error for the entire validation dataset is 13.32, 7.68 respectively. Table 1 summarizes these test results for each project in the validation dataset.

TABLE I.
COMPARISON OF EFFORT ESTIMATION RESULTS IN MRE

S.No.	Test Set Project ID	MRE (%) using COCOMO	MRE (%) using Proposed Model
1	61	13.1	12.5
2	50	3.08	1.98
3	59	8.66	4.92
4	62	6.22	9.73
5	47	16.4	10.06
6	63	19.95	12.84
7	38	50.98	15.34
8	45	5.35	4.59
9	40	12.4	11.1
10	30	6.49	3.24
11	5	7.44	5.08
12	12	19.83	6.8
13	29	3.29	1.63
MMRE (%)		13.32	7.68

“Fig. 2” shows the effort prediction accuracy of the neural network. MRE values were plotted for COCOMO and the proposed model that are aligned with each project in the validation dataset. Test set projects were taken along with x-axis and MRE values were represented along with y-axis. The chart clearly shows that there is a decrement in the relative error, so that the proposed model is more suitable for effort estimation. The preliminary results obtained suggest that the proposed architecture can be replicated for accurately forecasting the software development effort. The aim of this study is to improve the estimation accuracy of COCOMO model, so that the estimated effort is more close to the actual effort.

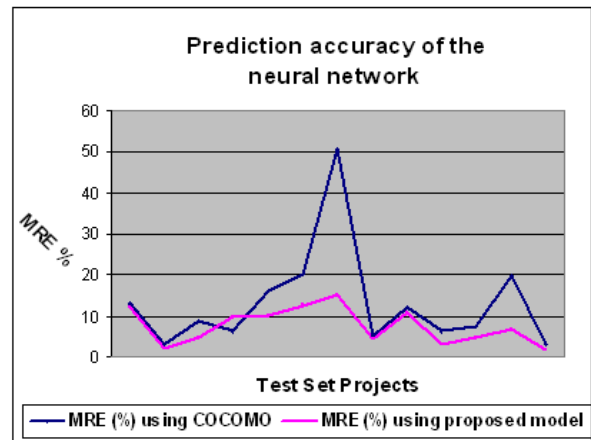


Figure 2. Chart showing the effort prediction accuracy of the neural network

V. CONCLUSIONS AND FUTURE WORK

A reliable and accurate estimate of software development effort has always been a challenge for both the software industrial and academic communities. There are several software effort forecasting models that can be used in forecasting future software development effort. We have constructed a cost estimation model based on artificial neural networks. Our idea consists in the use of a model that maps COCOMO model to a neural network with minimal number of layers and nodes to increase the performance of the network. The neural network that we have used to predict the software development effort is the multi layer feed forward neural network with the identity function at input, hidden and output units. The back propagation algorithm is used to train the network. We have used full COCOMO dataset to train and to test the network and it was observed that our neural network model provided significantly better cost estimations than the estimation done using COCOMO model.

Accordingly, we are convinced that the rationalization and interpretation of the knowledge stored in the architecture and synapse weights of the neural nets is very important to gain practitioners acceptance. Another great advantage of this work is that we could put together expert knowledge, project data and the traditional algorithmic model into one general framework

that can have a wide range of applicability in software cost estimation. This work can be extended by integrating this approach with fuzzy logic to effectively deal with imprecise and uncertain information associated with COCOMO. Therefore, a promising line of future work is to extend to the neuro-fuzzy approach.

REFERENCES

- [1] Anthony Senyard et al., "Software Engineering Methods for Neural Networks," IEEE Proceedings of the Tenth Asia-Pacific Software Engineering Conference, (APSEC'03), pages 468-477, 2003.
- [2] Boehm B., Clark B., Horowitz E., Madachy R., "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," Annals of Software Engineering, 1995.
- [3] Boehm, B.W., "Software Engineering Economics," Prentice-Hall, Englewood Cliffs, NJ, USA, 1994.
- [4] Dawson, C.W., "A neural network approach to software projects effort estimation," Transaction: Information and Communication Technologies, Vol. 16, pages 9, 1996.
- [5] D.E Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by Error Propagation," Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Cambridge, MA., MIT Press. Volume 1, 318-362, 1986.
- [6] Finnie, G. R. and Wittig, G. E., "AI Tools for Software Development Effort Estimation," In Proceedings of the IEEE International Conference on Software Engineering: Education and Practice, Washington, DC, pages 346-353, 1996.
- [7] Heiat, A., "Comparison of artificial neural network and regression models for estimating software development effort," Journal of Information and Software Technology, Vol. 44 (15), 911-922. 2002.
- [8] Hughes, R.T., "An evaluation of machine learning techniques for software effort estimation," University of Brighton, 1996.
- [9] Idri, A. Khoshgoftaar, T.M. Abran, A., "Can neural networks be easily interpreted in software cost estimation?," Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZ-IEEE'02, Vol.: 2, 1162-1167, 2002.
- [10] Jørgensen. M., "A Review of Studies on Expert Estimation of Software Development Effort," Journal of Systems and Software, Volume 70, pp. 37-60, 2004.
- [11] Jorgerson, M., "Experience with accuracy of software maintenance task effort prediction models," IEEE Transactions on Software Engineering, Volume 21 (8), 674-681, 1995.
- [12] Karunanithi, N., et al., "Using neural networks in reliability prediction," IEEE Software, pp. 53-59, 1992.
- [13] Kemerer, C., "An empirical validation of software cost estimation models," Communications of the ACM 30 (5), 416-429, 1987.
- [14] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris, "Software productivity and effort prediction with ordinal regression," Journal of Information and Software Technology, Volume 47, Issue 1, Pages 17-29, 2005.
- [15] 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.
- [16] Satish Kumar, B. Ananda Krishna and Prem S. Satsangi, "Fuzzy systems and neural networks in software engineering project management," Journal of Applied Intelligence, Volume 4, pages 31-52, 1994.
- [17] Schofield, C., "Non-algorithmic effort estimation techniques," Technical Report TR98-01, 1998.
- [18] Segee, B. E., "Using spectral techniques for improved performance in ANN," Proceedings of the IEEE International Conference on Neural Networks, pages 500-505, 1993.
- [19] Seluca, C., "An investigation into software effort estimation using a back propagation neural network," M.Sc.Thesis, Bournemouth University, UK, 1995.
- [20] S. Rajasekaran.g.a. Vijayalakshmi Pai, "Neural Networks, Fuzzy Logic, And Genetic Algorithms-Synthesis And Applications," Prentice-Hall of India Pvt.Ltd., PHI, 2004.
- [21] Srinivasan, K., Fisher, D., "Machine learning approaches to estimating software development effort," IEEE Transactions on Software Engineering, Volume 21 (2), 126-137, 1995.
- [22] Tadayon, N., "Neural network approach for software cost estimation," International Conference on Information Technology: Coding and Computing (ITCC 2005), Volume: 2, on page(s): 815- 818, 2005.
- [23] Wieczorek I., Briand L.C., Emam K.E. and Surmann D., "An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques," ISERN-1998-27.
- [24] Wittig, G., Finnie, G., "Estimating software development effort with connectionist models," Journal of Information and Software Technology, Volume 39 (7), 469-476, 1997.
- [25] Zhiwei Xu and T.M.Taghi M. Khoshgoftaar, "Identification of fuzzy models of software cost estimation," Journal of Fuzzy Sets and Systems, Volume 145, Issue 1, Pages 141-163, 2004.

Mr. Ch. Satyananda Reddy obtained his B.E. Degree in Computer Science and M.Tech Degree in Computer Science with specialization in Software Engineering from Jawaharlal Nehru Technological University, Hyderabad, India. He is currently a Research Scholar and working as Assistant Professor in the Department of Computer Science and Systems Engineering at Andhra University College of Engineering, Visakhapatnam, India. His Research interests include Software Engineering, Software Metrics, Software Quality Assurance, Cost Estimation, Neural Networks and Fuzzy Systems.



Dr. KVSUN Raju holds a PhD degree in Computer Science from IIT, Kharagpur, and is presently working as Professor in the department of Computer Science and Systems Engineering at Andhra University Engineering College, Visakhapatnam. His research interests include Software Engineering Data Engineering and Data Security.