Level of Confidence in Software Effort Estimation by an Intelligent Fuzzy - Neuro - Genetic Approach

Poonam Rijwani[1], Dr. Sonal Jain[2]
Research scholar[1], Associate professor[2]
JK Lakshmipat University
Jaipur, Rajasthan

***Abstract*—Organizations are struggling to deliver the expected software functionality and quality in scheduled time and prescribed budget. Despite availability of numerous advanced effort estimation techniques overestimation and underestimation occur on a vast scale and results in project failures and significant loss to the organization. The paper proposes machine learning based approach to calculate the optimized effort and level of confidence. Genetically trained neural network evaluates the optimum effort for given COCOMO II variables. The level of confidence is evaluated by fuzzy logic and indicates the percentage that the predicted effort will not exceed the limits.**

Keywords*—COCOMO II; artificial neural networks; genetic algorithm; fuzzy logic

I. Introduction

Human dependency on software is increasing continuously. Today most of the goods and services are realized with software systems. Software has become major driving force for progress even in domains that were traditionally reserved as completely mechanical or hardware systems, for instance major advances in automobile industry are being realized with software development. Companies spend 4-5 percentage of the revenue on software development [1]. The figure is as high as 10 percent in highly IT dependent sectors, for instance telecommunications and finance [1].Thus, functionality and complexity of software systems is increasing manifold. Simultaneously time to market and cost should be reduced to stay competitive. In United States 250 billion dollars are spent each year on IT development [2]. Project management and Effort estimation are key factors for success of a software project. Despite much research and technological advancement in effort estimation techniques, proportion of failed software projects is huge[2,3].According to the Chaos report submitted by Standish Group[2] only 16.2 percent of the projects are successful ,57.2 percent projects are over-budget and provides lesser functionalities than specified and 31.1 percent of the projects are cancelled during their development cycle. The percentage of the successful, challenged (over-budget with less functionality) and impaired (cancelled) projects is shown in Fig 1.

Project failure can be defined as combination of cost overruns, late deliveries, poor quality, and/or developing a product that does not get used. The two crucial reasons for failure of most software projects are Overestimation and Underestimation of the software effort [4]. Most projects either cost more than they return or fail to deliver required projects in the expected time. Both the scenarios lead to huge loss or may also result in termination of the whole project. R. Charette [3] suggested unrealistic project goals and inaccurate estimates of needed resources as principal factors that lead to project failure. A. Trendowicz and etal [2] pointed that most of the effort estimation techniques provides point estimates with hardly any support for risk management if project overruns the expected cost. Moses and etal[4] in their research concluded that in addition to estimates the effort estimation should also specify a Level Of Confidence associated with the calculated effort in order to compensate the uncertainty associated with effort estimation.

II. Related Work

A. What research has been Conducted so far?

For software effort estimation, numerous methods have been examined specifically data driven soft computing methods such as artificial neural networks, regression trees, evolutionary computing, rule-based induction, fuzzy logics etc. These methods exhibit many advantages like regression over other standard methodologies. Literature of software effort estimation endorse that important product feature characteristically reflects the software size which exactly impact efforts. Basically it is used to build cost models.

Initially, almost all models are based on the size of metrics which contemplates numerous coding lines coded for a software project i.e. lines of code (LOC) or thousands of source line code (KLOC), as shown in COCOMO [5], or function points (FP) which is there in models like Albrecht's FP i.e. Function Point Analysis [6].

Many researchers analyze the feasibility of evolving software effort estimation methods exhausting various methodology, parameters, datasets, etc. In the comparative analysis study given by [7], amalgamation of estimation methods may generate more reliable, accurate cost estimation for software development as it is displayed that no method is good or bad in all the situations. Review papers given by [8] [9] grasp a complete description of such studies. In review paper [10], the effort estimation was assessed by back propagation Artificial Neural Networks on datasets such as Desharnais and ASMA, generally through system size to determine the correlation of size with effort.

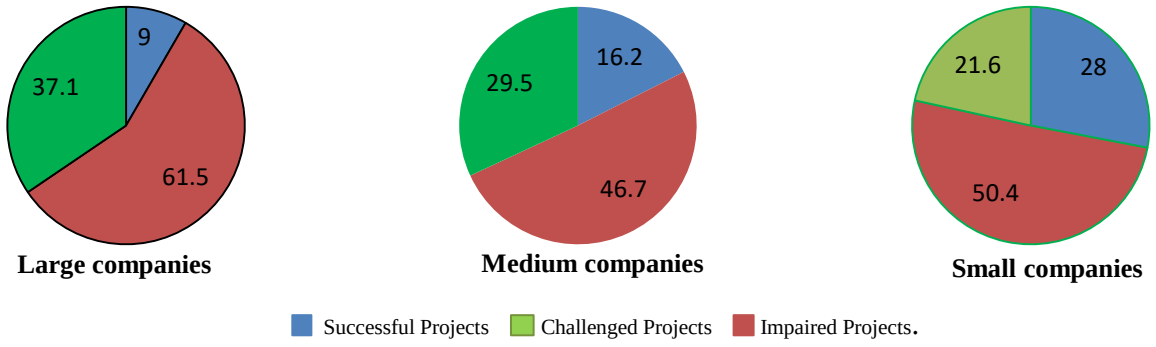

Fig. 1. Percentage of successful, challenged and impaired projects in large, medium and small organizations

The method produced reassuring estimates represent that the model need an additional methodical advance method to develop the architecture and parametric settings in order to achieve improved outcomes. In paper [11], the effort estimation evaluation of the relationship between effort and size analyzed using Genetic Programming technique in which advanced tree composition displaying number of power, linear, quadratic type common equations. The methodology lengthened to appropriate desirable stages of prediction correctness through hardly the size attribute but also approved to achieve additional enhancements.

Kumar et al. proposed a model exhausting Particle Swarm Optimization (PSO) for tuning the elements of primary COCOMO model to compute the effort accurately considering hardly KLOC factor [12].

Finnie et al. [13] conferred a comparison of statistical regression based model with other artificial Intelligence based estimation models for evaluation of software development effort. The researcher establish that statistical regression model underperformed for difficult and complex software projects as the Artificial Intelligence based models gives satisfying evaluation results. They studied dataset amidst Projects from 17 organization and Desharnais. As an estimation criterion MMRE was used. In 2002 another researcher Heiat [14] examined Feed Forward Neural Networks with function point and Radial Basis Neural Network with Source Lines of Codes for various datasets including projects of different generation languages. The results embodied that artificial neural network method is prudent with regression though a third generation language data set is used. P.Rijwani et-al [15] used hit & try method to determine best network architecture, in an experiment for the training network using back propagation. For software effort estimation a FLANN was proposed by Trimula Rao [16] which generates effort and hence processes the final layer output. It has a drawback that the relation between input and output is not equitable.

Investigation on back propagation Artificial Neural Network of 2-2-1 design based on dataset of NASA that includes eighteen projects. The inputs were development methodology and KDLOC and output was effort.

Attarzadeh [17] in which 17 cost drivers and 5 Scale factors were used as inputs. Sigmoid activation function is utilized while creating the network to achieve post architecture of COCOMOII model. The COCOMO algorithm is compared using Pred (0.25) and the results are shown in terms of MMRE. An innovative software development for effort estimation was proposed by Attarzadeh [24], exhausting neural networks, in which weights of the network were adjusted in such that it resulted in COCOMO II model. The neural network method suggested gives better result when related to COCOMO model after proper training. Even though back-propagation for neural networks is focused, complexity arises for adjusting weight and bias net parameters during training like slow convergence, sensitive to arbitrary initial weights, entering local minima, difficulty in selecting explicit optimum network configuration. The preeminent efficiency of genetically trained neural networks is mentioned in various researches. For instance After a series of experiments and simulations Shukla[22] concluded that the genetically trained neural networks outperforms back propagation trained and quick propagation trained neural networks in software effort estimation. The recommended model evaluates software development effort in the function of seventeen cost drivers and five scale factors. Thus it is evident that over the years improving the accuracy of software effort estimation has remained main concern of research with little attention being paid to quantifying the uncertainty involved in effort estimation techniques.

B. Loopholes in Existing Research

Many systematic surveys have been conducted on software effort estimation. Moløkken and Jorgensen [30] provide an exhaustive review of surveys in software effort estimation. They concluded that most of the projects (60-80 percent) suffer from effort and/or schedule overruns. However the percentage overrun (30-40%) is significantly lower than suggested by Chaos report by Standish Group (80%). Jorgensen and Sheppard [29] identified and reviewed 304 research papers published in 76 journals. They found that majority of the research in software effort estimation has remained concentrated on effort estimation methods and less research is done on uncertainty assessments, data set properties and measures of estimation performance. The authors also

proposed that most of the research relies on historical dataset for evaluation and validation of effort estimation models, only a few provide real life evaluations. The different estimation methods used in industries are shown in Fig 2.Presently Expert judgment is the prominent estimation method used by organizations [26, 40].The reason is it cannot be ascertained if formal methods are better or weaker than expert judgment [27]. Identification the human factors affecting effort estimation [28] and development of practical guidelines are crucial to get benefitted from expert judgment. Applications of software effort estimation is summarized in Fig 3.

Fig 4 shows the number of papers published from the year 1996 to 2016 related to software effort estimation J.Moses[4] suggested that Effort estimation using algorithmic models, statistical prediction systems or machine learning approaches, e.g. Case Based Reasoning (CBR), Artificial Neural Networks (ANN) or Rule Induction (RL), under perform when compared to subjective estimation given by human estimators. The author also added that the prediction methods does not provides any support for decision making in case if the actual effort is greater than or less than the predicted effort. The author developed an approach by using Bayesian Inference to improve effort estimation consistency. B. Clark and etal[20] developed a multiple regression based effort calibration strategy. D.Yang and etal [21] proffered that software industry suffers from frequent cost overruns, and the software cost estimation remains a challenging issue. The authors developed a model for accounting the uncertainty based on Bayesian belief networks.

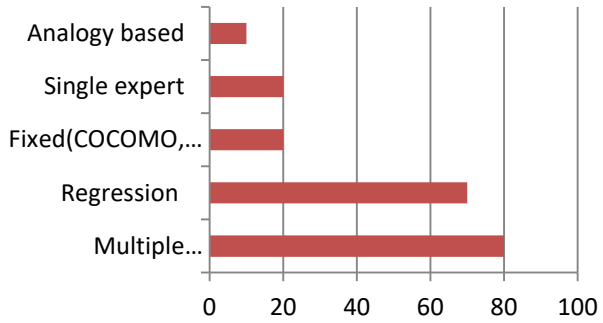

Fig. 2. Effort Estimation techniques used in industries

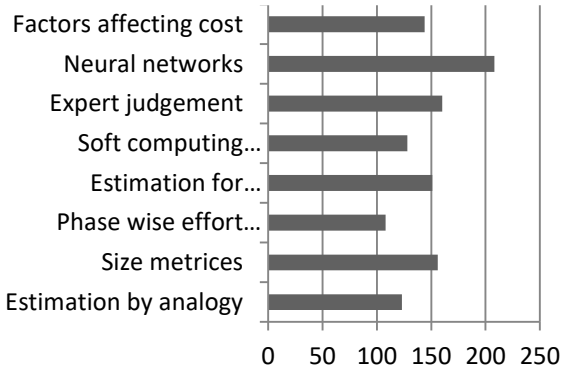

Fig. 3. [25]Papers published over past years about different estimation techniques.

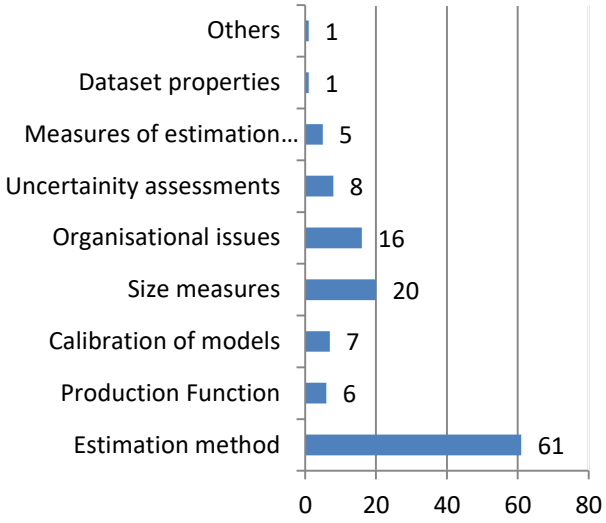

Fig. 4. Papers published over past years about different estimation techniques

III. Implementation Details

The block diagram of the adopted methodology is shown in Fig4. Genetic Algorithm is used for improving the output of neural network. Fuzzy inference and genetically trained neural networks are employed independently to evaluate level of confidence and optimized effort. The main form or GUI constructed in MATLAB is shown in Fig 5. . The GUI prompts user to input COCOMO II variables. The user can click on the optimized effort button on the calculate panel for the GANN [3] predicted effort in person months. Similarly level of confidence button is pressed to obtain the probability that the effort will not exceed the specified limits.

A. Dataset Generation

For the present work COCOMO 81 dataset available on PROMISE repository [18] was converted to COCOMO II dataset using the tool Rosetta stone [33].The tool is developed at IBM research in order to make COCOMO estimates functional with COCOMO II model. The output is development effort measured in man-months. The above mentioned COCOMO 81 dataset was established from exploration of sixty three developed software projects.

B. Network Topology

The model is created with one hidden layer in MLF() neural network .One hidden layer with arbitrary units is sufficient for "Universal Approximation Property"[23].There is no rule of thumb for determining number of hidden units to be used, as it depends upon critical factors such as number of training cases and complexity of classification and learning. A convenient way is to try many different networks, calculate the generalization error for each network and select the network with minimum generalization error [23].Following the above rule, we performed a number of hit and trial experiments from 2 to 20 nodes.

The optimum topology was found to be 23-10-1 i.e. 23 input nodes, 10 nodes in hidden layer and 1 node layer (Fig 5).

The 23 neurons in input layer correspond to the 23 input variables of COCOMO II model (kloc, 5 Scale Factors, 17 Effort Multipliers). The node in output layer represents the optimized effort. The input to neural network as scale factors and effort multipliers is be represented by binary vectors$x_j (j \in [1,23])$. The COCOMO II variables are converted into binary vectors by range normalization such that such that $x_j (x_j \in [0,1])$ The vector O represents output obtained in person-months.w_i And w_{ji} are weight parameters or synaptic strength connecting hidden layer to output layer and input layer respectively.

$$O\left(\sum_{j=1}^{23} x_j\right) = f\left(\sum_{i=1}^{10} w_i\, y_i\right) \quad (1)$$

$$y_i = f\left(\sum_{j=1}^{23} w_{ij} x_j\right) \quad (2)$$

C. Training The Network

Genetic algorithm is employed for training the neural network. Fig 6 represents the algorithm of training neural network by genetic algorithm. The preeminent efficiency of genetically trained neural networks is mentioned in various researches. For instance After a series of experiments and simulations Shukla [22] concluded that the genetically trained neural networks outperforms back propagation trained and quick propagation trained neural networks in software effort estimation The suitable values for control parameters of genetic algorithm have been found by running various simulations and have been listed in Table 1. We have used binary string chromosomes. Six features (very low- vl, low-l, nominal-n, high-h,very high-vh, extra high-xh) are considered for each cost driver and subsequent weights are encoded with 3 bits(0-n,1-vl,2-l,3-n,4-h,5- vh,6-xh,7-n).0 and 7 are assumed default values. Fitness function is reciprocal of MMRE as genetic algorithm maximizes the fitness function and a low Roulette Wheel selection:

Following are the steps for Roulette Wheel selection:

1) Evaluate the sum of the fitness value of all individuals in given population...

2) Calculate probability of selection of a particular individual by dividing its chromosome's fitness by the total fitness values of the population.

3) Divide the roulette wheel into sectors based on probabilities calculated in the second step.

4) Spin the wheel 'n' number of times. The individual corresponding to the sector pointed by the pointer is selected.

The probability that an individual is selected from a population of n individuals is given by equation, where is fitness value of element.

$$MMRE = \frac{1}{N}\sum_{J=1}^{N} \frac{EstEff_j - ActEff_j}{ActEff_j} \quad (3)$$

$$Fitness\ function = \frac{1}{MMRE} \quad (4)$$

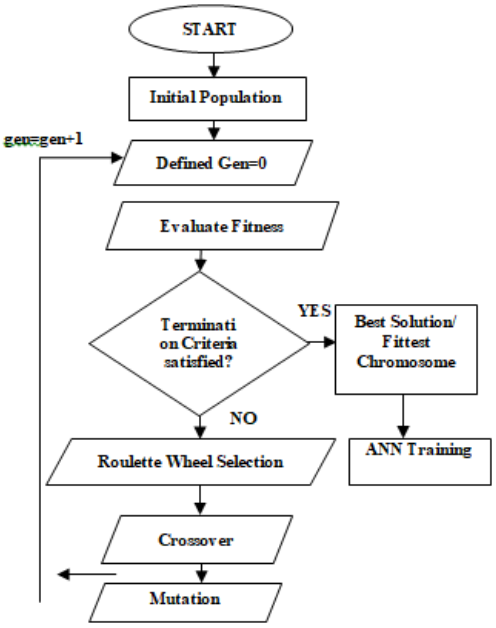

Fig. 5. Genetically trained neural network.

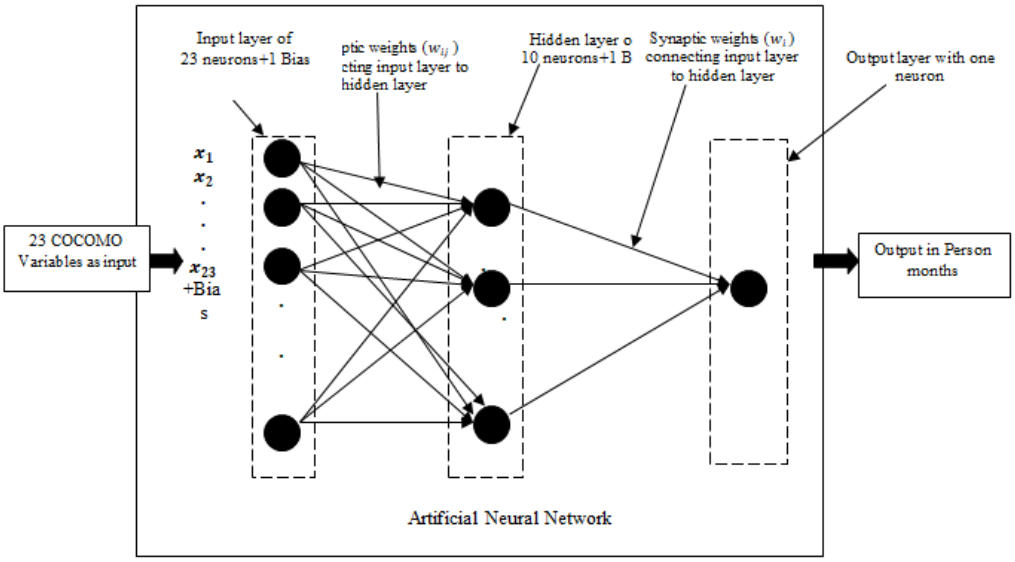

Fig. 6. Network Topology

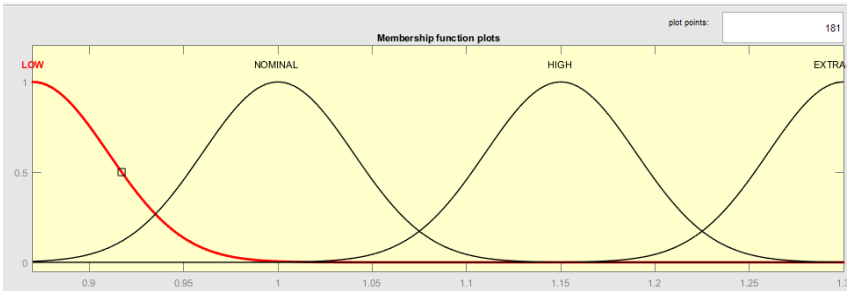

Fig. 7. Membership functions for STOR

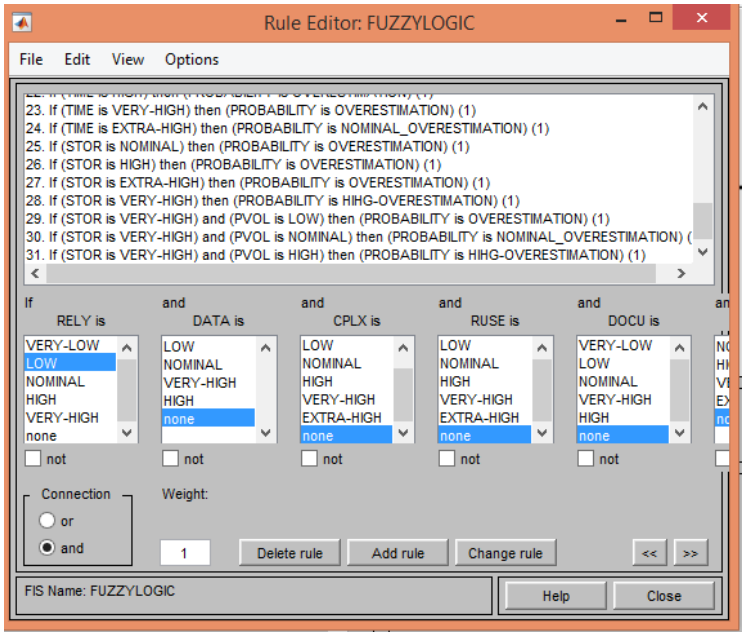

Fig. 8. Fuzzy rules Fuzzy Inference

Fuzzy logic is close to human interpretation of truth. This property can be utilized in effort estimation practices to balance the inherent imprecision with uncertainty to determine the level of confidence. The level of confidence indicates the uncertainty or the probability of overestimation or underestimation. The uncertainty can not only be used to improve the estimation consistency but can also be used in making statements to client indicating the chances of overestimation or underestimation.

Literature reveals that there are numerous approaches for incorporating fuzzy logic in effort estimation models. However available research on organization specific effort dependencies is scarce. For instance a company 'A' is facing frequent underestimation on its past projects. It is possible that the value of COCOMO variables used by them make it inclined to Underestimation. Suppose the assigned value of Programmer capability is high but due to some discrepancies the actual programmer capability is lower than expected. Such conditions can occur frequently while calculating software effort by formal methods because of following reasons:

1) A huge amount of information is required in the starting phase.

2) Error due to human factor.

The developed fuzzy inference system calculates probability of overestimation. This acts as a warning system for effort estimators and prompts them to review the process and/or set appropriate risk factors (Table2).

Dataset preparation: The historic dataset is converted into probability distribution functions of 23 input variables. In the proposed model conditional probability is used as basis for forming fuzzy rules. By probability theory conditional probability is defined as probability of occurrence of an event (A) by assertion that another event (B) has already occurred. The event A is hypothesis and the event B is observed evidence. It is expressed mathematically by equation

$$\mathbf{P(A/B) = \frac{P(A \cap B)}{P(B)} \quad , P(B) \neq 0} \qquad (5)$$

Here A denotes occurrences of overestimation and B denotes instances of values of input variables. For instance P (O/STOR=1) denotes the probability of overestimation provided that the selected value of STOR is nominal. The model is based on chances of overestimation as underestimation and overestimation are mutually exclusive events. Thus, if probability of one is known the probability of other can be calculated easily.

$$P(A \cup B) = 1 \qquad (6)$$

A and B are mutually exclusive events

The conditional probability for every value of each input variable is calculated and subsequently mapped into linguistic fuzzy rules. The developed fuzzy model uses 76 fuzzy rules to calculate output.

Input and Output Variables: In this next step the range of 23 input variables as effort multipliers scale factors and lines of code is defined. The output variable is probability of overestimation and it lies between 0 and 1.

Membership Functions: The membership functions are defined for each input as well as output variables. In our analysis we have considered Gaussian membership functions as it demonstrated by Kushwaha and Suryakane [31] that Gaussian membership function smoother transition in its intervals, and the achieved results were closer to the actual effort. The Gaussian membership function is governed by following equation. The Gaussian membership function for KLOC is given in Table3 [32] and STOR is shown in Fig7.

$$f(x;\sigma,\mu) = \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \quad (7)$$

Fuzzy Rules: Fig 8shows the subsequent fuzzy rules.

IV. Results And Discussions

In our project historical dataset of 63 projects is considered. The estimated effort and Mean Relative Error using COCOMO model, neural network model with back propagation and the genetically trained neural network model is shown in Table4 and Table5. Fig 10 demonstrates the error histogram obtained after training process. The histogram is centered on zero error Thus our selected topology is appropriate. Fig 12 shows the comparison between the two models. Mean square error found is 0.0124682 after 50 generations using 10000 populations. Fig 9 substantiates that the genetically trained neural network model outperforms the COCOMO model as well as BPNN by significant difference. The data of BPNN is taken from the authors' past research [19].

TABLE I. Genetic Algorithm Parameters

SNO	Control parameter	Value
1	Population Size	10000
2	Elite Count	4
3	Crossover	0.8
4	Generations	50
5	Initial Population	10000*251 double
6	Selection	Roulette Wheel
7	Crossover	Heuristic
8	Number of variables	251
9	Mutation	0.01

TABLE II. Risk Factor

Probability (Fuzzy Output)	Overestimation	Risk
0-0.3	Very low	low
0.3-0.5	low	Moderate
0.5-0.7	High	High
0.7-1	Very High	Very High

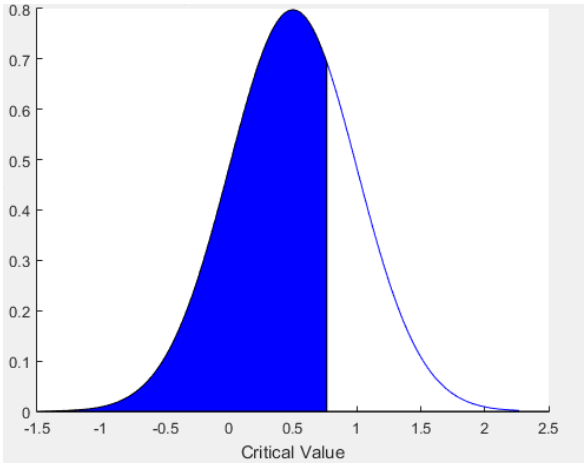

Fig. 9. Fuzzy Output.

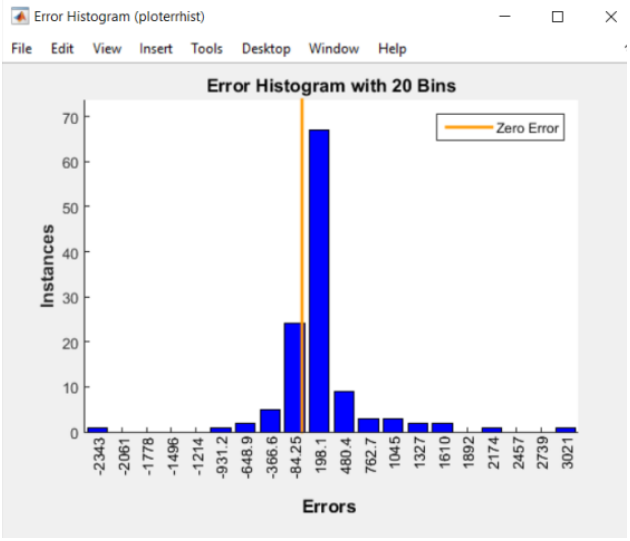

Fig. 10. Error Histogram.

TABLE III. MRE Comparison

KLOC	Value
0-50	small
50.1-128	Moderate
128.1-512	High
512.1-up	Very high

TABLE IV. MRE Comparison

SNO	Control parameter	Value
1	Population Size	10000
2	Elite Count	4
3	Crossover	0.8
4	Generations	50
5	Initial Population	10000*251 double
6	Selection	Roulette Wheel
7	Crossover	Heuristic
8	Number of variables	251
9	Mutation	0.01

TABLE V. EFFORT COMPARISON

SNO	Actual Effort	Effort using COCOMO model	Effort using Hybrid model (GANN)
1	117.6	180.8134786	127.4616
2	117.6	168.9821569	124.3451
3	31.2	43.65971251	34.8585
4	36	37.9393787	37.9985
5	25.2	49.47453739	32.8746
6	8.4	10.38903049	9.4435
7	10.8	19.08279263	14.8734
8	352.8	449.5306924	399.7262
9	72	45.9777269	68.6443
10	72	287.291445	112.9243
11	24	14.08522464	20.6532
12	360	287.291445	381.3423
13	36	50.44249685	41.9123
14	215	686.648327	300.3533
15	48	51.69199249	50.5451
16	360	615.4091318	401.2176
17	324	670.4677889	400.2432
18	60	162.0906366	80.6453
19	48	51.69199249	50.4434
20	60	207.7411868	112.5542

TABLE VI. MRE COMPARISON

SNO	Actual Effort	MRE using COCOMO model	MRE using Hybrid model (GANN)
1	117.6	0.53723	0.08385714
2	117.6	0.436923	0.04735629
3	31.2	0.39935	0.11726962
4	36	0.053872	0.05551389
5	25.2	0.963275	0.30454762
6	8.4	0.326789	0.12422619
7	10.8	0.766925	0.37716667
8	352.8	0.27418	0.13301077
9	72	0.361976	0.04646806
10	72	2.990159	0.56839306
11	24	0.413107	0.13945
12	360	0.201968	0.05928417
13	36	0.40118	0.16423056
14	215	2.193713	0.396999209
15	48	0.076917	0.05302292
16	360	0.70947	0.11449333
17	324	1.069345	0.23531852
18	60	1.701511	0.34408833
19	48	0.076917	0.05090417
20	60	2.462343	0.87590333

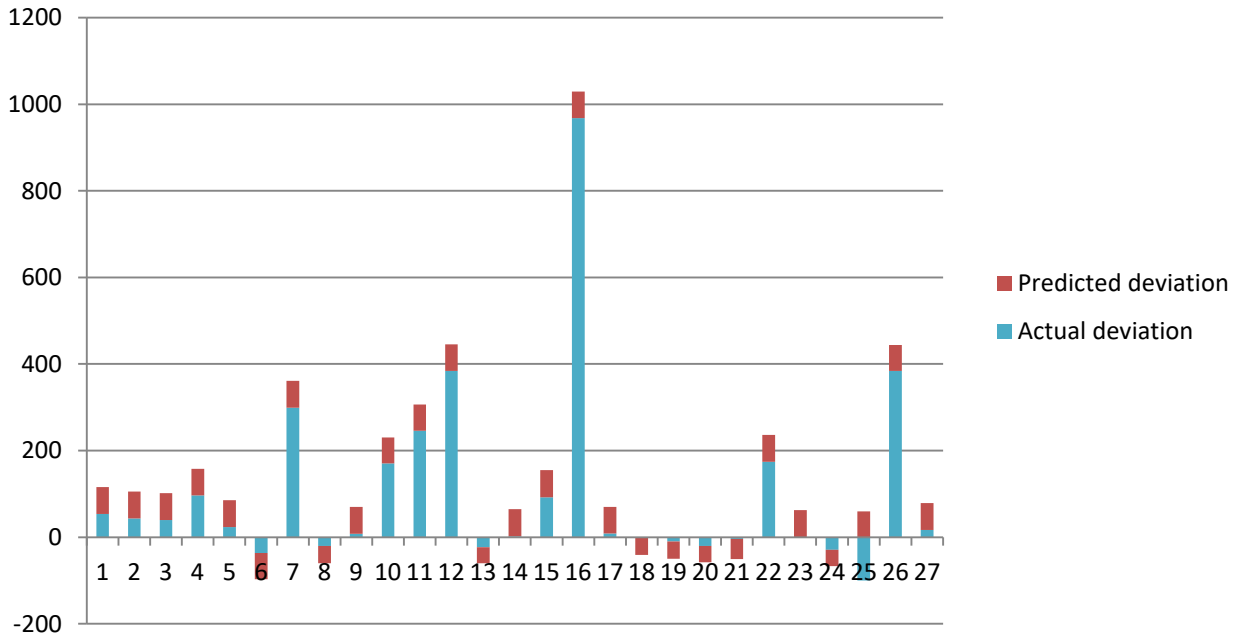

Fig. 11. Actual Deviation v/s Predicted deviation.

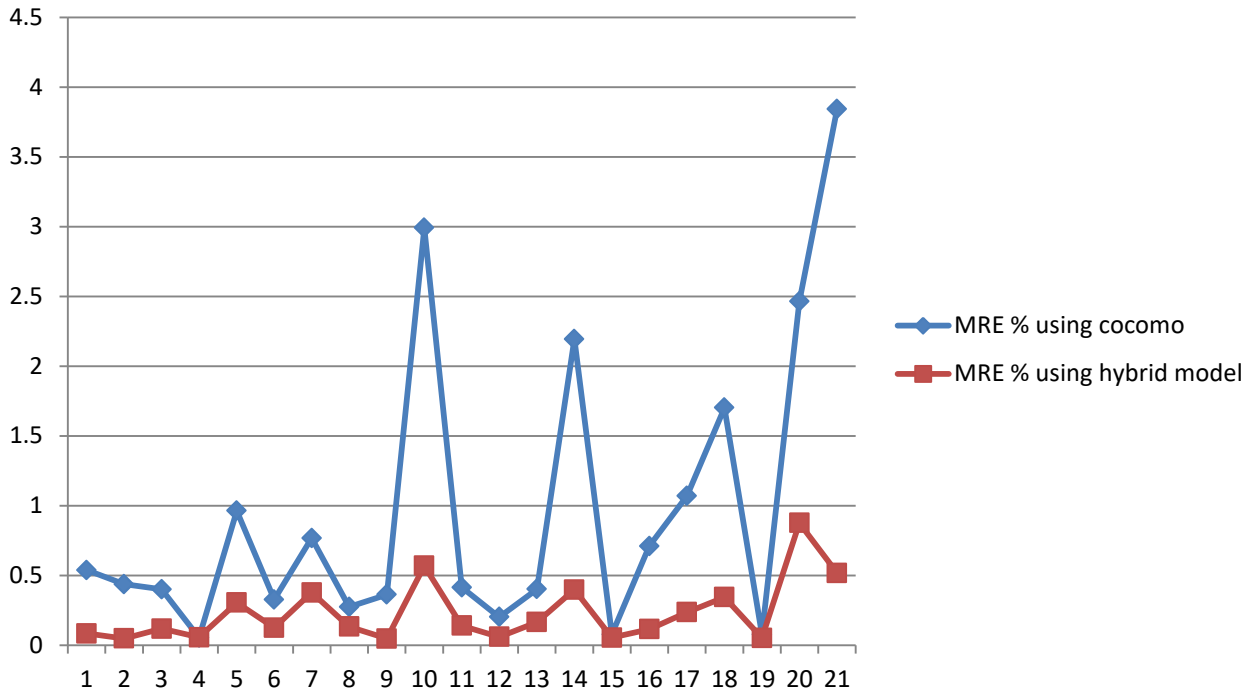

Fig. 12. MRE comparison

Fig 11 demonstrates the actual and predicted deviations from the actual effort. Where actual deviation denotes the actual percentage of overestimation/underestimation and predicted deviation denotes the predicted overestimation/underestimation. The Figure confirms that the developed model is accurate and provides optimistic view of uncertainty by effectively covering all range of deviation. Table 5 presents the computed effort values using COCOMO and proposed model. Subsequently, Table 6 presents the comparison of MRE values when effort is computed using COCOMO and with the hybrid model.

V. Conclusion and Future Scope

The learning exposes that the suggested fuzzy logic based COCOMO II model incapacitates the uncertainty and vagueness in the inputs that is present in the conventional COCOMO and hence increases the accurateness of software effort estimation. By determining additional appropriate fuzzy rule sets and by arraying technologies like type-2 fuzzy improbability can be handled further closely and hence more precise software effort estimation is thinkable.

References

[1] A. Trendowicz, J. Munch, and R. Jeffery, Adam Trendowicz, J¨urgen M¨unch, and Ross Jeffery, Software Engineering Techniques, Springer Heidelberg Dordrecht London NewYork,2008.

[2] The Standish Group: CHAOS Chronicles. Technical report, The Standish Group International, Inc. (2007).

[3] R. N. Charette,Why Software Fails. IEEE Spectrum ,September 2005.

[4] J.Moses., Measuring Effort Estimation Uncertainty to Improve Client Confidence, Software Quality Journal, 10, 135–148, 2002.

[5] Boehm, B.W., Abts, C., Clark, B., Devnani-Chulani, S.: COCOMO II Model Definition Manual. The University of Southern California (1997)

[6] Albrecht, A.J., Gaffney, J.R.: Software Function Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Transactions on Software Engineering 9(6), 639–648 (1983)

[7] Rijwani, P., Jain, S., &Santani, D. (2014). Software Effort Estimation: A comparison based Perspective. *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, *3*(12), 18-29.

[8] Briand, L.C., Wieczorek, I.: Resource Modeling in Software Engineering. Encyclopedia of Software Engineering 2 (2001)

[9] Jorgensen, M., Shepperd, M.: A Systematic Review of Software Development Cost Estimation Studies. Software Engineering, IEEE Transactions on Software Engineering 33(1), 33–53 (2007)

[10] Wittig, G., Finnie, G.: Estimating software development effort with connectionist model. Information and Software Technology 39, 469–476 (1997)

[11] Dolado, J.J.: On the Problem of the Software Cost Function. Information and Software Technology 43(1), 61–72 (2001)

[12] Anil Kumar, C. S. Yadav et al 2012. Parameter tuning of COCOMO Model for software effort estimation using PSO. ICIAICT ISBN 978-93-81583-34-0 pp 99-105

[13] G.R. Finnie and G.E. Wittig, ―A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models,‖ Journal of Systems and Software, vol. 39, pp. 281-289, 1997

[14] Heiat, Abbas. "Comparison of artificial neural network and regression models for estimating software development effort." *Information and software Technology* 44.15 (2002): 911-922.

[15] Rijwani, P., & Jain, S. (2016). Enhanced Software Effort Estimation Using Multi Layered Feed Forward Artificial Neural Network Technique. *Procedia Computer Science*, *89*, 307-312.

[16] B. T. Rao, et al., A novel neural network approach for software cost estimation using Functional Link Artificial Neural Network (FLANN), International Journal of Computer Science and Network Security, (9) (2009), pp. 126-131, 2009.

[17] I. Attarzadeh and S. H. Ow, Proposing a new software cost estimation model based on artificial neural networks, in 2nd International Conference on Computer Engineering and Technology (ICCET), (2010) pp. V3-487-V3- 491.

[18] http://promise.site.uottawa.ca/SERepository/datasets/ cocomo81.arff.

[19] P. Rijwani, and S .Jain, "Comparison and Analysis of Various Artificial Neural Networks for Software Effort Estimation". *International Journal of Advanced Research in Computer Science and Software Engineering*.

[20] Clark, Bradford, Sunita Devnani-Chulani, and Barry Boehm. "Calibrating the COCOMO II post-architecture model." Proceedings of the 20th international conference on Software engineering. IEEE Computer Society, 1998.

[21] Yang, Da, et al. "COCOMO-U: An extension of COCOMO II for cost estimation with uncertainty." *Software Process Workshop*. Springer, Berlin, Heidelberg, 2006..

[22] Hota, H. S., Shukla, R., & Singhai, S. (2015). Predicting Software Development Effort Using Tuned Artificial Neural Network. In *Computational Intelligence in Data Mining-Volume 3* (pp. 195-203). Springer, New Delhi.

[23] C. S. Reddy and K. V. S. V. N. Raju, "An optimal neural network model for software effort estimation," International Journal of Software Engineering, vol. 3, no. 1, pp. 63–78, 2010.

[24] I. Attarzadeh, et al., Proposing an Enhanced Artificial Neural Network Prediction Model to Improve the Accuracy in Software Effort Estimation, in Fourth International Conference on,Computational Intelligence, Communication Systems and Networks (CICSyN), (2012), pp. 167-172.

[25] S. K. Sehra, Y. S. Brar, N. Kaur and S. S. Sehra, Research Patterns and Trends in Software Effort Estimation, INFSOF 5836.

[26] M. Jørgensen, A review of studies on expert estimation of software development effort, Journal of Systems and Software 70 (1) (2004) 37–60. doi:10.1016/ S0164-1212(02)00156-5.

[27] M. Jørgensen, T. M. Gruschke, The impact of lessonslearned sessions on effort estimation and uncertainty assessments, IEEE Transanctions on Software Engineering 35 (3) (2009) 368–383. doi:10.1109/TSE. 2009.2.

[28] A. Magazinius, S. B¨orjesson, R. Feldt, Investigating intentional distortions in software cost estimation - an exploratory study, Journal of Systems and Software 85 (8) (2012) 1770–1781. doi:10.1016/j.jss.2012. 03.026.

[29] Magne Jørgensen and Martin Shepperd, A Systematic Review of Software Development Cost Estimation Studies, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 1, JANUARY 2007.

[30] Kjetil Moløkken and Magne Jørgensen, A Review of Surveys on Software Effort Estimation, Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03) 0-7695-2002-2/03.

[31] N.Kushwahal and Suryakane, Software Cost Estimation using the Improved Fuzzy Logic Framework, 978-1-4799-3064-7/14,IEEE.

[32] E. Manalif , L.F. Capretz, and D. Ho, Fuzzy Rules for Risk Assessment and Contingency Estimation within COCOMO Software Project Planning Model, DOI: 10.4018/978-1-4666-4785-5.ch006.

[33] Reifer D.J., Boehm B.W. and Chulani S., "The Rosetta Stone: Making COCOMO 81 files work with COCOMO II", University of South California.