A Genetic Neural Network based Application for Improving Software Development Effort Estimation

Poonam Rijwani, Sonal Jain
Research Scholar, Associate Professor
CSE Department,
JK Lakshmipat University, Jaipur, India

Abstract: Software Project effort estimation provides the estimation of various resources, effort and time vitally required to complete a project. Accurate estimation of effort is significant for developers and clients. The present study proposes to develop a machine learning based model to predict Software Project effort estimation. Paper attempts to evaluate the training of Artificial Neural Network using population based Genetic Algorithm. The code is generated in MATLAB for training artificial neural network of 22 input features of COCOMO dataset and single target feature. The COCOMO I, NASA98 data sets; and four project data from a software company were used in the evaluation of the proposed neuro Genetic COCOMO II (GANN-COCOMO II). The assessment of the attained results, using Mean of Magnitude of Relative Error (MMRE) and PRED(25%) evaluation methods, indicate that the GANN-COCOMO II produced the MMRE less than the original COCOMO and the value of PRED(25%) in the GANN-COCOMO II is higher than the original COCOMO. In addition, the GANN-COCOMO II showed improvement in terms of estimation accuracy using MMRE when compared with the original COCOMO.

IndexTerms - COCOMO, Machine Learning, Software Effort Estimation, Artificial Neural Network, Back Propagation, Genetic Algorithm, Optimization.

I. INTRODUCTION

Recently, the developments of large scale software products gain an emergent interest. For any industry to stay in the market competition, handling a sense of balance amongst quality and cost of software is extremely significant. Nowadays software industry has become competitive and software developers struggle for the reasons like delivering the product on time at the specified cost ensuring anticipated quality. This aids in understanding the significance of predicting effort, schedule in initial phases of development of the software. Early estimation and accurate prediction permits the significance of diminishing project risks and then helps to maintain the thoughtful decisions in software development. Overestimation of the software effort can affect in dropping the chance to win a bid and underestimation can lead to unfavorable effect on the quality of software or economical loss.

The manifesto from the literature survey reveals that several effort models have already been developed, popular ones are an early model developed by Putnam acknowledged as SLIM [1] in the year 1978. Later in the year 1981, another cost estimation model named COCOMO 81 (CONstructive Cost MOdel) given by Boehm [2] [3]. Numerous other algorithmic and parametric models are also been suggested and recommended like function point analysis [4] and Use case point [5]. These various techniques and models are based on applying statistical regression based techniques to historic projects data. Among all algorithmic empirical techniques, COCOMO (constructive cost model) is widely accepted one for the reason that its effortlessness and ease in approximating the effort in terms of person months intended for a development project at diverse phases [2]. But all these empirical methods have a shortcoming of inability in handling the vagueness and inaccuracy linked with the various attributes of the projects. The preciseness of the output of these models rests on precision in the input parameters. It is actual challenging to attain such precise inputs at the early stages of software project. Such concerns generate the need to introduce diverse innovative software cost prediction procedures expending neural networks and evolutionary techniques.

Several researchers and software investigators are incessantly functioning on developing novel machine learning-based approaches as an enhancement to empirical algorithmic methods to accomplish improved estimations. Because of the capability to learn and simulate the complex nonlinear connections, ANNs are competent in achieving great success in effort estimation. Artificial Neural Network denotes the computing systems inspired with the analogy of biological neural networks and is explained in depth in the literature [27]. In our paper, we put forward to develop an effort estimation model consistent with Boehm’s COCOMO II model using the technique of neural networks which has the capability to relate very complex input and output datasets, which were otherwise difficult to predict using empirical calculations. ANN is trained to find the optimum set of weights and biases. In this work, the neuro genetic approach is implemented for significant improvement in prediction accuracy.
The paper is systematized as the following order: Section II presents COCOMOII. Section III discusses the related literature work. Section IV presents the proposed GANN-COCOMO system. Section V gives the Evaluation criteria and experimentation planning. Section VI presents dataset preparation for training the model section VII presents results and finally section VIII gives conclusion of the paper and familiarizes the future research work.

II. COCOMO II EMPIRICAL MODEL

Originally, the COCOMO model is given by Barry Boehm in 1981 [2]. It is implemented after various investigations on 63 software projects. This empirical model provides effort in terms of cost and schedule for a development of a software project. In late 1990’s, Boehm offered a model named COCOMO II [3] to accommodate the environmental changes in the software industry. The purpose of COCOMO model is to direct effort using software size and a sequence of cost and scale factors, as given in the equation below:

\[ \text{Effort} = A \times [\text{size}]^{1.01+\sum_{i=1}^{5} SF_i} \times \prod_{i=1}^{17} EM_i \]

Here, A is the multiplicative constant, Software project Size is calculated in terms of KSLOC, SF, is the 5 Scale Factors that are used in the Post-Architectural Model and EM is the 17 Effort Multipliers which are assessed on scales from very low to extra high.

COCOMO II model pronounces seventeen cost drivers and five scale factors considering the Post Architecture model. They have an exponential effect to increase or decrease the extent of development effort. The five scale factors are: Precedentedness (PREC), Development flexibility (FLEX), Architecture/Risk resolution (RESL), Team cohesion (TEAM) and Process maturity (PMAT). The given features are graded using scale of very low, low, nominal, high, very high and extra high. Various effort multipliers or cost drivers are categorized into 4 types as Product factors, Computer factors, Personnel factors and Project factors. Individually each project is described by seventeen cost drivers which are reliability required (RELY), size of the database(DATA), product complexity(CPLX), required reusability(RUSE), documentation(DOCU), execution time constraint(TIME), main storage constraint(STOR), platform volatility(PVOL), analyst competence(ACAP), programmer capability(PCAP), applications experience(AEXP), platform experience(PEXP), language & tool experience(LTEX), personnel stability(PSTA), use of software tools(TOOL), multisite development (SITE) and required development schedule(SCED). The Range is given in Table 1.

<table>
<thead>
<tr>
<th>Effort Multiplier</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required software Reliability (RELY)</td>
<td>0.82-1.26</td>
</tr>
<tr>
<td>Database Size (DATA)</td>
<td>0.90-1.28</td>
</tr>
<tr>
<td>Product Complexity (CPLX)</td>
<td>0.73-1.74</td>
</tr>
<tr>
<td>Developed for reusability(RUSE)</td>
<td>0.95-1.24</td>
</tr>
<tr>
<td>Documentation match to life cycle needs (DOCU)</td>
<td>0.81-1.23</td>
</tr>
<tr>
<td>Execution Time Constraint(TIME)</td>
<td>1.00-1.63</td>
</tr>
<tr>
<td>Main storage constraint(STOR)</td>
<td>1.00-1.46</td>
</tr>
<tr>
<td>Platform Volatility (PVOL)</td>
<td>0.87-1.30</td>
</tr>
<tr>
<td>Analyst Capability (ACAP)</td>
<td>1.42-0.71</td>
</tr>
<tr>
<td>Programmer capability (PCAP)</td>
<td>1.34-0.76</td>
</tr>
<tr>
<td>Personnel Continuity (PCON)</td>
<td>1.29-0.81</td>
</tr>
<tr>
<td>Applications Experience (AEXP)</td>
<td>1.22-0.81</td>
</tr>
<tr>
<td>Platform Experience (PLEX)</td>
<td>1.19-0.85</td>
</tr>
<tr>
<td>Language and Tool Experience (LTEX)</td>
<td>1.20-0.84</td>
</tr>
<tr>
<td>Use of Software Tools (TOOL)</td>
<td>1.17-0.78</td>
</tr>
<tr>
<td>Multisite development (SITE)</td>
<td>1.22-0.80</td>
</tr>
<tr>
<td>Required Development Schedule (SCED)</td>
<td>1.43-1.00</td>
</tr>
</tbody>
</table>

The nominal weight assigned to each multiplier is 1.0. If a rating level has detrimental effect on effort, its corresponding multiplier is greater than 1.0. On the contrary, if the rating level reduces the effort then the corresponding multiplier is less than 1.0. The 5 scale factors account for the relative economies or diseconomies of scale met as a software project increases its size based on based on different nominal values and rating schemes [32].

III. RELATED WORK

Numerous methods had been inspected for software effort estimation, particularly data-driven soft computing methods, for instance, artificial neural networks, regression trees, evolutionary computing, rule-based induction, fuzzy logics etc., as they showcase a number of advantages above other standard methodologies such as regression.
To commence with, almost all models based on size metrics contemplate the number of coding lines coded for a software project i.e. lines of code (LOC) or thousands of source lines of code (KLOC), as in COCOMO [7], or function points (FP) which is there in models like Albrecht’s FP i.e. Function Point Analysis [8]. Various researchers examine the possibility of developing software effort estimation methods exhausting diverse methodologies, datasets, parameters, etc. In the comparative analysis review presented by [25], it is shown that no one method is good or bad in all the situations, so amalgamation of estimation methods may produce more reliable, accurate cost estimation for software development. Review papers presented by [9][10] embrace a thorough depiction of such studies. In [11] effort estimation was evaluated by backpropagation Artificial Neural Networks on datasets like Desharnais and ASMA, primarily by means of system size to decide the correlation of size with effort. The method generated encouraging estimates signifying that the model requires a further methodical advance method to create the architecture and parametric settings and therefore attain improved outcomes. In [12] the effort estimation calculation of the relationship amongst size and effort was explored using a technique known as Genetic Programming which developed tree configurations indicating numerous linear, power, quadratic type conventional equations. The methodology extended to judiciously worthy stages of prediction correctness by means of merely the size attribute but likewise recommended that added enhancements can be attained.

Kumar et al. suggested a model exhausting particle swarm optimization (PSO) for tuning the parameters of basic COCOMO model to calculate the effort precisely considering merely KLOC factor [13]. Praynlin and Latha confirmed that back propagation algorithm is more effective than a recurrent neural network [14]. Reddy, Raju utilized single layer feed forward neural network and backpropagation an algorithm used for training and resilient backpropagation algorithm for estimating the software development effort precisely [15]. Researcher Finnie et al. [16] in 1997 presented a comparison of statistical regression based model with other artificial Intelligence based estimation models for estimation of software development effort. The Researchers found out that statistical regression model underperformed for intricate and complex software projects while the Artificial Intelligence based models provide agreeable estimation results. They considered dataset among Projects from 17 organization and Desharnais. MMRE was used as an evaluation criterion. Another researcher in 2002, Heiat [17] investigated Feed Forward Neural Networks with function point and Radial Basis Neural Network with Source Lines of Codes for diverse datasets encompassing projects of varied generation languages. For every dataset separately, Heiat has given evaluation with a regression model. He utilized Kemerer dataset of fifteen projects and IBM DP service organization dataset of twenty four projects for the first investigation, and for the second trial, utilized twenty eight projects from Hallmark dataset. The IBM and Kemerer projects are developed using third generation languages while Hallmark projects are developed with fourth generation languages. The outcomes have revealed that artificial neural network method is modest with regression when a third generation language data set is used. But in the case of the 4GLs data set or mixed dataset were used, neural network methodology works expressively precise for software effort estimation.

P.Rijwani et-al [24] using trial and error approach investigated optimal network architecture, in an attempt using backpropagation for training the network. B.Trimula Rao[18] suggested a FLANN for software effort estimation. It generates effort and then processed final output layer. Its one shortcoming is that in this relation between inputs and outputs is not reasonable. Jaswinder Kaur, et-al.[19] instigated a backpropagation Artificial Neural Network of 2-2-1 design on NASA dataset comprises of eighteen projects. KDLLOC and development methodology were the inputs and effort was the output. MMRE was found to be 11.78 with his applied approach. Iman Attarzadeh [20] proposed a new model to accommodate COCOMO II. 5 Scale factors and 17 cost drivers were taken as input. To accomplish post architecture COCOMOII model, sigmoid activation function is utilized while creating the network. Results are shown in terms of MMRE, and Pred(0.25) to compare it with algorithmic COCOMO. Attarzadeh [21] projected a novel software development effort estimation model exhausting neural networks. In this, the Initial weights of the network were set in such a way that it lead to COCOMOII model. The suggested neural network approach provides improved result as related to COCOMO model after appropriate training.

**IV. PROPOSED METHODOLOGY**

Researchers became dissatisfied with conventional and neo-conventional models of estimation so they considered the techniques where they can find meticulous solutions to their problem. The motivation after this endeavor exists in evidence that the capability of ANN greatly influenced by the size, organization, and connectivity of the system and with the right parameter settings, results may be additionally improved. For diminishing the gap between expected output and the real output, weights have been attuned repeatedly by exhausting artificial neural network.

4.1 Artificial Neural Networks

ANNs are non-linear, high-performance logical tools that are capable of creating an association between input and output data deprived of any prior knowledge of the correlation among various variables involved. The principal architecture used in our proposed approach is a supervised and feedforward Multi-Layer Neural Network. In supervised learning, for each input value, the desired output values are available and ANN adjusts its synaptic weight to adapt to the desired functionality. Feed forward ANN takes an input vector in the input layer and forwards it to the neurons of hidden layer to the neuron at the output layer without having any information feedback. For effort prediction we have used a multi-layered feedforward Neural Network with 23 input neurons, each neuron corresponding to one of Boehm’s features i.e. 17 independent cost drivers, 5 scale factors, SIZE features. A hidden layer of neurons to develop the desired mapping and 1 output neuron in compliance to the predicted effort in person-months. A representative ANN network is shown in figure 1.
Each neuron computes the weighted sum of all its inputs (containing bias) by summing the product of input signals with associated synaptic weights. Then, an activation function is applied on this sum for producing output result of each neuron. After the results are received in the hidden layer, they will be processed and their outputs in turn will be distributed to the output layer representing effort. We consider real valued weights and the sigmoid activation function (i.e. $x = 1/(1 + e^{-x})$) is used. The formula of the designed three-layered feed forward neural network is as below:

Each hidden unit $(z_j, j=1,...,p)$ sums its weighted input signals, $Z_{inj}$

$$Z_{inj} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij} \tag{1}$$

Each output unit $(y_k, k=1,....,m)$ sums its weighted input signals,

$$y_{ink} = w_{ok} + \sum_{j=1}^{p} z_j w_{jk} \tag{2}$$

and applies its SIGMOIDAL function to calculate the output signals.

$$Y = f(y_{ink}) \tag{3}$$

The error information term is calculated as

$$\delta_j = \delta_{ink} f(z_{inj}) \tag{4}$$

A general network topology of Multi-Layer feed forward neural net has been presented in Figure 2.

Figure 2: The ANN architecture: feedforward Multi-Layer Neural Network

The supervised learning problem for our Neural Network model is to minimize Error ($W$) with respect to weight $W$; or, find an optimum weight vector $W_p$ that globally minimizes Error ($W$) over the training set. To efficiently solve the global optimization problem, and thereby train the Neural Network to expect effort precisely, we exploit genetic algorithm (GA), which is an efficacious global optimizer used by nature for the evolution of species. The training is performed with 120 NASA and industry data points.

4.2 Genetic Algorithms

Genetic Algorithms are based on biological evolutionary theories for solving optimization problems. Introduced by Holland, it involves a set of individual elements known as the population and a set of biologically inspired operators. According to “Survival of the fittest” theory, the most suited elements in a population are likely to survive and produce offspring, and transfer their biological heredity to the new generations. Due to Genetic Algorithm’s inherent capability of parallelism and
directed stochastic search, they are much more to conventional search and optimization techniques, implemented by recombination operators [29–31].

GA operates through a simple cycle of three stages:

1. Randomly create an initial population of individuals.
2. Perform the following sub steps iteratively for each generation until a termination condition is fulfilled:
   2.1. Evaluate the fitness of each individual in the population and save the best individual of all preceding populations.
   2.2. Create a new population by applying the genetic operators:
      2.2.1. Selection: it is based on the fitness i.e. the fitter an individual the greater the chance for this individual to get selected for reproduction and contribute for the next generation.
      2.2.2. Crossover: This operator takes two chromosomes and swaps part of their genetic information to produce new chromosomes.
      2.2.3. Mutation: It is implemented by sporadically varying a random bit in a string before the offspring are introduced into the new population
   2.3. Replace the current population by the new population.
3. Output the individual with the best fitness as the optimum solution.

Basic process of GA is illustrated in figure 3.

Figure 3: Basic process of genetic Algorithms

V. Evaluation Criteria

5.1 Fitness Function

The evaluation criterion to measure the performance of the developed GANN model is to calculate the mean square error. GAs can enable to determine optimized neural network interconnection weights. The MRE value is intended for each observation i of actual and estimated effort. The accretion of MRE over several observations (N) can be attained through the Mean of MRE (MMRE) as follows:

```matlab
function mse_calc = mse_test(x, net, inputs, targets)
    % 'x' contains the weights and biases vector
    % in row vector form as passed to it by the
    % genetic algorithm. This must be transposed
    % when being set as the weights and biases
    % vector for the network.
    net = setwb(net, x');
    % To set the weights and biases vector to the
    % one given as input
    net = netwb(net, x);
    % To evaluate the outputs based on the given
    % weights and biases vector
    y = net(inputs);
```
Calculating the mean squared error
\[
mse_{\text{calc}} = \frac{\text{sum}((y-\text{targets})^2)}{\text{length}(y)};
\]
end

The detailed information about genetic algorithms is given in [22] [26]. Figure 5 shows GANN implementation process.

Figure 5: Proposed Process implementing GANN

5.2 Experiment planning

To relate Genetic algorithm to neural network training, it requires binary representation of synaptic weight parameters so that genetic operators can be readily applied. To program a solution, projects with 23 effort multipliers are chromosomes, where genes are effort multipliers used as initial population. The fit individuals find more chances in reproduction phase these are proliferated into the population and this increases the opportunity of better solutions. All the weights in the network are joined to make one string. This string is then used in the GA as a member of the population. Each string represents the weights of a complete network. Fitness is measured by calculating the error (target – output) (i.e. fitness= 1/error) - the lower the error the higher the fitness.

Neural Network with 23 input neurons, 10 hidden layer neurons to progress input output mapping, and 1 output neuron to predict software development effort in person-months has been taken. Further, Genetic Algorithm has been used to train the network of weights of Network 23-10-1 in order to minimize the mean squared error over a training COCOMONASA data set. Here the network 23-10-1 is called the phenotype, and the string of weights of Network 23-10-1 is called the genotype. Genotype is a data structure which represents information about the phenotype and which is encoded for use in Genetic Algorithm. Since 23 neurons and one bias value are at input layer, 10 neurons and one bias value are at hidden layer, and 1 neuron is at output layer, so weight vector from input to hidden layer is 23×10, and weight vector from hidden to output layer is 11×1. The neural network 23-10-1 consists of 343 weights connecting various layers. These 241 weights are encoded in a chromosomal string. The real valued coding scheme has been used to form a string.

The GUI prepared in MATLAB for taking various inputs and make calculations is given in figure 6. To start with, the network designed using ANNs are trained in a supervised fashion. Among the inputs, randomly, 70% of the data samples are utilized for training, 20% for validation and 10% for testing the network.

```matlab
% create a neural network
net = feedforwardnet(n);
% configure the neural network for this dataset
for i=1:5
    variable = i
    net = configure(net,data1to5(:,i)',dataOfsum);
```
h = @(x) mse_test(x, net, data1to5(:,i)', dataOfsum);
ga_opts = gaoptimset('TolFun', 1e-2, 'Display', 'iter');
[x_ga_opt1(:,i), err_ga1(:,i)] = ga(h, 3*n+1, ga_opts);
end

for i=1:17
variable = i+5
net = configure(net, data1to17(:,i)', dataOfmul);
h = @(x) mse_test(x, net, data1to17(:,i)', dataOfmul);
ga_opts = gaoptimset('TolFun', 1e-2, 'Display', 'iter');
[x_ga_opt2(:,i), err_ga2(:,i)] = ga(h, 3*n+1, ga_opts);
end

Here feed forward neural network 23-10-1 is known as the phenotype, and the string of weights of feed forward neural network 23-10-1 is called the genotype. Genotype is a data structure that denotes information about the phenotype and that is encoded for use in genetic Algorithms.

Although GA presents an impressive way to optimize ANN weights, it is comparatively slow for local fine tuning in contrast with gradient techniques. A needed method in this instance would be to integrate a local gradient search with the GA.

**SOFTWARE PROJECT EFFORT ESTIMATION**

![Figure 6: GUI for proposed solution Approach](image)

**VI. Datasets Preparation for Training ANN using GA**

For executing the proposed model, data must be saved in spreadsheet using Microsoft Excel file (.xlsx). The input data is stored in one spreadsheet where all features of an input and output dataset must be positioned in one row of Spreadsheet whereas, the corresponding output data i.e. the target data of the input feature data must be sited in the first column of the first row of another sheet. The details of input data entries for historic projects are shown in figure 7. Here, we have 22 input features and one single output data i.e. software effort.

These datasets can be loaded in the Matlab workspace by following command:

```matlab
filename = 'Results.xlsx';
sheet = 2;
c1 = 'B2:B122';
c2 = 'Z2:Z122';
c3 = 'A2:A122';

act_eff = xlsread(filename, sheet, c1);
kloc = xlsread(filename, sheet, c2);
rec_num = xlsread(filename, sheet, c3);
```
VII. Results and Performance Evaluation

Throughout the work, it has been found that development effort of most of the projects is estimated accurately. For look around the goal in global search space, Genetic Algorithm has been tried to optimize the fitness function which is reducing error in terms of mean square error. Mean square error found is 0.0124682 after 50 generations using 10000 populations. Table 2 shows the sample of effort results which are far efficient when compared with other models. Figure 8 (a) (b) shows various comparative charts depicting the results of our proposed model.

Once the training gets finished the regression plot of the trained Artificial Neural Network (net_f). It is observed that regression coefficient R is 0.95365.

Now, the trained network can be functional to know the output of unknown project input values. To know the effort corresponding to any input values, we give command:

\[
\text{test_output} = \text{net_f(test_input)}
\]

Table 2: Effort Calculation Result
VIII. Conclusion and Future Scope

It is well agreed that the accurateness of the various distinct effort estimation models can be well-defined based on accepting the calibration of the software data and this has been confirmed using a hybrid estimation model in this research. A hybrid technique amalgamating neural network and Genetic Algorithm for estimating the effort precisely all through the primary phases of software development is described in this paper. The learning behavior of the algorithm was tested on actual software project examples. It was able to prove its potential and it exhibited good performance. We have clearly compared the approach with the traditional COCOMO II in terms of the performance evaluation factors such as Magnitude of Relative Error (MRE). The algorithm used is adaptable to dynamic factors like minor or abrupt project plan changes or other disturbances. In conclusion the hybridization representation model commands the success of any project is reliable on accurate estimations. This reliability ensures significantly higher probability of project success rate. The projected model later be extended to deal the SDLC phase particular concerns reasonably than just deal on one model for diverse phases where the roles and tasks along with results are exclusively different. This research can also be extended in trying other optimization concepts as well.

Compliance with Ethical Standards:

Author Poonam Rijwani declares that she has no conflict of interest. Author Sonal Jain declares that she has no conflict of interest. …

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

REFERENCES


