Effective ANN Model based on Neuro-Evolution Mechanism for Realistic Software Estimates in the Early Phase of Software Development

Ravi Kumar B N[1]
Dept. of Computer Science and Engineering
BMS Institute of Technology and Management
Bangalore, India

Dr. Yeresime Suresh[2]
Dept. of Computer Science and Engineering
Ballari Institute of Technology and Management
Ballari, India

***Abstract*—There is no doubt that the software industry is one of the fastest-growing sectors on the planet today. As the cost of the entire development process continues to rise, an effective mechanism is needed to estimate the required development cost to control better the cost overrun problem and make the final software product more competitive. However, in the early stages of planning, the project managers have difficulty estimating the realistic value of the effort and cost required to execute development activities. Software evaluation prior to development can minimize risk and upsurge project success rates. Many techniques have been suggested and employed for cost estimation. However, computations based on several of these techniques show that the estimation of development effort and cost vary, which may cause problems for software industries in allocating overall resources costs. The proposed research study proposes the artificial neural network (ANN) based Neural-Evolution technique to provide more realistic software estimates in the early stages of development. The proposed model uses the advantages of the topology augmentation using an evolutionary algorithm to automate and achieve optimality in ANN construction and training. Based on the results and performance analysis, it is observed that software effort prediction using the proposed approach is more accurate and better than other existing approaches.**

Keywords*—*Software cost estimation; COCOMO-II; neuro-evolution; artificial neural network; genetic algorithm

I. INTRODUCTION

The software industry is undoubtedly one of the greatest innovations in the modern world [1]. The software development process broadly requires various discrete actions such as understanding the client requirements, analysis, preparing the user requirement specification, technical requirement specification, software requirement specification, and hardware requirement specification in the initial stages [2]. Further actions architecture design of the software, design of the modules, coding, integration, testing, and debugging. The overall development cost estimation depends on the individual cost and efforts required for each of the actions involved in the SDP. However, estimating the cost in software development has been a challenge facing researchers and professionals in software engineering over the past few years. The purpose of cost estimation is to help with decisions made during the development of a software project. Many factors affect the accuracy of cost estimation. If the cost is underestimated, the project may be delayed, lack implemented features, or not be completed. On the other hand, an overestimated cost can lead to higher software costs, a waste of resources, and even loss of opportunities for competing markets [3]. These factors can have negative consequences for the project, the development organization, and the customers. Thus, the quality of estimates can affect the quality of the software project.

Many software cost estimation models have been developed and improved, which can be categorized into algorithmic and non-algorithmic models [4]. In algorithmic cost model (ACM), typically a mathematical model or expressions are formulated using factors like i) source line of codes (SLOC), ii) risk calculation, and iii) skill levels obtained from the historical records; however, it fails to enumerate many vital factors including i) complexities, ii) reliability and experiences of the projects and due to this, it leads to the imprecise estimation. The constructive cost model- COCOMO is the most popular method in this category [5]. Further, it has evolved as COCOMO-II and has been widely used to design software cost predictors with various strategies considering basic cost indicators like lines of codes (LOC) and the function points [6-7]. The non-algorithmic approach is basically concerned with soft-computing approaches that overcome the limitations of the algorithmic model. The soft-computing approaches handle a better approximation of the solutions of the complex problems where many nonlinear and uncertain parameters are involved. Table I highlights the comparison of algorithmic and non-algorithmic models. Specifically, the existing approaches for the estimation, such as COCOMO and iii) function point-based model, all lack providing desirable accuracy as they ignore many of the critical drivers. So, these methods limit their applicability in the real-time scenario. In order to address these challenges, the soft-computing approaches are being extensively attracted the focus of the researchers by including approaches either individual or by hybrid techniques like- swarm optimization, fuzzy logic, genetic algorithm, machine learning, and neural network [8-10]. The advantage of the soft-computing approach is that it approximates the solutions created by the mess due to nonlinear factors that are uncertain and imprecise. In recent years, neural networks have gained prominence in software development. However, the literature presents several studies on applying neural networks and machine learning techniques to estimate cost [11-12]. However, there is no consensus on which method best predicts software costs. The neural network architecture involves different configuration and hyperparameters such as layers, neuron nodes, transfer function, and learning parameters (weights and biases). Generally, the design of the learning model is specific to the particular data set and problem context. If the same model is introduced with a different dataset, it may not perform similarly. Therefore, the parameters mentioned above affect network performance. However, the evolution of models that produce good results in different environments is still a driving force for current research work. This paper suggests a unique approach to software development cost estimation based on Neuro-evolution. The proposed Neuro-evolution approach implements a mechanism of artificial intelligence (AI) that employs an evolutionary algorithm to generate optimal Artificial Neural Network (ANN) architecture. Further, the constructed ANN model in the proposed work is trained to adopt characteristics of software attributes using the previous dataset to produce accurate software estimates.

TABLE I. ANALYSIS OF ALGORITHMIC AND NON-ALGORITHMIC TECHNIQUES

Techniques	*Category*	*Advantages*	*Limitations*
Analogy	Non-Algorithmic	Independent of new resources	Dependent on past information & huge data requirement.
Expert-based		Highly responsive and fast process	Biased outcome
Bottom-Up		Stable	Inaccurate timings & needs huge data
Top-Down		Faster & low cost	less stable outcome & decisions
COCOMO	Algorithmic	Flexible analysis, input modification, & clear outcomes	Inaccurate estimates & practically infeasible
Function Point		Tool independent	Not good enough
Neural Network	Machine learning	Precise predictive estimates	Highly dependent on the dataset and no standard rule for implementation

The ANN model constructed is a feedforward neural network utilizing backpropagation learning mechanisms. The entire configuration and learning parameter is realized with the evolutionary algorithm, particularly a genetic algorithm (GA) implemented via the Neuro-evolution concept. The proposed study aims to achieve:

- A unique ANN model with an optimal selection of its parameters, including the size of hidden layers, number of neuron units at each layer, and transfer functions, from the given interval (linear, Relu, and sigmoid).
- The stable training process of the constructed ANN model that supports large training data samples.
- Self-adjustment in the weight and biases in an optimal manner from the training samples.
- Enhanced generalization in the training phase and efficient identification of dependencies of the predicted values from the input observations.
- Higher accuracy in the prediction to achieve realistic estimates of the cost required for the software development compared to the existing techniques.

The remaining sections of this paper are organized in the following manner: Section-II presents the review of the literature in the context of software cost and effort estimations; Section III discusses the material and methodology adopted in the proposed work; Section IV presents the system design and implementation procedure adopted in the proposed system; Section V presents the outcome and discusses the performance of the proposed system concerning its scope and effectiveness compared to the existing approaches, and finally, the entire contribution of the proposed work is summarized in Section VI.

II. RELATED WORK

Currently, the literature consists of several types of techniques and schemes for software cost estimation and prediction. This section discusses some of the recent research works carried in the context of enhancing prediction of the cost required for software development.

A. Algorithmic Approaches

The algorithmic approaches are concerned with mathematical models or expressions for cost predictions. To date, various methods have been suggested based on the algorithmic approaches. Work carried out by Kumawat, and Sharma [13] focuses on estimating the size metric for computing the cost required for the software project development (SPD). The authors have used the function point analysis (FPA) technique to compute cost estimates. The work of Khan et al. [14] suggested a cost estimation model by customizing features of the COCOMO-II that integrates additional cost drivers for computing the estimates of actual cost and effort required for SDP. Similarly, the study of Keil et al. [15] has introduced a different version of COCOMO-II to fit in the context of global software development (GSD). Two additional cost drivers are added in this version of cost drivers concerning collaboration and communication among different sites. The researchers in the above-discussed literature have tried to provide a significant contribution. All the factors are determined and devised based on the literature analysis and researchers' knowledge. However, there is a lack of empirical support, effective benchmarking, and validation of the scope of the suggested schemes. The authors in the study of Menzies et al. [16] have introduced a tool that encompasses case studies and previous experience to reduce the execution time, the effort required, and the number of defects in the project's development. Their results were obtained from small data sets, and they recommend conducting other tests where large volumes of information are handled. They do not explicitly use control indicators from other areas of knowledge, for example, to measure human and logistical resources. In the existing literature, few extensions to COCOMO were suggested, including dynamic multistage models to meet the analytical needs of prototyping SPD models. These models consider the dynamics of varying requirements, system design, and other strategies, but all lack desirable accuracy as they ignore many critical drivers. So, these methods limit their applicability with varied IDEs models, languages, and tools.

B. Non-Algorithmic Approaches

The non-algorithmic approach generally implies the soft-computing techniques that handle ambiguity and nonlinearity in the cost estimation techniques. The previous section discusses the conventional approaches regarding software cost and effort estimation. However, software project requirements constantly change over time, which also causes the estimates of cost and effort to change. The researchers realized the need for soft computing approaches that include machine learning techniques, fuzzy logic, and various metaheuristic method. This section discusses the existing soft computing approaches for software effort and cost estimation to analyze the current research trend. Nandal and Sangwan [17] a hybrid Bat and Grautational algorithm is used to estimate the effort of software, whereas fuzzy regression models are used to overcome the problem of imprecise in the dataset for the prediction software effort (Nassif et al. [18]). All these approaches provide a good solution but at the cost of huge computational complexity. The application of evolutionary algorithms like GA is used in the study of Zaidi et al. [19] and Reena et al. [20] to optimize the coefficients of different estimation models in the presence of nonlinear data. The approach of intelligent techniques like the neural network deals with the complexities and uncertainty in the software effort estimation is presented in Venkataiah et al. [21] [22]. Few recent research studies have also focused on applying the hybrid approach in the SPD process. The joint approach of nature-inspired algorithm and ML is adopted by authors in [23-25] to compute the estimates of effort in project development. The work of Singh et al. [26] evaluated different ML techniques in the software effort estimation. The outcome reported in this study showed better performance achieved by LR in terms of error percentage analysis. A neural network approach [27-28] has also been widely accepted in software cost estimation. In the work of Choetkiertikul et al. [29], a long short-term memory (LSTM) and recurrent highway network (RHN) are employed to estimate the effort required for completing user stories or issues. Also, Bayesian Network is used to estimate the work time required in the SPD process [30].

C. Motivation of the Research

A wide range of schemes and techniques have been described in the literature for predicting SPD's costs. The recent literature has been observed more focused on applying metaheuristic techniques, neural networks, and machine learning algorithms. Building a model based on the dataset is difficult due to the complexity and nonlinearity involved in the data attributes. Also, the learning model's design is affected by a variety of factors concerned with network parameters, data modeling, and feature engineering. Apart from this, the factors that determine the connectivity among nodes are complicated to analyze before the training phase to develop an ideal network. Generally, the building and training of the learning model involves a lot of human effort and is specific to the particular context, which is a significant concern as software attributes vary over time. However, even small changes in parameters can dramatically alter the result of the trained model.

A unique model with accurate estimation is presented based on the neuro-evaluation augmenting topology to evolve with an optimized ANN architecture to address and overcome these problems. This type of approach for the cost estimation problem has not yet been applied to the software cost estimation problem. The proposed study aims to explore the effectiveness of augmenting the topology mechanism to automate the construction and training of the ANN model that generates better solutions.

III. Materials and Methodology

The material used for evaluating the proposed model is the COCOMO dataset. The methodology used for designing and developing the proposed ANN model for cost estimation is based on the Neuro-evolution AI technique, which constructs an optimal ANN model using a genetic algorithm. This section briefly highlights the dataset and methodology adopted in the proposed system.

A. Dataset

The COCOMO (Constructive Cost Model) is a widely known software estimation model introduced by Barry Boehm [31]. This model utilizes an approach of statistical correlation between software attributes and lines of the code. In other words, it basically adopts regression analysis with the responsible parameters that are representative of the estimates of the cost required in software development. In the current research work, the study uses the COCOMO NASA-2 dataset publicly accessible at the promise software engineering repository. This dataset consists of a total of 24 vital cost attributes from 93NASA projects.

B. Artificial Neural Network

In recent years, ANN has received wide attention to address complex nonlinear problems in various fields such as computer vision, image processing, natural language processing, and many more. ANN can be viewed as a function approximator that takes an input from observation state and maps to the output state (decision), such that: $f(x) \rightarrow y$. Typically, the function approximators consist of neurons, often referred to as cells or units, composed of summation and activation functions. The typical function of ANN cell is described in Fig. 1 as follows:

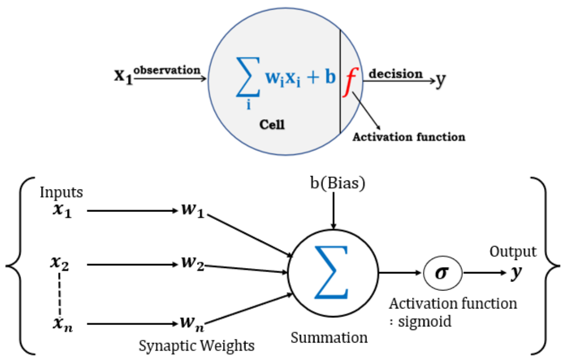

Fig. 1. Typical Function of ANN Cell.

In Fig. 1, the architecture of the basic ANN cell is described where x is the n input such that:$x \in [x_1, x_2, x_3 \cdots x_n]$, w indicates synaptic weight, such that: $w \in [w_1, w_2, w_3 \cdots w_n]$. Each weight 'w' are associated with input sample $'x'$ both together served as input to the cell function, where all x is multiplied with w and are summed with biased (b) using summation function as described as follows:

$$x \cdot w = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n) \quad (1)$$

Equation 25 describes the dot product of vector x and vector w, and their summation is given in equation 26 as follows:

$$\sum = x \cdot w \quad (2)$$

The weights'w_i' can be considered as a strength of the association between cells, and it also decides how much influence the given input will have on the cell's output. Another essential component of the ANN cell is the offset value added to the summation of dot product $x \cdot w$. This offset value is often called a bias that allows shifting the phenomenon of the nonlinear activation function to produce the expected result correctly to the output state. Moreover, the w and b are also often called learning parameters of the ANN model; the relationship between w and b can be numerically represented as follows:

$$(x \cdot w) + b \quad (3)$$

Equation 3 is then passed to the nonlinear function, which is generally a sigmoid function that enables nonlinearity in the ANN cell as numerically represented as follows:

$$y = \sigma(x \cdot w) + b \quad (4)$$

Where y denotes the output of the cell and nonlinear σ sigmoid function. Sigmoid or Logistic: takes a real-valued input and returns output in the range [0,1]. The ANN cells are arranged into several layers, typically classified as input layers, hidden layers and output layers all interconnected to each other.

Usually, the topological structure of the artificial neural network is selected based on empirical analysis, and the learning parameters are determined using the training process, which is related to the trial-and-error process. Therefore, developing an ANN model is not a big problem. However, training ANN models to accomplish certain tasks is a real challenge. In this regard, Neuro-Evolution can be an effective mechanism for determining the optimal topology of neural networks and learning parameters (weights and biases) to construct an ideal ANN model.

C. Neuro-Evolution of Augmenting Topologies

Neuro-Evolution of Augmenting Topology (NEAT) is a neuroevolutionary AI technology that deals with topology augmentation to automate the construction and training of ANN models using evolutionary algorithms (EA) [32]. The EA in NEAT is a kind of genetic algorithm (selection, crossover, and mutation), which allows the evolution of ANN units, learning parameters (weight and biases), and structure, trying to determine stability between the fitness of the obtained solution and assortment. Fig. 2 shows a sample visualization of the topology construction of ANN using the NEAT algorithm.

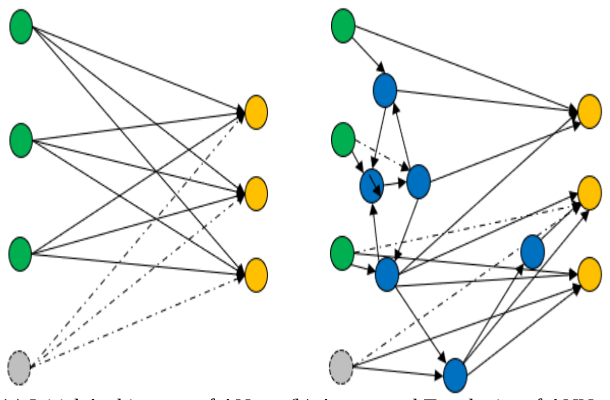
(a) Initial Architecture of AN
(b) Augmented Topologies of ANN.

Fig. 2. Topology Construction of ANN using NEAT.

In the above Fig. 2, visualization of initial topology (a) and final topology construction of ANN model (b) after several iterations is shown using NEAT. The flow process of topology augmentation in the construction and training of the ANN model is shown in Fig. 3.

The mechanism of topology augmentation for the optimal ANN model requires the initialization of variables concerning network hyperparameter and loss function. The initialization of hyperparameter variables (such as learning rate and the number of neurons) is crucial to determine the training performance of the network during the crossover and mutation process of EA. On the other hand, the loss function determines the optimality of the neuron genes (bias) and synapse genes (weight) in the learning phase. The loss function in NEAT is also regarded as a fitness function, and a set of neuron genes and synapse genes are called genomes.

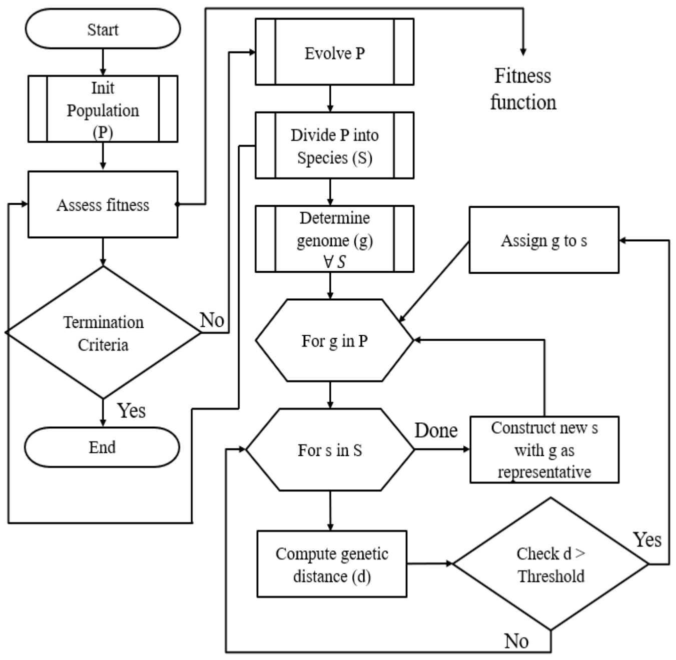

Fig. 3. Flow Process of Topology Augmentation using NEAT.

The algorithm generates a genome considering single input and output layer during the initialization of an initial set of solution candidates (population). Therefore, in the first generation, the genomes only vary in weights and biases but not network topology. After assessing the fitness value of each genome, the algorithm stops if the termination criterion is met. Otherwise, it generates a new set of solution candidates by executing crossovers phase y between genomes and then performs mutations in the subsequent offspring. All these processes are carried out randomly, and prior to computing the fitness of neuron genes and synapse genes, i.e., optimality of weight and biases, the algorithm splits the set of solution candidates into species (a particular class with the common characteristics) based on the computation of the genetic distance between each set of neuron weight and biases. The computation of the genetic distance is carried out using the following numerical equation:

$$d = d_b + d_w \tag{5}$$

The above equation 6 represents the computation of distance (d) based on the summation of neuron ($\mathrm{d_b}$ i.e., bias) and synapse ($\mathrm{d_w}$ i.e., weight). The computation of the $\mathrm{d_b}$ and $\mathrm{d_w}$ are shown in equations 6 and 7 as follows:

$$d_b = c_n \times \frac{\Delta_b}{\max(B(g_1), B(g_2))} \tag{6}$$

$$d_w = c_s \times \frac{\Delta_w}{\max(W(g_1), W(g_2))} \tag{7}$$

Where c_n and c_s are the user-defined variables for fine-tuning the model parameters.

IV. Proposed Cost Estimation Model

This section discusses the proposed cost estimation implementation procedure based on the ANN model determined using the NEAT algorithm discussed in the previous section. In the proposed study, the cost estimation problem is being studied as a regression problem rather than an optimization problem to predict kilo line of code (KLOC). The proposed cost estimation model design involves three core modules; namely, i) data exploration module ii) data preprocessing, and iii) design of ANN Model.

A. Dataset Exploration

In the current study, the data is available on the NASA website. The data is downloaded by sending an HTTP GET request to the respective URLs. When the request is sent, the data can be retrieved in the form of an a.arff file. However, this is not readable readily by our system. Hence, the data is sub-set from the 'Arff file', which contains 10 parts, including {Title, Past Usage, Relevant Information, Number of instances, Number of attributes, Attribute information, Missing attributes, Class distribution, Data}. The sub-set extracts only the Data. The Data Store stores the data in the form of a simple CSV file. Each column is separated by a (delimiter), and a new line character separates each sample. Many data science platforms can read and process this format, including pandas used in the current study. The data imported into the numerical computing environment (NCE) describes 124 entries ranging from the index number 0 to 123 with 24 columns. The dataset consists of 24 variables with type numeric and two categorical variables. The memory taken to upload the data is more than 25 KB. Table II presents a statistical description of all the 25 predictors and an output KLOC. The closer shows that the counts of all the parameters are identical to the number of samples, which indicates there are no missing values. The differential between the consecutive pair between {0, 25%}, {25%, 50%}, {50%, 75%} and {75%, 100%} sometimes are not less than standard deviation (σ) that means there is the presence of outliers in the data, as well if RMSE and MAE of the model have a difference more than mean KLOC then outliers need to be corrected. Another important observation on the dataset is that certain parameters show a specific correlation with the effort. The correlations are either negative correlation or positive correlation. In positively correlated parameters, the effort decreases with a decrease in the parameter's values, whereas, in negatively correlated parameters, the effort decreases if the parameters increase. The positively correlated parameters are the cost drivers (CD) ∈ {acap, pcap}, and negatively correlated parameters such that CD ∈ {rely, Cplx, data, time, stor, sced}. Further, on the analysis of co-efficient using linear regression analysis, it is found that reduced reusability (ruse) and 'site' have a higher multiplier effect on cost/effort compared to other CDs, as evident in Fig. 4. It is clear that the correlation of data points with the actual effort is highly non-uniform in nature. Therefore, a custom feature engineering process for the proposed ANN-based CEM is being carried out.

B. Preprocessing

In this section, the preprocessing operation is carried out from the perspective of the feature engineering task and the extraction of suitable input for the proposed learning model. The core module in this stage contains i) correlation analysis and ii) dataset normalization. In the correlation analysis, the relationships between various variables are analyzed using a mathematical approach that helps find correlations between various cost drivers. The formula for correlation is shown in the equation as follows:

$$\mathrm{r_{x,y}} = \frac{\sum(\mathrm{x_i}-\mathrm{x}').(\mathrm{y_i}-\mathrm{y}')}{\sqrt{\sum(\mathrm{x_i}-\mathrm{x}')^2.\sum(\mathrm{y_i}-\mathrm{y}')^2}} \tag{8}$$

Where, $\mathrm{x_i}$ and $\mathrm{y_i}$ denotes cost drivers, x' and y' are means values of the cost drivers and $\mathrm{r_{x,y}}$ is the correlation factor between x and y that ranges from -1 to +1. As it can be observed from the formula if x ∝ y, which means that x = ky, then the following outcome is achieved when the same is substituted in equation 9.

$$\mathrm{r_{x,y}} = \frac{\sum(\mathrm{x_i}-\mathrm{x}').\mathrm{k}(\mathrm{x_i}-\mathrm{x}')}{\sqrt{\sum(\mathrm{x_i}-\mathrm{x}')^2.\mathrm{k}^2\sum(\mathrm{x_i}-\mathrm{x}')^2}} \tag{9}$$

$$\mathrm{r_{x,y}} = \frac{\sum \mathrm{k}(\mathrm{x_i}-\mathrm{x}')^2}{\mathrm{k}\sqrt{(\sum(\mathrm{x_i}-\mathrm{x}')^2)^2}} \tag{10}$$

$$\mathrm{r_{x,y}} = \frac{\mathrm{k}\sum(\mathrm{x_i}-\mathrm{x}')^2}{\mathrm{k}\sum(\mathrm{x_i}-\mathrm{x}')^2} \tag{11}$$

$$\mathrm{r_{x,y}} = 1 \tag{12}$$

The above equation 12 proves that when the two cost drivers are proportional, the correlation between them is one. Similarly, when one cost driver reduces and another cost driver increases, in other words, x=k-ly, then the correlation is said to be -1 and considered as an ideal scenario when there is a perfect linear relationship between two CDs. However, a zero correlation refers to total randomness and no relation between two CDs. The correlation plot for among CDs is given in Fig. 5. It can be analyzed that there is a strong correlation between the 'prec', 'flex', 'resl' and 'team'. As it can be observed that except for exponential CDs such that {'prec', 'flex', 'resl', 'team' and 'pmat'} all other CDs have (>10%) correlation. Hence, all variable turns out to be significant while building an ANN model.

TABLE II. DESCRIPTIVE STATISTICS

Cost Drivers	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
ACT_EFFORT	124.0	563.334677	1029.227941	6.00	71.50	239.500	581.750	8211.00
prec	124.0	3.110000	1.292409	0.00	2.48	2.480	4.9600	4.960000
flex	124.0	2.618952	1.041618	0.00	103	2.030	4.0500	5.070000
resl	124.0	3.688871	1.403707	0.00	2.83	2.830	5.6500	6.010000
team	124.0	1.837097	1.094185	0.00	1.10	1.100	3.2900	4.660000
pmat	124.0	5.602984	1.288265	2.84	4.68	4.680	6.2400	7.800000
relay	124.0	1.078522	0.103427	0.85	1.00	1.100	1.1000	1.740000
Cplx	124.0	1.189892	0.163256	0.87	1.17	1.170	1.2125	1.740000
Data	124.0	1.014919	0.117179	0.90	0.90	1.000	1.1400	1.280000
Ruse	124.0	0.996935	0.014605	0.95	1.00	1.000	1.0000	1.070000
Time	124.0	1.124516	0.184476	1.00	1.00	1.000	1.2900	1.630000
Stor	124.0	1.107097	0.163149	1.00	1.00	1.000	1.1700	1.460000
Pvol	124.0	0.927406	0.095456	0.87	0.87	0.870	1.0000	1.150000
Acap	124.0	0.880276	0.101079	0.71	0.85	0.850	1.0000	1.016667
Pcap	124.0	0.918817	0.085625	0.76	0.88	0.895	1.0000	1.000000
pcon	124.0	1.000544	0.035766	0.81	1.00	1.000	1.0000	1.205000
Apex	124.0	0.925712	0.083496	0.81	0.88	0.880	1.0000	1.220000
Plex	124.0	1.004590	0.080974	0.91	0.91	1.000	1.0000	1.190000
ltex	124.0	0.966781	0.089415	0.91	0.91	0.910	1.0000	1.200000
Tool	124.0	1.115847	0.078542	0.83	1.09	1.170	1.1700	1.170000
Sced	124.0	1.043065	0.063760	1.00	1.00	1.000	1.1400	1.140000
Site	124.0	0.925040	0.017623	0.86	0.93	0.930	0.9300	0.947500
docu	124.0	1.024940	0.057830	0.91	1.00	1.000	1.1100	1.230000
Physical Delivered KLOC	124.0	103.443901	141.455891	0.00	20.00	51.900	131.7500	980.000000

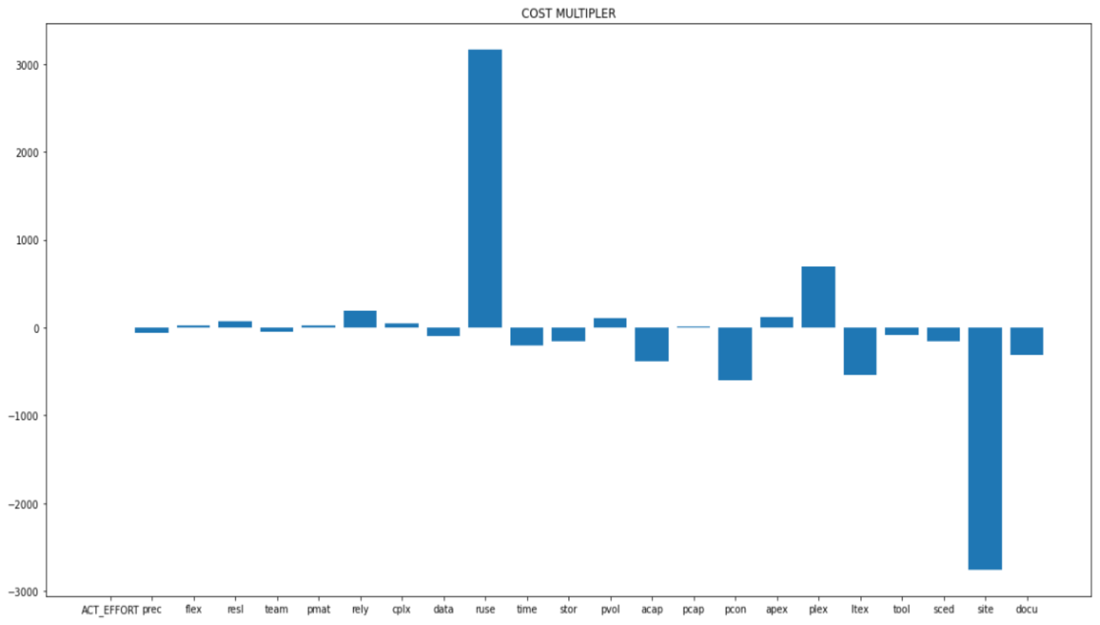

Fig. 4. Representation of Cost Multiplier.

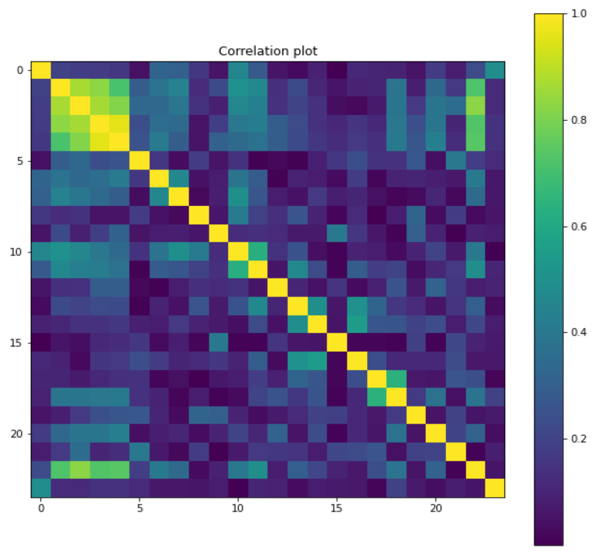

Fig. 5. Correlation Plot among CDs and KLOC.

In order to provide an input to a learning model, the input data is required to be in a vector form. Feature vectorization refers to converting a row of values into a usable vector. In this phase of implementation, the data is normalized with the help of the Min-Max scaling method. Further, each row is transposed and fed to neural networks. The typical formula for data normalization for feature vectors is numerically expressed in equation 13.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \quad (13)$$

Where, x is the input data, i.e., original CDs feature samples, which is normalized using min and max function and rescaled in the range of [0,1], and x' normalized CDs feature samples which are further fed to the proposed learning model.

C. *Design of the Proposed ANN Model*

This section discusses the ANN model design and its implementation procedure with the support of the algorithmic steps. The implementation procedure utilizes the NEAT library of python executed in the Anaconda distribution. The dataset is split into training and testing sets, where 80% of the dataset is kept for the model training, and 20% of the dataset is kept for model testing. The design configuration of the proposed ANN model is carried out using neural evolution mechanisms, where the features from the input observation are considered for determining weights and biases. In this process, the optimality of the ANN architecture is determined through topology augmentation using a genetic algorithm. The configuration parameters considered in the ANN construction consist of hidden layers, neurons unit at each hidden layer, and a set of transfer functions. The proposed study considers three transfer functions: linear, Relu, and nonlinear sigmoid. On the other hand, mean square error (MSE) is considered a fitness function. Since the proposed study has considered MSE, the fitness evaluation is carried out based the less error. Therefore, the inverse roulette selection (IRS) technique is considered for the proportionate fitness selection. The core configuration and training process of ANN construction using topology augmentation is shown in Fig. 6. The topology augmentation begins with the initialization of population (a set of candidate solutions), basically a pool of random neural networks. The process iterates several times, which is also called a generation where the algorithm chooses the optimal ANN based on the fitness value, which is then further cross overed according to the selection/decision process.

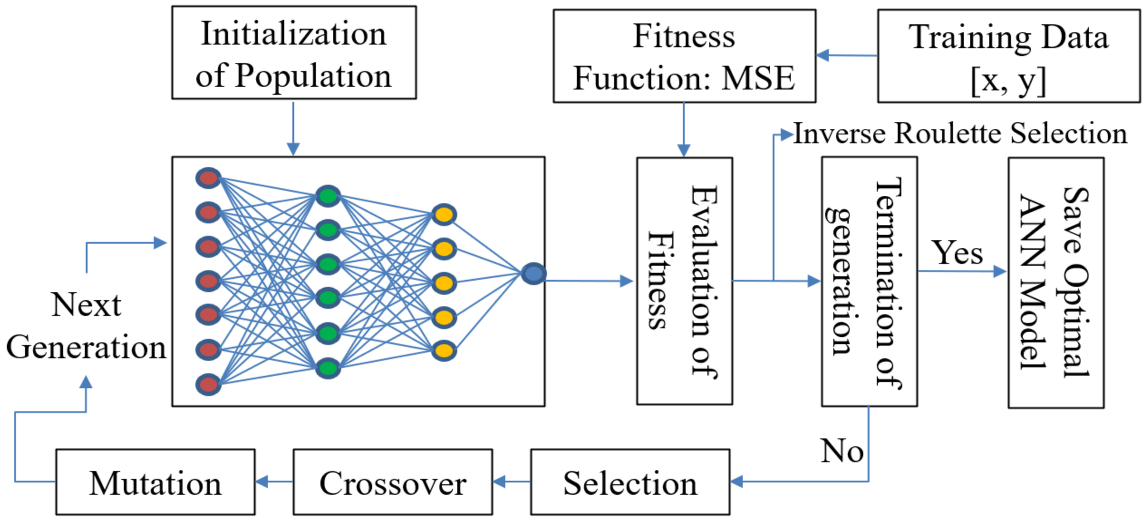

Fig. 6. Generation of Optimal ANN Model using Neuro-evolution Technique.

Afterward, a new ANN model is generated, and after mutation, an evolved version of the ANN model is further carried out for the training process. All these processes continue until the termination criteria are met. This termination criterion is based on the specified number of generations, wherein each generation, the trained model is evaluated and selected according to the prediction performance. The implementation steps for the above-discussed procedure are mentioned as follows:

Algorithm:1 Neuro evolution training

Step 1. Create population pool
In this step, the population pool is generated, a set of random neural networks with random layers and neurons and random activation functions. Inputs to the algorithm are given in the form of a finite number of layers and neurons and, at the same time, a set of activation functions. The activation functions allowed are, Sigmoid, Linear, and relu.

Step 2. Evaluate fitness of the population
The MSE fitness function measures the fitness of the population. The MSE of the input data is considered with the output in the training set.

Step 3. Select the fittest individual to reproduce
The inverse Russian roulette process selects the individuals for the repopulation pool. The lower the fitness function value, the higher the probability of the selection. The following equation decides the probability of selection.

$$P_i = 1 - \frac{MSE_i}{\sum_{i=1}^{n} MSE_i} \tag{14}$$

Step 4. Repopulate using copies of the fittest network
Most fit individuals among the population are selected and used for further processing. The crossover of these individuals is made here, and also mutation is applied according to the mutation probability.

Step 5. Introduce normally distributed mutations to the network weights
The neural networks are finalized in this step, and the newly formed networks are introduced to the population pool.

V. Result and Performance Analysis

This section discusses the performance metrics followed by outcome analysis to justify the scope and effectiveness of the proposed system.

A. Neuro Evolution Model Parameters

The design and development of the proposed system are done using python programming language and execution on Anaconda. The parameters considered for executing proposed neuroevolutionary technique for obtaining optimal ANN model is mentioned in Table III.

The parameter namely population size is the total number of offspring (networks) present in each generation and total number of generations is number of times the fitness is measured. In 15% of the cases a new neuron is added to the network. In 10% of the cases an existing neuron is deleted from the network. Addition and deletion of neurons happen within a single generation. Either relu, sigmoid or linear activation functions are chosen. Initial bias is assigned according to the normal distribution. Maximum value of weights and bias are set to 30 however the minimum weight is set to 0 in order avoid negative values. At the same time, minimum bias is set to -5 in order to cancel out certain values.

Mutation probability is 5%. This is necessary to display the stochastic nature of the system. After successful execution of the neuro-evolution training, the proposed algorithm returns optimal ANN model discussed in Table IV.

The architecture of the obtained ANN model is shown in Fig. 7. After evolution through several iteration, the neuro-evolution algorithm provides optimal number of layers and number of neurons unit at each layer as mentioned in Table IV.

TABLE III. Neuro-evolution Hyperparameters

Parameters	**Values**
Population size	200
Number of generations	100
Probability of adding a new neuron	0.15
Probability of deleting a neuron	0.1
Activation function	Sigmoid, Relu, Linear
Initial bias	according to normal distribution
Mutation probability	0.5
Minimum neuron bias	-5
Maximum neuron bias	30
Minimum weight	0
Maximum weight	30
Weight mutation probability	0.5

TABLE IV. Configuration Description of Obtained Optimal ANN Model

Layer	**Number of neurons**	**Trainable parameters**
Layer 1 (input)	24	N/A
Layer 2	10	(24*10) + 10 = 250
Layer 3	5	(10 * 5) + 5 = 55
Layer 4 (output)	1	(5 * 1) + 1 = 7
Loss Function (MSE)	-	-
Activation Function (Relu)	-	-
	Total neurons: 40	Total trainable parameters: 312

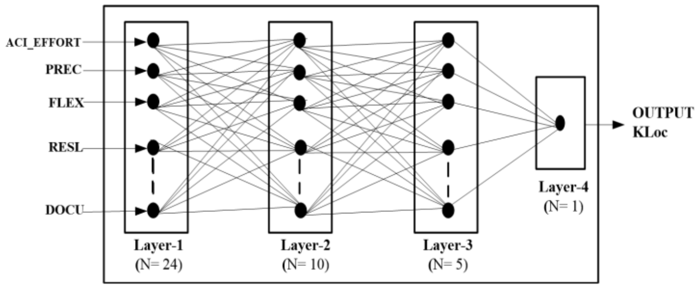

Fig. 7. Architecture of Optimal ANN Model.

B. Performance Metrics

1) MMRE (Mean Magnitude of Relative Error): The MMRE performance metric is the most common basis for the assessment of the effort estimation process. The matric MMRE is computed for the given dataset of software projects whose estimated efforts are compared with their actual efforts. The estimation process with minimum MMRE is considered to be the most accurate. The formula for calculating MMRE is given as Eq. 15.

$$\text{MMRE} = \frac{1}{N} \cdot \sum_{i=1}^{n} \frac{\lfloor(y-y')\rfloor}{y} \tag{15}$$

Where, y is the actual effort, and y' denotes estimated work effort for project pi, and N is the total project (PI) under consideration. Mathematically, MMRE gives an average percentage of error between y and y'.

2) MSE (Mean Squared Error): MSE is being calculated in proposed implementations to analyze the performance of proposed methods over other LR and SVR. MSE is more critical function while building better models while optimizing the learning model. The formula for calculating MSE is given as Eq. 16.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y-y')^2 \tag{16}$$

Where y is the actual effort, and y' denotes estimated work effort for project pi, and N is the total number of the project under consideration.

3) RMSE (Root Mean Square Error): Since the unit of MSE is squared, RMSE is the square root of MSE used since the unit of MSE is Nl^2 where Nl is the number of lines of code in the project. Though MSE is significant for optimizing the model, it would make no sense to human beings. Hence, the study considers $\text{RMSE} = \sqrt{\text{MSE}}$. Since the unit of RMSE is Nl, it can be assumed that the most probable range for y can be $y = y' \pm \text{RMSE}$. The computation of RMSE can be numerically represented as follows in eq. 17:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-y')^2} \tag{17}$$

4) MAE (Mean Absolute Error): This is similar to MMRE, representing average absolute error instead of providing average percentage error. In MAE abs function is used to remove the error from simple error, and the average is calculated. Due to this, some of the extreme points, like outliers, will provide less significance; hence this measure is less sensitive to outliers. MAE can be numerically represented as follows in eq. 18:

$$\text{MAE} = \frac{1}{N} \cdot \sum_{i=1}^{n}\lfloor(y-y')\rfloor \tag{18}$$

Since the unit of MAE and output (actual cost) is the same, MAE represents total cost overrun or underrun.

5) Pred: PRED is the de facto standard for cost model accuracy measurement. It is called the percentage of predictions falling within the K% of the actual known value. The formula for PRED calculation is shown in equation 19:

$$\text{PRED} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\text{EstimationEffort}-\text{ActualEffort}}{\text{Actual Effort}}\right| \text{K}\,\% \tag{19}$$

Where k% is the percentage error between AE and EE, PRED represents the percentage of a number of projects whose cost overrun or underrun is below 25% in some researches 30%.

C. Outcome Analysis

This section discusses the outcome obtained for the proposed system based on the comparative analysis. The proposed study implements two machine learning algorithms for the comparative analysis such as Linear regression (LR) and supports vector regression (SVR). In order to compare ANN with LR and SVR, the performance metrics MSE, RMSE, and MAE are considered. To justify the scope of the proposed optimal ANN model, the study also considers performance analysis with similar existing approaches such as estimation technique based on fuzzy-genetic [33] and based Dolphin optimization technique [34], Bat optimization [34], and combined Dolphin-BAT [34], the performance metric PRED and MMRE is used. The quantitative outcome obtained for the proposed system and its comparison is shown in Table V.

As it can be observed in Table V, that LR, SVR is associated with 151% and 128% errors, respectively, which means the predicted/estimated value could be more than twice as big as the actual value; therefore, making LR and SVR unfit for real-world implementations. However, even the most basic benchmarked algorithms (GA) are giving 29.9% error which is below 30%, which is an acceptable cost overrun ratio for software projects in general. It is also far below 77%, which is the average cost overrun ratio of the NASA project from which the dataset is collected. The overall numerical outcome shows the proposed ANN's effectiveness regarding the cost overrun ratio. Performance analysis regarding MAE is shown in Table VI.

TABLE V. QUANTITATIVE OBSERVATION IN TERMS OF MMRE

Methods	**Performance Metrics**
LR	1.510457
SVR	1.281522
GA	0.299469
BAT	0.1698
DOLPHIN	0.1665
DOLPHIN-BAT	0.14576
ANN	0.113518

TABLE VI. QUANTITATIVE OBSERVATION IN TERMS OF MAE

Methods	**Performance Metrics**
LR	119.266357
SVR	81.872095
ANN	22.151230

The performance metric MAE is used to calculate the performance of proposed methods over other LR and SVR. Since the unit of MAE is in Nl, the MAE value 22.15 obtained for ANN represents a number of lines of codes in the projects that may vary by 22,151 lines in ANN. An average developer writes 250 lines of production code per week (40 hours of working per week). An extra 22151 lines represent 88 weeks of work (3520-man hours). Considering that an average developer in the USA earns approximately $34 per hour, the total cost overrun might come to $119,680. In the cases of LR and SVR, the cost overrun is quite more than ANN, which is impractical for real-time implementation? The performance of the learning models implemented in this study regarding MSE is shown in Table VII.

The metric MSE is being considered in proposed implementations to assess the performance of the proposed ANN over other LR and SVR. MSE represents the overall training of the algorithm as it is used for optimization. Even though the MSE does not directly represent the algorithm's performance, it does represent the quality and level of training given to the algorithm. Lower MSE represents higher knowledge of the algorithm. More trainable parameters can store more knowledge among them. The MSE score is higher in both LR and SVR as they contain fewer trainable parameters than ANN. The quantified outcome indicates that ANN is less associated with error compared to LR and SVR. Therefore, it can be concluded that SVR and LR are subjected issue of underfitting. The performance analysis in terms of RMSE is mentioned in Table VIII.

Similarly, the metric RMSE is considered to evaluate the training performance of the learning models. The RMSE also helps to understand the requirement re-training model by the preprocessing step. From the quantified outcome, the proposed ANN scored 39.33 % RMSE and 22.15% MAE from Table VI, i.e., a difference of 17.18 % compared to mean KLOC of all projects, i.e., 103.44. This indicates minor variation with 16%-17%, which is within the acceptable limit of 20 %. The performance analysis regarding PRED is shown in Table IX.

PRED represents the ratio of projects which has less than a threshold percentage of cost overrun. Hence, this performance measurement is more practical than the other metrics since it represents the number of projects that will fall below the acceptable cost overrun ratio. In most of the studies, the threshold is set to 30%. In this study, 25% of the threshold value is considered to perform comparative analysis. From Table IX, it can be observed that the proposed model ANN achieved a higher PRED value, i.e., 68.91, compared to other ML methods and existing approaches. Bat, Dolphin, hybrid Dolphin-Bat, and the proposed ANN are more practical to implement as they have PRED value much higher than GA. But among them, the proposed ANN method has the highest PRED value, which indicates its suitability and scope in the real-world system. The following analysis mentions the overall improvement (%) of ANN concerning MMRE in Fig. 8 and PRED in Fig. 9 over other implemented ML models and existing approaches.

TABLE VII. QUANTITATIVE OBSERVATION IN TERMS OF MSE

Methods	**Performance Metrics**
LR	42545.810081
SVR	29240.145478
ANN	1547.247493

TABLE VIII. QUANTITATIVE OBSERVATION IN TERMS OF RMSE

Methods	**Performance Metrics**
LR	206.266357
SVR	170.997501
ANN	39.335067

TABLE IX. QUANTITATIVE OBSERVATION IN TERMS OF PRED

Methods	**Performance Metrics**
LR	2.335234
SVR	5.297425
GA	11.66
BAT	61.66
DOLPHIN	61.66
DOLPHIN-BAT	66.66
ANN	68.91522

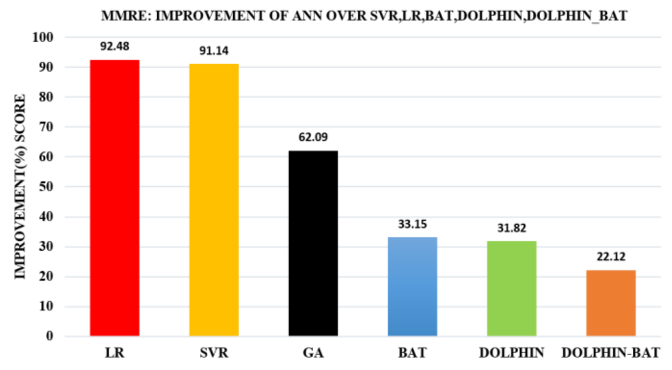

Fig. 8. MMRE Improvement (%) of ANN over SVR, LR and existing Methods.

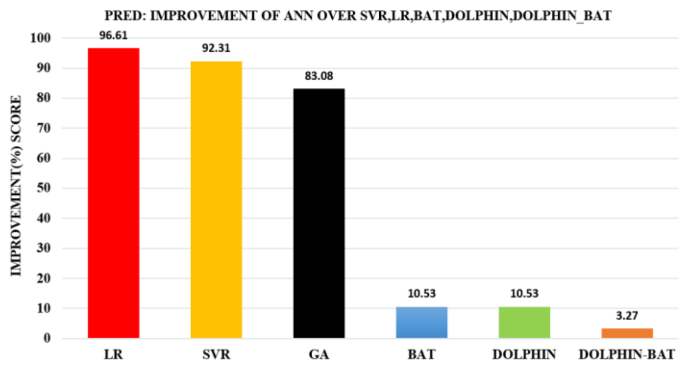

Fig. 9. PRED Improvements (%) of ANN over SVR, LR and Existing Methods.

The analysis from Fig. 8 shows ANN has achieved 92.4% improvement over LR, 91.14% improvement over SVR, and 62.09%, 33.15%, 31.82%, 22.12% over Fuzzy-GA, BAT, Dolphin, and Dolphin-Bat, respectively. The analysis from Fig. 9 shows that ANN has achieved 96.91% improvement over LR, 92.31% improvement over SVR, and 83.08%, 10.53%, 10.53%, 3.27% over Fuzzy-GA, BAT, Dolphin, and Dolphin-Bat, respectively. Hence, it can be seen that the proposed offers a good result regarding software cost estimates. The overall analysis shows effectiveness of the proposed neuro-evolution algorithm towards devising suitable learning model for achieving realistic estimates of the cost required in the initial stage of the software development process. Hence, the proposed research work suggested a technically-efficient method acquainted with recent trends and technologies to benefit real-world applications.

VI. Conclusion

The development of software projects involves various phases like initial planning, risk assessment, effort, and cost estimation. Among these, cost estimation is the key concern in the software industry. The conventional approaches do not provide accurate estimation due to the lack of precise system and cost drivers modeling. In this paper, the study has presented a novel and unique approach to predict realistic estimates of the cost needed to develop a software project. The proposed study applied a mechanism of neural evolution in conjunction with evolutionary technique, namely genetic algorithm top construct ANN, which predicts actual estimates of the cost required to develop a software. The application of neural evolution in ANN modeling proves its effectiveness and scope that it can compete with the existing techniques in terms of realistic estimates of the cost and effort. Once developed and trained, the proposed ANN can estimate the development costs in real-time as it computes cost estimates based on the responsible attributes required in the development of the software. The execution complexity grows linearly with the problem context and size of data samples. Based on the result analysis, it is observed that the proposed ANN is producing better results than other previously proposed algorithms and other machine learning models being implemented. The existing works adopted global optimization algorithms that require huge computing resources due to recursive operation in parallel. However, the proposed ANN model is constructed optimally using the mechanism of augmenting topology, and it better adopts generalization of the feature from the input observations, therefore, providing accurate estimates of the cost compared to the existing approaches.

References

[1] Alt, R., Leimeister, J.M., Priemuth, T. et al. Software-Defined Business. Bus Inf Syst Eng 62, 609–621 (2020).

[2] Trendowicz, A., 2013. Software Cost Estimation, Benchmarking, and Risk Assessment: The Software Decision-Makers' Guide to Predictable Software Development. Springer Science & Business Media.

[3] Mittas, N. and Angelis, L., 2013, September. Overestimation and underestimation of software cost models: Evaluation by visualization. In 2013 39th Euromicro Conference on Software Engineering and Advanced Applications (pp. 317-324). IEEE.

[4] Khan, B., Khan, W., Arshad, M. and Jan, N., 2020. Software Cost Estimation: Algorithmic and Non-Algorithmic Approaches. International Journal of Data Science and Advanced Analytics (ISSN 2563-4429), 2(2), pp.1-5.

[5] Kaushik, A., Chauhan, A., Mittal, D. and Gupta, S., 2012. COCOMO estimates using neural networks. International Journal of Intelligent Systems and Applications, 4(9), pp.22-28.

[6] Singh, B.K., Tiwari, S., Mishra, K.K. and Punhani, A., 2021. Extended COCOMO: robust and interpretable neuro-fuzzy modelling. International Journal of Computational Vision and Robotics, 11(1), pp.41-65.

[7] Coelho, E. and Basu, A., 2012. Effort estimation in agile software development using story points. International Journal of Applied Information Systems (IJAIS), 3(7).

[8] Bedi, R.P.S. and Singh, A., 2017. Software Cost Estimation using Fuzzy Logic. Indian Journal of Science and Technology, 10, p.3.

[9] Singh, B.K. and Misra, A.K., 2012. Software effort estimation by genetic algorithm tuned parameters of modified constructive cost model for nasa software projects. International Journal of Computer Applications, 59(9).

[10] Nassif, A.B., Azzeh, M., Capretz, L.F. and Ho, D., 2016. Neural network models for software development effort estimation: a comparative study. Neural Computing and Applications, 27(8), pp.2369-2381.

[11] Tayyab M.R., Usman M., Ahmad W. (2018) A Machine Learning Based Model for Software Cost Estimation. In: Bi Y., Kapoor S., Bhatia R. (eds) Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016. IntelliSys 2016. Lecture Notes in Networks and Systems, vol 16. Springer, Cham.

[12] Sakhrawi, Z., Sellami, A. & Bouassida, N. Software enhancement effort estimation using correlation-based feature selection and stacking ensemble method.

[13] Kumawat P., Sharma N. (2019) Design and Development of Cost Measurement Mechanism for Re-Engineering Project Using Function Point Analysis. In: Kamal R., Henshaw M., Nair P. (eds) International Conference on Advanced Computing Networking and Informatics. Advances in Intelligent Systems and Computing, vol 870. Springer, Singapore.

[14] J. A. Khan, S. U. R. Khan, T. A. Khan and I. U. R. Khan, "An Amplified COCOMO-II Based Cost Estimation Model in Global Software Development Context," in IEEE Access, vol. 9, pp. 88602-88620, 2021.

[15] P. Keil, D. J. Paulish, and R. S. Sangwan, "Cost estimation for global software development," in Proc. Int. Workshop Econ. Driven Softw. Eng. Res. (EDSER), 2006, pp. 7–10.

[16] Menzies T, Brady A, Keung J, Hihn J, Williams S, El-Rawas O, Green P, Boehm B. Learning project management decisions: a case study with case-based reasoning versus data farming. IEEE Transactions on Software Engineering. 2013 Sep 16;39(12):1698-713.

[17] D. Nandal and O. P. Sangwan, "Software cost estimation by optimizing COCOMO model using hybrid BATGSA algorithm," Int. J. Intell. Eng. Syst., vol. 11, no. 4, pp. 250–263, 2018.

[18] A. B. Nassif, M. Azzeh, A. Idri, and A. Abran, "Software development effort estimation using regression fuzzy models," Comput. Intell. Neurosci., vol. 2019, pp. 1–17, Feb. 2019.

[19] Zaidi SA, Katiyar V, Abbas SQ (2017) Development of a framework for software cost estimation: design phase. Int J Tech Res Appl 5(2):68–72.

[20] Reena, Bhatia PK (2017) Application of genetic algorithm in software engineering: a review. Int Refereed J Eng Sci 6(2):63–69.

[21] V. Venkataiah, R. Mohanty, M. Nagaratna, Prediction of software cost estimation using spiking neural networks, in: Smart Intell. Comput. Appl. Smart Innov. Syst. Technol., Springer, Singapore, 2019, pp. 101–112, http: //dx.doi.org/10.1007/978-981-13-1927-3_11.

[22] V. Venkataiah, R. Mohanty, M. Nagaratna, Prediction of software cost estimation using spiking neural networks, Smart Innov. Syst. Technol. 105 (2019) 101–112, http://dx.doi.org/10.1007/978-981-13-1927-3_11.

[23] S. Kumari, S. Pushkar, Cuckoo search based hybrid models for improving the accuracy of software effort estimation, Microsyst. Technol. 24 (2018) 4767–4774, http://dx.doi.org/10.1007/s00542-018-3871-9.

[24] M. Pandey, R. Litoriya, P. Pandey, Validation of existing software effort estimation techniques in context with mobile software applications, Wirel. Pers. Commun. 110 (2020) 1659–1677, http://dx.doi.org/10.1007/s11277-019-06805-0.

[25] S. Goyal, P.K. Bhatia, Feature selection technique for effective software effort estimation using multi-layer perceptrons, Lect. Notes Electr. Eng. 605 (2020) 183–194, http://dx.doi.org/10.1007/978-3-030-30577-2_15.

[26] A.J. Singh, M. Kumar, Comparative analysis on prediction of software effort estimation using machine learning techniques, SSRN Electron. J. (2020) 1–6, http://dx.doi.org/10.2139/ssrn.3565822.

[27] M. Qin, L. Shen, D. Zhang, L. Zhao, Deep learning model for function point based software cost estimation -an industry case study, in: Proc. - 2019 Int. Conf. Intell. Comput. Autom. Syst. ICICAS 2019, 2019, pp. 768–772, http://dx.doi.org/10.1109/ICICAS48597.2019.00165.

[28] V. Resmi, S. Vijayalakshmi, Kernel fuzzy clustering with output layer self-connection recurrent neural networks for software cost estimation, J. Circuits, Syst. Comput. 29 (2019) 1–17, http://dx.doi.org/10.1142/ S0218126620500917.

[29] M. Choetkiertikul, H.K. Dam, T. Tran, T. Pham, A. Ghose, T. Menzies, A deep learning model for estimating story points, IEEE Transactions on Software Engineering, 45 (2019), 637–656, http://dx.doi.org/10.1109/TSE.2018.2792473.

[30] Dragicevic, S., Celar, S., & Turic, M. (2017). Bayesian network model for task effort estimation in agile software development. Journal of Systems and Software, 127, 109- 119. DOI: 10.1016/j.jss.2017.01.027.

[31] http://promise.site.uottawa.ca/SERepository/datasets/cocomonasa_v1.arff

[32] Stanley, K.O., Miikkulainen, R., 2002a. Evolving neural networks through augmenting topologies. Evolutionary Computation 10 (2), 99–127.

[33] X Chhabra, S., Singh, H. Optimizing design parameters of fuzzy model based COCOMO using genetic algorithms. Int. j. inf. tecnol. 12, 1259–1269 (2020). https://doi.org/10.1007/s41870-019-00325-7.

[34] A. A. Fadhil, R. G. H. Alsarraj and A. M. Altaie, "Software Cost Estimation Based on Dolphin Algorithm," in IEEE Access, vol. 8, pp. 75279-75287, 2020, doi: 10.1109/ACCESS.2020.2988867.