IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 30, 2021, accepted February 4, 2021, date of publication February 8, 2021, date of current version February 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3057807

A New Approach to Software Effort Estimation Using Different Artificial Neural Network Architectures and Taguchi Orthogonal Arrays

NEVENA RANKOVIC[1], DRAGICA RANKOVIC[1], MIRJANA IVANOVIC[2], (Member, IEEE), AND LJUBOMIR LAZIC[1]

[1]School of Computing, Union University, 11000 Belgrade, Serbia
[2]Faculty of Sciences, University of Novi Sad, 21000 Novi Sad, Serbia

Corresponding author: Nevena Rankovic (nrankovic@raf.rs)

ABSTRACT In this article, two different architectures of Artificial Neural Networks (ANN) are proposed as an efficient tool for predicting and estimating software effort. Artificial Neural Networks, as a branch of machine learning, are used in estimation because they tend towards fast learning and giving better and more accurate results. The search/optimization embraced here is motivated by the Taguchi method based on Orthogonal Arrays (an extraordinary set of Latin Squares), which demonstrated to be an effective apparatus in a robust design. This article aims to minimize the magnitude relative error (MRE) in effort estimation by using Taguchi's Orthogonal Arrays, as well as to find the simplest possible architecture of an artificial Neural Network for optimized learning. A descending gradient (GA) criterion has also been introduced to know when to stop performing iterations. Given the importance of estimating software projects, our work aims to cover as many different values of actual efficiency of a wide range of projects as possible by division into clusters and a certain coding method, in addition to the mentioned tools. In this way, the risk of error estimation can be reduced, to increase the rate of completed software projects.

INDEX TERMS Software effort estimation, Taguchi method, artificial neural networks design, orthogonal array-based experiments, clustering, COCOMO81, COCOMO2000, NASA project dataset, Kemerer dataset.

I. INTRODUCTION

Accurate software development effort estimation is a crucial factor in software project's success and reducing risks. At the same time, each project has a special nature that makes it much harder to estimate the required effort for completion, and the task prediction becomes more challenging. Over the past years, there have been many Machine Learning (ML) approaches in the literature that have been applied to improve the software effort estimation. Many researchers [1]–[5], also proposed different methods for optimizing the parameters of three COCOMO-based models using some of the most popular Neural Networks. Neural networks are frequently used as a tool for software effort prediction because of their aptness for arbitrary accuracy. There is an enormous area of real-world applications [3]–[8], for which artificial Neural Networks (ANN) have proved very efficient due to their learning capability. Most of ANNs architecture learning refers to modifications of the values of neuron weights, which modulate signal transmission between interconnected neurons. ANN learning algorithms determine weight modifications, for example, in order to optimize the cost function. The most efficient, but rarely used, cost function is the summed magnitude relative error (MRE) between current ANN output and the desired (target) output, which can lead to the smallest value of cost. Contrary to these new approaches, which are based on ANN architec- ture and require a smaller number of iterations, traditional learning algorithms are based on numerical methods that require a huge number of iterations. In accordance with that, most

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Liu.

popular learning methods are based on Gradient Descent (GA) [5]–[10] optimization. This value is important for real-time applications and security in high-risk systems, which are expected to quickly learn and adapt to their environments, and for which fast learning methods are in high demand. Previous research [1], [2] has shown significant shortcomings in the design of Neural Network architecture. The main disadvantages indicate the choice of the optimal set of parameters during training when achieving the convergence rate and the required accuracy. In this article we propose two different ANN architectures, based on the Taguchi technique for robust design, which are using three software development attributes in COCOMO models as control variables (exponent – scale factors, cost factors, and software size). Robust design optimization through the Taguchi technique includes orthogonal arrays. At the same time, it is considered important to develop a prediction method that is less complex and much more useful. The Neural Network is associated with an Orthogonal Array (OA) with the number of parameters equaling the number of weights of the ANN. The interval containing the solution is gradually dwindling as the result of an iterative algorithm. In this study, the obtained results show that the method of Taguchi orthogonal arrays is an efficient approach for directing Neural Networks in learning speed and accuracy of obtained results in software effort assessment. Furthermore, we present the minimum value of magnitude relative error (MRE) for selected ANN architectures through orthogonal arrays and the new COCOMO2000 version. The MRE formula measures, for a given project, the difference between actual and estimated effort relative to the actual effort. The numerical value of every examination in data distribution is sensitive to individual predictions. The contribution of this approach lies in the application ANNs, using Taguchi search/optimization arrays and GA criteria in the domain of software effort prediction [11]–[15]. The critical decisions that defined the objective of our research are as follows:

- comparison of two different architectures of artificial neural networks and the obtained results concerning the minimum relative error obtained by the COCOMO formulas (1, 2),
- division of software projects into clusters, given the different nature and values of the actual effort of each, as well as the possibility of application to as many projects as possible,
- finding one of the most efficient methods of encoding and decoding input quantities,
- minimum number of iterations performed,
- testing and validation on different data sets.

This article is structured as follows: Section 2 provides an overview of the previous studies that applied ANNs for software effort estimation. Section 3 explains the Taguchi method for the Neural Network design that is used to formulate the basic idea through three main parts of the experiment: training, testing, and validation of two ANN architectures. Section 4 proposes the evaluation method, and Section 5 discusses the results. The concluding remarks are given in the last section.

II. LITERATURE REVIEW

The efficiency of software effort and cost estimation is one of the contributing factors that aim at developing a successful project. In the last decade, the software industry, the modern world and many researches are examining this key process inside the software development cycle. The following section presents the review of the state-of-the-art work done in this domain. Some of the authors claim that software project management has taken a new perspective of software quality assurance methodologies during all tasks in software development [12], while other authors [13], [14] state that the project manager has a dominant role in completing and satisfying all requirements for project scope, time, cost, risk, and subsequently on quality. Since recently, many researchers usually work with a new costs assessment process based on attractive and useful artificial Neural Networks. Some of the previous effort estimation models, used in various software development projects are: SLIM (Software Life Cycle Management) model [15], SEER-SEM (System Evaluation and Estimation of Resource Software Evaluation Model) [16], and the COCOMO81 (Constructive Cost Model) [17] and COCOMO2000 [18]. The COCOMO model is based on cost factors, scale factors, and software size for estimating effort requirements, cost, and the scheduling of a software development project. The most detailed version of the COCOMO2000 model is the Post Architecture and it is expressed in formulas (1, 2) [17], [18]. This model uses a size measurement and a number of cost drivers (scale factors and effort multipliers) to estimate the amount of the effort required. The estimated effort for software development project is expressed as person-months (PM) and can be calculated as follows:

$$PM = A \cdot Size^{E} \cdot \prod_{j=1}^{n} \cdot EM_i. \tag{1}$$

$$E = B + 0.001 \cdot \sum_{j=1}^{5} \cdot SF_j. \tag{2}$$

A and B are the baseline constants for calibration; KSLOC (thousands of Source Lines of Code) represents the size of the software project; SF_j stands for five scale factors; EM_i stands for seventeen effort multipliers; Cost drivers of the COCOMO2000 Post Architecture model together comprise these scale factors and effort multipliers. In [19], [20] the authors used the Back-Propagation Neural Networks technique through COCOMO dataset & NASA 2 dataset which consist of 60 and 93 projects respectively and the sigmoid function to evaluate software cost. The result showed less MRE value. The pieces of research [21] studied and merged the COCOMO & Neural Network technique into a single structure. It was shown that the COCOMO the evaluated cost is closer to the actual cost. Goyal S. and experiments by other authors listed [22] proposed the ML approach to improve COCOMO Model using ANNs, but the results of MRE were large. An interesting study that was conducted [23] included

the use of a Neural Network algorithm to evaluate the software cost with accuracy but neither of the methods mentioned is confidently better or worse than the other. Authors in [24] have used a two layer feed forward network to propose a model to minimize the MRE between actual cost and evaluated cost. The cost was estimated with 3 experiments: size estimation, cost estimation and time estimation. Authors used an intermediate model which gives better accuracy than the other two models. A study conducted by [25] utilized a new approach uniting k-means algorithm to estimate the software cost through MRE results. The study [26] experimented with a Neural Network technique using a perceptron learning algorithm to evaluate the software cost based on the COCOMO model. Authors in [27] proposed Neural Network model to evaluate the software effort also using a multilayer artificial Neural Network with identity activation function at the input, hidden and the output layer with sustainable result. Authors in [28] proposed a back propagation Neural Network algorithm. They compared their result with the COCOMO model and it improved the result of experimentation. Nassif *et al.* [29] experimented with four different neural network models and investigated the corelation between them, using the mean absolute residual criterion and other paramethers. It was supposed that the regression and UCP models were also used to forecast the software effort from the used experiment diagram. Authors in [30] enhanced the accurateness of software effort estimation using numerical formulas and the calculated result was acceptable with less Mean Magnitude Relative Error (MMRE). Boetticher [31] performed over 33,000 different experiments using Neural Networks on empirical data. Collected data were from separate corporate domains and assessed the contribution of different internal product metrics (size, vocabulary, complexity, and object) using Neural Networks. In [32]–[35] authors applied the artificial Neural Network to cost estimation and claimed that Neural Network is able to infer from a trained data set. Over a set of training data, the Neural Network learning algorithm constructs mappings and fits previously unseen data in an appropriate way. In [36] the authors states that the new idea of a Hybrid approach for improving the accuracy of software cost estimation through estimated result is very close to the actual result using COCOMO2000, Neural network and the Principal Component Analysis (PCA) technique. Sweta Kumari *et al.* [37] gave the pros and cons of basically two types of cost estimation models: algorithmic and non-algorithmic. The shortcomings of previous research are reflected in the lack of systematicity for the design and optimization of Neural Networks in complex systems. Our approach involves the use of ANNs and Taguchi orthogonal vector plans, through which we obtained significantly better results. In all the aforementioned studies, the value obtained for MRE is far greater than the error of estimation in our approach. The advantage is also in the rapid fulfilment of criteria for the completion of ANN training i.e. the number of iterations in our experiment is less than 10.

III. TAGUCHI METHOD FOR NEURAL NETWORK DESIGN

A. TAGUCHI METHOD FOR NEURAL NETWORK DESIGN

There is a huge number of businesses that invest in brand new software. It is necessary to help those companies to consider the cost regarding software development and also the period of time used for improvement. The Neural Network is used as the most prominent model for software estimation and also as an information processing technique. Due to its ability to learn from any dataset, it is possible to generated meaningful results from it. The general structure of ANNs is typically organized in three layers: the input layer, hidden layer and output layer. So, it includes a number of inputs that are applied by some weights which are combined together to give an output. Because of the fact that the number of hidden layers should be as small as possible, to avoid overfitting, we use only one hidden layer. This article presents the efficient application of Neural Networks based on Taguchi's orthogonal vector plans and the required accuracy for the convergence rate [1], [2]. The following steps are applied:

- Construction of ANN based on a matrix of orthogonal arrays and application of weighting factors during the learning process;
- The minimum number of hidden layers within the ANN;
- Improving the accuracy and reliability of the Neural Network; and
- Choice of methodology for convergence to the minimum value of magnitude relative error.

In the design of Neural Networks, the correct selection of design parameters, number of input sizes, hidden layers and output sizes is required, in order to obtain appropriate values for the specified experiment. The main contribution of this article is the application of Neural Networks using Taguchi orthogonal arrays, which allows the analysis of a large number of parameters in the smallest number of experiments. The originality and innovation of our approach is reflected in the application of simple architectures of ANN networks, a small number of observations for the training of ANN networks and rapid convergence - a small number of iterations.

B. ORTHOGONAL ARRAY TUNING METHOD

In this section, we propose the Orthogonal Array Tuning Method (OATM) that always gives good results and requires a much smaller number of experiments to find optimal the solution. When using the OATM method, the hyper-parameters are levels. In our proposed model, we use three levels: L1, L2 and L3 according to work [1] and based on these levels we build the F-L (factor-level) table. The table depends on W_i ($i = 1, 2, \ldots, 13$) in Figure 1 - Single hidden layer Neural Network (ANN27), or OA(27, 3^{13}) in our first proposed ANN structure in Table 1. It requires the number of to-be-tuned factors and the number of levels for each factor W_i to be defined. The levels should be determined by experience and literature [2]. We further suppose each factor has the same number of levels. The OATM is enabled to

optimize the hyper-parameters by utilizing a very small set of highly representative hyper-parameter combinations. The high efficiency can be demonstrated by a simple sample in Table 1. The OATM only takes 27 combinations (27 ANN during training process) which means the hyper-parameters (W_i to W_{13}) can be optimized by running the experiment for 27 times instead of $3^{13} = 1\ 594\ 323$ observations in full factorial plan i.e. exhausted factor combinations experiments. Therefore, through OATM, we can save about 99.99830649% (0.9999830649 = 1 - (27/1 594 323) work in the tuning procedure. The proposed methodology is based on two different kinds of ANN architectures that are summarized as follows: In the presented study, for two proposed Neural Networks, the stated relationship between performance measures and scheduling criteria is defined using weighting factors. This contributes to the accuracy and reliability of effort and time estimation. It was used a back propagation Neural Network for the chosen architectures. The first proposed architecture of a single hidden layer Neural Network is based on Taguchi Orthogonal Array (L27) with 13 parameters (W_i, i = 1, 2, . . . , 13) and three different levels: L1, L2 and L3, shown in Figure 1. Input nodes are COCOMO attributes from equation (1): X_1 = E, X_2 = EAF (Effort Adjustment Factor) i.e. multiplied EM_i, and X_3 = Size [KLOC] according to formula (1). After construction of the ANN

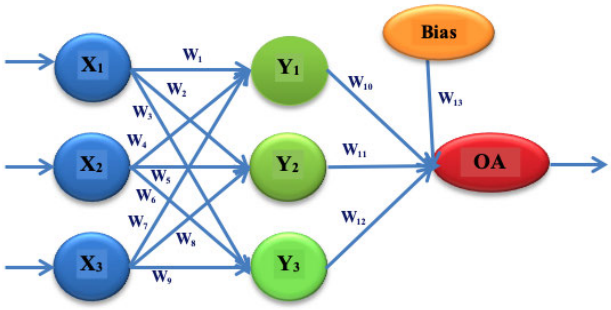

FIGURE 1. Single hidden layer Neural Network (ANN27).

architecture, we have to define the Orthogonal Array Tuning table i.e. Table 1 in this experiment. The presented table should heed the basic composition principles. Some commonly used tables are presented in references. An alternative way is to use the software which generates orthogonal array tables.

The second proposed architecture of single hidden layer Neural Network is based on Taguchi Orthogonal Array (L18) with 8 parameters (W_i, i = 1, 2, . . . , 8) and three different levels: L1, L2 and L3, shown in Figure 2. Contrary to the previous architecture, this one is combined network, because the first parameter has only two levels Table 2.

The essential concern is to get an ideal neural organization structure given a run of conceivable plan parameters and not to get a point by point understanding of the neural organization itself. As such, the Taguchi method will help to precisely determine every experiment. It makes a difference to decide

TABLE 1. Taguchi Orthogonal Array OA(3^{13}) for ANN27

Test	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1
2	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2	L2	L2	L2
3	L1	L1	L1	L1	L3	L3	L3	L3	L3	L3	L3	L3	L3
4	L1	L2	L2	L2	L1	L1	L1	L2	L2	L2	L3	L3	L3
5	L1	L2	L2	L2	L2	L2	L2	L3	L3	L3	L1	L1	L1
6	L1	L2	L2	L2	L1	L1	L1	L3	L3	L3	L2	L2	L2
7	L1	L3	L3	L3	L1	L1	L1	L3	L3	L3	L2	L2	L2
8	L1	L3	L3	L3	L2	L2	L2	L1	L1	L1	L3	L3	L3
9	L1	L3	L3	L3	L3	L3	L3	L2	L2	L2	L1	L1	L1
10	L2	L1	L2	L3	L1	L2	L3	L1	L2	L3	L1	L2	L3
11	L2	L1	L2	L3	L2	L3	L1	L2	L3	L1	L2	L3	L1
12	L2	L1	L2	L3	L3	L1	L2	L3	L1	L2	L3	L1	L2
13	L2	L2	L3	L1	L1	L2	L3	L2	L3	L1	L3	L1	L2
14	L2	L2	L3	L1	L2	L3	L1	L3	L1	L2	L1	L2	L3
15	L2	L2	L3	L1	L3	L1	L2	L1	L2	L3	L2	L3	L1
16	L2	L3	L1	L2	L1	L2	L3	L3	L1	L2	L2	L3	L1
17	L2	L3	L1	L2	L2	L3	L1	L1	L2	L3	L3	L1	L2
18	L2	L3	L1	L2	L3	L1	L2	L2	L3	L1	L1	L2	L3
19	L3	L1	L3	L2	L1	L3	L2	L1	L3	L2	L1	L3	L2
20	L3	L1	L3	L2	L2	L1	L3	L2	L1	L3	L2	L1	L3
21	L3	L1	L3	L2	L3	L2	L1	L3	L2	L1	L3	L2	L1
22	L3	L2	L1	L3	L1	L3	L2	L2	L1	L3	L3	L2	L1
23	L3	L2	L1	L3	L2	L1	L3	L3	L2	L1	L1	L3	L2
24	L3	L2	L1	L3	L3	L2	L1	L1	L3	L2	L2	L1	L3
25	L3	L3	L2	L1	L1	L3	L2	L3	L2	L1	L2	L1	L3
26	L3	L3	L2	L1	L2	L1	L3	L1	L3	L2	L3	L2	L1
27	L3	L3	L2	L1	L3	L2	L1	L2	L1	L3	L1	L3	L2

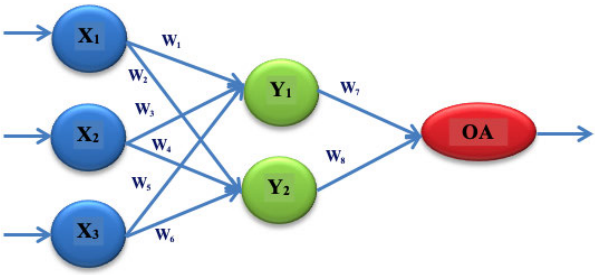

FIGURE 2. Single hidden layer Neural Network (ANN18).

TABLE 2. Taguchi Orthogonal Array ($2^1 + 3^7$)for ANN18

Test	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8
1	L1	L1	L1	L1	L1	L1	L1	L1
2	L1	L1	L2	L2	L2	L2	L2	L2
3	L1	L1	L3	L3	L3	L3	L3	L3
4	L1	L2	L1	L1	L2	L2	L3	L3
5	L1	L2	L2	L2	L3	L3	L1	L1
6	L1	L2	L3	L3	L1	L1	L2	L2
7	L1	L3	L1	L2	L1	L3	L2	L3
8	L1	L3	L2	L3	L2	L1	L3	L1
9	L1	L3	L3	L1	L3	L2	L1	L2
10	L2	L1	L1	L3	L3	L2	L2	L1
11	L2	L1	L2	L1	L1	L3	L3	L2
12	L2	L1	L3	L2	L2	L1	L1	L3
13	L2	L2	L1	L2	L3	L1	L3	L2
14	L2	L2	L2	L3	L1	L2	L1	L3
15	L2	L2	L3	L1	L2	L3	L2	L1
16	L2	L3	L1	L3	L2	L3	L1	L2
17	L2	L3	L2	L1	L3	L1	L2	L3
18	L2	L3	L3	L2	L1	L2	L3	L1

the particular plan data required for the optimization prepare for a least required number of tests.

As outlined within the primary steps underneath, the Taguchi method proposes an approach to arrange and plan

ideal neural systems efficiently. The architectures of single hidden layer networks shown in Figure 1 and Figure 2 have been processed through seven steps within Algorithm1.

Algorithm 1 Robust Model Designing

Step 1. Input layer.
Step 2. Coding method of input layer.
Step 3. Hidden layer functions and output layer.
Step 4. Decoding output values.
Step 5. Gradient Descent (GA) and determining Winner network.
Step 6. Training and Testing via Winner network.
Step 7. Validation using Kemerer dataset and random dataset composed of projects from different sources.

Step 1: Input layer

Based on the latest version of COCOMO2000 projects which a is publicly available dataset [Boehm *et al.* 00], the estimated effort in person-months (PM) is divided into three clusters: less than 90PM (small projects), between 90PM and 500PM (medium projects) and more than 500PM (large projects). The proposed architectures have 3 input variables, as we already stated ($X_1 = E$, $X_2 = PEM_i$ (Effort Multipliers Product), and $X_3 =$ Size [KLOC]) and 13 weights for ANN27 network, or 8 for ANN18 network, $W_i = 1$ ($i = 1$ to 13; $i = 1$ to 8), gradient descent (GA = 0.001) and the identity activation function is a sigmoid function to calculated the desired output of the network.

Step 2: Coding method of input layer

We are coding input values to increase accuracy as follows:

$$X_i = (X_i - X_{min})/(X_{max} - X_{min}). \quad (3)$$

Step 3: Hidden layer functions and output layer

By using a sigmoid function with formulas (4) – (7) in the hidden layer, an estimated value is obtained:

$$Y_1 = 1/(1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_4 + X_3 \cdot W_7)}). \quad (4)$$
$$Y_2 = 1/(1 + e^{-(X_1 \cdot W_2 + X_2 \cdot W_5 + X_3 \cdot W_8)}). \quad (5)$$
$$Y_3 = 1/(1 + e^{-(X_1 \cdot W_3 + X_2 \cdot W_6 + X_3 \cdot W_9)}). \quad (6)$$
$$EstEffNN = 1/(1 + e^{-(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + 1 \cdot W_{13})}). \quad (7)$$

Step 4: Decoding output values

Based on proposed method in Step 2. the proportional equation (denormalization) is given like (8):

$$X_i = (X_{min} + X_i(coded)) \cdot (X_{max} - X_{min}). \quad (8)$$

Step 5: Gradient Descent (GA) and determining Winner network After completing the number of iterations and output values (MRE) are lower than 0.001 (GA < 0.001) we have reached the stop criteria and we choose "Winner" NN with the smallest MRE value.

Step 6: Training and Testing via Winner network Each ANN re-enacted parametric run is proportionate to a test run (a "design of experiments" in terms of robust design terminology) which constitutes the preparatory stage for certain ANN. The number of ANN rises to the number of columns of the OA (which demonstrates the number of tests). The weights are gotten from the Orthogonal Arrays by supplanting the (ordered) level of each parameter (weight) with its current esteem within the look interim [1]. Controllable variables, in our experiment, for every search start are at the [−1, 1] interval for all weights W_i. The three levels of Wi (level L1, L2, L3) within the comparing OA can be related to the extremes and the center of the look interim, −1, 0, 1. In this way, we are supplanting the level number in Table 1 and Table 2 with the actual value of the weight at that level. For weight W_i, at the primary cycle, level L1 from Table 1 for ANN27 (L1 level of weight Wi) is supplanted by −1, level L2 is supplanted by 0, and level L3 is supplanted by 1. Thereafter a cost function is to be evaluated, see for each network in the test set. In our experiment it is the sum of magnitude relative error (MRE) between the network output (estimated effort value) and target output value (actual effort value). The cost to be minimized is the sum of cost functions for all ANNs from proposed architectures. For the primary iteration the real fetched esteem for each ANN recreation run (calculated as whole of greatness relative mistake for the given preparing set) is at that point utilized to assess the impact that each individual level had on the fetched work. This is often done by summing the fetched capacities for the systems (tests) in which that level was shown. Consider as an instance ANN weights W_1, Level L1 (of W_1) enters the primary nine networks (experiments) of L27. The cumulated consequences of Level L1 of W_1 (Eff L1,W_1) are given through the sum of prices for the networks wherein L1 intervenes. Thus Eff L1 $W_1 = \text{cost1} + \text{cost2} + \ldots + \text{cost9}$ [1].

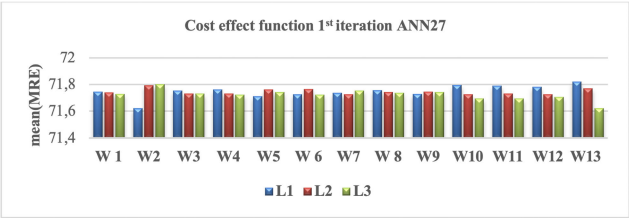

FIGURE 3. Cost effect function for W_1 to W_{13} on 3 levels (L1, L2, L3) in 1. iteration.

For this example, after the first iteration, the effect of each level value on the cost is synthetized in Figure 3. For next, 2. Iteration, modify the values of each weight (shrink the search interval for the weight) in the direction of the weight value that produced the minimum cost. From Figure 3, comparing the effects of the 3 levels of W_1, one notices that Level L3 of weight W_1 produces minimum cost, therefore the weights will move in its direction. The new levels of W_1, in the 2nd iteration, it is calculated as (9) – (11):

$$W_1L1new = W_1L2old. \quad (9)$$
$$W_1L2new = W_1L2old + (W_1L3old - W_1L2old)/2. \quad (10)$$
$$W_1L3new = W_1L3old. \quad (11)$$

Reducing the search interval at 1/2 its value and choosing to continue the search within the half that's highest to the extent of the load that created minimum cost could be an easy approach to calculate the new weight levels. By dividing the interval in two at every iteration, the search converges extremely quickly (the search interval for a weight shrinks three orders of magnitude in ten steps). Table 3 illustrates the

TABLE 3. Example of Convergence Rate for ANN27 on Medium Cluster

Iterations	1st		2nd		3rd		4th		5th	
ANN27	MRE	GA	MRE	GA	MRE	GA	MRE	GA	MRE	GA
1	0.4384	0.4384	0.4372	0.0012	0.4339	0.0032	0.4333	0.0007	0.4315	0.0018
2	0.6031	0.6031	0.4727	0.1304	0.4417	0.0310	0.4356	0.0061	0.4326	0.0030
3	1.1374	1.1374	0.9861	0.1513	0.5023	0.4838	0.4440	0.0583	0.4343	0.0097
4	1.0667	1.0667	0.6592	0.4076	0.4875	0.1717	0.4445	0.0430	0.4358	0.0086
5	0.4370	0.4370	0.4352	0.0018	0.4338	0.0013	0.4333	0.0005	0.4319	0.0014
6	0.5181	0.5181	0.4465	0.0716	0.4348	0.0117	0.4328	0.0020	0.4317	0.0011
7	0.7090	0.7090	0.5020	0.2069	0.4495	0.0525	0.4388	0.0107	0.4334	0.0054
8	1.0269	1.0269	0.6031	0.4238	0.4682	0.1349	0.4405	0.0277	0.4341	0.0065
9	0.4434	0.4434	0.4344	0.0090	0.4324	0.0020	0.4320	0.0004	0.4310	0.0010
10	0.9410	0.9410	0.5722	0.3688	0.4605	0.1117	0.4398	0.0207	0.4336	0.0061
11	0.4487	0.4487	0.4370	0.0117	0.4345	0.0025	0.4331	0.0014	0.4322	0.0009
12	0.6236	0.6236	0.4900	0.1336	0.4451	0.0449	0.4371	0.0080	0.4329	0.0042
13	0.4899	0.4899	0.4539	0.0360	0.4386	0.0153	0.4346	0.0040	0.4327	0.0020
14	0.7199	0.7199	0.4906	0.2292	0.4445	0.0461	0.4366	0.0079	0.4329	0.0038
15	0.5942	0.5942	0.4630	0.1312	0.4382	0.0249	0.4341	0.0040	0.4320	0.0021
16	0.4917	0.4917	0.4466	0.0451	0.4365	0.0100	0.4333	0.0032	0.4318	0.0015
17	0.7495	0.7495	0.5213	0.2281	0.4499	0.0714	0.4382	0.0117	0.4332	0.0051
18	0.5713	0.5713	0.4639	0.1074	0.4422	0.0217	0.4359	0.0063	0.4330	0.0029
19	0.6980	0.6980	0.4762	0.2218	0.4412	0.0349	0.4354	0.0058	0.4326	0.0028
20	0.9032	0.9032	0.5770	0.3262	0.4621	0.1149	0.4398	0.0223	0.4338	0.0060
21	0.4457	0.4457	0.4388	0.0069	0.4353	0.0035	0.4339	0.0013	0.4323	0.0016
22	0.6218	0.6218	0.4893	0.1325	0.4444	0.0449	0.4366	0.0078	0.4325	0.0041
23	0.4670	0.4670	0.4458	0.0212	0.4385	0.0074	0.4345	0.0039	0.4324	0.0021
24	0.7483	0.7483	0.5218	0.2265	0.4519	0.0698	0.4395	0.0124	0.4338	0.0057
25	0.5832	0.5832	0.4712	0.1120	0.4432	0.0280	0.4362	0.0070	0.4331	0.0032
26	0.5181	0.5181	0.4536	0.0645	0.4383	0.0153	0.4340	0.0043	0.4323	0.0018
27	0.6804	0.6804	0.4860	0.1944	0.4430	0.0430	0.4361	0.0069	0.4326	0.0035
Winner MRE	0.4370	27	0.4344	23	0.4324	21	0.4320	8	0.4310	0
Min. MMRE	0.6546		0.5065		0.4471		0.4364		0.4328	

quick convergence to the target worth of a given cost, therefore leading to a fast learning method that satisfies GA criteria. The Winner ANN among twenty-seven trained ANNs is one that is a calculable mean MRE. In the presented table, the minimum magnitude relative error for Winner NN5 is 0.4370 (43.70%) and the minimum mean magnitude relative error is 0.6546 (65.46%). Also, the number of Neural Networks that have a GA that is not less than 0.001 is 27, 23, 21, 8 and 0, respectively.

Step 7. Validation using Kemerer dataset and random dataset composed of projects from different sources. The first experiment is a training process, which is performed on three clusters. A training data set includes values from different projects in every cluster based on the two presented Neural Network architectures. As a result of the first experiment, it is possible to recognize the ''Winner'' Neural Network. The Neural network that is considered as a ''Winner'' is the one with the smallest value of MRE. Second experiment is a testing experiment, obtained on previously, founded ''Winner'' Neural Network. The third experiment is a process of validation. This experiment is performed on the Kemerer validation dataset divided in three clusters like as in step 1. In his papers [22], [38] he used data collected from 63 completed software projects to produce results. He also recognized the three main attributes which affect the results of the software productivity. For a predictive capability of effort estimation, it was necessary to test the Winner network for each cluster on the Kemerer data set. The second dataset that was used for validation consists of 25 projects divided also in three clusters based on the criteria in the first step. The random dataset which was composed of projects from different resources like: Aerospace, Loc. Calibration, etc. is used because there were projects that are not cast-off in previous datasets.

IV. EVALUATION METHOD

Software price estimation models ought to be quantitatively evaluated in terms of estimation accuracy to enhance the modelling process. Some rules or the measurements should be provided for the purpose of model assessment. This measurement of accuracy defines the calculabe results, no mather how shut they are with their actual value. Computer code cost estimates play important role in delivering software projects. As a result, researchers have projected the most widely used criterion of analysis to assess the performance of software prediction models, that is, the mean magnitude of relative error (MMRE), to gauge the luxuriousness of prediction systems. MMRE is typically computed by following commonplace analysis processes cross-validation. Comparisons will be created across information sets and prediction model sorts [39]. COCOMO computes effort based on the number of lines of codes. In intermediate COCOMO, Boehm [17], [18] used fifteen additional predictor variables known as price drivers, that are needed to calibrate the nominal effort of a project to a particular project environment. The values are set to every cost driver in step with the properties of a particular computer code project. These numerical values of 15 cost drivers in COCOMO81 and seventeen cost drivers in COCOMO2000 are increased to induce the trouble adjustment factor, that is, EAF. The performance of estimation [40] strategies is sometimes evaluated by many quantitative relation measurements of accuracy metrics [41] as well as RE (relative error), MRE (magnitude of relative error), MAE (mean absolute error) [42], and MMRE (mean magnitude of relative error) that are calculated as follows (12) – (16):

$$Deviation = |ActEffort - EstEffort|. \tag{12}$$

$$MAE_i = \frac{1}{n}\sum_{i=1}^{n} |ActEffort - EstEffort|. \tag{13}$$

$$MRE = Deviation/ActEffort. \tag{14}$$

$$MMRE = mean(MRE). \tag{15}$$

$$PRED(k) = count(MRE) < 25\%. \tag{16}$$

Decreasing of MMRE and increasing of PRED (Prediction) are the main aims of all estimation techniques.

V. DISCUSSION AND RESULTS

The proposed OATM is evaluated over three different tasks on three benchmark datasets. In order to carry out our experiments to evaluate the efficiency of our proposed model, we have chosen four datasets: For neural network training we chose the COCOMO2000 model consisting of 100 projects found on MATLAB repository (see www.mathworks.com), testing includes twenty COCOMO2000 projects, for validation we used NASA60 projects and a random dataset consisting of COCOMO81 [17] and Kemerer 15 projects [38], shown in Table 4.

TABLE 4. Datasets That are Used for all Three Parts of the Experiment

	Dataset name	Number of projects	Purpose
Dataset_1	COCOMO2000	100	Training
Dataset_2	COCOMO2000	20	Testing
Dataset_3	NASA60	60	Validation
Dataset_4	Random=COCOMO81+Kemerer	81+15	Validation

The first COCOMO2000 dataset is divided into a training set (80%) and a testing set (20%). All the networks were trained for maximum 10 iterations and the most accurate results were considered as a Winner ANN. To evaluate the proposed OATM, we designed an extensive experiment to tune the hyper-parameters.

1. The first experiment involves training two different architectures (ANN18 and ANN27) that were presented in previous section. For the proposed ANN27 architecture that is obtained on 100 projects from the COCOMO2000 dataset, MMRE results for all 27 estimators and the selected Winner network in accordance with a small, medium and large cluster, show the following: The lower MMRE values is reached with a medium cluster (43.2%), with Winner network performed by NN5 (43.1%), while the highest MMRE value was obtained over a small cluster (59.8%) with NN5 estimator (59.7%). The large cluster has the MMRE value between a small and medium cluster and amounts 46.7% and Winner network NN9 with 46.5% Table 5. Gradient descent (GA < 0.001) was achieved in the eighth iteration for the large cluster and small cluster, while for the medium cluster this has already been achieved in the sixth iteration, shown in Figure 4.

TABLE 5. MMRE Values for Each Selected Cluster and Winner Network (ANN27)

ANN27, number of iterations		1st	2nd	3rd	4th	5th	6th	7th	8th
Small cluster	MMRE(%)	102.4	75.7	67.0	64.3	63.0	60.4	60.1	59.8
Small cluster	Winner NN5(%)	61.2	61.0	61.0	60.9	59.9	59.8	59.7	59.7
Medium cluster	MMRE(%)	65.5	50.6	44.7	43.6	43.3	43.2	43.2	43.1
Medium cluster	Winner NN5(%)	43.7	43.4	43.2	43.2	43.1	43.1	43.1	43.1
Large cluster	MMRE(%)	334.0	148.4	77.5	50.9	49.9	47.0	46.8	46.7
Large cluster	Winner NN9(%)	60.3	56.9	52.9	59.3	53.3	48.4	46.7	46.5

For the proposed ANN18 architecture that is obtained on 100 projects from COCOMO2000 dataset, MMRE results for all 18 estimators and the selected Winner network in accordance with a small, medium and large cluster, show the following: The lower MMRE values are reached with medium cluster (44.2%), with Winner network performed by NN5 (43.8%), while the highest MMRE value was obtained over a large cluster (138.0%) with NN5 estimator (134.0%). The small cluster has the MMRE value of 63.7% and Winner network NN5 with 63.3% Table 6. Gradient descent (GA < 0.001) was achieved in the eighth iteration for the large cluster, while for the small cluster it was achieved in the sixth iteration, and this has already been achieved in the fifth iteration for the medium cluster, shown in Figure 5.

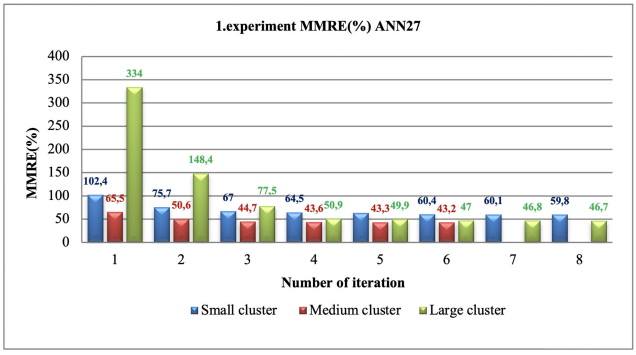

FIGURE 4. Graphical representation of all performed iterations on ANN27 architecture, for each selected cluster.

TABLE 6. MMRE Values for Each Selected Cluster and Winner Network (ANN18)

ANN18, number of iterations		1st	2nd	3rd	4th	5th	6th	7th	8th
Small cluster	MMRE (%)	93.0	75.0	67.8	65.3	64.1	63.7		
Small cluster	Winner ANN5(%)	67.0	64.8	63.9	63.5	63.4	63.3		
Medium cluster	MMRE (%)	61.4	50.7	46.3	44.7	44.2			
Medium cluster	Winner ANN5(%)	45.9	44.6	44.1	44.0	43.8			
Large cluster	MMRE (%)	320.5	234.0	187.3	161.8	146.6	141.1	140.6	138.0
Large cluster	Winner ANN5(%)	170.5	153.5	145.7	139.6	137.1	136.0	135.0	134.0

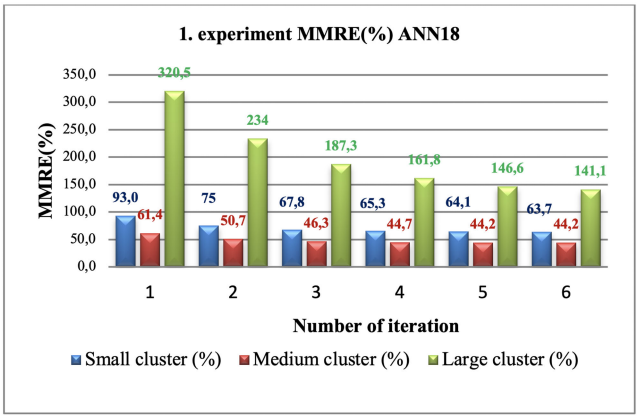

FIGURE 5. Graphical representation of all performed iterations on ANN18 architecture, for each selected cluster.

After the first experiment, which involved training, it can be seen that all clusters achieve better results using the two proposed architectures ANN27 and ANN18 compared

to COCOMO2000, with the ANN27 network giving better results than the ANN18 network, shown in Table 7, Figure 6.

TABLE 7. Values of MMRE of Each Selected Cluster for Both Architectures in the 1st Experiment [4], [22]

	MMRE				
	ANN18(%)	Winner18(%)	ANN27(%)	Winner27(%)	COCOMO2000(%)
Small cluster	63.7	63.1	59.8	59.7	455.2
Medium cluster	44.2	43.8	43.3	43.1	80.1
Large cluster	138.0	136.0	44.9	44.5	43.9

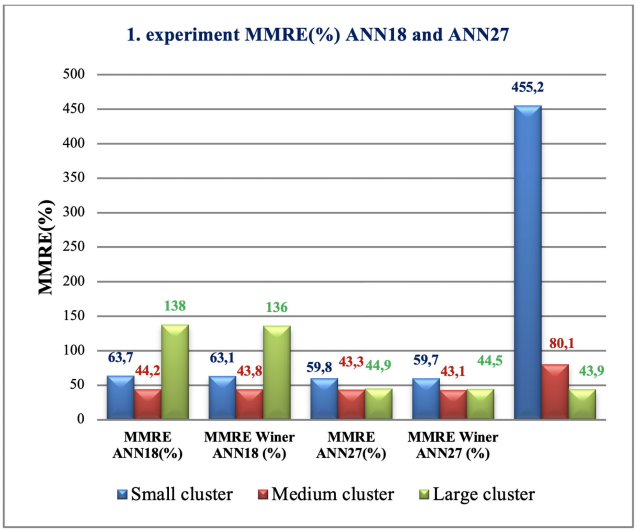

FIGURE 6. Graphical representation of MMRE values for both architectures in 1st experiment.

2. In the second experiment testing is performed on projects from the COCOMO2000 dataset (with ID from 101 to 120). For the small and medium cluster of ANN27 Neural Network, Winner network NN5 was used, while for the large cluster, Winner network NN9 was used. In each cluster of ANN18 Neural Network, Winner network NN5 was used. Obtained results are presented below in Table 8, Figure 7 and after testing the experiment, it can be concluded that the results of MMRE from ANN27 Neural Networks are better than the ones in ANN18 Neural Network. For example, in the case of a large cluster (projects with actual effort greater than 500PM) the MMRE using ANN18 is 163.5%, but with ANN27 the MMRE is almost eight times smaller (22.7%).

TABLE 8. Values of MMRE of Each Selected Cluster for Both Architectures in the 2nd Experiment

	MMRE	
	ANN18 WinnerNN5(%)	ANN27 WinnerNN5(%)
Medium cluster	48.9	41.9
Large cluster	43.0	38.9
Large cluster	163.5	22.7

3. The third experiment (validation) is obtained on the Kemerer dataset and random dataset, that include 63 projects

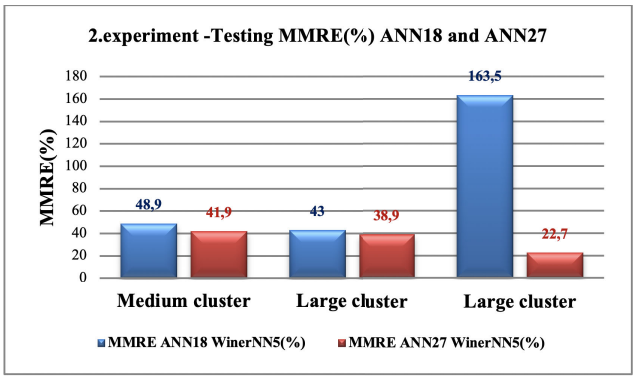

FIGURE 7. Graphical representation of MMRE values for both architectures in 2nd experiment.

and 25 projects, respectively. Projects in the Kemerer dataset are divided into three clusters: the small cluster (contains 30 projects), the medium cluster (contains 17 projects), and and large cluster (contains 17 projects). Projects in the random dataset are divided into three clusters: the small cluster (contains 11 projects), the medium cluster (contains 10 projects), and the large cluster (contains 4 projects). The following validation results were obtained for both architectures, using the appropriate Winner networks, shown in Table 9, Figure 8, Table 10, Figure 9.

TABLE 9. Values of MMRE (Kemerer dataset) of Each Selected Cluster for Both Architectures in the 3rd Experiment

	MMRE	
	ANN18 WinnerNN5(%)	ANN27 WinnerNN5(%)
Small cluster	66.3	66.1
Medium cluster	45.3	36.8
Large cluster	67.1	54.4

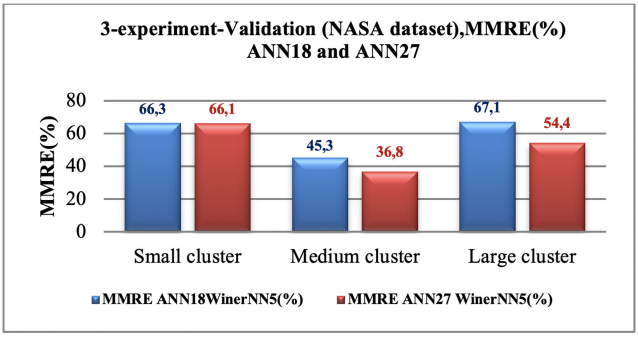

FIGURE 8. Graphical representation of MMRE (Kemerer dataset) values for both architectures in the 3rd experiment.

In both validation sets, the best results are achieved with the ANN27 network over the middle cluster and the MMRE is then at its lowest: for the Kemerer set it is 36.8%, while for the random set, it is 33.6%.

A. SHORT OVERVIEW OF THE RESULTS FOR ALL THREE PARTS OF THE EXPERIMENT

The results of all three parts of the experiment for the two representations of the architecture (ANN27 and ANN18),

TABLE 10. Values of MMRE (random dataset) of each selected cluster for both architectures in the 3ed experiment [4], [22].

	MMRE		
	ANN18(%)	ANN27(%)	COCOMO2000(%)
Small cluster	49.4	48.9	455.2
Medium cluster	34.3	33.6	80.1
Large cluster	55.6	29.9	43.9

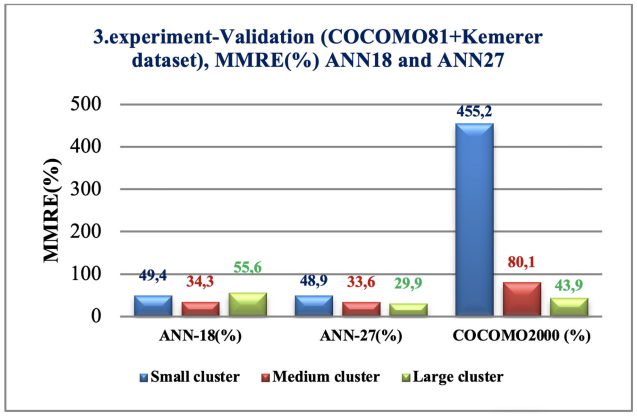

FIGURE 9. Graphical representation of MMRE (random dataset) values for both architectures in the 3rd experiment.

TABLE 11. Values of MAE and MMRE of Each Selected Cluster for Both Architectures in all Three Parts of the Experiment in Relation to COCOMO2000 [4], [22]

Cluster	Parts of the experiments	MAE		MMRE		
		ANN18[PM]	ANN27[PM]	ANN18(%)	ANN27(%)	COCOMO2000(%)
Small cluster	Training	25.9	11.4	63.7	59.9	455.2
Small cluster	Testing	35.3	14.5	48.9	41.9	455.2
Small cluster	Validation random Dataset	23.5	14.9	49.4	48.9	455.2
Small cluster	Validation Kemerer	38.9	14.3	66.3	66.1	455.2
Medium cluster	Training	182.2	117.6	44.2	43.2	80.1
Medium cluster	Testing	172.4	162.5	43	38.9	80.1
Medium cluster	Validation random Dataset	185.9	160.6	34.3	33.6	80.1
Medium cluster	Validation Kemerer	183.1	158.8	45.3	36.8	80.1
Large cluster	Training	1512.2	286.2	138	46.7	43.9
Large cluster	Testing	3019.3	1104.1	163.5	22.7	43.9
Large cluster	Validation random Dataset	3043.9	1813.5	55.6	29.9	43.9
Large cluster	Validation Kemerer	3015.9	1028.9	67.1	54.4	43.9

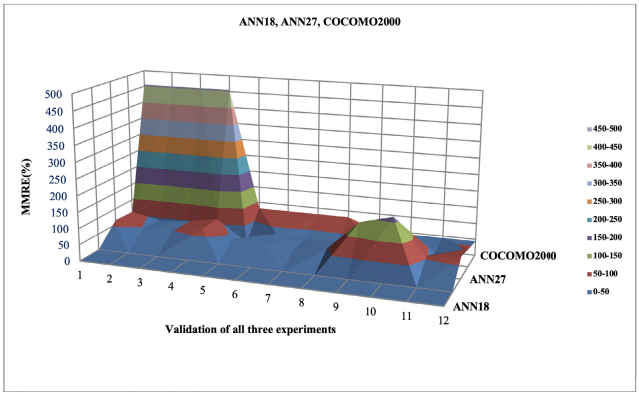

FIGURE 10. Graphical representation of MMRE (random dataset) values for both architectures in the 3rd experiment.

by clusters in relation to the COCOMO2000 values are as follows: It can be observed that the best results were achieved within the medium clusters for both architectures, even twice smaller estimated magnitude relative error compared to COCOMO2000 estimated results. With a small cluster for both architectures, the results were even seven times better than with COCOMO2000. For a large cluster, the results for ANN27 are approximately the same as for COCOMO2000, while for ANN18 the results obtained for a large cluster are three times worse than COCOMO2000, shown in Figure 10, Table 11.

VI. CONCLUSION

This study aimed to improve the accuracy of software evaluation methods. This article compares the impact of architectures of nonparametric models (two presented artificial Neural Network architectures) in relation to traditional parametric models such as COCOMO2000 for the purpose of estimating software effort and costs. Actual, published datasets (COCOMO81, COCOMO2000 and Kemerer datasets) were used for comparison [17], [18], [38]. In the decision-making process, a relationship exists between the performance measures and also the planning criteria. This relationship may be in terms of weights. These weights represent the relative importance of the scheduling criteria. The factors and their associated weights outline the operational policy. During this experimental study, a backpropagation Neural Network was chosen to determine the connection between the worldwide performance measures with regard to the operational policies and the work orders to be scheduled. A Neural Network approach was chosen for the most part as a result of analytical and simulation approaches that are neither sensible nor price-effective. The results that we obtain show that through three parts of the experiment: training, testing, and validation, the two mentioned Neural Network architectures give better results in achieving the minimum MMRE when assessing software effort compared to the use of traditional methods. Mean magnitude relative error values (MMRE) for ANN27 architecture compared to ANN18 architecture, give better results, in essence, lower value of relative error, so they can be considered the best-proposed technique of nonparametric models for software effort estimation. Also, as effort assessment is in practice the most restrictive process, additional criteria can be defined through these experiments, such as the introduced search optimization Gradient Descent (GA). In order for the whole process to be complete, it was important to validate all the results on different data sources in our experiment Kemerer and randomly selected datasets. The main advantages of this approach are as follows: a small number of iterations, a simple ANN architecture, more efficient and precise estimation, less magnitude relative error value, clustering due to the nature of software projects, the choice of a reliable coding and encoding method, great application possibilities, to achieve more reliable results. A possible disadvantage would be validation on additional datasets. There are no major limitations in the application of this approach because it covers a wide range of project values. In such experiments, large software companies may have several development teams to deal with the various stages of

this process. This approach can benefit managers, software engineers, and test engineers. The clustering technique made it possible to set clear boundaries when estimating the appropriate actual effort values, which resulted in actual rather than adjusted results. Future research will be focused on the application of different ANN architectures and the training, testing, and validation of such architectures on other datasets, value of the cost-effect function and other parameters. Error minimization for multiple input values using orthogonal vector plans will be considered. The Taguchi technique used in this analysis provides a scientific improvement to analyze the accuracy of the backpropagation Neural Network once given a collection of input data, and to lift the sensitivity of the network when it encounters different levels. It involves also the utilization of the orthogonal arrays to formulate the matrix experiments that provide an additional reliable estimate of the planning factors. In addition, fewer experiments are required with different strategies for the design of experiments. This technique includes a linear graph technique which permits the designer to check the results of interactions between the design factors.

REFERENCES

[1] A. Stoica and J. Blosiu, "Neural Leaming using orthogonal arrays," *Adv. Intell. Syst.*, vol. 41, p. 418, Jan. 1997.

[2] J. F. C. Khaw, B. S. Lim, and L. E. N. Lim, "Optimal design of neural networks using the Taguchi method," *Neurocomputing*, vol. 7, no. 3, pp. 225–245, Apr. 1995.

[3] N. Ghatasheh, H. Faris, I. Aljarah, and R. M. H. Al-Sayyed, "Optimizing software effort estimation models using firefly algorithm," 2019, *arXiv:1903.02079*. [Online]. Available: http://arxiv.org/abs/1903.02079

[4] A. A. Fadhil, R. G. H. Alsarraj, and A. M. Altaie, "Software cost estimation based on dolphin algorithm," *IEEE Access*, vol. 8, pp. 75279–75287, 2020, doi: 10.1109/ACCESS.2020.2988867.

[5] M. Hammad and A. Alqaddoumi, "Features-level software effort estimation using machine learning algorithms," in *Proc. Int. Conf. Innov. Intell. Informat., Comput., Technol. (3ICT)*, Nov. 2018, pp. 1–3, doi: 10.1109/3ICT.2018.8855752.

[6] S. Shukla and S. Kumar, "Applicability of neural network based models for software effort estimation," in *Proc. IEEE World Congr. Services (SERVICES)*, Jul. 2019, pp. 339–342, doi: 10.1109/SERVICES.2019.00094.

[7] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," *IEEE Trans. Softw. Eng.*, vols. SE–9, no. 6, pp. 639–648, Nov. 1983.

[8] V. Yurdakurban and N. ErdoGan, "Comparison of machine learning methods for software project effort estimation," in *Proc. 26th Signal Process. Commun. Appl. Conf. (SIU)*, May 2018, pp. 1–4, doi: 10.1109/SIU.2018.8404495.

[9] Y. Mahmood, N. Kama, A. Azmi, and M. Ali, "Improving estimation accuracy prediction of software development effort: A proposed ensemble model," in *Proc. Int. Conf. Electr., Commun., Comput. Eng. (ICECCE)*, Jun. 2020, pp. 1–6, doi: 10.1109/ICECCE49384.2020.9179279.

[10] C. ShekharYadav and R. Singh, "Tuning of COCOMO II model parameters for estimating software development effort using GA for PROMISE project data set," *Int. J. Comput. Appl.*, vol. 90, no. 1, pp. 37–43, Mar. 2014.

[11] R. Setiono, K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens, "Software effort prediction using regression rule extraction from neural networks," in *Proc. 22nd IEEE Int. Conf. Tools with Artif. Intell.*, vol. 2, Oct. 2010, pp. 45–52.

[12] T. R. G. Nair, V. Sharma, and S. Kumar, "Impact analysis of allocation of resources by project manager on success of software projects," 2014, *arXiv:1407.5319*. [Online]. Available: http://arxiv.org/abs/1407.5319

[13] L. Lazić and N. Mastorakis, "Two novel effort estimation models based on quality metrics in Web projects," *WSEAS Trans. Inf. Sci. Appl.*, vol. 7, no. 7, pp. 923–934, Jul. 2010.

[14] L. J. Lazić, I. Đokić, and S. Milinković, "Quantitative model for allocation of resources based on success rate of software projects using design of experiments," in *Proc. 7th Eur. Comput. Conf. (ECC)*, Dubrovnik, Croatia, Jun. 2013, pp. 264–269.

[15] L. H. Putnam and W. Myers, *Measures For Excellence: Reliable Software On Time, Within Budget*. Upper Saddle River, NJ, USA: Prentice-Hall Professional Technical Reference, 1991.

[16] D. D. Galorath and M. W. Evans, *Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance*. Boca Raton, FL, USA: CRC Press, 2006.

[17] B. Barry, *Software Engineering Economics*, 1st ed. New York, NY, USA: Prentice-Hall, Sep. 1981.

[18] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, *Software Cost Estimation With Cocomo II*, vol. 1. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.

[19] S. Pundhir, U. Ghose, and U. Bisht, "Performance evaluation of various ANN architectures using proposed cost function," in *Proc. 8th Int. Conf. Rel., Infocom Technol. Optim. (Trends Future Directions) (ICRITO)*, Jun. 2020, pp. 732–737, doi: 10.1109/ICRITO48877.2020.9197851.

[20] A. Kaushik, A. Soni, and R. Soni, "A simple neural network approach to software cost estimation," *Global J. Comput. Sci. Technol.*, vol. 6, no. 2, pp. 117–125, Feb. 2010.

[21] G. Kumar and P. K. Bhatia, "Automation of software cost estimation using neural network technique," *Int. J. Comput. Appl.*, vol. 98, no. 20, pp. 11–17, Jul. 2014.

[22] S. Goyal and A. Parashar, "Machine learning application to improve COCOMO model using neural networks," *Int. J. Inf. Technol. Comput. Sci.*, vol. 10, no. 3, pp. 35–51, Mar. 2018.

[23] D. Manikavelan and R. Ponnusamy, "Software cost estimation by analogy using feed forward neural network," in *Proc. Int. Conf. Inf. Commun. Embedded Syst. (ICICES)*, Feb. 2014, pp. 1–5.

[24] S. Mukherjee and R. K. Malu, "Optimization of project effort estimate using neural network," in *Proc. IEEE Int. Conf. Adv. Commun., Control Comput. Technol.*, May 2014, pp. 406–410.

[25] O. F. Sarac and N. Duru, "A novel method for software effort estimation: Estimating with boundaries," in *Proc. IEEE INISTA*, Jun. 2013, pp. 1–5.

[26] A. Kaushik, A. Chauhan, D. Mittal, and S. Gupta, "COCOMO estimates using neural networks," *Int. J. Intell. Syst. Appl.*, vol. 4, no. 9, pp. 22–28, Aug. 2012.

[27] M. Madheswaran and D. Sivakumar, "Enhancement of prediction accuracy in COCOMO model for software project using neural network," in *Proc. 5th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2014, pp. 1–5.

[28] A. Kaushik, A. K. Soni, and R. Soni, "An adaptive learning approach to software cost estimation," in *Proc. Nat. Conf. Comput. Commun. Syst.*, Nov. 2012, pp. 1–6.

[29] A. B. Nassif, M. Azzeh, L. F. Capretz, and D. Ho, "Neural network models for software development effort estimation: A comparative study," *Neural Comput. Appl.*, vol. 27, no. 8, pp. 2369–2381, Nov. 2016.

[30] I. Attarzadeh and S. H. Ow, "A novel algorithmic cost estimation model based on soft computing technique," *J. Comput. Sci.*, vol. 6, no. 2, pp. 117–125, Feb. 2010.

[31] G. Boetticher, "An assessment of metric contribution in the construction of a neural network-based effort estimator," in *Proc. Int. Workshop Soft Comput. Appl. Softw. Eng.*, 2001, pp. 59–65.

[32] C. Schofield, *Non-Algorithmic Effort Estimation Techniques*, Standard ESERG TR98-01, 1998.

[33] J. Li and G. Ruhe, "Analysis of attribute weighting heuristics for analogy-based software effort estimation method Aqua+," *Empirical Softw. Eng.*, vol. 13, no. 1, pp. 63–96, Feb. 2008.

[34] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 4, pp. 491–502, Apr. 2005.

[35] N.-H. Chiu and S.-J. Huang, "The adjusted analogy-based software effort estimation based on similarity distances," *J. Syst. Softw.*, vol. 80, no. 4, pp. 628–640, Apr. 2007.

[36] L. V. Patil, R. M. Waghmode, S. D. Joshi, and V. Khanna, "Generic model of software cost estimation: A hybrid approach," in *Proc. IEEE Int. Advance Comput. Conf. (IACC)*, Feb. 2014, pp. 1379–1384.

[37] S. Kumari and S. Pushkar, "Performance analysis of the software cost estimation methods: A review," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 7, pp. 229–238, 2013.

[38] C. F. Kemerer, "An empirical validation of software cost estimation models," *Commun. ACM*, vol. 30, no. 5, pp. 416–429, May 1987.

[39] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Feature subset selection can improve software cost estimation accuracy," in *Proc. Workshop Predictor Models Softw. Eng. (PROMISE)*, 2005, pp. 1–6.
[40] M. Jankovic, S. Zitnik, and M. Bajec, "Reconstructing de facto software development methods," *Comput. Sci. Inf. Syst.*, vol. 16, no. 1, pp. 75–104, 2019.
[41] T. Beranic and M. Hericko, "Comparison of systematically derived software metrics thresholds for object-oriented programming languages," *Comput. Sci. Inf. Syst.*, vol. 17, no. 1, pp. 181–203, 2020.
[42] 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.

NEVENA RANKOVIC was born in Valjevo, Serbia, in 1994. She received the B.Sc. and M.Sc. degrees in software engineering from the Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, in 2015 and 2018, respectively, where she is currently pursuing the Ph.D. degree with From 2018 to 2019, she was a Teaching Assistant with the Department of Mathematics and Informatics, University of Novi Sad. She is currently a Teaching Assistant with the School of Computing, Union University, Belgrade on subjects like software testing, software engineering, and software project management. Her research interests include software engineering, applied artificial intelligence, business intelligence, agile development, web technologies, software quality, requirements engineering, software project management, software testing, and software metrics.

DRAGICA RANKOVIC was born in Valjevo, Serbia, in 1963. She received the Ph.D. degree in mathematics and informatics from the Technical Faculty, University of Novi Sad, in 2009. She is currently with the School of Computing, Union University, Belgrade as a Teaching Lecturer on subjects like mathematical analysis, linear algebra, and analytic geometry. Her research interests include fuzzy theory, mathematical analysis, and applied artificial intelligence.

MIRJANA IVANOVIC (Member, IEEE) has been a Full Professor with the Faculty of Sciences, University of Novi Sad, Serbia, since 2002. She has also been a member of the University Council for informatics for more than 10 years. She has authored or coauthored 13 textbooks, 13 edited proceedings, 3 monographs, and of more than 440 research articles on multi-agent systems, e-learning and web-based learning, applications of intelligent techniques (CBR, data and web mining), software engineering education, and most of which are published in international journals and proceedings of high-quality international conferences. She is/was a member of program committees of more than 200 international conferences and general chair and program committee chair of numerous international conferences. Also, she has been an invited speaker at several international conferences and a visiting lecturer in Australia, Thailand, and China. As a leader and researcher, she has participated in numerous international projects. She is currently an Editor-in-Chief of *Computer Science and Information Systems* Journal.

LJUBOMIR LAZIC received the B.S., M.S., and Ph.D. degrees in electrical and computer engineering from the University of Belgrade, Belgrade, Serbia. Before he joined the School of Computing, he was an Associate Professor with Metropolitan University, Belgrade. He is currently a Professor with the School of Computing, Union University, Belgrade. He has authored about 140 articles published in international journals, book chapters, and conference proceedings, and invited speaker (keynote speaker at three International Conference on Software QA and Testing on Embedded Systems). His current research interests, as a project leader, are in two projects supported in part by the Ministry of Science and Technological Development of the Republic of Serbia under Grant TR-1318 (2008–2011) and TR-35026 (2011–2020) involving optimal software project management, software metrics, and effort estimation modeling. His research interests include software engineering, software project management, software testing, human computer interaction, and component-based engineering.