Turkish Journal of Computer and Mathematics *Vol.12No.14(2021),4186-4202*

Research Article

APPLICATION OF ARTIFICIAL NEURAL NETWORKS IN PREDICTION OF SOFTWARE DEVELOPMENT EFFORT

Zainab Rustum Mohsin

University of Thi-Qar, College of Education for Pure Science, Department of Computer Science, Thi-Qar/ Iraq

E-mail : zainabrustum@utq.edu.iq

Abstract: Over the past two decades, there has been a great enhance in researches dealing with the software development effort estimation utilizing machine learning (ML) approaches with the objective of improving the accuracy of the estimates. Among these ML methods, artificial neural network approaches have gained significant scholarly attention thanks to their capability to learn and model non-linear and complex functions. In this paper, artificial neural network technique was considered for modelling software development effort estimation. Datasets considered for estimation were COCOMO. Evaluation measures used were MMRE and correlation *R*. After building and testing the ANN model, and based on the comparison between the test results of the ANN model and the SLIM, Function Points, and COCOMO-basic models it could be concluded that the ANN was a suitable model in the estimation of the effort. ANN is recommended to be used as a predictive model for software development effort estimation.

Keywords: Machine learning, COCOMO database, Artificial neural network, Software effort estimation

1. Introduction

One of the main objectives of the software development community is to develop models that are practically applicable. Another important goal for the community is to evaluate how accurately a model can predict effort involved in developing software. Effort estimation is the prediction of the development time and cost required to develop the software product. Estimating of software development effort is essential and important issue to manage of complex and large software projects (Papatheocharous & Andreou, 2010). Accurate and reliable prediction for software effort enables project managers to effectively plan and allocate resources during software design and development. Overestimation can cause resources misallocation, affecting the development of other important projects. Conversely, underestimating of software effort may lead to delay and cost over-run, that may result in failure of project (Moosavi & Bardsiri, 2017). Therefore, during the last years, numerous effort estimation approaches were established using various theoretical concepts (Jorgensen & Shepperd, 2006) and combining existing estimation techniques (MacDonell & Shepperd, 2003; Mittas & Angelis, 2008). Because of the importance of accurate estimation of effort, extensive research has been conducted in this field. The current techniques can be categorized as below (Boehm, Abts, & Chulani, 2000) :

(1)Parametric models: SLIM (Putnam & Myers, 1991), SEER-SEM (Jensen, 1983), and COCOMO (Boehm, 1981)

(2)Expertise-based methods: Delphi technique and work breakdown structure based techniques (Jørgensen, 2004)

(3)Dynamics based approaches (Madachy, 1994)

(4)Regression based techniques: robust regression and ordinary least square regression (Costagliola, Ferrucci, Tortora, & Vitiello, 2005)

(5)Learning oriented methods: analogy based estimation (Shepperd & Schofield, 1997) and machine learning techniques (Mohsin, 2021; Oliveira, 2006).

Although previous models have produced improvements in prediction of software effort, the attempts to develop more reliable and accurate models are ongoing. Recently, machine learning techniques, such as artificial neural networks (ANN), have gained popularity as a way to model the complex relationship between effort and software attributes. Within this context, Wen et al. conducted an extensive literature review for relevant published studies in 1991–2010 and 84 primary empirical studies were selected (Wen, Li, Lin, Hu, & Huang, 2012).

They found that eight types of ML approaches were used in software development effort estimation models: case-based reasoning (CBR), decision trees (DT), artificial neural networks (ANN), support vector regression (SVR), Bayesian networks (BN), genetic programming (GP), genetic algorithms (GA), and association rules (AR). The most frequently utilized among them are ANN, DT, and CBR. Their study also revealed that the prediction accuracy of most ML models is close to the acceptable level and is outperforms the non-ML models in terms of MMRE and Pred (0.25). In fact, different ML approaches have different strengths and weaknesses and the efficiency of any model is influenced by the database characteristics utilized to build the model (size of dataset, outliers, categorical features, and missing values).

Because the artificial neural network technique is adaptable and nonparametric, predictive models can be designed to the data at a specific site. The present study investigated the application of a powerful machine learning approach namely ANN to estimate the software development effort. The COCOMO database (Boehm, 1981) have been used for training and testing the proposed ANN model.

The remainder of this paper is organized as follows: Section 2 presents the previous work related to software effort estimation using different machine learning techniques. Section 3 gives a description of the ANN technique. Section 4 illustrates the datasets and performance criteria employed to assess the performance of the ANN model. Section 5 presents the results obtained by the proposed model and comparison of the results is discussed. Finally, section 6 concludes the research and suggests future work.

2. The Related Work

Accurate and reliable estimation of development effort is a key element of effective software projects management. Although the numerous studies over the last decades, the software community remain faces significant challenges when it comes to estimate software effort. Time to time, different methods for the same have been discussed by authors. Karunanitthi et al. (Karunanithi, Whitley, & Malaiya, 1992) developed neural networks model for software reliability prediction. They performed analysis with both ANN and Jordan networks and the cascade correlation learning algorithm. Samson et al. (Samson, Ellison, & Dugard, 1997) applied Albus multiplayer perceptron to estimate software effort on the Boehm's COCOMO dataset and they compare a neural networks method with a linear regression method. The software development effort using wavelet neural network (WNN) was estimated by Kumar et al. (Kumar, Ravi, Carr, & Kiran, 2008). The WNN performance is compared with radial basis function network, multilayer perceptron, dynamic evolving neuro-fuzzy inference system, multiple linear regression, and support vector machine. The mean

magnitude relative error (MMRE) was used as the accuracy criterion. Two sets of data were used: The Canadian financial comprising 24 projects and the IBM data processing services comprising 37 projects. Their results indicated that the WNN performed better than the other methods. Heiat (Heiat, 2002) compared the performance of two types of NNs (a radial basis function network and a multilayer perceptron) with the performance of a regression approach for software effort estimation. The three datasets used were from IBM DP Services Organization, Kemerer dataset, and Hallmark dataset, having 24, 15, and 28 software projects, respectively. The performance criterion was the MMRE. Park and Baek (Park & Baek, 2008) proposed a neural network model for software development estimation. The model was developed used a set of 148 software projects that were published from 1999 to 2003, which included a wide range of software techniques, project periods, and project scales, as well as projects from a variety of industries. Kultur et al. (Kultur, Turhan, & Bener, 2009) predicted software effort using an ensemble of neural networks with associative memory (ENNA). Five sets of data were used: the NASA dataset comprising 60 projects (these projects are from 1980 to 1990), the COCOMO 81 dataset comprising 63 projects created before 1981, the Softlab Data Repository dataset comprising 24 projects, the NASA 93 dataset comprising 93 projects completed between 1970 and 1980, and the Desharnais dataset comprising 77 projects from the late 1980s. Lopez-Martin et al. (Lopez-Martin, Isaza, & Chavoya, 2012) proposed a model to predict the software development effort using general regression neural network (GRNN) approach. The International Software Benchmarking Standards Group (ISBSG) database was used for training and testing the proposed model. Khoshgoftaar et al. (Khoshgoftaar, Allen, & Xu, 2000) performed a research that used real time software for forecasting the testability for each module based on source code static measures. They applied ANNs approach for developing predictive models, because they can model nonlinear relationships. Sree et al. (Sree & SNSVSC, 2016) applied a fuzzy model using subtractive clustering for software effort estimation. The NASA 93 dataset was used for estimation. Nassif et al., (Nassif, Azzeh, Capretz, & Ho, 2016) used four ANN methods to predict the software development effort. Models considered for estimation are multi-layer perceptron, radial basis function neural networks, general regression neural network, and cascade correlation neural networks. The ISBSG dataset was used to develop the proposed model. Their results indicated that the cascade correlation neural network outperformed other compared methods.

Machine Learning approaches especially ANN are the prominent techniques to develop predicting models. Thus, newer machine learning-based techniques can always be used to develop more accurate predictive precision models. ANNs have been effectively used for solving problems in area like engineering, business, physics, and medicine. They can be utilized as predictive models because they are developing approaches capable of modeling non-linear and complex relationships. In this vision, our work is aimed to predict software development effort of a project size in source lines of code and other effort and cost drivers using artificial neural networks (ANN) technique.

3. Artificial Neural Networks (ANNs)

An artificial neural network (ANN) is a computational model inspired in the biological neural networks, which comprises of highly interconnected nodes, called neurons. (Tanarslan, Secer, & Kumanlioglu, 2012). The neuron calculates the sum of weighted input signals and produces an output signals if the sum is greater than a value, called the threshold. If the associated connection is excitatory, the weight is positive; if the connection is inhibitory, the weight is negative. The procedure continuous until one or more outputs are produced. Figure (1) shows the architecture of the neural network consisting of three layers (Baughman & Liu,

2014): (i) the input layer, which introduces input data into the network (ii) hidden layer as an intermediate layer, and (iii) the output layer, which consists nodes that represent the results of NN predictions. Also, ANN can involve a bias term acting on a neuron like an offset. The objective of the bias term is providing a threshold for the activation of neurons. The bias can be connected all neurons in network.

The neurons' number in the input layer is the same as the number of input data into the network, and the neurons' number in the output layer represents the number of target output data. Hidden layers comprises from one or more layers and the number of neurons in the hidden layer depends on the application of the network. Neural networks are parallel and distributed framework composed of a large number of interconnected processing units called nodes. Each node is connected with the other by varying connection strengths (or weights). Each node has a mathematical process that involves each input is multiplied by its weight, calculating the sum of the product, and then the activation function are applied to generate the desired output (Jeon & Rahman, 2008). Back-propagation is commonly used algorithm for training feedforward neural networks. The network is trained by feeding a pairs of input - output data.

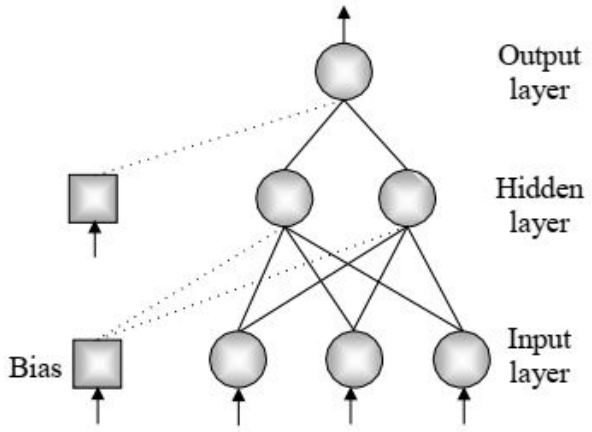


Figure 1. Basic structure of ANN

ANN model applications are generally carried out in two phases. At the first phase, the network is built and trained, and in the second phase, the network is tested with new input data. The main goal to train the model is updating the connection weights to achieve a satisfactory reduction of error between the values of actual output and desired output. During model training, net information is flow to the output layer along with connection weights by using Equation (1):

$$net = \sum_{i=1}^{n} x_i w_{ik} + b \quad (1)$$

where *net* is the total input of the node, x_i is the value of input, w_{ik} is the value of weight, and b is the neuron's negative threshold value and is referred to as the bias of the neuron.

The transfer (activation) function is used to produce the neuron output. The most commonly used activation function is the sigmoid transfer function, expressed as given in Equation (2):

$$f(net) = \frac{1}{1+e^{-net}} \quad (2)$$

where *f* (*net*) is the value of the output of the neuron and the process of obtaining *f* (net) is depicted in Figure 2.

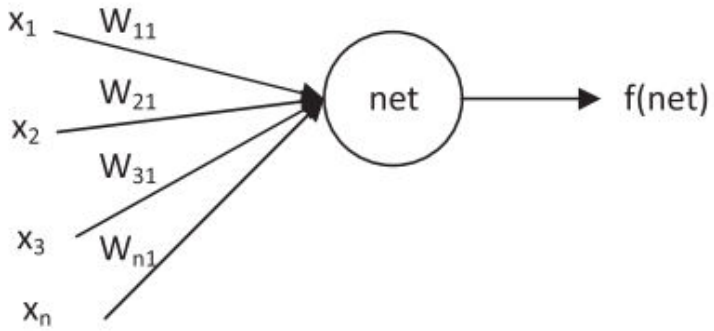


Figure 2. Neuron output

The error of the ANN model is calculated using actual output values and neurons output values. If the value of the error exceeds the acceptable range, the error is propagated backwards from output layer to the input layer. This process continues until the output error is minimized. In the steepest descent direction, the back-propagation algorithm updates the connection weights between neurons. After the training phase, the corresponding trained weights of the model are tested using new input variables that was not used in training. In this study, the resilient BPNN training algorithm is used. The structure and procedures of BPNN have been explained in details elsewhere in the literature (Mashrei, Abdulrazzaq, Abdalla, & Rahman, 2010; Rafiq, Bugmann, & Easterbrook, 2001). This paper presents the used of feed-forward back-propagation neural network (BPNN) technique to the actual data for software effort estimation utilizing MATLAB program (Higham & Higham, 2000).

4. Data Description and Performance Criteria

4.1. Description of the database

In the present study, COCOMO dataset (Boehm, 1981) was applied to determine the software effort utilizing ANN method. The COCOMO database was selected because it is a public domain database that has been used for various methods already. This dataset consists of 63 projects. Table 1 presents effort driver variables of the COCOMO dataset considered in this study. The dependent variable was the total effort, as represented by the variable MMACT (the number of man-hours spent on software development). The software size measured in thousands of source lines of code (ADJKDSI). The software size measured in term of ADJKDSI (thousands of lines of code). All cost drivers have qualitative rating levels ('extra low' to 'extra high') which express their influence on development effort. The cost driver at a nominal level always has the effort multiplier (EM) of 1.00, which means the estimated effort remains unchanged. If the rating level increases the software development effort, then the associated EM is greater than 1.0. Conversely, if a rating level causes less effort, then the associated EM is lower than 1.0 (BW Boehm et al., 2000). The cost drives considered in this study and their levels are presented in Table 2. Size, identified as ADJKDSI, is the only numerical independent variable in this work. This variable was then transformed into its corresponding logarithmic value to get better fitting and estimation results.

The COCOMO dataset was divided into two groups. The first data set training dataset was used for training and the second data set testing was used for testing the proposed model. The

assessment of the efficiency of the model similar to the procedure used by Kitchenham (Kitchenham, 1998), using a separate testing dataset. Following the Kitchenham's procedure, by creating six different training and testing datasets couple. The training dataset has been created by removing every sixth project beginning from a different project number.

For example, training set number 1 was created by removing projects 1, 7, 13, 19, 25, 31, 37, 43, 49, 55, 61(according to numbering scheme of Boehm); training set number 2 was created by removing projects: 2, 8, 14, 20, 26, 32, 38, 44, 50, 56, 62, and so on. The projects that were removed from the training set have been utilizing as the test dataset. Since the dataset consists from 63 projects, there were 52 projects in the first three training subsets and 53 projects in the next three. Although data can be randomly selected and only one training and testing dataset can be used, this collection of six couples of dataset may result in decrease the bias and prevent the recurrence of bias for data sequences in a possible software project.

Table 1. COCOMO features

Variable	Full name
ACAP	Analyst capability
TOOL	Use of software tools
AEXP	Applications experience
RELY	Reliability
MODP	Use programming modern practices
DATA	Database size
VEXP	Experience with virtual machine
PCAP	Programmers capability
CPLX	Process complexity
LEXP	Programming language experience
TIME	Restriction of time
TURN	Computer turnaround
SCED	Schedule constraint
VIRT	Virtual machine volatility

Table 2. Cost driver.

Cost drive	Levels
ACAP	1. Super high 2. Very high 3. High 4. Nominal 5. Low 6. Very low 7. Super low
RELY	1. Extra low 2. Very low 3. Low 4. Nominal 5. High 6. Very high
STOR	1. Nominal 2. High 3. Very high 4. Extra high 5. Super high

TOOL	1. Extra high 1. Very high 2. High 3. Nominal 4. Low 5. Very low
AEXP	1. Very high 2. High 3. Nominal 4. Low 5. Very low
MODP	1. Extra high 2. Very high 3. High 4. Nominal 5. Low 6. Very low
DATA	1. Low 2. Nominal 3. High 4. Very high 5. Extra high
VEXP	1. High 2. Nominal 3. Low 4. Very low
PCAP	1. Super high 2. Very high 3. High 4. Nominal 5. Low 6. Very low 7. Super low
LEXP	1. High 2. Nominal 3. Low 4. Very low
CPLX	1. Very low 2. Low 3. Nominal 4. High 5. Very high 6. Extra high 7. Super high
TIME	1. Nominal 2. High

	3. Very high 4. Extra high 5. Super high
VIRT	1. Low 2. Nominal 3. High 4. Very high
TURN	1. Very high 2. High 3. Nominal 4. Low 5. Very low
SCHED	1. Lax 2. Nominal 3. Compressed 4. Very Compressed
RVOL	1. Low 2. Nominal 3. High 4. Very high 5. Extra high 6. Super high

4.2. Performance Criteria

(1) Mean Magnitude Relative Error (MMRE): MMRE is commonly used to assess the performance of any estimation method. It measures the percentage of the absolute relative errors values from whole dataset (Baker, 2007).

$$MMRE = \frac{1}{N}\sum_{1}^{N}\frac{|Actual\ Effort - Predicted\ Effort|}{Actual\ Effort} \quad (3)$$

where *N* is the amount of projects.

(2) The Correlation coefficient (R): is a statistical measure that determines the degree to which the two different variables are associated. The value of the R ranges between 0 and 1. The best model that give R value that are as near to 1 as possible. A negative value of R indicates that the data and the model have no correlation.

$$R = 1 - \sqrt{\frac{\sum_{i=1}^{N}(Actual\ Effort - Predicted\ Effort)^2}{\sum_{i=1}^{N}(Actual\ Effort)^2}} \quad (4)$$

5. Results and Discussion

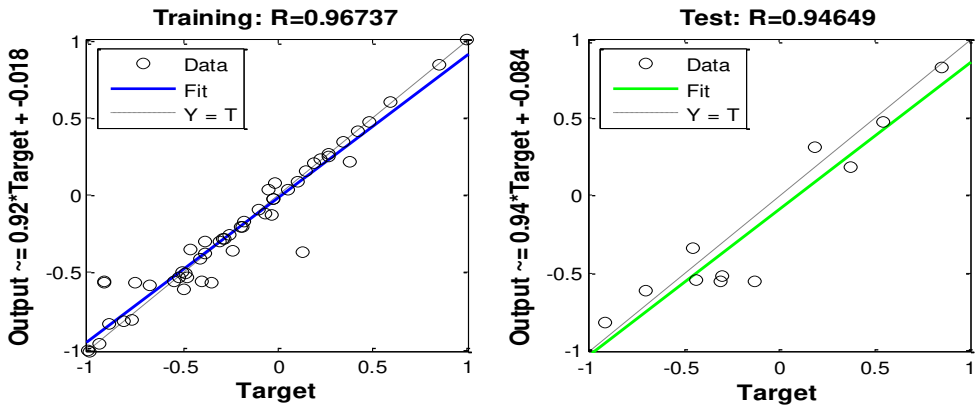
5.1. Proposed Approach

The MATLAB NN toolbox ("Neural network toolbox user's guide: for Use with MATLAB," 2009) was used for creating the proposed ANN model. Feed forward - backpropagation algorithm was used for ANN modeling. A sigmoid function was used as a transfer function for all the neurons in the input and hidden layers, whereas for the output layer a linear one was applied. Through using trail-and-error process, the optimal number of hidden layers, number of neurons in hidden layer, and epochs was chosen. During the training process, the model's training convergence is based on the minimizing the tolerance error to root mean squared error (RMSE) and evaluating the predictive accuracy of the proposed model by comparing the outputs. After the errors are minimized, testing is performed to ensure that the estimated values are close to actual values.

In the developing ANN, there is an input layer in which input variables are provided to network and an output layer, with one neuron giving software effort estimation. The model with one hidden layer and 18 nodes in the hidden layer gave the optimal configuration with minimum RMSE. Table 1 presents the input data used for the training process of the proposed model to predict the effort. In order to find the optimum network for solving the problem, the training process was repeated ten times. Besides, different ANN architectures were investigated and the best model has been selected. Table 3 presents the results of the ANN model with the best-performing ANN model on the testing dataset. Figures 3-8 show the actual (target) values versus predicted values during the training and testing periods. A head-to-head comparison of performance for actual (target) values and proposed ANN model for testing set are shown in Figures 9-15.

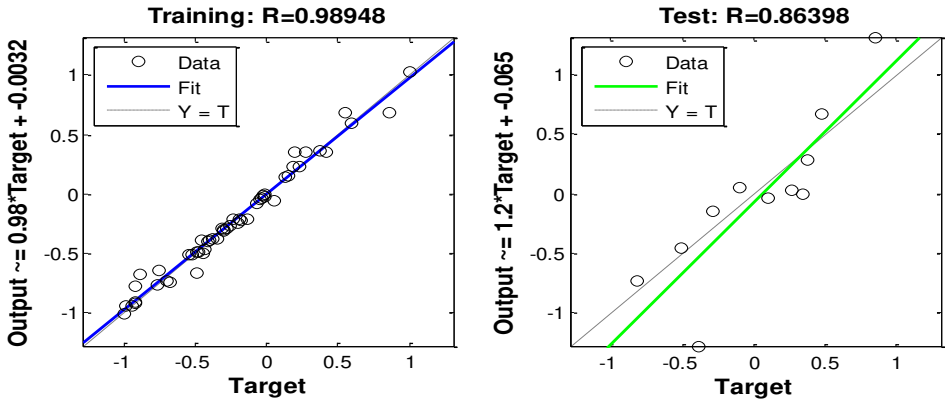
Table 3. Effort estimation using ANN approach

Set 1		Set 2		Set 3		Set 4		Set 5		Set 6	
Actual	Estimated	Actual	Estimated	Actual	Estimated	Actual	Estimated	Actual	Estimated	Actual	Estimated
2040	1513.56	1600	3235.94	243	204.17	240	501.19	33	34.67	43	31.6
8	11.75	1075	759	423	776.25	321	218.78	218	338.84	201	87.1
79	32.36	60	2.00	61	457.09	40	69.18	9	50.12	11400	11220.2
6600	5888.44	6400	36307.8	2455	1778.28	724	588.84	539	501.19	453	457.1
523	851.14	387	223.87	88	24.55	98	213.8	7.3	5.89	5.9	6.0
1063	524.81	702	281.84	605	724.44	230	144.54	82	60.26	55	56.2
47	72.44	12	16.22	8	11.48	8	9.55	6	20.89	45	93.3
83	36.31	87	147.91	106	95.50	126	109.65	36	14.45	1272	288.4
156	31.62	176	309.03	122	63.10	41	165.98	14	19.50	20	3.6
18	25.70	958	257.04	237	42.66	130	39.81	70	50.12	57	38.9
50	33.11	38	45.71	15	10.47						



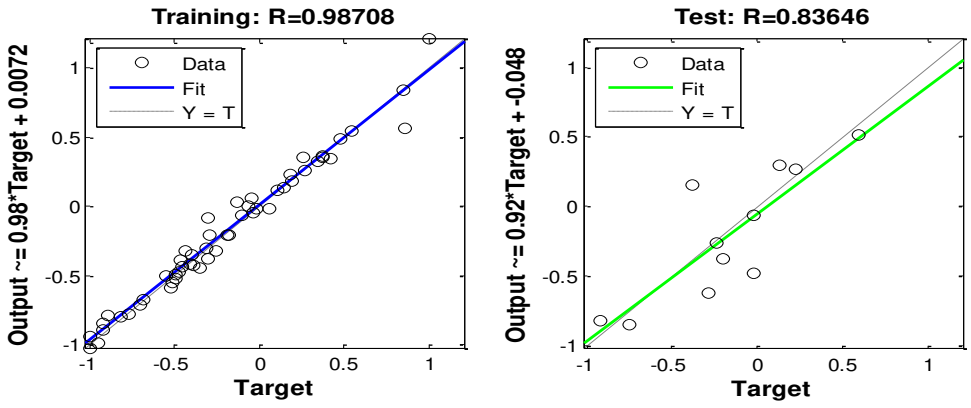
(a) (b)

Figure 3. Comparison between actual results and ANN results for set 1 for (a) training dataset and (b) testing



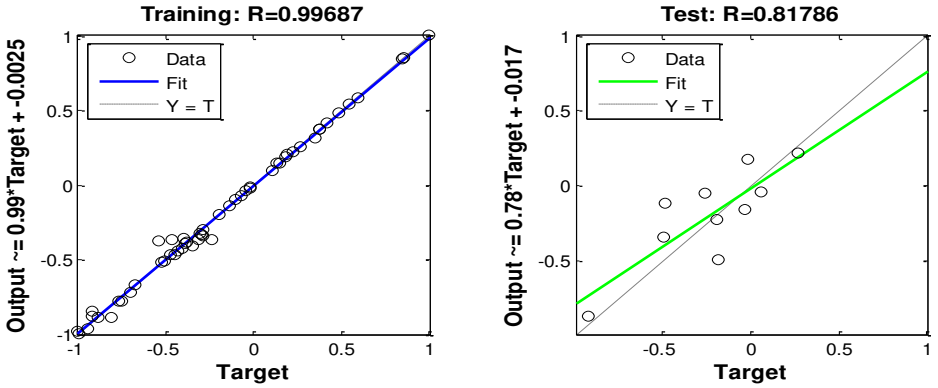
(a) (b)

Figure 4. Comparison between actual results and ANN results for set 2 for (a) training dataset and (b) testing



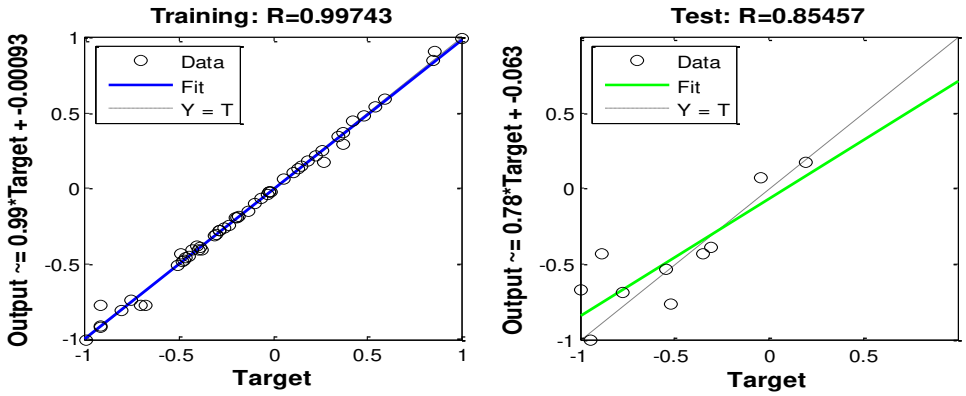
(a) (b)

Figure 5. Comparison between actual results and ANN results for set 3 for (a) training dataset and (b) testing



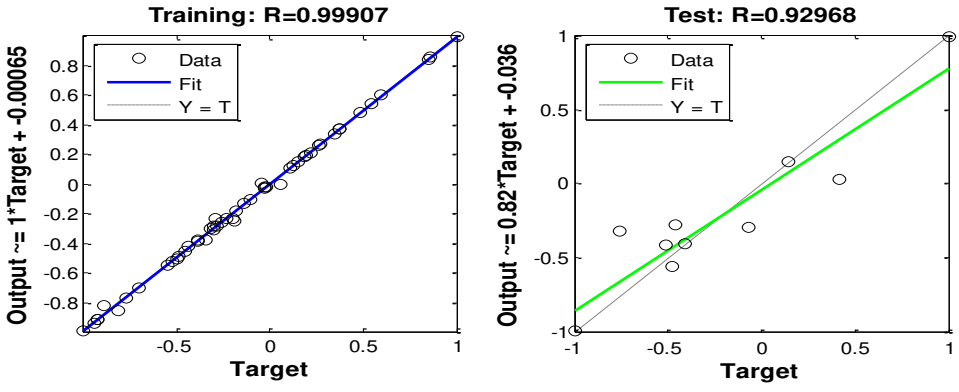
(a) (b)

Figure 6. Comparison between actual results and ANN results for set 4 for (a) training dataset and (b) testing



(a) (b)

Figure 7. Comparison between actual results and ANN results for set 5 for (a) training dataset and (b) testing



(a) (b)

Figure 8. Comparison between actual results and ANN results for set 6 for (a) training dataset and (b) testing

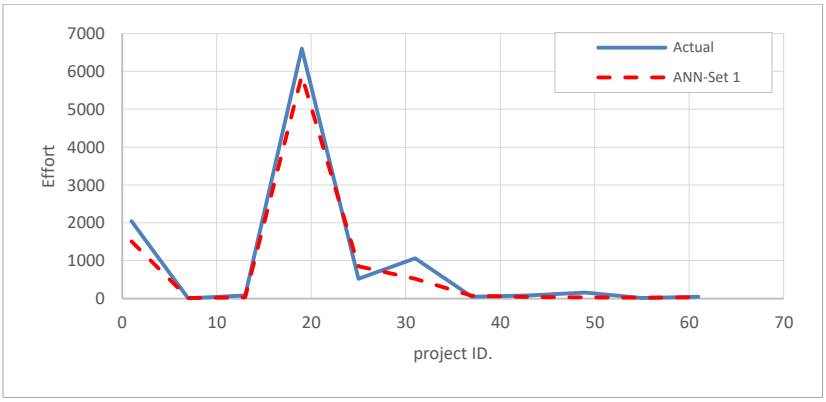


Figure 9. Actual and predicted software effort for set 1 (testing dataset)

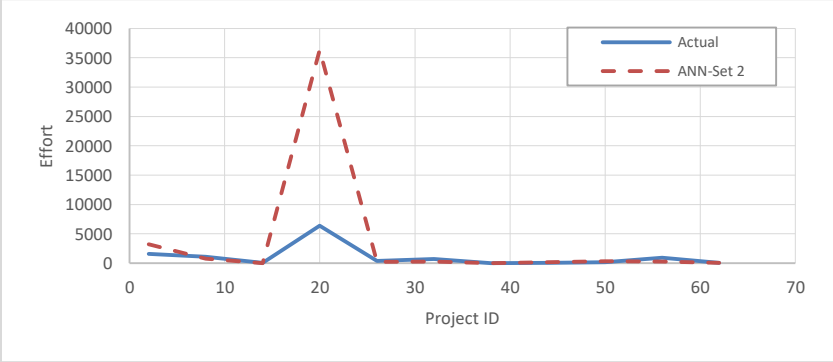


Figure 10. Actual and predicted software effort for set 2 (testing dataset)

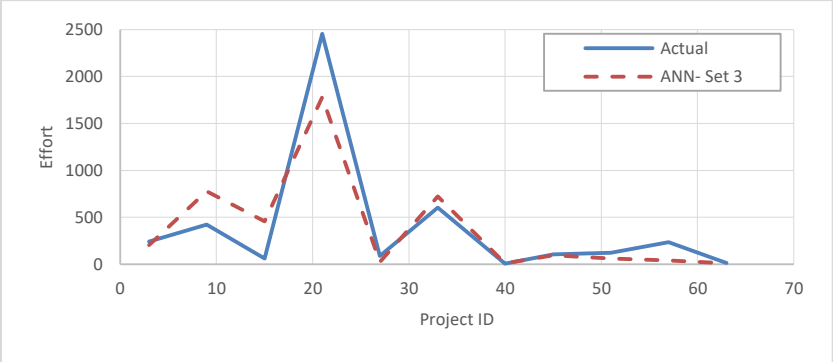


Figure 11. Actual and predicted software effort for set 3 (testing dataset)

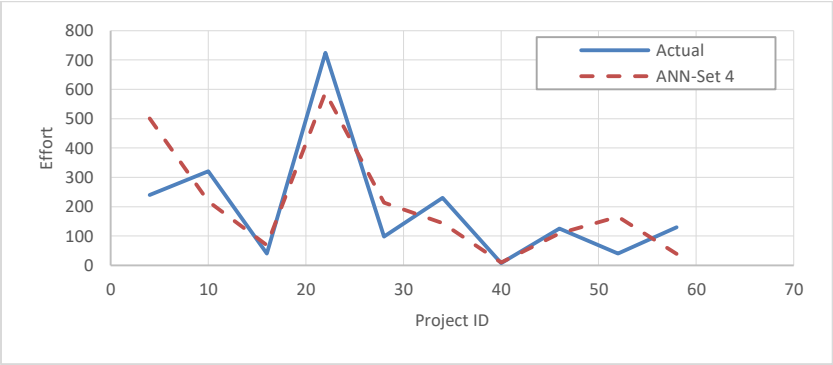


Figure 12. Actual and predicted software effort for set 4 (testing dataset)

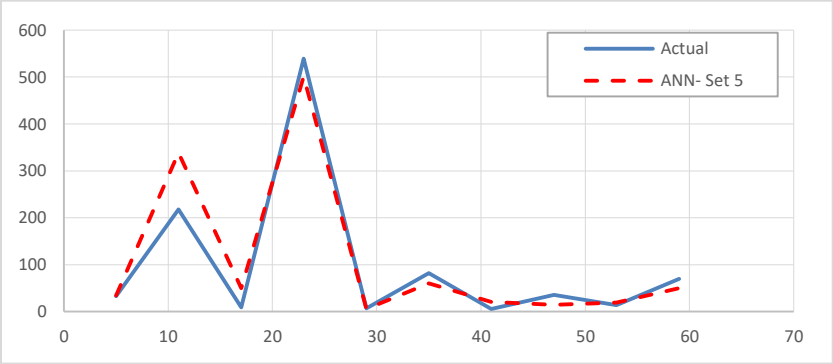


Figure 13. Actual and predicted software effort for set 5 (testing dataset)

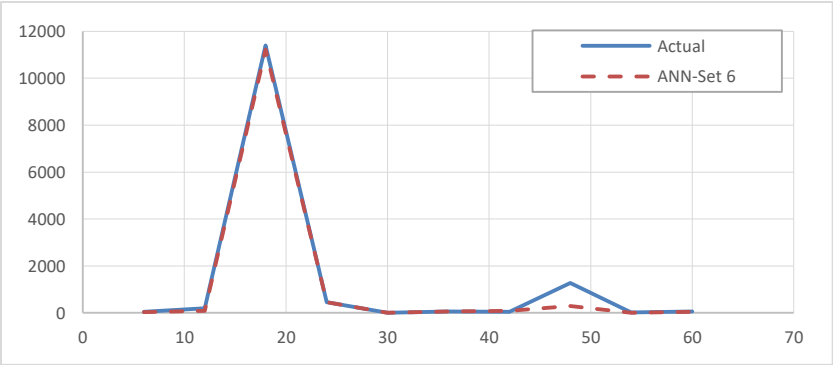


Figure 14. Actual and predicted software effort for set 6 (testing dataset)

5.2. Comparison study

Table 4 presents the values of MMRE the correlation R resulting from the estimates conducted using ANN model with the six testing datasets. According to Table 4, the best performer could be attributed to the 6th dataset. The sixth dataset gives the best results with minimum MMRE. The correlation R is quite high.

Table 4. The test dataset predictive accuracy

Test dataset	MMRE	*R*
1	47.50	0.946
2	97.44	0.864
3	98.37	0.836
4	79.41	0.818
5	94.6	0.854
6	38.83	0.929

Figure 16 also confirms this result when comparing the MMRE and the correlation R for all test dataset. Kemerer (Kemerer, 1987) obtained the results shown in Table 5 using three software cost estimation models, which are COCOMO-basic, Function Points, and SLIM models.

These results demonstrate a good agreement in the predicting values compared with the actual effort values for all test dataset. In term of MMRE, the ANN model outperforms the COCOMO 's performance (610), the SLIM and the Function Point Analysis models (772 and 103, respectively) in Kemerer's experiments. With regard to correlation R, the ANN has better performance rather than the Function Points and COCOMO-basic methods. The results show that the ANN model outperformed the SLIM method for the first and sixth dataset in term of correlation R. But the SLIM model provides a slightly better value of correlation R than ANN model for the remaining dataset. It should be noted that if the training data set represent most of the probable software effort scenarios the ANN method will perform much better. Thus, it is necessary to collect additional data in order for the model to learn (be trained on) all of the data's intrinsic features.

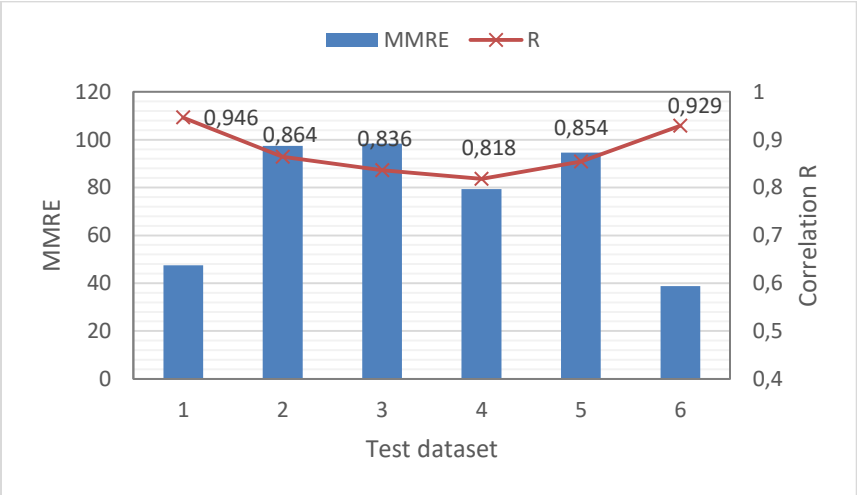


Figure 15. Comparative analysis of test datasets

Table 5. Traditional models on COCOMO dataset

Model	MMRE	*R*
SLIM	772	0.89
Function Points	103	0.58
COCOMO-basic	610	0.70

6. Conclusion and Future Works

In this investigation, the artificial neural network (ANN) model were developed to estimate the software development effort. The Boehm's COCOMO dataset was used to train and test the network. Six different groups of training and testing data set were generated by following the Kitchenham's procedure. The training data set was constructing by eliminating every sixth project beginning from a different software project number. The testing data set were comprising from the projects that were eliminated. The optimal ANN models were chosen after experimenting different model architecture. The structure of the proposed ANN model has one hidden layer with 18 nodes, with sigmoid function as transfer function, and one output nodes with a linear transfer function. The evaluation was conducted using two assessment criteria: MMRE and correlation R. The ANN results showed a good agreement with the actual data for the six test datasets. The ANN model was also compared with SLIM, Function Points, and COCOMO-basic methods. It was found that the ANN model provided better results compared to the other models. Therefore, the ANN can serve as an economical, efficient, and reliable tool for software development effort estimation.

In future research, a hybrid model will be developed to improve prediction of software development effort.

References

Baker, D. R. (2007). *A hybrid approach to expert and model based effort estimation*: Citeseer.

Baughman, D. R., & Liu, Y. A. (2014). *Neural networks in bioprocessing and chemical engineering*: Academic press.

Boehm, B. (1981). *Software engineering economics*: Prentice-Hall, Englewood Cliffs, NJ.

Boehm, B., Abts, C., Brown, A., Chulani, S., Clark, B., Horowitz, E., . . . Steece, B. (2000). Software cost estimation with COCOMO II. Prentice Hall PTR. *Upper Saddle River, NJ*.

Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches—A survey. *Annals of software engineering, 10*(1), 177-205.

Costagliola, G., Ferrucci, F., Tortora, G., & Vitiello, G. (2005). Class point: an approach for the size estimation of object-oriented systems. *IEEE Transactions on software engineering, 31*(1), 52-74.

Heiat, A. (2002). Comparison of artificial neural network and regression models for estimating software development effort. *Information and Software Technology, 44*(15), 911-922.

Higham, D. J., & Higham, N. J. (2000). *MATLAB guide*: Philadelphia: SIAM.

Jensen, R. (1983). *An improved macrolevel software development resource estimation model.* Paper presented at the 5th ISPA Conference.

Jeon, J., & Rahman, M. S. (2008). Fuzzy neural network models for geotechnical problems.

Jørgensen, M. (2004). Top-down and bottom-up expert estimation of software development effort. *Information and Software Technology, 46*(1), 3-16.

Jorgensen, M., & Shepperd, M. (2006). A systematic review of software development cost estimation studies. *IEEE Transactions on software engineering, 33*(1), 33-53.

Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992). Using neural networks in reliability prediction. *IEEE Software, 9*(4), 53-59.

Kemerer, C. F. (1987). An empirical validation of software cost estimation models. *Communications of the ACM, 30*(5), 416-429.

Khoshgoftaar, T. M., Allen, E. B., & Xu, Z. (2000). *Predicting testability of program modules using a neural network.* Paper presented at the Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology.

Kitchenham, B. (1998). A procedure for analyzing unbalanced datasets. *IEEE Transactions on software engineering, 24*(4), 278-301.

Kultur, Y., Turhan, B., & Bener, A. (2009). Ensemble of neural networks with associative memory (ENNA) for estimating software development costs. *Knowledge-Based Systems, 22*(6), 395-402.

Kumar, K. V., Ravi, V., Carr, M., & Kiran, N. R. (2008). Software development cost estimation using wavelet neural networks. *Journal of Systems and Software, 81*(11), 1853-1867.

Lopez-Martin, C., Isaza, C., & Chavoya, A. (2012). Software development effort prediction of industrial projects applying a general regression neural network. *Empirical Software Engineering, 17*(6), 738-756.

MacDonell, S. G., & Shepperd, M. J. (2003). Combining techniques to optimize effort predictions in software project management. *Journal of Systems and Software, 66*(2), 91-98.

Madachy, R. J. (1994). *A software project dynamics model for process cost, schedule and risk assessment.* University of Southern California.

Mashrei, M. A., Abdulrazzaq, N., Abdalla, T. Y., & Rahman, M. (2010). Neural networks model and adaptive neuro-fuzzy inference system for predicting the moment capacity of ferrocement members. *Engineering Structures, 32*(6), 1723-1734.

Mittas, N., & Angelis, L. (2008). *Combining regression and estimation by analogy in a semi-parametric model for software cost estimation.* Paper presented at the Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement.

Mohsin, Z. R. (2021). Investigating the Use of an Adaptive Neuro-Fuzzy Inference System in Software Development Effort Estimation. *Iraqi Journal For Computer Science and Mathematics, 2*(2), 18-24.

Moosavi, S. H. S., & Bardsiri, V. K. (2017). Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation. *Engineering Applications of Artificial Intelligence, 60*, 1-15.

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

Neural network toolbox user's guide: for Use with MATLAB. (2009).

Oliveira, A. L. (2006). Estimation of software project effort with support vector regression. *Neurocomputing, 69*(13-15), 1749-1753.

Papatheocharous, E., & Andreou, A. S. (2010). *On the problem of attribute selection for software cost estimation: Input backward elimination using artificial neural networks.* Paper presented at the IFIP International Conference on Artificial Intelligence Applications and Innovations.

Park, H., & Baek, S. (2008). An empirical validation of a neural network model for software effort estimation. *Expert Systems with Applications, 35*(3), 929-937.

Putnam, L. H., & Myers, W. (1991). *Measures for excellence: reliable software on time, within budget*: Prentice Hall Professional Technical Reference.

Rafiq, M., Bugmann, G., & Easterbrook, D. (2001). Neural network design for engineering applications. *Computers & Structures, 79*(17), 1541-1552.

Samson, B., Ellison, D., & Dugard, P. (1997). Software cost estimation using an Albus perceptron (CMAC). *Information and Software Technology, 39*(1), 55-60.

Shepperd, M., & Schofield, C. (1997). Estimating software project effort using analogies. *IEEE Transactions on software engineering, 23*(11), 736-743.

Sree, P. R., & SNSVSC, R. (2016). Improving efficiency of fuzzy models for effort estimation by cascading & clustering techniques. *Procedia Computer Science, 85*, 278-285.

Tanarslan, H., Secer, M., & Kumanlioglu, A. (2012). An approach for estimating the capacity of RC beams strengthened in shear with FRP reinforcements using artificial neural networks. *Construction and Building Materials, 30*, 556-568.

Wen, J., Li, S., Lin, Z., Hu, Y., & Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology, 54*(1), 41-59.