Indian Journal of Science and Technology, Vol 9(29), DOI: 10.17485/ijst/2016/v9i29/90093, August 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Application of Artificial Neural Network for Software Reliability Growth Modeling with Testing Effort

Subburaj Ramasamy and Indhurani Lakshmanan*

School of Computing, SRM University, Kattankulathur - 603203, Tamil Nadu, India; subburaj.r@ktr.srmuniv.ac.in, indhurani.a@gmail.com

Abstract

Background/Objectives: To design a relatively simple Software Reliability Growth Model (SRGM) with testing effort function using Artificial Neural Network approach. **Methods/Statistical Analysis:** The results evaluation of the proposed SRGM using Artificial Neural Network (ANN) is measured by calculating the three vital criterians namely; AIC (Akaike Information Criterian), R^2 (Coefficient of determination) and RMSE (Root Mean Squared Error). **Findings:** Traditional time-based models may not be appropriate in some situations where the effort is varying with time. Estimating the total effort required for testing the software in the Software Development Life Cycle (SDLC) is important. Hence, a multi-layer feed-forward Artificial Neural Network (ANN) based SRGM using back propagation training is proposed in this paper by incorporating test effort. The proposed ANN based model provides consistent performance for both exponential and S-shaped growth of mean value functions witnessed in software projects. **Application/Improvements:** The proposed SRGM using ANN will be performed to be eminently useful for software reliability applications, since it is able to maintain its performance in all situation.

Keywords: Artificial Neural Network, Back Propagation, Software Reliability Growth Model, Software Testing, Testing Effort Estimation

1. Introduction

American National Standards Institution (ANSI) defines the software reliability as the probability of failure-free operation of a software system for a specified period of time in a specified environment[1]. The growth of software reliability increases through the correction of the faults causing failures, during the test phase. Software reliability growth models estimate the present reliability by estimating the model parameters and predict the future reliability of a system using practical failure data. The modeling performance of SRGMs depends on the nature of the failure data set. Although more than 200 SRGMs have been proposed since the year 1972, still the research is going to develop more accurate predictive models.

Typically there are two classifications of Software reliability growth models: Parametric models and non-parametric models. Non-Homogeneous Poisson Process (NHPP), Markovian and Bayesian models are three types of parametric models. The application of NHPP models are used in the field of hardware and software reliability. Thus researchers mostly use NHPP models in software reliability engineering research[2]. In[3] presented the first NHPP model based on continuous distribution. Then different types of S-Shaped NHPP models were presented by[4]. Logarithmic Poisson Execution Time model and Basic Execution Time model were proposed by[2]. The imperfect debugging models proposed by[5,6]. Later on,[7] proposed Generalized NHPP model and[8,9] presented Generalized NHPP weibull model. In[10] presented a study of NHPP based SRGMs to improve software quality. In[11] discussed an NHPP software reliability growth model with the concept of imperfect debugging. In[12] evaluated SRGMs by improving their predictive accuracy using historical projects failure data.

Traditional parametric SRGMs have two characteristics in common. First, the models are built by assuming that

*Author for correspondence

the failure process follows a predetermined distribution[13]. Second, the parameters, as well as the dependent and independent variables in the parametric SRGM, have meaningful interpretations such as a cumulative number of failures or failure detection rate. However, there is sensitivity of model parameters i.e., the parameters vary with additional data assuming to be available.

On the other hand, Artificial Neural Network (ANN) based non-parametric software reliability models estimates the model parameter without any distributions and assumptions. It is used in researching regression and classification problems by the simulation of biological neural networks. The estimation accuracy of ANN-based models can be improved by increasing number of hidden layers of the multi-layer feed-forward back propagation neural network[14]. ANN based software reliability model improves with better parameter estimation and predictive accuracy than parametric traditional models[15,16]. In[16] proposed first ANN based software reliability concept for prediction. In[17] presented the comparison between the neural networks and parametric models to predict the reliability. In[18] applied back-propagation neural network for software reliability to predict the next failure time. In[19] presented a neural network approach combining dynamic creation of weighted models for software reliability estimation and prediction. In[20] compared an artificial neural network with three traditional statistic NHPP models to predict software reliability. In[21] applied neural network concept to the traditional software reliability growth models to improve the accuracy of software reliability prediction. In[22] built an approach with various ANNs for complicated software to improve the accuracy of software reliability estimation. In[23] proposed a neuro-genetic approach on the logistic model to improve the software reliability prediction accuracy. In[24] they presented a survey on intelligent techniques in Decision making. Back propagation and hippomy algorithm for statistical classification are discussed in[25]. Software reliability models are classified to improve software reliability in[26].

During the software testing phase, efforts like manpower, test cases and computing time are consumed. The fault detection and correction will depend on efforts consumed. The Exponential, Rayleigh, Logistic and Weibull functions have been studied to describe testing effort consumption[27–31]. In this paper, we propose an SRGM with testing effort function based on the multi-layer feed-forward artificial neural network architecture and back propagation training algorithm which is used to determine the gradient weight.

This paper is discussed as follows: Section 2 presents a method to construct the architecture of proposed model using the elements of neural network. Section 3 presents the testing effort concept. Section 4 presents the implementation for software reliability estimation of the proposed model. Section 5 discusses the experiment results by using real practical failure data set by evaluating the performance of the traditional base model, neural network-based model, and proposed model. Section 6 describes Summary and conclusion.

2. Elements of Neural Network

A neural network is a network of interconnected nonlinear neurons inspired from the studies of the biological neuron system. The function of a neural network is to produce an output pattern when presented with an input pattern[34]. The proposed ANN-based SRGM with testing effort function is constructed by three basic elements: Neurons, Network architecture and learning algorithm.

- **Neurons:** A neuron is an important element for the process of a neural network which maps the input data elements with the output data elements through an activation function. The process of a single neuron is mathematically defined as

$$y = f(h) \text{ and } h = \sum_{i=1}^{M} \text{witi} \quad (1)$$

here,

f(h) = activation function which maps the input data elements with the final output data elements.

Σ = summation function which sums the input data elements with the corresponding input weights.

wi = w_1, w_2, w_3, …w_m are corresponding input data elements weights

M is total number of input data elements i.e., t_1, t_2, t_3, …t_m and

y = Final output of a single neuron.

- **Neural Network Architecture:** A simple multi-layer Feed-Forward Neural Network is used to design the proposed model. Here the calculation of data goes in a forward direction, starting from the input layer for accepting the input elements to the output layer to produce the output through hidden layer to map the input and output elements. The proposed ANN-based SRGM with testing effort function is constructed with single input and output layer each has single neuron and three hidden layers each has single hidden neuron.

- **Training algorithm:** The proposed ANN based SRGM is trained with back-propagation training algorithm by feeding a neural network with the training input data and finding the difference between estimated output and the actual output by adjusting weights until the stopping criteria are met. The backward propagation refers the output error at the time of training on the network which is found by calculating the difference between the estimated output of the neural network and the actual output. This output error element is sent back through the propagation of network architecture in the backward track by optimizing gradient weights of the neural network to improve the evaluation performance of model architecture.

3. Testing Effort Modelling

In[28] proposed a testing effort dependent software reliability growth model based on Goel-Okumoto model[3]. If W(x) is the testing effort spent at time x then the mean value function μ(x) of the SRGM is:

$$\mu(x) = n\left[1 - e^{-sw(x)}\right] 0 < r < 1, n > 0 \quad (2)$$

where, s = fault detection rate, interpreted as per the testing effort at unit time x.

W(x) = amount of total spending testing effort in time interval [0,x].

In[29] proposed the Weibull curve as a testing effort function for an exponential SRGM. The total consumed testing effortin time interval [0,x] is given by

$$W(x) = a\left(1 - e^{\left(bx^c\right)}\right) \quad (3)$$

Where a is the total effort that will be consumed if testing is continued till the infinite time, b is scale parameter, and cis shape parameter. When c = 1, we obtain an exponential testing effort function. We choose log-power model [32]due to its inherent properties and time transform[2] the model to transform it into effort based model.

4. Experiment Implementation

4.1 Implementation of the Proposed ANN-based SRGM with Testing Effort

The steps for implementation of the experiment that are needed to design the ANN-based SRGM with testing effort function are as follows:

- Select simple software reliability growth model and testing effort function as the support models to design the neural network architecture with suitable activation functions for the hidden and output layer neurons.
- Train the network architecture by giving the normalized practical failure data set to the neural model using back-propagation training algorithm.
- Using the trained neural network, estimate the respective weights of neurons and measure the performance analysis of the proposed ANN-based SRGM with testing effort function.

4.2 Selection of Base Model

The ANN based SRGM with testing effort function proposed in this paper is designed with single neuron in the input and output layer and three hidden layers each has single neuron. The number of hidden layer neurons is found by the chosen support models and activation functions selected to design ANN based software reliability model. An infinite failure log power model[32] with exponential testing effort function is selected in the proposed model since they have the simplicity and ability to give the best fit.

Figure 1 represents the ANN-based SRGM with testing effort function using back-propagation training algorithm. In the traditional log power SRGM, the time 'x' is substituted with exponential testing effort function by applying the time transformation as applicable to NHPP model. If W(x) is the exponential testing effort spent at time t then the mean value function μ(x) of the log power SRGM[32] is given as follows:

$$\mu(x) = a\log\left(1 + W(x)\right)^c \quad (4)$$

and

$$W(x) = N\left(1 - e^{(-bx)}\right) \quad (5)$$

where, a, b and c are constants.

W(x) = amount of spending test effort in time interval [0,x].

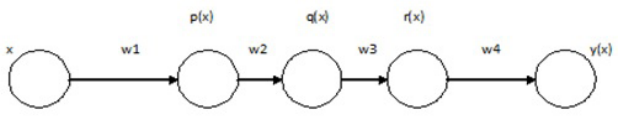

Figure 1. Proposed multilayer feed forward ANN based testing effort model.

Thus, the mean value function μ(x) of the log power SRGM with exponential testing effort function is as follows:

$$\mu(x) = a\log\left(1+N(1-e^{(-bx)})\right)^c \quad (6)$$

The output of the proposed multi-layer feed-forward ANN based SRGM with testing effort function is as follows:

$$y(x) = w4\log\left(1+w2\left(1-e^{(-w1x)}\right)\right)^{w3} \quad (7)$$

Where, w1, w2, w3 and w4 (>0) are the weights of the proposed ANN-based software reliability model and the weight values are found by the back-propagation training algorithm.

According to the need for the activation functions in the ANN-based software reliability growth model, the activation functions are designed as simple, continuous, easily differentiable and the function of the output can be mapped approximately as a compound form g(f(x)). The output of the proposed multi-layer feed-forward ANN based SRGM with testing effort function can be derived from its mean value function as a compound form as the following: Equation (10), is the mean value function of the proposed ANN-based SRGM with testing effort function.

Assume that

$$f(x) = 1-e^{(-bx)}, g(x) = Nx, h(x) = log(1+x)^c \text{ and } i(x) = ax$$

Then we get,

$$\begin{aligned} i\left(h\left(g\left(f(x)\right)\right)\right) &= i(h(g\left(1-e^{(-bx)}\right) = i(h\left(N\left(1-e^{(-bx)}\right)\right) \\ &= i(\log\left(1+N\left(1-e^{(-bx)}\right)\right)^c \\ &= a(\log\left(1+N\left(1-e^{(-bx)}\right)\right)^c) \end{aligned} \quad (8)$$

From Equation (6) and (8), we see that the mean value function of the proposed the ANN-based SRGM with testing effort function is composed off (x), g(x), h(x) and i(x).

From the above derivation, we see that the hidden layer neurons have the activation functions , x, log(1+x) and a linear activation function i(x) = x is used in the output layer neuron of the proposed ANN-based SRGM with testing effort function. Note that, the activation functions can be continuous and differentiable everywhere, so they are by the conditions of the activation function. The activation functions used for hidden layer neurons are designed from the respective mean value function of the selected SRGM and testing effort function. If we construct a neural network with the activation functions as , x and log(1+x) in the hidden layers and x for the output layer having the weights w1, w2, w3 and w4 then we get the output equation for hidden layers and output layer as below.

$$p(x) = 1-e^{(-w1x)} \quad (9)$$

$$q(x) = w2\left(1-e^{(-w1x)}\right) \quad (10)$$

$$r(x) = \log\log\left(1+w2\left(1-e^{(-w1x)}\right)\right)^{w3} \quad (11)$$

and

$$y(x) = w4\left(\log\left(1+w2\left(1-e^{(-w1x)}\right)\right)^{w3}\right) \quad (12)$$

Compare Equation (6) and (12), It is the log power SRGM with exponential testing effort function and w1 = b, w2 = N, w3 = c and w4 = a.

Extending the same method, we can simply construct a neural network with the suitable activation functions and respective weights for any given SRGM.

4.3 Normalization of Failure Data

The software failure data are arranged in pairs {xi,yi} where xi is the cumulative test time and yi is the corresponding cumulative number of failures. The failure data set needs to be normalized in the range of [0, 1] on their maximum values before feeding to the ANNs.

4.4 Back-propagation Training Algorithm

The proposed ANN-based SRGM with testing effort function is trained through back-propagation training algorithm with the normalized real software failure data sets, and the weights of the neural network are adjusted to reduce the error. First, the weights of the network are initialized randomly. The weights are updated by gradient descent method in which the weight change is proportional to the partial derivative of the error on the each weight of the network. The weights of the network are iteratively trained with the errors propagated back from

the output layer. This iterative adjustment of the network weights continues until the error is less than a predefined tolerance limit.

5. Experiment Evaluation Results

5.1 Failure Data Sets

Two practical failure data sets[33] Musa P1 and Musa P14c are used to test the evaluation of the proposed ANN-based SRGM with testing effort. Here these two data sets are referred as DS-1 and DS-2 respectively. The datasets[34] are described as follow:

- DS-1: 136 failures with 21,700 lines of code for a project of real time control system which was collected by Musa at Telephone Laboratories.
- DS-2: 101 failures from 1,80,000 lines of code which was taken from Musa's military application project.

Most of the researchers take DS-1 to confirm their model performance since the testing process was under control and exhibited neither the learning phenomenon nor fluctuations in data. On the contrary, DS-2 provides challenges to SRGMs because of the fact that wide fluctuation in data as well as the learning phenomenon of the testing team. The later project appears to have exhibited turbulent behaviour.

5.2 Model Evaluation Criteria

To compare the performance of software reliability models, the researchers proposed some criteria to evaluate the models. We measure the performance of the proposed model by using R^2 (Coefficient of determination), RMSE (Root Mean Square Error) and AIC (Akaike Information Criterion). The value of R^2 may vary from 0 to 1. The closer R^2 is to 1; the better is the fit. RMSE is the criterion to measure the difference between the actual and predicted values. AIC is a measure of goodness of fit of an estimated statistical model. AIC is considered to be a measure which can be used to rank the models, and it gives a penalty to a model with more number of parameters. RMSE and AIC are computed as follows:

$$\mathrm{RMSE} = \sqrt{1/p}\sum\left(E-O\right)^2 \tag{13}$$

Where p is the total number of points taken for testing performance, E is the estimated testing error and O is the actual observed testing error. The lesser RMSE represents the best fit performance.

Table 1. Comparisons of performance

	M1			M2			PRANN		
	R^2	RMSE	AIC	R^2	RMSE	AIC	R^2	RMSE	AIC
DS-1	0.994	0.093	10.47	0.996	0.061	7.81	0.998	0.038	4.72
DS-2	0.971	0.272	12.63	0.989	0.098	9.42	0.994	0.082	5.56

M1: Traditional time-based log power model without testing effort.
M2: Neural network based log power model without testing effort.
PRANNTEM: Proposed Artificial Neural Network Testing Effort model.

$$\mathrm{AIC} = p\log\left(\frac{\mathrm{RSS}}{p}\right)+2W \tag{14}$$

Where p indicates the number count for observations, RSS represents residual sum squares, and W is the respective weights in the network architecture. Lesser AIC indicates the best goodness of fit network architecture.

5.3 Performance Analysis

We evaluate the performance analysis of the proposed model using ANN based approach with and without incorporating testing effort to prove that the former is better.

The result analysis of R^2, RMSE and AIC are given in Table 1. As expected the effort based models seem to give better performance than time based traditional model. It is also clear that estimation of parameters using neural network improves the performance further. The data set DS-2 has wide fluctuations due to the learning phenomenon of the testing team as compared to DS-1 as revealed by all the goodness of fit indices. Traditional models are sensitive to wild fluctuations in data. But proposed ANN-based SRGM with testing effort function will perform better even when the learning phenomenon occurs, and there are reasonable fluctuations in data. AIC is considered to provide overall performance index of a model for a given dataset. Table 1 provides the ranking for the performance of the models for the chosen datasets.

6. Conclusion

A large number of time-based SRGMs were proposed in the past and such SRGMs assume constant efforts during the entire testing period. However, this is not a reality. Hence, a few effort based SRGMs were proposed in the past. In this paper, we proposed a simple log power SRGM with exponential testing effort function estimated param-

eters using the artificial neural network. We compare the performance of the SRGM when parameters were estimated in the traditional manner and with ANN, with and without incorporating testing effort. The study confirms that ANN based testing effort models describe failure data even when there are wide fluctuations in data which prohibit modelling.

7. Acknowledgment

This research work is funded and supported by the Department of Science and Technology, Government of India under the Grant DST/Inspire Fellowship/2013/849.

8. References

1. Michael RL. Handbook of software reliability engineering. IEEE Computer Society Press. New York: McGraw Hill; 1996.
2. Xie M. Software reliability modelling. Singapore: World Scientific; 1991.
3. Goel AL, Okumoto K. Time-dependent error-detection rate model for software reliability and other performance measures. IEEE trans on Reliability. 1979; 28(3):206–11.
4. Yamada S, Osaki S. Software reliability growth modeling: Models and applications. IEEE Transactions on Software Engineering. 1985; 11(12):1431–37.
5. Ohba M. Chou XM. Does imperfect debugging affect software reliability growth? Proc 11th International Conf Software Eng; New York. 1989. p. 237–44.
6. Kapur PK, Garg RB. A software reliability growth model for an error removal phenomenon. Software Engineering Journal. 1992; 7(4):291–4.
7. Goel AL. Software reliability models: Assumptions, limitations and applicability. IEEE Trans on Software Eng. 1985; 11(12):1411–23.
8. Subburaj R, Gopal G. Generalized exponential poisson model for software reliability growth. International J of Performability Eng. 2006; 2(3):291–301.
9. Subburaj R, Gopal G, Kapur PK. A software reliability growth model for estimating debugging and the learning indices. International J of Performability Eng. 2012; 8(5):539–49.
10. Lai R, Garg M. A detailed study of NHPP software reliability models. Journal of Software. 2012; 7(6):1296–306.
11. Roy P, Mahapatra GS, Dey KN. An NHPP software reliability growth model with imperfect debugging and error generation. International Journal of Reliability, Quality and Safety Engineering. 2014; 21(2):1–32.
12. Rana R, Staron M, Berger C, Hansson J, Nilsson M, Torner F. Selecting software reliability growth models and improving their predictive accuracy using historical projects data. Journal of Systems and Software. 2014; 98:59–78.
13. Roy GSP, Dey MKN. Neuro-genetic approach on logistic model based software reliability prediction. Expert systems with Applications. 2015; 42(10):4209–718.
14. Zhixin J, Hong-bin Z. Research in reliability modelling of WEDM based on neural network. Proc IEEE Symp Measuring Technology and Mechatronics Automation (ICMTMA'09); Zhangjiajie, Hunan, China. 2009. p. 277–80.
15. Karunanithi N, Whitley D, Malaiya YK. Using neural networks in reliability prediction. IEEE Software. 1992; 9:53–9.
16. Karunanithi N, Whitley D, Malaiya YK. Prediction of software reliability using connectionist models. IEEE Trans on Software Eng. 1992; 18(7):563–74.
17. Sitte R. Comparison of software-reliability-growth predictions: Neural networks vs parametric recalibration. IEEE Trans on Reliability. 1999; 48(3):285–91.
18. Cai KY, Cai L, Wang WD, Yu ZY, Zhang D. On the neural network approach in software reliability modeling. J Systems and Software. 2001; 58(1):47–62.
19. Su YS, Huang CY. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. J of Systems and Software. 2007; 80(4):606–15.
20. Zheng J. Predicting software reliability with neural network ensembles. Expert Systems with Applications. 2009; 36(2):2116–22.
21. Wang G, Li W. Research of software reliability combination model based on neural network. IEEE second WRI World Congress on Software Eng. 2010; 2:253–6.
22. Kapur PK, Yadavalli VSS, Khatri SK, Basirzadeh M. Enhancing software reliability of a complex software system architecture using artificial neural-networks ensemble. International Journal of Reliability, Quality and Safety Engineering. 2011; 18(3):271–84.
23. Roy P, Mahapatra GS, Rani P, Pandey SK, Dey KN. Robust feed forward and recurrent neural network based dynamic weighted combination models for software reliability prediction. Applied Soft Computing. 2014; 22:629–37.
24. Das TK. Intelligent techniques in decision making: A survey. Indian Journal of Science and Technology. 2016; 9(12).
25. Jabarullah BM, Babu CNK. BPNN-hippoamy algorithm for statistical classification. Indian Journal of Science and Technology. 2015; 8(14).
26. Gayathry G, Selvi RT. Classification of software reliability models to improve reliability of software. Indian Journal of Science and Technology. 2015; 8(29).

27. Kapur PK, Grover PS, Younes S. Modeling an imperfect debugging phenomenon with testing effort. Proceedings 5th International Symposium on Software Reliability Engineering; Monterey, CA. 1994, p. 178–83.
28. Yu C. An assessment of testing-effort dependent software reliability growth models. IEEE Transactions on Reliability. 2007; 56(2):198–211.
29. Yamada S. Software reliability growth model with testing effort. IEEE Transactions on Reliability. 1986; 35(1):422–24.
30. Yamada S. Software reliability growth model with Weibull testing effort. IEEE Transactions on Reliability. 1993; 42(1):100–05.
31. Kapur PK, Yadavalli VSS, Khatri SK, Basirzadeh M. Enhancing software reliability of a complex software system architecture using artificial neural-networks ensemble. International Journal of Reliability, Quality and Safety Engineering. 2011; 18(3):271–84.
32. Zhao M, Xie M. On the log-power NHPP software reliability model computing. IEEE Transactions on Reliability. 1992; 14–22.
33. Musa JD. DACS software reliability dataset. Data and Analysis Center for Software; 1980 Jan.
34. Indhurani L, Subburaj R. An artificial neural network approach to software reliability growth modelling. Procedia Computer Science. 2015; 57:695–702.

Appendix

Back Propagation training algorithm Pseudo code

Step 1

Present the normalized input and output pattern.

Initialize the weight to small random numbers.

Set the condition criteria (number of iterations or error rate).

Step 2

$v_m \rightarrow$ output of a neuron n.

- Output of an input neuron = The value of the corresponding input data.
- Output of a hidden neuron or an output neuron are $v_n = F(NET)$ where NET = $\Sigma_m w_{nm}. v_m$

F is the activation function, w_{nm} is the respective weight from m to n and v_m is corresponding input value.

Step 3

Back Propagation training

Starting from the output layer and go back to the hidden layers to find error gradient ∂_j

Error $E = E + \frac{1}{2}(e-a)^2$ where e is the estimated output and a is the actual observed output.

Error for output layer neurons $\partial_0 = (e-a) * (f^1(NET))$

Similarly find the error for hidden layer neurons.

Step 4

Weight Adjustments

Weight adjustments for output layer $w_{ji} = w_{ji} + \Delta w_{ji}$ where $w_{ji} = c.\partial_0.v_i$ here c is the learning rate co-efficient.

Similarly do the weight adjustments for hidden layer neurons.

Repeat the feed-forward calculation from step 2 to step 4 until the stopping criteria is met.

Here gradient descent method is used to upgrade the weights i.e. if the weight adjustment increases then the derivative of the weight adjustment error will also increase with the respective weights of the neural network architecture.