Cluster Computing (2019) 22:S3095–S3102
https://doi.org/10.1007/s10586-018-1942-4

Software reliability modeling using increased failure interval with ANN

K. Kumaresan[1] · P. Ganeshkumar[2]

Received: 26 December 2017 / Revised: 24 January 2018 / Accepted: 30 January 2018 / Published online: 3 March 2018

Abstract
In growing Software industry, whenever engineers developed new software, they want to make sure that it is failure free and reliable. With the increasing reliability of hardware and growing complexity of software, the software reliability is a rising concern for both developer and users. Software reliability is the key part of quality and customer satisfaction. For the last three decades, many software reliability models have been successfully utilized in practical software reliability engineering. However, no single model can obtain the accurate prediction for all cases. This paper proposed the software reliability model with the increased number of training data set and neural networks. The back propagation algorithm has been chosen and applied for a learning process. The obtained results show that the proposed model increases the accuracy of the software reliability prediction.

Keywords Software reliability · Failure prediction · Software reliability prediction · Neural network

1 Introduction

The software has been increasingly used in our daily life and has become a crucial part of critical and non-critical applications [1]. Because of this increasing use and importance, the assurance of software quality becomes an issue of critical concern. Software quality can be expressed by quality requirements or attributes such as reliability, availability, safety, security, and performance [2]. Among these software quality attributes, software reliability is generally considered as the most important factor. It quantifies software faults and failures, which can lead to serious consequences in safety–critical systems as well as in normal business. Therefore, assessing, estimating, and predicting software reliability have been increasingly demanded in projects in order to achieve highly reliable software systems [3].

Software reliability is a process of providing failure-free solutions until the lifetime of the software. Software reliability Modelling has the benefits of increasing the profit of the organization in many ways which include Reduction in time to deliver, Reduction in total lifecycle cost, Improvement in levels of quality, Improvements on customer satisfaction ratings, an improvement on the supplier relationships and delivery of complete specified functionality [4]. This software reliability measurement is a set of mathematical techniques that can be used to estimate and predict the reliability behaviour of software during its development and operation.

Software reliability growth models refer to those models that try to predict software reliability from test data. These models try to show a relationship between fault detection data (i.e. test data) and known mathematical functions such as logarithmic or exponential functions. The goodness of fit of these models depends on the degree of correlation between the test data and the mathematical function [5]. Typically two broad categories of software reliability growth models (SRGMs) include parametric models and nonparametric models. Most of the parametric models are based on non-homogeneous Poisson process (NHPP) that has been widely used successfully in practical software reliability engineering [6].

In the past few years, a large number of software reliability models/statistical models have been developed. Jelinski–Moranda Model, Shooman Model, Musa Model, geometric models of Moranda and Ramamoorthy–Bastani

✉ K. Kumaresan
kumaresankstu@outlook.com

[1] Department of Computer Science & Engineering, Study World College of Engineering, Coimbatore, India

[2] Department of Information Technology, Anna University Regional Campus, Coimbatore, India

and Schick–Wolverton model are some example for statistical models [7]. Every model is based on certain assumptions. So the predictive capability of different models is different for different datasets. There exist no single models that can best suit in all cases. The statistical models are also influenced by different external parameters [8]. To overcome these problems, non-parametric models like neural network and support vector machines have been used for last few years [9]. The non-parametric models are not influenced by any external parameter and also not based on assumptions which are unrealistic in real situations.

Traditional modelling approaches are provides solution, if failure and repair data were always available [10]. This is not a very realistic assumption and although it might be possible to use parameter estimates from the many handbooks for different models. The Bayesian approach might help the researchers to solve this issue and provides better estimation with help of available additional data [11]. Gathering the failure data is not very easy because many different industries practising very poor route cause failure analysis. At that time, the failure modes are not clearly identifiable from the databases, from a more practical view, does not provide the information needed to accurately fit statistical models for reliability analyses [12].

Soft computing techniques emerged from the studies of natural systems. In the past, soft computing methods were difficult to be implemented because of computational power limitations. Examples of bio-inspired techniques are Artificial Immune systems (AIS), Fuzzy systems (FS), Artificial Neural Networks (ANN), Swarm Intelligence (SI) and Evolutionary Computing (EC). These techniques are also known as Computational Intelligence (CI) techniques and are part of Artificial Intelligence (AI) research area [13]. Sometimes one or more models combined among themselves and with stochastic methods in order to develop a solution for solving complex engineering problems.

The Neural network is the techniques, which is inspired from the behaviour of human brain. It is a collection of the number of artificial neurons. It can be customized according to the needs and problems. As each artificial neuron takes a number of inputs and provides the single output [14]. An artificial neuron performed the complex calculation on input on the basis of a used transfer function to produce the desired output. As the size of the neural network increases the complexity of the system also increases. Therefore there is need to use the number of layers, the number of neurons, transfer function as less as possible. The Neural network is capable to perform a complex function in many fields like in pattern recognition, classification, speech recognition, vision and control system.

2 Literature survey

Kaswan et al. [15] presented neural network based model to predict the failures of the system. They used execution time as the input of the neural network. They used different networks like Feed Forward neural networks, recurrent neural networks like Jordan neural network and Elman neural network in their approach. They used two different training regimes like Prediction and Generalization in their study. They compared their results with some statistical models and found better prediction than those models.

Cai et al. [16] also used connectionist models for software reliability prediction. They applied the Falman's cascade Correlation algorithm to find out the architecture of the neural network. They considered the minimum number of training points as three and calculated the average error (AE) for both endpoint and next-step prediction. Their results concluded that the connectionist approach is better for endpoint prediction.

Lakshmanan et al. [17] provided the model to improve the accuracy of software reliability prediction combine the software reliability models with the neural networks (NN). Particle swarm optimization (PSO) algorithm has been chosen and applied for learning process to select the best architecture of the neural network. Neural networks trained by particle swarm optimization (PSO) has shown to be an effective nonparametric technique for software reliability prediction by optimizing the mean squared error, Selecting the best architecture of the network are also concerned for enhancing the performance of the model.

Roy et al. [18] developed ensemble models to forecast software failure. One non-linear and three linear ensembles are implemented. Various statistical and intelligent techniques are investigated against the ensembles. They are multivariate adaptive regression splines, TreeNet, multiple linear regression, dynamic evolving neuro–fuzzy inference system and back propagation neural network. Finally concluded the non-linear ensemble technique provided better performance compared to other ensemble techniques.

Provided approach is regarded as an extension of separate ANN model under the same modeling framework and is a complement to analytical models which only describe the influence of FDP on FCP as a time delay. With practical software testing data, this approach shows its advantage in incorporating more information than the separate ANN model and paired analytical model. Also, within the combined ANN models, both feed forward and recurrent frameworks perform well with the given dataset. The combined ANN models are beneficial in incorporating the correlation between FDP and FCP. They model the

software debugging process more realistically, with more accurate predictions.

Proposed Artificial Feed Forward network with exponential and logarithmic functions. The result of encoding and various parameters have been analysed. The effect of changing hidden node size has also noted. The experimental result highlights the proposed approach gives acceptable results for various data sets using the same neural network architecture.

Proposed a neural network based method for software reliability prediction. They used back propagation algorithm for training. They used multiple recent 50 failure times as input to predict the next-failure time as output. They evaluated the performance of the approach by varying the number of input nodes and hidden nodes. They concluded that the effectiveness of the approach generally depends upon the nature of the handled data sets.

3 Research methods

In this section, we present the proposed system for software reliability prediction based on the recurrent neural network with subdivided intervals.

3.1 Software reliability data

Sixteen software failure data sets from (Table 1) Musa Data collections are used to check the performance and validity of the proposed method.

Table 1 Musa data collection

Data sets	Number of failure	Software type
DS1	136	Real time command & control
DS2	54	Real time command & control
DS3	38	Real time command & control
DS4	53	Real time command & control
DS5	831	Real time commercial
DS6	73	Commercial subsystem
DS7	36	Real Time
DS8	38	Military
DS9	41	Military
DS10	101	Military
DS11	112	Operating system
DS12	375	Operating system
DS13	277	Operating system
DS14	192	Time sharing system
DS15	278	Word processing system
DS16	196	Operating system

Figures 1 and 2 show the intervals of DS1 and DS6. The software failure data arranged in the pair of (Eti,Iti,Cfi) where Eti, the execution time for ith software failure. Iti = E ti _E ti_1 is the time interval between the (i−1)th and ith software failures. Cfi is the Cumulative failure. The goal of software reliability prediction model is to predict the number of failures in future execution time with help of software failure dataset. The failure datasets are normalized in the range of [0, 1] on their maximum values before feeding into the ANN to maintain the large variation of prediction and forecasting.

3.2 Neural network architecture

Figure 3 shows the architecture of an artificial neural network. It is designed with an input layer, output layer and one or more hidden layers. Every single node from input layer is interconnected to a node from hidden layer and each node from the hidden layer is interconnected to a node in the output layer. Usually every connection is associated with some weights. Input layer represents input data that is fed into the network. This part of the network is never changing its values. Every single input node value is sends to the nodes in the hidden layer. Hidden Layer accepts value from the input layer and modifies them using some weight value, this new value is then send to the output layer but it will also be modified by some weight from the connection between hidden layer and the output layer. Output layer process the value received from the hidden layer and produces an output. This value is then transforming the activation level of a value into an output signal using activation function.

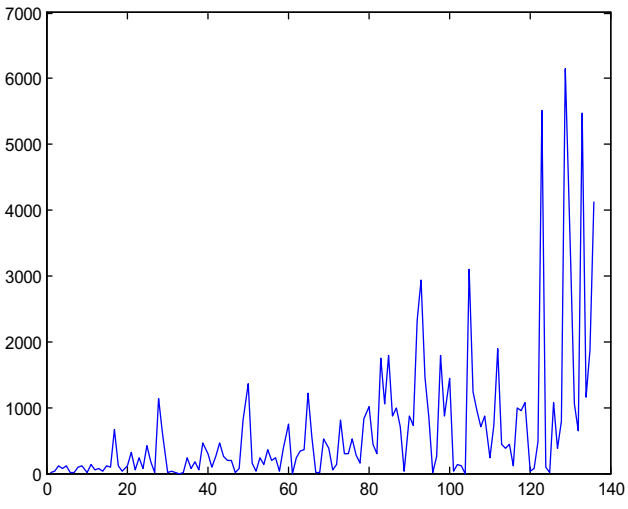

Fig. 1 DS1 failure intervals

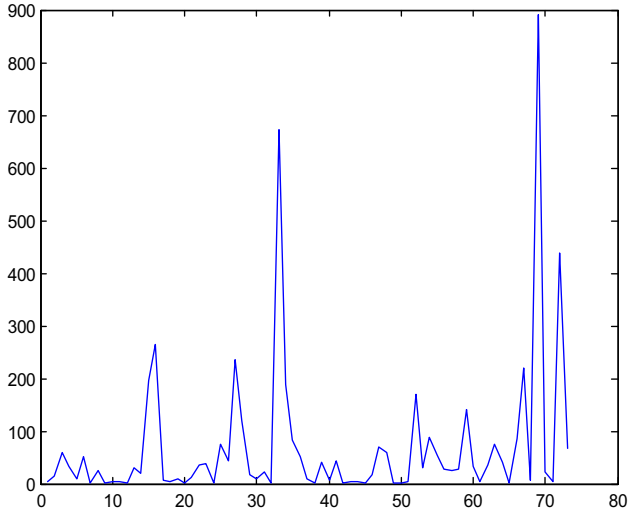

Fig. 2 DS6 failure intervals

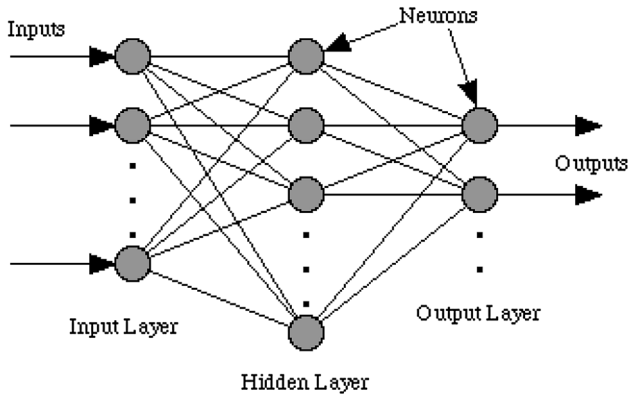

Fig. 3 Architecture of a neural network

3.2.1 Number of nodes and layers

Selecting the number of nodes for each layer is depends upon the problems and its environment parameters like types of the data networks are dealing with, quality of data and some other parameters. Finding the number of nodes in hidden layer could be a tricky task. If the hidden layer nodes are increased, the number of possible computations that algorithm has to deal with increases. Picking just a few nodes in hidden layer can prevent the algorithm of its learning ability. Right balance needs to be picked. It is very important to monitor the progress of NN during its training, if results are not improving, some modification to the model might be needed.

3.2.2 Setting weights

The way to control NN is by setting and adjusting weights between nodes. Initial weights are usually set at some random numbers and then they are adjusted during NN training. The Focus should not be at changing one weight at the time, changing all the weights should be attempted simultaneously. Some NN are dealing with thousands, even millions of nodes so changing one or two at the time would not help in adjusting NN to get desired results in a timely manner. The Logic behind weight updates is quite simple. During the NN training weights are updated after iterations. If results of NN after weights updates are better than the previous set of weights, the new values of weights are kept and iteration goes on. Finding the combination of weights that will help us minimize error should be main aim when setting weights. This will become bit more clear once the learning rate; momentum and training set are explained.

3.2.3 Training neural network

When training neural network, the network is feeding with the set of examples that have inputs and desired outputs. Choosing the learning rate and momentum will help with weight adjustment. Setting right learning rate could be a difficult task, if learning rate is too small, the algorithm might take a long time to converge. On the other hand, choosing large learning rate could have opposite effect, the algorithm could diverge. Sometimes in NN, every weight has its own learning rate. Learning rate of 0.35 proved to be popular choice when training NN. This proposed work uses Learning rate of 0.45 but this value is used because of simple architecture of NN used in example. Large values of momentum term will influence the adjustment in the current weight to move in same direction as previous adjustment.

3.2.4 Activation function

In Fig. 4 activations function is needed for the hidden layer of the NN to introduce nonlinearity. Without them, NN would be same as plain perceptions. If the linear function were used, NN would not be as powerful as they are. Activation function can be linear, threshold or sigmoid function. The Sigmoid activation function is usually used for hidden layer because it combines nearly linear behaviour, curvilinear behaviour and nearly constant behaviour depending on the input value.

Here x1, x2, x3,……xn are n inputs to the artificial neuron. w1x1, w2x2, ….wnxn are the weights attached to the input links. The initial bias assigned to the neural network hidden layer and output layer is zero. The output of the Neural Network calculation process is defined as follows.

$$y = f(s) \quad \text{and} \quad s = \sum_{i=0}^{n} wixi \tag{1}$$

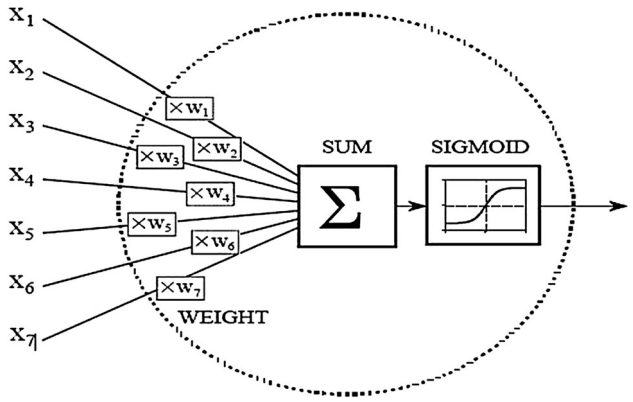

Fig. 4 Activation process

where

y = output of the network

f() = activation function

$\sum$ = collection of the output nodes from hidden layer that have been multiplied by connection weights, added to get single number and put through sigmoid function

n = input elements

wi = w1,w2,w3,…wn are weight of respective inputs

Input to sigmoid is any value between negative infinity and positive infinity number while the output can only be a number between 0 and 1.

3.2.5 Back propagation (BP) algorithm

One of the most popular NN algorithms is backpropagation algorithm. Back Propagation algorithm could be broken down into four main steps. After choosing the weights of the network randomly, the backpropagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following steps:

Algorithm:

i) Feed-forward computation

ii) Backpropagation to the output layer

iii) Backpropagation to the hidden layer

iv) Weight updates

4 Implementation and result

In this work, used Software Reliability Dataset was compiled by John Musa of Bell Telephone Laboratories. His objective was to gather failure interval dataset to help the software engineers in monitoring test status and assist the software researchers in validating software reliability models. Execution time was reported in terms of wall-clock seconds except DS6 and faults in terms of cumulative faults.DS6 Executive time was reported in CPU seconds. The failures are not raised at the particular time rather it already starts at the end of the previous failure. This fact leads to sub divide the both cumulative failure interval and cumulative fault due to this the Neural network get more data set. The increased number of data sets improves the training performance of the Neural Network. Cumulative execution time and cumulative faults are normalized between 0.1 and 0.9.

Neural network cannot predict future faults without learning the software's failure history. Any prediction without training is equivalent to making a random guess. Most training algorithms sets the neural network weights with random values at the starting state of training, which bring about the network to converge to different weight at the end of each training.In Proposed technique, used different hidden layer size for analysing the performance. The same set of hidden layer size used both increased training data set model and without increasing dataset model. Trainlm training function is used to updates weight and bias values according to Levenberg–Marquardt optimization.

4.1 Performance measures

In Tables 2 and 3 variable-term prediction has been used in the proposed approach, which is commonly used in the software reliability research community. Only part of the failure data is used to train the model and the trained model (Figs. 5 and 6) is used to predict for the rest of the failure data available in the data set. For a given execution time ti, if the predicted number of failure is Ni', then Ni' is compared with the actual number of failures i.e. Ni to calculate three performance measures such as Average relative Error (AE) and RMSE which are defined as follows:

$$\mathrm{AE} = \frac{1}{k}\sum_{i=1}^{k} abs\left(\frac{N_i' - N_i}{N_i}\right) * 100 \tag{2}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{k}\sum_{i=0}^{k}\left[N_i - N_i'\right]^2} \tag{3}$$

where k is the number of observation for evaluation.

It is shown that no single parametric model can fit well to both two datasets. Among the four parametric models, (Tables 4, 5 and 6) Duane model has the best performance for DS1 but the worst performance for DS6. The best parametric model for DS6 is G–O model. It also can be observed that the proposed system has lower prediction error than the other model system for both datasets, which confirms our initial intention that the neural network with subdivided failure intervals shows dataset can be a better software reliability predictor than other Models.

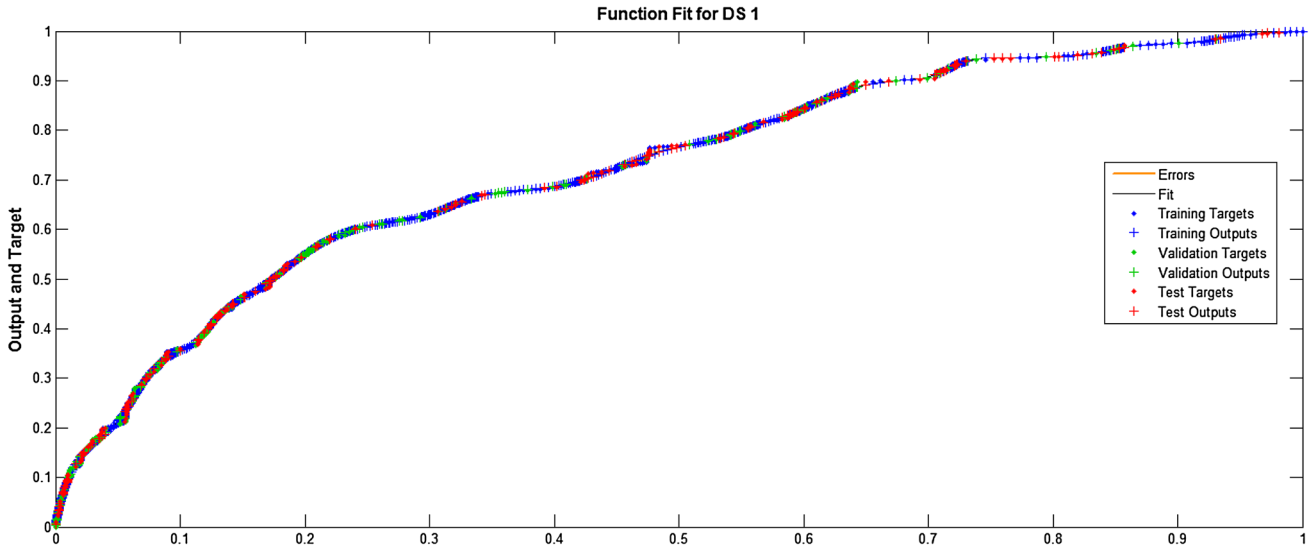

Fig. 5 Performance of DS1

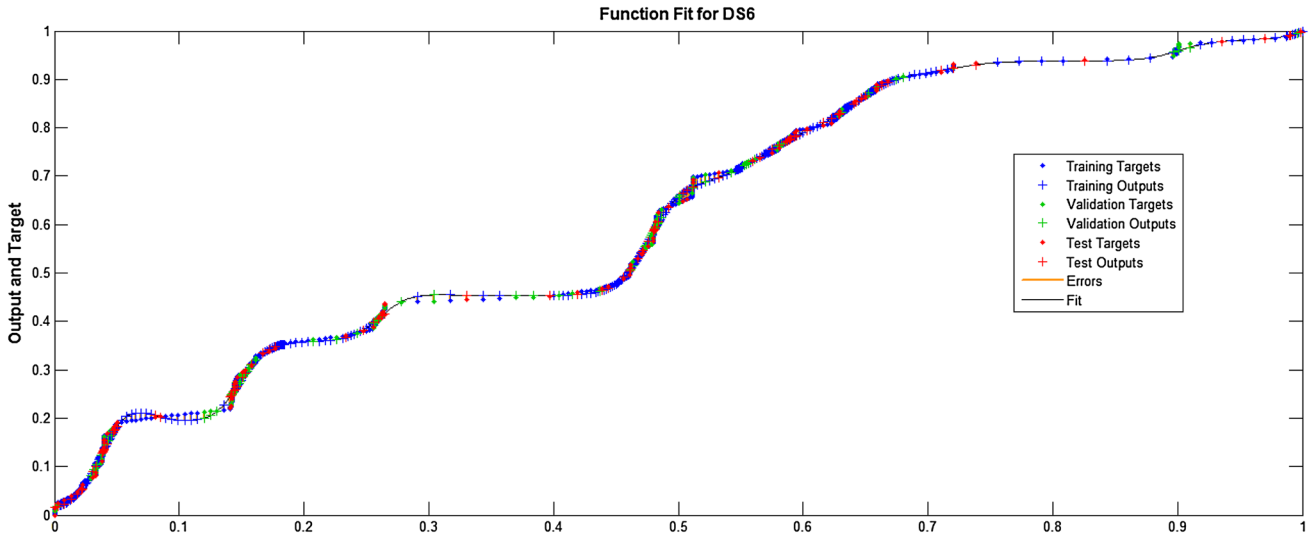

Fig. 6 Performance of DS6

Table 2 DS1 performance before increasing training data size

S. No	Hidden layer	Epoch	Performance	Training performance	Validation performance	Test performance
1	12	23	0.000047802	0.000036715	0.000047560	0.000101260
2	15	17	0.000048165	0.000043085	0.000083487	0.000037230
3	20	14	0.000192470	0.000036550	0.001100000	0.000080215
4	30	10	0.000151640	0.000034955	0.000812740	0.000050615
5	40	16	0.000123760	0.000021526	0.000465160	0.000273050

Table 3 DS1 performance after increasing training data size

S. No	Hidden layer	Epoch	Performance	Training performance	Validation performance	Test performance
1	12	95	0.000032365	0.000032357	0.000035889	0.000028880
2	15	124	0.000028650	0.000027107	0.000028516	0.000035985
3	20	27	0.000029749	0.000029879	0.000028076	0.000030816
4	30	20	0.000021964	0.000022387	0.000019013	0.000022937
5	40	283	0.000009291	0.000008815	0.000011581	0.000009226

Table 4 DS6 performance before increasing training data size

S. No	Hidden layer	Epoch	Performance	Training performance	Validation performance	Test performance
1	12	26	0.000121310	0.000116430	0.000109040	0.000156180
2	15	18	0.000165250	0.000129100	0.000236220	0.000261910
3	20	8	0.001700000	0.000299410	0.009000000	0.001200000
4	30	10	0.002800000	0.000056605	0.000773330	0.017300000
5	40	8	0.036400000	0.000254400	0.160800000	0.079500000

Table 5 DS6 performance after increasing training data size

S. No	hidden layer	Epoch	Performance	Training performance	Validation performance	Test performance
1	12	14	0.000106160	0.000104440	0.000108520	0.000111800
2	15	69	0.000081161	0.000077475	0.000086973	0.000092439
3	20	70	0.000060078	0.000058856	0.000053318	0.000072502
4	30	44	0.000051543	0.000052655	0.000056224	0.000041707
5	40	71	0.000469530	0.000035602	0.001500000	0.001400000

Table 6 Results of the training and testing dataset obtained using the proposed model

Dataset	Proposed method	G–O model	Duane model	S-shaped model
DS1	1.21	11.1551	2.8431	18.0817
DS6	1.06	97.135	121.8711	9.8745

5 Conclusion

Software reliability prediction is a significant factor for increasing software quality and getting better customer satisfaction. This Proposed work highlights the software failure prediction technique based on increased failure interval Dataset with ANN back propagation learning algorithm. The experimental results show that the proposed system achieves significantly lower prediction error compared with the traditional Software Reliability Prediction Models, which proves that the neural network back propagation algorithm and sub divided failure intervals are providing better software reliability prediction. On the other hand, it exposed the Neural Network method proposed in this paper using back propagation algorithm provides good fit for different types of datasets while changing hidden layer size and activation function. Nevertheless, the result of the method is differs according to the failure datasets, network architecture and simulation tools.

References

1. Huang, C-Yu., Lyu, M.R.: Estimation and analysis of some generalized multiple change-point software reliability models. IEEE Trans. Reliab. **60**(2), 498–514 (2011)
2. Zheng, J.: Predicting software reliability with neural network ensembles. Expert Syst. Appl. **36**, 2116–2122 (2009)
3. Amin, A., Grunske, L., Colman, A.: An approach to software reliability prediction based on time series modeling. J. Syst. Softw. **86**, 1923–1932 (2013)
4. Rita, G., Nada, N.S.: Software reliability prediction using artificial techniques. Int. J. Comput. Sci. Issues **10**(4), 274–281 (2013)
5. Tkachtenberg, M.: A general theory of software-reliability modeling. IEEE Trans. Reliab. **39**(1), 92–96 (1990)
6. Bisi, M., Goyal, N.K.: Software reliability prediction using neural network with encoded input. Int. J. Comput. Appl. **47**(22), 46–52 (2012)
7. Salgado, M.F., Caminhas, W.M., Menezes, B.R.: Soft computing approaches in reliability modeling and analysis of repairable systems. In: *IEEE, Annual Reliability and Maintainability Symposium (RAMS)*, pp. 1–5 (2010)
8. Kumar, R., Gupta, D.L.: Software bug prediction system using neural network. Eur. J. Adv. Eng. Technol. **3**(7), 78–82 (2016)
9. Karunanithi, N., Whitley, D., Malaiya, Y.K.: Prediction of software reliability using connectionist models. IEEE Trans. Softw. Engg **18**(7), 563–573 (1992)
10. Karunanithi, N., Whitley, D., Malaiya, Y.K.: Using neural networks in reliability prediction. IEEE Softw. **9**(4), 53–59 (1992)
11. Raj Kiran, N., Ravi, V.: Software reliability prediction by soft computing techniques. J. Syst. Softw. **81**(4), 576–583 (2008)
12. Hu, Q.P., Xie, M., Ng, S.H.: Software reliability predictions using artificial neural networks. Stud. Comput. Intell. **40**, 197–220 (2007)
13. Gokhale, S.S., Lyu, M.R.: A simulation approach to structure-based software reliability analysis. IEEE Trans. Softw. Eng. **31**(8), 643–656 (2005)
14. Almering, V., van Genuchten, M., Cloudt, G., Sonnemans, P.J.: Using software reliability growth models in practice. IEEE Softw. **24**(6), 82–88 (2007)
15. Kaswan, K.S., Choudhary, S., Sharma, K.: Software reliability modeling using soft computing techniques: critical review. Inf. Technol. Softw. Eng. **5**(1), 144 (2015)
16. Cai, K.Y., Cai, L., Wang, W.D., Yu, Z.Y., Zhang, D.: Onn the neural network approach in software reliability modeling. J. Syst. Softw. **58**, 47–62 (2001)
17. Lakshmanan, I., Ramasamy, S.: Application of artificial neural network for software reliability growth modeling with testing effort. Indian J. Sci. Technol. **9**(29), 1–7 (2016)
18. Roy, P., Mahapatra, G.S., Dey, K.N.: Neuro-genetic approach on logistic model based software reliability prediction. Expert Syst. Appl. **42**(10), 4709–4718 (2015)

K. Kumaresan received his B.E. degree in Computer Science and Engineering from Anna University, Chennai, Tamil Nadu, India in 2006, M.Tech in Information Technology from Anna University, Chennai, Tamil Nadu, India in 2012. Having 5 years teaching experience, 3 years Industry experience. His research interest includes Software Engineering, Software Reliability, Software Design and Architectures, Information Management. At present, he is working as Assistant Professor, with Department of Computer Science and Engineering in Study World College of Engineering, Coimbatore, Tamil Nadu, India.

P. Ganeshkumar received his B.Tech., M.S.(by research), and Ph.D. in Information Technology in 2003, 2008 and 2012 from the University of Madras, Anna University Chennai, and Anna University Coimbatore, India respectively. He completed his Post Doc from the School of Computer Science and Engineering, Kyungpook National University, South Korea from April 2015 to March 2016. His research interest includes the development of soft computing techniques for problems in data mining, wireless sensor networks, bio informatics, Smart Grid, Image Processing, Software Reliability Models and big data analytics. He is the resource person for delivering technical talk in UGC, AICTE, TEQIP, ICMR, IETE, and IEEE sponsored seminars in various technical institutions. At present, he is working as Assistant Professor, with Department of Information Technology in Anna University Regional Campus, Coimbatore, Tamil Nadu, India.