International Journal of Electrical and Computer Engineering (IJECE)
Vol. 6, No. 4, August 2016, pp. 1929 – 1938
ISSN: 2088-8708, DOI: 10.11591/ijece.v6i4.9991

Software Reliability Prediction using Fuzzy Min-Max Algorithm and Recurrent Neural Network Approach

Manmath Kumar Bhuyan[*], Durga Prasad Mohapatra[**], and Srinivas Sethi[***]
[*]Computer Science Engineering and Application, Sarang, Utkal University, Vanivihar, India
[**]Computer Science Engineering, National Institute of Technology, Rorkela, India
[***]Computer Science Engineering and Application, IGIT, Sarang, India

Article Info

Article history:
Received Dec 23, 2015
Revised May 23, 2016
Accepted Jun 8, 2016

Keyword:
Fuzzy Min-Max
K-Means
Software Reliability Prediction
Recurrent Neural Network
Back-propagation

ABSTRACT

Fuzzy Logic (FL) together with *Recurrent Neural Network (RNN)* is used to predict the software reliability. Fuzzy Min-Max algorithm is used to optimize the number of the k-gaussian nodes in the hidden layer and delayed input neurons. The optimized recurrent neural network is used to dynamically reconfigure in real-time as actual software failure. In this work, an enhanced fuzzy min-max algorithm together with recurrent neural network based machine learning technique is explored and a comparative analysis is performed for the modeling of reliability prediction in software systems. The model has been applied on data sets collected across several standard software projects during system testing phase with fault removal. The performance of our proposed approach has been tested using distributed system application failure data set.

Corresponding Author:
Manmath Kumar Bhuyan
Utkal University
Vanivihar, Bhubaneswar
Phone: 09437931445
Email: manmathr@gmail.com

1. INTRODUCTION

Software reliability prediction in software systems plays a key part for any software organization to produce quality and reliable software. The key part of software quality is it's reliability. So predicting software reliability plays a key part for producing good quality software. As per IEEE Standard Glossary of Software Engineering, a definition of software reliability is the probability of the failure free operation of a computer program for a specified period of time in a specified environment [1, 2, 3, 4]. *Computational Intelligence* (CI) can offer promising approaches to software reliability prediction and modeling, because they require only failure history as input without any assumption [5].

As per IEEE Standard Glossary of Software Engineering, a definition of software reliability is the probability of the failure free operation of a computer program for a specified period of time in a specified environment [1, 2, 3, 4]. The time duration between successive failures or the cumulative failure time is a vital factor of software reliability [6, 7]. CI can offer promising approaches to software reliability prediction and modeling, because they require only failure history as input without any assumption. In reply to this, neuro-fuzzy approach has been applied to software reliability assessment. Fuzzy Min-Max algorithm is used to optimize the neural network architecture after every occurrence of software failure time data.

The main contribution of this paper is to propose hybrid model of fuzzy logic and neural network to handle dynamic data set of software reliability between number of observed failure along with successive software failures. We propose an adaptive software reliability prediction model *fuzzy min-max with recurrent neural network* (FMM-RNN) approach based on multiple-delayed-input single-output architecture. We structured the data set relationship between failure sequence number and failure time data. Optimization technique is used to model the inter-relationship among software failure time data. In addition, we made a comparative study about the performance of some well-known existing software reliability prediction models against our approach model. Neuro-Fuzzy system is used in

predicting the software reliability from the aspects of prediction ability for short-term prediction.

The paper is organized as follows: Section 2. describes the related work proposed so far in reliability prediction. Section 3.1. presents our proposed model *fuzzy min-max with recurrent neural network (FMMRNN)* architecture. The basic terminologies, application, architecture development is discussed in Section 3.. Section 4. focus on computation of measure criterion of the propose model and observations are presented. In Section 5., the concluding remarks and future work are included.

2. BACKGROUND OF RELATED WORK

Karunanithi et al. [8] first propose a neural network for software reliability prediction. The authors [9, 10, 11, 12] developed a connectionist model for reliability prediction. Raj Kiran et al. [13, 14] implemented the use of *wavelet neural networks (WNN)* and employed wavelets as transfer function to predict software reliability. Pai et al. [15] used support vector machines and simulated annealing algorithms for reliability forecasting. They used lagged data in their analysis by dividing the 101 observations such as: 33 observations for training, 8 observations for validation, 60 observations for test. Since it is not a standard method of splitting the data set for experimentation. [16] used neural network approach for software defect prediction and pointed out that the approach is poor at at predicting number of software defects, but qualitatively good at classifying program modules. Sitte [17], analyzed two methods (i.e. *Neural Networks (NN)* and 2) parametric recalibration models) for reliability prediction.

An early reliability prediction approach at design phase is proposed by Mohanta et al. [18]. The fault is estimated using product metrics collected during design phase of the components. Then these product metrics are used for reliability prediction. Adnan et al. [19] and Cai et al.[20] determined the number of input neurons and the number of neurons in hidden layers were determined using a pre-specified range of values (i.e. 20, 30, 40, and 50 input neurons selected in Cai et al. [20], while 1, 2, 3, and 4 input neurons were selected in Adnan et al. [19] and Cai et al.[20] used genetic algorithm as an optimization search scheme to determine the optimal or near optimal network architecture. Tian and Noor [6] predicted software reliability using RNN. Su et al. [21] build a dynamic weighted combinational model using NN. Lo [22] designed a model examines several conventional software reliability growth models.

3. PROPOSED MODEL FMMRNN RELIABILITY PREDICTION

In this section, we discuss about our proposed model FMMRNN architecture and its applicability in software reliability prediction. Fuzzy Min-Max is a specific type of neuro-fuzzy that has high efficiency rather than other computational methods [23]. The FMMRNN architecture comprises of two steps 1) Network optimization, 2) Reliability prediction. The complete architecture frame work of our propose model is shown in Fig. 1. The model receive the failure data as input then Fuzzy Min-Max algorithm is use to optimizing the neural network architecture. The Fuzzy Min-Max algorithm optimization process determines the optimal or near-optimal numbers of 'k' hidden neurons and initializes the k-centers. On the basis of numbers of neurons in the hidden layer, the network is framed. The cumulative execution time is taken as input and the number of cumulative failures is taken as output to the networks. As this is a supervised learning, so the network is trained with input and output data proved to the model. This information is then used to dynamically reconfigure the neural network architecture for predicting the next-step failure $\hat{d}_{i+1}$.

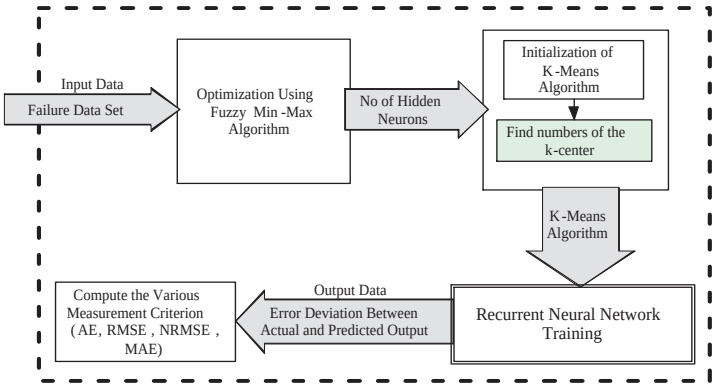

Figure 1. A simple architecture of FMMRNN model

3.1. Recurrent Neural Network

The RNN is based on standard feed-forward neural networks. RNN is a dynamic network as it is a network with output feedback [24].

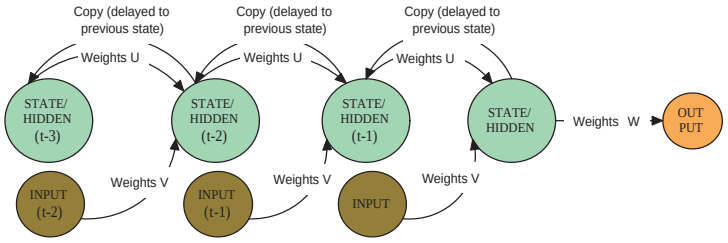

Figure 2. An graphical presentation of unfolding network associated with recurrent back-propagation through time learning.

Fig. 2 shows the RNN with back-propagation through time learning that consists of cycles with its states. In FMMRNN model, the hidden layers are recursive relationship in nature. In recurrent network an extra layer of neurons which copy the current activations in memory (i.e. in the hidden layer neurons)and move forward. Later on, it delays these values for one time instant [25], feed them back as additional inputs into the hidden layer neurons as shown in Fig. 2. As a result of this, each node[1] sends activation along a recurrent connection, has at least τ number(s) of copies. In the supervised learning, an error deviation is the Euclidean distance between the predicted output of the network and actual output. That error deviation is propagated through time [26]. Each time the error is calculated and the weights are folded back and added with error to compute the new updated weights using network training method.

$$y_k(t) = f_2(net_k(t)), \tag{1}$$
$$where,\ net_k(t) = \sum_{j}^{m} y_j(t) w_{kj} + \theta_k$$

Here, f_2 is an activation function between the hidden and output nodes.

In this paper, we consider y_i as the output (predicted failure) with activation function f(.) and w_{ij} as the weight from node j to node i associated with this link. Here the input nodes x_i receive external inputs (i.e. the failure number). The desired state of unit i denoted as d_i corresponding to x_i. The accumulated cost function (i.e. the *Summed Square Error* (SSE)) 'L' in Equation 2 measures the deviation (i.e. difference between the actual and desired values) of the network outputs $y_i(t)$ from the desired functions $d_i(t)$ from $t = t_0$ to $t = t_1$ for all copies of the output nodes.

$$L = \frac{1}{2}\sum_{k=1}^{n}(d_k(t) - y_k(t))^2 = \frac{1}{2}\sum_{k=1}^{n} L_k^2 \tag{2}$$

Where $L_k = \begin{cases} d_k - y_k & , k^{th}\ output\ node \\ 0 & , otherwise \end{cases}$

Here, n is the total no of output nodes and is an index over the training sequence $d_k(t),\ y_k(t)$ (these are the desired and predicted output functions of time respectively).

The change in weights ΔW and ΔV are calculated as

$$\Delta W(h) = \gamma_m W(h-1) + \gamma_g L_k f'(h) y_k \tag{3}$$
$$\Delta V(h) = \gamma_m V(h-1) + \gamma_g L_k g'(h) x_k \tag{4}$$

where $\gamma_m, \gamma_g \epsilon$ [0,1] are the constant parameters. The last step in the training process is the updation of net weights using Equations 3 & 4, which is given below: The training process aim is to update the net weights using Equations 5 and 6.

$$W(new) = W(old) + \Delta W \tag{5}$$
$$V(new) = V(old) + \Delta V \tag{6}$$

The aim of this weight updation is to minimize the error deviation between the desired output and actual output.

[1] In this paper the term unit, node and neuron are used interchangeably.

3.2. FMMRNN Training

This section gives a brief discussion about RNN training using back-propagation learning rule. We can interpret number of failures as a function of cumulative execution time x_i. Both cumulative execution time and number of failures are normalize in the range 0 to 1. Suppose f is the function of x_i, it can be written as $f(x_i) = d_i$. The normalized values of the input to the network such as $f(x_1), f(x_2)...f(x_i)$ are used to predict the $\hat{d}_{i+1}$. In other way, we can forecast $\hat{d}_{i+1}$ by using $\{x_1, d_1\}, \{x_2, d_2\}, ...\{x_i, d_i\}$, where d_{i+1} is target value is known as *short term prediction* or *1-step ahead prediction* or *next-step prediction*. In this study, we assume that the logistic function binary sigmoidal $F(x) = 1/(1 + e^{-\lambda x})$ is used for each neuron, where λ is the steepness parameter. The range of this transfer function varies from 0.0 to 1.0. The logistic transfer function is used to reduce the computational burden during training. The cross-validation process splits the entire representative data set into two sets: a) a training data set, used to train the network, b) a test data set used to validate the output of the model. We split the data set as follows: 80% for training and 20% for testing.

The training process is continued and the weights are updated until the last hidden layer state is reached. In the first epoch, the weights are typically initialized to a small random value. On next onwards a set of weights are chosen at random and the weights are adjusted in proportion to their contribution to error [27].

We employed MATLAB Version 7.10.0 environment for prediction purpose. The weights are initialized with small random value before first epoch starts. After then, the weights are adjusted randomly. The error tolerance for back-propagation algorithm is E_{min}=0.005. The network model FMMRNN is trained with initial weights and continues until the stopping criteria gets satisfied and the best weights are recorded for next-step-prediction of the reliability.

4. EXPERIMENTAL RESULTS AND OBSERVATIONS

In our reliability prediction experiment, we considered the failure data during system testing phase of distributed system application having defect severities 2 and 3 [28] as provided in Table 1. As per our experimental

Table 1. Defect severities level

Sl. no.	Severities level	Type	Description	Need of solution
1	0	No impact	Can tolerate	No need
2	1	Minor	Can tolerate	Solution eventually
3	2	Major	Can tolerate	Solution needed
4	3	Critical	Intolerable	Solution urgently needed

requirements, we have taken a) Failure Number, b) Time Between Failures (TBF) for our analysis. Below, we present, the list of some prediction parameters used in our approach.

- **The Average Error (*AE*)**, how adequately a model determines all over the system testing phase [29]. AE, measures how well a model predicts throughout the testing phase [30]. Average Error(%):$AE_i = (|(F_i - D_i)/D_i|) * 100$

- **The Root Mean Square Error (*RMSE*)**, is used to determine how far on average the error (i.e. between actual and target value) is from 0. The lower is RMSE, the higher is prediction accuracy. Mathematically, $RMSE = \left[\sqrt{\sum_1^n (F_i - D_i)^2}\right]/n$
  Normalized Mean Square Error (NRMSE)=$\left[\sqrt{\sum_1^n (F_i - D_i)^2}\right] / \sum_1^n {F_i}^2$

- **Mean Absolute Error (*MAE*)** is an average of an absolute error that computes how close predictions are to the final result. The MAE and RMSE are used together to analyze the variation in the errors on data set. $MAE = [\sum_1^n |(F_i - D_i)|]/n$

Here, F_i denotes the predicted output and D_i denotes the desire output of i^{th} node. In our experiment, we consider *data set*(DS) [31] for prediction and analysis purpose.

The number of hidden layers is one and the numbers of neurons present in the hidden layer calculated by optimization process are recorded from 10 to 50. Table 5 represent the result of the model during validation. After the proposed network model is successfully trained with 80% data, then the network undergoes for next-step predictions

Table 2. Some Data Sets Used For Reliability Prediction [28] & [31]

Project Code	Project Name	Number of Failures	Development Phases
DS1	Real Time Command & Control System	136	System Test Operations
DS2	Military	101	System Test
DS3	Commercial System	73	Subsystem Test
DS4	Real Time Command & Control System	54	System Test Operations
DS5	Real Time Command & Control System	53	System Test Operations
DS6	Military	41	System Test
DS7	Real Time Command & Control System	38	System Test Operations
DS8	Military	38	System Test
DS9	Real Time	36	System Test
DS10	Distributed System [31]	191	System Test

and validation using rest 20% test data. The MAE and RMSE are used together to analyze the variation in the errors on data set. The values of various parameters such as AE, RMSE, and NRMSE of our experiment are listed in Table 5. We observed that the network model produce best result at 45 numbers of neurons in the hidden layer. Fig. 3 and 4 summarize in terms of AE and 6 in terms of RMSE.

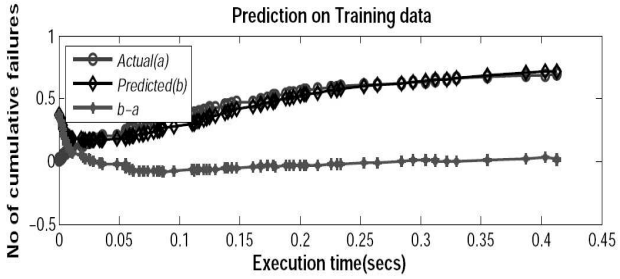

Figure 3. Next-step prediction on training data.

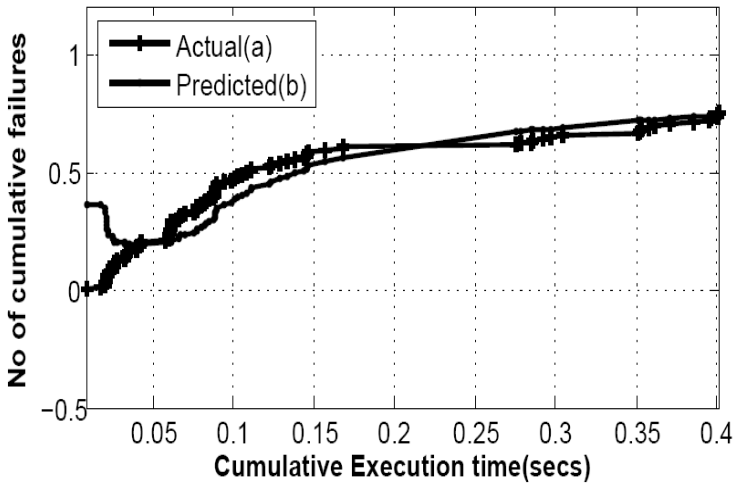

Figure 4. Next-step prediction on test data.

The best results found for data set for *short-term prediction (STP)* are as follows: the values of AE, RMSE, NRMSE, and MAE are 3.0019, 0.00438, 0.0261, and 0.0331 respectively. The STP for measurement unit AE on training is shown in Fig. 3 (i.e. the desired output and predicted output) and deviation between actual and forecasted value. Figure 4 show the prediction graph of actual data and predicted result for data set DS10. The corresponding deviation between the actual and computed output is shown in Figure 5. The figure shows how close predictions are to the predicted results.

The accuracy of AE, we found in this experiment has been greatly improved and is consistent than some well-known methods that are arrived at Table 6. The performance of the network during training is presented in Fig 6 in terms of RMSE during training. It is drawn in the form of number of epochs vs error rate in terms of RMSE during training. It shows how the error rate decreases with number epochs during training the network.

Table 3. Data set by Iyer and Lee [31] for DBS10

Failure No	C E Time	Failure No	C E Time	Failure No	C E Time	Failure No	C E Time
1	9.9898	49	472.18	97	1048.3	145	1661.8
2	18.747	50	483.2	98	1062	146	1669.4
3	28.962	51	494.25	99	1075.8	147	1677.5
4	40.719	52	505.39	100	1089.6	148	1686.3
5	52.872	53	516.55	101	1103.3	149	1695.5
6	61.037	54	527.7	102	1117.1	150	1705.1
7	70.447	55	539.81	103	1130.9	151	1715
8	80.03	56	551.94	104	1145.4	152	1727.8
9	88.819	57	563.97	105	1159.9	153	1740.6
10	100.3	58	576.01	106	1174.5	154	1753.5
11	110.31	59	588.08	107	1189	155	1766.3
12	117.3	60	600.39	108	1203.5	156	1779.1
13	124.36	61	612.71	109	1218.3	157	1792.4
14	130.85	62	625.03	110	1233.3	158	1806.6
15	137.48	63	637.37	111	1248.2	159	1820.8
16	143.67	64	650.49	112	1263.1	160	1835.1
17	149.64	65	664.14	113	1278	161	1847.8
18	154.47	66	677.97	114	1284.3	162	1861.5
19	164.37	67	691.79	115	1296.5	163	1875.7
20	177.25	68	705.63	116	1309.4	164	1890.3
21	183.9	69	719.47	117	1322.4	165	1904.9
22	191.83	70	733.31	118	1336.2	166	1916.9
23	200.02	71	747.19	119	1349.9	167	1930.2
24	208.79	72	761.09	120	1363.9	168	1943.5
25	218.06	73	775	121	1377.8	169	1957.9
26	227.6	74	788.92	122	1383	170	1972.3
27	237.5	75	802.83	123	1388.2	171	1986.7
28	247.47	76	816.77	124	1396.9	172	2001.3
29	257.56	77	830.72	125	1409.4	173	2015.8
30	267.7	78	844.69	126	1422.5	174	2025.7
31	280.18	79	858.68	127	1436.2	175	2038.1
32	292.96	80	872.67	128	1449.8	176	2050.9
33	305.79	81	879.07	129	1463.7	177	2062.3
34	319.6	82	885.46	130	1478.2	178	2075.6
35	328.15	83	889.07	131	1485.7	179	2088.9
36	336.82	84	902.73	132	1496.8	180	2102.8
37	345.49	85	916.75	133	1509.7	181	2113.5
38	354.17	86	930.94	134	1522.8	182	2124.3
39	362.81	87	945.24	135	1536.9	183	2135.6
40	369.61	88	959.53	136	1551.3	184	2147.4
41	379.51	89	973.83	137	1565.9	185	2160.1
42	391.11	90	988.13	138	1580.5	186	2172.8
43	403.37	91	993.81	139	1595.2	187	2186
44	417.38	92	1001.5	140	1609.8	188	2199.1
45	431.38	93	1009.5	141	1620.8	189	2212.3
46	445.39	94	1017.5	142	1628.6	190	2225.5
47	453.99	95	1025.9	143	1639.5	191	2238.7
48	462.8	96	1034.6	144	1650.7		

Table 4. Training Result of Data Set DS10

Neurons in each layer	AE	RMSE	NRMSE	MAE
1,8,1	4.6714	0.02321	0.0544	0.0423
1,23,1	4.4113	0.0178	0.0432	0.0328
1,30,1	3.6215	0.0021	0.0159	0.0331
1,38,1	3.7614	0.0113	0.017	0.0333
1,40,1	3.4311	0.0093	0.01501	0.0351
1,46,1	4.6715	0.00572	0.0261	0.0331
1,47,1	3.0017	0.00433	0.0262	0.0489
1,49,1	3.9912	0.01989	0.0493	0.0416

Table 5. Selection for number of neurons in hidden.

Neurons in each layer	AE	RMSE	NRMSE	MAE
1,8,1	4.6714	0.02321	0.0544	0.0423
1,10,1	4.8714	0.00798	0.0534	0.0423
1,23,1	4.4113	0.0178	0.0432	0.0328
1,25,1	4.5113	0.00547	0.0441	0.0328
1,38,1	3.7614	0.0113	0.0317	0.0333
1,45,1	3.0019	0.00438	0.0261	0.0331
1,49,1	3.9912	0.01989	0.0493	0.0416
1,50,1	5.6312	0.00993	0.0493	0.0446

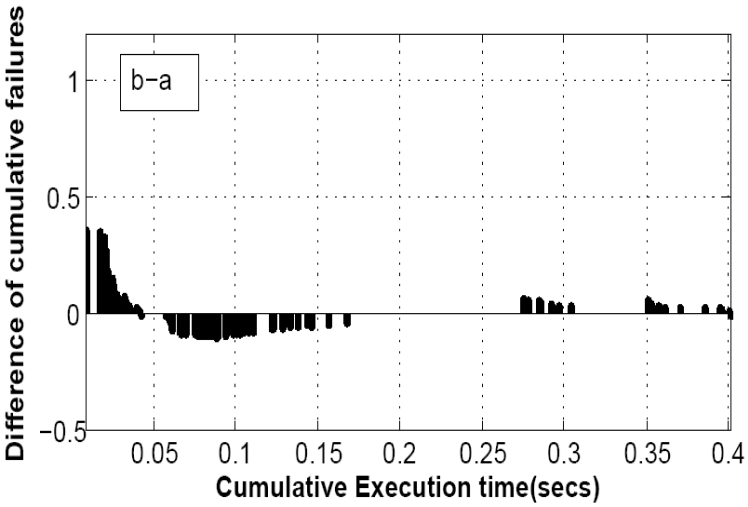

Figure 5. Eror Deviation of Next-step prediction on test data.

4.1. Observations

The comparative data are shown in Table 6 for the given data set with various models. It is also observed that the training results shows better accuracy than the prediction result.

The next-step prediction in Table 6 shows that our proposed model has less NRMSE value i.e 0.0261 and AE is 3.0019. Beside, the measurement criteria NRMSE is also found minimum than the various reliability prediction models [13, 14, 32] and RMSE value with [10, 33, 34].

Model quality is observed if its predictions are close to the ideal line passing through the zero error [8]. Fig. 3, 4 ans 5 show the prediction closeness between the actual value and prediction value.

Some observations on software reliability prediction using our proposed feed forward neural network model are listed below:

- The training results of the proposed model are better than the prediction result of the corresponding trained neural network. It means that producing good result at approximating does not certainly good at forecasting.
- Unlike statistical techniques, no unrealistic assumption is made in recurrent neural network approach.
- As we in the category of black-box model approach, so some useful information is ignored.

Table 6. A comparison with different model

Approach Model	Measure Parame-ter	Value
Mohanty et al. [32]	NRMSE	0.07292
FMMRNN(Proposed)	NRMSE	0.0261
Su et al. [21]	AE	3.24
FMMRNN(Proposed)	AE	3.0019

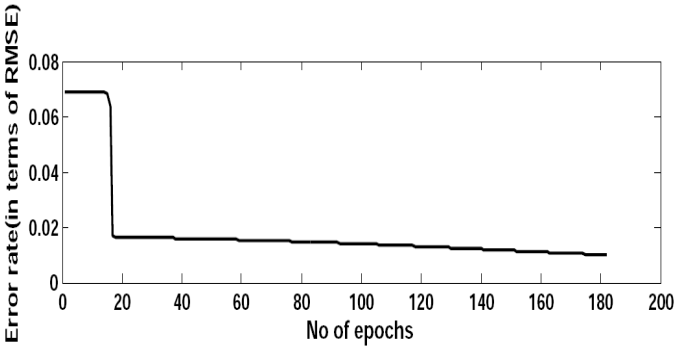

Figure 6. Performance result during training.

- Our model is a generic model that can work in any stabilize smooth trend data set and in any environment.

.

4.2. Threats to Validity

Below we discuss the possible threats to the validity of our work.

- Arbitrary data set partitioning for training and testing the network can be a limiting factor.
- As our experiment uses MATLAB for computation, so it suffers the same threats to validity as MATLAB does.
- As we discussed in Section 4., The weights of the neural network are chosen as random variables with specified distributions. So computed value may not produce the same result for every run. That is, even if for same input dataset and the same learning scheme are employed, expecting the same output is difficult.
- So far there is no such criteria on range for training and testing partitioning with respect to the performance validation.

The FMMRNN model shows that it yields a lower average relative prediction error and Normalized Mean Square Error compared to other model [13, 14, 32] approaches.

5. CONCLUSION

In this approach, we presented a novel technique for software reliability prediction using fuzzy min-max algorithm together with recurrent neural network technique. We presented experimental evidence showing that fuzzy max-min algorithm with recurrent network (using back propagation learning) is giving the accurate result comparable to other methods. Software reliability prediction is used to improve software process control and achieve high software reliability. This finding gives a good sign of prediction capabilities of the developed fuzzy-neural networks model for estimating the software reliability. More datasets and other types of computational intelligence and simulation tools need to use for further justify our findings.

REFERENCES

[1] IEEE, "Standard glossary of software engineering terminology," Standards Coordinating Committee of the IEEE Computer Society, 1991.
[2] P. J. Boland, "Challenges in Software Reliability and Testing," Department of Statistics National University of Ireland, Dublin Belfield - Dublin 4 Ireland, Technical report, 2002.
[3] K. Khatatneh and T. Mustafa, "Software Reliability Modeling using Soft Computing Technique," *European Journal of Scientific Research*, vol. 26, no. 1, pp. 154–160, 2009.
[4] J. D. Musa and K. Okumoto, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," in *ICSE*, T. A. Straeter, W. E. Howden, and J.-C. Rault, Eds., IEEE Computer Society. Orlando, Florida, NJ, USA: Proceedings of the 7th International Conference on Software Engineering, March 1984, pp. 230–238.
[5] M. K. Bhuyan, D. P. Mohapatra, and S. Sethi, "A Survey of Computational Intelligence Approaches for Software Reliability Prediction," *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 2, pp. 1–10, March 2014.

[6] L. Tian and A. Noore, "Software Reliability Prediction Using Recurrent Neural Network with Bayesian Regularization," *International Journal of Neural Systems*, vol. 14, no. 3, pp. 165–174, June 2004.
[7] ——, "On-line Prediction of Software Reliability using an Evolutionary Connectionist Model," *Science Direct, The Journal of Systems and Software*, vol. 77, no. 2, pp. 173–180, August 2005.
[8] N. Karunanithi and D. Whitley, "Prediction of Software Reliability Using Feed-forward and Recurrent Neural Nets," in *Neural Networks, 1992. IJCNN*, vol. 1. Baltimore, MD: IEEE, June 1992, pp. 800–805.
[9] T. M. Khoshgoftaar, A. S. Pandya, and H. More, "A Neural Network Approach For Predicting Software Development Faults." Research Triangle Park, NC: Proceedings of Third International Symposium on Software Reliability Engineering, October 1992, pp. 83–89.
[10] Y. Singh and P. Kumar, "Prediction of Software Reliability Using Feed Forward Neural Networks," in *Computational Intelligence and Software Engineering (CiSE)*, I. Conference, Ed. IEEE, 2010, pp. 1–5.
[11] M. M. T. Thwin and T. S. Quah, Eds., *Application of Neural Network for Predicting Software Development Faults using Object-Oriented Design Metrics*, vol. 5. Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02), November 2002.
[12] L. Zhao, J. pei Zhang, J. Yang, and Y. Chu, "Software reliability growth model based on fuzzy wavelet neural network," in *2nd International Conference on Future Computer and Communication (ICFCC)*, vol. 1. Wuhan: IEEE, May 2010, pp. 664–668.
[13] N. RajKiran and V. Ravi, "Software Reliability Prediction using Wavelet Neural Networks," in *International Conference on Computational Intelligence and Multimedia Applications*, vol. 1. Sivakasi, Tamil Nadu: IEEE, December 2007, pp. 195 – 199.
[14] ——, "Software Reliability Prediction by Soft Computing Techniques," *Journal of Systems and Software*, vol. 81, no. 4, pp. 576–583, April 2008.
[15] P.-F. Pai and W.-C. Hong, "Software reliability forecasting by support vector machines with simulated vector machines with simulated annealing algorithms," *Journal of Systems and Software, ELSEVIER*, vol. 79, no. 6, pp. 747–755, June 2006.
[16] K.-Y. Cai, *Software Defect and Operartional Profile Modeling*. MA, USA: Kluwer Academic Publishers Norwell, 1998, vol. 1.
[17] R. Sitte, "Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration," *Reliability, IEEE Transactions*, vol. 48, no. 3, pp. 285–291, September 1999.
[18] S. Mohanta, G. Vinod, and R. Mall, "A Technique For Early Prediction of Software Reliability Based on Design Metrics," *Springer, International Journal of System Assurance Engineering and Management*, vol. 2, no. 4, pp. 261–281, December 2011.
[19] W. Adnan, M. Yaakob, R. Anas, and M. Tamjis, "Artificial neural network for software reliability assessment," in *TENCON Proceedings of Intelligent Systems and Technologies for the New Millennium*, vol. 3, Fac. of Eng., Univ. Putra Malaysia, Selangor, Malaysia, 2000, p. 446451.
[20] K.-Y. Cai, L. Cai, Wei-Dong, Z.-Y. Yu, and D. Zhang, "On the neural network approach in software reliability modeling," *The Journal of Systems and Software, ELSEVIER*, vol. 1, no. 58, pp. 47–62, August 2001.
[21] Y.-S. Su and C.-Y. Huang, "Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models," *The Journal of Systems and Software, ELSEVIER*, vol. 80, no. 4, pp. 606–615, August 2007.
[22] J. H. Lo, "The Implementation of Artificial Neural Networks Applying to Software Reliability Modeling," *Control and Decision Conference, 2009. CCDC '09. Chinese*, pp. 4349 – 4354, June 2009.
[23] A. Joshi, N. Ramakrishman, E. N. Houstis, and J. R. Rice, "On neurobiological, neuro-fuzzy, machine learning, and statistical pattern recognition techniques," *IEEE TRANSACTIONS ON NEURAL NETWORKS*, vol. 8, no. 1, pp. 18–31, January 1997.
[24] D. R. Hush and B. G. Herne, "Progress in supervised neural networks," *Signal Processing Magazine, IEEE*, vol. 10, no. 1, pp. 8–39, January 1993.
[25] A. C. Tsoi and A. D. Back, "Locally recurrent globally feedforward networks: A critical review of architectures," vol. 5, no. 2, pp. 229–233, March 1994.
[26] Y. Maeda and M. Wakamura, "Simultaneous perturbation learning rule for recurrent neural networks and its fpga implementation," *IEEE TRANSACTIONS ON NEURAL NETWORKS*, vol. 16, no. 6, pp. 1664–1672, NOVEMBER 2005.
[27] N. Karunanithi, Y. Malaiya, and D. Whitley, "Prediction of Software Reliability Using Neural Networks," in *Proceedings IEEE International Symposium on Software Reliability Engineering*. Austin, TX: IEEE, May 1991, pp. 124–130.
[28] J. D. Musa, "Software Reliability Data," Data & Analysis Center for Software, January 1980.

[29] N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Prediction of Software Reliability Using Connectionist Models," *IEEE Trans. Software Eng.*, vol. 18, no. 7, pp. 563–574, July 1992.
[30] Y. K. Malaiya, N. Karunanithi, and P. Verma, "Predictability of software reliability models," *IEEE Transactions on Reliability*, vol. 41, no. 4, pp. 539–546, December 1992.
[31] R. Iyer and I. Lee, *Measurement-based analysis of software reliability, Handbook of Software Reliability Engineering*. McGraw-Hill, 1996, pp. 303 – 358.
[32] R. Mohanty, V. Ravi, and M. R. Patra, "Hybrid Intelligent Systems for Predicting Software Reliability," *Applied Soft Computing*, vol. 13, no. 1, pp. 189–200, August 2013.
[33] Y. Singh, A. Kaur, and R. Malhotra, "Empirical Validation of Object-Oriented Metrics for Predicting Fault Proneness Models," *Journal of Software Quality Control, Springer Science Business Media, LLC*, vol. 18, no. 1, pp. 3–35, July 2009.
[34] E. O. Costa, S. R., V. Aurora, and P. G. Souza, Eds., *Modeling Software Reliability Growth with Genetic Programming*. Chicago, Illinois: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, November 2005.

BIOGRAPHIES OF AUTHORS

Manmath Kumar Bhuyan is currently a Ph.D. candidate in the Department of Computer Science and Engineering at Utkal University, Vani Vihar, INDIA. M Tech in Computer Science and Engineering from NITTR, Kolkata. He worked with various MNC company as member in R&D group. He worked as an Asst prof in Computer science & engineering department for more than 10yrs.

Durga Prasad Mohapatra received the M E degree in computer science engineering in 2000 from REC and the Ph D degree in Computer Science Engineering in 2005 from Indian Institute of Technology, INDIA . He is an Associate Professor in the Department of Computer Science Engineering at National Institute of Technology. He has published more than 70 papers in the areas of software engineering, neural networks and genetic algorithms. He serve a members of Technical Societies IEEE, Institution of Engineers (I), CSI

Srinivas Sethi is an Associate Professor in the Department of Computer Science Engineering in Indira Gandhi Institute of engineering and Technology. He received the master degree in computer application (MCA) in 1995 from Berhampur University, INDIA, and the Ph D degree in Computer Science in 2011 from Berhampur University, INDIA, He is an Assistant Professor in the Department of Computer Science Engineering & Application at IGIT Sarang. He has published more than 30 papers in the areas of software engineering, networking, cloud computing, etc.