Recurrent neural network for software failure prediction

Mohamed Benaddy[1], Brahim El Habil[2], Othmane El Meslouhi[3], Salah-Ddine Krit[4]
Department of Mathematics Computer Sciences and Management
Laboratory of Engineering Sciences and Energy
Polydisciplinary Faculty of Ouarzazate, Ibn Zohr University
BO 638, Ouarzazate
Morocco
[1]: m.benaddy@uiz.ac.ma, [2]: b.elhabbil@uiz.ac.ma, [3]: o.elmeslouhi@uiz.ac.ma, [4]: s.krit@uiz.ac.ma

ABSTRACT

Software failure occurs when the software runs in an operational profile. Controlling failures in software require that one can predict problems early enough to take preventive action. The prediction of software failures is done by using the historical failures collected previously when they occur. To predict software failures, several models are proposed by researchers. In this paper, we present a recurrent neural network (RNN) to predict software failure using historical failure data. The proposed RNN is trained and tested using collected data from the literature; the obtained results are compared with other models and show that our proposed model gives very attractive prediction rates.

CCS CONCEPTS

• **Computer systems organization → Dependable and fault-tolerant systems and networks → Reliability**

KEYWORDS

Software Reliability, Software Failure Prediction, Neural Networks, Recurrent Neural Networks.

1 INTRODUCTION

Failure and defect detection at the test phase of software development process is an important issue, to correct their associated errors in the source code [33,34,35].

Due to the complexity of software, the fault and failure detection is hard task during the test phase. To fix this problem, software engineers use a number of tools to predict failures and defects in the software during the development process and to plan the testing and delivering processes. The software failure prediction uses data collected during software testing and or operational process. Several models for software failure prediction are developed by researchers. These models can be classified into parametric models and non-parametric ones [36,37,38,39,40]. The non-parametric models are more solicited than the parametric ones because of their undependability from parameters. Neural networks approach has been used to evaluate the software failure prediction and reliability, it has proven to be a universal approximates for any non-linear functions [14,5,17,28,30]. Karunanithi, et al. [23,15] were the first to propose using neural networks is software reliability prediction. Aljahdali, et. al., [27,26], Adnan, et. al., [32], Park, et. al., [24] and Liang, et. al., [29,31] have also made contributions to software reliability predictions using neural networks, and have gained better results compared to the traditional analytical models with respect to predictive performance. Most of the cited models are feed-forward neural networks based models. Alternatively, recurrent neural networks are used for software reliability modeling and for failure prediction, these models proved its robustness in software failure and reliability modeling. In this paper we present a software failure prediction using recurrent neural networks, the proposed approach is experienced using datasets from the literature and compared with other models. The rest of this paper is organized as follow; the section 2 is devoted for presenting the related works in the field of software reliability and failure prediction. Section 3 gives more details about our developed model using recurrent neural networks. Section 4 gives the obtained results of the proposed model. The final section is devoted to conclusion of this paper.

2 RELATED WORKS

Because of most recent development in the power of hardware, recently neural networks receives more attention by researchers in different field as pattern recognition [1,16] prediction and forecasting [6, 2] and so on. We have proposed a hybrid models based on neural networks trained by genetic algorithms [20,19] and simulated annealing [18], the obtained results are encouraged

ICEMIS'18, June 19–21, 2018, Istanbul, Turkey.
Copyright 2018 ACM ISBN 978-1-4503-6392-1/18/04…$15.00.
DOI: https://doi.org/10.1145/3234698.3234714

compared with other models such as regression models. M. K. Bhuyan et al. [22] proposed a detailed feed-forward back propagation network model for predicting reliability using failure data, the obtained results show a good of fit compared with other models. Jinyong Wang et al. [13] proposed a deep learning model based on the recurrent neural network (RNN) encoder–decoder for software reliability prediction, the architecture of the proposed RNN-encoder-decoder model uses one input layer, two hidden layers, and one output layer. As shown in the paper the obtained results are the best compared with four neural networks (NN) models using 14 fault datasets in terms of end-point predictions and next-step predictions. M. K. Bhuyan et al. [21] used a hybrid technique for software reliability prediction using fuzzy min-max algorithm with recurrent neural network technique. The authors showed that the proposed approach, gives the accurate result comparable to other methods [21]. Q.P. Hu et al. [25] proposed an efficient Elman combined recurrent network and feed-forward network for fault detection and correction processes as shown in their paper compared with other classical analytical models.

3 Recurrent neural networks for software failure prediction architecture

As mentioned in the previous section both feed-forward and recurrent neural networks are applied successfully in software failure prediction. The recurrent architectures has its advantage to share information over the time, from one time step to the next one. This kind of neural networks is suitable for predicting cumulative failure as the present failure is summed with the past one. In this paper we adopted a simple recurrent neural network which consist of three layers (i.e. one input layer, one hidden layer and one output layer) as shown in figure **Error! Reference source not found.** [8].

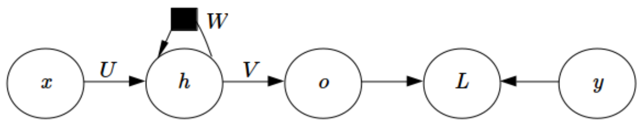

Figure 1: Proposed RNN architecture

At each time step t, the input vector x_t is the actual four failures, the hidden layer activations are $h^{(t)}$, the outputs are $o^{(t)}$, the targets are $y^{(t)}$ and the loss is $L^{(t)}$. As shown in the figure **Error! Reference source not found.** the hidden layer has a connection from the past time step *t-1*. The *U* matrix is the weights of the input layer to the hidden layer. The *W* matrix is the weights of the hidden layer to itself, and the *V* matrix represents the weights for the hidden layer to the output layer. The loss *L* is computed in function of the output $\hat{y}^{t)}$ and target y values according to the equation 5. At time step t we compute all the values according to the equations 1, 2, 3 and 4.

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \tag{1}$$

$$h^{(t)} = h^{(t-1)} + log\big(exp\big(a^{(t)}\big) + 1\big) \tag{2}$$

$$o^{(t)} = c + Vh^{(t)} \tag{3}$$

$$\hat{y}^{(t)} = \frac{1}{\left|o^{(t)}\right|}\sum_{i}\left(o_i^{(t)}\right) \tag{4}$$

$$L^{(t)} = \left[\hat{y}^{(t)} - y^{(t)}\right]^2 \tag{5}$$

where b and c are the biases of the input to hidden layer and the hidden to the output layer.

4 Prediction procedure

The prediction process is divided into two stages; the training stage and the testing stage. At the training stage the network is trained with 70% of available collected data. The testing stage is performed after the training by using 30% of the collected data. The process is repeated a number of training epochs. At each epoch step the network is retrained for new prediction and testing for each data inputs within the project. The prediction procedure is explained in the following subsections.

4.1 Software failure Datasets

Datasets for software reliability are collected and compiled by John Musa of Bell Telephone Laboratories [12]. The data sets consist of failures interval time, these data are used by software managers in monitoring test status and predicting schedules and to assist software researchers for developing and validating software reliability models. The data sets consist of software failure data of 16 projects. Careful controls were employed during data collection to ensure that the data would be of high quality. It represents projects from a variety of applications including real-time command and control, word processing, commercial, and military applications. To compare our obtained results with other proposed models we use data from three different projects. They are Military (System Code: 40), Real Time Command & Control (System Code: 1) and Operating System (System Code: SS1C). The data contain the day of occurrence of the failure, time between two successive failures and the number of failure.

4.2 Data Processing

Before predicting future software failures using the collected failures $x=\{x_1, x_2, ..., x_n\}$, with x_i is the discovered number of failure on day i. The number of failure are summed to obtain the cumulative failures $x=\{x_1, x_1 + x_2, x_1 + x_2 + x_3, ..., x_1 + x_2 + x_3 + ... + x_{n-1} + x_n\}$ then these data are normalized. To normalize data thirty one techniques are proposed by Jahan, A., Edwards [12] according to benefit criteria or cost criteria conditions of use. In this paper we adopted a linear normalization sum-based method [12] using the equation 6.

$$x_i^{(norm)} = \frac{x_i}{o^{(t)}}\sum_{i}^{n} x_i \tag{6}$$

The normalization procedure does not alter the failures behaviour but simply puts the input values in a range more suitable for the standard activation functions [9]. For notation simplification we assume that the failures $x=\{x_1, x_2, ..., x_n\}$ are already normalized.

4.3 Training the RNN

The normalized data $x=\{x_1, x_2, ..., x_n\}$, are used to train the RNN to predict the future cumulative failures. To do so the data sequence should be grouped into vectors of length w (sliding window) that can be denoted as $x=\{x_{k-w}, x_{k-w+1}, ..., x_{k-1}\}$, $k = w, w+1, ..., n$ for any k the vectors contain w inputs. The target data $y=\{y_1, y_2, ..., y_n\}$ contains the same number of data in input data. The two training data sets map the relationships of the network. The training data sets x and y are used to train the network for adjusting its weights W, U and V and biases b and c. The weights and biases are randomly initialized within the interval [-0.1,0.1] and by ones successively, the weights can be initialized using Gaussian distribution with mean 0 and deviation 0.1, but there is no directions which initialization method to use for initializing weights and biases [9]. With this random initialization several experimentations should be realized to obtain the correct initial values. Designing improved initialization strategies is a difficult task because neural network optimization is not yet well understood [8]. The backpropagation optimization algorithm is the most and simplest classical method used to train neural networks (feed-forward and recurrent) [9]. But there exist several modern methods which are developed and tested with a variety of networks architectures, such as AdaGrad algorithm [4], RMSProp [7,8], Adam [3] known as Algorithms with Adaptive Learning Rates [8]. In this paper we tested the three methods, as the result the Adam method [3] gives more accuracy compared to other methods. These algorithms called optimization algorithms try to find the accurate network parameters (weights and biases) to fit the outputs $\hat{y}$ to targets y, by minimizing the deviation between the network outputs $\hat{y}$ and targets y called the loss function indicated in equation 5. The training process is repeated a number of epochs for each project data set, until it stabilizes its parameters.

4.4 Testing the RNN

After the training phase, the network with the optimized parameters is used for testing, by presenting the rest of 30% of data to the trained network. In this stage the MSE (equation 7) is computed and stored, that is we can compare the obtained results obtained in different executions.

4.5 Remark

At the end of the training and testing phases, if a negative value is obtained it is substituted with the value 0.

4.6 Evaluation criterion

In the training and testing stages, the network is evaluated at each epoch by computing the mean of the sum of squared (MSE) errors for each project according to the equation 7.

Where n is the number of failures used during the training process. $\hat{y}_i$ and y_i are the predicted output and the actual number of failures during the same process.

5 Experimental results

5.1 Training phase

The RNN weights are initialized randomly within the interval [-0.1,0.1] and the biases by ones. The Adam optimization algorithm's parameters are as follows: *learning_rate = 0.003*, *beta1 = 0.9*, *beta2 = 0.999*, *epsilon* $= 10^{-08}$. The number of epochs is set to 40, the sliding window $w = 4$. *70%* of the data are used to train the RNN for each project in the data set. The proposed model is implemented using Pyhton3.5 [10] and Tensorflow [11]. To be more accurate and comparable with the results obtained in [19] and [18] the MSE is computed according to the equation 8, because of the normalization process. The obtained results are summarized in table 1, compared with feedforward neural network trained by real-coded genetic algorithm (RCGA) [19] and simulated annealing (SA) [18]. The obtained results show that our proposed RNN is more efficient than a feedforward one trained by the RCGA and SA, another advantage of the proposed RNN is the number of training epochs which is very small compared with the other methods.

$$MSE_{new} = MSE \sum_{i}^{n} x_i \tag{7}$$

Table 1: Obtained results for the three projects during the training phase compared with other models

Project Name	Military	Real Time Control	Operating System
# failures	101	136	277
Training ratio		70 %	
# epochs		40	
Proposed Recurrent Neural Network trained with Adam			
MSE	0.012424	0.2207779	0,0169715
Neural Network trained with RCGA [19]			
MSE	2.859155	2.0515463	2.0515463
Neural Network trained with SA [18]			
MSE	1.7323943	2.4329896	1.9329897

5.2 Testing phase

For testing phase 30% of the dataset is used for each project. Table 2 summarize the obtained MSE for each project compared with these obtained by feedforward neural network trained with RCGA [19] and SA [18].

Table 2: Obtained results for the three projects during the testing phase compared with other models

Project Name	**Military**	**Real Time Control**	**Operating System**
# failures	101	136	277
Testing ratio		30 %	
	Proposed Recurrent Neural Network trained with Adam		
MSE	0.010273	0.0007407	5.8794258E-05
	Neural Network trained with RCGA [19]		
MSE	4.2277226	2.6617646	3.0758123
	Neural Network trained with SA [18]		
MSE	2.8415842	2.4044118	2.4187725

5.2 Plotting the obtained results

In figures 2, 3, 4 and 5 are plotted the obtained results and its related error differences during training and testing phases for Military Application. In figures 6, 7, 8 and 9 we are showing the obtained results and their related errors differences in the training and testing phases for Real Time and Control Application. The figures 10, 11, 12 and 13 are represented the obtained cumulative failures and the error differences during training and testing phases, using our proposed RNN approach.

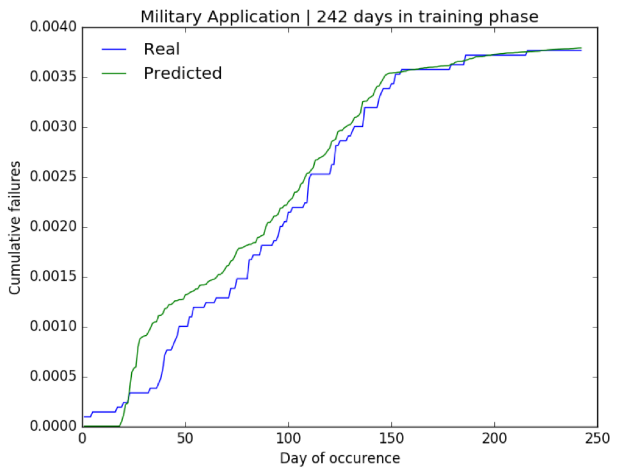

Figure 2: Real and predicted cumulative failures for Military application data during the training phase.

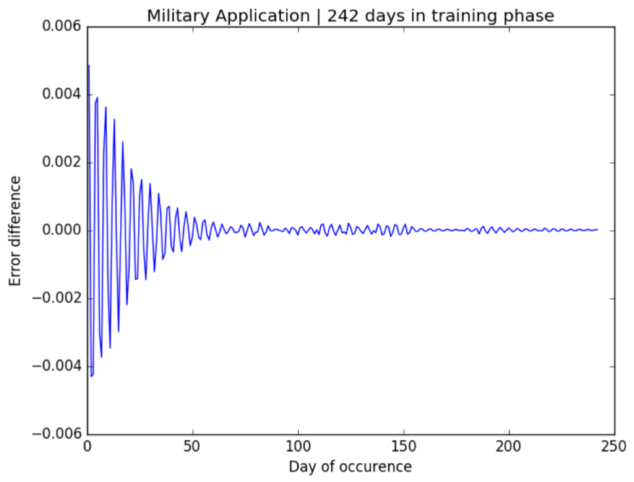

Figure 3: Relative prediction error for Military application during the training phase.

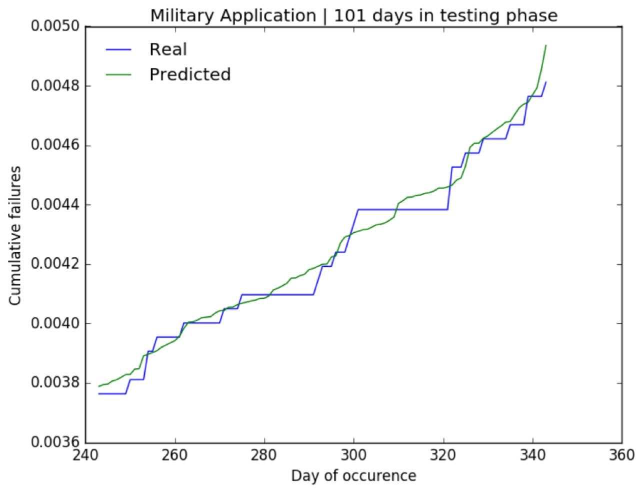

Figure 4: Real and predicted cumulative failures for Military application data during the training phase.

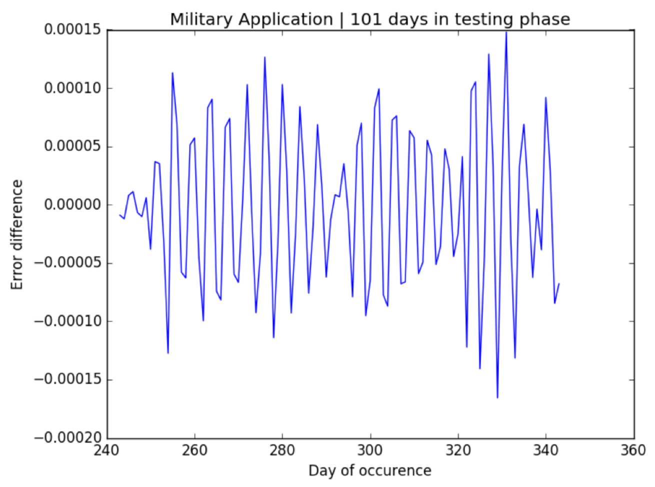

Figure 5: Relative prediction error for Military Application during the testing phase.

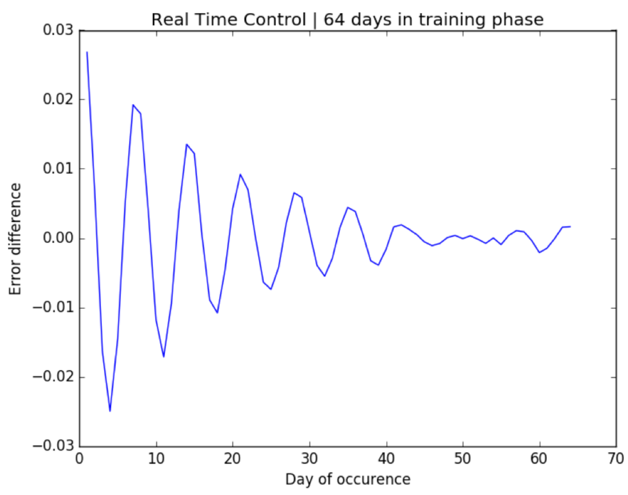

Figure 7: Relative prediction error Real Time and Control application during the training phase.

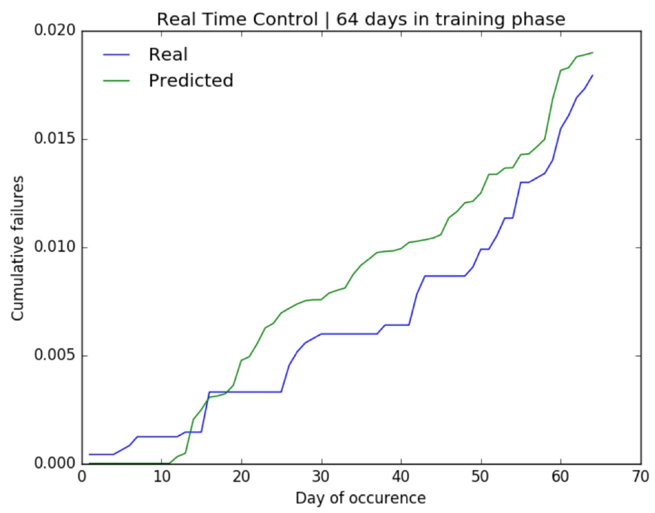

Figure 6: Real and predicted cumulative failures for Real Time and Control application data during the training phase.

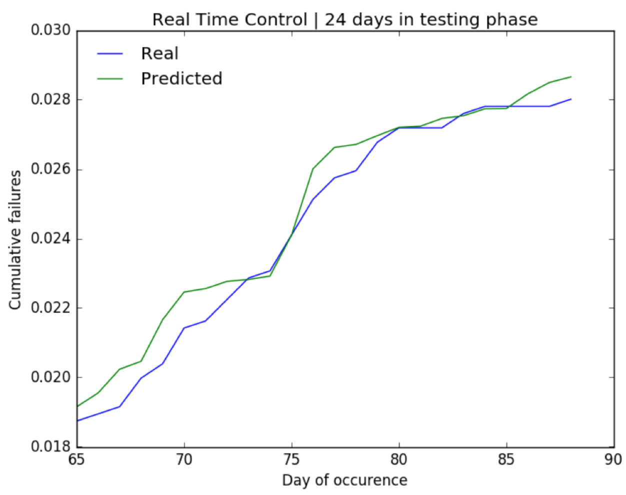

Figure 8: Real and predicted cumulative failures for Real Time and Control application data during the testing phase.

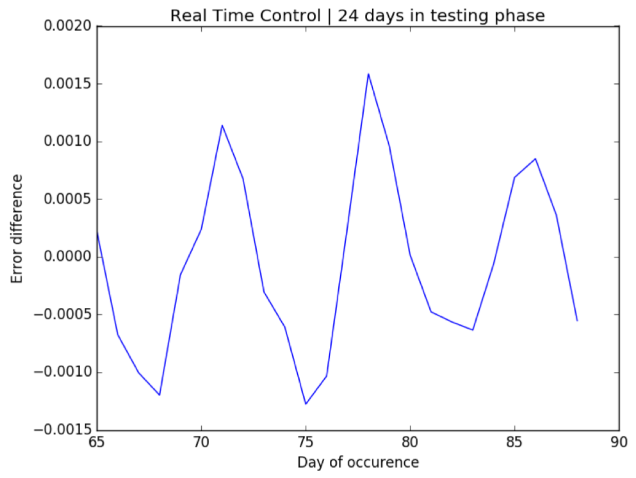

Figure 9: Relative prediction error for Real Time and Control application during the testing phase.

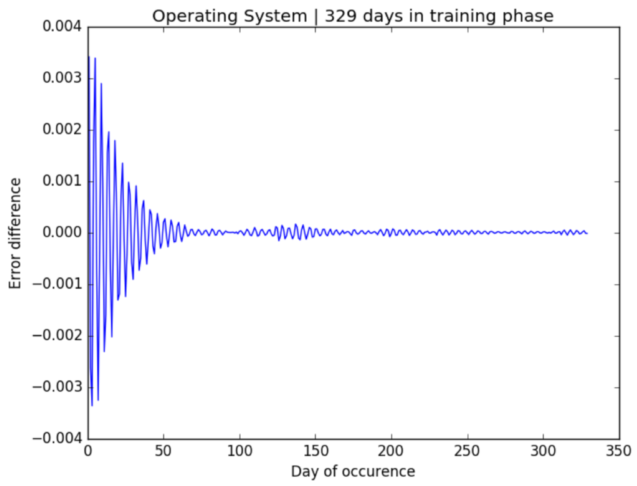

Figure 11: Relative prediction error for Operating System application during the training phase.

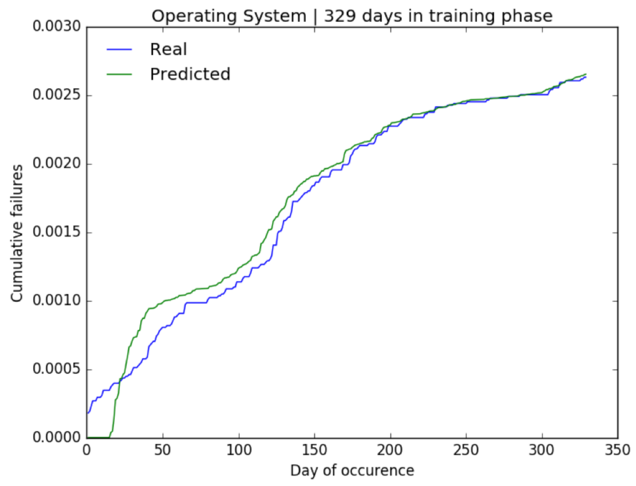

Figure 10: Real and predicted cumulative failures for Operating System application data during the training phase.

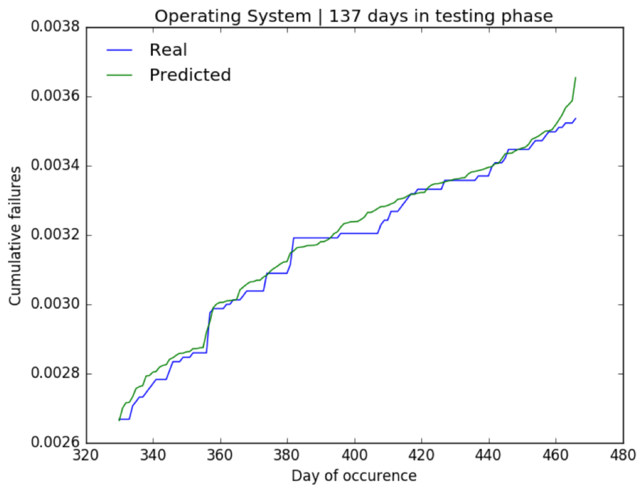

Figure 12: Real and predicted cumulative failures for Operating System application data during the testing phase.

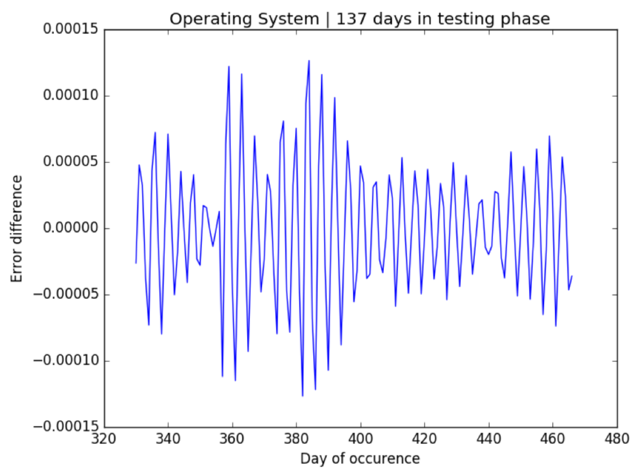

Figure 13: Relative prediction error for Operating System application data during the testing phase.

4 CONCLUSION

In this paper, a recurrent neural network model for software cumulative failure prediction is proposed. To optimize the proposed architecture Adam optimization algorithm is adopted. 70% and 30% of the data set from three projects are used to train and test the model successively. Experimental results show that our proposed model adapts well across the three different projects, compared with the results obtained with a feedforward neural network trained with a real-coded genetic algorithm and simulated annealing. As a future work our aims is to use the entire data sets projects and compare its results with other neural networks architectures from the literature.

REFERENCES

[1] S. Fernandez Bertolami H. Bunke A. Graves, M. Liwicki and J. Schmidhuber. May 2008. A Novel Connectionist System for Unconstrained Handwriting Recognition. IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, pp. 855868 (May 2008).

[2] Fatima El Jamiy James Higgins Brandon Wild Abd El Rahman El Said, Travis Desell. 2017. Optimizing Long Short-Term Memory Recurrent Neural Networks Using Ant Colony Optimization to Predict Turbine Engine Vibration. CoRR, vol. Abs/1710.03753. (2017).

[3] Jimmy Lei Ba Diederik P. Kingma. 2015. Adam: A method for stochastic optimization. ArXiv preprint arXiv:1412.6980. Published as a conference paper at ICLR (2015).

[4] Hazan E. Duchi, J. and Y. Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research. (2011).

[5] S. H. Ling F. H. F. Leung, H. K. Lam and P. K. S. Tam. 2003. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. IEEE Transactions on Neural Networks vol. 14, no. 1, pp. 7988. (2003).

[6] N Srivastava Geoffrey Hinton and Kevin Swersky. 2012. Neural Networks for Machine Learning, Lecture 6a overview of minibatch gradient descent. Coursera Lecture slides https://class.coursera.org/neuralnets-2012-001/lecture (Online) (2012).

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org.

[8] A. Graves. 2012. Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence. Springer. 374 , 395 , 411 , 460 (2012).

[9] A. Graves. 2012. Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence. Springer. 374 , 395 , 411 , 460 (2012).

[10] https://www.python.org/downloads/. accessed on January 27th 2018.(accessed on January 27th 2018.).

[11] https://www.tensorflow.org/. accessed on January 27th 2018.. (accessed on January 27th 2018.).

[12] Edwards K.L. Jahan, A. 2014 A state-of-the-art survey on the influence of normalization techniques in ranking: Improving the materials selection process in engineering design. Materials and Design, doi: http://dx.doi.org/10.1016/j.matdes.2014.09.022. (2014).

[13] Ce Zhang Jinyong Wang. 2017. Software Reliability Prediction Using a Deep Learning Model based on the RNN Encoder-Decoder. Reliability Engineering and System Safety, doi: 10.1016/j.ress.2017.10.019. (2017).

[14] W. D. Wang Z. Y. Yu K. Y. Cai, L. Cai and D. Zhang. 2001. On the neural network approach in software reliability modeling. J. Syst. Softw., vol. 58, pp. 4762 (2001).

[15] Whitley D. Karunanithi, N. and Y.K. Malaiya. 1992. Prediction of software reliability using connectionist models. IEEE Trans. Soft. Eng. 18, 563-574 (1992).

[16] Mamalet F. Garcia C. Lefebvre G., Berlemont S. 2015. Inertial Gesture Recognition with BLSTM-RNN. In: Koprinkova-Hristova P., Mladenov V., Kasabov N. (eds) Artificial Neural Networks. Springer Series in Bio-/Neuroinformatics, vol 4. Springer, Cham. (2015).

[17] M. R. Lyu. 1996. Handbook of Software Reliability Engineering. IEEE Computer Society Press and McGraw-Hill Book Company.

[18] M. Wakrim M. Benaddy. 2012. Simulated annealing neural network for software failure prediction. International Journal of Software Engineering and Its Applications 6 (4), 35-46 (2012).

[19] M. Wakrim M. Benaddy, S. Aljahdali. 2011. Evolutionary prediction for cumulative failure modeling: A comparative study. In Information Technology: New Generations (ITNG2011) Eighth International Conference on pp. 41-47.

[20] Sultan Aljahdali M. Benaddy, M. Wakrim. 2009. Evolutionary neural network prediction for cumulative failure modeling. In Computer Systems and Applications. AICCSA 2009. IEEE/ACS International Conference on, pp. 179-184.

[21] Durga Prasad Mohapatra Manmath Kumar Bhuyan and Srinivas Sethi. 2016. Software Reliability Prediction using Fuzzy Min-Max Algorithm and Recurrent Neural Network Approach. International Journal of Electrical and Computer Engineering (IJECE), Vol. 6, No. 4, pp. 1929 1938. (2016).

[22] Srinivas SETHI Manmath Kumar BHUYAN, Durga Prasad MOHAPATRA. 2016. Software Reliability Assessment using Neural Networks of Computational Intelligence Based on Software Failure Data. Baltic J. Modern Computing, Vol. 4 , No. 4, pp. 10161037. (2016).

[23] D. Whitley N. Karunanithi and Y.K. Malaiya. 1992. Using Neural Networks in Reliability Prediction. IEEE Software, vol. 9, no. 4, pp. 53-59 (1992).

[24] J. H. Park J. Y. Park and S. U. Lee. 1999. Neural network modeling for software reliability prediction from failure time data. Journal of Electrical Engineering and information Science, vol. 4, no. 4, pp. 533538. (1999).

[25] S.H. Ng G. Levitin Q.P. Hu, M. Xie. 2007. Robust recurrent neural network modeling for software fault detection and correction prediction. Journal of Reliability Engineering and System Safety 92, pp. 332340 (2007).

[26] A. Sheta S. Aljahdali and D. Rine. 2001. Prediction of Software Reliability: A Comparison between regression and neural network non-parametric Models. In Proceeding of the IEEE/ACS Conference, 25-29.

[27] K.A. Buragga S. Aljahdali. 2007. Evolutionary Neural Network Prediction for Software Reliability Modeling. The 16th International Conference on Software Engineering and Data Engineering, SEDE-2007 (2007).

[28] L. Tian and A. Noore. 2005. Evolutionary neural network modeling for software cumulative failure time prediction. Reliability Engineering System Safety, vol. 87, no. 1, pp. 45 51. (2005).

[29] L. Tian and A. Noore. 2005. Evolutionary neural network modeling for software cumulative failure time prediction. Reliability Engineering &amp; System Safety, vol. 87, no. 1, pp. 45 51. (2005).

[30] L. Tian and A. Noore. 2005. On-line prediction of software reliability using an evolutionary connectionist model. Journal of Systems and Software, vol. 77, no. 2, pp. 173 180 (2005).

[31] L. Tian and A. Noore. 2005. On-line prediction of software reliability using an evolutionary connectionist model. Journal of Systems and Software, vol. 77, no. 2, pp. 173 180 (2005).

[32] M.H. Yaacob W.A. Adnan. 1994. An integrated neural-fuzzy system of software reliability prediction. In Proceeding of the First International Conference on software Testing, Reliability and Quality Assurance, New Delhi, India.

[33] Shadi A. Aljawarneh, Muneer Bani Yassein, and We'am Adel Talafha. 2018. A multithreaded programming approach for multimedia big data: encryption system. Multimedia Tools Appl. 77, 9 (May 2018), 10997-11016. DOI: https://doi.org/10.1007/s11042-017-4873-9

[34] Vangipuram Radhakrishna, Shadi A. Aljawarneh, Puligadda Veereswara Kumar, and Kim-Kwang Raymond Choo. 2018. A novel fuzzy gaussian-based dissimilarity measure for discovering similarity temporal association patterns. Soft Comput. 22, 6 (March 2018), 1903-1919. DOI: https://doi.org/10.1007/s00500-016-2445-y

[35] Shadi A. Aljawarneh, Muneer Bani Yassein, and We'am Adel Talafha. 2017. A resource-efficient encryption algorithm for multimedia big data. Multimedia Tools Appl. 76, 21 (November 2017), 22703-22724. DOI: https://doi.org/10.1007/s11042-016-4333-y

[36] Shadi A. Aljawarneh, Ali Alawneh, and Reem Jaradat. 2017. Cloud security engineering. Future Gener. Comput. Syst. 74, C (September 2017), 385-392. DOI: https://doi.org/10.1016/j.future.2016.10.005

[37] Shadi A. Aljawarneh, Radhakrishna Vangipuram, Veereswara Kumar Puligadda, and Janaki Vinjamuri. 2017. G-SPAMINE. Future Gener. Comput. Syst. 74, C (September 2017), 430-443. DOI: https://doi.org/10.1016/j.future.2017.01.013

[38] Muneer Bani Yassein, Shadi A. Aljawarneh, and Esraa Masadeh. 2017. A new elastic trickle timer algorithm for Internet of Things. J. Netw. Comput. Appl. 89, C (July 2017), 38-47. DOI: https://doi.org/10.1016/j.jnca.2017.01.024

[39] Shadi A. Aljawarneh, Mohammed R. Elkobaisi, and Abdelsalam M. Maatuk. 2017. A new agent approach for recognizing research trends in wearable systems. Comput. Electr. Eng. 61, C (July 2017), 275-286. DOI: https://doi.org/10.1016/j.compeleceng.2016.12.003

[40] Muneer O. Bani Yassein and Shadi A. Aljawarneh. 2016. A Conceptual Security Framework for Cloud Computing Issues. Int. J. Intell. Inf. Technol.12, 2 (April 2016), 12-24. DOI=http://dx.doi.org/10.4018/IJIIT.2016040102