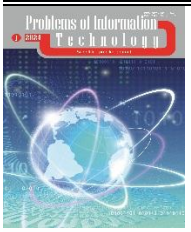

Available online at www.jpit.az

15 (1)
2024

Predicting the reliability of software systems using recurrent neural networks: LSTM model

Tamilla Bayramova

Institute of Information Technology, B. Vahabzade str., 9A, AZ1141 Baku, Azerbaijan

toma_b66@mail.ru

https://orcid.org/0000-0002-8377-3572

ARTICLE INFO

Keywords:

Software Reliability Growth Models
Recurrent neural network
LSTM
Deep learning
Parametric model
Non-parametric model

ABSTRACT

The dynamics and complexity of processes occurring in complex software systems, as well as the emergence of new types of malicious threats, further complicate the issues of ensuring software reliability. Despite the development of hundreds of models for increasing the reliability of software systems, this issue still remains relevant. Research shows that the use of neural networks in predicting the reliability of software systems allows one to obtain more accurate results. In this paper, to predict reliability, we used a neural network model with long short-term memory, which is a type of recurrent neural networks. Seven real-world software crash datasets were used to test the model's performance. The experiments were carried out in Python. Both parametric and nonparametric models were taken for comparison. The experimental results showed the practical significance of using the proposed model in predicting the reliability of software systems.

1. Introduction

Advances in science and technology have created ample opportunities for the development of high-performance equipment and high-quality software systems. However, in contrast to the rapid development of hardware technologies, the development of software technologies lags behind in all respects (quality, reliability, performance, security, etc.). As a result of errors in software systems, accidents can occur, ranging from minor inconveniences to large economic losses and human lives (Lai & Garg, 2020).

Software systems have become an integral part of modern society. With the increase in the number of functions assigned to these systems, the number of errors and failures in these systems increases. All components of software systems must be monitored before they are presented to the user. Prediction and reliability assessment have become one of the important problems that need to be solved when developing software systems. It determines the end time of testing a software system, thereby saving time and budget. Reliability in safety-critical systems (medical equipment, nuclear power plants, defense and aviation industries, etc.) requires more accurate assessment.

It is not possible to directly assess the reliability of software, and since the requirements for reliability in critical applications are very high, the volume of work performed in this area increases several times. To ensure the reliability of software systems, it is important to develop and implement effective methods and models that warn about and prevent errors, while at the same time allowing the program to continue functioning when these errors occur.

Models for improving the reliability of software systems are divided into two groups: parametric and non-parametric models (Fig. 1) (Sahu et al., 2021). Parametric models are built based on certain considerations and assumptions. Traditional parametric models such as non-homogeneous Poisson process (NHPP) models have been successfully used in software reliability engineering.

Received 27 September 2023, Received in revised form 30 November 2023, Accepted 14 December 2023
http://doi.org/10.25045/jpit.v15.i1.07

However, no parametric model can accurately predict all cases. The importance of predicting the reliability of software systems has turned the development of more accurate and reliable methods and models into one of the urgent tasks.

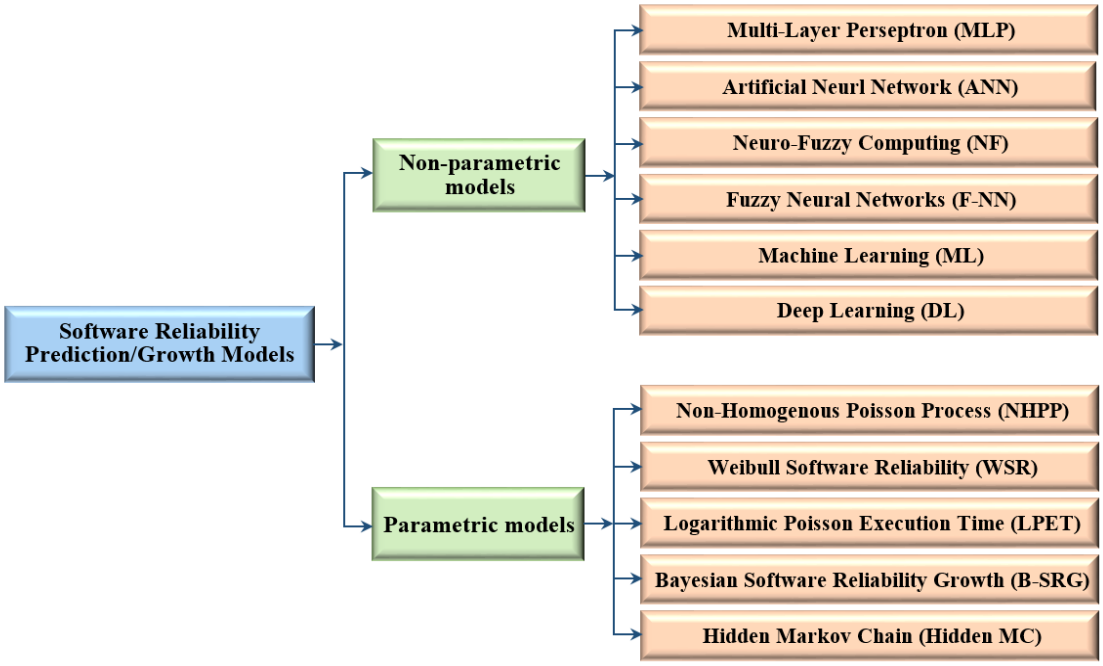

Fig. 1. Software reliability growth models

In recent years, to overcome the difficulties that arise in real problems in the field of modeling and predicting the reliability of FT, methods of machine learning and soft computing have been used. Modern and more promising methods for predicting the reliability of software systems, especially the intensity and duration of failures, are based on nonparametric models. Such models eliminate the main disadvantages of parametric models, since no assumptions or other restrictions are made about the nature of errors. These methods are based on the time history of failures based on historical data presented as time series. In recent years, various types of neural networks have been used for time series forecasting: multilayer perceptron (MLP), convolutional neural networks (CNN), recurrent neural networks (RNN), and other deep learning (DL) models. The neural network model requires only software failure history as input and predicts the timing of future failures more accurately than parametric models (Lakshman & Ramasamy, 2017; Roy et al., 2014; Musa, 2004; Bayramova, 2023).

Karunaithi et al. (1991) first used a neural network to evaluate software reliability. To evaluate reliability, they used bug detection history as input and cumulative number of bugs detected as output. In their research, they used feedforward neural network, Jordan network and Elman network. Comparing the results with several statistical models, they showed that they obtained better results. Khoshgoftaar and Lenning (1995) used a neural network as a tool to predict the number of errors in programs. They presented a new approach to reliability modeling and experimentally proved the superiority of neural network models.

In this study, a long short-term memory neural network model, which is a special type of recurrent neural networks, is used to predict the reliability of a software system. Scientific novelty of the article: to select the hyperparameters of the article, the AutoML hyperparameter optimization method was used. This model gives good results for both large and small data sets. Data from real software projects was used to train and test the model's performance, and a series of experiments were conducted to compare it with other competing approaches.

2. Releated work

The increasing demand for the quality of software systems requires the development of accurate modeling methods for predicting the reliability of software systems. Software reliability models are very useful for estimating the probability of failures in software systems.

Cai et al. (2001) proposed a method for predicting software reliability based on neural networks. For training, they used a

backpropagation algorithm. Based on the last 50 failures, the study predicted the time of the next failure. Lu and Ma (2018) assessed software reliability using a modified Whale algorithm. The article proposes a three-stage model. The experimental results showed that the accuracy of the three-stage model in predicting software errors is higher than that of the one-stage model.

Tian and Noor (2005) combined a neural network with an evolutionary algorithm to evaluate reliability. They applied a genetic algorithm to optimize the number of input and hidden layers. Granitto et al. (2005) showed through experiments that it is possible to improve the performance of neural network models by combining multiple neural networks.

Hu et al. (2006) proposed an artificial neural network model to predict the reliability of current projects based on failures in past projects. Aljahdali and Buragga (2008) combined multilayer perceptron, radial basis function, Elman recurrent neural network, and fuzzy neural network models to predict software reliability.

Jheng (2009) used an ensemble of neural networks to predict the reliability of software systems. The experiment was carried out on two databases and the result was compared with a neural network model and several statistical models. Experiments have shown that an ensemble of neural networks has better performance. Singh and Kumar (2010) used a feedforward neural network model to predict the reliability of a software system.

Ramasamy et al. (2016) proposed a neural network-based dynamic weighted combinatorial model (DWCM) for software reliability prediction. In this model, they took three different parametric models as the activation function in the hidden layer. They applied the result to two data sets and compared it with the results of those statistical models separately. Experiments have shown that the neural network model gives better results.

Munir et al. (2021) used a combination of GRU-LSTM to predict defects. They did not choose historical testing data as data for the forecast, but identified 32 program code operator level metrics. Performance tests cover a variety of programs and sets of variants written by thousands of programmers. The recall, accuracy, precision, and F1 measurement of the GRU-LSTM-based model increased by 1%, 4%, 5%, and 2%, respectively.

In (Yangzhen et al., 2017), a software reliability prediction model based on the long short-term memory network was proposed. To improve performance, the authors added layer normalization to the model. The results were compared with other neural network models, and the proposed approach showed better predictive performance.

3. Recurrent neural networks

Machine learning is a type of artificial intelligence technology that aims to develop methods and algorithms that make predictions or decisions based on data. Deep learning refers to machine learning techniques and consists of a neural network (deep neural network) consisting of several layers. Deep learning models can identify complex patterns in data, but require more parameters and computational resources.

Deep learning methods are used in a number of areas to solve various problems (classification of images, video and audio data, speech recognition, sentiment analysis, time series forecasting, etc.). The main advantage of these models is the automatic extraction of the best features from the input data using a general purpose learning procedure (LeCun et al., 2015). The history of the development of deep learning models and their application in artificial neural networks is explored in detail by Schmidhuber et al. (2015).

Time series forecasting has been the focus of machine learning researchers for over 40 years. In recent years, many experiments have proven that recurrent neural networks provide the best results in this area (Sezer et al., 2020).

Recurrent neural networks (RNN) are a type of deep learning network that work better when analyzing time series or sequential data. It was first developed in 1980 for forecasting time series and sequential data. One of the main ideas of a recurrent neural network is that it can use the information obtained in the previous stage in the current task. Unlike simple feed forward neural networks (FNNs), RNNs have internal memory to process input data. The architecture of an RNN model consists of different numbers of layers and different types of blocks at each layer. The main difference between RNN and FNN is that each block receives both current and previous inputs simultaneously. The result also depends on previous data. In this way, memory is formed in the network (Fig. 2) (Hewamalage et al., 2021).

The basic architecture of an RNN consists of an input block, an output block and hidden blocks. Hidden blocks perform all the calculations and generate the output by adjusting the weights.

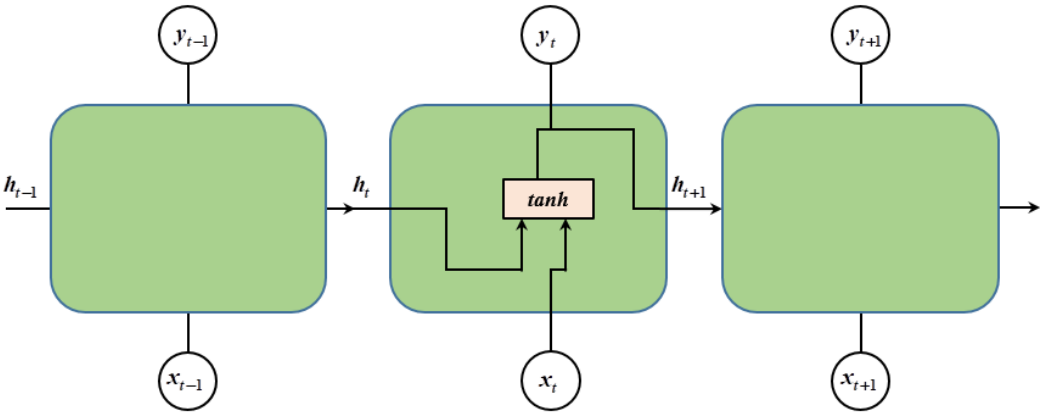

Fig. 2. Recurrent neural networks

The RNN is called "recurrent" because it does the same thing for each element of the sequence, and the result depends on previous calculations. The network changes the weights by comparing the current hidden layer error with the previous hidden layer error.

Recurrent neural networks use backpropagation through time (BPTT) as the training algorithm. Recurrent neural networks adjust weights using BPTT algorithm. The learning algorithm is used to correct the weights in the neural network and ensures that the value obtained at the network output is closer to the actual value. During training, serial data is fed into the network and the output is compared with the expected value. The difference between the received and expected output price is calculated using the loss function. The goal of training is to adjust the network weights to minimize this function.

BPTT calculates the gradients of the network output loss function at each step as a function of time. The gradient is a vector used to update the neural network's weights (determining in which direction the function is incremented). Because gradients are propagated back in time to update the network's weights, updating the weights depends on the entire data sequence in addition to the input and output data. This allows RNNs to learn long-term dependencies and better cope with problems associated with data sequences.

An optimization algorithm (SGD, RMSProp, ADAM) is used to adjust the weights. LSTM networks use various activation functions. The most commonly used activation functions are sigmoid (1), ReLU (rectified linear unit) (2), laky-ReLU (3), hyperbolic tangent (4) and softmax (5):

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{1}$$

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2}$$

$$R(z) = \max(0, z) \tag{3}$$

$$R(z) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x) \tag{4}$$

$$soft\max(z_i) = \frac{\exp z_i}{\sum_j \exp z_i} \tag{5}$$

The hyperparameters of a recurrent neural network determine the architecture of the network, and the correct choice of these parameters affects the performance of the network. The hyperparameters of the network are the number of hidden layers, number of epochs, activation functions, learning rate, batch size, iteration size, etc. Proper selection of these hyperparameters is a major challenge and ensures accurate performance of the model (Reimers & Gurevych, 2017).

The main disadvantage of a recurrent neural network is the problem of gradient vanishing and exploding gradient. The vanishing gradient problem occurs when the strength of the gradient is too small because the layers cannot learn. This makes the learning process more difficult. An exploding gradient problems occurs as a result of the accumulation of large error gradients. This causes the neural network's weights to be updated frequently during training. As a result, the model cannot be trained on training data.

4. Long Short-Term Memory

Recurrent neural networks cannot store very large data vectors. To overcome this problem, Long Short-Term Memory (LSTM) neural networks are used,

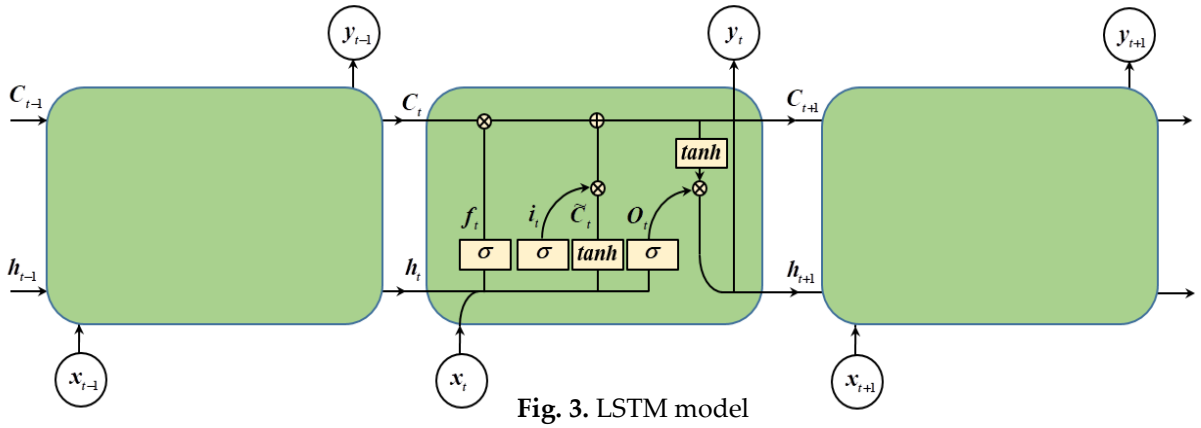

Fig. 3. LSTM model

which are a further improved type of recurrent neural network. Fig. 3 shows examples of a typical LSTM network (Kianimoqadam & Lapp, 2023).

LSTM is a special type of recurrent neural networks first developed by S. Hochreiter and J. Schmidhuber in 1997 and later improved by many researchers. These networks are used to solve a wide range of problems, and in recent years, due to their good predictability of sequences, they have become widely used for predicting stocks and other stock market data. The ability to store information for a long time is one of the main advantages of these networks.

A recurrent network can be described in the form of repeating modules; these modules consist of layers with a tangent activation function. Modules in LSTM networks have a rather complex structure and consist of 4 layers interconnected with a certain pattern. These networks add an internal state cell, the ability to extract the most significant features from the input data and decide which of them have the most impact and which should be passed on to the output.

An LSTM network consists of modules. These modules combine to form the layers of the network. Modules consist of cells. The internal structure of LSTM cells is shown in Fig. 3. A cell state component C_t is added to the LSTM module, which is able to store data for a long time and transfer it to the next step. In Fig. 3, this component is shown as a horizontal line crossing the upper part of the cell. The state of the cell guarantees that all data will be transferred unchanged or deleted. This process is controlled by gates, each cell has 3 types of gates: forget gate (f_t), input gate (i_t) and output gate (o_t). Filters are layers that allow data to be transmitted under certain conditions. The activation function of these layers is sigmoidal and hyperbolic tangent (Salman et al., 2018).

The operating principle of an LSTM cell consists of the following steps:

1. The LSTM determines what information should be removed or retained from the cell state (C_{t-1}). This decision is made by the forgetting filter. f_t is estimated in the interval [0,1] based on the inputs h_{t-1} and x_t. Based on the calculated value, it is determined which part of the information will be transmitted. 0 means no information will be passed, 1 means all information will be transmitted.
2. A decision is made about what information will be stored in the cell state. In this step, the input filter i_t first determines which values to update. A new vector of candidate values $\tilde{C}_t$ is then calculated to be added to the cell state.
3. The old state of the cell C_{t-1} is replaced by the new state C_t.
4. The output value h_t is calculated. To do this, first the output gate o_t is estimated in the interval [0,1] based on the data h_{t-1} and x_t passed to the input and passing through a layer with a sigmoid activation function. The cell state is then estimated in the interval [-1,1] by passing through a layer with a tangent activation function. Based on them, the output value of the LSTM cell is calculated.

The sequence for calculating these parameters is given below:

$$f_t = \sigma(W_f \cdot h_{t-1} + U_f \cdot x_t + b_f)$$

$$i_t = \sigma(W_i \cdot h_{t-1} + U_i \cdot x_t + b_i)$$

$$\tilde{C}_t = \phi(W_C h_{t-1} \cdot U_C x_t + b_C)$$

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

$$h_t = o_t \cdot \varphi(C_t)$$
$$LSTM_{out} = h_t$$

W, *U* - weights, *b* – bias.

5. Experiments

In this section, a model based on recurrent neural networks is proposed for predicting the reliability of software systems. Proper selection of hyperparameters protects the model from overfitting and underfitting problems. In this study, the AutoML library was used in Python for hyperparameter optimization.

3 metrics were used to evalute the performance of the model:

1. Root Mean Square Error - RMSE:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_i'\right)}$$

Here y_i and y_i' are the actual and predicted failure values at time t_i. (x_i, y_i) is a set of data collected from n number of observations. A small *RMSE* value indicates good model performance.

1. The Average Error – *AE*

$$AE = 100\% \times \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i' - y}{y}\right|$$

A small *AE* value indicates good model performance.

2. Coefficient of determination

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - y_i')^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

Here y_i and y_i' are the actual and predicted failure values at time t_i and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.

The coefficient of determination shows how well the model matches the database; it takes a value in the interval [0;1]. For the model to be accepted, the coefficient of determination must be at least 0.5. If this coefficient is above 0.8, the model can be considered quite good.

The experiments were conducted in the Python programming environment. To test the performance of the proposed model, 7 real databases of software product failures were used (Table 1) (DATA Directory in the CD-ROM).

Failures that occur in software systems are recorded during testing and operation. This data is designated as (t_i, N_i). Here N_i is the total number of failures up to time t_i. In this case, the goal of predicting the reliability of a software system is to calculate the total number of future failures based on this data (DS2, DS3, DS7).

The data can also be written as (x_i, T_i). Here T_i is the total time until failure x_i occurs. In this case, the goal of predicting the reliability of a software system is to calculate the time of the next failure based on these data (DS1, DS4, DS5, DS6).

In this article, the prediction was made based on both types of data.

The DS1 database was used to compare the performance of the proposed model with existing models in terms of root mean square error and coefficient of determination (Table 2). This data was compiled by John D. Musa of Bell Telephone Laboratories from 136 faults observed in a real-time command and control system containing 21,700 lines. This study used 3 data sets of approximately equal size. In (Yangzhen et al., 2017), the authors predicted software reliability using an LSTM network. They obtained *AE*=2.01 for the DS1 data. In the proposed model, *AE* = 0.6, since the optimal hyperparameters are selected.

Data from DS1, DS2, DS3, DS4, DS5 and DS6 were used to show the model's performance in terms of Average Error. Different models are taken for comparison (Wang & Zhang, 2018). 5 parametric (heterogeneous Poisson processes), 4 nonparametric (feed forward neural networks) models were considered (Table 3).

Table 4 shows the comparison of the model with the PRNNDWCM (Roy et al., 2014), PNNE (Zheng et al., 2020) and DWCM (Su & Huang, 2007) hybrid neural network models proposed in recent years according to the AE indicator.

Figs. 4-8 show results on real, training and test data sets for DS2, DS3, DS5, DS6 and DS7, respectively. From the figures it can be seen that the proposed model shows good prediction results.

6. Conclusion

Despite the existence of hundreds of models for predicting and assessing the reliability of software systems, the development of models based on new, modern technologies has become a necessity. This study uses seven datasets to examine the effectiveness of the proposed approach.

Table 1. Software failure data sets

Dataset	Faults detected	LOC	Software type
DS1	136	21.700	Realtime Command and Control
DS2	46	40.000	On-line Data Entry
DS3	535	870.000	Realtime Control Application
DS4	213	38.500	Flight Dynamic Application
DS5	266	-	Realtime Control Application
DS6	181	-	-
DS7	81		Brazilian Electronic Switching System

Table 2. Comparison of the performance of the proposed model with other models in terms of RMSE and R^2

	Neural Net	Saburag-Goel	Goel-Okumoto	LSTM
RMSE	0.0952	2.539	3.926	**0.0051**
R^2	0.9958	0.9911	0.9901	**0.9995**

Table 3. Comparison of the performance of the proposed model with different models

Data	Models	*AE*	Data	Models	*AE*
DS1	FFN-Generalization	3.52	DS4	FFN-Generalization	6.83
	FFN-Prediction	2.32		FFN-Prediction	5.46
	JordanNet-Generalization	3.11		JordanNet-Generalization	4.30
	JordanNet-Prediction	3.21		JordanNet-Prediction	4.71
	Logarithmic	3.83		Logarithmic	23.97
	Inverse Polynomial	3.98		Inverse Polynomial	21.08
	Exponential	8.57		Exponential	24.84
	Power	3.94		Power	31.08
	Delayed S-shape	10.99		Delayed S-shape	11.36
	DNN-RED	0.12		DNN-RED	0.02
	LSTM	**0.006**		LSTM	**0.005**
DS2	FFN-Generalization	10.24	DS5	FFN-Generalization	13.90
	FFN-Prediction	12.32		FFN-Prediction	12.71
	JordanNet-Generalization	6.96		JordanNet-Generalization	13.73
	JordanNet-Prediction	9.52		JordanNet-Prediction	11.26
	Logarithmic	12.48		Logarithmic	19.12
	Inverse Polynomial	13.29		Inverse Polynomial	18.27
	Exponential	15.87		Exponential	30.54
	Power	12.95		Power	53.75
	Delayed S-shape	27.10		Delayed S-shape	32.98
	DNN-RED	0.11		DNN-RED	0.05
	LSTM	**0.05**		LSTM	**0.036**
DS3	FFN-Generalization	6.43	DS6	FFN-Generalization	8.32
	FFN-Prediction	6.80		FFN-Prediction	8.80
	JordanNet-Generalization	2.27		JordanNet-Generalization	2.41
	JordanNet-Prediction	1.31		JordanNet-Prediction	3.01
	Logarithmic	13.20		Logarithmic	17.00
	Inverse Polynomial	13.10		Inverse Polynomial	28.87
	Exponential	14.56		Exponential	9.15
	Power	24.71		Power	42.04
	Delayed S-shape	17.76		Delayed S-shape	30.41
	DNN-RED	0.07		DNN-RED	0.10
	LSTM	**0.006**		LSTM	**0.02**

The results were compared with those of both parametric and nonparametric reliability models. Experimental results prove the effectiveness of the proposed model. The main advantage of this model is that the prediction results are positive in datasets of different sizes collected from different application domains. The performance of the proposed model depends on the choice of hyperparameters. Selecting the optimal configuration and training algorithm is one of the difficult tasks facing machine learning specialists. In the future, it is possible to combine LSTM and other (Gated Recurrent Unit) models to predict defects in software systems, since hybrid models show the best results in prediction.

Table 4. Comparison of LSTM and hybrid models

	PRNNDWCM	PNNE	DWCM	LSTM
AE	0.0112	0.0129	0.0176	0.0039

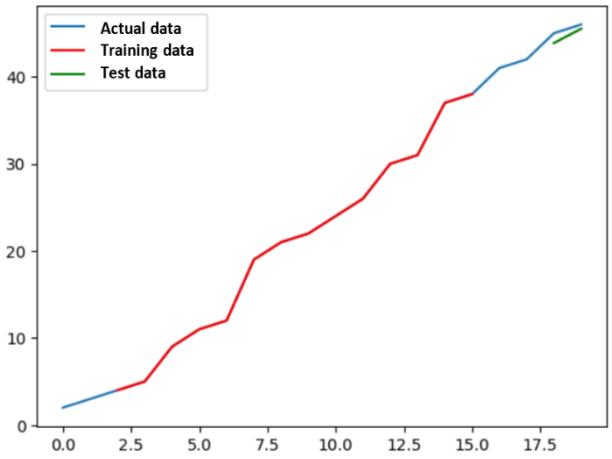

Fig. 4. Training and prediction results for DS2

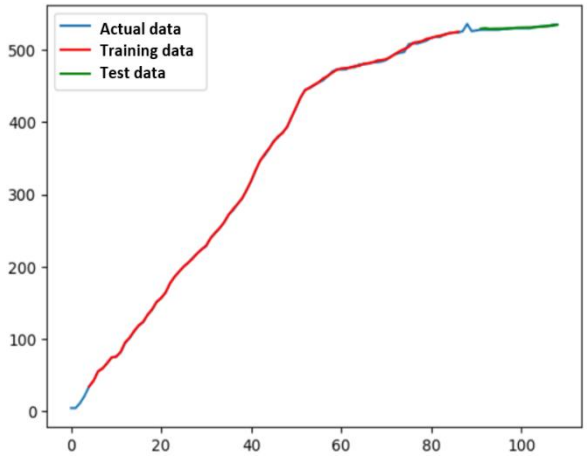

Fig. 5. Training and prediction results for DS3

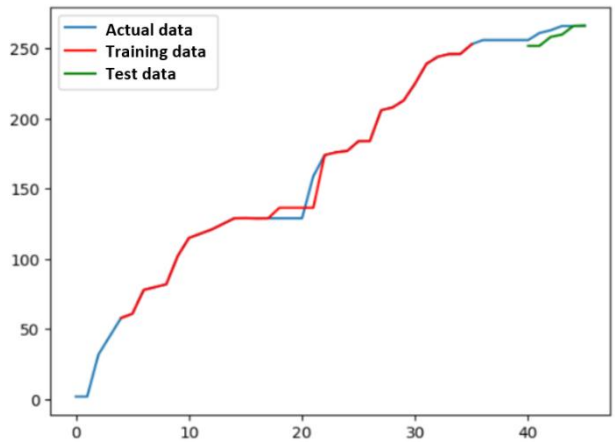

Fig. 6. Training and prediction results for DS5

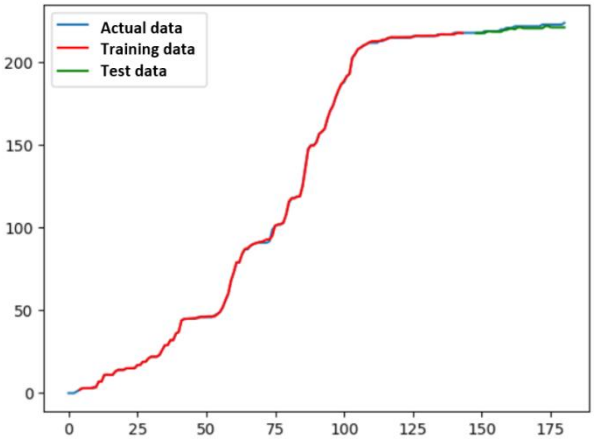

Fig. 7. Training and prediction results for DS6

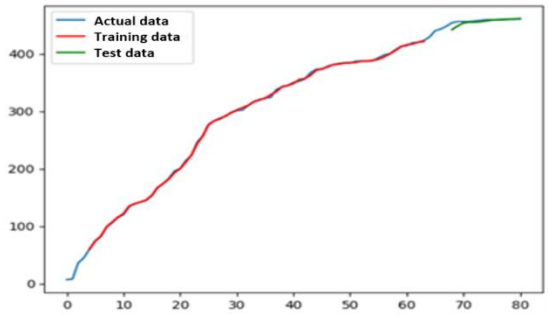

Fig. 8. Training and prediction results for DS7

References

Aljahdali, S. H., & Buragga, K. A. (2008). Employing four ANNs paradigms for software reliability prediction: an analytical study. ICGST-AIML Journal, ISSN, 1687-4846.

Bayramova, T. A. (2023). Development of a Method for Software Reliability Assessment using Neural Networks. Procedia Computer Science, 230, 445-454.

Cai, K. Y., Cai, L., Wang, W. D., Yu, Z. Y., & Zhang, D. (2001). On the neural network approach in software reliability modeling. Journal of Systems and Software, 58(1), 47-62. https://doi.org/10.1016/S0164-1212(01)00027-9

DATA Directory in the CD-ROM. https://www.cse.cuhk.edu.hk/~lyu/book/reliability/data.html

Govindasamy, P., & Dillibabu, R. (2020). Development of software reliability models using a hybrid approach and validation of the proposed models using big data. The Journal of Supercomputing, 76(4), 2252-2265. https://doi.org/10.1007/s11227-018-2457-8

Granitto, P. M., Verdes, P. F., & Ceccatto, H. A. (2005). Neural network ensembles: evaluation of aggregation algorithms. Artificial Intelligence, 163(2), 139-162. https://doi.org/10.1016/j.artint.2004.09.006

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. International Journal of Forecasting, 37(1), 388-427. https://doi.org/10.1016/j.ijforecast.2020.06.008

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hu, Q. P., Dai, Y. S., Xie, M., & Ng, S. H. (2006). Early software reliability prediction with extended ANN model. In 30th Annual International Computer Software and Applications Conference (COMPSAC'06), Chicago, USA, September 2006 (pp. 234-239). https://doi.org/10.1109/COMPSAC.2006.130

Karunanithi, N., Malaiya, Y. K., & Whitley, L. D. (1991). Prediction of software reliability using neural networks. In ISSRE (pp. 124-130).

Khoshgoftaar, T. M., & Lanning, D. L. (1995). A neural network approach for early detection of program modules having high risk in the maintenance phase. Journal of Systems and Software, 29(1), 85-91. https://doi.org/10.1016/0164-1212(94)00130-F

Kianimoqadam, A., & Lapp, J. (2023). Calculating the view factor of randomly dispersed multi-sized particles using hybrid GRU-LSTM recurrent neural networks regression. International Journal of Heat and Mass Transfer, 202, 123756. https://doi.org/10.1016/j.ijheatmasstransfer.2022.123756

Lai, R., & Garg, M. (2012). A detailed study of NHPP software reliability models. J. Softw., 7(6), 1296-1306. doi:10.4304/jsw.7.6.1296-1306

Lakshmanan, I., & Ramasamy, S. (2017). Improving software reliability estimation using multi-layer neural-network combination model. International Journal of Innovative Computing and Applications, 8(2), 113-121, https://doi.org/10.1504/IJICA.2017.084897

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444. https://doi.org/10.1038/nature14539

Lu, K., & Ma, Z. (2018, October). Parameter estimation of software reliability growth models by a modified whale optimization algorithm. In 2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), Wuxi, China, October 2018 (pp. 268-271). https://doi.org/10.1109/DCABES.2018.00076

Mičko, R., Chren, S., & Rossi, B. (2022). Applicability of Software Reliability Growth Models to Open Source Software. In 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Gran Canaria, Spain, August - September 2022 (pp. 255-262). https://doi.org/10.1109/SEAA56994.2022.00047

Musa, J. D. (2004). Software reliability engineering: more reliable software. Faster Development and Testing, 632.

Munir, H. S., Ren, S., Mustafa, M., Siddique, C. N., & Qayyum, S. (2021). Attention based GRU-LSTM for software defect prediction. Plos one, 16(3), e0247444. https://doi.org/10.1371/journal.pone.0247444

Ramasamy, S., & Preetha, C. D. (2016). Dynamically weighted combination model for describing inconsistent failure data of software projects. Indian Journal of Science and Technology, 9(35), 1-4. http://doi.org/10.17485/ijst/2016/v9i35/90211

Reimers, N., & Gurevych, I. (2017). Optimal hyperparameters for deep LSTM-networks for sequence labeling tasks. arXiv preprintarXiv:1707.06799. https://doi.org/10.48550/arXiv.1707.06799

Roy, P., Mahapatra, G. S., Rani, P., Pandey, S. K., & Dey, K. N. (2014). Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction. Applied Soft Computing, 22, 629-637. https://doi.org/10.1016/j.asoc.2014.04.012

Sahu, K., Alzahrani, F. A., Srivastava, R. K., & Kumar, R. (2021). Evaluating the Impact of Prediction Techniques: Software Reliability Perspective. Computers, Materials & Continua, 67(2). https://doi.org/10.32604/cmc.2021.014868

Salman, A. G., Heryadi, Y., Abdurahman, E., & Suparta, W. (2018). Single layer & multi-layer long short-term memory (LSTM) model with intermediate variables for weather forecasting. Procedia Computer Science, 135, 89-98. https://doi.org/10.1016/j.procs.2018.08.153

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117. https://doi.org/10.1016/j.neunet.2014.09.003

Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Applied soft computing, 90, 106181. https://doi.org/10.1016/j.asoc.2020.106181

Singh, Y., & Kumar, P. (2010). Application of feed-forward neural networks for software reliability prediction. ACM SIGSOFT Software Engineering Notes, 35(5), 1-6. https://doi.org/10.1145/1838687.1838709

Stisen, A., Blunck, H., Bhattacharya, S., Prentow, T. S., Kjærgaard, M. B., Dey, A., Sonne, T., & Jensen, M. M. (2015). Smart devices are different: assessing and mitigating mobile sensing heterogeneities for activity recognition. 2015 ACM Conference on Embedded Networked Sensor Systems (SenSys), Seoul, South Korea, November 2015 (pp. 127-140). https://doi.org/10.1145/2809695.2809718

Su, Y. S., & Huang, C. Y. (2007). Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. Journal of Systems and Software, 80(4), 606-615. https://doi.org/10.1109/ACCESS.2020.2972826

Tian, L., & Noore, A. (2005). Evolutionary neural network modeling for software cumulative failure time prediction. Reliability Engineering & system safety, *87*(1), 45-51. https://doi.org/10.1016/j.ress.2004.03.028

Wang, J., & Zhang, C. (2018). Software reliability prediction using a deep learning model based on the RNN encoder–decoder. Reliability Engineering & System Safety, 170, 73-82. https://doi.org/10.1016/j.ress.2017.10.019

Yangzhen, F., Hong, Z., Chenchen, Z., & Chao, F. (2017, July). A software reliability prediction model: Using improved long short term memory network. In 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Prague, Czech Republic, July 2017 (pp. 614-615). https://doi.org/10.1109/QRS-C.2017.115

Zhen, L., Liu, Y., Dongsheng, W., & Wei, Z. (2020). Parameter estimation of software reliability model and prediction based on hybrid wolf pack algorithm and particle swarm optimization. IEEE Access, 8, 29354-29369. https://doi.org/10.1109/ACCESS.2020.2972826

Zheng, J. (2009). Predicting software reliability with neural network ensembles. Expert systems with applications, 36(2), 2116-2122. https://doi.org/10.1016/j.eswa.2007.12.029