

Article

Deep-Learning Software Reliability Model Using SRGM as Activation Function

Youn Su Kim ¹, Hoang Pham ² and In Hong Chang ^{1,*}¹ Department of Computer Science and Statistics, Chosun University, 309 Pilmun-daero, Dong-gu, Gwangju 61452, Republic of Korea; imk92315@chosun.kr² Department of Industrial and Systems Engineering, Rutgers University, 96 Frelinghuysen Road, Piscataway, NJ 08855-8018, USA; hopham@rci.rutgers.edu

* Correspondence: ihchang@chosun.ac.kr; Tel.: +82-62-230-6621

Abstract: Software is widely used in various fields. There is no place where it is not used from the smallest part to the entire part. In particular, the tendency to rely on software is accelerating as the fields of artificial intelligence and big data become more widespread. Therefore, it is extremely important to evaluate the reliability of software because of the extensive damage that could occur if the software fails. Previously, software reliability models were developed based on mathematical and statistical grounds; however, immediate response was difficult. Therefore, in this study, a software reliability model was developed that depends on data using deep learning, and it was analyzed by replacing the activation function previously used in deep learning with the proposed software reliability model. Since the sigmoid function has a similar shape to the software reliability model, we utilized this to propose a deep learning software reliability model that replaces the activation function, the sigmoid function, with the software reliability function. Two datasets were compared and analyzed using 10 criteria, and the superiority of the proposed deep-learning software reliability model was proved. In addition, the results were compared by changing the parameters utilized in the proposed deep-learning software reliability model by -10% , -5% , 5% , and 10% , and it was found that the larger the parameters, the smaller the change.

Keywords: deep-learning software reliability model; software reliability growth model; activation function; sensitivity analysis



Citation: Kim, Y.S.; Pham, H.; Chang, I.H. Deep-Learning Software Reliability Model Using SRGM as Activation Function. *Appl. Sci.* **2023**, *13*, 10836. <https://doi.org/10.3390/app131910836>

Academic Editor: Jérôme Morio

Received: 3 September 2023

Revised: 25 September 2023

Accepted: 27 September 2023

Published: 29 September 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Software is widely used in many fields. There is no place where it is not used from the smallest part to the entire part. Currently, the fields of artificial intelligence and big data have gained considerable attention and are based on software along with high-performance hardware. These fields require many calculations. Therefore, to make the calculations as simple as possible and to learn quickly, the code, structure, and configuration of the software are extremely important. As the artificial intelligence and big data industries develop, they must rely heavily on software. If part of the developed software fails owing to a code error or an error in the calculation process, it can cause many problems. The error in the computation may produce unexpected results, or the error may cause a malfunction in a machine that operates in real time; therefore, research to improve software reliability has been steadily conducted [1–4].

There is plenty of study in the field of system reliability that extends beyond software reliability. The study is being conducted on how to ensure reliability and service in more complex systems than software. It has been shown that information about system failures and errors is important to increase the reliability of a system and to troubleshoot problems. In addition, sustainable development is becoming a very important topic in system reliability. Sustainable development means operating production processes and developing products in a way that is environmentally friendly and considers the environment and

economic aspects of workers. To create a sustainable production environment, breakdowns and energy waste must be minimized or eliminated, and maintenance is essential. For sustainable development, technologies equipped with intelligent learning self-diagnosis and self-adaptive technologies that monitor, control, and process data are needed [5,6].

Software reliability is a field in which how well software works is measured, and software reliability growth model (SRGM) is developed based on software failure. Because software failure data follows a Poisson distribution that indicates the number of failures that occurred during a period of time, the SRGM currently being studied is based on the SRGM that assumes a nonhomogeneous Poisson process (NHPP) [7]. The SRGM assuming NHPP, which started with Goel-Okumoto [8], has been extended to SRGMs reflecting various assumptions such as independent assumptions, uncertain operating environments, and dependent assumptions. From a parametric perspective, SRGMs are developed by estimating optimal parameters. In addition, software reliability models from a non-parametric perspective that depend on data have been continuously studied, based on deep learning or other machine learning algorithms such as deep neural networks and recurrent neural networks [9,10]. However, in deep learning, the typical nonlinear activation function used can cause training difficulties due to the saturation behavior, so new activation functions such as noisy or self-regularized non-monotonic activation function are being studied [11,12].

In this study, based on the theoretical background of the problem of activation function, we would like to propose a deep learning software reliability model by combining a software reliability model from a parametric perspective and a software reliability model using deep learning from a non-parametric perspective. Because the structure of the existing software reliability model is similar to the sigmoid function, the proposed software reliability model utilizes the shape of the sigmoid function and applies it to learning instead of the activation function of deep learning. This is a deep learning software reliability model by using an activation function with mathematical assumptions to solve the problem of deep learning that only depends on data. The superiority of the model is demonstrated by comparing 10 criteria using two datasets. Furthermore, the performances of the proposed deep-learning software reliability model are compared when the parameters of the model are changed. This paper is composed as follows: In Section 2, we introduce related research. In Section 3, the theoretical background of software reliability models is provided and the proposed software reliability model is introduced. In Section 4, numerical examples are provided. The results of the sensitivity analysis are presented in Section 5. Discussions and conclusions are given in Sections 6 and 7.

2. Related Research

The study of software reliability is essential for reducing software failures and managing them better. Past SRGMs were developed based on mathematical and statistical assumptions. Early SRGMs were studied by Goel and Okumoto [8] under the assumption that independent software failures are continuous problems. Based on this, the SRGM was proposed by assuming that software failures would occur in an S-shape [13–15]. Zhang et al. [16] proposed a software reliability model that assumes incomplete debugging with a functional form of the fault detection rate. Yamada et al. [17] developed a SRGM with a constant defect detection rate by assuming incomplete debugging, in which defects detected during the testing phase are not corrected or removed. An incomplete debugging about defect detection rate, in which the defect detection rate is a function, has also been studied [18–20]. Roy et al. [21] presented a software reliability model that considers the case of incomplete debugging and error detection in the testing phase, with a sharp increase in the early stages and then a constant form of defect detection. Because the software operating environment is different for software, comparison is hard; therefore, SRGMs that consider the uncertain operating environment have been developed [22–24]. Chang et al. [25] proposed a software reliability model with a fault detection rate function that considers the uncertainty of the operating environment and a new test coverage, and

Song et al. [26] proposed an uncertain operating environment software reliability model using a fault detection rate function that includes three parameters.

In addition, SRGMs which assume that software failures occur dependently have been studied because the developed software is composed of complexity [27,28]. A dependent SRGM was developed that assumes that past errors or faults will continue to contribute to software failures if they are not corrected [29,30]. Furthermore, Lee et al. [31] proposed a dependent SRGM that assumes an uncertain operating environment.

SRGMs with mathematical and statistical assumptions make predictions based on failure trends. However, because it is not known when and how failures will occur, there is an ongoing debate about software reliability from a nonparametric perspective, as opposed to an immediate failure perspective [32]. Miyamoto [33] developed a software reliability model for software failures that occur in open-source software using deep neural networks (DNNs). In addition, because software failures have time-series characteristics, a software reliability model using long short-term memory (LSTM) was proposed [34,35]. Wu et al. [36] proposed a structure in which the weights of the software reliability model are learned through the weights of the DNN. Batool et al. [37] proposed a model to predict software errors and defects by synthesizing not only deep learning but also various machine learning techniques. Sreekanth et al. [38] improved current methodologies such as RE and MRE by evaluating the estimation required for software development. Bhuyan et al. [39] demonstrated the superiority of the proposed feedforward neural network through a comparison between the feedforward neural network and the existing parametric SRGM. Mittelman [40] proposed a convolutional neural network-based time series model utilizing software failure data with time series data characteristics. Pan et al. [41] proposed a convolution neural network model that solved problems such as limited data size and insufficient number of experiments. To solve these problems, they built and utilized the PROMISE Source Code (PSC) dataset, which is an extension of the original dataset.

3. Software Reliability Model

3.1. NHPP Software Reliability Growth Model

Software reliability is the ability of software to successfully perform its intended function for the intended duration under given conditions of use. The reliability function for evaluating software reliability is shown in Equation (1), which is the probability that the software will operate for at least t hours.

$$R(t) = P(T > t) = \int_t^{\infty} f(u) du \quad (1)$$

The probability density function $f(t)$ assumes that the time to failure of the software is a random variable T . To measure the reliability function $R(t)$, the mean number of failures is assumed to follow an exponential distribution with parameter λ . The parameter λ follows the NHPP, a function that changes over time. The model is derived by Equation (2), where $N(t)$ is the number of failures that have occurred by time t , and n is the number of failures.

$$\Pr\{N(t) = n\} = \frac{\{m(t)\}^n}{n!} e^{-m(t)}, n = 0, 1, 2, \dots, t \geq 0 \quad (2)$$

where a mean value function $m(t)$ is the integral of $\lambda(t)$, the intensity function, which is the number of failures at time t , from time 0 to time t , which is the $m(t)$ from 0 to time t and follows Equation (3). The intensity function $\lambda(t)$ is the number of instantaneous failures at time t .

$$m(t) = \int_0^t \lambda(s) ds \quad (3)$$

The $m(t)$ is used to develop a software reliability model through a differential equation, with $a(t)$ representing the number of failures and $b(t)$ representing the failure detection rate. It is expressed as Equation (4), which is the most basic SRGM.

$$\frac{dm(t)}{dt} = b(t)[a(t) - m(t)] \quad (4)$$

When Equation (4) is multiplied by η , it is defined as a SRGM with assumptions about an uncertain operating environment added. When the $m(t)$ is multiplied once more, it is defined as a SRGM which assumes that software failures occur dependently. By solving each differential equation with respect to $m(t)$, one can derive the SRGM that satisfies each condition. The $a(t)$, $b(t)$, and η are expressed in variable shape. The functions contain a number of parameters with meanings such as number of software failures, expected number of failures, failure detection rate, etc. The parameters included in each model can have different meanings for the same parameter, so please refer to the references for each model. The software models developed on this basis are as follows: Models 1–9 are SRGMs obtained through the basic form of differential equations, Models 10–12 are SRGMs assuming an uncertain operating environment, Models 13 and 14 are SRGMs assuming a dependent failure, and Model 15 is the SRGM assuming a dependent failure in an uncertain operating environment. Table 1 shows the descriptions for the 15 models.

Table 1. Software reliability growth models.

No.	Model	Mean Value Function	Note
1	Goel–Okumoto (GO) [8]	$m(t) = a(1 - e^{-bt})$	Concave
2	Ohba (IS) [15]	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$	S-shaped
3	Zhang et al. (ZFR) [16]	$m(t) = \frac{a}{p - \beta} \left[1 - \left(\frac{(1 + \alpha)e^{-bt}}{1 + \alpha e^{-bt}} \right)^{\frac{c}{b}(p - \beta)} \right]$	S-shaped
4	Yamada et al. (YID 1) [17]	$m(t) = \frac{ab}{\alpha + b} (e^{\alpha t} - e^{-bt})$	Concave
5	Yamada et al. (YID 2) [17]	$m(t) = a(1 - e^{-bt}) \left(1 - \frac{\alpha}{b} \right) + \alpha at$	Concave
6	Pham–Zhang (PZ) [18]	$m(t) = \frac{((c + a)[1 - e^{-bt}] - [\frac{ab}{b - \alpha}](e^{-at} - e^{-bt}))}{1 + \beta e^{-bt}}$	Both
7	Pham et al. (PNZ) [19]	$m(t) = \frac{a(1 - e^{-bt})(1 - \frac{\alpha}{b}) + \alpha at}{1 + \beta e^{-bt}}$	Both
8	Teng–Pham (TP) [20]	$m(t) = \frac{a}{p - q} \left[1 - \left(\frac{\beta}{\beta + (p - q) \ln \left(\frac{c + e^{bt}}{c + 1} \right)} \right)^{\alpha} \right]$	S-shaped
9	Roy et al. (RMD) [21]	$m(t) = a\alpha [1 - e^{-bt}] - \left[\frac{ab}{b - \beta} (e^{-\beta t} - e^{-bt}) \right]$	Concave
10	Pham (Vtub) [24]	$m(t) = N \left[1 - \left(\frac{\beta}{\beta + a^{bt} - 1} \right)^{\alpha} \right]$	S-shaped
11	Chang et al. (TC) [25]	$m(t) = N \left[1 - \left(\frac{\beta}{\beta + (at)^b} \right)^{\alpha} \right]$	Both
12	Song et al. (3P) [26]	$m(t) = N \left[1 - \left(\frac{\beta}{\beta - \frac{a}{b} \ln \left(\frac{(1 + c)e^{-bt}}{1 + ce^{-bt}} \right)} \right)^{\alpha} \right]$	S-shaped
13	Kim et al. (DPF1) [29]	$m(t) = \frac{a}{1 + \frac{a}{h} \left(\frac{1 + c}{c + e^{bt}} \right)^{\frac{a}{h}}}$	S-shaped, dependent
14	Lee et al. (DPF2) [30]	$m(t) = \frac{a}{1 + \frac{a}{h} \left(\frac{b + c}{c + be^{bt}} \right)^{\frac{a}{h}}}$	S-shaped, dependent
15	Lee et al. (UDPF) [31]	$m(t) = N \left[1 - \left(\frac{\beta}{\alpha + bt - \ln(bt + 1)} \right)^{\alpha} \right]$	S-shaped, dependent

3.2. Software Reliability Model Using Deep Learning

3.2.1. Deep Neural Network

An artificial neural network is organized through nonlinear connections and modeled after the biological system of human thinking. Based on neurons (nodes), the most basic unit of information, the artificial neural network comprises input, output, and hidden layers. The input layer takes the information, the output layer considers or makes judgments based on the information received, and the hidden layer transmits the information between the input and output layers. It not only transmits information but also performs information transmission through a nonlinear combination. After the weights and biases of each layer are combined, along with the information obtained, the sum and product are combined, and the result is derived through a nonlinear transformation using the activation function. When there are many hidden layers in the structure, it is a DNN, as shown in Figure 1. The circles in the figure represent neurons. The data put to the input layer is informed to the next hidden layer through a combination of sums and products with weights, biases, and activation functions. This process is repeated until the output layer [42,43].

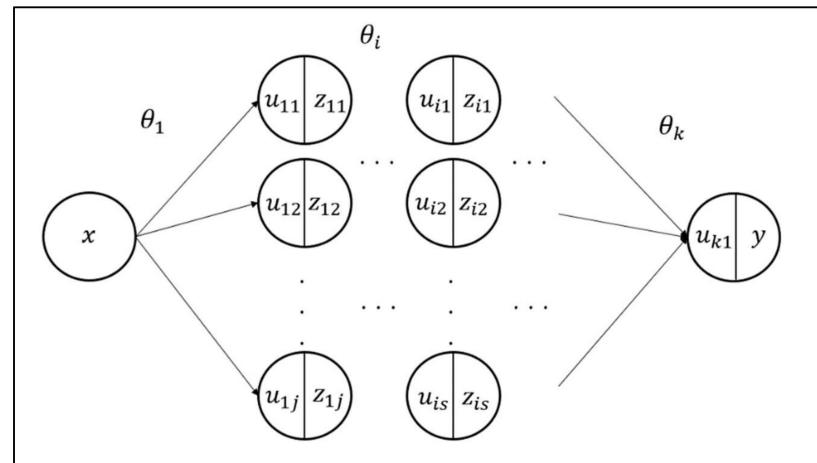


Figure 1. Structure of DNN model.

When transferred from layer to layer, the sigmoid function, hyperbolic tangent function, or rectified linear unit (ReLU) function is used to convert the nonlinear functions. In addition, the result transmitted to the output layer uses a sigmoid or softmax function for categorical variables and an identity function for continuous variables. In the field of software reliability, the sigmoid function is used because the software failure itself is a structure that increases in a sigmoid form, and the identity function is used because the dependent variable is a continuous variable. If the final predicted value y is obtained through the above process, a loss function representing the difference between the actual value and the predicted value is learned based on the partial derivative value for a specific parameter, where θ represents the weight and bias parameters, and α represents the learning rate. It is expressed as Equation (5).

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial}{\partial \theta} \text{Loss}(\theta) \quad (5)$$

3.2.2. Recurrent Neural Network

A recurrent neural network is a model with cells that remember information from previous points in time, and it is specialized for learning sequential data. It is designed so that information from past points in time can influence future points in time. Figure 2a shows the structure of the RNN model represented by Equation (6). h_{t-1} is information about the previous layer, x_t is information about the current layer, θ_h is the weight and

bias of the previous layer, and θ_x is the weight and bias of the current layer. The activation functions utilized are a hyperbolic tangent function ($\tanh$) [44].

$$h_t = \tanh(\theta_h h_{t-1} + \theta_x x_t) \quad (6)$$

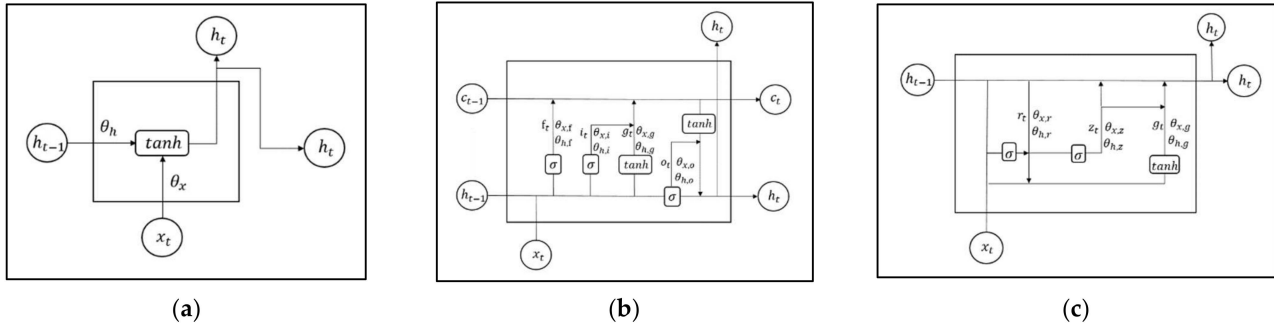


Figure 2. (a) Structure of RNN model, (b) structure of LSTM model, and (c) structure of GRU model.

However, while recurrent neural networks can influence later time points, they suffer from the problem that, as the time horizon becomes longer, the information from farther back in time is not reflected as well. To solve this problem, a model that has the ability to train long-term dependence, LSTM, was developed. An LSTM has a memory cell c . Memory cell c consists of a forget gate (f_t), an input gate (i_t), a new memory cell (g_t), and an output gate (o_t). The f_t determines how much previous information is reflected, the i_t contains new information, the g_t determines how much new information is reflected, and the o_t indicates the amount of previous information passed to the next layer. The $\theta_{x,f}$, $\theta_{x,g}$, $\theta_{x,i}$, and $\theta_{x,o}$ are the weight for each gate connected to the input data. The $\theta_{h,f}$, $\theta_{h,g}$, $\theta_{h,i}$, and $\theta_{h,o}$ are the weight of each gate for the previous layer. The c_t contains information about the long-term state by combining previous information and current information, and the h_t contains information about the short-term state. The activation functions utilized are a sigmoid function (σ) and the $\tanh$. Memory cells can be used to solve the problem of vanishing gradients. The structure of the LSTM follows Equation (7). Figure 2b shows the structure of the LSTM model [45,46].

$$\begin{aligned} f_t &= \sigma(x_t \theta_{x,f} + h_{t-1} \theta_{h,f}) \\ g_t &= \tanh(x_t \theta_{x,g} + h_{t-1} \theta_{h,g}) \\ i_t &= \sigma(x_t \theta_{x,i} + h_{t-1} \theta_{h,i}) \\ o_t &= \sigma(x_t \theta_{x,o} + h_{t-1} \theta_{h,o}) \\ c_t &= f_t \odot c_{t-1} + g_t \odot i_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \quad (7)$$

LSTM solved the problem of the vanishing gradient for RNN. However, the larger the data size, the more computational effort is required by multiplying the number of parameters, resulting in inefficiency. To improve this, a gated recurrent unit (GRU) model was proposed. It reduces the four operations of the LSTM to three, creates an update gate (z_t) that combines the f_t with the i_t in the LSTM into one, and replaces the o_t with a reset gate (r_t). The r_t and z_t determine the size of previous information forgotten and how much previous information is retained. The g_t is the gate that determines the information candidates at time t . $\theta_{x,r}$, $\theta_{x,z}$, and $\theta_{x,g}$ are the weight for each gate connected to the input data. $\theta_{h,r}$, $\theta_{h,z}$, and $\theta_{h,g}$ are the weight of each gate for the previous layer. The h_t , which integrates the c_t and h_t of the LSTM, contains information from the previous and current

layers together. It learns as it transfers to the next layer through Equation (8). Figure 2c shows the structure of the GRU model [47,48].

$$\begin{aligned} r_t &= \sigma(x_t \theta_{x,r} + h_{t-1} \theta_{h,r}) \\ z_t &= \sigma(x_t \theta_{x,z} + h_{t-1} \theta_{h,z}) \\ g_t &= \tanh(x_t \theta_{x,g} + (r_t \circ h_{t-1}) \theta_{h,g}) \\ h_t &= (1 - z_t) \circ g_t + z_t \circ h_{t-1} \end{aligned} \quad (8)$$

The structure of the deep-learning model used in this study is as follows: The DNN consists of four hidden layers. The activation function is a sigmoid function, and the number of nodes in each layer is used as the training size of the dataset. The parameter size is equal to the number of nodes, and the number of nodes is defined by the data size. All of the recurrent neural network types have a first-order difference. All deep-learning models are trained equally with 100 epochs to minimize overfitting, and the optimization method utilizes Adam, which simultaneously considers the acceleration of the previous training and the structure in which the learning rate is not constant and changes with each training.

3.2.3. Proposed Software Reliability Model

Typical failure occurrences follow either an S shape, in which failures occur in increasing numbers and then decrease past a certain point, or a concave shape, in which many failures occur at the beginning and end. The SRGMs developed thus far are based on the assumption of these shapes. The S-shaped form is similar to the sigmoid function in deep-learning activation functions. The sigmoid function has an S-shaped or sigmoidal curve with $1/2$ when x is 0 in Equation (9) [49–51].

$$f(x) = \frac{1}{1 + \exp(-x)} \quad (9)$$

In this study, the sigmoid function was the activation function used because it most closely resembles the shape of the software reliability model, and a generalized software reliability model was developed with a $b(t)$ function whose input value is not a simple indexing number but the failure detection rate of the software reliability model. The failure detection rate function $b(t)$ utilizes $\frac{b^2 t}{bt+1}$ wherein parameter b is the shape parameter and has a structure in which the failure probability changes with time. The final model follows Equation (10).

$$m(t) = \frac{1}{1 + \exp\left(-\frac{b^2 t}{bt+1}\right)} \quad (10)$$

As mentioned previously, this model is a modified sigmoid form. In this study, instead of using the well-known sigmoid or ReLU activation function in deep learning, the proposed model was employed by converting it into a modified form of the proposed software reliability model. This is the proposed deep-learning software reliability model, which applies mathematical and statistical assumptions instead of only data-dependent models, and the structure of this model is the same as that of the model defined in Section 3.2.1. There are a total of four hidden layers, and the number of nodes in the hidden layer is the same as the number of data. We do not have more nodes than the number of data, as this can lead to overfitting problems. The activation function utilizes the proposed software reliability model that is similar to the form of the sigmoid function. An optimizer method utilizes Adam, which simultaneously considers momentum and adaptive learning rate [52]. Based on this, we structure a deep learning software reliability model and train it for 100 epochs. Table 2 shows a description of the software reliability model using deep learning and the proposed deep learning software reliability model. In addition, 20 models (the four software reliability models using the deep learning introduced in Sections 3.2.1 and 3.2.2,

the proposed deep-learning software reliability model introduced in Section 3.2.3, and the 15 models introduced in Section 3.1, are compared.

Table 2. Software reliability models using deep learning and the proposed deep-learning software reliability model.

No.	Model	Mean Value Function and Structure
16	DNN	Hidden Layer = 4, Activation Function = Sigmoid, Optimizer = Adam
17	RNN	Difference = 1, Optimizer = Adam
18	LSTM	Difference = 1, Optimizer = Adam
19	GRU	Difference = 1, Optimizer = Adam
20	Proposed Model	$m(t) = \frac{1}{1 + e^{-\frac{t^2}{bt+1}}}$

4. Numerical Examples

4.1. Data Information

Two datasets were used. Dataset 1 contains the failure data of critical, major, and minor errors, representing the data of 231 faults over a period of 38 weeks [53]. Dataset 2 contains the failure data generated by AVRO using the Apache open-source software. AVRO is a program that provides communication technology with external modules and serialization functions to support external storage and input/output, and a total of five releases from version 1.0.0 to version 1.7.7 were considered. Monthly data were collected on bugs, new features, improved features, and other fixed issues in each AVRO version. The period was five years, from July 2009 to July 2014, with 949 cumulative failures per month [54]. The two datasets show that the number of failures remains more or less constant as the time to failure increases, rather than a trend of a significant decrease. For Datasets 1 and 2, 90% of the training data were used for training and parameter estimation, and the remaining 10% were used to compare the predictions of the trained and estimated models.

4.2. Criteria

Twenty SRGMs, software reliability models using deep learning and the proposed deep-learning software reliability model, were compared using 10 criteria to determine the superior model. These 10 criteria were based on the difference between the actual (y_i) and predicted values ($\hat{m}(t_i)$). The mean squared error (MSE) is defined as the sum of the squares of the difference between the predicted number and the actual number of failures [55]. The mean absolute error (MAE) measures the difference between the predicted number of failures and the observed value, considering the number of observations as the sum of absolute values [56]. Predictive ratio risk (PRR) is measured by dividing the distance from the predicted number of failures to the actual number of failures by the predicted value with respect to model estimation [57]. Predictive power (PP) is measured as the distance from the actual number of failures to the predicted number of failures divided by the actual value [58]. R^2 measures the success of the fit in explaining the variation in the data [59]. Root mean square prediction error (RMSPE) estimates the closeness of the model to the predicted observations [60]. The mean error of prediction (MEOP) sums the absolute value of the deviation between the actual number of observations and the estimated curve and is defined elsewhere [61]. The Theil statistic (TS) is the ratio of the average deviation of all periods to the actual observations [62]. Pham's criterion (PC) considers the tradeoff between the uncertainty of the model and the number of parameters in the model [63]. The prediction sum squared error (preSSE) represents the sum of the differences between the predicted and actual values [64]. Table 3 shows a description of the 10 criteria.

Table 3. Criteria for model comparisons.

No.	Criteria	Formula
1	MSE	$\frac{\sum_{i=1}^n (\hat{m}(t_i) - y_i)^2}{n}$
2	MAE	$\frac{\sum_{i=1}^n \hat{m}(t_i) - y_i }{n}$
3	PRR	$\sum_{i=1}^n \left(\frac{\hat{m}(t_i) - y_i}{\hat{m}(t_i)} \right)^2$
4	PP	$\sum_{i=1}^n \left(\frac{\hat{m}(t_i) - y_i}{y_i} \right)^2$
5	R^2	$1 - \frac{\sum_{i=1}^n (\hat{m}(t_i) - y_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$
6	RMSPE	$\sqrt{\text{Variance}^2 + \text{Bias}^2}$
7	MEOP	$\frac{\sum_{i=1}^n \hat{m}(t_i) - y_i }{n+1}$
8	TS	$100 * \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{m}(t_i))^2}{\sum_{i=1}^n y_i^2}} \%$
9	PC	$\frac{n}{2} * \log \left(\frac{\sum_{i=1}^n (\hat{m}(t_i) - y_i)^2}{n} \right) + \left(\frac{n-1}{n} \right)$
10	preSSE	$\sum_{i=k+1}^n (\hat{m}(t_i) - y_i)^2$

Based on the above criteria, the existing SRGMs and software reliability models using deep learning are compared with the proposed deep-learning software reliability model. The R^2 value is closer to 1, and the other nine scales are closer to 0, indicating a better fit. Using R and MATLAB, we estimated the parameters of each model using the least-squares estimation (LSE) method, calculated the goodness of fit, and compared their performance. The LSE method estimates the parameters that minimize errors. We want to estimate the optimal parameters of each model through Equation (11) [65].

$$LSE = \sum_{i=1}^n (y_i - \hat{m}(t_i))^2 \quad (11)$$

The proposed deep learning software reliability model selects the parameter b through the LSE method, and the parameters of deep learning are trained by the given data. The model is trained 100 times for each selected parameter to present the best results.

4.3. Results of Dataset 1

Table 4 lists the estimated parameter values of the SRGM assuming an NHPP using Dataset 1 and the structure of the software reliability model using deep learning. It also shows the estimated parameter values and structure of the proposed deep-learning software reliability model. The parameter b of the proposed deep-learning software reliability model is estimated to be 0.52. Based on the estimated parameter b , the structure of the deep-learning software reliability model has four hidden layers. The Adam optimization method is used, and the learning rate is set to 0.000001. Figure 3 shows the estimated value of $m(t)$ for each model and deep-learning structure.

Table 4. Parameter estimation and structure of model from Dataset 1.

No.	Model	Estimation and Structure
1	GO	$\hat{a} = 405.3473, \hat{b} = 0.01966$
2	IS	$\hat{a} = 398.5632, \hat{b} = 0.02049, \hat{\beta} = 0.02548$
3	ZFR	$\hat{a} = 48.7172, \hat{b} = 0.00231, \hat{\alpha} = 0.04331,$ $\hat{\beta} = 0.23542, \hat{c} = 0.17070, \hat{p} = 0.35612$
4	YID 1	$\hat{a} = 205.0374, \hat{b} = 0.04056, \hat{\alpha} = 0.01222$
5	YID 2	$\hat{a} = 98.4888, \hat{b} = 0.08904, \hat{\alpha} = 0.04632$
6	PZ	$\hat{a} = 27.1595, \hat{b} = 0.06871, \hat{\alpha} = 0.02132,$ $\hat{\beta} = 2.16867, \hat{c} = 232.0570$
7	PNZ	$\hat{a} = 40.2796, \hat{b} = 0.29103,$ $\hat{\alpha} = 0.13231, \hat{\beta} = 0.05003$
8	TP	$\hat{a} = 17840.283, \hat{b} = 0.02703, \hat{\alpha} = 0.00375, \hat{\beta} = 0.01292,$ $\hat{c} = 15.38103, \hat{p} = 0.60263, \hat{q} = 0.10827$
9	RMD	$\hat{a} = 42.38821, \hat{b} = 0.02091,$ $\hat{\alpha} = 10.00201, \hat{\beta} = 0.00959$
10	Vtub	$\hat{a} = 1.82256, \hat{b} = 0.50401, \hat{\alpha} = 0.52648,$ $\hat{\beta} = 12.54998, \hat{N} = 399.1401$
11	TC	$\hat{a} = 0.04460, \hat{b} = 0.84140, \hat{\alpha} = 12.34453,$ $\hat{\beta} = 220.2973, \hat{N} = 2627.4388$
12	3P	$\hat{a} = 0.09480, \hat{b} = 0.000000315, \hat{\beta} = 8.18127,$ $\hat{c} = 0.01147, \hat{N} = 706.1525$
13	DPF 1	$\hat{a} = 209.7896, \hat{b} = 0.00754,$ $\hat{c} = 12.9492, \hat{h} = 31.45712$
14	DPF 2	$\hat{a} = 206.8508, \hat{b} = 0.00316,$ $\hat{c} = 5.22432, \hat{h} = 28.1848$
15	UDPF	$\hat{b} = 0.01562, \hat{\alpha} = 0.49790,$ $\hat{\beta} = 0.49731, \hat{N} = 472.0649$
16	DNN	Hidden layer = 4, Optimizer = Adam, Epoch = 100
17	RNN	Difference = 1, Optimizer = Adam, Epoch = 100
18	LSTM	Difference = 1, Optimizer = Adam, Epoch = 100
19	GRU	Difference = 1, Optimizer = Adam, Epoch = 100
20	Proposed model	$\hat{b} = 0.52$, Hidden layer = 4, Optimizer = Adam, Epoch = 100

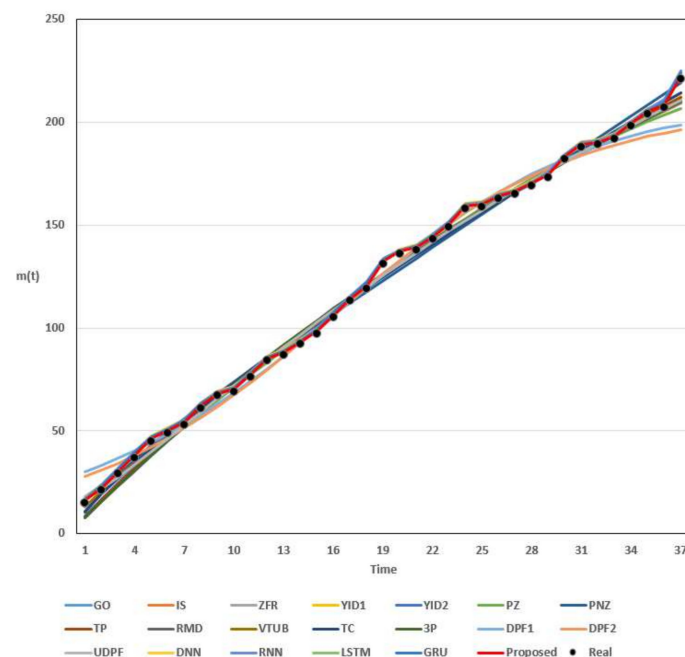
**Figure 3.** Prediction of all models for Dataset 1.

Table 5 presents the results for the 10 criteria based on the estimated values. The proposed deep-learning software reliability model had the best results for MSE, PRR, PP, R^2 , RMSPE, MAE, MEOP, TS, and PC, with values of 1.5764, 0.0173, 0.0191, 0.9995, 1.255, 1.255, 1.2197, 0.9968, and 8.7077, respectively, for all criteria. The recurrent neural network types showed good results, as did the LSTM. In addition, among the previous SRGMs, PZ showed the best results in MSE, R^2 , RMSPE, MAE, MEOP, TS, and PC with 3.5923, 0.9977, 2.6764, 2.1584, 2.0967, 2.0933, and 33.9348, respectively, compared with other SRGM, and Vtub showed good results in PRR and PP (0.0578 and 0.0507, respectively). In terms of prediction, preSSE was the best model for the proposed deep-learning software reliability model with a result of 4.728, followed by LSTM with 12.845. The best model among the NHPP SRGMs was TC with a result of 52.177.

Table 5. Comparison of all criteria from Dataset 1.

No.	Model	MSE	MAE	PRR	PP	R2	RMSPE	MEOP	TS	PC	preSSE
1	GO	13.9920	3.1187	1.0978	0.4134	0.9954	3.7951	3.0296	2.9696	45.8249	138.994
2	IS	14.0172	3.1232	1.1019	0.4145	0.9954	3.7983	3.0339	2.9723	45.8555	145.058
3	ZFR	13.9948	3.1191	1.0976	0.4133	0.9954	3.7955	3.0300	2.9699	45.8282	140.045
4	YID 1	13.6219	3.1393	0.9056	0.3590	0.9955	3.7452	3.0496	2.9301	45.3691	73.479
5	YID 2	13.3450	3.1495	0.7134	0.3016	0.9956	3.7073	3.0595	2.9002	45.0200	52.222
6	PZ	6.9523	2.1584	0.0684	0.0789	0.9977	2.6764	2.0967	2.0933	33.9348	232.611
7	PNZ	14.2497	3.1105	0.2259	0.1308	0.9953	3.8316	3.0216	2.9969	46.1351	70.068
8	TP	13.1042	3.0831	0.8088	0.3311	0.9957	3.6736	2.9950	2.8739	44.7105	79.662
9	RMD	13.9770	3.1179	1.0934	0.4122	0.9954	3.7931	3.0288	2.9681	45.8067	137.644
10	Vtub	7.9408	2.3265	0.0578	0.0507	0.9974	2.8602	2.2601	2.2372	36.1948	109.486
11	TC	10.9547	2.8162	0.2133	0.1326	0.9964	3.3595	2.7357	2.6276	41.6646	52.177
12	3P	13.7490	3.1007	1.0277	0.3939	0.9955	3.7623	3.0121	2.9437	45.5270	120.537
13	DPF 1	20.3656	3.0736	0.4568	1.4588	0.9933	4.5774	2.9857	3.5827	52.2060	672.499
14	DPF 2	17.4659	3.2987	0.3703	0.9978	0.9942	4.2419	3.2044	3.3179	49.5949	886.313
15	UDPF	8.7018	2.4885	0.0855	0.0947	0.9971	2.9942	2.4174	2.3419	37.7505	113.299
16	DNN	5.9825	2.4459	0.0600	0.0726	0.9980	2.4459	2.3760	1.9418	31.3809	17.932
17	RNN	5.6534	2.3462	0.0606	0.0734	0.9981	2.3787	2.2772	1.9312	29.5519	36.121
18	LSTM	2.5088	1.5576	0.0285	0.0324	0.9991	1.5847	1.5118	1.2865	16.1464	12.845
19	GRU	3.2422	1.7118	0.0364	0.0418	0.9989	1.8033	1.6614	1.4625	20.3779	17.703
20	Proposed model	1.5764	1.2555	0.0173	0.0191	0.9995	1.2555	1.2197	0.9968	8.7077	4.728

4.4. Results of Dataset 2

Table 6 lists the estimated parameter values of the SRGM assuming an NHPP using Dataset 2 and the structure of the software reliability model using deep learning. It also shows the estimated parameter values and structure of the proposed deep-learning software reliability model. The parameter b of the proposed software reliability model was estimated to be 0.76. The structure of the proposed deep-learning software reliability model is the same as that for Dataset 1. Figure 4 shows the computed estimates of $m(t)$ for each model and deep-learning structure.

Table 7 presents the results for the 10 criteria based on the estimated values. The proposed deep-learning software reliability model had the best results of 34.9817, 0.9995, 5.9145, 5.9145, 5.9145, 5.9145, 5.9145, 5.8070, 0.9571, and 96.9617 for the MSE, R^2 , RMSPE, MAE, MEOP, TS, and PC criteria, respectively, for seven of the ten criteria. Among the recurrent neural networks, LSTM showed the second-best results of 265.2696, 1.6538, 0.9958, 16.2892, 16.1755, 15.8759, 2.6661, and 148.8709 for MSE, PP, R^2 , RMSPE, MAE, MEOP, TS, and PC, while RNN showed the best results in PRR. In addition, among the previously proposed SRGMs, PNZ exhibited the best results for all the criteria. For prediction, preSSE yielded the best result of 133.128 for RMD, followed by 146.460 and 174.911 for GO and the proposed model, respectively.

Table 6. Parameter estimation and structure of models from Dataset 2.

No.	Model	Estimation and Structure
1	GO	$\hat{a} = 1200.6573, \hat{b} = 0.02594$
2	IS	$\hat{a} = 1090.2373, \hat{b} = 0.03435, \hat{\beta} = 0.21312$
3	ZFR	$\hat{a} = 395.6997, \hat{b} = 0.01072, \hat{\alpha} = 0.13188,$ $\hat{\beta} = 1.29491, \hat{c} = 1.07014, \hat{p} = 1.06829$
4	YID 1	$\hat{a} = 735.9191, \hat{b} = 0.04609, \hat{\alpha} = 0.00821$
5	YID 2	$\hat{a} = 560.7125, \hat{b} = 0.06328, \hat{\alpha} = 0.01755$
6	PZ	$\hat{a} = 0.02306, \hat{b} = 0.00172, \hat{\alpha} = 0.00172,$ $\hat{\beta} = 0.02247, \hat{c} = 0.92125$
7	PNZ	$\hat{a} = 249.3358, \hat{b} = 0.44707,$ $\hat{\alpha} = 0.05410, \hat{\beta} = 9.86286$
8	TP	$\hat{a} = 16.13260, \hat{b} = 0.01055, \hat{\alpha} = 0.07844, \hat{\beta} = 0.00020,$ $\hat{c} = 0.86459, \hat{p} = 1.07014, \hat{q} = 1.06829$
9	RMD	$\hat{a} = 53.30543, \hat{b} = 0.02660,$ $\hat{\alpha} = 22.98703, \hat{\beta} = 0.01071$
10	Vtub	$\hat{a} = 8.59529, \hat{b} = 0.50025, \hat{\alpha} = 0.00184,$ $\hat{\beta} = 17.01916, \hat{N} = 39, 113.5827$
11	TC	$\hat{a} = 0.07169, \hat{b} = 1.02720, \hat{\alpha} = 0.32387,$ $\hat{\beta} = 2.09164, \hat{N} = 3111.1052$
12	3P	$\hat{a} = 1.44644, \hat{b} = 0.00468, \hat{\beta} = 2.31208,$ $\hat{c} = 30.67715, \hat{N} = 1663.2977$
13	DPF 1	$\hat{a} = 912.3438, \hat{b} = 0.00016,$ $\hat{c} = 0.56449, \hat{h} = 145.4555$
14	DPF 2	$\hat{a} = 955.6838, \hat{b} = 0.00280,$ $\hat{c} = 34.39205, \hat{h} = 177.6015$
15	UDPF	$\hat{b} = 2.75584, \hat{\alpha} = 27.11226,$ $\hat{\beta} = 3.15487, \hat{N} = 1490.17003$
16	DNN	Hidden layer = 4, Optimizer = Adam, Epoch = 100
17	RNN	Difference = 1, Optimizer = Adam, Epoch = 100
18	LSTM	Difference = 1, Optimizer = Adam, Epoch = 100
19	GRU	Difference = 1, Optimizer = Adam, Epoch = 100
20	Proposed model	$\hat{b} = 0.76$, Hidden layer = 4, Optimizer = Adam, Epoch = 100

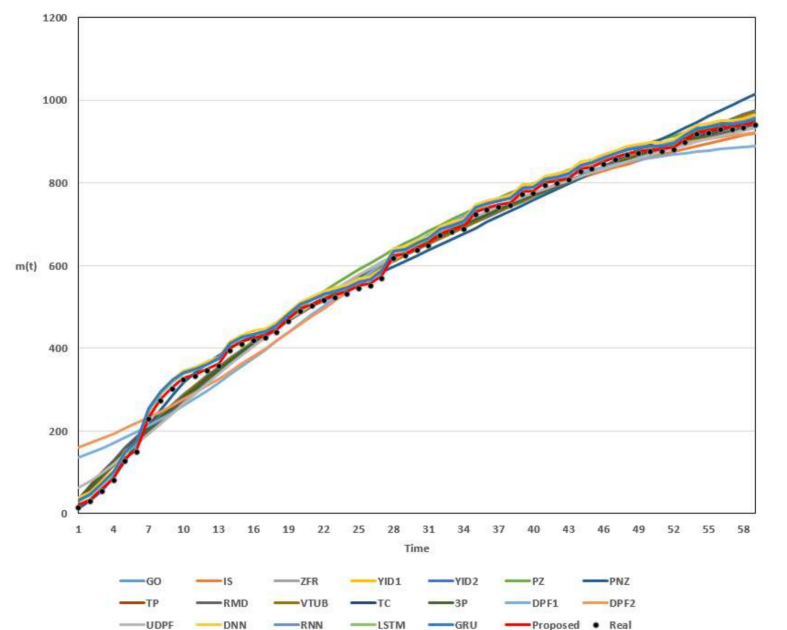
**Figure 4.** Prediction of all models for Dataset 2.

Table 7. Comparison of all criteria from Dataset 2.

No.	Model	MSE	PRR	PP	R2	RMSPE	MAE	MEOP	TS	PC	preSSE
1	GO	417.7167	1.0046	3.0998	0.9936	20.6300	15.4403	15.1595	3.3075	163.9212	146.460
2	IS	473.8190	0.9943	3.0358	0.9927	21.9712	17.7702	17.4471	3.5226	167.3238	2358.03
3	ZFR	423.4813	1.0000	3.0371	0.9935	20.7719	15.5361	15.2536	3.3302	164.2912	190.859
4	YID 1	369.1211	1.0917	4.1090	0.9943	19.3920	14.9690	14.6969	3.1091	160.5819	2702.19
5	YID 2	369.8950	1.1528	4.7333	0.9943	19.4118	14.9997	14.7270	3.1124	160.6384	3432.36
6	PZ	753.2823	1.0465	3.3918	0.9884	27.6128	24.8414	24.3897	4.4415	179.8414	2233.55
7	PNZ	301.2450	0.1147	0.1115	0.9954	17.5194	14.3911	14.1294	2.8087	155.0954	17,417.7
8	TP	365.2466	1.1372	4.5168	0.9944	19.2898	14.4809	14.2176	3.0928	160.2969	1390.88
9	RMD	416.1812	1.0054	3.1091	0.9936	20.5921	15.3974	15.1175	3.3014	163.8217	133.128
10	Vtub	306.2931	0.8128	2.1844	0.9953	17.6654	13.7406	13.4908	2.8322	155.5441	3147.84
11	TC	367.9481	1.0283	3.4991	0.9943	19.3617	14.7116	14.4441	3.1042	160.4959	487.299
12	3P	380.4131	1.0555	3.6816	0.9941	19.6872	14.9196	14.6484	3.1563	161.3954	245.064
13	DPF 1	1686.1575	2.6213	86.7665	0.9740	41.4468	30.7013	30.1431	6.6451	201.5971	10,674.2
14	DPF 2	1947.4844	2.7591	125.063	0.9700	44.5237	29.2403	28.7087	7.1415	205.4874	1217.06
15	UDPF	618.8834	1.5780	14.0278	0.9905	25.1109	18.8242	18.4820	4.0258	174.5352	577.008
16	DNN	521.7809	0.8215	3.3469	0.9920	22.8425	22.8425	22.4272	3.6966	169.9272	2606.87
17	RNN	266.8270	0.5646	1.7034	0.9958	16.3355	16.3008	15.9989	2.6739	149.0260	1192.65
18	LSTM	265.2696	0.5696	1.6538	0.9958	16.2892	16.1755	15.8759	2.6661	148.8709	1034.38
19	GRU	298.8604	0.6227	1.8510	0.9953	17.2917	17.0695	16.7534	2.8299	152.0305	978.347
20	Proposed model	34.9817	0.1325	0.2242	0.9995	5.9145	5.9145	5.807	0.9571	96.9617	174.911

5. Sensitivity Analysis

Results of Variation for Changes in Parameters

The software reliability model estimates the number of failures $m(t)$ by estimating a parameter that contains mathematical assumptions. Whenever the parameters change, the $m(t)$ in the software reliability model also changes; therefore, it is extremely sensitive to small changes in the estimated parameters [66,67]. To evaluate the sensitivity of the model to changes in parameters, the parameters of the proposed model were varied by -10% , -5% , 5% , and 10% . Figure 5a shows the estimates of the time-varying parameters for Dataset 1. From -10% to 10% , the change in parameter b did not cause much change in $m(t)$, so the last eight time points are presented with a zoomed-in graph. Figure 5b shows the estimates of the time-varying parameters for Dataset 2, with the results for the last nine time points zoomed in because they are similar to those for Dataset 1.

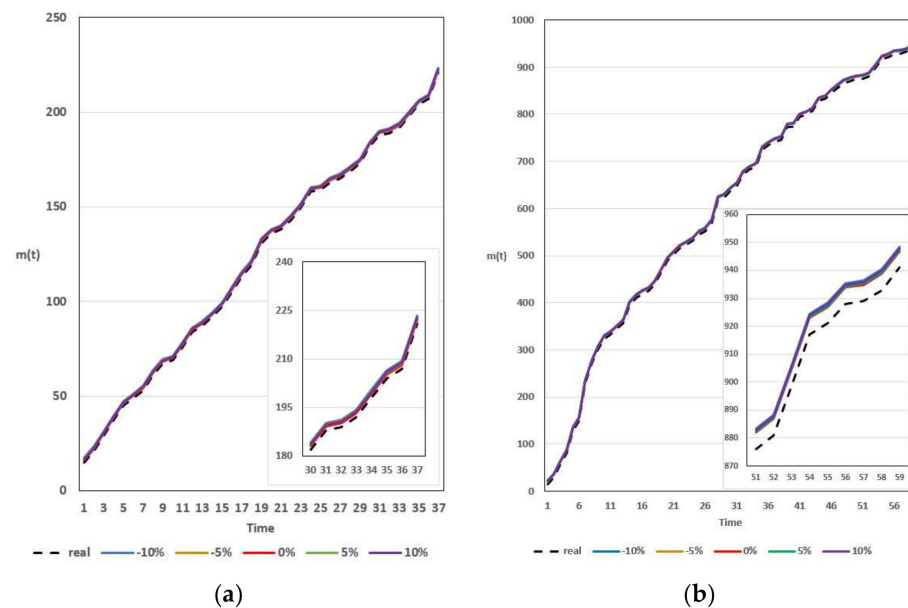


Figure 5. (a) Predicted change in parameter b for Dataset 1 and (b) predicted change in parameter b for Dataset 2.

Table 8 and Figure 6 show the results of comparing the MSE values based on the changed estimates for Datasets 1 and 2 with the MSE values of the proposed deep-learning software reliability model for a numerical comparison. The comparison criterion was the MSE of the proposed deep-learning software reliability model, and the MSE of the changed parameters was divided by that of the proposed deep-learning software reliability model. The results for Dataset 1 show that the MSE increases by factors of 3.378 (−10%) and 2.459 (−5%) as b becomes smaller, and by factors of 1.670 (5%) and 2.054 (10%) as b becomes larger compared with the previously estimated values. The results for Dataset 2 show that the MSE increases by factors of 1.587 (−10%) and 1.258 (−5%) as b becomes smaller, and by factors of 1.119 (5%) and 1.350 (10%) as b becomes larger compared with the original estimate. This demonstrates that the MSE increases more for smaller than for larger values of the estimated fault detection rate b , indicating a poor estimate. This is because Datasets 1 and 2 have a high percentage of failures that either decrease in number or recur, even as the number of time points increases.

Table 8. Comparison of MSE by parameter b change.

Data	−10%	−5%	5%	10%
Dataset 1	3.3781	2.4595	1.6995	2.0537
Dataset 2	1.5870	1.2583	1.1185	1.3501

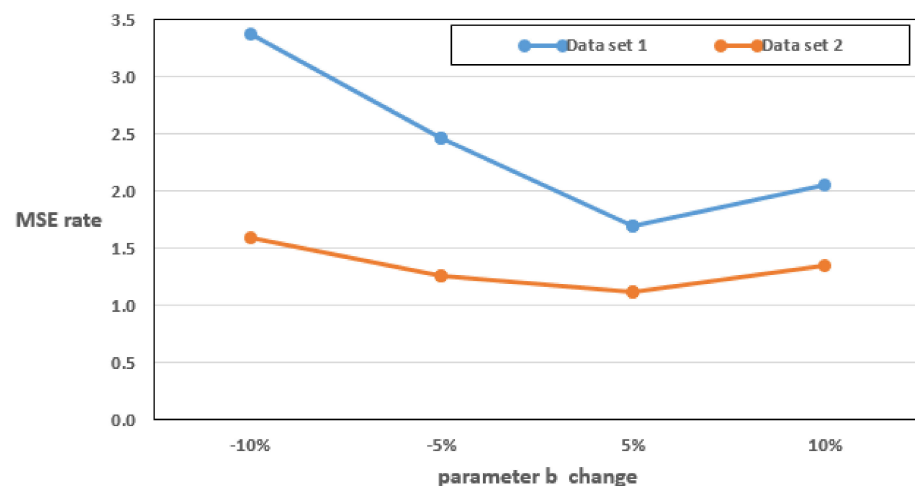


Figure 6. Comparison of MSE by parameter b change.

6. Discussions

A deep-learning software reliability model was developed by combining an existing software reliability model developed from a parametric perspective with the software reliability model using deep learning developed from a nonparametric perspective. We compared with traditional software reliability models and demonstrated the superiority of our proposed model using two datasets. However, since the proposed deep learning software reliability model only depends on parameter b , we believe that additional assumptions about uncertain operating environments or dependent failures as described in Section 3.1 are needed. We believe that this will allow us to manage software failures for more general situations.

7. Conclusions

In this study, we developed a deep learning software reliability model to replace the existing software reliability model because software failures are similar in shape to sigmoid functions, and we compared the traditional SRGMs and the software reliability model using deep learning. The first dataset had the best results on 10 criteria, and the second dataset had the best results on all 10 criteria except PRR, PP, and preSSE. The proposed

deep-learning software reliability model did not show much difference from the actual value as the time point became larger than the existing software reliability model, and it provided a better fit than the software reliability model using deep learning. In addition, the change in the estimation value according to the change in parameter b of the proposed model was examined, and the smaller the parameter b , which is the failure detection rate, the worse the estimation value. Even when b was larger, the estimation value was good compared with a smaller one. It is believed that this is because the number of failures did not decrease owing to the characteristics of the data used.

This study is meaningful because the previously developed software reliability model using deep learning does not provide a mathematical basis and relies solely on data, whereas the proposed deep-learning software reliability model is based on data and fits the mathematical basis. Therefore, further research is planned on other models that transform the software reliability model into an activation function based on mathematical and statistical assumptions that include multiple parameters rather than a single one.

Author Contributions: Conceptualization, H.P.; funding acquisition, I.H.C.; software and simulation, Y.S.K.; sensitivity analysis, I.H.C.; writing—original draft, Y.S.K.; writing—review and editing, I.H.C. and H.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Science Research Program of the National Research Foundation of Korea (NRF), funded by the Ministry of Education (NRF-2021R1F1A1048592).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are available in a publicly accessible repository.

Acknowledgments: This study was supported by the National Research Foundation of Korea (NRF).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hong, Y.; Lian, J.; Xu, L.; Min, J.; Wang, Y.; Freeman, L.J.; Deng, X. Statistical perspectives on reliability of artificial intelligence systems. *Qual. Eng.* **2023**, *35*, 56–78. [\[CrossRef\]](#)
2. Bastani, F.; Chen, I.R. Assessment of the reliability of AI programs. *Int. J. Artif. Intell. Tools* **1993**, *2*, 235–247. [\[CrossRef\]](#)
3. Sheptunov, S.A.; Larionov, M.V.; Suhanova, N.V.; Il'ya, S.K.; Alshynbaeva, D.A. Optimization of the complex software reliability of control systems. In Proceedings of the 2016 IEEE Conference on Quality Management, Transport and Information Security, Information Technologies (IT&MQ&IS), Nalchik, Russia, 4–11 October 2016; pp. 189–192.
4. Ryan, M. In AI we trust: Ethics, artificial intelligence, and reliability. *Sci. Eng. Ethics* **2020**, *26*, 2749–2767. [\[CrossRef\]](#)
5. Martyushev, N.V.L.; Malozyomov, B.V.; Sorokova, S.N.; Efremenkova, E.A.; Valuev, D.V.; Qi, M. Review Models and Methods for Determining and Predicting the Reliability of Technical Systems and Transport. *Mathematics* **2023**, *11*, 3317. [\[CrossRef\]](#)
6. Antosz, K.; Machado, J.; Mazurkiewicz, D.; Antonelli, D.; Soares, F. Systems Engineering: Availability and Reliability. *Appl. Sci.* **2022**, *12*, 2504. [\[CrossRef\]](#)
7. Jelinski, Z.; Moranda, P.B.; Freiberger, W. Statistical computer performance evaluation. *Softw. Reliab. Res.* **1972**, 465–484.
8. Goel, A.L.; Okumoto, K. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Trans. Reliab.* **1979**, *28*, 206–211. [\[CrossRef\]](#)
9. Kumar, P.; Singh, Y. An empirical study of software reliability prediction using machine learning techniques. *Int. J. Syst. Assur. Eng. Manag.* **2012**, *3*, 194–208. [\[CrossRef\]](#)
10. Jaiswal, A.; Malhotra, R. Software reliability prediction using machine learning techniques. *Int. J. Syst. Assur. Eng. Manag.* **2018**, *9*, 230–244. [\[CrossRef\]](#)
11. Misra, D. Mish: A Self Regularized Non-Monotonic Neural Activation Function. *arXiv* **2019**, arXiv:1908.08681.
12. Gulcehre, C.; Moczulski, M.; Denil, M.; Bengio, Y. Noisy activation functions. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 3059–3068.
13. Hossain, S.A.; Dahiya, R.C. Estimating the parameters of a non-homogeneous Poisson-process model for software reliability. *IEEE Trans. Reliab.* **1993**, *42*, 604–612. [\[CrossRef\]](#)
14. Yamada, S.; Ohba, M.; Osaki, S. S-shaped reliability growth modeling for software fault detection. *IEEE Trans. Reliab.* **1983**, *32*, 475–484. [\[CrossRef\]](#)
15. Ohba, M. Inflexion S-shaped software reliability growth models. In *Stochastic Models in Reliability Theory*; Osaki, S., Hatoyama, Y., Eds.; Springer: Berlin, Germany, 1984; pp. 144–162.

16. Zhang, X.M.; Teng, X.L.; Pham, H. Considering fault removal efficiency in software reliability assessment. *IEEE Trans. Syst. Man. Cybern. Part Syst. Hum.* **2003**, *33*, 114–120. [\[CrossRef\]](#)
17. Yamada, S.; Tokuno, K.; Osaki, S. Imperfect debugging models with fault introduction rate for software reliability assessment. *Int. J. Syst. Sci.* **1992**, *23*, 2241–2252. [\[CrossRef\]](#)
18. Pham, H.; Zhang, X. An NHPP software reliability models and its comparison. *Int. J. Reliab. Qual. Saf. Eng.* **1997**, *4*, 269–282. [\[CrossRef\]](#)
19. Pham, H.; Nordmann, L.; Zhang, X. A general imperfect software debugging model with S-shaped fault detection rate. *IEEE Trans. Reliab.* **1999**, *48*, 169–175. [\[CrossRef\]](#)
20. Teng, X.; Pham, H. A new methodology for predicting software reliability in the random field environments. *IEEE Trans. Reliab.* **2006**, *55*, 458–468. [\[CrossRef\]](#)
21. Roy, P.; Mahapatra, G.S.; Dey, K.N. An NHPP software reliability growth model with imperfect debugging and error generation. *Int. J. Reliab. Qual. Saf. Eng.* **2014**, *21*, 1–3. [\[CrossRef\]](#)
22. Yang, B.; Xie, M. A study of operational and testing reliability in software reliability analysis. *Reliab. Eng. Syst. Saf.* **2000**, *70*, 323–329. [\[CrossRef\]](#)
23. Huang, C.Y.; Kuo, S.Y.; Lyu, M.R.; Lo, J.H. Quantitative software reliability modeling from testing from testing to operation. In Proceedings of the International Symposium on Software Reliability Engineering, IEEE, Los Alamitos, CA, USA, 8–11 October 2000; pp. 72–82.
24. Pham, H. A new software reliability model with Vtub-Shaped fault detection rate and the uncertainty of operating environments. *Optimization* **2014**, *63*, 1481–1490. [\[CrossRef\]](#)
25. Chang, I.H.; Pham, H.; Lee, S.W.; Song, K.Y. A testing-coverage software reliability model with the uncertainty of operation environments. *Int. J. Syst. Sci. Oper. Logist.* **2014**, *1*, 220–227.
26. Song, K.Y.; Chang, I.H.; Pham, H. A Three-parameter fault-detection software reliability model with the uncertainty of operating environments. *J. Syst. Sci. Syst. Eng.* **2017**, *26*, 121–132. [\[CrossRef\]](#)
27. Huang, C.Y.; Kuo, S.Y.; Lyu, M.R. An assessment of testing-effort dependent software reliability growth models. *IEEE Trans. Reliab.* **2007**, *56*, 198–211. [\[CrossRef\]](#)
28. Ahmad, N.; Khan, M.G.; Rafi, L.S. A study of testing-effort dependent inflection S-shaped software reliability growth models with imperfect debugging. *Int. J. Qual. Reliab. Manag.* **2010**, *27*, 89–110. [\[CrossRef\]](#)
29. Kim, Y.S.; Song, K.Y.; Pham, H.; Chang, I.H. A software reliability model with dependent failure and optimal release time. *Symmetry* **2022**, *14*, 343. [\[CrossRef\]](#)
30. Lee, D.H.; Chang, I.H.; Pham, H. Software reliability model with dependent failures and SPRT. *Mathematics* **2020**, *8*, 1366. [\[CrossRef\]](#)
31. Lee, D.H.; Chang, I.H.; Pham, H. Software reliability growth model with dependent failures and uncertain operating environments. *Appl. Sci.* **2022**, *12*, 12383. [\[CrossRef\]](#)
32. Cai, K.Y.; Cai, L.; Wang, W.D.; Yu, Z.Y.; Zhang, D. On the neural network approach in software reliability modeling. *J. Syst. Softw.* **2001**, *58*, 47–62. [\[CrossRef\]](#)
33. Miyamoto, S.; Tamura, Y.; Yamada, S. Reliability assessment tool based on deep learning and data preprocessing for OSS. *Amer. J. Oper. Res.* **2022**, *12*, 111–125. [\[CrossRef\]](#)
34. Oveisi, S.; Moeini, A.; Mirzaei, S.; Farsi, M.A. LSTM encoder-decoder dropout model in software reliability prediction. *Int. J. Reliab. Risk Saf. Theory Appl.* **2021**, *4*, 1–12. [\[CrossRef\]](#)
35. Raamesh, L.; Jothi, S.; Radhika, S. Enhancing software reliability and fault detection using hybrid brainstorm optimization-based LSTM model. *IETE J. Res.* **2022**, 1–15. [\[CrossRef\]](#)
36. Wu, C.Y.; Huang, C.Y. A study of incorporation of deep learning into software reliability modeling and assessment. *IEEE Trans. Reliab.* **2021**, *70*, 1621–1640. [\[CrossRef\]](#)
37. Batool, I.; Khan, T.A. Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review. *Comput. Electr. Eng.* **2022**, *100*, 107886. [\[CrossRef\]](#)
38. Sreekanth, N.; Rama, D.J.; Shukla, A.; Mohanty, D.K.; Srinivas, A.; Rao, G.N.; Alam, A.; Gupta, A. Evaluation of estimation in software development using deep learning-modified neural network. *Appl. Nanosci.* **2023**, *13*, 2405–2417. [\[CrossRef\]](#)
39. Bhuyan, M.K.; Mohapatra, D.P.; Sethi, S. Software Reliability Prediction using Fuzzy Min-Max Algorithm and Recurrent Neural Network Approach. *Int. J. Electr. Comput. Eng.* **2016**, *6*, 1929–1938.
40. Mittelman, R. Time-series modeling with undecimated fully convolutional neural networks. *arXiv* **2015**, arXiv:1508.00317.
41. Pan, C.; Lu, M.; Xu, B.; Gao, H. An improved CNN model for within-project software defect prediction. *Appl. Sci.* **2019**, *9*, 2138. [\[CrossRef\]](#)
42. Karunanithi, N.; Whitley, D.; Malaiya, Y.K. Using neural networks in reliability prediction. *IEEE Softw.* **1992**, *9*, 53–59. [\[CrossRef\]](#)
43. Tamura, Y.; Yamada, S. Software reliability model selection based on deep learning with application to the optimal release problem. *J. Ind. Eng. Manag. Sci.* **2016**, *1*, 43–58. [\[CrossRef\]](#)
44. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to sequence learning with neural networks. *Adv. Neural Inf. Process. Syst.* **2014**, *27*, 3104–3112.
45. Schmidhuber, J.; Hochreiter, S. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780.

46. Chen, L.; Zheng, J.; Okamura, H.; Dohi, T. Software reliability prediction through encoder-decoder recurrent neural networks. *Int. J. Math. Eng. Manag. Sci.* **2022**, *7*, 325.
47. Che, Z.; Purushotham, S.; Cho, K.; Sontag, D.; Liu, Y. Recurrent neural networks for multivariate time series with missing values. *Sci. Rep.* **2018**, *8*, 6085. [[CrossRef](#)] [[PubMed](#)]
48. Munir, H.S.; Ren, S.; Mustafa, M.; Siddique, C.N.; Qayyum, S. Attention based GRU-LSTM for software defect prediction. *PLoS ONE* **2021**, *16*, e0247444. [[CrossRef](#)]
49. Jónás, T. Sigmoid functions in reliability based management. *Period. Polytech. Soc. Manag. Sci.* **2007**, *15*, 67–72. [[CrossRef](#)]
50. Kyurkchiev, N. A note on a hypothetical piecewise smooth sigmoidal growth function: Reaction network analysis, applications. *Int. J. Differ. Equat. Appl.* **2022**, *21*, 1–17.
51. Lu, K.; Ma, Z. A modified whale optimization algorithm for parameter estimation of software reliability growth models. *J. Algorithms Comput. Technol.* **2021**, *15*, 17483026211034442. [[CrossRef](#)]
52. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980.
53. Misra, P.N. Software reliability analysis. *IBM Syst. J.* **1983**, *22*, 262–270. [[CrossRef](#)]
54. Wang, J. Model of open source software reliability with fault introduction obeying the generalized pareto distribution. *Arab. J. Sci. Eng.* **2021**, *46*, 3981–4000. [[CrossRef](#)]
55. Inoue, S.; Yamada, S. Discrete software reliability assessment with discretized NHPP models. *Comput. Math. Appl.* **2006**, *51*, 161–170. [[CrossRef](#)]
56. Chiu, K.C.; Huang, Y.S.; Lee, T.Z. A study of software reliability growth from the perspective of learning effects. *Reliab. Eng. Syst. Saf.* **2008**, *93*, 1410–1421. [[CrossRef](#)]
57. Haque, M.A.; Ahmad, N. An effective software reliability growth model. *Saf. Reliab.* **2021**, *40*, 1–12. [[CrossRef](#)]
58. Zhao, J.; Liu, H.W.; Cui, G.; Yang, X.Z. Software reliability growth model with change-point and environmental function. *J. Syst. Softw.* **2006**, *79*, 1578–1587. [[CrossRef](#)]
59. Cameron, A.C.; Windmeijer, F.A. An R-squared measure of goodness of fit for some common nonlinear regression models. *J. Econom.* **1997**, *77*, 329–342. [[CrossRef](#)]
60. Pillai, K.; Nair, V.S. A model for software development effort and cost estimation. *IEEE Trans. Softw. Eng.* **1997**, *23*, 485–497. [[CrossRef](#)]
61. Anjum, M.; Haque, M.A.; Ahmad, N. Analysis and ranking of software reliability models based on weighted criteria value. *Int. J. Inform. Tech. Comp. Sci.* **2013**, *2*, 1–14. [[CrossRef](#)]
62. Sharma, K.; Garg, R.; Nagpal, C.K.; Garg, R.K. Selection of optimal software reliability growth models using a distance based approach. *IEEE Trans. Reliab.* **2010**, *59*, 266–276. [[CrossRef](#)]
63. Selvakumar, K.; Lokesh, S. Retracted Article: The prediction of the lifetime of the new coronavirus in the USA using mathematical models. *Soft Comput.* **2021**, *25*, 10575–10594. [[CrossRef](#)]
64. Dhaka, R.; Pachauri, B.; Jain, A. Two-dimensional software reliability model with considering the uncertainty in operating environment and predictive analysis. In *Data Engineering for Smart Systems*; Springer: Singapore, 2022; pp. 57–69.
65. Musa, J.D.; Iannino, K.; Okumoto, K. *Software Reliability Measurement Prediction Application*; McGraw-Hill: New York, NY, USA, 2006.
66. Lo, J.H.; Huang, C.Y.; Chen, Y.; Kuo, S.Y.; Lyu, M.R. Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure. *J. Syst. Softw.* **2005**, *76*, 3–13. [[CrossRef](#)]
67. Li, X.; Xie, M.; Ng, S.H. Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points. *Appl. Math. Model.* **2010**, *34*, 3560–3570. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.