Applied Mathematics, Modeling and Computer Simulation
C.-H. Chen et al. (Eds.)

doi:10.3233/ATDE221036

Confidence Evaluation Algorithm of Aerospace Software Based on Deep Auto-Encoding Network

Ben ZHANG, Tianwen YAO, Zhi ZHANG, Jun PENG[1]
Systems Engineering Institute of Sichuan Aerospace, 610100, Sichuan, China

Abstract. In this paper, a software confidence evaluation model based on software fault tree analysis and deep auto-encoding network is established to calculate the confidence of rocket fire control software. First, a fault tree is established accroding to the common architecture of rocket fire control software, and a metric set for rocket fire control software is constructed. Second, an autoencoder is used to perform feature dimension reduction and confidence estimation on historical data, so as to calculate top-event occurrence rate, that is, the probability of the software failure. Finally, an example analysis of the propoded method was carried out, and compared the rusults with the results of the traditional exponential model. The case study show that the confidence evaluation model established in this paper is effective, which can be used for the confidence evaluation of rocket fire control software in the engineering development process. And it can also be extended to other types of software.

Keywords. Aerospace software, fault tree analysis, deep auto-encoding network, confidence evaluation

1. Introduction

Software confidence refers to the ability of the software to run for a certain period of time without causing system failure under certain conditions. There are three types of traditional models of software confidence: exponential model, non-exponential model and Bayesian model[1].Taking the exponential model as an example, the software confidence is described by the mean time before system failure (MTTF):

$$\mathrm{MTTF} = \int_0^\infty \mathrm{R(t)dt} \tag{1}$$

where R(t) is the confidence function with the software running time t as the independent variable, which can be expressed as:

$$\mathrm{R(t)} = \exp\left[-\int_0^t \mathrm{z(x)dx}\right] \tag{2}$$

In the formula, z(x) is the failure rate function with the software running time x as the independent variable. The description of failure rate function includes Schncidewind model, Jelinski-Moranda model and generalized exponential model. Therefore, when evaluating software confidence, it is necessary to select an appropriate evaluation model,

[1] Corresponding Author, Jun PENG, Systems Engineering Institute of Sichuan Aerospace, 610100, Sichuan, China; E-mail: 15988124520@163.com.

and then estimate the parameters based on the collected software failure data[2]. Data collection is the key to software confidence evaluation. Software confidence data mainly includes failure time (the specific time when a failure occurs) and failure count (the number of failures that occur within a period of time), etc. The collection of these data needs to be established on the basis of long-term test and program trial run. The longer the test and trial run time, the more sufficient the data could be collected, and the more accurate the evaluation of the software confidence.

Rocket fire control software is mainly used for pre-launch testing and execution of the launch process, and its confidence plays a key role in the success of the launch mission. However, in the process of engineering development, due to various factors such as tight engineering schedules and late matching of hardware resources, fire control software often does not have sufficient trial run time to collect confidence data, so that it is often faces the situation that up with the case that the software is directly summoned to the "battlefield" once the developer testing is completed. In recent years, with the further development and extensive application of machine learning, more and more researches use machine learning methods to conduct research on software defect prediction. Through machine learning algorithm to train software historical data, the accuracy of software defect prediction is getting higher and higher[3].

In this paper, based on the characteristics of rocket fire control software, a software confidence evaluation model is established based on software fault tree analysis (SFTA) and deep auto-encoding network. The model can be used to to calculate software confidence, which can effectively solve the problem of insufficient software trial run time in the development process.

2. Confidence Evaluation Model for Fire Control Software

2.1. Methodology Overview

Software fault tree analysis can be used to explore all the causes and combinations of causes that cause undesired system failures or catastrophic hazardous events. The tree structure is refined from top to bottom to create a fault tree from top events to bottom events. When basic data are available, the probability of occurrence of top events and other quantitative indicators available[4].

The process of creating a fault tree of a software is specifically as follows: First, takeing the fault of software as the top event T, the middle subprogram module fault of control process as the intermediate event E_i, and the software unit as the bottom event X_i. Second, connecting the fault modes of the software units through logic gates to form a software fault tree. Then the fault tree can be used to evaluate the confidence of the software.

Rocket fire control software is essentially a communication software, which completes the function of sending and receiving data according to corresponding instructions of the operator. It is usually a three-layer architecture, and the uppermost layer is the functional modules of the application layer. These functional modules use one or more communication protocols to perform operations such as data analysis and framing. The middle layer is the communication protocol distribution layer, and the bottom layer a layer that responsible for the data stream transmission of the hardware communication interface.

For typical rocket fire control software, figure 1 shows the fault tree with software fault as the top event. The event T is the fault of the fire control software. E_1, E_2, ... $E_{(n-2)}$ are each function implemented by the fire control software. X_3, X_4, ..., X_n are application layer's units of these functions. X_1 is the bottom layer's data transmission unit, and X_2 is the middle layer's protocol distribution unit. Using the minimum cut set method, the fault tree is expressed as:

$$T = X_1 + X_2 + \cdots + X_n \tag{3}$$

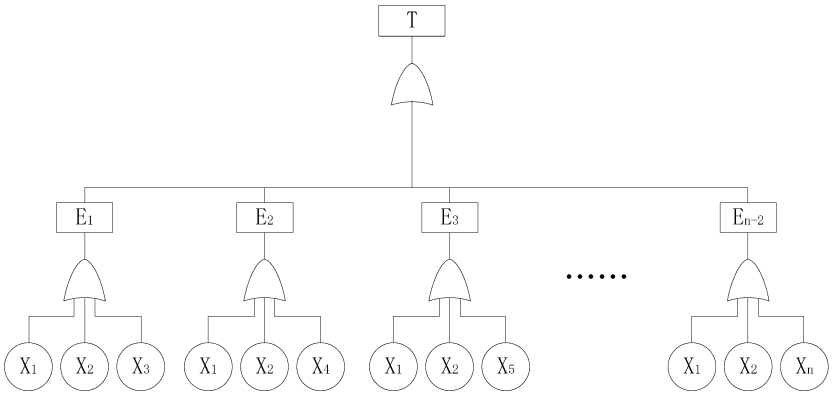

Figure 1. Fault tree of rocket fire control software

Since each event Xi (i=1,2,...,n) is finally connected to the event T through a logical OR gate, the probability P of the occurrence of event T can be calculate though probabilities of the occurrence of events X1, X2, … Xn :

$$P = 1 - (1 - P_1)(1 - P_2) \cdots (1 - P_n) \tag{4}$$

where P_i is the occurrence probability of event X_i. According to formula (4), the key to calculating the probability of software failure is to obtain the probability of occurrence of all bottom events.

2.2. Convolution Kernel Improvement for Insulator Cracks

Autoencoder[5] is an unsupervised learning algorithm. Its function is to perform feature recognition and feature optimization on the training data. As shown in figure 2, the algorithm uses the back propagation algorithm to make the target value as equal to the input value as possible. Through continuous training to adjust the parameters in the network, the weight value of each layer can be obtained. Among them, the middle layer can be used as an approximate representation of the original input to achieve unsupervised feature dimensionality reduction. The autoencoder can be regarded as a three-layer neural network. The top layer is the input layer, and the bottom layer is the output layer. The number of nodes in the top layer is equal to that of the bottom layer. The middle layer is the compression representation of the top layer, which means the features of input data after dimensionality reduction, so this layer has fewer nodes than the input layer and output layer.

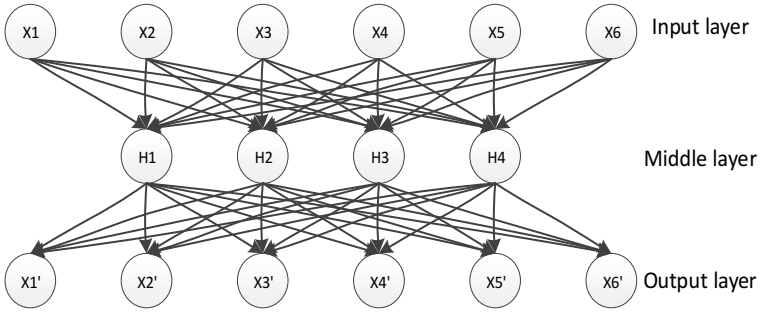

Figure 2. Schematic diagram of automatic encoder

When encoding, let each input sample be represented by a vector, as follows:

$$(x_1, x_2 \cdots x_n) \tag{5}$$

Where x_1~x_n refers to the dimension of each sample, that is, the value of different metric elements. In the network, the final output is need to be a probability value in the range of 0 to 1 However, As the numerical unit of each metric element is different, the value of the data set must be normalized. The specific normaliz method is to divide the specific value of the metric by the the maximum value of the metric value to obtain the numerical ratio, whose value range is [0,1], which meets the output requirements.

The activation function of the middle layer uses the sigmoid function:

$$\mathrm{sigmoid}(\mathrm{x}) = 1/(1 + \mathrm{e}^{-\mathrm{x}}) \tag{6}$$

When encoding, use the encoding function $H_n = f(x)(H_n \in [0,1]^m)$ to encode metrics in input leyer to get the values of middle layer in the network, where the encoding function is defined as:

$$\begin{cases} \mathrm{H_n(x)} = \mathrm{f}(\omega \mathrm{x} + \mathrm{b}) \\ \mathrm{f(x)} = 1/(1 + \mathrm{e}^{-\mathrm{x}}) \end{cases} \tag{7}$$

Where $\omega \in \mathrm{R}^{\mathrm{m*n}}$, $\mathrm{R}^{\mathrm{m*n}}$ is the weight matrix, m is the number of values in middle layer, and n is the number of samples.

When decoding, the decoding function is:

$$\mathrm{x}' = \mathrm{G_n(x)} = \mathrm{f}(\omega^{\mathrm{T}} \mathrm{x} + \mathrm{b}) \tag{8}$$

In the formula, $\omega^{\mathrm{T}} \in \mathrm{R}^{\mathrm{n*m}}$, f(x) is the same as formula (5). Using the decoding function to reconstruct the middle layer, the output value$\mathrm{x}'(\mathrm{x}' \in [0,1]^{\mathrm{n}})\text{^}\mathrm{n})$ can be obtained. can see that the structure of the auto-encoding network is a symmetrical structure, its objective function is as fallows:

$$\frac{1}{n}\sum_{i=1}^{n} L(x_i, x_i') \tag{9}$$

$$\mathrm{L(x, x')} = \|\mathrm{x} - \mathrm{x}'\|^2 \tag{10}$$

After multiple rounds of training are performed on the input sample data, when the parameters converge, a stable network structure can be obtained, in which, $\mathrm{x}' \approx \mathrm{x}$. Since the input layer can be approximately reproduced by the output layer, and encoding and decoding operations are performed from the input layer to the output layer, so the lower-dimensional middle layer can approximately represent the input layer, thereby realizing the feature reduction of the sample.

Furthermore, for a deep auto-encoding network structure composed of multiple single-layer auto-encoding networks from top to bottom. In the deep auto-encoding network, the middle layer of the previous network is no longer connected to the output

layer, but as the input layer of the next network. Use the greedy algorithm for machine learning training in this layer-by-layer auto-encoder to obtain network parameters that can make the one layer converge. Use the same method to repeat the training of the multi-layer auto-encoder to obtain the parameters of a complete set of deep auto-encoder network structure. Finally, a classifier is added to the network structure, using the deep auto-encoding network to perform feature reduction on the software data samples, data dimension of the n metric features of a sample will reduced to three dimensions to represent the sample characteristics.

3. Case Study

This article selects the NASA warehouse defect data for research, which contains 21 software metrics[6]. Those metrics are mainly divided into two categories: The first category is Halstead[7] variables, including the number of unique operators, the number of unique operations, the total number of operands and operators, the number of branches, capacity, program length, difficulty, Halstead intelligence, program efficiency, workload estimation, time, code lines, comment lines, blank lines, total code and comment lines. The second category is McCabe[8] variable, including three types of metrics. The first is cyclomatic complexity, the second is basic complexity, and the third is design complexity.

In the NASA data warehouse, seven historical models of rocket fire control software data are selected for this research, four of them are used as the training data set of the deep auto-encoding network, and the remaining three are used as the test data set. the data of these seven software is shown in table 1.

Table 1. Experimental data set

Software project	Data set type	Number of software units	Number of metric
Fire Control Software A	Training data set	656	21
Fire Control Software B	Training data set	564	21
Fire Control Software C	Training data set	558	21
Fire control software D	Training data set	452	21
Fire Control Software E	Test data set	661	21
Fire Control Software F	Test data set	553	21
Fire Control Software G	Test data set	756	21

The steps of the case study are as follows:

Step 1: Calculate the 21 software metric metadata of each unit, and establish a database based on the failure conditions in the test and operation records of the 7 fire control software;

Step 2: Take the metric element as input, take whether the unit has test or operation failure as the classifier, and use the deep auto-encoding network to perform parameter training on the data of fire control software A, B, C, and D;

Step 3: Use the trained parameters to predict the defects of the software units of the three test software, obtain all software units failure probabilities of the fire control software E, F, G;

Step 4: Calculate the failure probability P(E), P(F), P(G) of the fire control software E, F, G according to formula (3), and make a comparison between them and the values that calculated though the traditional Jelinski-Moranda model. The comparison results is shown in table 2.

Table 2. Comparison of failure probability prediction of test data

Software project	1-P	Total running time t (unit: h)	Confidence R(t)	Two methods deviation
Software E	0.9636	189	0.9612	0.24%
Software F	0.9323	210	0.9356	-0.35%
Software G	0.	2039516	0.9501	0.16%

It can be seen from the table that for rocket fire control software with similar architecture and functions, the software failure probability P calculated by the evaluation model established in this paper is similar to the confidence R calculated by the Jelinski-Moranda model, so the model can be used for software confidence evaluation .

4. Conclusion

This paper established a software confidence evaluation model based on software fault tree analysis and deep auto-encoding network. According to the common architecture of rocket fire control software, a fault tree is established, and the metrics for software defect prediction are constructed. The deep auto-encoding network is used to perform feature reduction and defect prediction based on historical data, thereby through calculating the probability of the top event, that is, the software failure. Comparing the calculated software failure probability with the confidence calculated by the Jelinski-Moranda model, the two calculation results are similar. Therefore, the software confidence evaluation model proposed in this paper can be used for the confidence evaluation of rocket fire control software in the engineering development process, and it can also be extended to other types of software.

References

[1] JELINSKI Z, B MORANDA P. Software reliability research[C]Proc of GREIBERGER. Statistical Computer Performance Evaluation. New York: Academic Press:1972:465-484.
[2] Fengzhe Zhu, Xiaoyi Lv. Improvement of Reliability Evaluation method Based on Defect Independence[J]. Information Research, 2017, 43(3):24-28.
[3] Zhou Mo, Xu Ling, Mengning Yang. Software defect prediction based on deep autoencoder networks[J]. Computer Engineering & Science, 2018, 40(10):1796-1804.
[4] Liu Boning, Zhang Peng, Jianye Zhang. Method for software reliability assessment based on fuzzy fault tree analysis[J]. Application of Research of Computers, 2012, 29(10):3783-3786.
[5] Bengio Y, Lamblin P, D Popovici. Greedy layer-wise training of deep networks[C]Twenty-First Annual Conference on Neural Information Processing System. NIPS:2007:
[6] Shepperd M Song Q. B. Sun. Data quality: Some comments on the NASA software defect datasets[J]. IEEE Transactions on Software Engineering, 2013, 39(9):1208-1215.
[7] MH Halstead. Elements of Software Science (Operating and programming systems series)[M].Elsevier Science Inc., 1977.
[8] TJ Mc Cabe. A complexity measure[J]. Software Engineering, IEEE Transactions on, 1976,(4):308-320Wang Meng. Research on defect detection method based on insulator image [D]. Huazhong University of science and technology, 2019