ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(44), DOI: 10.17485/ijst/2016/v9i44/93114, November 2016* ISSN (Online) : 0974-5645

Finding the Effectiveness of Prioritized Test Suite using Neural Networks

V. Chandraprakash, M. R. Narasingarao, M. Sindhu Saraswathi, N. Dorothy William, T. Lavanya Kumari and K. Venkata Sujith

Department of Computer Science and Engineering, K. L. University, Green Fields, Vaddeswaram, Guntur - 522502, Andhra Pradesh, India;
vchandrap@kluniversity.in, ramanarasingarao@kluniversity.in, mssaraswathi123@gmail.com, wills9324@gmail.com, lavanyaraju.t@gmail.com, sujithchowdary8989@gmail.com

Abstract

Objective: To find out whether a trained neural network can predict the Average percentage of Faults Detected (APFD) value for the given prioritized test suite without having the knowledge of computing APFD formula. **Methods:** To generate a test suite, A fault matrix containing faults and test cases is considered and for each possible permutation of test cases. The APFD value is computed for each of such test suite. The test suites with their respective APFD values are given to the neural network during training. During testing, a prioritized test suite is fed to the neural network. The APFD value predicted by the neural network is noted down. **Findings:** The neural network has learnt to predict APFD value of the given prioritized test suite. The predicted APFD value is compared with computed APFD value using the Root Mean Square Deviation. The deviation is found to be very low, showing that the values are very nearer. **Applications/Improvements:** Number of hidden layers can be increased in order to reduce the deviation further.

Keywords: Artificial Neural Network, Average Percentage of Faults Detected, Test Case Prioritization

1. Introduction

Regression testing is to find bugs in already tested software. It is done to find if any new bugs have been generated in the software due to changes in code for improvement, addition, bug fixing, etc. The regression testing is done to know if bugs arise while modifying the code. Regression test suite is just a collection of test cases that developers have created beforehand, and that have been stored with the goal that they can be utilized later to perform regression testing. Generally, all the tests which have been performed on the software are again performed in regression testing. This is done to check if faults which were fixed earlier have again emerged in the software.

1.1 Early Detection of Faults

If the regression testing is not performed completely on the software under test (SUT), the software product may not be reliable. If this unreliable product is released, then the software product may fail in its functionality or may even crash. To avoid this situation most of the faults should be detected at an earlier stage. The early detection of faults helps the debuggers to fix the code even before completion of testing. The early detection of faults also lessens the probability of software failure. To detect faults earlier the test cases have to be rearranged in an order.

1.2 Fault Matrix

In the fault matrix, the test cases of the regression test suite are represented as columns in matrix and the faults are represented as rows in the matrix. Each test case (a column) shows the faults (represented by*) that can be detected. An example of a fault matrix is shown in Table 1.

1.3 Test Case Prioritization

Test case prioritization methods mean to enhance the adequacy of regression testing, by re-arranging the test cases so that the most useful are executed first with more

* *Author for correspondence*

priority. The test case prioritization works on an objective to improve some functions, one of which is the rate at which a fault is detected. In test case prioritization, the ordering of test cases plays an important role and the ordering of which is dependent on the number of faults that can be detected earlier. This is the main goal of test case prioritization. The fault detection rate is the number of faults detected within short period of testing. As the rate of fault detection increases, the debuggers can start fixing of faults at an early stage due to prompt feedbacks from testing teams.

1.4 Average Percentage of Faults Detected (APFD)

Gregg Rothermel et al.[1] coined the word APFD which is an acronym for Average Percentage of Faults Detected. It is a metric to find the effectiveness of how early the faults are detected by execution of test cases in a particular order. APFD gives a value ranging from 0 to1 which represents the rate of fault detection of an ordered test suite. The greater the APFD value means the more effective is the test suite in detecting the faults early. Consider the fault matrix shown in Table 1. The APFD of the test suite represented in fault matrix in Table 1 can be computed using the formula.

$$APFD = 1 - \frac{TF1 + TF2 + TF3 + \cdots + TFm}{n * m} + \frac{1}{2n}$$

Where TFi = the first test case in the test suite which identifies fault i,

m = The number of faults contained in the program under testing,

n = Number of test cases performed on the program.

The APFD of the test suite (shown in Table1*) is 0.65.*

Table 1. Test Suite with faults

	T1	T2	T3	T4	T5	T6
F1				*	*	*
F2						*
F3	*	*	*	*	*	*
F4	*			*		
F5	*	*		*	*	*

The test suite shown in Table 1 is rearranged in a different order as shown in Table 2.

The APFD of the rearranged test suite is 0.78 which indicates that this rearranged test suite is much better than the earlier one in detecting faults earlier.

Table 2. Rearranged test suite

	T1	T2	T6	T4	T3	T5
F1			*	*		*
F2			*			
F3	*	*	*	*	*	*
F4	*			*		
F5	*	*	*	*		*

1.5 Neural Networks

Artificial neural networks is an important application of artificial intelligence and it can be normally represented as systems of interconnected "neurons" present in the input, hidden and output layers. The neurons present in the layers are connected by synapses and each neuron in each layer is connected to other neuron in the other layer. It works similar to a human brain in obtaining knowledge through learning. The synapses have numeric weights and these synaptic weights change based on experience during training, making neural nets adaptive to inputs and capable of learning.

Neurons: These are the basic units of the neural network which are similar to biological neurons present in human brain. The artificial neurons accept the input signals which are processed and yield output signals. The neurons have an activation function which is responsible for transformation of inputs into output.

Network architecture: In most of the applications, feed-forward networks are used for the purpose in which the input is processed from input to hidden and from hidden to output layer.

1.6 Problem Definition

We want to investigate whether an intelligent system can predict the effectiveness of the given prioritized test suite using an artificial neural network, without the knowledge of the actual formula to compute APFD of the ordered test suite. This problem comes under the category of machine learning.

The authors[1] have proposed different techniques for test case prioritization, and these techniques gave results measuring the effectiveness of these techniques for improving the fault detection rate.

The authors[2] have reported the results of several experiments which contain techniques that order test

cases based on their estimated ability to reveal faults in the code components that they cover, to various test suites for various programs.

The authors[3] have incorporated varying test costs and fault severities into test case prioritization and have given a new metric for assessing the rate of fault detection of prioritized test cases.

The authors[4] have analysed the fault identification rate that resulted from various programs and updated versions by applying various prioritization techniques.

The authors[5] have talked about a value-driven approach to system-level test case prioritization called the prioritization of requirements for test (PORT). The results have shown that, PORT prioritization at the system level improved the rate of detection of severe faults.

The authors[6] have proposed various methods of test prioritization based on state-based models have been developed by changing the model and the system. For prioritizing the tests, Information about the model after execution is used for the purpose. They also have given an analytical framework for assessment of test case prioritization methods.

The authors[7] described a novel approach of test case prioritization which is dependent on the presence of the coverage requirements in the outputs of test cases.

The authors[8] have developed a technique to select subsets of the test cases which reduce the time consumed for evaluation of a new software version and also to continue to identify the defects that were seeded.

The authors[9] have presented a paper on assessment of test case prioritization by using the mutation faults. Two controlled experiments have been created to improve fault detection rate of test cases and which is done by analysing the ability of prioritization techniques the results of which were measured with respect to mutation faults.

According to [10]the authors have done an empirical study of the application of various greedy, meta-heuristic and evolutionary search algorithms to six programs to find the suboptimal results which denote local minima within the search space.

The authors[11] have developed a new test case prioritization algorithm to calculate average number of faults detected per unit time and based on which the test suite can be rearranged based on priorities. The objective to determine the effectiveness of prioritized and non-prioritized cases with the help of APFD.

The authors[12] have discussed a model-based prioritization techniques and discussed several model-based test prioritization heuristics. The authors[13] have given an empirical study to find the impact of test case prioritization on the effectiveness of fault localization.

The authors[14] have proposed clustering based prioritization which has used APFD metric to support their efforts.

The authors[15] have described new prioritization technique based on hamming distance and to present the effectiveness of the algorithm. APFD metric is used to present the effectiveness of the proposed algorithm. The authors[16] have done a survey on test case prioritization.

The authors[17] have analyzed the metric APFD and investigated the gain of measuring Test Case Prioritization techniques from a control theory viewpoint. In[18] the authors have presented experimental results by using a neural network for reducing the test suite.

The authors[19] have described a method to predict software faults identification using neural networks during testing phase.

The authors[20] have presented a novel approach of using ANN in software testing for automating the oracle software.

The authors[21] have elucidated an approach for automating the prioritization of test cases process using GA by using Multi-Criteria Fitness function.

The authors[22] have described a method to prioritize test cases using parameters like traceability, customer priority, etc.

A new algorithm has been proposed by the authors[23] to prioritize the test cases for fault detection and reduction of cost. Heuristic Methods like Genetic Algorithm and Simulated Annealing are used for the purpose.

2. Investigations and Findings

In this paper, we proposed a methodology using which we can solve the above mentioned problem. As per the methodology we developed the necessary programs and tabulated the results obtained.

a. Methodology

A Neural Network has to be trained (machine learning) so that it can be used to predict the APFD values for a given test suite.

- Create a test suite containing different test cases (represented as columns in the fault matrix) and the

corresponding faults detected (represented as rows in the fault matrix);

- For all possible permutations of test cases in test suite
  {
  Calculate APFD; //using formula
  }
- Train the neural network with some selected ordered test suites (with their APFD values);

Convert the test suite (fault matrix) into binary form. i.e., every blank element in the matrix is replaced with a '0' and every '*' is replaced with a '1'. Feed all the rows to the neural network as input. For the prioritized test suite shown in Table 2 the input to the neural network is given below.

00110100100011111110010011101

For the prioritized test suite the APFD is computed as 0.78.

Feed 0.78 as output to the neural network.

- Let the neural network predict APFD for the given set of ordered test suites;
- Compare the original APFD values of the given set of ordered test suites with the predicted APFD values;

b. Results

A back propagated neural network (BPNN) with one input layer, one hidden layer, and one output layer is selected for our research. 23 neurons have been taken into consideration as hidden neurons for this purpose. A set of 100 ordered test suites and the corresponding APFD values are fed to the neural network as input for training the network, and for testing, we have selected 50 ordered test suites for the purpose in ascertaining the accuracy of the network. For each ordered test suite, we have noted down the output of the neural network which is regarded as a predicted APFD value.

$$Absolute\ Difference = abs(computed\ APFD - predicted\ APFD)$$

The sum of absolute differences is calculated. Then, we increased the number of ordered test suites for training to 200, 300, 400, 500, and 600. We plotted a graph taking number of test suites in training on x-axis and sum of absolute differences of 50 test suites in testing on y-axis. We have repeated this experiment by varying the number of neurons in hidden layer from 1 to 23 and plotted different lines. Then we superimposed all the lines on a single graph which is given in (Figure 1). It is observed that the sum of absolute difference is minimum when the number of training set is around 400. So we fixed up 400 as the number of test suites for training.

We want to find the optimum number of neurons in hidden layer for which the network gives better results. We reconfigured back propagation neural network with number of neurons in the hidden layer ranging from 1 to 23. The Figure 2 represents the relationship between the numbers of neurons in hidden layer vs. Sum of absolute differences. It is observed that sum of absolute difference is minimum when the number of neurons in the hidden layer is equal to 21. Hence, the number of neurons in hidden layer is fixed as 21.

A set of 40 ordered test suites was considered for testing the neural network. The APFD value predicted by the neural network for each of the ordered test suite is noted down. The computed APFD, the corresponding predicted APFD and the absolute difference are shown in the Table 3.

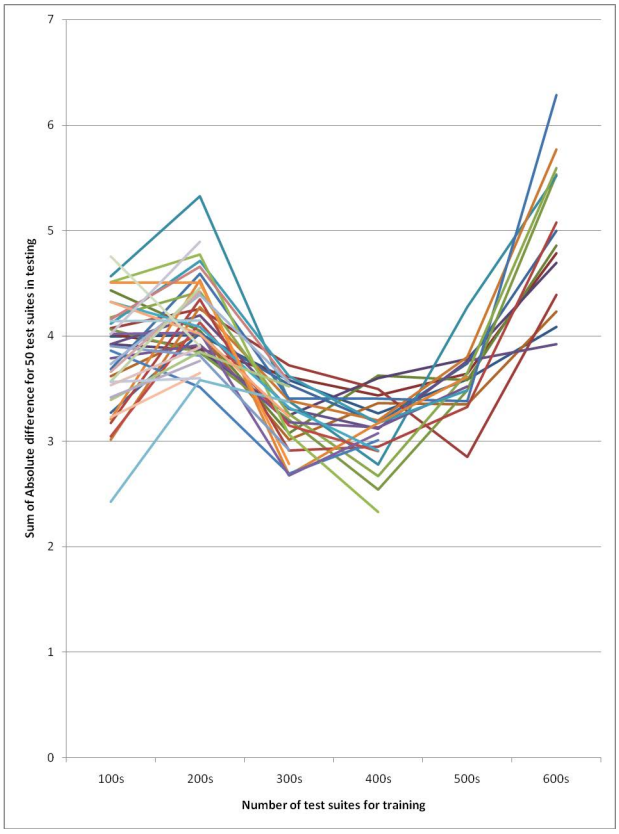

Figure 1. Number of test suites vs. Sum of absolute differences.

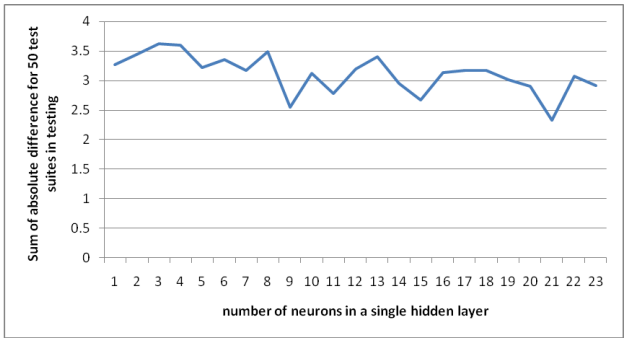

Figure 2. Number of neurons vs. Sum of absolute differences.

Table 3. Computed APFD vs. Predicted APFD

S.No.	Computed APFD	Predicted APFD	Absolute difference
1	0.82	0.82	0.00
2	0.72	0.72	0.00
3	0.75	0.74	0.01
4	0.82	0.81	0.01
5	0.75	0.76	0.01
6	0.72	0.73	0.01
7	0.72	0.73	0.01
8	0.82	0.81	0.01
9	0.78	0.77	0.01
10	0.78	0.79	0.01
11	0.78	0.80	0.02
12	0.75	0.73	0.02
13	0.75	0.73	0.02
14	0.78	0.80	0.02
15	0.78	0.76	0.02
16	0.75	0.78	0.03
17	0.82	0.79	0.03
18	0.78	0.81	0.03
19	0.82	0.79	0.03
20	0.75	0.78	0.03
21	0.75	0.79	0.04
22	0.82	0.78	0.04
23	0.75	0.79	0.04
24	0.75	0.79	0.04
25	0.82	0.78	0.04
26	0.65	0.70	0.05
27	0.72	0.77	0.05
28	0.75	0.80	0.05
29	0.68	0.73	0.05
30	0.88	0.83	0.05
31	0.82	0.76	0.06
32	0.68	0.74	0.06
33	0.65	0.71	0.06
34	0.88	0.81	0.07
35	0.72	0.79	0.07
36	0.88	0.81	0.07
37	0.68	0.75	0.07
38	0.85	0.78	0.07
39	0.85	0.78	0.07
40	0.85	0.78	0.07

c. Comparative Analysis

In order to find whether a predicted APFD value is comparable with the computed APFD value, Root Mean Square Deviation (RMSD) is calculated.

$$RMSD = \sqrt{\frac{1}{n} * \sum_{t=1}^{n} (x - \hat{x})^2}$$

where x = computed value

$\hat{x}$ = predicted value

n = number of observations

The RMSD value for the values in table 3 is found to be 0.0436 which is very low. Hence the predicted APFD value of the given ordered test suite is very close to the computed APFD of the test suite.

3. Conclusions and Future Scope

In this paper, we have proposed a new way of predicting APFD of the given ordered test suite using a trained neural network with back propagation algorithm. The neural network has no knowledge of the formula for computing APFD of the given ordered test suite. As per our proposed methodology, we have trained the neural network by giving inputs as ordered test suits and unknown APFD values have been predicted for the purpose. The predicted APFD value of the ordered test suite was compared with computed APFD value using statistical means i.e., by finding root mean square deviation. The deviation is found to be very low and hence we conclude that the predicted APFD value of the given ordered test suite correlates with computed APFD value. There is also one more advantage with our proposed methodology. In future, even if the APFD formula is revised, still the same methodology works out without any change to predict the revised (modified) APFD value.

In future, this research work can be carried further in order to reduce the root mean square deviation. For example, we can try it out by increasing the number of hidden layers.

4. References

1. Rothermel G, Untch RH, Chengyun C and Mary Jean H. Test case prioritization: An empirical study. Proceedings of the International Conference on Software Maintenance. 1999; p. 179-88.

2. Rothermel G, Untch RH, Chengyun C and Mary Jean H. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering. 2001; 27(10):929-48.
3. Elbaum S, Malishevsky AG and Rothermel G. Incorporating varying test costs and fault severities into test case prioritization. Proceeding of the International Conference on Software Engineering. 2001; p. 329-38.
4. Elbaum S, Rothermel G, Satya K and Malishevsky AG. Selecting a cost-effective test case prioritization technique. Software Quality Journal. 2004; 12(3):185-210.
5. Srikanth, Hema, Laurie W and Jason O. System test case prioritization of new and regression test cases. International Symposium on Empirical Software Engineering. 2005; 1(10).
6. Korel B, Luay HT and Harman M. Test prioritization using system models. Proceedings of the International Conferenceon Software Maintenance. 2005; p. 559-68.
7. Jeffrey D and Neelam G. Test case prioritization using relevant slices. International Computer Software and Applications Conference COMPSAC'06. 2006; p. 411-20.
8. Simao AD, Rodrigo FD and Luciano JS. A technique to reduce the test case suites for regression testing based on a self-organizing neural network architecture. International Computer Software and Applications Conference COMPSAC'06. 2006; p. 93-6.
9. Hyunsook D and Rothermel G. On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Transactions on Software Engineering. 2006; 32(9):733-52.
10. Zheng L, Harman M and Hierons RM. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering. 2007; 33(4):225-37.
11. Srivastava PR. Test case prioritization. Journal of Theoretical and Applied Information Technology. 2008; 4(3):178-81.
12. Korel B, George K and Luay HT. Application of system models in regression test suite prioritization. International Conference on Software Maintenance ICSM. 2008; p. 247-56.
13. Jiang B, Zhenyu Z, Tse TH and Chen TY. How well do test case prioritization techniques support statistical fault localization. International Computer Software and Applications Conference COMPSAC'09. 2009; p. 99-106.
14. Arvind KU and Misra AK. Prioritizing Test Suites Using Clustering Approach in Software Testing. International Journal of Soft Computing and Engineering. 2012; 2(4).
15. Maheswari RU and Jeya Mala D. A novel approach for test case prioritization. Internal Conference on Computational Intelligence and Computing Research (ICCIC). 2013; p. 1-5.
16. Catal C and Deepti M. Test case prioritization: a systematic mapping study. Software Quality Journal. 2013; 21(3):445-78.
17. Junpeng L, Yin B and Cai KY. On the Gain of Measuring Test Case Prioritization. International Computer Software and Applications Conference (COMPSAC). 2013; p. 627-32.
18. Anderson, Charles, Von Mayrhauser Anneliese and Mraz Rick. On the use of neural networks to guide software testing activities. Proceedings of the International Test Conference. 1995; p. 720-29.
19. Khoshgoftaar, Taghi M and Szabo MR. Using neural networks to predict software faults during testing. IEEE Transactions on Reliability. 1996; 45(3):456-62.
20. Vanamali M, Mark L and Abraham K. Using a neural network in the software testing process. International Journal of Intelligent Systems. 2002; 17(1):45-62.
21. Ahmed A, Abdelbaky, Shaheen M and Kosba E. Software testing suite prioritization using multi-criteria fitness function. Proceeding of the 22nd International Conference on Computer Theory and Applications. 2012; p. 160-66.
22. Chandu P, MSS and Sasikala T. Implementation of regression testing of test case prioritization. Indian Journal of Science and Technology. 2015; 8(8):290-93.
23. Maheswari RU, Mala DJ. Combined Genetic and Simulated Annealing Approach for Test Case Prioritization. Indian Journal of Science and Technology. 2015; 8(35):1-5.