Deep Learning-Based Fault Localization with Contextual Information

Zhuo ZHANG†, Nonmember, Yan LEI†††, ††††, Member, Qingping TAN††, Xiaoguang MAO†, Ping ZENG†, and Xi CHANG†, Nonmembers

SUMMARY Fault localization is essential for solving the issue of software faults. Aiming at improving fault localization, this paper proposes a deep learning-based fault localization with contextual information. Specifically, our approach uses deep neural network to construct a suspiciousness evaluation model to evaluate the suspiciousness of a statement being faulty, and then leverages dynamic backward slicing to extract contextual information. The empirical results show that our approach significantly outperforms the state-of-the-art technique Dstar.

key words: fault localization, dynamic slice, deep learning, contextual information

1. Introduction

Software faults are inevitable with software’s increasing scale and complexity. However, debugging activity is an expensive and time-consuming process. Thus, many efforts have been made to provide assistance in guiding programmers to the locations of faults that cause incorrect outputs [1]. The core work of fault localization is to evaluate which statement being suspiciously faulty. Among existing localization methods, spectrum-based fault localization (SBFL) is the most popular one by using spectrum-based suspiciousness formulas to assign suspiciousness values to the statements and selecting a subset of statements among suspicious statements [2]. Xie et al. [2], [3] has theoretically showed the maximal effectiveness that SBFL can reach, that is, the potential of SBFL has been fully explored. Thus, it is necessary to propose a novel approach different from the view of SBFL to evaluate which statements being suspiciously faulty for further improvement.

We notice that machine learning techniques were successfully applied in the field of fault localization [4], [5]. However, many models have drawbacks such as restricted generalization ability for intricate problem and inadequate ability in using limited data set to express complicated functions. With these shortcomings, the faults cannot be analyzed exactly. In recent years, deep learning has witnessed a rapid development to tackle these limitations. With its ability of showing tremendous improvement in robust and accuracy, deep learning has already been rapidly evolved into making sense of data such as images, text and sound [6]. It means that with this technology, we could construct a deep neural network with multiple hidden layers to extract features from the huge amount of data collecting from test cases and then assign different suspiciousness to sentences and rank them. Another drawback of many techniques is that they ignore the contextual information of the relationship among suspicious statements [4]. Hence, program slicing technology could be taken into consideration for its ability of extracting data and/or control dependencies of program statements and selecting a subset of statements affecting the incorrect output [7], [8]. There are static slicing and dynamic slicing, where dynamic slicing gathers run-time information along the execution path and cut down the size of the slice noticeably in comparison with static slicing [9]. Hence, we use dynamic slicing to construct the context.

Thus, we propose a fault localization approach using deep neural network and dynamic backward slicing technique. Our approach adopts deep neural network to assign different suspiciousness values to the statements, then we construct a suspicious context with the utilization of dynamic slicing to slice the output of a failed test case [10]. Consequently, our approach provides useful context with its elements ranked in descending order to provide the examining guidance for developers. We conduct an empirical study on 10 representative open-source Java programs. The results show that our approach can reduce almost 43.48% up to 88.85% of the average cost of examined code over the state of the art technique Dstar [11].

2. Approach

2.1 Overview

Deep neural network is an artificial neural network with multiple hidden layers, each node at the same hidden layer uses the same nonlinear function to map the feature input from the layer below. Deep neural network’s structure is very flexible and demonstrates excellent capacity to fit the highly complex nonlinear relationship between inputs and outputs.
The basic idea of our approach is to adopt deep neural network to quantify the suspiciousness of the statements, and then use dynamic backward slicing to construct a suspicious context with the failed outputs. Since the suspicious context defined in our approach can show useful information of data/control dependencies and present its statements with different suspiciousness in descending order, it can help developers to distinguish which statements are more suspicious and should be assigned priority to be checked.

**Evaluating suspiciousness using deep neural network:** We adopt deep neural network to compute the suspiciousness of each statement. First, a deep neural network is constructed with three parts: the input layer, hidden layer and output layer. Then this model is trained with the utilization of test cases’ results and statement coverage data which is collected indicating that whether the statements are executed or not as input, we further get suspiciousness of each statement by testing the trained model using virtual test suit. Specifically, given a program \( P \) with \( N \) executable statements, it is executed by \( M \) test cases \( T \) which contains at least one failed test case (see Fig. 1). \( x_{ij} = 0 \) indicates that statement \( j \) is not executed under test case \( i \), and \( x_{ij} = 1 \) otherwise. The error vector \( e \) represents the test results. The element \( e_j \) is equal to 0 if test case \( i \) passed, and 1 otherwise. Based on the coverage vector as input, the network is trained iteratively. The complex nonlinear relationship between the statement’s coverage information and test case’s result can be reflected after training. At last, a set of virtual test cases (see Fig. 2) that is an \( N \)-dimensional unit matrix is constructed as the testing input, the output vector is the suspiciousness of statements.

After suspiciousness calculation by deep neural network, a ranking list of all statements is produced in descending order of their suspiciousness. In our model (see Fig. 3), there are one input layer with the number of nodes according to the number of executable statements, one output layer with the number of nodes according to test cases’ scale, appropriate number of hidden layers with the number of nodes in each one estimating by the following formula:

\[
\text{number} = \text{round}(n/30 + 1) \times 10
\]

where, \( n \) represents the number of executable statements. The hidden layers extract features from the input layer. Transfer function reflects the complex relationship between input and output, we use \( \text{relu} \) function in the hidden layer and output layer as nonlinear transfer function and \( \text{sigmoid} \) function to output the result:

\[
\text{relu}(x) = \max(0, x)
\]

\[
\text{sigmoid}(x) = 1/(1 + e^{-x})
\]

where, \( x \) is the output vector from the last layer. The learning rate impacts the speed of convergence. In our model, dynamic adjusting learning rate is adopted for it has the benefit of making large changes at the beginning of the training procedure when larger learning rate values are used and decreasing the learning rate therefore smaller training updates are made to weights later, this method can get good weights more quickly. In the following formula, one \( \text{Epoch} \) means completing all training data once, \( LR \) represents learning rate, \( \text{DropRate} \) is the amount that learning rate is modified each time while \( \text{EpochDrop} \) is how often to change the learning rate.

\[
LR = LR \times \text{DropRate}(\text{Epoch} + 1)/\text{EpochDrop}
\]

Impulse factor is adjusted according to the size of the sample. We use Back Propagation algorithm to fine-tune the parameters (weight and bias) of the model, and the goal is to minimize the difference between error vector \( e \) (see Fig. 1) and training result \( y \).

**Construct suspicious contexts:** The suspicious context is constructed by the dynamic slice of the faulty output of a failed test case. Specifically, a dynamic slice consists of an execution trace and a slicing criterion. We randomly choose a failed test case, using its execution trace as the execution trace for the dynamic slice. Furthermore, we use the output statement with its variable outputting the faulty result of the failed test case as the slicing criterion. Thus, we can obtain a dynamic slice. Since the dynamic slice shows how a set of statements affect the faulty output to cause a failure, we define it as the suspicious context.

**2.2 An Illustrative Example**

Figure 4 shows an example illustrating how our approach is to be applied with program \( P \) and a faulty statement \( s_3 \). The cells below each statement represent whether the statement...
Fig. 4 Example illustrating our approach based on deep learning.

is covered by the test case showed in the left cell (1 for executed and 0 otherwise). The rightmost cells indicate whether the test case is failed or not (1 for fail and 0 otherwise). The concrete process is as follows: Firstly, construct the deep learning model with the number of input layer nodes being 16, 3 hidden layers with the number of each one's nodes being 10 according to Eq. (1) and the number of output layer nodes being 1. Secondly, we input the vector \( t_1 = (1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0) \) and its result 1, then vector \( t_2 = (1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1) \) and its result 0 into the input layer until the coverage data and execution results are all inputted into the network. After that, we train the network iteratively to get the final relationship between the coverage information and test cases' results. Thirdly, we construct the virtual test set which is a 16 dimensional unit matrix and put it into the network, then get the suspiciousness values. Finally, dynamic slice is utilized to get the context and rank the statements in the context by using a slicing criterion \( (t_1, s_{14}, x) \), where \( t_1 \) is a failed test case and \( s_{14} \) is the statement that outputs the faulty result with its variable \( x \).

Based on these information, Dstar and deep neural network both output a ranking list of all statements in descending order. It reveals that the faulty statement \( s_3 \) is ranked 8th by Dstar and ranked 4th by deep neural network while ranked 1st in the context \( \{s_1, s_3, s_7, s_{14}\} \) by our approach, the statements in the context are all responsible for the failed output of \( t_1 \). Therefore, our approach obtains a better localization result than original Dstar.

3. An Experimental Study

3.1 Experimental Setup

The experiment selects 10 Java programs. Table 1 lists the name of these programs, function description, the number of faulty versions, the lines of code and the number of test cases. The 6 small-scale subject programs and 4 large-scale subject programs are all widely used in the fault localization community.

Our experiments adopt widely used metrics EXAM \[8\] and RImp \[12\], the former is defined as the percentage of executable statements to be examined before finding the actual faulty statement, the latter is to compare the total number of statements that need to be examined to find all faults by our approach versus the number that need to be examined by using other fault localization technique. Lower values of both metrics show better improvement. As a reminder, our system uses EMMA† to collect coverage information and JSlice†† to perform slicing.

3.2 Data Analysis

Since our approach has the effect of deep learning and context (dynamic slicing), the comparison has four cases: Deep Learning (context), Deep Learning, Dstar (context), and Dstar, among which Deep Learning (context) is our proposed approach. Figure 5 shows the EXAM distribution among four cases in all faulty versions. As shown in Fig. 5,

### Table 1 Subject programs.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
<th>Versions</th>
<th>Loc</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print_token1</td>
<td>Lexical analyzer</td>
<td>5</td>
<td>587</td>
<td>4,071</td>
</tr>
<tr>
<td>Print_token2</td>
<td>Lexical analyzer</td>
<td>10</td>
<td>571</td>
<td>4,056</td>
</tr>
<tr>
<td>Schedule1</td>
<td>Priority scheduler</td>
<td>9</td>
<td>422</td>
<td>2,650</td>
</tr>
<tr>
<td>Schedule2</td>
<td>Priority scheduler</td>
<td>23</td>
<td>411</td>
<td>2,710</td>
</tr>
<tr>
<td>Tot_info</td>
<td>Information measure</td>
<td>37</td>
<td>173</td>
<td>1,052</td>
</tr>
<tr>
<td>Jcasis</td>
<td>Collision avoidance</td>
<td>14</td>
<td>198</td>
<td>1,608</td>
</tr>
<tr>
<td>NanoXML_v1</td>
<td>XML parser</td>
<td>7</td>
<td>5,369</td>
<td>206</td>
</tr>
<tr>
<td>NanoXML_v2</td>
<td>XML parser</td>
<td>7</td>
<td>5,650</td>
<td>206</td>
</tr>
<tr>
<td>NanoXML_v3</td>
<td>XML parser</td>
<td>10</td>
<td>8,392</td>
<td>206</td>
</tr>
<tr>
<td>NanoXML_v4</td>
<td>XML parser</td>
<td>7</td>
<td>8,795</td>
<td>206</td>
</tr>
</tbody>
</table>

†EMMA, http://emma.sourceforge.net/
††JSlice, http://jslice.sourceforge.net/
Fig. 5  EXAM comparison among our approach, Deep Learning, Dstar (context), and Dstar.

Fig. 6  RImp of our approach over Deep Learning, Dstar (context), and Dstar on all faulty versions of each small-scale program.

Fig. 7  RImp of our approach over Deep Learning, Dstar (context), and Dstar on all faulty versions of each large-scale program.

our approach significantly outperforms the other three techniques. Furthermore, we have the effectiveness relationship:

- Deep Learning (context) > Dstar (context) and Deep Learning > Dstar.
- Deep Learning (context) > Deep Learning and Dstar (context) > Dstar.

The relationship means that both deep learning and context have a beneficial effect on fault localization. Based on the gap between each of two curves, the effect of context is a bit higher than that of deep learning.

Figures 6 and 7 illustrate the RImp score of our approach over Deep Learning, Dstar (context), and Dstar in all faulty versions of each program\(^1\). The data in the cells show the detailed RImp values, and the fraction in the parentheses shows the computation of RImp, among which the numerator is the total number of the statements examined by our approach to locate all faults in all faulty versions of a program and the denominator is the total number of the statements examined by other fault localization technique. Take Print\_tokens in Dstar as an example. Our approach needs to check a total number of 34 statements to locate all faults of Print\_tokens, while Dstar requires to examine a total number of 305 statements to locate all faults of Print\_tokens. It means our approach takes up a 34/305 = 11.15% of the examined statements of Dstar to locate all faults in Print\_tokens. We can observe that our approach sharply decreases the statements that need to be examined over the three techniques, accounting for a range from 26.98% to 92.86% of the examined statements of Dstar.

4. Conclusion

In this paper, we propose an approach using deep neural network to establish an effective suspiciousness model and

\(^1\)Complete data of RImp on each fault version are available at https://github.com/ifuleyou/Rimp/blob/master/RImp.txt
using dynamic slicing to construct contexts for fault localization, and the results show a preliminary benefit on fault localization. In the future, we plan to promote the accuracy of our approach and explore more in the potential of deep learning for fault localization.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grant No.61502296, No.61602504 and No.61672529, the Scientific Research Fund of Hunan Provincial Education Department (Grant No.15A007), the Municipal Natural Science Foundation of Shanghai (Grant No.15ZR1417000).

References


