DeepBugs: A Learning Approach to Name-based Bug Detection

Michael Pradel and Koushik Sen

To appear at OOPSLA 2018
Software has bugs
Software has bugs

0.5-25/KLoC

in delivered software
Static Bug Detection

- Lightweight static analysis
- General framework & set of checkers for specific bug patterns

Error Prone

Google

Infer

SpotBugs
The Problem

- Existing bug detectors miss most bugs (see our ASE’18 paper)

- Main reasons:
  - Bugs are domain-specific
  - Bug detectors cover only a small fraction of all bug patterns
Traditional Approach

How to create a new bug detector?

Human expert → Time-consuming process → Program analysis
Traditional Approach

How to create a new bug detector?

- **Heuristics**, e.g., to avoid spurious warnings
- Carefully **tuned algorithms**, e.g., to ensure scalability

Program analysis

Time-consuming process

Human expert
Learning to Find Bugs

Train a model to distinguish correct from buggy code

Buggy code → Train machine learning model
Correct code → New code

Classifier

Buggy/Okay
Learning to Find Bugs

Train a model to distinguish correct from buggy code

How to get training data?
- Here: Insert artificial bugs via simple program transformations
- Other options, e.g., from version histories
Learning to Find Bugs

Train a model to distinguish correct from buggy code

Buggy code $\rightarrow$ Train machine learning model $\rightarrow$ Classifier $\rightarrow$ Buggy/Okay
Correct code $\rightarrow$

How to represent code?
- Here: Embeddings of natural language elements in code
- Other options, e.g., token-based, graph-based
Benefits of Learning Bug Detectors

Simplifies the problem
- Before: Writing a program analysis
- Now: Providing examples of buggy and correct code

Catches otherwise missed bugs
- Learns conventions from corpora of existing code
- ML can handle natural language in code, which expresses domain-specific knowledge
Name-related Bugs

What’s wrong with this code?

```javascript
function setPoint(x, y) { ... }

var x_dim = 23;
var y_dim = 5;
setPoint(y_dim, x_dim);
```
Name-related Bugs

What’s wrong with this code?

```javascript
function setPoint(x, y) { ... }

var x_dim = 23;
var y_dim = 5;
setPoint(y_dim, x_dim);
```

Incorrect order of arguments
Name-related Bugs (2)

What’s wrong with that code?

```c
for (j = 0; j < params; j++) {
    if (params[j] == paramVal) {
        ...
    }
}
```
Name-related Bugs (2)

What’s wrong with that code?

```java
for (j = 0; j < params; j++) {
    if (params[j] == paramVal) {
        ...
        Should be params.length
    }
}
```
Overview of DeepBugs

Code corpus → Generate training data → Correct code, Buggy code → Represent code as vectors → Correct vectors, Buggy vectors → Train classifier → Classifier → Predict bugs in new code → Bugs

- Generate training data
- Correct code, Buggy code
- Represent code as vectors
- Correct vectors, Buggy vectors
- Train classifier
- Classifier
- Predict bugs in new code
- Bugs
Generating Training Data

Simple code transformations to inject artificial bugs into given corpus
Generating Training Data

Simple code transformations to inject artificial bugs into given corpus

1) Swapped arguments

\[ \text{setPoint}(x, y) \rightarrow \text{setPoint}(y, x) \]
Generating Training Data

Simple **code transformations to inject artificial bugs into given corpus**

2) Wrong binary operator

\[ i \leq \text{length} \quad \rightarrow \quad i \% \text{length} \]

Randomly selected operator
Generating Training Data

Simple code transformations to inject artificial bugs into given corpus

3) Wrong binary operand

\[\text{bits} \ll 2 \rightarrow \text{bits} \ll \text{next}\]

Randomly selected operand that occurs in same file
Representing Code as Vectors

- Insight: **Natural language** in identifiers conveys **semantics** of code

- Compute **word embeddings** of **identifier names**
  - Train **Word2Vec** on corpus of code
  - Tokens $\approx$ words

*Efficient Estimation of Word Representations in Vector Space* (Mikolov et al., 2013)
Word Embeddings

- **Known problem in natural language processing**

- **Word embeddings**
  - Continuous vector representation for each word
  - Similar words have similar vectors
Word2Vec

Learn embeddings from corpus of text

- ”You shall know a word by the company it keeps”
- Context: Surrounding words in sentences
Word2Vec

Learn embeddings from corpus of text

- "You shall know a word by the company it keeps"
- Context: Surrounding words in sentences

Input layer: Context words

Hidden layer

Output layer: Word
Word2Vec for Source Code

Natural language
- Sentences
- Words

Programming language
- Program
- Tokens
function setPoint(x, y) { ... }

var x_dim = 23;
var y_dim = 5;
setPoint(y_dim, x_dim);
Word2Vec for Source Code

Natural language

- Sentences
- Words

Programming language

- Program
- Tokens

```javascript
function setPoint(x, y) { ... }
```

```javascript
var x_dim = ...
var y_dim = ...
setPoint(y_dim, x_dim);
```

Context of x:
```
function - setPoint - (- , - y -)
```
Example: Embeddings
Code Snippets as Vectors

Concatenate embeddings of names in code snippet

1) Swapped arguments

```python
someObj.someFun(arg1, arg2)
```

For each argument: Name, type, and formal parameter name
Code Snippets as Vectors

**Concatenate embeddings of names in code snippet**

2) + 3) Wrong binary operator/operation

```
i <= length
```

For each operand:
- Name and type
- Parent and grand-parent
- AST node type
Learning the Bug Detector

- Given: Vector representation of code snippet
- Train neural network: Predict whether correct or wrong

Vector representation of code snippet ➔ Hidden layer ➔ Probability that correct

Hidden layer: size=200, dropout=0.2
RMSprop optimizer with binary cross-entropy as loss function
Predicting Bugs in New Code

- Represent code snippet as vector
- Sort warnings by predicted probability that code is incorrect
Evaluation: Setup

68 million lines of JavaScript code

- 150k files [Raychev et al.]
- 100k files for training, 50k files for validation

<table>
<thead>
<tr>
<th>Bug detector</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
</tr>
<tr>
<td>Swapped arguments</td>
<td>1,450,932</td>
</tr>
<tr>
<td>Wrong binary operator</td>
<td>4,901,356</td>
</tr>
<tr>
<td>Wrong binary operand</td>
<td>4,899,206</td>
</tr>
</tbody>
</table>
# Accuracy of Classifier

<table>
<thead>
<tr>
<th>Bug detector</th>
<th>Validation accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swapped arguments</td>
<td>94.70%</td>
</tr>
<tr>
<td>Wrong binary operator</td>
<td>92.21%</td>
</tr>
<tr>
<td>Wrong binary operand</td>
<td>89.06%</td>
</tr>
</tbody>
</table>
Examples of Detected Bugs

// From Angular.js

browserSingleton.startPoller(100, function(delay, fn) {
    setTimeout(delay, fn);
});
Examples of Detected Bugs

// From Angular.js

browserSingleton.startPoller(100,
    function(delay, fn) {
        setTimeout(delay, fn);
    });

First argument must be callback function
Examples of Detected Bugs

// From DSP.js

for(var i = 0; i<this.NR_OF_MULTIDELAYS; i++){
    // Invert the signal of every even multiDelay
    mixSampleBuffers(outputSamples, ...,
        2%i==0, this.NR_OF_MULTIDELAYS);
}

Examples of Detected Bugs

// From DSP.js
for(var i = 0; i<this.NR_OF_MULTIDELAYS; i++){
    // Invert the signal of every even multiDelay
    mixSampleBuffers(outputSamples, ..., 2%i==0, this.NR_OF_MULTIDELAYS);
}

Should be i%2==0
## Precision

<table>
<thead>
<tr>
<th>Bug detector</th>
<th>Inspected Bugs</th>
<th>Code quality</th>
<th>False pos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swapped args.</td>
<td>50</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>Wrong bin. operator</td>
<td>50</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>Wrong bin. operand</td>
<td>50</td>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>150</strong></td>
<td><strong>95</strong></td>
<td><strong>7</strong></td>
</tr>
</tbody>
</table>
## Precision

<table>
<thead>
<tr>
<th>Bug detector</th>
<th>Inspected Bugs</th>
<th>Code quality</th>
<th>False pos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swapped args.</td>
<td>50</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>Wrong bin. operator</td>
<td>50</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>Wrong bin. operand</td>
<td>50</td>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>150</td>
<td>95</td>
<td>7</td>
</tr>
</tbody>
</table>

68% true positives. High, even compared to manually created bug detectors
Importance of Embeddings

How many true positives do we miss with random embeddings?

- Misses 11 out of 102 true positives
- Example:

  \[
  \text{transform} = \text{is(obj, value)} \mid \text{is(func, value)};\]
Importance of Embeddings

How many true positives do we miss with random embeddings?

- Misses 11 out of 102 true positives
- Example:

```plaintext
transform = is(obj, value) | is(func, value);
```

Bitwise OR for logical OR of booleans:
Inefficient and error-prone
Efficiency

- Data extraction and learning: 28 minutes – 59 minutes (depending on bug detector)

- Prediction of bugs: Less than 20ms per JavaScript file

48 Intel Xeon E5-2650 CPU cores, 64GB of memory, 1 NVIDIA Tesla P100 GPU
Open Challenges

- Bug detection based on other code representations
  - Token-based, graph-based, etc.
  - One representation for many bug patterns

- Support more bug patterns
  - Learn code transformations from version histories
  - A single model for multiple bug patterns
Conclusion

- **Bug detection as a learning problem**
  - Classify code as buggy or correct

- **DeepBugs: Name-based bug detector**
  - Exploit natural language information to detect otherwise missed bugs
  - Learning from seeded bugs yields classifier that detects real bugs

*DeepBugs: A Learning Approach to Name-based Bug Detection* (Pradel & Sen, OOPSLA’18)