Analyzing Software using Deep Learning

Lecture 5: Name-based Program Analysis

Prof. Dr. Michael Pradel
Software Lab, TU Darmstadt
Plan for Today

- **Name-based bug detection**
  Based on "Deep Learning to Find Bugs" by Pradel and Sen, 2017

- **Predicting meaningful identifier names**
  Based on "Context2Name: A Deep Learning-Based Approach to Infer Natural Variable Names from Usage Contexts" by Bavishi et al., 2018
Focus: Natural Language in Code

Source code contains natural language information

Example:

```python
filteredNames = filter(userNames);
sortedNames = sort(filteredNames);
printToScreen(sortedNames);
```
Focus: Natural Language in Code

Source code contains natural language information

Example:

\[ a = b(c); \]
\[ d = e(a); \]
\[ f(d); \]
Source code contains natural language information

Example:

```python
filteredNames = filter(userNames);
sortedNames = sort(filteredNames);
printToScreen(sortedNames);
```

Identifiers convey the intended semantics
Challenges

Identifier names are informal
- Jargon instead of universal vocabulary
- Ambiguity in meaning

Identifiers names are diverse
- Abbreviations, e.g., message versus msg
- Meaningless names, e.g., a, b, c
- Compound names, e.g., LinkedList
Challenges

Identifier names are **informal**
- Jargon instead of universal vocabulary
- Ambiguity in meaning

Identifiers names are **diverse**
- Abbreviations, e.g., message versus msg
- Meaningless names, e.g., a, b, c
- Compound names, e.g., LinkedList

Great match with deep learning: Reasoning about probabilities
Name-based Bug Detection

Goal: Learn a bug detector that reasons about identifier names.

Source code → DeepBugs → Warnings
Motivating Example

What’s wrong with this code?

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

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

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
Overview of DeepBugs

Idea: Train a model to distinguish correct from incorrect code

- Buggy code → Train model → Classifier → Buggy/correct
- Correct code → Train model

New code
Challenge 1: Reason about Names

How to reason about identifier names?

Prior work: **Lexical similarity**
- $x$ similar to $x_{\text{dim}}$

Want: **Semantic similarity**
- $x$ similar to $\text{width}$
- $\text{list}$ similar to $\text{seq}$
Word Embeddings

- Known problem in natural language processing

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

- State-of-the-art technique to learn word embeddings: Word2Vec

- Learn embeddings from context in which a word occurs
  - ”You shall know a word by the company it keeps”
  - Context: Surrounding words in sentences
Word2Vec: Overview

- Given: Sequences of words
  \[ w_1, w_2, w_3, w_4, w_5, w_6 \]

- Represent each word via one-hot encoding
  \[ w_1 \rightarrow 00001000 \]
  \[ w_2 \rightarrow 01000000 \]
  length = size of vocabulary (large!)

- Learn shorter, semantics-encoding representation of \( w_i \)
  from \( k \) surrounding words (= context)
  \[ \text{E.g., } k=4: \ w_3 \text{ has context } w_1, w_2, w_4, w_5 \]

Based on fixed vocabulary of frequent words
Variant 1: Continuous Bag of Words (CBOW)

Predict word from context

\[ h = \frac{1}{k} \cdot U \cdot (\sum_{j \neq i} w_j) \]

\[ y = \text{softmax}(V \cdot h) \]

(here: context of size \( k = 4 \))
Variant 2: Ship-gram

Predict context from word

\[ h = U \cdot x \]

\[ y = \text{softmax}(V \cdot h) \]
Getting the Embedding

Once the network has become good at its task (through training):
Use hidden layer as embedding for word \( w \).
Quiz: Word2Vec

Consider a sequence of words "the quick brown fox jumps over the lazy dog". Which of the following are correct?

- Word2Vec learns an embedding for the entire sentence.
- Word2Vec learns an embedding for each individual word.
- Skip-gram learns to predict how likely it is that "quick" and "brown" occur around "fox".
- CBOW learns to predict the probability that "over" occurs in the context of "jumps" and "the".
Quiz: Word2Vec

Consider a sequence of words “the quick brown fox jumps over the lazy dog”. Which of the following are correct?

- Word2Vec learns an embedding for the entire sentence.
- Word2Vec learns an embedding for each individual word.
- Skip-gram learns to predict how likely it is that “quick” and “brown” occur around “fox”.
- CBOW learns to predict the probability that “over” occurs in the context of “jumps” and “the”.
Word2Vec for Source Code

Natural language
- Sentences
- Words

Programming language
- Program
- Tokens
Word2Vec for Source Code

Natural language
- Sentences
- Words

Programming language
- Program
- Tokens

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

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

Natural language

- Sentences
- Words

Programming language

- Program
- Tokens

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

var x_dim =
var y_dim =
setPoint(y_dim, x_dim);

Context of x:
function - setPoint - (-, -y- )
Challenge 2: Training Data

Effective learning requires millions of examples

- To learn a bug detector:
  Need both correct and buggy code examples

- Most available code is correct: Easy to get correct examples

- How to get many examples of buggy code?
  - Want: Buggy due to the same bug pattern
Generate Buggy Code

Idea: *Artificially introduce bugs*

![Diagram showing the process from Corpus of code to Buggy seeding to Buggy code to Correct code]
Example

For swapped function arguments:

- Visit every function call with $\geq 2$ arguments
- **Positive example**: Original order of arguments
- **Negative example**: Swap first two arguments

\[\text{setPoint}(x, y); \quad \rightarrow \quad \text{setPoint}(y, x);\]
Deep Bugs: Putting Everything Together

Code example
setPoints \((x, y)\)

Extract identifiers

Sequence of identifiers
"setPoints", "x", "y"

Represent as vectors (based on learned embeddings)

Input vector: Concatenate

Hidden layer

Output: Probability that correct
Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values
- Incorrect binary operators
- Swapped operands of binary operations
Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values
  ```javascript
  var callback = function() { .. }
  ```

- Incorrect binary operators

- Swapped operands of binary operations
Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values
  
  ```javascript
  var callback = function() { .. }
  ```

- Incorrect binary operators

- Swapped operands of binary operations
Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values

  ```javascript
  var callback = function() {
    "abc"
  }
  ```

- Incorrect binary operators

  ```javascript
  if (x == undefined) ...
  ```

- Swapped operands of binary operations
Same idea works for other bug patterns

- Assignments of incorrect values
  
  ```javascript
  var callback = function() { .. } "abc"
  ```

- Incorrect binary operators
  
  ```javascript
  if (x >= undefined) ...
  ```

- Swapped operands of binary operations
Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values
  ```javascript
  var callback = function() {
  "abc"
  };
  ```

- Incorrect binary operators
  ```javascript
  if (x == undefined) ...
  ```

- Swapped operands of binary operations
  ```javascript
  bytes[i + 1] >> 4
  ```
Beyond Swapped Arguments

Same idea works for other bug patterns

- Assignments of incorrect values
  ```javascript
  var callback = function() { .. } "abc"
  ```

- Incorrect binary operators
  ```javascript
  >
  if (x !== undefined) ... 
  ```

- Swapped operands of binary operations
  ```javascript
  4 >> bytes[i + 1]
  bytes[i + 1] >> 4
  ```
How Well Does it Work?

- Evaluation with 69 million lines of JavaScript and three bug detectors

- Results
  - 89%–95% accuracy
  - 102 real-world bugs with 68% true positive rate
  - Less than 20 milliseconds per checked file
Examples of Bugs

// From Angular.js

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

// From Angular.js

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

First argument must be callback function
// 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 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
Plan for Today

- **Name-based bug detection**
  Based on "Convolutional Neural Networks over Tree Structures for Programming Language Processing" by Mou et al., 2016

- **Predicting meaningful identifier names**
  Based on "Context2Name: A Deep Learning-Based Approach to Infer Natural Variable Names from Usage Contexts" by Bavishi et al., 2018
Predicting Natural Names

- **Goal:**
  Predict *natural names for variables*

- **Usage scenario:**
  Understand *minified code*
function http(e, n, t, s, i, f) {
    s = JSON.stringify(s);
    e.open(t, n, i);
    if (i) {
        e.onreadystatechange = function() {
            if (e.status == 200) {
                f(e.responseText);
            }
        }
    }
    ...
}
Motivating Example

```javascript
function http(req, url, method, s, i, f) {
    body = JSON.stringify(body);
    req.open(method, url, async);
    if (async) {
        req.onreadystatechange = function() {
            if (req.status == 200) {
                response(req.responseText);
            }
        }
    }
    ...
}
```
Overview of Context2Name

Train a model to predict a suitable name from the way a variable is used.

Corpus of natural code → Machine learning → New code

→ Variable name predictor

→ New code with natural names
Challenge 1: Usage as a Vector

How to represent the usage of a variable as a compact vector?

```javascript
function http(e, n, t, s, i, f) {
    s = JSON.stringify(s);
    e.open(t, n, i);
    if (i) {
        e.onreadystatechange = function() {
            if (e.status == 200) {
                f(e.responseText);
            }
        }
    }
    ...
}
```
Challenge 1: Usage as a Vector

How to represent the usage of a variable as a compact vector?

```javascript
function http(e, n, t, s, i, f) {
    s = JSON.stringify(s);
    e.open(t, n, i);
    if (i) {
        e.onreadystatechange = function() {
            if (e.status == 200) {
                f(e.responseText);
            }
        }
    }
    ...
}
```
Extracting Usage Contexts

- One usage context per occurrence of each variable
- Context = Tokens before & after occurrence

```javascript
function http(e, n, t, s, i, f) {
    s = JSON.stringify(s);
    e.open(t, n, i);
    if (i) {
        e.onreadystatechange = function() {
            if (e.status == 200) {
                f(e.responseText);
            }
        }
    }
    ...
}
```
Compressing Usage Vectors

Compress usage contexts via auto-encoder

Input layer: Tokens in context
LSTM layer
Repeat layer
LSTM layer
Output layer: Tokens again
Challenge 2: Predict Name

Given: Representation of usage contexts

How to predict a suitable name?
Predicting Variable Names

- Learn from corpus of natural code
- Recurrent neural network

Input layer: Sequence of contexts
LSTM layer
Output: Names (length=60000)
Replace Names in Code

- For each minified name, select name predicted with maximum probability.
- Consistently replace all occurrences of the variable.

```javascript
function http(e, n, t, s, i, f) {
    s = JSON.stringify(s);
    e.open(t, n, i);
    if (i) {
        ...
    }
    ...
    ...
}
```
Replace Names in Code

- For each minified name, **select name predicted with maximum probability**
- **Consistently replace** all occurrences of the variable

```javascript
function http(req, n, t, s, i, f) {
    s = JSON.stringify(s);
    req.open(t, n, i);
    if (i) {
        ...
    }
    ...
}
```
Replace Names in Code

- For each minified name, select name predicted with maximum probability
- Consistently replace all occurrences of the variable

```javascript
function http(req, url, t, s, i, f) {
    s = JSON.stringify(s);
    req.open(t, url, i);
    if (i) {
        ...
    }
    ...
}
```
How Well Does it Work?

- Evaluation with **69 million lines** of JavaScript

- Results
  - Correctly predicts **48%** of all minified names
  - Complements existing non-deep learning approaches: **5.3% additional names**
  - Prediction time: **2.9 milliseconds per name**
Open Challenges

- Name-based program analysis is in its infancy
- Many more problems to tackle
  - Bug detection beyond the currently covered bug patterns
  - Suggest better identifier names than those chosen by developers
  - Classify code, e.g., to identify authors
Name-based program analysis

- Use natural language information in source code
- Reason about identifiers based on learned embeddings
- Useful for bug detection and to de-obfuscate source code