Thinking Like a Developer?
Comparing the Attention of Humans with Neural Models of Code

Michael Pradel
Software Lab – University of Stuttgart
Joint work with Matteo Paltenghi
Executive Summary

Direct comparison:

Developers vs. neural models of code

- Humans still (clearly) outperform models
- Partial agreement on what code to focus on
- Models ignore some tokens that developers deem important
- Human-model agreement correlates with prediction accuracy

Should try harder to mimic humans
Neural Software Analysis

Learning developer tools from large software corpora

Source code
Execution traces
Documentation
Bug reports
etc.

Machine Learning

Predictive tool

Neural Software Analysis, CACM’22
Neural Software Analysis

Learning developer tools from large software corpora

Source code
Execution traces
Documentation
Bug reports
etc.

Machine Learning

New code, execution, etc.
Predictive tool
Information useful for developers

Neural Software Analysis, CACM’22
Common Tasks

Type prediction

Code summarization

Bug detection

Code completion

Program repair
Common Tasks

- Type prediction
- Code summarization
- Bug detection
- Code completion
- Program repair

Humans could also do it.

→ Added value: Automation
Understanding Models of Code

- Emphasis of most papers: Accuracy
- Mostly unclear: What do these models actually learn?
  - Intellectually unsatisfying
  - Risk of coincidental accuracy
Developers vs. Neural Models

Do neural models reason about code similarly to human developers?

- If yes: Good news
- If no: Should mimic developers more closely
Methodology
Idea: Compare Humans & Models

- Same task
- Same code examples
- Measure attention and effectiveness

Developers vs. Machine Learning

Neural models of code
Task 1: Code Summarization

```java
{
    if (!prepared(state)) {
        return state.setStatus(MovementStatus.PREPPING);
    } else if (state.getStatus() == MovementStatus.PREPPING) {
        state.setStatus(MovementStatus.WAITING);
    }
    if (state.getStatus() == MovementStatus.WAITING) {
        state.setStatus(MovementStatus.RUNNING);
    }
    return state;
}
```

Input: Method body

Output: Method name

updateState

Dataset: 250 methods from 10 Java projects

*A Convolutional Attention Network for Extreme Summarization of Source Code, ICML'16*
Task 2: Program Repair

```java
public double sqrt(double x, double epsilon) {
    double approx = x / 2d;
    while (Math.abs(x - approx) > epsilon) {
        approx = 0.5d * (approx + x / approx);
    }
    return approx;
}
```

Input: Method with a buggy line

Output: Fixed line

```java
while (Math.abs(x - approx * approx) > epsilon) {
```

Dataset: 16 bugs from QuixBugs (Java) *

* QuixBugs: A Multi-Lingual Program Repair Benchmark Set Based on the Quixey Challenge, SPLASH’17 (Companion)
Capturing Human Attention

- **Goal:** Track human attention while performing the task
- **Approach:** Unblurring-based web interface
  - Initially, all code blurred
  - Moving mouse/cursor temporarily unblurs tokens
Capturing Human Attention

Task 1: Code Summarization

Inspect the code and select the correct method name:

1. testDeepConflictingReturnTypes
2. testAction
3. testInitializingDoesntTakeReadAction
4. testToStringDoesntExhaustIterator
5. disableSyncScrollSupport
6. calculateTimestamp
7. testCorrectProgressAndReadAction

Manager.getInstance()
Capturing Human Attention

- **91 participants**: Undergrads, graduate students, crowd workers
- **1,508 human attention records**
- **5+ records for each of 250 methods**
- **On average per record:**
  - 1,271 mouse-token events
Capturing Human Attention

Task 2: **Program Repair**

- **Help**: Hover to see the instructions again. Hover if you need more information on the algorithm.
- **Submit**: Please press the submit button once you have fixed the bug, or indicate that you are not able to.
- **Snippet Info**
- **Code Editor**
- **Buggy Line**
Capturing Human Attention

- **27 participants**: Software engineers, graduate students
- **98 bug fixing records**
- **5–7 records for each of 16 bugs**
- **On average per record**: 983 unblur events and 13 edit events
Capturing Human Attention

Summarize fine-grained attention record into **attention map**:

```java
public class SQRRT {
    public static double sqrt(double x, double epsilon)
    double approx = x / 2d;
    while (Math.abs(x - approx) > epsilon) {
        approx = 0.5d * (approx + x / approx);
    }
    return approx;
}
```
Model Attention

Task 1: Code summarization

- Convolutional sequence-to-sequence (CNN)
  *A Convolutional Attention Network for Extreme Summarization of Source Code*, ICML’16

- Transformer-based, sequence-to-sequence model
  *A Transformer-based Approach for Source Code Summarization*, ACL’20

- Both models:
  Regular attention and copy attention
Model Attention

Task 2: Program repair

- LSTM-based, sequence-to-sequence: 
  **SequenceR**
  *SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair, TSE’21*
  - Regular attention and copy attention

- AST-based transformer: **Recoder**
  *A Syntax-Guided Edit Decoder for Neural Program Repair, FSE’21*
  - Regular attention only
Results
Human-Model Agreement

Do developers and models focus on the same tokens?

- Given for each code example
  - Human attention vector $\vec{h}$
  - Model attention vector $\vec{m}$
- Measure agreement between them
  - Spearman rank correlation:
    \[
    \frac{\text{cov}(rg\vec{h}, rg\vec{m})}{\sigma_{rg\vec{h}}, \sigma_{rg\vec{m}}}
    \]
Results: Summarization

Human-human agreement:

Developers mostly agree on what code matters most
Results: Summarization

Human vs. copy attention:

Empirical justification for copy attention
Results: Summarization

Humans vs. regular attention:

Lots of room for improvement!
Results: Program Repair

Human-human agreement:

![Histogram of Spearman Rank Coefficient]

- Mean 0.56
- Median 0.59

Developers mostly agree on what code matters most.
Results: Program Repair

Human-model agreement:

Some room for improvement
Divided vs. Selective Attention

How to distribute attention over the given code?

- One extreme: Equally distribute over all tokens
- Other extreme: Focus on a few tokens only
Results: Summarization

More dented curve: Focus on few tokens only
Results: Summarization

No model closely matches developers

Overspecialization to a few tokens

More dented curve: Focus on few tokens only
Results: Program Repair

Focus on **buggy line vs. code context:**

<table>
<thead>
<tr>
<th></th>
<th>Buggy line</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developers</td>
<td>37%</td>
<td>63%</td>
</tr>
<tr>
<td>SequenceR</td>
<td>67%</td>
<td>33%</td>
</tr>
<tr>
<td>Recoder</td>
<td>13%</td>
<td>87%</td>
</tr>
</tbody>
</table>

Again, no model closely matches developers.
Results: Program Repair

Human attention evolves over time:

Models could mimic human behavior:
First understand, then fix
What **kind of tokens to focus on?**

- Different kinds: Identifiers, separators, etc.
- For each kind, compute **distance from uniformity**
  - $0$ means uniform attention
  - $-1$ means no attention at all
  - $>0$ means more than uniform attention
Results: Summarization

Distance from uniformity:

- Identifier
- Separator
- Keyword
- Operator
- String
- Boolean

Distance from Uniformity:

- Regular - CNN
- Regular - Transformer
- Copy - CNN
- Copy - Transformer
- Human
Results: Summarization

Distance from uniformity:

Identifiers are deemed important
Results: Summarization

Distance from uniformity:

Models mostly ignore some kinds of tokens
Results: Summarization

Example from Transformer model:

```java
int status = httpClient.executeMethod(method);  // If (status != 200)
    throw new Exception("Error logging in: " + method.getStatusLine());

document document = new saxBuilder(false).build(method.getResponseBodyAsStream()).getDocument();
xPath path = xPath.newInstance("/response/token");
element result = (element) path.selectSingleNode(document);
if (result == null)
    throw new Exception("Error logging in": error.getText());

myToken = result.getTextTrim();
```

Regular attention of neural model

```java
int status = httpClient.executeMethod(method);  // If (status != 200)
    throw new Exception("Error logging in: " + method.getStatusLine());

document document = new saxBuilder(false).build(method.getResponseBodyAsStream()).getDocument();
xPath path = xPath.newInstance("/response/token");
element result = (element) path.selectSingleNode(document);
if (result == null)
    throw new Exception("Error logging in": error.getText());

myToken = result.getTextTrim();
```

Human attention
Results: Summarization

Example from Transformer model:

```java
log.debug("Requesting new token");
int status = getHttpClient().executeMethod(method);
if (status != 200) {
    throw new exception("Error logging in: " + method.getStatusCode());
}
document document = new saxBuilder(false).build(method.getResponseAsStream()).getDocument();
xPath path = XPath.newInstance("/response/token");
    element result = (element) path.selectSingleNode(document);
    if (result == null) {
        element error = (element) xpath.newInstance("/response/error").selectSingleNode(document);
        throw new exception(error == null ? "Error logging in": error.getText());
    }
    myToken = result.getTextTrim();
```

Regular attention of neural model

Model “wastes” attention on understanding syntax

Human attention
Results: Summarization

Example from Transformer model:

```java
log.debug("Requesting new token");
int status = getHttpClient().executeMethod(method);
if (status != 200) {
    throw new exception("Error logging in: " + method.getStatusLine());
}
document document = new saxBuilder(false).build(method.getResponseBodyAsStream()).getDocument();
xPath path = xPath.newInstance("/response/token");
element result = (element)path.selectSingleNode(document);
if (result == null) {
    element error = (element)xPath.newInstance("/response/error").selectSingleNode(document);
    throw new exception("Error == null? Error: " + error.getText());
}
myToken = result.getTextTrim();
```

Model ignores tokens important to developers

Human attention
Effectiveness

Comparing developers and models w.r.t. their effectiveness at solving the task

- Strengths and weaknesses?
- Can current models compete with developers?
Results: Summarization

Comparing different kinds of methods:

Models underperform on non-trivial methods
## Results: Program Repair

### Success rate during program repair:

<table>
<thead>
<tr>
<th></th>
<th>Plausible patch ratio</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top-5</td>
<td>Top-100</td>
<td></td>
</tr>
<tr>
<td>SequenceR</td>
<td>2/80 (2.5%)</td>
<td>17/1395 (1.2%)</td>
<td></td>
</tr>
<tr>
<td>Recoder</td>
<td>2/80 (2.5%)</td>
<td>10/908 (1.1%)</td>
<td></td>
</tr>
</tbody>
</table>
Results: Program Repair

Success rate during program repair:

<table>
<thead>
<tr>
<th></th>
<th>Plausible patch ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top-5</td>
</tr>
<tr>
<td>SequenceR</td>
<td>2/80 (2.5%)</td>
</tr>
<tr>
<td>Recoder</td>
<td>2/80 (2.5%)</td>
</tr>
<tr>
<td>5-7 developers/bug</td>
<td></td>
</tr>
<tr>
<td>Developers</td>
<td>68/98 (69.4%)</td>
</tr>
</tbody>
</table>

Models are far from human effectiveness
Effectiveness vs. Agreement

Are models more effective when they agree more with developers?
Results: Summarization

Human-model agreement for all vs. accurate predictions:

<table>
<thead>
<tr>
<th>Method</th>
<th>All methods</th>
<th>Methods with F1 ≥ 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN (regular)</td>
<td>0.08</td>
<td>0.24</td>
</tr>
<tr>
<td>CNN (copy)</td>
<td>0.49</td>
<td>0.55</td>
</tr>
<tr>
<td>Transformer (reg.)</td>
<td>-0.20</td>
<td>0.02</td>
</tr>
<tr>
<td>Transformer (copy)</td>
<td>0.47</td>
<td>0.55</td>
</tr>
</tbody>
</table>

More human-like predictions are more accurate
Implications

- **Direct human-model comparison**
  - Helps understand why models (do not) work

- **Should create models that mimic humans**
  - Use human attention during training
  - Design models that address current weaknesses
    - E.g., understanding string literals
Conclusions

- Available for future research:
  - Interface for capturing human attention
  - Datasets of human attention records

- More details:

  *Thinking Like a Developer? Comparing the Attention of Humans with Neural Models of Code*, ASE’21