Bidirectional Transformer Language Models for Smart Autocompletion of Source Code

Felix Binder, Johannes Villmow, Adrian Ulges

AI-based support in software engineering has recently emerged as a research field: Recommenders for software commits [Da16], predicting code changes [Zh19], semantic code search [Hu19] or code captioning [ALY18] have been developed. These are usually based on machine learning components, trained on vast amounts of source code and documentation from open-source platforms such as GitHub. Another challenge – and the subject of this paper – is smart autocompletion: As the developer types source code, an AI-based system suggests names for methods/interfaces to use next. To do so, the system infers the plausibility of method calls from the local code context. Take a look at the following example: The AI system (more specifically, a neural method ranking network) analyzes a position in the current code (red, left), and infers that – out of the class TextField’s methods – addActionListener seems most plausible. The network has learned this suggestion from a vast training set of Java projects on GitHub, which contain similar usages of GUI components as the target code:

```java
public class NameEntry {
    ...
    TextField name;
    ...
    public void setup() {
        name = new TextField(20);
        l = new Label(this.name);
        add(l, Layout.WEST);
        add(name, Layout.EAST);
        h = new NameHandler();
        name.  ???
        pack();
    }
    ...
}
```

We refer to this challenge of ranking an object’s method names by their plausibility in a given code context as method ranking. While previous work has used n-grams [Hi12],

[RheinMain University of Applied Sciences, DCSM Department, Wiesbaden/Germany]
felix.binder@student.hs-rm.de, [johannes.villmow, adrian.ulges]@hs-rm.de
recurrent networks [Wh15] and left-to-right language modeling [In20], we evaluate the transformer network BERT [De18] based on masked language modeling. While masked language modeling has been very successful in natural language processing, it has not been used for method ranking / smart autocompletion in source code yet. Our approach first pre-trains a BERT model (more precisely, RoBERTa [Li19]) on a dataset of 10,414 open-source projects (250 million lines of code) from the GitHub Java Corpus [AS13]. Training is done by masking out tokens (more precisely, BPE tokens [SHB16]) in pieces of source code and forcing the model to predict those missing tokens. We call the resulting model JavaBERT.

To utilize JavaBERT for method ranking, we address the fact that method names may consist of multiple tokens (e.g., add-Action-Listener). We suggest two alternatives:

1. **JavaBERT-unsup**: The pre-trained (unsupervised) JavaBERT is applied by masking out variable numbers of tokens. JavaBERT’s predictions on token-level are then combined in a probabilistic reasoning to predictions on method level.

2. **JavaBERT-sup**: JavaBERT is fine-tuned in an additional supervised training as a binary classifier, estimating whether a certain method call is plausible or not in context.

We evaluate both approaches in quantitative experiments on a set of random samples from the test split of the GitHub Java Corpus. Our results indicate that masked language modeling is surprisingly accurate, with a top-3 accuracy of up to 98%. We also study the impact of different contexts, e.g. only the code up to the target method call, or shorter vs. larger pieces of code.

**References**


