A Common Graph Representation for Source Code and Natural Language

Abstract

Semantic information plays a key role in the code search and synthesis settings. In this work, we propose a graph-based representation for code search and document recommendation which incorporates semantic and relational features from the type system and dataflow graph. We connect this graph to a natural language corpus of developer documents, and demonstrate the effectiveness of a graph-based representation on three downstream tasks: code search, document recommendation and entity linking within developer documentation.

1 Background and motivation

In addition to its syntactic structure, source code contains a rich denotational and operational semantics [Henkel et al., 2018]. In order to reason about code in semantically similar but syntactically diverse settings, developers must build mental models which incorporate semantic features from the call graph and surrounding typing context. Many semantic features, like data and control flow can be represented by a directed acyclic graph (DAG), which admits linear-time solutions to a number of graph problems, e.g. topological sorting, single-source shortest path and reachability queries.

Some programming languages allow users to specify which type of values will inhabit a given variable at runtime. Types allow the compiler to reason about certain properties like nullity [Ekman and Hedin, 2007] and shape [Considine et al., 2019]. While types many not appear explicitly in source code, they can often be inferred from the surrounding context using a dataflow graph (DFG). Java, one of the most popular programming languages today, recently introduced local variable type inference Liddell and Kim [2019], which allows variable types to be omitted, and later inferred by the compiler.

DAGs also have important applications in natural language parsing [Sagae and Tsujii, 2008, Quernheim and Knight, 2012]. Consider the sentences in the previous paragraph, which can be shuffled without altering its meaning. In contrast, deranging the sentences in this paragraph would produce a much less coherent text. Various attempts to represent permutation-invariant properties when constructing language models have been suggested [Vinyals et al., 2015]. These allow us to model semantic relationships between natural language entities, such as co-reference, entailment, dependence and other properties which are difficult to capture with a purely sequence-based approach.

Source code for popular software projects is often accompanied by web-based developer documentation. Such documentation is often represented by tree-based markup languages like HTML or Markdown, which contain a collection of natural language, links to other documents, and instructions for how the content should be visually rendered. Both the link graph and the AST of the parent document contain useful information: the syntax tree describes the text in relation to the other entities in the same document, while the link graph describes the relationshp between the document and related documents or source code entities.

Prior work has explored the association between comments and source code entities [Panthaplackel et al., 2020]. Si et al. [2018] introduce a control-flow representation for source code which incorporates elements of the syntax tree and surrounding context.

34th Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

2 Proposed approach

In order to relate the graph of documents to source code, a heuristic is needed. For source code, a good heuristic is the presence of an unambiguous token. This token can be a code-like fragment or other entity such as text.

It is often the case that two documents share a common token. If the token is rare, the co-occurence indicates they refer to a common entity. But which entity? In order to determine the referent, we need a representation of the surrounding context that . While many documents occasionally link to source code directly, source code very seldomly contains links to a HTML document.

We would like to infer which documents which are relevant to a particular section of code, based on the graph of documents and the graph of code. To infer links between these two domains requires building a multi-relational graph representation. We also need an AST of statically typed computer programs from GitHub. We choose Kotlin, which has a variety of parsing tools for source code [Kovalenko et al., 2019] and natural language [Grella and Cangialosi, 2018].

3 Data availability and computational requirements

References

Breandan Considine, Michalis Famelis, and Liam Paull. Kotlin∇: A shape-safe eDSL for differentiable programming. `https://github.com/breandan/kotlingrad`, 2019.

Torbjörn Ekman and Görel Hedin. Pluggable checking and inferencing of nonnull types for java. *Journal of Object Technology*, 6(9):455–475, 2007.

Matteo Grella and Simone Cangialosi. Non-projective dependency parsing via latent heads representation (lhr). *arXiv preprint arXiv:1802.02116*, 2018.

Jordan Henkel, Shuvendu K Lahiri, Ben Liblit, and Thomas Reps. Code vectors: Understanding programs through embedded abstracted symbolic traces. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 163–174, 2018.

Vladimir Kovalenko, Egor Bogomolov, Timofey Bryksin, and Alberto Bacchelli. Pathminer: a library for mining of path-based representations of code. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 13–17. IEEE Press, 2019.

Clayton Liddell and Donghoon Kim. Analyzing the adoption rate of local variable type inference in open-source java 10 projects. *Journal of the Arkansas Academy of Science*, 73(1):51–54, 2019.

Sheena Panthaplackel, Milos Gligoric, Raymond J. Mooney, and Junyi Jessy Li. Associating natural language comment and source code entities. In *AAAI*, 2020.

Daniel Quernheim and Kevin Knight. Dagger: A toolkit for automata on directed acyclic graphs. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 40–44, 2012.

Kenji Sagae and Jun'ichi Tsujii. Shift-reduce dependency dag parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 753–760. Association for Computational Linguistics, 2008.

Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning loop invariants for program verification. In *Advances in Neural Information Processing Systems*, pages 7751–7762, 2018.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015.