A Common Graph Representation for Source Code and Developer Documentation

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Semantic information plays a key role in the code search and synthesis settings. In this work, we propose a graph-based representation for source code and natural language which incorporates semantic and relational features from both domains. We apply this graph to a parsing a corpus of code and developer documents, and demonstrate the effectiveness of a common graph-based representation on three downstream tasks: code search, document recommendation and link prediction.

1 Background and motivation

In addition to its syntactic structure, source code contains a rich denotational and operational semantics [Henkel et al., 2018]. To effectively reason about code in semantically similar but syntactically diverse settings requires models which incorporate features from the call graph [Gu et al., 2016] and surrounding typing context [Allamanis et al., 2017]. Many semantic features, such as data and control flow [Si et al., 2018] can be represented by a directed acyclic graph (DAG), which admits linear-time solutions to a number of graph problems, including 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]. Various attempts to build semantic representations for natural language have been proposed, notably the pointer network architecture [Vinyals et al., 2015b,a]. Pointer networks help to capture permutation-invariant semantic relations between natural language entities, and have important applications in dependency parsing [Ma et al., 2018], named-entity recognition [Lample et al., 2016], and other tasks where sequence representations fall short. Li et al. [2017] extend this work with a copy-mechanism to handle out-of-vocabulary tokens for source code.

Predicting doc-to-doc and code-to-code is a straightforward application of link prediction [Zhang and Chen, 2018] and code embedding [Gu et al., 2018] techniques, but cross-domain transfer largely remains unsolved. Robillard and Chhetri [2015] first explore the task of predicting reference API documentation, but do not use machine learning. Prior work also associates code comments and source code entities [Panthaplackel et al., 2020] using machine learning, but only in source code files.

Maintainers of widely-used software projects often publish web-based documentation, typically stored in markup languages like HTML or Markdown. These files contain a collection of natural language sentences, markup, and hyperlinks to other documents. Both the link graph and the document AST contain important semantic information: the markup describes the text in relation to the other entities in the document hierarchy [Yang et al., 2016], while the link graph describes the relationship between the parent document and related documents or source code entities. Documents occasionally link to source code, but source code rarely contains links to developer documents.

2 Proposed approach

Our goal is as follows: given a single token and its semantic context in source code or developer documentation, to recommend relevant entities in the either source code or documentation. In order to relate the document graph to source code entities, a heuristic is needed. For source code, a good heuristic is the co-occurrence of a salient token. This token can be a code-like fragment or other entity which refers to the selected token.

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

It is often the case that two documents share a common token. If the token is rare, the co-occurrence indicates they refer to a common entity. But which entity? In order to determine the referent, we need a representation of the surrounding context and the contexts in which the referent occurs.

3 Data availability and computational requirements

Our dataset consists of Java projects collected from the Zeal developer docs, and their accompanying source code, collected from GitHub. All projects have a collection of code and collection of documents.

References

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*, 2017. URL `https://arxiv.org/pdf/1711.00740.pdf`.

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. URL `http://www.jot.fm/issues/issue_2007_10/paper23.pdf`.

Matteo Grella and Simone Cangialosi. Non-projective dependency parsing via latent heads representation LHR. *arXiv preprint arXiv:1802.02116*, 2018. URL `https://arxiv.org/pdf/1802.02116.pdf`.

Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642, 2016. URL `https://arxiv.org/pdf/1605.08535.pdf`.

Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 933–944. IEEE, 2018. URL `https://guxd.github.io/papers/deepcs.pdf`.

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. URL `https://arxiv.org/pdf/1803.06686.pdf`.

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. URL https://github.com/JetBrains-Research/astminer.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016. URL https://arxiv.org/pdf/1603.01360.pdf.

Jian Li, Yue Wang, Michael R Lyu, and Irwin King. Code completion with neural attention and pointer networks. *arXiv preprint arXiv:1711.09573*, 2017. URL https://www.ijcai.org/Proceedings/2018/0578.pdf.

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. URL https://scholarworks.uark.edu/cgi/viewcontent.cgi?article=3346&context=jaas.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. Stack-pointer networks for dependency parsing. *arXiv preprint arXiv:1805.01087*, 2018. URL https://arxiv.org/pdf/1805.01087.pdf.

Sheena Panthaplackel, Milos Gligoric, Raymond J. Mooney, and Junyi Jessy Li. Associating natural language comment and source code entities. In *AAAI*, 2020. URL https://arxiv.org/pdf/1912.06728.pdf.

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. URL https://www.aclweb.org/anthology/W12-6207.pdf.

Martin P Robillard and Yam B Chhetri. Recommending reference api documentation. *Empirical Software Engineering*, 20(6):1558–1586, 2015. URL https://www.cs.mcgill.ca/~martin/papers/cr2014a.pdf.

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. URL https://www.aclweb.org/anthology/C08-1095.pdf.

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. URL https://papers.nips.cc/paper/8001-learning-loop-invariants-for-program-verification.pdf.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015a. URL https://arxiv.org/pdf/1511.06391.pdf.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015b. URL https://arxiv.org/pdf/1506.03134.pdf.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016. URL https://www.cs.cmu.edu/~./hovy/papers/16HLT-hierarchical-attention-networks.pdf.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018. URL https://papers.nips.cc/paper/7763-link-prediction-based-on-graph-neural-networks.pdf.