(brief) Abstract

Evaluate the performance of using a richer AST representation with a Transformer model for code summarization, while ideally also incorporating methodological improvements.

Relevance

Neural code summarization is a topic we've covered quite a bit in this class. It is a popular research question in the intersection of AI and software engineering. More specifically, incorporating additional features, both structural and semantic, has been a common theme in recent neural programming language modeling literature, while Transformers are the dominant architecture for working with sequences. As far as I know, there have been few fruitful attempts to directly incorporate AST information to Transformers: Feng et al. discuss unsuccessful attempts to do so with CodeBERT; Alon et al. suggest that their technique of incorporating AST information in code2seq is superior in this domain to at least a vanilla Transformer; Ahmad et al. report that they were unable to improve the code summarization performance of a tuned Transformer model with linearized ASTs.

This project explores some topics that I saw while working on my midterm paper.

What

- Extend a Transformer implementation with AST information
    - I tentatively plan to use Ahmad et al.'s Transformer model. The code is available, and they demonstrated that their modified Transformer worked well.
        - The CodeBERT authors discuss incorporating AST information, but BERT typically uses a lot of data and compute, ~~potentially well beyond what I have access to~~ (the original BERT by Google using 16-64 TPUs).
            - Supposedly CodeBERT only used 2x P100s (the wording a is a bit ambiguous on whether it's finetuning or the actual pretraining), I have a single 3090, which should be a fair bit more powerful than 1x P100.
    - ~~For the AST representation, I tentatively plan to look into code2seq's model of generating compositional AST path vectors, though there's still thought required on how that will actually fit in a Transformer. Their code is also available.~~
        - ~~Barring that, I'll need to investigate how to fit a tree in a Transformer, for which there is some literature.~~
    - I looked into one of the papers Aria discussed in her midterm paper presentation (An Empirical Study of Transformers for Source Code, https://arxiv.org/pdf/2010.07987.pdf). It surveyed a couple use cases (variable misuse, code completion, function naming) for Transformers for source code, but it did not look at code summarization. The authors outline how to collect the data (and they actually also try to follow the same improved methodology I was interested in).
        - Feasible from a computational perspective, as did all their work on a single GPU and gave training times (2-3 hours per epoch, with model performance plateauing at around 10-15 epochs).
        - They were able to show improvements from incorporating structural information in several tasks, but I find it interesting that despite using the

Ahmad et al. Transformer model, they didn't make a direct comparison on summarization.

- My plan is probably to adapt the best performing AST method that they identified, and compare it with a text only model for summarization.

Why

One thing that bothered me when reading the Ahmad et al. Transformer paper was that they use what seems to have been shown as an inferior AST representation and then claim that AST information was not useful for Transformers in code summarization because Transformers implicitly capture structural features. I don't necessarily disagree with the claim as much as I find that using a weaker representation when richer representations exist isn't as convincing. The topic itself seems relevant to recent research and somewhat untouched (rich AST representation + Transformer), so it could be an opportunity to tread some new ground.

Additionally, the two papers I saw on methodological problems in Transformer research in general and in code summarization motivate me to try to incorporate their recommended practices in my own work.