“TreeBERT: A Tree-Based Pre-Trained Model for Programming Language”

Paper: Xue Jiang, Zhuoran Zheng, Chen Lyu, Liang Li, Lei Lyu

Presentation: David Baines and Matt Buchholz

CSCI 5535: Fall 2023

Agenda

David

1. Context
2. Problem Statement
3. Why This Problem is Interesting
4. Research Up to This Point
5. TreeBERT’s Novel Contribution
6. Drawbacks / Things to Consider
7. Opportunities for Future Research
8. Conclusion
9. Questions and Answers

1. Context

David

- How can learning models “learn” a programming language?
- Pre-trained models are great for learning natural languages (NLs), but not so great for learning programming languages (PLs)
    - Think transformers in general or the the BERT architecture more specifically
- To learn a language, you need some “pre-training tasks” to help you learn the language’s “rules”
    - Models learn these rules contextually (not logically) through a large amount of training data
- For PLs, pre-trained models exist, but they’re based in code sequences instead of something more complete like an abstract semantic tree (AST)
    - Run through many, many code snippets to find patterns, then you’ve “learned” the PL
- Learning ASTs can be difficult, since you don’t know where you are in the tree for a given node

2. Problem Statement

David

There are two main issues when applying pre-trained natural language (NL) models to programming languages (PLs):

- It's difficult to determine a finite set of logical "rules" when a PL is modeled as a sequence of words, such as when training on NLs.
- Learning tasks for PLs don't necessarily follow the same "priority" as learning tasks for NLs.

Thinking of a PL in terms of an AST and not a sequence of words when learning it is a critical paradigm shift in the literature on "learning" PLs.

3. Why This Problem is Interesting

David

- If we can “learn” PLs not through context but their underlying rules of logic, we can:
  - Create better generative PL models
  - Auto-comment
  - Enhance or automate verification
  - Automate performance optimizations
    - Think Lawrence, Emily, and Kartik’s presentation on DrAsync and the “anti-patterns” they discussed
  - Teach PLs more easily
  - Create self-repairing code

4. Research Up to This Point

David

1. Modeling language in a general sense
    a. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (Devlin et al, 2019)
2. Modeling programming languages
    a. "Learning and Evaluating Contextual Embedding of Source Code" (Kenade et al, 2020)
    b. "CodeBERT: A Pre-Trained Model for Programming and Natural Languages" (Feng et al, 2020)
3. Exploring and learning ASTs
    a. "Deep Code Comment Generation" (Hu et al, 2018)
    b. "A Neural Model for Generating Natural Language Summaries of Program Subroutines" (LeClair et al, 2019)
4. Learning models
    a. "Attention Is All You Need" (Vaswani et al, 2017)

5. TreeBERT’s Novel Contribution

Matt

1. TreeBERT, a PL-oriented, tree-based, pre-trained modeling architecture
2. Representing the AST as a set of constituent paths and the introduction of node position embedding
   a. “Tree-Masked Language Modeling” (TMLM): Using an encoder-decoder framework to both learn the structure of the AST but also infer properties of the AST that may not be known yet
   b. “Node Order Prediction” (NOP): When learning the AST, NOP allows nodes to be sequenced
      i. Think “Expr” must come after “then”, which must come after “if”
3. Verify theoretical contributions with empirical tests which show TreeBERT’s performance improvement over existing pre-training methods

5. TreeBERT’s Novel Contribution

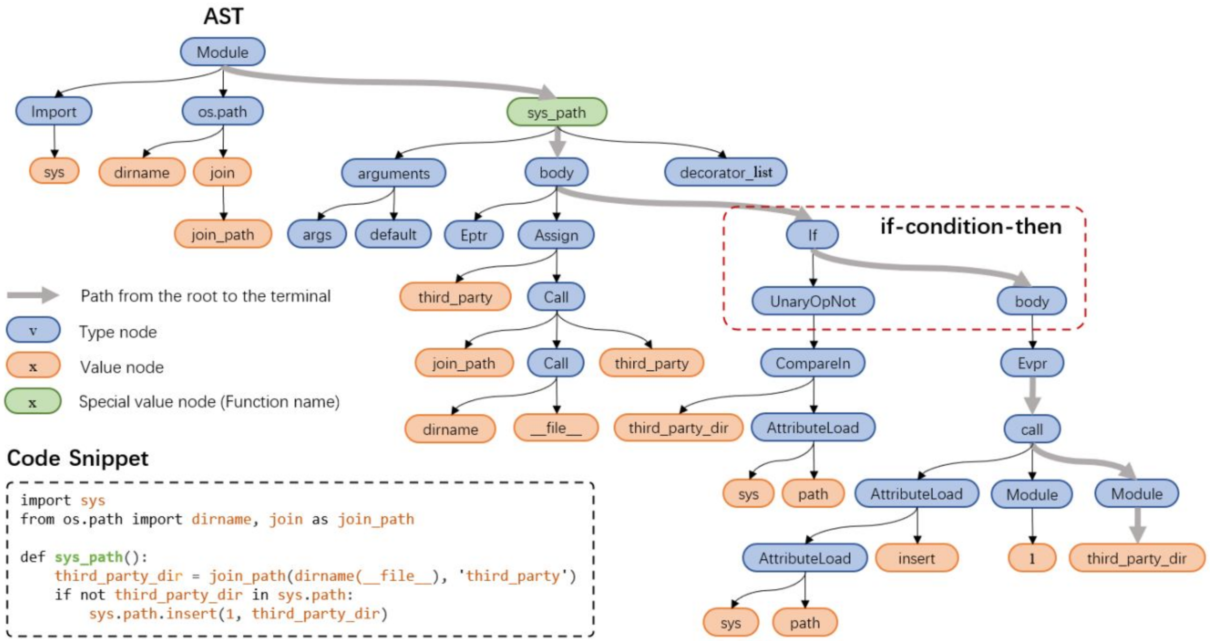

Figure 1: **AST representation of the code snippet**. When we represent AST, the terminal node is its corresponding value attribute, and the non-terminal node is its corresponding type attribute, except for the function name that acts as a non-terminal node but uses the value attribute.

5. TreeBERT's Novel Contribution

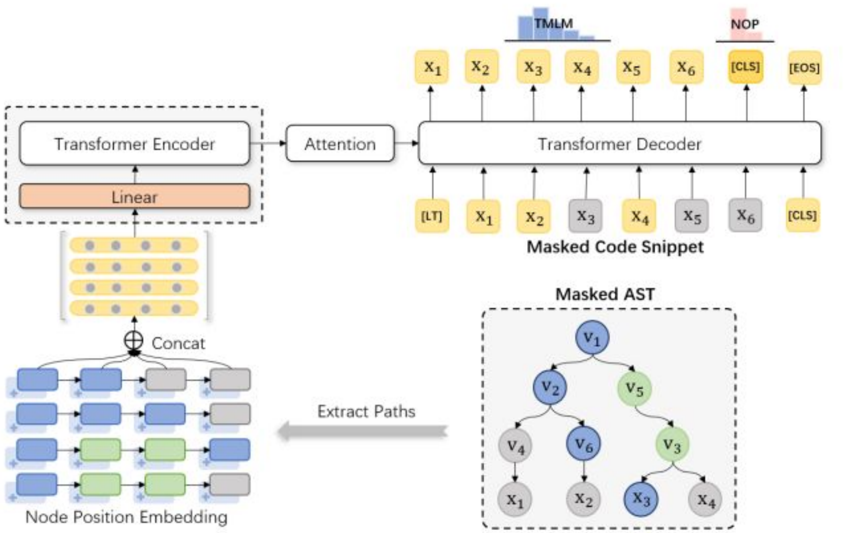

Figure 2: **Overview of TreeBERT.** The gray nodes indicate that the nodes (or tokens) are masked, and green nodes mean that the nodes (e.g., v_3 and v_5) exchange their positions.

6. Drawbacks / Things to Consider

Matt

- What if the PL can't be represented as an AST?
  - Or a "complete" AST? How would we know?
- What if the PL is a compilation of other PLs?
  - Think modules in Python that call C scripts
- Can we use this pre-trained model in low-resource environments where data and / or training resources are limited?
- What does the evolution of the PL mean for training?

7. Opportunities for Future Research

Matt

- Can a TreeBERT-like architecture be applied to LLMs that work specifically with PLs? (Our research project)
  - Can we use TMLM and NOP in LLMs?
- Can a tree-based learning approach be enacted on an AST that is a compilation of smaller, distinct ASTs?
- Besides code documentation and code summarization, what other tasks can TreeBERT be used for?
- How can more information about the AST be gleaned in the training process?
  - Think using AST, graphs, and sequencing simultaneously

8. Conclusion

Matt

- We can learn NLs via modern learning architectures, but PLs can be more difficult
- PLs aren't series of code sequences as much as they're a system of logical rules that can be represented (completely) via an AST
- A tree-based learning approach (TreeBERT) can "learn" a PL effectively, since it's learning the PL's underlying AST
- Defining hierarchical training tasks is a critical element in effectively learning the AST
- The tree-based approach is also more performant that context- or sequence-based approaches
- More work can be done on applying tree-based learning approaches to more complex "PLs", such as LLMs

Thank you.