Deep Learning on Code with an Unbounded Vocabulary

ML4P, July 2018

Milan Cvitkovic, Badal Singh, Anima Anandkumar

In a nutshell

If you're familiar with the following three ideas:

- Abstract Syntax Tree (AST)
- Graph Neural Network (GNN)
- Out-of-Vocabulary (OoV) words in Natural Language Processing

then here's a summary of this work:

> *We develop models for general supervised learning tasks on source code. Our models make predictions by:*
>
> 1. *Parsing the input code into an AST*
> 2. *Adding edges to this AST to represent semantic information like data- and control-flow*
> 3. *Adding nodes/edges to this AST to represent the words (including OoV words) in the code*
> 4. *Consuming this augmented AST with a GNN*

Our problem, and why care

- We’re interested in doing supervised learning on source code
    - Supervised learning task = pairs (input data, desired output)
    - For source code, examples of supervised learning tasks include
        - Suggesting variable names
        - Finding bugs
        - Etc.
- Why bother?
    - It’s hard to hand-craft rules for many tasks, but we may be able to learn rules with enough data

Our starting point: deep models for NLP

- Why deep models?
  - Learn general representations useful for a variety of tasks
- Why NLP models?
  - Natural language closest analog to code among modern ML topics

Summary: Challenges of applying NLP methods to code

- Code semantics are extremely sensitive to syntax
- The vocabulary of written code is unusual
- It isn’t obvious how to read code
- Changes to code matter as much as the code
- Practical challenges

Code semantics are extremely sensitive to syntax

- Natural language sentences can be ill-formed and still get their point across
- Referents are more numerical than natural language
  - Arithmetic comparisons
  - Hardcoded numerical values
- Reuse of terms in different lexical scopes

The vocabulary of written code is unusual

- Natural language is mostly composed of words from a *large, but fixed*, vocabulary
- Code operates over an *unbounded* vocabulary, containing many newly-coined words:
    - Brand names
    - Abbreviations/Acronyms
    - Technical terms
    - Etc.

It isn't obvious how to read code

- Code doesn't have an unambiguous written (or even execution) order
- Most code in a software package isn't relevant to any single query about that package
    - Code typically references many dependencies, most of which are sparsely used

Changes to code matter as much as the code

- The central object of modern software engineering is the diff, not static code
  - True, diffs can be additions of big, standalone blocks of code, but they usually aren’t
- There isn’t an analogous object of study in NLP

Practical Challenges

- It can be hard to get training data, and deep NLP is data-hungry
    - Often can't crowdsource
    - The more advanced the task we'd like to get labeled data for, the rarer those data are
    - Big tech companies have lots of data, but it's not accessible to most
- It can be hard to incorporate models usefully into the development workflow
    - Deep NLP models are often computationally expensive, even in deployment
    - Given the fallibility of machine-learned models, one needs to find inherently safe deployments

Summary: Challenges of applying NLP methods to code

- **Code semantics are extremely sensitive to syntax**
- **The vocabulary of written code is unusual**
- It isn't obvious how to read code
- Changes to code matter as much as the code
- Practical challenges

This work is about addressing (part of) the first and second bullets

Desiderata

- Syntax: Give the model a way to reason about syntactic structure
  - Model should understand relations between syntactic elements
- Vocab: Flexibly handle new words, but recognize old ones
  - E.g. upon seeing method “set_ml4p_dictionary” and variable “ml4p_dict” model:
    - Can utilize the fact that unknown word “ml4p” is in both
    - Can utilize learned understanding of “set”, “dictionary”, and “dict”
  - Usual strategy of fixed vocabulary or character-level understanding doesn’t work

Prior work: deep models for relational data

- Recursive Neural Networks
  - [C. Goller and A. Kuchler, 1996] assumed fixed tree structure
  - [R. Socher et al., 2011] general formulation
  - [M. White et al., 2016] and others use on ASTs of code
- Aggregate representations of children at every node of a tree, process from leaves to root

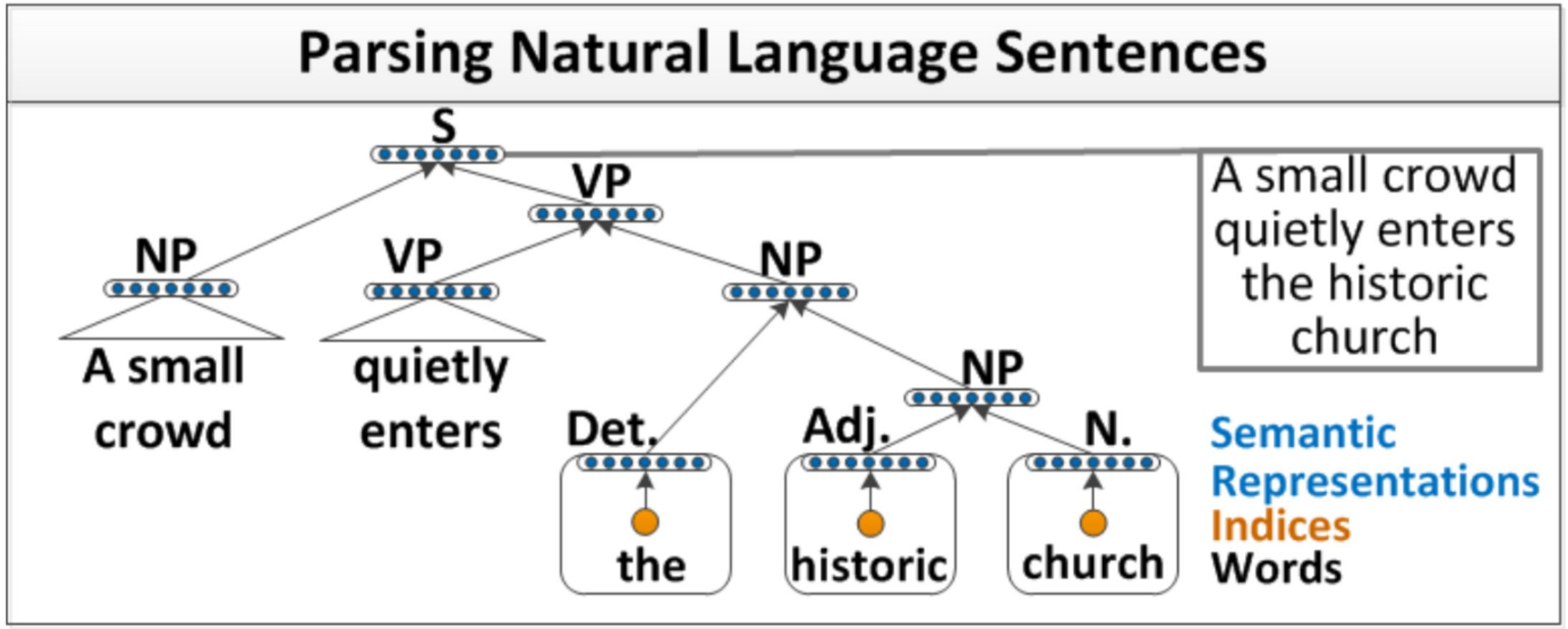


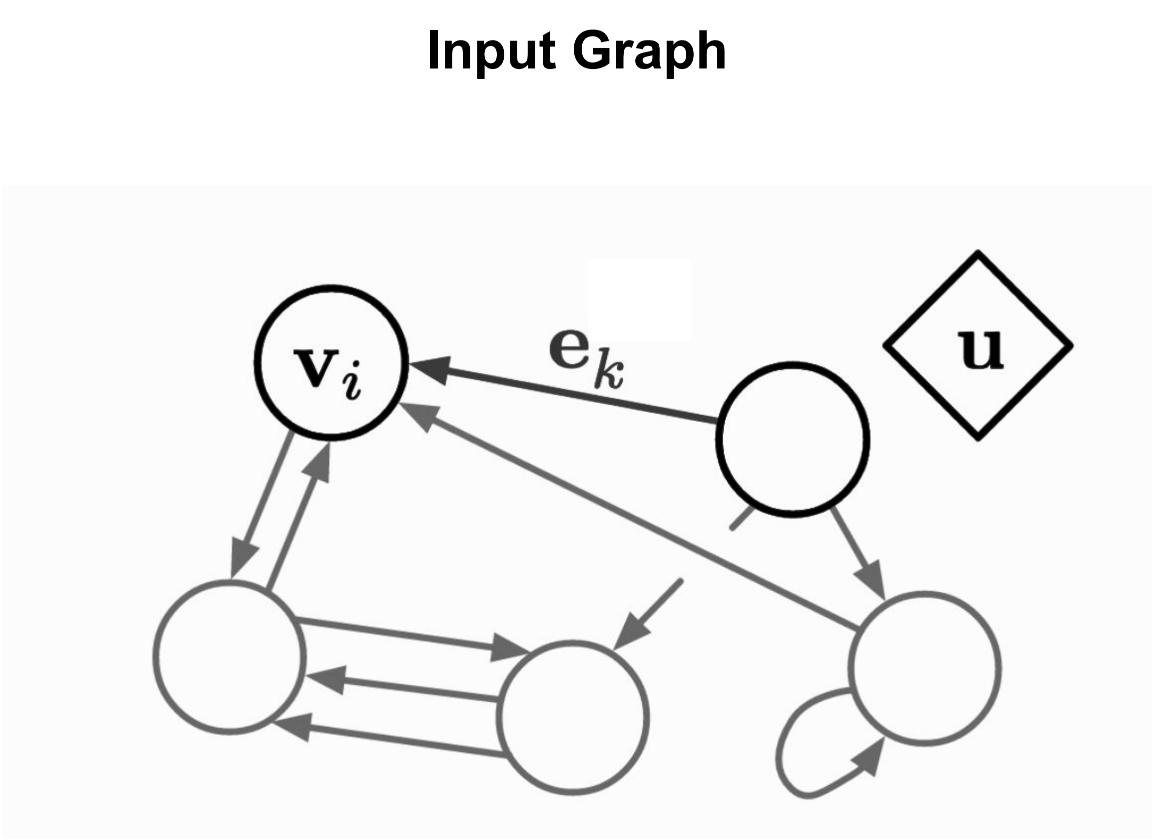
Image credit: [R. Socher et al. 2011]

Prior work: deep models for relational data

- Graph (Neural) Networks
  - Evolved from Recursive Neural Networks [M. Gori et al., 2005]
  - Message Passing Neural Networks framework intro'd in [J. Gilmer et al., 2017]
  - Graph Networks intro'd in [P. W. Battaglia et al., 2018]
  - [R. Kondor and S. Trivedi, 2018] gives rigorous characterization via permutation group representation theory
- Aggregate representations of neighbors at every node (and/or edge), repeat, combine into output
- [M. Allamanis, et al., 2017] and others apply to supervised learning on code

Prior work: deep models for relational data

- Graph Networks [P. W. Battaglia et al., 2018]



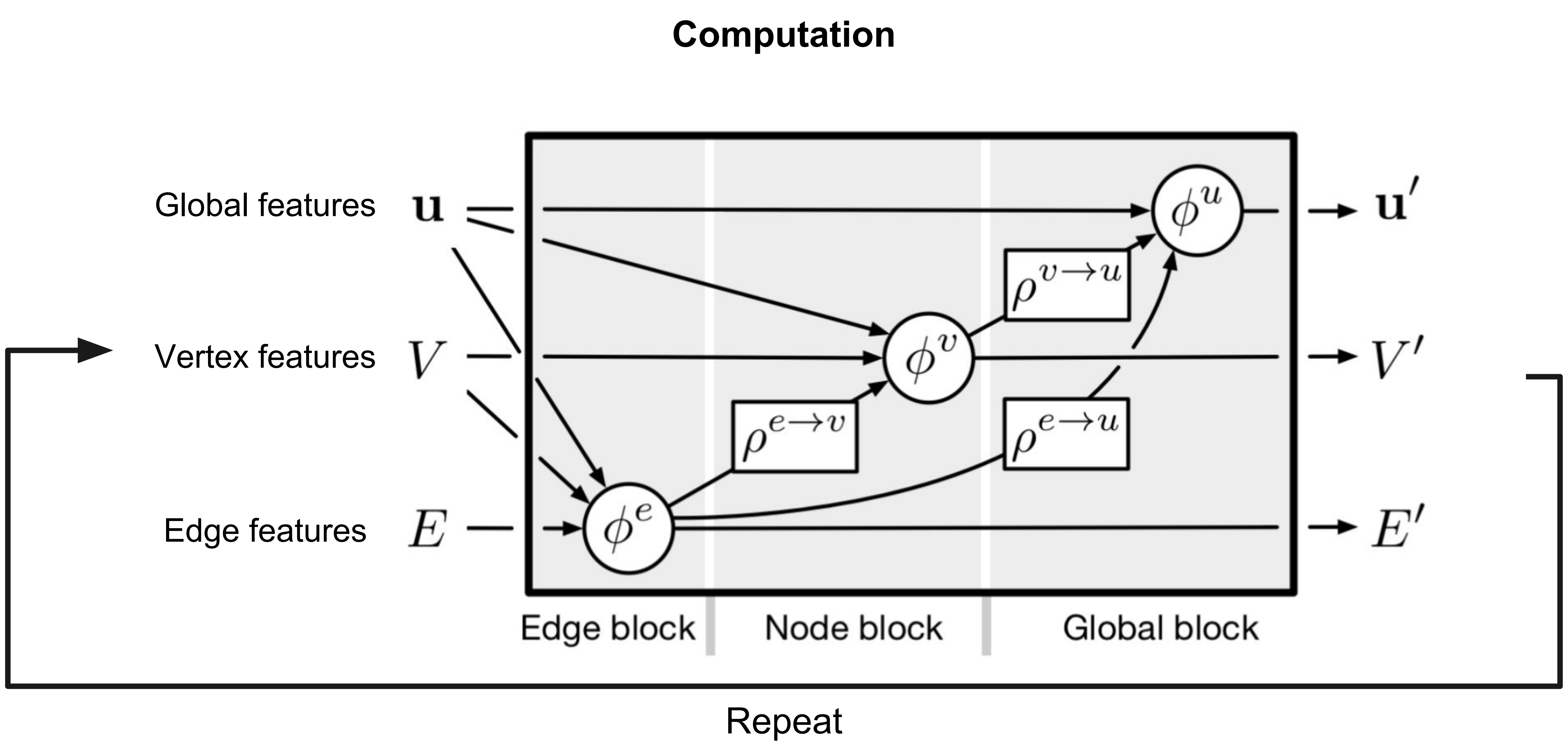


Image credit: [P. W. Battaglia et al. 2011] (slightly modified)

Prior work: unbounded domains of discourse

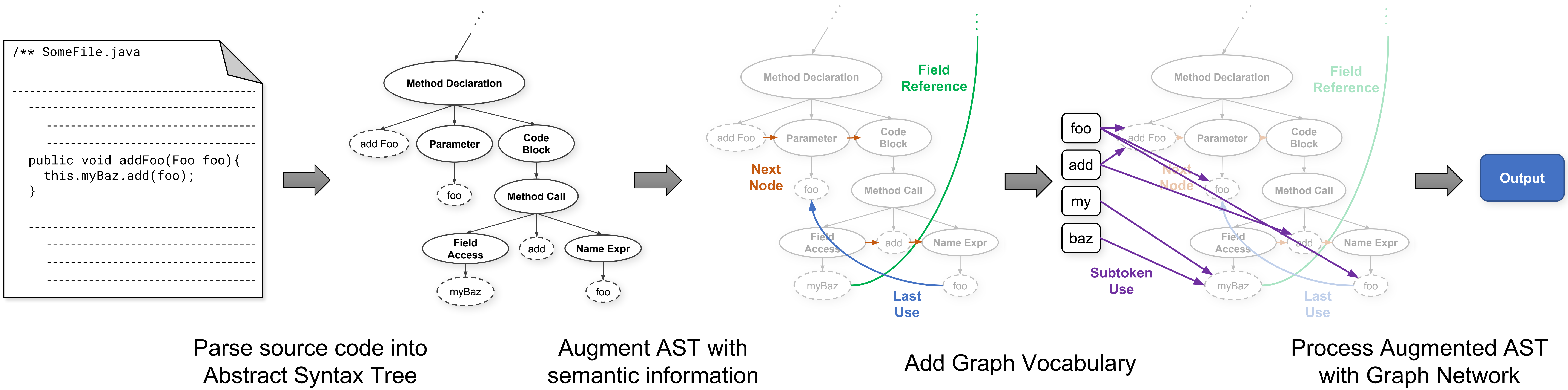
- Neural Attention
  - Networks outputs scalar values for each element of a (potentially variable size) set
  - Larger values = more attention
- Pointer Networks [O. Vinyals et al., 2015]
  - Generate ordered outputs by successively attending ("pointing") to elements of a set
- Pointer Sentinel Mixture Models [S. Merity et al., 2016]
  - Keep a cache of recently seen words in text
  - Can include them in outputs by pointing to them

Our contribution: Graph Vocabulary

- We're already constructing a graph of abstract entities - why not include words?

Our contribution: Graph Vocabulary

- **We're already constructing a graph of abstract entities - why not include words?**
- Our full model:



Fill-In-The-Blank Task

- Task: hide single use of variable in code, model predicts what which variable we hid
- Accuracy:

		Fixed Vocab	CharCNN Only	Graph Vocab (ours)
Unseen files from seen repos	AST	0.58	0.60	0.89
	Augmented AST	0.80	0.90	**0.97**
Entirely unseen repos	AST	0.36	0.48	0.80
	Augmented AST	0.59	0.84	**0.92**

Variable Naming Task

- Task: hide all uses of a variable in code, model generates name via Recurrent NN
- Full-name reproduction accuracy (char-wise edit distance):

		Fixed Vocab	CharCNN Only	Graph Vocab (ours)
Unseen files from seen repos	AST	0.23 (7.22)	0.22 (8.67)	0.49 (3.87)
	Augmented AST	0.19 (7.64)	0.20 (7.46)	**0.53 (3.68)**
Entirely unseen repos	AST	0.05 (8.66)	0.06 (8.82)	0.38 (4.81)
	Augmented AST	0.04 (8.34)	0.06 (8.16)	**0.41 (4.28)**

Takeaways

- Graph Networks allow flexible reasoning over arbitrary entities and their relations
  - Nice way to combine “logical” and “learning” methods while letting both play to their strengths
- Using a Graph Vocabulary:
  - Shouldn’t ever hurt your model - it can always learn to ignore the new nodes
  - Helps in all cases we tried, sometimes significantly

Future Directions

- Many advances in Graph Networks to be tried
    - In particular, adding the right kinds of invariances/equivariances
- Many other entities and relations potentially worth including beyond AST structure and vocabulary
    - Compound words
    - Types (along with their hierarchies)
        - Useful for working with snippets
    - VCS history
        - Useful for working with diffs

Acknowledgments

- Miltos Allamanis
- Hyokun Yun
- Haibin Lin

Our code, for use on your code

https://github.com/mwcvitkovic/Deep_Learning_On_Code_With_A_Graph_Vocabulary--Code_Preprocessor

https://github.com/mwcvitkovic/Deep_Learning_On_Code_With_A_Graph_Vocabulary

Questions, comments, concerns?