Project Area: Formal Analysis of Differentiable Stack Neural Networks

Majors: Computer Science & Linguistics

Advisors: Dana Angluin (Computer Science), Bob Frank (Linguistics)

Background:

A differentiable data structure is a generalized version of a discrete data structure where values outputted from the data structure (represented as real-valued vectors) are a differentiable function of the values inputted to the data structure (which are also vectors). This type of data structure is useful because the differentiability allows it to be connected to a neural network which can then be trained via gradient descent to utilize the data structure to solve some task.

One differentiable data structure of particular interest is the differentiable stack. Due to the connection between stacks and context-free parsing, stacks are a useful tool for parsing hierarchical structure in natural language data. Thus, integrating a stack into a neural network is a promising direction of research for NLP and computational linguistics.
The Project:

Last semester, along with the rest of the Computational Linguistics at Yale (CLAY) lab in the Linguistics department, I worked on implementing a differentiable stack based on the formalism described by Grefenstette et al. We connected this stack to a finite-state controller that could be trained to model linguistic transduction tasks. This model was able to learn how to solve simple tasks (reversing a string, parentheses language modelling, computing bitwise parity, etc.) using intuitive strategies. This project culminated in a paper, but there are still many opportunities for extending it.

In particular, I want to extend the project by investigating the difference between a stack-augmented neural controller and the conventional LSTM. Recent literature has suggested that a good theoretical model for LSTM computation is a finite-state controller augmented with $k$ counter variables (Weiss et al.). Similarly, the direct automata-theoretic counterpart of a stack RNN is a finite-state controller with stack memory (also known as a pushdown automaton). Thus, one way to think about the difference between these architectures is to analyze the difference between the classes of languages that their theoretical counterparts can accept.

There is already existing theory on this topic going back to the 1960s. However, it is not particularly concerned with the real-time case, which is what is relevant for
neural network models. I hope to extend on the existing theory and provide answers to some of the following (or related) questions:

1. What kinds of languages are real-time acceptable by $k$-counter machines, but not by pushdown automata (and vice-versa)?

2. For a language in the symmetric difference of these two classes, can we empirically verify that one type of neural network can learn to model it, whereas the other is not?

3. What additional computational power is obtained by giving a finite-state controller access to both counters and a stack?

4. What is the relationship between the $k$-counter real-time acceptable languages and the mildly context-sensitive languages?

I think this would be a valuable contribution to the field of NLP because it provides some theoretical foundation for the types of models that are often used as black boxes in NLP. From a personal point of view, it would be a meaningful capstone project because it combines ideas from machine learning, automata theory, and complexity theory, which are all subjects that I have enjoyed studying throughout my time at Yale.