Natural Language to Code

Project Proposal

Sai Krishna Karanam karanam.s@husky.neu.edu
Nischal Mahaveer Chand mahaveerchand.n@husky.neu.edu
Varun Sundar Rabindranath rabindranath.v@husky.neu.edu

I. Introduction

The task of converting Natural Language to Code is well established. With recent research, we have achieved basic code generation from annotated text and code comments. Although the techniques used for the same are state-of-the-art, our team feels there are parts that still need improvement.

Take for example the recent paper by Microsoft [1]. The paper describes one such technique where code is generated from the description. The above generates only a limited set of machine instructions and is domain specific. Our team proposes to create a code generator for a high level language, like Python.

With this project, we aim to reduce development time and cost for building efficient, readable, and reliable software. It should be noted that the project does not aim at providing a framework to build industry standard deployable code, but is rather a proof-of-concept. With further research, the former might be possible. As another use case, artificial code generators can also generate exception routines for common exceptions that human programmers sometimes fail to identify, like checking bounds when indexing an array, or checking for zero during dividing. With the introduction of a new language everyday, it becomes more and more important to establish an interface between natural language and programming languages.

For this project, our plan is to use the comments in the code to train models, where the comments would act as labels for training and the code as the expected output. Apart from this, we also plan on using data from other sources as described in the Dataset and Evaluation section.

II. Related Work

Techniques to generate regular expressions [2], input parsers [3], UML diagrams, object oriented class layout, and general purpose code from Natural Language (NL) specification have been researched for a long time. In this project we will focus only on generating executable code from NL.

Recent approaches in converting NL to executable source code predominantly use neural network techniques. These approaches follow two basic patterns. One, is to directly convert NL to source code by modeling the task as a Sequence-to-Sequence conversion task, using Recurrent Neural Networks (RNN) (Lili Mou et al., 2015) [4] to that end. The other is to perform semantic parsing on the NL input and then convert the produced syntax tree to code (Pengcheng Yin et al., 2017) [5]. However, research [5] shows that the second approach is better than the first and emphasizes the use of syntax information of the target language, in NL to code tasks.

We consider the recent research paper by Pengchen Yin et. al. [5] , on syntactic neural model for code generation as the primary motivation for this project. They propose an approach where they translate NL to an Abstract Syntax Tree (AST) first, thus using the grammar of the target language as a prior knowledge. They report a 10% absolute improvement in accuracy compared to the previous state-of-the-art on standard datasets. Since, converting from AST to code can be done deterministically, their system always produces syntactically correct, executable code. This aspect is lacking in the Sequence-to-Sequence models, where the correctness of the generated code is not guaranteed. They use a bidirectional LSTM network for encoding each word, in the NL input, as a context specific embedding and an RNN as a decoder to generate an output AST.

Maxim Robinovich et. al. [6], propose an approach similar to [5] for converting NL to AST. They confirm the experimental results of [5], thus again emphasizing the use of target language syntax. However Maxim Robinovich et. al., have evaluated their system on a wider range of datasets achieving near state-of-the-art results on average.

Pengcheg Yin et.al [5], report that 25% of the errors incurred by their system, on one of the datasets, is due to the generated code only partially implementing the required functionality. We hypothesize that this could be due to insufficient training data leading to the encoder-decoder network not seeing enough data to capture certain semantic nuances. Robin Jia et. al. [7], propose a novel data recombination technique that induces a high-precision generative model from the training data, and then samples from it to generate new training examples. These new training samples capture important conditional independence properties commonly found in semantic parsing. They report that such a technique beats existing state-of-the-art on one of the datasets they evaluated. We hypothesize that adding this technique to Pencheng Yin et. als approach will significantly improve their accuracy.

III. Project Plan

For this project, we plan to have four different phases,

1) Phase 1: *Literature Survey*
   This phase mainly deals with the team reading multiple papers to familiarize ourselves with the terminologies

and jargons used for the task of converting natural language to code.

2) Phase 2: *Modeling*
   The team will build multiple models for the task. At this moment, we plan to implement Hidden Markov Models, Semantic Parsers, and Multilayer Neural Networks.
3) Phase 3: *Evaluation*
   Each model will be carefully evaluated and examined for potential optimizations. The best model will be used for further optimization and improvement.
4) Phase 4: *Testing*
   The selected model will be tested against various data points, finding any weak spots to improve on, or edge cases to cover.

Maxim Rabinovich et. al. report that their system fails to handle an imperative description of complex logic. They also report that their model fails to enforce executeability, well-typedness and semantic coherence in the output. Executeability is handled by [5], while the other issues present open research opportunities. We wish to investigate each of these issues and hope to discover novel approaches to solve them.

IV. Dataset and Evaluation

We consider two datasets, HearthStone and Django. We find that these two datasets are used widely for NL to code generation tasks.

HearthStone is a card based computer game. The HearthStone dataset consists of Python classes implementing the cards, based on the card description. HearthStone dataset is considered difficult due to the terse card descriptions from which the card‘s class code should be generated.

Django is a web development framework written in Python. The Django dataset consists of lines of code in Python manually annotated with NL descriptions. The Django dataset is relatively more general compared to HearthStone dataset, in the sense that it covers a wider variety of programming use cases.

As is commonly used for NL to code tasks, we will use accuracy, the fraction of correctly generated code samples, and BLEU-4 as our primary and secondary metrics respectively.

References

[1] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, “Deepcoder: Learning to write programs,” *arXiv preprint arXiv:1611.01989*, 2016.

[2] N. Kushman and R. Barzilay, “Using semantic unification to generate regular expressions from natural language,” in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 826–836.

[3] T. Lei, F. Long, R. Barzilay, and M. Rinard, “From natural language specifications to program input parsers,” in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2013, pp. 1294–1303.

[4] L. Mou, R. Men, G. Li, L. Zhang, and Z. Jin, “On end-to-end program generation from user intention by deep neural networks,” *arXiv preprint arXiv:1510.07211*, 2015.

[5] P. Yin and G. Neubig, “A syntactic neural model for general-purpose code generation,” *arXiv preprint arXiv:1704.01696*, 2017.

[6] M. Rabinovich, M. Stern, and D. Klein, “Abstract syntax networks for code generation and semantic parsing,” *arXiv preprint arXiv:1704.07535*, 2017.

[7] R. Jia and P. Liang, “Data recombination for neural semantic parsing,” *arXiv preprint arXiv:1606.03622*, 2016.