Neural Program Synthesis With Soft Actor Critic

Lucas Kabela

March 7 2020

1 Objective

This project aims to further deepen the array of techniques applied to neural program synthesis. One such area with numerous unexplored approaches is reinforcement learning [5]. Through drawing inspiration from the works of [1], [7], [9], and [10], this project seeks to formulate programming synthesis as a Markov Decision Process and apply techniques from RL. However, issues with formulating neural program synthesis as a RL task include the sparseness of reward functions, difficulty in generating samples, and consequently issues with convergence.

As such, this project takes inspiration from a recent development in the policy gradient family of actor critic methods, which vastly reduce the variance, improve sample efficiency, and bolster convergence guarantees of reinforcement learning methods. In particular, this proposal seeks to explore Soft Actor Critic [4], which makes use of off-policy samples to retain maximum entropy while simultaneously achieving stability. It is conjectured that the benefits in variance reduction, sample efficiency, and stability of the method will enable more efficient exploration of larger search spaces than previous reinforcement learning methods applied to neural program synthesis.

2 Targeted DSL

To maintain a simple yet expressive program space, we propose to use a DSL of string transformations following [2] and [3]. A program P: $Str \rightarrow Str$

uses a concat operator to concatenate constant strings, substring expressions, or nested expressions. Substrings follow the convention of positive indices from the left, or negative indices from the right, as well as regex matching. Nesting enables simple text transformations in between such as casing or delimiting.

3 Example Input/Output and Specification

Input (and consequentially specification) will be similar to [2] and will take the from of I/O examples, where the input is randomly generated strings, and output is the results of passing these through some target program. The target program itself will be randomly generates by piecing together some number of expressions (capped at length 10 for efficiency purposes), and will inform the reward function.

One such example is learning a name reversal (taken from [2]), which given "John Smith" outputs "Smith, John" by learning a program similar to Concat(GetToken(Alpha, -1), ',' , ' ', GetToken(Alpha,1))

4 Algorithmic Approaches

The first phase of the project will be to use more simplistic RL methods such as REINFORCE and Monte Carlo Tree Search to establish a baseline for RL and policy gradient methods. Once this is completed, a vanilla actor critic method or A2C will be implemented to establish an actor critic baseline. Finally, the project will implement Soft-Actor Critic as mentioned in [4] to gauge the sampling efficiency and convergence of this method in a neural program synthesis setting.

5 Candidate Infrastructure

The policy network will follow the works of [1], [6], [8], and [10]. The model architecture will likely follow a encoder/decoder format, with a sequential LSTM or RNN [8], as such models codify the recurrent structure/syntax of programs. The value network will follow the approach of [10], and will consist of a task encoder, which will be a sequential LSTM or RNN, and a value prediction layer, which can simply be a fully connected network.

6 Conclusion

In conclusion, it is our hope that this project pushes the boundary for reinforcement learning techniques in Neural Program Synthesis by utilizing Actor Critic methods. We plan to primarily build upon the work by [10], and plan to use a DSL of restricted string expressions similar to [2]. Our actor network will then encode observations into continuous space where it will learn a policy to output programs for the task using a encoder/decoder schema. Similarly, our critic network will learn value predictions with a similar schema. We anticipate compute power and sparse rewards may limit the project, but hope to overcome this issue by the vast RL literature, such as replay buffers. We believe such an endeavour may yield surprising results by reducing variance and improving convergence of RL algorithms in Neural Program Synthesis.

References

[1] Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. *CoRR*, abs/1805.04276, 2018.

[2] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o, 2017.

[3] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *PoPL'11, January 26-28, 2011, Austin, Texas, USA*, January 2011.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[5] Neel Kant. Recent advances in neural program synthesis. *CoRR*, abs/1802.02353, 2018.

[6] Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder, 2017.

[7] Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc Le, and Ni Lao. Memory augmented policy optimization for program synthesis and semantic parsing, 2018.

[8] Emilio Parisotto, Abdel rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis, 2016.

[9] Riley Simmons-Edler, Anders Miltner, and Sebastian Seung. Program synthesis through reinforcement learning guided tree search, 2018.

[10] Yifan Xu, Lu Dai, Udaikaran Singh, Kening Zhang, and Zhuowen Tu. Neural program synthesis by self-learning, 2019.