Neural Program Synthesis With Soft Actor Critic

Lucas Kabela

Februrary 21 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 [3]. Through drawing inspiration from the works of [1], [5], [8], and [9], 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 [2], 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

Following approaches such as [7], [8], and [9] which seek to apply stochastic searches to a Markov formulation of program spaces, the targeted DSL will be a restricted subset of X86 or some other form of assembly. This project most likely will follow the work of [9], which made use of "4 CPU registers (%eax, %ebx, %ecx, %edx), and 4 main data transfer / integer instructions (addl, subl, movl, imull), 10 digit numbers, and optionally 4 RAM positions (-0(%rbp), -4(%rbp), -8(%rbp), -12(%rbp))". Results (and time) permitting, the DSL will be expanded to include 4 more CPU registers (%r8d, %r9d, %r10d, %r11d), and the next 4 RAM positions.

3 Example Input/Output and Specification

Input (and consequentially specification) will be similar to [8] and [9] - specification will be of the form input, output from some underlying goal function (the function will be chosen before hand, with random sampling providing the input to the function, and output produced from said function). The neural program synthesis will then output a program in the DSL matching the specification.

One such example is learning a "map" function - that is, given the input [0, 1, 2, 3] in registers [%eax, %ebx, %ecx, %edx] respectively, the output specified is [f(0), f(1), f(2), f(3)]. A concrete example would be "subtract 1", where the specification/goal function output would be [-1, 0, 1, 2]. A program the model would output which could produce this would be

MAP:

```
subl $1, %eax
subl $1, %ebx
subl $1, %ecx
subl $1, %edx
```

4 Algorithmic Approaches

The first phase of the project will be to replicate results from [9] with RL methods such as REINFORCE and Monte Carlo Tree Search to verify and 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 [2] to gauge the sampling efficiency and convergence of this method in a neural program synthesis setting. Stretch goals for the project include moving from the restricted/simple DSL presented in section 2 to a larger DSL capable of bit twiddling programs from Hacker's Delight [7].

5 Candidate Infrastructure

The policy network will follow the works of [1], [4], [6], and [9]. The model architecture will likely follow a encoder/decoder format, with a sequential LSTM or RNN [6], as such models codify the recurrent structure/syntax of programs. The value network will follow the approach of [9], 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 [9], and plan to use a similar DSL of restricted x86 with input/output examples serving as specification. 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. While compute power and transforming the space of programs into a continuous, learnable space may limit the project, 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] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

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

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

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

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

[7] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization, 2012.

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

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