Program Generation as an Evolution Operator

Thibault Gauthier

Czech Technical University in Prague, Czech Republic

Abstract

We propose a framework for program synthesis via recursive program generation, where programs function as generators for other programs. These generators are implemented in a domain-specific language (DSL) inspired by the architecture of Recurrent Neural Networks (RNNs). Unlike traditional genetic programming approaches, our method eliminates the need for explicit mutation and crossover operators. Instead, variation arises implicitly from the stochastic nature of the program outputs.

Basic Idea

Our approach is based on the concept of programs generating programs recursively. Rather than directly solving a specific task, a program defines a stochastic process that produces other programs. These descendants may, in turn, generate further generations of programs. This perspective is inspired by artificial life systems [3], but it is closer in spirit to neural networks generating other neural networks [2]. A key distinction from evolutionary frameworks such as NEAT [6] is that reproduction occurs through program interpretation itself; thus, there is no requirement for an explicit mutation operator.

This research seeks to generalize the successful "Alien coding" approach [1] for program synthesis of integer sequences. In that work, a recurrent neural network (NMT [4]) trained on successful program-sequence pairs serves as an improvement operator over the space of OEIS [5] programs. Synthesizing a program generator as capable as the NMT is the long-term goal of the current framework.

Primitives

We define a compact DSL where programs operate on lists of square matrices of fixed dimensions (M^*). The language includes matrix operations (multiplication, transposition, line permutation, scaling, ...) and list operations (push, pop, and fold). To allow programs to manipulate the probability distributions of their own output space, the language includes *constant matrices representing tokens*, denoted by a hash prefix (e.g., `#relu`). Each `#token` corresponds to a constant matrix where the first column is a one-hot vector for that specific primitive. An example of a randomly generated program in this DSL is shown below:

```
(flip (relu (fold (pop (up (mult #permute (up #up)))) (down (down (up (mult #relu
#pop)))) (mult #rand (transpose (relu #permute))))))
```

Reproduction

From a parent program p, a child p' is produced token-by-token. The program p takes two inputs: its previous output and a one-hot encoded list of tokens generated so far. The output is a list of matrices; the first column of the head matrix is interpreted as a real-valued vector. After a softmax transformation, this vector represents a probability distribution over the DSL primitives. A token is sampled and appended until the program is syntactically complete. Although any of these programs may be used to generate other programs, these programs are

given a level depending on their specific usage. A level-0 program is directly evaluated according to its performance on a given task. A level-1 program generates level-0 programs. In general, a level-n+1 program generates level-n programs.

Experiment

Instead of generating level-0 programs randomly, we evaluate whether an evolving population of level-1 programs could improve the quality of level-0 programs. For this experiment, we choose to score a level-0 program by the number of unique programs it can generate in 100 samples. A parent's score is the score of its best child at each iteration. Each parent has 10 children per iteration. In Table 1, we compared an evolving population of 20 level-1 programs where the top 10 best parents were retained across iterations (Best Selection) against a control where the 10 preserved parents were chosen at random (Random Selection).

	Best Selection		**Random Selection**	
Iteration	Best	Avg	Best	Avg
1	72	63.00	66	61.50
2	74	63.70	72	63.60
3	70	63.90	72	64.00
4	70	63.50	70	63.70
5	68	63.65	73	63.85
6	68	62.80	72	63.30
7	65	61.80	69	62.70
8	68	63.55	69	63.70
9	70	62.90	71	63.50
10	77	64.35	68	63.35

Table 1: Evolution and scores of two populations of 20 level-1 programs over 10 iterations. Average scores are computed on the 10 level-1 programs preserved at the end of each iteration.

It is not clear whether the Best Selection population has an advantage over the Random Selection group. Therefore, a larger population size and more iterations will be necessary to see if the results can be distinguished from random noise.

Future Work

We plan to scale these experiments using CPU-vectorization and introduce primitives that allow programs to access the most successful examples from the level below. This would enable a form of supervised learning within the hierarchy. Finally, we aim to investigate if evolving a population of higher-level programs is beneficial and if it can eventually collapse the hierarchy by learning to produce high-quality versions of themselves.

Resources

The source code is available at: https://github.com/barakeel/evolution.

References

[1] Thibault Gauthier, Miroslav Olsák, and Josef Urban. Alien coding. *CoRR*, abs/2301.11479, 2023.

[2] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[3] Eugene M. Izhikevich, John H. Conway, and Anil Seth. Game of life. *Scholarpedia*, 10(6):1816, 2015.

[4] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[5] Neil J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants, 14th Symposium, Calculemus 2007, 6th International Conference, MKM 2007, Hagenberg, Austria, June 27-30, 2007, Proceedings*, volume 4573 of *Lecture Notes in Computer Science*, page 130. Springer, 2007.

[6] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural network through augmenting topologies. *Evol. Comput.*, 10(2):99–127, 2002.