Improvements in Program Synthesis for Integer Sequences

Thibault Gauthier[1], Miroslav Olšák[2], and Josef Urban[1]

[1] Czech Technical University in Prague, Czech Republic
[2] Institut des Hautes Etudes Scientifiques Paris, France

Abstract

We present improvements for a self-learning algorithm for synthesizing programs for integer sequences. It discovers on its own programs for more than 90,000 OEIS sequences.

Introduction The search for abstract patterns is one of the principal occupations of mathematicians. The discovery of similar patterns across different mathematical fields often leads to surprising connections. When a symbolic representation (e.g. formula) describing a pattern is found, a mathematician can start reasoning and deriving additional facts about the theory in which the pattern occurs. Integer sequences are a very common kind of mathematical patterns. A compilation of such sequences is available in the On-Line Encyclopedia of Integer Sequences OEIS [11] curated by Neil Sloane. Our initial system was able to discover initially patterns, in the form of programs, for 27,000 sequences [5]. In this abstract, we present improvements made to the self-learning system, leading it to discover from scratch programs and explanations for more than 90,000 sequences [3].

Programming Language The programs are synthesized using a programming language containing a few primitives. This facilitate the learning and encourage the system to come up with its own solutions for more complex programs (e.g. prime numbers). It contains the constants 0, 1, 2, the functions $+$, $-$, $\times$, *div*, *mod*, two arbitrary-precision integers variables x, y, the conditional operator *cond* and three looping operators *loop*, *loop2*, *compr*. To generate a sequence of integers, we take a program p that represents a function $f : \mathbb{Z} \to \mathbb{Z}$ and compute $f(0), f(1), f(2), etc.$ If these outputs match all the terms provided for an OEIS sequence on its OEIS web interface, we say that the OEIS sequence has a solution.

Our Approach To find programs generating integer sequences, our approach relies on a self-learning loop that alternates between three phases. During the search phase, a neural network synthesizes programs for targeted sequences. Then, during the checking phase, the proposed programs are checked to see if they generate their target sequence, or any other OEIS sequences. In the learning phase, a neural network trains on these examples to translate the "solved" OEIS sequences into the best (e.g. smallest) discovered program(s) generating it. This updates the weights of the network which influences the next search phase. Each iteration of the self-learning loop leads to the discovery of more solutions, as well as to the optimization of the existing solutions. This approach differs from inductive logic programming systems (such as Popper [10]) as we use statistical learning across many instances to arrive to a solution instead of trying to find logical patterns from a single instance (sequence).

Improvements We have replaced a tree neural network (TNN) [2] by a neural machine translation model (NMT)) [9]. We run the NMT on GPU using larger embeddings (500 vs 64) The NMT is also trained on both the fastest and smallest solutions (instead of only the smallest) for each sequence leading to the discovery of faster programs. Other technical improvements include different training strategies and faster checking techniques.

Results After multiple months of self-learning, The NMT run has currently reached 90000 solutions and is still finding more than 100 solutions every day. By analyzing the solutions, we found that the NMT essentially relied a lot on synthesizing pairing functions to create complex programs. This allows it to essentially use an arbitrary number variables (our language has only two variables) by encoding multiple variables into one. Given 100 additional terms provided in OEIS b-files for 40,577 sequences, we measure the percentage of generalizing programs (producing matching additional terms) to be 90.57. A common error is reliance on an approximation for a real number, such as π.

Resources The code for our project is available in this repository [6]. A user can also test an interactive version of our old system (using the TNN) via the web interface [7]. The first 78,118 sequences and their associated programs which were discovered during the previous experiment can be inspected in this repository [4].

Ongoing Experiments: Memory The pairing functions implemented by the NMT are not very efficient and also makes the final program less readable. Thus, we are currently running an experiment where the variables x, y are replaced by variables xl, yl representing list of arbitrary integers. And we add push and pop operations to store and retrieve integers. This removes the need for pairing functions and we have already observed that the new programs using the pop/push operations are much shorter and more readable than the ones relying on pairing functions. In collaboration with the LODA developers [8], we also currently running experiments with the LODA programming language which is very similar to ours but also has array assignment and lookup as primitives.

Ongoing Experiments: Definitions The process of refactoring and making definitions is an important part of the job of a programmer. We are experimenting with the introduction of local and global definitions [1] to our programming language in the hope that crystallizing commonly occurring patterns in definitions will reduce the size of our programs and facilitate the use of these definitions by the neural network. We already observed that the presence of too many definitions (more than 1000) is detrimental to learning and therefore we should focus on creating a few (less than 100) high-quality definitions instead.

Ongoing Experiments: Mathematical Problems We are currently experimenting with the application of the aforementioned self-learning approach to generate Ramsey graphs and Hadamard Matrices. Given a program implementing a function $f : \mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{Z}$, we can construct a candidate matrix $A = (a_{ij})$ for a Hadamard matrix as follows: $a_{ij} =$ if $f(i, j) \leq 0$ then 1 else -1. Furthermore, we can construct a candidate matrix $B = (B_{ij})$ for the adjacency matrix of a Ramsey graph as follows: $b_{ij} =$ if $f(i, j) \leq 0$ then 1 else 0. During self-learning, objective functions and normalization functions are used to select the 10000 best (with respect to the objective) and unique (with respect to normalization) training examples for the next generation. Early results indicate that solely guessing/conjecturing may not be enough to obtain new results on these open problems. Relying on a combination of statistical conjecture-making steps and deductive-style reasoning steps could be more successful.

References

[1] Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Lucas Morales, Luke B. Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: bootstrapping inductive program synthesis with wake-sleep library learning. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 835–850. ACM, 2021.

[2] Thibault Gauthier. Tree neural networks in HOL4. In Christoph Benzmüller and Bruce R. Miller, editors, *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*, volume 12236 of *Lecture Notes in Computer Science*, pages 278–283. Springer, 2020.

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

[4] Thibault Gauthier, Miroslav Olsák, and Josef Urban. Synthesized programs accompanying the paper "Alien coding". https://github.com/Anon52MI4/oeis-alien, 2023. Accessed: 2023-04-03.

[5] Thibault Gauthier and Josef Urban. Learning program synthesis for integer sequences from scratch, 2022.

[6] Thibault Gauthier and Josef Urban. Software accompanying the paper "Learning program synthesis for integer sequences from scratch". https://github.com/barakeel/oeis-synthesis, 2022. Accessed: 2023-04-03.

[7] Thibault Gauthier and Josef Urban. Web interface accompanying the paper "Learning program synthesis for integer sequences from scratch". http://grid01.ciirc.cvut.cz/~thibault/qsynt.html, 2022. Accessed: 2023-04-03.

[8] Christian Krause. Loda software: an assembly language, a computational model and a tool for mining integer sequences. https://github.com/loda-lang, 2022.

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

[10] Bruce Nielson and Daniel C. Elton. Induction, popper, and machine learning. *CoRR*, abs/2110.00840, 2021.

[11] 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.