Solving One-Third of the OEIS from Scratch

Thibault Gauthier[1] and Josef Urban[1]

Czech Technical University in Prague, Czech Republic

Abstract

We have automatically discovered explanations for about one-third of the sequences in the Online Encyclopedia of Integer Sequences (OEIS). We briefly describe the basic setting consisting of a feedback loop that starts from zero knowledge and iterates between guessing the explanations, their verification, and training of the guessing methods. Then we describe several additions and experiments that led to the current set of solutions found in 600 iterations of the loop. We also analyze some of the solutions discovered.

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 [4] curated by Neil Sloane. In this abstract, we present improvements made to the self-learning system, leading it to discover from scratch programs and explanations for 122,888 sequences in 600 iterations. This improves over our previously published result of 78,000 sequences [1]. We describe the changes to our system that were implemented and their effect on the self-learning loop.

Programming Language The programs are synthesized using a programming language containing a few primitives. This facilitates the learning and encourages 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. We have recently added two extra operators *push* and *pop* to allow our programs to reason about list integers if necessary. This addition allowed the discovery of programs for sequences that are naturally implemented using lists as well as the simplification of existing ones. Such examples will be given during the presentation. To speed up the execution of programs, we have also implemented memoization for our loops. This can only be done when the initial values of the loop do not contain variables. This speeds up the evaluation by a factor of 10 and also allows our system to find solutions that would have been discarded because of the time limit.

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 [3]) 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).

Results About three months and 190 iterations were necessary to reach the 78,000 sequences from scratch, and after a bit more than a year and 600 iterations in total, we obtained solutions for almost 123,000 sequences. During this later year, we experimented with different approaches, described in the following paragraphs, to improve the quality of the programs synthesized by the NMT neural network [2].

New Training Regimes Instead of training on the full dataset at every generation, it can be interesting to only train on a specific part of the data set. These parts of the dataset are created using a simple clustering algorithm based on weighted syntactic features. This creates specialized neural networks that are better at detecting patterns in a particular mathematical domain. We also experimented with increasing the embedding size from 500 to 1000 and with a transformer architecture but the results so far have been inconclusive.

A New Task: Translating Programs to Programs The original task is to translate a sequence of integers into a program generating that sequence. In order to propagate innovations faster across all programs we have experimented with translating programs to programs that are close according to a syntactic distance. In practice, to create a training example for this task, we pick for each program P_1 a program P_2 at random among the 10 closest neighbors of P_1. Since P_1 and P_2 are syntactically close, the neural network trained on such examples can learn to emulate various tasks implemented manually e.g. in ILP, evolutionary, and abstraction-refinement systems, such as *generalization* (changing a term into a variable), *specialization*, replacement of subroutines by shorter or more efficient code, etc. In general, such methods may be useful to emulate various reasoning and inductive/abductive steps that human thinkers might do when playing with the sequences and programs. Running this new translation task, on the set of all programs discovered so far, produces a significant number of solutions (sometimes more than the original approach) at each generation on top of our original approach.

A New Objective: Programs that are Both Small and Fast In our original setup, we are collecting in the evaluation phase only the fastest and the smallest program discovered so far for a particular sequence. Smaller programs typically generalize better [1] and thus are preferred but can be too slow to evaluate. In a recent change, we collected a third solution for each sequence which is a compromise between a small and a fast program. It minimizes the measure $time^{0.5} \times size$. The coefficient 0.5 was experimentally determined to be the best at producing solutions that are different from the fastest and smallest solutions to maximize the diversity of our training data. After adding these extra solutions, we did not observe any significant effect (positive or negative) on the speed of discovery of new solutions.

Resources The code for our project is available in this repository https://github.com/barakeel/oeis-synthesis.

References

[1] Thibault Gauthier, Miroslav Olsák, and Josef Urban. Alien coding. *Int. J. Approx. Reason.*, 162:109009, 2023.

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

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

[4] 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.