Toward artificial intelligence that learns to write code

June 17 2019, by Kim Martineau

Researchers have developed a flexible way of combining deep learning and symbolic reasoning to teach computers to write short computer programs. Here, Armando Solar-Lezama (left), a professor at CSAIL, speaks with graduate student Maxwell Nye. Credit: Kim Martineau

Learning to code involves recognizing how to structure a program, and how to fill in every last detail correctly. No wonder it can be so frustrating.

A new program-writing AI, SketchAdapt, offers a way out. Trained on tens of thousands of program examples, SketchAdapt learns how to compose short, high-level programs, while letting a second set of algorithms find the right sub-programs to fill in the details. Unlike similar approaches for automated program-writing, SketchAdapt knows when to switch from statistical pattern-matching to a less efficient, but more versatile, symbolic reasoning mode to fill in the gaps.

"Neural nets are pretty good at getting the structure right, but not the details," says Armando Solar-Lezama, a professor at MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL). "By dividing up the labor—letting the neural nets handle the high-level structure, and using a search strategy to fill in the blanks—we can write efficient programs that give the right answer."

SketchAdapt is a collaboration between Solar-Lezama and Josh Tenenbaum, a professor at CSAIL and MIT's Center for Brains, Minds and Machines. The work will be presented at the International Conference on Machine Learning June 10-15.

Program synthesis, or teaching computers to code, has long been a goal of AI researchers. A computer that can program itself is more likely to learn language faster, converse fluently, and even model human cognition. All of this drew Solar-Lezama to the field as a graduate student, where he laid the foundation for SketchAdapt.

Solar-Lezama's early work, Sketch, is based on the idea that a program's low-level details could be found mechanically if a high-level structure is provided. Among other applications, Sketch inspired spinoffs to automatically grade programming homework and convert hand-drawn diagrams into code. Later, as neural networks grew in popularity, students from Tenenbaum's computational cognitive science lab suggested a collaboration, out of which SketchAdapt formed.

Rather than rely on experts to define program structure, SketchAdapt figures it out using deep learning. The researchers also added a twist: When the neural networks are unsure of what code to place where, SketchAdapt is programmed to leave the spot blank for search algorithms to fill.

"The system decides for itself what it knows and doesn't know," says the study's lead author, Maxwell Nye, a graduate student in MIT's Department of Brain and Cognitive Sciences. "When it gets stuck, and has no familiar patterns to draw on, it leaves placeholders in the code. It then uses a guess-and-check strategy to fill the holes."

The researchers compared SketchAdapt's performance to programs modeled after Microsoft's proprietary RobustFill and DeepCoder software, successors to Excel's FlashFill feature, which analyzes adjacent cells to offer suggestions as you type—learning to transform a column of names into a column of corresponding email addresses, for example. RobustFill uses deep learning to write high-level programs from examples, while DeepCoder specializes in finding and filling in low-level details.

The researchers found that SketchAdapt outperformed their reimplemented versions of RobustFill and DeepCoder at their respective specialized tasks. SketchAdapt outperformed the RobustFill-like program at string transformations; for example, writing a program to abbreviate Social Security numbers as three digits, and first names by their first letter. SketchAdapt also did better than the DeepCoder-like program at writing programs to transform a list of numbers. Trained only on examples of three-line list-processing programs, SketchAdapt was better able to transfer its knowledge to a new scenario and write correct four-line programs.

In yet another task, SketchAdapt outperformed both programs at converting math problems from English to code, and calculating the answer.

Key to its success is the ability to switch from neural pattern-matching to a rules-based symbolic search, says Rishabh Singh, a former graduate student of Solar-Lezama's, now a researcher at Google Brain. "SketchAdapt learns how much pattern recognition is needed to write familiar parts of the program, and how much symbolic reasoning is needed to fill in details which may involve new or complicated concepts."

SketchAdapt is limited to writing very short programs. Anything more requires too much computation. Nonetheless, it's intended more to complement programmers rather than replace them, the researchers say. "Our focus is on giving programming tools to people who want them," says Nye. "They can tell the computer what they want to do, and the computer can write the program."

Programming, after all, has always evolved. When Fortran was introduced in the 1950s, it was meant to replace human programmers. "Its full name was Fortran Automatic Coding System, and its goal was to write programs as well as humans, but without the errors," says Solar-Lezama. "What it really did was automate much of what programmers did before Fortran. It changed the nature of programming."

More information: Learning to Infer Program Sketches. arXiv:1902.06349 [cs.AI] arxiv.org/abs/1902.06349