ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Prof. R. Wattenhofer

Neural Program Induction

Program Induction is a thrilling field of research that gained significant traction over the past decade due to the advances in machine learning. Simply put, the goal is to have systems that take in some sort of specification of program behaviour and produce a concrete programs that fit this input. It is hard to underestimate how desirable it is to have at least one such system that is reliable.

The problem of program induction or synthesis can be approached in both top-down and bottom-up fashion. Those championing the latter would likely want to learn from input-output examples (i.e. to "program by example"), while those in favour of the former would prefer to process high-level specifications of what the program should do.

Here are a few of the things we'd like to do, some of which we've already started on.

1. **Inducing Functional Programs from Examples.** Given a sequence such as $1, 1, 2, 3, 5, 8, ...$, can we find a short functional program (such as the one computing the Fibonacci numbers) that maps $1, 2, 3, 4, 5, 6, ...$ to the elements of this sequence?
2. **Inferring Context Free Grammars from Positive and Negative Examples.** A paper recently written in our group presents an approach to inferring regular grammars. We now want to move up the Chomsky hierarchy a little and extend to context-free grammars.
3. **Synthesising Sequential Circuits.** Given Boolean tables across time, can we neurally synthesise a sequential circuit that fits them?
4. **Synthesising List Programs.** Given atomic operations such as *reduce*, *map*, *head*, or *sort*, synthesising short linear programs using these atoms from examples is almost a classical problem in program synthesis. We have a solid idea of how we could approach this with neural networks and in greater generality.
5. **Synthesising Short Assembly Programs.** Moving values around from register to register while jumping if conditions are satisfied. It sounds easy, though it really isn't, but the problem can be approached with a particular type of neural networks.
6. **Learning Group Programs.** Actions of (semi-)groups are perhaps the simplest class of programs that can be considered. Can we generalise the search or transformer architectures to make it possible to learn them from examples? Can we generalise graph neural networks to learn these programs from Cayley tables?
7. **Neural Lambda Calculus.** Neural Turing Machines already exist, but they struggle with learning where, when, and what to write into the memory. This is not a problem faced by lambda calculus programs, where the program and the memory are the same thing in rewriting.

Candidate Profile. Generally speaking, a good candidate is a competent programmer in the language of his/her choice, has good knowledge of or solid experience with TensorFlow or (Py)Torch, and is interested in one or more of the following fields: programming language theory, program induction, program synthesis.

Interested? Please contact us to learn more!

Contact

- Peter Belcak: belcak@ethz.ch, ETZ G61.3