TTIC 31230, Fundamentals of Deep Learning

David McAllester, Autumn 2020

AGI: Bootstrapping

Bootstrapping

Programs were first written in binary machine code.

An assembler was written in binary.

A Fortran interpreter was written in assembler.

A Fortran compiler was written in Fortran.

⋮

Will we eventually just describe programs in informal English.

Bootstrapping

Let an ultraintelligent machine be defined as a machine that can far surpass all the intellectual activities of any person however clever. Since the design of machines is one of these intellectual activities, an ultraintelligent machine could design even better machines; there would then unquestionably be an 'intelligence explosion,' and the intelligence of humanity would be left far behind. Thus the first ultraintelligent machine is the last invention that humanity need ever make, provided that the machine is docile enough to tell us how to keep it under control.

I.J. Good, 1969

Representing Functions by Programs

High level scripting languages such as Python seem to be the most productive programming languages for human programmers.

Does Python represent a particularly effective universal learning bias?

Productivity in programming seems to be greatly enhanced by functional expressions (functional programming) and object-oriented programming (objects, classes an inheretitance).

This seems crucial if we want to somehow achieve I. J. Good's intelligence explosion.

The Turing Tarpit

But in theory the choice of programming language does not matter.

For any two Turing universal languages, say Python and Assmbler, there exists an interpreter I for Python written in Assembler where write $I(h)$ for the assember interpreter I applied to Python program h. We then get

$$|I(h)|_{\text{Assembler}} = |h|_{\text{Python}} + |I|_{\text{Assembler}}$$

Bootstrapping layers of language can make the interpreter small.

The Turing Tarpit

$$|I(h)|_{\text{Assembler}} = |h|_{\text{Python}} + |I|_{\text{Assembler}}$$

Up to the additive constant of the interpreter, assembler gives just as good a learning bias as Python.

Yet we know that the choice of language does matter — Python is clearly better than assembler.

Presumably this is because the search over Python is easier than the search over assembler.

Searching over Programs: Levin Search, 1973

Leonid Levin observed that one can construct a universal solver. The solver takes as input a solution tester and returns as output a solution whenever a solution exists.

Levin's solver is universal in the sense that it is not more than a constant factor slower than any other solver for the given tester.

It follows that for any problem in NP, the universal solver provides a P-time algorithm whenever such an algorithm exists.

Levin's Universal Solver

We time share all programs giving time slice $2^{-|h|}$ to program h where $|h|$ is the length in bits of h.

The run time of the universal solver is at most

$$O(2^{-|h|}(h(n) + T(n)))$$

where $h(n)$ is the time required to run program h on a problem of size n and $T(n)$ is the time required to check the solution.

Here $2^{-|h|}$ is independent of n and is technically a constant.

Bootstrapping the Search

The Baldwin Effect: Learning Facilitates Adaptation

In a 1987 paper entitled "How Learning Can Guide Evolution", Goeffrey Hinton and Steve Nowlan brought attention to a paper by Baldwin (1896).

The basic idea is that Learning facilitates evolution — if arm control is learned then arm structure is easier to change.

Meta-Baldwin

The Baldwin effect should apply to brain modules as well, such as vision or the motor cortex.

Learning should facilitate the evolution of brain modules.

Meta-Baldwin: Learning should then facilitate the evolution of learning.

Bootstrapping the Search


Schmidhuber proposes a learning algorithm for learning an optimal universal solver. The Optimally Ordered Problem Solver (OOPS), Schmidhuber, 2002.

The ideas are similar to Meta-Baldwin — we learn to search over programs, or learn the learner.

But as in Levin search, Schmidhuber's optimal problem solver still does an exponential search over programs. This has not produces useful results to date.

END