Jürgen Schmidhuber - PowerPlay: training an increasingly general problem solver by continually searching for the simplest still unsolvable problem (2013)

Tom Rochette <tom.rochette@coreteks.org>

December 21, 2025 — 77e1b28a

0.1 Context

0.2 Learned in this study

0.3 Things to explore

- Given a single task and an agent which is internally composed of millions (or more) of small solver modules, with some modules producing the right solution while others producing invalid solutions, how should the agent pick the solution to use (or indirectly, the appropriate module to use)?
- Given an unconstrained task invention space, how can the PowerPlay agent deal with contradicting (task, solution) combination?
  - Basically, can an agent learn "non-functions?"
- Given that the brain has limited capacity/resources, should we expect it to be unable to learn how to solve new tasks without forgetting?
- It seems that a problem solver that can grow, unlike a pre-wired, unmodifiable topology feedforward NN, is going to be more flexible after the FNN has been saturated (can solve its maximum amount of tasks)
  - If we allow the FNN to "call" other subnetwork FFNs with similar topology, can we conjecture that this makes it as powerful as a program that can have infinite length, or will there be limitations in what it is capable of solving?
- How can a system generate meaningful tasks? It seems to me that the biggest issue is that the system will generate "random" tasks that are of no benefit and that only add a burden to the construction of a more complex agent

1 Overview

- The agent receives as part of its input a task identifier or query that lets it decide what it is trying to solve

2 Notes

2.1 1. Introduction

- Given a realistic piece of computational hardware with specific resource limitations, how can one devise software for it that will solve all, or at least many, of the a priori unknown tasks that are in principle easily solvable on this architecture?

    - How to build a practical general problem solver, given the computational restrictions?
- How do initially helpless human babies become rather general problem solvers over time? Apparently by playing
    - Infants continually seem to invent new tasks that become boring as soon as their solutions become known
    - Easy-to-learn new tasks are preferred over unsolvable or hard-to-learn tasks

2.1.1 1.1. Basic Ideas

- In traditional computer science, given some formally defined task, a search algorithm is used to search a space of solution candidates until a solution to the task is found and verified
- To automatically construct an increasingly general problem solver, let us expand the traditional search space in an unsual way, such that it includes all possible pairs of computable tasks with possibly computable solutions, and problem solvers
    - Given an old problem solver that can already solve a finite known set of previously learned tasks, a search algorithm is used to find a new pair that provably has the following properties:
        * the new task cannot be solved by the old problem solver
        * The new task can be solved by the new problem solver (some modification of the old one)
        * The new solver can still solve the known set of previously learned tasks
- Smart search orders candidate pairs of the type (task, solver) by computational complexity, using concepts of optimal universal search, with a bias toward pairs that can be described by few additional bits of information and that can be validated quickly

2.2 2. Notation and Algorithmic Framework PowerPlay (Variant I)

- The computational architecture of the problem solver may be a deterministic universal computer, or a more limited device such as a finite state automaton or a feedforward neural network
    - All such problem solvers can be uniquely encoded or implemented on universal computers, such as universal Turing Machines
- The problem solver's initial program is called s_0
- There is a set of possible task descriptions $\mathcal{T} \subset B^*$ (B^* is the set of finite sequences or bitstrings over the binary alphabet)

2.3 3. Task Invention, Solver Modification, Correctness Demo

- Three main jobs
    - Task invention
    - Solver modification
    - Correctness demonstration

2.4 4. Implementations of PowerPlay

2.4.1 4.1. Implementation Based on Optimal Ordered Problem Solver OOPS

- The big difference to previous implementations of OOPS is that PowerPlay has the additional freedom to define its own tasks

2.5 7. Discussion

2.5.1 7.6 Opposing Forces: Improving Generalization Through Compression, Breaking Generalization Through Novelty

- Two opposing forces are at work in PowerPlay. On the one hand, the system continually tries to improve previously learned skills, by speeding them up, and by compressing the used parameters of the problem solver, reducing its effective size.

  - The compression tends to improve generalization performance, according to the principles of Occam's Razor and Minimum Description Length and Minimum Message Length
- On the other hand, the system also continually tries to invent new tasks that break the generalization capabilities of the present solver

2.6 8. Words of Caution

- Unlike, say, traditional virus programs, PowerPlay-based systems will continually change in a way hard to predict, incessantly inventing and solving novel, self-generated tasks, only driven by a desire to increase their general problem-solving capacity, perhaps a bit like many humans seek to increase their power once their basic needs are satisfied

3 See also

4 References

- Schmidhuber, Jürgen. "Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem." Frontiers in psychology 4 (2013): 313.