AGI black box

Tom Rochette <tom.rochette@coreteks.org>

August 30, 2025 — 861fb9d0

0.1 Context

The ultimate goal of a simple AGI program is to be able to receive a set of inputs and to produce the desired output while having as little input from the user.

0.2 Learned in this study

0.3 Things to explore

- How do you recognize that a problem is time dependent?
- Should every type of input/output be converted into a generic format such as a number or a tensor?
  - Given the brain functions "only" on electrical current signals, then it should be possible

1 Overview

2 Supported problems

- X to C (constant) mapping
  - (a, b, c, ..., z) -> alphabet
  - (0, 1, 2, ..., 1000) -> number
- X to X mapping (identity)
  - a -> a, b -> b, c -> c, ..., z -> z
  - 0 -> 0, 1 -> 1, 2 -> 2, ..., 1000 -> 1000
- X to Y mapping
  - 0 -> a, 1 -> b, 2 -> c, ..., 26 -> z
  - a -> 0, b -> 1, c -> 2, ..., z -> 26
- Delayed X to Y mapping (the Y value should be produced with a delay t in the output sequence)
  - a -> ., b -> ., c -> a, d -> b, e -> c, ..., z -> x
- X to Y mapping where the mapping changes over time
  - 0 -> 5, 10 -> 25, 0 -> 10, 10 -> 30, 0 -> 12, 10 -> 23

3 Types of inputs/outputs

- Numbers: integer, float
- Strings
- Lists
- Tensors
- Complex data structure

4 Requirements

- The user must define inputs and outputs that are to be learned in order to reduce the problems' space
- In some cases, the black box agent should be able to generate optimal solutions given constraints
- The black box should be told the degree of freedom it has to update itself
  - In some cases we want the black box function to be very rigid
- It should be able to generate simulated inputs and generate outputs, then verify whether its computed output is accurate

5 What the black box can do?

- Record every input/output pair that goes through it

6 Example case

Given a set of numbers mapping to another set of numbers (e.g., 1 -> 13, 23 -> 14, 36 -> 67, . . .), we want the agent to learn as efficiently as possible this mapping.

The best solution for this is to maintain a map of input -> output, and simply output the known output when a know input is received. This is known as the lookup table.
In an optimal case, the agent would be able to discover whether a simpler mathematical formulate appears to generate the output and would replace its lookup table with this function, hence saving a lot of memory at the expense of doing a bit of calculation.

In order to learn the mapping, the agent must receive sufficiently enough examples to build a complete lookup table of the inputs that it will expect to receive over its lifetime. This implies that the more complex/large the lookup table has to be, the more memory the agent needs to have in order to appear accurate. It can exchange memory at the cost of making more mistakes.

An agent for this simple use case has many strategies it can deploy when receiving unseen inputs. It can randomly pick any output value it knows. It can pick the most (or least) common output value. It can use retention policies (similar to cache policies) such as LRU, MRU, LFU, FIFO, LIFO, etc. to determine an element to return.

7 Data structure case

In the case of the data structure inputs/outputs, then the problem becomes highly complex. Each field in itself becomes a problem that needs to be solved on its own as well as together with the other fields.
In most programming languages, integers are bound to a certain range, strings can form infinite sequences of integers

8 See also

9 References

- https://en.wikipedia.org/wiki/Cache_replacement_policies