Mathematics based AGI

Tom Rochette <tom.rochette@coreteks.org>

August 30, 2025 — 861fb9d0

0.1 Context

0.2 Learned in this study

- Much of machine learning is geared toward fitting a given function as closely as possible without under/over-fitting the function in question.

0.3 Things to explore

- Consider getting an optimal AGI as an optimization in itself (gradient descent)
- A lot of the reasoning about AGI in the "Reasoning on AGI" section seems also to relate to the idea of genetically engineered functionalities, where under a certain length no functionality exists
  - This also seems to match the idea of Turing/von Neumann where certain things only emerge after a certain size/complexity

1 Overview

2 Functions

As human beings in relatively well educated societies, we're taught early on about mathematics. We're taught about numbers and how they relate to one another, how to manipulate them, transform them and use them to solve complex problems.

One of the fundamental tools we're presented are `functions`. Functions allow us to transform a set of values (called `domain`) into a second set of values (called `image`).

Functions are easy to think about and are a powerful tool as well. Through an equation, is it possible to express a table of an infinite amount of `domain` → `image` relations.

x	y
0	1
1	2
...	...
n	n+1

This simple relation can be represented with the equation y = x + 1. With that equation, it is possible to represent a relation over an infinite amount of integers (or reals). Thus, we could say that we have reduced the space of all integers over all integers + 1 using a single equation composed of 5 symbols.

In this sense, we can say that functions are generators, in that they are not values until you replace the variables that they contain. Once you do fill out the variables and generate solutions, you are increasing your "generated solution space".

Since functions allow us to express the relations between two sets much more concisely than by enumerating each and every possible case, it means that is must be an important tool in our toolbox for summarizing data sets. In other words, if we are able to produce a function that can replace or approximate a data set as closely as possible, we can reduce the amount of space required to store this data, as long as we're willing to lose in precision and sometimes make mistakes.

Much like lossy compression, functions can be approximated with a certain degree of error. Using approximation allows us to have better compression through generalization at the cost of a higher rate of error.

One can consider a function as two different things:

1. A transformation of inputs into outputs
2. A sequence of transformations applied on the initial output, where the output are assigned to internal variables and possibly transformed by other functions within the initial function (functions calling functions)

3 Integer-based reasoning

Concepts are basically numbers, that is, they can be uniquely represented by an integer value
Everything can be represented by a number (basically the concept of primary key → set of data)
We learn to associate images, sounds, experiences (samples) with a specific concept (number)

4 Logic/Rule-based AGI

Conjecture: An AGI can be built from the composition of millions of rules (if-then constructs).

This conjecture is based on the idea that anything can be reduced to a (binary) string of yes/no (if/then) questions being answered. For instance, to construct this sentence, given the space of lowercase/uppercase letters, we'd have 26 + 26 options to go through each time we'd want to select a letter. This in turn would amount to doing a binary search within this alphabet tree to pick the appropriate letter.

But to simplify our thinking, it's easier to think of question spaces. If we want to do a given action, then we have to enter the appropriate question space, which then has the appropriate questions to answer. Thus, as a whole, the process is always about answering yes/no, but through a higher level layer, it is simpler to think of just picking the right set of questions and computing the appropriate answers. In the end, these values are yes/no (binary).

- How to efficiently pick the appropriate rules to execute?
  Context
- How to pick the appropriate context?
  Rules that define the presence of a context
- How are rules composed?
  Small programs which can evaluate things at time t
- How are rules generated?
  Trial and error (random) then guided by previous successes
- How to evaluate success/failure?
  Observe others successful at the task and derive one or many metrics/evaluation functions
  If no observations are available, consider success as being something different than the last state and failure as staying in the same state. Success/failure is determined by the agent's own sense of value, which may sometimes be controlled by a process similar to reinforcement learning, where certain things make the agent feel worse/better (how "feel" is evaluated is not defined, which is an issue)
- How do you find appropriate examples of success?
  Either the task we want to improve came from observing others (thus we were provided with initial examples) or we want to learn about the rules of a specific domain (which are to be determined)

4.1 Structures for efficient selection of rules

Rules are decomposed into their parts, each part is a key in a dictionary such that when a part is triggered, its corresponding rules are fetched and readied to be intersected with the next triggered part.

4.1.1 Deduction

What is the probability of finding an AGI program? To answer this question, we go through a deductive process which is detailed below.

- the probability of finding an AGI program is based on the existence of such a program
- the probability is based on the minimum length of a program exhibiting AGI, as the set of longer programs will necessarily include this smaller program within them
- thus, past a certain length, we have 100% certainty that the AGI program exist within the set of programs

4.1.1.1 Steps

- a program is composed of an alphabet Σ of s symbols
  - Σ: Alphabet
  - s: Number of symbols in the alphabet
    - s cannot be 0 nor 1 as we need to convey information, which requires at least two symbols (true/false, yes/no)
- the smallest AGI program has a given length l
  - $l = |p_{AGI}|$: length of the smallest AGI
- how many programs are of length $|p|$? $s^{|p|}$
- $s^{|p_{a,b}|}$ number of programs of length $|p_{a,b}|$
- $P(\text{p exhibits AGI} \mid |p_{a,b}| = l) = \frac{1}{s^{|p_{a,b}|}} = \frac{1}{s^{|p_{AGI}|}} = \frac{1}{s^l}$
  - If we want to be more "precise", we would want this to express the number of programs, starting from the empty program, not just the programs of length l, thus, it would be a sum from 0 to l
- $p_{x,y}$ contains $p_{a,b} = p_{AGI}$, thus $|p_{x,y}| \geq |p_{a,b}| = |p_{AGI}|$
- when the length difference $d = |p_{x,y}| - |p_{a,b}|$ increases, we have permutations (where X is any symbol of the alphabet and _ is the program $p_{a,b}$):
  - length d = 2 XX_, X_X, _XX => $3s^2$
  - length d = 3 XXX_, XX_X, X_XX, _XXX => $4s^3$
  - length d = n => $(n+1)s^n$
- $L_{p \to AGI} = |p_{x,y}| - |p_{AGI}|$ the length difference between a program $p_{x,y}$ longer than p_{AGI}
- for $s > 1$ and $l = |p_{AGI}| > 1$, (something is not right with this equation as the probability can go over 1 if $|p_{x,y}| - |p_{AGI}| + 1 > s^{|p_{AGI}|}$)

$$
\begin{aligned}
P(\text{p exhibits AGI} \mid |p_{x,y}| > l \wedge p_{AGI}) &= \frac{\text{Number of programs containing the AGI subprogram}}{\text{Number of programs of length (y - x)}} \\
&= \frac{l}{s^{y-x}} \\
&= \frac{\sum_{i=0}^{l}(n+1)s^i}{s^{y-x}}
\end{aligned}
$$

4.1.2 Observations

- The larger the smallest AGI program is, the more difficult it is to find it in the solution space ($P(\text{p exhibits AGI} \mid |p_{a,b}| = l) = \frac{1}{s^l}$)
  - The difficulty increases exponentially with the length of the program
- As the program p length tends to infinity, the probability it contains the AGI program increases to almost 100% certainty, $P(\text{p exhibits AGI} | \lim_{(y-x)\to\infty} |p_{x,y}|) \approx 1$

5 See also

6 References