General AI Challenge - Round 1

Tom Rochette <tom.rochette@coreteks.org>

November 2, 2024 — 36c8eb68

0.1 Context

0.2 Learned in this study

0.3 Things to explore

- One of the problem of this challenge is to answer the question "which task am I trying to accomplish at this time step?". This is an interesting question because, given a generic event loop program with a "fixed" timestep, we'd like to know which task we should execute within this iteration (should we observe, process observations, compute a step of solution resolution, enact a specific policy, etc.)

1 Approach

- Use a set of solvers
- Each solver tracks its success history
- The solver with the longest sequential reward is selected as the current solver to output an action
  - This could be replaced with the best overall reward over a specified time period
  - When the reward is 0 from all solvers or the reward is the same for a set of solvers, randomly pick one

2 Gradual learning

- Sequence learning
- Reinforcement learning based (using a reward signal)
- Different tasks, no signal as to the type of task to solve or whether the task has changed

3 Problem formalization

- At each time step t
  - The agent receives an input $i \in [0, 255]$
  - In response, the agent emits an action $a \in [0, 255]$
  - The environment emits a reward $r \in \{-1, 0, 1\}$
- During a task T, the agent is expected to produce a sequence of actions A in response to a sequence of inputs I

$$A^*_{i:j} = T_{i:j}(I_{i:j})$$

- $A^*_{i:j}$ the expected/correct actions from time i to time j
- $I_{i:j}$ the inputs from time i to time j
- $T_{i:j}$ the task currently under execution by the environment from time i to time j

- The reward R_j at time step $t = j$ is based on the action A_j, thus indirectly by T and I

$$R_j = \begin{cases} -1 & \text{if } A_j \neq A_j^* \\ 0 & ? \\ 1 & \text{if } A_j = A_j^* \end{cases}$$

- R_j the reward at time j
- A_j the action taken by the agent at time j
- A_j^* the expected/correct action at time j
- Since we want to maximize R over the agent lifetime, we would like to know what action A_j to take at time step $t = j$ to ensure a non-negative reward
  - The action A_j we want to take is A_j^*
  - To determine A_j^*, we need to compute $T_j(I_j)$
  - We do not know T_j, thus cannot compute A_j^*
  - Thus we would like to know what T_j is, so that we may then compute A_j^*
  - We can try to construct $T_j(I_j)$ by providing an A_j and observing the reward
- Given an attempt A_j in response to input I_j,

$$\begin{cases} T_j(I_j) = A_j = A_j^* & \text{if } R_j \geq 0 \\ T_j(I_j) \neq A_j & \text{otherwise} \end{cases}$$

- We can conclude that if the reward at R_j was non-negative, then the appropriate action A_j was taken
- Otherwise, it means that the action A_j taken was wrong
- In the simplest cases, the reward R_j is based on the current input I_j and action A_j, where $A_j = I_j$ (instant reward, no memory)
- In more complicated cases, the reward R_j is based on the current input I_j and action A_j as well as some prior knowledge K, which is a function that maps I to A, $A = K(I)$ for the current task T (instant reward, memory required)
- ... (delayed reward, no memory)
- In more difficult cases, the reward R_j is based on the sequence of inputs $I_{i:j}$ and actions $A_{i:j}$ as well as some prior knowledge K, which is a function that maps I to A, $A = K(I)$ for the current task T (delayed reward, memory required)
- The prior knowledge K is a function that reproduces the behavior expected by the task T (in other words, we would like $K = T$)
  - Given a vector I, K is expected to produce the vector A^*, as T would, in which case a positive reward r at time t will be emitted

$$K_{i:j}(I_{i:j}) = T_{i:j}(I_{i:j}) = A_{i:j}^*$$

- K does not need to be equal T over its whole domain, however it needs to be for the interval i, j evaluated
- At time j, we would like to know if a positive reward is going to be emitted
- At time j, we would like to know the time step interval $i : j$ we need to lookup in order to compute the appropriate action A using our prior knowledge K
- At time j, we would like to know the task T that is currently executed
- Tasks are executed for a certain numbers of steps, after which a new task replaces them. The nature of the task may be the same as the previous one, or it may have changed. In other words, this means that the family of functions the two tasks have are different.
- An evaluation contains multiple tasks execution

$$E = T_1, \ldots, T_n$$

- The agent does not know when a task begins/ends, it is up to the agent to determine that
- In the optimal scenario, the agent knows when a task has begun and when it will end, as well as what the task is

- In a non-optimal scenario, the agent is expected to make as few errors as possible while it determines the new task being executed

3.1 Alternative formalization

- A task T is a program that, given an input I, generates an output A
- A task T is a program that, given an input I and A, generates a reward R
- Given all valid programs P, a task T can be completed by a given subset P_T of programs
- The requirements are that for all inputs I of the task T, a program of the subset P_T must produce the expected output A^*
  - This means that for all other inputs $I_{rest} = I_U \setminus I$, the program $p \in P_T$ may have any behavior it wants (even an infinite loop)
  - This is because the program p is expected to run only for the specified inputs I
- Given that all programs P run in parallel, we expect that for the duration d of a task T, the programs in P_T will perform optimally, which is to say that the sum of the rewards over d is maximal

$$\forall P_T \subset P p(I_i) = T(I_i) \sum_{i=1}^{d} T(I_i) \sum_{i=1}^{d} R(T, I_i)$$

- It can be seen that the task T is an example of a program p that is part of the subset P_T
- There exist other classes of programs in P that are suboptimal
  - $P_{T_{sub}}$ the class of programs that will have a sum of the reward between 1 and the maximum reward - 1
  - the class of programs that will have a reward of 0
  - the class of programs that will have a negative reward

4 Problems

- When have we switched task?/Can we determine the iteration at which a new task began?
  - When we were getting many sequential positive rewards and suddenly we do not anymore
  - When rewards are negative
  - When what used to work (give reward) does not work (give reward) anymore
  - In more advanced tasks, there is enough structure to know when the task instance has changed
- When should we drop all knowledge we think we've acquired since we moved onto a new task?
  - As soon as we're confident enough we're working on a new task
- Is it possible to determine how much previous knowledge we can keep if we discovered we have switched task?
- How can we design a curriculum for which we have to punish as little as possible?
- Can we determine if a given task is easier than another task? Is the Kolmogorov complexity a good metric that can be used as a way to order these?

5 Systems/Functions

(Here, task may be replaced with program)

- Task detector/classifier: Which task appears to be under execution?
- Task switch detector: Have we switched onto a new task?
- Task executor: Execute the task that is thought to be under way.
- Task encoder: Encode the knowledge required to accomplish the task.

5.1 Functions

- Set internal memory: Used to remember previous input and/or input/output that led to a reward.

- Unset internal memory: Used to forget about a previously set memory block as the task may have changed since and the information within the memory block is now irrelevant/incorrect.
- Copy a memory block: Used to copy a memory block within the working memory workspace. The copied memory block can then be manipulated if required.

6 Comments on microtasks

- I think it would make more sense if the microtask 2.4 - Learning to copy input to output was the first task of the agent. The idea is that initially an agent does not know anything. The next thing it should be able to do is mimic the environment, namely, copy it exactly. All the tasks prior to this task assume that the agent knows the other valid symbols that can be inputted/outputted, while a simply copy agent would only need to know about what it has seen so far.

7 Additional microtasks

- Print any of the previous input symbols (of the current task).
- Generate any action in response to any input.
- Return no action (space/silence) to any input.
- Generate anything you know how to generate: Babies are not known to be able to speak right out of the womb. Thus, it makes little sense to expect a machine to do the same. However, babies are able to express themselves through some minimalistic communication methods, which they will improve over time. In a similar fashion, the agent may only be able to express itself through a binary signal, which it would have to learn how to concatenate 0/1 to produce the appropriate byte signal.
- Copy with n-iterations delay.
- Simple Caesar cipher (special case of "Learning a mapping between symbols from input to Output: 1 to 1").
- The goal here would be to induce the complete relation out of a few examples.
- Tasks with stochastic reward: positive and negative rewards are sometimes given, but it may happen we don't return any reward.
- Learning a mapping between symbols from input to output: N to M: Map a single symbol to more than one other symbols. The purpose is to show that more than one answer may be correct.
- Regex learning

The following tasks purpose are to introduce complexity in the agent decisions.

- Task with opposite reward: return positive reward when it should be negative and negative when it should be positive (punishing for the expected behavior).
- Stochastic reward for good/bad behavior.

Note: I need a way to determine in what order these tasks should be accomplished in relations to the other tasks.

8 Other types of microtasks

This section describes other types of ways an agent could learn.

- Multiple stream association: Instead of being given a single byte stream, the agent observes 2..n byte streams and must discover the association between the streams (if any).
- Microtasks attached with an external agent: Different agents have different ways to reward for a given task, some may punish (negatively reward) you for a specific task while another agent may reward you.
- Offer an initial set of input-reward on which the agent can bootstrap itself (learning by examples).
- Have the agent initiate the interaction with the environment, the environment deciding whether or not to reward the agent.

9 Things that have been tried

- Naive byte map.
- A solver per difficulty/problem step, with a solver selector that select the solver that has the currently highest reward streak to determine the next action to take based on input (works up to Micro5Sub1Task).
- A similar system, but instead of writing the solvers myself, I used the learners provided in the test_micro_tasks file.
  - These learner are expected to learn only a single instance, which means they break as soon as a new instance would be given to them. Furthermore, some of the learners have been written knowing the problem subset of byte allowed, which means that it breaks for the general case.
- A solver per difficulty/problem step, with a solver selector that select the action that is the most suggested.

10 Current WIP approach

- Provide the agent with a fixed-size memory of the input/output/reward stream so that it may reason using it.
- Process input as a stream of tokens, similar to how a parser would process a bit/stream of code.
- Similar to how a computer works, an action (operation) would have operands.
- At some point you start to need a callstack and the ability to stack function calls.

11 Ideas

- A system that can copy/move/duplicate/multiply/map symbols
- NTM where one part is dedicated to determining which function/task to use, and an LSTM to process the sequence and use the selected function

12 Questions

- Given two agents, which one is better as an indicator of success:
  - a lifetime reward (positive and negative)
    - Indicates whether it has been more right than it has been wrong
  - a lifetime positive reward (positive only)
    - Indicates only the number of times it was right
    - Does not consider mistakes

13 TODO

- Observe the input samples presented
- Learn from the mistakes of others (most solvers are only learning the good behavior, but not the bad ones)
- Extract a list of features based on the different micro tasks and build a neural network using this feature vector

14 Observations/Notes

- In a multi-agent system, it is only when the system takes the action of the agent that this agent may be directly punished/rewarded. If its action is not taken, but the other action ends up being rewarded, it does not necessarily mean that the agent action would have been wrong (we cannot conclude anything).
- Gradual learning does not mean that we can't learn orthogonal tasks (or that we have to learn by composition over previous tasks all the time). Learning new things without having to reuse existing

knowledge makes it generally easier to grasp the new concept, which can then be merged with existing concepts.

- To accomplish a task, you need to have the basic components to get to the solution. If we were to map this to the linear algebra domain, we'd say that the the basic components are basis vectors, for which a task is the composition of a sequence of basis vectors and where a solution is a point in space. Thus, the through the composition of basis vectors it is possible (or not) to get to a solution. Furthermore, like in linear algebra, if you do not have the necessary basis vectors (components) to reach all point in the space, it may be impossible to reach certain solutions.

14.1 Transcribed notes

- Sequence of inputs, parts of a more complex input
- Hierarchical recognition of more complex constructs
- Construction of complex token detection and association with a task
- State machine based on token parsed
- Input chains and output chains (chains are sequence of symbols)
- Recognize/Categorize similar types of tasks by observing the input/output/reward streams
- Adaptive attention span, which increases the length of the memory buffer allocated to tasks as we progress toward more difficult tasks
- How do you get an agent that is more advanced to recognize earlier tasks?
  - Observe positive/negative rewards (and their associated input/ouput chains)
  - Hypothesize what the current task is
    - Implies that the agent knows how many different tasks there are and what those tasks are
      - Might be open to unsupervised learning/clustering?
- How do you recognize you are dealing with "commandX" vs "c" "o" "m" "m" "a" ... (a particular sequence vs a random sequence)?
- How do you detect (input) and construct (output) structure?
- Track the sequences of symbols and their count
  - Variable length buffer and the count for each time they were seen

Length	States
1	$255^1 = 255$
2	$255^2 = 65025$
3	$255^3 = 16581375$
4	$255^4 = 4228250625$
5	$255^5 = 1.08 \times 10^{12}$
...	...
n	255^n

- How can you reconstruct the task set only given the input/output/reward stream?

15 Similar problems

- Predicting the stock market (when are we moving to a different trend/problem? Based on the current trend/problem, what is the appropriate action?)

16 Ideas based on Andreas Ipp solution to the General AI Challenge: Round 1

Source: https://mirror.general-ai-challenge.org/data/andreasipp_whitepaper.pdf

- Code is DNA

- A genetic algorithm generates alternative code in order to accomplish a given task
- The biggest problem is the size of programs that needs to be generated (in his paper, he lists a 152 bytes long program, and he has 42 ops)
- How many operations are part of DNA? (a sequence of nucleotide forms a op, like a sequence of bits forms an op)
- How can the genetic algorithm decompose a complex task into subtasks it can learn and create the appropriate modules to solve

17 See also

- Machine Super Intelligence

18 References

- https://mirror.general-ai-challenge.org/challenge_first_round_specifications.pdf
- https://mirror.general-ai-challenge.org/data/gradual_round_evaluation.pdf
- http://wiki.opencog.org/w/CogPrime_Overview#Measuring_Incremental_Progress_Toward_Human-Level_AGI