OpenAI:

Emergent Tool Use from Multi-Agent Interaction

Report

Iñigo Ortega, Andoni Rodríguez

October 8, 2019

Subject: **A**dvanced **T**echniques in **A**rtificial **I**ntelligence

Degree: Computer Science

School Year: 2019/2020

Contents

1 Background

1.1 General explanations

1.1.1 Main Game: Hide and Seek

The main objective of this experiment was to make some agents play against each other on a *Hide and Seek* game in a physics-based environment. Hiders task is to avoid getting sighted by seekers, and seekers' task is to keep vision on hiders.

The line-of-sight of an agent is a cone of 135 degrees in front of it. In case of seekers, if an entity is not in that range it is masked on the policy. Agents also have a 30 range sensors arrayed evenly around them. In addition, each agent can observe his and the opposite team as well as how much time is left in the preparation phase.

On the environment *Hide and Seek* game is carried out, there are objects scattered around the floor with which agents can interact by grabbing or pushing them. There are also some randomly generated walls and rooms (sometimes without doors) that agents must learn to navigate, but cannot be modified in any way by them.

Before the game actually starts, there is a preparation phase in which hiders are given a chance to run away or change the environment while seekers are immobilized.

Each game instances are called episodes. Each one lasts 240 time steps and the first 40% of them are used on the preparation phase.

1.1.2 Learning Policy

There are no explicit incentives for agents to perform any specific action. The only supervision given is through the *Hide and Seek* objective.

Agents are given a team based reward depending on how they perform during an episode. Hiders are given a reward of 1 if all of them were able to successfully hide away from seekers. However, they are given a -1 if any hider is seen by a seeker.
Seekers are given the opposite reward: -1 if all hiders are hidden, otherwise 1.

To avoid seekers running away infinitely, agents are penalized if they go too far away from the play zone.

1.1.3 Environment Specifications

The environment is simulated on the *MUJOCO* physics engine.

Besides randomly generated static walls and rooms, the world is populated with 1 to 3 hiders, 1 to 3 seekers, 3 to 9 movable boxes, which at least 3 are elongated and 2 movable ramps.

Agents are simulated as spherical objects and can choose between 3 action types every time step. They may *move* by setting a discretized force along their x and y axis and rotate around their z-axis.
They have a binary *grab* action for binding themselves to the closest object within a predefined distance, but only one at a time and the agent has to be directly looking at it.
Agents are also able to lock objects in place to avoid agents on the opposite team manipulating them. That's because only team members of the agent that locked an object can unlock or move it, including itself.

1.2 Policy

Agents are trained using *self-play* which acts as a natural curriculum as agents always play against opponents of an appropriate level.

Agent policies are composed of two networks with different parameters: a policy network which produces an action distribution and a critic network which predicts the discounted future returns.

Policies are optimized using Proximal Policy Optimization (PPO) and Generalized Advantage Estimation (GAE).

Execution is decentralized. That means, each agents acts only given its own observations and memory state.

Training is centralized, which means that at optimization time no information is masked due to visibility like it is on execution. Then a centralized omniscient value function is used for each agent.

There is also a 5% chance of using a policy uniformly sampled from past versions, which is used to improve policy robustness.

1.3 Auto-curricula and Emergent Behavior

As agents train against each other in *Hide and Seek*, new strategies emerge solely by the result of autocurriculum induced by multi-agent competition which creates a previously non-existing pressure for agents to progress to the next stage. That means agents are not incentivized for interacting with objects or for exploring. Consequently, agents discover new strategies and counter-strategies by themselves.

In all stages of strategy agents must learn to coordinate within their team. Because of that, hiders learn efficient ways of labor division, for example, when building a shelter, each agent separately goes for his own box. Furthermore, hider teams of 2 or 3 members are able to lock 25% and 36% more boxes, respectively, than a single hider team, indicating teams with many members are able to divide labor and accomplish their objectives.

1.4 Theoretical Foundations

1.4.1 State of Art

Very first approaches in this area were made using genetic algorithms.

Years later, first trials were made using the RL, but with different evaluating methods such as:

- Count-based exploration, where agents are incentivized to reach infrequently visited states by maintaining state visitation counts.
- Transition-based methods, in which agents are rewarded for high prediction error in a learned forward or inverse dynamics model. In this case, the used method is intrinsic motivation method, as an alternative exploration baseline to multi-agent autocurricula.

1.4.2 RL description

Even if it has not mentioned before, the base of this experiment is the Reinforcement Learning (RL). To have an idea, RL is founded on behaviourist psychology, as agents in an environment have different rewards depending on their actions. But how?

To understand this, we can take the Pavlov's Dogs idea. In this experiment, a dog was sat in front of a cup of water. When water was served, the saliva of the dog was measured and a bell rang. This water serving-bell ringing process was repeated until dog's saliva quantity was as big as when it saw the cup. So, he learned that the cup and the bell were related.

Therefore, the correlation between psychology and AI is easy to see now. The main idea is that an agent (or many agents, or agent groups) has to choose between different actions which of them is the best one. Many trials will be done and the agents is in theory going to learn which of those actions are rewarded or penalized. As we work on computers, this trials can be made so fast and we can train the agents to do hardly anything, as the "want" to maximize their rewards. Therefore, this RL techniques are so useful for different areas such as linear programming,

2 Author Contributions

2.1 Their results

When the simple environment has been tried with many agents, results tend to be next ones as more episodes happen:

- Episode 0, Random: The agents move randomly, so rewards are similar for both groups.
- Episodes 0–2.69 million, Chasing: Seekers learn to chase hiders, do rewards are much bigger for chasers.
- Episodes 2.69–8.62 million, Door Blocking: Hiders learn to grab and move boxes to block the doors, so rewards change drastically in favour of hiders.
- Episodes 8.62–14.5 million, Ramp Use: Seekers learn to use the ramp to jump over obstacles, so rewards are again traded-off.
- Episodes 14.5–43.4 million, Ramp Defense: Hiders learn to move the ramp inside the room to prevent seekers from using it, causing the game get stuck.

In addition, when the same idea is used in different and more complex environments, same development happens: chaos at the beginning, then a learning phase where agents interact with the available tools and finally a finishing state where they get stuck.

In those more complex environments, episode numbers are considerably higher and more unexpected strategies may appear.

2.2 Their measurement

In the experiment intrinsic motivation has been qualitatively compared. However, as environments increase in scale, so will the difficulty in qualitatively measuring progress. Therefore, they propose using a suite of domain-specific intelligence tests that target capabilities we believe agents may eventually acquire. These tests consist of duties that agents must do and that are easy to quantify:

- Object counting: The agent is pinned in place and asked to predict how many objects have gone right or left, testing the agent's memory and sense of object permanence.

- Lock and return: The agent must find the box, lock it, and return to its original position, which tests the agent's long term memory of its location.
- Sequential lock: The agent must lock boxes in an order unobserved to the agent. Boxes can only be locked in the correct order, so the agent must remember the status of boxes it has seen.
- Blueprint construction: The agent must move boxes to the target locations.
- Shelter construction: The agent must construct a shelter around the cylinder.

2.3 Their unfinished

There were also unexpected behaviours that no one could explain, like:

- Box surfing: Since agents move by applying forces to themselves, they can grab a box while on top of it and "surf" it to the hider's location.
- Endless running: Without adding explicit negative rewards for agents leaving the play area, in rare cases hiders will learn to take a box and endlessly run with it.
- Ramp exploitation: Hiders abuse the contact physics to remove ramps from the play area or, on the case of seekers, to launch themselves upward.

3 Personal Opinion

3.1 Software testing

In order to test how the system works and have a first-hand experience on it, I tried to install the software provided by *OpenAI* itself and run the command available.

Installation

Installation is pretty straight forward, but I had several problems because of the versions used on the packages.

They use not too old but neither too new versions of many packages. For example, tensorflow's latest release is 2.0.0 but they used 1.9.0, which is from July 2018.

What's more, they use python3.6 and the latest version is 3.7. All these caused package conflicts and installation or execution breakages.

The fix for that was using *virtualenv* to use python3.6 and older versions of the packages I needed.

virtualenv also need bash or a bash-compatible shell to work properly. Basically, not doing so caused a breakage.

Usage

On `https://github.com/openai/multi-agent-emergence-environments` there are enough instructions on how to use the program but generally the command to be run is the following:

$ bin/examine.py examples/<properties>.jsonnet examples/<saved policy>.npz

This runs the program from a seed with given environment properties and policy.

On *examples/* directory there are several files to select from already built up by developers.

It is possible to modify the properties of the environment but policies cannot be changed modifying their files, as they are binarized.

Experience

The visual experience of the program itself is pretty basic, actually pretty more archaic compared to the graphics used on the demonstration videos provided by OpenAI.

Apart from the graphics, we didn't get to run the program from a 0 episode policy. We always had to use premade policies.

In terms of usability of the program itself, there is not much to say as for a complete learning process too many episodes must be made.

3.2 Implications

Papers like this shows us the emerging power AI field is taking. Simulating multi-agent behaviours like this, more and more complex problems can be solved. For example, we can replace Hide and Seek problem and insert any useful automatised job, such as doing something and adapting to new situations.

It obviously does have other ethical implications, and can have two distinguished points of view:

- Automatizing is good, so this paper makes daily life easier and more convenient. We can try agents in useful environments so that they learn to do their jobs better than us.
- Jobs can be destroyed, as they work faster, cheaper and better than the human. If technologies like this were developed and enhanced, it can be a risk for our sustainability.

4 Conclusions

It is clear that Reinforcement Learning is a groundbreaking method on Artificial Intelligence in order to achieve goals using self-supervised multi-agent systems, which makes the development of the whole much more simpler, although that means that the system will need a considerable amount of training hours in order to achieve the relevant skills to complete the task. However, making a self-supervised system like this also contributes to scalability and constant improvement as there are no predefined rules set on the agents in order to accomplish specific tasks, allowing them to explore and discover indefinitely what a programmer may not come up with. Example of that are the cases where agents exploit bugs on the simulation environment itself (2.4 Their unfinished).

5 References

https://openai.com/blog/emergent-tool-use/
https://arxiv.org/abs/1909.07528
https://github.com/openai/mujoco-worldgen
https://github.com/openai/multi-agent-emergence-environments