Diversity, and Reinforcement Learning - A short extension

The game involves firing dynamic bodies (Planets) around static bodies (Stars) that have gravity. The goal of the game is to maximise the time that planets remain in orbit around the stars. Some possible options that can be changed are the number of stars, the number of planets, the range of mass (size) for the planets and stars, the size of the game-window and more.

Agents were trained using Reinforcement Learning with 2 optimisation algorithms, PPO and NEAT. Given this, every run has a reward. Each run is a single attempt by the AI agent to fire the planet(s) around a star(s).

The PDF will discuss some investigations of how the performance of the game was impacted by implementing principles of Software diversity.

Explaining the data collected

A reward function determines the reward. In summary, when an agent does well, the reward function increases reward, and when an agent does something bad, the reward function decreases reward. My reward function punished agents when planets collided with one another, planets collided with a star, and when planets went off-screen. And agents received a reward every few timesteps for every agent still in orbit.

To prevent infinite game lengths when perfect orbits were achieved, I limited each run to a fixed number of timesteps, 1024 timesteps. When an agent kept all planets alive for a cumulative total of 1024 timesteps, that run is said to be a success. With all this said there are 2 possible numerical values we can use to understand the performance of each agent. Whether or not the run was successful, and the reward achieved by the agent.

2 of the questions I answer demand numerical values for the potential benefits that we could gain via diversity. More specifically, the benefits to overall AI performance that we'd get if we used a perfect 1oo2 component that selected the better agent at the start of every run.

Operationalising “potential benefits gained via diversity”:

1. Train the agents via one shared domain (one `data_handler` object).
2. Run the agents 1000 times on the domain.
3. Randomly choose a subdomain (A subset of the possible settings specified by the `data_handler` object), and using this we compare the performance of the agents using their reward
    1. For example, agent 1 is better when the mass of the sun <25.
4. Make note of the difference in reward
5. This difference in performance on a particular subdomain tells us that hypothetically if we had a mechanism that could predict which agent would be superior, the difference would be n units. (n is how much better the better agent is a given subdomain)
6. Repeat steps 3,4,5 to get an average on many different subdomains to ensure we can be certain of our answer.

A higher number means more benefits to reward would be achieved via diversity (1oo2 component) and vice versa.

Is implementing principles of diversity more or less beneficial as the Agents improve in competency?

To answer this question, I used a function called `benefits_via_diversity` which contains the implementation of the above algorithm. From now on I'll use the acronym PBGVD for potential benefits gained via diversity. I had 4 agents. Incompetent_A, Incompetent_B, Competent_A, and Competent_B. A/B refer to the specific agent. Meaning Incompetent_A comes from a checkpoint of Competent_A.
[The actial agents are 1Amodel84-2303(04:14).zip, 2Amodel87-2403(14:36).zip, 1Bmodel114-2303(12:30).zip, 2Bmodel156-2503(10:49).zip in the same order as above]

Competence refers to the performance of the agents. This was determined by using a field named `ep_rew_mean` visible during the `stable_baselines3` training procedure. As you expect it tracks the average reward over an episode which isn't necessary to explain, but as `ep_rew_mean` increases, the performance of the agent increases. I picked `ep_rew_mean` values of 100 for the incompetent agents and 300 for the competent ones.

I then used the `benefits_via_diversity` method to find a PBGVD on the 2 incompetent agents and got a 74 units. And again I used the method on the competent agents and got 51 units. From this, we can conclude that as agents improve their performance, the potential benefits that can be gained with diversity will decrease.

Are the benefits gained from diversity more substantial across implementations of different types of AI agents (eg PPO vs NEAT) or different implementations of the same type of agent (eg PPO vs PPO)?

To answer this question I trained 4 agents, PPO_A, PPO_B, NEAT_A, and NEAT_B until they became as good as I could get them. I then calculated the PBGVD between Across PPO/NEAT agents. And then between the best PPO/NEAT agents.
[The actial agents are 1Bmodel114-2303(12:30).zip, 2Bmodel156-2503(10:49).zip, winner04032324, winner04040105 in the same order as above]

The PBGVD between NEAT agents was 66.41, and for PPO it was 24.79 for an average of 45.6. I then did 4 comparisons between different agent types (PPO_A/NEAT_A, PPO_A/NEAT_B, PPO_B/NEAT_A, PPO_B/NEAT_B). The average PBGVD is 252.23. From this value, it’s clear that the benefits gained between different types of Implementations outweigh the difference between agents of the same type. Unfortunately though, in our case this difference is certainly because is because PPO agents are much more successful when playing the game so trivially, comparing results between different agents will result in a massive difference. In one test, NEAT_B won 19.5% of the time and PPO_A won 29.57% of the time. So while we have an answer to the question, it comes with an asterisk attached.

Is it better to use multiple specialist agents or 1 large generalist agent?

I used 4 agents to answer the question. We'll name the agents PPO_A, PPO_B, and PPO_AB. A and B represent the specialised domains that the agents were trained to perform in. Domains A and B were equal other than the range of potential mass for both planets and stars.
[The actual agents are 1model17010-0404(12:56).zip, 2model17150-0404(03:46).zip, model66-2303(00:25).zip]

I first trained an agent on PPO_A (on domain A), and after ~1 million steps, the agent could satisfy the win conditions of the game 37.33% of the time. Agent PPO_B was trained for ~1 million steps and won 34.20% of the time.

I trained a combined agent PPO_AB for ~4 million timesteps. Even after this much larger training time, it only managed to satisfy the win conditions 7.53% of the time.

With much less training, splitting an AI into 2 specialist agents managed to get better results in less time. Given this, we can conclude that splitting the agents into smaller specialists is an efficient approach to take when using AI agents to solve complex problems.