Gekitai with Reinforcement Learning

João Sousa Miguel Rodrigues Ricardo Ferreira

May 31, 2022

Specification

Considerations

- In this assignment the main goal is to develop an AI capable of playing gekitai using reinfocement learning algorithms.
- Since the gekitai game is very simple, the goal for our agent is to win games against a more traditional algorithms/heuristics for generating moves.
  - In this case we used a heuristic in which the opponent generates random moves but with more probability of choosing central spaces - more valuable.

Tools and algorithms

- For this project, we choose python as the main programming language, since it offers a lot of utilitaries and lots of libraries targeted to RL projects.
- For the environment we used OpenAI gym.
  - This was a challenge since gym API is best suited for single-agent environments. This means that `step()` would require some adaptation, i.e. `step()` would play for both the agent and its opponent.
- The implementation of the RL algorithms will be provided by Stable Baselines3.

Implementation

Considerations

- The first step of this assignment was develop the environment in which we could simulate a game play of gekitai. As stated before we have used gym since must of the well-known implmentations follow its API.
- The next step was putting our agent to learn using the appropriate RL algorithms. For this assignment we used the following algorithms:
  - DQN
  - PPO
  - A2C
- As expected, it is possible to pass hyperparameters to all of the algorithms allowing for fine tunning.

Environment

- While developing the environment the main challenge was to develop the `step()`. As previously said a step in the environment means that both players make their turns.
- Here is a simplified code snippet of `step()`:

```python
def step(action):
    board = move(board, action)
    reward = 0
    done, info = is_over(board)

    if done:
        reward = 1 if info['winner'] == AGENT else -1
    return board.flatten(), reward, done, info
```

RL algorithms used

Deep Q-Network

- The DQN algorithm is based in the Q-learning algorithm.
  - Basically, the Q-table which store the Q-values for each pair `(state, action)` from the latter is substituted by a a neural network which is trained to estimate that same Q-value, in other words, Q-learning is for a discrete `observation_space` what DQN is for a continuous `observation_space`.,
  - Both DQN and Q-learning have the charateristic of being **off-policy**, meaning that the behaviour of the agent is completely independent from the produced estimates for the value function.

Proximal Policy Optimization

- The PPO algorithm is an algorithm developed at OpenAI.
  - It tries to find a balance between different aspects as **ease of tunning**, **sample efficiency** and **code complexity**.
  - It works with an **on-line** basis, meaning that unlike DQN there is no replay buffer where the agent can learn from current and past actions but rather the agent only learns from what the current action is and it only processes that same action once for the entire lifespan of an episode - in order to update the model's policy gradient.

Advantage Actor Critic

- A2C is a policy gradient algortihm and it is part of the on-policy family of RL algorithms.
  - It consists of 2 networks, the **actor** and the **critic** who work together in order to solve a particular problem based on an advantage function which calculates the agent's temporal difference error.
  - This means that the A2C algorithm works on a temporal difference learning paradigm by using error prediction, very similar to how the human brain also learns new things[1].

[1] Article by Mike Wang

Conclusion

- After training our agent with DQN, PPO and A2C we reached several conclusions:
  - Looking at the average episode reward[2] it is visible that DQN is the algorithm with areward value closer to 1.
    - This means that the agent wins the majority of the episodes/games against its opponent who uses the developed heuristic.
  - Looking at the average episode length[3] it is also visible that DQN episodes took a smaller number of moves.
    - In this particular case the agent learned to take advantage of playing in the outter spaces!
  - Finally, both PPO and A2C had similar performance metrics.

[2] Average episode reward plot
[3] Average episode length plot

References

References

- Some of the references for the work already carried out:
  - Gekitai Rules
  - IA's course page @ moodle
  - Reinforcement Learning
  - OpenAI gym
  - Stable Baselines3
  - DQN Explation
  - PPO by OpenAI
  - PPO by Arxiv Insights
  - A2C by Alvaro Durán Tovar
  - A2C by Mike Wang