Deep RL for Starcraft II

Andrew Chang (agchang1@stanford.edu)

Problem

Starcraft II is a competitive multi-agent game where an agent must combine both long-term planning and real-time mechanics to perform well. DeepMind released SCLE [1] which is a RL environment for building agents that can play the full game. In this project, I considered a simpler setting using one of the provided minigames, namely **DefeatRoaches**. The goal is kill as many roaches (an enemy unit) in a limited episode time. The roaches are strong units that can attack back, so the agent must learn a good policy to perform well. As an example of a good policy, human players use a mechanic known as "*microing*", which involves micromanaging your units in way to maximize the number alive at any name, e.g. by moving damaged units behind healthier units.

For the minigame, the agent starts with 9 marines and must fight 4 roaches. A reward of +10 is given for each roach killed and -1 for each marine killed. If all roaches are killed within an episode, a new group of 4 is spawned and the agent gets 5 more marines. The episode is terminated when either all marines die, or 120s passes.

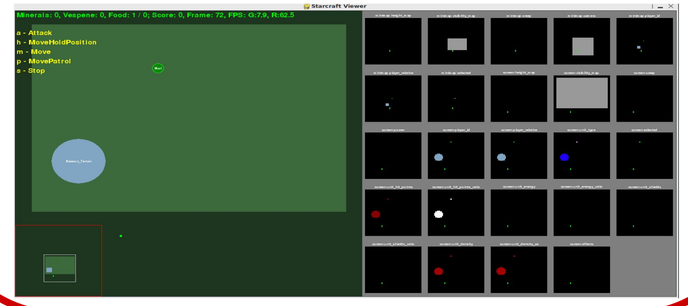
Environment

The environment exposes a standard RL interface for agents where an agent can enter a loop where it receives a state, performs an action, and the receives the next state and a reward. The environment allows the agent to act every N game steps, and the agent is not time-bound to make a decision. The standard configuration is for the agent to run every **8 game steps** which corresponds to **180 APM**. The action space is defined as a set (524) of base actions a_0 and a variable number of arguments $a_1,...,a_l$ where l differs depending on the base action chosen. The flattened action space is **101938719 actions.** Actions are chosen to be similar to a human's interface to prevent giving agents unfair advantages. For example, the agent must select a unit before issuing a command -- previous work allowed agents to issue commands to all units simultaneously, which gives a big advance. An example action to move a selected unit:

[331/Move_screen, 3/queued [2], 0/screen [84, 84]

Features

The spatial state space contains **feature layers** instead of raw pixels which have either scalar or categorical variables. There are two sets of spatial feature layers corresponding to the **screen** (primary view) and **minimap** (smaller fixed map view) of sizes **[n,n,13]** and **[64, 64, 7]** respectively. n is chosen to be 64 for this project. There is also sets of non-spatial features, the ones chosen for this project are: general player information **[13,1]**, multi-select **[50,7]**, and available actions **[524]**. Flattening this means the state vectors are **82807** dimensional.



Approaches

- A standard A3C implementation which uses a conv net to model multiple policy functions $\boldsymbol{\pi}_i(a_t|s_t)$ and a value function $\mathbf{V}(s_t)$. The goal of the policy is gradient is to maximize by expected sum of rewards by directly differentiating to get the gradient of the policy. The standard A3C gradient is therefore:

$$\underbrace{(G_t - v_\theta(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\text{policy gradient}} + \beta \underbrace{(G_t - v_\theta(s_t))\nabla_\theta v_\theta(s_t)}_{\text{value estimation gradient}} + \eta \underbrace{\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)}_{\text{entropy regularization}},$$

- The policy gradient contains the "advantage" term which intuitively increases probability of actions that do better than expected (advantage over baseline), the value estimation regresses the value function to the advantage, and the entropy term encourages exploration by incurring lost for low entropy distributions.
- The convnet architecture same used in DeepMind's Atari paper [3], and the network outputs a policy for the base action and each argument independently, including the spatial arguments where it outputs x,y independently.

Results

- Agent achieves **17.64** average with **46 max** over 50 episodes
- Random policy achieves **1.04** average with **21 max**
- Analyzing the learnt policy reveals a simple strategy to flank the roaches on the top or bottom so all marines focus fire on a single roach and the bottom roach needs to move closer to attack.
- Interestingly the policy used the patrol action since it is a convenient way of moving and attacking.

Discussion

- Although learnt policy is better than random, it is still not high-skilled human level (scores well over 100).
- With more compute and ability to experiment, could have better results, original paper cites 100 experiments with randomly sampled hyperparameters for each mini-game, ran for over 500M game steps. Having limited hardware definitely impacts ability to evaluate models (training time is relatively slow).
- Training can plateau easily at a non-optimal strategy, or even diverge and perform worse than baseline.
- This project truly illuminates how difficult RL for applied environments (massive state and action space).

References:

[1] Starcraft II: A New Challenge for Reinforcement Learning
https://arxiv.org/abs/1708.04782
[2] Asynchronous Methods for Deep Reinforcement Learning
https://arxiv.org/abs/1602.01783
[3] Playing Atari with Deep Reinforcement Learning
https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf