"Game of Booms": Proximal Policy Optimization for Pommerman

Binbin Xiong, Minfa Wang
Stanford University

Abstract

In this project we propose to participate in the Pommerman competition, which is a play on Bomberman. Our goal is to develop a single agent under FFA mode (Free For All, where four agents enter and one leaves and the board is fully observable) that can beat all other players. More detailed game description could be found here.

Background

Our task is to design a model which reads a state (as a dictionary of observation), outputs an action.

The **state** of the game for each agent is represented as a dictionary of following fields:

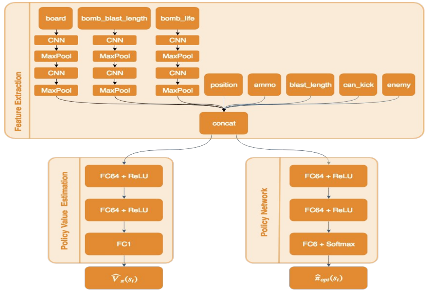
- **Board**: 121 Ints. The flattened board.
- **Position**: 2 Ints, each in [0, 10]. The agent's (x, y) position in the grid.
- **Ammo**: 1 Int. The agent's current ammo.
- **Blast Strength**: 1 Int. The agent's current blast strength.
- **Can Kick**: 1 Int, 0 or 1. Whether the agent can kick or not.
- **Teammate**: For FFA, this will be -1.
- **Enemies**: 3 Ints, each in [-1, 3]. Which agents are this agent's enemies.
- **Bombs**: List of Ints. The bombs in the agent's purview, specified by (X int, Y int, BlastStrength int).

In each time step, an agent can choose from one of six **actions**:

Actions(state) = { Stop, Up, Left, Down, Right, Bomb }

As can be seen from the setup, this is a very high dimension yet super sparse reward problem.

Model Architecture



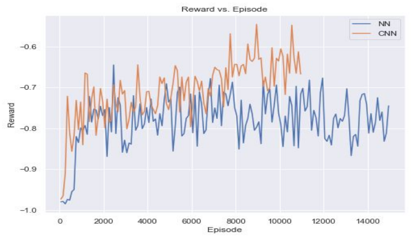
We show our training framework as PPO and loss function on the right, and the feature extraction architecture above.
Feature Extraction/Embedding:
Extract dense features from the raw state.
Policy Value Estimation:
Estimate the expected reward by following policy pi from the current state s and onward. Standard 3-layer neural network architecture.
Policy Network:
Compute the estimated optimal policy from the current state s. Similar architecture as Policy Value Estimation.

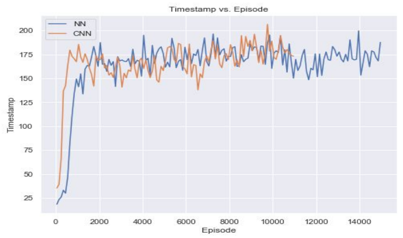
Algorithm 1 PPO, Actor-Critic Style

```
for iteration=1,2,... do
    for actor=1,2,...,$N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \le NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

Experimental Result





Metric Definition

we choose simple agent as our enemy to compete against to. Each timestamp, each agent will be given an **reward** = { +1 if it's the only survivor and the game is end, +0.1 if it kills another agent, +0.1 if it eats special power, 0 otherwise }

Model	Reward
Baseline-Heuristic	-0.46
NN with PPO	-0.64
CNN + NN with PPO	-0.53

Future work

Playback: Learn from close-to-end states.
Learn from humans:
Try to educate the agent in the first a few thousand episodes to emulate the way human plays.

Reference

N-Person Minimax and Alpha-Beta Pruning: extend minimax to multi-agents scenario, and use alpha-beta pruning for efficient searching.
Depth-limited search: reduce search space with evaluation function.
Proximal Policy Optimization Algorithms (PPO): Efficient policy gradient updates.
Opponent Modeling in Deep Reinforcement Learning: Learn strategy patterns of opponents through encoding observations of opponents into a deep Q-Network (DQN).