2023 Summer REU @ OSU

Reinforcement Learning with Ray RLlib

Yuni Jeong

Mentors | [redacted]

Table of Contents

Overview

Topic
Reinforcement Learning

Project Theme
Multi-agent Reinforcement Learning (MARL; cooperative case)

Project: FireDrones
MARL project built with Ray RLlib where multiple agents (drones) are trained to control wildfire in a 2D grid forest.

Reinforcement Learning 101

Reinforcement Learning (RL)

One of basic machine learning paradigms alongside supervised and unsupervised learning. Agent learns in an interactive environment by trial and error using feedback from its own actions and experiences.

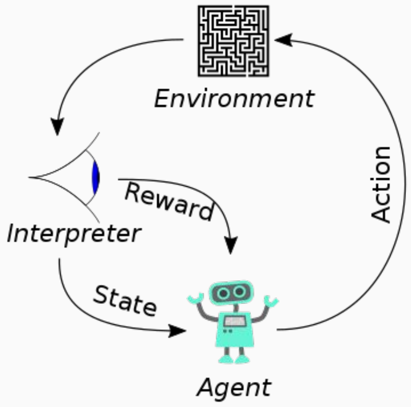
vs. Supervised Learning

RL uses reward as signal for desired behavior, not labelled input/output pairs for training.

vs. Unsupervised Learning:

RL goal is finding policy that would maximize cumulative reward, not identifying data patterns from an unlabeled dataset.

RL is the closest to the kind of learning that humans and other animals do!



Reinforcement Learning 101 (cont.)

Markov Decision Process (MDP)
Mathematical framework that describes almost all RL environment. A MDP is a 4-tuple of:

- Set of environment states *S*;
- Set of agent actions *A*;
- State transition probability function *P_a(s', s)*=*Pr(s' | s, a)*=probability that action *a* in state *s* at time *t* will lead to state *s'* at time *t+1*;
- Reward function *R_a(s, s')*=immediate reward after transition from *s* to *s'* with action *a*

MDP Goal
Find a policy–a mapping from state to action–that will maximize some learning objective; typically the expected discounted sum of rewards over a potentially infinite horizon.
The amount of “discount” is represented by the discount factor γ that has range [0,1]. γ controls the degree of emphasis on immediate vs. delayed reward.

Reinforcement Learning 101 (cont.)

RL Algorithms

Goal is to solve MDP problem (aka find the best possible policy)

⇒ Use agent's past experience to find out which actions lead to higher cumulative rewards.

= Value function estimation

Value Function (under policy *π*)

Maps state *s* to the expected return when starting in *s* and following π thereafter.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

For any π and s, the following recursive relationship holds:

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},$$

This equation expresses a relationship between the value of a state and the values of its successor states, and is called the Bellman equation for value function *v_π*.

Reinforcement Learning 101 (cont.)

Bellman Equation for v_π and Finding Policy
Solving the Bellman Equation finds the value function, which maps states to expected returns.
With the value function, we can find the function that describes optimal action as a function of state: the policy function.

Multi-agent Reinforcement Learning

RL vs. MARL

- Optimal policy for single agent vs. Optimal policies for multiple agents.
- State transition caused by single agent vs. State transition occurs based on joint action of agents.

MARL Applications and Challenges

- Autonomous driving, multi-robot warehouse management, automated stock training, etc.
- Scaling in number of agents, optimality of policies and equilibrium selection, non-stationarity caused by learning agents

Project FireDrones

source

Agents’ objective
Extinguish fire in a grid-like forest as soon as possible ⇒ Cooperative MARL Problem

Technology used
Ray RLlib. Native support for multi-agent environment and has a number of state-of-the-art RL algorithms with comprehensive documentation

Motivation
- Gain experience in MARL
- Lack of up-to-date tutorials on Ray RLlib
- Showcase potential of MARL in similar problems (e.g. capturing wild animals, cleaning up oil spills or algae bloom in the ocean).

FireDrones Problem Definition

Environment
A 2D integer grid. At the beginning of each episode, (1) tree is planted in each cell with some probability, (2) several trees are set on fire, and (3) agents are placed in cell (0,0). A cell can be an empty cell, a tree cell, or a fire cell. Tree and fire locations change for each episode.

Agents
Drones that move around the grid and extinguish fire by spraying water. At each timestep, a drone can move in one of the 8 directions or stay in the same cell and spray water (9 actions in total).

Observation (state)
A drone has local vision of the map gets one observation per timestep, which contains information of (1) the agent's current coordinate on grid, and (2) its surrounding cells' state (fire or not, etc.). Number of “surrounding cells” depends on agent’s vision level.

Reward
+1 for extinguishing a fire cell, -1 at each timestep (time penalty)

FireDrones Problem Definition (cont.)

Environment Update
Fire spreads to neighboring trees with some cell-specific probability at each timestep.
Independent of agent actions.

Episode Termination
Each episode ends when all fires are extinguished OR timestep reaches 100 for that episode.
Hence shorter episode length implies faster completion of the firefighting task.

Experiment Setup: Environment Configuration and Algorithm

Experiment Configuration

```
env_config = {
"height": 10, # grid (forest) size
"width": 10,
"prob_tree_plant": 0.5, # Probability of each cell being a tree
"num_fires": 2, # Fire severity: initial number of trees on fire
"prob_fire_spread_high": 0.2,
"prob_fire_spread_low": 0.05,
"timestep_limit": 100, # End an episode after this many timesteps
"num_agents": 10, # Number of drones
"agents_vision": 1, # How far can an agent observe. 1=>3x3, 2=>5x5, etc.
"time_penalty": -1,
"fire_ext_reward": 1,
"do_render": True, # Render training process with Pygame
}
```

Experiment Setup: Environment Configuration and Algorithm

Algorithm: Proximal Policy Optimization (PPO)
A policy gradient (PG) method; iteratively updates the policy by sampling trajectories from the environment and computing the policy gradient (the direction that improves the expected return).

PPO Characteristic

Uses clipped objective function that penalizes large changes in policy
⇒ Keeps the deviation from previous policy relatively small
⇒ Avoids overfitting or collapsing the policy to a suboptimal solution.

PPO Advantages

Balance between ease of implementation, sample complexity, and ease of tuning.

- Easily implemented with standard deep learning frameworks
- High sample efficiency; learns from fewer interactions with the environment
- Does not require complex hyperparameter tuning or sophisticated optimization techniques
- Can handle discrete and continuous action spaces, multiple agents, and parallel environments

Forest Example

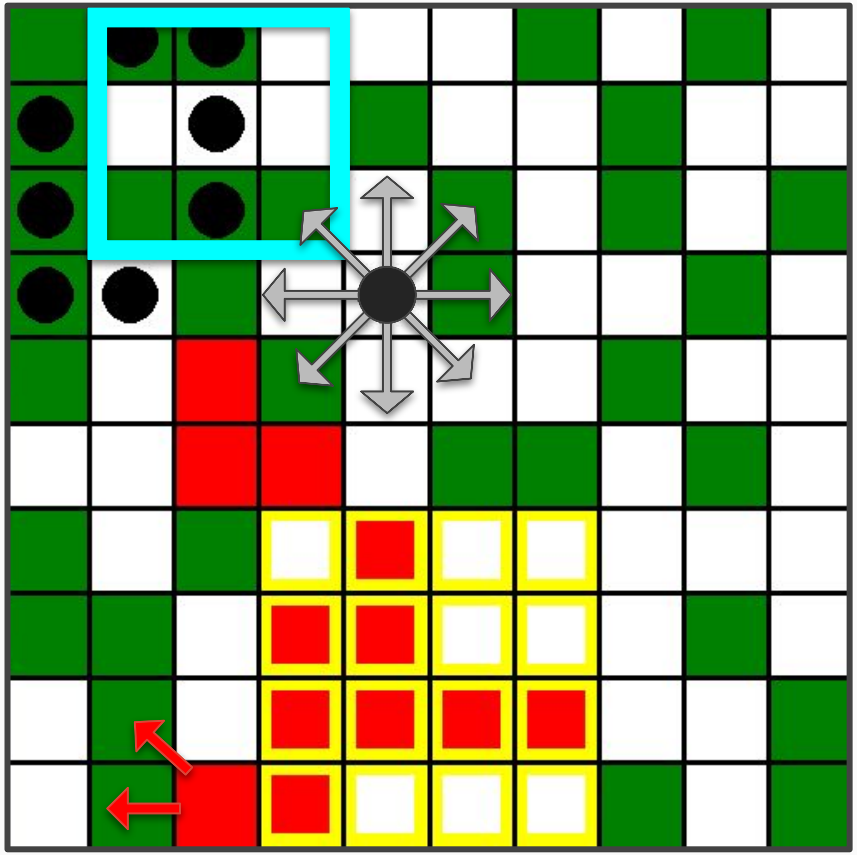
Blue rectangle: observable area for the agent in the middle with agent_vision=1

Gray arrow: possible next coordinates of a drone

Red arrow: possible directions of fire spread

Yellow border: cells with high probability of fire-spread

[⚫=drone, 🟩=tree, 🟥=fire, ⬜=empty]



Screenshot of Pygame animation; taken during a training episode

Training Results

Training length
100 iterations (40 episodes per iteration)

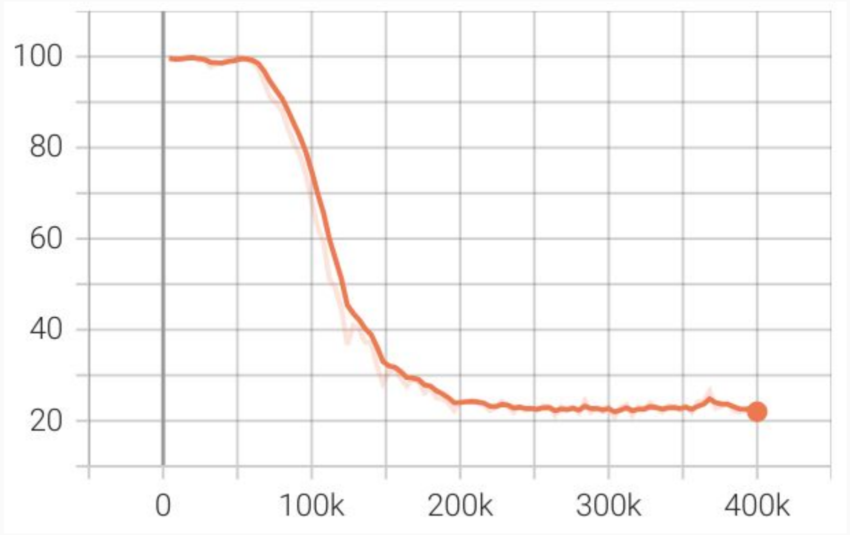
Performance metrics

- Episode length: faster fire suppression -> shorter length
- Reward at episode termination: more fires extinguished -> higher reward
- Fraction of burnt trees: more unburnt trees -> lower fraction

Baseline test
Random policy; agents take random actions at any given time (no learning happens)

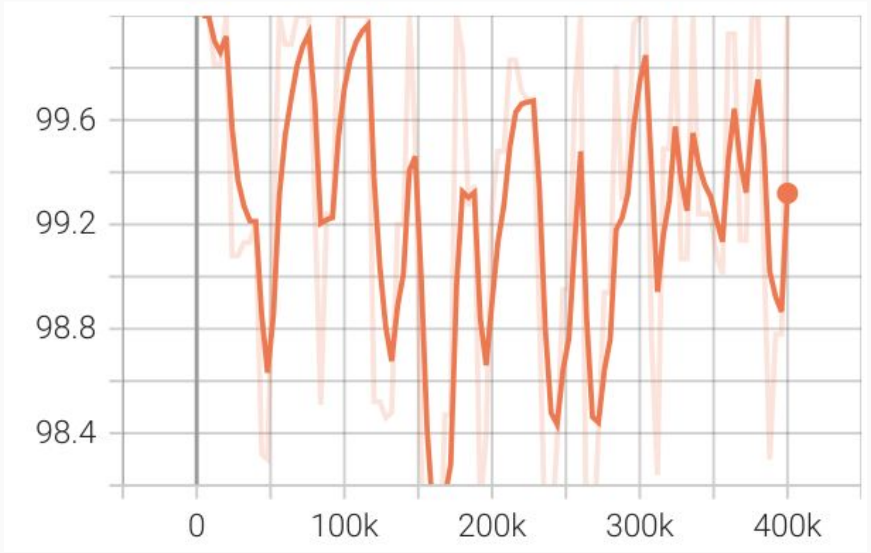
Episode Length

PPO



- Decreased over time from 100 to 20

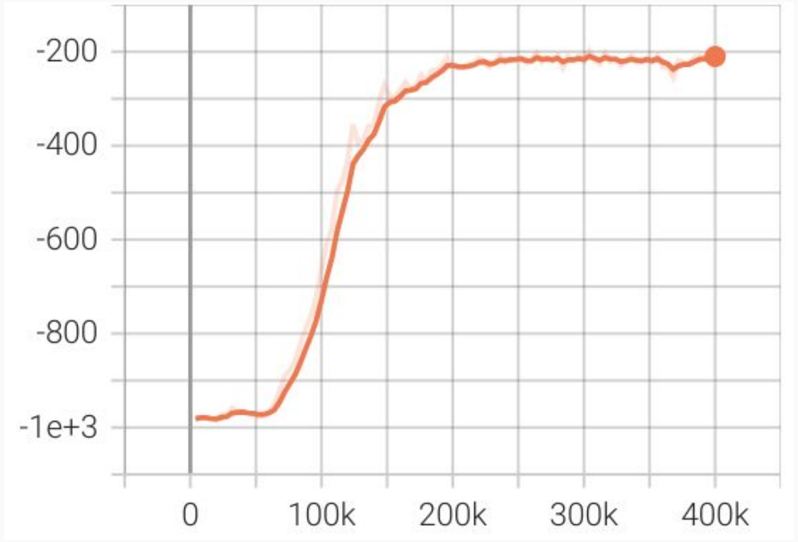
Random



- Fluctuates between 100 and 98
- No decrease over time

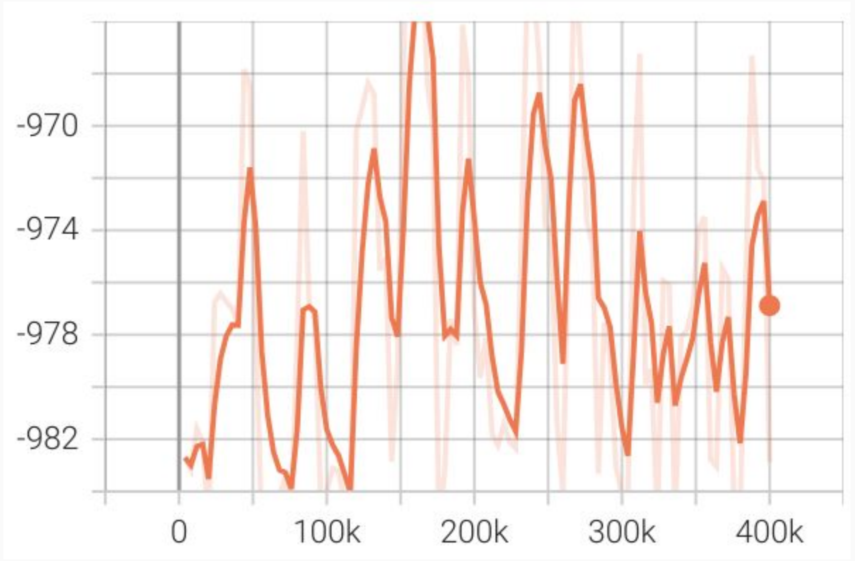
Reward at episode termination

PPO



- On average, increased over time from -100 to -20 (value divided by the number of agents, 10)

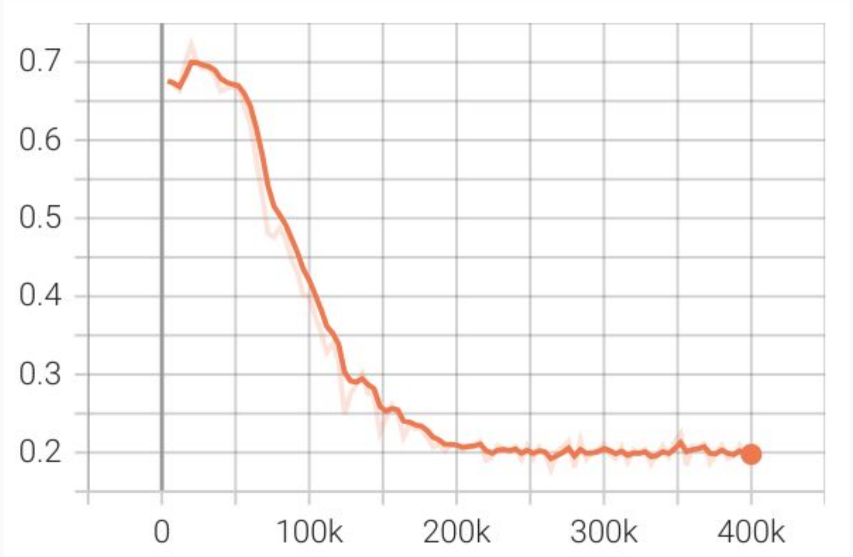
Random



- On average, fluctuates between -98.2 and -96.5 per agent
- No increase over time

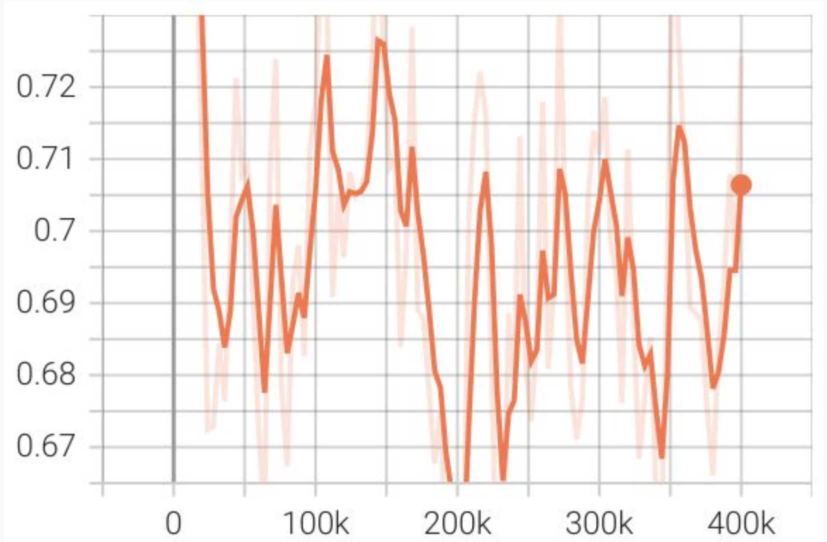
Fraction of burnt trees

PPO



- Decreased over time from 0.7 to 0.2

Random



- Fluctuates between 0.75 and 0.65 per agent
- No decrease over time

Conclusion

Training agents with PPO yields much better-performing agents compared to taking random actions.

Extending the project
Considerations that can be incorporated:

- Drones must replenish its battery and water
- Communication between drones
- Add environmental factors (e.g. wind affects fire spread direction)
- Experiment different observation space
- Make state transitions that depends on the agent action and are stochastic (e.g. let the spread of fire dependent on drone movement)

Thank you :)

Any questions?

Project repo: Search **FireDronesRL** on GitHub