Team Tron RL

Andranik Saakyan
saakyana@uci.edu

Arya Kashani
akashan1@uci.edu

Abstract

We are interested in exploring the emergence and evolution of cooperative behavior (trapping, sacrificing, communicating, etc) by playing Tron with teams. We are also interested in observing how the behavior of the agents changes as a response to changes in the environment (e.g. imposing constraints on communication, adding obstacles, etc).

In this project we train teams using three reward systems and have them compete with each other to determine which reward system and policy result in the highest cooperation and/or win ratio.

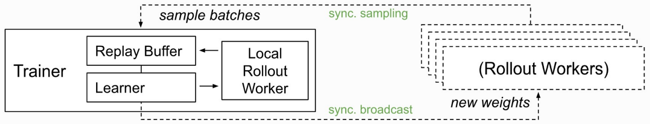
1 Introduction

Our project is in the exciting realm of multi-agent reinforcement learning. Applications of multi-agent reinforcement learning include large-scale fleet management, swarm systems, and traffic light control [1]. While Q-learning with Deep Q-Networks has resulted in strong performance in single agent settings, the introduction of multiple agents complicates the learning process and brings forth novel challenges such as non-stationarity and partial observability [1]. The non-stationarity of the environment in multi-agent settings means the agents' policies are dependent on each other and must be constantly updated. The fact that the environment is dynamic violates the Markov assumptions required for the convergence of Q-learning algorithms such as DQN [2]. Partial observability of the environment further complicates the learning process, but we have left this challenge for future research and provide the full grid as input to each agent. Many strategies have been proposed to tackle these problems (LOLA, RIAL, Q-MIX), however training multi-agent policies using single-agent RL algorithms can yield surprisingly strong results [2].

This is the approach we are using. We have created a new Tron teams environment that supports having multiple players and teams. An agent dies if it collides with a wall or another agent, and the winner of each game is the team of the last remaining agent. Each team trains a single policy with it's strategy depending on it's reward system. In our experiments, after 1000 epochs the balanced team beat the socialist team in almost 70% of the games, the balanced team beat the favored agent team in about 80% of games, and the favored agent team beat the socialist team in about 85% of games.

2 Background

Our project extends the Tron environment in the ColosseumRL platform. We use RLlib and Tensorflow for training our teams. We chose the Deep Q-Learning algorithm with the following architecture for our DQN:



DQN architecture

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

The Convolutional Neural Network has 3 filters: (64, 5, 2), (128, 3, 2), (256, 3, 2) where the tuple values represent output channels, kernel size, and stride, respectively. The network has 128 hidden nodes and 3 nodes in the output layer which represent the approximated Q-values associated with each action in the action space.

3 Problem Statement

The Tron teams environment is a fully-observable 13x13 grid with 4 players and 2 teams. Agents are individually rewarded 1 point for every move they are alive, 1 negative point for death, and a reward of 10 points for winning. To encourage cooperation and intelligent team behavior, we have created three reward systems: balanced, favored agent, and socialist. The reward system for each team is as follows:

Balanced

reward[player1] = reward[player1] + 0.5 · reward[player2]

reward[player2] = reward[player2] + 0.5 · reward[player1]

Favored Agent

reward[player1] = reward[player1] + 0.25 · reward[player2]

reward[player2] = reward[player2] + 0.75 · reward[player1]

Socialist

reward[player1] = reward[player1] + reward[player2]

reward[player2] = reward[player2] + reward[player1]

Regarding the favored agent, we were interested to see if this reward system would incentivize the lower-weighted player to sacrifice itself in situations where doing so would lead to its teammate winning the game, receiving 10 points, and thus increasing its own reward.

We divided our project into three milestones. In milestone 1 we focused on understanding the fundamentals of reinforcement learning/RLlib and implementing a working version of single player Tron. In milestone 2, we created the Tron teams environment and trained teams using a DQN. Finally, in milestone 3, we planned on experimenting with various reward systems and hyperparameters in order to observe cooperative team behavior.

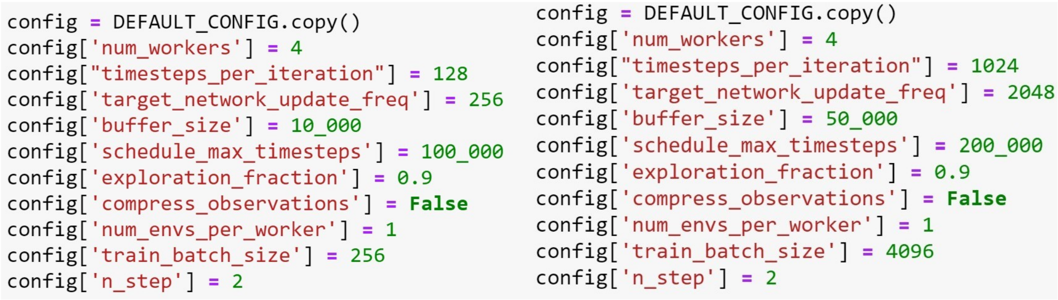
4 Method

Our initial approaches were not very successful. We started the project using tabular Q-learning due to its simplicity; however, its simplicity was one of the reasons it was unsuccessful. We implemented a Q table for storing state-action pairs and chose an action based on the epsilon-greedy policy, but our agent had poor performance. Tabular Q-learning quickly becomes infeasible for complex action and state spaces. Therefore, we decided to create a sequential neural network using the keras library. The inputs to the network were: x and y coordinates of the agent, direction, distance to the edge of the board in each direction, and whether there is a player in adjacent blocks. This approach was also unsuccessful as our network did not perform well.

At this point, we returned to RLlib and created a DQN with a buffer size of 10,000, exploration fraction of 0.9, and training batch size of 256. Our preprocessor extracts the board and simplifies it for the network by marking the location of all heads and walls, and rotates the board to make it look the same from each player's point of view. We then train a policy for each team, where each team employs one of the reward systems described in the previous section.

5 Experiments

We ran multiple experiments tweaking the values of epsilon, learning rate, and other hyperparameters shown in the figure. The experiments were ran on a machine with an i7 processor and 16gb of RAM. Due to a lack of computational power, configuration (b) was very slow and did not result in much



```
config = DEFAULT_CONFIG.copy()
config['num_workers'] = 4
config["timesteps_per_iteration"] = 128
config['target_network_update_freq'] = 256
config['buffer_size'] = 10_000
config['schedule_max_timesteps'] = 100_000
config['exploration_fraction'] = 0.9
config['compress_observations'] = False
config['num_envs_per_worker'] = 1
config['train_batch_size'] = 256
config['n_step'] = 2
```

(a)

```
config = DEFAULT_CONFIG.copy()
config['num_workers'] = 4
config["timesteps_per_iteration"] = 1024
config['target_network_update_freq'] = 2048
config['buffer_size'] = 50_000
config['schedule_max_timesteps'] = 200_000
config['exploration_fraction'] = 0.9
config['compress_observations'] = False
config['num_envs_per_worker'] = 1
config['train_batch_size'] = 4096
config['n_step'] = 2
```

(b)

Figure 1: Testing sets of DQN hyperparameters

improvement. However, after modifying parameters such as training batch size and timesteps per iteration, training speed and performance improved significantly. For our final project, we used configuration (a) with an initial epsilon of 1.0, final epsilon of 0.2 (annealed over 10,000 timesteps), and learning rate = 0.0005. We also experimented with different reward systems and policies, such as intelligent team vs untrained team and different combinations of games using our three reward systems.

6 Results

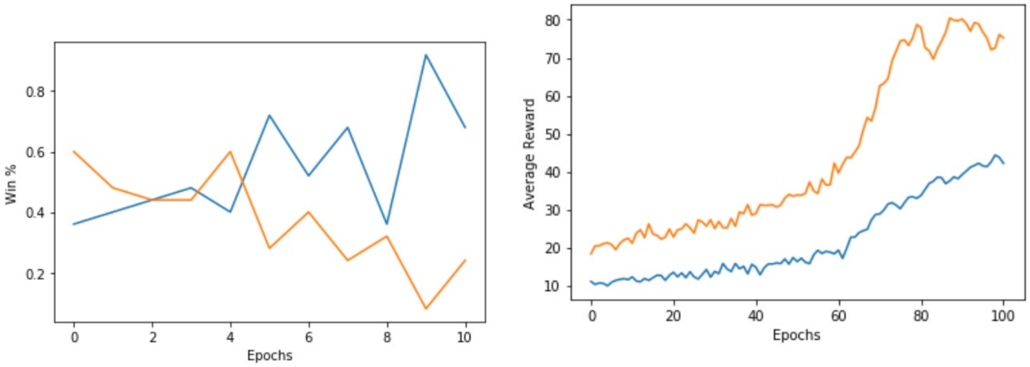
We ran 25 games every 100 epochs and plotted win ratio for each team. The balanced team was most successful, beating the socialist team in about 70% of games and the favored team in about 80%. The favored team also beat the socialist team in about 85% of games. The learning curve is plotted next to each matchup, however it is important to note that the difference between average reward between the teams is not meaningful since each team has a different reward system. For example, we would expect the average reward of the socialist team to be higher regardless of its actual performance due to the way the reward is computed.

Although we achieved some interesting results, we noticed the training method is an area of improvement for future research. Given enough epochs, the average reward converges; however, the initial randomness due to exploration often changes the winner, making it hard to compare teams simply based on reward systems. In future research, we would like to experiment with new training methods to tackle the non-stationarity problem and to make the Tron environment more complex. One idea is to make the environment partially-observable, train a policy for each agent, and give agents the additional action of being able to communicate with their teammate if they are within a certain distance of one another. We believe these constraints would make the emergence of cooperative behavior more natural and interesting to study.

7 Team Members

Andranik Sahakyan
Project idea, brainstorming, coding, experimenation, writing.

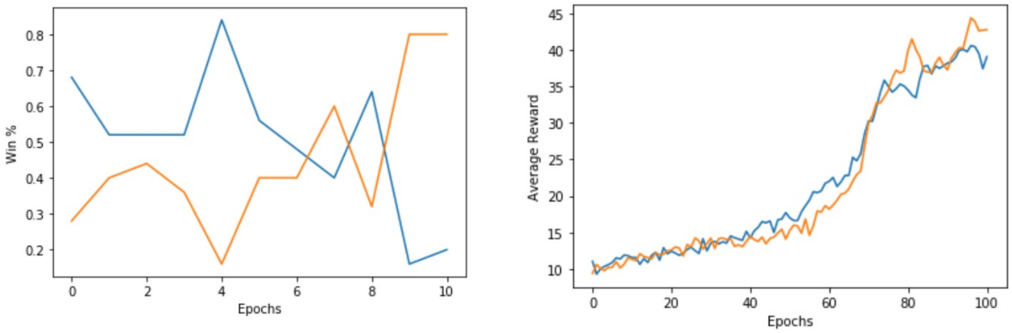
Arya Kashani
Brainstorming, coding, experimentation, data analysis, creating plots.



(a) Win Ratio per 100 epochs

(b) Average Reward per 10 epochs

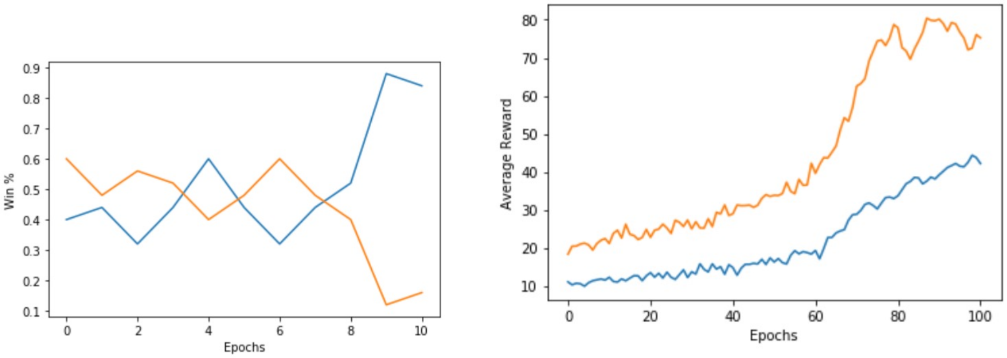
Figure 2: Balanced (blue) vs Socialist (orange)



(a) Win Ratio per 100 epochs

(b) Average Reward per 10 epochs

Figure 3: Balanced (blue) vs Favored (orange)



(a) Win Ratio per 100 epochs

(b) Average Reward per 10 epochs

Figure 4: Favored (blue) vs Socialist (orange)

References

[1] Nguyen, T.T., Nguyen, N.D., & Nahavandi, S. (2018). Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications. ArXiv, abs/1812.11794.

[2] Seita, D. (n.d.). Scaling Multi-Agent Reinforcement Learning. Retrieved from `https://bair.berkeley.edu/blog/2018/12/12/rllib/`