HIGH SCHOOL EDITION
Journal of Student Research
Volume 13 Issue 1 (2024)

Imitation Learning as a Tool for Modeling Cooperative Agent Games

Nikhil Alladi[1] and Sam Showalter[#]

[1]Thomas Jefferson High School for Science and Technology, USA
[#]Advisor

ABSTRACT

The modern day sees constant application of team-level cooperation, and game theory offers a way to model these tasks in a way that provides applicable results. We created a game using Python 3 that had 3 or more agents from one team surround an agent from another team to consume them, with the end goal of consuming the other team. These agents were then trained in a centralized training, decentralized execution schema to mimic an expert's behavior using imitation learning. Agents trained under this imitation learning schema performed significantly better than agents who followed a random policy, and as more training was supplied to the agents to learn off of, the agents performed better compared to other lesser trained teams.

Introduction

Multi-agent games, such as team-based video games and real-world economic simulations, are valuable models for understanding phenomena in the modern world. Many tasks in the modern era require team-level cooperation; creation of accurate and useful models of these tasks' dynamics is one of the most practical applications of game theory today. Applications of multi-agent learning include modeling swarms of cells inside the blood system, large groups of humans in economic contexts, and more. Through the use of computer simulations, researchers can empirically simulate complex behavior to evaluate theoretical findings about multi-agent strategy. In this paper, we explore a self-created game with multi-agent dynamics using methods previously established on the topic of swarm theory and multi-agent imitation learning, a subdomain of Reinforcement Learning (RL). For simplicity, we model these swarm dynamics under Markovian assumptions in order to make learning tractable. We expand on our implementation details in the Methodology section below as well as review existing research contributions in this space.

Literature Review

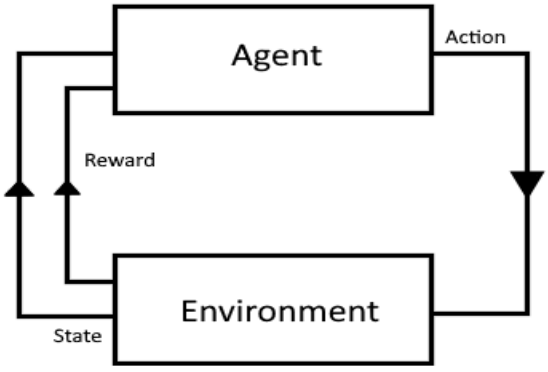

Figure 1. A Reinforcement Learning diagram. A diagram outlining the general procedure of Reinforcement Learning. Agents use a policy to make decisions, and than environment than infers a reward on the agent along with a new state.

The concept of Reinforcement Learning in the context of a single agent has been the subject of hundreds of studies and experiments. Reinforcement learning has many formulations but in this paper we focus on those that can be modeled by Markov Decision Processes (MDPs). MDPs, visualized in Figure 1 above, determine the action to be taken based off of a given state and a reward function associated with that state. We govern the behavior of an agent by choosing from a finite set of actions in order to place the agent in states that confer high reward across time. Put simply, our goal is to learn a control policy $a' = \pi(a|s)$ that yields the maximum return, defined by $R_t = \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$ where T represents the maximum time that the game can be run. The environment then returns a new state given the current state and action, more formally defined by $s_{t+1} = p(s|s_t, a_t)$.

Imitation learning, a subdomain of reinforcement learning, aims to accurately replicate an expert's behavior ("imitation" of experts). The premise of imitation learning involves storing data played by some expert source and then making decisions based on what the expert may have done in the same position. Experts will likely not provide data for all states, so at times this behavior must be approximated.

Multi-agent reinforcement learning represents a more modern advancement in this line of research. Multi-agent reinforcement learning has also been defined in numerous ways. We explore methods that have teams of agents trained through a central model that each incorporate the same reward functions and feedback of single-agent reinforcement learning. Comparatively, Leibo et al. (2017) use a decentralized training, decentralized execution structure where agents are trained and move individually of each other without shared coordination opportunities. This solves the issue of agents taking a large amount of time to train and move, allowing for models to achieve more sophisticated state-action policies without straining resources. By contrast, Lowe et al. (2020) implement a centralized training, decentralized execution structure to train agents under one concurrent algorithm but give agents limited vision. In doing so, they partially mitigate the time constraints involved with multi-agent learning and training without having to deal with the issue of agents having unnatural vision, which would lead to little practicality in the real world.

Multi-agent reinforcement learning proves to be significantly more computationally intensive during training than single-agent reinforcement learning. This can be attributed to the amount of agents that need to be trained, time taken to perform computations, and the increase in simulation complexity that results from an increase in agents. The methods for training agents mentioned in the above paragraph (centralized training, decentralized execution and decentralized training, decentralized execution) are ways of mitigating this time constraint, as otherwise applying the same computational resources of single-agent reinforcement learning would be untenable. Additionally, multi-agent training algorithms have to deal with the issue of observability, especially if training is centralized. Information learned from one agent might not be available to another agent even if their observable states are similar.

In the context of combined training to reduce computational intensity, Shalev-Schwartz et al. (2016) apply a centralized training, decentralized execution structure to train cars to pass through intersections without any collisions in minimal time. The structure closely resembles experiments of coordinated automated driving, where information is fed to a collective to send back decisions to a car. The cars themselves then act communally based on the information they have, continuing the decentralized execution schema. Finding a middle ground, Jin et al. (2018) use a mix of centralized and decentralized training optimal for their game of bidding for advertisement space, with a decentralized execution structure to complete agents' policies. They combine agents in "clusters" due to size and time constraints, which are trained in a decentralized fashion. The individual

agents within these clusters follow a centralized training structure, as they all share the common goal of getting the highest bid within their cluster.

Determining whether to centralize or decentralize training and execution is a vital decision in creating the training model for this simulation. Each side has its own pros and cons, and this paper decided on a centralized training, decentralized execution format. Additionally, avoiding the issue of environment dimensionality also proves to be a significant issue; Growth of the environment size also increases the sparsity of the observable state space, which prompts us to leverage partial observability MDPs, also known as POMDPs. We elaborate on these design decisions below.

Methods

In order to empirically analyze multi-agent game dynamics, we built a game in Python 3.11 using PyGame. Agents are placed in a square grid surrounded by impassable borders, and can move up, down, left, or right. If no moves are available, the agent does not move. If an agent is surrounded by three or more agents from any opposing team, the agent is consumed. The game ends when the turn limit is passed or no agents remain on a team. That is, this is a coordinated, zero-sum (between teams), team game where one team intends to consume the other by engulfing them. If the turn limit is reached, then a winner is decided by the team with the most agents.

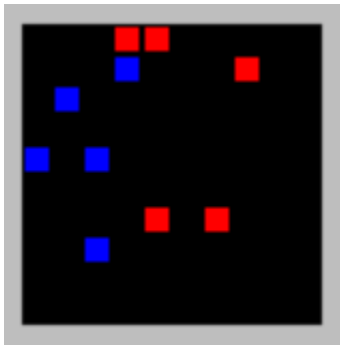

Figure 2. The Game Board. A snapshot of the gameboard at an early timestep of the game.

Data from human gameplay will be stored into a database where Markov chains, defined below, will be implemented and used for imitation learning to simulate an intelligent player. The implementation of the Markov chain works by the assumption that only the current state of the agent is needed to dictate its next move (also known as Markovian dynamics). In other words, the policy of the agent is independent of all previous states given its current state. This assumption is made for two primary reasons. Most importantly, sequential reasoning over discrete spaces incurs an exponential growth with each successive timestep, making inference over real-world sequences intractable. The benefits of Markov chains are primarily present in their time and space saved. In the case of our game, the agent has one of four possible moves, and basing the policy of an agent off of all of its previous moves would result in 4^n possibilities, where n is the amount of moves played in the game so far. Storage of these moves would also be considerably more cumbersome if full game-length chains were used. Secondly, even if we were able to store an exponentially large amount of data, we still have to contend with the Curse of Dimensionality. In this setting, as the sequence length grows, the policy must operate over a similarly large space, reducing performance. Practically, storing full-game chains of moves would result in new game positions having little or no moves at all for an agent to replicate. As such, we decided

to have our agents have their policy determined exclusively by the current state of the board, a Markov chain of order n=1.

Rewards were conferred to agents based on their actions in the game and chains were learned on an agent-centric partially observed slice of the board. This slice placed the agent at the center of the observability window and captured a 3x3 grid of surrounding cells. If an agent makes a move that results in no change in position, a negative penalty is incurred. Being eaten also incurs a large negative penalty. Any agents involved in consuming another agent will be given a large bonus, scaling with the amount of opposing agents on the team (the smaller the size of the team, the larger reward the agents are given). Entering a "danger zone" (defined as moving into a tile that two or more opposing agents already border) incurs a negative reward, whereas leaving one results in a large positive reward. The winning team gets a large positive reward, and the losing one gets a larger negative reward. The agent that consumes the most opposing agents receives an extremely large positive reward. Data from these tests were plotted in Excel.

Results

For the following experiments, the maximum timestep of the game was set to 1000 timesteps, and the game board's size was set to 10 by 10. Two teams were placed in the game board with 5 agents each. A total of 3,000 simulations against random agents were conducted using the created Markov chain; this chain was learned from 30 human-played games. Agents following the Markovian policy are expected to perform better than those who follow a random policy. Average reward per timestep is plotted against time in Fig. 3.

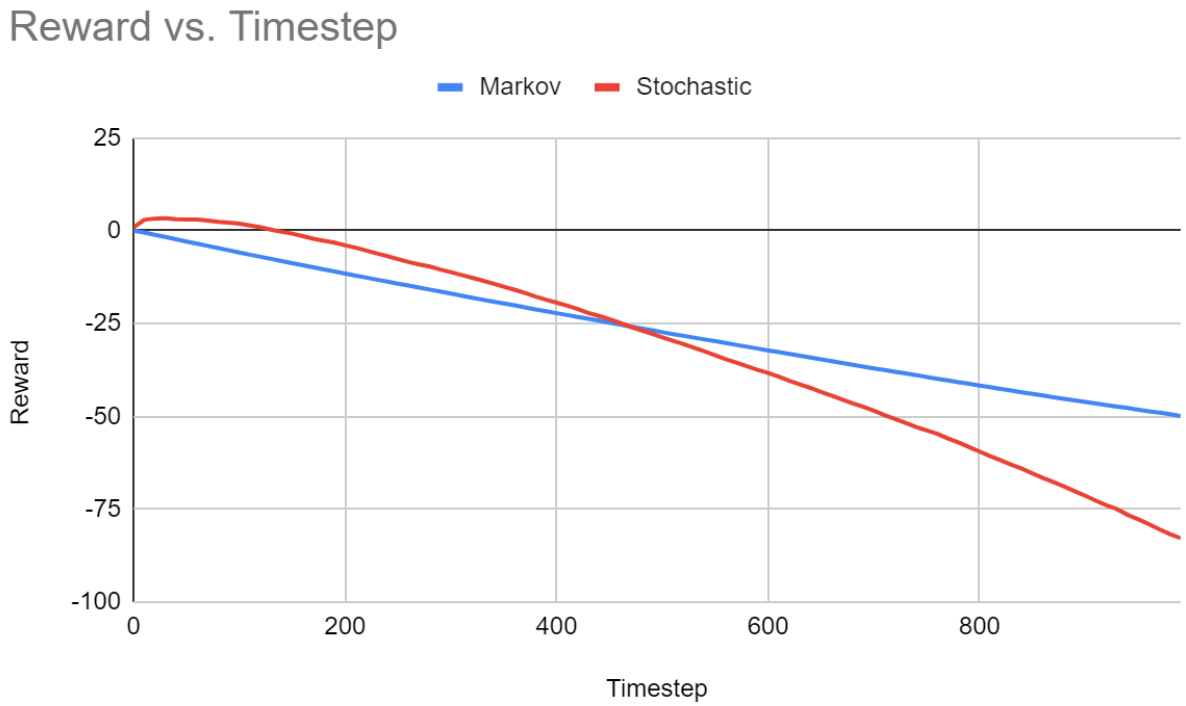

Figure 3. Reward vs. Timestep. The stochastic (random) policy overtakes the Markovian policy in reward for the first 480 timesteps, and than falls behind for the remaining time.

The stochastic policy maintains a reward above 0 for the first 130 timesteps, then slowly experiences a linear decline for the remainder of the timestep. The overall stochastic policy can be modeled by the linear

function $R(t) = -0.0904t + 13.3$. The Markov policy never exceeds 0 reward, however, it maintains a constant decline which can be modeled by the equation$R(t) = -0.0504t - 1.4$. The stochastic policy's decline is nearly 80% steeper than the Markov policy's decline, which ultimately results in the Markov policy's reward being greater than the stochastic policy at timestep 480 and beyond.

Table 1. Win Rates for Teams. The agents following a Markovian policy saw a significantly higher win rate than those who followed a random policy.

Winning Team	Markov-Trained	Random	No Winners
Number of Wins	2,051	780	169
Win Percentage	68.4%	26.0%	5.6%

Agents following the Markovian policy won a majority (68.4%) of the games, whereas agents following a stochastic policy won 26%. 5.6% of games ended in a draw. Over all the games, Markov-trained agents tended to not be consumed much at all compared to random agents, as can be seen in Figure 2. Markov-trained agents followed a linear path in average agents lost (2,160 timesteps/agent lost), whereas random agents tended towards a logarithmic decay in agents lost modeled by the equation $(7.53 - 0.896\ ln\ x)$.

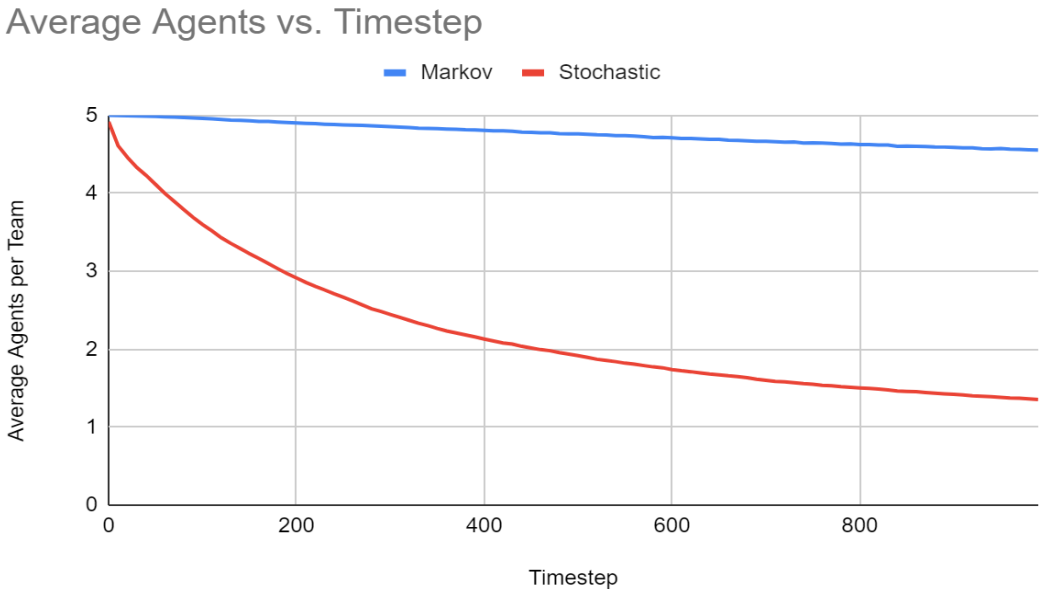

Figure 4. Average Agents vs. Timestep. The agents following a Markovian policy maintained a much higher average amount of agents than those who followed a stochastic policy.

The percentage of games that reached a certain timestep can be seen in Figure 3 below. Games often continued well past the 500 timestep mark, with over 90 percent of the games reaching this point. It takes 840 timesteps for the percentage of games to reach a timestep to drop below 75%. Around 400 timesteps, the lost amount of games levels out, with about 9% of games finishing per 200 timesteps after that timestep.

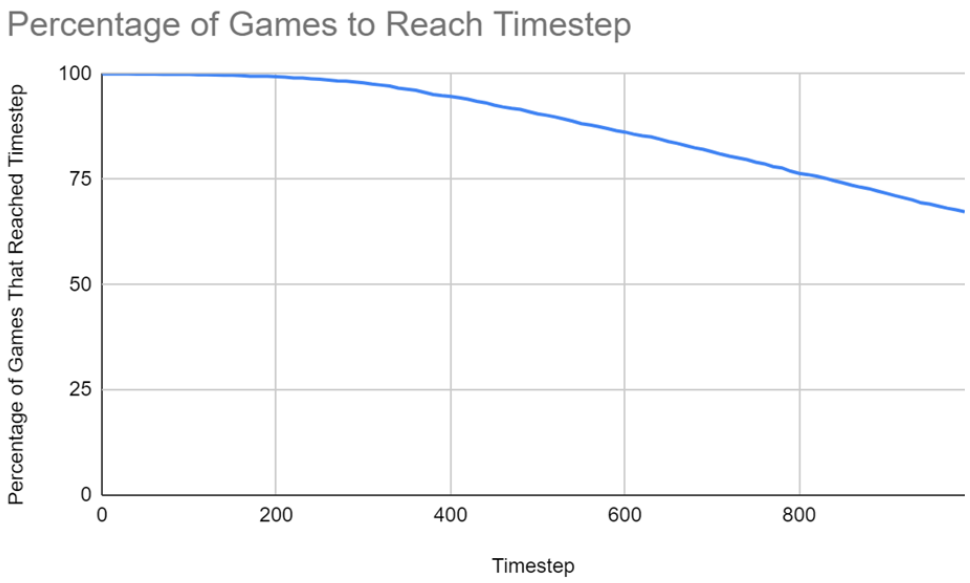

Figure 5. Percentage of Games to Reach Timestep. Nearly every game reaches the 200 timestep mark, but a decline in the amount of games to reach further timesteps follows shortly after.

Additionally, subsets of Markov policies were trained using a limited scope of the data given. In the given matchups, agents that learned from a smaller percentage of games are expected to do worse than those who learned from a bigger percentage. Matchups that have the same amount of training for both teams are expected to be tied. 500 games were played per variant of Markov training for a total of 12,500 games. Comparative win rates based off of the perspective of team 1 can be seen below.

Table 2. Win Rates for Partially Trained Policies. This table shows the win rates of team 1 based on their trained policy (left column). The percentage represents the amount of data used to train the team. The same applies for team 2 (upper row).

	Data Percentage (Team 2)				
Data Pct. (Team 1)	20%	40%	60%	80%	100%
20%	50.8%	43.8%	31.8%	30.2%	33.0%
40%	55.4%	44.2%	17.8%	18.8%	23.4%
60%	65.0%	78.8%	46.8%	49.8%	57.2%
80%	62.6%	78.8%	49.0%	47.8%	50.8%
100%	58.0%	76.0%	44.2%	43.6%	50.0%

Despite slight variances in win rates, the expected trend is mostly held across the 25 averages shown. Outliers can be seen in the 20% vs 60% and 40% vs 60% games, where the win rate for the 20% team is noticeably higher than the win rate for the 40% team. The same pattern holds for 20% vs 80% and 40% vs 80%.

Discussion

Agents with random movement saw noticeably higher rewards between timesteps 60 and 100, followed by a sharp decline in reward gain. The contrast between this and the agents with Markov chains as their policy can be attributed to fewer chances for sporadic movement in random agents due to a smaller average population size compared to the Markov chain controlled agents, which often keep all five of their agents in close ranges to each other. Further points were lost due to constant triggering of the reward loss state upon nearing the vicinity of opposing agents. In the context of this game, loss of agents on a team corresponds to a major decrease in average reward gain. However, time seems to be the biggest factor in all of the reward vs. time functions, as both reward vs. time functions eventually approach linear decay as the game enters a stalemate.

The higher reward of agents following a stochastic policy within the first 200 timesteps could be attributed to the reward function favoring the avoidance of danger zones. Random agents are completely unaware of their presence in danger zones, and since their movement is unpredictable, agents following a Markovian policy will lose their danger zone presence and random agents will be rewarded.

The outliers for win rates can be attributed to the transition between stochastic and trained policies. As agents begin to be trained with more data, they become more predictable. The point at 20% of data likely results in agents being predictable enough to trap without being able to trap other agents, resulting in a considerably lower win rate overall.

Conclusion

In summary, in the context of the game we have created, agents under a stochastic policy perform significantly worse than ones under an intelligent model that mirrors the actions of a human. However, both policies eventually approach linearly decreasing reward functions as the game reaches a stalemate.

Future work regarding this project could involve expanding the datasets that the agents learn off of, and potential recursive learning where the top games played by the Markov chain are then fed back into the Markov chain itself to refine the chain's best moves. Moreover, we could consider more modern modeling solutions to these games, as this served as a promising first step to understanding multi-agent dynamics in games.

Limitations

If time had allowed, a bigger dataset with more simulations run and data to compare against would provide a higher degree of specificity to our analysis. Additionally, more human-play data could have been collected in order to decrease the likelihood of the Markov chain agents running into an unencountered scenario (where the agents then just sample from a random move).

Acknowledgments

I would like to thank [Author] for his continued support and teaching throughout the project.

References

Jin, J. (n.d.). Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising. arXiv.

Leibo, J. (n.d.). A multi-agent reinforcement learning model of common-pool resource appropriation. arXiv.

Lowe, R. (n.d.). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. arXiv.

Shalev-Shwartz, S. (n.d.). Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. arXiv.