Multiagent Reinforcement Learning and Multiarmed Bandits

Michael Krumdick, Geoffrey Liu

1 Introduction

Multi-agent reinforcement learning (MARL) problems are about finding a set of agents that define an optimal set of policies for some environment. Optimality is usually interpreted in terms of exploitability, meaning if we swapped out one of our agents with a new agent, they could not achieve a higher level of performance than the agent they replaced. Most people associate these sorts of equilibrium with the nash equilibrium, but the MARL framework is typically more general.

Most MARL algorithms involve reasoning over empirical games. These games are abstractions over a possibly complex (extensive form) environment that allow us to compute things like the nash equilibrium by framing a meta-game as a simple normal-form game. The most common formulation of these meta-games is to define the actions to be the set of currently existing agents, $\{\pi_1, \pi_2, \dots \pi_n\}$ and the payoff to be the expected payoff of playing the agents against each other. If we can find a equilibrium solution to this game using some kind of meta-solver, we can then find a new agent that is a best response to this equilibrium by training it against this distribution using standard reinforcement learning techniques. We can then add this agent to our population of agents and repeat. This general framework is known as Policy Space Repeating Oracle (PSRO) [4]. For the right meta-solver and best response algorithm, this iterative process will coverage to the solution for our actual game.

One commonly used and powerful meta-solver is α-rank. [6] This computes a ranking over the agents by analyzing the payoff matrix, M. For the two player symmetric case, this matrix stores the expected payoff, or ν of using agent i, π_i, versus agent j, π_j, such that $M_{i,j} = \nu(\pi_i, \pi_j)$. Importantly, in the most common setting for α-rank, the only thing that matters is whether or not $\nu(\pi_i, \pi_j) > \nu(\pi_j, \pi_i)$. Meaning, that we can replace our dense payoff matrix, M, with an equivalent binary one.

Most analyses for α-rank, including its introduction, assume you have a perfect estimate of M and ignore the computation required to compute it from their analysis. However, this is not realistic in most cases. When we are interested in stochastic agents and environments, our M becomes a random variable, $\hat{M}$, that we need to estimate via simulation. Often times, these simulations are expensive to run. Not only that, but our payoff matrix grows exponentially as we add new players and strategies. Although α-rank itself is tractable once given $\hat{M}$, the computation of $\hat{M}$ is a significant bottleneck for real-world use cases. [1]

We can think of computing $\hat{M}$ as an interesting mulit-armed bandit problem: how do we allocate a limited number of samples to each agent configuration to have the lowest probability of making an error with our meta-solver? [9] introduced this formulation and provide some algorithms and bounds for doing so, including the sample complexity required to have a $1 - \delta$ error guarantee for our meta-solution. There have been extensions of this work that increase the performance by using other multi-armed bandit insights [8], but every analysis so far has limited it's scope to solely the computation of α-rank. We want to see if we can provide any new insights by looking at the entire α-rank and PSRO process [5] through this multi-armed bandit lens. At the very least, we would like to perform an empirical analysis of the convergence properties of this setup under different kinds of realistic settings for computing $\hat{M}$. Hopefully, our analysis will lead us to some insight that may help speed up the convergence or decrease the memory complexity of this family of algorithms. One idea that we have is to see if we can learn anything from the "streaming bandits" formalism to aid in the deletion of agents.

2 Motivation

For the past few decades, the large majority of research being done in reinforcement learning only considered the single-agent or independent learning case. Here, the environment is static. Anything being simulated as part of the environment will never learn how to adapt to the agent. For example, the common set of Atari baselines [3] include games that simulate other agents, i.e. the other paddle in pong, but these

agents will never adapt in response to the agent. MARL involves learning agents any type of environment in which each agent has to learn an optimal behavior in the presence of other agents who are trying to optimize their own behaviors.

MARL has received large amounts of interest recently, as many people have identified it as a major gap for the practical use of reinforcement learning. Real world environment often involve dealing with other agents who are non-static. These approaches broadly have application areas including war gaming, advertising, trading, and policy simulation to name a few.

3 Plan

Our plan is to follow in a framework similar to [2]. We will first:

1. **Exploration:** Implement and analyze baselines in simple normal-form games. This gives a quick iterative feedback loop that we can use to compare our approximate algorithm to the ground truth solution.
2. **Design:** Through the lens of multi-armed bandit problems, attempt to modify the existing algorithms to increase performance along some dimension.
3. **Application:** Modify this algorithm to work with deep reinforcement learning agents and demonstrate its application on a more complex environment.

Since this is a research project, we want to make sure that we have a good number of off-ramps when things do not go as planned. Once we have implemented our baselines, if we cannot develop an improved algorithm, we can pivot to an application setup where we apply some of the mutli-agent algorithms to an interesting problem and perform a comparative analysis of their convergence. The only step that is mandatory is the completion of our exploratory setup.

In terms of environments, we want to find something that is scalable. Meaning, we want a problem that can both work as a normal-form game (or similar) and by updating the parameters it can scale to something that will require deep reinforcement learning. By having a parameter or set of parameters that allow us to smoothly scale complexity, we can treat this as another dimension in our analysis. Meaning instead of having a sliding scale of separate environments that represent different levels of complexity, we can plot complexity as a function of a specific parameter. One possible environment that we current have in mind are Erdos-Selfridge-Spencer(ESS)[7] games. These not only fit the previous description, they also have an analytical optimal solution that can always be computed as a linear function of the state. In other words, we can compute the exploitability of an agent analytically without relying on an estimate. However, the main types of ESS games seem to have action spaces that would complicated to simulate as a simple normal-form game and they are lacking open source implementations. We are also interested in Kuhn and Leduc Poker, since they are a more common baseline for these types of algorithms.

4 Related Works

We have cited a few related works throughout this document, but the most important ones are: the original α-rank paper [6], the paper about connecting α-rank with PSRO [5], and the paper about interpreting the payoff matrix estimation as a multi-armed bandit problem [9]. This paper gave a PAC bound on the computation of the α-rank ordering, but did not connect it with the PSRO version. The PSRO version gives a convergence analysis for their algorithm with respect to the computation of the α-rank, but they do not connect it to the estimate of $\hat{M}$. We are hoping to be able to bridge that gap.

References

[1] α^{α}-rank: Scalable multi-agent evaluation through evolution. *CoRR*, abs/1909.11628, 2019. Withdrawn.

[2] Thomas Anthony, Tom Eccles, Andrea Tacchetti, János Kramár, Ian Gemp, Thomas C. Hudson, Nicolas Porcel, Marc Lanctot, Julien Pérolat, Richard Everett, Roman Werpachowski, Satinder Singh, Thore Graepel, and Yoram Bachrach. Learning to play no-press diplomacy with best response policy iteration, 2020.

[3] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012.

[4] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning, 2017.

[5] Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Pérolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, Zhe Wang, Guy Lever, Nicolas Heess, Thore Graepel, and Rémi Munos. A generalized training approach for multiagent learning. *CoRR*, abs/1909.12823, 2019.

[6] Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M. Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. alpha-rank: Multi-agent evaluation by evolution, 2019.

[7] Maithra Raghu, Alex Irpan, Jacob Andreas, Robert Kleinberg, Quoc V. Le, and Jon Kleinberg. Can deep reinforcement learning solve erdos-selfridge-spencer games?, 2018.

[8] Tabish Rashid, Cheng Zhang, and Kamil Ciosek. Estimating α-rank by maximizing information gain. *CoRR*, abs/2101.09178, 2021.

[9] Mark Rowland, Shayegan Omidshafiei, Karl Tuyls, Julien Pérolat, Michal Valko, Georgios Piliouras, and Rémi Munos. Multiagent evaluation under incomplete information. *CoRR*, abs/1909.09849, 2019.