Game Theoretical Approaches in Multi-Agent Reinforcement Learning

Sai Ganesh Nagarajan* Student ID: 1003206

Jin Xing Lim* Student ID: 1003964

1 Introduction

We study the game theoretical problems that arise in multi-agent learning. In particular, we are interested in the simplest possible setting of symmetric zero-sum games,i.e, games that look like rock-paper-scissors, in the sense that there is no dominant agent/strategy. A recent paper that tries to address this problem [2], talks about a variant of previously proposed multi-agent reinforcement learning algorithm [4] called the Policy Space Response Oracles (PSRO), where they showed improved performance compared to learning independent reinforcement algorithm. We will look at these two algorithms and interpret the game theoretical intuition behind them. In addition, we re-created some experiments in [2] to understand the working of the algorithms.

2 Policy Space Response Oracles

In this section, we describe the fundamental multi-agent learning algorithm (Figure 1), first described in [4], based on which more recent developments have been made for the symmetric zero-sum case, which discusses variants of PSRO.

Algorithm 1: Policy-Space Response Oracles

input : initial policy sets for all players Π
Compute exp. utilities U^{Π} for each joint $\pi \in \Pi$
Initialize meta-strategies $\sigma_i = \text{UNIFORM}(\Pi_i)$
while *epoch* e *in* $\{1, 2, \cdots\}$ **do**
  for *player* $i \in [[n]]$ **do**
    for *many episodes* **do**
      Sample $\pi_{-i} \sim \sigma_{-i}$
      Train oracle π_i' over $\rho \sim (\pi_i', \pi_{-i})$
    $\Pi_i = \Pi_i \cup \{\pi_i'\}$
  Compute missing entries in U^{Π} from Π
  Compute a meta-strategy σ from U^{Π}
Output current solution strategy σ_i for player i

Figure 1: *PSRO Algorithm*

The main intuition of the algorithm arises from how we analyze games in general, when analyzing the strategies taken by agent i, we fix the other agents' strategies and see the effect on the payoff to agent i, if i changes his/her strategy.

The high-level description of the algorithm is as follows:

1. Initially all agents have their own initial policies and say they have an uniform probability distribution over these initial policies.

* Indicates equal contribution.

2. Next, an empirical game matrix is calculated by computing the expected payoffs that each agent would get for using policy π_i. Enumerate the matrix with all possible combinations of the policies that each agent can take.
3. Given agent i and fixing all the agents stochastic policies, i.e, using the current probability distribution over their policies (also referred to as meta strategies), agent i then learns a "best response" using an oracle, which may be a standard single agent reinforcement learning algorithm, or as simple as a function that implements gradient descent.
4. Once all the agents repeat the above step, these policies are added to the empirical game matrix and the remaining entries are filled.
5. Finally, the agents update their meta strategies over their "deterministic" policies based on the empirical payoff matrix and using an update rule such as MWU, or RD or PRD (to encourage exploration).

The nature of the algorithm is to decouple the game theory aspect (obtaining the meta strategies using an update rule) and the reinforcement learning aspect, which goes into the oracle and helps us obtain a best response policy when fixing the others. This allows to study them separately. In addition, the algorithm's limit behavior depends on the nature of the empirical game and the update rule being used. We know that most of the discrete time update rules for high dimensional games are chaotic and diverging [1]. Even in continuous time, the dynamics are known to be recurrent for zero-sum games [5]. This in itself raises many interesting questions with regards to training of the agents in these settings. Since, the general case is complex, we will try to analyze a much simpler setting and look at the problem of training agents when playing a symmetric zero-sum game. The main question is how do we aim to get "better" agents while playing a symmetric zero-sum game, where there is no clear winner (or dominant agent).

3 Towards Population Objectives in Zero-Sum Games

The main argument in [2], is that in the case when agents are playing a symmetric zero-sum game, where there is no one best agent, one way to train is by diversifying the strategies that the agents play, by trying to discover new strategies. So this requires maintaining a population of agents and improving this population over time, as opposed to self-play where the best agent trains against itself and improves over time. The paper argues that self-play is not effective in games where there is no best agent and hence defines certain classes of games that can be studied by keeping this in mind.

3.1 Zero-sum Games: Transitive + Cyclic

The main idea is that, there are some games that are transitive meaning, any local improvements will eventually lead to global optimum. We can imagine these games to have "gradient-line" behavior. These are the class of games where self-play may suffice, for instance, Chess, Go, Laser-tag etc. There are other games which are cyclic, in the sense that there is no best agent. These are the games that look like rock-paper-scissors. For instance, more complex games such as DOTA, Starcraft may exhibit this type of behavior where you require different skills in a team and diversification to win matches.

3.2 Training Objectives

We establish some notations to talk about the algorithm. Consider $v, w \in \mathcal{W}$, where v, w maybe weights of a neural net. We will henceforth refer to them as agents. The agents are chosen from the space of agents $\mathcal{W}$. We define $\phi(v, w)$ to be the expectation of v winning against w and according to this definition we have $\phi(v, w) = -\phi(w, v)$, since it is a symmetric zero-sum game. We can then define a population of agents $\beta = \{v_1, v_2, \ldots, v_K\}$, where this population contains K agents. Now, we can populate the empirical game matrix (or evaluation matrix A_β) as the payoff matrix when population β competes with itself. Formally, $A_\beta^{i,j} = \phi(v_i, v_j)$, where $A_\beta^{i,j}$ is the $(i, j)^{th}$ element of A_β. Finally, with the population at hand, we can look at training agents when, i.e, getting a best response oracle when the other agents in the population plays a Nash mixture which is the notion that has replaced a "best agent" in the symmetric zero-sum case and this leads to two algorithms which are modifications of the PSRO for self play in symmetric-zero sum games. For illustration, we look at

one algorithm called PSRO -rectified Nash ($PSRO_{rN}$), which is explained in the next section. Also, the Nash equilibrium for this game can be computed via a simple linear program (which however, grows with the number of agents in the population).

3.3 Comparison between Self-play and $PSRO_{rN}$

Two of the algorithms that were used for comparison in Balduzzi et al paper [2] are Self-play and $PSRO_{rN}$.

Algorithm 2 Self-play

input: agent $\mathbf{v}_1$
for $t = 1, \ldots, T$ **do**
  $\mathbf{v}_{t+1} \leftarrow \text{oracle}\left(\mathbf{v}_t, \phi_{\mathbf{v}_t}(\bullet)\right)$
end for
output: $\mathbf{v}_{T+1}$

Figure 2: *Self-play Algorithm*

For Self-play algorithm (Figure 2), it essentially trains the next agent using the current agent via an oracle. This algorithm has been shown to be effective in training agents in games like Chess, Go and other games [8]. However, Self-play requires the game to be transitive in order for the training to be effective. Training through self-play on cyclic games, such as disc game below, is not effective as success against one agent does not guarantee success against the others.

Algorithm 4 Response to rectified Nash (PSRO_{rN})

input: population $\mathfrak{P}_1$
for $t = 1, \ldots, T$ **do**
  $\mathbf{p}_t \leftarrow$ Nash on $\mathbf{A}_{\mathfrak{P}_t}$
  for agent $\mathbf{v}_t$ with positive mass in $\mathbf{p}_t$ **do**
    $\mathbf{v}_{t+1} \leftarrow \text{oracle}\left(\mathbf{v}_t, \sum_{\mathbf{w}_i \in \mathfrak{P}_t} \mathbf{p}_t[i] \cdot \lfloor \phi_{\mathbf{w}_i}(\bullet) \rfloor_+\right)$
  end for
  $\mathfrak{P}_{t+1} \leftarrow \mathfrak{P}_t \cup \{\mathbf{v}_{t+1} : \text{updated above}\}$
end for
output: $\mathfrak{P}_{T+1}$

Figure 3: $PSRO_{rN}$ *Algorithm*

As such, the paper introduces another $PSRO_{rN}$ (Figure 3) that works well in both transitive and cyclic games. In each iteration, each agent, with support under the Nash equilibrium, is trained against the Nash-weighted mixture of agents that it beats or ties. The idea is to make agents to "further strengthen on their strengths and ignore their weaknesses". For transitive games, this algorithm will reduce to classical self-play as the Nash equilibrium would be $(0, ..., 0, 1)$. For cyclic games, this algorithm will generate a more diversified population of strategies to counter different variations of strategies. The intuition behind this is that, the oracle produces an agent that maximizes the convex combination of non-negative functions as shown in the figure. So this "maximum" agent has to tie with every other agent or perform better and this has a net effect of improving (or discovering) strategies that perform better and since each significant agent in the population is trained, this leads to a net improvement in the population.

4 Experiments

We ran several experiments to show how the populations from Self-play and $PSRO_{rN}$ algorithms evolved for disc game (cyclic) and transitive version of the lotto game separately.

4.1 Disc Game [2]

Let $v, w \in \mathbb{R}^2$, where the payoff function is

$$\phi(v, w) = v_1 w_2 - v_2 w_1.$$

Thus, every point in the $\mathbb{R}^2$ is considered as an agent where the value of both coordinates are considered as its strategy. Thus, the payoff of agent v against agent w is measured by $\phi(w, v)$. This game is cyclic (see Figure 4, which is taken from Balduzzi et al paper).

Note that rock-paper-scissors can be embedded in this disc game via the following transformation:

$$r_\epsilon = \frac{\sqrt{3}\epsilon}{2}\big(\cos 0, \sin 0\big), p_\epsilon = \frac{\sqrt{3}\epsilon}{2}\big(\cos\frac{2\pi}{3}, \sin\frac{2\pi}{3}\big), s_\epsilon = \frac{\sqrt{3}\epsilon}{2}\big(\cos\frac{4\pi}{3}, \sin\frac{4\pi}{3}\big)$$

to obtain the following evaluation matrix:

$$A_{\{r_\epsilon, p_\epsilon, s_\epsilon\}} = \begin{bmatrix} 0 & \epsilon^2 & -\epsilon^2 \\ -\epsilon^2 & 0 & \epsilon^2 \\ \epsilon^2 & -\epsilon^2 & 0 \end{bmatrix}$$

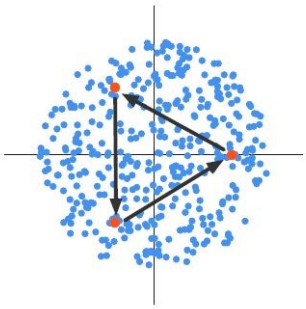

Figure 4: *Disc game*. The points (blue and red) represent a set of strategies from the disc game. The three cyclic strategies' (rock-paper-scissors) relations are shown as red points.

We ran Self-play and $PSRO_{rN}$ algorithms with the initial population as $\{r_1, p_1, s_1\}$ (as shown in the transformation above with $\epsilon = 1$). For training under Self-play, the population evolved spirally outwards as shown in Figure 5. However, as for the population under the $PSRO_{rN}$ algorithm training, the population spread out from each of the initial conditions as shown in Figure 6. We can see that population from the $PSRO_{rN}$ algorithm diversified much better than the Self-play algorithm, which only improves the next agent based on the current agent, causing it to move out in a spiral manner.

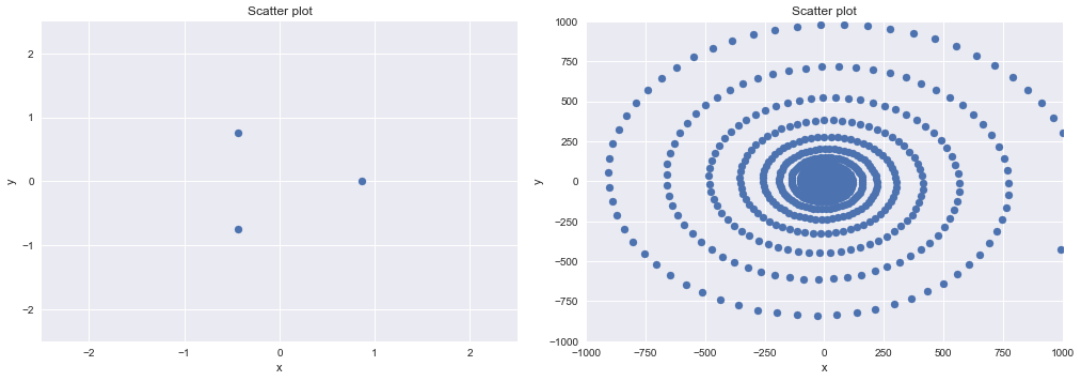

Figure 5: *Evolution of population through Self-play.* Left: Initial population $\{r_1, p_1, s_1\}$. Right:Final population with size of 1536 (after epochs of training)

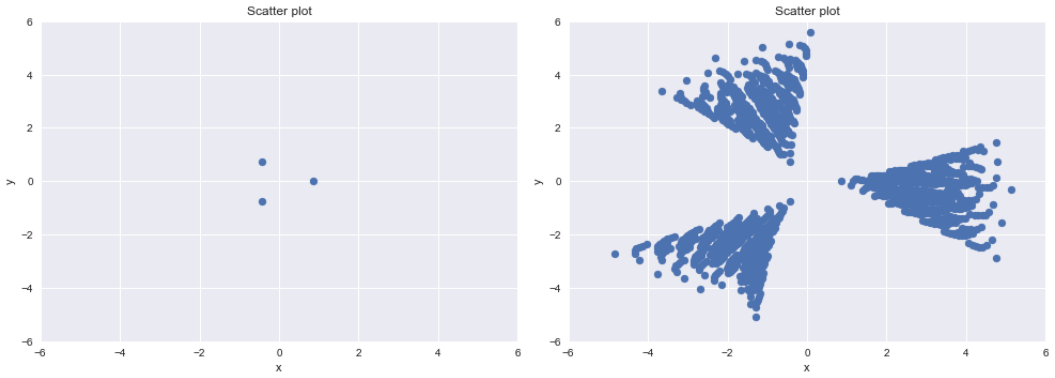

Figure 6: *Evolution of population through* $PSRO_{rN}$. Left: Initial population $\{r_1, p_1, s_1\}$. Right:Final population with size of 1536 (after epochs of training)

4.2 Transitive Lotto

This game is a resource allocation game defined over $[-1, 1]^2$ space in the $\mathbb{R}^2$. There is a fixed set C of n customers, $\{c_1, ..., c_n\}$, each being a point in the space. Each agent's strategy $(\mathbf{p}, \mathbf{v}) = \{(p_1, \mathbf{v_1}), ..., (p_k, \mathbf{v_k})\}$, distributes weight p_i over server $\mathbf{v_i} \in [-1, 1]^2$ for $1 \leq i \leq k$. The payoff function between two agents, $(\mathbf{p}, \mathbf{v})$ and $(\mathbf{q}, \mathbf{w})$, is defined as

$$\phi((\mathbf{p}, \mathbf{v}), (\mathbf{q}, \mathbf{w})) = \sum_{i,j=1}^{c,k} (p_i v_{ij} - q_j w_{ij})$$

where the scalars v_{ij} and w_{ij} depend on the distance between customer i and the servers:

$$(v_{i1}, ..., w_{ik}) = softmax(-||\mathbf{c_i} - \mathbf{v_1}||^{\mathbf{2}}, ..., -||\mathbf{c_i} - \mathbf{w_k}||^{\mathbf{2}}).$$

This game is a transitive version of the lotto game.

We ran Self-play and $PSRO_{rN}$ algorithms for different number of servers ($k = 3, 9, 15$). As mentioned above, we observed that for all cases, the $PSRO_{rN}$ algorithm works the same as Self-play for this transitive lotto game. Regardless of the number of servers ($k = 3, 9, 15$), all servers eventually converge to the centroid of the 9 customers (See Figure 7, 8 and 9), which is the best location to serve all customers. As for the probability distribution of the servers, eventually it will converge to uniform distribution.

Figure 7: $k = 3$ *servers.* Right points: Location of 9 customers. Blue points: Location of 3 servers. Magenta point: Centroid of the 9 customers. Left: Initial locations of the servers. Right: Final location of the servers after 100 epoch

Figure 8: $k = 9$ *servers.* Right points: Location of 9 customers. Blue points: Location of 9 servers. Magenta point: Centroid of the 9 customers. Left: Initial locations of the servers. Right: Final location of the servers after 100 epoch

Figure 9: $k = 15$ *servers.* Right points: Location of 9 customers. Blue points: Location of 15 servers. Magenta point: Centroid of the 9 customers. Left: Initial locations of the servers. Right: Final location of the servers after 100 epoch

5 Future Works

For future work we can try to extend the idea of self-play to multi-agent settings, via the idea of network (or polymatrix) games, as the understanding of game dynamics [7, 6] and the tractability of Nash equilibria [3] maybe helpful in achieving effective learning in more complicated multi-agent settings by encoding the interaction through the edges of the network.

6 Conclusion

We analyze experimentally and provide intuition to some game theoretical problems surrounding multi-agent learning by looking at some recent learning algorithms.

References

[1] J. P. Bailey and G. Piliouras. Multiplicative weights update in zero-sum games. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 321–338. ACM, 2018.

[2] D. Balduzzi, M. Garnelo, Y. Bachrach, W. Czarnecki, J. Pérolat, M. Jaderberg, and T. Graepel. Open-ended learning in symmetric zero-sum games. *CoRR*, abs/1901.08106, 2019.

[3] Y. Cai, O. Candogan, C. Daskalakis, and C. Papadimitriou. Zero-sum polymatrix games: A generalization of minmax. *Mathematics of Operations Research*, 41(2):648–655, 2016.

[4] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Perolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4190–4203. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7007-a-unified-game-theoretic-approach-to-multiagent-reinforcement-learning.pdf`.

[5] P. Mertikopoulos, C. Papadimitriou, and G. Piliouras. Cycles in adversarial regularized learning. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2703–2717. SIAM, 2018.

[6] S. G. Nagarajan, S. Mohamed, and G. Piliouras. Three body problems in evolutionary game dynamics: Convergence, periodicity and limit cycles. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 685–693. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[7] G. Piliouras and J. S. Shamma. Optimization despite chaos: Convex relaxations to complex limit sets via poincaré recurrence. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 861–873. SIAM, 2014.

[8] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.