Reinforcement Learning and Decision Making

Project 3: Correlated-Q

Hanxiao Wu, hwu429@gatech.edu

Last Git Hash: 0e118e32aebf3dc0a21999f50e3a0a023a836bb9

1 INTRODUCTION

In this analysis, we will replicate the experiments done by Greenwhald on how different multi-agent reinforcement learning algorithms work for the soccer game. The soccer game is a zero-sum game without deterministic equilibrium policies. To frame it as a reinforcement learning environment, we need to specify the state, action and reward.

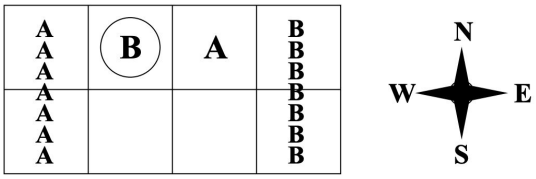

Figure 1 — Soccer game.

The state consists of the location of player A, location of player B, and who owns the ball. As shown in the figure, there are eight possible locations for each player, and the ball is either with player A or player B. All the state components are discrete. So the state can be represented as a matrix with three dimensions: (8,8,2).

There are five actions available: N, S, E, W, and stick. Each action represents how the player moves in the grid. The players' actions are executed in random order. When a player with the ball tries to move to another player's current position, then the ball changes possession.

The reward is calculated when the player with the ball hits the goal. When player A moves into his own goal, then he will receive +100, and player B will receive -100; if player A moves into the other player's goal, he will receive -100 and player B will receive +100. The game then ends.

2 METHODOLOGY

In the paper, there are four algorithms implemented to solve the soccer game. All of them are based on the Q-Learning framework with some modification.

2.1 Multi-Agent Q-Learning

In one-player Markov game, Q-Learning uses the Q function as the following:

$$Q^*(s,a) = (1-\gamma)R(s,a) + \gamma \sum P[s'|s,a]V^*(s')$$

$$V^*(s) = \max(a, Q^*(s,a))$$

The Q(s,a) is the sum of immediate reward and the expected discounted long term reward from taking the optimal policy thereafter.

In a multi-player Markov game, it will use action vectors instead of action. The action vectors $\vec{a}$ represent the action of each player. The Q function of the player *i* is defined as below:

$$Q_i(s, \vec{a}) = (1-\gamma)R_i(s, \vec{a}) + \gamma \sum P[s'|s, \vec{a}]V_i(s')$$

The challenge here is that, unlike the one-player Q function where the V* is simply the reward of taking the best action that maximizes the long term reward, here the value function of the player *i* will be affected by other players' actions as well. There are multiple ways proposed to define the value function.

The simplest way is to continue to use the regular Q-Learning and ignore other players' actions in the value function. In other words, player *i* will consider other players behavior as part of the environment and simply gets V by choosing the optimal action on it's own that maximizes Q.

2.2 Friend-Q

Littman proposed the Friend-Q value function for coordination game, for which all the players' reward functions are equivalent. In this setting, every player should assume that others would cooperate to maximize the total reward. The value function is defined as below:

$$V_i(s) \ = \ \max(\vec{a},\, Q(s, \vec{a}))$$

In Friend-Q, the value of state s is set by choosing the best joint action (action vector $\vec{a}$) that maximizes the reward. As the reward functions are the same across players, each player should assume that others will cooperate and choose the best joint action as well.

2.3 Foe-Q

Littman proposed the Foe-Q mixmax value function for zero-sum game, for which one player's gain is the other player's loss. In this minmax equilibria, each player assumes that opponents are adversarial and will choose actions that minimize the reward others receive. The value function is defined as below:

$$V_1(s) \ = \ \max(\pi_1,\ \min(a_2,\ \textstyle\sum_{a1} \pi_1(a_1)\, Q(s,\, a_1, a_2))) \ = \ -V_2(s)$$

In Foe-Q, the goal of the players is to find a policy π that maximizes the minimum reward result given that opponents are adversarial. In order to find that policy, which is the probability of choosing each action, linear programming is implemented to find the probability that maximizes the value function.

2.4 Correlated Equilibria Q

Correlated Equilibria (EQ) is different from Nash Equilibria where the probability distributions over actions are independent among each agent. EQ is a probability distribution over the joint space of actions. Greenwhald discussed four CE-Q variants, and in this experiment, we'll use *u*CE-Q. *u*CE-Q assumes the opponent will always find a correlated equilibrium that gives the highest sum of average returns. The objective function is defined as below:

$$\sigma \ \in \ arg\max(\sigma,\ \textstyle\sum_i \sum_{\vec{a}} \sigma(\vec{a}) Q(s, \vec{a}))$$

The goal is to find the probability distribution of the joint actions, which can be computed easily via linear programming.

3 EXPERIMENTS REPLICATION

I'm going to replicate four experiments in the paper. The chart below shows Q-value difference over iterations. The Q-value here refers to the player A's Q value where player A location = 2, player B location = 1, ball owner = B, player A action = South and player B action = stick.

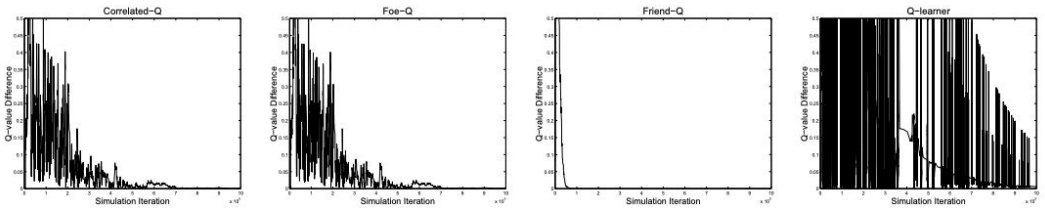

Figure 2 — Convergence in the soccer game (Greenwald paper).

3.1 Regular Q-Learning Replication

In regular Q-learning, where the player simply ignores the opponent's action, the process is pretty similar to single-agent Q learning. Each player will keep their own Q tables, and the Q tables are updated with (s, a, r, s') experience tuples at each step, independently.

The paper states experiment setting to be gamma = 0.9, epsilon → 0.001, alpha → 0.001. In this regular Q learning replication, I have epsilon and alpha both start from 1 and assume a constant decay rate that makes epsilon and alpha end at 0.001 after 1M iterations. So the alpha = 1, decay rate = 0.999995, and alpha min = 0.001. Same schedule for epsilon.

Figure 3 shows a similar result to the one in the paper. Though the Q-value difference decreases overtime, it hasn't converged within 1M iterations. The difference decreases simply because our learning rate is decaying overtime. This is consistent with our expectations. Intuitively, regular Q-learning is hard to converge as players have no assumption about the opponent and are not able to adjust actions based on assumption.

3.2 Friend-Q Replication

In Friend-Q, the goal is to find the joint action that maximizes the reward, assuming the other player will cooperate. There are two Q-tables, and since joint action is used, each Q table has action space of 5x5. At each step, the player will choose the optimal joint action with an optimistic assumption that the other player will do the same.

With the same parameters as in regular Q-learning, I'm not able to replicate the Friend-Q result in paper. After experimenting different learning schedules, I found that with a learning rate starting from 0.2 and a 0.99997 decay rate, our replication resembles the paper perfectly.

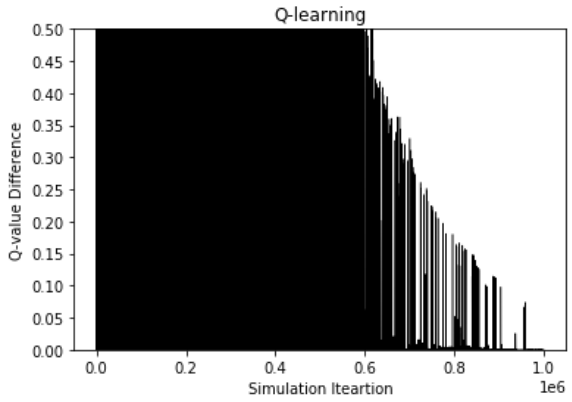

Figure 3 —Q-Learning Replication.

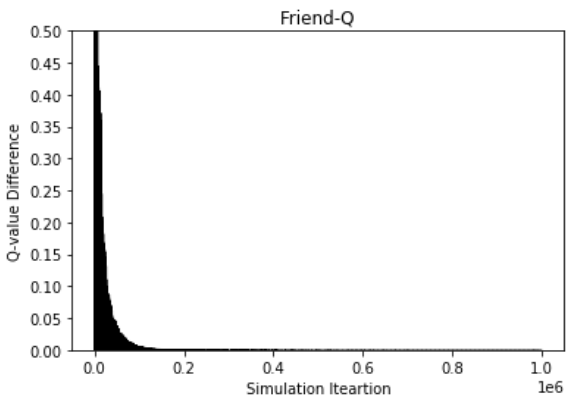

Figure 4 —Friend-Q Replication.

As shown in Figure 4, Friend-Q converges pretty fast within the first 0.1M iterations. However, it converges to a deterministic policy for player B and the solution is not rational, since both players assume that the opponent will score for them.

3.3 Foe-Q Replication

In Foe-Q, the goal is to find a policy π that maximizes the minimum reward result given that opponents are adversarial. In this experiment, two more policy tables are used to record the probability of actions at each state. The policies are found using linear programming. LP will also return the utility of performing the action probability at this state, which will be used as the value function in the Q table update process. In the experiment, most parameters are the same as regular Q-learning replication. One modification after searching is to use a learning rate decay rate of 0.99995 to match the scale of Q-value difference in the original paper.

Figure 5 is similar to the one in the paper. The Q-value difference decreases over iterations and Foe-Q converges within 1M iterations. It's also consistent with the theory that minmax function is guaranteed to converge.

What is different is that our replication converges at around 0.5M iterations while in the paper it converges at around 0.7M. There are several hypotheses. One hypothesis is that the learning rate decay schedule is a bit different, especially in the later stage. For example, between 0 to 0.3M, the replication matches the paper. However, after 0.4M, our replication starts to converge much faster. If more time were allowed, I would explore more decay schedules.

Another hypothesis is that linear programming contributes to the difference. In this Feo-Q replication, I used the package cvxpy with default solver. If another linear programming method is used in the paper, it's likely that the convergence schedule will be a bit different.

3.4 uCE-Q Replication

In uCE-Q, the goal is to find a policy π, which is the probability distribution over the joint space of actions, that maximizes average reward. Thus there is a policy table that records the probability of action pairs. The policy is found using linear programming. The LP will return

the optimal V and also the policy used to achieve the best reward. In the experiment, all parameters for Q-learning are the same as Foe-Q replication.

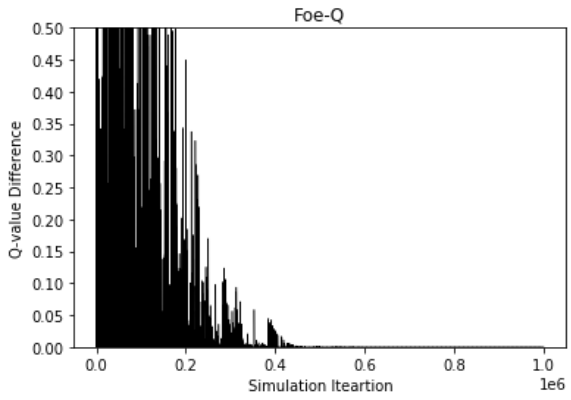

Figure 5 —Foe-Q Replication.

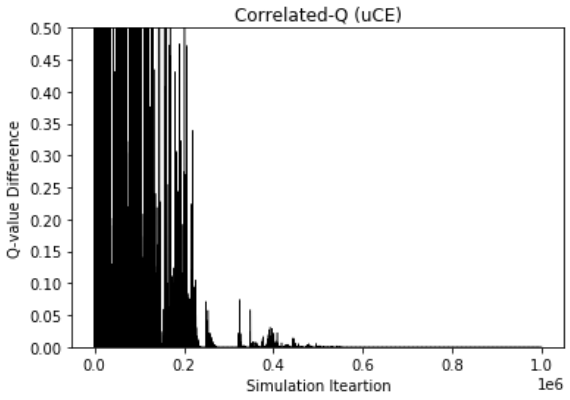

Figure 6 —uCE-Q Replication.

Figure 6 is similar to the paper. uCE-Q converges within 1M iterations, and the performance is very similar to Foe-Q. This is expected because as stated in the paper, in two-player zero-sum games, uCE-Q learns the minimax equilibria, which is exactly what is learnt in Foe-Q.

Compared to the paper, there are two differences. One is that here the uCE-Q converges a bit faster than in the paper, which it is also observed and discussed in the Foe-Q replication. Another difference from the paper is that, uCE-Q performance looks not exactly the same as Foe-Q. This is because when replicating uCE-Q, I used a different linear programming method. Since uCE-Q has more complicated constraints and needs much more time to train, I changed to the cpyopt package with GLPK solver, and also set timeout to prevent long time loops. Those changes are likely to result in slightly different convergence schedules.

4 PITFALLS AND LEARNINGS

There are two challenges in this project. One is about linear programming. It took time to understand the constraints and object functions in Foe-Q and uCE-Q, and to form them correctly in the LP solver. Another challenge is that in the paper there is little description on the implementation details. I was confused about how to set Q tables and policy tables at the beginning, and had to try a bunch of parameters to match the figures in the paper. Luckily, there are a lot of valuable discussions on Piazza and OH that helps me speed up. For example, I learned to switch to cvxopt from cvxpy for uCE-Q and it saves me lots of time.

5 REFERENCES

Greenwald, Amy, Keith Hall, and Roberto Serrano. "Correlated Q-learning." *ICML*. Vol. 20. No. 1. 2003.

Littman, Michael L. "Friend-or-foe Q-learning in general-sum games." *ICML*. Vol. 1. 2001.

Littman, Michael L. "Markov games as a framework for multi-agent reinforcement learning." *Machine learning proceedings 1994*. Morgan Kaufmann, 1994. 157-163.

Soccer game Robocup Environment: https://github.com/zengliX/SoccerGame