Reinforcement Learning Project 3

Peter Lucia
plucia3@gatech.edu

April 14, 2020

Git hash: b1bfdd531efd0bc57254c24d7518719020ad4c5a

Abstract

In this work, I develop a system to replicate results of the soccer game experiment of Greenwald et. al in [1]. The soccer game environment is a general-sum Markov game in which there are no deterministic equilibrium policies. Therefore, it is an ideal environment in which to expose the tradeoffs of four Q-learning based algorithms: Q-Learning, Friend-Q, Foe-Q, and Correlated-Q Learning. The implementation and results of each replicated experiment are discussed in context of the game and the algorithm, including any similarities, differences, pitfalls, or assumptions that had to be made.

1 Introduction

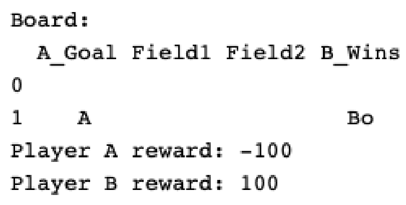
1.1 Soccer Game Environment

The soccer field is represented by a two dimensional grid with two rows and four columns, and is shown in Figure 2. The two squares in the first column are goal positions for Player A, while the two squares making up the last column make up the goal positions for Player B. The soccer ball is represented by a circle, and is always either in the possession of Player A or B. The possible actions are move North, South, East, West, or Stick.

At the start of the game, Player B has the ball in the top row and second column (0,1), while Player A is positioned one block over in position (0, 2). If Player B takes a deterministic policy, B might be subject to indefinite blocking by Player A, while if B takes a non-deterministic policy, B can expect to eventually bring the soccer ball to the goal [1].

It is randomly decided whether Player A or player B moves first. If the player without the ball moves into the same position as the player with the ball, no more moves occur for that turn. However, if the player with the ball moves second and ends up moving into the same position as the player without the ball, the player without the ball will then get possession of the ball [1].

One source of difficulty surrounding the rules of the soccer game environment description from [1] was how to handle collisions and ball possession changes between the two players. The description in [1] regarding whether two players could occupy the same square at the same time in every possible circumstance can be interpreted in multiple ways. Therefore, in these experiments, the assumption was made that two players could in fact occupy the same square at the same time if a steal occurred, where the player that has the ball moves second and also accidentally enters the cell of his opponent. This assumption may increase the likelihood of either player scoring, since both players cannot absolutely block their opponent by taking a particular position. While replicating the experiments, this was considered as one reason for dissimilarities between original and replicated graphs. So, two experiments were run with Q-learning and the Friend-Q algorithms, as well as a modified environment implementation that prevented players from occupying the same squares. The results showed neglible differences when compared with the original environment implementation that allowed players to occupy the same positions in the event of a steal. Therefore, in these experiments, the players are allowed to occupy the same position in this one circumstance.



```
Board:
  A_Goal Field1 Field2 B_Wins
0
1    A                    Bo
Player A reward: -100
Player B reward: 100
```

Figure 1: Rendered Environment

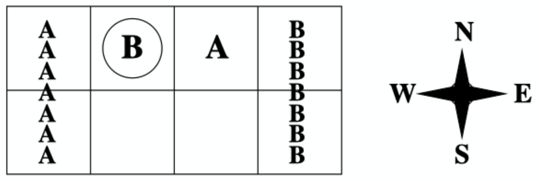


Figure 2: Soccer Field

If a player moves into their respective goal position while having possession of the ball, they gain a reward of +100, the opposing player receives a penalty of -100, and the game ends. The soccer game environment implementation is modeled after the open-ai gym environment structure, but doesn't subclass any gym environment directly, as this was found to be unnecessary during later stages of the development process. Each square on the field is given an index, such that the top row of positions have indices 0,1,2, and 3 from left to right, while the bottom row positions have indices 4,5,6, and 7. A minimalistic render function is implemented and its output, shown in Figure 1, is used to visualize the actual positions of the two players, the ball, the field, and any rewards received after taking each action.

1.2 Q-Learning

1.2.1 Q-Learning Implementation

With the Q-Learning strategy, each player focuses independently on their strategy to maximize reward without making any assumptions about the intention of their opponents.

There is no information sharing between the agents, other than the fact that both agents are able to observe the position of the opponents at all times. Therefore, each agent has their own respective Q table and takes an ϵ-greedy approach to action selection, increasingly exploiting their knowledge of which actions produce the highest utility at each state as the game progresses. Each state of the game is represented by the positions of the two players as well as the current player with possession of the ball.

In this experiment, each agent's Q-values can be defined over states and action-vectors instead of a state and action pair [1]. Therefore, the Bellman equation is extended to describe multiple agents, each having respective Q tables mapping states to action-vectors:

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

where s is the index of eight possible positions for the agent, $\vec{a}$ consists of the actions available $\vec{a} = (a_1, a_2, ...a_n) =$ (N,S,E, W, and Stick), γ is the discount factor, $R_i(s,\vec{a})$ is agent i's reward for the state and all actions in the action set, P is the probability transition function, and $V_i(s')$ is the value function for state s' [1].

The Q-Learning algorithm, shown in Figure 1, is extended from a single-agent implementation in [3] to support the separate Q-tables for each agent and to track the Q-value difference error for state s and action a, where s is the reset state and action a is move south.

The following hyperparameters were used in the replicated Q-learning experiment: $\alpha = 1.0$, alpha_decay = 0.9999, alpha_min = 0.001, $\gamma = 0.9$ as in [6], $\epsilon = 1.0$, epsilon_min = 0.001, and epsilon_decay = 0.9999.

Algorithm 1 Multi-Agent Q-Learning Algorithm

```
Initalize all $Q1(S_1, A_1) \leftarrow 0$
Initalize all $Q2(S_2, A_2) \leftarrow 0$
for $i = 1 \rightarrow$ num_iterations do
    $s_1, s_2 \leftarrow$ starting positions (0,1) and (0,2)
    while not terminated or i < num_iterations do
        Choose $a_1, a_2$ using $\epsilon$-greedy policy
        Take actions $a_1, a_2$ using Step Algorithm
        Observe $\langle s_1', s_2', r_1, r_2 \rangle$
        $Q1(s_1, a_1) \leftarrow Q(s_1, a_1) + \alpha[r_1 + \gamma \max_a Q(s_1', A_1) - Q(s_1, a_1)]$
        $Q2(s_2, a_2) \leftarrow Q(s_2, a_2) + \alpha[r_2 + \gamma \max_a Q(s_2', A_2) - Q(s_2, a_2)]$
        $s_1 \leftarrow s_1'$
        $s_2 \leftarrow s_2'$
    end while
    $\epsilon \leftarrow \max(\text{epsilon\_min}, \epsilon \cdot \text{epsilon\_decay})$
    $\alpha \leftarrow \max(\text{alpha\_min}, \alpha \cdot \text{alpha\_decay})$
end for
```

1.2.2 Q-Learning Results

Figure 3 shows the replicated Q-learning results from Figure 3d in [1]. As was expected, the Q-Learning algorithm did not converge. This is the same result reported by Greenwald et. al in [1]. The main reason the Q-learning algoritm does not converge is because it searches for determinsitic optimal policies, but in this soccer game, none exist [1]. This result is significant because it shows the limits of the multi-agent Q-learning algorithm when applied to general-sum Markov games, and motivates investigation into other classes of algorithms toward this class of problems.

While the replicated Q-learning results also did not converge, there are some differences in the characteristic of the graph compared to the original in [1]. Specifically, in the original result, between 3.5 and 4.5 $\times 10^6$ iterations, there is a sharp decrease in the standard deviation of the Q-Value differences. This is not found in the replicated version of Figure 3. Also, in the original, there is a sharper decrease after 7×10^5 iterations than in the replicated version. Given these differences, the replicated version still shows that the Q-value differences do not converge

In the original version, the Q-Value differences decrease as the number of simulations increases proportionally to a decrease in α. This is also present in the replicated version, but to a lesser degree. One reason we may see less steep decrease is because we don't know the starting alpha used in the original experiment, we only know the minimum value allowable for α (0.001). Therefore, an assumption had to be made and the starting value for α was set to 1.0, although this may have been different in the original.

One purpose of experimenting with the multi-agent Q-learning algorithm in playing the soccer game is to provide a foundation for which to measure alterations or augmentations to it. The Friend-Q, Foe-Q, and CE-Q algorithms are all based on the Q-learning algorithm and have different mechanisms of action selection, q-table updates. They also may use independent or joint policy tables. On-policy q-learning is also typically successful in grid-games [1]. Therefore, the Q-learning algorithm is a useful baseline algorithm for solving the multi-agent soccer grid-game problem.

The largest difficulty or pitfall encountered while testing with the Q-learning algorithm was performing hyperparameter tuning, specifically finding the best decay rates of α and ϵ to better match the original. This information was not found in [1], therefore, finding the optimal decay rates was computationally expensive and involved some guesswork. Ultimately, 0.9999 was selected as the decay for both α and ϵ, but it is quite possible that other decay rates may have been used in the original.

1.3 Friend-Q Learning

1.3.1 Friend-Q Experiment

With the Friend-Q strategy, each player i assumes that the other players are allies, and are working together to maximize i's value [5].

The Friend-Q algorithm is essentially the same as Q-learning but accounts for the combined action space of the two players [5]. The Friend-Q value function is as follows:

$$V_i(s) = \max_{\vec{a} \in A(s)} Q_i(s,\vec{a})$$

where $\vec{a}$ denotes the combined action space of the two players in place of a single action, while the Q-learning value function only contains the action space of a single player:

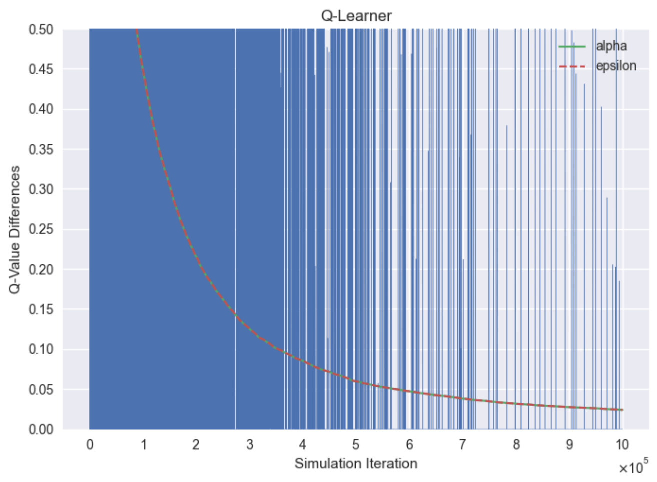


Figure 3: Q-Learning

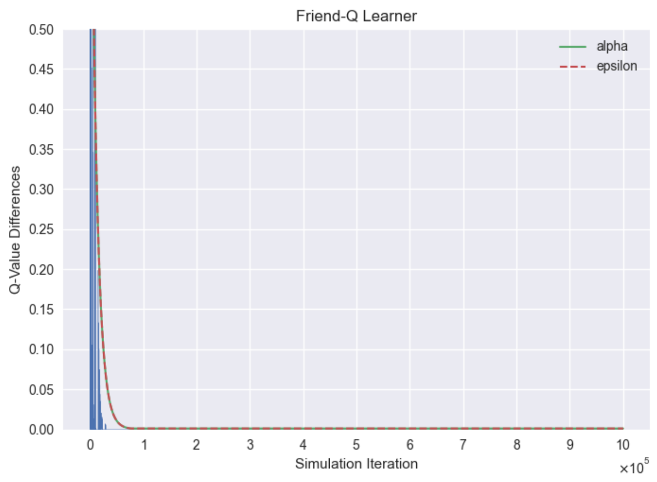


Figure 4: Friend-Q

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a)$$

[1].

As a result, the Q tables for both players in the Friend-Q algorithm are initialized to have an extra dimension representing the action space of the opposing player. The e-greedy action selection method is also extended to support the extra dimension, such that instead of selecting actions for both players with

$$\underset{i}{\operatorname{argmax}}\ Q(s, a_i)$$

we use the following instead:

$$\underset{j}{\operatorname{argmax}}\ \{\max_i Q(s, a_i, a_j)\}$$

The Q-Table update is also different from the original Q-Learning algorithm to account for this extra dimension in the opponent player's action space.

The following hyperparameters were used to replicate the original results from [1]: α=1.0, alpha_decay =0.99991, alpha_min=0.001, γ=0.9, ϵ=1.0, epsilon_min=0.001, and epsilon_decay=0.99991. Similar to the Q-learning experiment, some experimentation was required to settle on optimal decay rates as these were not identified in [1].

1.3.2 Friend-Q Results

Figure 4 shows the replicated Friend-Q result. Both the original and replicated graph show convergence after about 1.5 $\times 10^5$ iterations. In [1], Greenwald et. al reported that the Friend-Q algorithm converges to a deterministic policy that benefits only one player. In the replicated experiment, this was sometimes observable while rendering the gameplay, as one of the players might score without being challenged by the opponent. In this situation, is seemeed the player would naively trust his opponent to score for him. In other cases, the opponents would converge to irrational policies by moving up and down inside their opponent's goal area, delaying the progress of the game.

While the replicated Friend-Q results does show convergence about halfway through 1 $\times 10^5$ iterations, there is more variance in the data. In the original graph before convergence, there are no points where the q-value difference was 0. In the replicated version, there are more oscillations between large and near-zero q-value differences. One possible reason for this is that there may have been some post-processing of the data, such as averaging or thresholding of points outside the standard deviation. Since convergence occurs quickly, there is not a lot of data to examine in the original, and since no post-processing or graphing techniques are discussed in the original work, any such suggestions can only be speculative given the current information available.

Compared to the Q-learning algorithm, the Friend-Q algorithm does converge, but the convergence is to a policy that is irrational [1]. For example, it is irrational for one player to always trust an opponent that never acts in his best interests. Compared to the Foe-Q algorithm, the Friend-Q algorithm's guarantees are considerably weaker because the learner may not achieve its learned value even if the opponent is a friend. This is due to the possibility of incompatible coordination equilibria between opponents [5]. The significance of this result is that convergence doesn't necessarily prove mastery over the game, since a better solution to the game would provide better handling of incompatible coordination equilibria between opponents. The result also motivates investigation into the Foe-Q algorithm, where each player assumes the other is a true adversary or into the CE-Q algorithm, where agents share knowledge to reach a common goal.

1.4 Foe-Q

1.4.1 Foe-Q Experiment

In the Foe-Q algorithm, each player's goal is to minimize the maximum value of the opponent's reward. To implement Foe-Q, the minimax-Q algorithm from [6] served as a theoretical basis in concert with the following definition of the value function from Greenwald et. al in [1]:

$$V_1(s) = \max_{\sigma_1 \in \Sigma_1(s)} \min_{a_2 \in A_2(s)} Q_1(s, \sigma_1, a_2) = -V_2(s)$$

where, $\Sigma_i(s)$ is the probabilistic action space of player i at state s and $Q(s, \sigma_1, a_2) = \sum_{a_1 \in A1} \sigma_1(a_1) Q(s, a_1, a_2)$, and where σ_1 is the probability distribution over all actions of player 1, and a_1 and a_2 are actions in the action spaces A_1, A_2 of the two players [1].

To solve this problem efficiently, we can make use of linear programming techniques to determine the optimizing strategy (i.e. probabilities for each action) and value of the problem. Specifically, we can dynamically generate the coefficients of the constraints and declare a function to maximize, forming a linear program.

The following linear program is implemented to update the policy and value tables for both agents:
The function to maximize is z
with constraints

$$z + \sum_j Q(s, a_i, a_j) x_a \leq 0 \quad \forall\ i \in Q(s, a_i, a_j) \text{ and } \forall\ a \in A$$

$$x_a \geq 0 \quad \forall\ a \in A, \text{ and } \sum_{a \in A} x_a = 1$$

where A is the action space of the environment and separate constraints are written for each $i \in Q(s, a_i, a_j)$. Intuitively, a_i and a_j form the coefficients in the constraint equations mapping the q-values of all possible actions taken by both players for the current state, s.

1.4.2 Foe-Q Results

Figure 5 shows the replicated Foe-Q results from Figure 3b. in [1]. The replicated figure's characteristic matches with the original reasonably well when examining the overall trend of the peaks of the q-value differences as a function of iterations. However, the convergence in the replicated version occurred later than the original, and the initial errors in the first 3 $\times 10^5$ iterations were higher overall. As may have been the case with all graphs published in the greenwald paper, one theory is that this descrepancy may be due to undisclosed post-processing techniques such as averaging or thresholding to smooth the data.

Similar to the original, the replicated q-value differences have a characteristic of a slight uptrend between 4×10^5 and 6×10^5 iterations before it asymptotically approaches 0 from 0.05 after about 5×10^5 iterations.

One advantage of the Foe-Q algorithm over CE-Q is that there is no reliance on a central planner. The algorithm makes an assumption about its adversary's strategy and sticks with it, so it is more useful in a distributed setting. Extending the CE-Q algorithm to learn correlated equilibrium policies in Markov games would be a useful innovation, and while it is proposed as a future area of research in [1], the proposed CE-Q algorithm does not offer that capability.

The Friend-Q algorithm would not be as advantageous as the Foe-Q algorithm, even though it converges much faster, because it converges to an irrational policy. The Q-learning algorithm causes players to make no assumptions about the strategy of their opponents, forcing a determination about an opponent's strategy from observations alone. This requires that each state of an agent is fully observable to her opponents instead of hidden, and the learning of the opponents strategy requires additional training time. Assuming an opponent is taking a particular strategy at the beginning gives the Foe-Q learner a time advantage over the Q-learner. Therefore, by

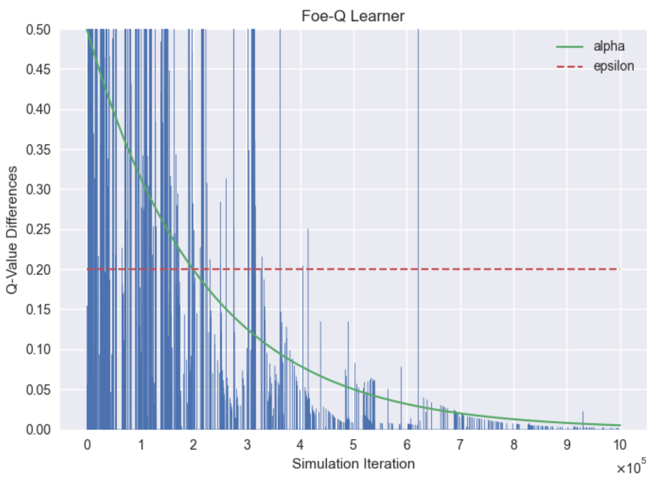


Figure 5: Foe-Q

assuming all players are adversaries, Foe-Q, is better suited to quickly reaching coordination equilibria than Q-learning.

The most difficult aspect of implementing the Foe-Q algorithm was creating a working linear program to update the policy and value tables for both players. To overcome this, the rock paper scissors problem served as a convenient foundation on which to develop a linear program that would extend to the Foe-Q algorithm. Fortunately, the number of constraints required for the Foe-Q algorithm was limited by the size of the action space. Compared to CE-Q, the required complexity of the linear program for Foe-Q is relatively small.

1.5 Correlated Equilibrium Q-Learning (CE-Q)

1.5.1 CE-Q Experiment

In the Correlated Equilibrium Q-Learning algorithm, the agents share their knowledge with one another and work together to achieve the highest common goal. In this particular experiment, the common goal is to maximize the sum of their rewards. Utilizing linear programming, CE-Q is a more easily computable approach to finding equilibrium policies in general-sum Markov games than finding Nash equilibria, and is intended to generalize both Nash and Friend-and-Foe-Q [1]. While a Nash equilibria tells you the independent probability distributions over actions for each agent, CE-Q permits information sharing through the use of joint probability distributions between the agents [1].

To augment the multi-agent Q-Learning algorithm and solve the constant sum markov game, Greenwald et al in [1] propose an alternative definition for the value function, $V_i(s)$:

$$V_i(s) \in \text{CE}_i(Q_1(s), ., Q_n(s)) = \text{CE}_i(\vec{Q}(s))$$

Here, CE is the correlated equilibrium that allows for a joint probability distribution between all of agent i's actions and each action of the other players for a given state [1]. Since our focus is to maximize the sum of each player's reward, we apply the *utilitarian* (uCE-Q) objective function from [1]:

$$\sigma \in \arg \max_{\sigma \in \text{CE}} \sum_{i \in I} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

Given

$$V_i(s) \in \mathrm{CE}_i(\vec{Q}(s))$$

$$\mathrm{CE}_i(\vec{Q}(s)) = \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

from [1], we can precisely define the value of state s in a uCE-Q algorithm that maximizes multi-agent rewards as follows:

$$V_i(s) \in \left\{ \sum_{\vec{a} \in A} \left\{ \arg \max_{\sigma \in \mathrm{CE}} \sum_{i \in I} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a}) \right\} Q_i(s, \vec{a}) \right\}$$

Intuitively, the innermost quantity finds the action probability distribution that maximizes rewards between the two agents, based on each σ and $Q_i(s, \vec{a})$ for the state and action-vector. Stepping one quantity out, if we multiply the argument maximizing rewards by $Q_i(s, \vec{a})$ we will then generate the set of potential values for $V_i(s)$. Since there are multiple ways of selecting the optimal value, it may be sufficient to say that $V_i(s)$ exists in this set, but a more specific definition that describes how we find the value that maximizes the total rewards for both agents requires applying the max operator over the set. Therefore, the max operator was applied to this entire quantity during the implementation.

The following linear program from [7] provided a theoretical foundation for the linear program implementation that determines the optimal strategies and values of $Q_1(s)$ and $Q_2(s)$:

$$\sum_{\bar{s} \in S_{-p}} (u^p_{i,\bar{s}} - u^p_{j,\bar{s}}) x_{i,\bar{s}} \geq 0 \quad \forall p \text{ and } \forall i, j \in S_p$$

$$x_s \geq 0 \; \forall s \in S$$

$$\sum_{s \in S} x_s = 1$$

where $x_{i,\bar{s}}$ is the probability that player p takes strategy i while everyone else plays $\bar{s}$, $u^p_{i,\bar{s}}$ is the payoff to player p for taking strategy $i \in S_p$ while everyone else plays $\bar{s}$, and S_p is the strategy set defining the strategy profile for player p [7]. Before the more general linear program was attempted, it was first useful to implement a linear program that solves the game of chicken because the constraint equations can be derived mathematically first before the coefficients are hard-coded into a table. Building the constraint equations for the soccer game was a more challenging task since the end result would produce approximately 65 total equations with 25 variables. The underlying cvxopt linear program solver from [8] did occasionally experience internal errors instead of safely reporting that no solution could be found in the final result when solving this large linear program. Therefore it was necessary to gracefully handle this error and skip updates to the joint policy and value tables when it occurred.

1.5.2 CE-Q Results

Figure 6 shows the replicated CE-Q results from Figure 3a in [1]. Like the original, the replicated CE-Q errors approach 0 with increasing simulation iterations in a similar manner as Foe-Q. The significance of this result is that CE-Q can learn minimax equilibrium policies in the soccer game environment without making assumptions about the strategies of opponents [1]. Thus, it is a more general solution to the two-player, zero-sum game of soccer.

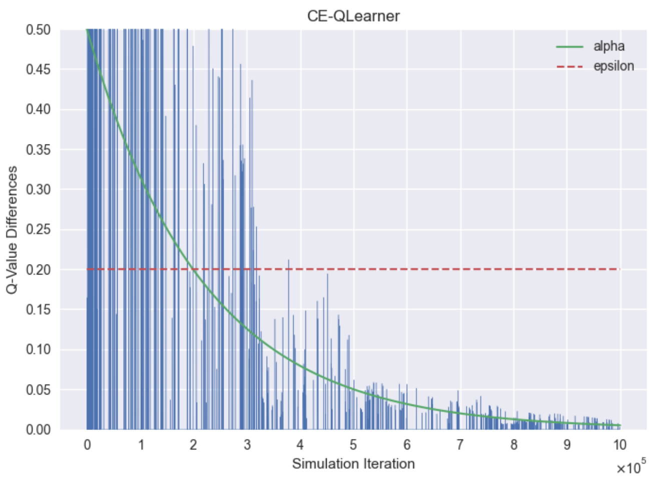


Figure 6: CE-Q

With the goal in [1] being to improve the design of multiagent systems, there is some debate about whether to have a central planner dictating agent behavior or to allow agents to specify their own behavior. The advantage of CE-Q is that it is not as susceptible to miscordination as the other algorithms are and that it pre-specifies rational agent behavior [1]. On the other hand, one difficulty of CE-Q is that the underlying linear program call could be simplified with an adaptive algorithm instead. Also, in some general sum markov games, the sharing of Q-tables between agents might not be allowed and in that case algorithms such as Foe-Q and Friend-Q that presuppose the strategy of the opponent may perform better.

References

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

[2] Sutton, Richard S. "Learning to predict by the methods of temporal differences." Machine learning 3.1 (1988): 9-44.

[3] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[4] Mitchell, Tom M. "Machine learning." (1997).

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

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

[7] https://web.stanford.edu/~saberi/lecture6.pdf

[8] https://cvxopt.org