Project #3 Correlated-Q

Hui Xia (Hxia40)

Georgia Institute of Technology

Git hash: c7756d6295b9c8f1c9684ecb717403b5d94f5266

Abstract— **In this project, we aim to create an implementation to reproduce Figure 3 (parts a-d) from a reinforcement learning paper "Correlated-Q learning". Here we describe our implementations, then compare the resulted figures with the original paper, and analyze the difference between them.**

I. Introduction

Reinforcement learning is a major category in machine learning. In 1988, Richard Sutton described a reinforcement learning method, which is temporal difference (TD) [1]. The TD method is now well-received by the computer science community for solving multi-step Markov Decision Process (MDP), which involves a decision-making process of to taking various actions among various states. The TD method has been developed into on-policy TD methods such as State-Action-Reward-State-Action (SARSA), and off-policy TD methods such as Q-learning [2].

In practice, the reinforcement learning agent in MDP usually needs to interact with other agents, in either a cooperative or competitive manner, or both. Various attempts are made to design learning algorithm that can be implemented on multiagent MDPs. Adopting the game theory [3], reinforcement algorithms have been explicitly designed for investigating multiagent MDPs, which is thereby referred as Markov games. Hu and Wellman [4] proposed Nash-Q algorithm, which try to find Nash equilibrium policies of Markov games. This method is limited in general-sum games, as in such case Nash-Q typically does not converge. Littman [5] proposed friend-Q and foe-Q algorithms, which always converges when the relationship between two agents can be decided. Greenwald and Hall [6] proposed Correlated-Q (CE-Q), which converges to correlated equilibrium policies on a more general form of Markov games. In this project we will firstly implement a 'soccer game' environment, then reproduce Figure 3 of Greenwald and Hall's, work, which plots the error between each episode of the reinforcement environment of soccer game using four algorithms: on-policy Q-learning (i.e. SARSA), friend-Q, foe-Q, and Correlated-Q (CE-Q). We will compare our reproduction work with the original figure, then analyze on the difference between them.

II. Method

A. The Soccer Game Environment

The Soccer Game is a zero-sum grid game which has multiple versions. In Greenwald and Hall's work, this environment is based on a 2 x 4 grid:

0	$1(\boldsymbol{B})$	$2(A)$	3
4	5	6	7

In the grid world above, two players, A and B, are playing a soccer game. The player that is boldened (e.g. player B in the above-shown state. Hereafter the above-shown state will be referred as state s) is with the ball. For each episode of experiment, points 1 and 2 are player B and player A's starting point, respectively, and player B always to start with the ball. Points 0 and 4 are player A's goal, while Points 3 and 7 are player B's goal. When either player that is with the ball move to a goal, the owner of the goal will score + 100, and the opponent player - 100. In other words, own goal is valid. Other points in the grid world (i.e. points 5 and 6) are flat ground that all players can move through.

Both players could perform five actions: N (grid-value -4, same below), S (+4), E (+1), W (-1), and stick (+0). Greenwald and Hall did not mention what will happen when a player moves out of the grid world. Here we assume that when such an action happens (e.g. in state s, player B choose the action N), the player will not move, and such action will be considered as performing the action "stick". For each round of the game, both player will choose one action, and the players' actions are executed in random sequence. If this sequence of action causes the players to collide, then neither of the player moves. But if the player with the ball moves second, then the ball changes possession, as described by Greenwald and Hall.

We implemented the soccer environment using OpenAI Gym, which is a powerful reinforcement environment assist, but typically does not support multiagent Markov game. To accommodate, we update the actions of both players as two-value tuples, with the first value represent player A's action, and the second value represent player B's action. The environment state is also updated in tuples, with the first value represent the right of ball (0 means player A is with the ball, and 1 means player B is with the ball), and the second and third value represent the positions of player A and player B, respectively.

B. Solver Algorithms

TD algorithm trains an agent to choose and execute the optimal action based on the state of an MDP. For single-agent MDP, "optimal action" is deterministic and can be calculated using Bellman's equation, and the respective value of executing

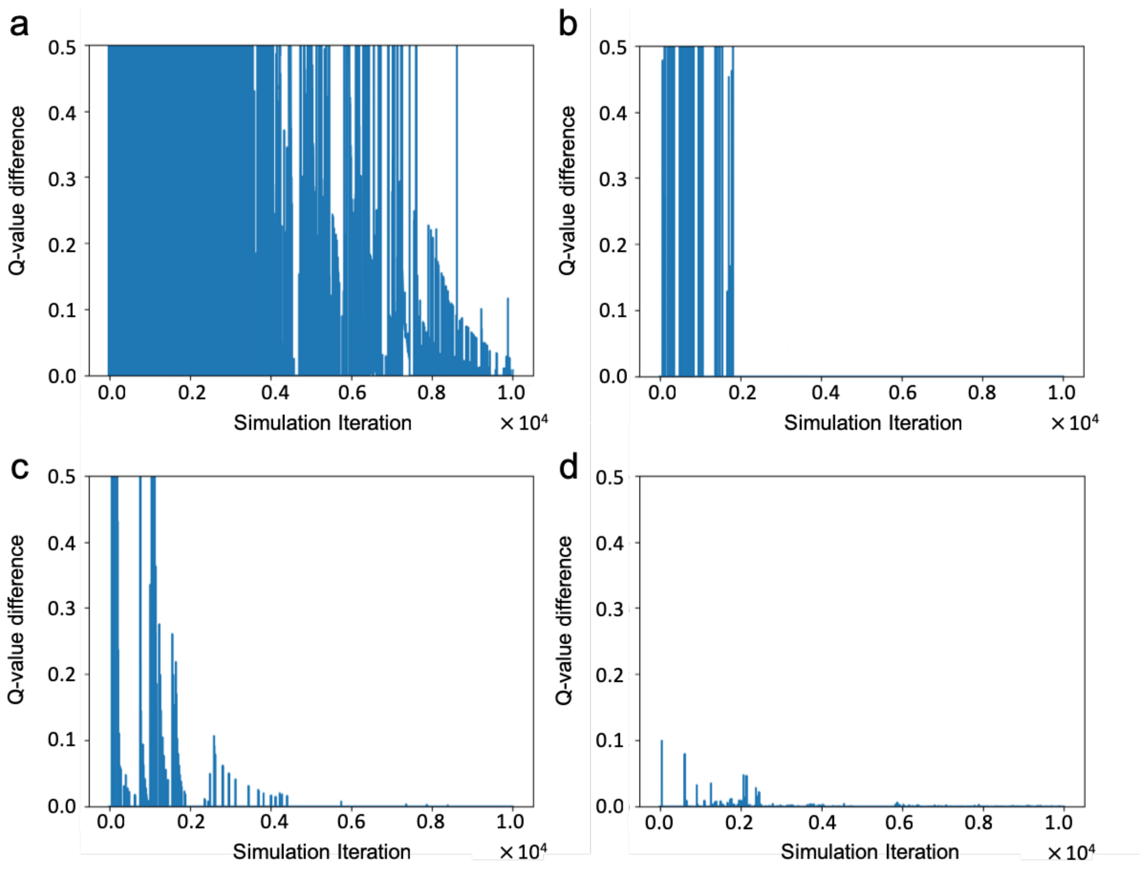

Figure 1. Q-value difference for each simulation iteration. (a) Using SARSA, represent player A's Q-values, corresponding to state *s* and action S. (b-d) Using friend-Q, foe-Q, and CE-Q, respectively, represent player A's Q-values, corresponding to state *s*, when player A takes action S and player B sticking. Initial alpha = 0.5, minimum alpha = 0.001, initial epsilon = 0.5, minimum epsilon = 0, gamma = 0.9, alpha decay = 0.9995, epsilon decay = 0.9995, random seed = 1.

an action under a given state is referred as Q-value. For each step in the MDP, the Q-value is updated using the equation below:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P[s'|s,a] V^*(s') \qquad (1)$$

In the equation above, $Q^*(s,a)$ stands for the update of the Q-value prediction in a given step of the episode, given the state and action of (s,a), and $r(s,a)$ is the immediate reward at that step. Gamma (γ) is the discount factor, which controls how much weight it gives to future rewards in the algorithm. Considering that in the soccer game, only making goals will grant the players reward of +100 points, we would use a high value of $\gamma = 0.9$ for the experiments performed in this project. The value function $V^*(s)$ is defined as the value that maximizes $Q^*(s,a)$ overall actions, and the collection of such actions that maximizes $Q^*(s,a)$ is the optimal policy:

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a) \qquad (2)$$

MDP could be considered as single-agent Markov games. In case of Markov games, compared with the deterministic Q-value update equation shown above, in Markov games, player i's optimal Q-values are determined by probabilistic action vectors:

$$Q^*(s,\vec{a}) = R_i(s,\vec{a}) + \gamma \sum_{s'} P[s'|s,\vec{a}] V_i^*(s') \qquad (3)$$

The value function $V_i^*(s')$ is also become probabilistic and is used to describe friend-Q and foe-Q, respectively using eqs. (4) and (5) below:

$$V_i^*(s) = \max_{\vec{a} \in A(s)} Q_i^*(s,a) \qquad (4)$$

$$V_1^*(s) = \max_{\overline{a_1} \in A_1(s)} \min_{\overline{a_2} \in A_2(s)} Q_1^*(s,\sigma_1,a_2) = -V_2^*(s) \qquad (5)$$

In friend-Q, an agent will naively assume that its opponent will always be friendly and choose the action that will maximize the reward of the former agent itself, and *vice versa*. For example, in the state *s* of the soccer game shown above, player B's policy should converge to always move right, trying to pass the soccer ball to player A, and expecting that player A will then score an own goal for player B. On the opposite, in foe-Q, an agent will assume its opponent will try to minimize the former agent's reward and will accordingly try to maximize its reward based on above assumption. For zero-sum two-player Markov games, this max-min algorithm taken by foe-Q agents can converge to Nash equilibrium. However, for more generalized n-player general-sum games, no convergence can be found for Nash equilibrium. To solve such problem, Greenwald and Hall described CE-Q, which find empirical convergence to correlated equilibrium, rather than Nash equilibrium:

$$V^*(s) \in CE_i(Q_1^*(s), \ldots, Q_n^*(s)) \qquad (6)$$

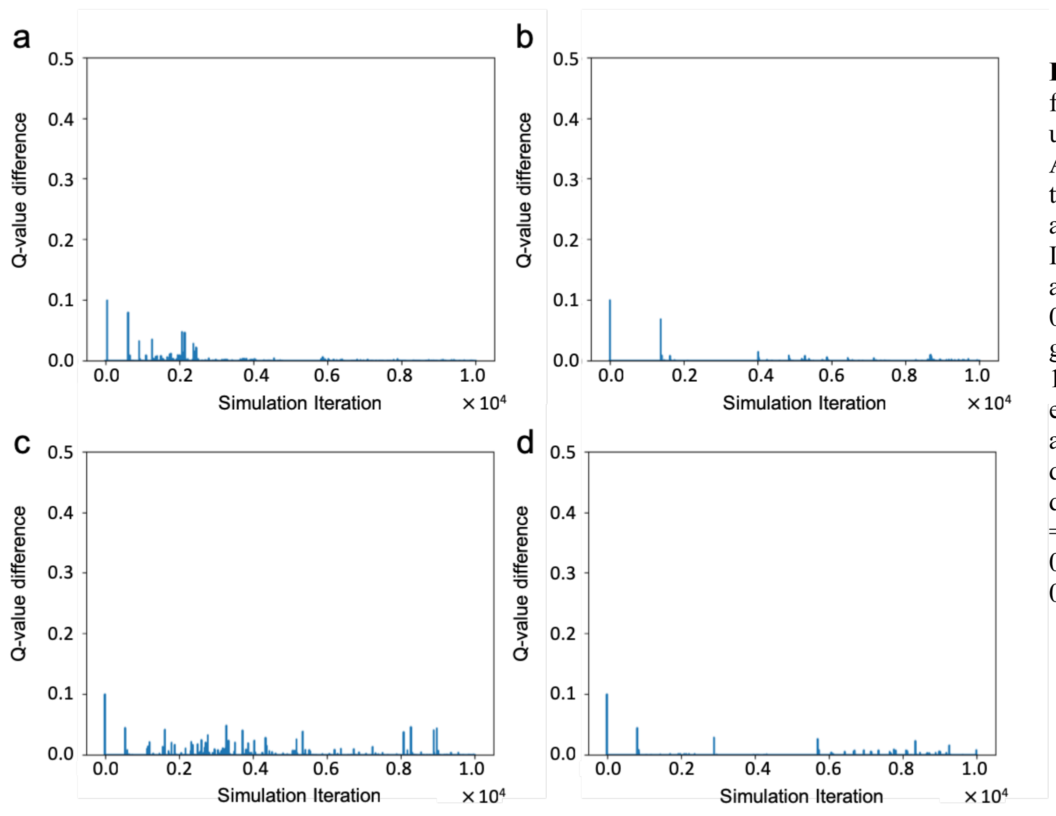

Figure 2. Q-value difference for each simulation iteration using CE-Q, represent player A's Q-values, corresponding to state *s*, when player A takes action S and player B sticking. Initial alpha = 0.5, minimum alpha = 0.001, initial epsilon = 0.5, minimum epsilon = 0, gamma = 0.9, random seed = 1. (a) alpha decay = 0.9995, epsilon decay = 0.9995. (b) alpha decay = 0.9995, epsilon decay = 0.9999. (c) alpha decay = 0.9999, epsilon decay = 0.9995. (d) alpha decay = 0.9999, epsilon decay = 0.9999.

In this project, we implemented all algorithms by updating Q tables and V tables for each of the soccer players. Note that although the soccer game is zero-sum, and thus the value (V) table for player B should just equal to the negative of player A's, as shown in eq. (5), we still maintained V-tables for both players in the foe-Q algorithm, for better comparation with other algorithms. The max-min and correlated equilibria for foe-Q and CE-Q learning are calculated by CVXOPT linear programming inspired by Novotny's method [7]. Due to the limitation in computational power, we run 10 thousand episodes per experiment, rather than 1 (or 10, the original figure is too blurry to distinguish) million episodes per experiment in Greenwald and Hall's original paper. Also, to decrease the run time, we implemented a timeout-cap of 1000, which force the game to terminate if it does not finish in 1000 steps.

III. Results and Discussion

Figure 1 demonstrates the Q-value difference for on-policy Q-learning (SARSA), friend-Q, foe-Q, and CE-Q on state *s*, in endeavor of reproducing Figure 3 from Greenwald and Hall (G & H). One major difference between G & H and our reproduction is that all four figures from G & H are plotted using a smooth continuous line. This feature is more pronounced in G & H's friend-Q plot, as well as at higher iterations in G & H's foe-Q and CE-Q plot, when these algorithms are nearly converged. On the other hand, we can see significant variations between episodes in our reproduced plots for all four algorithms. This observation is counter-intuitive, as we just discussed in the **Methods** section, due to run time difficulty, we can only perform ten thousand episodes per experiment, which is much less than the episodes performed by G & H. However, our plots seem denser than theirs. We reason that it is rather unbelievable that plotting error difference between each episode can be a continuous line, unless G & H were using a very fast decaying epsilon. Alternatively, G & H did not plot their graphs using all data points, but rather, for example, every i'th point. Without detailed description on G & H's methods, we will put our focus on reproducing the shape and trend of our error plot, rather than data density.

As shown in **Figure 1(a)**, our SARSA graph generally resembles the shape and trend of G & H's. Both our and the original graph demonstrate significant variations between updates. This observation is due to that a Q-learning agent seeks for deterministic optimal policy, which does not exist in two-agent Markov games. Since a Q-learning agent choose its actions without factoring its opponent's movements, in the Soccer Game environment, this is equal to learning from a noisy dynamic environment. Thus, it is rather intuitive that no convergence can be found. After the experiment is finished, both players' Q-tables show that neither of the players have converged to any deterministic policy.

As shown in **Figure 1(b)**, our Friend-Q converges in a fast manner. Although, proportionally speaking, it takes more episodes for our plot to converge, compared with G & H's. This could due to the difference of initial value and the decaying factor of hyperparameters such as alpha and/or epsilon. Afterall, G & H did not specify all the hyperparameters except that their minimum alpha is 0.001. A friend-Q agent assumes its opponent will always try to maximize the reward received by the friend-Q agent itself. Thus, running the Soccer Game

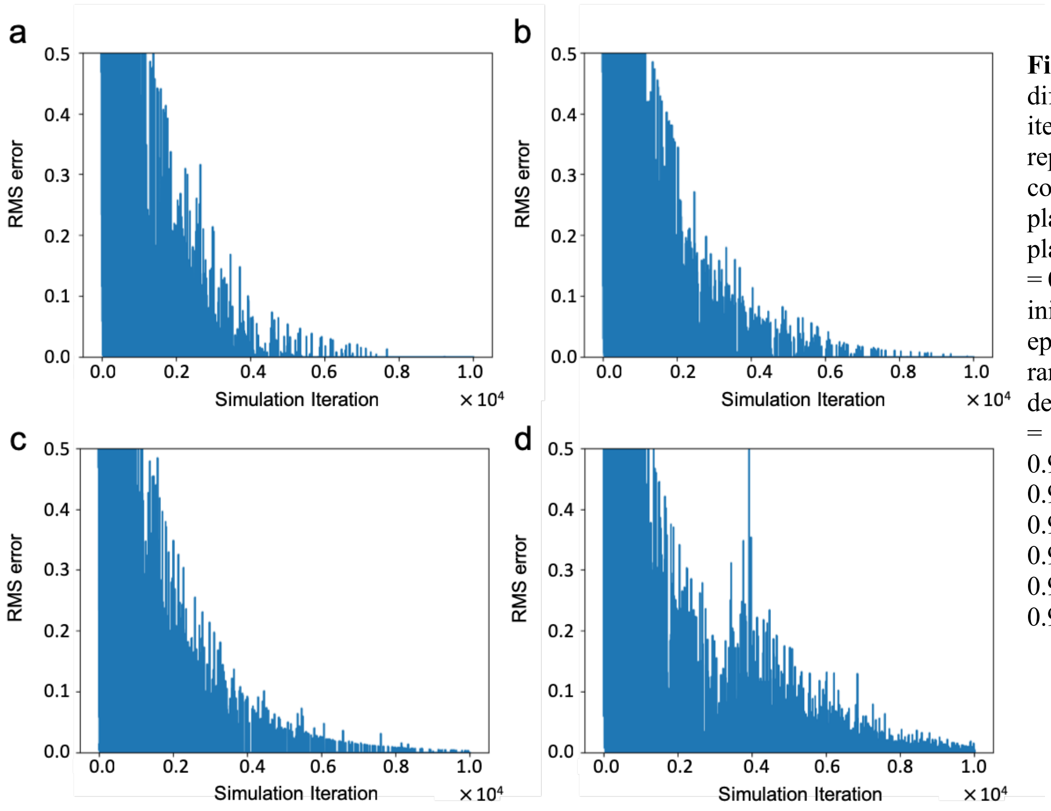

Figure 3. Overall RMS difference for each simulation iteration using CE-Q, represent player A's Q-values, corresponding to state *s*, when player A takes action S and player B sticking. Initial alpha = 0.5, minimum alpha = 0.001, initial epsilon = 0.5, minimum epsilon = 0, gamma = 0.9, random seed = 1. (a) alpha decay = 0.9995, epsilon decay = 0.9995. (b) alpha decay = 0.9995, epsilon decay = 0.9999. (c) alpha decay = 0.9999, epsilon decay = 0.9995. (d) alpha decay = 0.9999, epsilon decay = 0.9999.

with two friend-Q agents will result in a deterministic, but rather naïve, strategy for both players. After the algorithm converged, both players' Q-tables show that at state *s*, player A's policy converged to randomly choosing among all five actions, as it anticipates that player B will directly perform an own goal for player A itself. On the other hand, player B will always choose action E, trying to pass the ball to player A so that player A can then perform an own goal for player B. This observation in in agreement with G & H's paper.

As shown in **Figure 1(c)**, using the foe-Q learner, our Q-value difference decreases over increasing simulation iteration, and is no longer detectable below six thousand iterations, indicating convergence of the algorithm. This observation is in agreement with Littman's original friend-Q and foe-Q paper that in a dual-agent Markov game, if both agents are using minimax algorithm, then both agent's policy should converge. Similarly, as shown in **Figure 1(d)**, using the CE-Q learner, our Q-value difference decreases over increasing simulation iteration, and is no longer detectable below six thousand iterations, indicating convergence of the algorithm. This observation is in agreement with G & H's paper, that in a dual-agent Markov game, if both agents are using CE algorithm, then both agent's policy should converge.

Although the shape of our foe-Q plot resembles G & H's plot, we noticed that our CE-Q plot demonstrates a much lower Q-value difference compared with its counterpart in G & H's paper. Also, after all simulation iterations are finished, both players' Q-tables show that although both our foe-Q and CE-Q converge to nondeterministic policies for both players, each player randomizes between sticking, heading south, and heading toward each other. This observation does not with G & H's paper, which suggests that after foe-Q and CE-Q converge, both players should converge to a randomized policy between sticking and heading south. Furthermore, in Q-tables we obtained using SARSA and friend-Q, we can see that all states have been updated. However, in the Q-tables generated using foe-Q and CE-Q, the Q-value in many states have not been updated and remained as 1, the initialized value in the Q-table. These observations indicate that our for-Q and CE-Q may have been converged to local 'optimum' due to lack of exploration and could contribute to the reason why the Q-value difference in our CE-Q plot is much lower compared with in the G & H paper.

In endeavor of to improving the exploration of our CE-Q solver, and to increase the Q-difference in our CE-Q plot to better match its G & H counterpart, we investigate on how alpha and epsilon decay rates will affect the Q-difference in our CE-Q plot. We performed experiment using CE-Q over a series of alpha and epsilon decay rates: 0.99999, 0.9999, 0.9995, and 0.999. Some of the representative combinations are shown in **Figure 2**. None of the test combinations resulted in an increase in Q-value difference. On the other hand, using higher alpha decay rate and epsilon decay rate will result in the CE-Q to converge slower.

To investigate whether our CE-Q has actually converged, we calculated the root mean squared error (RMSE) over the change of all Q values between each simulation over a combinations of a series of alpha and epsilon decay rates. **Figure 3(a)** demonstrates the RMSE of CE-Q using the exact same condition that have been plotted in **Figure 1(d)**. The

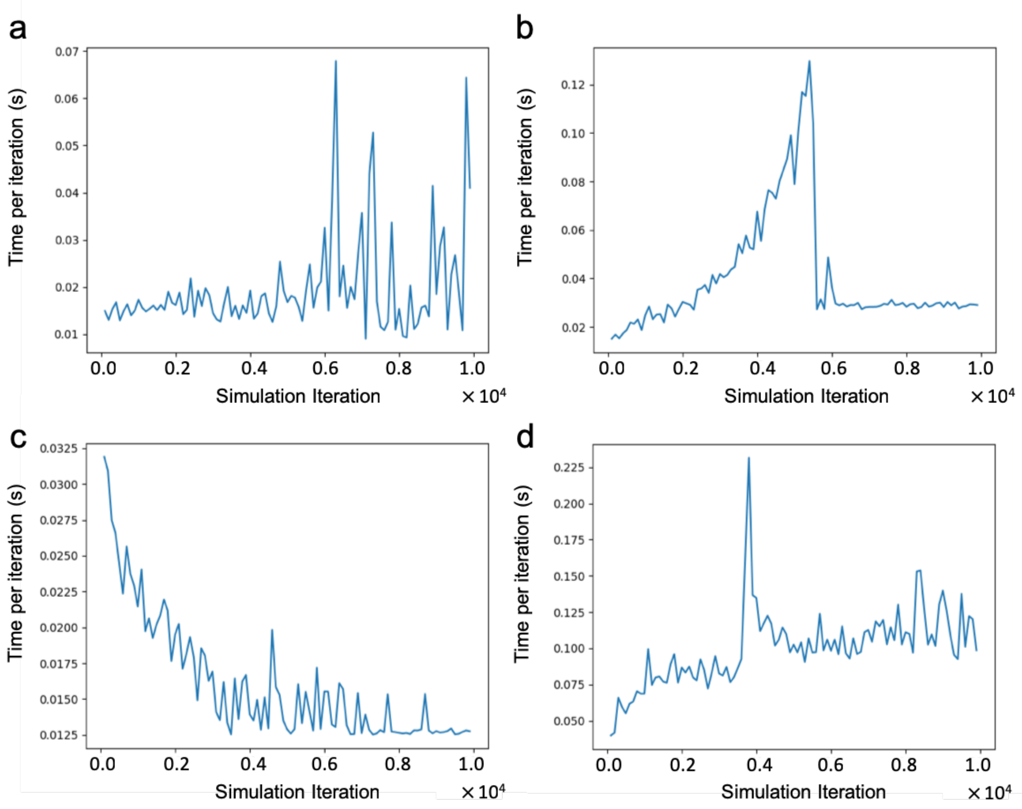

Figure 4. Average time consumption per 100 simulation iterations. (a) Using SARSA, represent player A's Q-values, corresponding to state *s* and action S. (b-d) Using friend-Q, foe-Q, and CE-Q, respectively, represent player A's Q-values, corresponding to state *s*, when player A takes action S and player B sticking. Initial alpha = 0.5, minimum alpha = 0.001, initial epsilon = 0.5, minimum epsilon = 0, gamma = 0.9, alpha decay = 0.9995, epsilon decay = 0.9995, random seed = 1.

RMSE did converge at about 8000 iterations. The rest of **Figure 3** also demonstrates that using higher alpha decay rate and/or epsilon decay rate will require more iterations to reach convergence for CE-Q.

Another side proof on the convergence state of all the four algorithms is shown is **Figure 4**, which plots the average time consumption per 100 simulation iterations. When an algorithm has converged, the time consumption will be kept in a rather narrow range, indicating that the policy for neither of the players are making major changes. **Figure 4** suggests that friend-Q, foe-Q, and CE-Q has converged after 6000, 4000, and 4000 iterations, respectively, while SARSA did not converge until the end of the experiment. Note that after the algorithm converged, the time consumption of foe-Q and CE-Q still oscillate in a larger range compared with friend-Q, which can be explained by that foe-Q and CE-Q converge to nondeterministic policies, while friend-Q will converge to a deterministic policy. Based on the above-demonstrated evidences, we justified our reasoning earlier, that our CE-Q solver are trapped in local 'optimum', which is presumably due to lack of exploration. Thus, to fully reproduce G & H's plot on CE-Q, we will need to further optimize our code, so that running more episode iterations per experiment will be possible, allowing the code to perform more through exploration and fully converge.

Conclusions

Overall speaking, in this project, we implemented a soccer game Markov game environment, and then implemented four solvers: SARSA, Friend-Q, Foe-Q, and CE-Q to solve this environment. We have closely resembled the SARSA, friend-Q, and foe-Q plot from the original paper. While our CE-Q plot need further improvement to closely resemble the original plot, via this process, we gained understanding on Nash equilibrium, correlated equilibrium, and multi agent Markov games.

References

[1] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning,* vol. 3, no. 1, pp. 9-44, 1988.

[2] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning,* vol. 8, no. 3-4, pp. 279-292, 1992.

[3] O. Morgenstern and J. Von Neumann, *Theory of games and economic behavior*. Princeton university press, 1953.

[4] J. Hu and M. P. Wellman, "Nash Q-learning for general-sum stochastic games," *Journal of machine learning research,* vol. 4, no. Nov, pp. 1039-1069, 2003.

[5] M. L. Littman, "Friend-or-foe Q-learning in general-sum games," in *ICML*, 2001, vol. 1, pp. 322-328.

[6] A. Greenwald, K. Hall, and R. Serrano, "Correlated Q-learning," in *ICML*, 2003, vol. 20, no. 1, p. 242.

[7] A. Novotny. (2017). *Linear programming in Python: CVXOPT and game theory*. Available: https://medium.com/@excitedAtom/linear-programming-in-python-cvxopt-and-game-theory-8626a143d428