CS7642 Project 3 Report – Correlated-Q

Minghan Xu (minghan.xu@gatech.edu)
c9e530f9c810ba7ad5780ecaf14c7cc3a49da5fc (Git hash)

Abstract— **This project report discusses the theoretical foundation of the game theory and the application of various multiagent Q-learning techniques including Q-learning, Friend-and-Foe, and Correlated-Q learning in solving the Robocup environment.**

I. INTRODUCTION

In homework 6, we have practiced to solve for the Nash equilibrium using linear programming (LP). The goal is to derive a set of strategy (pure or mixed) for the players to reach the equilibrium state. The equilibrium conditions are translated into an objective function and a set of LP constraints for the LP solver to generate the probability distribution of the strategy. In project 3, we extend the techniques of finding the equilibria to the environment of Robocup Soccer. Four different alogirthms: Q-learning, Friend-Q, Foe-Q and Correlated-Q are implemented to estimate the Q-values and value functions V. Figure 3 of Greenwald's paper is reproduced and analyzed.

II. GAME THEORY AND REINFORCEMENT LEARNING

A. Nash Equilibrium

Nash equilibrium is a set of strategy for a non-cooperative game involving multiple players in which none of the player are better-off by switching to any other strategy, given each player knowing the equilibrium strategy of the other players. The mathematical definition as: Given $S = S_1 \times S_2, \times ... S_n$ is set of strategy profiles for n players and $f(x) = (f_1(x), f_2(x), ...f_n(x))$ is the utility function, strategy profile $x^* \in S$ is a Nash equilibrium if and only if:

$$\forall i, x_i \in S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*)$$

where x_i is the strategy profile of player i and x_{-i} be the strategy profile for other players.

Discussed in the lectures, for a finite game, there is always at least one Nash equilibrium. In some cases, like the game of chicken, multiple Nash equilibria exist. Figure 1 demonstrates two possible pure-strategy Nash equilibria exist for this general-sum game.

For the strategy of (Row, Column) players, (Chicken, Dare) and (Dare, Chicken) are in Nash equilibrium as switching to other strategy for each player only yields a lower utility. These two pure strategies give utilities (2,7) or (7,2) respectively. If such game is played for multiple rounds, a mixed Nash equilibrium can be established by each player Dare $\frac{1}{3}$ of the time and Chicken $\frac{2}{3}$ for the remaining time. Such mixed strategy can improve the average return to $4\frac{2}{3}$.

	Dare	Chicken
Dare	0,0	7,2
Chicken	2,7	6,6

Fig. 1. Game of Chicken

B. Correlated Equilibrium

From the lecture, it is proposed that when a third party randomly draws a card from three combinations: (Chicken, Chicken), (Dare, Chicken), and (Chicken, Dare) and the two players strictly follow the actions shown on the card, the average return can further improve to 5. At the high-level, this third party brings in some coordination between two players and eliminates the combination of (Dare, Dare) which generates the worst-case utility. Hence the average return is improved. Similar example like traffic lights have been shared in Greenwald and Littman's paper. Correlated equilibrium can be perceived as a generalization of Nash equilibrium. It permits inter-dependencies among agent's probability distribution.

One additional property that makes correlated equilibrium stand out is that Linear Programming (LP) is efficient in solving for the correlated equilibrium. Unlike Nash equilibrium where no efficient method of computation is known, correlated equilibrium can be formulated as a set of constraints and an objective function for an LP solver to solve.

C. MDP to Markov Games

Greenwald's paper states that stochastic games generalize repeated games (games being played for multiple rounds) and Markov Decision Process (MDP). A stochastic game is defined as a tuple $\langle I, S, (A_i(s))_{s\in S, 1\leq i \leq n}, P, (R_i)_{1\leq i\leq n}\rangle$, where I as set of n players, S as set of states, $(A_i(s))_{s\in S, 1\leq i\leq n}$ as player's set of action at state s, P as the state transition function conditioned on joint actions and past states, and $R_i(s,\ \vec{a})$ as i-th player's reward with the joint action $\vec{a} \in A(s) = A_1(s) \times ... \times A_n(s)$. The Q-value function is defined as:

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

MDP is a special-case single-player Markov game. Hence the concept of state-value function can be extended from

MDP to Markov games. To define the value function, Greenwald proposes:

$$V_i(s) \in CE_i(Q_1(s), ...Q_n(s))$$

where the CE_i correspond to the reward of player i at correlated equilibrium in the general-sum game. This formulation also generalizes the condition of Nash equilibrium $V_i(s) \in Nash_i(Q_1(s), ...Q_n(s))$, where the Nash equilibrium is a special case of CE and each individual player's action space is fully independent.

III. Robocup Environment

The Robocup soccer game is a two-player, finite-space zero-sum game. On the grid of 4×2 cells, there are two players A & B and a ball. The objective of this game is for the player to carry the ball and reach to the designated goal cells and receive +100 reward. Carrying the ball to opponent's goal cells or allowing the opponent to score get the reward of -100. There are 5 valid actions for each step movement: N,S,E,W or stick. For each movement step, the sequence of action execution is at random. When a certain joint action pair results in a collision (two players occupy the same cell), the ball possession might change from the second mover to the first mover. The game ends when there is a goal from either player A or B.

Figure 2 demonstrates the state s defined in Greenwald's paper. In the implementation, we also assumes that after an previous game ends, a new game will be re-initialized to state s. The top-left cell is indexed as (0,0) and the bottom-right cell is indexed as (1,3). Player A and B are indexed as 0 and 1 respectively. Five legal actions are represented as the 2-dimensional movement on the grid: [-1,0] for N, [0,1] for E, [1,0] for S, [0,-1] for west and [0,0] for stick. When player A carries the ball and moves to cell (0,0) or (1,0), A scores +100, B scores -100; If A carries the ball but moves to cell (0,3) or (1,3), A scores -100, B scores +100. As such game is symmetrical, the same rules applies to B.

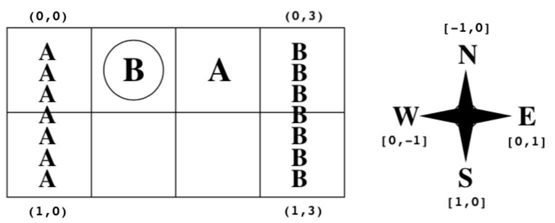

Fig. 2. Soccer Game with state s

The algorithm 1 illustrates the high-level implementation of the Soccer environment. The environment takes in current positions of both players, ball possession and new action selections from player A and B. It first checks whether the input actions are in valid range, then randomly decides either A or B to take the first move. The first mover performs the action and the collision checking mechanism decides whether there is a change of ball. It then checks whether the player has scored for himself or the opponent. After the first mover finishes his step, if there is no goal, the second mover performs the action and goes through the same process. The step update finishes and returns to the caller for the updated states of two players, scores and game termination flag.

Algorithm 1: Soccer Environment

```
Input: current positions, ball possession and new
       actions from player A and B
Output: [$state\_A$, $state\_B$, $ball\_possession$], $scores$,
        $done$
1  Init scores
2  Check input actions and randomly decide sequence of
     movement of A & B
3  First mover moves
4  if collision then
5      check ball status and exchange possession
6  else
7      update positions of first mover
8      if first mover scores for himself or opponent then
9          update scores, done=1, return
10 end
11 Second mover moves
12 if collision then
13     check ball status and exchange possession, return
14 else
15     update positions of second mover
16     if second mover scores for himself or opponent then
17         update scores, done=1, return
18 end
19 return [$state\_A$, $state\_B$, $ball\_possession$], $scores$,
     $done$
```

IV. Multiagent Q-learning

Greenwald's paper shares a general template for multi-agent Q-learning. The concept of multiagent is there are multiple agents to update the value functions and Q-values concurrently for multiple players of Markov Games. Much like the single-agent Q-learning architecture implemented in the project 2, multiagent learners have preset parameters including discount factor: γ, learning rate: α, decay factors: α-decay & ϵ-decay, and total training steps: T. Based on the flavor of the learning algorithm (Q-learning, Friend-or-Foe and Correlated-Q), for each step, the learners takes in joint states, actions and rewards to perform updates on Q-values and applies the magic function f to select the value functions V.

The following algorithm generalizes the architecture of the multiagent learner adopted for the Soccer game. The agent initializes the hyperparameters, Q-values, Value V and Policy π. For 1 million steps specified in Greenwald's paper, the agent generates actions using ϵ-greedy, passes the action to the Soccer environment discussed in the previous chapter and receives the next states, rewards and termination flag $done$. Next the agent performs Q-value update using:

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

Based on the flavor of the Q-learning algorithm. the agent may require to use the magic function f and formulate an LP to solves for value function V and policy π. From Greenwald, ERR_i^t is defined as the difference in Q-value for state s: $ERR_i^t = |Q_i^t(s,\vec{a}) - Q_i^{t-1}(s,\vec{a})|$, where $\vec{a}$ stands for the joint action player A going S and B sticking. For each step of Q-learning update, such error is captured and later plotted with the completion of 1 million training steps.

Algorithm 2: Multiagent Q-Learning

```
   Input: $f$, $\gamma$, $\alpha$, $\alpha$-decay, $\epsilon$-decay, $T$
   Output: $Q_A, Q_B, V_A, V_B, \pi_A, \pi_B$, $error\_list$
1  Init hyperparameters, i = 0
2  Init $Q_A, Q_B, V_A, V_B, \pi_A, \pi_B$, $error\_list$
3  while $i < T$ do
4      while True do
5          Map 2-dimensional positions of players and ball
             possession into states
6          Generate new actions for two players using
             $\epsilon$-greedy
7          i += 1, decay $\epsilon$
8          Perform generated actions and get new states,
             rewards and the termination flag done from
             env
9          if done then
10             calculate ERR, break, and start new game
11         else
12             Q-learning update
13             use magic function $f$ and LP to solve for $V$
                 and $\pi$
14             decay $\alpha$
15         end
16     end
17 end
```

A. Q-learning

The implementation of Q-learning follows the standard structure of the off-policy Q-table update. Two Q-tables are constructed independently for player A and B, each with a dimension of (8,8,2,5). The first two dimensions represent the 8 possible positions for player A and B. The third dimension is the indicator of ball possession, and the last dimension is the 5 legal actions for the player. As standard Q-learning does not take opponent's action into the construct of Q tables, The update formula of Q-value only involves the action of player himself: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$.

B. Friend-Q

In Littman's paper on Friend-or-Foe, from Theorem 4, with the condition B+: There exists a coordination equilibrium for the entire game and for every game defined by the Q-functions during the learning process, we can derive the Nash equilibrium as:

$$Nash_i(s, Q_1, Q_2) = \max_{a_1 \in A_1, a_2 \in A_2} Q_1[s, a_1, a_2]$$

This formulation shows, when we expect the opponent to be a friend (coordination equilibrium), the function value comes from the joint actions that maximize the Q-value. This is a natural extension of Q-learning by adding an additional dimension of opponent's action to the Q-table. In this implementation, the Q-table has dimension of (8,8,2,5,5) where the last two dimensions represent the action space of the opponent and the player.

C. Foe-Q

Similar to Friend-Q, from Theorem 4, with the condition A+: There exists an adversarial equilibrium for the entire game and for every game defined by the Q-functions during the learning process, we can derive the Nash equilibrium as:

$$Nash_i(s, Q_1, Q_2) = \max_{\pi \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} \pi(a_1) Q_1[s, a_1, a_2]$$

This formulation shows, when we expect the opponent to be fully adversarial, the function value comes from the maximization efforts given the worst-case situation posted by the opponent. It follows the structure of minimax (maxmin) and can be solved through linear programming (LP).

D. Correlated-Q (CE-Q)

Greenwald in his paper states the difficulty of equilibrium selection problem and proposes four variants of Correlated-Q algorithms: utilitarian (uCE-Q), egalitarian(eCE-Q), republican(rCE-Q) and libertarian (lCE-Q). The flavors of the CE decides the magic function f used and ensures the equilibrium value of the game is unique though not necessarily the uniqueness of the equilibrium policy. Discussed in the learning session, as the soccer game is an zero-sum game, the flavor of CE does not matter to the policy selection and value generation. The equilibrium requires the formulation of an LP including the probability and rationality constraints.

At the high level, the rationality constraints of CE can be described as: The expected payoff from playing the recommended strategy is no worse than playing any other strategy.

The mathematical formulation in LP as:

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

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

$$\sum_{\bar{s}} x_s = 1$$

V. Experiments & Results

In this section, the reproduction of figure 3 in Greenwald's paper will be illustrated in technical details with the analysis and discussion of each of the four Q-learning algorithms.

A. *Q-Learning*

Figure 3 shows ERR_i^t for Q-learning algorithm across 1 million training steps. It shows a decreasing Q-value difference at state s. However, this is sorely due to the learning rate α decay. As the paper points out, simple Q-learning computes the Q-value for player's own action without factoring opponent's action into the consideration. Simple Q-learning works on a static environment. However in the game theory, if the agent does not learn from the opponent, it learns from a noisy dynamic environment hence convergence cannot be guaranteed. From another angle, Q-learning seeks pure/deterministic optimal policy, in the Soccer game, such policy simply does not exist, as any deterministic policy will be exploited by the opponent.

The graph generally resembles the shape in Greenwald's paper. The paper does not specify all the hyperparameter values. However, it does disclose the learning rate $\alpha \rightarrow 0.001$. By adjusting the ϵ-decay and α-decay, figure 3 is produced to look like the original graph. Moreover, the original graph demonstrates significant variations of ERR_i^t between updates (the white spaces between the black lines). This is partially observed in the reproduced version. It could be due to the difference in parameter initialization on Q-table or different decaying functions/rates applied.

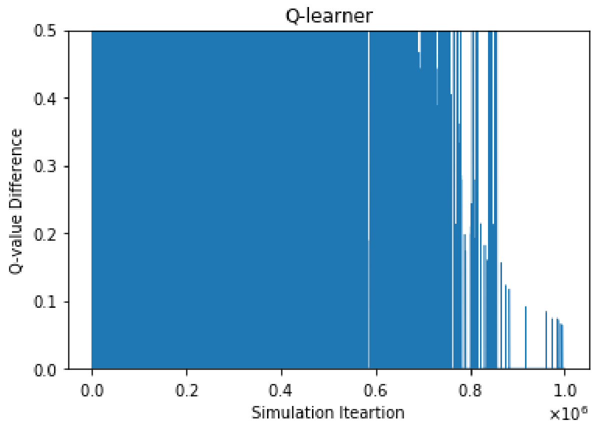

Fig. 3. Convergence in the Soccer Game – Q-learning

B. *Friend-Q*

Figure 4 shows ERR_i^t for Friend-Q algorithm across 1 million training steps. Due to the fast convergence of Friend-Q. Only first 1500 iterations are shown in the graph. It demonstrates a very fast convergence of the Q-value. For Friend-Q agent, it assumes the opponent will fully collaborate to maximize its benefits. In the paper Greenwald argues that Friend-Q results in a pure/deterministic strategy where both player anticipates the opponent to score for himself.

The graph closely resembles the shape in Greenwald's paper, though in the original paper it takes more steps for convergence. This could result from a difference decaying function applied. In the reproduced version, exponential decay is applied hence the learning rate reduces much faster in the early iteration steps. By changing the exponential decaying to a linear decaying mechanism, the convergence might behave more like the one in the original paper.

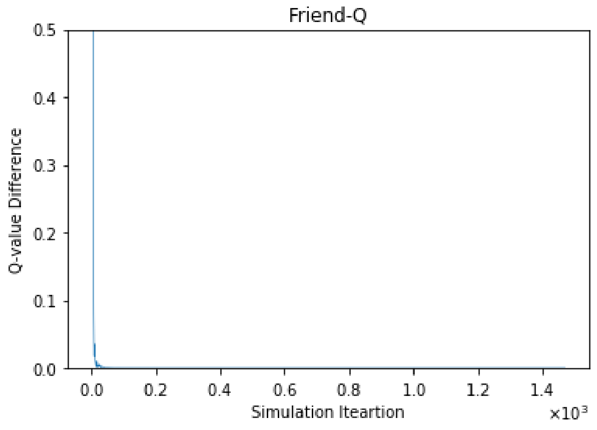

Fig. 4. Convergence in the Soccer Game – Friend-Q

C. *Foe-Q*

Figure 5 shows ERR_i^t for Foe-Q algorithm across 1 million training steps. It shows an a decreasing Q-value difference at state s, indicating the convergence of the algorithm. This aligns with the theoretical foundation discussed in Littman's paper that Foe-Q follows the convergence property of minimax-Q and is guaranteed to converge.

The graph resembles figure 3 in Greenwald's paper, both graphs converges around 700,000 steps. The original graph shows more local variation in ERR_i^t. This could be due to slightly different initialization of ϵ and α. As the maxmin (Foe-Q) algorithm requires an LP solver to calculate value function and probability distribution over actions, different solver used might contribute to such difference.

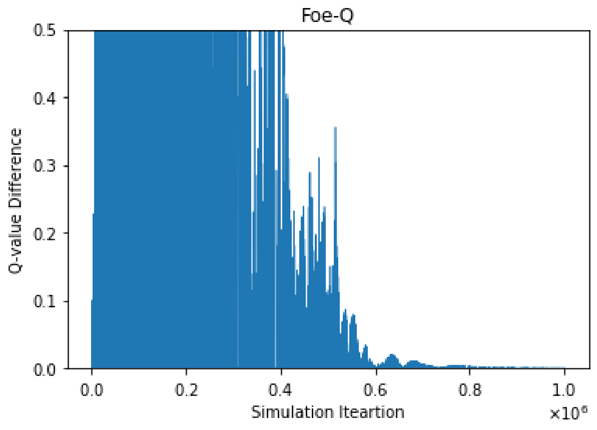

Fig. 5. Convergence in the Soccer Game – Foe-Q

D. Correlated-Q

Figure 6 shows ERR_i^t for Foe-Q algorithm across 1 million training steps. It shows an a decreasing Q-value difference at state s, indicating the convergence of the algorithm. Moreover, the graph closely resemble the graph of Foe-Q in terms of convergence. There are few factors contributing to this observation. First, mentioned in the earlier section, Soccer game is a zero-sum finite-space game, hence different flavors of the CE-Q algorithms will generate same values for all equilibria. Secondly, as explained in the original paper, CE-Q learns the same set of Q-values as Foe-Q. As Foe-Q is the minimax equilibrium (maxmin), Greewald also draws the conclusion that CE-Q learns minimax equilibrium policies in the two-player, zero-sum game.

The graph well aligns with the figure 3 in Greenwald's paper, both graph converges around 700,000 steps. The minimal difference in local variation might results from the use of different LP solver, state representation or hyperparameter selections.

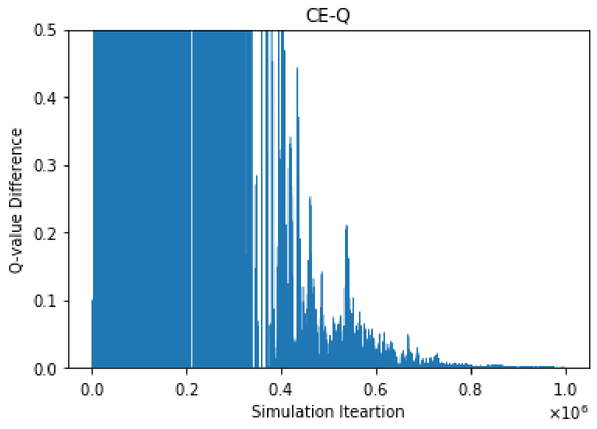

Fig. 6. Convergence in the Soccer Game – CE-Q

VI. CONCLUSIONS

The author of this report has walked through in details on game theory foundation from Nash equilibrium and correlated equilibrium to the generalization of MDP as Markov Games. The construction of soccer environment has been explained in details. The environment takes the states of players and the ball from the agent and returns the next state, reward and the termination signal. The report later discusses the general framework of multiagent Q-learning and the formulations of four implemented Q-learning algorithms: Q-Learning, Friend-Q, Foe-Q and CE-Q, followed by the detailed analysis of reproduced convergence graphs and comparisons to the original papers. The author has gained deeper understanding on Markov Games and CE-Q from technical documents and online materials as this topic involves knowledge and skills across multiple fields including reinforcement learning, game theory and optimizations.

REFERENCES

[1] Littman, M. L. (2001). Friend-or-Foe Q-learning in General-Sum Games. Paper presented at the Proceedings of the Eighteenth International Conference on Machine Learning.

[2] Greenwald, A., & Hall, K. (2003). Correlated-Q learning. Paper presented at the Proceedings of the Twentieth International Conference on International Conference on Machine Learning, Washington, DC, USA.

[3] Amy Greenwald, K. H., Martin Zinkevich. (2005). Correlated Q-Learning. Retrieved from Brown University

[4] Saberi, A. (2009). Correlated Equilibria MS&E 334: Computation of Equilibria.

[5] Roughgarden, T. (2016). The Minimax Theorem and Algorithms for Linear Programming. Retrieved from Stanford University

[6] Jiang, J. (2018). Reinforcement Learning Implementation of Correlated-Q Learning on RoboCup Game. Retrieved from https://github.com/Jerryxe