CS-7642 Project 3 Report – Correlated-Q

Puneeth Kumar Chana Reddy (preddy61@gatech.edu)
4e4189451af5359ab2b2ed42b4098271377ebab3 (Git hash)

***Abstract*—In this paper, we discuss about fundamentals of Game theory, Multi agent learning algorithms like Friend-Q, Foe-Q and CE-Q to solve Robo-Soccer cup environment. Furthermore, the paper replicates the graphs presented in Greenwald-Hall-2003 paper using different multi agent Q-learning algorithms.**

I. Introduction

MUlti-Agent Reinforcement Learning (MARL) focuses on how multiple agents can collectively collaborate, interact and learn from each other in an environment to efficiently work with each other. In a multi agent reinforcement learning, the environment is presented as stochastic game where the actions of the other agents lead to non stationary transitions in the environment, which becomes a biggest challenge to overcome. However, the transitions remains stationary in a single agent environment. Therefore the algorithms used for single agent doesn't perform well in an multi agent stochastic environment. A new set of multi agent algorithms like Friend-Q, Foe-Q and CE-Q will studied in detail and the Figure 3 of Greenwald-Hall paper will be replicated and analyzed in the coming sections.

II. Theory of Markov Games

The theory of games are designed to understand multi-agent interactions. Markov games is an extension of game theory to MDP-like environments. Recently, there have been several multi agent learning algorithms that learns equilibrium policies in general-sum Markov games. Littman proposed Friend-or-Foe-Q (FF-Q) and Hu and Wellman proposed an algorithm called Nash-Q. The Nash-Q converged to Nash equilibrium under restrictive conditions and there are no efficient methods to solve for Nash equilibrium. On the other hand FF-Q as per Littman always converged to equilibrium policies for a restricted class of games e.g. two-player, constant-sum Markov games. Linear programming could be used to solve Foe-Q which converges to minimax equilibria whereas Friend-Q more likely works for co-ordination games.

A. Nash Equilibrium

Nash equilibrium is an optimal solution in a non-cooperative game involving multiple players in which none of the player gain anything by switching to any other strategy, given each player knowing the equilibrium strategy of the other players. Formally, let S_i be the set of all possible strategies for player i, where i=1, …, N.

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i^*, s_{-i}^*) \text{ for all } s_i \in S_i \quad (1)$$

where s_i is the strategy profile of player i and s_{-i} is the strategy profile for other players.

B. Game of Chicken

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

Fig. 1. Payoff matrix for the game of Chicken

In the above game of "Chicken" there are three Nash equilibria. Two strategies (Dare, Chicken) and (Chicken, Dare) are pure but there is also a mixed strategy equilibrium where each player Chickens with probability 2/3 and Dares with probability 1/3. It results in expected payoffs of 4.667 for each player.

C. Correlated Equilibrium

In game theory, a correlated equilibrium is a solution concept that is perceived as generalization of Nash equilibrium. At a high level, a third party brings in coordination between the agents to eliminate combinations that generate worst utility, thereby further increasing the expected payoff of Correlated Equilibrium compared to that of the mixed strategy Nash equilibrium.

In the game of Chicken, an oracle draws one of three combinations (Chicken, Chicken), (Dare, Chicken), and (Chicken, Dare) uniformly at random and assigns the actions to the players. This kind of strategy assignment eliminates (Dare, Dare) which provides worst utility, therefore increasing expected payoff for this equilibrium to 5 compared to 4.667 of the mixed strategy Nash equilibrium.
Furthermore, Correlated equilibrium is computationally less expensive compared to Nash equilibrium. Linear Programming (LP) is efficient in solving for the correlated equilibrium. Unlike Nash equilibrium where no efficient method of computation is known.

D. Markov Games

A Markov Decision Process (MDP) is defined by a set of states (S), actions (A), the transition function $T : S \times A \rightarrow$PD(S) and the reward function $R : S \times A \rightarrow R$. In broad terms, the agent's objective is to find an optimal policy π^* that is deterministic so as to maximize the expected sum of discounted reward. Hence an MDP is called as a one-player Markov game. The update equation is dependent on state-action pair as show below

$$Q^*(s,a) = (1-\gamma)R(s,a) + \gamma \sum_{s'} P[s^{'}|s,a]V^*(s^{'}) \quad (2)$$

However, in the real world, other agents in the environment are not stationary, they interact with each other engaging in collaborations, competitions to maximize their own rewards. These interactions lead to a stochastic environment.

In its general form, a Markov game, sometimes called a stochastic game, is defined by a set of states S, and a collection of action sets, $A_1 \ldots A_k$, one for each agent in the environment. State transitions are controlled by the current state and one action from each agent $T : S \times A_1 \times \cdots \times A_k \rightarrow$PD(S). Reward function for each agent is defined as, $R_i : S \times A_1 \times \cdots \times A_k \rightarrow R$.

In Markov games, player i's Q-values are defined over states and action-vectors $\overrightarrow{a}$. The update equation is dependent on state and action-vectors as show below

$$Q_i(s,\overrightarrow{a}) = (1-\gamma)R_i(s,\overrightarrow{a}) + \gamma \sum_{s'} P[s^{'}|s,\overrightarrow{a}]V_i(s^{'}) \quad (3)$$

III. Soccer Game Environment

Soccer field is a grid of size $2 \times 4 (rows \times columns)$, There are two players A & B whose possible legal actions are N,S,E,W and Stick. For each step, the player's actions are executed random. If this sequence of actions causes the players to collide (two players occupy the same cell), the ball possession changes from the second mover to the first mover. If the player with the ball moves into a his own goal, then he scores $+100$ and the other player scores -100, on the other hand, if he moves to other's goal with the ball then he scores -100 and the other player scores $+100$. In either case, the game ends.

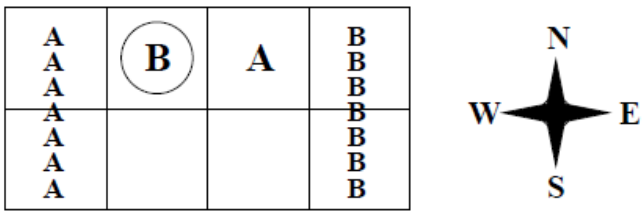


Fig. 2. Soccer Game

A. Implementation

Algorithm 1: Soccer Grid World

Input: New actions from player A and B
Output: [position A, position B, ball possession], scores,done
Init scores;
Check input actions;
Randomly select players to move;
if *unknown actions* **then**
  return default position
else
end
Move First Player
if *no collision* **then**
  update position of the first player;
  if *scores for himself or opponent* **then**
    return scores, done=1;
  else
else
  check ball status and exchange possession;
end
Move Second Player
if *no collision* **then**
  update position of the second player;
  if *scores for himself or opponent* **then**
    return scores, done=1;
  else
else
  check ball status and exchange possession;
end
return [position A, position B, ball possession], scores, done

IV. Multiagent Q-learning

A general template for multi-agent Q-learning is described in Greenwald's paper. The idea of multi-agent learning is that they update the Q-values and the value functions jointly. Various parameters like α (learning rate), γ (discount factor), ϵ(epsilon), α-decay (alpha decay), ϵ-decay (epsilon decay) that were used in the single agent is also applicable in multi-agent Q-learning.

In game theory, as described above Nash equilibrium is highly general and assumptions are quite restrictive, therefore to overcome this several alternative approaches are proposed like Littman's FF-Q algorithm (Friend and Foe Q), Correlated-Q limited to finite, two-player Markov games which can be solved using Linear Programming.

Algorithm 2: Multiagent Q-learning

Input: f, γ, α, α-decay, ϵ-decay, T
Output: Q value error for state 's'
Initialize qDiffValues
Initialize Q values for two players
while $t < T$ **do**
  Reset env to get current state;
  while *True* **do**
    Store Q value for state 's'
    Generate new actions for two players using ϵ-greedy
    Generate next state, rewards and done flag by performing env.step()
    if *done* **then**
      calculate error and store in qDiffValues
    else
      Q-learning update
      Use equilibrium selection function f and LP to solve for V and pi
      decay ϵ using ϵ-decay
      decay α using α-decay
    end
  end
end

A. Q-learning

Q-learning uses a standard off-policy update. In the soccer environment, two Q-tables are constructed for each of the players independently with the dimensions (8,8,2,5). The first two dimensions contain 8 possible states for both players, the third dimension '2' hold the ball possession and the last dimension '5' contains legal actions for both the players. The Q-tables are updated independently without taking opponent's actions into consideration. The Q-learning follows the below update equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)] \tag{4}$$

B. Friend-Q

Littman in his paper shows that in a two player zero sum game the opponent is considered to be a friend, and the value function comes from the joint actions that maximize the Q-values. The Friend-Q agent selects the maximum Q value from each column of the game matrix then selects a maximum from previously chosen column Q-values and any tie among the values are broken randomly. In the soccer environment, two Q tables of dimensions (8,8,2,5,5) constructed for both the players, the last two dimensions represent the action space of the player and the opponent. The Nash equilibrium in this scenario transforms to:

$$Nash_i(s, Q_1, Q_2) = \max_{a_1 \in A_1, a_2 \in A_2} Q_1[s, a_1, a_2] \tag{5}$$

C. Foe-Q

Littman in his paper shows that in a two player zero sum game the opponent is considered to be a foe, and the value function comes from the maximization efforts given the worst-case situation posted by the opponent. The Foe-Q agent uses the game matrix similar to Friend-Q but calculates min-max instead max-max. The Nash equilibrium in this scenario transforms to:

$$Nash_i(s, Q_1, Q_2) = \max_{\pi \in (A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} \pi(a_1) Q_1[s, a_1, a_2] \tag{6}$$

The minimax function is solved using Linear programming. To do this, a value V is introduced as the minimal utility, the opponent's action can cause the agent to have, given a policy π_s(a) under below probability constraints and certain inequalities as show below

$$\pi_s(N)Q(s, N, N) + \pi_s(N)Q(s, S, N) + \cdots + \pi_s(Stick)Q(s, Stick, N) \tag{7}$$

The above inequality is for Player A taking all legal actions against action 'N' of Player B. On the similar lines another four inequalities are constructed for remaining four actions of Player B.
In addition to the above inequalities, probability constraints are also used as shown below

$$\pi_s(a) \geq 0 \tag{8}$$

$$\sum_{a \in A(s)} \pi_s(a) = 1 \tag{9}$$

D. Correlated-Q (CE-Q)

The Correlated-Q (CE-Q) algorithm calculates correlated equilibria. As described above in a correlated equilibria no player is motivated to deviate unilaterally, in other words deviating will not lead player to gain additional reward.
Greenwald in his paper proposes four variants of objective functions to resolve equilibrium selection problem:

- **utilitarian (uCE-Q)**: This function maximizes the sum of all player's rewards
- **egalitarian (eCE-Q)**: This function maximizes the min of all player's rewards
- **republican (rCE-Q)**: This function maximizes the max of the player's rewards
- **libertarian (lCE-Q)**: This function maximizes the maximum of each individual player's rewards

In the context of the soccer game, we mainly focus on utilitarian (uCE-Q). The LP formulation is done using the probability constraints as per equations 8 and 9, the objective function σ as per the below equation:

$$\sigma \in \arg\max_{\sigma \in CE} \sum_{i \in N} \sum_{\overrightarrow{a} \in A} \sigma(\overrightarrow{a}) Q_i(s, \overrightarrow{a}) = 1 \tag{10}$$

and the rationality constraints as shown below:
A set of four inequalities state that when player A is instructed to choose 'N', he doesn't gain additional reward on deviating to alternative actions.

$$\pi_s(N, N)Q_1(s, N, N) + \ldots \pi_s(N, Stick)Q(s, N, Stick) \geq \pi_s(N, N)Q_1(s, S, N) + \ldots \pi_s(N, Stick)Q(s, S, Stick) \tag{11}$$

$$\ldots$$

$$\pi_s(N, N)Q_1(s, N, N) + \ldots \pi_s(N, Stick)Q(s, N, Stick) \geq \pi_s(N, N)Q_1(s, Stick, N) + \ldots \pi_s(N, Stick)Q(s, Stick, Stick) \tag{12}$$

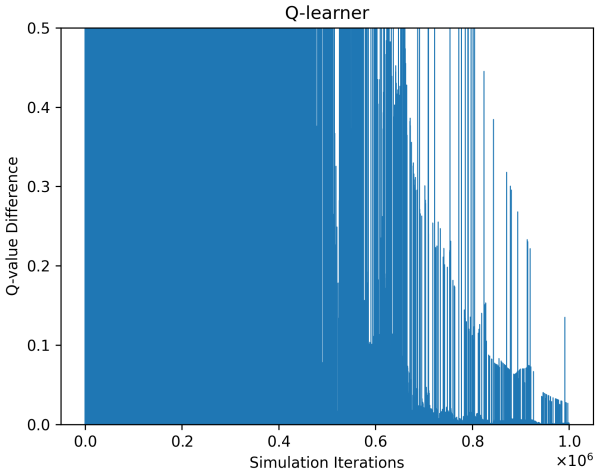
On the similar lines set four equations are built for other four actions taken by Player A making it 20 inequalities for player A and another 20 inequalities for Player B.

V. Experiments and Results

In this section, the experiments specified in Greenwald's paper are performed on the soccer environment and the graphs in Figure 3 of the paper are replicated. Different Q-learning algorithms are run for 1 million steps. Before each step the player A's Q-value corresponding to state 's' with player A taking action 'S' and player B 'Sticks' is recorded. Next a Q-values update is performed based flavor of the Q-learning algorithm and then the Q-value for player A at the same state 's' is recorded again. The error between the two recording is defined by the below equation:

$$ERR_i^t = |Q_i^t(s, \overrightarrow{a}) - Q_i^{t-1}(s, \overrightarrow{a})| \tag{13}$$

A. Q-learning

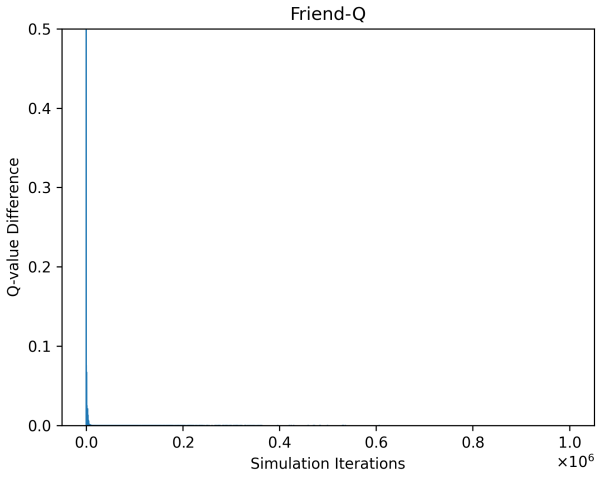


The above figure shows Q-value difference (ERR) for state 's' across 1 million training steps. The graph shows the decreasing Q-diff values from the iterations 800K and above, it is mainly due to reducing learning rate of α=.001. As expected, a simple Q-learning is good for stationary environments but doesn't suit for stochastic environment like the soccer game involving multiple players.

Q-learning simply computes the Q-values of player A independently without considering the actions of player B, hence the convergence is not guaranteed. Also Q-learning looks for a deterministic policy which doesn't exist in a stochastic environment, because any deterministic policy can be easily exploited by the opponent.

The graph generally resembles the shape in Greenwald's paper. The original graph contains a white spaces at regular intervals but the reproduced graphs doesn't show so many white spaces, this may be due to the difference in parameter initialization on Q-table or different decaying rates applied.

B. Friend-Q

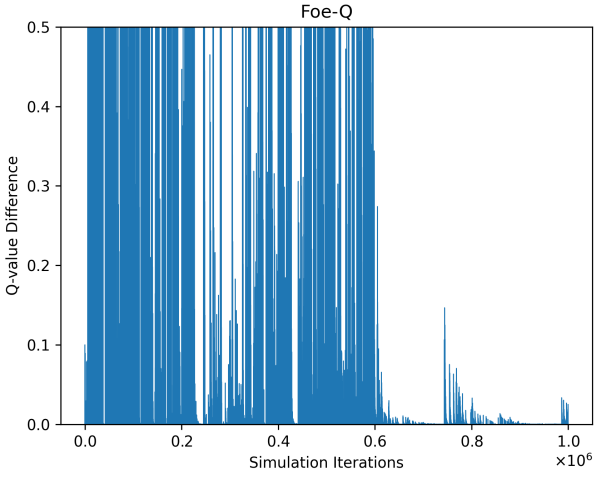


The above figure shows Q-value difference (ERR) for state 's' across 1 million steps. Friend-Q algorithm converges quickly within the first 1000 steps as shown in the graph. Friend-Q algorithm assumes that the player B will completely co-operate with player A to maximize the rewards. In other words both players anticipate that the opponent player will score himself. Hence the Friend-Q converges to a deterministic policy for player B at state 's', namely action E.

The above graph closely matches to that of Greenwald's, but the original graph takes few more iterations to converge. This could be because of exponential α-decay and ϵ-decay function that was applied which lead to quicker convergence, whereas by applying some kind of linear decay mechanism would have resulted in a more closely matching graph.

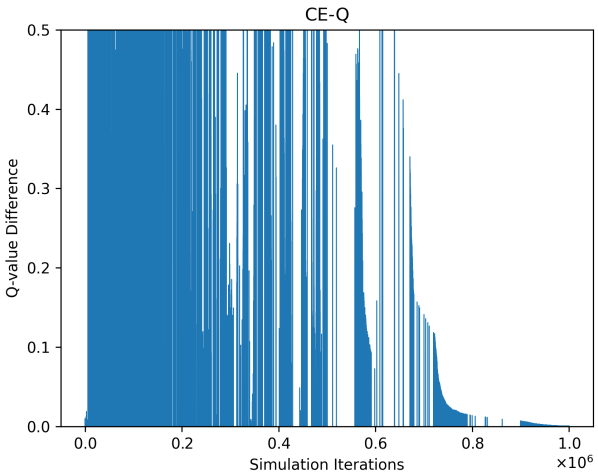
Also paper doesn't specify anything about the initial values of α-decay, ϵ-decay, γ, and Q-table initialization. This could be also be reason for the difference in the graphs.

C. Foe-Q



The above figure shows Q-value difference (ERR) for state 's' across 1 million steps. It shows a decreasing Q-value difference after 600K steps indicating the convergence of the algorithm. As per the Littman's paper Foe-Q follows the convergence property of minimax-Q and is guaranteed to converge.
The graph shows slight variation to the original graph in terms of small spike in Q-value difference at around 780-820K steps. This could be because of different initialization of α-decay, ϵ-decay, γ values and Q-table initialization.
Linear programming is used to calculate the value function and probability distribution over actions. In the current implementation cvxopt (glpk) solver is used which may be different from the solver used in Greenwald's paper leading to this slight difference in the graphs.

D. Correlated-Q



The above figure shows Q-value difference (ERR) for state 's' across 1 million steps. It shows a decreasing Q-value difference after 700K steps indicating the convergence of the algorithm. Furthermore, the graph generally looks similar to the one from the original paper but with some slight differences in terms convergence happening only at around 700K steps whereas in the original paper convergence happen much earlier. This could be because of different initialization of α-decay, ϵ-decay, γ values and Q-table initialization.
Linear programming is used to calculate the value function and probability distribution over actions. In the current implementation cvxopt solver is used which may be different from the solver used in Greenwald's paper leading to this slight difference in the graphs.
Secondly, as per the original paper CE-Q learns the same set of Q-values as Foe-Q which is also seen in the experiments here. Greewald also draws the conclusion that CE-Q learns minimax equilibrium policies in the two-player, zero-sum game.

VI. Implementation Pitfalls

- **Hyper-parameter tuning**: Tuning all the hyper-parameters to achieve results close to the graphs presented in Greenwald's paper is not only difficult but also time consuming. Number of episodes were reduced to e.g. 200k until a right set of parameters were found which showed the some convergence. After finding a good hyper-parameter combination, the algorithms were run for a complete 1 million steps.
- **Solving Linear Programming**: Understanding linear programming and building the constraints for For-Q and CE-Q was tricky. Reading a lots of discussions on Piazza helped in figuring out multiple ways of solve LP.

VII. Future Work

Author would have evaluated above mentioned algorithms for performance by increasing the size of the soccer field. Author would have also liked to train the agents with different Q-learning algorithms and let them compete against each other to see how they perform in the soccer environment.

VIII. Conclusion

The report has explored the basics of Markov games and various challenges faced in stochastic environments. Next, it also gave an insight into Nash equilibrium and correlated equilibrium to the generalization of MDP as Markov Games. Soccer game and the implementation was discussed in detail and various Q-learning algorithms were studied in the context of multi player soccer game. Finally, graphs in the Greenwald's paper were replicated by conducting experiments as described and the reasons causing some slight variation in the graphs compared to the original ones were discussed in detail to gain more understanding of the Q-learning, FF-Q and CE-Q algorithms.

References

[1] Amy Greenwald, Keith Hall, and Roberto Serrano. Correlated Q-learning. In: ICML. Vol. 20. 1. 2003, p. 242.
[2] Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.
[3] Michael L. Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In Proceedings of the Eleventh International Conference on International Conference on Machine Learning (ICML'94). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 157–163.
[4] Michael L. Littman. 2001. Friend-or-Foe Q-learning in General-Sum Games. In Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 322–328.