CS7642 - Decoding Greenwaldś Correlated Q-Learning

d9a2cfbbeb86b6e8d93e8e691b840d025e1ff441 (Git Hash)

Preprint, compiled November 23, 2020

Sangeet Moy Das (sangeet.das@gatech.edu)

College of Computing, Georgia Tech

Abstract

The objective of this project report is to replicate and verify the performance of Correlated Equilibrium Q-Learning against Foe-Q, Friend-Q and Q-Learning in an multiagent zero-sum soccer game as presented by Amy Grrenwald and Keith Hall Greenwald in their paper titled Correlated Q-Learning [1]. Alongside this, the report also discusses the theoritcal foundation of Game Theory and the application of the above multiagent Q-Learning techniques in solving the Soccer game as discussed in Section 5 of Greenwald's paper.

1 Introduction

A classic single agent reinforcement learning deals with having only one actor in the environment. Imagine yourself playing soccer alone without knowing the rules of how the game is played. You'll begin with randomly wandering the soccer field without receiving any 'reward'. After some time, you would start to hit the ball (although not very often) in the goal making the expected reward significantly low. Later on, after several trials (and errors), you would 'learn' that you have to hit the ball in the goal post and try to achieve that consistently. In contrast, in the case of multi-agent reinforcement learning, you wouldn't be alone playing the game. For simplicity, assume that you have a single opponent and you both have your individual goalposts, in this case you will have to ensure that you get as many goals (rewards) as possible irrespective of how your opponent behaves. However, your opponent is again in the same environment maximizing the same objective trying to learn the game. Therefore, his actions would impact how your actions change the environment. In this case, if you hit the ball, he shall try to defend it from going into the post! Notice the key difference here: your goals are conflicting! In a classical multi-agent environment, multiple agents would simultaneously act affecting the learning process of each of the other agents. The challenge is: how can your agent behave optimally? In other words, how can you as a player take actions hitting as many goals as possible in the presence of an opponent who also shares the same objective? Such a setting would transform most of the multi-agent RL into game-theory problems.

2 Background

Previously in homework 6, we were tasked with solving for Nash Equilibrium using Linear Programming where the goal was to derive a set of strategy for the players to reach the equilibrium state. These equilibrium conditions are translated into an objective function and a set of constraints are set for our solver to generate the probability distribution of the strategy. In this project report, we have extended the techniques of finding the equilibria to the environment of "Soccer Game" and estimate the Q values and value functions V. As described in Michael Littman's paper [2] multi-agent RL addresses the sequential decision-making problems, but with more than one agent involved. In particular, both the evolution of the system state and the reward received by each agent are influenced by the joint actions of all agents. More intriguingly, each agent has its own long-term reward to optimize, which now becomes a function of the policies of all other agents.

2.1 Nash & Correlated Equilibrium

John Hillas, in his paper [3] correctly points out that when Noncooperative games are examined from the point of view of an outside observer who believes that the players are rational and that they know at least as much as the observer. The observer is assumed to be able to observe many instances of the play of the game; these instances are identical in the sense that the observer cannot distinguish between the settings in which different plays occur. If the observer does not believe that he will be able to offer beneficial advice then he must believe that the players are playing a correlated equilibrium, though he may not initially know which correlated equilibrium. If the observer also believes that, in a certain sense, there is nothing connecting the players in a particular instance of the game then he must believe that the correlated equilibrium they are playing is, in fact, a Nash equilibrium.

CS7642 discusses that, for a finite game, there is always atleast one Nash Equilibrium. In some cases, like the game of chicken (Figure 1), there are three Nash Equilibria. The two pure strategy Nash Equilibria are (Dare, Chicken Out) and (Chicken Out, Dare). There is also a mixed strategy equilibrium where each player Dares with probability 1/3.

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

Figure 1: A Game of Chicken (Source: wiki)

From the lecture, it is proposed that when a third party randomly draws a card from three combinations: (Chicken Out, Chicken Out), (Dare, Chicken Out), and (Chicken Out, Dare) and the two players strictly follow the actions shown on the card, the average return can further improve to $7(1/3) + 2(1/3) + 6(1/3) = 5$. At the highlevel, 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.

2.2 *Markov/Stochastic Games*

One direct generalization of MDP that captures the intertwinement of multiple agents is Markov games (MGs), also known as stochastic games [4]. Originated from the work of Littman [2], the framework of Markov Games has long been used in the literature to develop multi-agent RL algorithms. A Markov game is defined by a tuple $\left(N, S, \left\{A^i\right\}_{i \epsilon N}, P, \left\{R^i\right\}_{i \epsilon N}, Y\right)$, where $N = \{1, ..., N\}$ denotes the set of $N > 1$ agents, S denotes the state space observed by all agents, A^i denotes the action space of agent i. Let $A := A^1 \times ... \times A^N$, then $P : S \times A \rightarrow \Delta(S)$ denotes the transition probability from any state $s \epsilon S$ to any state $s' \epsilon S$ for any joint action $a \epsilon A; R^i : S \times A \times S \rightarrow \mathbb{R}$ is the reward function that determines the immediate reward by agent i for a transition from (s, a) to $s'; \gamma \epsilon [0, 1]$ is the discount factor. 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's proposes:

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

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

3 SOCCER GAME ENVIRONMENT

One of the games used by Greenwald and Hall in their experiments is so called "Soccer game" first introduced in Littman in "Markov games as a framework for multiagent reinforcement learning" [2]. The game is taking place in a grid world of size 2x4.

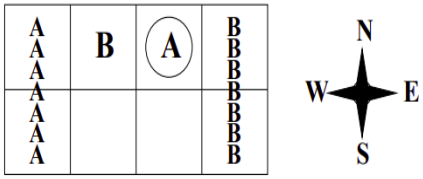

Figure 2: Soccer Game. The circle represents the ball. If player A moves W, he loses the ball to player B; but if player B moves E, attempting to steal the ball, he cannot.

(Source: Greenwald's Paper[1])

There are two players. At each iteration both players choose one of 5 actions: move N, S, W, E or don't move. Both players actions executed in random order with following rules: if player is trying to move to the cell occupied by another player then the movement doesn't take place, but if the moving player had a ball at that moment then the ball is going to the standing player. If player that has ball moves to one of two targets, then player assigned to that target receives +100 points and other player loses 100 points, this makes it a zero-sum game. Original paper doesn't specify what happens if player is attempting to move outside of the grid boundaries, so in our experiments we're assuming that player doesn't move in this case.

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*

```
Init scores
Check input actions and randomly decide sequence of movement of A & B
First mover moves
if collision then
    check ball status and exchange possession
else
    update positions of first mover
    if first mover scores for himself or opponent then
        update scores, done=1, return
end
Second mover moves
if collision then
    check ball status and exchange possession, return
else
    update positions of second mover
    if second mover scores for himself or opponent then
        update scores, done=1, return
end
return [state_A, state_B, ball_possession], scores, done
```

Algorithm 1: Defining our Soccer Environment

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.

4 MULTIAGENT Q-LEARNING

Greenwald and Hall introduced four different reinforcement learning algorithms for Markov games: standard Q-Learning, Friend Q-Learning, Foe Q-Learning and Correlated Equilibrium Q-Learning.

4.1 Q-Learning

Q-Learning uses Bellman update rule to update its state-action value table and later use it to calculate policy. Standard Q-Learning takes into consideration only its own actions. This prevents Q-Learning from learning mixed strategies, moreover, its deterministic policies don't take into consideration opponent's actions. The formula of Q-value only involves the action of the player himself:

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

4.2 Friend Q-Learning

Main difference of Friend Q-Learning from standard Q-Learning is modified update rule. Friend Q-Learning uses pair of actions (its own and opponents) instead of just its own action. This should allow it to adapt to opponents actions. But Friend Q-Learner only maximizes over action pairs, which is equivalent to expecting that opponent will actually help the player, which is usually not the case in zero-sum games. We can derive Nash equilibrium as:

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

4.3 Foe Q-Learning

In opposite to Friend Q-Learning Foe Q-Learner is trying to learn actions that will maximize minimal rewards, as if opponent's trying to minimize players rewards. One way of implementing such policy is by using linear programming. By putting constraints on minimal value of expected rewards one can not only find maximum of this value but also calculate probability distribution for actions that can help achieve this expected value. We can derive Nash equilibrium as:

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

4.4 Correlated Equilibrium Q-Learning

Main interest of Greenwald and Hall in this paper was CE Q-Learning algorithms. These algorithms are introduced as an equivalent to Nash Q-Learning, which finds policies for players that are considered to be in Nash equilibrium, when neither player gains from changing the policy if opponent is sticking with its own. CE Q-Learning optimizes in respect to joint probability distributions of players actions. CE Q-Learning allows among action probabilities, so agents can optimize with respect to one another's probabilities, conditioned on their own. Unlike Nash Equilibrium, Correlated equilibrium can be efficiently computed using Linear Programming. But there might be many numbers of correlated equilibria. Greenwald and Hall introduced four different methods of choosing one equilibria over another. We'll cover one of them: utilitarian CE Q-Learning, when total sum of expected payoffs of both players is maximized. This condition is used in linear programming as a target maximization function.

5 EXPERIMENT

In this section we will discuss about the reproduction of the results in figure-3 from Greenwald's paper[1] and discuss the analysis for the four Q-Learning algorithms. The experiment started with initializing the game world for a 2 player zero-sum Soccer game, for which all the algorithms were implemented. The Q-table for each independent player was updated using Bellman's equation, where the best action was implemented using a epsilon-greedy approach with epsilon starting from 0.9 and linearly reducing to 0.01. In our experiment Q-Learning did not converge which was expected. The reduction in Q-value difference came from reducing the alpha in our algorithm, where alpha started at 1 and eventually decayed to 0.001 linearly. However reducing alpha exponentially proved a little useful, but still the author failed to achieve results similar to Greenwald's paper[1]. The huge spikes in Q-Value in the paper is not seen in our case, as the exact method of decaying the alpha down to 0.001 is unknown.

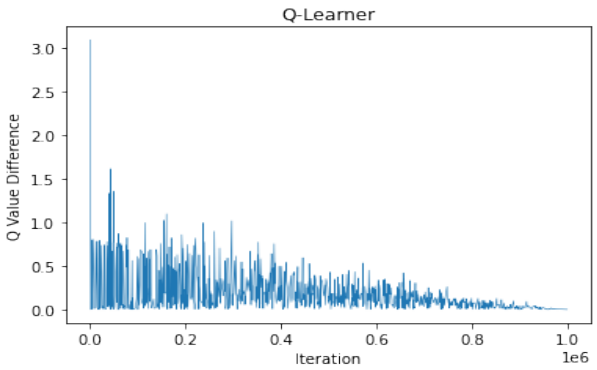

Figure 3: Convergence of Q-Learning

The Friend Q-Learning will converge sooner as the opponent is helping Player A reach a maximum score, by passing the soccer ball and letting Player A score. Very little learning happens after it converges in a few iterations. Q-Learner performs on-policy learning and only randomizes it's action based on the epsilon parameter. But on the other hand Friend-Q, Foe-Q and Correlated Equilibrium-Q performs off-policy learning. The Algorithms take random actions and update their Q-Tables according to the present and discounted future rewards of those actions. In the paper, the gamma used for discount was 0.9 which was also used in all our experiments.

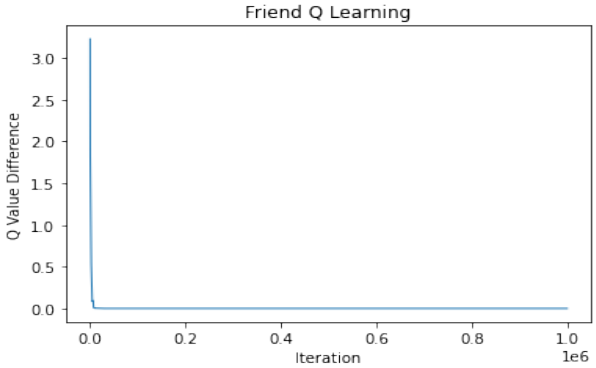

Figure 4: Convergence of Friend Q-Learning

Our third algorithm, Foe-Q was modeled using a single Q Table. It was assumed that the foe was acting solely to reduce Player A's return, which implied that Foe's rewards are the negation of Player A's returns which accurately reflects the zero-sum nature of the game. This is crucial because Nash Equilibrium and minimax strategies coincide in zero sum games which is why our Foe-Q algorithm has no issue converging. Running the minimax algorithm allowed us ot model an adversarial player. A series of linear equations were constructed and linear programming was used to solve the probabilities for each action to take in state S prime. These probabilities were then multiplied by the Q-Value of those future actions. Then the current state's Q value was updated using discounting according to gamma and the learning rate.

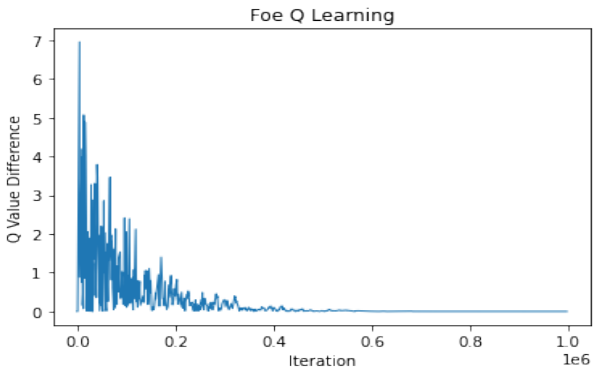

Figure 5: Convergence of Foe Q-Learning

In a pure strategy minimax zero-sum game, th results are the same even if the opponent can predict what the player will do. In this implementation of Foe-Q, the strategy was mixed and both the players chose action which were unknown to the other. However, the mixed strategy that the player chooses has an corresponding best strategy for the opponent. It's like, for a pure strategy game, there exists a unique solution which provides the best possible payoff.

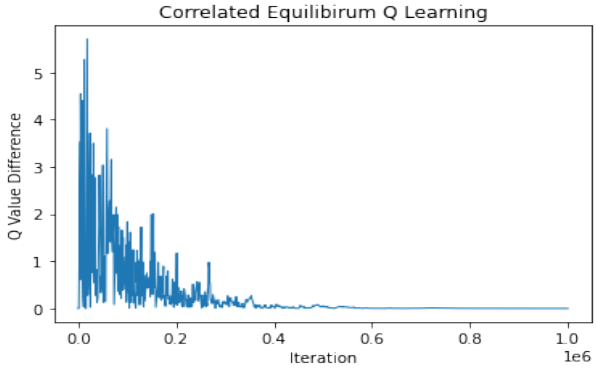

Figure 6: Convergence of Correlated Equilibrium Q-Learning

To solve an Correlated Equilibrium (CE) problem, a set of linear equations were created which represented the constraints of each player. Greenwald introduces CE Q-Learning algorithm as an equivalent to Nash Q-Learning, which find policies for players that are considered to be in Nash equilibrium, when neither player gains from changing the policy if opponent is sticking with its own. CE Q-Learning optimizes in respect to joint probability distributions of players actions. CE Q-Learning allows among action probabilities, so agents can optimize with respect to one another's probabilities, conditioned on their own. Unlike Nash Equilibrium, Correlated equilibrium can be efficiently computed using Linear Programming. But there might be many numbers of correlated equilibria. Greenwald and Hall introduced four different methods of choosing one equilibria over another. In the context of this report we have considered utilitarian CE Q-Learning, where total sum of expected payoffs of both players is maximized. This condition is used in linear programming as a target maximization function.

6 PITFALLS

- **Assumption:** Since we made an assumption that player can take actions that would lead it outside the field and still stay on the same spot, learners didn't see any differences between these actions and action to stick to the same spot. These lead in many cases for learners to ignore latter, which resulted in flat Q-Values update graphs. But since Q-values were initialized randomly some learning attempts were quite consistent with original paper and learned similar strategies.
- **Computation Time:** Also some fast linear programming implementations have proven to be unstable in case of CE Q-Learning, perhaps because of sometimes unpredictable changes in Q-Values and strict restrictions in those implementation. But other implementations showed just as good results even though they didn't run as many training iterations as other q-learners due to limited computational time.

7 CONCLUSION

In this project report the author was successful in repeating Greenwald and Hall's experiments and achieved similar results (but not exact), which once again confirmed efficiency of Q-Learners in multiagent systems. Just like in the original paper

choice of proper Q-Learner algorithm should depend on task's goal and type of the task. Also it was shown that Foe Q-Learning and CE Q-Learning can be efficiently computed using linear programming giving advantages of such algorithms as Nash Q-learning and at the same time making it computationally efficient. Additionally it was also observed Foe-Q, CE-Q and Friend-Q converge to policy quickly whereas the results are identical for Foe-Q and CE-Q as their probability distribution on joint action space is identical due to zero sum.

REFERENCES

[1] 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.

[2] Michael L. Littman (1994). Markov games as a framework for multi-agent reinforcement learning.

[3] John Hillas, Elon Kohlberg, & John Pratt (2007). Correlated Equilibrium and Nash Equilibrium as an Observer's Assessment of the Game.

[4] Shapley, L. S. (1953). Stochastic games. Proceedings of the National Academy of Sciences