Greenwald & Hall's Correlated-Q Learning

Adarsh Gouda
Mechanical Engineer — AI Enthusiast
Edmonton, Canada
adhi.pesit@gmail.com

***Abstract*—This paper describes and replicates the Soccer environment, a two-player, zero-sum Markov game from Greenwald and Hall.[1] Four variants of Q-Learning algorithms are applied to the Soccer game and convergence to equilibrium policies is validated. This paper also discusses the theoretical foundations of multiagent Q-learning techniques.**

I. Introduction

Q-Learning, a specific instance of learning techniques for MDPs in Reinforcement Learning, has remarkable properties of global convergence to an optimal policy. When multiagent Q-Learning is applied to two-player constant-sum games, Q-Learning converges to unique equilibrium.[2] However, for general-sum Markov games, multiagent Q-Learning does not yield strong convergence properties because there are often multiple equilibria. Greenwald and Hall,[1] introduced Correlated-Q (CE-Q) learning which is based on the correlated equilibrium solution concept. CE-Q is shown to outperform Friend & Foe-Q (FF-Q)[3] and Nash-Q[4] methods. In fact, CE-Q generalizes both Nash-Q and FF-Q. Nash-Q, which is based on Nash Equilibrium (NE), learns a Probability Distribution (PD) vector over actions, independent for each agent. NE's are difficult to compute which is why CE-Q was introduced. CE-Q learns PD over joint action space which can be easily computed using Linear Programming(LP). This learned PD is then used to select an ϵ-greedy action in solving the Soccer environment.

II. Soccer & Markov Games

The Soccer game presented in the Section 5 of Greenwald and Hall is a two-player, zero-sum, 4x2 grid game. This is shown in Figure 1 and is a *stochastic game* for which the probability transitions satisfy the Markov property. A stochastic game is defined as a tuple $\langle I, S, (A_i(s))_{s\in S, 1\leq i\leq n}, P, (R)_{1\leq i\leq n}\rangle$ where I is a set of n players, S is a set of states, $(A_i(s))_{s\in S,1\leq i\leq n}$ is player's set of actions given a state s, P is state transition function based on joint actions and past state, $R_i(s,\vec{a})$ is the player's reward function for state s and joint action $\vec{a} \in A(s) = A_1(s)\times\cdots\times A_n(s)$. In Markov games, player i's Q-values are defined over state and action-vectors $\vec{a} = (a_1, ..., a_n)$ and the Q-value function is defined as:

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

MDP then is a special case of single-player Markov game. As compared with the Q-value function of MDP, Markov games have n additional dimensions for each agent(or, player). Since our Soccer game is a two-player Markov game, $n = 2$.

In the Soccer game, Player A always starts in Cell 2. Player B always starts with the ball from Cell 1. This starting state is the primordial state, s. The leftmost and the rightmost two cells 0,4 and 3,7 are Player A and B's goals respectively. The available actions for each player are [N, E, S, W, Stick]. At each step of the game, A and B determine their move simultaneously. However, the execution of those moves is completely random. Indexing the state as shown in Figure 1 helps in simplifying the implementation of players' movements in the grid world by indexing the actions [N, E, S, W, Stick] = [-4, 1, 4, -1, 0]. For instance, Player B moving south [S] from its primordial state 1 would land in state 1 + 4 = 5. If a player enters the assigned goal while in

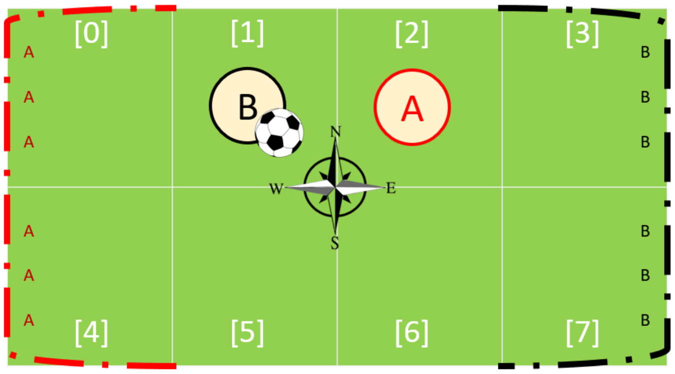

Fig. 1. An illustration of the Soccer grid game environment. This illustration is showing the primordial state of the game, state s. Player B is in possession of the ball.

possession of the ball, the player will receive a +100 reward, and the opponent, this being a zero-sum game, receives a -100 reward. On the flip side, if a player enters the opponent's goal while in possession of the ball, the player will receive a -100 reward while the opponent receives a +100 reward. At any given step in the game, the terminal state is only checked after both the players have executed their actions. If both the players incidentally act to move into the same cell, the player acting second loses possession of the ball. If a player runs into the boundary of the grid world, it is assumed that they bounce back into the same cell. The game is always initiated at the primordial state and also reset to its primordial state after termination.

III. Markov Game Algorithms

For a single agent, Equation 1 reduces to Equation 2, which we know as the Q-value update rule for an MDP. The value function $V^*(s)$ at state s is defined as the value that maximizes $Q^*(s,a)$ over all actions a and is given by Equation 3. The actions that maximize $Q^*(s,a)$ at each state s describe the deterministic optimal policy π^*: i.e., Equation 4.

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

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a) \quad (3)$$

$$\pi^*(s) \in \arg\max_{a \in A(s)} Q^*(s,a) \quad (4)$$

If we can compute the optimal strategy π^* for a Markov game, we will be able to compute $V_i(s')$ which can then be substituted in Equation 1. But Markov games do not necessarily exhibit deterministic equilibrium policies. Therefore, all players maximizing their respective rewards with respect to each other's actions is not adequate, since deterministic actions that satisfy all the simultaneous equations need not exist. Few alternative definitions of the value functions are discussed below.

A. Minimax-Q Algorithm

Littman[5] proposed a minimax-Q algorithm specifically designed for two-player zero-sum stochastic games. The minimax-Q algorithm uses von Neumann's minimax principle[6] to solve for players' Nash equilibrium strategies and values of states for two-player zero-sum stochastic games. The value function is given in Equation 5 and the Q-value update rule is given in Equitation 7

$$V_i^*(s) = \max_{\pi_i(s,\cdot)} \min_{a_{-i} \in A_{-i}} Q_i^*(s, a_i, a_{-i})\pi(s, a_i) \quad (5)$$

$$Q_i(s, a_i, a_{-i}) \leftarrow (1-\alpha)Q_i(s, a_i, a_{-i}) + \alpha[R_i + \gamma V_i(s')] \quad (6)$$

where $-i$ denotes player i's opponent, $\pi_i(s,\cdot)$ denotes all the possible strategies of player i at state s, and $Q_i^*(s, a_i, a_{-i})$ is the expected reward when player i and its opponent choose action $a_i \in A_i$ and $a_-i \in A_-i$, respectively, and follow their Nash equilibrium strategies after that. Linear Programming is used to solve Equation 5. The minimax-Q algorithm can guarantee the convergence to a Nash equilibrium, irrespective of the strategy followed by the opponent, if all the possible states action combinations are visited infinitely often.

B. Nash-Q Learning

The Nash Q-learning algorithm, first introduced by Hu and Wellman,[4] extends the minimax-Q algorithm from zero-sum stochastic to general-sum stochastic games.

$$\begin{aligned} Q_i(s, a_1, ..., a_n) \leftarrow (1-\alpha)Q_i(s, a_1, ..., a_n) + \\ \alpha[(R_i + \gamma Nash_i(s, Q_1, ...Q_n)] \end{aligned} \quad (7)$$

$$Nash_i(s, Q_1, ...Q_n) = Q_i(s, \pi_1, ..., \pi_n) \quad (8)$$

The Q-value update rule for Nash-Q Learning is given in Equation 7, where πs is Nash equilibrium defined by Q-functions at state s. The Nash-Q algorithm converges to a Nash equilibrium strategy only when there is a global optimum for all time-steps at all states. The Nash Q-values need to be calculated at each state in order to update the action-value functions and find the equilibrium strategies. All these conditions are quite restrictive and computationally demanding.

C. Friend-or-Foe Q-Learning

For general-sum stochastic games, Littman[3] proposed a Friend-or-Foe Q-learning (FF-Q) algorithm such that a learner is told to treat the other players as either a "friend" or a "foe". FF-Q starts by letting each agent maintain their own Q-function, contrary to Nash-Q learning. We assume that agents are aware of others agents' action spaces but are not aware of the actions others might choose at any given moment. The Q-value update rule for both Friend-Q and Foe-Q algorithms is the same and is given in Equation 9.

$$\begin{aligned} &Q_i(s, a_1, ..., a_{n_1}, o_1, ..., o_{n_2}) \\ &\leftarrow (1-\alpha)Q_i(s, a_1, ..., a_{n_1}, o_1, ..., o_{n_2}) + \\ &\alpha[(R_i + \gamma Nash_i(s, Q_1, ...Q_n)] \end{aligned} \quad (9)$$

For two player version of FF-Q, which is relevant to our Soccer game, the value-function takes the form of Equation 10 if the opponent is a friend and Equation 11 if the opponent is a foe. Solving Equation 10 is straightforward while Equation 11 could be solved using Linear Programming.

$$Nash_i(s, Q_1, Q_2) = \max_{a_1 \in A_1, a_2 \in A_2} Q_i(s, a_1, a_2) \quad (10)$$

$$Nash_i(s, Q_1, Q_2) = \max_{\pi \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q_i(s, a_1, a_2) \quad (11)$$

D. Correlated Q-Learning

CE-Q generalizes both Nash-Q and FF-Q. In general-sum games, the set of correlated equilibria contains the set of Nash equilibria. In constant sum games, the set of correlated equilibria contains the set of minimax equilibria.

In general-sum games, there exist multiple equilibria with multiple payoff values. Greenwald and Hall, in their paper,[1] attempt to resolve this equilibrium selection problem by introducing four variants of CE-Q, based on four equilibrium selection functions. They are utilitarian, egalitarian, republican, and libertarian CE-Q learning. The soccer game being a zero-sum game, pure stationary equilibrium policies do not exist; all variants of CE-Q coverage everywhere and equilibria at all states are known to have equivalent values indicating all variants of CE-Q produce identical outcomes. Also, CE-Q converges to the same solution as Foe-Q with identical Q-values. Therefore, CE-Q learns minimax equilibrium policies in the two-player, zero-sum game.

A Correlated Equilibrium (CE) is a joint distribution over actions from which no agent is motivated to deviate unilaterally. It is more general than a NE in that it allows for dependencies among agents' strategies. Intuitively, there is a

”referee” who recommends a policy to all agents to follow. If the policy is a CE, all agents should follow the referee’s recommendation. Deviating from the recommendation will not provide any additional rewards when all other players abide by the recommended policy.

IV. Assumptions, Interpretations, Challenges

The challenging part I faced was associating and relating various concepts and algorithms from different papers. After reading several related papers and reference books, I was able to map all these learning algorithms into a sensible context and develop a holistic view on the subject matter. Section III of this paper is the result of this rigorous endeavor. The other challenge was understanding the game itself and developing the environment in Python. I tried different approaches before settling with the code that is implemented which is simple and easier to work with. It was indeed a very good exercise and it certainly developed a confidence in me that I could attempt to replicate a given RL research paper in its entirety and not depend on readily available environments like OpenAI.

A. Dichotomy of On and Off-Policy

The difference in the definition of On-Policy and Off-Policy between Sutton[7] and Greenwald[1] was confusing. After careful literature review, I understood the finer differences in the definition.
Sutton's On-Policy $\rightarrow$ learn $Q(s,a)$ from actions chosen using the current policy $\pi(a|s)$. And, $\epsilon = 1$ or $\epsilon \rightarrow 0.001$
Sutton's Off-Policy $\rightarrow$ learn $Q(s,a)$ from actions chosen using a different policy $\hat{\pi}$ either mixed (from LP) or pure (using argmax). And, $\epsilon = 1$ or $\epsilon \rightarrow 0.001$
Greenwald's On-Policy $\rightarrow$ learn $Q(s,a)$ from actions chosen using a different policy $\hat{\pi}$ either mixed (from LP) or pure (using argmax). And, $\epsilon \rightarrow 0.001$
Greenwald's Off-Policy $\rightarrow$ learn $Q(s,a)$ from actions chosen using a different policy $\hat{\pi}$ either mixed (from LP) or pure (using argmax). And, $\epsilon = 1$

B. Let's just iterate

The notion of an ”episode” is absent in Greenwald’s paper. An episode is the duration of an episodic task. It is the timeframe between the primordial state in the environment and the last possible state (end state) or the terminal state. Greenwald’s implementation of iteration is not 10^6 episodes but it is 10^6 cumulative total steps regardless of the duration of each episode. Without this understanding, the computation time would be significantly longer, and also the plots would not match with what is presented in Greenwald’s paper.

C. Hyperparameters

There is no clarity on the choice of hyper-parameter values in the paper. After several trials, I settled with hyperparameters that would provide consistent results across all four plots. Table 1 shows the hyperparameter values chosen along with the references.

TABLE I
Hyperparameter Values

Hyperparameter	Value	Reference
Discount Factor, γ	0.900	Littman, 1994
Learning Rate, α	1.000	Greenwald, 2005
α-minimum	0.001	Greenwald, 2005
Greedy Policy, ϵ	1.000	Greenwald, 2005
ϵ-minimum	0.001	Greenwald, 2005
ϵ-decay	$10^{\log(\epsilon-min/10^6)}$	Littman, 1994
α-decay	$10^{\log(\alpha-min/10^6)}$	Littman, 1994

Algorithm 1: Multiagent Q-Learning with strategy f

```
Choose a strategy
  $f \in \{Q, Foe-Q, Friend-Q, CE-Q\}$
def Q-Learner(env, PlayerA, PlayerB):
    Initialize: $Q_A, Q_B, V_A, V_B, \pi_A, \pi_B$
    Initialize: error = [ ], iterations = $10^6$
    Initialize: $\epsilon, \gamma, \alpha, end = 0$
    for $i \leftarrow 0$ to iterations do
        if $end = 0$ then
            Reset $env$
        end
        Choose and execute $\epsilon$-greedy action from $\vec{a}$
        Observe $s', R_1...R_n$ and $end$ from $env$
        Record $s \doteq old$
        for $i \in \{PlayerA, PlayerB\}$ do
            $V(s') \leftarrow f(Q_1(s'), ..., Q_n(s'))$
            $Q_i(s, a_1, ..., a_n) \leftarrow$
              $(1-\alpha)Q_i(s, a_1, ..., a_n)+$
            $\alpha[(1-\gamma)R_i + \gamma V_i(s') \cdot end]$
        end
        Record $s \doteq new$
        error.append($old - new$)
        Decay $\epsilon, \alpha$
    end
    return error
```

V. Implementation

Algorithm 1 generalizes the method used by each agent in learning the Soccer game. Like the Q-Learning architecture used in Project 2, multiagent learners too initialize predetermined values for discount factor: γ, learning rate: α, probability factor for greediness in action selection: ϵ and the corresponding decay factors for α and ϵ. For 10^6 iterations, as set up in Greenwald’s paper, the agent generates actions based on ϵ-greedy and executes it in the Soccer environment, env. The env provides joint states, actions, and rewards at each step in the iteration as a response to the action executed. This new observation is then used to update the Q-values. As shown in Algorithm 1, a strategy function f is appropriately chosen based on the type of Q-Learning followed by the agent and evaluates the value-function, V. If the observed state is a terminal state, the env updates the flag $end = 0$, and in the subsequent iteration, the environment is reset to its primordial state. Depending on the type of strategy f, Linear Programming(LP) could be used to solve V and the policy π.

A. *Error Evaluation*

The error at the primordial state s of Player A and Player B while A is moving South and B sticking is evaluated at each iteration. This is computed as shown in Equation 12.

$$ERR_i^t = |Q_i^t(s, \vec{a}) - Q_i^{t-1}(s, \vec{a})| \tag{12}$$

B. *Q-Learning*

In implementing multi-agent Q-Learning, two Q-tables are created for each agent. These Q-tables are independent and are updated separately for Player A and Player B. The Q-tables comprise of 5 dimensions i.e., $Q(8, 8, 2, 5)$. The first two dimensions are state-space for Player A and B. The third dimension, of size 2, is an indication of ball possession; if Player B is in possession of the ball, the value is 0 and if A has the ball, the value is 1. Since a standard Q-Learner does not take the opponent's action into the construct of Q-tables, the update of the Q-value involves only the action of the player. The primordial state at which the error ERR_i^t is evaluated corresponds to $Q_a[2, 2, 1, 2]$.

C. *Friend-Q*

From Equations 9 and 10, it can be inferred that when the opponent is a friend, the value function comes from the joint actions that maximize the Q-value. Therefore, Friend-Q is an extension to Q-Learning by adding one additional dimension to the Q-table i.e., action-space of the opponent. The Q-table of each agent will be $Q(8, 8, 2, 5, 5)$. And, the primordial state at which ERR_i^t is evaluated corresponds to $Q_a[2, 2, 1, 2, 4]$. The Friend-Q algorithm calculates $\pi(s, a)$ by performing a max-max; first pick the max Q-value from each column in $Q(s, a_i, a_o)$ then pick the max among those max values. If there is a tie, break it randomly.

D. *Foe-Q*

From Equations 9 and 11, it is clear that Foe-Q has the same state-action space as that of Friend-Q, $Q(8, 8, 2, 5, 5)$ and the primordial state is the same $Q_1[2, 2, 1, 2, 4]$ at which ERR_i^t is calculated. However, the Foe-Q algorithm calculates $\pi_i(s, \vec{a})$ where $i \in \{agent, opponent\}$ for each player by performing a min-max instead. The min-max can be computed using Linear Programming(LP) method. To do this we define V_i which is the minimum utility that the agent can get as a result of its opponent's action over their respective policies. The five inequality constraint for agent's actions, $\vec{a}$ = [N, E, S, W, St(ick)] are shown in Equation 13.

$$\begin{aligned}
&\pi_i(s, N)Q_i(s, N, N) + \pi_i(s, N)Q_i(s, S, N) + \cdots + \\
&\qquad \pi_i(s, N)Q_i(s, St, N) \geq V_i \\
&\vdots \\
&\pi_i(s, N)Q_i(s, N, St) + \pi_i(s, N)Q_i(s, S, St) + \cdots + \\
&\qquad \pi_i(s, N)Q_i(s, St, St) \geq V_i
\end{aligned} \tag{13}$$

$$\pi_i(s, a) \geq 0; \quad a \in \vec{a} \tag{14}$$

$$\sum_{a \in \vec{a}} \pi_i(s, a) = 1 \tag{15}$$

Now we consider two more equations for boundaries of probabilities shown in Equations 14 and 15.

We then $maximize\ V_i$ to compute minimax and $\pi_i(s, \vec{a})$ is then solved using Linear Programming. Each agent maintains its own probability distribution of its action at each state.

E. *Correlated-Q*

The Q-table setup of CE-Q is similar to Foe-Q in that both algorithms maintain separate $Q_i(8, 8, 2, 5, 5)$ for the agent and the opponent. The CE-Q learner also has an identical algorithm for Linear Programming. However, for CE-Q learner, the probabilities are joint-probabilities over their combined action $\pi(s, \vec{a}_i, \vec{a}_{-i})$. In other words, all agents optimize with respect to one another's probabilities, conditioned on their own. To solve CE-Q, we need three sets of inequality equations. The first set, Equation 16, defines the essence of Correlated-Equilibrium.

$$\begin{aligned}
&\sum_{a_{-i} \in A_{-i}(s)} \pi(s, a_{-i}, a_i) Q_i(s, a_{-i}, a_i) \geq \\
&\sum_{a_{-i} \in A_{-i}(s)} \pi(s, a_{-i}, a_i) Q_i(s, a_{-i}, a_i^{'})
\end{aligned} \tag{16}$$

The second set determines the boundary of probabilities which are Equation 14 and 15. The third set defines the objective function of the CE which is either of the 4 flavors of CE presented in Greenwald's paper - utilitarian, egalitarian, plutocratic, and dictatorial. I have used utilitarian CE-Q, the objective function of which is given by Equation 17.

$$\max_{\pi(s,a) \in \Delta(A(s))} \sum_{j \in N} \sum_{a \in A(s)} \pi(s, a) Q_j(s, a) \tag{17}$$

VI. Results & Discussions

It was a real struggle in choosing the best decay rate for α. After trying out several different ways such as linear, geometric, exponential etc., I finally settled with Table 1. The decay rate has some influence on the way the plots appear, if not significant. All the algorithms, I have implemented, use the same values for hyperparameters from Table 1 for consistency. Although, it is possible to play around with those hyperparameters for individual algorithms to make each plot closely resemble in appearance with the plots from the paper. However, the primary objective is to recreate the idea behind each plot. Therefore, less time was spent on cherry-picking the hyperparameter values. Python(Google Colab) was used as the platform for code implementation.

A. *Q-Learner*

Q-Learning is not guaranteed to converge and it is clear from Figure 2(a) that the algorithm fails to converge. Essentially, the Q-Learner is looking for a deterministic optimal

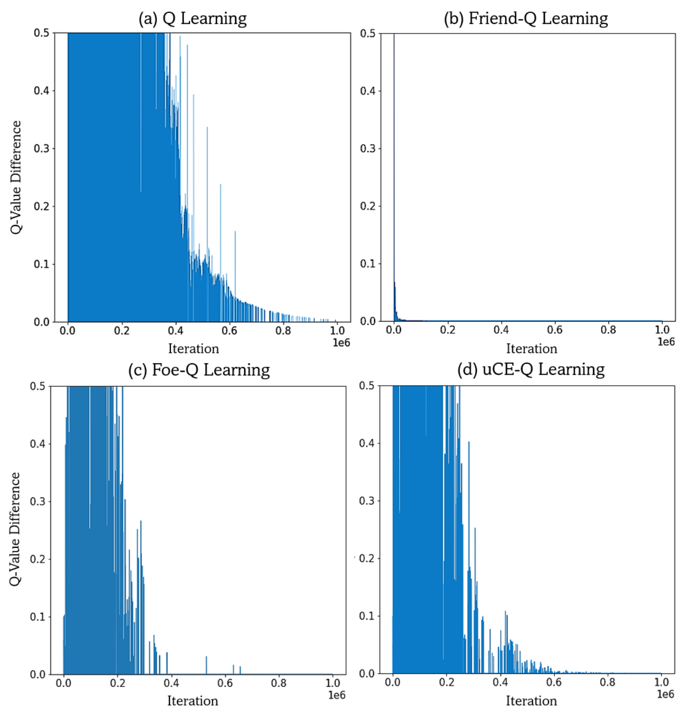

Fig. 2. Convergence in Soccer game. All algorithms except Q converge.

policy but none exists in this game. The reduction in the Q-value difference comes mostly from decaying α. The large spikes at the trailing end are absent but the plot follows the same curve as presented in Greenwald, 2005. Greenwald carefully pointed out the influence of α and $\alpha - decay$ on the plot but did not specify the implementation details and this, I have assumed in Table 1.

B. Friend-Q Learner

The Friend-Q learner quickly converges since the opponent is actively helping Player A reach a maximum score. The plot in Figure 2(b) clearly suggests that there is no learning past the early convergence. This plot closely resembles the plot in the paper. Greenwald argues that Friend-Q converges to deterministic policy for Player B but provides no guarantees to the learned values.

C. Foe-Q Learner

Implementing Foe-Q was not an easy task. CVXPY library was used in the first version of the Foe-Q implementation code. After facing issues with long computation times, I later switched to the CVXOPT library with few suggestions on Piazza and in OH. Besides the plots from Greenwald's paper, having no other reference to verify my results, I found it challenging to understand if my code was after all working as intended. There was no Q-value benchmark for calibration or correction. I resorted to performing thought experiments with the resulting Q-values for various move scenarios with Player A's Q-values. I ran several test cases with varying ϵ, α values, and the decay rate just to make sure that my code response is predictable and cross-verify if my assumptions, in the beginning, were wrong to start with. The Foe-Q plot in Figure 2(c) shows convergence. This agrees with Littman's paper that Foe-Q follows convergence properties of minimax-Q and is guaranteed to converge.[3] But the plot does not exactly match the plot in Greenwald's paper and there are some subtle deviations. Greenwald's plot shows much finer and local variations in the error. This is perhaps due to the difference in ϵ or α initialization combined with the ERR update rate. Future work would involve using a different LP solver to check if different solvers influence the outcome in any way.

D. CE-Q Learner

The CE-Q learner is similar to Foe-Q except that the LP implementation is much more complex. Although, it took a considerable amount of time and effort in researching all the related foundation topics like NE, CE, and Convex Optimization, this section of the paper was the most rewarding in terms of knowledge gain. LP implementation was tried with and without the use of the SCIPY library in Python. It was found that with SCIPY, the code execution was faster and stable. Like Foe-Q learner, uCE-Q learner too converges and this is confirmed in Figure 2(d). Greenwald also draws the conclusion that CE-Q learns minimax policies in two-player zero-sum games. The two learners seem to converge at approximately the same number of iterations ($\approx 0.7e6$) with CE-Q taking few more iterations to converge. The appearance of the plot has the same shortcomings discussed in Foe-Q learner. I suspect that the LP implementation requires improvement. My future work on this would involve minimizing the Q-value difference between Foe-Q and CE-Q since the paper states that both the learners converge to the same values.

VII. Conclusion

Future work will involve implementing a single Q-table for Foe-Q learners to look for improvements in performance. Recreating published papers requires making assumptions and experimenting with different approaches. In this paper, the decay rates are assumed which clearly has a significant influence on the results making it difficult to reproduce exact plots. In addition to some missing information, several other related papers had to be read and understood to wrap my head around Greenwald's paper. Overall, this was an extremely challenging project and it definitely put me out of my comfort zone in several areas. Through this exercise, I have learned a great deal about Markov games, CE-Q, and RL in general.

References

[1] Greenwald, A. and Keith B. Hall. "Correlated Q-Learning." ICML (2003).Vol 20.1.2003, p.242
[2] M. Littman and C. Szepesvari. A generalized reinforcement learning model: Convergence and applications. ICML, pages 310–318, 1996.
[3] Littman "Friend-or-Foe Q-learning in General-Sum Games." ICML (2001).
[4] Hu and Wellman. 2003. Nash Q-learning for general-sum stochastic games.
[5] Littman, "Markov games as a framework for multi-agent reinforcement learning," ICML 1994, pp. 157–163, 1994.
[6] J. von Neumann and O. Morgenstern. The Theory of Games and Economic Behavior. Princeton University Press, Princeton, 1944.
[7] Sutton and Barto, Reinforcement Learning: An Introduction