Project 3 - Correlated Q

*Hash for latest commit to the GitHub repository - c6ee64b662066c232c398e4ae981de77de768542

Prithi Bhaskar

pbhaskar8@gatech.edu

***Abstract*—This report aims to describe the experiments found in [GH03] that introduces Correlated-Q (CE-Q)learning, a multiagent Q-learning algorithm based on the correlated equilibrium (CE) solution concept. It is shown that CE-Q generalises Friend-Q and Foe-Q. It also attempts to describe the process of implementing the experiments and replicating the results found in [GH03]. As shown in [GH03], the performance of various learning algorithms like Friend-Q, Foe-Q and Q-Learning are analysed in comparison with CE-Q on a general-sum Markov game called [Lit94].**

I. Correlated-Q Learning

[GH03] introduces Correlated-Q (CE-Q) learning, a multiagent Q-learning algorithm based on the correlated equilibrium solution concept. CE-Q generalizes both Nash-Q and FF-Q: in general-sum games, the set of correlated equilibria contains the set of Nash (and thus, coordination) equilibria; in constant-sum games, where Nash and minimax equilibria coincide, the set of correlated equilibria contains the set of minimax equilibria.

Correlated Equilibrium (CE) is more gen- eral than a Nash Equilibrium (NE) in that it permits dependencies among the agents' probability distributions, while maintain- ing the property that agents are optimizing. Unlike Nash equilibria, CE can be computed easily via linear programming. Also, CE that are not NE can achieve higher rewards than NE, by avoiding positive probability mass on less desirable outcomes, unlike mixed strategy Nash equilibria.

II. Q-Learning Algorithms

[GH03] demonstrates empirical convergence to equilibrium policies for CE-Q learning algorithm and also compares the performances of other variants of Q-learning on a testbed of Markov games. The variants include Friend-Q, Foe-Q and Q-learning algorithms. This report attempts to explain the replication of the results presented in [GH03] for the game of [Lit94]

In the case of a single-agent MDP, the optimal state-action values for the MDP can be calculated using Bellman's equations as shown below:

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

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

In Markov games, player i's Q-values are defined over states and action-vectors $\vec{a}$= (a1, . . . , an), rather than state-action pairs:

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

In the case of Markov games, several alternative definitions of the value function have been proposed. [Lit01] studied two-player, zero-sum games and presented Foe-q learning which describes the value function as below:

$$V_1(s) = \max_{\sigma_1\in\Sigma_1(s)} \min_{a_2\in A_2(s)} Q_1(s,\sigma_1,a_2) = -V_2(s) \quad (4)$$

At the opposite extreme, Littman's [Lit01] value function is suited to coordination games—games for which all the players' reward functions are equivalent—with uniquely-valued equilibria:

$$V_i(s) = \max_{\vec{a}\in A(s)} Q_i(s,\vec{a}) \quad (5)$$

For the general case of n-player, general-sum games, the definition of the value function that uses Nash Equilibrium can be written as:

$$V_i(s) \in Nash_i(Q_1(s),...,Q_n(s)) \quad (6)$$

where $Nash_i(X_1, \ldots ,X_n)$ denotes the i^{th} player's reward according to some correlated equilibrium in the general-sum game determined by reward matrices $X_1, \ldots ,X_n$.

[GH03] propose an alternative definition of the value function for Markov games:

$$V_i(s) \in CE_i(Q_1(s),...,Q_n(s)) \quad (7)$$

where $CE_i(X_1, \ldots ,X_n)$ denotes the i^{th} player's reward according to some correlated equilibrium in the general-sum game determined by the rewards $X_1, \ldots ,X_n$.

Four variants of correlated-Q learning are presented in [GH03] and each variant has been shown to resolve the equilibrium selection problem. For the purpose of reproducing the results shown in Figure 3 of [GH03] , a variant - uCE-Q was implemented. uCE-Q, otherwise known as utilitarian CE-Q, attempts to maximise the sum of all the players' rewards. This is given by the equation:

$$CE_i(\vec{Q}(s)) = \sum_{\vec{a}\in A} \sigma(\vec{a})Q_i(s,\vec{a}) \quad (8)$$

where σ satisfies the below equation:

$$\sigma \in arg \max_{\sigma \in CE} \sum_{i \in I} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a}) \tag{9}$$

where $\sigma(\vec{a})$ is the probability distribution over the entire action space of all players.

III. Soccer

The game of [Lit94] is used as the environment to present the efficiency of correlated Q algorithm in [GH03], mostly following the environment details given in the paper. For details that were not listed explicitly in the paper, [Lit94] was used to draw assumptions.

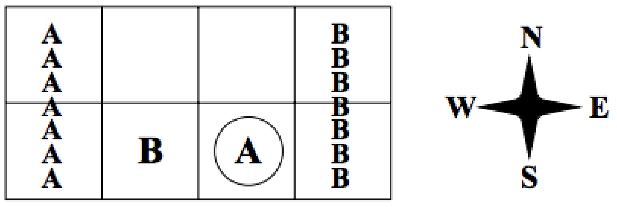


Figure 1. State s of Soccer Game.

The soccer field is a grid of 2x4. The circle represents the ball. There are two players, whose possible actions are N, S, E, W, and stick. The players' actions are executed in random order. If this sequence of actions causes the players to collide, then only the first moves. But if the player with the ball moves second, then the ball changes possession. If the player with the ball moves into a goal, then he scores +100 if it is in fact his own goal and the other player scores 100, or he scores 100 if it is the other player's goal and the other player scores +100. In either case, the game ends.

A. Implementation

The Soccer game was implemented with details from the previous section. A grid of 2x4 was used with cells indexed from 1 to 8. Each state of the game was represented by three factors: position of player A, position of player B, and possession of the ball, thus leading to a total of 112 states. The starting position was always set to the state s shown in Figure 1. This was done in order to make frequent updates to the Q value of the state(with player 1 taking action S and player 2 sticking) and also since neither player has any advantage from the state. However, possession of the ball goes to one or the other player at random(assumption drawn from [Lit94]). Another assumption that was made was that when a player takes an action that takes them out of the field -i.e., if a player moves N from the top row or S from the bottom row, the player stays in the same state and the possession of the ball remains the same. This assumption was made as there were no explicit mentions of penalty or change in possession of ball for invalid moves.

IV. Replication

A. Correlated-Q Learning

Utilitarian CE-Q, as shown in (7) was implemented in order to find optimal strategies for the game of soccer. Specifically, the q values and strategy at state s (shown in Figure 1) were analysed and plotted for replicating results from [GH03]. The game of soccer was played repeatedly for 1 million time steps with an initial alpha value of 0.2, alpha decay of 0.999995 and a minimum alpha value of 0.001 and actions were chosen at random for both players at each time step. These values were found empirically after trying different values and values that replicated the graphs from [GH03] closely were chosen as the final values and used for generating graphs presented in the following sections.

1) Implementation: uCE-Q was implemented as a linear programming problem in order to find a strategy that maximises the sum of the players' rewards. This was done with the help of following constraints:

$$p(a_i, a_j) \geq 0 \tag{10}$$

where $p(a_i, a_j)$ is the probability that the action $set(a_i, a_j)$is chosen from the set of joint actions of the two players.

$$\sum_{a_i, a_j \in \vec{a}} p(a_i, a_j) = 1 \tag{11}$$

$$\sum_{a_i, a_j \in \vec{a}} p(a_i, a_j) u_k(a_i, a_j) \geq \sum_{a_i, a_j \in \vec{a} - a_i} p(a_i, a_j) u_k(a_i -, a_j) \tag{12}$$

where k $\in$ (1,2). $u_k(a_i, a_j)$ denotes the payoff to the player k for player k choosing action a_i and the other player choosing action a_j. $u_k(a_i -, a_j)$ denotes the payoff to the player k for player k choosing any action other than a_i and the other player choosing action a_j.

The objective function was defined as the sum of the expected rewards of both players for each strategy and it was maximised subject to the constraints (9), (10) and (11).

2) Comparison of Results:

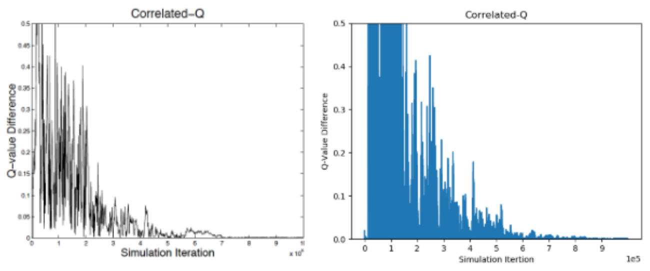
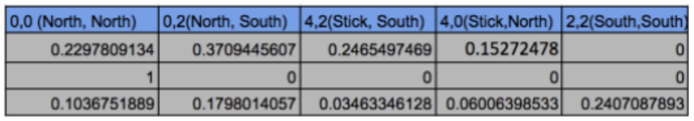


Figure 2. (left)Figure 3(a) of [GH03]. (right)Replication of 3(a)

As can be observed from Figure 2, the graph follows the same trend as the one presented in the paper. i.e., the q-values converge after 700000 time steps. However, between spikes, the difference of q values settles at 0 as opposed to non-zero values in the original graph. This can be explained by the fact that the q-value of the state-action pair: state s and player A taking action S and player B sticking was not updated at all the time steps. It can also be explained by the difference in methods by which the actions are chosen by both players at each time step. As mentioned earlier, actions were chosen uniformly at random for both players as it generated a graph most similar to the one presented in the original paper. Also, since it is important for all the state-action pairs to be visited, it helped to choose actions randomly rather than epsilon-greedy.

The non-zero difference between the spikes in the original graph might be because of the use of a strategy like showing the differences only when there is an update to the Q-values.

3) Policy:



0,0 (North, North)	0,2(North, South)	4,2(Stick, South)	4,0(Stick,North)	2,2(South,South)
0.2297809134	0.3709445607	0.2465497469	0.15272478	0
1	0	0	0	0
0.1036751889	0.1798014057	0.03463346128	0.06006398533	0.2407087893

Figure 3. Policies learnt during the last few time steps of training

Figure 3 shows the policies learnt by CEQ algorithm during the last few iterations. As mentioned in the original paper, the algorithm converges to non-deterministic policies, where the players randomise mainly between Sticking and heading South and the probabilities for rest of the joint actions are almost always 0. In addition to sticking, the agent also sometimes randomises between heading North and sticking. This is because of the assumption mentioned earlier - if a player takes action N from the top row or action S from the bottom row, the player stays in the same state and the possession of the ball remains the same. Similar to state s, the agent converges to non-deterministic policies for all states.

4) Problems Encountered: One of the major problems encountered during implementation of CEQ algorithm was the structuring of the problem as a linear program. After a steep learning curve(which in itself was a time-consuming process owing to the limited resources available for linear programming), the process of structuring the problem as a linear program took a considerable amount of time and required a lot of iterations to be implemented correctly in order to get the desired results.

Another major problem was tuning the hyper parameters - alpha, alpha decay and the strategy for selecting actions at each iteration. Since the original paper did not provide information regarding any of the hyper parameters or strategies, the entire range of acceptable values were to be tried for each of the parameters. For instance, the entire range of 0 to 1 for alpha and even wider range for alpha decay. Different values were passed to the parameters and the results were not as expected. However, [Lit94] provided some information regarding alpha decay and initial alpha values for successful convergence of Foe-Q learning and the same values were tried and resulted in a graph similar to the one presented in the original paper.

B. Foe-Q Learning

Foe-Q learning was implemented using the value function given by (4). As with CE-Q, the q values and strategy at state s (shown in Figure 1) were analysed and plotted for replicating results from[GH03]. The game of soccer was played repeatedly for 1 million time steps with an initial alpha value of 0.2, alpha decay of 0.999995 and a minimum alpha value of 0.001 and actions were chosen at random for both players at each time step. These values were found empirically after trying different values and values that replicated the graphs from [GH03] closely were chosen as the final values and used for generating graphs presented in the following sections.

1) Implementation: Foe-Q learning was implemented as a linear programming problem in order to find a mixed strategy as the game of soccer does not have deterministic equilibrium policies. The constraints set up are as follows: For all a $\in \vec{a}$ for player i,

$$p(a) \geq 0 \tag{13}$$

where p(a) is the probability of choosing an action a.

$$\sum_{a \in \vec{a}} p(a) = 1 \tag{14}$$

The objective function was defined as one that was lesser than or equal to the smallest expected reward. In other words, the minimum value that the player can expect, being in a state and assuming an adversarial opponent. Then the objective function was maximised subject to the constraints listed above.

2) Comparison of Results:

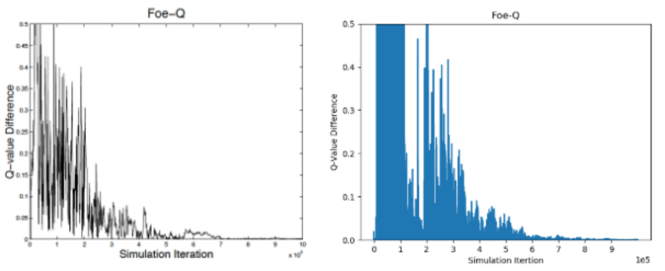
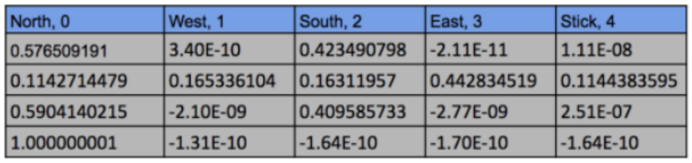


Figure 4. (left)Figure 3(b) of [GH03]. (right)Replication of 3(b)

As can be observed from Figure 3, the graph follows the same trend as the one presented in the paper. i.e., the q-values converge after 700000 time steps. However, like CEQ, the difference of q values settles at 0 as opposed to non-zero values in the original graph(for reasons cited under Correlated-Q learning section). Also, as in the original paper, the graphs for Foe-Q and CE-Q are similar although not the same. There are no major differences between the two i.e. the trends are just the same and in both cases Q-values converge at around the same time step. However, there are subtle differences in the magnitude of differences noticed at various time steps. This can be explained by the difference in sequence of actions taken randomly by the players at each time step. Perhaps, seeding the random number generator would have generated just the same plots but it was not tried due to time constraints.

3) Policy and Q values:



North, 0	West, 1	South, 2	East, 3	Stick, 4
0.576509191	3.40E-10	0.423490798	-2.11E-11	1.11E-08
0.1142714479	0.165336104	0.16311957	0.442834519	0.1144383595
0.5904140215	-2.10E-09	0.409585733	-2.77E-09	2.51E-07
1.000000001	-1.31E-10	-1.64E-10	-1.70E-10	-1.64E-10

Figure 5. Policies learnt for player 1 during the last few time steps of training

As seen in Figure 4, the Foe-Q algorithm converges to the policy specified in the original paper, which is just the same as the one obtained using CE-Q algorithm: the player randomizes between heading south and sticking. Here going North is equivalent to sticking as per the assumptions(mentioned in Section 3.A) made during implementation. But overall, the policies give weights to either action 2(south) or 0(North/sticking) or give equal weights to both. This is the same as the one explained in the original paper. Also, as mentioned in the original paper, the Q-values learnt using Foe-Q coincide exactly with the Q-values learnt using CE-Q and hence both algorithms converge to the same non-deterministic policy for all states including state s.

4) Problems Encountered: Similar to CEQ, one of the major problems encountered when implementing Foe-Q was the structuring of the problem as a linear program. However, unlike CEQ, Foe-Q was much easier and straightforward to implement as it has lesser constraints when compared to CEQ. Nevertheless, the problem of tuning the hyperparameters still persisted and took a considerable amount of time before finding out the correct values that led to graph similar to the original paper.

C. Friend-Q Learning

Friend-Q was implemented using the value function given by (5). [Lit01] studied friend-Q for co-ordination games where it is optimal for both players to maximise each other's reward. Hence Friend-Q is expected to converge to a deterministic policy for all states.

1) Implementation: The game of soccer was played repeatedly for 1 million time steps with an initial alpha value of 0.1, alpha decay of 0.9999954 and a minimum alpha value of 0.001. Epsilon was set as 1.0 initially with a decay rate of 0.999954. Actions were chosen at random for both players epsilon times and the rest of times actions were chosen so as to maximise the agent's reward in that state . These values were found empirically after trying different values and values that replicated the graphs from [GH03] closely were chosen as the final values and used for generating graphs presented in the following sections.

2) Comparison of Results:

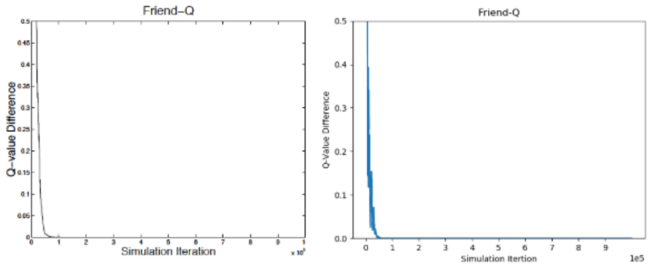


Figure 6. (left)Figure 3(c) of [GH03]. (right)Replication of 3(c)

As shown in Figure 4, the difference between q values converge to 0 between 0 and 100000 iterations. The curve in the original graph is smoother. This might be because of the use of a seed which influenced the sequence of random actions taken. However, as mentioned in the original paper, the algorithm converges to a deterministic policy for player A. i.e.For state s, the policy for player A converges to action 4(sticking), as the player assumes that from the current state, player B's immediate action will be to score a goal for player A. Similarly, the policy for player B converges to action 3(heading east), as the player expects the opponent to pass the ball towards its own goal.

3) Problems Encountered: Even though the implementation of Friend-Q algorithm was much easier when compared to both CEQ and Foe-Q, getting the graph for Friend-Q similar to the graph presented in the original paper involved a lot of challenges. For example, the graph presented in the original paper shows a continuous curve without any values of 0 before convergence meaning that the Q-value of the state action pair was updated at every time step. But this seems unlikely as at each time step, only the Q-values pertaining to the state the agent is in at that time step gets updated. Hence, in order to generate a graph similar to the one shown in [GH03], the difference was shown as non-negative only when there was a real update happening to the Q-value of state s. Otherwise the previous Q value was preserved.

Another issue was with the tuning of hyper parameters. Unlike CEQ and FoeQ, FriendQ did not produce the required graphs with alpha values mentioned earlier. In addition to alpha values, epsilon values had to be tuned as well in order to get the correct graph. A wide range of values had to be tried for each of the parameters before finding the ones that produced the graph most similar to [GH03].

D. Q-Learning

Q-Learning was implemented using the conventional q-learning algorithm given by equations (1) and (2), except that as mentioned in the original paper, the Q-learners compute Q-values for each of their own possible actions, ignoring their opponents' actions. And the actions that maximize Q(s,a) at each state describe the optimal policy π^*, given by:

$$\pi^*(s) \in arg \max_{a \in A(s)} Q^*(s,a) \tag{15}$$

1) Implementation: The game of soccer was played repeatedly for 1 million time steps with an initial alpha value of 0.4, alpha decay of 0.9999954 and a minimum alpha value of 0.001. Epsilon was set as 0.5 initially with a decay rate of 0.999954. Actions were chosen at random for both players epsilon times and the rest of times actions were chosen so as to maximise the agent's reward in that state . These values were found empirically after trying different values and values that replicated the graphs from [GH03] closely were chosen as the final values and used for generating graphs presented in the following sections.

2) Comparison of Results:

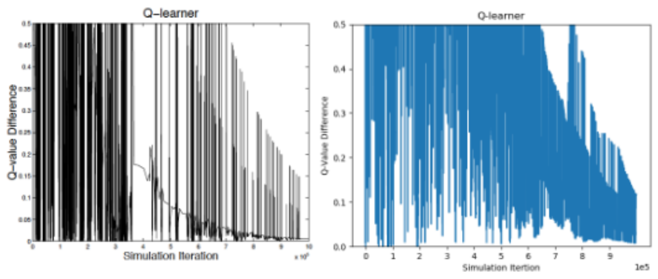


Figure 7. (left)Figure 3(d) of [GH03]. (right)Replication of 3(d)

As in the original paper, Q-learning does not converge. This is because the agent does not take in to account, the opponent's actions and each time the agent encounters a state, a different optimal action is found and hence the algorithm keeps changing policies. Also, Q-learning searches for deterministic policies but as mentioned earlier, the game of soccer does not have deterministic equilibrium policies. However, the magnitude of differences still decreases over time and this is because of the decay of the learning rate as the agent learns the game. Also, towards the end of 1 million time steps, the difference of Q-values settle at around 0.15, which is the same as presented in the original paper.

3) Problems Encountered: Just as Friend-Q, Q-learning did not produce the required graphs with alpha values mentioned earlier for Foe-Q or CEQ. In addition to alpha values, epsilon values had to be tuned as well in order to get the correct graph. A wide range of values had to be tried for each of the parameters before finding the ones that produced the graph most similar to [GH03].

References

[Lit94] Michael L. Littman. "Markov Games as a Framework for Multi-Agent Reinforcement Learning". In: *In Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994, pp. 157–163.

[Lit01] Michael L. Littman. "Friend-or-Foe Q-learning in General-Sum Games". In: *Proc. 18th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 2001, pp. 322–328.

[GH03] Amy Greenwald and Keith Hall. "Correlated-Q learning". In: *In AAAI Spring Symposium*. AAAI Press, 2003, pp. 242–249.