Correlated-Q Learning

Git Hash: 7facde3ed99bf324a77619e764f82418d448538d

Devanathan Ramachandran Seetharaman
CS 7642 - Reinforcement Learning
Georgia Institute of Technology
Atlanta,GA
rsdevanathan@gatech.edu

***Abstract*—This project document details the implementation, discussion, and analysis of various multi-agent Q-learning algorithms for soccer game example from the paper "Correlated-Q Learning" by Amy Greenwald and Keith Hall [1].**

I. Introduction

The objective of the project is to implement the soccer game experiment from [1], using Correlated-Q learning proposed by authors. The experiment compares Correlated-Q learning with three other learning algorithms, Q-Learning, Friend-Q Learning, and Foe-Q learning. The experiment compares the convergence using the plot of Q value difference for the first player. This document compares each of these plots for my implementation with the author's implementation plots and analyzes the causes for the discrepancies. Also, this document discusses the hyperparameters used, assumptions made and the challenges faced in the implementation of each of these algorithms.

II. Multi Agent Q-Learning

The soccer game implemented in this project is a multi-agent zero-sum game. A multi-agent environment considers more than one agent interacting with one another. The standard Q-Learning algorithm is mostly used for single-agent learning games. For multi-agent problems like a soccer game, which has a non-deterministic policy, the algorithm tries to maximize the reward of one of the players. Since the action of another player is completely random, it usually does not converge to the optimal policy. The standard Q-Learning is implemented for the soccer game and discussed in detail in section (V).

In [2], Littman proposes a Friend or Foe-Q(FFQ) learning algorithm that always converges to the optimal policy in the multi-agent zero-sum games. In these algorithms, an opponent player is considered either a Friend or Foe.

In the Friend-Q algorithm, the players intend to maximize the overall reward of the game and hence a coordination equilibrium exists between the players' actions and it is a maximization algorithm. The value function for the Friend-Q learning algorithm is given by

$$V = \max_{a_1 \in A_1, \alpha_2 \in A_2} Q_1 \left[s, a_1, a_2\right]$$

In the Foe-Q algorithm, the players intend to maximize their reward and minimize the opponent's reward. Foe-Q is a min-max algorithm and adversarial equilibrium exists between the players' actions. The value function for the Foe-Q algorithm is given below.

$$V = \max_{\pi \in \Pi I(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} \pi\left(a_1\right) Q_1\left[s, a_1, a_2\right]$$

In [1], authors propose a new algorithm called as "Correlated-Q" algorithm which is an extension and generalization of Littman's FFQ algorithm. The algorithm is explained in detail in the next section.

III. Correlated-Q Learning

In [1], Greenwald and Hall introduce correlation equilibrium(CE-Q) which generalizes the coordination equilibrium and adversarial equilibrium proposed by Littman in In [2]. For CE-Q learning, authors propose the below value function,

$$V_i(s) \in \mathrm{CE}_i\left(Q_1(s), \ldots, Q_n(s)\right)$$

The author introduces four variants of CE-Q, utilitarian, egalitarian, republican, and libertarian for equilibrium selection from multiple equilibria available. below are the objective function for each of the variant

- Utilitarian - maximize sum of player's rewards

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

- Egalitarian - maximize the minimum of player's rewards

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

- Republican - maximize the maximum of player's rewards

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

- Libertarian - maximize the maximum of individual player's rewards

$$\sigma^i \in \arg \max_{\sigma \in \mathrm{CE}} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

In general, the multi-agent learning can be expressed by the below algorithm as given by authors in [1].

Algorithm 1 Multi Agent Q-Learning

for $t = 1$ to T **do**
  Initialize $s, a_1, \ldots, a_n$ and $Q_1, \ldots, Q_n$
  simulate actions $a_1, \ldots, a_n$ in state s
  observe rewards $R_1, \ldots, R_n$ and next state s'
  for for $i = 1$ to n **do**
    $V_i(s') = f_i(Q_1(s'), \ldots, Q_n(s'))$
    $Q_i(s, \vec{a}) = (1-\alpha)Q_i(s, \vec{a}) + \alpha[(1-\gamma)R_i + \gamma V_i(s')]$
  end for
  agents choose actions $a'_1, \ldots, a'_n$
    $s = s', a_1 = a'_1, \ldots, a_n = a'_n$
  decay α according to S
end for

.

In the above algorithm γ, α, S, T are hyperparameters.

In this project, we implemented Utilitarian CE-Q along with friend-Q, Foe-Q, and Q-learning algorithms for the soccer game briefed below.

IV. Soccer Game

In this project, all the above multi-agent Q-Learning algorithms are implemented for a soccer game shown below which is a grid game. The game is a zero-sum game and it does not have deterministic equilibrium policies.

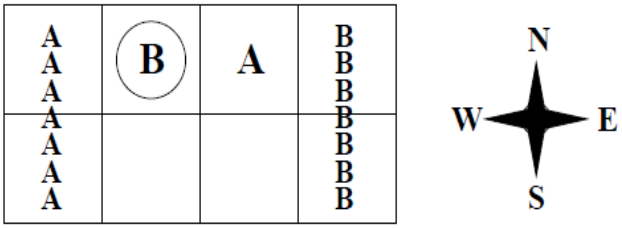

Fig. 1. Soccer Game

There are two players in the game, A and B and each of them can take 5 actions, move North, South, East, West, or Stick to the position. The ball is represented by the circle. The order of players' actions is randomly chosen. If the player's collided, only the first player moves and the second player sticks to the position. Whenever the player with the ball collides with the player without the ball, the first player loses the ball to the second. The player gets the reward of 100 if he scores the goal and the opposite player gets -100. If the player scores an own side goal he gets the negative reward of -100 and the other player gets the reward of 100.

Environment Implementation

I implemented the above soccer game environment from scratch to mirror the Open AI gym environment. The environment is implemented as 2*4 Grid and with 2 players A and B.Action space are created with 5 possible actions in the game. Observation space is created with 128 possible states(4*2 positions for player A, 4*2 positions for player B, 2 to indicate ball possession).

Each of the states is assigned with the state id and the value will contain the x and y coordinates of player A, x, and y coordinates of player B and the binary variable ball possession(0 for Player A and 1 for player B).

The step function for the environment is implemented to find the next state and reward given the current state and action. The order of player action is selected randomly for each step and collision rules are adjusted as per the order of player action. The reward is assigned to the players in this step if the goal and game are marked 'done' in this step if the ball reaches one of the goal columns.

The environment also provides the functions to reset the game, i,e, initialize the state and fetch the random action for the players when needed. All of these functions are implemented in such a way to mimic the Open AI gym.

The environment created is used in all the four experiments explained below.

V. Experiment 1 - Q-Learning

From the implementation perspective, Q- learning is the simplest of the four algorithms. For each episode, the state is initialized for the environment and the action for Player A is chosen using $\epsilon-greedy$ method i,e. random action is chosen with the probability of ϵ and max value action is chosen with the probability of (1-ϵ). Epsilon value is initialized as 1 and decayed until 0.001.

The action for Player B will always be random for the Q-Learning algorithm. Based on the action chosen and the current state, the next state and reward are calculated from the environment.

The Q table is initiated only for Player A with state-action(128*5). With the next state and reward, the Q value for the current state is updated using the below equation.

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

The key hyperparameters used were Steps(T) = 1M, Discount factor(γ) = 0.9 , Learning rate decay($alpha - Decay$) = 0.999995.Q-Value differences are recorded for the fixed state s(given in soccer diagram game) and the action A(south). The selection of s and A as well as other hyperparameter are explained in section (IX), Assumptions and challenges.

Below are the Q value difference plots for Q-Learning from my implementation and Greenwald and Hall's [1].

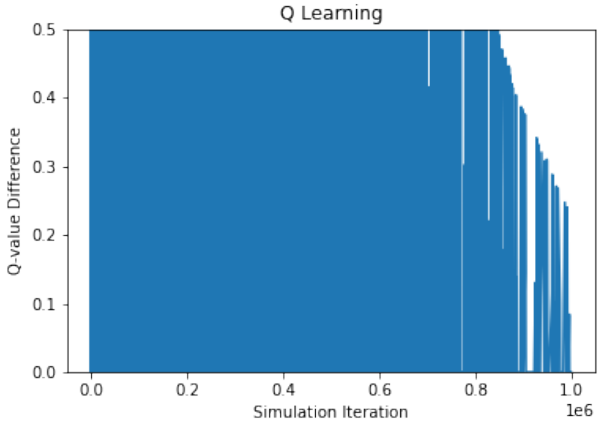

Fig. 2. Q-Learning - My Implementation

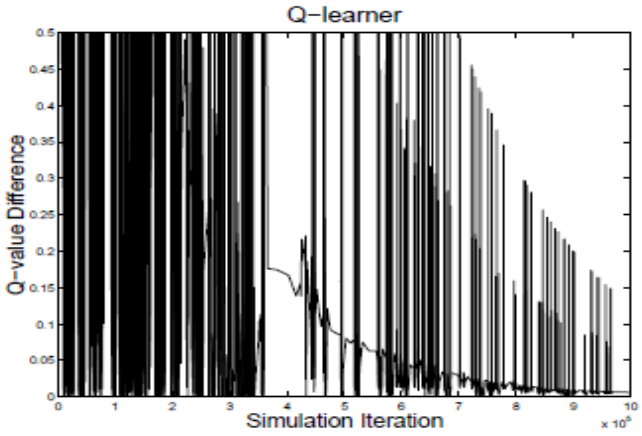

Fig. 3. Q-Learning -From Paper [1]

As expected, the Q-Learning does not converge due to the non-deterministic nature due to the random actions of player B. However the policy is quite different from [1]'s plot which is visible in the plots. Especially I could not replicate the wide gap's in the middle of the plot. The possible reasons could the different state s and action A for which the Q-value differences are recorded as it is not mentioned in the paper [1]. the hyperparameters could also be different and it is discussed in the Assumptions section.

VI. Experiment 2 - Friend Q-Learning

In Friend-Q learning, action pair is chosen for both the players, in such a way that it maximizes the overall reward. From an implementation standpoint, Friend-Q learning is implemented very similarly to the Q-Learning, the main difference being Q tables would have joint actions for both the players.

The current state is initialized similar to the Q-Learning but $\epsilon - greedy$ is used to select the actions for both the players. next state and reward are calculated from the environment using the current state and action chosen. The value, V is calculated by taking the maximum future value for the next state across all the actions.

Q table is initialized for both the players with the joint action(128*5*5) and the Q value for the current state is updated for both the players in every step using the reward from the environment and the Value V, calculated from the future rewards.

The key hyper parameters used were same as the one's used for Q-Learning, Steps(T) = 1M, Discount factor(γ) = 0.9 , Learning rate decay($alpha - Decay$) = 0.999995.Q value difference is recorded for state s, when Player 1 takes Action A(South) and Player 2 sticks to the current position as mentioned in [1]. Below are the plots from my implementation and [1].

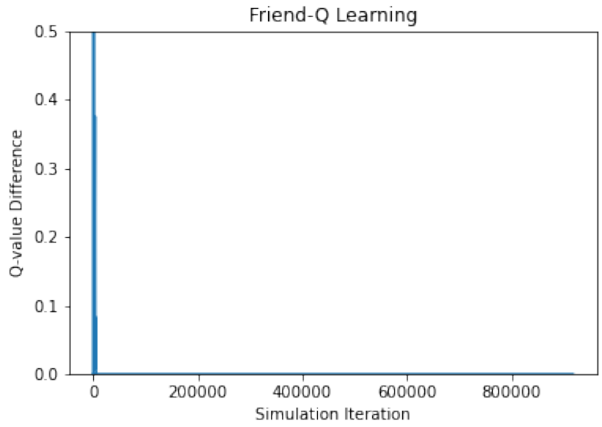

Fig. 4. Friend Q-Learning - My Implementation

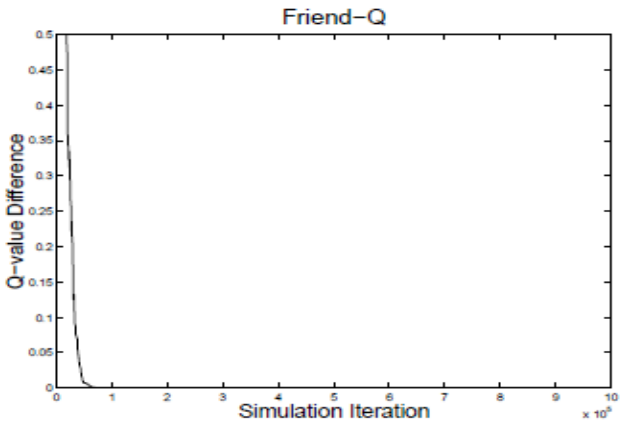

Fig. 5. Friend Q-Learning -From Paper [1]

The plot converges very quickly since the actions are taken in coordination and the plot looks close to the plot from [1]. The minor difference in shape could again be due to the hyperparameter selection and unclarity in state and action for which the differences are recorded.

VII. Experiment 3 - Foe Q-Learning

In Foe-Q learning, the players act with adversarial equilibrium, where the player tries to maximize their reward given the opponent's worst-case action.

The value function and policy for each player is calculated using linear programming and the "cvxopt" package from python is used for implementing linear programming and the "glpk" solver is used. The objective function for the LP is to maximize the rewards and the constraints would contain probability and rationality constraints for each action. Once the Value is calculated for each player, the Q table for each player is updated.

Again, the key hyper parameters used were Steps(T) = 1M, Discount factor(γ) = 0.9 , Learning rate decay($alpha-Decay$) = 0.999995.Q value difference is recorded for state s, Action A(South) for Player 1 and Player 2 sticks to the position as mentioned in [1]. Below are the plots from my implementation and [1].

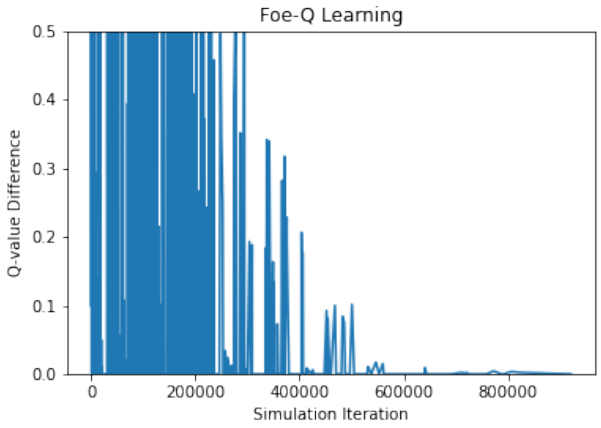

Fig. 6. Foe Q-Learning - My Implementation

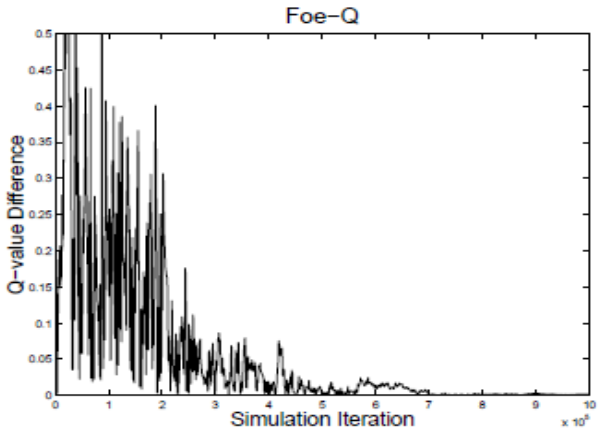

Fig. 7. Foe Q-Learning -From Paper [1]

The algorithm converges as expected from [1]. The convergence in my implementation happens around the same steps(600K-800K) as [1] however the shape of the curve is quite different. I could not replicate the policy from [1] even after experimenting with multiple hyperparameters and state and action for recording Q-Value differences.

VIII. Experiment 4 - Correlated Q-Learning

Correlated Q-Learning is implemented similar to Foe-Q learning using linear programming to calculate the value function and the policy. However, instead of calculating individual probability distribution for the policy, the joint probability is calculated using the linear programming for each pair of actions(5*5).

The objective function for the linear program is defined to maximize the sum of the rewards for both the players(utilitarian CE-Q) and the probability constraints(5*5) and the rationality constraints(5*4*2) are defined and "glpk" solver is used to solve the Lp and calculate the joint probabilities.

The joint probabilities are then used for calculating the value for player 1 and player 2 which is then used for updating the Q table for both the players. Below are the Q-Value difference plots from my implementation and [1].

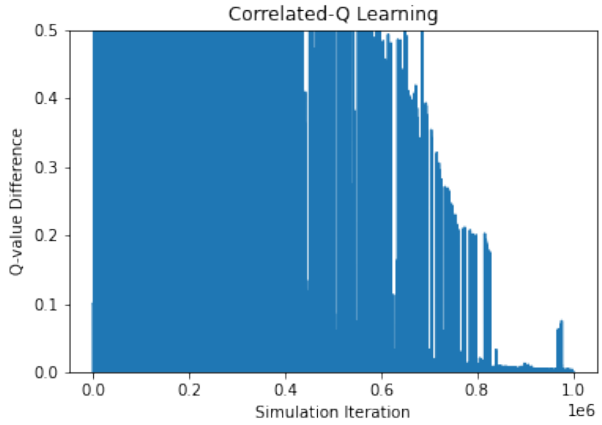

Fig. 8. Correlated Q-Learning - My Implementation

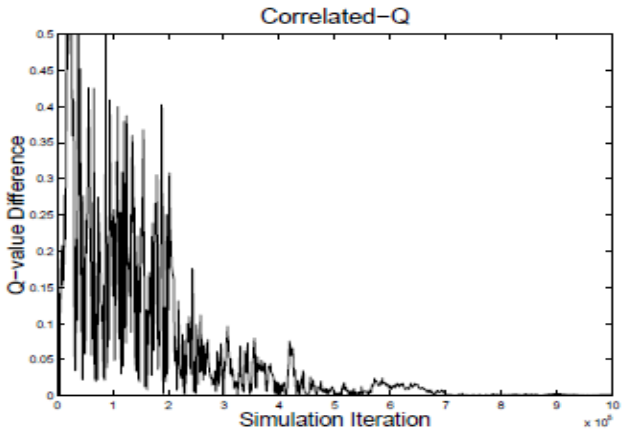

Fig. 9. Correlated Q-Learning -From Paper [1]

The plots look completely different and the Q-Value difference barely converges in my implementation. I tried multiple parameters and tried different LP packages(cvxpy,cvxopt) and different solvers within those packages but could not replicate the plot from [1]. I debugged potential errors in my code in both LP and Q value iterations but could not figure out the reason for the discrepancies. Some of the technical challenges faced in the implementation are explained in the below section.

IX. Assumptions and Challenges

One of the first assumptions made in the implementation is about the way Q value differences are recorded. For Q-Learning, I recorded the Q-Value differences for state s in each step. However, based on the plot from [1], it is not clear if it is recorded for each step as there are wide gap's in the plot. For the other three algorithms, the state s and Player 1's action A for which the differences are recorded is not clear in [1]. I recorded for the state shown in the soccer game grid diagram and the action "South", the only non-collision and non-goal action possible for player A in that state. however, it is not clear if the same is used by the authors.

Also, Player B sticking to the position might mean that Player B took the action "Stick", or took some other action but got stuck due to the collision. I chose the former assumption but not sure if it is the correct assumption.

For Foe-Q learning and Correlated-Q learning, I was not sure whether the action should be taken randomly or using the probability calculated from the linear programming. Taking the random actions did not converge properly and hence I took the action based on the probabilities from LP.

The time step is mentioned as 1 million, but it is not clear if is episodes are total steps across the episodes. I took the assumption of later as it produced a graph closer to [1].

The main challenge is in choosing the hyperparameters γ and α. I arbitrarily chose 0.9 after experimenting with 0.9,0.95 and 0.99. However, I could not do extensive tuning due to time constraints. For the learning rate, I initialized α as 1 and experimented with different decay factors, and found 0.999995 and 0.999997 to be producing graphs similar to [1].

The rationality constraints for the Correlation-Q algorithm are not clearly explained in the paper although the example is given for the chicken game. Multiple Piazza posts helped me in understanding the rationality constraints for player A and Player B in the correlation-Q algorithm.

Apart from these, the significant challenge faced is the performance of linear programming packages. Foe-Q and Correlation-Q algorithms took around 4 to 5 hours to run in both cvxpy and cvxopt packages, although cvxopt is marginally better. This made debugging and experimenting very tedious and time-consuming. In some scenarios, I had to implement the linear programming using cvxopt for the RPS game or chicken game to debug and use them on soccer games to save time.

X. Conclusion

In this project I implemented Correlation-Q learning along with Friend-Q, Goe-Q and standard Q learning algorithm for the multi-agent zero-sum game of soccer example from [1] and compared the Q-Value difference plots with the author's plot. The similarities and discrepancies were analyzed in this document.

References

[1] Amy Greenwald and Keith Hall. 2003. Correlated-Q learning. In Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML'03). AAAI Press, 242–249.

[2] Littman, M. (2001). Friend-or-Foe Q-learning in General-Sum Games. ICML.