Mehmet Oguz KAZKAYASI git hash: 6cfbd692fd60399db1cccf391d313c06d75bb62a

PROJECT-3 Robocup Correlated-Q Edition

1. The Soccer Game

The game described in the Amy Greenwald and Keith Hall paper and Littman (1994) is composed of 2 players (A and B). There is a 2 rows 4 colums environment and the leftmost column is goal for player A, which means 100 points for him and -100 for playerB. And the rightmost column is goal for playerB.

The game starts with the same position, A is at [0,2] and B is at [0,1] and one of them possesses the ball in the beginning randomly. A and B makes their decisions simultaneously, but they move in order randomly. Whenever the player who possesses the ball tries to moves to the grid that is occupied by the other player, that player loses the ball. When a collision happens, players can’t move.

This game is different than Project-2 game (Lunar Lander) because there is more than environment. Now, our agent’s decisions depend on the other player’s decisions and this fits in Game Theory context.

For Lunar Lander, we used an advanced version of Q-learning algorithm. Even though we didn’t see all states, we used the information we gathered from similar states on them. In this project, we don’t have that kind of requirement because we have a fairly small state space. But, the challenge is to play this game against other player, who is behaving rationally, too.

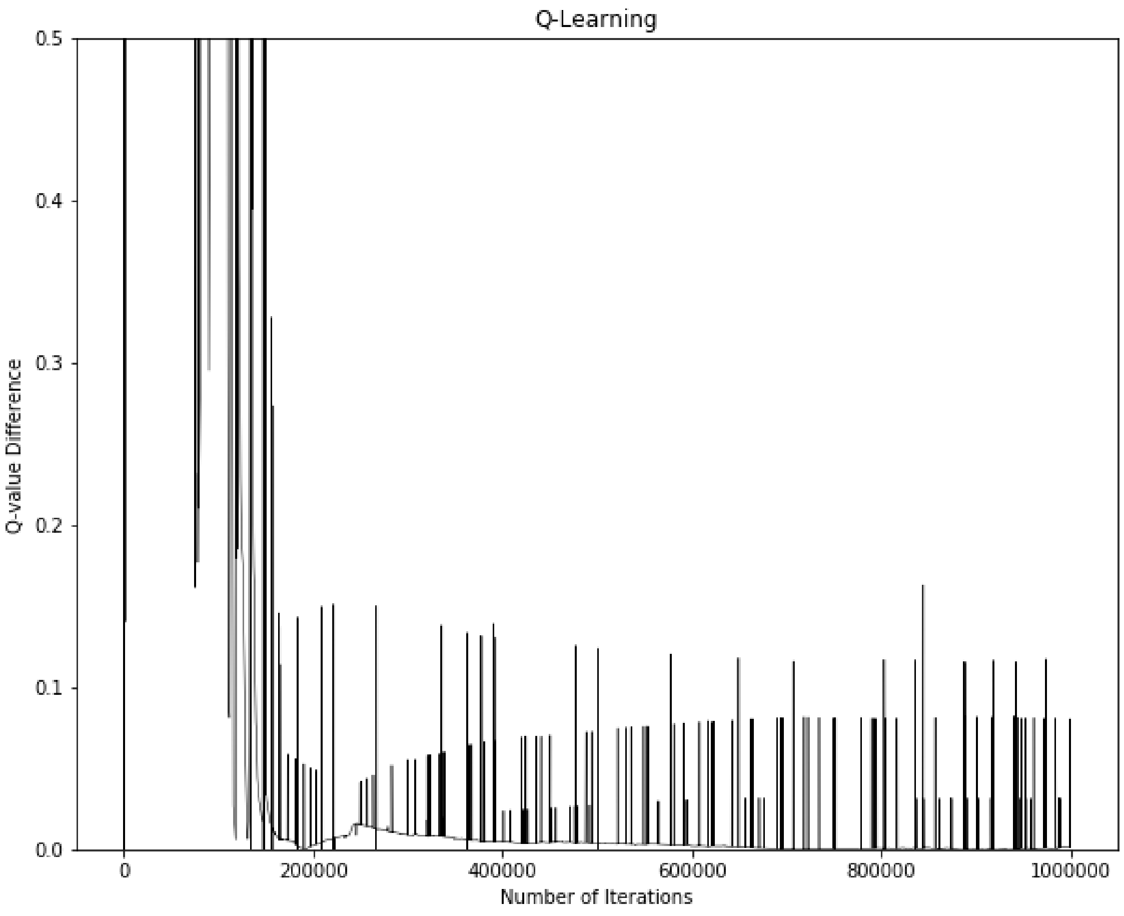
2. Algorithms

The project required 4 different algorithms to be used to play the soccer game.

a. Q-Learning

Q-Learning algorithm is easy to implement because the state space and action space was relaitvely small and 1M time-steps is more than needed for this environment. But, the problem for Q-Learning algorithm has a huge disadvantage in this game. It does not take other player's decisions into account. It just behaves the other player as a part of the environment. So, in my opinion, Q-Learning algorithm can't totally comprehend the game. As a result, it cannot converge to a value and even though sometimes it wanders around a value, it changes from game to game. Q-Learning algorithm is not enough to solve 2-player games.

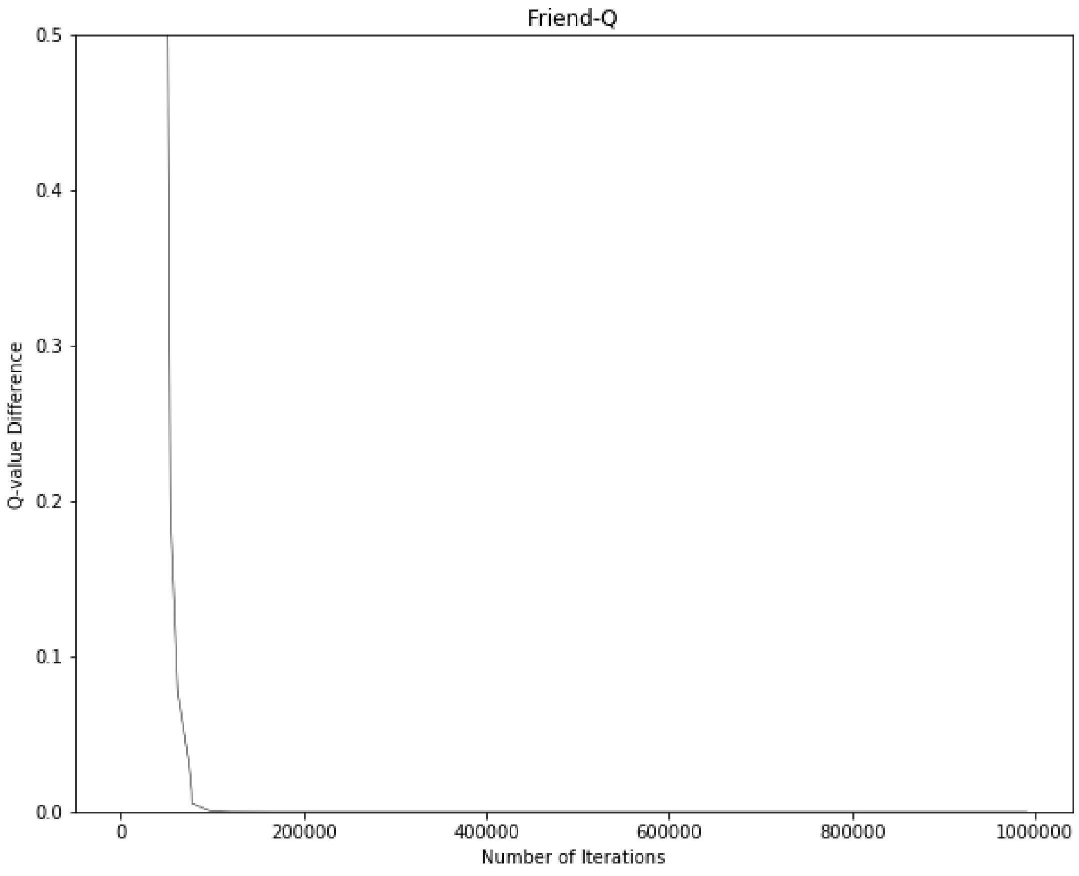
In other algotithms, we have an assumption on how other player is gonna behave. I think one of the reason that Q-Learning doesn't converge is the lack of that assumption. Even though it learns how the environment behaves, it is not enough without learning how the opponent will behave.



b. Friend-Q Algorithm

Friend-Q algorithm assumes that the opponent will serve for our agent, as well. This algorithms, as its name implies, won’t see the opponent as an opponent. And, it doesn’t set its behaviours according to an enemy. It will choose the maximum value of (state, action, opponent_action) as its Q value and it will decides its actions on that criteria. As expected, this algorithm converges into 100 for soccer game.

Also, as it can be seen in the figure below, this algorithm converges pretty quick. I think this is because it doesn’t think it will be challenged by the other player. So that, the 100 value it reaches around 100K iterations doesn’t change.



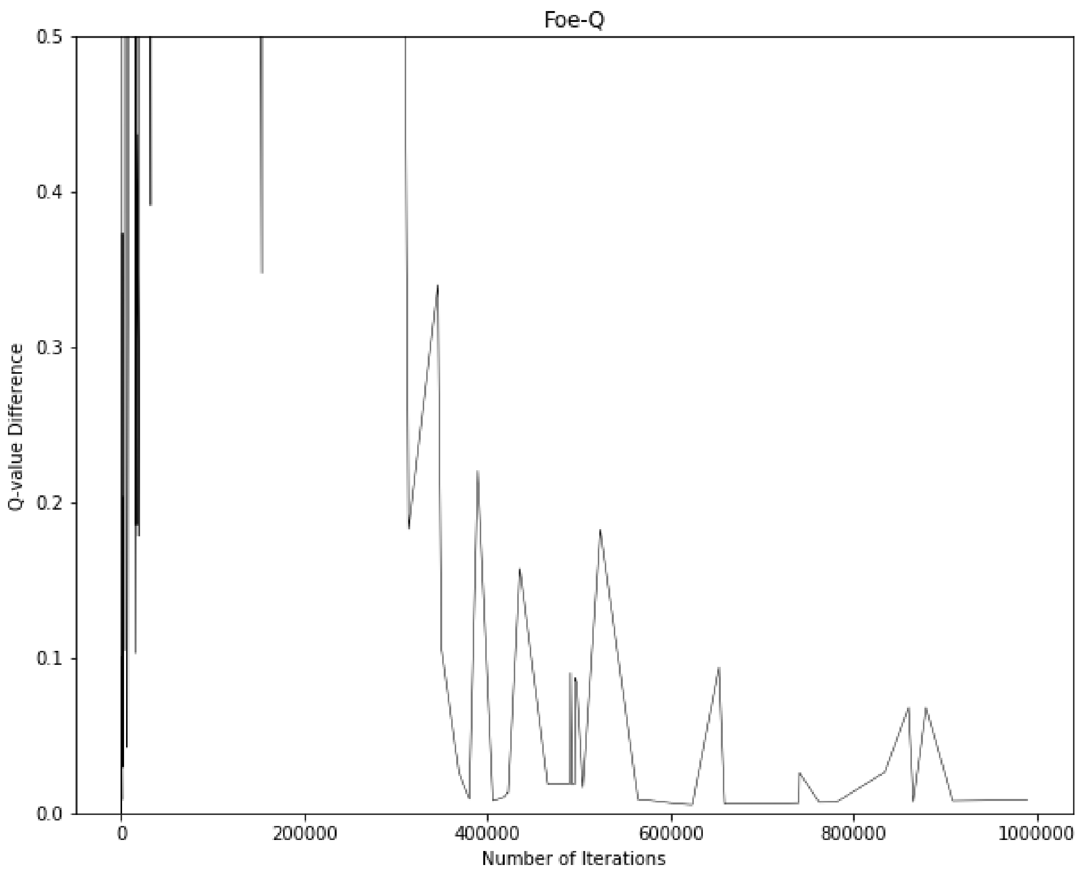
c. Foe-Q Algorithm

In this algorithm, the game is seen as a zero-sum game (which actually is) by the agents.The agent assumes that the opponent will try its best to reduce the agent's points. This is challenging comparing to Friend-Q algorithm and I think this causes some unexpected behaviours from the opponent even after 800K episodes. This algorithm converges much slower.

This is the first algorithm that required a Linear Programming solution. The solution was very similar to Rock-Scissors-Paper game solution, which is also a zero-sum game.

In my solution, I only kept track of the error values when the Q value (state:s, action:S, opponent_action:stick) is changed. I can't see any other was to keep track of the error. Because of that my graphs have less data points comparing to the paper's graphs, visually.

This can be caused by a lower value of randomness comparing to the paper. It can be seen that in my implementation (state:s, action:S, opponent_action:stick) is not experienced too many times. When I set the randomness value higher, this caused divergence. So, I kept it that way.



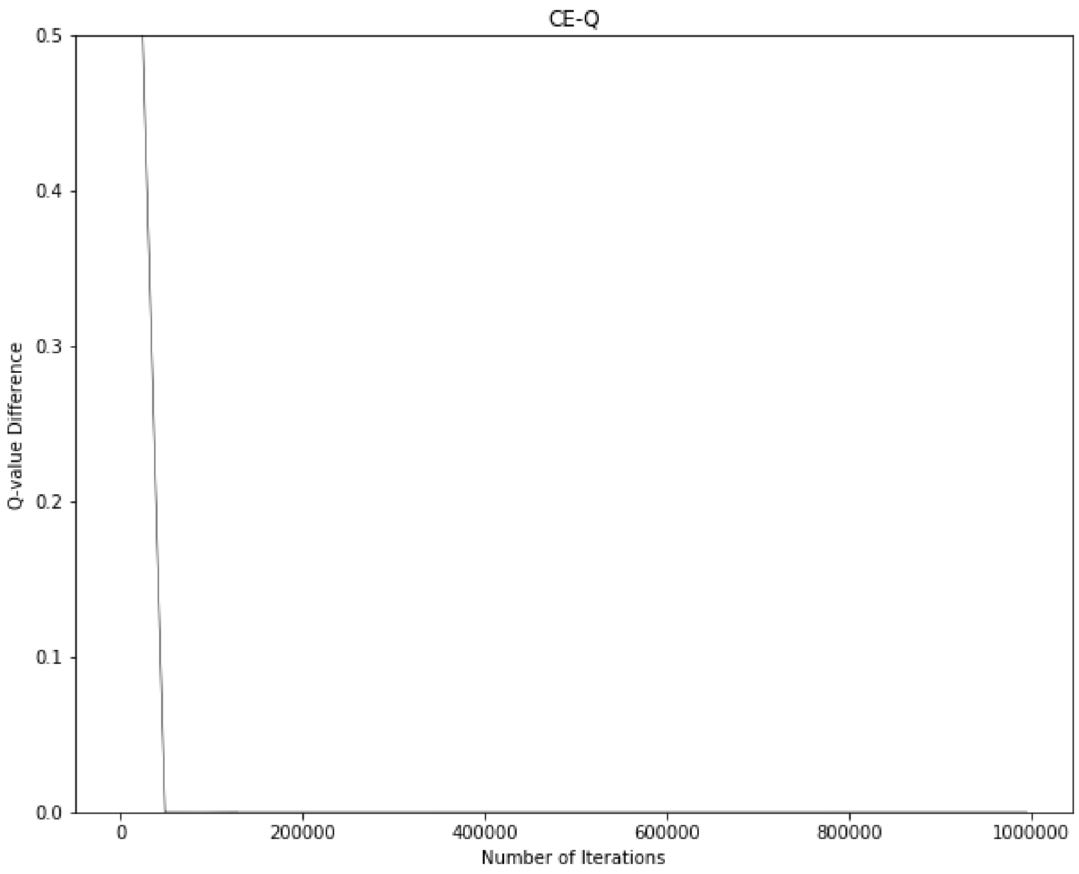
d. Utiliterean Correlated Q-Learning (uCE-Q) Algorithm

Utiliterean Correlated Q-Learning wants to maximize the sum of the two players' rewards. When this is used by both agents, it can result in some good coordination. But, the soccer game described in the paper is not such a case because it is zero sum. So, it is obvious that the sum of two agents' rewards will add up to 0.

This algorihtm needed the most trials for hyperparameters. I think in this case, it is difficult to converge for uCE-Q (assuming I implemented correctly). This algorithms requires two Q-tables that are used at the same time for players and the LP solution is similar to Chicken game solution.

Setting the constraints from matrices was the toughest part for this one and since my solution solves the examples at Piazza, I assumed it is doing good for the soccer game, as well.

On the other hand, my graph doesn't resemble the one in paper, at all. I think there must be something wrong with my implementation since this can't be explained by hyperparameters. Also, I have seen that the q value goes way above 100 using some different hyperparameters. I think this is also an indicator of wrong implementation.


CE-Q
Q-value Difference
Number of Iterations