Emily Broom

Git Hash: 0cf37d6312a426efc9cb9213f70f0fb66cd3f16b

Soccer with Correlated Q Learning

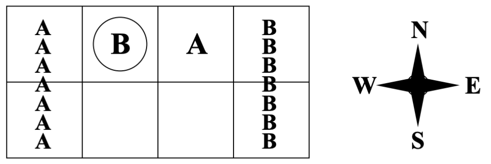

Figure 1

I. Soccer

The image above shows the soccer game field. The pitch is 8 cells, 4 columns with 2 rows with the first column being the goal for A and the last column being the goal for B. Greenwald-Hall did not specify the initial conditions for this game, so I assumed the initial conditions to be as seen in figure 1. A starts at cell 3, B at cell 2. B starts with possession of the ball. There could be other implementations of the same game where that A and B could be randomly placed on the pitch or that ball control could be randomly given to one of the two players, however for simplicity I chose this static initial state. The actions that A and B can take are N, S, E, W, and X. X here means to stick or not move.

The goal of the game for each player is for them to score. If B carries the ball to their goal on the right, they score 100 points while A loses 100. The reverse happens if A scores in their goal on the left. This is a zero-sum game. Both players chose actions simultaneously, but the order in which they occur is random. If A goes first and collides into B, nothing changes. However, if B goes first and collides into A, they both stay in their same position, but A now has control of the ball. "Possession of the ball goes to the stationary player and the move does not take place" [2]. If there is a set of moves that end with a collision (say A is at cell 4 and B is at cell 2, A moves W and B moves E with A moving first), then only the first player moves and only if that cell is unoccupied. In my example, A would move to cell 3, B would stay at cell 2, but possession changes from B to A. B tried to move, couldn't, and lost possession because of the previous rules.

This is the logic I used to recreate the soccer game. A small callout is that the rules, due to being written in natural language, were a little ambiguous and could be interpreted different ways. In the previous example maybe Greenwald or Littman meant the action isn't completed at all and B stays with possession if A collides with B before B collides with A.

II. Overview of Algorithms

All equations given as figures come from the Greenwald-Hall paper. It is assumed from previous assignments and projects that the reader understands Q learning and this paper is to explain a few different variations of Q learning.

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

Figure 2: Bellman Equation for Multiagent Markov Games

Figure 2 shows the Bellman Equation slightly modified for multiagent Markov Games. Different algorithms based from Q-learning come from different value functions. The reason for having different value functions is because the deterministic Q-learning policy for when all agents maximize their own reward may not exist. In fact, this deterministic policy often doesn't exist like in the Rock-Paper-Scissors game in the Greenwald paper.

A. Q-Learning

Greenwald starts with traditional Q-learning. In this algorithm both players try to find deterministic optimal policies, only thinking of their own actions and not considering the other player's actions.

This algorithm never converges for this game. For the scenario given in figure 1, say B thinks it's found an optimal policy: going south to get around player A. This may work a couple times and then A realizes that its optimal policy is to also go south to block player B. Player B readjusts its policy to stick while player A goes south and the whole cycle begins again. The policy is always changing, and Q-learning cannot decide on the best policy, because there is none when the algorithm is nondeterministic. This is like in Rock-Paper-Scissors (RPS) when you find out your friend's game plan. You change yours accordingly and then so should your friend. The optimal move is not deterministic.

B. Friend-Q

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

Figure 3: Value Function for Friend-Q

Friend-Q is a version of Q-learning that both players maximize the same reward and is more suitable for coordination games. The value function is given in Figure 3. Both players maximize on the same Q table that has the joint actions of both players.

The best actions for the state in Figure 1 would be for B to go south and A to stick according to this algorithm. Intuitively, you might think it better if A goes south and B goes east but remember that actions are ordered randomly and if B went first and went east, it would lose the ball to A. B has no chance of giving the ball to A if it goes south and can easily score. A lets B score because it believes they are both working together.

This algorithm for the soccer game converges quite fast. However, it converges to the wrong policies, because this is not a cooperative game. We want for both players to score roughly the same as they are both trying to win.

Comparing again to RPS, this is like figuring out your friend's game plan but then allowing them to win as you think you are also winning.

C. Foe-Q

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

Figure 4: Value Function for Foe-Q

Foe-Q's value function is given in figure 4 where $\sum_i(s)$ is the probabilistic action space of player i at state s. This value function is specifically for zero sum games. Player A is trying to get the maximum reward knowing that B is going to try and minimize A's reward. This can be done using one Q table as the maximum Q-value for B is simply the opposite of the minimum Q-value for A.

For the soccer game, this created probabilistic optimal policies. From the state indicated from figure 1, both players know that either south or sticking is the best policy depending on what the other player does. With 50% probability, B choses to go either south or stick. This probabilistic policy causes the Q values to converge.

How Foe-Q actually finds these values is through linear programming. A has to find the value that B would pick that would minimize A's reward, given that B knows A's Q-table. We wind up with a series of linear constraints. A has 5 actions, all of which have to have a probability >= 0. All probabilities must add up to 1. If A picks an action, the sum of all the policies for each action B could pick must be >= 0 where all the coefficients for the policies are from A's Q table, shown below:

$$\sum_{j \in ACT(B)} Q[i,j]\pi_{ij} \geq 0$$

Where i is the action A took, j is an element of the set of the actions of B (N, S, E, W, X). J here is indexed properly (N = 0, S=1, etc.). Each action A can take is a constraint equation (given above), so there will be five more constraints.

There are 11 constraints in total, 1 of which is an equality constraint. A, knowing that B is going to minimize the reward A can get (also maximizing their own reward since this is a zero-sum game), can get the action that B will choose due to linear programming. A then can maximize its own action knowing this information.

D. *Correlated Q-Learning*

Correlated Q-Learning is used to find correlated equilibrium (CE). Different from a Nash equilibrium, CE allows for the possibility of dependencies in agents' randomizations. A CE is a probability distribution over the joint action space and all agents optimize with respect to the other players' probabilities [1].

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

Figure 5: Value function for CE Learning

Figure 5 shows the value function for CE learning where Q_i is the i-th player's reward in a correlated equilibrium. Table 1 from Greenwald's paper (2003) shows the algorithm for multiagent Q-learning. In this algorithm, there is a function that will be used in the value function with all the Q_i's as input. The soccer game was only concerned with one of the four functions that Greenwald presented, which is the utilitarian correlated q-learning algorithm.

There can be multiple CE in a problem, so we have to pick a function to choose one of these equilibria. The utilitarian selection function is given in figure 6 which maximizes the sum of the players' rewards.

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

Figure 6: Utilitarian CE Selection Function

Going back to the value function, how one finds the correlated equilibria is by using linear programming (LP). This is unlike a Nash equilibria which cannot be found using LP. There are many constraints for this problem. All joint action pairs must have a policy that is >= 0. This is 25 constraints. All policies must sum to equal 1. The other constraints for player A follow as so:

$$\sum_{j \in ACT(B)} Q[i,j]\pi_{ij} - \sum_{j \in ACT(B)} Q[i_2,j]\pi_{ij} \geq 0$$

i and i_2 are two actions that A can take, j is an action from the set of actions that B can take. The policies when action is i, using the coefficients from the Q table for action i and j, must be greater than the policies when the action is i, using the coefficients from the Q table for action i_2 and j. Subtracting the two gives the equation shown above.

This may be a little confusing. What this means is that if you look at a table, take the coefficients from the first row, R1 and the coefficients of any other row, R2. The policies of the first row times R1 subtracted from the same policies times R2 is greater than or equal to 0. This is exactly how Greenwald creates the constraints in the "Chicken" problem. This creates 20 constraints for the Q table for A and 20 constraints for the Q table for B.

Maximizing on these constraints, we sum the results of the policies times the Q table as in figure 6 to maximize the sum of the players' rewards. Like all the other algorithms, we update the Q table as normal, using this sum as the maximum expected future reward.

III. RESULTS

The graphs given in figures 7-10 show the absolute value of the difference between the Q value at the state represented in figure 1 with action A being south and action B being stick from iteration i to i+1. This value is used to show convergence. As time increases, the absolute difference between the Q values at that state should decrease with the limit equaling 0 as time goes to infinity.

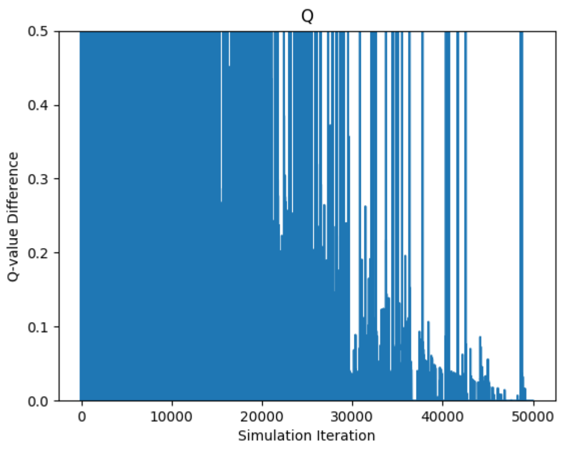

Figure 7: Q-Learning

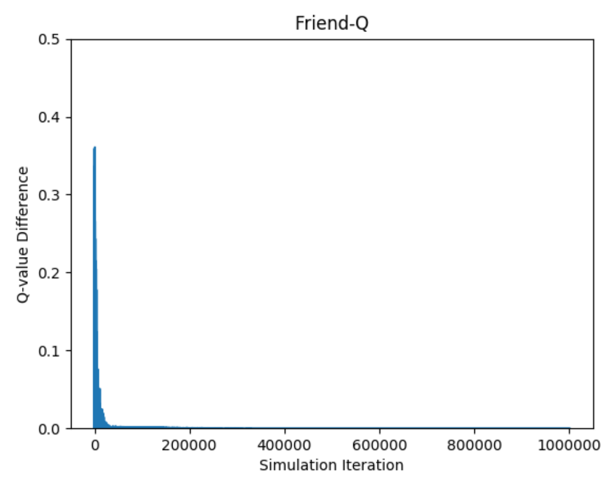

Figure 8: Friend-Q

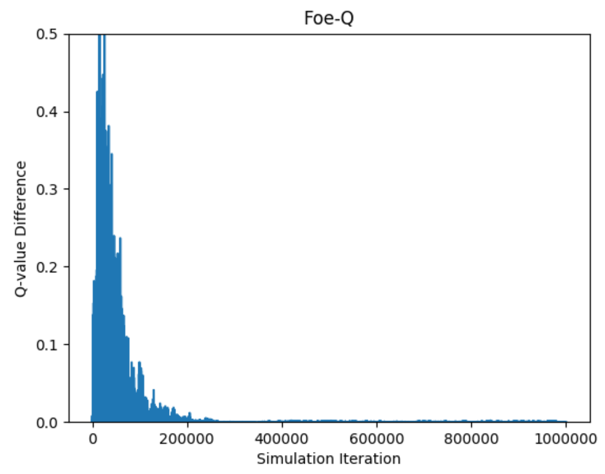

Figure 9: Foe-Q

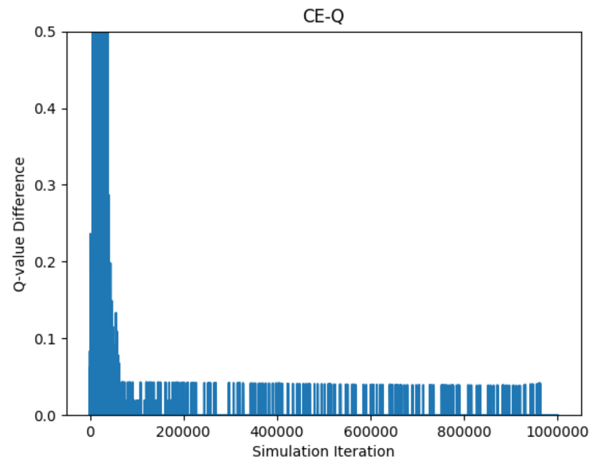

Figure 10: CE-Q

A. Q-Learning

Due to the fact that Q learning doesn't converge and there is no deterministic policy for this problem, the game ran slower and slower as time went on. For that reason, I stopped the game at 50000 instead of 1000000. However, figure 7 shows pretty clearly that Q learning does not converge, the same outcome that Greenwald's paper had. The downward trend here is the same, which is that the differences in Q are getting smaller due to alpha going to 0.001. I started alpha at 0.9 and epsilon at 0.99.

B. Friend-Q

Friend-Q has very similar results to that of Greenwald's in figure 8. As mentioned before, it converges very fast (although not to the policy that would be desired for this game). The alpha had to be taken to be very small to start with, 0.03, or else the graph looked like a small spike at 0 with no other points. I wanted the algorithm to converge not quite as quickly so the trend could be shown, so a low alpha value worked.

C. Foe-Q

Foe-Q also had very similar results to Greenwald's. in figure 9 It does not converge as fast as Friend-Q, but it does converge. It also converges to the correct, nondeterministic optimal policies explained earlier in the *Foe-Q* section of *Algorithms*. The starting alpha value here was also small, 0.08, but it was not as small as Friend-Q. The same behavior happens with a larger alpha value, where the trend of the chart isn't seen. Therefore, a small alpha value had to be taken in order to show the behavior.

D. Correlated-Q

Correlated-Q should look very similar to Foe-Q. This is because they converge to the same solution and the Q-values learned by both algorithms should be identical in theory.

My CE-Q graph in figure 10 looks similar to my Foe-Q graph, but with some interesting end behavior. The difference between Q values at the end is very small, less than 0.05, however it's not infinitesimal at the end of the trials like Foe-Q. This tells me that there is probably a small bug in my code (that I have yet to find) perhaps a rounding error or something similar that would cause the Q values to be off by about 0.04.

However, the idea is there. CE-Q should behave like Foe-Q in this experiment with the optimal policy for the state in figure 1 is for player B to switch between south and sticking and it does this probabilistically. As shown in the graph, CE-Q starts to converge at the same time as Foe-Q.

I tried altering starting alpha states, the initial Q table, the decay rate of alpha. However, a downside of CE-Q is that it is slower than the other algorithms mentioned. Linear programming is a polynomial algorithm and this CE-Q implementation had 66 constraints. If given more time I'm sure I would have found the bug and been able to recreate a better graph. However, it took several hours to run one whole experiment.

IV. Implementation Details

The algorithms were solved using Python 3 using the cvxopt module for the linear programming solver. The environment for the game was taken from [3], as topic 826 in Piazza said we were allowed to use others' environments.

CE-Q, Friend-Q, and Foe-Q were all off-policy with alpha going to 0.001. The policy used to train these algorithms was simply randomly choosing actions, taking full advantage of exploration. Q-learning was on-policy with an ε-greedy policy with ε going to 0.001. The Q-learning algorithm had alpha starting at 0.9 and epsilon at 0.99, multiplied by 0.9999 every iteration until they were both at

0.001. Friend-Q started with an alpha of 0.03, Foe-Q started with alpha of 0.08, and CE-Q started with an alpha of 0.8.

Each experiment was run 1,000,000 times with the exception of Q-learning which was only ran 50000 times due to time.

V. Problems and Pitfalls

I struggled a lot at first with the rules of the game. If A runs into B and then B runs into A, does B lose the ball or is the entire move canceled since there is a collision? What happens if the two players want to move into the same space? What if A collides into B but B wants to go south? Is the move still cancelled? I took what I thought was the best answer to these, which is that if there is a collision, the move afterward still happens or if the second move cannot happen for another collision, possession will change accordingly (as in, if the second move is the player with the ball colliding with the player who moved first, the possession switches). However, it would be reasonable for others to make different judgements on these cases since the paper was not extremely clear.

Second was understanding the constraints. How Greenwald went from the table of rewards in "Chicken" to the correlated equilibria constraints was not obvious. Thankfully there was an office hours session that helped a lot to understand this logic. Even after I understood how he coefficients were picked for the example used, the Bayesian logic was not explained well from how they went from $\pi(L|T)*\pi(T)$ to π_{TL}. This part of the assignment was a huge time sink.

Thirdly was just getting used to cvxopt and how it worked. I have not used a linear programming module before and there was not much documentation or examples on how it worked. It was also hard to know if the answer was right until the program had finished running all the trials.

Lastly was getting the alpha values. Foe-Q and Friend-Q have very low starting alphas because otherwise the graph is very unusable since they both converge so early. There was a lot of tweaking to the alpha values as the original alpha values were not given in the Greenwald paper. Thankfully they did give a couple of the other hyperparameters like gamma and the limit for alpha and epsilon.

VI. References

[1] Greenwald, A. and Hall, K. (2003) *Correlated-Q Learning*

[2] Littman, M. (1994) *Markov games as a framework for multi-agent reinforcement learning.*

[3] Akash29 *Soccer Simulator.* https://github.com/akash29/Soccer/tree/master/Environment