Correlated-Q Learning

Mikey Ling
Georgia Institute of Technology
mling7@gatech.edu
https://github.gatech.edu/mling7/CS7642_Project3

Abstract

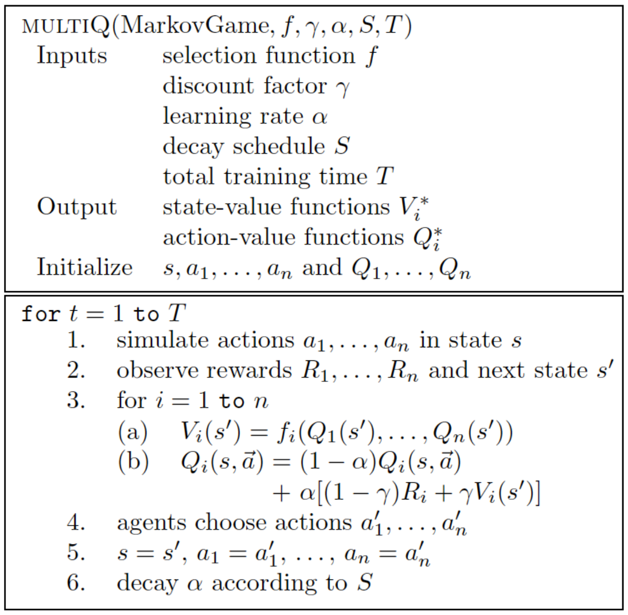
The purpose of this project is to attempt to reproduce the results of four different Q-Learning algorithms when playing a simple grid game, as featured in Amy Greenwald and Keith Hall's paper, "Correlated Q-Learning". Correlated-Q (CE-Q) learning is a multiagent Q-learning algorithm that focuses on the correlated equilibrium (CE) concept to converge to a solution. In addition to CE-Q learning, the paper also features regular Q-Learning, Friend Q-Learning, and Foe Q-Learning to use as a comparison benchmark.

I. Introduction

Thus far, all of the reinforcement learning problems the course has featured have been single-agent problems. A very interesting extension of reinforcement learning is multi-agent applications. We determine a solution has "converged" when the Nash Equilibrium policy has been reached. The Nash Equilibrium is defined a solution agents arrive at where no single player can increase its expected utility by changing its policy. Littman took the concept of Nash Equilibria and further extrapolated: adversarial and coordination. Adversarial equilibria are solutions where agents attempt to maximize their own utility while minimizing their opponents'. On the other hand, coordination equilibria are solutions where the agents work together to achieve a maximum utility for all players throughout the space-action set. In this project, the performance of both types of equilibria will be visualized and analyzed for the sake of comparison.

II. Correlated, Friend, Foe, and Regular Q-Learning

Greenwald's paper does an excellent job describing how the four-featured learning algorithms differ in the following figure:



```
MULTIQ(MarkovGame, $f, \gamma, \alpha, S, T$)
  Inputs      selection function $f$
              discount factor $\gamma$
              learning rate $\alpha$
              decay schedule $S$
              total training time $T$
  Output      state-value functions $V_i^*$
              action-value functions $Q_i^*$
  Initialize  $s, a_1, \ldots, a_n$ and $Q_1, \ldots, Q_n$

for $t = 1$ to $T$
    1. simulate actions $a_1, \ldots, a_n$ in state $s$
    2. observe rewards $R_1, \ldots, R_n$ and next state $s'$
    3. for $i = 1$ to $n$
        (a) $V_i(s') = f_i(Q_1(s'), \ldots, Q_n(s'))$
        (b) $Q_i(s, \vec{a}) = (1 - \alpha)Q_i(s, \vec{a})$
                $+ \alpha[(1 - \gamma)R_i + \gamma V_i(s')]$
    4. agents choose actions $a'_1, \ldots, a'_n$
    5. $s = s'$, $a_1 = a'_1$, $\ldots$, $a_n = a'_n$
    6. decay $\alpha$ according to $S$
```

Figure 1 Template for Multiagent Q Learning

Figure 1 features a generic formulation for multiagent Q-Learning. The only thing that differs between the four-featured algorithms is the "selection function" *f*, which computes the value function *V*, given the matrix-vector Q=(Q_1, ….Q_n). Littman's Friend or Foe Q computes V according to the two following equations, respectively:

$$\text{Nash}_1(s, Q_1, Q_2) = \max_{a_1 \in A_1, a_2 \in A_2} Q_1[s, a_1, a_2]$$

Figure 2 Friend Q Selection Function

$$\text{Nash}_1(s, Q_1, Q_2) = \max_{\pi \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} \pi(a_1) Q_1[s, a_1, a_2]$$

Figure 3 Foe Q Selection Function

The Correlated-Q selection function was actually broken down into four sub-functions: utilitarian, egalitarian, republican, and libertarian. For this project, we focused on utilitarian:

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

So, when writing the software to implement these four algorithms (regular Q-Learning's selection function was not shown), the code stemming from steps 1,2,4,5, and 6 were all *identical*. The only code that differed was the code relating each algorithm's selection function, *f.*

III. Soccer Game

We tested the performance of the two-player, zero-sum Markov Decision Problem soccer game mentioned in Greenwald and Hall's paper. The soccer field state-space is visualized in the following figure:

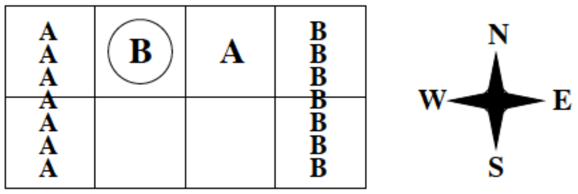


Figure 4 Soccer Game State Example

The athlete (A or B) with the circle around it has possession of the ball, and the first and fourth columns represent the respective athlete's target goal. If the athlete with possession of the ball enters its target goal, a reward of +100 is received. If the athlete with possession of the ball enters its *opponent's* target goal, a reward of -100 is received. Once the ball enters any goal area at all, the game is over (it can be thought of as: if the reward is *not* 0, the game is over). In all other states, athletes get a reward of 0.

The players' actions are chosen randomly (North, East, South, West, or Stick) and simultaneously. The order in which the players perform their actions is also randomized (50% athlete A, 50% athlete B). If the player without the ball moves into the player with the ball, he cannot. But if the player with the ball moves into the player without the ball, a change of possession occurs. This form of the game is featured in Littman's paper on multiagent Q Learning.

IV. Results

Standard Q-Learning:

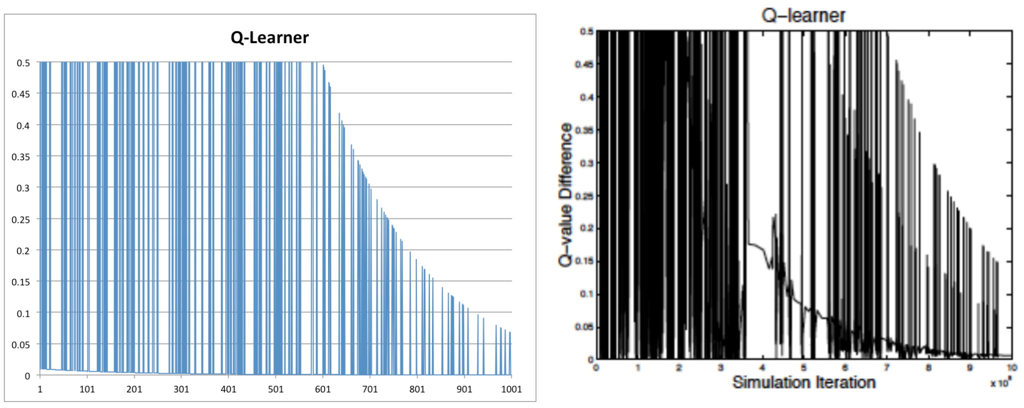


Figure 5 Experimental Results (left) vs Greenwald's Results (right) for Standard Q-Learning

The results for our Q-Learner compared to Greenwald's are fairly similar. One will notice the "gap" of noise in Greenwald's graph around 4x10^5 iterations that is *missing* from ours. This could potentially be the cause of our agents choosing different actions throughout the experiment because the nature of the experiment requires agents to *randomly* choose actions. This difference in action sequence could have caused our Q tables to differ at the same number of iteraitons.

Friend Q-Learning:

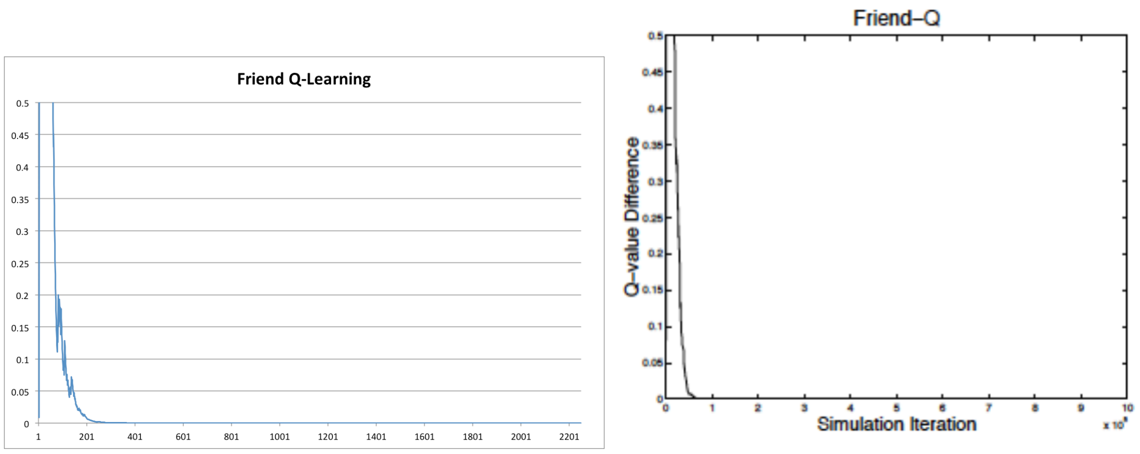


Figure 6 Experimental Results (left) vs Greenwald's Results (right) for Friend Q-Learning

The Friend Q-Learning graphs are also very similar; void of the slight noise our graph contains. Again, the noise could be the result of different actions being taken; however, we are very pleased with our results. The reason this algorithm converges so quickly is because Athlete A assumes Athlete B will score *for* him because, by definition of the algorithm, athletes assume the opponent is working to maximize their own rewards.

Foe Q-Learning:

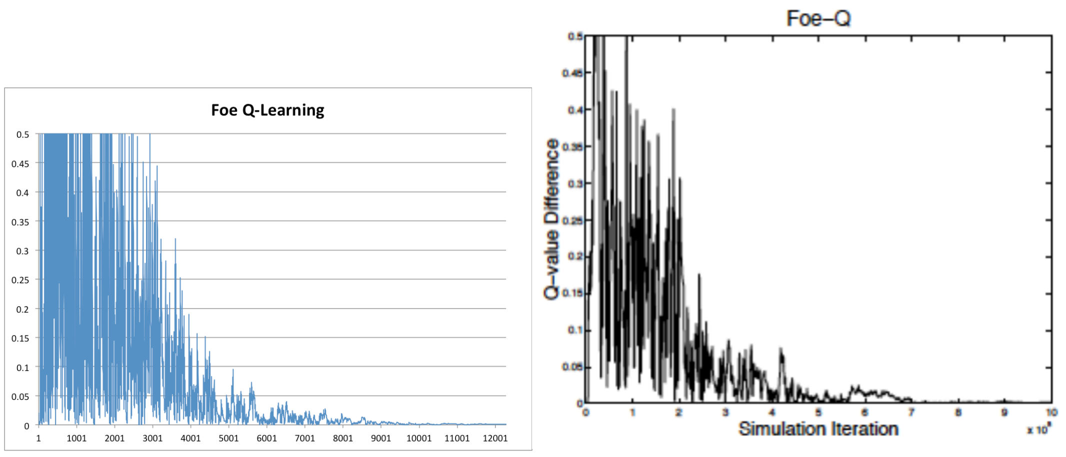


Figure 7 Experimental Results (left) vs Greenwald's Results (right) for Foe Q-Learning

The Foe-Q diagrams are very similar. One difference is it seems Greenwald's converges slightly quicker than ours. This could be due to differing alpha-decay rates (more on this at the end of this section). The algorithm converges to a nondeterministic policy where both players either stay or head south so as to minimize the maximum reward for the opponent (or maximize the minimum reward for themselves).

Correlated Q-Learning

Unfortunately, my results for Correlated-Q did not show any similarity to the figure featured in Greenwald's paper. This is most likely due to an insufficient/inaccurate representation of all of the constraint functions necessary to converge to a solution. It is fairly certain our attempted solution contains the correct probability constraints (to ensure the solution is a proper probability distribution); however, creating the correct rationality constraints (maximizing the minimum utility value based on the opponent's actions) has proved to be difficult.

In due time, I hope to grasp a better understanding of rationality constraints and eventually create an effective Correlated Q-Learning algorithm. Unfortunately a working algorithm could not be developed in the allotted time for this project.

All four algorithms were performed with identical hyperparameters. Alpha was initialized to a value of 0.2, and was multiplied by 0.99999 after each iteration until it reached a value of 0.001 (mentioned in Greenwald's paper). Gamma was initialized to a value of 0.9 and remained constant throughout the experiments. The number of iterations the algorithms performed was 1,000,000.

Each game was initialized to the state depicted in Figure 4, and the Q-value Differences were calculated by finding the absolute value of the different between Q-values for the same state *before* and *after* performing a Q-update. For example, let's assign state S to be the state where Athlete A is in position (2,0), Athlete B is in (1,0), Athlete B has possession of the ball, Athlete A's next action is South, and Athlete B's next action is Stay. The absolute value of the difference between the value of Q[S] before and after performing an update was taken, logged, and graphed. This method of data acquisition was used for all four algorithms.

V. Conclusion

As always, the hardest part of attempting to replicate the results of these highly sophisticated experiments is turning the provided pseudo-code into a working reinforcement model. It's also hard to replicate the graphs provided in the experiments because the methods used to acquire the data/graphs is very ambiguous. For example, it is quite convincing the results of Greenwald's Friend Q-Learner were formed using some sort of moving-average to create a smoother line. However, it is these little ambiguities that force us to dive into the details, look for other resources, and create solutions that, regardless of if they work or not, help us become better practitioners of machine learning.

VI. References

- Santiago L. Valdarrama. Piazza Post @658
- http://cvxopt.org/userguide/modeling.html
- Amy Greenwald and Keith Hall. Correlated-q learning. In In AAAI Spring Symposium, pages 242–249. AAAI Press, 2003
- Michael L. Littman. Friend-or-foe q-learning in general-sum games. In Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, pages 322–328, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In IN PROCEEDINGS OF THE ELEVENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, pages 157–163. Morgan Kaufmann, 1994.