cac42cbe4eecf69a60de27b29fc1f2fb25bc918f

Project 3 Report

Bing Yang

Abstract. In this project, by following the soccer game experiment from Greenwald and Hall (2003), I compared differences in performance of four Q-learning based multi-agent algorithms: traditional Q-learning, Friend-Q, Foe-Q and Correlated-Q (CEQ). My results support the main conclusions that Q-learning does not converge in a general-sum game without deterministic equilibrium while the other three converge.

1. Introduction

A Markov game is a multi-step process in which the state of the game is determined by the actions of the participants stochastically. Based on the current state of the game and their respective actions, each participant gains different levels of rewards (Greenwald, Hall and Zinkevich, lecture notes, 2005). This concept is similar to a Markov decision process, except that in a Markov game there are multiple agents taking actions in each iteration. Several algorithms have been proposed to produce a 'best' policy for the agents in a Markov game. In this project, four algorithms are considered and compared: the traditional Q algorithm, Friend-Q, Foe-Q and Correlated-Q (CEQ).

1.1. Q-learning algorithm

The Q-learning algorithm solves the problem by treating each player's game as an independent Markov decision process, regardless of other opponents' actions. Namely, the update of the Q-table (the state-action value table) for each player is only based on his/her own state-action pair. The actions of the opponents are only used to make the game move forward. Thus, the update function is exactly the same as in a Markov decision process:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha * ((1-\gamma) * r + \gamma * V(s))$$

$$V(s) = \max_{a'} Q(s,a')$$

, in which α is the learning rate and γ is the scaling factor. Notice that the term $\max_{a'} Q(s,a')$ essentially calculates the state-value following the Bellman's equation. The other three algorithms are similar to the Q-learning algorithm except for how they calculate the state-value function based on the Q-table.

1.2. Friend-Q algorithm

In Littman 1994, two algorithms are proposed to look for good policies in a constant-sum two-players Markov game. One of them is Friend-Q. Ignoring all the mathematical details, the basic idea of the Friend-Q algorithm is that the two players share the same reward function so that they are willing to cooperate. In a zero-sum game, like the Soccer game discussed below, this means that one player is sacrificing his own benefit for the other player to win. The state-value, in this case, is calculated as the maximum Q-value over all the possible action pairs:

$$V(s) = \max_{a,a'} Q(s,a,a')$$

Notice that unlike traditional Q-algorithm, the Q-table now is indexed over actions from the two players instead of only one. The update equation is otherwise similar to that of Q-algorithm.

1.3. Foe-Q algorithm

Foe-Q algorithm is proposed together with Friend-Q algorithm. However, instead of cooperating, the opponent now tries to minimize the reward for the player under consideration. In this sense, Foe-Q is suitable for a zero-sum two-player game in which both players intend to optimize their own benefits. The state-value function is calculated as:

$$V(s) = \max_{\sigma} \min_{a'} \sum_{a} \pi(a) * Q(s,a,a')$$

The term $\pi(a)$ is the probability of choosing action a when following policy π. Notice that even though the Q-table is indexed over combinations of actions, the policies are independent for the two players. This is a minmax problem and can be solve by linear programming for the policy probabilities.

1.4. Correlated-Q algorithm

The idea of the Correlated-Q (CEQ) algorithm is that players take actions simultaneously and randomly in combination. It would be easier to understand the concept if there is a referee in the game together with all the players. In each time step, the referee recommends a combination of actions for all the players randomly following the combinatorial policy. Each player can choose to either follow the recommendation or not. The CEQ algorithm tries to look for a policy so that each player cannot do better if they reject the recommendation, namely,

$$\sum_{a_{-i}} \pi(a_i, a_{-i}) Q(s, a_i, a_{-i}) \geq \sum_{a_{-i}} \pi(a'_i, a_{-i}) Q(s, a'_i, a_{-i})$$

In this equation, a_i is the recommended action and a'_i is all the other possible actions under state s. With these restrictions (notice that the above inequality should be true for all players), the state-value can be calculated under different assumptions. The assumption used in this project is the utilitarian assumption, namely,

$$V_i(s) = \sum \pi(a_i, a_{-i}) Q_i(s, a_i, a_{-i})$$
$$\pi(a_i, a_{-i}) \in arg \max_{\pi \in CE} \sum_i \sum_{a_i, a_{-i}} \pi(a_i, a_{-i}) Q_i(s, a_i, a_{-i})$$

In above equations indicate that the selected policy π is defined over the space of joint action from all players. It should meet the restrictions defined above ($\pi \in CE$) and maximize the sum of expected rewards for all the players when follow this policy. Once this policy (or a set of policies) is found, the state-value can be calculated for each player separately.

Notice that the set of restrictions and other constraints so that π defines a probability distribution forms a polytope. This means that the problem of finding the policy can be solved by linear programming.

2. Methods

2.1. Soccer game

The Markov game used in this project is the Soccer game proposed in Littman 1997. Figure 1 shows the basic structure of the game.

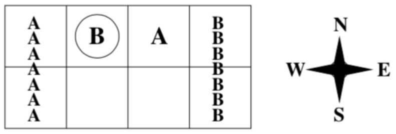

Figure 1. Illustration for the Soccer game (copy from Greenwald and Hall, 2003).

In this game, two players try to score for rewards. The goal gate for each player is designated by a column of his corresponding name. At all steps, exactly one player possesses the ball. If the ball is in A's goal gate, then A gets 100 points and B -100 points. Otherwise A gets -100 and B gets 100. Note that this game allows own goal.

In each time step, player can take 5 actions: UP, DOWN, LEFT, RIGHT and STICK. When take actions, the players cannot cross the boundary of the field. For example, if A takes UP action in Figure 1, then he/she stays in the same place.

In each time step, the two players choose actions simultaneously. However, the order of executing the two actions is random, with equal chance of taking action first for either player. When taking an action results in a collision, the player who takes that action stays in the same place. The possession of the ball changes when the player with the ball tries to invade the position of the other player. In another word, if B tries to move RIGHT in Figure A, then he/she loses the ball to A. This game does not allow stealing ball actively. That means if A moves LEFT, he/she cannot steal the ball from B. In summary, change of ball possession occurs when collision happens, and the static player always takes the ball (the static player might already have the ball and keeps the ball in that case).

2.2. Implementation of algorithms

Q-learning is implemented as described in section 1.1. Q-table is created for each player separately. An epsilon-greedy on-policy algorithm is implemented for Q-learning. The epsilon-greedy procedure is done independently for the two players. The learning rate α starts at 0.9 and decays to 0.001 in a uniform linear fashion over all iterations, and this is true for all the following algorithms. The epsilon value starts at 1.0 and decays to 0.01 in a uniform linear fashion. The decay parameter gamma is always 0.9.

The Q-table is indexed as a two-dimensional array, with the first dimension representing the number of states and second dimension representing the number of actions. The state index is calculated as follows. The field is indexed as 0-7 from bottom left to top right. Thus, the position of each player can be described by a single octal digit. The current state of the game can be represented by a 7-bit integer (1 for ball and 3 for each player's position) so that there are in total 128 different states. Note that not all of the 128 states can happen because A and B cannot stay in the same spot. Nevertheless, these states won't be updated by the learning process. In summary, the size of the Q-table is 128*5 for each player. The indexing of the states is used for all the following algorithms.

The game is simulated until some player scores. The game is then restarted until a pre-specified number of iterations have been performed. Q-table for each player is update once in each iteration. The Q-value for the state showing in Figure 1 with A taking action DOWN is recorded and used for plotting the absolute change of Q-values in all figures below. This is true for all the following implementations.

Friend-Q learning is implemented as describe in section 1.2. An off-policy algorithm is used for the implementation in which the two players choose actions randomly. The Q-table in this case is a three-dimensional array (128*5*5) and the state value is calculated as the maximum of all 25 pairs of actions with respect to player A's reward for the current state in each iteration. Notice that in a zero-sum game, this means player B's reward is minimized. Only the Q-table for player A is used for plotting.

Foe-Q learning's implementation is similar to Friend-Q except that state-value is calculated by solving a minmax value problem using the cvxopt package. The constraints for the LP problem include (1) policy values should be larger than 0 and (2) policy values should sum up to 1 for each player so that they are probabilities. The state value to update the Q-table is the solved target of the LP problem (it is actually the negative of the target because of the formulation used by cvxopt). Note that this is a direct calculation of the expectation over all actions pairs because the policy probabilities calculated from the LP problem are not used to sample actions in each iteration. Only the Q-table for player A is updated and used for plotting.

CEQ learning's implementation is similar to Foe-Q except that the state-value is calculated by solving a minmax problem with more restrictions compared to Foe-Q. The added constraint is the CE restriction described in section 1.4. Once the LP is solved, the target is not the right state-value to be used because it is the sum of rewards for all the players. To calculate the state-values for both players, the policy probabilities are extracted and equations in section 1.4 are used. The Q-tables for both players are updated while only the values from player A are used for plotting.

3. Results

The results are summarized in Figure 2 below.

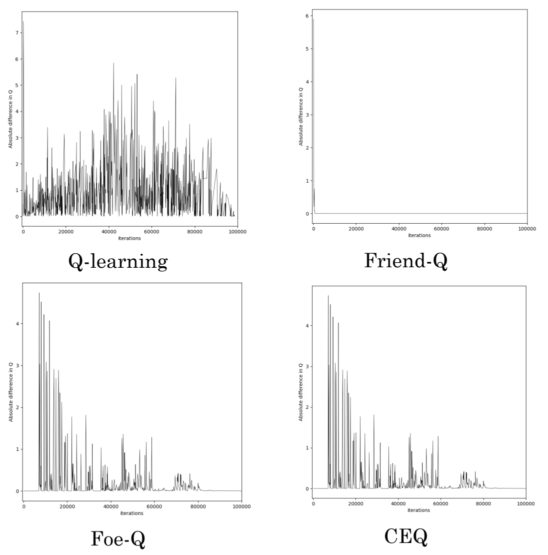

Figure 2. Change of absolute Q-value differences over time for the four algorithms. Niter = 10^5

Compared to Figure 3 in the original paper, my results support the following conclusions. First, Q-learning does not show convergence. The decrease on the absolute differences are caused, as explained in the paper, by the small learning rate at the end of simulation. Second, Friend-Q, Foe-Q and CEQ converged and the converged policy is the same as described in the paper.

However, there are a few differences between my results and the original figure. Below I will discuss those differences in detail.

First, before discussing the differences, one important technical difficulty needs to be brought up. In the paper, the authors talked about the decay schedule *S* for learning rate in *Table* 1. However, besides giving the end target, the authors did not specify what decay schedule they exactly used in their simulation. The results shown are produced by following a uniform linear decay schedule. I have also tried exponential decay and temperature decrease from simulated annealing. Even though the general conclusions are supported in all simulations, the dynamic patterns are drastically different. Sometimes the Q-learning even shows converged pattern. This is best illustrated by Figure 3, in which the only difference between Figure 2 and Figure 3 is the number of iterations. One can see that by changing the rate of decay (because the number of iterations is different), the patterns change drastically, with Foe-Q and CEQ converge almost at the end of the simulation.

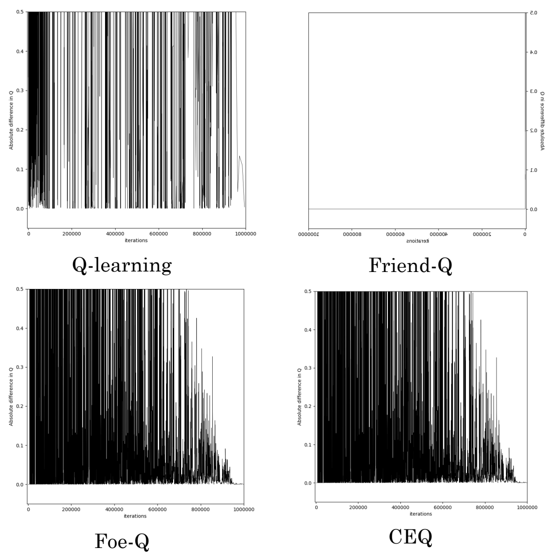

Figure 3. Change of absolute Q-value differences over time for the four algorithms. Niter = 10^6. The limit of y axis is restricted to 0 ~ 0.5, same as the original figure.

Thus, all the differences discussed below might be partly attributed to the different decay schedule used. However, it is not meaningful to find the 'best' decay schedule to match the original result. In any event, the basic conclusions are always supported so that the results from the original paper is still reasonable.

The first difference is that in the original paper, the Q-learning result shows an oscillating behavior at the end of the simulation, and the amplitude is scaled with the learning rate. This is not observed in my simulation. The authors did not explain why there are oscillations at the end. My thought was that because the Soccer game is a zero-sum game, the learning process might oscillate between best policy for A and for B. However, in my simulation even though I created two Q tables for both players (while not update them jointly), I did not see the oscillation. As mentioned above, this could be due to the different decaying schedule used.

The second difference is that in my Friend-Q learning, the convergence is almost instantly reached at the beginning of the simulation, unlike the gradual decrease at least during the first 1/10th iterations in the original paper. Again, there could be multiple reasons for that, including different random seed, different decaying schedule and so on.

One interesting point is that, like the original result, the dynamic patterns of Foe-Q and CEQ are exactly the same when used with the same random seed. In the paper, the author used the word 'coincide' to refer to this phenomenon. My thought is that this might be a general outcome for a two-player zero-sum game with utilitarian version of the CEQ algorithm. My logic is as follows. In a two-player zero-sum game, the behavior of the two players are more or less mirror to each other. Maximize the minimum reward for each player will lead to the same policy and a zero sum of rewards. This could be exactly the solution for the correlated equilibrium solution in this case. When simulated with the same random seed, the two algorithms will converge to the same set of policies in this situation. Thought I could not give a rigorous proof for my thoughts, the 'coincidence' in all my simulations could be a systematic property under the specific condition.

4. Discussion

Overall, even though there are some differences between my results and that of the original paper, my results support the general conclusion of the paper, namely Q-learning does not converge while the other three converge. Also, the Friend-Q converges to an irrational policy while the other two converge to a more reasonable mixed strategy. Replicating results from previous papers deepens my understanding of all these algorithms. By solving the linear programming method on my own, I now have better sense of the technical details for these algorithms. Also, by reproducing the results in slightly different settings, one can check the generality of the conclusions from the paper.

5. References

Greenwald, Hall and Zinkevich. *Correlated Q-learning*. Brown University Technical Reports (2005) CS-05-08

Greenwald and Hall. *Correlate-Q learning. ICML-2003.*

M. Littman. *Markov games as a framework for multi-agent reinforcement learning. ICML-1994.*