Zheng Fu / zfu66@gatech.edu

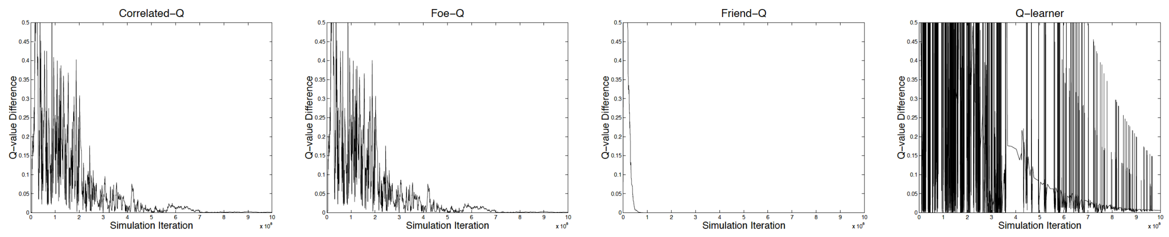
Project 3: Evaluating multi-agents Q-learning algorithms in zero-sum Markov game

I. Motivation

Stochastic game is a standard framework for modeling multi-agents reinforcement learning algorithms. It could be represented by a tuple (***I, S, A, P, R***), where

I = {1,2,3, …, *i*} is a set of agents;
S is a finite set of states;
A = {A_1, A_2, … , A_i} is a set of actions and A_i is a finite set of all possible actions for a given agent *i*;
P: *S* x A_1 x … x A_i x *S* -> [0,1] is the probability transition function to depict the probability of transiting among states based on agents' joint actions;
R = {R_1, R_2, … , R_i} is a set of rewards and R_i : *S* x A_1 x … x A_i is the reward of agent *i* conditioned on state and agents' joint actions.

If ***P*** fulfills Markov property, such stochastic game is called Markov game. In 2003 Amy Greenwald and Keith Hall published a paper named "Correlated-Q Learning", and they proposed soccer, a zero-sum Markov game, to test four Q-learning algorithms: CE-Q, Friend-Q, Foe-Q and ordinary Q-learning. The primary goal of this project is to replicate the experiments of this soccer game, as well as their results in Greenwald's 2003 paper, Figure 3(see below):



(a) Correlated-Q (b) Foe-Q (c) Friend-Q (d) Q-learning

Figure 3. Convergence in the soccer game. All algorithms—except Q-learning—converge. As above, the CE-Q algorithm shown is uCE-Q.

II. Implementation

1. Soccer game environment

As mentioned in section I, the Markov game is a 5-tuple thus generating these 5 elements is the key to successfully implementing the soccer game environment.

(1) Set of agents:

This soccer game is a two-players game. So I generated two players and each one has two features: its position and its name ("A" or "B").

(2) Set of actions:

According to the paper, the player has five actions: N, S, E, W, stick, and I use 0,1,2,3,4 to stand for these five actions, respectively.

Challenge: I need to write few statements to judge if an action is allowed. Say if a player is in the second row and the action is "go South", then it won't be permitted and the position of this player will not be changed.

(3) Set of States:

The soccer field is a 2 X 4 grid (see Figure 4 in Greenwald's 2003 paper) and it doesn't allow the players occupying the same cell. The all combinations of players' positions should be 8 * (8-1) = 56. And the ball also has two states: with player A or with player B. So the total number of states in this game is 56 * 2 = 112.

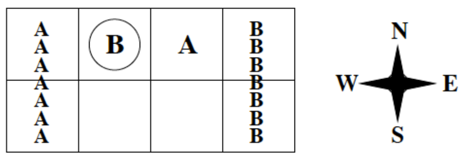


Figure 4. Soccer Game. State s.

Challenge: The tricky part is to determine the ball state, since the players' actions are performed in random order. In my program if the first-moving player moves into the other player's position with the ball, and the second-moving player choses sticking, the ball state will be changed. Also I assume if the player without ball intends to steal the ball, neither its position nor ball state will be changed.

(4) Set of Rewards:

If ball is in player A's goal region, A scores 100 and B scores -100. If ball is in player B's goal region, A scores -100 and B scores 100. In either case the game is done.

Challenge: There is no variable in states space to describe the ball's position. So I need to check both players' positions and ball state to determine if the game ends.

(5) Probability transition function:

This is a Markov game so the next state of game only depends on the current state and the player's action.

2. Agents

(1) Q-learning theory:

For single agent in Markov decision processes (MDPs), Q-learning is an algorithm to find out the optimal action-selection policy for the agent. It updates the quality value of a state-action combination based on Bellman equation:

Eq1:

$$Q(s,a) := Q(s,a) + \alpha[r + \gamma \cdot \max_a Q(s',a) - Q(s,a)]$$

where s is the current state, a is agent's action, s' is the next state, α is the learning rate, γ is the discount factor, r is the reward of action. For multi-agents Markov game, the updating rule is similar with Eq1.

Eq2:

$$Q_i(s,\vec{a}) := Q_i(s,\vec{a}) + \alpha[r + \gamma \cdot V(s') - Q_i(s,\vec{a})]$$

where s is the current state, $\vec{a}$ is the joint agents' actions, s' is the next state, α is the learning rate, γ is the discount factor, r is the reward of $\vec{a}$ and V(s') is the value function of next state(aka future reward). In this project I implemented four algorithms using different approaches to estimate V(s') and evaluate the Q-value convergence. Since this soccer game has finite, discrete states space, I use Q table to trace the Q-value updates.

Challenge: The Eq2 is different from the updating rule in Greenwald's 2003 paper's Table 1, which times (1- γ) on current reward. I did few tests on both rules with different Q-learning algorithms and the results were pretty much the same. Thus in this project I use Eq2 as the multi-agents Q-value updating rule.

(2) Ordinary Q-learning:

Ordinary Q-learning is the same as the single agent in MDPs and $V(s') = \max_a Q(s',a)$. That means player A and player B have their own Q-table (112 states X 5 actions) and update independently. The Q-table is initialized with all zeros. To balance the exploration and exploitation, I use epsilon-greedy search to determine player A's action and player B's action is always randomly chosen. The epsilon value is set to 0.90 with a linear decay after each iteration. The discount factor is equal to 0.90. The Q-value error was computed on player A's Q-value corresponding to state s in Figure 4 in Greenwald's 2003 paper and player A takes action S. Such Q-value error will not be estimated if the current state is not state s during a single iteration. Once the game is done, the state will be reset to state s. Figure 1D shows the Q-value convergence tendency after 1 million iterations using ordinary Q-learning

Challenge: It is hard to match the Figure3(d) in Greenwald's 2003 paper. Since the Q-value error decreasing only rely on the learning rate reduction, I tried different alpha decay values and finally found if alpha starts with 1.00, alpha = alpha * alpha_deacy after each iteration where alpha_deacy = 0.999993, and the alpha decay stops if alpha is no more than 0.001. Under these conditions my figure closely matches the Figure3(d) in Greenwald's 2003 paper.

(3) Friend Q-learning:

The idea of friend Q-leaning is to assume all agents have the same reward function. The difference between ordinary Q-learning and friend Q-learning is the latter one's Q-values are over joint actions:

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

Thus only one Q-table is necessary (112 states X 5 actions X 5 actions) for the learning steps and is initialized with all zeros. Both players take actions randomly. The discount factor is equal to 0.90. The Q-value error was computed on player A's Q-value corresponding to state s in Figure 4 in Greenwald's 2003 paper and A takes action S, B takes action sticking. Once the game is done, the state will be reset to state s. Figure 1C shows the Q-value convergence tendency after 1 million iterations using friend Q-learning.

Challenge: The Q-value of friend Q-learning is converged very quickly. Yet in Figure3(c) in Greenwald's 2003 paper, the Q-values different become to 0 after 50000 iterations. So I tried few learning rate values and found if alpha starts with 0.04, alpha = alpha * 0.999993 after each iteration, and the alpha decay stop if alpha is no more than 0.001. Under these conditions my figure closely matches the Figure3(c) in Greenwald's 2003 paper.

(4) Foe Q-learning:

The strategy of foe Q-learning is not maximize $Q(s', \vec{a})$ but the player needs to consider the adversary's actions. It assumes the adversary's goal is to minimize the player's reward. Hence $V_1(s') = -V_2(s')$ and it is the essence of zero-sum game. In this scenario

$$V(s') = \max_{\pi_s} \min_{a_2} \sum_{a_1} Q(s, a_1, a_2)\pi_s(a_1)$$

where a_1 is player1's action, a_2 is player2's action, s is current state, $\pi_s(a_1)$ is the probability of taking action a_1 while following strategy π_s. I convert the above equation into a linear programming problem:

Object: maximum U

Subject to: $P(a_1)Q(s, a_2=0) <= U$; $P(a_1)Q(s, a_2=1) <= U$; $P(a_1)Q(s, a_2=2) <= U$; $P(a_1)Q(s, a_2=3) <= U$; $P(a_1)Q(s, a_2=5) <= U$; all elements in $P(a_1) > 0$; the sum of all elements in $P(a_1) = 1$.

Note that $P(a_1)$ is a vector containing 5 elements, which represent the probabilities of player1 taking 5 actions, respectively. Based on the LP object and constraints, I construct matrix A, b and c (please see my codes comments for more details) and use package 'cvxopt' to solve the linear programming problem in each iteration. The value of U is the V(s') and I use it to update the Q-table. Again only one Q-table is necessary (112 states X 5 actions X 5 actions) for the learning steps and is initialized with all zeros. Both players take actions randomly. Discount factor = 0.90. Learning rate starts on 1.00 with decay factor = 0.999993, and the alpha decay stop if alpha is no more than 0.001. The way of computing Q-value error was as same as that in friend Q-learning. Figure 1B shows the Q-value convergence tendency after 1 million iterations using foe Q-learning.

Challenge: The first challenge is that the Q-value differences become to zero after 50000 iterations, whereas in Figure3(b) in Greenwald's 2003 paper, it occurs after about 700000 iterations. Then I abandon to reset the state to state s after game ends, instead I randomly assign player A and B's positions (not in goal region) as well as ball state after game ends and it fixes this issue.

The second challenge is how to properly construct matrix A, b and c, especially for the constraint "the sum of all elements in $P(a_1) = 1$". I initially set x = [U, P0, P1, P2, P3, P4]^T, A[n,] = [0,1,1,1,1,1], b[n,] = [1], and I always have U = 0. The I add one more line in A and b: A[n+1,] = [0,-1,-1,-1,-1,-1], b[n+1,] = [-1] to fix this issue, since the requirement is $Ax^T \le b$.

(5) CE Q-learning:

Unlike the foe Q-learning, the goal of CE Q-learning is not to find an probability vector for each player's actions, but to identify the joint probability distribution of players joint actions to fulfill correlated equilibrium. Thus the V(s') should be expressed as:

$$V(s') = CE[Q_1(s), Q_2(s) \dots, Q_n(s)]$$

where $CE[Q_1(s), Q_2(s) \dots, Q_n(s)]$ is a player's reward based on some correlated equilibrium and a Q matrix vector $Q_1(s), Q_2(s) \dots, Q_n(s)$. And I convert it into the following linear programming problem:

Object: $max \sum_{a,i,j,i\neq j} P(a)U(a_{i1,} a_{j2})$

Subject to: $\sum_{a,i,j,i\neq j} P(a)[\, U(a_{i1,} a_{j2}) - U(a'_{i1}, a_{j2})] \ge 0,\ P(a) > 0, \sum_a P(a) = 1$

where a belongs to the joint actions space, $P(a)$ is the probability of taking a given joint action a, $U(a_{i1,} a_{j2})$ is the reward, a_{i1} is player 1 taking action i, a_{j2} is player 2 taking action j, a'_{i1} is player 1 not taking action i. Based on the LP object and constraints, I construct matrix A, b and c and use package 'cvxopt' to solve the linear programming problem in each

iteration. Two Q-tables are generated (112 states X 5 actions X 5 actions) for player A and B and are initialized with all zeros. The vector $P(a)$ dot multiplies the reward vector of 25 joints actions based on Q-table of player A is the V(s') of player A. Since it is a zero-sum game, the V(s') of player B is equals to the V(s') of player A times -1. Both players take actions randomly. Discount factor = 0.90. Learning rate starts on 1.00 with decay factor = 0.999993, and the alpha decay stop if alpha is no more than 0.001. The way of computing Q-value error was as same as that in friend Q-learning. Figure 1A shows the Q-value convergence tendency after 1 million iterations using CE Q-learning.
Challenge: Since utilitarian (uCE-Q) is used in Figure3(a) in Greenwald's 2003 paper, the reward vector of 25 joints actions based on Q-table of player A, should add the reward vector of 25 joints actions based on Q-table of player B to build up the matrix c for solving the linear programming problem.

III. Conclusions

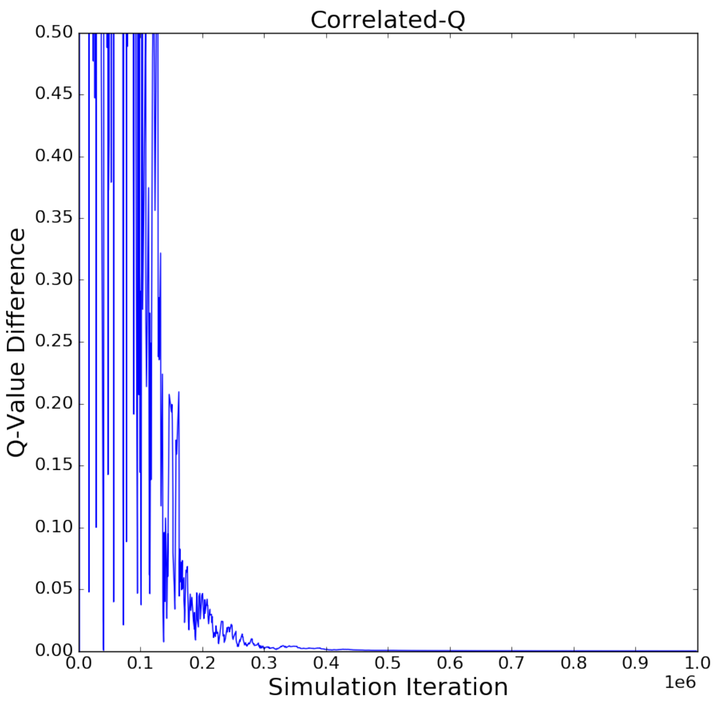
Figure 1D shows the simulation results of using ordinary Q-learning in soccer game after 1 million iterations, and we could see the Q-value differences are not converged. Since ordinary Q-learning looks for an optimal pure strategy in this game yet it doesn't exist. Figure 1A, B and C reveal that the Q-value differences are converged after 1 million iterations by using CE Q-learning, foe Q-learning and friend Q-learning, respectively. The convergence speed of friend Q-learning is the fastest since it converges to a pure strategy starting from state s: player A sticks -> player B moves to East and transfers ball to player A -> player A moves to East -> player B scores 100 and game ends. Yet this strategy is irrational. For zero-sum Markov game, it exists Nash equilibrium at a saddle point on reward function surface, and foe Q-learning finally converges to that saddle point using minimax-Q methods. The purpose of CE Q-learning is to figure out the multiple equilibria selection issue in Markov game. In this game, both CE Q-learning and foe Q-leaning select the same equilibrium strategy. Overall my experiments are successful and all figures are closely matched their counterparts in Figure3 of Greenwald's 2003 paper.

IV. References

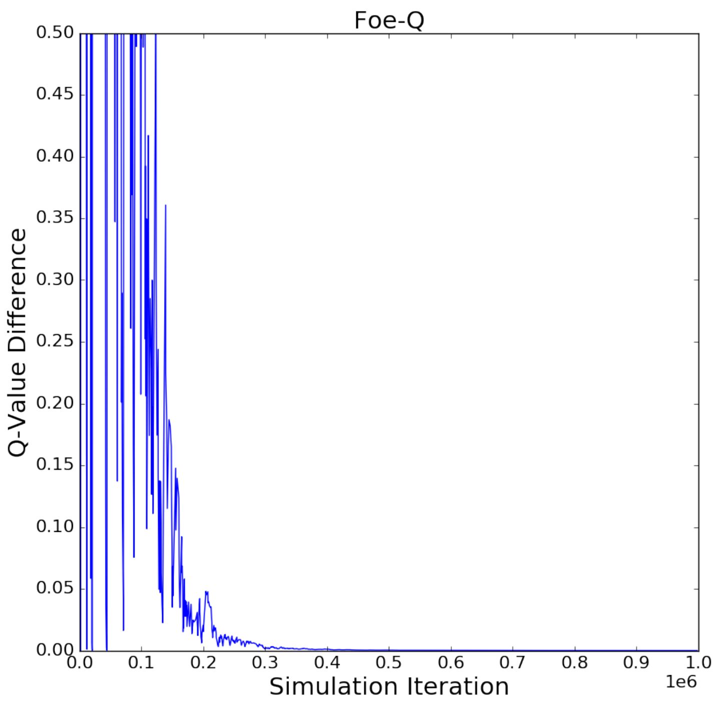
1. "Correlated-Q learning", Amy Greenwald, Keith Hall, ICML'03 Proceedings of the Twentieth International Conference on International Conference on Machine Learning, Pages 242-249.
2. "Markov games as a framework for multi-agent reinforcement learning", Michael L. Littman, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, July 10–13, 1994, Pages 157–163.

Figure 1. Q-values convergence in the soccer game.

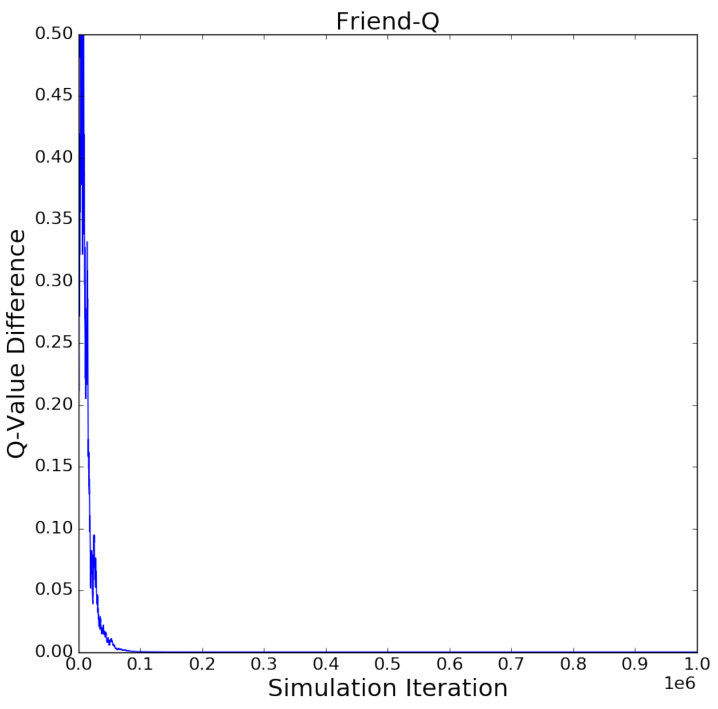
A. Correlated-Q



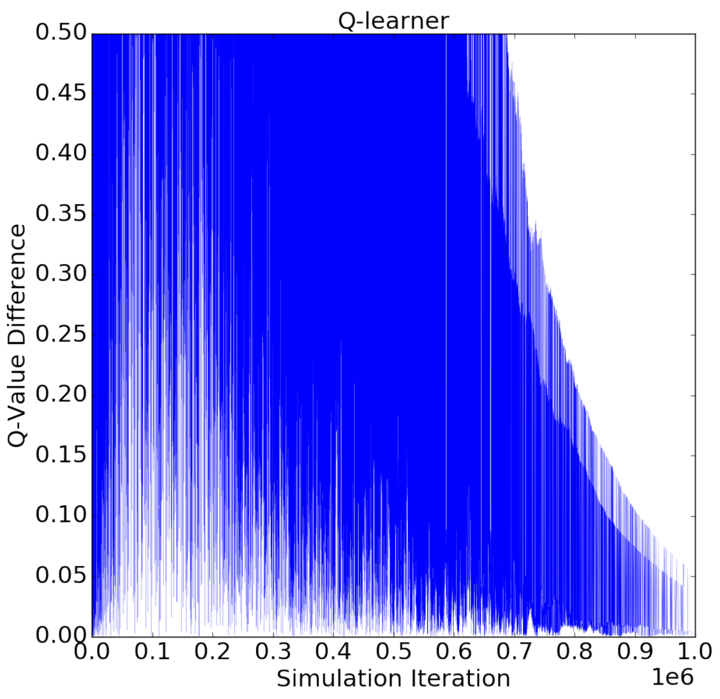
B. Foe-Q



C. Friend-Q



D. Q-learning


Q-learner
Q-Value Difference
Simulation Iteration
1e6