Project 3: RoboCup

Matthew Molinare
mmolinare@gatech.edu
CS 7642 Reinforcement Learning

Abstract

I describe several Q-learning algorithms that use different definitions of the state-action value function to determine optimal strategies in multiagent general-sum games. These include the Friend-Q, Foe-Q, and Correlated-Q learners. I study the behavior of such algorithms in a simple, non-deterministic, two-player Markov grid game known as RoboCup. Furthermore, I discuss methods used to successfully replicate the convergence results of Greenwald and Hall.

1 Introduction

The Q-learning algorithm learns optimal policies in Markov decision processes (MDPs) by maximizing an agent's expected sum of discounted rewards, or action-value function, according to the value iteration update,

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma V(s')) \tag{1}$$

where α is the learning rate, $\gamma \in [0,1)$ is the discount factor, and r is the reward received when transitioning from state s to state s' after taking action $a \in A(s)$. For single-agent Q-learning, the state-action value $V(s)$ is defined as

$$V(s) = \max_{a \in A(s)} Q(s,a) \tag{2}$$

In multi-agent environments, Q-values are defined over states and joint action-vectors $\vec{a} = (a_1, ..., a_n)$, where n is the number of agents. One such environment is a Markov game where players synchronously choose actions and state transitions are described by a probability function p that satisfies the constraint,

$$\sum_{s'} p(s'|s,\vec{a}) = 1 \tag{3}$$

Littman [1] defines the Friend-of-Foe Q-learning (FFQ) method for learning a Q function for a player interacting with other players in the game. From the perspective of player 1 in a two-player, zero-sum Markov game where player 2 is considered a friend, the value function is the maximum Q-value over the action-space,

$$V_1^{\text{Friend-Q}}(s) = \max_{\vec{a} \in A(s)} Q_1(s,\vec{a}) \tag{4}$$

If player 2 is considered a foe, the value function is the minimax,

$$V_1^{\text{Foe-Q}}(s) = \max_{\pi_1 \in \Pi_1(s)} \min_{a_2 \in A_2(s)} \sum_{a_1 \in A_1} \pi_1(a_1) Q(s,a_1,a_2) \tag{5}$$

where $\Pi_i(s)$ is the probabilistic action space of player i at state s. Eq. (5) can be computed using linear programming by expressing the problem in the following form:

$$\begin{aligned} \text{maximize} \quad & V_1(s) \\ \text{subject to} \quad & \pi_1(a_1) \geq 0 \text{ for all } a_1 \in A_1 \\ & \sum_{a_1 \in A_1} \pi_1(a_1) = 1 \\ & \sum_{a_1 \in A_1} \pi_1(a_1) Q(s,a_1,a_2) \geq V_1(s) \end{aligned} \tag{6}$$

The canonical form for linear programs with both equality and inequality constraints is

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & Gx \leq h \end{aligned} \tag{7}$$

where $c^T x$ is the objective function and x is the vector of variables to be computed. The matrix A and vector b contain the coefficients of the equality constraints. Likewise, the matrix G and vector h contain the coeffecients of the inequality constraints.

Greenwald and Hall [2] propose a definition of the value function based on a correlated equilibrium (CE) probability distribution over the joint action-space from which no agent is motivated to deviate unilaterally. This is a generalization of a Nash equilibrium in which the players' actions are independent random variables. The value function in a Markov game involving a set of players I can be represented by a set of correlated equilibria,

$$V_i^{\text{CE-Q}}(s) \in \sum_{\vec{a} \in A(s)} \pi(\vec{a}) Q_i(s, \vec{a}) \tag{8}$$

where,

$$\pi \in \arg\max_{\pi \in \text{CE}} \sum_{i \in I} \sum_{\vec{a} \in A(s)} \pi(\vec{a}) Q_i(s, \vec{a}) \tag{9}$$

The correlated equilibria can be computed using the linear programming formulation,

$$\begin{aligned} \text{maximize} \quad & V(s) \\ \text{subject to} \quad & \pi(\vec{a}) \geq 0 \text{ for all } \vec{a} \in A(s) \\ & \sum_{\vec{a} \in A(s)} \pi(\vec{a}) = 1 \\ & \sum_{\vec{a} \in A(s)} \pi(\vec{a}) Q_i(s, \vec{a}) \geq \sum_{\vec{a} \in A(s)} \pi(\vec{a}) Q_i(s, \vec{a}_{-i}) \\ & \sum_{i \in I} \sum_{\vec{a} \in A(s)} \pi(\vec{a}) Q_i(s, \vec{a}) = V(s) \end{aligned} \tag{10}$$

where $\vec{a}_{-i} = (a_1, ..., a_{i-1}, a_{i+1}, ..., a_n)$. Assuming the number of possible actions $\mathcal{A}$ is fixed over all states, the total number of rationality constraints is $2\mathcal{A}(\mathcal{A} - 1)$.

2 Environment

2.1 RoboCup

Greenwald and Hall study the convergence of the Q-learning, Foe-Q, Friend-Q, and CE-Q methods in a zero-sum Markov game. The environment consists of a 2×4 grid representing a soccer field. There are two players whose possible actions, selected at random and executed in random order, are to move up (N), down (S), right (E), left (W), or remain in the current cell (stick). A player is unable to move into a cell occupied by the other player. If the player with possession of the ball attempts to do so, the possession changes. A player with possession of the ball receives a reward of +100 if he moves into his own goal and a reward of -100 if he moves into the other player's goal. The other player receives the negation of this reward. Figure 1 is a depiction of the state $\boldsymbol{s}$ at which Q-value convergence is measured.

2.2 Implementation

The CVXOPT Python library [3] is used to solve the linear programs in the Foe-Q and CE-Q value iteration updates (Eqs. (5, 8)). The size of the action-space for a single player in the RoboCup environment is $\mathcal{A} = 5$. The Foe-Q minimax objective function is

$$\begin{aligned} x_{6\times 1} &= \begin{bmatrix} \pi_1(N) & \pi_1(S) & \pi_1(E) & \pi_1(W) & \pi_1(\text{stick}) & V_1(s) \end{bmatrix}^T \\ c_{6\times 1} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}^T \end{aligned} \tag{11}$$

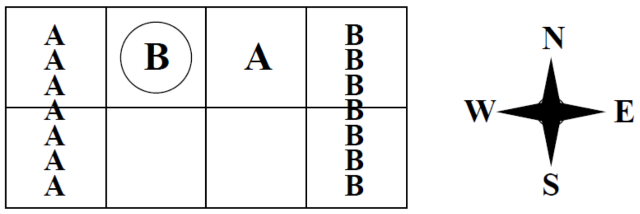

Figure 1: State s of the RoboCup environment. Player A may indefinitely block player B from reaching his own goal if player B is to pursue a deterministic policy whereby, to avoid losing possession of the ball, he never chooses to move into the cell occupied by player A. Therefore, there exists no deterministic equilibrium policies for this game.

The factor of -1 results in a maximization of the value function. From Eqs. (6, 7), the coefficients of the inequality constraints are

$$\begin{aligned} G_{10\times 6} &= \begin{bmatrix} -I_5 & 0 \\ -Q(s,a_1,a_2)^T & V_1(s) \end{bmatrix} \\ h_{10\times 1} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \end{aligned} \tag{12}$$

The matrix $Q(s, a_1, a_2)$ has shape $\mathcal{S} \times \mathcal{A} \times \mathcal{A}$, where the size of the state-space $\mathcal{S} = 8^2 \cdot 2 = 128$ for the 8 possible positions of each player and a boolean signifying which player has possession of the ball. The equality constraints are

$$\begin{aligned} A_{1\times 6} &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \\ b_{1\times 1} &= \begin{bmatrix} 1 \end{bmatrix} \end{aligned} \tag{13}$$

In the CE-Q algorithm, action-values for each player are stored in separate Q-tables, $Q_1(s, a_1, a_2)$ and $Q_2(s, a_1, a_2)$. From Eq. (10), the number of rationality constraints is 40 and may be best represented by the pair of inequalities,

$$\begin{aligned} \sum_{a_1\in A_1(s),\ a_2\in A_2(s)} \pi(a_1,a_2)Q_1(s,a_1,a_2) &\geq \sum_{a_1\in A_1(s),\ a_2\in A_2(s)} \pi(a_1,a_2)Q_1(s,a_{-1},a_2) \\ \sum_{a_1\in A_1(s),\ a_2\in A_2(s)} \pi(a_1,a_2)Q_2(s,a_1,a_2) &\geq \sum_{a_1\in A_1(s),\ a_2\in A_2(s)} \pi(a_1,a_2)Q_2(s,a_1,a_{-2}) \end{aligned} \tag{14}$$

The resulting inequality constraint coefficient matrix G has shape 65×26.

3 Results

I applied my implementations of the Q-learning, Friend-Q, Foe-Q, and Correlated-Q algorithms to the RoboCup game environment in order to replicate the convergence results of Greenwald and Hall. Here, convergence is defined as the absolute difference in the Q-value (pre- and post-update) corresponding to state $\boldsymbol{s}$ (Figure 1), with player A taking action S and player B sticking. Like the authors, I used a discount factor of $\gamma = 0.9$ and an exponential decay learning rate schedule. The Friend-Q, Foe-Q, and Correlated-Q learners use an initial learning rate of $\alpha = 0.2$ and a final learning rate of 0.001 after one million simulation iterations. The Q-learner uses an initial learning rate of $\alpha = 0.4$ and a final learning rate of 0.003. This was necessary to match the decay patterns in the Q-learner plot (starting around 700,000 iterations) with those in the reference paper.

I was able to reproduce Greenwald and Hall's results for all four learners. The Friend-Q, Foe-Q, and Correlated-Q learners all converge, while the Q-learner does not (see Figure 2). In my simulation, I used a fixed initial state every time the game was reset. The choice of initial state, whether fixed or random, had no impact on convergence, so long as the state was valid and non-terminal. As shown by Greenwald and Hall, the Foe-Q and Correlated-Q plots (bottom-left and bottom-right) are identical, converging at approximately 750,000 iterations. The Friend-Q result (top-right) is also consistent with their findings, converging nearly monotonically after just 50,000 iterations.

The Q-learning algorithm, although the simplest to implement, was the most problematic in terms of generating a curve that matched Greenwald and Hall. Their plot shows distinct bands of intermediate

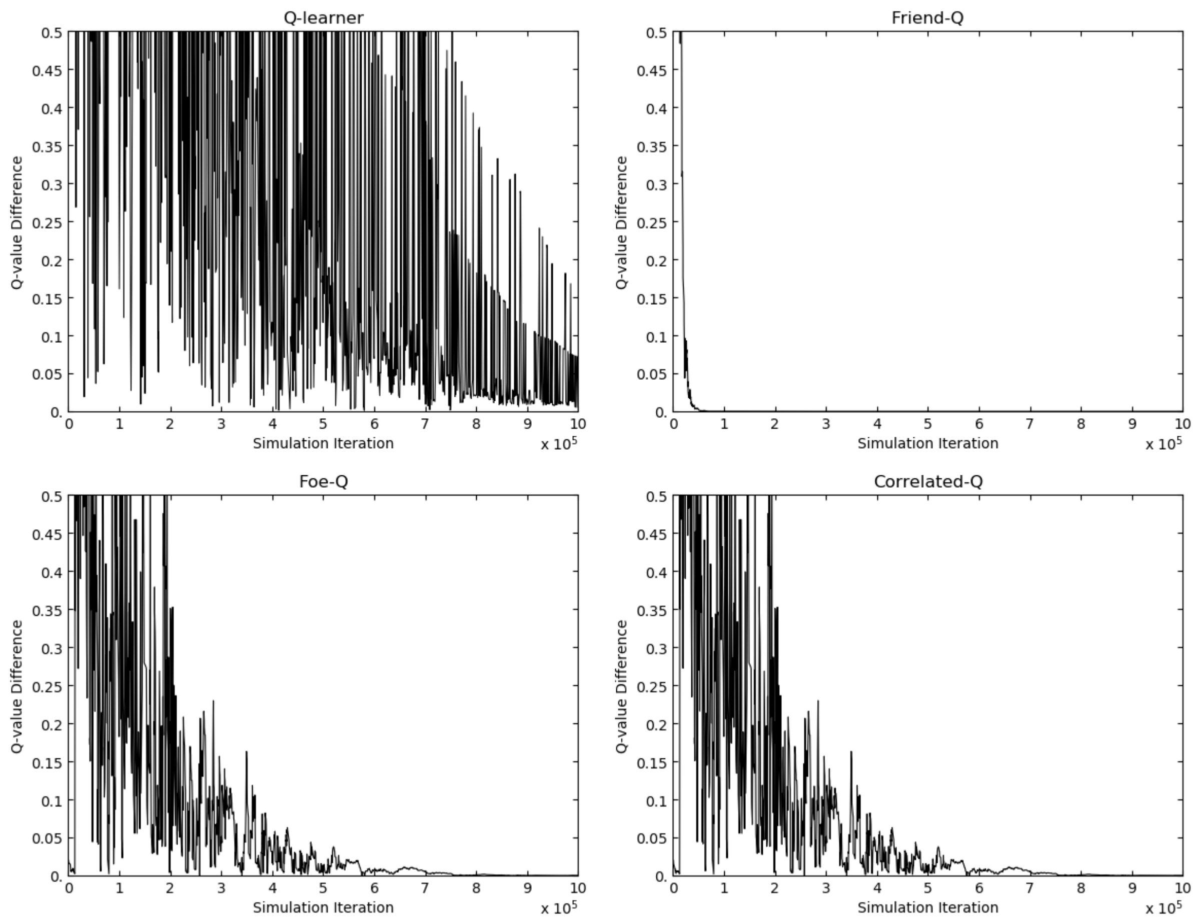

Figure 2: Convergence of Q-values in the RoboCup game. Games are initialized using a fixed starting state. Only the Q-learner fails to converge.

values around 400,000 iterations and again at 700,000 iterations. My difference values were notably noisier, containing numerous data points at extreme values (< 0.05 and > 0.5). This was overcome by plotting only every fourth data point (top-left), allowing these intermediate values to be better visualized.

Perhaps the greatest obstacle in replicating Greenwald and Hall's results was defining a concrete set of rules for the RoboCup game. Note the authors' description:

The players' actions are executed in random order. If this sequence of action causes the players to collide, then only the first moves.

It was not clear, then, the outcome of a collision between two players caused by the first player moving into the cell occupied by the second player who chooses to stick. Since the first player cannot move into a cell already occupied by the second player, I assumed that no player is able to move in this scenario, and the ball goes to the second player if he does not already have possession.

Another pitfall I encountered was the authors' definition of the update rule,

$$Q_i(s,\vec{a}) \leftarrow (1-\alpha)Q_i(s,\vec{a}) + \alpha((1-\gamma)r_i + \gamma V_i(s')) \tag{15}$$

This deviates from the original description of Q-learning by Watkins [4] by the inclusion of the $1-\gamma$ term. Removing this term, Eq. (15) reduces to the following form,

$$Q_i(s,\vec{a}) \leftarrow (1-\alpha)Q_i(s,\vec{a}) + \alpha(r_i + \gamma V_i(s')) \tag{16}$$

which is a generalization of Eq. (1) for player i in a multiagent environment. This was the form used to generate the results shown in Figure 2. Use of Eq. (15) with the same discount factor and learning rate schedule results in an exceedingly rapid decay of Q-value differences (see Figure 3).

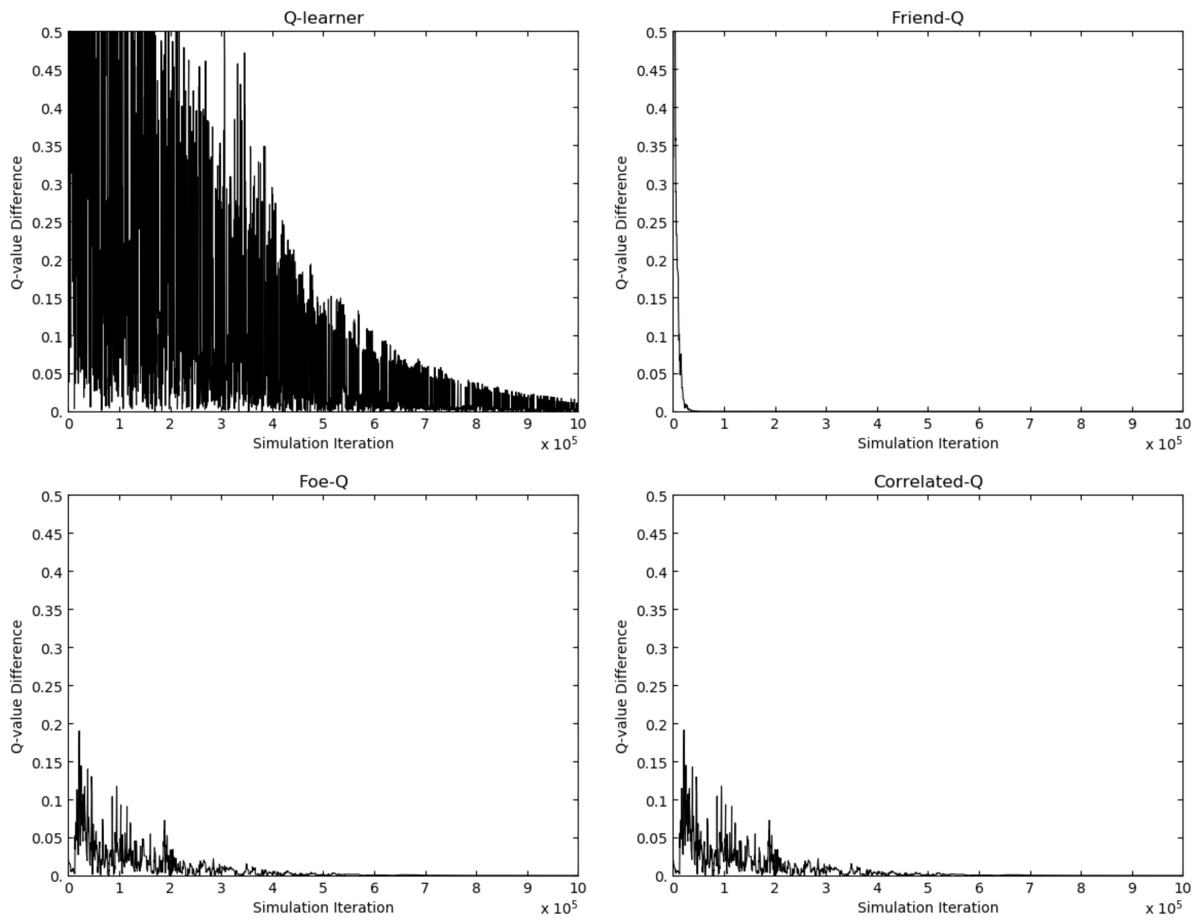

Figure 3: Convergence of Q-values in the RoboCup game using Greenwald and Hall's definition of value iteration.

4 Conclusion

I was able to implement the Foe-Q, Friend-Q, and Correlated-Q algorithms for multiagent learning and replicate the convergence results of Greenwald and Hall for a two-player Markov game. I made assumptions about the rules of the game regarding the collision of two players and the authors' expression for the Q-value update. Nevertheless, I believe my results closely match those of Greenwald and Hall and correctly demonstrate the unique properties of the various learners.

References

[1] Littman, Michael L. *Friend-or-foe Q-learning in general-sum games.* ICML. Vol. 1. 2001.

[2] Greenwald, Amy, Keith Hall, and Roberto Serrano. *Correlated Q-learning.* ICML. Vol. 3. 2003.

[3] CVXOPT: Python Software for Convex Optimization. https://cvxopt.org/

[4] Watkins, Christopher JCH, and Peter Dayan. *Q-learning.* Machine Learning 8.3-4 (1992): 279-292.