Multiagent Q-Learning for Zero-Sum Markov Games Using Correlated Equilibria

Michael Benjamin Burns

mburns49@gatech.edu

Git Hash: 6bd9913bcfff81ae56cc66117d91744addccfda8

***Abstract* - This paper details and contrasts the implementation and effectiveness of 4 different Q-learning algorithms: correlated-Q, foe-Q, friend-Q, and Q-learning; for the purpose of multiagent reinforcement learning in the context of a zero-sum game in which no deterministic equilibrium policy exists. It will be shown how these algorithms make use of the concept of correlated equilibria to find an effective mixed strategy. We will show how and why linear programming can be used to find the probabilities of signal presentations to the agents, which will guide their decision making.**

Index Terms - Correlated Q, Correlated Equilibria, Linear Programming, Game Theory,

I. Introduction

The problem space of Q-learning as an approach to reinforcement learning in single agent, deterministic MDPs is relatively well explored. A more challenging matter is that of multi-agent markov games, specifically those using multiple agents with mixed strategies. In this paper we will attempt to create a simple, zero-sum soccer game environment, as used by Greenwald and Hall in their 2003 paper, "Correlated-Q Learning", [1] and Littman in his 1994 paper, "Markov games as a framework for multi-agent reinforcement learning", [2] as a means of testing the 4 multiagent Q-learning algorithms to try to find convergence on an optimal strategy for each agent. These Bellman equation based algorithms all use the same overall structure aside from their selection function which determines which strategy for achieving equilibrium to use. We will also see that these algorithms are prime candidates for the use of linear programming because of their complexity and their requirement for maximization of probable rewards.

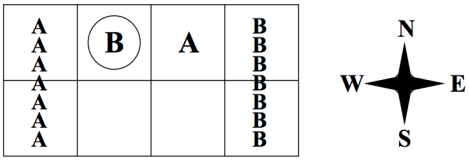

Fig. 1 Game environment grid with arbitrary agent positions (From Greenwald/Hall [2])

II. Environment

The soccer game environment that will be used to test the agents will be made to reflect that which is defined in Greenwald/Hall 2003. Littman's 1994 paper is referenced by Greenwald/Hall so it is assumed that any ambiguities in the definition of the Greenwald/Hall soccer environment can be informed by Littman's implementation. The environment is zero-sum in its rewards- that is any time player A gets a reward, player B gets an equivalent negative reward and vice-versa.

It will be a grid with 2 rows and 4 columns[1] wherein the 2 agents (A and B) independently occupy a grid space at a time. The starting state of the board is not defined by the Greenwald/Hall paper, and the board size is different in the Littman paper (4x5), however a board start state in the Littman paper had the agents diagonal from each other in the center of the board so that form was adopted for this paper wherein the start state has A at position (1,2) and B at (0,1). Either one of the agents can possess the ball at a time. If either agent with the ball enters either of the far left 2 spaces, it will score for agent A, giving a reward of +100 to A, and thus a -100 reward to B. The inverse is true for the 2 far right spaces, giving -100 and +100 to A and B, respectively. In either case, the game is reset to the starting state again.

Agents have 5 available actions: North, South, West, East, and Stay. "Stick" was chosen by Greenwald/Hall instead of "Stay" as defined by Littman, but we will use "Stay" here as it is less likely to be mistaken for another action. The 4 cardinal directions move the agents 1 grid square in those directions, and stay remains in the same cell without moving. When an agent takes an action that would move it off of the grid, that agent remains in the same space. Agents choose the actions they wish to take prior to any action occurring. Once they have chosen, one of the agents is chosen at random to take their action first.

If an agent attempts to move into a space that is occupied, the agent does not move. If an agent has possession of the ball, is chosen to move *second*, and attempts an action that would move it into a space that is occupied by the other agent, the agent gives its ball to the other agent and does not

move. This is the only way the ball can change possession. Aside from the constraints presented, all actions are deterministic. In this environment, each state is determined by the positions of all players as well as who has possession of the ball.

[1] *When presenting tuples representing coordinate space in this paper, the row will be first, and the column second.*

III. Algorithms

It is readily visible that these multiagent problems do not have a purely deterministic strategy that is dominant. If an agent were to always pick a single action, the other agent could recognize this and exploit their determinism. A mixed strategy, that is one where each possible action is assigned a probability that is then chosen from stochastically, is required to find an equilibrium. Another way of saying this is that there is no Nash Equilibrium wherein both agents will never want to change their pure strategy regardless of the actions or strategy of the other agent. There are however *correlated equilibria* that will allow for convergence. A correlated equilibrium (CE) is more general than a Nash equilibrium in that it allows for dependencies on the strategies of other agents through signals indicating the probable actions of those agents. One may consider a stop-light one such signal for a CE in that it does not dictate the actions of other drivers but it does allow for coordination by understanding the probable actions of those drivers (driving through a red light would have a low expected outcome for either party).

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 α according to S

Fig 2. Pseudocode for setting up multiagent Q-learning [2]

The general structure of the agent learning uses Q-learning based upon the Bellman equation. As the agents take actions, and they opponents do, they record the results of these pairings as a means of predicting what the probably rewards of those actions in those states will be in the future. The agents start by picking actions at random in an exploration phase, controlled by the epsilon hyperparameter, and epsilon slowly decays over time to lean more towards exploiting the strategies with the highest recorded expected discounted reward (EDR). The amount that the EDR is factored into the prediction of a state's value is controlled by the lambda hyperparameter. The amount that a Q value is influenced when learning is determined by the alpha hyperparameter. In this implementation, as defined by Greenwald/Hall, the alpha value is equal to (1 / the number of times that state was visited), which allows policies to stabilize over time and they change less drastically. The initial hyperparameters presented by Greenwald/Hall are alpha = 1, epsilon = 1 (except for the pure Q-learning strategy where it is 0.01), and lambda = 0.9. Minimum values are 0.001 for epsilon and alpha. Greenwald/Hall nor Littman define a rate of decay for epsilon, but through experiment, 0.001 was found to be an acceptable value. Epsilon would then be recalculated every iteration as 1 - exp(epsilon_decay - number of iterations), with a minimum value of 0.001.

The 4 different Q-learning strategies implemented for testing are Correlated-Q (CE-Q), Foe-Q (Minimax), Friend-Q, and Q-Learning.

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

Fig. 3 Equation for utilitarian correlated-Q algorithm

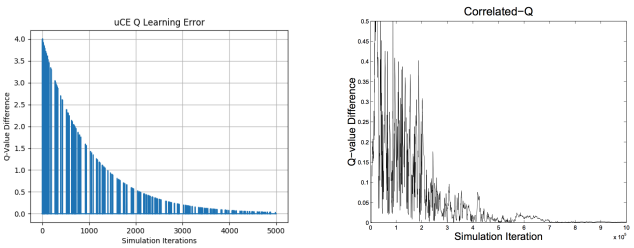

Fig. 4 Comparison of produced results for correlated-q algorithm (left) and results from Greenwald/Hall 2003 [2] (right)

A. Correlated-Q

Correlated-Q is presented with different variants that use different equilibrium selection mechanisms in Greenwald/Hall's paper, however in this paper we will focus on the use of the "utilitarian" variety which seeks to maximize the sum of all agent's rewards. This is accomplished by summing the product of each probability of choosing each action with the EDR of that action. This is done for both players and passed back in one common policy. The optimal probability of picking a given action is determined using the EDR through linear programming, discussed in Section IV. The differences in the graphs are that my curve seems much smoother which I suspect could be because I'd used a linearly decaying alpha value, whereas Greenwald/Hall used an alpha that was dependant on the number of times a Q state-action pair was visited. Convergence seems to be around the same time in both. My errors are also much higher, perhaps they'd used some kind of normalization. Also, there are gaps between the values in my graph, leading me to believe that they did not include zero values in theirs, or perhaps if a value was zero they'd used the previous error for smoothing.

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

Fig. 5 Equation for foe-Q algorithm [2]

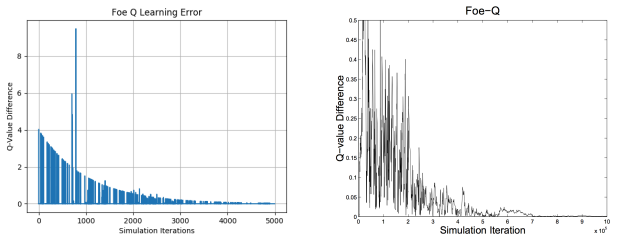

Fig. 6 Comparison of produced results for foe-q algorithm (left) and results from Greenwald/Hall 2003 [2] (right)

B. Foe-Q

Foe-Q uses the minimax strategy wherein each agent tries to pick the action that will give them the highest EDR while expecting that their opponent will choose the action that will give them the lowest EDR for all possible actions. This is similar to Correlated-Q, and in fact achieves similar rewards. My results seem again to be much smoother and the same thoughts lie with these as the prior uCE.

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

Fig. 7 Equation for friend Q

$$V^*(s) = \max_{a \in A(s)} Q^*(s, a)$$

Fig. 8 Equation for regular Q-learning

IV. Linear Programming

In this environment, each state is determined by the positions of all players as well as who has possession of the ball. This would mean that 8 grid spaces * 7 other grid spaces * 2 possible ball possessors = 112 possible states. For each of these states, there are then 25 actions represented in the Q table per state for each agent, because for each agent's 5 possible actions, the agent must also consider each of the other agent's 5 possible actions. 112 states * 25 Q values per state * 2 agents = 5600 possible Q values to manipulate for this very small grid size.

Because of the complexity that this problem presents, linear programming is critical to the computability of correlated learning scenarios such as this. The role of the LP is to give the probabilities that are being signalled to the agents, not to actually determine the actions directly for the agents. A linear program minimizes some objective function using a set of constraints, either as inequalities or equations. The objective function (c) contains a set of variables that are being solved for. When determining the inequality constraints for a zero-sum, 2 player matrix game, one should visualize each cell of the matrix to have its own probability.

	N	S	E	W	Stay
N	p1	p2	p3	p4	p5
S	p6	p7	p8	p9	p10
E	p11	p12	p13	p14	p15
W	p16	p17	p18	p19	p20
Stay	p21	p22	p23	p24	p25

Fig. ?? Matrix of agent A (rows) and agent B (cols) with probabilities to be used for constructing constraints for linear program solver

Each of these probabilities are a variable in the objective function. The constraints then are ensuring that for a given action for either agent, every other action that that agent could take is less rewarding. For example, when writing constraints for the row player, one would check that the rewards expected, through the agent's Q table, for a given action, are more than that of each other action. So to check N against S for the the row agent (A), the equality would be: p1*R1(N,N) + p2*R1(N,S) + p3*R1(N,E) + p4*R1(N,W) + p5*R1(N,Stay) ≥ p1*R1(S,N) + p2*R1(S,S) + p3*R1(S,E) + p4*R1(S,W) + p5*R1(S,Stay) [7]. This needs to happen for each other action (E,W,S) against N. There are in total, then, 20 constraints for each player, and then 40 in total. Each of these needs to be put into standard form as well, that is setting them against 0 on the right side. In addition there are constraints ensuring that all values are ≥ 0 and sum in total to 1. Once all of these conditions are met, the solver can create a convex hull from which to find an equilibrium payoff matrix that can be used to determine the expected rewards for our agents. Note that for this paper, the excitedAtom [9] solver was used.

References

[1] C. Daskalakis, " Topics in Algorithmic Game Theory, Lecture 2," 08-Feb-2010. [Online]. Available: http://people.csail.mit.edu/costis/6896sp10/lec2.pdf. [Accessed: 14-Apr-2019].

[2] A. Greenwald and K. Hall, "Correlated-Q Learning," ICML'03 Proceedings of the Twentieth International Conference on International Conference on Machine Learning, pp. 242–249, Aug. 2003.

[3] A. Greenwald, K. Hall, and M. Zinkevich, "Correlated Q-Learning," Jul. 2005.

[4] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning ," In Proceedings of the Eleventh International Conference on Machine Learning, pp. 157–163, 1994.

[5] M. L. Littman, "Friend-or-Foe Q-learning in General-Sum Games," In Proceedings of the 18th Int. Conf. on Machine Learning, 2001.

[6] A. Pilkington, "Strategy Lines and Optimal Mixed Strategy for R," 06-Nov-2018. [Online]. Available: https://www3.nd.edu/~apilking/Math10120/Lectures/Topic 28.pdf. [Accessed: 14-Apr-2019].

[7] https://web.archive.org/web/20170829232257/http://www3.ul.ie/ramsey/Lectures/Operations_Research_2/gametheory4.pdf

[8] https://github.com/axonal/cvxopt-tutorial/blob/master/Linear%20Programming.pdf

[9] https://github.com/excitedAtom/blogs/blob/master/cvxopt/cvxopt_examples.py