Correlated Q Learning*

*Hash: faae53188d213c01ff8579d14284e8137de539aa

1st Sizhang Zhao (szhao334)
Computer Science Department
Georgia Tech

***Abstract*—MDP can be generalized into game theory when single agent setting changes to multi-agent settings. Dependent on how agent choose for equilibrium actions, several different models can be used. In this paper, we will take a simple environment, soccur game, as an example and implement 4 different multi-agent Q-learning methods, Q learning, foe Q learning, friend Q learning and correlated Q learning and analyze their convergence behavior.**

***Index Terms*—game theory, multi-agent, Q Learning, Friend-Foe-Learning, Correlated Q Learning**

I. Introduction

Game theory cares for equilibriums. Under multi-agent setting, behavior for agents will stay consistent (aka convergence when it talks from algorithm perspective) when they come to equilibrium over time. However, not all of the policy will guarantee equilibrium exists. Four different algorithms with different policy will be presented in this paper under Robocup soccur environment and their convergence bahaviror will also be analyzed.

II. Theory

A. Nash Equilibrium

Nash equilibrium is achieved when no player can do better by unilaterally changing his or her strategy. Mathmatically, it can be defined for n agent as,

$$U_i(a_i^*, a_{-i}) \geq U_i(a_i, a_{-i}) \qquad \forall\ 0 \leq i \leq n, a_i \in \mathcal{A}_\rangle \quad (1)$$

where U_i is the utility function of ith agent and $\mathcal{A}_\rangle$ is his action space, a_i^* is the equilibrium action chosen. It can be easily found that at equilibium, agent i has no incentive to change its action no matter what actions others take.

Also note that if we define utility as the expected utility given a mixed strategy profile s rather than a deterministic function, i.e.

$$U_i(s) = \sum_{a \in \mathcal{A}} U_i(a)\mathbb{P}(a|s) \quad (2)$$

then based on the defination, we can get mixed strategy nash equilibrium, where single agent no longer take a pure strategy in terms of actions but they rather take a probability over the action space.

Nash proved that if we allow mixed strategies (where a pure strategy is chosen at random, subject to some fixed probability), then every game with a finite number of players in which each player can choose from finitely many pure strategies has at least one Nash equilibrium. However, this is no uniqueness guarantee for Nash Equilibirum, meaning there might exists more than one Nash equilibrium.

B. Correlated Equilibrium

Correlated equilibrium is more general than nash equilibrium. Public signal is incorporated into the model. Each agent no longer only assign probability to his or her action space but assigns an action to every possible observation a player can make. And similar to Nash Equilibrium, if no player would want to deviate from the recommended strategy (assuming the others don't deviate), the distribution is called a correlated equilibrium. Mathematically, it can be presented as,

$$\sum_{s:s_i=\bar{s_i}} p(s)U_i(s) \geq \sum_{s:s_i=s_i'} p(s)U_i(s) \quad (3)$$

where $s = (s_1, ..., s_n)$ as the joint mixed strategy, and $\bar{s_i}$ means the equlibrium strategy of agent i.

Correlation Equilibrium is similar to mixed strategy Nash Equilibrium and only differs in mixed strategy Nash Equilibrium models marginal probability distribution but Correlation Equilibrium models the joint probability distribution among agents.

C. MDP Games and Q Learning

MDP is the nothing different from a one agent Markov game, and we can maintain Q table for learning and planning. The Q table can be calculated by bellman equation,

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} \mathbb{P}(s'|s,a)V(s') \quad (4)$$

$$V(s) = max_{a \in \mathcal{A}(s)} Q(s,a) \quad (5)$$

To get a good estimate of Q table, we usually use Q Learning with update formula,

$$\begin{aligned} Q(S_t, A_t) =& Q(S_t, A_t) + \\ & \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \end{aligned} \quad (6)$$

For a more general Markov game with more than one agent, the single update as show in equation (5) that agent simply choose the action which have the maximum Q value might no longer works, since actions satisfy these simultaneous equations need not exist under multi-agent settings.

However, if we become flexible for how to choose policys and calculate Q value, bellman equation and thus Q learning can be generalized to multi-agent settings, in general form, it can be presented as,

$$V_i(s) = Nash_i(Q_1(s), ..., Q_n(s)) \tag{7}$$

We will be presenting 4 different choices in the later sections.

III. Experiment

A. Robocup Soccer Game

Soccer game is a grid game, it can be presented in figrue 1. There are totally 8 grids in the game, and

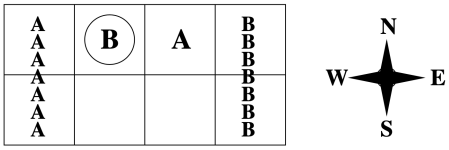

Fig. 1. Soccer Game

two players will be playing and one single ball will either belong to player A or player B. The starting state is player A stays in (0,2) and player B stays in (0,1) and ball belongs to B. Each player can choose from 5 actions, N, S, E, W and stick. For each move, the actions from players are executed in random order. And when a player executes an action that would take it to the square occupied by the other player, possession of the ball goes to the stationary player and the move does not take place. If any player bring the ball to the goal area of A, A scores 100 and B scores -100, and similarly if the ball go into the area belongs to goal area of B, B scores 100 and A scores -100. Please note there is no deterministic equilibrium policy for the game.

The algorithm for implementation of soccer game is shown in algorithm 1 and used move function are shown in algorithm 2.

B. Q Learnings

As we briefly mentioned the generalization of bellman equation to multi-agent settings, we will go into deeper descriptions in the section for all the four models we will analyze later on.

Algorithm 1: Agent Step Process

Result: Step
initialization:
current position for two players
current ball belongings
legal movement
goal place
Input [actions from two players]
Output [A position, B position, ball belongings, rewards, done]
first move agent = random()
(A position, B position, ball belongings, rewards, done) = move(first move agent, first move agent action)
if *done* **then**
  return *[A position, B position, ball belongings, rewards, done]*
end
else
  (A position, B position, ball belongings, rewards, done) = move(second move agent, second move agent action)
end
return *[A position, B position, ball belongings, rewards, done]*

Similar to the generalization to bellman equation, we can also generalize Q learning to multi-agent setting, so instead of having equation (6), we will have,

$$\begin{aligned} Q(S_t, A_t) =& Q(S_t, A_t) + \\ & \alpha[R_{t+1} + \gamma V_i(s') - Q(S_t, A_t)] \end{aligned} \tag{8}$$

where $V_i(s')$ is defined similar in equation (7), will be the expected value of next steps with the policy chosen.

To research into the convergence behavior of different policies. We will use the same method as Greenwald [1]. $ERR_i^t = |Q_i^t(s, \vec{a}) - Q_i^{t-1}(s, \vec{a})|$ will be used to track difference between Q table over time. In our experiments, we will always choose the initial state with agent A choose S and agent B choose to stick as the tracking Q entry. The general Q learning algorithms is shown in algorithm 3.

The difference between different Q learning is mainly in how the policy is defined, aka how do agent choose actions based on the current Q values and how does next state value estimated. We will introduce the 4 different ones here, Q learning, friend Q learning, foe Q learning and correlated Q learning.

1) Q Learning: Q value is the most basic one, it is the same as we defined in equation (6) and value got estimated by equation (5), action is taken by,

$$a = argmax_{a \in \mathcal{A}(s)} Q(s, a) \tag{9}$$

Algorithm 2: Move Function for single agent

Result: Move
Input [agent, action]
Output [A position, B position, ball belongings, rewards, done]
if *Collison happend and agent has the ball* **then**
  ball possetion change
end
else if *Action didn't make agent out of grids* **then**
  agent move
  if *agent didn't have the ball* **then**
    return *[A position, B position, ball belongings, rewards, done]*
  end
  if *agent move into goal area* **then**
    update reward based on which goal area belongs to
    done = True
    return *[A position, B position, ball belongings, rewards, done]*
  else
    return *[A position, B position, ball belongings, rewards, done]*
  end

This is a determinsitic pure stratrgy, which means π in our general framework doesn't fit in here. Agent won't take actions based on any probability distribution. And also in multi-agent setting, this means we don't have equilibrium for the game.

There is another similar algorithm, Sarsa, it is an on policy algorithm. The difference between Q learning and Sarsa is that the value estimator of Sarsa comes directly from Q value of next state other than that from equation (5).

2) Friend Q Learning: Littman [2] proposed a friend Q learning algorithm, not only they prove the algorithm coverges, but also they shows that as long as condition: There exists a coordination equilibrium for the entire game and for every single game defined by the Q functions encountered during the learning process, is satisified, Friend Q can learn through the Nash Equilibrium policy. Mathematically, this means,

$$\begin{aligned} V_i(s) &= Nash_i(Q_1(s,\vec{a}), Q_2(s,\vec{a})) \\ &= max_{a_1 \in \mathcal{A}_1(s), a_2 \in \mathcal{A}_2(s)} Q_1(s, a_1, s_2) \end{aligned} \tag{10}$$

and

$$a_1 = argmax_{a_1 \in \mathcal{A}_1(s), a_2 \in \mathcal{A}_2(s)} Q_1(s, a_1, s_2) \tag{11}$$

This is a simple generalization of the one agent Q learning in that we also take actions from other agents into consideration. We naively assume the other agent will help us achieve the highest utility and choose actions based on the joint Q value.

Algorithm 3: General Q Learning Framework

Result: Q Learning
initialization:
α, γ, ϵ, their min value and decay, number of iteration
$Q_a, Q_b, V_a, V_b, \pi_a, \pi_b$, errors, i = 0
env = Soccer Game
while $i \leq$ *number of iteration* **do**
  state = env.reset()
  update π, V with the new state
  while *True* **do**
    record tracking Q value
    choose action with ϵ-greedy and based on policy with π
    (A position, B position, ball belongings, rewards, done) = env.step()
    state new = (A position, B position, ball belongings)
    if *done* **then**
      update Q table with V(s) = 0
    else
      update Q table with equation (8) state = state new
    end
    record tracking Q value and get the difference before and after move
  end
end
plot error

3) Foe Q Learning: Littman [2] also propsed another foe Q learning algorithm. Similarly, if condistion: There exists a adversarial equilibrium for the entire game and for every single game defined by the Q functions encountered during the learning process, is satisified, Foe Q can learn through the Nash Equilibrium policy. And mathematically, this means:

$$\begin{aligned} V_i(s) &= Nash_i(Q_1(s,\vec{a}), Q_2(s,\vec{a})) \\ &= \max_{\pi \in \Pi(\mathcal{A}_1(s))} \min_{a_2 \in \mathcal{A}_2(s)} \sum_{a_1 \in \mathcal{A}_1(s)} \pi(a_1) Q_1(s, a_1, s_2) \end{aligned} \tag{12}$$

And the agent will no longer choose the pure strategy, rather he will use mixed strategy based on π. And the policy is chosen based on min max algorthm so a LP can solve π and min max value simultanously after we fed the Q table for a given state.

This further generalized Q learning and now we assume our opponent is completely adversarial and he will try to decrease our utility as much as possible. So what the agent does is to maximum the utitility under the worst scenario.

4) Corrlated Q Learning: We already discussed about correlated equilibrium in previous sections. The idea of

correlated equilibrium can be generalized into Q learning as well. Similar to Foe learning, we will also need to model probability distribution but not on action space of single agent but rather on joint action spaces of all the agents. Greenwald [1] proposed 4 different correlated Q learning, utilitarian(uCE-Q), egalitarian(eCE-Q), republican(rCE-Q) and libertarian (lCE-Q). And the difference between them are mainly for how the equilibrium is calculated. All of them can be calculated by LP efficiently. We take uCE-Q as an example in this paper and it can be represented mathematically as,

$$\pi \in argmax_{\pi \in CE} \sum_{i \in I} \sum_{\vec{a} \in \mathcal{A}} \pi(\vec{a}) Q_1(s, \vec{a}) \tag{13}$$

s.t.

$$\sum_{\bar{a} \in \mathcal{A}_{-p}} (Q_p(s, a_p^i, \bar{a}) - Q_p(s, a_p^j, \bar{a}))\pi(a_p^i, \bar{a}) \geq 0 \quad \forall p,\ \forall i, j \in \mathcal{A}_p \tag{14}$$

$$\sum_{\vec{a}} \pi(\vec{a}) = 1 \tag{15}$$

$$\pi(\vec{a}) \geq 0 \quad \forall \vec{a} \in \bar{\mathcal{A}} \tag{16}$$

So each agent are targeted to maximize the combined utility among them and while they shouldn't have motivation to move away from the current strategy they are following.

C. Results

1) General Settings: For the later sections, we all assume a constant $\gamma = 0.9$, learning rate starts with 1.0 and eventually decay to 0.001 and ϵ starts as 1.0 and eventually decay to 0.001 and number of iteration is 1000000, same as descrived in the original paper.

2) Q Learning and Sarsa: We firstly present the results for off policy Q learning, result is shown in Figure 2. Greenwald [1] also mentioned they are using on policy

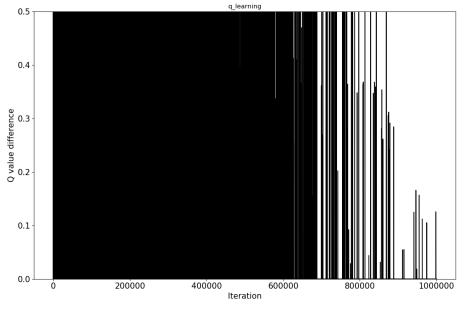

Fig. 2. ERR_i^t with respect to number of iteration of Q Learning

Q learning, so Sarsa is also implemented and result is shown in Figure 3. Both of them looks similar to the results in the original paper. Even though it looks like the

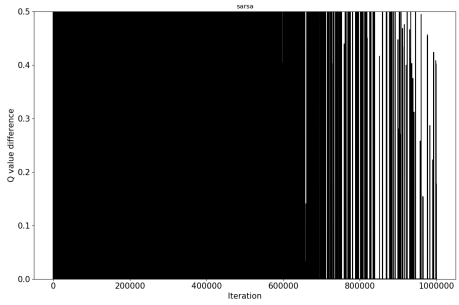

Fig. 3. ERR_i^t with respect to number of iteration of Sarsa

learning algothms started to converge eventually (which is more obvious in q learning in figure 2). But as stated in the orginal paper, it's mainly because the decay of learning rate. Q learning is suitable for static enviroment one agent learning but when there are another agent, the environment becomes noisier. And we also know there is no deterministic equilibrium for the soccer game, so Q learning should never actually converge.

Also please note the learning rate play a key role in the replication, choosing a more suitable learning rate decay strategy can help the results look much more similar to the original paper.

3) Friend Q Learning: The ERR_i^t changes for Friend Q learning is presented in figure 4. The convergence of

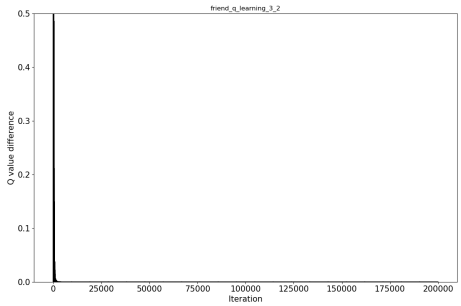

Fig. 4. ERR_i^t with respect to number of iteration of Friend Q Learning

Friend Q learning is super fast and thus only the first 200000 iteration is presented. However, as Greenwald pointed out the equilibrium of friend Q learning is irrational as each agent assume his opponent to score for himself.

Same as the findings as in q learning, learning rate decay (also epsilon decay) plays an importent role for the shape of the chart. Choosing a different decay strategy might help my chart converge in a more similar speed as in the original paper.

4) Foe Q Learning: The ERR_i^t changes for Foe Q learning is presented in figure 5. Foe Q learning also converges as we already briefly discussed in the early

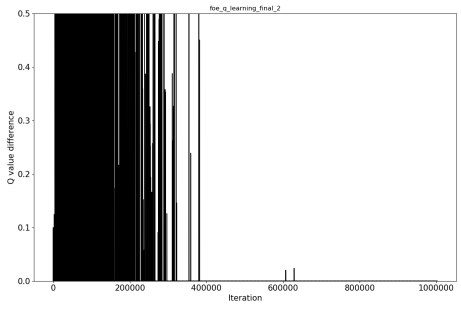

Fig. 5. ERR_i^t with respect to number of iteration of Foe Q Learning

sections (Foe Q follows the convergence property of minimax equilibrium and thus it's guaranteed to converge). And the convergence speed are similar to the original paper, though it looks like mine converges a slightly faster.

Still, learning rate decay (also epsilon decay) is the key. With a linear decay initially, the convergence is faster. It needs careful choice of dacay algorithms to make sure the final chart looks more similar to the original paper.

5) Corrlated Q Learning: The ERR_i^t changes for Correlated Q learning is presented in figure 6. The

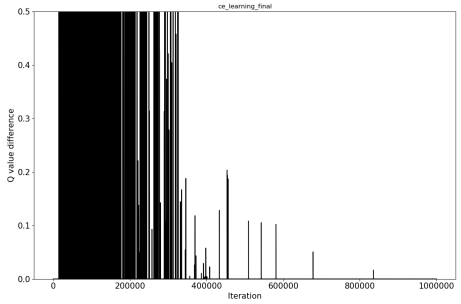

Fig. 6. ERR_i^t with respect to number of iteration of Correlated Q Learning

convergence behavior is very similar to foe Q learning if same decay strategy is given. As described in the original paper, both learning algorithm will converge for both agents, where each one of them randomizes between stick and head north. Also, under 2 players, zero sum settings, correlated Q learning is the same as minimax equilibrium and that's also why correlated Q learning have very similar convergence property with Foe learning. But please note at each step, Foe learning needs 2 separate LP to figrue out policy for each agent while correlated Q learning will only need one overall LP for the joint actions, so correlated Q learning runs faster.

Still same as the previous sections, learning rate and epsilon decay are the key to resemble the original paper.

IV. Conclusion

We presented 4 different Q learning methods under a simplied game - Robocup Soccer, where there are two players interacting with the environment. We also analyzed the convergence behevior of them and found Q learning, which is suitable for one agent single agent setting, didn't really converge. Friend Q learning which assume the opponent will help with the agent will converge but to a irrational equilibrium. Both Foe Q learning and Correlated Q learning will converge and come to the equilibrium same as minimax equilibrium.

References

[1] Greenwald, A., & Hall, K. (2003). Correlated-Q learning. Paper presented at the Proceedings of the Twentieth International Conference on International Conference on Machine Learning, Washington, DC, USA.

[2] Littman, M. L. (2001). Friend-or-Foe Q-learning in General-Sum Games. Paper presented at the Proceedings of the Eighteenth International Conference on Machine Learning.

[3] Amy Greenwald, K. H., Martin Zinkevich. (2005). Correlated QLearning. Retrieved from Brown University