CS7642 Project 3 – Solving Soccer Game with Q-Learning, Friend/Foe-Q and Correlated-Q

Qinghui Ge

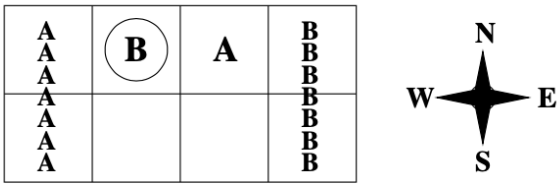


Fig. 1. Soccer game, state s (Figure 4 or Greenward 2003 paper [1])

***Abstract*—In this project, we try to reproduce Figure 3 of Amy Greenward's paper, *Correlated Q-learning*, 2003 [1]. Correlated Q-learning is an algorithm tries to find the correlated equilibrium in Markov games. Correlated-Q generalize Nash-Q and Friend/Foe-Q, and can be solved with linear programming. In general sum games, correlated equilibria contains all Nash equilibria, while in constant sum games correlated equilibria contains minimax equilibria. We implemented Correlated-Q, as well as regular Q-learning, Friend-Q, Foe-Q and compare their behavior in solving soccer game — a zero sum game. We show that in this particular game, Correlated-Q reaches the save solution as Foe-Q.**

git hash:

I. Soccer Game

The soccer game is a two-player zero-sum game. On a 4×2 grid, there are two players A,B and a ball. For each player, the objective is to carry the ball to his or her designated goal area and receive +100 reward. Carrying the ball to opponent's area or allow the opponent to score would receive -100 reward. There are five possible actions for each player: N, S, E, W and Stick. Fig.1 shows the initial state of soccer game.

There is certain ambiguity in the game rules described in Greenward's paper [1]. We implement the rule as following: First determine which player (A or B) moves first, then execute two players' actions based on the random order. For each player, if the next position do not collide with other player's current position, he can move to the new position, otherwise, if he losses ball to the other player(if he has it). After executing two actions, check if the termination criterion is met and assign rewards. The game is implemented like an Opengym environment, with the "step"-like function defined in Alg.1. The *step* function takes a tuple of action $\mathbf{a}$=(a_A, a_B), and returns the state index based on two players position and ball ownership, reward as a tuple $\mathbf{R}$=(R_A, R_B), and boolean variable *done* to indicator whether the game has ended. For the soccer game, there are $8 \times 7 \times 2 = 112$ states.

Algorithm 1 soccer game environment implementation

function ENV.STEP($\mathbf{a}$=(a_A, a_B))
  determine player move order
  get first player proposed move
  if no collision **then**:
    first player move
  else if first player has ball **then**
    pass ball to second player
  end if
  get second player proposed move
  if no collision **then**:
    second player move
  else if second player has ball **then**
    pass ball to first player
  end if
  if ball in player A's goal area **then**: ▷check game status
    $\mathbf{R}$, done = (100,-100), True
  else if ball in player B's goal area **then**:
    $\mathbf{R}$, done = (-100,100), True
  else
    $\mathbf{R}$, done = (0,0), False
  end if
  Return s(pos_A, pos_B, ball), $\mathbf{R}$, done
end function

II. Learner implementation

We follow a general off-policy learning template (Alg. 2) for all training process, each learner (Q, Friend-Q, Foe-Q, CE-Q) has to implement it's own way of action selection (for ϵ-greedy to call if greedy action is needed), and the function to update Q values. The general form of updating Q values of agent i is:

$$Q_i(s, \mathbf{a}) = (1-\alpha)Q_i(s, \mathbf{a}) + \alpha\big((1-\gamma)R_i + \gamma V_i(s')\big) \quad (1)$$

where α is learning rate, γ is discount factor. $\mathbf{a} = (a_1, a_2 \dots a_N)$ is a combination of all agent actions (although we will see in Q-Learning each agent only need to track the Q-table of its own action, while for other learners each agent also keep track of other agent's actions). Each learner defines state value function $V_i(s)$ differently.

Our implementation of Foe-Q is currently restricted to only two agents, while Q-Learning, Friend-Q, and CE-Q are generalized to handle multi-agents, with each agent potentially having different action choices. The joint action space of

all agent is $\mathbf{A} = \{A_1, A_2 \dots A_N\}$. As Littman has point-out, Friend/Foe-Q can be combined into the same frame framework, with some agents as friend, and some agents as foe [2].

In Greenward's 2003 paper [1], it is unclear how learning rate is decayed, and whether Friend/Foe-Q, CE-Q used ϵ-greedy for action exploration. In Greenward's 2005 extented paper [3], the authors decay learning rate according to the number of times each state-action pair is visited, i.e. $\alpha(s, \mathbf{a}) = 1/n(s, \mathbf{a})$. The 2005 paper also employs $\epsilon = 1$ (random action selection) for Friend/Foe-Q, CE-Q. We implemented this procedure described in Greenward's 2005 paper, as well as other possibilities such as decaying ϵ and α exponentially.

Algorithm 2 off_policy_learning template

```
function OFF_POLICY_LEARNING(learner, env)
    for each episode do
        reset environment, get current state s
        for t in episode do
            a = ε-greedy(learner, env, s, ε)
            take action a, observe s', R, done
            learner.update(s, s', a, R done, α(s,a), γ)
            decay α, e.g. α(s,a) = 1/n(s,a)
            optional: decay ε, e.g. ε *= ε_decay
            s = s'
        end for
    end for
end function
```

A. Q-Learning

In Q-Learning implementation, each agent i maintains its own Q-table of size $nS \times nA_i$. The greedy action is each agent taking it's own greedy action, i.e:

$$\mathbf{a}^* = (a_1^*, a_2^* \dots a_N^*)$$
$$a_i^* = argmax_{a_i} Q_i(s, a_i), a_i \in A_i$$

The value function used in Eq.1 is:

$$V(s) = max_{a_i} Q_i(s, a_i) = Q_i(s, a_i^*) \tag{2}$$

B. Friend-Q

In Friend-Q, each agent maintain a copy of Q-table of size $nS \times nA_1 \times \dots nA_N$. Each agent assumes other agents will corporate to maximize the Q values, and the greedy action is defined as:

$$\mathbf{a}^* = (a_1^*, a_2^* \dots a_N^*)$$
$$a_i^* = argmax_{\mathbf{a}} Q_i(s, \mathbf{a})[i], \mathbf{a} \in \mathbf{A}$$

The value function is defined as:

$$V(s) = max_{\mathbf{a}} Q_i(s, \mathbf{a}) = Q_i(s, \mathbf{a}^*) \tag{3}$$

C. Foe-Q

In Foe-Q learning, each agent maintain a of Q-table of size $nS \times nA_{opponent} \times nA_{self}$. The agents can follow a mixed policy: each agent i takes action a_i with probability $\sigma_i(a_i)$. The greedy policy is obtained from solving the minimax problem of σ:

$$\sigma_i^*(a) = max_{\sigma_i} min_{a_o \in A_{opponent}} \sum_{a_i \in A_{self}} Q_i(a_o, a_i)\sigma_i(a_i) \tag{4}$$

and the greedy action can be sampled from the minimax probability: $a_i^* \sim \sigma_i^*(a)$.

Let the value function $V_i = min_{a_o} \sum_{a_i} Q_i(a_o, a_i)\sigma_i(a_i)$, the minimax equation can be formulated in the following linear programing problem:

maximize V_i

1. nA_o inequality constraint:
   $V_i - \sum_{a_i \in A_i} Q_i(a_o, a_i)\sigma_i(a_i) \leq 0, \forall a_o \in A_{opponent}$
2. nA_i inequality constraint : $\sigma_i(a_i) \geq 0, \forall a_i \in A_i$
3. 1 equality constraint: $\sum_{a_i \in A_i} \sigma_i(a_i) = 1$ (5)

The second inequality constraint and equality constraint are simply probability restrictions. Foe-Q follows Alg.3 to solve the minimax problem and update Q values. Note we first update $Q(s)$ with $V(s')$ then solve for $V(s)$ with the updated $Q(s)$, alternatively we can solve $V(s')$ from $Q(s')$ first, then update $Q(s)$. I believe the order shouldn't matter – more like a chicken or egg problem. But with the first way we can initialize $V = 0$ and no need to treat terminal state specially, the second way we don't need to save V in memory.

Algorithm 3 foe-Q update

```
function FOE-Q.UPDATE(s, s', a, R)
    Q1(s, a2, a1) = (1−α)Q1(s, a2, a1) + α((1−γ)R1 + γV1(s'))
    Q2(s, a1, a2) = (1−α)Q2(s, a1, a2) + α((1−γ)R2 + γV2(s'))
    V1(s), σ1(s) = minimax(Q1(s))
    V2(s), σ2(s) = minimax(Q2(s))
end function
```

D. Correlated-Q

In CE-Q, each agent maintain a copy of Q-table of size $nS \times nA_1 \times \dots nA_N$. All agents follows a mixed policy based on join probability $\sigma(\mathbf{a}) = \sigma(a_1, a_2 \dots a_N)$. The joint probability has to satisfy rational constraint as well as regular probability constraint. For utilitarian CE-Q, the objective function is to maximize the sum of the agents' rewards, and the uCE-Q can be formulated in the following linear programing problem:

$$\text{maximize} \sum_{i}^{n_{agent}} \sum_{\mathbf{a}\in\mathbf{A}} Q_i(\mathbf{a})\sigma(\mathbf{a})$$

1. Each agent contribute $nA_i(nA_i - 1)$ rational constraint:
$$\sum_{\mathbf{a_o}} Q_i(\mathbf{a_o}, a_i')\sigma(\mathbf{a_o}, a_i) \leq \sum_{\mathbf{a_o}} Q_i(\mathbf{a_o}, a_i)\sigma(\mathbf{a_o}, a_i)$$
$$\forall a_i \neq a_i' \in A_i$$
$$\mathbf{a_o} \in \{A_1, A_2 \dots A_{i-1}, A_{i+1} \dots A_N\}$$
2. $\Pi_i nA_i$ inequality constraint : $\sigma(\mathbf{a}) \leq 0, \forall \mathbf{a} \in \mathbf{A}$
3. 1 equality constraint: $\sum_{\mathbf{a}\in\mathbf{A}} \sigma(\mathbf{a}) = 1$ (6)

The rational probability can be interpret as following: assuming there is a referee recommend all agent to act according to joint probability distribution $\sigma(\mathbf{a})$. Assuming all other agent follows the referee's advise, for agent i, the expected reward for playing the recommended action a_i is $\sum_{\mathbf{a_o}} Q_i(\mathbf{a_o}, a_i)\sigma(\mathbf{a_o}, a_i)$, while for playing another action a_i' $\sum_{\mathbf{a_o}} Q_i(\mathbf{a_o}, a_i')\sigma(\mathbf{a_o}, a_i)$. The rational constraint simply says, at correlated equilibrium, an agent receives maximum reward if he follows referee's recommendation, given other agents also follow the referee.

After solving the above LP and obtain the joint probability, the state value function can be calculated as: $V_i(s) = \sum_{\mathbf{a}} Q_i(s, \mathbf{a})\sigma(s, \mathbf{a})$. The uCE-Q updating procedure is summarized in Alg.4.

Algorithm 4 Correlated-Q update

function CE-Q.UPDATE(s, s', **a**, **R**)
  for i = 1,2... n_{agent} **do**
    $Q_i(s, \mathbf{a}) = (1-\alpha)Q_i(s, \mathbf{a}) + \alpha\big((1-\gamma)R_i + \gamma V_i(s')\big)$
  end for
  $\sigma(s, \mathbf{a}) = uCE(\{Q_1(s), Q_2(s) \dots Q_N(s)\})$
  for i = 1,2... n_{agent} **do**
    $V_i(s) = \sum_{\mathbf{a}} Q_i(s, \mathbf{a})\sigma(s, \mathbf{a})$
  end for
end function

III. Experiment Results

While training all learners, we keep track of the Q value of state s (as show in Fig.1), with player A taking action S and player B sticking to reproduce figure.3 in Greenward 2003 paper. Since in state s, taking action N would bump into the wall and result in stick effectively, we will add up the Q value of player B taking N and stick together. It is unclear to me whether the iteration in x-axis means each time step or each episode in Greenward's paper, but given the large number I would assume it is every time step – although this doesn't make material difference that affect our conclusion.

For Q-Learning, we decay $\alpha \to 0.001$ and $\epsilon \to 0.001$ exponentially with decay rate 0.99995. For Friend-Q, Foe-Q and CE-Q, we decay α according to the number of visits to each state-action pairs: $\alpha(s, \mathbf{a}) = 1/n(s, \mathbf{a})$, and use random exploration for choosing actions ($\epsilon = 1$). All learner are trained for 200000 iterations. The discount factor $\gamma = 0.9$.

Q-Learning does not converge. In Fig.4(a), we can see the difference in Q-value decreases at the beginning, but this is the decaying of learning rate. When the learning rate reduce to minimum (0.001), we can see there is constant fluctuation of the error. Q-Learning tries to find a pure policy, while for this game, the equilibrium is a mixed policy.

From Fig.4(b), we can see the Q values converges quickly for friend-Q. However, the resulting actions are not rational. From the Q-table of state s, we can see for player B, action E dominate other actions, so player B would move right and expect player A to move aside or score for him. The best Q-value for player B is both A and B take action E, so there is 50% chance B can score +100 in the current round. While the Q-table for player A has $Q_A[:, W] = 10$, this means A can take any action and expect B to move W and score for him. Note the maximum possible Q-value ± 10 results from the $(1-\gamma)R_i$ term when we have $\gamma = 0.9$.

Foe-Q and uCE-Q also converges. If the initialization are the same and apply Greenward 2005 paper settings (decay α according to $n(s, a)$, random action exploration), the learning curves are nearly identical for both learners. The final Q-value difference are with in 10^{-7} and both learner gives a mixed policy with both player randomly choose between stick and heading south: $\sigma_A(stick/N) = 0.423, \sigma_A(S) = 0.577, \sigma_B(stick/N) = 0.606, \sigma_B(S) = 0.394$. Note for uCE-Q each players policy is marginal probability computed from joint probability $\sigma_i(a_i) = \sum_{\mathbf{a_o}} \sigma(\mathbf{a_o}, a_i)$.

However, if I decay α exponentially in action exploration, I am unable to converge Foe-Q and uCE-Q to the exact Q-values. I am uncertain if this is because the hyperparameters are not optimal or undetected bugs in my code. I assume having a state-action dependent learning rate might be more beneficial so that less frequently visited states can be updated effectively. On the other hand, I verified that when passing a same Q table, the Foe-Q minimax solver and the uCE-Q solver would give the same policy and state value – so I am a bit more confident with my implementation.

The final Q-table of state s for both Foe-Q and CE-Q is shown in Fig.3. Interestingly, in Foe-Q and CE-Q, two players Q-table preserve the property $Q_A = -Q_B$ during training (If not initialized this way, it empirically converge to something close). For Friend-Q, the converged Q-table does not satisfy $Q_A = -Q_B$ (also see Fig.2). As far as I understand, in zero-sum game, there isn't a collaborative equilibrium, so friend-Q does not necessarily converge to Nash equilibrium. Foe-Q converge to Nash equilibrium because there is adversarial equilibrium, and the collaborative equilibrium coincide with Nash equilibrium in case of zero-sum game.

IV. Pitfalls

There are several pitfalls in this project that requires future work.

player A \ player B	stick	W	E	N	S
stick	9.0	10.0	8.551	9.000	8.999
W	9.0	10.0	8.776	9.000	8.992
E	9.0	10.0	-0.935	9.000	9.000
N	9.0	10.0	8.553	9.000	9.000
S	9.0	10.0	8.085	8.993	9.000

(a) player A

player A \ player B	stick	W	E	N	S
stick	8.527	-10.0	9.000	8.525	8.087
W	8.517	-10.0	8.762	8.524	8.088
E	8.094	-10.0	9.495	8.096	8.100
N	8.526	-10.0	8.996	8.523	8.100
S	8.099	-10.0	9.000	8.092	8.487

(b) player B

Fig. 2. Q table of state s of trained Friend-Q learner

First, Greenward's paper uses on-policy/SARSA update for Q-Learner, while we used off-policy so that everything can be fit into the general off-policy update template 2.

It would also be interesting to implement other variant of CE-Q learners. For eCE-Q and rCE-Q, we may need to introduce extra variable $u = \sum_{\mathbf{a}} \sigma(\mathbf{a}) Q_i(\mathbf{a})$, solving $max(u)$ with extra constraint $u > (or\ <) \sum_{\mathbf{a}} \sigma(\mathbf{a}) Q_i(\mathbf{a}), \forall i$. lCE-Q would require an decentralized algorithm that each agent solves one LP.

REFERENCES

[1] A. Greenwald, K. Hall, and R. Serrano, "Correlated q-learning," in *ICML*, vol. 20, no. 1, 2003, p. 242.

[2] M. L. Littman, "Friend-or-foe q-learning in general-sum games," in *ICML*, vol. 1, 2001, pp. 322–328.

[3] A. Greenwald, K. Hall, and M. Zinkevich, "Correlated q-learning," 2005.

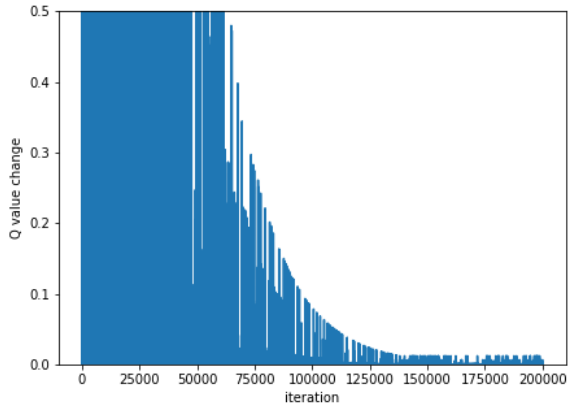
player A \ player B	stick	W	E	N	S
stick	-1.985	10.0	1.884	-1.938	-3.639
W	-1.930	10.0	-0.038	-1.931	-5.097
E	-3.449	10.0	-9.060	-3.448	-6.582
N	-1.960	10.0	1.873	-1.962	-3.614
S	-3.089	10.0	-2.941	-3.060	-1.878

(a) player A

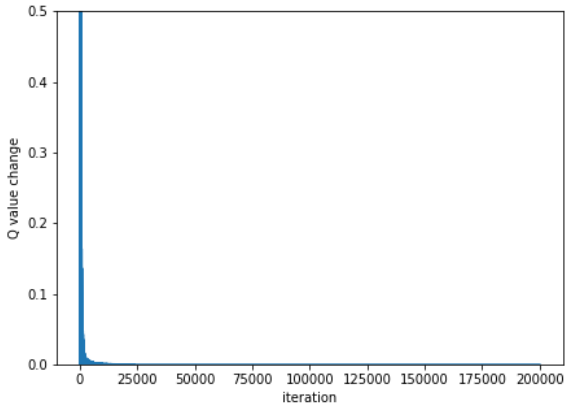
player A \ player B	stick	W	E	N	S
stick	1.985	-10.0	-1.884	1.938	3.639
W	1.930	-10.0	0.038	1.931	5.097
E	3.449	-10.0	9.060	3.448	6.582
N	1.960	-10.0	-1.873	1.962	3.614
S	3.089	-10.0	2.941	3.060	1.878

(b) player B

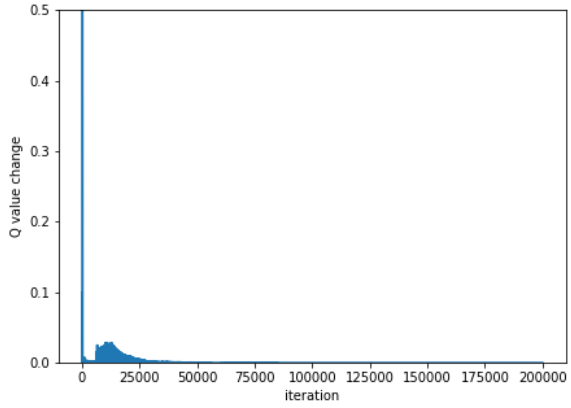
Fig. 3. Q table of state s of trained Foe-Q or CE-Q learner



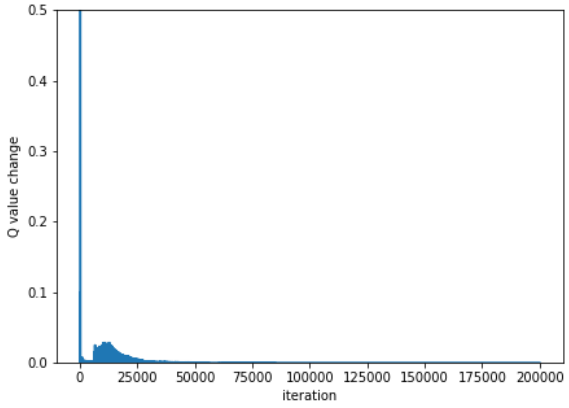
(a) Q-Learning



(b) Friend-Q



(a) Foe-Q



(b) uCE-Q

Fig. 4. The difference of player A's Q value for state s and player A taking action S and player B sticking/N in each iteration, $|Q_A^t(s, S, Stick/N) - Q_A^{t-1}(s, S, Stick/N)|$ (the TD error). For Q-Learning, $\alpha \to 0.001$, $\epsilon \to 0.001$ with decay rate 0.99995. For Friend-Q, Foe-Q, CE-Q, $\alpha(s, \mathbf{a}) = 1/n(s, \mathbf{a})$, $\epsilon = 1$. $\gamma = 0.9$