Correlated-Q

Tiantian Guo
tguo60@gatech.edu

***Abstract*—In this project, agents capable of correlated-Q, Foe-Q, Friend-Q and Q-learning [1] are developed. The theory of those four algorithms are discussed and an soccer game environment is utilized to implemented the agents.**

***Index Terms*—Correlated-Q, Foe-Q, Friend-Q, Q-learning, Soccer Game**

I. Introduction

In this project, we extend linear programming (LP) solution for the Nash equilibrum to the environment of soccer game. Four different algorithms: Q-learning, Friend-Q, Foe-Q and Correlated-Q are implemented to estimate the Q-values and value functions V. Figure 3 of paper "Correlated-Q Learning" [1] is replicated and analyzed.

II. Reinforcement Learning and Game Theory

A. Nash Equilibrium

Nash equilibrium is a set of strategy for a non-cooperative game involving multiple players in which non of the player are better-off by switching to any other strategy, given each player knowing the equilibrium strategy of the other players. For a finite game, there is always at least one Nash equilibrium. Strategy profile $x^* \in S$ is a Nash equilibrium if and only if:

$$\forall i, x_i \in S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{i-1}^*) \qquad (1)$$

where x_i is the strategy profile of player i and x_{-i} be the strategy profile for other players. $S = S_1 \times S_2, \times ...S_n$ is set of strategy profiles for n players and $f(x) = (f_1(x), f_2(x), ...f_n(x))$ is the utility function.

B. Markov Games and Correlated Equilibrium

Markov or stochastic games can be visualized as multi-player MDPs and can be represented in form of tupe $I, S, (A_i(s)), P, R_i$, with I as number of players, S as states, $A_i(s)$ as joint action vector for the players, P the probabilistic transition function and $R_i(s, \vec{(a)}$ as the i^{th} player's reward. Instead of individual player's action, multi-players' actions are considered to be a joint vector for all the players.

The paper presented by Greenwald and Hall [1] extends the idea of a multi-agent learning technique to correlated Q-learning. This method generalizes both Nash-Q, Friend and Foe-Q methods [2] [3]. Expanding on Nash-Q algorithm, which is based on Nash equilibrium, learns via a vector of independent probability distributions over actions where all the agents optimize based on each others probabilities. Linear programming is efficient in solving for the correlated equilibrium.

C. Multi-agent Q-Learning

The strategy selection function f from the algorithm 1 is the determining factor for each variant of multiagent Q-Learning. In regular Q-learning, the agents are unaware of the existence of other players, but can observe the state of the game, so every agent just learn its own Q function, thus an agent maximizes values of its own actions max$Q(s, a)$. In friend-Q [3], a particular agent A maximizes its own payoff at each state, and assumes the other players will coordinate their choices to let agent A receive the maximum payoff, thus an agent evaluate state value using max$Q(s, \vec{a}$. Foe-Q [3] incorporate mixed strategies for individual actions at Nash equilibrium using maximin algorithm. Correlated-Q(CE-Q) employs mixed strategies for joint actions at correlated equilibrium.

Algorithm 1 Multi-agent Q-Learning.

Input f - strategy selection function; γ - discount; α - learning rate; D - decay; T - total training steps
Output action-value function Q_i^*;

Initialize s and Q_1,...Q_n
for t ∈ 1,...,T **do**
  choose and execute actions a_1,...,a_n at state s
  observe rewards R_1,...,R_n and the next state s'
  for i ∈ 1,...,n **do**
    $V(s') \leftarrow f_i(Q_1(s'), ..., Q_n(s'))$
    $Q_i(s, a_1, ..., a_n \leftarrow (1-\alpha)Q_i(s, a_1, ..., a_n) + \alpha[(1-\gamma)R_i + \gamma V_i(s')]$
  end for
end for
$s \leftarrow s'$
$\alpha \leftarrow D(\alpha)$

The update algorithms of Q values are different. Foe-Q updating is based on

$$V_1(s) = \max_{\sigma_1 \in A_1(s) a_2 \in A_2(s)} \min Q_1(s, \sigma_1, \sigma_2) = -V_2(s) \qquad (2)$$

Friend-Q updating follows

$$V_i(s) = \max_{\vec{a} \in A(s)} Q_i(s, \vec{a}) \qquad (3)$$

CE-Q learning updating is based on

$$\sigma \in \arg\max_{\sigma \in \text{CE}} \sum_{i \in I} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a}) \qquad (4)$$

The intention is to apply these equations on to a special case of Markov Games: Two-Player, Zero-Sum in form of grid soccer and compare how these converge based the error metric in

$$ERR_i^t = |Q_i^t(s, \vec{a}) - Q_i^{t-1}(s, \vec{a})| \quad (5)$$

III. Soccer Game Environment Setup

The soccer game is a two-player, finite-space zero-sum game. On the grid of 2 rows and 8 columns, there are two players A, B and a ball, as shown in Fig.1. The soccer field is a grid. The circle represents the ball. The rules for both player A and B can be summarized as:

1) The top-left cell is indexed as [0,0] and the bottom-rightcell is indexed as [1,3]. Player A and B are indexed as 0 and 1 respectively. The game is always initiated or reset to state s, as shown in Fig 1, in which B holds the ball, occupying position $[0, 1]$ and A occupies position $[0, 2]$.
2) Each player has five possible actions, stick the ball or go to direction N, S, E, W. The players' actions are executed in random order. Five actions are represented as the two-dimensional movement on the grid: [-1,0] for N, [0,1] for E, [1,0] for S, [0,-1] for west and [0,0] for stick.
3) At each step of the game, A and B determines their moves simultaneously, but the execution of those two moves are always in random order.
4) When a player with the ball is trying to move into the position already occupied by the other player, then the ball changes possession, and neither player changes position.
5) When the ball is in a player's goal, then that player scores 100, its opponent scores -100, and game ends, no matter who has the ball at the time.
6) After an previous game ends, a new game will be re-initialized to state s

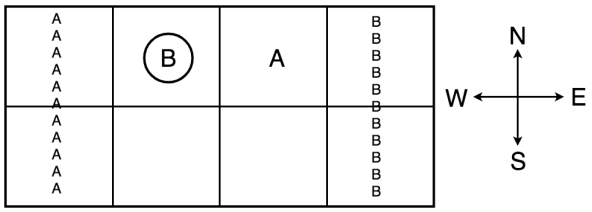


Fig. 1. Soccer Game. State s.

The algorithm 2 illustrates the implementation of the soccer game environment. The environment takes in current positions of both players, ball possession and new action selections from player A and B. It first checks whether teh input actions are in valid range, then randomly decides either A or B to take the first move. The first mover performs the action and the collision checking mechanism decides whether there is a change of ball. It then checks whether the player has scored for himself or the opponent. After the first mover finishes his step, if there is no goal, the second mover performs teh action and goes through teh same process. The step update finishes and returns to the caller for the updated states of two players, scores and game termination flag.

Algorithm 2 Soccer game environment.

Input Current positions, ball possession and new actions from players A and B;
Output A state, B state, ball possession, scores, done;

```
Init scores
Check input actions and randomly decide sequence of movement of A and B
First mover moves
if collision then
    check ball status and exchange possesion
else
    update positions of first mover
    if GOAL for any players then
        update scores, done=1, return
    end if
end if
Second mover moves
if collision then
    check ball status and exchange possesion, return
else
    update positions of second mover
    if GOAL for any players then
        updage scores, done=1, return
    end if
end ifreturn A state, B state, ball posession, scores, done
```

IV. Experiments and Results

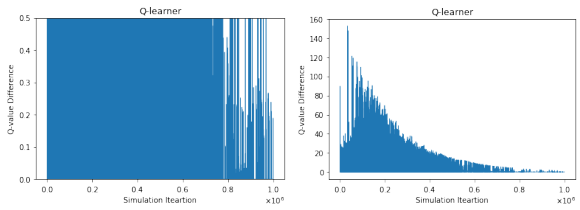
Experiments represented in Fig 3 in [1] were replicated to show ERR_i^t across 1 million training steps for Q-learning, friend-Q, foe-Q and CE-Q on soccer environment using different parameters, and their results and performances were compared. The Q values at a particular state action pair for player A, Q(state=s,action=S) is recorded during training iterations.

All four training algorithms are tested with ϵ-greedy with ϵ: $1 \rightarrow 0.001$ with the ϵ decay rate as 0.999995, discount γ = 0.9, leanring rate α from $1.0 \rightarrow 0.001$ with the α decay rate as 0.999995 or 0.99. The y axis in left two figures are limited to 0.5 and there is no y axis uplimit in right two figures.

A. Q-learning

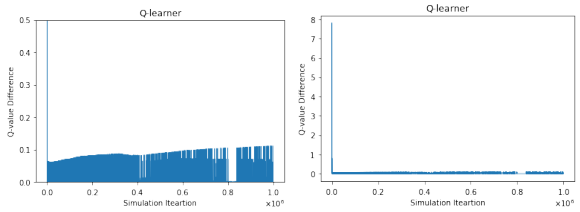
In Q-learning, the two agents are not aware of the actions taken by their opponents, however they can observe the state of the game. Player A can observe position of B, as well as possession of the ball, vice

versa. Since a player is only aware of its own action, it simply choose the action with the largest Q value at the particular state to evaluate the utility of that state.



(a) Zoom in (b) Zoom out

Fig. 2. α decay: 0.999995



(a) Zoom in (b) Zoom out

Fig. 3. α decay: 0.99

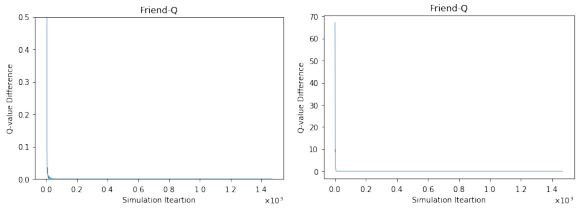
The effect of α decay rate is shown in Fig. 2 and 3. The left figure in Fig 2 almost replicates the figure (d) in Fig.3 in paper [1]. But the figures are not exactly the same due to the different selections of hyperparameters like γ, ϵ, α, α decay etc. As we can see the plot difference between $\alpha decay = 0.999995$ and $\alpha decay = 0.99$. α decay 0.999995 performs better than α decay 0.99. The plot from α decay 0.99 does not show any convergence trend at all. As long as we try various combination of hyperparameters, we can achieve same plot as in [1].

Q-learning algorithm results do show a decreasing trend but even after million iterations it does not fully converge.

B. Friend-Q

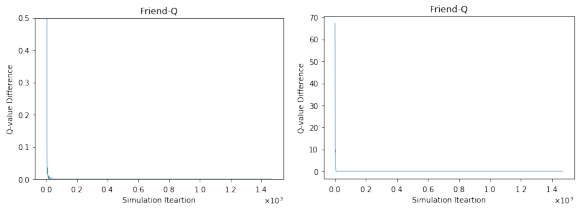
In friend-Q, a player, A, is aware of actions of all players, as well as the full state of game, however in evaluating the utility of a particular state, it chooses the joint action with the highest Q value, and assumes its opponent will cooperate and let it choose that joint action. The opponent behaves the same, assuming A will cooperate with it.

In Fig 4 and 5, the effect of α decay is minimal to the Q value difference. Only first 1500 iterations are shown in the graph. It demonstrates a very fast convergence of the Q-value. The graph closely resembles the shape in Greenwald's paper [1], though the original paper it takes more steps for convergence. Some hyperparamters combinations might achieve the exact shape and the same number of converge iterations.



(a) Zoom in (b) Zoom out

Fig. 4. α decay: 0.999995

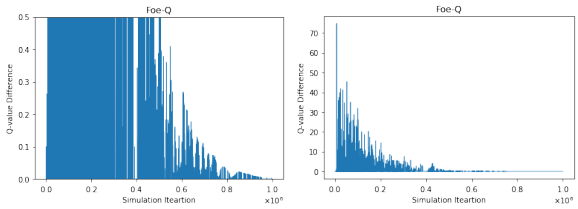


(a) Zoom in (b) Zoom out

Fig. 5. α decay: 0.99

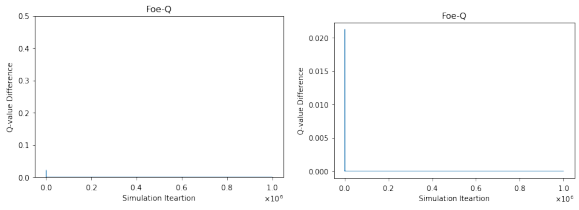
C. Foe-Q

In foe-Q, each agent tries to maximize its own reward and minimize its opponents, and the utility of a particular state $V(s)$ is evaluated to be opposite for the two players.



(a) Zoom in (b) Zoom out

Fig. 6. α decay: 0.999995



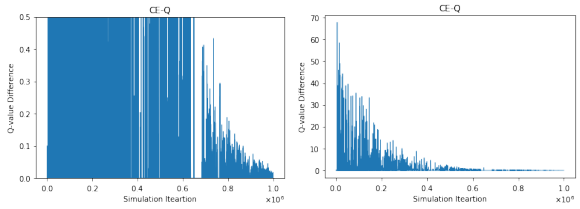
(a) Zoom in (b) Zoom out

Fig. 7. α decay: 0.99

The graph replicates Fig 3 in Greenwald's paper [1], in our replication, the graph begins to converge at around 800,000 steps while in [1], the graph converges at around 700,000 iterations. As the maxmin (Foe-Q) algorithm requires an linear programming solver to calculate value function and probability distribution over actions, different solver used might contribute to such difference.

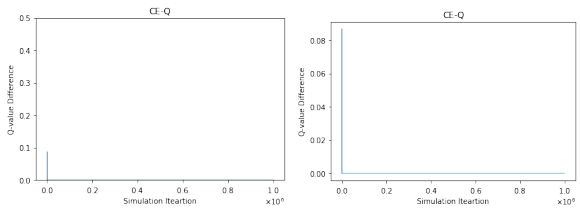
D. Correlated-Q

CE-Q evaluates the utility at a particulate state $V(s)$ using the mixed strategy of joint actions obtained at a correlated equilibrium. CE-Q learner learns two separate Q tables for players A and B, and using Q values from both tables to generate a strategy of joint actions using linear programming.



(a) Zoom in (b) Zoom out

Fig. 8. α decay: 0.999995



(a) Zoom in (b) Zoom out

Fig. 9. α decay: 0.99

Fig 8 and 9 shows ERR_i^t for Foe-Q algorithm across 1 million iterations. It shows a decreasing Q-value difference at state s, indicating the convergence of the algorithm. Moreover, the graph closely resemble the graph of Foe-Q in terms of convergence. There are few factors contributing to this observation. First, soccer game is a zero-sum finite-space game, hence different CE-Q algorithms will generate same values for all equilibria. Secondly, as explained in the original paper, CE-Q learns the same set of Q-values as Foe-Q. The difference of graphs generated in this report and in the paper [1] might result from the use of different linear programming solver and hyperparameter selections.

V. Conclusion

The details on game theory foundation from nash equilibrium and correlated equilibrium to the generalization of MDP as markov games are explained. This report implemented four multi-agent Q learning algorithms: Q-Learning, Friend-Q, Foe-Q and CE-Q in the soccer game environment. The error differences plots are compared with the results shown in [1]. Deeper understanding is gained on Markov Games and CE-Q.

References

[1] Greenwald, A., and Hall, K., Correlated-Q learning *International Conference on International Conference on Machine learning*, 2003.

[2] Junling Hu, Michael P. Wellman, Nash Q-Learning for General-Sum Stochastic Games *Journal of Machine Learning Research*, 4, 1039-1069, 2003.

[3] Michael L. Littman, Friend-or-Foe Q-learning in General-Sum Games *ICML*, 322-328, 2001.