Reproducing results from Soccer game – Greenwald, Hall 2003

SAHIL DHINGRA

CS-7642 Reinforcement Learning
Project 3
GitHub hash – ***177e991***

1. Introduction

This article describes implementation of Soccer game – a multi-agent zero-sum Markov game [1] for which there do not exist deterministic equilibria – and experimental results from using four algorithms – Q-learning, Friend-Q, Foe-Q and Correlated-Q.

2. Soccer game environment

The game is played on a 2x4 grid as depicted in Figure 2. The two players, A and B, occupy distinct squares of the grid and can choose one of 5 actions on each turn: N, S, E, W, and stick. Once both players have selected their actions, the two moves are executed in random order.

- *Goal* - The circle in the figures represents the "ball". When the player with the ball steps into a goal (left for A, right for B), the player corresponding to the goal scores +100 points and the other player -100, and the board is reset to a random configuration – each of the players outside of the goal states. Possession of the ball goes to one or the other player at random.
- *Collision* – 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. The problem can be summarized as a continuous space RL problem, where we need to find the optimal policy – represented in a 4-dimensional continuous action space – corresponding to a continuous 8-dimensional state space.

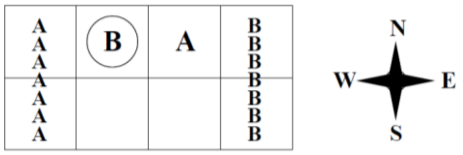

Figure 1. Soccer game state *s*, where error is measured at each successive iteration

Certain aspects of the game were not specified in the paper or unclear, and hence the description from *Littman 1944* was followed, or otherwise assumed to be the following in other cases –

1. *Initial state* – Since it isn't specified, a random initial state was chosen, where both players start from any of the non-goal states, with ball going to either player at random.
2. *Order of movement* – The order of execution of actions in a normal and collision scenario is unclear and *Littman's* description was followed here.
3. *Off-policy training* – The agent was trained off-policy for correlated-Q, foe-Q and friend-Q algorithms and on-policy for Q-learning, as the paper doesn't specify that for the soccer game, but does so for the Grid games.
4. *Hyper-parameters* – Values for γ & α_{min} were used as $\gamma = 0.9$ and $\alpha_{min}=0.001$, same as that provided in Grid Games. Since no α value is specified, a suitable value was used for each algorithm. Three additional parameters, that come into play for on-policy Q-learning – ε, ε_{min} *and* $\varepsilon_{decay\ rate}$ – are not specified either for the soccer game; $\varepsilon_{min} = 0.001$ is used from the Grid games section.
5. *Decay function* – A decay schedule is used for α & ε in training, however, no particular function is specified in the paper. This affects the shape of the top of the curve. For reproducing the experiments, the following decay schedules were followed.

$$\alpha = \alpha_{min} + (\alpha_{min} - \alpha_{min})\, e^{-decay\_rate*episode}$$
$$\varepsilon = \varepsilon_{min} + (\varepsilon - \varepsilon_{min})\, e^{-decay\_rate*episode}$$

3. Implementation

Four different policies were learned with an offline training approach, except vanilla Q-learning with . Q-learning algorithm was followed for all training methods, however, the Value function update is completed using different functions.

$$V_i(s') = f_i(Q_A(s'), Q_B(s'))$$

where, *f* represents the multi-agent algorithm in each game-play approach for the *i*th player

Q-values are updated as per the following equation for both players A & B:

$$Q_i(s, \boldsymbol{a}) = (1 - \alpha)\, Q_i(s, \boldsymbol{a}) + \alpha\, [(1- \gamma)\, R_i + \gamma\, V_i(s')]$$

where, ***a*** represents the action vector.

State representation – For each player, a state represents the state of the game and is represented as a combination of the position of each player and possession of the ball. Hence it is represented as an 8x8x2 vector – [Position of Player A, Position of Player B, Ball Possession (*A or B*)]

3.1. Q-Learning

In Q-Learning approach, each player aims only to optimize their rewards and does not have access to opponent's Q-table. Hence, at every state, *Q(s, **a**)* is a 1-D vector with one value for each of the 5 actions. From the perspective of player A, its Value is updated with the maximum of the Q-values for the 5 actions for the player.

$$V_A(s) = max(Q_A(s))$$

3.2. Friend-Q

Friend-Q is just ordinary Q-learning in the combined action space of the two players. Each player aims to maximize the rewards against the action space of both players and maintains a Q-table which keeps track of opponent's actions as well, corresponding to its own actions for every state. *Q(s)* is a 5x5 matrix, corresponding to the 5x5 action space [Action A, Action B]. From the perspective of player A, its Value is updated with the maximum of the Q-values for the 5x5 action space.

$$V_A(s) = max(Q_A(s, Action\ A, Action\ B))$$

3.3. Foe-Q

In Foe-Q, each player aims to find a policy that maximizes rewards for itself while minimizing rewards for the opponent, which can be formulated as a minimax scenario – maximizing your rewards in the worst case. The optimal policy is evaluated by solving a linear program, to obtain a probabilistic policy. The LP can be formulated as

$$\boldsymbol{\pi_A(s)}*Q_A(s, action_A) >= U$$

where, each of the policy components is greater than 0 and sum to 1. Thus, this returns the optimal policy and value function corresponding to the equilibrium.

3.4. Correlated-Q (CE-Q)

In CE-Q, each player has access to opponents Q-value table and evaluates the equilibrium policy using LP. The utilitarian version of the algorithm was used for this article. A single LP is formulated using Q-values from both players, where the policy is maximised against all other actions for the player. A 25 variable LP is solved to obtain the optimal policy.
CE-Q converges to the same policy as the minimax approach as well, the one followed while training the Foe-Q agent.

4. Results

1. **Convergence plots** – Similar curves were achieved by testing different hyper-parameter values and different random seeds for initialization. For several reasons, the plots don't exactly match. Some of these can be attributed to the assumptions taken while training the agent, as well as other uncontrollable factors such as random values used.

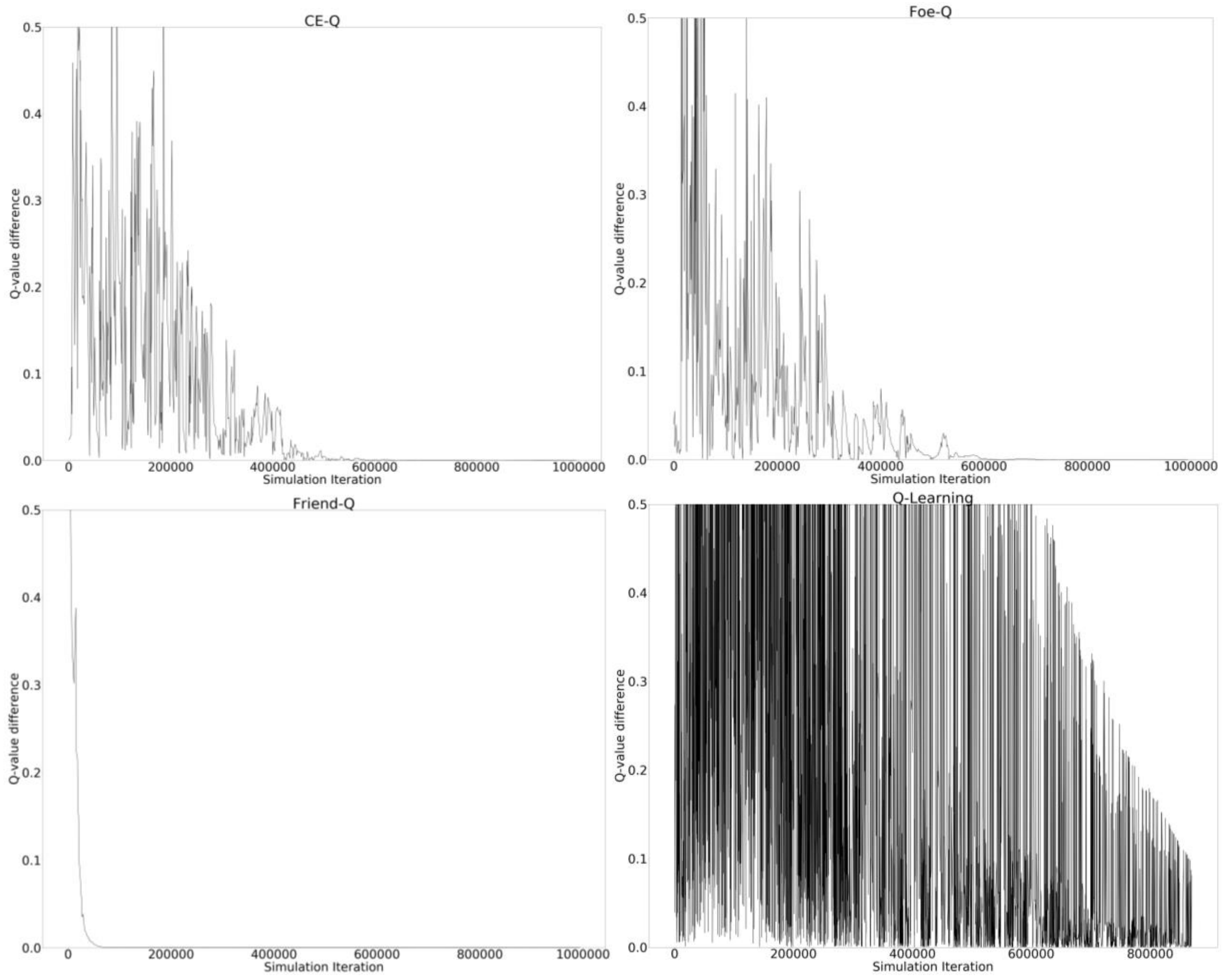

Figure 1. Convergence in the soccer game. All algorithms—except Q-learning—converge. As above, the CE-Q algorithm shown is uCE-Q

5. Analysis of results

5.1. Hyper-parameter values – $\alpha = 0.2$ was used for all but Foe-Q (0.4) algorithm, while the decay-rate used for α was $5\text{x}10^{-10}$ for all but Friend-Q ($5\text{x}10^{-8}$).

5.2. Difference in results due to random seed – Due to randomness in the training process – using random initial states for game initialization or taking random actions while learning off-policy, the shape of the plot can vary, despite the overall trend not changing significantly.

5.3. Convergence – Only Q-learning algorithm doesn't converge, since it seeks deterministic policies which do not exist for this game. Friend-Q learning converges to wrong policies. CE-Q and Foe-Q converge to optimal policies.

References

[I] Michael L. Littman (1994); Markov games as a framework for multi-agent reinforcement learning

[II] Greenwald, Hall (2003); Correlated-Q Learning