CS8803-O03 Reinforcement learning Project 3 report

Rohan D. Kekatpure
Email: rdk@gatech.edu

I. Introduction

Multi-agent Q-learning is a nontrivial departure from deterministic action, single Q function MDPs in two ways: (1) state values are functions of multiple Q's (2) optimal policies are allowed to be stochastic. The Greenwald paper [1] demonstrates the inadequacy of regular Q learning and convergence of multi-agent Q learning learning. Project 3 aims to deepen our understanding of multi-agent (adversarial or cooperative) Q learning through reproduction of Greenwald's results for a 2×4 grid game called soccer.

II. Quick theory tour

The paper presents four multi-agent Q learning options:

1) **Q learning:** Ignore the opponent's Q function and actions. Agents are only coupled through rewards. The policy is deterministic and Q function at each state for each agent is simply a vector of n values ($n = 5$ in our case). The state value is the max for Q_i:
$$V_i = \max_{a \in A_i} Q_i(s, a) \tag{1}$$
2) **Friend Q:** Ignore the opponent's Q function, but consider its actions. Optimal policy is deterministic, and Q function at each stage is a $n \times n$ matrix. The value for each state is the max across this matrix:
$$V_i = \max_{\vec{a} \in A_1 \times A_2} Q_i(s, \vec{a}) \tag{2}$$
3) **Foe Q:** Ignore opponent Q function, consider its actions, but calculate the value function using **maximin** instead of simple max. This requires linear programming (LP). Maximin also allows **stochastic** optimal policies represented as a probability distribution over n action values. The Q is an $n \times n$ matrix and the state value is:
$$V_i = \max_{\pi \in \text{PD}(A)} \min_{o \in O} \sum_{a \in A_i} \pi(a) Q_i(s, a, o) \tag{3}$$
4) **CE Q:** Consider joint actions and compute state value as a function of agents' Q values
$$V_i = \sum_{\vec{a} \in A_1 \times A_2} \sigma(\vec{a}) Q_i(s, \vec{a}) \tag{4}$$
where σ is a probability distribution over the n^2 action pairs computed using the *u*CE-Q objective out of the four CE options provided. In simplified notation:
$$\sigma = \arg\max_{\sigma \in \text{CE}} \sum_{\vec{a} \in A_1 \times A_2} \big[Q_1(s, \vec{a}) + Q_2(s, \vec{a})\big] \tag{5}$$

III. Implementation

A. Game representation

The soccer world is a 2×4 grid with row index $r \in [1, 2]$ and column index $c \in [1, 2, 3, 4]$. The state of the game is represented as:
$$s \triangleq \text{tuple}((r_A, c_A), (r_B, c_B), \text{ball}) \tag{6}$$
The individual action space is $A_1, A_2 \in [N, W, E, S, T]$ ($T =$ stick) and the joint action space is $A_1 \times A_2$. Since the number of states $= 8 \times 7 \times 2 = 112$ is small, the Q values can be maintained in memory as a nested dictionary:
$$Q \triangleq \{s : \{\vec{a} : (Q_A, Q_B)\}\} \tag{7}$$
where the inner dictionary maps 25 action pairs to a tuple of Q values for agents A and B, and the outer dictionary maps this structure to the state. The structure can also be thought as a mapping from each of the 112 states to a $5 \times 5 \times 2$ matrix of Q values for A and B for each action pair.

B. Simulation

A **move** is defined as one action each by A and B carried out in a random order. An **episode** is a set of moves until a goal is scored. An episode can: (1) in theory continue forever and (2) end mid-move if a goal is scored.

To simulate a move, we select an action pair (a_1, a_2) from $A_1 \times A_2$ and assign it to (A, B), who execute in random order. After executing its action, each agent checks for the game-end condition and receives rewards for reaching the next state. Rewards are zero except when the game ends. In addition to the prescribed rules, we observed the following additional conventions: (1) an episode never starts in the end condition, (2) goal is counted even when a stationary empty-handed agent in a goal position is handed a ball by the mobile ball-carrying agent; i.e. A could lose if it is in B's goal position and handed a ball.

After each move, values V_i, $i \in [A, B]$ for each state are calculated and used to update the two Q-functions according to the update equations in Table 1 in the Greenwald paper. Because of the small number of states, we learn via an **off-policy** Q learning algorithm (i.e. actions are always random).

C. Recording Q-function difference

The paper reports the evolution of the Q_A values for the state $s = ((1, 3)_A, (1, 2)_B, B)$ (a.k.a the monitor state) for the action pair ST. The evolution of the Q values for the monitor

state are stored in an array and updated only when that state is actually encountered in an episode. A global timestep counter, which increments across episodes, is maintained to record the value of the timesteps at which the monitor state is updated. The Q-function difference is calculated exactly according to the prescription: $\text{ERR}_A^t = |Q_A^t(s, \vec{a}) - Q_A^{t-1}(s, \vec{a})|$.

D. Linear programming

Correlated Q and Foe Q implementation require LP. After some research [2], we chose the `CVXOPT` LP package with the GNU `glpk` solver. This resulted in over $10\times$ speedup relative to `PuLP`, `SciPy` and `CVXOPT`'s default solver. However, setting up the problem in `CVXOPT`'s matrix language requires care (proper negations, column ordering etc).

E. Software dependencies

The only external packages for our implementation are `CVXOPT`, `numpy` for core computation and `matplotlib` and `Pandas` for postprocessing.

F. Hyperparameters

We experienced that the convergence curves are extremely sensitive to the choice of initial learning rate and the decay schedule. We've collected the hyperparemeters of our implementation in Table I.

Hyperparameter	Value
Discount γ	0.9
Decay schedule	$\alpha = \max(\alpha_{\min}, \alpha_0 \alpha_d^t)$
α_0	0.15
$\alpha_{\min}$	10^{-3}
α_d	0.9999954

TABLE I: Hyperparameters for all Q-learning algorithms.

IV. Results

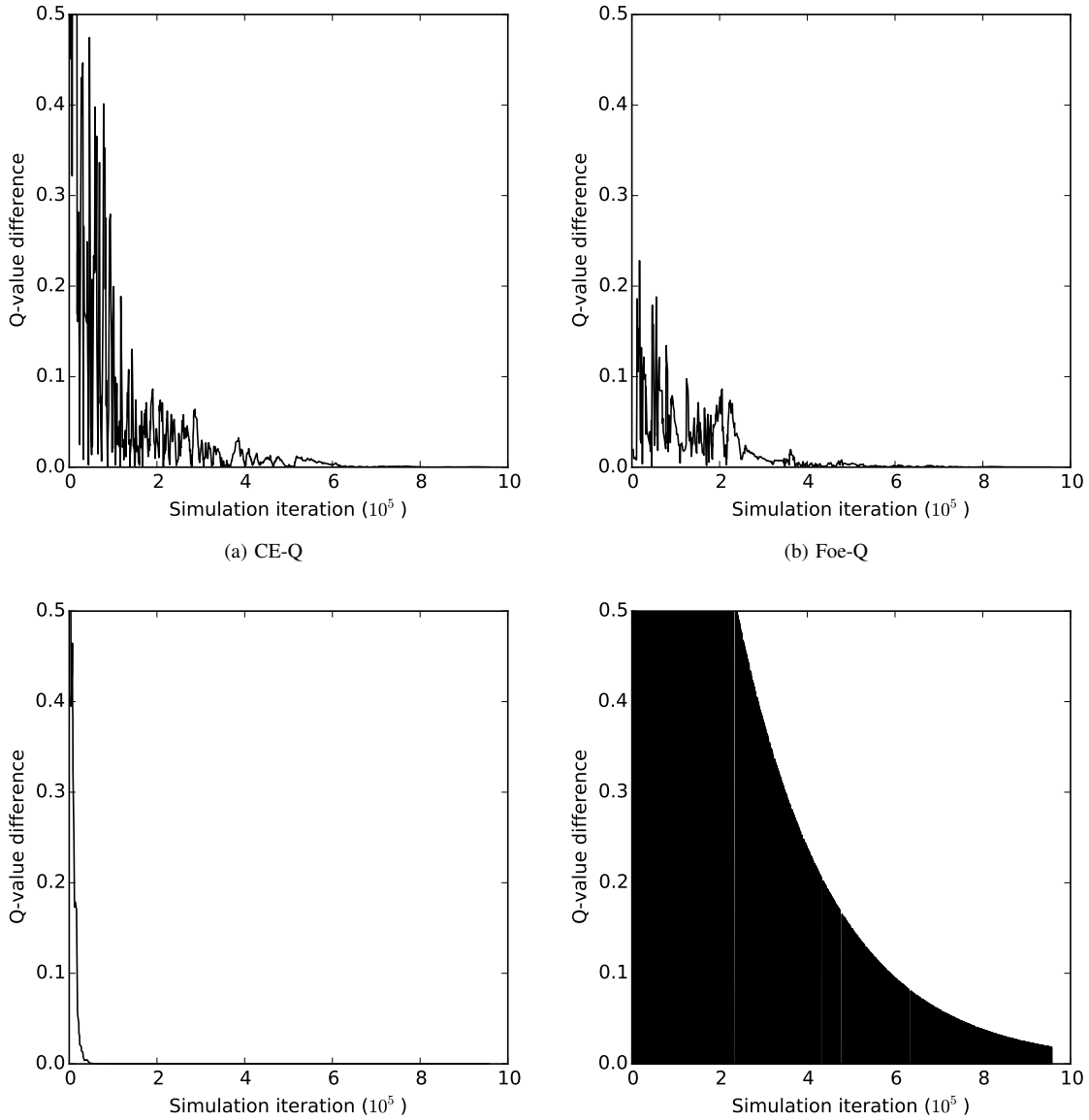
The results of the simulation of the soccer game are displayed in Fig. 1.

Summary

References

[1] A. Greenwald and K. Hall, "Correlated Q learning," *ICML*, 2001.

[2] S. Caron,"Linear Programming in Python with CVXOPT," *https://scaron.info/blog/linear-programming-in-python-with-cvxopt.html*



(a) CE-Q

(b) Foe-Q

(c) Friend-Q

(d) Q-learning

Fig. 1: Results of the soccer game in the Greenwald paper. (a) Correlated Q, (b) Foe Q, (c) Friend Q and (d) Q learning.