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. The Greenwald paper [1] (1) demonstrates that policies in multi-agent situations could be stochastic, (2) formulates a computable algorithm, called "correlated Q learning" to find the stochastic optimal policies and (3) shows a way to circumvent the NP-hard Nash equilibrium problem. In Project 3, we aim to reproduce 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:** Ignores the opponent's Q function and actions. Agents are coupled only through rewards. The policy is deterministic and Q function at each state for each agent is a vector of n values ($n = 5$ in our case). The state value for agent i is max over the Q vector:
$$V_i = \max_{a \in A_i} Q_i(s, a) \tag{1}$$
2) **Friend Q:** Ignores the opponent's Q function, but works with joint action space. Optimal policy is deterministic, and Q function at each stage for each agent is a $n \times n$ matrix. The state value for agent i is the max over the Q matrix:
$$V_i = \max_{\vec{a} \in A_1 \times A_2} Q_i(s, \vec{a}) \tag{2}$$
3) **Foe Q:** Ignores opponent's Q function, but calculates the value using **maximin** instead of simple max and requires linear programming (LP). Maximin also allows **stochastic** optimal policies represented as a probability distribution over n action values. The Q function at each state for each agent is an $n \times n$ matrix and the state value is given by:
$$V_i = \max_{\pi \in \text{PD}(A)} \min_{o \in O} \sum_{a \in A_i} \pi(a) Q_i(s, a, o) \tag{3}$$
4) **CE Q:** Considers opponent's Q as well as joint actions and computes state value as a function as:
$$V_i = \sum_{\vec{a} \in A_1 \times A_2} \sigma(\vec{a}) Q_i(s, \vec{a}) \tag{4}$$
where $\sigma(\vec{a})$ is a probability distribution over the n^2 action pairs $\vec{a}$ 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

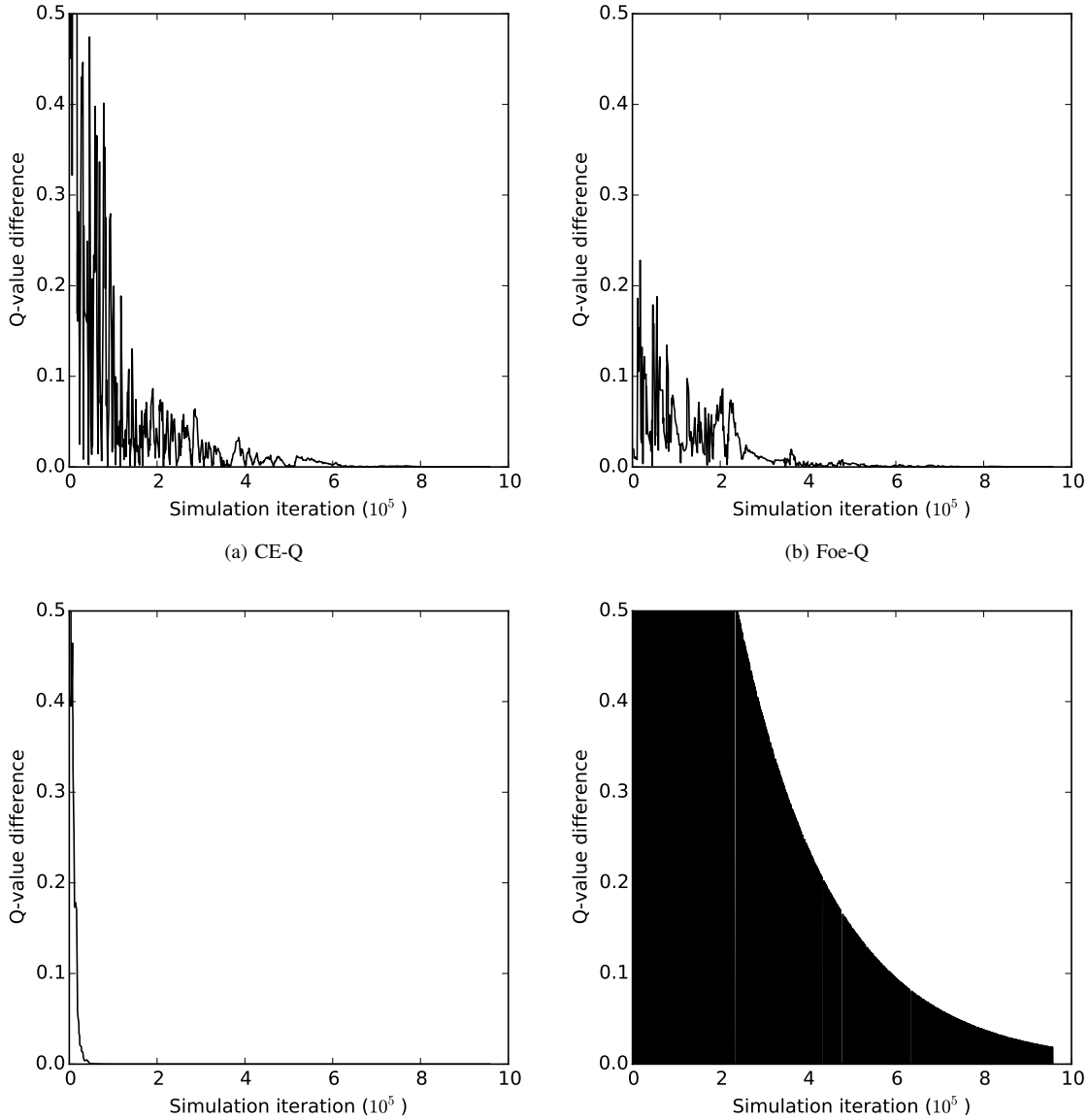
The soccer world is a 2×4 grid with row index $r \in [1, 2]$ and column index $c \in [1, 2, 3, 4]$. The state of the game is represented as:
$$s \triangleq \text{tuple}((r_A, c_A), (r_B, c_B), \text{ball}) \tag{6}$$
The individual action space is $A_1, A_2 \in [N, W, E, S, T]$ ($T =$ stick) and the joint 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:
$$Q \triangleq \{s : \{\vec{a} : (v_A, v_B)\}\} \tag{7}$$
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:
$$Q_i^{t+1}(s, a) \leftarrow (1 - \alpha_t) Q_i^t(s, a) + \alpha_t \big[(1 - \gamma) R_i^t + \gamma V_i^t(s)\big] \tag{8}$$
where the action variable a is a single action in case of Q-learning and Foe Q and an action vector, $\vec{a}$, for Friend Q and CE Q. Because of the small number of states, we learn via an **off-policy** Q learning algorithm (i.e. actions are always random).



(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.

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 $\vec{a} = ST$. 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 **sensitive** to the choice of initial learning rate and the decay schedule.

The converged values are less sensitive. We've collected the hyperparemeters of our implementation in Table I.

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

TABLE I: Hyperparameters for all Q-learning algorithms.

IV. Results

Before discussing the actual results, note that the **spirit** of the assignment is (1) to demonstrate that regular and Friend Q learning do not work in stochastic multi-agent situations and (2) to discover stochastic optimal policies for multi-agent environments. While we aim to reproduce Greenwald results, we don't try to eliminate small cosmetic differences, as long as optimal policies match.

The results of the simulation of the soccer game are displayed in Fig. 1 and are in the same order as [1]. Below is a summary of results:

1) Regular Q-learning, shown in Fig. 1(d), does not converge, but instead simply traces the decay schedule. The trend of our Q-learning curve and the general conclusion is consistent with [1].
2) Friend Q shown in Fig. 1(c) converges, but it converges to a deterministic action by design (i.e. in Friend Q formulation we simply don't seek a probability distribution). Our results converge at ~ 50000 timesteps, which is close to what can be estimated from the paper.
3) Foe Q, shown in Fig. 1(b) allows for stochastic actions by computing probability distribution over *independent* action vectors. The Foe Q probability distribution for A converges to $\pi_A(s) = [0.00, 0.00, 0.00, 0.32, 0.68]$ and $\pi_B(s) = [0.00, 0.00, 0.00, 0.68, 0.32]$ for actions $[N, W, E, S, T]$. So The agent is randomizing between sticking and heading south, consistent with description in the paper. The figure appears a bit different than the paper; this is to be expected given the differences in the learning rates, decay schedules and randomizer seeds.
4) Correlated Q results are plotted in Fig. 1(a). The overall graph appears similar to the paper. We also obtain a probabilistic action space for A and B which agrees with the optimal policy for Foe Q (highest probability mass assigned to action pairs SS and TT). Our Foe and CE Q graphs are dissimilar. In the paper, the graphs are **identical**. We are unable to reproduce this aspect.

Summary

Project 3 was a challenging exercise that forced a synthesis of the multi-agent systems.

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*