CS7642 Project 2 – Solving LunarLander problem with Double Deep Q-Learning

Qinghui Ge

***Abstract*—git hash:**

I. SOCCER GAME

II. LEARNER IMPLEMENTATION

We follow a general off-policy learning template (Alg. 1) 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 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 $\mathbf{A} = \{A_1, A_2 \dots A_N\}$.

In Greenward's 2003 paper, 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, 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.

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:

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

Algorithm 1 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
```

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:

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

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:

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

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:

$$\text{maximize } V_i \tag{5}$$

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: $\sum_{a_i \in A_i} \sigma_i(a_i) = 1$ (6)

(7)

The second inequality constraint and equality constraint are simply probability restrictions. Foe-Q follows Alg.2 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 2 foe-Q update

```
function FOE-Q.UPDATE(s, s', a, R)
    Q_1(s, a_2, a_1) = (1 − α)Q_1(s, a_2, a_1) + α((1 − γ)R_1 + γV_1(s'))
    Q_2(s, a_1, a_2) = (1 − α)Q_2(s, a_1, a_2) + α((1 − γ)R_2 + γV_2(s'))
    V_1(s), σ_1(s) = minimax(Q_1(s))
    V_2(s), σ_2(s) = minimax(Q_2(s))
end function
```

D. Correlated-Q

In CE-Q, each agent maintain a copy of Q-table of size $nS \times nA_1 \times \ldots nA_N$. All agents follows a mixed policy based on join probability $\sigma(\mathbf{a}) = \sigma(a_1, a_2 \ldots a_N)$. 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:

$$\sum_{\mathbf{a_o}} (Q_i(\mathbf{a_o}, a_i') - Q_i(\mathbf{a_o}, a_i))\sigma(\mathbf{a_o}, a_i) \leq 0$$

$$\forall a_i \neq a_i' \in A_i$$

$$\mathbf{a_o} \in \{A_1, A_2 \ldots A_{i-1}, A_{i+1} \ldots A_N\}$$

2. $\Pi_i nA_i$ inequality constraint : $\sigma(\mathbf{a}) \leq 0, \forall \mathbf{a} \in \mathbf{A}$
3. 1 equality constraint: $\sum_{\mathbf{a} \in \mathbf{A}} \sigma(\mathbf{a}) = 1$ (8)

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.3.

Algorithm 3 Correlated-Q update

```
function CE-Q.UPDATE(s, s', a, R)
    for i = 1,2... n_agent do
        Q_i(s, a) = (1−α)Q_i(s, a)+α((1−γ)R_i+γV_i(s'))
    end for
    σ(s, a) = uCE({Q_1(s), Q_2(s) ... Q_N(s)})
    for i = 1,2... n_agent do
        V_i(s) = Σ_a Q_i(s, a)σ(s, a)
    end for
end function
```

III. Comparison between Foe-Q and Correlated-Q