Solving Stochastic Games with Multiagent Reinforcement Learning

Josep Famadas
jfamadas95@gmail.com

June 18th, 2018

Contents

1 Introduction

In this document it is presented the implementation of an algorithm known as Opponent Modeling Q-Learning, and evaluate its performance applied in a game.

First, it is given an introduction on what is a stochastic game, then the game and the algorithm are explained and finally the results are presented.

2 Stochastic Game

Stochastic games are the natural extensions of Markov Decision Processes (multi-state games where the Markov assumption is made) to multiple agents.

A stochastic game can be defined with a tuple $(n, S, A_{1...n}, T, R_{1...n})$ where n is the number of agents, S the set of states, A_i the set of actions avaliable for the $i-th$ agent (A is the joint action space $A_1\ x\ ...\ x\ A_n$), T is the transition function $S\ x\ A\ x\ S \rightarrow [0,1]$ and R_i is the reward received by the $i-th$ agent $S\ x\ A \rightarrow \mathbb{R}$.

3 Game

The Stochastic Game decided for this document is inspired in the one described in [1]. It consists on the $3x3$ grid described in Figure 1 which is the initial state. The transition probabilities are 1 for every wall except for the special walls marked by '- - -' where the probability is 0.5, e.g. If PL1 selects as action ($\uparrow$), there is a 50% probability that it moves to the above square and a 50% probability it remains in its current square.

Another special rule is that both players can not be in the same square (except for the reward square), so if in the initial state PL1 does ($\rightarrow$) and PL2 does ($\leftarrow$) one will move to the objective square and the other one will stay in its current square (the one that moves is decided with a coin flip).

The formal definition of the game is $(n, S, A_1, A_2, T, R_1, R_2)$:

- $n = 2$.
- S = Set of 73 states (72 with the combinations of both players in the board and one extra for the state in which both have reached the reward square).
- A_1 and A_2 = ($\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$, *Stay*).
- T = The transitions probabilities described in this section.
- R_1 and R_2 = 1 if they reach the reward square, -0.04 otherwise (if both reach the reward it is 1 for each).

The trials of the game end when one of the two players (of both) reach the reward square.

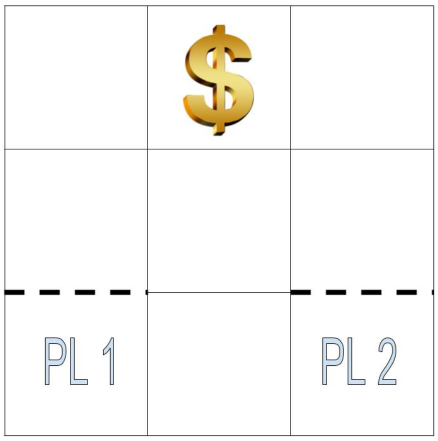

Figure 1: Game board. PL1: Player 1, PL2: Player2, $: Reward, - - - : Special wall

4 Algorithm: Opponent Modeling Q-Learning

Opponent modeling is a Reinforcement Learning algorithm that allows each agent to learn explicit models of its opponents as stationary distributions over their actions. These distributions combined with the learnt Q are used to select the action to perform on each state.

In the algorithm several parameters are used. Seen from the point of view of a player, a_i is the action performed by him, a_{-i} is the joint action performed by its opponent(s) and a is the joint action of all the players. For example, in a game with 4 players that in a given state do (P1:$\uparrow$, P2:$\downarrow$, P3:$\leftarrow$, P4:$Stay$), from the point of view of the P3, a_i would be ($\leftarrow$), a_{-i} would be ($\uparrow$, $\downarrow$, $Stay$) and a would be ($\uparrow$, $\downarrow$, $\leftarrow$, $Stay$). The other parameters are:

- $C(s, a_{-i})$: Number of times that in the state s the opponent(s) have done the joint action a_{-i}. (Every player has its own C).
- $n(s)$: Number of times that the state s has been visited. (This parameter is the same for all the players).
- $Q(s, a)$: Matrix of shape ($len(States)$ x $len(A_1)$ x ... x $len(A_n)$) with the long-term expected return of (in state s) doing a_i and the other players doing a_{-i}. (Each player has its own Q).

The algorithm described in algorithm 1 is from the point of view of 1 agent. The others do the same in parallel:

```
initialize Q: e.g. $Q(s,a) \leftarrow 0.01$;
$C(s,a_{-i}) \leftarrow 0$;
$n(s) \leftarrow 0$;
$s \leftarrow initialState$;
**while** *True* **do**
    (a) Select action $a_i$ following equation (1) with probability $1-\epsilon$ or following
      equation (2) with probability $\epsilon$;
        $$a_i = \operatorname*{argmax}_{a_i} \sum_{a_{-i}} \frac{C(s,a_{-i})}{n(s)} * Q(s, < a_i, a_{-i} >) \quad (1)$$
        $$a_i = randomElement(A_1) \quad (2)$$
    where:
        $$\epsilon = \max(\min(\frac{1,000}{n^o\ trial}, 1.0), 0.1)$$
    (b) Observing other agents' actions $a_{-i}$, the resulting state $s'$ and your reward $r$;
        $$Q(s,a) \leftarrow (1-\alpha) * Q(s,a) + \alpha * (r + \gamma * V(s'))$$
        $$C(s,a_{-i}) \leftarrow C(s,a_{-i}) + 1$$
        $$n(s) \leftarrow n(s) + 1$$
        $$s \leftarrow s'$$
      where:
        $$V(s') = \max_{a_i} \sum_{a_{-i}} \frac{C(s',a_{-i})}{n(s)} * Q(s', < a_i, a_{-i} >)$$
    (c) **if** *Trial finished (any player reached the reward)* **then**
            $s \leftarrow initialState$
    **end**
**end**
```

Algorithm 1: Opponent Modeling Q-Learning.

5 Experimental Results

In this section are presented the experimental results of applying the Opponent Modeling Q-Learning algorithm described in section 4 in the game described in section 3.

For the learning parameters, since the environment is not fully deterministic due to the special walls, a big learning rate (α) is not optimal, so the selected value is 0.1.

When it comes to the discount (γ), since the environment is really small and by doing random actions is nearly impossible to have an infinite length trial, the value can be big, for the experiments we have selected 0.9.

Finally, the exploration parameter (ϵ) is described in algorithm 1. It starts at 1.0 (which means all actions are random) until the 1,000th trial, when starts decreasing until reaching 0.1 in the 10,000th trial, when it stops decreasing.

In the following experiments, first it is studied the performance of the algorithm in terms of number of trials needed to reach a stable behaviour. Then, it is studied how the agents react to different rewards.

5.1 Basic Rewards

- Step reward: -0.04. Reward given for each turn that the agent does not arrives to the reward square.
- Reward square: 1.0. Reward given to the agent that reaches alone the reward square.
- Both reward: 1.0. Reward given to **both** agents (it is not splitted) if both reach the reward square at the same turn.

As it can be seen in Figure 2, the algorithm converge to 3 episodes per trial (which is the minimum to reach the reward square).

If we look at the Q values for each action in the initial state, we can see that for player 1, whatever action selects player 2, its preferred action is ($\rightarrow$). The same happens with player 2 with ($\leftarrow$). If from this we follow the Qs of the possible following states we can see that both players try to take the bottom-middle position and go upwards to the reward position.

This is an equilibrium, since none of the two players have an incentive to change its behaviour, both will get the reward 50% of the trials, and if, for example, player 1 tries to go upwards from the beginning instead of right, it will still reach the reward only 50% of the trials because of the special wall.

However, this is not the optimal equilibrium, since if player 1 changed its behaviour as explained before, it would reach 50% of times (which means it would lose nothing) but player 2, going to the center and then upwards, would reach the reward square 100% of the times. In order to try to modify this we apply some changes on the rewards.

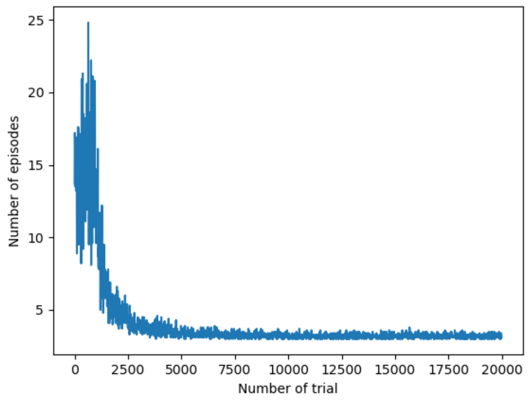

Figure 2: Average number of episodes for each 10 trials.

5.2 Increased Both Reward

- Step reward: -0.04. Reward given for each turn that the agent does not arrives to the reward square.
- Reward square: 1.0. Reward given to the agent that reaches alone the reward square.
- Both reward: 5.0. Reward given to **both** agents (it is not splitted) if both reach the reward square at the same turn.

Here comes the point in which it can be proven that the Opponent modelling Q-Learning is a powerfull algorithm for solving stochastic games, because despite the fact that PL1 does not know the action PL2 is going to perform, it manages to wait for PL2 so both reach the reward together.

In this situation, if we look at the Q values for each action in the initial state, when can see that player 1 will try to go right and player 2 will try to go up. This is the desired behaviour explained before, in order to get the 100% P1 / 50% P2. However, if player 2 can not go through the special wall the player 1 will wait for the player 2 until it goes through the wall, and both can reach the reward square together. My intuition is that the problem that leads to this is that the step reward is too low (in absolute value) and player 1 decides to lose a bit of reward because the final of reaching both at the same time will be much higher.

In order to solve this, the Step reward is increased (in absolute value).

5.3 Increased Both Reward

- Step reward: [-0.04 – -0.9]. Reward given for each turn that the agent does not arrives to the reward square.
- Reward square: 1.0. Reward given to the agent that reaches alone the reward square.
- Both reward: 5.0. Reward given to **both** agents (it is not splitted) if both reach the reward square at the same turn.

After sweeping the step reward from -0.04 to -0.9 it reaches a point in which the behaviour changes from 'waiting the other so both can get the reward at the same time' to 'both trying to get the central column and go upwards from it'. That what makes me think that, the Opponent Modeling Q-Learning makes the agent come to an equilibrium point but there is no way to force this equilibrium to be the optimal one (the one in which the expected added reward from both is maximum).

References

[1] Udacity. Stochastic games and multiagent rl - georgia tech - machine learning. `https://www.youtube.com/watch?v=ZdwT6YHZOt0&t=`.