UNIVERSITY OF MICHIGAN-DEARBORN

CIS 579: ARTIFICIAL INTELLIGENCE

INSTRUCTOR: DR. LUIS E. ORTIZ

FALL 2017

On-ramp Merging with Multi-Agent Reinforcement Learning

Author:
Heng Liu

Student ID:
62774047

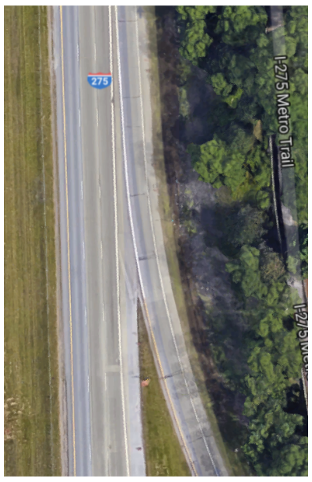
December 21, 2017

Contents

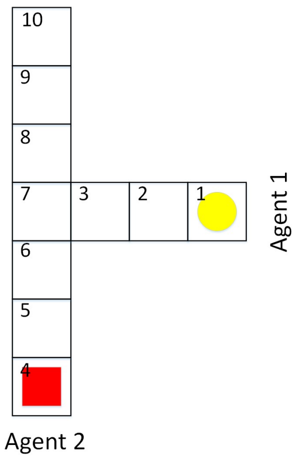
1 Introduction

Lane-merging can be a challenging problem in the era of autonomous driving when all vehicles are smart and connected. In this project we hope to develop a simulator based on the multi-agent reinforcement learning (MARL) technology to study the possible actions and consequences from two autonomous vehicles in lane merging scenario. As shown in Figure 1a, vehicles travel on the on-ramp with the goal of merging with approaching in-lane vehicle.

To simplify the problem while keeping all necessary elements, a two-agent gridworld is considered as in Figure 1b. Merge player Agent 1 (marked as yellow circle) moves from right to left while lane player Agent 2 (marked as red rectangular) moves from bottom to top.



(a) Real-world Scenario



(b) Simulated Scenario

Figure 1: On-ramp merge

2 Methodology

2.1 Single-Agent Q-learning

In single-agent reinforcement learning, the environment of the agent is described by a Markov decision process. The general learning framework is based on the model of Figure 2, where an agent interacts with the environment by selecting actions to take and then perceiving the effects of those actions, a new state and a reward signal indicating if it has reached some goal (or has been penalized, if the reward is negative). The objective of the agent is to maximize Environment some measure over the rewards, like the sum of all rewards after a number of actions taken [7].

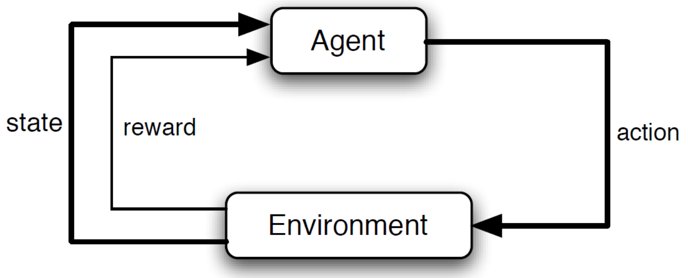


Figure 2: The general reinforcement learning framework

Q-earning is a popular and widely used form of reinforcement learning which determines optimal actions in a Markovian domain. Q-learning iteratively calculate the Q-function (action-value function) [10]:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}} \tag{1}$$

where r_t is the reward observed for the current state s_t, and α is the learning rate $(0 < \alpha \leq 1)$.

Before learning has started, Q is initialized to a possibly arbitrary fixed value (chosen by the programmer). Then, at each time t the agent selects an action a_t and observes a reward r_t and a new state s_{t+1} that may depend on both the previous state s_t and the selected action, Q is updated [1].

2.2 Multi-Agent Reinforcement Learning

The multi-agent framework is based on the same idea of Figure 2. However, there are several agents deciding on actions over the environment. The environments is no longer stationary - each agent probably has some effect on the environment and actions can have different outcomes depending on what the other agents are doing.

To model such a domain, game theory was introduced and the framework of stochastic games was put forth to model multi-agent systems. In a stochastic game, agents choose actions simultaneously in a discrete state space and action space.

2.2.1 Fictitious Play

Fictitious Play is a belief-based learning rule, i.e., players form beliefs about opponent play from the entire history of past play and behave rationally with respect to these beliefs.

Each player assumes that his opponent is using a stationary mixed strategy, and updates his beliefs about this stationary mixed strategies at each step. Players choose actions in each period to maximize that period's expected payoff given their prediction of the distribution of opponent's actions, which they form according to [6]

$$\mu_i^t(a_{-i}) = \frac{\eta_i^t(a_{-i})}{\sum_{a_{-i}\in A_{-i}} \eta_i^t(a_{-i})}$$

i.e., player i forecasts player $-i'$s strategy at time t to be the empirical frequency distribution of past play

Given player i's forecast rule, he chooses his action at time t to maximize his payoffs, so

$$a_i^t \in arg\ max\ r_i(a_i, \mu_i^t)$$

2.2.2 Nash Q-learning

The Nash Q-learning [4] [3] algorithm resembles standard single-agent Q-learning in many ways, but differs on one crucial element: how to use the Q-values of the next state to update those of the current state. Nash Q-learning algorithm updates with future Nash equilibrium payoffs, whereas single-agent Q-learning updates are based on the agent's own maximum payoff. learn these Nash equilibrium payoffs, the agent must observe not only its own reward, but those of others as well.

Table 1: Definitions of Q-values

	Multi-agent	Single-agent
Q function	$Q(s, a^1, ..., a^n)$	$Q(s, a)$
Optimal Q value	Current reward + Future rewards when all agents play specified Nash equilibrium strategies from the next period onward	Current reward + Future rewards by playing the optimal strategy from the next period onward

Table 1 summarizes the difference between single agent systems and multi-agent systems on the meaning of Q-function and Q-values.

This Nash Q-learning algorithm is summarized in Figure 3.

```
Initialize:
  Let $t = 0$, get the initial state $s_0$.
  Let the learning agent be indexed by $i$.
  For all $s \in S$ and $a^j \in A^j$, $j = 1,\dots,n$, let $Q_t^j(s, a^1,\dots,a^n) = 0$.
Loop
  Choose action $a_t^i$.
  Observe $r_t^1,\dots,r_t^n; a_t^1,\dots,a_t^n$, and $s_{t+1} = s'$
  Update $Q_t^j$ for $j = 1,\dots,n$
    $Q_{t+1}^j(s, a^1,\dots,a^n) = (1-\alpha_t)Q_t^j(s, a^1,\dots,a^n) + \alpha_t[r_t^j + \beta NashQ_t^j(s')]$
    where $\alpha_t \in (0,1)$ is the learning rate
  Let $t := t+1$.
```

Figure 3: Nash Q-learning algorithm

where $NashQ_t^i(s')$ is agent i's payoff in state s' for the selected equilibrium. Nash equilibria is solved using 'support enumeration' algorithm [8].

3 Implementation

3.1 Problem Representation

- Number of agents : n = 2
- Action space for agent i: $A_i = \{maintain, decelerate, accelerate\}$
- State space: $S = (1,4), (2,5), ...$ where a state $s = (l^1, l^2)$ represents the agents' joint location.
- Reward function for agent i:

$$f(x) = \begin{cases} 1000 & (L(l^1,a^1) = 10 \; or \; L(l^2,a^2) = 10) \; and \; L(l^1,a^1) \neq L(l^2,a^2), successful \; merge \\ -10000 & L(l^1,a^1) = L(l^2,a^2) \; or \; (l^1 < l^2 \; and \; L(l^1,a^1) > L(l^2,a^2)) \; or \; (l^1 > l^2 \; and \; L(l^1,a^1) < L(l^2,a^2)), collision \\ -1 & L(l^i,a^i) - l_i = 1, decelerate \\ 0 & L(l^i,a^i) - l_i = 2, maintain \; speed \\ -1 & L(l^i,a^i) - l_i = 3, accelerate \end{cases} \quad (2)$$

where $L(l, a)$ is the potential new location resulting from choosing action a in position l.

3.2 Modeling and Simulation

Both algorithms were implemented in Python and a simulator was built with TKinter. All the source code can be found on GitHub: https://github.com/HLiuUS/cis579.

4 Results

4.1 Factitious Play

Table 2 shows the Q values of Agent 1 and Agent 2 at starting position $(1, 4)$ after 3000 episodes of play. It is obvious that the optimal joint action for (Agent 1, Agent 2) is (decelerate, accelerate), which leds the agents to a new joint position $(2, 7)$. We can also note that if both agents choose the joint action (accelerate, accelerate), there will be large value penalty, which aligns with the observation that they would collide.

Table 2: Q-values at state (1, 4) after 3000 episodes

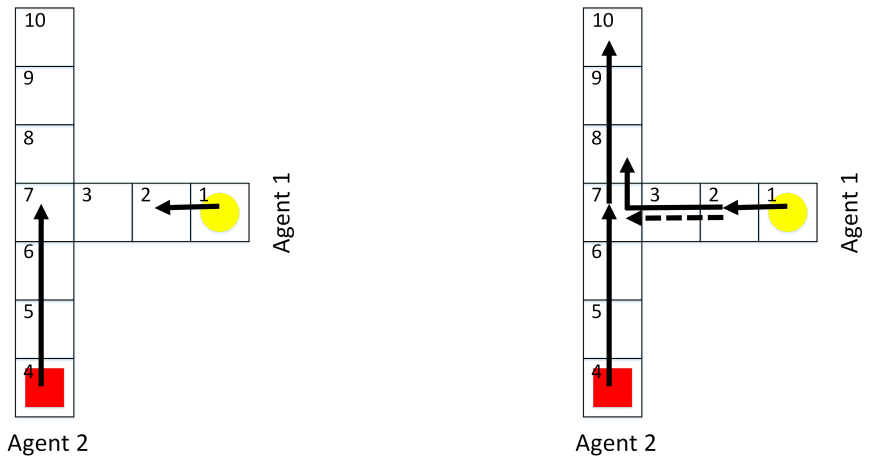
	decelerate	maintain	accelerate
decelerate	-741.16, -741.16	-1.31, 0.00	461.52, 461.52
maintain	7.76, 6.21	-3733.57, -3733.57	401.61, 400.21
accelerate	444.07, 444.07	361.72, 363.08	-19001.90, -19001.90

Table 3 shows the Q values of Agent 1 and Agent 2 at the new position $(2, 7)$. The Q values for joint action (decelerate, accelerate) and joint action (accelerate, accelerate) are the same. So Agent 1 should keeping accelerating while Agent 2 can either accelerate or decelerate to have a successful merge.

Table 3: Q-values at state (2, 7) after 3000 episodes

	decelerate	maintain	accelerate
decelerate	-211.07, -211.07	-0.10, 0.00	521.18, 521.18
maintain	-186.39, -186.75	-6190.93, -6190.93	409.51, 409.10
accelerate	-6216.83 , -6216.83	-851.73, -851.24	521.18, 521.18

The path resulted from fictitious play is shown in Figure 4.



(a) Fictitious Play path at (1, 4) (b) Fictitious Play path at (2, 7)

Figure 4: Results from Fictitious Play

4.2 Nash Q-learning

Table 4 shows the Q values of Agent 1 and Agent 2 at starting position $(1, 4)$ after 3000 episodes of play. The optimal joint action for (Agent 1, Agent 2) is (decelerate, accelerate), which is exactly the same as from fictitious play and again results in a new position $(2, 7)$.

Table 4: Q-values in state (1, 4) after 3000 episodes

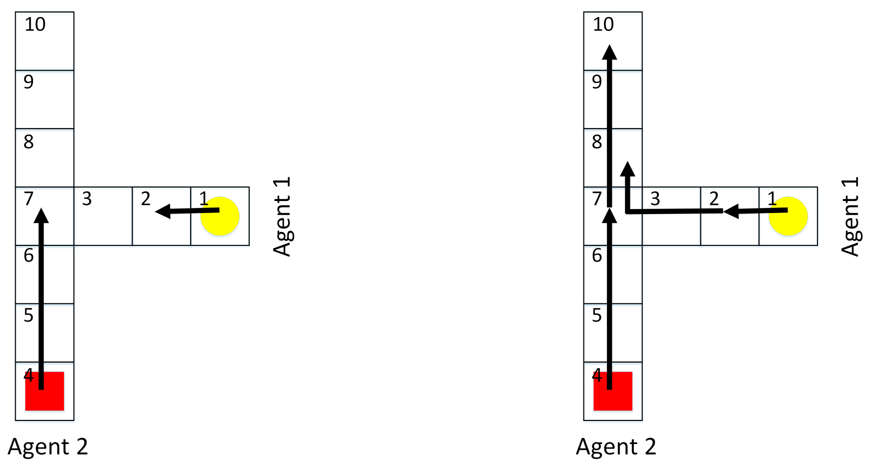
	decelerate	maintain	accelerate
decelerate	-743.09, -743.09	65.22, 67.13	851.05, 851.05
maintain	0.00, -1.31	-5524.48, -5524.48	366.65, 365.29
accelerate	506.10, 506.10	236.09, 237.33	-10897.53, -10897.53

Table 5 shows the Q values of Agent 1 and Agent 2 at new position $(2, 7)$, and now the optimal joint action for (Agent 1, Agent 2) is (accelerate, accelerate), which results in a successful merge. The Q values for action (decelerate, accelerate) and action (accelerate, accelerate) are pretty close, which helps to explain the tie between these actions in fictitious play results.

The path resulted from Nash Q-learnging is shown in Figure 5.

Table 5: Q-values in state (2, 7) after 3000 episodes

	decelerate	maintain	accelerate
decelerate	-214.77, -214.77	264.17, 265.38	946.72, 946.72
maintain	-1790.13, -1791.25	-4125.34, -4125.34	911.37, 910.46
accelerate	-8452.73, -8452.73	-3478.32, -3477.04	999.00, 999.00



(a) Nash equilibrium path at (1, 4) (b) Nash equilibrium path at (2, 7)

Figure 5: Results from Nash Q-learning

4.3 Conclusion

The optimal strategy for lane merging generated from fictitious play and Nash Q-learning are similar. They both favor that Agent 2 (lane player) always accelerate when it comes to lane merging. Agent 1 (merge player) should decelerate firstly and then accelerate.

5 Future Works

We envision that there are a few other topics that is worth further pursuing.

1. `Add more agents to increase complexity`. With more agents being added into the environment, the complexity of the simulation would increase exponentially due the fact that all the agents will interact with each other all the time. To solve this problem, for example, Hu and Gao [5] proposed a multi-agent learning architecture with sparse interactions.

2. `Explore other MARL algorithms`. Nash Q-Learning is used in this work. However, concerns have been voiced regarding their usefulness. For instance, in [2] it is argued that the link between stage-wise convergence to Nash equilibria and performance in the dynamic stochastic game is unclear. A family of possible algorithms [2] are listed below:

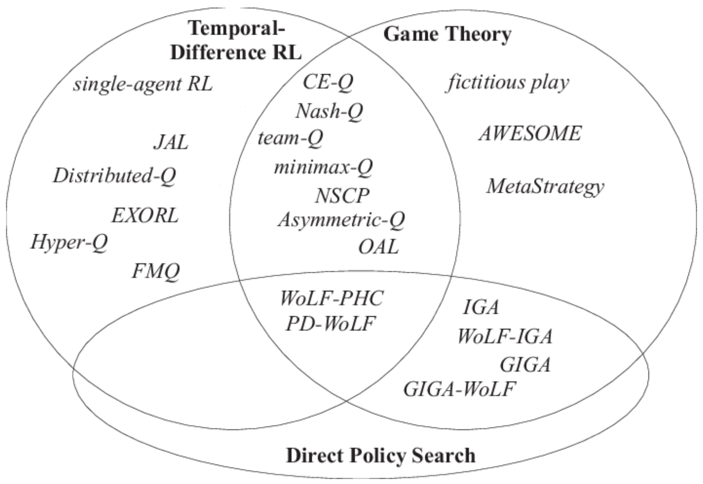


Figure 6: MARL techniques

3. `Deep Q-learning`. The vehicles' state space and action space are actually continuous instead of discrete, which makes it impractical to use tabular setting

as in basic Q-learnings. Instead, Q-function approximation is a good way to deal with problems in a continuous state/action space [9].

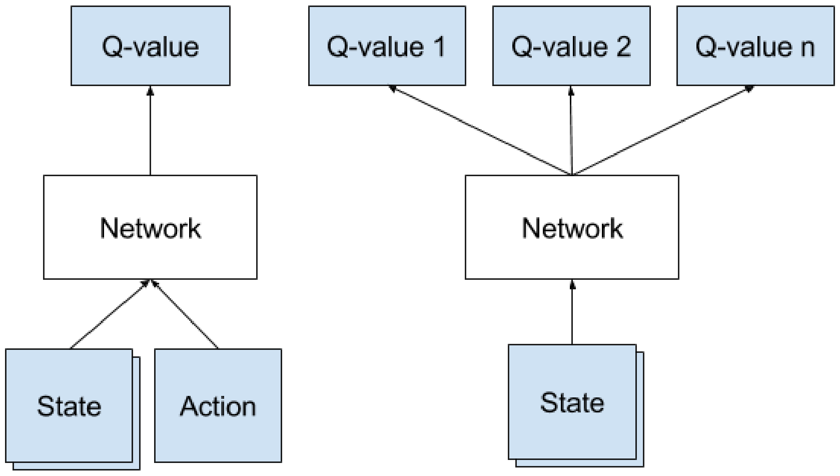


Figure 7: Deep Q-learnins

References

[1] Q-learning. `https://en.wikipedia.org/wiki/Q-learning`. Accessed: 2017-12-3.

[2] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*, 2008.

[3] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov):1039–1069, 2003.

[4] Junling Hu, Michael P Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pages 242–250, 1998.

[5] Yujing Hu, Yang Gao, and Bo An. Learning in multi-agent systems with sparse interactions by knowledge transfer and game abstraction. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 753–761. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[6] Jonathan Levin. Supermodular games. *Lectures Notes, Department of Economics, Stanford University*, 2003.

[7] Gonçalo Neto. From single-agent to multi-agent reinforcement learning: Foundational concepts and methods. *Learning theory course*, 2005.

[8] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a nash equilibrium. *Games and Economic Behavior*, 63(2):642–662, 2008.

[9] Pin Wang and Ching-Yao Chan. Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge. *arXiv preprint arXiv:1709.02066*, 2017.

[10] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.