Multi-Agent Reinforcement Learning for Multi-Car Racing

Kaiqing Zhang Sahika Genc

August 6, 2019

Abstract

In this project, we study the problem of multi-car racing using multi-agent reinforcement learning (MARL), with application to the platform of DeepRacer.

1 Introduction

2 Problem Formulation

In this section, we present some preliminary background for multi-agent reinforcement learning, game theory, and multi-car racing for DeepRacer.

2.1 Multi-Agent Reinforcement Learning

In order to model the interaction among agents, a general framework of *Markov games* has been used in the literature of MARL (Littman, 1994). In particular, a Markov game $\mathcal{G}$ is usually characterized by a tuple

$$\mathcal{G} := \langle \mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, \{R^i\}_{i\in\mathcal{N}}, P, \gamma \rangle,$$

where $\mathcal{N}$ denotes the set of N agents, $\mathcal{S}$ denotes the state space that is common to all agents, $\mathcal{A}^i$ denotes the action space of agent $i \in \mathcal{N}$. $R^i : \mathcal{S} \times \mathcal{A}^1 \times \cdots \times \mathcal{A}^N \to \mathbb{R}$ represents the reward function of agent i, which is dependent on the state and the joint action of all agents. $P : \mathcal{S} \times \mathcal{A}^1 \times \cdots \times \mathcal{A}^N \to \Delta(\mathcal{S})$ represents the state transition probability that is a mapping from the current state and the joint action to the probability distribution over the state space. $\gamma \in (0, 1]$ is the discounting factor.

At each time t, each agent selects its own action $a_t^i \in \mathcal{A}^i$ in face of the system state s_t, according to its own policy $\pi^i : \mathcal{S} \to \Delta(\mathcal{A}^i)$, which is a mapping from the state space to the probability distribution over action space $\mathcal{A}^i$. Then the system transits to the next state s_{t+1} and each agent i receives the instantaneous reward $r_t^i = R^i(s_t, a_1^1, \cdots, a_t^N)$. The goal of each agent i is to maximize the long-term return J^i calculated using r_t^i, i.e.,

$$\max_{\pi^i} \quad J^i(\pi^i, \pi^{-i}) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t^i \,\middle|\, s_0, a_t^i \sim \pi^i(\cdot|s_t)\right] \tag{2.1}$$

where $-i$ represents the indices of all agents except agent i, and $\pi^{-i} := \prod_{j \neq i} \pi^j$ refers to the joint policy of all agents except agent i. Note that different from the setting of single-agent RL, the objective of agent i not only depends on its own policy π^i, but also on others' joint policy π^{-i}. Therefore, from the perspective of a single-agent i, the problem is no longer stationary and Markov (Busoniu et al., 2008). In the same vein, one can define the value and action-value(Q)-function for each agent i as follows

$$V^i(s) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t^i \,\middle|\, s_0 = s, a_t^i \sim \pi^i(\cdot|s_t)\right],$$

$$Q^i(s, a^1, \cdots, a^N) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t^i \,\middle|\, s_0 = s, a_0^i = a^i, a_t^i \sim \pi^i(\cdot|s_t)\right].$$

Due to the coupling of agents' policies in J^i, the solution concept of maximizing the return of a single agent is unattainable. Instead, one commonly used solution concept is the *Nash equilibrium* (NE) of the game. Specifically, the NE is defined as the point of a joint policy $\pi_* := (\pi_*^1, \cdots, \pi_*^N)$ at which

$$J^i(\pi_*^i, \pi_*^{-i}) \geq J^i(\pi^i, \pi_*^{-i}), \quad \forall i \in \mathcal{N}, \tag{2.2}$$

namely, given all other agents' equilibrium policy π_*^{-i}, there is no motivation for agent i to deviate from π_*^i. Hence, the goal of MARL is to solve for the NE of the Markov game $\mathcal{G}$ without the knowledge of the model.

On the other hand, if all the opponents policy π^{-i} is fixed, the problem for agent i reduces to a single-agent decision-making problem $\mathcal{M}^i := \langle \mathcal{S}, \mathcal{A}^i, \widetilde{R}^i, \widetilde{P}^i, \gamma \rangle$, where

$$\widetilde{R}^i(s, a^i) := \int_{\mathcal{A}^{-i}} R^i(s, a^i, a^{-i}) \pi^{-i}(a^{-i}|s) da^{-i}$$

$$\widetilde{P}^i(\cdot|s, a^i) := \int_{\mathcal{A}^{-i}} P(\cdot|s, a^i, a^{-i}) \pi^{-i}(a^{-i}|s) da^{-i}.$$

To simplify the notation, we also define the *best-response* operator $\mathcal{B}^i : \Pi^{-i} \to \Pi^i$, where Π^i represents the policy space for agent i, as follows:

$$\mathcal{B}^i(\pi^{-i}) := \operatorname*{argmax}_{\pi^i} \; J^i(\pi^i, \pi^{-i}).$$

Let $\mathcal{B} := (\mathcal{B}^1, \cdots, \mathcal{B}^N)$, then the NE is the fixed point of the operator $\mathcal{B}$, i.e., $\pi_* = \mathcal{B}(\pi_*)$. In addition, viewing the NE using the best-response operator motivates the use of *opponent-modeling* in the algorithm design later.

2.2 Multi-Car Racing for DeepRacer

In general, multi-car racing with the current DeepRacer system configuration can be modeled as a Markov game with partial-observability. For notational simplicity, we consider the setting with only two cars[1]. Specifically, the state $s = (s^1, s^2)$ is the position of both cars on the track, where $s^i = (x^i, y^i)$ denotes the coordinates of the car i on the map. The action of each car i is its velocity $a^i = v^i = (v^i_x, v^i_y)$. The transition probability model is the physical model that deterministically describes the advance of all cars, i.e.,

$$s^i_{t+1} = s^i_t + \Delta t * (a^i_t).$$

Each car has its own reward function $R^i(s, a^1, a^2)$, which characterizes the goal of both finishing the racing as soon as possible and avoiding collision with the opponent during racing. Note that both goals are dependent on the state and joint action of both agents. Details of engineering the reward function can be found in the quip document Zhang (2019) and will not be discussed here.

In the current single-car setting, the DeepRacer has access to the image stream from the camera it carries. The image, which is in fact the *observation*, has been treated as the *state*, and has achieved success previously in both simulations and sim-to-real experiments. The observation is denoted by $O^i(s', a^1, a^2)$. This is mainly due to the fact that the state in single-car setting is the position of the car, which can be identified by the image observation (almost) without ambiguity. However, such partial observability will cause great challenge in the multi-agent setting, since the state here depends on the position of all cars, which cannot be completely captured by the image of the camera on a single car. This problem can be generally modeled as a partially-observed Markov/Stochastic game (POSG) (Hansen et al., 2004), which has been notoriously known to be intractable since its algorithmic complexity grows exponentially with the number of agents, and double-exponentially with the horizon. Specifically, this is because in POSG, different agents have different observations, which leads to different beliefs over the state. In fact, it has been show that the optimal strategy of a single-agent in POSG relies on the belief not only over the state, but also over the strategies of the opponents (Hansen et al., 2004). An alternative solution is to enable each agent to have access to the *perfect observations* of all other agents, so that each agent will main identical belief over the state, which reduces the multi-agent decision making to a single-agent one (but each still needs to solve a game even with such perfect information). Nonetheless, such an approach requires unlimited bandwidth as well as instantaneous and noiseless communication between agents, which is impractical for real systems like DeepRacer, especially in the racing setting where all agents are competing instead of cooperating and may not have the motivation to share such information.

To tackle such challenges, we propose two potential solutions that are prac-

[1] Note that our formulation and algorithm design later can be readily generalized to the setting with $N > 2$ cars.

tical, considering the current software and hardware assets we have on DeepRacer, as to be introduced below.

2.2.1 The Use of *God Camera*: Remove Partial Observability

The first solution is to use the so-called *God Camera*, which can be mounted on the ceiling of the racing room and can observe the position of all cars. This is easy to implement and is viable hardware-wise, especially with the aid of the *DeepLense* sensor from Amazon. All cars can have access to the image stream collected from the God Camera, which brings back the problem to the fully observed setting, with the God Camera image being the state. This is also relatively easy to implement in the simulation, by simply adding a camera stream fed to all agents in Gazebo. This minor change of hardware greatly simplifies the problem, so that we can test some computationally feasible MARL algorithms developed for the fully observed setting, for example, minimax-Q learning (Littman, 1994), policy gradient for zero-sum Markov games (Pinto et al., 2017).

Due to its easiness to implement, we can use this solution as the baseline environment for testing MARL algorithms. Additionally, we note that the use of this God Camera may help improve the performance of single-car racing as well.

2.2.2 Centralized Training & Decentralized Execution

The second solution is to use the idea of *Centralized Training & Decentralized Execution* originated from Foerster et al. (2016); Lowe et al. (2017). Specifically, during training which is conducted in simulations for DeepRacer, extra information, for example, observations and actions of other agents, can be used to ease the training. While during execution, these pieces of information are not available and only local observations can be used as input to the policy. Notationally, the Q-value function at each agent i is heuristically approximated by $Q^i(o^1,\cdots,o^N,a^1,\cdots,a^N)$, i.e., a function of joint observation $o=(o^1,\cdots,o^N)$ and joint action $a=(a^1,\cdots,a^N)$, so is the value function $V^i(o^1,\cdots,o^N)$. In contrast, the policy at each agent i is *local* in the sense that it only takes o^i as input, i.e., $\pi^i(a^i\,|\,o^i)$.

In this setting, it is unnatural to develop Q-learning or other value-based RL algorithms, since in general Q-function cannot contain different information at training and testing time (Lowe et al., 2017). Hence, actor-critic/policy-gradient methods become feasible choices, since it is the critic that uses other agents' information, and as long as the actor is trained to use only local observations as input, it will remain decentralized during testing. After testing MARL algorithms using the first solution setup, we can switch to this setup and compare the performance with the baseline above.

3 Algorithms

According to the formulation in §2, we propose three stages of algorithm design tasks for multi-car racing. The three stages are categorized based on the types of opponent behavior. After that, we discuss the approaches to do opponent modeling, a core immediate step in most of the three stages.

3.1 Fixed-policy Opponent

In this stage, we consider the opponent as an agent with a fixed policy that is not changing over time. For example, in the setting of two-car racing, the opponent car is either staying still or moving with a constant speed along the track. The goal of the learning car is to avoid collision with the opponent car and finish the race quickly. As mentioned in §2.1, from the perspective of the learning car, the problem is a single-agent decision-making problem, with certain reward and transition model that depend on the opponent policy. Since the opponent is not making decisions simultaneously, and the environment is stationary and Markovian for the learning car. Therefore, the optimal policy for the learning agent can be obtained via standard single-agent RL algorithms. For example, we can start with the proximal policy optimization (PPO) algorithm (Schulman et al., 2017) that has been used in single-car racing. Now we develop algorithms for both solution settings in §2.2.

With God Camera

With full observability of the state, if the opponent policy is additionally fixed, then the problem becomes an MDP. This way, we only need to: i) change the reward function, so that collision is penalized; ii) change the state from the local camera image to the God Camera image. See details of designing collision-avoiding reward in Zhang (2019).

Centralized Training & Decentralized Execution

This setting allows partial observability of each agent. We can build up our algorithm based on the actor-critic algorithm in Lowe et al. (2017). This reduces to the same information structure as the current *object-avoidance* task. What we need to do is to use a centralized trainer to provide the joint action information to the learning car. Essentially, this changes our current single-agent PPO algorithm to the actor-critic one in Lowe et al. (2017), just with the environment changed from the object-free one to the one with object.

We note that in both cases, there is no need to *model the opponent*, since its policy is not changing over time. In fact, any instantaneous action of the other agent reflects the behavior of the time-invariant policy. Specifically, for the fully observed setting, we can view (s_t, a_t^{-i}) as the joint state, which preserves the Markov and stationary property. Such a property also holds even in the second setting with partial observability.

3.2 Adaptive-policy Opponent

In this stage, the opponent is also not assumed to be strategic, i.e., the opponent is not a game player and its policy is not optimal in its own best interest. The opponent policy may be fixed for a period of time, but can change over time. Therefore, it is not sufficient to only use instantaneous action of the opponent to infer its behavior. Instead, the learning car needs to use the historical data of the opponent's action to approximate its policy, and adapt to learn a best-response to that policy. Moreover, the learning car may also need to update the approximation of the opponent policy on the fly, so that it keeps track of the change of the opponent behavior over time. In order to make the setting sensible, we assume that the change of the opponent policy is slower in comparison to the adaptation of the learning policy in response to it.

With God Camera

The algorithm may be implemented in a double-loop fashion. In particular, in the inner loop, the learning car can model the opponent policy as a mapping from the state space, i.e., the God Camera image, to its own action space, using techniques to be introduced in §3.4. Here, the instantaneous action of the opponent a_t^{-i}, i.e., its velocity at time t, need to be observed. This is available in training in the simulations. It then uses single-agent RL algorithms to learn the best-response policy. In the outer loop, the opponent policy is updated as the dataset of the opponent actions updates, every several inner loops.

Centralized Training & Decentralized Execution

The algorithm is also double-loop as above. The main difference is that in the inner loop, the opponent policy is modeled as a mapping from its observation space, i.e., its local camera image, to its action space. As a result, the opponent's instantaneous observation is also needed, in addition to its instantaneous action. In the centralized trainer, both pieces of information are available.

3.3 Game-theoretic Opponent

In this stage, both the learning the opponent cars are players of the Markov game, i.e., both agents are strategic. Specifically, both agents aim to maximize the long-term return corresponding to their own reward functions, while considering the involvement of the other agent in decision-making.

With God Camera

In the fully observed setting, it can be shown that the Markov game can be solved by solving each stage normal-form game in a dynamic programming manner (Littman, 1994). Hence, as long as it is assumed that the opponent is a rational learner, each stage game can be solved by solving a linear program (Myerson, 2013).

We note that there is no need to model/infer the opponent policy using historical data, since its behavior is more or less predictable, and can be obtained in only the learning car's mind. For example, if value-based approaches, say, minimax-Q learning (Littman, 1994), are used, then the opponent policy can be obtained implicitly by solving the stage games. Moreover, for policy-based approaches, the learning car can utilize policy gradient to improve the opponent policy in an online fashion (Pinto et al., 2017).

We also note that if the convergence of the opponent policy is faster than that of the learning car's policy, namely, the opponent policy is almost unchanged during the policy optimization of the learning car, such an approach can be viewed as the *fictitious play* in game theory (Brown, 1951; Monderer and Shapley, 1996). The basic idea behind fictitious play is that each player maintains an approximation of the opponent policy using the historical action data, and takes either best-response or better-response to the fictitious policy. It has been shown that if all agents follow this play, the algorithm converges to the Nash equilibrium of certain normal-form games (Monderer and Shapley, 1996; Shamma and Arslan, 2005). However, for stochastic/Markov games with states, it has been shown that vanilla fictitious play may diverge (Schoenmakers et al., 2007). Even though, such an idea has been widely applied in empirical RL studies, and is worth trying in our multi-car racing problem.

Centralized Training & Decentralized Execution

For the partially observed setting, the actor-critic based approach in Lowe et al. (2017) directly applies, since it can handle both cooperative and competitive settings. Note that here in the centralized trainer, we can either model/infer the opponent policy as in He et al. (2016); Lowe et al. (2017), or we can use alternative policy improvement as in the fully observed setting in Pinto et al. (2017).

3.4 Opponent Modeling

One core immediate step that is most of the algorithm design above is opponent modeling/inference. Several common approaches are summarized in the sequel. Note that we consider the partially observed setting as an example, since the fully observed one is included by letting the state be the observation. For notational convenience, we assume agent 1 is the learning car, and agent 2 is the opponent car.

Log Probability Maximization

The opponent policy is approximated by $\pi^2_{\theta^2_1} : \mathcal{O}^2 \rightarrow \mathcal{A}^2$, a mapping from its observation space to the probability distribution over its action space, that is parameterized by θ^2_1. Here we use θ^2_1 to denote the parameter of the policy of agent 2 from agent 1's perspective. Then, the parameter θ^2_1 is determined by solving the following optimization problem:

$$\theta^2_1 := \underset{\theta}{\text{argmax}} \quad \mathbb{E}_{a^2,o^2}\Big\{\log \pi^2_{\theta^2_1}(a^2 \,|\, o^2) + \lambda H\Big[\pi^2_{\theta^2_1}(\cdot \,|\, o^2)\Big]\Big\}, \tag{3.1}$$

where o^2 and a^2 represent the observation and action of agent 2, $\lambda > 0$ is a coefficient that balances the two objectives, and $H\Big[\pi^2_{\theta^2_1}(\cdot|o^2)\Big]$ denotes the entropy of the probability distribution $\pi^2_{\theta^2_1}(\cdot|o^2)$, defined as

$$H\Big[\pi^2_{\theta^2_1}(\cdot|o^2)\Big] := \sum_{a^2 \in \mathcal{A}^2} -\pi^2_{\theta^2_1}(a^2|o^2) \cdot \log \pi^2_{\theta^2_1}(a^2|o^2).$$

Note that the objective (3.1) aims to maximize the log probability of the data in $\mathcal{D}^2$ occurs, and also adds the entropy regularization to incentive exploration for the inferred policy.

Given a replay buffer $\mathcal{D}^2 := \Big\{a^2_1, o^2_1, \cdots, a^2_T, o^2_T\Big\}$, the objective function in (3.1) can be approximated as

$$\begin{aligned} &\mathbb{E}_{a^2,o^2}\Big\{\log \pi^2_{\theta^2_1}(a^2|o^2) + \lambda H\Big[\pi^2_{\theta^2_1}(\cdot|o^2)\Big]\Big\} \\ &\quad \approx \frac{1}{T} \sum_{(a^2_t, o^2_t) \in \mathcal{D}^2} \Big\{\log \pi^2_{\theta^2_1}(a^2_t|o^2_t) + \lambda H\Big[\pi^2_{\theta^2_1}(\cdot|o^2_t)\Big]\Big\}. \end{aligned} \tag{3.2}$$

The approximate objective can be maximized by any optimization techniques. As the replay buffer $\mathcal{D}^2$ changes, the optimization of (3.2) is re-solved on the fly.

Self-Play

Another way to model the opponent is to assume that the opponent uses the same policy as the learning agent. This is also known as *self-play* in the game-theoretic settings. Therefore, whenever the the learning car maintains a policy, we can just use its current or previous versions as the opponent policy. This way, the learning car does not need to access the observation and action history of the opponent, which enables a *decentralized training* architecture.

Implicit Modeling With Neural Nets

Motivated by He et al. (2016), we can also use the hidden layer of the neural network to model the opponent policy. In contrast to the explicit opponent policy modeling, the opponent behavior is encoded in the architecture of the neural networks. See §3.2 in He et al. (2016) for more details.

Representation Learning Approach

Recently, a relatively new facet of opponent modeling is enabled by learning representations for MARL using neural networks. Such approaches impose a certain model structure to compute the *representation* of the opponent, which takes the opponent observation as input, and predicts specific information about the opponent, such as their actions (Grover et al., 2018) or returns (Tacchetti et al., 2018) received by the modeled agent. The subtle difference from the explicit opponent policy modeling above is that the *model* here may be more complicated than just the policy. The policy network of the learning car is then trained by receiving its own observations concatenated with output representations from the representation network. Compared to explicit opponent policy modeling, this representation learning approach has been claimed to have better generalizability to opponents that have yet been encountered.

References

Brown, G. W. (1951). Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, **13** 374–376.

Busoniu, L., Babuska, R. and De Schutter, B. (2008). A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*.

Foerster, J., Assael, Y. M., de Freitas, N. and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*.

Grover, A., Al-Shedivat, M., Gupta, J. K., Burda, Y. and Edwards, H. (2018). Learning policy representations in multiagent systems. *arXiv preprint arXiv:1806.06464*.

Hansen, E. A., Bernstein, D. S. and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *AAAI*, vol. 4.

He, H., Boyd-Graber, J., Kwok, K. and Daumé III, H. (2016). Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning*.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P. and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*.

Monderer, D. and Shapley, L. S. (1996). Fictitious play property for games with identical interests. *Journal of Economic Theory*, **68** 258–265.

Myerson, R. B. (2013). *Game Theory*. Harvard University Press.

Pinto, L., Davidson, J., Sukthankar, R. and Gupta, A. (2017). Robust adversarial reinforcement learning. In *International Conference on Machine Learning*.

Schoenmakers, G., Flesch, J. and Thuijsman, F. (2007). Fictitious play in stochastic games. *Mathematical Methods of Operations Research*, **66** 315–325.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shamma, J. S. and Arslan, G. (2005). Dynamic fictitious play, dynamic gradient play, and distributed convergence to nash equilibria. *IEEE Transactions on Automatic Control*, **50** 312–327.

Tacchetti, A., Song, H. F., Mediano, P. A., Zambaldi, V., Rabinowitz, N. C., Graepel, T., Botvinick, M. and Battaglia, P. W. (2018). Relational forward models for multi-agent learning. *arXiv preprint arXiv:1809.11044*.

Zhang, K. (2019). Reward function design for multi-car racing. https://quip-amazon.com/fpfoA9DUYyG1.