Learning to Drive using Inverse Reinforcement Learning and Deep Q-Networks

Conference Paper · December 2016

CITATIONS
24

READS
868

4 authors, including:

Sahand Sharifzadeh
Ludwig-Maximilians-University of Munich
8 PUBLICATIONS **34** CITATIONS
SEE PROFILE

Rudolph Triebel
Technische Universität München
99 PUBLICATIONS **2,383** CITATIONS
SEE PROFILE

Daniel Cremers
Technische Universität München
477 PUBLICATIONS **25,131** CITATIONS
SEE PROFILE

Learning to Drive using Inverse Reinforcement Learning and Deep Q-Networks

Sahand Sharifzadeh[1] **Ioannis Chiotellis**[1] **Rudolph Triebel**[1,2] **Daniel Cremers**[1]
[1] Department of Computer Science, Technical University of Munich, Germany
{sharifza, chiotell, triebel, cremers}@in.tum.de
[2] Institute of Robotics and Mechatronics, Department of Perception and Cognition
German Aerospace Center (DLR), Oberpfaffenhofen-Weßling, Germany
rudolph.triebel@dlr.de

Abstract

We propose an inverse reinforcement learning (IRL) approach using Deep Q-Networks to extract the rewards in problems with large state spaces. We evaluate the performance of this approach in a simulation-based autonomous driving scenario. Our results resemble the intuitive relation between the reward function and readings of distance sensors mounted at different poses on the car. We also show that, after a few learning rounds, our simulated agent generates collision-free motions and performs human-like lane change behaviour.

1 Introduction

Robots and autonomous systems are becoming more and more a part of everyday life by assisting us in various different tasks. Therefore one important requirement for these systems is that they behave in a human-acceptable, socially normative way, e.g. by respecting personal spaces or treating groups based on the social relations of the individuals [1, 2]. This also means that humans should not just be regarded as obstacles and that an optimal robot motion must also consider human comfort metrics [3]. One of the most popular instances of autonomous systems in the current decade is self-driving cars. The ultimate goal of autonomous cars is to drive the passengers from one point to another without any human input, while assuring the comfort of human passengers. Defining "comfort" is not straight forward and this makes it hard to define a suitable objective function for motion planning. One widely used method aiming to fulfil this objective is introduced by Werling et al. [4]. They propose to provide "ease and comfort" by producing jerk-optimal trajectories. Later, lane change experiments conducted by Tehrani et al., in Japanese highways showed that the human lane change behavior cannot be modelled by a single stage of jerk-optimal trajectories [5, 6]. Instead they proposed a two-stage model. Many other models exist similar to these. However, they mostly fail to produce human-like behaviors. Modeling the human driving behavior becomes even more complicated when considering scenarios such as driving in large cities with many intersections, traffic lights, pedestrians, etc. Therefore, applying machine learning methods to extract the models directly from the expert demonstrations appears more promising.

In a recent work, Bojarski et al. [7], proposed an end-to-end supervised learning approach that maps the front facing camera images of a car to steering angles, given expert data. However, they require a large amount of data from different possible driving scenarios in order to give a good approximation of the policy. Still, they might fail when facing scenarios that are very different from the ones in the training data. A more promising formulation for this problem is based on Markov Decision Processes (MDPs). In this framework, one can apply Inverse Reinforcement Learning (IRL) to extract the unknown reward function of the driving behavior [8]. By approximating this function rather

Workshop on "Deep Learning for Action and Interaction", Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

than directly learning the state-action pairs in a supervised fashion, one can handle new scenarios better. Finding the reward function of an MDP using IRL is proposed by Ng and Russel [8] and further improved by Abbeel and Ng [9]. Since then, several variations of IRL have been proposed such as Bayesian IRL [10], maximum entropy based methods [11] and max margin prediction [12]. Most of the recent methods have been inspired by these works. For example, Wulfmeier et al. [13] proposed a maximum entropy based approach that handles nonlinear reward functions using deep neural networks. However, most of these approaches are limited to small state spaces that cannot fully describe real-world driving scenarios. One of the main reasons is the difficulty of applying the Reinforcement Learning (RL) step in large state spaces. While approximating the Q-function for large state spaces has been effectively addressed by Mnih et al., using a Deep Q-Network (DQN) [14], to the best of our knowledge, it has not been used in IRL methods before.

In this paper, we address the exploding state space problem and build upon the projection-based IRL method by [9] using a DQN. We implemented a highway driving simulator and evaluated the performance of our approach by analyzing the extracted rewards. The evaluation is presented in the Section 3.

2 Problem Formulation

A Markov Decision Process (MDP) is defined as the tuple (S, A, T, γ, R), where S and A are the state and action spaces, T is the transition matrix, $\gamma \in [0, 1]$ is the discount factor and $R : S \times A \to \mathbb{R}$ is the reward function. A *policy* $\pi : S \to A$ maps a state to an action. It can also be defined as a probability distribution over actions at each state. A *value* or *state-value* function $V^\pi(s_0)$ is defined as the expected discounted future reward if we start from initial state s_0 and act according to policy π:

$$V^\pi(s_0) = \mathbb{E}[\Sigma_t \gamma^t R(s_t)|\pi] \tag{1}$$

The discount factor applies the amount of uncertainty that we have about the future rewards. The action-value function Q represents the value that we can gain if we start from state s_0 and take action a, thereby applying policy π:

$$Q^\pi(s_0, a) = \mathbb{E}[\Sigma_t \gamma^t R(s_t)|\pi, a] \tag{2}$$

The motion planning problem can be formulated as a Markov Decision Process, in which finding the optimal action-value function, sometimes referred to as Q-function, is the goal. Given an unknown MDP, except for the reward function, Q-Learning methods can find the optimal Q-function [15][16]. Approximating the Q-function can help extend these methods to problems with larger state spaces. Using neural networks for this purpose had been shown to cause instabilities or divergence[17] in the past. In order to address these problems, Mnih et al., proposed two key ideas [14]. First, to randomly sample training data from the sequence of past experiences ("experience replay"). Second, to separately train a copy of the Q-network using predictions of the original network and only periodically replace the original network with the copy.

In motion planning, similar to many other real-world applications, the MDP, including the reward function, is unknown. Inverse Reinforcement Learning methods have been shown to effectively find the underlying reward function given expert demonstrations, when the MDP is small and known, except for the reward function [8] [11] [10]. These approaches are mainly based on an iterative refinement of the reward function followed by a Reinforcement Learning step. In order to apply such approaches to large state spaces, we propose using a Deep Q-Network [14] as the Reinforcement Learning step. Here, we apply this approach to the projection-based IRL method [9] but other IRL techniques can also benefit from it.

Given expert demonstrations, we want to generate policies π whose values are close to the value of the expert policies π_E:

$$\|V^\pi(s_0) - V^{\pi_E}(s_0)\| \leq \epsilon \tag{3}$$

Every state s_i is spanned by d-dimensional feature vectors $\phi(s_i)$ such as speed, acceleration, sensor readings, etc. The reward function is defined as a weighted linear combination of these features:

$$R(s_i) = w \cdot \phi(s_i)\,, \tag{4}$$

where $w \in \mathbb{R}^{\mathrm{d}}$ is the weight vector and $\|w\| \leq 1$. Plugging (4) into (1), we get

$$V^\pi(s_0) = w \cdot \mathbb{E}[\Sigma_t \gamma^t \phi(s_t)|\pi] \tag{5}$$

Algorithm 1: Projection-based IRL using DQN

1. Randomly initialize the parameters of DQN and some policy $\pi^{(0)}$, compute or approximate its features expectations $\mu^{(0)}$, and set $w^{(1)} = \mu_E - \mu^{(0)}$, $\bar{\mu}^{(0)} = \mu^{(0)}$ and $i = 2$.

2. Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T (\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T (\mu^{(i-1)} - \bar{\mu}^{(i-2)})} (\mu^{(i-1)} - \bar{\mu}^{(i-2)})$

3. Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$ and $t^{(i)} = \|w\|_2$.

4. If $t^{(i)} \leq \epsilon$, terminate

5. Update the DQN using $R = (w^{(i)})^T \phi$ and compute the optimal policy $\pi^{(i)}$.

6. Compute or estimate features expectations $\mu^{(i)}$ of the newly extracted policy.

7. Set $i = i + 1$ and go to step 2.

Furthermore, feature expectations are defined as:

$$\mu(\pi) = \mathbb{E}[\Sigma_t \gamma^t \phi(s_t) | \pi] \, . \tag{6}$$

Thus, the problem is reduced to generating trajectories whose feature expectations are similar to those of the expert. Abbeel & Ng proposed an iterative projection-based method to solve it [9]. In this paper, we propose to use a DQN in the RL step of their algorithm. The details of our method are given in Algorithm 1.

3 Evaluation

In this section, we present the evaluation results of the proposed approach. We considered the driving scenario in a highway and implemented a simulator for collecting expert trajectories and testing. The simulating environment was programmed in Python. The user interface is shown in Figure 2 on the right side. The red car is the agent being trained to drive. The dynamics of this car are implemented based on the single track model [18], with three degrees of freedom. The Deep Q-network architecture used in our approach is shown in Figure 1. It consists of an input layer of features, 2 fully connected hidden layers with 160 units each and rectifying nonlinear units, followed by a fully connected output layer to the actions.

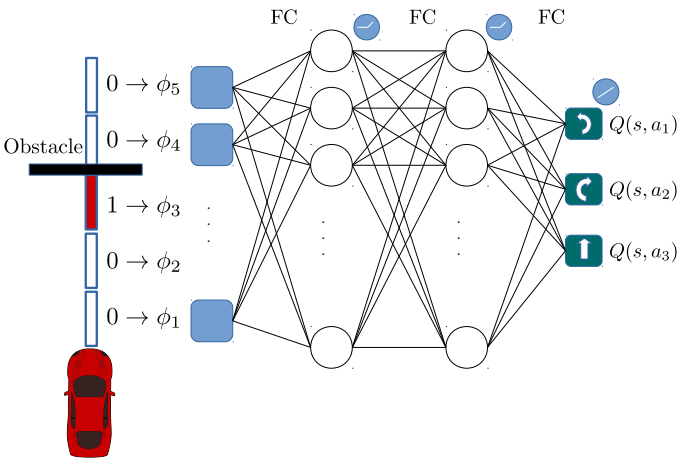

Figure 1: The architecture of the proposed Deep Q-Network. The input is the set of features and the output layer consists of action-values for three possible actions, to steer left, steer right or not steer.

In these experiments we used 13 sensors. The sensors had a maximum sensing radius 64% of the environment length, discretized to 16 bins of equal size. Each feature ϕ_k, indicated whether or not there was an obstacle in the interval of each bin. If the sensor was not sensing any obstacles, the maximum possible reading distance was assigned to it. Therefore, we had a total of 208 binary features which gave rise to 2^{52} possible states.

In the driving experiments by [9], the car could only have discrete transitions to the lane on its left or right. However, in our experiments, we allowed steering with three different angles $(0, \frac{\pi}{12}, -\frac{\pi}{12})$ giving rise to more realistic, continuous transitions. For simplification in our experiments we set the acceleration to zero. The highway in our experiments had 3 lanes and at most two other cars could appear in front of the agent. For training, we collected 90 expert demonstrations from this setup.

The algorithm achieved satisfactory results after only 6 IRL iterations with 3000 inner loop iterations each. Since we did not have access to the true reward function of driving, we evaluated our proposed algorithm in the following ways:

a. **Interpreting the extracted weights:** The extracted weights for features from 7 sensors are plotted in Figure 2. The color-code guide of each sensor is shown in the upper right corner. As shown in this figure, there is a nonlinear relation between readings of the same sensor and their extracted weights. This means that if the sensor readings had not been discretized into binary features, the algorithm would not have been able to capture the weights correctly. We have represented the

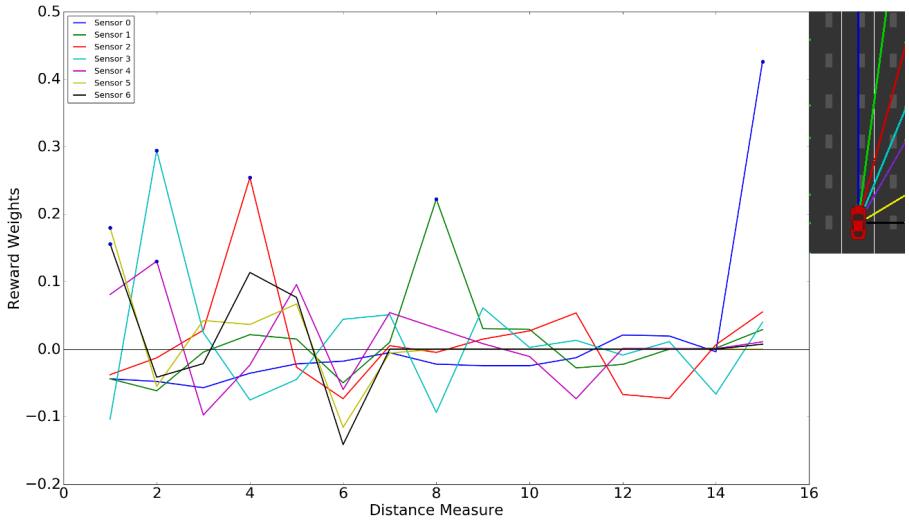

Figure 2: Weights extracted using the IRL algorithm plotted for features of 7 sensors. Each sensor has been assigned a color which is shown in the right corner.

maximum weight of features for each sensor with blue dots. As one can see, for Sensor 0 (blue), the higher the reading is, the larger the weight is. Meaning, the agent learns to keep as far away from the obstacles as possible. In the optimal state, where the car stays as far away from all obstacles and walls, the sensor readings form an ellipse. Therefore, as the angle of the sensor from the vertical axis increases, the reward weights peak at a smaller distance.

Another notable observation is that Sensor 6 (side sensor), gets the highest weight when reading the smallest distance from the obstacles. The explanation is that at this distance, the car is placed next to a highway wall. This has been the expert's preference in the demonstrations. This reward weight immediately drops when the reading is increased, which perfectly explains that staying in the lanes is preferable to driving between the lanes. The same can be observed for most of the other sensors. Note that some of the distances were never read by the sensors during the experiments and the extracted weights for these cases is 0. For example Sensors 5 and 6 never see obstacles in distances higher than the width of the highway.

b. **Comparing feature expectation values of the expert to the trained agent:** We trained the DQN using the rewards computed by the final weights. Then, we let the agent drive for several

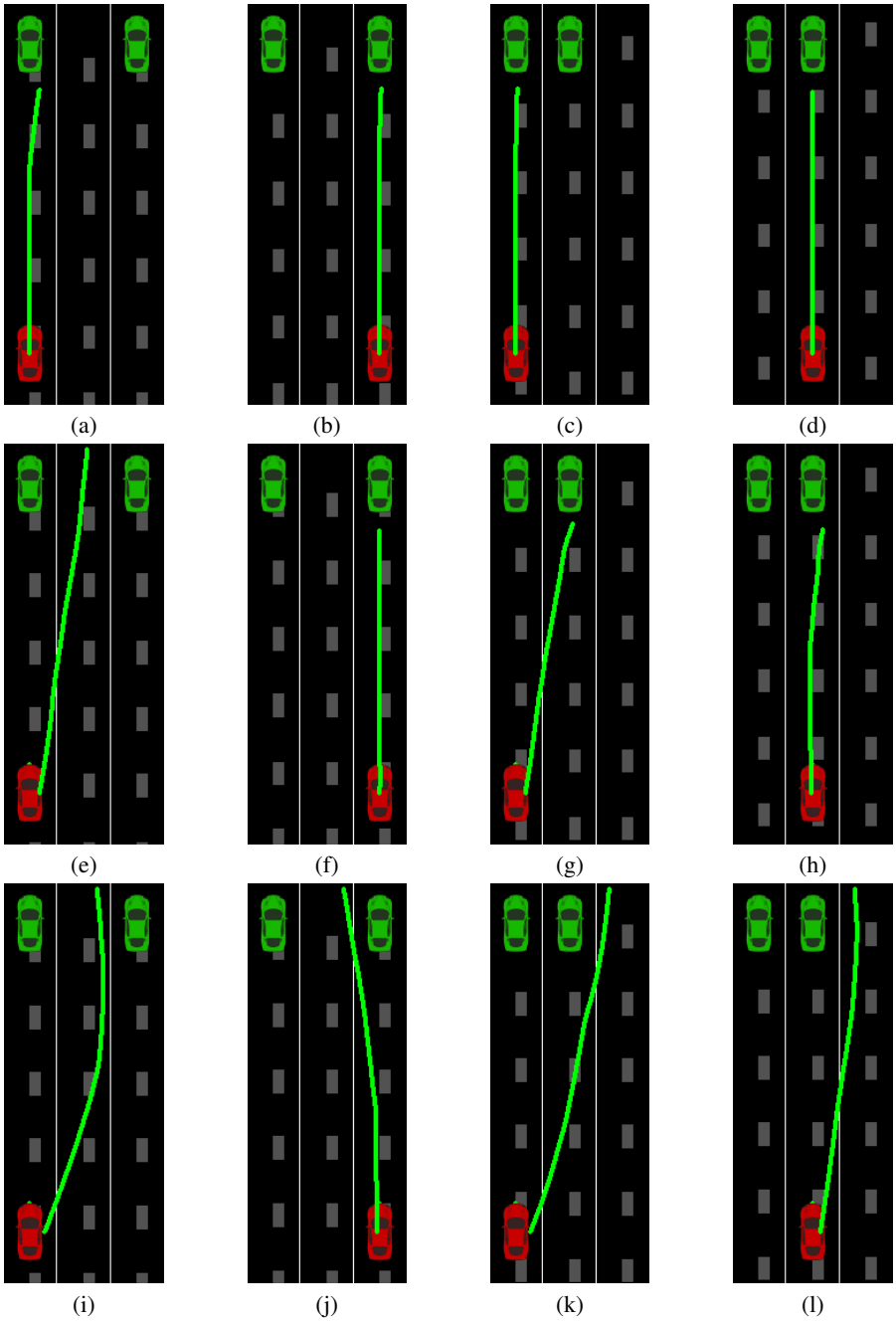

Figure 3: The agent's motion when facing four particular scenarios during training. The top row depicts the planned motion after the first IRL iteration (with 3000 DQN inner iterations), the middle row after four and the bottom row after 6 IRL iterations.

scenarios, acting according to the policy of the trained network. The mean difference over feature expectations in these scenarios was computed. Part of these values are presented in Table 1.

c. **Evaluating using classical motion planning objectives:** Classical motion planning methods are evaluated based on their performance in obstacle avoidance, jerk optimality, driving in the lanes, etc. In our experiments, similar to the expert, the agent avoided the walls and obstacles in 100% of the scenarios, while maintaining its position in the lane except during obstacle avoidance. Jerk values were also found to be close to the expert's. Figure 3 demonstrates the agent's planned motion based on the rewards extracted at different stages of the training phase.

		Sensor						
		0	1	2	3	4	5	6
$\lvert\hat{\mu}_E - \hat{\mu}_A\rvert$	1	0.000	0.000	0.000	0.004	0.158	0.005	0.011
	2	0.000	0.001	0.005	0.136	0.209	0.110	0.167
	3	0.001	0.004	0.006	0.173	0.079	0.126	0.085
	4	0.003	0.004	0.016	0.116	0.121	0.158	0.026
	5	0.002	0.016	0.094	0.096	0.004	0.039	0.030
	6	0.001	0.041	0.095	0.000	0.176	0.045	0.038
	7	0.001	0.039	0.057	0.046	0.017	0.000	0.000
	8	0.006	0.075	0.002	0.156	0.016	0.000	0.000

Table 1: The absolute differences between the trained and expert mean feature expectations. Each column refers to one of the first 7 sensors and each row refers to one of the first 8 distance bins.

4 Conclusion

In this paper we proposed using Deep Q-Networks as the refinement step in Inverse Reinforcement Learning approaches. This enabled us to extract the rewards in scenarios with large state spaces such as driving, given expert demonstrations. The aim of this work was to extend the general approach to IRL. Exploring more advanced methods like Maximum Entropy IRL and the support for nonlinear reward functions is currently under investigation.

References

[1] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1726–1743, 2013.

[2] R. Triebel, K. Arras, R. Alami, L. Beyer, S. Breuers, R. Chatila, M. Chetouani, D. Cremers, V. Evers, M. Fiore, H. Hung, O. A. I. Ramírez, M. Joosse, H. Khambhaita, T. Kucner, B. Leibe, A. J. Lilienthal, T. Linder, M. Lohse, M. Magnusson, B. Okal, L. Palmieri, U. Rafi, M. van Rooij, and L. Zhang, "Spencer: A socially aware service robot for passenger guidance and help in busy airports," in *Proc. Field and Service Robotics (FSR)*, 2015.

[3] B. Okal and K. O. Arras, "Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2889–2895.

[4] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 987–993.

[5] H. Tehrani, K. Muto, K. Yoneda, and S. Mita, "Evaluating human & computer for expressway lane changing," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 382–387.

[6] H. Tehrani, Q. H. Do, M. Egawa, K. Muto, K. Yoneda, and S. Mita, "General behavior and motion model for automated lane change," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 1154–1159.

[7] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[8] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, 2000, pp. 663–670.

[9] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[10] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," *Urbana*, vol. 51, no. 61801, pp. 1–4, 2007.

[11] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, 2008, pp. 1433–1438.

[12] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 729–736.

[13] M. Wulfmeier, P. Ondruska, and I. Posner, "Deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[17] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE transactions on automatic control*, vol. 42, no. 5, pp. 674–690, 1997.

[18] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.