Implementing deep IRL and GAIL based techniques in Donkeycar

Harshita Chaudhary
harshita@tamu.edu
UIN: 529005682

Minh Dang
minhdangtn@tamu.edu
UIN: 131005649

May 3, 2021

Demo Video:
https://youtu.be/NFqOEZVZxHI
Github:
https://github.com/harshita-chaudhary/gail-donkeycar

Abstract

A common technique used in autonomous driving is to imitate the emergent behavior of humans while practically using the sensor data to guide the learning process. This can be done using the following two algorithms. The first one is Inverse reinforcement learning(IRL). IRL flips the problem and instead attempts to extract the reward function from the observed behavior of an agent. Generative Adversarial Imitation Learning (GAIL) is another imitation learning algorithm that is model-free and performs well due to its capability to imitate complex behaviors. The use of these techniques results in safer driving models as expert demonstrations are used for this, and are thus very helpful in environments where safety is paramount.

1 Problem Statement

Our goal for this project is to implement an optimized IRL/GAIL algorithm that would provide Autonomous Driving for a car in a simulated environment. One of the bottlenecks with Autonomous Driving is that it requires a large amount of data from different driving scenarios to be able to learn the correct behavior in a certain situation. This is where IRL/GAIL comes into play. We map the hidden states of the car while driving as a Markov Decision Process (MDP) and apply IRL/GAIL with Q-Learning to plan the actions accordingly. We will try to implement either or both of the two reference papers:

1. Imitating Driver Behavior with Generative Adversarial Networks [7].

2. Learning to Drive using Inverse Reinforcement Learning and Deep Q-Networks [11].

2 Literature Review

2.1 Background

The autonomous driving problem requires a large amount of data to train and create an efficient model to control the actions of the car while driving. More than that, it is very difficult to define a reward function that could approximate the correct reward for each action corresponding to a state due to the large state space of the problem (Sharifzadeh et al.). Hence, using normal Reinforcement Learning methods could not be suffice for the autonomous driving problem. The driving behavior of humans has been modeled in several ways in the past, with rule-based methods and behavior cloning (BC) being some of the most traditional ones and using IRL as a generative adversarial network (GAN) being one of the more recent ones.

2.2 Inverse Reinforcement Learning

To address the state space problem, Sharifzadeh et al. suggests that we could combine Inverse Reinforcement Learning(IRL) with two different Deep Q-Network and let the model approximates the optimal reward function while training could drastically improve the efficiency and result of the model. By mapping the problem to a Markov Decision Process (MDP), then keeping each state-transition in a memory and randomly sample the training data from the memory to train one network and periodically switch the original network with its copy. This can lower the uncertainties and help the model converge and generalize the problem quicker. The idea behind using two different Deep Q-Network is create by Mnih et al.. This method achieves the desirable result compare to the expert behaviors and it will be the fundamental of our implementation.

Another difficulty in autonomous driving is that we would like the model to drive the car in a human-like manner, even if the surrounding environment could change rapidly. Wang et al. propose a method called Augmented Adversarial Inverse Reinforcement Learning (AIRL) to tackle this problem. They create a discriminator to identify the expert-behavior and a generator that could generate an expert-like behavior that will be fed in the discriminator for training. They set up a semantic reward, which is dependent on how efficiently the model controls the car, at the output of the discriminator network to help the model understand the policy of the reward. Only on relevant information around the cars is being input into the model such as nearby cars, lanes, etc. to reduce the amount of information that is needed for training. However, in our implementation the Donkey car GYM environment will not change drastically so we found this method could slow down the performance of our model.

Other approaches such as using Maximum Entropy and Semantic Category Observations prove to perform on a similar efficiency with the above methods. However, Maximum Entropy could not perform well in a large and dynamic state space compare to Semantic Category due to the fact that Maximum Entropy only using a linear function approximate for the reward function (Huang et al.). Semantic Category Observations on the other hand utilizes the multi-class occupancy map and a deep fully convolutional cost encoder to build the cost model. This helps the model performs wonderful on the CARLA autonomous driving simulator. (Wang et al.).

2.3 Generative Adversarial Imitation Learning

The traditional models do not learn the emergent behavior of humans while driving. The rule-based methods rely on fixed assumptions whereas the behavior cloning methods suffer from cascading errors problem, where the model leads to poor predictions when it encounters states that haven't been covered very well in the training data.

GAIL was proposed by Ho and Ermon(2016) [3]. In behavior cloning(BC), the cascading errors problem is very common as the model does not know what to do when it visits states that are not included in the training samples used for supervised learning. BC needs expert interaction if this problem needs to be overcome. On the other hand, even though the IRL addresses this problem of cascading errors and does not require continuous expert interaction with the environment, it is computationally very expensive which makes its use infeasible for domains that have high dimensions or features. This is mostly due to the additional cost of learning a cost function when what we really need is a behavioral policy that is similar to an expert human. In GAIL, a GAN is used to train a policy that aims to imitate expert human behavior by giving appropriate rewards for confusing the discriminator that compares the expert policy with the generated one. This approach has the advantage of being model-free and good sample efficiency, thus it works well for complex environments.

Kuefler et al.(2017) applied GAIL to autonomous driving to overcome the cascading errors problem and demonstrated that it performed better in learning from expert humans[6]. Neural networks were used to handle non-linearity and high dimensionality and the stochasticity in human behavior was cap-

Algorithm 1 Generative adversarial imitation learning

Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0

for $i = 0, 1, 2, \ldots$ **do**

Sample trajectories $\tau_i \sim \pi_{\theta_i}$

Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{17}$$

Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\begin{aligned} &\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta), \\ &\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \end{aligned} \tag{18}$$

end for

Figure 1: GAIL algorithm as described by Ho et al. [3]

tured by creating a Gaussian distribution over the output actions. The results were compared with the baseline model using the Root Weighted Square Error, Kullback-Leibler Divergence, and emergent behavior(lane change rate, the offroad duration, the collision rate, and the hard brake rate) metrics. It was shown that GAIL performed better and resulted in stable long-term trajectories, thus making it extremely useful in modeling human behavior in simulation software.

Another common approach in autonomous driving is Apprenticeship via inverse reinforcement learning(AIRP). The recovery of unknown reward functions can be computationally heavy, therefore in apprenticeship learning, the reward function is not recovered, and instead, a policy is derived using a model-free approach that is at least as good as expert policy. Ho et al. (2016) improved upon the policy gradient algorithm using direct policy optimization in their apprenticeship learning formalism of imitation learning[4]. Instead of designing the cost function classes which are used in apprenticeship learning, they focused on optimizing the policy gradients that make it useful in large high-dimensional and continuous environments.

Morton et al. studied the effectiveness of using Recurrent Neural Networks, specifically long short-term memory (LSTM) in vehicle acceleration predictions on highways [9]. The use of neural networks resulted in efficient function approximations and thus provided more realistic acceleration distributions due to their high expressiveness. They used two output distributions for the LSTM, which are the Gaussian mixture (GM) and the Piecewise Uniform (PM). LSTMs are resistant to vanishing gradient issues and thus result in more accurate models. The results showed that recurrent neural networks replicate the qualitative behavior of human driving very closely. Furthermore, the Gaussian mixture LSTM was shown to perform better than Piecewise Uniform LSTM in all the evaluation areas.

Wierstra et al. introduced Recurrent Policy Gradients(RPGs)[14], which involved a policy-free method for doing reinforcement learning on Partially Observable Markov Decision Problems (POMDPs). They too inherently used LSTMs to memorize past events. In RPGs, the input of the recurrent neural network is the observation and the target reward, and the output is the distribution over the action space. The backpropagation method was slightly different from other standard backpropagation ways as they used eligibility-backpropagation through time. Thus, RPGs can prove to be fruitful in driving tasks when we cannot capture all the

state information due to sensor errors or occlusions that prevent the driver from observing the complete driving state needed by the model.

3 Preliminaries

3.1 Markov Decision Process (MDP)

We define our problem as an MDP with the following definitions:

3.1.1 State Space

The state space of the MDP comprises of the image which that the donkey car is seeing, i.e. the state space consists of the RGB image being captured from the front facing camera on the simulated donkey car. The set of all these states is defined by S, and a state at any given time-step t is denoted as s_t. Since, the state space is infinitely large we use image preprocessing to generalize to a large set of states.

3.1.2 Action Space

The action space is a 2 dimensional vector of continuous type which consists

- **steer value** which describes the direction in which the car is to be steered as a value between -1 to 1 (left to right) with0 being the center.
- **throttle value** which describes the amount by which the car accelerates (basically simulation of gas pedal), as a value between -1 and 1, 1 being full throttle forward and -1 being full throttle reverse.

3.1.3 Reward Function

As this is an imitation learning problem, there is no reward or discount factor.

3.2 Simulator

3.2.1 Donkeycar

Donkey is a Python library for self driving cars and donkeycar simulator is a self driving simulator built over the Unity platform. It has APIs to control the car using a web interface and methods such as joystick, gamepad, and device tilt. Donkeycar also has a gym wrapper that makes it easy to integrate with common RL algorithm libraries. One can control the donkey car using these APIs and generate expert trajectories that can be used for algorithms like behavior cloning and GAIL.

The donkey car Gym wrapper has Box2D state space which consists of images of size (120, 160, 3) from the front camera of the car. When the simulator is manually used to generate the expert trajectory, the simulator generates these images at a fixed frequency(20Hz) and records the throttle and angle values in a catalog file.

Donkeycar has configurable sensors, such as the camera type can be changed depending on the use case. Also, the simulator records multiple telemetry parameters such gyro, acceleration, car position, and cross track error(cte) along with the two action parameters, which are the throttle and steering values.

Due to the simplicity of the donkey simulator and ease of integrating it with the Open AI Gym environment and stable baselines, it was selected for our experimentation and project. Also, donkeycar is more lightweight than Carla making it easier to run in machines with low configuration.

3.3 Stable Baselines

For the base code, we use the GAIL algorithm from the Stable Baseline Github repository. Stable Baselines is a collection of Reinforcement Learning Algorithms based on OpenAI Baselines with improvement in usability and readability.

Besides the ease of implementation with the code-base, Stable Baselines models can be used to generate expert-behavior trajectory to feed into the training network to train. This can help save time in collecting real world data. However. the original code is still lacking in working with captured images. Hence, we add methods to enable the model to learn with captured image as observation space using Lane Segmentation (we will discuss this in the upcoming section).

4 Experiments

4.1 First Approach

4.1.1 Idea

GAIL involves using expert trajectories in order to obtain a cost function before learning a policy. Therefore, the first approach for our project involved focusing on generating this expert trajectory. We used Proximal Policy Optimization(PPO) algorithm to do do so. We also tried other algorithms such as A2C and TRPO with MLP and CNN policies, but PPO resulted in the best model. Therefore, we used PPO algorithm of the stable baselines repository [10] to generate 100 episodes of expert trajectory with 1 million time-steps.

The expert trajectory was fed into the GAIL model. The GAN's discriminator is used to distinguish the expert trajectories of the generated learned policy from the expert trajectory. However, the results weren't good as the expert trajectory learnt using PPO wasn't perfect that resulted in the GAIL model to learn from not so expert trajectories, thus leading to zigzag behavior while moving forward.

To overcome this problem, we tried to generate the expert trajectory manually by running the car in the Donkey simulator using joystick. This resulted in better expert trajectory as the car wiggles less when we drive it manually. Though this method is not perfect, it is slightly better than the earlier one.

4.2 Second Approach

4.2.1 Idea

The second approach involved generating expert trajectory manually using a joystick to drive the car. This resulted in a more stable expert trajectory as it overcame the problem of car wiggling while moving forward. The expert trajectory includes the observation space, which is the image and the action space, which is the Throttle and Angle of steering at each state. These values are stored as numpy arrays in a .npz file. The image path are stored instead of images themselves as it takes lesser space than the images. Various different hyperparameters were tuned while running the GAIL Model which can be found in table 1.

Table 1: Different hyper-parameters were tried for running the GAIL model:

Hyper-parameters	Values
Policies	MLP policy, CNN policy
Time-steps	50000, 100000, 1000000
Hidden size for MLP	50, 100, 500
Different sizes of expert data-set	1000, 2000, 3000 images
Activation function for hidden layert	tanh, relu

Adversary model: The adversary model is comprised of three convolutional layers followed by fully connected layers to compute features from the stacked image input. The image features are then concatenated with the actions to form a transition. This transition is passed to three fully connected layers. Figure 3 shows the model architecture for the adversary.

Simulator settings:

- start_delay 5.0
- max_cte 5.0
- frame_skip 2
- cam_resolution (120, 160, 3)

4.2.2 Image preprocessing

Since we are using the images taken directly from the car camera, we will need to extract the feature from the image to make the learning process become more meaningful. Following the article [1], we apply a technique called Lane Segmentation to achieve a desirable output.

The advantage of this image preprocessing is that it helps us to reduce the noisy background from of the image and just focus on extracting the lane which the car is driving on. This Lane Segmentation approach utilizes the Canny Edge Detector to extract all the edges, Hough Line Transform to find the straight lines and then compute the slope of each lines to identify the left and right lane [15].

After the preprocessing, we will stack the 4 frames together and randomly selected them as the input for the model. We apply Frame Skipping here to help the model generalize the action of a state in a 4-frame time step [2]. The

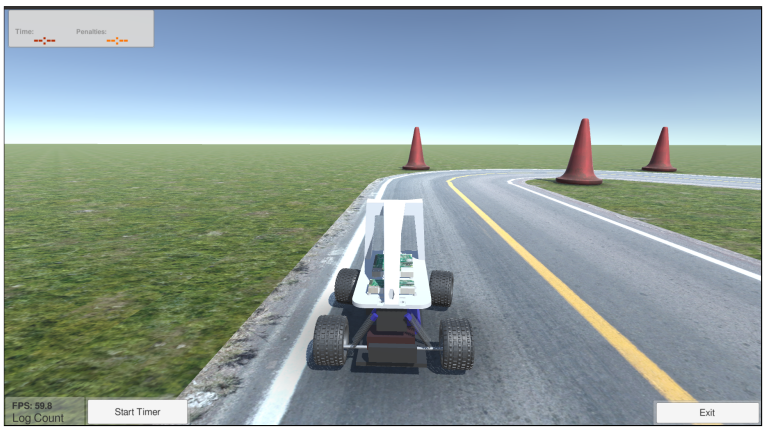


Figure 2: Donkeycar learning to drive by sampling in the donkey simulator

```
activ = tf.nn.relu
layer_1 = activ(conv(obs, 'c1', n_filters=32, filter_size=8, stride=4, init_scale=np.sqrt(2)))
layer_2 = activ(conv(layer_1, 'c2', n_filters=64, filter_size=4, stride=2, init_scale=np.sqrt(2)))
layer_3 = activ(conv(layer_2, 'c3', n_filters=64, filter_size=3, stride=1, init_scale=np.sqrt(2)))
layer_3 = conv_to_fc(layer_3)
_input = tf.concat([layer_3, actions_ph], axis=1)  # concatenate the two input -> form a transition
p_h1 = tf.contrib.layers.fully_connected(_input, self.hidden_size, activation_fn=tf.nn.tanh)
p_h2 = tf.contrib.layers.fully_connected(p_h1, self.hidden_size, activation_fn=tf.nn.tanh)
logits = tf.contrib.layers.fully_connected(p_h2, 1, activation_fn=tf.identity)
```

Figure 3: Adversary Model Architechture

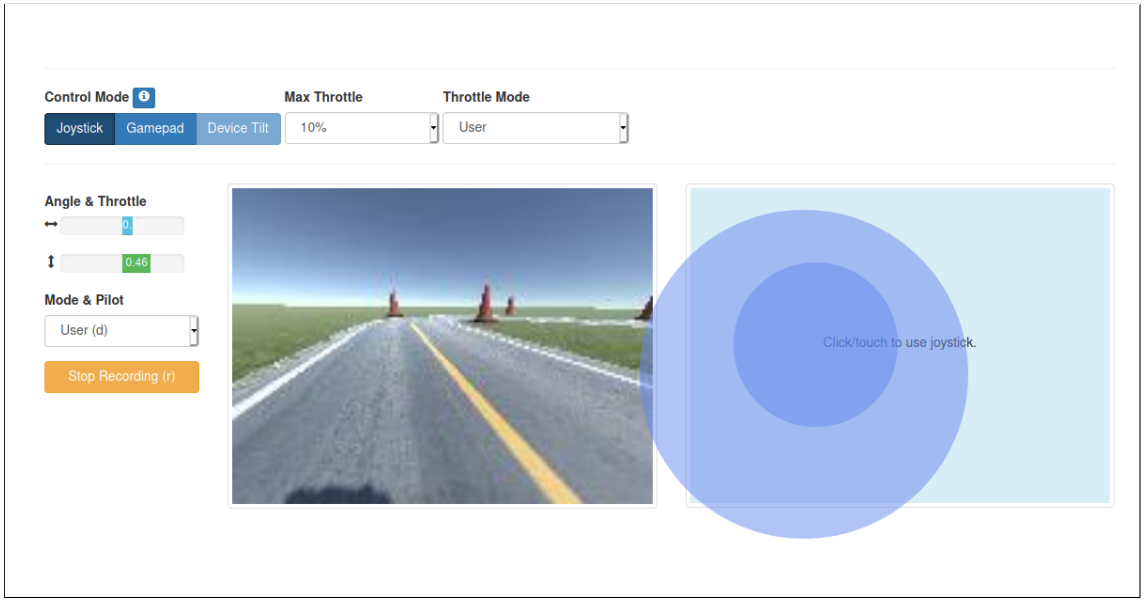


Figure 4: Generating expert trajectory by driving the car manually in the donkeycar simulator

Table 2: Observed metrics after iteration 200:

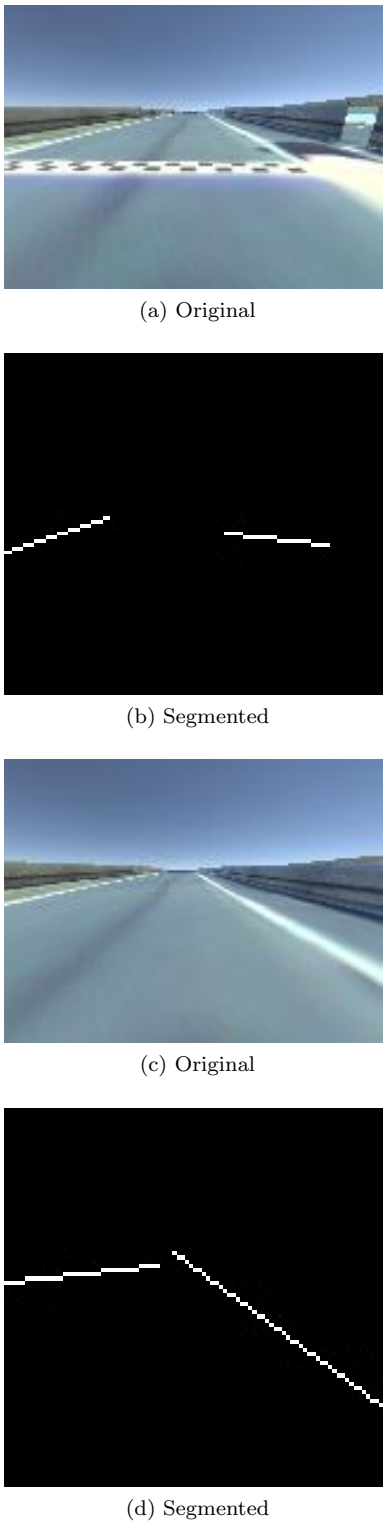
Metric	Values
generator_loss	0.96885
expert_loss	0.30522
entropy	0.38185
entropy_loss	-0.00038
generator_acc	0.58789
expert_acc	0.83398

intermediate outputs of preprocessing can be seen in 5.

4.2.3 Result

The results obtained at the end of iteration 16 can be found in the table 2. The training hyperparameters include

- step size 5.0
- batch size 5.0
- policy CNN
- total timesteps 1M



(a) Original

(b) Segmented

(c) Original

(d) Segmented

Figure 5: Image Preprocessing

5 Limitations

5.1 Inaccuracies in expert data

Since the model trains by making the discriminator compare the expert policy with the policy generated by the model, the performance highly depends on the quality of the expert data. In the first approach, this trajectory was generated using an expert model. However, it resulted in wiggling movements while driving. In the second approach, the expert data is generated manually using the web interface of Donkeycar. Since it is difficult to control the car accurately an in a stable manner using the joystick in the web interface, the expert trajectory created using this process is not perfect. We believe that a controller or physical steering wheel would be more sensitive and provide more accurate behavior of human drivers than controlling the simulation using a mouse. We think it is a major limitation in our project as even after lots of practice in using the simulator, the path followed is not as accurate as can

be achieved using other devices to control due to the sensitivity and controlability limitations.

5.2 Insufficient training

Training a model that consumes images requires high computational power and iterations to train. We believe that due to limited time and resources, the model could not be trained a good enough number of iterations for it to perform well.

5.3 Driving slowly

While training the model, we observed that as the iterations increase, the car learns to drive at a low speed than optimum. This is a limitation that could be arising due to the expert trajectory not having continuous throttle or the car not moving ahead to avoid collision and high negative rewards, which needs further investigation.

6 Conclusion

In this project, we trained the Generative Adversarial Imitation Learning algorithm for the problem of lane following in autonomous driving. Several experiments were done to obtain the expert trajectory to be given as an input to the GAIL model, and hyperparameters were tuned in order to learn a good GAIL model. However, due to time and resource limitations, the results were not as we expected. We had begun with an initial plan of using the Carla simulator but due to integration issues, we switched to Donkeycar which was easier to integrate in our experiment.

We believe our image preprocessing is still not accurate for some of the images captured from the camera. Some images are not segmented correctly due to the noisy background which can cause the algorithm to detect the wrong lines in the images.

7 Future Work

Learning from our limitations, we definitely could improve our work by focusing on these points:

1. Instead of using the simulator, training on a real life donkey car with human input could increase the correctness and fluid of the input.

2. Capturing the expert trajectory using a steering wheel as it is more sensitive and closer to human driving behavior.

3. Improving the preprocessing module by capturing features extracted using CNN networks and comparing it with the features extracted using the current method of image segmentation and stacking.

4. Fine tuning different hyper parameters within the GAIL model to get a higher accuracy.

5. Training the model for longer episodes and epochs to fit the model better.

References

[1] Train donkey car in unity simulator with reinforcement learning. https://flyyufelix.github.io/2018/09/11/donkey-rl-simulation.html. Accessed: 2021-04-30.

[2] Frame skipping and pre-processing for deep q-networks on atari 2600 games. https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/. Accessed: 2021-04-30.

[3] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[4] Jonathan Ho, Jayesh K. Gupta, and Stefano Ermon. Model-free imitation learning with policy optimization, 2016.

[5] Zhiyu Huang, Jingda Wu, and Chen Lv. Driving behavior modeling using naturalistic human driving data with inverse reinforcement learning, 2021.

[6] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. 01 2017.

[7] Alex Kuefler, Jeremy Morton, Tim Allan Wheeler, and Mykel John Kochenderfer. Imitating driver behavior with generative adversarial networks. *CoRR*, abs/1701.06699, 2017. URL http://arxiv.org/abs/1701.06699.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

[9] J. Morton, T. A. Wheeler, and M. J. Kochenderfer. Analysis of recurrent neural networks for probabilistic modeling of driver behavior. *IEEE Transactions on Intelligent Transportation Systems*, 18(5): 1289–1298, 2017. doi: 10.1109/TITS.2016.2603007.

[10] OpenAI. Openai-baselines. https://github.com/hill-a/stable-baselines.

[11] Sahand Sharifzadeh, Ioannis Chiotellis, Rudolph Triebel, and Daniel Cremers. Learning to drive using inverse reinforcement learning and deep q-networks, 2017.

[12] Pin Wang, Dapeng Liu, Jiayu Chen, and Ching-Yao Chan. Adversarial inverse reinforcement learning for decision making in autonomous driving. *CoRR*, abs/1911.08044, 2019. URL http://arxiv.org/abs/1911.08044.

[13] Tianyu Wang, Vikas Dhiman, and Nikolay Atanasov. Inverse reinforcement learning for autonomous navigation via differentiable semantic mapping and planning, 2021.

[14] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5):620–634, 09 2009. ISSN 1367-0751. doi: 10.1093/jigpal/jzp049.

[15] Felix Yu. Donkey car. https://github.com/flyyufelix/donkey_rl.