Imitating Driving Behavior in an Urban Environment

Malik Boudiaf
Department of Aeronautics and Astronautics
Stanford University
mboudiaf@stanford.edu

Ianis Bougdal-Lambert
Department of Aeronautics and Astronautics
Stanford University
ianisbl@stanford.edu

Abstract

To be deployed safely, autonomous vehicles need to adapt to a wide range of situations, even those that are rarely encountered by human drivers, if ever. In this paper we apply Behavioral Cloning to teach a car to imitate human driving behavior in a simulation environment. We compare its performance to GAIL – a novel algorithm of generative adversarial imitation learning – and show that GAIL performs better, with encouraging results even in an urban setting.

1 Introduction

Accurately modeling human driving behavior is critical for realistic simulation of driving scenarios. Addressing this challenge has the potential to significantly advance research in automotive safety. Initial approaches in autonomous driving tried to imitate human drivers' behavior by describing it as a set of fixed rules to follow. These early, rule-based approaches tried to fit a parametric model to real driving data using strong assumptions on road conditions and human behavior [1]. As a consequence, they failed to generalize well to new driving scenarios or situations that are rarely encountered in real life. It is clear that robust, safe autonomous vehicles cannot be deployed in a complex environment using a simple rule-based model, hence the need for more advanced approaches.

2 Related Work

Recent approaches are more data driven. The first approach is Imitation Learning (IL). Expert data is provided by human demonstrations and algorithms try to learn a policy that would optimally reproduce human behavior in all possible states. Within that framework, algorithms typically fall into one of two categories: Behavioral Cloning (BC) or Reinforcement Learning (RL).

Behavior Cloning BC algorithms directly map observed states to actions by fitting the expert data [2]. Just like early approaches, they suffer from a limited capacity of generalization. Because the training set contains very few extreme situations (close to collision, front vehicle suddenly braking, vehicle drifting towards the edge of the lane...), the policy learned will perform poorly when tested on these cases, resulting in a well-documented phenomenon of cascading errors [3].

Reinforcement Learning Given a known reward function $r(s, a)$ defined for every state/action pair, the RL approach tries to learn the expert policy that would maximize that reward. Because the reward function is hard to model, Inverse Reinforcement Learning (IRL) is often preferred [4]. IRL assumes that the expert follows an optimal policy with respect to an unknown reward function that the algorithm tries to recover. Once it is recovered, classic RL can be used to find the target policy.

Generative Adversarial Imitation Learning Recovering the reward function in Inverse Reinforcement Learning can be computationally expensive, and recent efforts have tried to bypass this step and directly optimize the target policy. Generative Adversarial Imitation Learning (GAIL) [5] is

one of these approach. In GAIL, a policy generator is trained alongside a discriminator in a GAN fashion. Given a state, the generator tries to output the optimal action, while the discriminator tries to distinguish between real and generated state/action histories.

Scope of this project In [6], GAIL was shown to perform significantly better than other state-of-the art algorithms on a highway dataset. But highways are relatively controlled environments and it still needed to be shown whether or not GAIL could adapt to a more complex setting. To do so, we extend the work done in [6] by applying GAIL and BC to a more urban dataset. We implement 3 architectures of behavioral cloning and compare their performance with GAIL on the urban dataset. The code framework represents months of work by itself and was therefore forked from the SISL's github repository [7] for most part. This project is shared with the CS236 class on Deep Generative Models. Although most of the infrastructure and data were shared, the CS230 part of the project consisted in implementing and testing the different models of Behavioral Cloning, while for the CS236 part we focused on the GAIL architectures. For that reason, we will focus on the BC approaches in this report. For more details on GAIL architectures, please refer to the project report for CS236.

3 Dataset and Features

3.1 Notations

We regard driving as a sequential decision making task in which the driver obeys a policy $\pi(a_t|s_t)$ mapping a current state of the car and its surroundings, denoted as s_t, to an action a_t. In the BC approach, we try to learn a policy π_θ parametrized by θ that best imitates human driving behavior.

States/Actions The state s of a vehicle is represented by a 64-dimensional vector. We can separate this vector into a set of core features, a set of sensor features, and a set of boolean features. Core features include the speed of the vehicle, its dimensions, its position within the lane... Sensor features consist of range data extracted from lidar beams emitted from the car in the simulation environment, while boolean features indicate when a dangerous situation is encountered (collision, driving in reverse, off-road...). Actions a are modeled by a 2-dimensional vector representing the forward acceleration as well as the turn rate.

3.2 Data Generation

The data was extracted from the NGSIM database [8], which compiles real vehicle trajectories over periods of 30 minutes. The original GAIL paper was tested on two highways in Los Angeles. For our project, we chose a section of Lankershim Boulevard, also located in LA. Contrary to the highways, our dataset includes intersections and traffic lights, with vehicles often stopping, changing lanes or entering/exiting the road. The data had to be formatted to be used in the GAIL framework. We extracted the roadway's centerlines using AutoCAD and fed them along with the raw car data to the NGSIM.jl package [9], that generates the final roadway data and the cleaned states/actions.

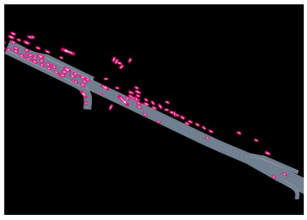

Figure 1: Section of Lankershim Boulevard with vehicles superimposed.

3.3 Simulation Environment

In order to generate artificial trajectories, we need to roll-out the actions output by the policy to get the corresponding next states. We use the rllab simulation environment, part of the ngsim_env package [10] developed by the SISL. It propagates the environment – ego vehicle + surrounding vehicles – at every time step.

4 Methods

4.1 Representing a Policy

The policy π_θ takes as input a state s_t and outputs an action a_t. To account for the complexity of a human driver's policy, it is modeled as a neural network that takes s_t as input. Depending on the BC models implemented, the network either directly outputs an action a_t or a mean μ_t and covariance matrix Σ_t, the action then being sampled from a Gaussian distribution: $a_t \sim \mathcal{N}(\mu_t, \Sigma_t)$.

4.2 Training

Behavioral Cloning models were trained using Batch Gradient Descent. Each batch consists of a number of state/action pairs. We chose the negative log-likelihood of the outputted action as the loss function:

$$L(\hat{a_t}, a_t) = -\log p(\hat{a_t}|a_t) = -\log[\mathcal{N}(\hat{a_t}|a_t, \Sigma_t)]$$

GAIL models include a discriminator that tries to distinguish real from generated trajectories. The policy and discriminator are trained simultaneously in a GAN fashion. See the CS236 report for more details on the training process.

5 Experiments and Results

5.1 Models implemented

The Behavioral Cloning (BC) baseline consists of a deterministic and direct mapping from state s_t to action a_t. It features 3 hidden layers of decreasing size. The model is fit directly to the expert data in a supervised way and not in a GAN fashion, hence the absence of discriminator. The **larger BC model** is a deeper version of the baseline and contains 2 additional layers.

The Static Gaussian (SG) model features two separate feed-forward networks for μ and Σ, with only two hidden layers of 32 units each. It is trained in a similar way as the BC models.

The GAIL models, finally, share a simlar policy architecture as the SG model. GAIL-MLP's policy is non-recurrent, while GAIL-GRU features an extra GRU layer of hidden size 64 for recurrence. The discriminator is also modeled by a 3-layers feedforward network.

Model	π_θ		D_ψ
	μ_θ	Σ_θ	
Baseline BC-MLP small	*(128,128,64)*		
BC-MLP (5 layers)	(256,128,64,64,32)		
Static Gaussian	(32,32)	(32,32)	
GAIL-MLP	(128,128,64)	(128,64)	(128,128,64)
GAIL-GRU	(128,128,64) + (64)	(128,64) + (64)	(128,128,64)

Figure 2: Architectures implemented. GAIL models include a discriminator and are trained in a GAN fashion.

5.2 Training Parameters

All experiments were performed on a Google Cloud instance with a 10.5 Go memory and 4 CPUs. For the BC models, we used a train/dev split of $90/10$, with mini-batches of 64 trajectories of 10s each. The training was performed for 50 epochs using the Adam Optimizer with a decaying learning rate ($\alpha_0 = 0.004, \gamma = 0.5$) and gradient clipping ($c = 5$). The GAIL models were trained for 1000 epochs, with batches of 50 trajectories. Training a BC model took a few hours, while a single GAIL model took about 3 days to train.

5.3 Evaluation Metrics

Root Mean Squared Error (RMSE) captures the deviation over time of a trajectory generated by our policy from the true trajectory. Let m be the number of expert trajectories extracted for testing. For each expert trajectory i, we generate n corresponding trajectories j starting as the same state, using our trained policy. Ideally, each of the n generated trajectories should remain relatively close to the corresponding, real trajectory i. The RMSE of a metric v at time t is computed as follows:

$$RMSE(t) = \sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(v_t^{(i)} - \hat{v_t}^{(i,j)})^2}$$

KL divergence. RMSE might not perfectly reflect the performance of the policy. If our policy chooses to change lanes after some time while the real vehicle doesn't, this can result in a bigger RMSE, even if the generated trajectory looks very realistic. KL divergence addresses this issue by measuring the distance between the real-world distribution of a metric and the generated distribution of the same metric over all trajectories. It is approximated by sampling values from a given metric both from the expert data and from the generated trajectories. A piece-wise uniform distribution is then formed using B equally spaced bins. The estimated KL divergence is then computed:

$$D_{KL}(p_{data}||p_{model}) = \sum_{b=1}^{B} p_{b,data} log(\frac{p_{b,data}}{p_{b,model}})$$

5.4 Results

5.4.1 Validation

To test the performance of our trained policies, we sampled 50 expert trajectories. For each of them, we generated 200 trajectories starting at the same initial state. For each model, generating these 1000 trajectories took about 1 hour. We present below the Root Mean Square Error of the position and velocity, as well as the KL divergence of the velocity and turn-rate.

5.4.2 Root Mean Squared Error

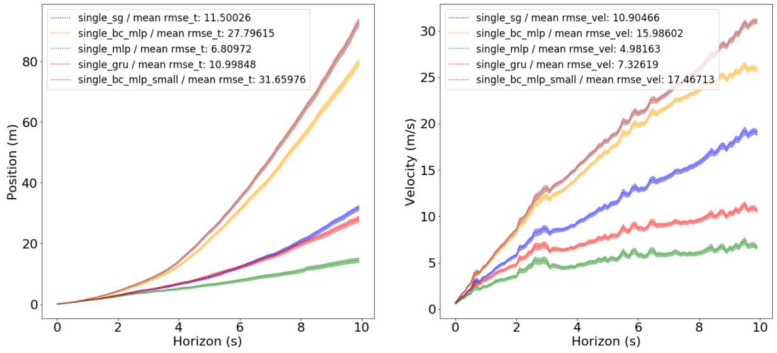

Figure 3: Root Mean Squared Error of position and velocity for the different models tested.

We can see that the two GAIL models – in green and red – perform much better than the BC approaches on position and speed. Directly fitting a policy to the data results in trajectories that quickly diverge from the real ones after 10s. Note that the Static Gaussian performs significantly better than the other two BC models. This could be accounted for by the greater flexibility of the architecture, that allows for less overfitting.

5.4.3 KL divergence

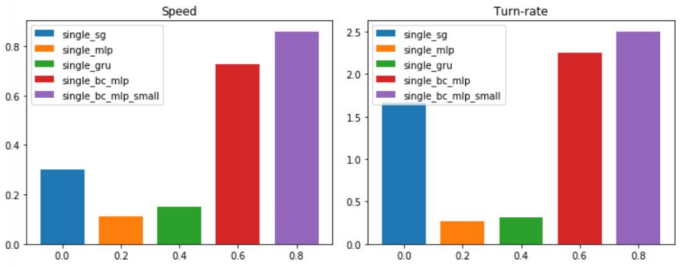

Figure 4: KL divergence for the position and velocity for the different models.

Once again, we see that the models trained with GAIL – in orange and green – clearly outperform the BC models. Training with GAIL results in trajectories that look much more "human". The SG model once again outputs trajectories that are somehow more realistic, especially when it comes to the distribution of speed.

5.5 Analysis

Overall, these results match our expectations. First, models trained with GAIL clearly outperform simpler models, both on RMSE and on KL divergence. They result in smoother trajectories with characteristics that are closer to human ones. If we compare with the graphs presented in [6], we see that GAIL's performance on highways is hardly better than on our urban dataset. On a highway environment, GAIL's error on position increased to about 10m after 10s, which is about the same as the final error on position for our best-performing policy. The KL divergence of speed and turn-rate on highways was around 0.1 and 0.5, respectively. We show similar values on speed, and even better ones on turn-rate. This is very encouraging and shows that GAIL is very robust to different environments.

6 Conclusion

We tested BC and GAIL architectures on a new urban dataset. We showed that policies trained with GAIL are much better at generating human-like trajectories than policies trained in a supervised learning fashion. GAIL seems to reduce the issue of cascading errors that can clearly be seen on BC models, with trajectories that show a significantly smaller divergence over time.

Policies have only been tested on 10 seconds trajectories. It would be interesting to validate them on longer trajectories to see if GAIL could be implemented in a real environment. While the RMSE seems to show that even GAIL-trained policies diverge after a short time, we saw that KL-divergence is a more interesting metric that reflects how close generated trajectories are to real ones overall. Computing the KL divergence on other metrics such as collision rate or off-road duration would be particularly helpful in determining whether the policies have the potential to be implemented in real life. There is also some additional work to be done on visualizing trajectories for validation, and on clearly separating training and development/test sets, a work that hasn't been done yet in the ngsim_env source code and that we unfortunately did not have time to focus on for the scope of this project.

Finally, we showed that GAIL's performance on an urban roadway can be similar to its performance on a highway. This is very encouraging given that urban contexts are much more complex. Additional work could also be done on adapting GAIL to this specific kind of environment to try and improve its performance further. We would also like to test the newly-developed multiagent version of GAIL. In this framework, multiple ego-vehicles are simulated in parallel and interact with one another. This is much closer to what would happen in real life where the actions of the ego-vehicle have an influence on surrounding agents.

7 Contributions

The code for our project can be found on the following github repository: https://github.com/mboudiaf/imitation-driving

For this project, Malik Boudiaf focused on coding, training and testing the Behavioral Cloning architectures, as well as on updating most of the source code. Ianis Bougdal-Lambert focused on generating the data and on training and testing the GAIL architectures.

References

[1] Gipps, P.G. (1981) A behavioral car-following model for computer simulation. *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp.105-111.

[2] Bojarski, M. & Del Testa, D. & Dworakowski, D. & Firner, B. & Flepp, B. & Goyal, P. & Jackel, L. & Monfort, M. & Muller, U. & Zhang, X. & Zhao, J. (2016) End to end learning for sel-driving cars. *IEEE Computer Vision and Pattern Recognition (CVPR)*.

[3] Ross, S. & Bagnell, J. (2010) Efficient reductions for imitation learning. *International Conference on Artificial Intelligence and Statistics*.

[4] Abbeel, P. & Ng, A.Y. (2004) Apprenticeship learning via inverse reinforcement learning. *International Conference on Machine Learning (ICML)*.

[5] Ho, J. & Ermon, S. (2016) Generative adversarial imitation learning. *Advances in Neural Information Processing Systems (NIPS)*.

[6] Kuefler, A. & Morton, J. & Wheeler, T. & Kochenderfer M.J. (2017) Imitating Driver Behavior with Generative Adversarial Networks. *IEEE Intelligent Vehicles Symposium (IV)*.

[7] Stanford Intelligent Systems Laboratory (SISL) github repository. *https://github.com/sisl*.

[8] Next Generation Simulation (NGSIM). Department of Transportation, Federal Highway Administration. *https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm*.

[9] Ngsim.jl. Stanford Intelligent Systems Laboratory. *https://github.com/sisl/NGSIM.jl*.

[10] Ngsim_env.jl. Stanford Intelligent Systems Laboratory. *https://github.com/sisl/ngsim_env*.