DeepRLDrive - Deep Reinforcement Learning for interpretable end-end autonomous driving

Amirtha Varshini A S
College of Computing
Georgia Institute of Technology
Atlanta, The United States of America
asindhanai3@gatech.edu

Vyshnavi Gutta
College of Computing
Georgia Institute of Technology
Atlanta, The United States of America
vgutta7@gatech.edu

***Abstract*—Existing works in Deep Learning for autonomous driving are modularized and lack generalizability in a continuous world setting. However, with deep Reinforcement Learning (RL), we can learn a control policy that is adaptive, scalable, and end to end. In this paper, we propose a deep reinforcement learning method based on the best performing model-free RL algorithm (Truncated Mixture of Continuous Distributional Quantile Critics (TQC)) with a Variational Autoencoder (VAE) to compress the input state representation. We further incorporate experience replay techniques to improve navigation of the model car in an autonomous driving simulator - Donkeycar. The performance of our method is much better than many baselines including Deep Q Learning (DQN), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC). Also, we have trained an external autoencoder module to provide interpretability to the learning process by generating a semantic bird-eye view mask.**

Keywords: Deep Reinforcement Learning, Autonomous Driving, End-to-end driving, Autoencoder, Interpretability.

I. Introduction

End-to-end autonomous driving (AD) [1] is a subfield of AD wherein the driving policies are learned without hand-engineered involvement by integrating the perception-planning-navigation pipeline. This method is also generalizable and scalable to the ever-complex needs of real-time driving. Reinforcement learning (RL) based AD is a popular choice for realizing this [3]. However, existing end-to-end RL methods are challenged by three main shortcomings:

1) Requires heavy computing from relying on complex networks due to high dimensional inputs
2) Lack of smooth driving policy and poor Average returns due to improper reward shaping
3) Lack of interpretability in how the agent understands the environment

To learn the end-end driving policy, a 2D Donkeycar simulator with input as the monocular camera image, the observed vehicle speed, and steering angle is used. [2] As the RL algorithm, we choose Truncated Quantile Critics (TQC). This provides the best performance compared to other algorithms like DQN, DDPG, and SAC.

To address the first shortcoming, this paper proposes to use a Variational Autoencoder (VAE) that generates a compressed latent space representation of input and improves rewards. For the second issue of smooth control, which is lack of continuity in the steering, various approaches are combined - conditioning on the reward function, constraining the change in steering angle while wrapping the input with the history of previous steering and throttle commands. For interpretability, a semantic segmentation mask is generated using another pretrained autoencoder.

We further propose an approach using experience replays to sample effectively from previous inputs, be tolerant of missing/faulty episodes and improve performance. Our experiments demonstrate the benefits of leveraging Prioritized Experience Replay (PER) and Hindsight Experience Replay (HER).

In addition, the first and third tasks could be trained jointly in a single model, according to the paper [4] by combining maximum entropy RL with sequential latent variables to model the environment. This approach generates a semantic bird view mask for explaining the algorithm's decision and also encodes the complex urban driving environment into a lesser dimension. However, [4] uses a CARLA simulator which is computationally expensive to train. Due to the absence of ground truth semantic segmentation masks in other low-cost simulators like Donkeycar, we will use a pretrained encoder network to visualize the input image representation.

II. Related Work

In regards to Reinforcement learning-based AD, Wolf et al. [6] use DQN to learn to steer an autonomous car and maintain track in simulation. Lillicrap et al. [11] demonstrated the first application of deep RL to real-world autonomous driving.

Experience replay [7] allows for online reinforcement learning (RL) agents to re-use past experiences. It is used to break correlations between weight updates and to avoid forgetting experiences. We hope to jointly leverage PER with MaxEntRL for boosting the performance quality in AD.

Another problem with MOST learning-based approaches for autonomous driving is that we can't visualize their understanding of the environment. However, Kim et al. [8] used a visual attention model with a causal filter to visualize the attention heatmap. [4] integrates sequential latent model learning and reinforcement learning using PGM for problem formulation, showing great potential in end-to-end learning of deep policies with high-dimensional inputs.

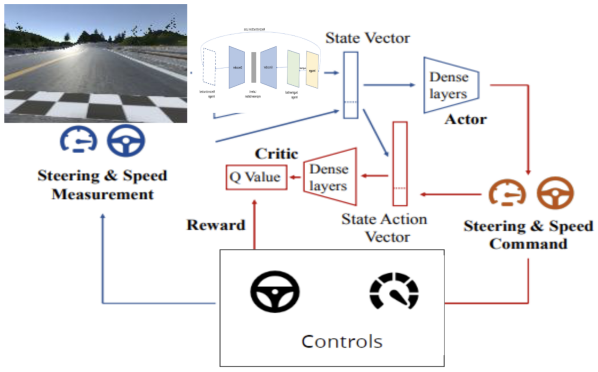

Fig. 1. Model architecture

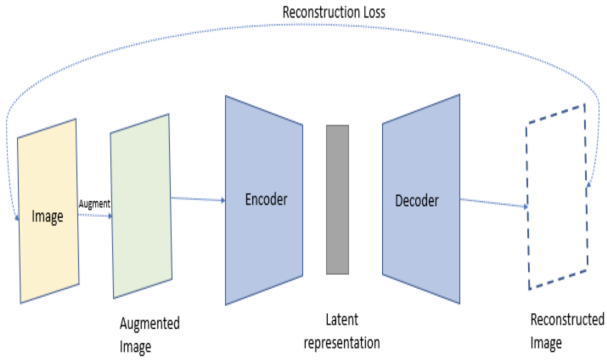

Fig. 2. VAE architecture

The field of autonomous driving is rich with complex problems, attracting state-of-the-art research. Simulators like CARLA while open-source, demand huge computational resources. So there is a need for training models on simpler and low-cost simulators such as Donkeycar Simulator [9].

III. METHOD

Our main objective is to incorporate experience replay with model-free RL algorithms like Soft-actor critic (SAC) and Truncated Quantile Critics (TQC) for navigation of a model car in autonomous driving simulators. For improved performance, we utilize sequential latent space models which reduce the dimensionality of the state space making the solution more scalable. The problem of interpretability would also be solved as latent space models facilitate the semantic bird-eye view mask generation.

Figure 1. shows the architecture of the algorithm Truncated Quantile Critics (TQC). TQC builds on SAC, TD3, and QR-DQN, making use of quantile regression to predict a distribution for the value function (instead of a mean value).

For training the policy, the images are first encoded using a feature extractor such as Variational autoencoder (VAE) (Figure 2.) and concatenated with a history of the last ten actions taken (throttle and steering). The control policy is represented by a neural network that learns to output steering angle and throttle. Training VAE after each episode leads to instabilities in the policy training since the features are changing over time. Furthermore, training without GPUs is time-consuming, so we would like to avoid retraining. Hence, the policy is trained with a fixed feature extractor to compress the image to a lower-dimensional space. Using VAEs also facilitates decoupling feature extraction from policy learning.

We were not able to train the interpretability module along with the RL module because most of the existing 2D car simulators do not generate the semantic segmentation ground truth. Hence, we have used an external network U-Net [11] trained on the Comma10k dataset which offers pretrained semantic segmentation.

IV. SIMULATORS

A. 2D simulators

We have considered 2 different yet simplistic simulator environments under 2D,

1) *Parking V0*
   In this environment, the goal is to make sure the ego vehicle is parked in the appropriate space.
2) *Carracing v0*
   In this environment, the goal is to maneuver the car along the track effectively.

B. Donkey simulator

Donkey [9] is a Self Driving Car Platform for hobby remote control cars. The goal is to make the car go as fast as possible while staying on a track, given only an image from its onboard camera as input. To facilitate learning, an open-source self-driving platform was developed using a unity simulator featuring the robot. A training episode is visualized in Fig 7.

$$reward = \begin{cases} -10 - w_1 \times \text{throttle} & \text{when getting off the track} \\ +1 + w_2 \times \text{throttle} & \text{when on the track} \end{cases}$$

We employ the above as the reward function. A reward of +1 is awarded when the car stays on track and a penalty of -10 for getting off. To control speed, the throttle parameter is employed. We have also conditioned reward in terms of smoothness by reducing max steering with a history wrapper. The new reward is formulated as follows,

$$reward = (1 - delta\_steering) * reward - continuity\_cost.$$

C. High Definition simulator

CARLA is an open-source simulation platform for autonomous driving research. The training with CARLA is expensive and not suited for multiple experiments, thus we have evaluated with a more realistic-environment low-cost simulator Donkeycar.

The observation space, action space, and rewards for the 2D Parking and CarRacing, CARLA, and Donkeycar environments are described in Fig 3.

	Parking-V0	Car racing-V0	Carla	DonkeyCar
Observations	Position, Speed	set of 96x96x3 vectors encoding the image	Camera image - 64×64×3 tensor, Lidar inputs - 64×64×3 tensor	Image observations from an on-board camera view
Actions	Accelerations, Steering angles	set of 3D vectors of the form [s,t,b] where each letter specifies the steering angle (s), throttle (t) and brake (b)	Acceleration and steering - Discrete and Continuous	Throttle, steering
Rewards	Reaching the goal	-0.1 for every frame and +1000/N for every track tile visited, where N is the total number of tiles in the track	Depends on collision, longitudinal speed, speed range, staying in lane, steering angle, lateral acceleration.	Reward of 1 for every step that it stays on the road and a penalty of -10 when it drives off.

Fig. 3. Comparison of various simulator environments

V. Experimental setup

We use OpenAI Gym and Stable Baselines3 environment [10] for simulation of the 2D driving environments. For the CARLA environment, training is performed on Ubuntu 20.1.4 with Nvidia Tesla P4 GPU and Google Cloud Platform. For Donkeycar, the OpenAI gym wrapper is used around the Self Driving Sandbox donkey simulator (sdsandbox) and runs in Ubuntu 20.1.4.

We have implemented the following as part of our evaluation. It is to be noted that not all methods are equally compatible with the considered simulator environments.

1) **Baselines:**
   a) *Deep Q Learning (DQN)* on Carracing V0 env.
   b) *Deep Deterministic Policy Gradient (DDPG)* on Parking V0 env.
   c) *Proximal Policy Optimization (PPO)* on 2D environments.
   d) *Soft Actor Critic (SAC)* on all the environments.
   e) *Truncated Quantile Critics (TQC)* on Donkeycar sim
2) **Alphas:**
   a) *SAC* with *Hindsight Experience replay (SAC+HER)* on Parking V0 environment.
   b) *SAC* with *Prioritized Experience replay (SAC+PER)* on Car racing V0 environment.
   c) *TQC* with with *Hindsight Experience replay (TQC+HER)* on DonkeyCar sim.
   d) Reduced high dimensional input size.
   e) Smooth control.
   f) Generic semantic segmentation with external model.

We chose the Average reward metric (Average of the rewards for the episodes seen so far) for performance comparison.

VI. Results

In this section, we discuss the simulation experiments for validating the proposed algorithm. We design the experiments to answer the following.

1) Do maximum entropy RL algorithms such as SAC give better performance?
2) Does Experience replay always give better results than baseline RL algorithms?
3) Will using an autoencoder for reducing state space also lead to improved rewards?
4) Can interpretability be enabled by decoding the latent space representation of the image input?

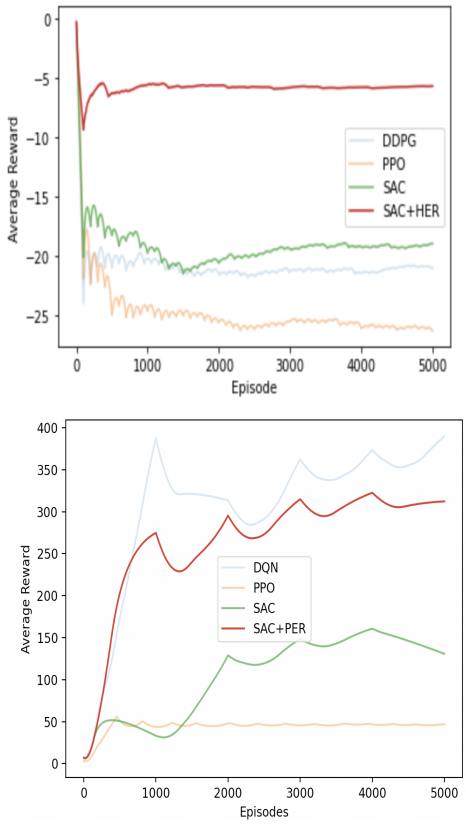

Fig. 4. Reward vs episode for Parking V0 (top) and Car Racing-V0 (bottom)

Training results of parking-v0 are shown in Fig. 4a, SAC performed well compared to baselines- DQN, DDPG as expected. Furthermore, SAC with HER gave the best perfor-

mance. Initial behavior with DQN was the incorrect parking orientation, this may be attributed to discrete action space. Upon changing to continuous action space and using algorithms DDPG, SAC the car was able to park with good orientation but not in the required region marked in blue. Upon increasing training timesteps to 5e6 from 5e3, the parking task is successfully performed in the algorithms. Unexpected behavior was that the reward graphs are flat without major improvement. We learned that this problem is seen with Stable Baselines3 and that the training curve is a good proxy but underestimates the agent's true performance. In the 2D Car-Racing environment (Fig. 4b), SAC with PER was expected to give the highest returns as the important samples with high TD Error are drawn more frequently for training. However, DQN performed better than SAC+PER, SAC, and PPO as seen in Fig. 4b as entropy regularization doesn't improve performance causing instability in training. On further training of TQC, we observe that it gives better returns than SAC in the parking environment (Fig 5).

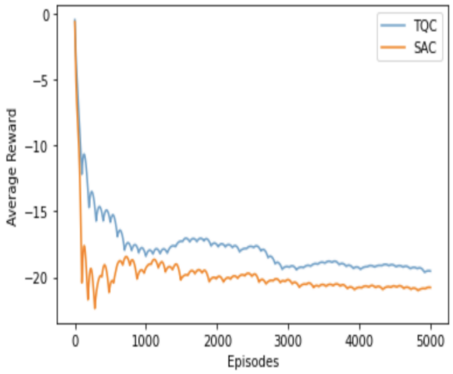

Fig. 5. Performance comparison of SAC and TQC.

Moving on to results from our main simulator of interest Donkeycar with TQC and Hindsight Experience Replay, we observe better performance after introducing an autoencoder as in Fig 6. This is intuitive as the complexity of the model is reduced. We have also changed the reward function from a simplistic control to a more complex one including continuity cost and delta of steering to achieve a smoother driving policy. The performance increased post this step as well.

For interpretability, on the 3D environment CARLA, the baseline SAC + Sequential latent model was able to generate a semantic mask. However, due to the lack of ground truth semantic segmentation masks on the Donkeycar simulator, we use an external autoencoder module [11] to generate the bird-eye view masks as shown in Fig 8.

Thus summarizing the results of our key research questions:

1) Maximum entropy RL algorithms such as SAC do not always give better performance if the reward function or the simulator is not suitable
2) Experience replay almost always gives better results than baseline RL algorithms when combined with it. For example, SAC with any experience replay gives better results than vanilla SAC
3) Using an autoencoder for reducing state-space did lead to improved rewards as observed in the TQC architecture
4) Interpretability can be enabled by decoding the latent space representation of the image input fed to the model, as shown in the resulting diagram (Figure 8)

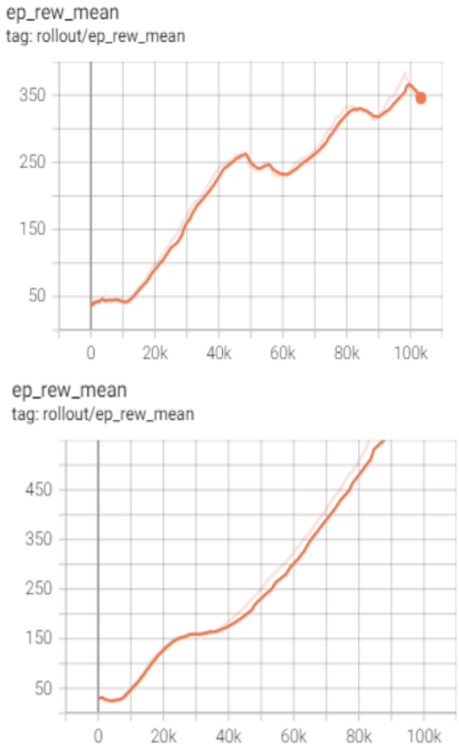

Fig. 6. Performance comparison of SAC and TQC. Comparison of average returns without and with autoencoder

VII. Conclusion

The objective was to incorporate experience replay with the latent sequential environment model and SAC for improved performance as well as interpretability. We verified higher performance with experience replay on the 2D environments and compared it against standard baselines. The best performing model was that of TQC and we used it with the Donkeycar simulator environment. After hyperparameter tuning, we obtained a good driving policy which was able to mostly stay in lane and complete a lap around the track. The performance increased further after reducing high dimensional input size with an autoencoder, by providing a smooth driving control using reward shaping and history wrapping with inputs. There is no existing literature for interpretability (generating a semantic segmentation mask) from an RL algorithm trained on a low-cost simulator like DonkeyCar. We demonstrated using a pretrained autoencoder for the same. For the next steps, we would like to train further with reward shaping for better performance. Also, using an actual high fidelity simulator like CARLA would be beneficial since it provides ground truths for the semantic mask using which we could jointly learn the interpretability module with the RL driving algorithms.

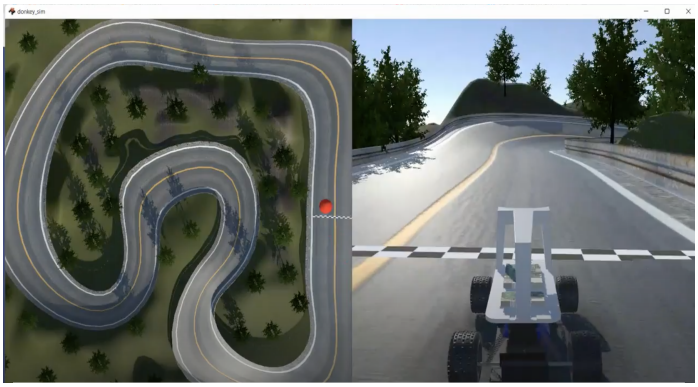

Fig. 7. Donkey simulator

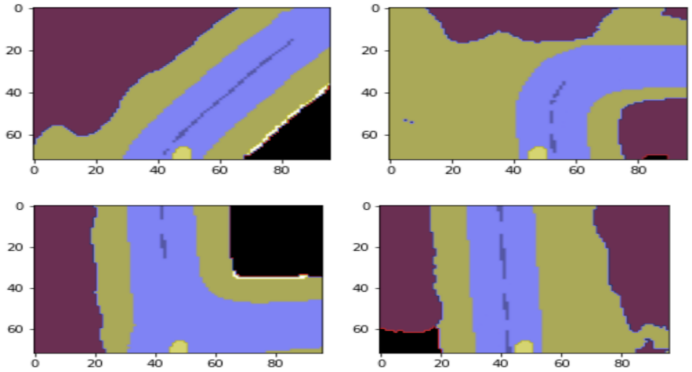

Fig. 8. Semantic mask of Donkey car environment

REFERENCES

[1] 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.

[2] A. Kendall et al., "Learning to Drive in a Day," 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 8248-8254, doi: 10.1109/ICRA.2019.8793742.

[3] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, Patrick Pérez, "Deep Reinforcement Learning for Autonomous Driving: A Survey" arXiv:2002.00444v2, 2020

[4] Jianyu Chen, Shengbo Eben Li, Masayoshi Tomizuka, "Interpretable End-to-end Urban Autonomous Driving with Latent Deep Reinforcement Learning" arXiv:2001.08726, 2020

[5] Ari Viitala, Rinu Boney, Yi Zhao, Alexander Ilin, and Juho Kannala, "Learning to Drive (L2D) as a Low-Cost Benchmark for Real-World Reinforcement Learning" arXiv:2008.00715, 2020

[6] P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Hartl, F. D ¨ urr, and J. M. ¨ Zollner, "Learning how to drive in a real-world simulation with deep q- ¨ networks," in 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 244– 250, IEEE, 2017.

[7] Tom Schaul, John Quan, Ioannis Antonoglou, David Silver, "Prioritized Experience Replay " arXiv:1511.05952 , 2016

[8] J. Kim and J. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," in Proceedings of the IEEE international conference on computer vision, pp. 2942–2950, 2017.

[9] Raffin, A., Sokolkov, R.: "Learning to drive smoothly in minutes. https://github.com/araffin/learning-to-drive-in-5-minutes/ ", 2019

[10] https://stable-baselines.readthedocs.io/en/master/guide/rltips.html

[11] https://github.com/milesial/Pytorch-UNet