BLG 604E Deep Reinforcement Learning Term Project: TORCS Self Driving Cars

Kıvanç Güçkıran
Yildiz Technical University
Davutpasa, İstanbul
Email: kivancguckiran@gmail.com

Can Erhan
Istanbul Technical University
Maslak, İstanbul
Email: erhanc@itu.edu.tr

Onur Karadeeli
Istanbul Technical University
Maslak, İstanbul
Email: karadeli18@itu.edu.tr

Fig. 1. Torcs Car simulator and Open Racing environment

***Abstract*—In this project, we will be training reinforcement learning agent(s) in the Torcs environment which is a popular racing game environment among the researchers. All other participants of the project will be competing with each other in a 1vs1 tournament. In the project we are given a gym like environment, which's specifications given below, to train our agent. In the environment, there are 6 bots (pre-programmed racers) as well as our agent(s). In the tournament event, racers of all the participants will be tested in unseen race tracks. In order to generalize well, we may train your agent in multiple tracks. We will be using the multi-agent paradigm(Train with two racers controlled by agents).**

I. Introduction

TORCS is a highly portable multi platform car racing simulation. It is used as ordinary car racing game, as AI racing game and as research platform.

http://torcs.sourceforge.net/

Competitors can create bots (or agents) to compete in the Torcs racing environment. These bots can range from basic scripting based bots to complex neural network based ones.

In this project we have designed and implemented a Deep Reinforcement Learning based Agent to compete in the Torcs environment. For the deep learning architecture we have used the Pytorch libraries.

II. Preliminaries

Reinforcement Learning is part of Machine Learning practices with supervised and unsupervised learning. Unlike these methods, reinforcement learning creates its own data by interacting with environment. There are two main approaches for solving problems via reinforcement learning, value function based and policy based methods. We have implemented both of these approaches with different algorithms. Below is the summary of the studies we have used in this project.

A. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor (SAC) [?] [1]

SAC uses a modified RL objective function using maximum entropy formulation. Algorithm tries to maximize entropy, in addition to policy updates. This way, agent is encouraged to explore unseen and unknown states. There are two Q networks to estimate expected rewards for policy updates. These two networks are used to minimize Q value overshooting like Double Q learning [2]. Q function is trained with another V function and since these two networks are dependant on each other, some instabilities may occur. To overcome this, SAC utilizes a target value network and updates it with Polyak averaging [3]. Since Q values are the policy's target density function, to achieve differentiation, reparameterization trick is used. Overall, Soft Actor Critic with Maximum Entropy Learning is a data efficient and stable algorithm.

B. Rainbow DQN [4]

Rainbow DQN utilizes multiple improvements on DQN [5] together. These improvements are;

- Double Q Learning [2] - This method is used to overcome overestimation problem on Q networks.
- Priority Experience Replay [6] - Experiences for update are picked with a priority. Mostly used parameter for priority is TD error.
- Dueling Networks [7] - Sometimes, choosing the exact action does not matter too much, but the value function estimation is still important. This way value is calculated nevertheless.
- Noisy Networks [8] - Exploration in environments with Q learning mostly depends on action exploration with epsilon greedy methods. Noisy Networks introduces the

capability of parameter space exploration. Parameter change drives state and action exploration.

These algorithms together forms the Rainbow DQN approach. We have also used C51 output to obtain further improvements [9].

III. Implementation

This section emphasizes on our implementation of the studies explained beforehand. We have tried multiple algorithms like PER-DDPG [10], TD3 [11], SAC, Rainbow DQN, to overcome TORCS problem, and had significant success with two of them, namely SAC and Rainbow DQN. The codebase is heavily borrowed from https://github.com/medipixel/rl$_algorithms$.

A. Architecture

1) SAC: Our neural network architecture for SAC consists of three layers with 512, 256, and 128 weights respectively. We have used linear layers with ReLU activation on hidden layers and gaussian distribution on actions with TanH activation on output layer. We observed improvements on SAC when we have added a single LSTM layer before output layer. Hyper parameters for SAC-LSTM are below:

- Gamma: 0.99
- Tau: 10^{-3}
- Batch Size: 32
- Episode Buffer: 10^3
- Actor Learning Rate: 3.10^{-4}
- Value Learning Rate: 3.10^{-4}
- Q Learning Rate: 3.10^{-4}
- Entropy Learning Rate: 3.10^{-4}
- Policy Update Interval: 2
- Initial Random Action: 10^4

We have used auto entropy tuning using log probabilities. Before LSTM, we have also deployed NSTACK mechanism. We serialized 4 past states and used as input. When we noticed that LSTM outperforms NSTACK approach, we have abandoned SAC and NSTACK mechanism.

2) Rainbow DQN: DQN network consists of three layers of 128 weights. The activation functions of hidden layers are ReLU like, SAC architecture, but outputs are 51 atom distribution over Q values, namely C51. As stated before, we are using Noisy-Net for exploration as opposed to epsilon greedy mechanism. Hyper parameters for Rainbow DQN are below:

- N-Step: 3
- Gamma: 0.975
- Tau: 10^{-3}
- N-Step Weight Parameter: 1
- N-Step Q Regularization Parameter: 10^{-7}
- Buffer Size: 10^5
- Batch Size: 32
- Learning Rate: 10^{-4}
- Adam Epsilon: 10^{-8}
- Adam Weight Decay: 10^{-7}
- PER Alpha: 0.6
- PER Beta: 0.4
- PER Epsilon: 10^{-6}
- Gradient Clip: 10
- Prefill Buffer Size: 10^4
- C51 - V Minimum: -300
- C51 - V Maximum: 300
- C51 - Atom Size: 1530
- NoisyNet Initial Variance: 0.5

B. Reward Shaping

Like many TORCS agent developers, we have noticed the fast left right maneuvers on a straight track. We have tried multiple reward functions to stabilize the car. In addition to this, when the agent steers off track and turns backwards, environment resets. This way agent does not try to recover from the state. We have also tried to recover from this.

1) Termination: We have changed some of the termination conditions provided. Our terminal judge starts after 100 timesteps, similar to this, we have provided agent 100 timesteps to recover from turning backwards. This way we want to see agent try to get on track after spins.

2) Reward Functions: The parameters used in reward functions are defined as,

- V_x: Longitudinal velocity
- V_y: Lateral velocity
- θ: Angle between car and track axis
- $trackpos$: Distance between center of the road and car

Reward functions we have tried are formulated below. We have noticed that in the original reward function, negative angles with sine penalties become positive. This makes agent biased towards left.

- No Trackpos: $V_x \cos\theta - |V_x \sin\theta|$
- Trackpos: $V_x \cos\theta - |V_x \sin\theta| - |V_x trackpos|$
- EndToEnd [12]: $V_x(\cos\theta - |trackpos|)$
- Extra [13]: $V_x \cos\theta - |V_x \sin\theta| - |2V_x \sin\theta trackpos| - V_y \cos\theta$
- Sigmoid: $V_x sigmoid(\cos\theta * 3) - V_x \sin\theta - V_y sigmoid(\cos\theta * 3)$

We have also further tried penalizing not turning at turns using lidar values. Our agent uses "Extra" reward function formulated above. We did not have enough time to throughly test "Sigmoid" function, it looks promising to overcome fast maneuvers on a straight track since it soft clips the cosine rewards.

C. Exploration

Exploration in this environments is done by maximizing entropy in SAC and NoisyNets in DQN algorithm. But learning to utilize brake is a challenge since using brake decreases reward. We have employed Stochastic Braking for this.

1) Try-Brake: Try Brake mechanism is like Stochastic Braking [13]. After a certain amount of timesteps, agent is forced to use brake 10% of the time, again for a certain amount of time. This way we hope that agent will learn to speed up in a straight track and brake before and while turns.

D. Generalization

We have further added 13 more tracks to train and test on to generalize agent's behaviour on unseen tracks. This way, we try to prevent agent to overfit and memorize the tracks. All added tracks are road tracks. We avoided using Spring track since it is very long.

E. Environments

We have changed environment action and step functions to make learning easier for agent. Below are two of the implementation and explanations of the environments we have tried.

1) Continuous Environment: In this environment, we have reduced to action size to 2. First action value is used for both accelerating and braking. Since agent won't try to use them together, defining two actions for them is unnecessary. Smaller values than zero are used for brake values and greater values are used for accelerating. Second action value is used for straight forward steering. We use this environment for SAC algorithm.

2) Discretized Environment: Since our DQN algorithm is suitable to discrete actions, we have dicretized action space into 21 actions. There are 7 steering points on 3 modals. First modal is for accelerating and steering, second modal is only steering and last modal is for braking and steering.

IV. Experiements

We divide our experiements into 3 categories.

- Score vs Loss curves
- Algorithm Comparisons
- Track Comparisons

For the x axis , We decided to use **timestep** feature instead of the **episode** to treat algorithms equally independent of the underlying hardware running simulation.

SAC Algorithm with LSTM

With the SAC algorithm the max speed can reach upto 180 km/h while the average speed is around 125 km/h. These results were obtained around 1.4billion times steps.

The loss against totals steps are depicted in Figure 3

Finally the Reward curve is shown in Figure 4. It is noticable there are steep falls in the Reward around 0.6 billion. This is due to the resume of a previously trained model where some bad rewards are taken.

DQN

We have observed DQN with the following loss and reward figures as in figure 5 and 6.

Algorithms against Tracks We have used 19 different tracks for the training. Some tracks are straightforward like e-track-1 where on the other some of them have difficulties with sharp turns or even gravitational effects.

The success among those tracks are shown in Figure 7 :

Finally the perforance in terms of Max Speed of algorithms in tracks are shown in Figure 8:

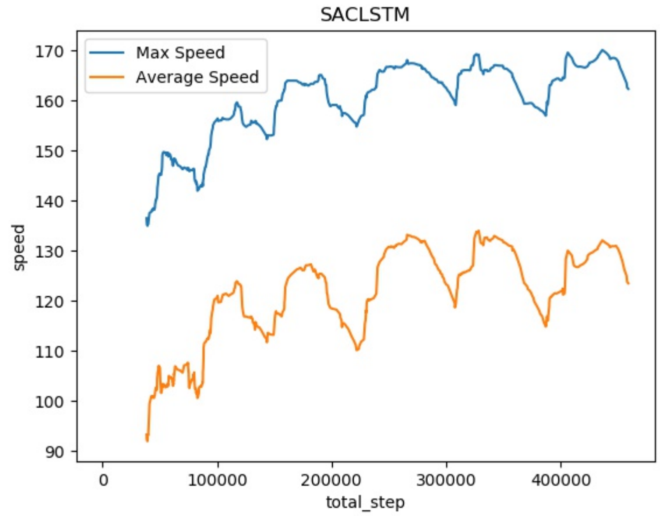

Fig. 2. Max Speed - Average Speed vs total steps with SACLSTM

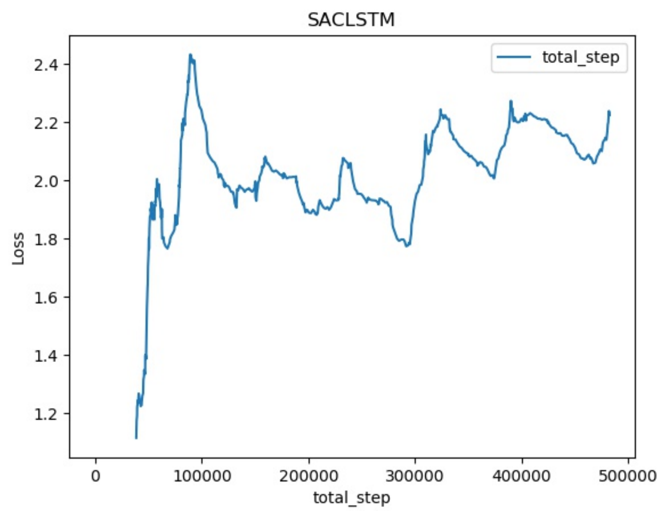

Fig. 3. Total Loss vs total steps with SACLSTM

V. Discussions

TODO

References

[1] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[2] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[3] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992.

[4] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] 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, p. 529, 2015.

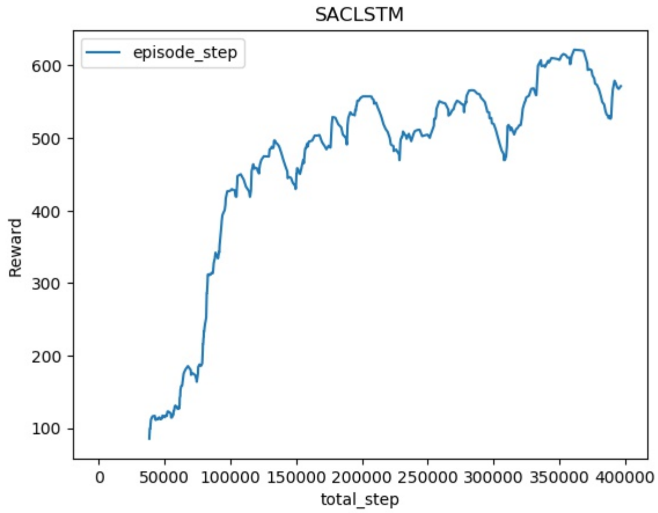

Fig. 4. Reward vs total steps with SACLSTM

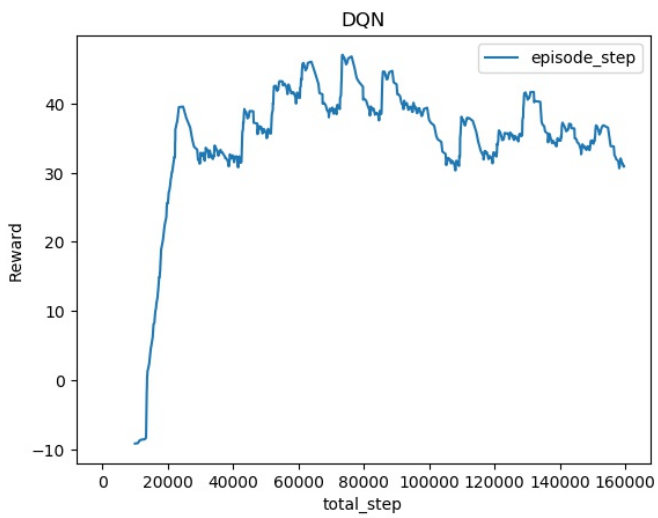

Fig. 5. Reward vs total steps with DQN

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[7] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[8] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[9] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 449–458.

[10] Y. Hou, L. Liu, Q. Wei, X. Xu, and C. Chen, "A novel ddpg method with prioritized experience replay," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2017, pp. 316–321.

[11] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[12] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2070–2075.

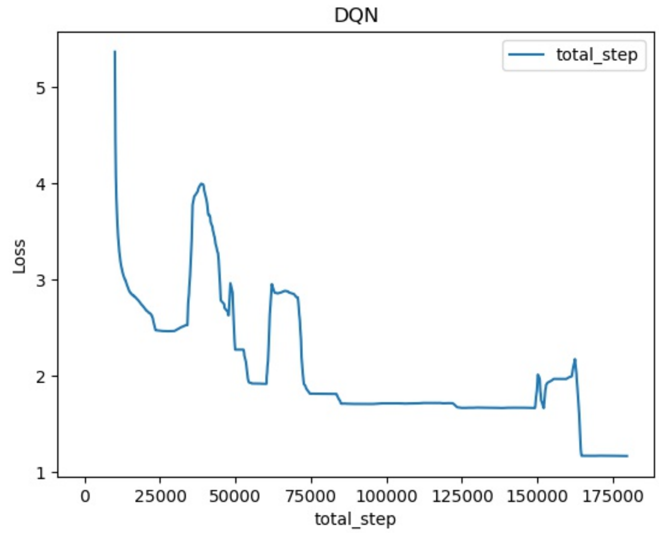

Fig. 6. Loss vs total steps with DQN

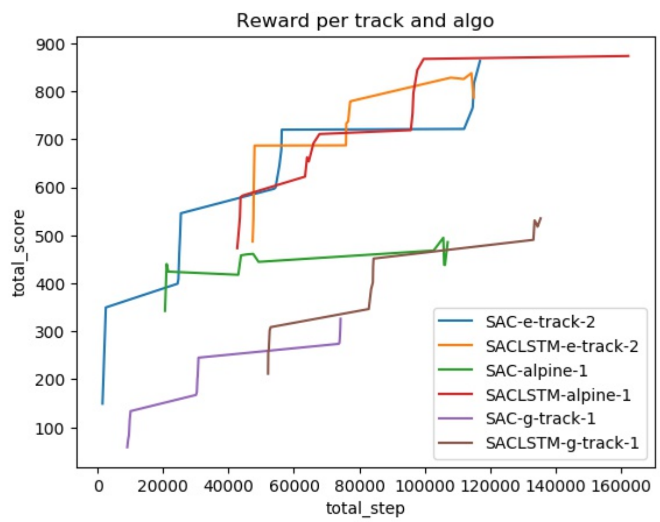

Fig. 7. SAC and SACLSTM algorithms against 3 different tracks

[13] B. Renukuntla, S. Sharma, S. Gadiyaram, V. Elango, and V. Sakaray, "The road to be taken, a deep reinforcement learning approach towards autonomous navigation," https://github.com/charlespwd/project-title, 2017.

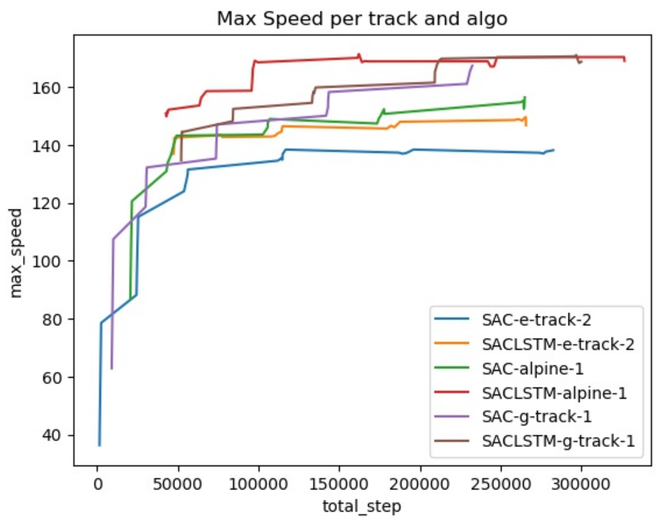

Fig. 8. SAC and SACLSTM algorithms against 3 different tracks