Autonomous Car Racing in Simulation Environment Using Deep Reinforcement Learning

Kıvanç Güçkıran[1], Bülent Bolat[2]
[1]Electronic and Communication Engineering Department, Yildiz Technical University, Istanbul, Turkey
kivancguckiran@gmail.com
[2]Electronic and Communication Engineering Department, Yildiz Technical University, Istanbul, Turkey
bbolat@yildiz.edu.tr

***Abstract*—Self-Driving Cars are, currently a hot topic throughout the globe thanks to the advancements in Deep Learning techniques on computer vision problems. Since driving simulations are fairly important before real life autonomous implementations, there are multiple driving-racing simulations for testing purposes. The Open Racing Car Simulation (TORCS) is a highly portable open source car racing -self-driving- simulation. While it can be used as a game in which human players compete with scripted agents, TORCS provides observation and action API to develop an artificial intelligence agent. This study explores near-optimal Deep Reinforcement Learning agents for TORCS environment using Soft Actor-Critic and Rainbow DQN algorithms, exploration and generalization techniques.**

***Keywords*—Deep Reinforcement Learning, TORCS, Self-Driving Car**

I. Introduction

Self-driving cars are one of the most important and promising assets of our time. It has been a challenge and inspiration for researchers and engineers throughout the world for decades. With the introduction of Deep Learning practices and computer vision techniques, autonomous vehicles in the near future is not a dream [1]. The development cycle of the self-driving cars must start from simulations since creating driving data for training would be inefficient, risky and time-consuming. TORCS, The Open Racing Car Simulator, is open, flexible and has a portable interface for AI development [2]. Example race is shown in Figure 1.

Achieving autonomous agents is a very complicated task. There are multiple practices in machine learning to train agents to learn and act. First one is supervised learning, in which the dataset contains the data and the ground truth labels, agents will try to predict true labels. Second one is unsupervised learning, instead of having labels, this time only data is provided and the agents need to correlate and group the data together. The last one, which is our method, reinforcement learning, creates its own data by interacting with the environment [3]. This is the best-suited approach since most of the time there will not be any ground truth actions for agents to learn.

Latest improvements in Deep Learning affected every area of computation, and one of the areas affected is Reinforcement Learning. With the GPUs' huge performance increase, Reinforcement Learning practices with Deep Learning techniques

Figure 1: Example race

became accessible. In this study, we will try to find a near-optimal driver for TORCS environment using Deep Reinforcement Learning techniques. The rest of the paper is organized as follows. In Section II, information regarding our approaches is given. Our methodology, algorithms and strategies we have used are explained in Section III. In Section IV, results of our approaches are given and the last section has concluding remarks.

II. Preliminaries

Reinforcement Learning (RL) is a goal-oriented machine learning practice which tries to maximize the agent's cumulative rewards. The reward is given when an agent behaves towards the goal or achieves the goal. Negative reward as punishment is also plausible in some cases.

RL agents also receive observations/states from environment and acts upon them. This cycle goes on until an optimal agent is found. Figure 2 depicts this behaviour. This behavior is formulated in Markov Decision Process and MDPs are defined with five components, which are:

- S: State space
- A: Action space
- R: Reward function

978-1-7281-2868-9/19/$31.00 ©2019 IEEE

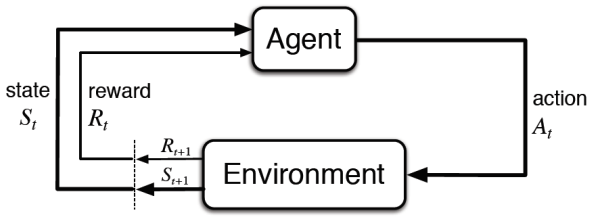


Figure 2: Reinforcement Learning Setting [3]

- P: Transition function
- γ: Discount factor

State space depends on the environment. All possible agent's perception of the environment forms the state space. Similar to state space, action space defined as all possible actions that agents can use and utilize. Reward function depends on state and action and defines when and how much reward is received with a state-action couple. Transition function addresses which state is transitioned to, after an action is taken in a state. The last one is the discount factor, which calculates how much an agent takes future rewards into consideration.

When MDP is known, there are common practices as Dynamic Programming to solve MDP via visiting all state action pairs recursively [4]. On the other hand, when MDP is not known to the agent, Reinforcement Learning practices are used. In RL practices trajectories are sampled from environment and agents learn from them. There are mainly two approaches to Reinforcement Learning problems, value-based methods, and policy-based methods. There are also, hybrid methods like Actor-Critic, which in addition to policies, value networks are also present within the algorithms.

Value-based methods define their policy via acting greedily to the value function and this way, it uses its current knowledge on the environment. This leads to sub-optimal policies since agents need to explore new trajectories to obtain optimality. There are multiple approaches to this dilemma like $epsilon$-greedy strategy, where sometimes agent acts randomly using $epsilon$ value.

Policy-based methods try to maximize their cumulative reward, mapping states to actions. This time, exploration is generally done via adding noise to actions. There are also hybrid methods which utilize a value function within policy-based methods. These methods are called Actor-Critic methods.

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

SAC uses a modified RL objective function using maximum entropy formulation [5]. The algorithm tries to maximize entropy, in addition to policy updates. This way, the 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 overestimating like Double Q learning [6]. Q function is trained with another V function and since these two networks are co-dependent on each other, there is a significant chance that value functions become unstable. To prevent this, SAC utilizes a target value network and updates it via Polyak averaging [7].

Q values are the policy's target density function. To achieve differentiation for the policy updates, the re-parameterization trick is used. Overall, Soft Actor-Critic with Maximum Entropy Learning is a data efficient and stable algorithm.

B. Rainbow DQN

Rainbow DQN [8] utilizes multiple improvements on DQN [9] together. These improvements are;

- Double Q Learning [6] - This method is used to overcome overestimation problem on Q networks.
- Priority Experience Replay [10] - Experiences for updates are picked with a priority. Mostly used parameter for priority is TD error.
- Dueling Networks [11] - Sometimes, choosing the exact action does not matter too much, but the value function estimation is still important. This method guarantees value calculation in all cases.
- Noisy Networks [12] - Exploration in environments with Q learning mostly depends on the action-exploration with epsilon-greedy methods. Noisy Networks introduces the capability of parameter space exploration. Parameter change drives state and action exploration indirectly.

These algorithms together form the Rainbow DQN approach. We have also used C51 output to obtain further improvements on Q value distribution [13].

C. TORCS

TORCS provides an API for AI agents to act and learn from. This API has several observations like angle, speed, damage, gear, etc. We are using 6 observations with a total of 29 dimensions from API, which are:

- Angle - 1: Angle between the tangent of the track and the car
- Track - 19: Lidar sensor on the front of the car scanning 180 degrees
- TrackPos - 1: Distance from the middle of the track, greater than 0.5 if off the track
- Speed - 3: Cartesian speeds where the x-axis is always pointing the front of the car
- Wheel speeds - 4: Angular speeds(rad/s) for each wheel
- RPM - 1: Engine speed

For actions, we are using acceleration, brake and steer.

III. METHODOLOGY

This section emphasizes on our implementation of the studies explained beforehand. First, architectures for the algorithms used in this study will be explained, then reward shaping and termination topics will be detailed, and lastly, exploration, generalization and environmental changes will be explained thoroughly. Our codebase and implementation can be found at https://github.com/kivancguckiran/torcs-rl-agent.

A. Architecture

1) SAC: Each neural network architecture for SAC consists of fully connected layers with 512, 256, and 128 weights respectively as seen in Figure 3. We have used ReLU activations on hidden layers and Gaussian distribution on actions with TanH activation on the output layer. We have observed improvements on SAC when we added a single LSTM layer before the output layer. Hyperparameters for SAC-LSTM are below. These parameters are selected via trial and error using a simple grid search.

- Gamma: 0.99
- Tau: 10^{-3}
- Batch Size: 32
- Step Size: 16
- 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 implemented a custom buffer for the LSTM. Since LSTM needs sequential samples, standard experience replay is inappropriate. Thus, we buffer a whole episode sequentially and select episodes randomly according to batch size on training time. Then, we randomly pick an index from each episode and train with the following 16 samples.

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 it and continued with LSTM.

2) Rainbow DQN: DQN network consists of three fully conected layers of 128 weights each as seen in Figure 4. 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 the epsilon-greedy mechanism. Hyperparameters for Rainbow DQN are below. These parameters are selected via trial and error using a simple grid search.

- N-Step: 3
- Gamma: 0.99
- 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

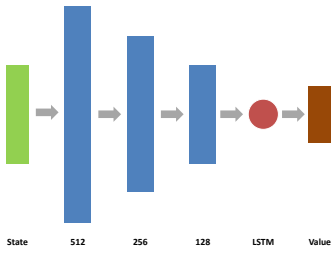
Like many TORCS AI developers, we also have noticed the fast left-right maneuvers (slaloming) on a straight track. We have tried multiple reward functions to stabilize the car. In addition to this, when the agent steers off track and turns backward, environment resets. In this situation, agent does not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

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

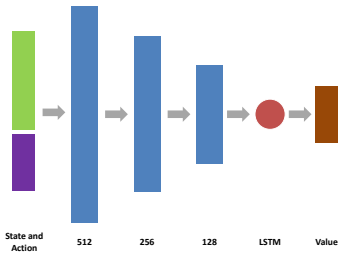
- V_x: Longitudinal velocity
- V_y: Lateral velocity
- θ: Angle between car and track axis
- $trackpos$: Distance between center of the road and car

Reward functions that we have tried are formulated below.

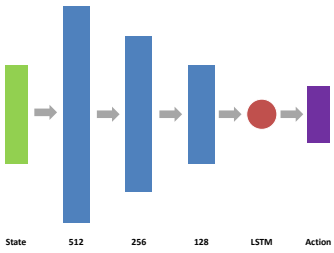
- No Trackpos: Track position is ignored.
$$V_x \cos\theta - |V_x \sin\theta|$$
- Trackpos: Track position is taken into consideration.
$$V_x \cos\theta - |V_x \sin\theta| - |V_x trackpos|$$
- EndToEnd [14]: Car's angle as penalty is not used.
$$V_x(\cos\theta - |trackpos|)$$
- DeepRLTorcs [15]: Track position penalty is discounted with car's angle. Car's angle as penalty is used. Additionally, lateral velocity as penalty is used with discounting towards car's angle.
$$V_x \cos\theta - |V_x \sin\theta| - |2V_x \sin\theta trackpos| - V_y \cos\theta$$



(a) SAC V Network



(b) SAC Q Network



(c) SAC Policy Network

Figure 3: SAC Architecture

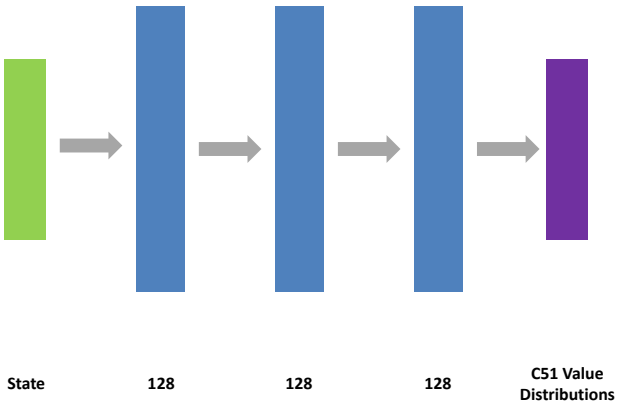


Figure 4: DQN Architecture

- Sigmoid: Same as above reward function, only difference is rewards are flatlined at center to overcome slaloming.

$$V_x sigmoid(3\cos\theta) - V_x \sin\theta - V_y sigmoid(3\cos\theta)$$

Our agent uses "DeepRLTorcs" reward function formulated above. We did not see any improvements with "Sigmoid" function but, it looks promising to overcome slaloming on a straight track since it soft clips the cosine of the longitudinal velocity. Reward shaping with plateaus in the center of the track might overcome slaloming in the future.

2) Termination: The active episode is terminated if no progress is achieved within 100 timesteps. Progress is defined as achieving 5km/h. Similar to this, the agent is provided with an additional 100 timesteps to recover from turning backward. This way we want to see agent try to get on track after spins.

C. Exploration

Exploration in this environment is done by maximizing entropy in SAC and NoisyNets in DQN algorithm. But learning to utilize brake is a challenge since using the brake action decreases reward. This way agent avoids using the brake action altogether. We have employed the Try-Brake mechanism to overcome this problem.

1) Try-Brake: Try Brake mechanism is like Stochastic Braking [15]. After a certain amount of timesteps, the 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. These trials are done according to a Gaussian distribution as seen in Figure 5.

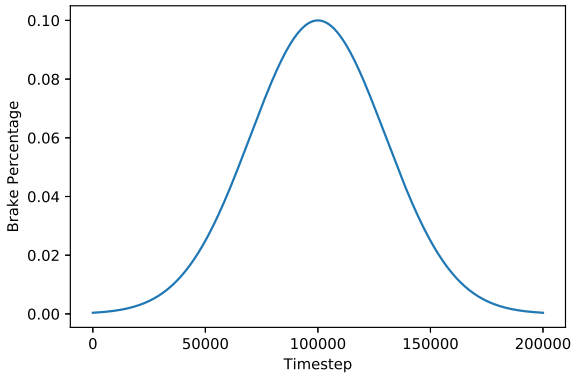


Figure 5: Try-Brake Distribution

D. Generalization

We have added nearly every road track to train and test on to generalize agent's behavior on unseen tracks. This way, we try to prevent agent to overfit and memorize the tracks. We have avoided using Spring track since it is very long.

The list of tracks used for training and test are listed in the first column of Table IV. Agents are trained in these tracks in a circular fashion. Each track is trained for 5 episodes then skips to next track.

E. Action Spaces

We have prepared optimal action spaces to make learning easier and faster for the agent. Below are two of the implementations and explanations of the environments we have tried. State and action values are normalized between -1 and $+1$.

1) Continuous Action Space: In this environment, we have reduced to action size to 2. First action value is used for both accelerating and braking. Since the agent should not 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 steering. We use this environment for SAC algorithm.

2) Discretized Action Space: Since our DQN algorithm is suitable for discrete actions, we have discretized action space into 21 actions. There are 7 steering points on 3 intervals. The first interval is for accelerating and steering, the second interval is only steering and the last interval is for braking and steering. Actions are depicted in Table I.

Acceleration	Brake	Steer
+1	-1	{-1, -0.66, -0.33, 0, 0.33, 0.66, 1}
-1	+1	{-1, -0.66, -0.33, 0, 0.33, 0.66, 1}
-1	-1	{-1, -0.66, -0.33, 0, 0.33, 0.66, 1}

Table I: Discretized Actions

IV. Results

We have achieved significant success with both of these approaches. SAC-LSTM agent was more data-efficient and performant in terms of track consistency and speed. Episode versus Score can be seen in Figure 6. Scores, which is the accumulated rewards, are aggregated over 250 episodes for brevity.

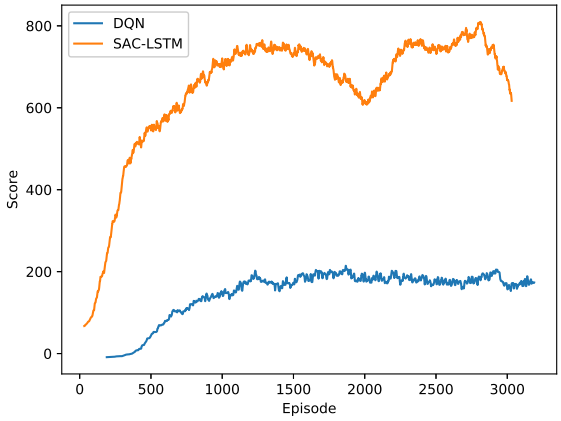


Figure 6: Episode vs Score. Maximum and Standard Deviation can be seen in Table II.

Algorithm	Standard Deviation	Maximum
DQN	306.25	1255.94
SAC-LSTM	433.22	1395.0

Table II: Standard Deviation and Maximum values of Scores

Further analysis will be done on the SAC-LSTM agent. Since this study is designed for a car race, speed is the crucial factor. Performance of the agent in terms of speeds versus episode can be seen in Figure 7. Values are aggregated over 250 episodes.

As discussed before, there are multiple tracks in the environment with different difficulties. Table IV shows results for each track for SAC-LSTM algorithm.

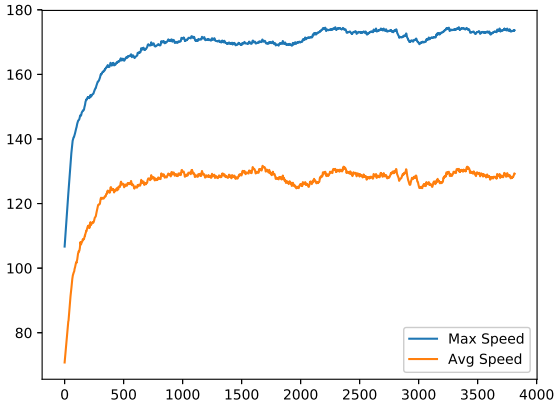


Figure 7: Episode vs Speeds. Maximum and Standard Deviation can be seen in Table III.

Type	Standard Deviation	Maximum
Max Speed	24.94	216.42
Avg Speed	28.52	164.59

Table III: Standard Deviation and Min, Max values of Speeds

Final agents were trained around 6.10^6 timesteps, 5.10^3 episodes on hardware with Intel i9-9900k CPU and GeForce RTX 2060 GPU.

V. Conclusion

We have implemented two different algorithms for TORCS with great success. Agents complete tracks most of the time around 140 km/h average speed and around 190 km/h maximum speed. It is observed that agents generalize well on unseen tracks. Before Try-Brake implementation, the agent did not use the brake action at all to prevent obtaining low rewards. This seems to be fixed with the implementation of the Try-Brake mechanism.

Another problem we have faced was fast left-right maneuvers, also mentioned as slaloming, partly solved via reward shaping and LSTM. We noticed that this problem is an issue of the reward function, thus better reward shaping might overcome this in the near future. Since these maneuvers happen frequently at high speeds, cases other than racing might not face this problem. Race with SAC-LSTM agent against scripted bots and races between Rainbow DQN and SAC-LSTM agents from our trained agent's perspective can be viewed from https://youtu.be/f82EBvPKyDI.

We argue that the reasons behind SAC and SAC-LSTM performed better from Rainbow DQN algorithm are because of the exploration methods and the continuous action space. SAC tries to maximize entropy and this allows the agent to explore uncertain regions of the action space. Additionally, since SAC's policy network contains continuous actions, braking, accelerating and steering can be controlled with continuous actions, unlike Rainbow DQN's discretized 27 actions. This difference might have been helpful to obtain stability on the road.

Track	Max Score	Avg Score	Max Speed	Avg Speed
forza	**1395.0**	742.95	216.0	143.27
g-track-1	954.0	611.36	200.36	131.56
g-track-2	1182.0	**912.06**	205.65	149.32
g-track-3	1154.0	574.16	182.0	109.11
ole-road-1	1103.0	209.70	**216.42**	121.65
ruudskogen	1294.0	575.58	199.78	117.08
street-1	1208.0	597.32	198.0	117.44
wheel-1	1248.0	758.45	206.05	139.16
wheel-2	1152.0	636.49	216.19	128.87
aalborg	922.0	183.84	189.16	80.54
alpine-1	1264.0	822.09	203.18	118.94
alpine-2	1119.0	677.13	194.43	103.02
e-track-1	1080.0	325.40	207.54	119.02
e-track-2	1212.0	907.00	179.65	104.55
e-track-4	1293.0	881.47	214.73	**152.13**
e-track-6	1201.0	583.34	213.59	130.12
eroad	1264.0	883.87	200.59	131.87
e-track-3	1383.0	895.41	211.87	137.35

Table IV: Scores on different tracks. Max value achieved for each column represented as bold.

This study is a step towards using Deep Reinforcement Learning practices for self-driving cars. It can be seen that these methods are capable of learning to drive without supervision. Furthermore, these agents are easily transferable to the real world robotics platforms. Together with other machine learning practices, Deep Reinforcement Learning methods are expected to be used actively in the autonomous car industry.

ACKNOWLEDGEMENTS

This work was part of the term project for BLG604E Deep Reinforcement Learning course from ITU. The project consisted of an autonomous car race with other participants from the class. Our agent became the fastest and won first place.

We thank Can Erhan, for his contributions to the code base and implementations. We also want to thank Onur Karadeli, for his points regarding reward functions.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Software available at http://torcs. sourceforge. net*, vol. 4, no. 6, 2000.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.