Comparative Analysis of Exploration Algorithms in Deep Reinforcement Learning for Autonomous Driving

How does epsilon-greedy, random network distillation, bootstrapped DQN affect training and the robustness of final policies under various testing conditions in autonomous driving?

Efe Sözen[1]

Supervisor(s): Matthijs Spaan[1]**, Moritz Zanger**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Efe Sözen
Email: e.sozen@student.tudelft.nl
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan, Moritz Zanger, Elena Congeduti

Abstract

Autonomous driving is a rapidly evolving field that aims to enhance road safety and reduce accidents through the use of advanced software and hardware technologies. Reinforcement learning (RL) combined with deep neural networks has emerged as a promising approach for training autonomous agents. This research paper investigates three exploration algorithms —Epsilon-Greedy, Random Network Distillation (RND), and Bootstrapped Deep Q-Network (DQN)— within the context of autonomous driving. Performance is assessed based on episodic returns in training and testing environments, as well as the time required to train the networks. The results show significant improvement in learning capability using Bootstrapped DQN without critical differences in training time. There also exists a potential to increase episodic returns further given an increase in the number of steps to train the models.

1 Introduction

Deep reinforcement learning (DRL) has become an intriguing area of research since deep learning has started to develop. The attempts of exceeding human performance on Atari [1][18][19] has inspired many other disciplines to utilize DRL, such as finance [17], robotics [13], human movement [3] and autonomous driving [14].

Autonomous driving is a complex problem with computational challenges and specialized hardware requirements [14]. Autonomous vehicles can be trained with DRL while avoiding the limitations of a real-world environment by utilizing simulators that create virtual environments [8].

DRL uses deep neural networks as well as a policy generated by the reinforcement learning algorithm, which allows for better performance in complex environments. While reinforcement learning algorithms fail to improve performance on larger state spaces, the ability of neural networks to generalize on extensive state spaces allows for the increase in performance metrics with DRL.

A relevant question for the training of an autonomously-driving agent with DRL, which would also apply to many other RL agents, is how to balance the agent's exploration and exploitation. In the setting of this research paper, exploration refers to accumulating new information about the environment the agent is in, whereas exploitation indicates using the already gathered information.

While Epsilon-Greedy is a common dithering algorithm used to balance exploration and exploitation, deep exploration is argued to accomplish improved performance in complex environments with larger state spaces. Some algorithms that could be used as examples for deep exploration are Random Network Distillation (RND) [21], Bootstrapped Deep Q Networks (DQN) [20], Noisy Networks [9] and Diversity-Driven Exploration [11]. There are additional methods of exploration, however, this paper focuses on two of the deep exploration methods, which are RND and Bootstrapped DQN.

These exploration methods are critical in the performance of the trained network as the efficiency of the exploration of the agent is a direct contributor. Bootstrapped DQN is shown to learn faster than DQN and increases the cumulative rewards by orders of magnitude in the Atari environment [20]. A network trained with RND and Proximal Policy Optimization (PPO) [24] has had significantly better results than PPO alone on Montezuma's Revenge [21]. These results beg the question of how much of an improvement can these exploration algorithms provide to an agent learning to autonomously drive in a simulated environment.

The research efforts for autonomous driving and deep exploration has been mostly separate. Of the papers published for the aforementioned exploration algorithms, not many have been tested specifically for autonomous driving, which motivates the purpose of this research paper. The details of the current state of research are discussed in Section 2.

Therefore the question that is being attempted to be answered in this research paper is: "How does Epsilon-Greedy, Random Network Distillation, Bootstrapped DQN affect training and the robustness of final policies under various testing conditions in autonomous driving?". As a result, this paper presents a comparison of the performances for Epsilon-Greedy, Bootstrapped DQN, and Random Network Distillation made in terms of robustness and time spent for training, where robustness is measured by the episodic return for three different maps. These maps are from the high-fidelity driving simulator CARLA [8]. To have the methods of exploration as the only independent variable in the experiments, all exploration algorithms are implemented with Deep Q Networks as the reinforcement learning method.

In Section 2, the exploration algorithms as well as the use of DRL in autonomous driving are explained further. Section 3 discusses the methodology of this research. Section 4 demonstrates the implementation details of the aforementioned algorithms. Section 5 discusses the results obtained from the experiments. In Section 6 critics of responsible research are made, which is followed up by Section 7 that discusses the limitations of the experiments. Finally, in Section 8 final remarks are made about the experiment and future work.

2 Related Work

This section provides an overview of the literature relevant to the research paper. It begins by discussing Markov Decision Processes which are a foundational concept for RL. Then Deep Q Networks are elaborated upon, which combine deep neural networks with Q-learning to learn optimal policies for sequential decision-making problems. Subsequently, the section explores the use of deep reinforcement learning in the field of autonomous driving. The importance of exploration methods in reinforcement learning, specifically Epsilon-Greedy and Random Network Distillation, is addressed. Finally, the extension of the DQN algorithm known as Bootstrapped DQN, which aims to enhance exploration and policy robustness, is discussed.

2.1 Markov Decision Process

A Markov Decision Process (MDP) is a widely used mathematical framework for modeling sequential decision-making problems in uncertain environments [23] [25]. In an MDP, an agent interacts with an environment over a series of discrete time steps, where it observes the current state, selects an action, and receives a reward based on the transition to the next state. The MDP assumes the Markov property, stating that future states and rewards depend only on the current state and action. The goal of an MDP is to find an optimal policy that maximizes the cumulative expected rewards over time, driving the development of effective reinforcement learning algorithms [23] [25].

2.2 Deep Q Networks

Deep Q-Networks are a class of reinforcement learning algorithms that combine deep neural networks with Q-learning [26], a popular model-free reinforcement learning technique [19]. DQNs are designed to learn optimal policies for sequential decision-making problems in an end-to-end manner, directly from raw sensory inputs.

The core idea behind DQNs is to approximate the Q-function, which represents the expected cumulative rewards for taking a particular action in a given state. By estimating the Q-values for different state-action pairs, DQNs can make decisions to maximize long-term rewards.

DQNs utilize deep neural networks as function approximators to handle high-dimensional and complex input spaces [19]. The network takes the state of the environment as input and produces Q-values for all possible actions as output. During training, the network parameters are updated iteratively by minimizing the difference between the predicted Q-values and the target Q-values, which are computed using a variant of the Bellman equation.

To mitigate issues related to correlation and instability in the learning process, DQNs employ an experience replay mechanism. This mechanism stores past experiences (state, action, reward, next state) in a replay memory buffer, from which batches of experiences are sampled randomly during training [15]. This random sampling breaks the sequential correlations and improves data efficiency.

Additionally, DQNs incorporate a target network to stabilize the learning process. The target network is a separate copy of the main network, which is periodically updated with the current network's parameters. This fixed target network provides more stable targets for Q-value approximation during training [19].

DQNs have been successfully applied to various domains, including Atari games, controlling robotic systems, and optimizing complex tasks. Their ability to learn directly from raw sensory inputs and handle high-dimensional state spaces has made them a powerful tool in reinforcement learning research.

2.3 Deep Reinforcement Learning for Autonomous Driving

Deep reinforcement learning has emerged as a promising approach for autonomous driving, enabling vehicles to learn complex driving policies directly from sensory inputs. Several notable papers have demonstrated the application of deep reinforcement learning in this domain.

In the work of Pomerleau [22], reinforcement learning was applied to autonomous driving, demonstrating the effectiveness of the approach. The system, known as ALVINN (Autonomous Land Vehicle in a Neural Network), learned to drive by training on a large dataset of expert demonstrations and refining the learned policy through interactions with the environment.

In a different paper [2], an end-to-end approach for autonomous driving using deep reinforcement learning is presented. The authors trained a convolutional neural network to map raw pixels from a front-facing camera directly to steering commands, effectively learning to drive in a simulated environment.

Many other papers have been proposed to train with DRL in CARLA [8], which is critical for the context of this paper. For instance, Latent DRL was used to show end-to-end driving capabilities in urban driving scenarios [5]. A model-free approach has also been used [6], which performed well in roundabouts and complex driving environments. A novel combination of DRL, imitation learning, and prioritized experience replay was proposed [16], which compared the performances of different learning algorithms on CARLA. In a different paper [10], a DRL algorithm that utilizes Proximal Policy Optimization has been proposed to learn complex intersections.

2.4 Exploration

An exploration algorithm is a technique that enables agents to explore their environment effectively while learning optimal policies. Exploration algorithms aim to strike a balance between exploiting the knowledge gained so far and actively seeking out new and uncertain regions of the environment. These algorithms play a crucial role in discovering and learning optimal strategies in RL problems.

Epsilon-Greedy

Epsilon-Greedy is a popular algorithm commonly used in reinforcement learning. It involves selecting the action with the highest expected reward probabilistically while occasionally choosing a random action with a probability determined by the exploration parameter, epsilon.

By incorporating randomness into action selection, Epsilon-Greedy allows the agent to explore uncharted regions of the action space, enabling the discovery of potentially better policies.

Random Network Distillation

Random Network Distillation is an exploration technique in reinforcement learning that leverages the concept of curiosity to drive agent behavior. RND utilizes two neural networks: a target network and a predictor network. The target network is a randomly initialized network that generates a fixed set of random target vectors. The predictor network learns to predict these target vectors based on the observed states encountered during exploration [3]. The prediction errors serve as intrinsic rewards, incentivizing the agent to explore novel or less familiar states [3].

By having the predictor network learn from the target network's predictions, RND fosters curiosity-driven exploration, allowing the agent to uncover unknown aspects of the environment and improve its policy learning capabilities. This approach can be particularly valuable in autonomous driving scenarios, where exploring a wide range of driving situations is essential for robust and adaptive decision-making.

Bootstrapped DQN

Bootstrapped DQN is an extension of the original DQN algorithm that aims to enhance exploration and policy robustness. In Bootstrapped DQN, multiple value estimation networks, also known as value heads, are employed to approximate the action-value function [20]. Each value head learns a different estimate of the action values, enabling the agent to explore multiple potential policies simultaneously.

During action selection, Bootstrapped DQN leverages Thompson sampling, a probabilistic approach, to choose an estimate from one of the value heads. This introduces exploration by encouraging the agent to consider multiple potential policies simultaneously. By combining the bootstrapping principle with Thompson sampling, Bootstrapped DQN promotes exploration in the action space, allowing the agent to achieve improved training stability and robustness in different environments.

3 Methodology

This section describes the methodology employed to compare the performance of three algorithms, namely Epsilon-Greedy, Bootstrapped DQN, and Random Network Distillation with DQN, in the context of autonomous driving. The experiments were first conducted using the CarRacing [1] environment from OpenAI's Gym and then extended to CARLA.

CarRacing environment's similarity in terms of input and output to CARLA, combined with the significant computational ease at training models have led to the decision to train models in CarRacing first. The main point of doing this was to test the algorithms in terms of correctness and to make sure the models learn how to drive. However, due to the simplicity of CarRacing, it is impossible to decide whether or not the models learn to drive in more realistic traffic. Therefore the final training and robustness tests are made in CARLA. A library for the integration of the Gym environment into CARLA is *gym-carla* [4], which adapts the interactions with the CarRacing environment to CARLA. Example images from the CarRacing and *gym-carla* environments are shown in Figure 4 and Figure 1 respectively.

The training process took place on DelftBlue [7], which is a high-performance computer specialized for parallel computing. This was crucial for the training process in CARLA because CARLA requires advanced hardware to be able to train models.

Epsilon-Greedy, Bootstrapped DQN, and RND were implemented over a modified DQN implementation on CleanRL [12] (implementation details on Section 4) and adapted to the specific requirements of the CarRacing and CARLA environments. Each algorithm was trained independently to learn the optimal policy for driving autonomously. A fixed training duration of 500 thousand steps was established for consistency and to ensure comparability across the algorithms. After training in the CarRacing environment, the algorithms were evaluated based on their episodic returns per step. The scores were compared to see if the algorithms learn better or worse than each other.

To further investigate the robustness and generalization capabilities of the algorithms, and to understand whether or not these algorithms cause a difference in robustness or training time, models were trained again on CARLA. The training and robustness tests were done on different CARLA maps[2]. The training took place in Town03, where the robustness tests were done on Town03, Town04, and Town05 for each of the three algorithms. Town03 is an urban map with roundabouts, Town04 is a small town embedded in mountains and Town05 is a squared grid town with cross junctions. The variability in the driving conditions in the different maps from CARLA tests the robustness of the models trained with different exploration algorithms.

The models were evaluated using an evaluation function that used the trained models to drive the simulated cars in the environments and collect rewards. The evaluation function ran the internal function that drives and completes a circuit 10 times. The episodic returns for each run are collected and averaged for comparison. The higher the average episodic return for an algorithm, the more learning it could do, which means that the exploration method used was more efficient since all else was kept equal.

Figure 1: Example image that is used as input from gym-carla

4 Implementations

In this section, the implementations of the exploration algorithms utilized in this research project are presented. These algorithms are implemented using the CleanRL framework, which has shown promising performance in various Atari games and is well-suited for image-related environments like CarRacing and CARLA. With minor changes to the CleanRL implementation, the network architecture was made fit for CARLA. Each algorithm incorporates specific modifications

[1] https://www.gymlibrary.dev/environments/box2d/car racing

[2] https://carla.readthedocs.io/en/latest/core_map/

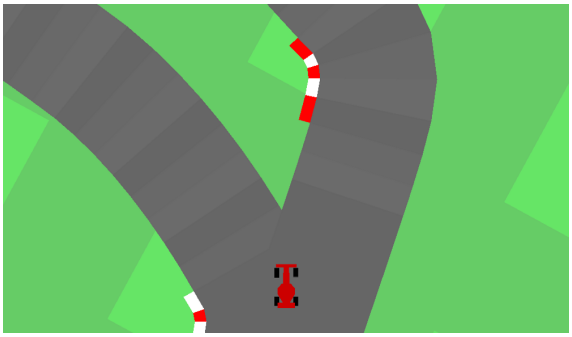

Figure 2: Example image that is used as input from CarRacing

to the default DQN implementation to enhance exploration and improve the agent's learning capabilities.

Additionally, since it is not possible to directly utilize CleanRL's implementations on CARLA, which are compatible with the Gym environment, *gym-carla* was used.

4.1 Epsilon-Greedy and DQN

The Epsilon-Greedy implementation that is utilized in this research paper is the default implementation of DQN in CleanRL with an additional convolutional layer added to the network structure. Since CleanRL performs well with many different Atari games, it is a fitting implementation for image-based environments like CarRacing and CARLA.

The implementation of Epsilon-Greedy is as follows, a random value is selected in each step of the algorithm and checked if the value is smaller than epsilon or not. If it is smaller, a random action is taken, if it is greater than or equal to epsilon, the action with the highest Q-value is executed. The epsilon value is decayed over time to ensure higher rates of exploration at first and an increase in exploration as the algorithm progresses.

4.2 Random Network Distillation

The RND algorithm is written on the DQN implementation of CleanRL. For the RND extension, two neural networks with 5 convolutional layers and 2 fully connected layers are utilized. One of the neural networks is the target and the other is the predictor. The intrinsic reward that the agent is rewarded with for exploring lesser-known states is calculated by the mean-squared error of the output of the prediction network and the output of the target network. The prediction network learns from the target network every *train_frequency*, which was set to 10, gradient-steps of the algorithm. As the prediction network gets better at predicting the output of the target network for some particular input, it is inferred that this input state has been visited enough times for the network to learn the output of the target network. Since the predictor can predict the target at this point, the intrinsic reward is lower.

There are differences between the RND implementation in [21] and the implementation utilized for this experiment. Although RND could be used with other DRL algorithms, the original paper [21] has found an increase in exploration ability using PPO. For the sake of the experiments, DQN was fitting to be able to compare the efficiency of the exploration across the same algorithm.

Next, the paper suggests that a non-episodic approach, where the end-game does not lose the agent as many points, results in better exploration. This was not possible due to a similar reason as to why PPO was not used. To be able to keep the exploration method as the only independent variable, it was compared to Bootstrapped DQN and Epsilon-Greedy in the same way these algorithms have obtained their episodic returns.

The paper also mentioned the importance of observation normalization [21]. For the implementation used in this experiment, the calculation has caused a significant decrease in steps per second. This resulted in a sharp increase in training time, therefore observation normalization was not used.

4.3 Bootstrapped DQN

The DQN algorithm from CleanRL is also utilized to implement Bootstrapped DQN. Contrary to DQN, now five heads with separate Q-networks are used, and the weights for the networks are randomly initialized. This is crucial as the random initialization of weights has shown to be adequate to drive more exploration [20]. For *train_frequency*, which was also set to 10, one of the heads is chosen randomly and is used to pick the action that presents the highest Q-value possible at the moment within that head. The previous experiences are recorded in a replay buffer, where it is bootstrap sampled for each head and the samples are used to be trained during the training phase.

In the original paper [20], a bootstrap head is shown to be selected every episode, instead of every *train_frequency* steps. The use of *train_frequency* instead of episodes to choose a bootstrap head is an intentional choice for this paper as the episodic returns obtained from choosing a bootstrap head every *train_frequency* steps, were higher.

Another difference from the original paper is that the original paper suggests the use of a masking distribution, while the current implementation does not apply masking to decrease complexity.

5 Results

The results section presents an analysis of the training outcomes for three algorithms: Epsilon-Greedy, RND, and Bootstrapped DQN. The evaluation encompasses episodic returns and training times to provide insights into the performance and efficiency of each algorithm. Furthermore, a comparative examination is conducted to assess the robustness of these algorithms across different maps in the CARLA environment.

5.1 Training

After each algorithm is trained on CARLA Town03 with 500 thousand steps to learn, the episodic returns and training times are compared. The training times did not differ significantly, and all of the algorithms have approximately needed twenty and a half hours to train on DelftBlue, which is demonstrated in Figure 3.

The expectation for the episodic returns was that the episodic returns of Bootstrapped DQN and RND would outrun the Epsilon-Greedy implementation because, during the training on the CarRacing environment, Bootstrapped DQN

and RND have returned significantly better episodic returns than Epsilon-Greedy. The comparison of RND and Epsilon-Greedy alone is displayed in Figure 4.

Although Bootstrapped DQN has satisfied the expectations, RND did not have better episodic returns for the Town03 environment of CARLA, shown in Figure 3.

This could be caused by several factors such as the training environment, specific implementation details of RND, the hyperparameters of the model, and training time. The differences in implementation details of the original paper [21] and the implementation of this experiment have been detailed in Section 4. A single one or a combination of these differences could explain the inferior ability to learn for the implementation that used RND.

Secondly, the learning environment in CARLA could have considerably more novel states than CarRacing due to a more complicated map with other car entities simulating traffic. Since there are many more novel states, RND might have incentivized the agent to explore these states, instead of exploiting the already explored paths that bring rewards.

One other reason for the poor performance of RND could be the fact that hyperparameters could not be optimized for CARLA. Since it requires significant computational resources to train on CARLA, it was not possible to try and utilize many different combinations of hyperparameters such as *rnd-weight* or *learning-rate-rnd*. More optimized hyperparameters could lead to more efficient exploration which would cause better returns.

Increasing training time could also help the model learn. With more steps given to the model to learn, the model might gain further capability by exploring novel states and collecting intrinsic rewards.

Bootstrapped DQN, contrary to RND, has produced considerably higher returns in CARLA than Epsilon-Greedy, showing improved ability in learning. This was aligned with the performance previously recorded on CarRacing, where Bootstrapped DQN reached the highest levels of returns around 50k steps earlier than RND and around 600k steps earlier than Epsilon-Greedy.

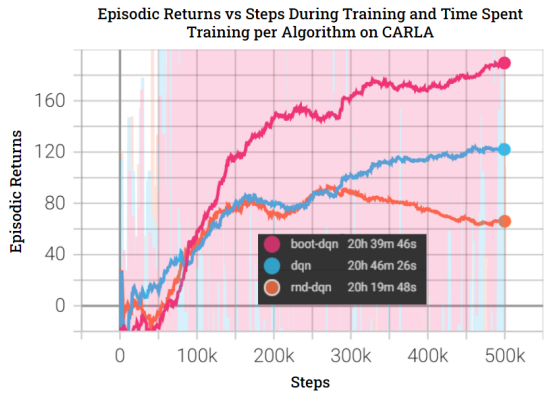

Figure 3: Graph showing episodic returns vs steps during training as well as time spent for training per algorithm on CARLA

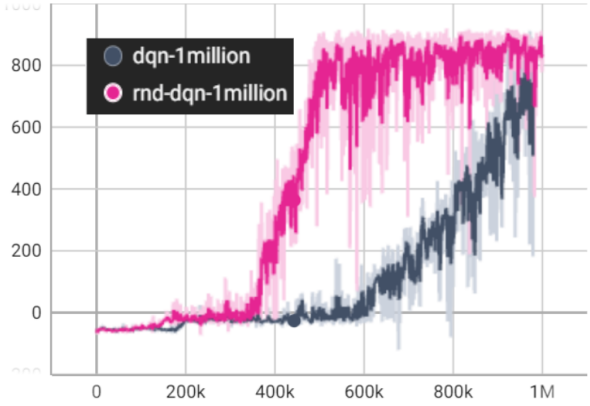

Figure 4: Graph showing episodic returns vs steps during training on CarRacing for DQN (uses Epsilon-Greedy) and RND-DQN

5.2 Robustness

The robustness tests were done on three different maps, Town03, Town04, and Town05 on CARLA. The models were evaluated using an evaluation function that drove the simulated cars in the environments and collected rewards. The mean rewards accumulated by each exploration method for each of the maps are displayed in Figure 5. Bootstrapped DQN has outperformed Epsilon-Greedy and RND by a large margin.

RND, as shown in Figure 5 performed the poorest among the three, consistent with the episodic returns obtained from the training phases. Although the obtained mean episodic returns are low, the standard deviation of episodic returns for RND was significantly high on each of the maps, demonstrated in Figure 6. This leads to the conclusion that the episodic returns of the model that was trained with RND were differing significantly, showing insufficient learning.

Another intriguing result is the serious increase in performance on map Town04 for all algorithms. This might be due to the shape of Town04, in which most of the road length is covered in a figure-8-shaped highway, displayed in Figure 7. While Town03 and Town05 are both urbanized with highways around the maps, there are also many opportunities for the agent to engage in inner-city driving with intersections and lights, which could slow learning.

Town04 also contributed to the most standard deviation in episodic returns on evaluation. The fact that the agent was obtaining very high and very low results might also be caused by the terrain in which the agent was driving at. The model might have had significant ease at driving on the highway while struggling in a complex intersection scenario.

6 Responsible Research

The field of autonomous driving holds tremendous potential for enhancing road safety and transportation efficiency. However, it also raises ethical concerns regarding the well-being and security of the individuals involved [3]. In this paper,

[3] https://www.moralmachine.net/

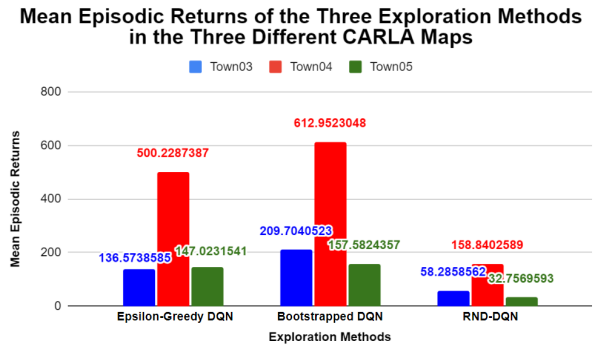

Figure 5: Graph showing mean episodic returns of the three exploration methods in three different CARLA maps

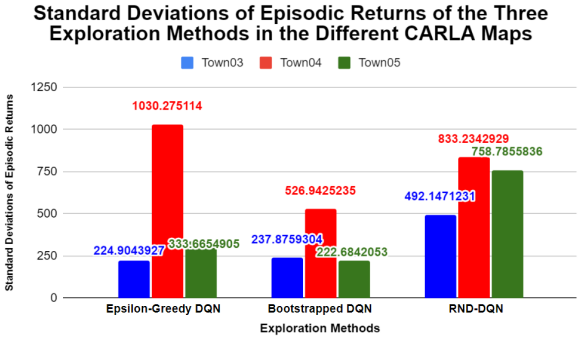

Figure 6: Graph showing standard deviations of episodic returns of the three exploration methods in the different CARLA maps

the examination of reinforcement learning techniques within simulated environments is concentrated on, without involvement in real-world testing or the deployment of autonomous vehicles. Therefore, ethical considerations associated with real-world deployments, such as potential harm to human lives, do not directly apply.

Furthermore, reproducibility constitutes a fundamental principle of scientific research, enabling the validation and verification of findings. In the domain of deep reinforcement learning, achieving exact reproducibility poses challenges due to the stochastic nature of neural networks. Although this is an inherent limitation, providing comprehensive details of the experimental setup, methodologies, and hyperparameters to facilitate reproducibility. While exact replication of results may prove challenging due to the stochasticity inherent in neural networks, it is the expectation that other researchers employing similar methods, models, and hyperparameters should obtain similar trends and outcomes.

It is important to acknowledge that the implementation of the algorithms mentioned in this paper might vary when attempting reproduction, potentially leading to slight differences in the obtained results. The complex nature of deep reinforcement learning algorithms often involves numerous design choices and parameter settings that can influence the outcome. Factors such as hardware configurations or software versions affect the overall behavior and performance of

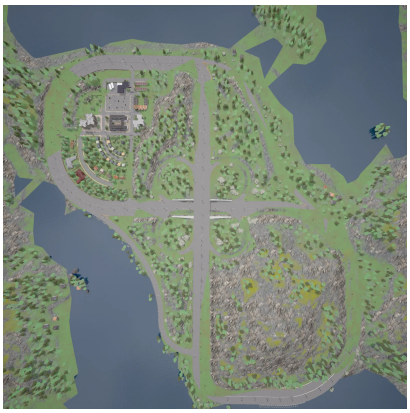

Figure 7: Town04 aerial view from CARLA

the models. Therefore, while striving for reproducibility, it is crucial to consider the inherent variability in results that may arise due to implementation discrepancies.

7 Limitations

There are several limitations associated with this research that should be considered. Firstly, the difficulty in tuning hyperparameters is a notable constraint, primarily due to the extensive time required to train models in the CarRacing and CARLA environments, with CARLA being particularly time-consuming. This limitation is specifically displayed in the under-performance of RND in a more complex environment like CARLA, as the setup for the network and the hyperparameters could have been optimized. The lack of time and computational resources restrict the thorough exploration and optimization of hyperparameter settings, affecting the overall performance and generalizability of the models.

Secondly, the limited time available for training the models in different environments in CARLA poses a constraint on the robustness of the models. Due to time constraints, it was not possible to cover a wide spectrum of available maps and environments. Consequently, the trained models may lack exposure to certain scenarios, which could limit their ability to generalize effectively to unseen environments.

Subsequently, the models have not been trained on the different sensory options *gym-carla* offered. Since the bird-eye view allowed for the simplest representation of the environment, a front camera that had a high-resolution representation of the environment, as well as the Lidar sensor data, was not used in training. Using these different environments in training would also contribute to the robustness of the models created.

Lastly, it is important to acknowledge that the trained models have not been tested on many different environments. Although the models showed promising performance in the tested scenarios, which are Town03, Town04, and Town05, CARLA's large offering of maps and environments could be utilized further. Evaluating the models on a larger set of diverse environments, encompassing various road conditions, weather conditions, and traffic patterns, would provide a more comprehensive understanding of their capabilities and limita-

tions.

Despite these limitations, this research offers valuable insights into the application of deep reinforcement learning in complex driving scenarios. Future research endeavors should aim to address these limitations by dedicating more time and resources to hyperparameter tuning, training models on a broader range of environments, and thoroughly testing the models on diverse and challenging scenarios.

8 Conclusions and Future Work

This research paper has aimed to answer "How does epsilon-greedy, random network distillation, bootstrapped DQN affect training and the robustness of final policies under various testing conditions in autonomous driving?" To achieve this, implementations of Epsilon-Greedy, RND, and Bootstrapped DQN have been trained on CARLA and evaluated for robustness. Bootstrapped DQN has produced marginally better episodic returns compared to the other two, while RND's implementation struggled to learn.

This incapability might have been caused by many different factors such as differences between the original paper that proposed RND [21], training environment, under-optimized hyperparameters, and lack of necessary time to train.

Future improvements should include further hyperparameter optimization not only for RND but also for the DQN implementation that uses Epsilon-Greedy as well as Bootstrapped DQN. The agents should also be trained in more maps on CARLA to be accounted for the differences in learning environments. CARLA environment parameters could also be changed during training to increase robustness. The changed parameters could include the number of vehicles, the number of walkers, and the observation range.

Due to the fact that Bootstrapped DQN was still showing an upwards trend in learning, it could be assumed that the learning has not peaked yet. This leads to the conclusion that the models could be trained with more than 500k steps.

Finally, more algorithms could be trained on CARLA using different exploration methods besides Epsilon-Greedy, Bootstrapped DQN, and RND. NoisyNets [9] and Diversity-Driven Exploration [11] could be candidates for further research.

References

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016.

[3] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation, 2018.

[4] Jianyu Chen. gym-carla. https://github.com/cjy1992/gym-carla, 2020.

[5] Jianyu Chen, Shengbo Eben Li, and Masayoshi Tomizuka. Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning, 2020.

[6] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Model-free deep reinforcement learning for urban autonomous driving, 2019.

[7] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[8] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. 2017.

[9] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration, 2019.

[10] Rodrigo Gutiérrez-Moreno, Rafael Barea, Elena López-Guillén, Javier Araluce, and Luis M. Bergasa. Reinforcement learning-based autonomous driving at intersections in carla simulator. *Sensors*, 22(21):8373, 2022.

[11] Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, and Chun-Yi Lee. Diversity-driven exploration strategy for deep reinforcement learning, 2018.

[12] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[13] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, jan 2021.

[14] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey, 2021.

[15] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

[16] Haochen Liu, Zhiyu Huang, Jingda Wu, and Chen Lv. Improved deep reinforcement learning with expert demonstrations for urban autonomous driving, 2022.

[17] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance, 2022.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. December 2013.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[20] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. February 2016.

[21] Ian Osband, Benjamin Van Roy, Daniel Russo, and Zheng Wen. Deep exploration via randomized value functions. March 2017.

[22] Dean A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. San Francisco, CA: Morgan Kaufmann, 1989.

[23] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[25] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[26] Christopher J. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

9 Appendix

9.1 Hyperparameters

The hyperparameters used in the experiments are shown below in 1.

total-timesteps: This specifies the total number of timesteps or steps the agent will take during training.

train-frequency: This hyperparameter controls how frequently the agent performs a training update.

buffer-size: This determines the size of the replay buffer, which is a storage mechanism used to store and sample past experiences for training.

gamma: This is the discount factor used in the computation of the cumulative discounted rewards. It determines the importance of future rewards relative to immediate rewards. A value of 0.99 means that future rewards are weighted more heavily than immediate rewards.

tau: This hyperparameter controls the rate at which the target network parameters are updated.

learning-starts: This specifies the number of timesteps the agent will take before starting the training process.

start-e and **end-e**: These specify the starting and ending values of the exploration rate (epsilon) used in epsilon-greedy action selection.

target-network-frequency: This determines how often the target network is updated with the primary network's parameters.

learning-rate-rnd: This is the learning rate used for updating the parameters of the RND network during training. It controls the step size taken during each parameter update of the RND network.

rnd-weight: This specifies the weight or scaling factor applied to the RND intrinsic reward. It determines the relative importance of the RND intrinsic reward compared to the extrinsic reward. A higher **rnd-weight** assigns more importance to the RND intrinsic reward, encouraging exploration.

exploration-fraction: This determines the fraction of total timesteps during which the exploration rate (epsilon) decreases.

batch-size: This is the number of experiences sampled from the replay buffer for each training update.

num-heads-bdqn: This hyperparameter specifies the number of the bootstrap heads used in the Bootstrapped DQN.

Hyperparameter	Value
total-timesteps	500k
train-frequency	10
buffer-size	50k
gamma	0.99
tau	1
learning-starts	10k
start-e	1
end-e	0.05
target-network-frequency	1000
learning-rate-rnd	0.001
rnd-weight	0.2
exploration-fraction	0.5
batch-size	128
num-heads-bdqn	5

Table 1: Hyperparameters and Values