Effects of action space discretization and DQN extensions on algorithm robustness and efficiency

How do the discretization of the action space and various extensions to the well-known DQN algorithm influence training and the robustness of final policies under various testing conditions?

Mehmet Alp Sozuduz[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 23, 2023

Name of the student: Mehmet Alp Sozuduz
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan, Moritz Zanger, Elena Congeduti

Abstract

Reinforcement Learning (RL) has gained attention as a way of creating autonomous agents for self-driving cars. This paper explores the adaptation of the Deep Q Network (DQN), a popular deep RL algorithm, in the Carla traffic simulator for autonomous driving. It investigates the influence of action space discretization and DQN extensions on training performance and robustness. Results show that action space discretization enhances behaviour consistency but negatively affects Q-values, training performance, and robustness. Double Q-Learning decreases training performance and leads to suboptimal convergence, reducing robustness. Prioritized Experience Replay also performs worse during training, but consistently outperforms in robustness testing, reward estimation and generalization.

1 Introduction

Ever since the first deep Reinforcement Learning algorithm [12] was demonstrated playing basic Atari games, Deep Reinforcement Learning has been at the forefront of the artificial intelligence world. Since then they started to appear in various parts of human life from playing games, and simulating behaviour to fueling auto-driving vehicles, each bringing unique opportunities and challenges. One such challenge is the categorization of the action space, which actions the agent can take within the world, into two: continuous and discrete. While discrete action spaces can be found in more simplistic environments such as board/video games, more realistic situations such as robotic movement, or simulations utilize continuous action spaces [17]. However, not all algorithms, Deep Q Network (DQN) for instance, could work with a continuous actions space and needs the space to be converted into a discrete space [21]. How many distinct actions could the space be divided into? One of the aims of this research is to answer this question by comparing different levels of discretization of an auto-driving vehicle in a traffic simulator called, Carla [3]. Through this research, the effects of discretization could be explored which in turn would lead to RL algorithms that could both respond to different situations they are presented with and take an efficient path in their training.

The other facet of this research is to check the effect of DQN extensions, such as Prioritized Experience Replay and Double Q-Learning, on the efficiency and robustness of the algorithm when applied to Carla. To explain these extensions, the RL algorithm utilizes the data, such as the state of the world, as a way to decide their next action. These data, referred to as experiences, can be stored and reused to increase the efficiency of the algorithm and reduce training times. Conventionally an experience is drawn with uniform sampling and utilized. Prioritized Experience Replay proposes to add prioritization to the experience sampling process such that experiences that teach more to the agent are utilized more [15]. The other extension stems from the fact that as the Q-Learning algorithms utilize the same weights for both action selection as well as evaluation, it can lead to more optimistic value estimations. This overestimation has the potential to negatively affect the performance of the agent as well as increase the risk of divergence [18]. To counter this, this process is decoupled into two with each action selection and evaluation utilizing their own Q-estimators [19]. These are just two of the many extensions proposed over time to improve DQN. The two were chosen as they are feasible to be implemented in two months, the duration of the project, and have been shown to have a noticeable effect on training times and robustness.

"How do the discretization of the action space (as required for several RL algorithms) and various extensions (e.g., Prioritized Experience Replay, Double Q-Learning) to the well-known DQN algorithm influence training and the robustness of final policies under various testing conditions?" is the question that will be answered in the following sections. This can be decoupled into three distinct steps:

- **Adaptation into a relevant problem**: The problem chosen for this project is auto-driving in a simulated driving environment of Carla. It allows the agent to train in a relatively complex environment with other actors. It contains multiple different maps to test the agent under differing conditions and offers functionality to control the action space. All of these reasons made it a good candidate for this project.
- **Comparison of training performance**: One of the objectives of the research questions is to understand the influence of these different methods on training efficiency. In the context of this research, *efficiency* will refer to several factors including: how fast the agent's performance stabilizes, and how Q-values, the agent's estimation of its performance, compare to the rewards it achieves each episode.
- **Comparison of robustness under testing conditions**: The other objective is to see the relationship between the proposed methods and the robustness of the algorithm. Robustness refers to the ability of the agent to adapt to varying conditions, which includes situations it may not have encountered during its training period. For this paper, the robustness will be tested by running the agent in different maps Carla offers.

2 Background Information

This section explores the literature on Deep Q Networks (DQN) and outlines the information necessary to understand the paper. It starts with describing the inner workings of the DQN algorithm in 2.1. Then discusses the problem of action space discretization in 2.2 and explains the two DQN extensions, Double Q-Learning and Prioritized Experience Replay, in 2.3.

2.1 Deep Q Network (DQN)

In the current Reinforcement Learning problem, the agent has a set of actions that it can select at each step, and selecting each action leads to a different step. As the observation of each step is an image of the agent's surroundings, and the

agent mainly relies on that image to make its decision, it is possible to model this problem as a Markov decision process (MDP). This gives us the ability to apply algorithms designed to tackle common MDPs in the current environment [12].

Deep Q Network is a Reinforcement Learning algorithm that combines Q-Learning with neural networks to have an agent that can adapt to environments by approximating an action-state value [13]. It inputs the observation, the current state of the world, to its internal neural network. This network outputs values for each of the actions that the agent can do at that given state, and then one of these action-value pairs is selected to determine the next action of the agent. The following subsections will describe the important internal mechanism of the DQN algorithm.

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha \cdot (r + \gamma \cdot max_{a'} Q(s',a')) \quad (1)$$

Q-Learning

Q-Learning is an off-policy temporal difference algorithm to find the optimal policy by updating the state-action value function (Q-function) at every step using the Bellman Optimality equation [9]. This is achieved by a data structure called Q-table, which maps each state-action pair to an expected reward, known as Q-values. The Bellman Equation describes how to update this Q-function after each iteration, it is shown in Equation 1 [9]. Translating this equation, it means that the value of the state-action par s, a at time step t can be determined by the estimated best optimal future value. r refers to the reward from taking an action, α refers to the learning rate of the algorithm and γ is the discount factor that makes rewards lose value over time so that immediate rewards are prioritized. As the agent tries out different actions, these Q-values converge and become accurate at describing the potential reward an action may give in a particular state.

Epsilon-Greedy Exploration Strategy

At each step, an agent can choose between two categories of actions: exploitation and exploration. Exploitation is when the agent utilizes their previous knowledge to choose the action that it believes will maximize the reward. On the other hand, exploration means the agent will take a random action instead. These two categories are important for the agent to learn new strategies and optimize the knowledge it has. The probability of choosing one of these is determined by a parameter ϵ. At the beginning of each step, a random number is generated and if the number is less than ϵ, the agent will choose exploration and exploitation otherwise. In the beginning, the agent has no knowledge of the environment so exploration is prioritized. As the training progresses, the agent starts to use exploitation more and more and this change is known as *epsilon-decay*.

Experience Replay

As the agent continues its training, it stores the experiences inside a memory called a replay buffer. During learning, these experiences are randomly sampled to contribute to the Q-value updates [13]. This introduces nonsequentiality to the training and reduces the bias that may have been caused by the current episode. It increases the data efficiency as the agent can learn multiple times from a single experience. Furthermore, it increases the robustness of the algorithm as the agent becomes less sensitive to sub-optimal experiences.

Q Network & Target Network

DQN utilizes a neural network called Q Network instead of a Q-table to map a state-action pair to the expected reward. The input to his neural network is the observation space and the output of the neural network is the Q-values for each action. At each step, if *exploit* is chosen, the maximum valued action is selected by the algorithm.

$$L_t(\theta_t) = E_{s,a,r,s' \ \rho(.)}[((r+\gamma \cdot max_{a'} Q(s',a';\theta_t^-) - Q(s,a;\theta_t))^2] \quad (2)$$

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(s,a;\theta_t)) \cdot \nabla_{\theta_t} Q(s, a:\theta_t) \quad (3)$$

$$Y_t^Q = r + \gamma \cdot max_{a'} Q(s',a';\theta_t^-) \quad (4)$$

DQN has another internal neural network, identical to the Q Network, called *Target Network*. Differing from the Q Network, this neural network is only updated every C number of steps, where C is a customizable parameter. These networks utilize the loss function in Equation 2 [13] and the update equation in Equation 3 and Equation 4 to internally train the neural networks. In the equations, θ_t refers to the set of weights the Q Network has, while θ_t^- is the set of weights of the Target Network.

2.2 Action Space Discretization

The first part of the research is concerned with discretizing, the process of dividing a continuous domain into a finite amount of values that can represent that domain fully. While there have been proposals such as Actor-critic and Normalized Advantage Functions (NAF) [6] to make a DQN algorithm compatible with a continuous action space, this research aims to convert the action space of Carla into a discrete one rather than utilizing one of these techniques.

The tool we utilize to train the agent on the Carla simulator, *gym-carla* [1], provides the ability to customize the action space by allowing users to select between continuous and discrete. In the case of discrete action spaces, it supports further customization by specifying the values of acceleration, how fast the car can go, and the angle of steering, how sharp a turn the car can make. In other words, the RL algorithm's output will be one of these values. By utilizing different values for these parameters, it is possible to see the influence of action space discretization on the training performance as well as the robustness of the DQN algorithm.

There has not been a lot of research that has directly varied the possible action outcomes to see its influence on algorithm performances within the traffic domain. Therefore, it is difficult to speculate the results as well as the impact of the amount of discretization. On one hand, the more choices the algorithm has means it should be more suitable to deal with different situations. On the other hand, being able to make more movements might delay the learning process and even negatively affect performance since safety is more important than being able to go when it comes to traffic. The results

from this paper should shed light on how far should the discretization be taken for it to accurately represent a continuous domain and whether discretization leads to better adaptability under test conditions.

2.3 DQN Extensions

The second part of the paper is regarding two DQN extensions, namely Double Q-Learning [19] and Prioritized Experience Replay [15]. These extensions have been proposed over the years to improve various points of the traditional DQN algorithm. Following subsections will give further information about these extensions.

Double Q-Learning

While DQN has shown to be capable of learning, it has some faults. One of these faults is the fact that it tends to overestimate action values in some cases, generating over-optimistic results. This not only negatively affects the performance [19], but could also cause divergence which in turn would render the network model useless. The main reason for this is that the maximum operator in DQN utilizes the same values to both select and evaluate an action. This makes it more likely to generate positive outcomes, which is a bad thing as it introduces overestimation to the model.

$$Y_t^Q = r + \gamma \cdot Q(s', max_{a'} Q(s', a'; \theta_t); \theta_t) \quad (5)$$

$$Y_t^{DoubleQ} = r + \gamma \cdot Q(s', max_{a'} Q(s', a'; \theta_t); \theta'_t) \quad (6)$$

Double Q-Learning, is an extension to the standard DQN algorithm to help with this issue. The idea behind it is to decouple the max operation into two: action selection and action evaluation. This can be achieved through utilizing two different Q Networks with each responsible for one of the operations. The equations Equation 5 and Equation 6 [19] show the difference between normal DQN and DQN with Double Q-Learning when updating the network gradient. Double Q, utilizes a different set of weights, denoted as θ'_t, in its update function to independently evaluate the action taken. As the DQN algorithm already comes with an additional network called *Target Network*, it is possible to utilize this network for the action evaluation. This is also what is recommended in the original paper for the Double Q-Learning [19].

The way to test the effect of this extension is to compare a DQN implementation without it and a DQN with it. Nothing should be changed and one important thing to note is that the seeds should also be the same to eliminate any unfairness due to randomness. While the literature agrees on the fact that Double Q-Learning lowers the overestimation, there has not been explicit research on the domain of traffic simulators [4]. These simulators are different in the fact that they have lots of moving parts that the DQN agents are in no control of. This means that due to lucky events increasing rewards in some episodes could reinforce unwanted behaviours and since the action selection and evaluation are done by the same network, this could aggregate into the model not performing as well as it can. Double Q-Learning could be used to disincentivize this over-eager behaviour as there is a different network that would evaluate the action.

Prioritized Experience Replay

Experience Replay is an integral part of the DQN algorithm. It is a process in which the agent stores the previous data it has seen in a memory buffer and then *randomly* samples this data during the learning process to improve its decision-making [12]. While this method has been shown to have improved the performance of the DQN algorithm, it is still not optimal in the fact that the agent can still sample bad experiences and learn nothing from them [13].

Prioritized Experience Replay, as its name implies, assigns importance to each experience that is stored within it to improve performance. This prioritization is often determined by temporal difference (TD) error, which is the difference between the actual reward and the estimated reward by the Q Network [15]. A high TD error means that in that experience an event that greatly surprised the agent has happened. This means that this is an event that could teach the agent to do or avoid something. The extensions achieve reduced training times and improved performance by compelling the agent to not disregard rare and valuable experiences, which could have been overlooked in uniform sampling.

One crucial faucet in Prioritized Experience Replay is to tune parameters α and β, which determine the degree of sampling skewing. α determines how aggressive the prioritization of the experiences should be whereas β determines the amount of correction applied after sampling [15]. This correction is there to help reduce bias that may have been caused by the difference in actual distribution and the sample's own distribution. In other words, the higher these values are, the less random the sampling process is. If these parameters are too low, then this is no different from normal experience replay, and if they are high then the agent can only focus on these unexpected behaviours and over-tune itself. This is especially true within Carla as the behaviour of other cars could directly create these unlikely situations. If these are learned to the appropriate degree, the agent can adapt to the complex nature of traffic much easier. All in all, this paper will utilize the previously explained methodologies to test the influence of experience prioritization on training performance as well as robustness.

3 Methodology

Since the research question is divided into two parts, it was advised to also divide the research itself into said parts. This section will illustrate the chosen methodology for each of these parts.

Due to technical limitations during the development of the algorithm as well as extensions, Carla was not utilized in the preliminary experiments. Instead a similar, in terms of input as well output, yet much more computationally cheap environment is utilized. This environment is the *CarRacing*[1] environment from the Gym framework that OpenAI has created. However, while this environment can indicate that the algorithm can learn through images and apply that to driving a car, it does not give insight into the algorithm's performance in realistic traffic. For this reason, for comparing the training as well as robustness Carla is preferred.

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

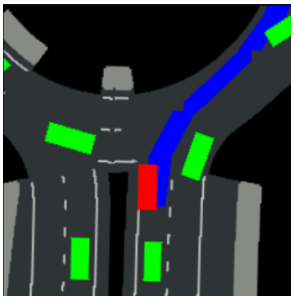

Figure 1: The birdeye view *gym-carla* provides to the algorithm.

Through the repository *gym-carla* [1], it is possible to seamlessly integrate the algorithm developed with the Gym environment into Carla. This repository converts the interactions that the Carla simulator has to a format compatible with the Gym framework. Furthermore, it offers the option to customize the action space, which makes it possible to answer the *action space discretization* part of the research question. An example of the algorithm will get as image input from this wrapper is given in Figure 1.

The following subsections explain the exact methods that will be used to conduct the experiments.

First part: action space discretization and its effect on training and robustness.

1. Adapt DQN into Carla's environment.
2. Train DQN with by varying the action parameters:
   (a) **steering angle**: the angle at which the vehicle can be rotated in a single frame.
   (b) **acceleration rate**: the velocity at which the vehicle will be sped up or slowed down by in a single frame.
3. Compare the various training data such as how fast the rewards converge to a value, and how Q-values evolve over training.
4. Run both versions of these algorithms by changing the map in which the algorithm operates. Log the results to check the differences that will occur.

Second part: DQN extensions and their effect on training and robustness.

1. Utilize the implemented DQN algorithm within the first part.
2. Add Prioritized Experience Replay into the DQN algorithm.
3. Compare the training using by using the workflow in the first part of the bare-bone implementation and DQN with extension.
4. Do a similar robustness test to the action space discretization between DQN with extension and normal DQN.
5. Implement Double Q-Learning into the DQN algorithm.
6. Run the tests for training times as well as robustness.

The workflow for testing the training performance and robustness are the same to ensure consistency within the research.

Name of the map	**Carla Description**
Town03	A larger, urban map with a roundabout and large junctions.
Town04	A small town embedded in the mountains with a special "figure of 8" infinite highway.
Town05	Squared-grid town with cross junctions and a bridge. It has multiple lanes per direction. Useful to perform lane changes.

Table 1: Carla maps and their descriptions [3]

4 Experiments

The DQN algorithm has many different implementations. The one that we have used for this paper is taken from ClearnRl by Shengyi Huang [8]. It is a repository containing *clean*, which means maintainable and easy-to-alter, implementations of many reinforcement learning algorithms with DQN being one of them. However, the network architecture of the internal Q Network was not suitable for learning from the image format that Carla produces. Therefore, the network architecture was replaced with the one used in the Deep Mind Atari paper [12]. The common hyperparameters that were constant throughout all of the experiments are given in Appendix A.

For the environment, Carla simulator [3] with version *0.9.13v* was used. The algorithm was adapted to this environment by utilizing a tool called *gym-carla* [1]. This is a tool developed to make Carla compatible with the Gym framework. The parameters this tool takes are given in Appendix B. As the hardware requirements for Carla are high, DelftBlue [2], the supercomputer cluster under TU Delft. One constraint of Delft Blue was that it only allowed training for 24 hours, and only 500k steps could be completed within this duration. Checkpointing was employed during the initial Carla tests; however, continuing from a checkpoint lengthened the experiment time by a factor of four to five. Due to this reason, 500k steps were used to train each of the agents.

For testing the robustness of the algorithm, we elected to change the maps that the agent will be tested in. The maps and their descriptions are given in Table 1. These maps were chosen as they present a multitude of real-life situations to test the agent against. All of the agents are initially trained in *Town03*. Then for evaluation, agents ran 10 episodes in each of these maps with each episode having differing initial conditions. For all of the models, both training and robustness testing, 0 was selected as a seed to eliminate any unfair advantage randomness may cause.

4.1 Action Space Discretization

In Carla, the action space is a combination of two parameters: acceleration value and steering angle. The acceleration values refer to the change of speed the agent's vehicle can have and the steering angle refers to the degree of turn the vehicle can make. Therefore, to test action space discretization, three different levels of discretization were selected at 3 values, 5 values, and 7 values for each of these parameters. These are given in Table 2. While the domain of the action space remains the same, by adding more values the degree of control

Number of values	Acceleration Value	Steering Angle
3 values	-3.0, 0.0, 3.0	-0.2, 0.0, 0.2
5 values	-3.0, -1.5, 0.0, 1.5, 3.0	-0.2, -0.1, 0.0, 0.1, 0.2
7 values	-3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0	-0.2, -0.133, -0.066, 0.0, 0.066, 0.133, 0.2

Table 2: Acceleration and Steering values for different agents

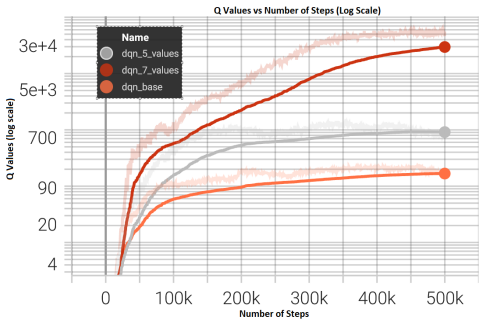

Figure 2: Graph showing the evolution of Q-values per agent during training

increases.

Training Performance

For comparing the training performance, changes in Q-values and episodic rewards were plotted for each step. These are highlighted in Figure 2 and Figure 3 respectively. For Q-values, it can be seen that they increase exponentially, as the graph is in logarithmic scale, with each increment of discretization. Another observation is that Figure 3 shows that an increase in the level of discretization delays the training and is consistent with results found in research by Yunhao Tang [17]. One explanation of these results could be about the equation in Figure 2, it shows that the loss is directly calculated by estimating the maximum amount of rewards the agent can get if it follows an optimal policy for the **rest** of the episode. As the level of discretization increases, the algorithm is more likely to find higher-rewarding actions resulting from the high-grained control. These compound over time and cause the algorithm to overestimate during the training. Another possible way to explain this is that the algorithm

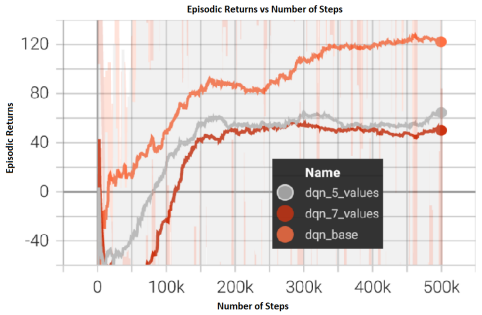

Figure 3: Graph showing the evolution of episodic returns per agent during training

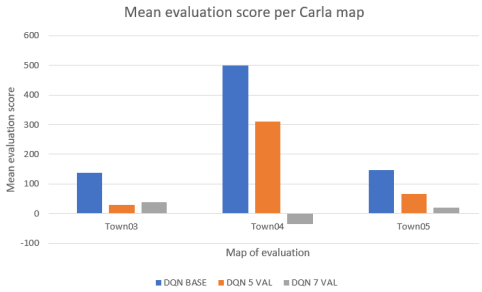

Figure 4: Graph showing the mean evaluation scores per agent across Carla maps

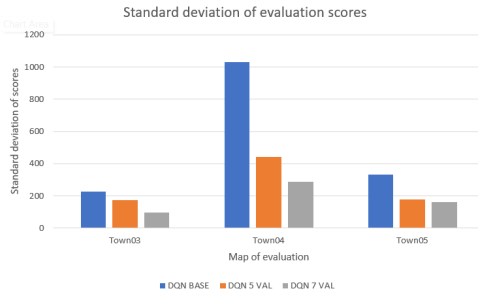

Figure 5: Graph showing the standard deviation of evaluation scores per agent across Carla maps

may need more time as well as data to adapt to the expanded output neurons and to properly converge its Q Network gradients. These results suggest that, at least for the Carla simulator, discretization increases overestimation and causes a performance decrease for training.

Robustness Testing

For testing the robustness of the final agent, three different maps in Table 1 were utilized. This was to ensure that the agent would encounter a different environment that it has no knowledge of. Each agent was run ten times per map with different starting conditions and the reward for each episode was collected. Figure 4 shows the mean evaluation score for each map. This graph suggests that the level of discretization lowers the overall expected reward from evaluations. This is in correlation with the training performance shown in Figure 3. The standard deviation for these scores is depicted in Figure 5. The graph illustrated a decrease in the standard deviation of scores as the discretizations increase. One reason for worse evaluation performance at higher discretization steps could be the divergence of Q-values in training. The previous section showed that a higher level of discretization correlated to an exponential increase in Q-value estimations. This may mean that agents with highly discretized spaces may not have had the opportunity to converge to an optimum neural network. Another potential explanation is that the simulation prioritizes bigger behaviour, larger turns or higher acceleration. This would mean lower discretization that only offers these larger movements would expect overall higher rewarding, but more varied, behaviour and this is supported by the standard deviations shown in the graphs. We can infer that

with the increase in the level of control high discretization provides, the agent's behaviour becomes more consistent, as it can choose more values rather than just backwards-stop-forwards.

4.2 DQN Extensions

In order to compare the influence of DQN on training performance, two DQN agents each utilizing one extension were trained in Carla. For the Double Q extension, the implementation of Van Hasselt [19] was utilized. In this architecture, the target network of the agent is utilized as an objective judge to evaluate the actions. As for Prioritized Experience Replay (PER), a Segment Tree was used as a Priority Queue with *TD loss* as keys and experiences as values. For the two hyperparameters PER introduces, a value of 0.2 was selected for α and a value of 0.6 was determined as the initial value of β. As the training progress, β slowly decays to 0 because the need for bias correction decreases.

Training Performance

Figure 6 describe the reward all three agents have gotten for each step and figures 7 and 8 illustrate the evolution of Q-values and TD loss respectively. It can be seen that base DQN converges to a higher reward value as training comes to an end. An interesting observation is that Figure 8 shows the agent with PER having a smaller TD loss compared to other agents.

These agents having worse performance does not agree with the original research papers [19] [15] for these extensions as both show an increased training performance compared to a base DQN instance. One explanation for the decrease in performance due to Double Q could be due to underestimation. Zhizhou Ren [14] describes that in complex environments, Double Q-Learning could underestimate the Q-values during training and converge to a less optimal point. This can be seen in other figures as Double Q consistently has lower Q-values and TD loss compared to base DQN, showing that the algorithm is better at estimating the Q-values like described in the original paper [19].

PER having significantly less TD loss is consistent with Tom Schaul's research [15] as the agent can train more on rare situations, it is better able to anticipate the reward that will come from it. For the decrease in training performance, Jincheng Mei [11] describes two potential limitations: outdated priorities and insufficient sample coverage. The first stems from the inability of the agent to update the priorities inside the replay buffer at each step and the second is observed when the replay buffer is not large enough compared to the sample space [5]. Carla environment is especially prone to insufficient sample coverage as it is a complex environment with many moving parts. So while the experience with high TD loss could have been sampled, the more normal experiences could be underrepresented due to prioritization.

Robustness Testing

Results of robustness testing are depicted in Figure 9 and Figure 10. These show that the agent with PER has higher mean evaluation scores and lower variation, which indicates that it outperforms other agents in robustness and agrees with [15]. On the other hand, Double Q-Learning under-performs

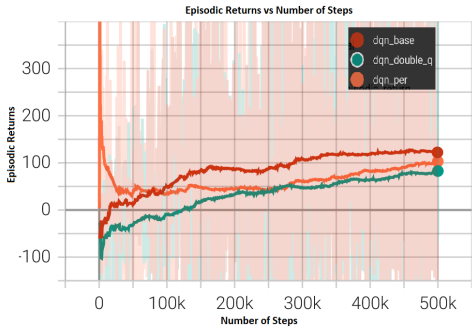

Figure 6: Graph showing the evolution of episodic returns per agent during training

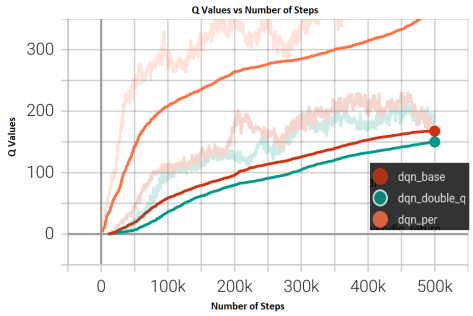

Figure 7: Graph showing the evolution of Q-values per agent during training

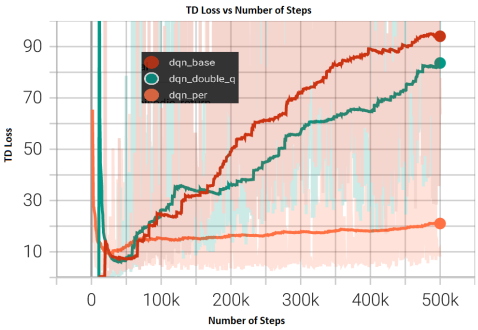

Figure 8: Graph showing the evolution of TD loss per agent during training

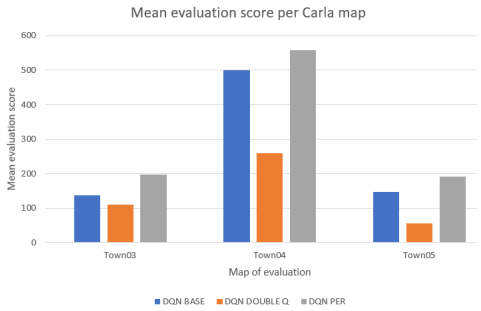

Figure 9: Graph showing the mean evaluation scores per agent across Carla maps

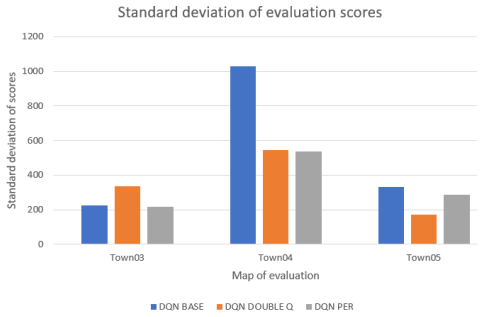

Figure 10: Graph showing the standard deviation of evaluation scores per agent across Carla maps

both agents significantly having a lower mean reward. This follows directly from the less-than-ideal convergence of its training rewards as well. In the discussion for training performance, we have explained underestimation bias as a possible reward for the loss of performance [14]. Another reason could be the Target Network, which was the chosen evaluator for the Double Q-Learning. The second network is supposed to be decoupled from the Q Network responsible for selecting the actions. However, the target network in DQN architecture still has ties to the original network it is just that it lags behind due to having lower frequency updates. While it helped to reduce the overestimation as seen by the decrease in TD loss, it caused the network to get consistently lower rewards. A secondary network which updates every step could overcome this issue. The most likely hypothesis for PER performing better than other agents could be explained through its mechanism of training more on unique experiences. We believe that these situations may have been encountered in other maps as well because traffic movement remains consistent even with different environments. Since an agent with PER has trained more on these unique encounters, it is able to more effectively deal with them in test episodes; hence leading to higher overall evaluation scores.

5 Responsible Research

Autonomous driving is a vital field of the future to improve traffic safety and driving performance. However, many of these autonomous vehicles will eventually hold individuals and this, naturally, creates ethical problems about the quality of any research within this domain. While our research is autonomous driving, the main environment was a simulation and no real people were involved in it. Nevertheless, the results of these experiments may be utilized to develop real-life autonomous vehicle agents, and we advise verifying our results with more time and better hardware to reduce any bias these constraints may have caused us. These constraints are described in more detail in Section 6.

Randomness is an inherent aspect of machine learning, and this paper's deep RL algorithms are no different. Choosing a favourable initial state, selecting between exploitation or exploration within a step, or updating the neural network in the correct direction at the beginning when most of the decisions are arbitrary only contribute to this randomness. To counter this, researchers use what is called a *seed*, essentially a number that can be specified so that the randomness can be reproduced. Another research could utilize the provided seeds in Section 4 to generate a model that could behave similarly.

Unfortunately, this seeding creates another problem such that whether the good or bad results obtained were due to the quality of the seed. To combat this during the research, we selected a *representitive seed*. This is a seed which will perform average results and is able to represent a multitude of seeds. Several runs of the base DQN algorithm at different seeds were done to see the progress of the training graph. Upon these, a seed was selected as the representative. This is not the best solution since the results still come from a single trial rather than aggregated results of multiple. However, within the limited time frame and the heavy hardware constraints Carla has imposed upon researchers, it was not possible to run multiple trials for all different algorithm variants and record the training performance for each of them. For testing the robustness of different algorithm variants, the trained models ran 10 trials, each starting with differing conditions, for each town map to eliminate any bias that could have been caused by good initial positioning.

6 Limitations

One issue we encountered during the research is the high amount of resources and time each experiment took. Carla has high hardware requirements that our computers were not able to run the experiments locally. For this reason, Delft Blue [2], the super-computer cluster of TU Delft was utilized. However, the month of May had the submission deadlines for NeurIPS[2] and this caused Delft Blue to delay the running of experiments. This lateness coupled with the fact that each experiment taking at least 24 hours to conclude and the research had a duration of two months, meant that the agents were not trained to a global optimum. This especially caused some negative bias for agents whose training performance was not as good as the base DQN algorithm. While we have tried to mitigate this limitation as discussed in Section 4, they nevertheless impacted the experiment results. It is advised for another research that would follow this one, to be aware of this limitation and, ideally, train the agents until the upward trends cease.

[2]https://nips.cc/

Another negative impact caused by the limited time and resources is that we were not able to perfectly tune the hyperparameters of the models. This may have negatively affected the overall training and testing performance of the agents. Especially with PER agent, this difficulty in tuning may have caused the results to be negatively biased against this extension, as it is heavily affected by two hyperparameters α and β. Again, it is recommended for future research within this domain, to spend more time tuning hyperparameters to have each agent perform at its best.

7 Conclusions and Future Work

In a nutshell, the paper aimed to answer the question "How do the discretization of the action space (as required for several RL algorithms) and various extensions (e.g., prioritized experience replay, double-q-learning) to the well-known DQN algorithm influence training and the robustness of final policies under various testing conditions?". For the action space discretization, experiments show that the main positive effect of the action space discretization is the increase in the behaviour consistency of the agent. The other metrics, such as Q-values, training performance, and robustness evaluation, negatively correlated to the degree of action space discretization. As for the second part of the research questions, we have concluded that both Double Q-Learning and Prioritized Experience Replay lengthened the training times. In the case of Double Q-Learning, the complexity of the environment has led to the agent converging to a less ideal position and leading to worse performance in robustness testing. In contrast, Prioritized Experience Replay has consistently outperformed other agents during robustness testing. Being able to learn more from unique situations, lead to better reward estimation as well as generalization performance.

All of the figures suggest that both higher discretized action spaces and DQN extensions have reduced performance during the training process. A possible improvement would be to train for longer than 500k steps. Some of the figures still show an upwards trend, and this could lead to them converging to a better optimum. This would possibly lead to agents giving better results in robustness testing. One further research avenue we have theorized would be to merge Double Q-Learning with action space discretization. Experiment results show that higher discretization could lead to gross overestimation of values, as Double Q is mainly utilized for countering this overestimation, its impact could be larger on action spaces with higher discretization. Another research could be done to validate other extensions of DQN, such as Dueling DQN [20]. Or one could also test against Rainbow, a combination of six proposed DQN extensions [7]. A similar methodology to the current paper could be employed to check the training and testing performance of other extensions.

References

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

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

[3] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017.

[4] Eyal Even-Dar and Yishay Mansour. Convergence of optimistic and incremental q-learning. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1499–1506. MIT Press, 2001.

[5] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. *CoRR*, abs/2007.06700, 2020.

[6] Shixiang Gu, Timothy P. Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. *CoRR*, abs/1603.00748, 2016.

[7] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

[8] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *CoRR*, abs/2111.08819, 2021.

[9] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019.

[10] Patrick Mannion, Jim Duggan, and Enda Howley. *An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control*, pages 47–66. Springer International Publishing, Cham, 2016.

[11] Jincheng Mei, Yangchen Pan, Martha White, Amirmassoud Farahmand, and Hengshuai Yao. Beyond prioritized replay: Sampling states in model-based RL via simulated priorities. *CoRR*, abs/2007.09569, 2020.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[14] Zhizhou Ren, Guangxiang Zhu, Hao Hu, Beining Han, Jianglun Chen, and Chongjie Zhang. On the estimation bias in double q-learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and

Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 10246–10259, 2021.

[15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. November 2015.

[16] Marnix Suilen, Thiago D. Simão, David Parker, and Nils Jansen. Robust anytime learning of markov decision processes. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 28790–28802. Curran Associates, Inc., 2022.

[17] Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. *CoRR*, abs/1901.10500, 2019.

[18] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *CoRR*, abs/1812.02648, 2018.

[19] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[20] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

[21] Jie Zhu, Fengge Wu, and Junsuo Zhao. An overview of the action space for deep reinforcement learning. In *Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence*, ACAI '21, New York, NY, USA, 2022. Association for Computing Machinery.

A DQN Parameters

Name of parameter	Value
total-timesteps	500000
learning-rate	2.5e-4
buffer-size	50000
gamma	0.99
tau	1
target-network-frequency	500
batch-size	128
start-e	1
end-e	0.05
exploration-fraction	0.5
learning-starts	10000
train-frequency	10

Table 3: Parameters used for DQN

B Carla Parameters

Listing 1: Carla environment parameters

```
    {
            'number_of_vehicles': 100,
            'number_of_walkers': 0,
            'display_size': 256,  # screen size of bird-eye render
            'max_past_step': 1,  # the number of past steps to draw
            'dt': 0.1,  # time interval between two frames
            'discrete': True,  # whether to use discrete control space
            'discrete_acc': [-3.0, 0.0, 3.0],  # discrete value of accelerations
            'discrete_steer': [-0.2, 0.0, 0.2],  # discrete value of steering angles
            'continuous_accel_range': [-3.0, 3.0],  # continuous acceleration range
            'continuous_steer_range': [-0.3, 0.3],  # continuous steering angle range
            'ego_vehicle_filter': 'vehicle.lincoln*',
# filter for defining ego vehicle
            'host': 'localhost',
            'port': 2000,  # connection port
            'town': 'Town03',  # which town to simulate
            'task_mode': 'random',  # mode of the task
            'max_time_episode': 1000,  # maximum timesteps per episode
            'max_waypt': 12,  # maximum number of waypoints
            'obs_range': 32,  # observation range (meter)
            'lidar_bin': 0.125,  # bin size of lidar sensor (meter)
            'd_behind': 12,  # distance behind the ego vehicle (meter)
            'out_lane_thres': 2.0,  # threshold for out of lane
            'desired_speed': 8,  # desired speed (m/s)
            'max_ego_spawn_times': 200,  # maximum times to spawn ego vehicle
            'display_route': True,  # whether to render the desired route
            'pixor_size': 64,  # size of the pixor labels
            'pixor': False,  # whether to output PIXOR observation
        }
```