A Study in Reinforcement Learning with Deep Q-Network and Double Deep Q-Network on Low-Dimensional State Space

By

Hoe Tyng Chang (20113422)

hfyhc1@nottingham.ac.uk

Contents

ABSTRACT

In this paper, a set of questions was asked and answered to investigate the hyperparameters and performance of Deep Q-Network (DQN) and Double Q-Learning (DDQN). Firstly, this paper demonstrates how the minimum epsilon value will affect the training performance of Q-Learning. Second, this paper demonstrates the training performance between Deep Q-Network (DQN) and Double Q-Learning (DDQN) in low dimensional state space. Thirdly, this paper demonstrates the performance difference between seen and unseen environments for DQN and DDQN. Fourth, this paper is also able to identify emergent behaviour and provided ways to eradicate the emergent behaviour presented by the agents by retuning the reward system

1 INTRODUCTION

The topic surrounding reinforcement learning (RL) has been on the rise in recent years where researchers have achieved good results. The main concept of RL directly mirrors the real world on how an infant or animal learns when interacting with its surrounding where it learns about how the environment is responding to its behaviour. Reinforcement learning is the computational approach to learning which allowed computers to learn the ways of interacting with the environment.

Although there are results widely available online for DQN, DDQN and various other RL algorithms. However, most papers only report on the results of their RL algorithms without reporting how different hyperparameters and reward system affects their algorithms. Furthermore, most papers also ignore if their agents contained emergent behaviour or did not test on seen (same seeds) and unseen (different seeds) environment to further evaluate their RL algorithms. Other than that, the performance difference between DQN and DDQN on lower-dimensional state space is also unclear as it is originally designed for high-dimensional state space.

Hence, this paper aims to answer the questions above by conducting some experiments on a 2D-Driving environment with discrete action space and low-dimensional state space. The questions are listed below:

1. How does the minimum epsilon value affect the training performance of a reinforcement learning algorithm over a specific number of episodes?
2. The difference in training performance between DQN and DDQN in lower-dimensional state space?
3. What is the difference in performance between DQN and DDQN when testing on unseen seed and seed that is trained on over a specific number of episodes?
4. Is there any emergent behaviour in both models?
5. Will changing the reward system helps or does it cause another emergent behaviour to happen?

2 BACKGROUND

2.1 Reinforcement Learning

The beginning of reinforcement learning traces its roots to two main disciplines which consist of Animal Learning by Trial and Error and Optimal Control (Sutton et al., 2018). The first discipline is inspired by the psychology of animal learning where animals often learn by trial and error. The approach to solving the second discipline was developed in the mid-1950s by Richard Bellman who had come up with the idea of dynamic programming and Markovian Decision Processes (MDP) (Bellman, 1954, 1957).

Reinforcement Learning (RL) is a type of machine learning approach to predict how intelligent agents take actions to maximize the cumulative rewards received by their agents. The basic principle of RL modelled Markov Decision Process (MDP). MDP is defined by a tuple $\{S, A, P, R, \gamma\}$ where:

- *S is a finite set of states*
- *A is a finite set of actions*
- *P is a state transition probability matrix,* $P_{ss'}^{a} = \mathbb{P}[S_t = s, A_t = a]$
- *R is a reward function,* $R_s^a = [R_{t+1} | S_t = s, A_t = a]$
- *γ is a discount factor,* $\gamma \in [0,1]$

In MDP, the transition to the next state S_{t+1} depends on the current state S_t and the action made by the agent at the current state A_t. Each state-agent pair also comes with a reward R_t and the cumulative reward at each time step can be written as $G_t = R_{t+1} + R_{t+2} + \cdots$.

2.2 Deep Reinforcement Learning

Tesauro (1994) introduces one of the first successful Deep Reinforcement Learning algorithms – TD-Gammon which plays Backgammon and achieved staggering results where its performance is identical to the best player at that time. However, TD-Gammon failed to succeed in other games as researchers believed that the algorithm only works on Backgammon.

After that, Deep Q-Learning (DQN) was introduced by Mnih et al. (2013) which is the first deep learning model that successfully learned from a high-dimensional visual input (210 × 160 RGB) to play Atari games. DQN almost outperforms every algorithm at that time and even surpasses human experts on some of the games.

However, DQN has been prone to overestimation of Q-value in some algorithms. Hence, Van Hasselt et al. (2013) propose Double Q-Learning (DDQN) which reduces the overestimation of Q value during prediction in DQN.

2.2.1 Deep Q-Network (DQN)

Deep Q-Network (DQN) introduced by Mnih et al. (2013) is an off-policy deep reinforcement learning algorithm that utilizes a convolutional neural network (CNN) to approximate Q-value. The main component of the DQN algorithm is the target network θ and experience replay.

Off-policy algorithm uses epsilon-greedy strategy ($\boldsymbol{\varepsilon}$) where the agent will select a random action if the random number is lower than ε. ε will decay every step according to the epsilon decay rate until it reaches the minimum threshold.

Experience replay uses replay memory to store experiences that are collected by the agent. Replay memory comprises four tuples $\{s_t, a_t, r_t, s_{t+1}\}$ where s_t is the current state when the agent made an action a_t and r_t is the reward it received before moving to the next state s_{t+1}. Experience replay is first executed when the length of replay memory is larger than the batch size. Then, the experience replay sample randomly from the replay memory every episode with the size of the sample being the batch size. For each experience replay, the target used by DQN is:

$$Y_t^{DQN} \equiv R_{t+1} + \gamma\, max_a Q(s_{j+1}, a, \theta_t) \tag{1}$$

Finally, a gradient descent step is performed according to the target to update the target network θ.

2.2.2 Double Q-Learning (DDQN)

Double Q-Learning (DDQN) introduced by Van Hasselt et al. (2013) produces more accurate value estimates which leads to a better overall performance of the CNN. The max function in (1) causes DQN to overestimate the Q-value as it uses the same values to predict and evaluate action. Hence, DDQN uses two networks with two sets of weights θ and θ' where the target is:

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma\, Q(s_{j+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta'_t) \tag{2}$$

θ' is used to evaluate the policy and θ' is used to update θ on every episode.

3 IMPLEMENTATION

3.1 Environment Setup

This paper uses a 2D highway environment (Leurent, 2018) in OpenAI where it is designed for reinforcement learning tasks. The environment supports multiple different types of actions and observations. The observation space for this paper uses a 2-Dimensional table that has an observation size of (number of vehicles * 5). Each row is defined by a tuple $\{v, x, y, v_x, v_y\}$ where (x, y) being the location of the vehicle, (v_x, v_y) being the velocity of the vehicle and v is the Boolean value of the presence of the vehicle. The action uses a discrete action space of 5 where:

Action	Definition
0	Steer Left
1	Idle
2	Steer Right
3	Accelerate
4	Decelerate

Table 1: Definition of different actions on the environment.

Before training the algorithm, the environment is seeded to ensure the consistency of the environment during training.

3.2 Network Architecture

The model setup is similar for both Deep Q-Network (DQN) and Double Q-Learning (DDQN) and the network architecture is shown in Figure 1.

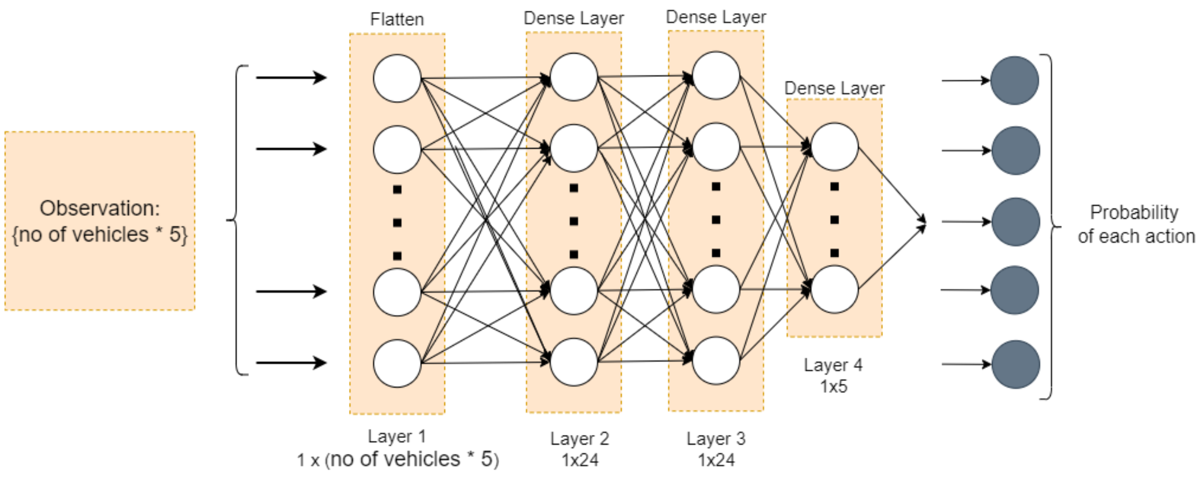
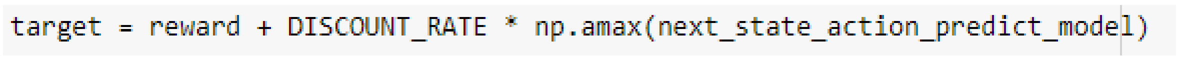


Figure 1: The Network Architecture of our DQN and DDQN

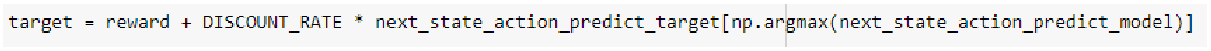
The network was built using Keras. The first layer is a Flatten layer that flattens the input of the observation. The second layer is a Dense layer that accepts the flattened observation layer and outputs a layer of 1x24. The third layer is a Dense layer that accepts and outputs a layer of 1x24. The last layer is a Dense layer that accepts 1x24 and outputs a 1x5 layer that shows the probability of every action. The loss function for architecture is Mean Squared Error.

The only difference between DQN and DDQN is how the target is selected and the equation is stated in (1) and (2). The code for DQN and DDQN is shown in Figure 2 and Figure 3.



```
target = reward + DISCOUNT_RATE * np.amax(next_state_action_predict_model)
```

Figure 2: Target Selection for Deep Q-Network (DQN)



```
target = reward + DISCOUNT_RATE * next_state_action_predict_target[np.argmax(next_state_action_predict_model)]
```

Figure 3: Target Selection for Double Q-Learning (DDQN)

3.3 Training and Testing

3.3.1 Training with different Epsilon Rate (ε) for Deep Q-Network (DQN)

Firstly, three DQN models are trained for 500 episodes each with epsilon rate being 0.05, 1e-5, and 0.9 with the cumulative reward being recorded for each episode. The cumulative reward collected is then plotted to find the rolling average and the distribution of the reward received by the agent. The hyperparameters were the same for three of the models except for their epsilon rate. The hyperparameters are shown in Appendix A.

3.3.2 Training Double Q-Learning (DDQN)

The best epsilon rate is then selected and chosen to train DDQN with the hyperparameters and number of episodes identical to DQN. The cumulative reward for each episode is also recorded to compare the rolling average and the distribution of cumulative rewards. After that, it is plotted together with DQN and DDQN to compare the results of both algorithms.

3.3.3 Testing models on seen (same seed) and unseen (different seed) environment

The testing section is divided into two parts:

1. The environment has the same seed during the training phase.
2. The environment has different seeds during the training phase.

Each section is tested with 3 different algorithms: DQN, DDQN, and Random Action with 100 episodes each. The rewards received by the agent are then recorded to plot the average rewards by each algorithm in each section. The speed of the agent and the actions taken by the agent is also recorded at every step for each algorithm to evaluate the emergent behaviour of the agent.

3.3.4 Investigating behaviour of agents to analyse emergent behaviour

Firstly, the sum of each action taken by the agent in different algorithms is plotted to analyse how different algorithms had chosen to react in the environment. Since there is emergent behaviour presented in the results, the average speed of each agent in the algorithm is also plotted to further investigate the behaviour of the agent.

3.3.5 Tuning the reward system and retraining DQN and DDQN

The reward system is first modified in a way to encourage agents to switch lanes and drive faster by increasing the minimum speed threshold before distributing rewards, increasing the reward if the agent reached the minimum speed threshold and distribute some rewards to the agent if they switch lane. The configuration is shown in Appendix C. After that, the agent is retrained again with 750 episodes for both algorithms on the new policy and both agent is tested where the speed and actions taken by the agent along with the cumulative rewards per episode is collected to plot a bar chart showing the cumulative reward, actions, and the average speed of each agent during testing. The reward system in the testing environment is the same as the reward system used in 3.3.3 to ensure fairness and accurate results during comparison.

4 RESULTS

4.1 Training performance of DQN for Minimum Epsilon (ε) of 0.05, 1e-5, and 0.9.

Figures 4 and 5 show the distribution of rewards and the running average obtained from 500 episodes of training of DQN with minimum epsilon of 0.05, 1e-5 and 0.9.

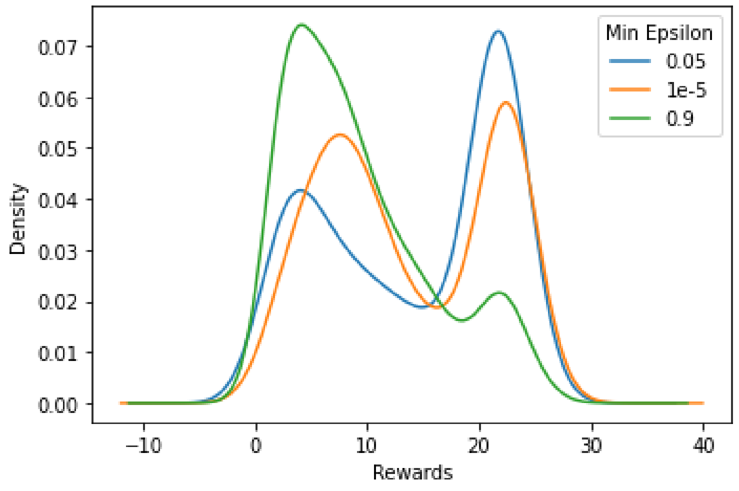


Figure 4: Distribution of rewards for Minimum Epsilon (ε) 0.05, 1e-5, 0.9 of DQN.

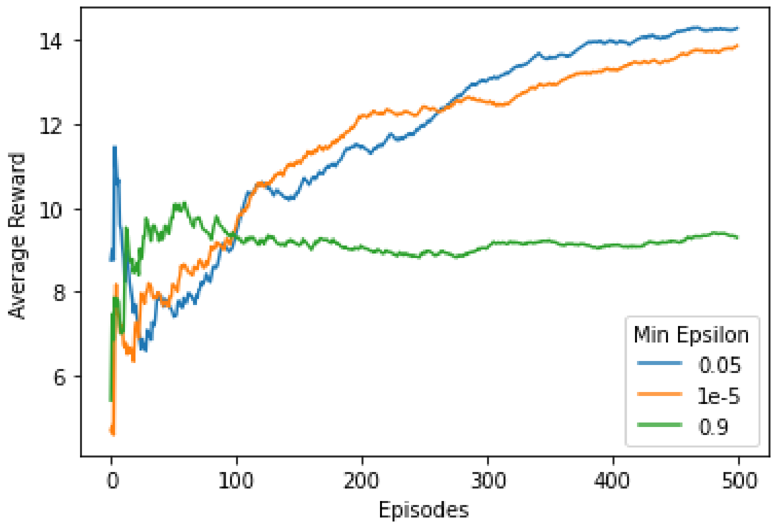


Figure 5: Running average for Minimum Epsilon (ε) 0.05, 1e-5, 0.9 of DQN.

According to Figures 4 and 5, minimum epsilon of 0.05 have a significantly higher positive mode than a negative mode and a higher running average after 100 episodes of training when compared to minimum epsilon of 1e-5 and 0.9 which indicates that the agent trained with minimum epsilon of 0.05 crashed less frequently when performing the task. Other than that, minimum epsilon of 1e-5 has a similarly positive and negative mode while minimum epsilon of 0.9 has a higher negative mode than a positive mode. Both minimum epsilon of 1e-5 and 0.9 also have a lower running average after 100 episodes of training.

4.2 Training performance comparison between DQN and DDQN (Min Epsilon = 0.05)

Figure 6 shows the moving average obtained from 500 episodes of training of DQN and DDQN with a moving average window of 75.

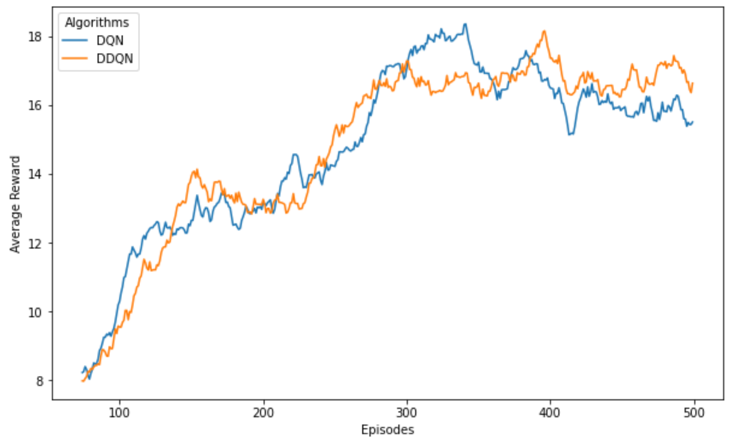


Figure 6: Moving average (Window = 75) of Deep Q-Network (DQN) and Double Q-Learning (DDQN) during training of 500 Episodes

At 500 episodes, DDQN performed slightly better than DQN. DDQN had an average reward of 14.41 whereas DQN had an average reward of 14.29. However, the average reward for DDQN is constantly improving and is still improving at the end of 500 episodes whereas the moving average of DQN started to decrease after 300 episodes.

4.3 Testing performance comparison between DQN and DDQN on seen (same seed) and unseen (different seed) environment.

Figure 7 shows the average rewards per episode by DQN, DDQN and random action on the seen and unseen environment after 100 episodes of testing per algorithm.

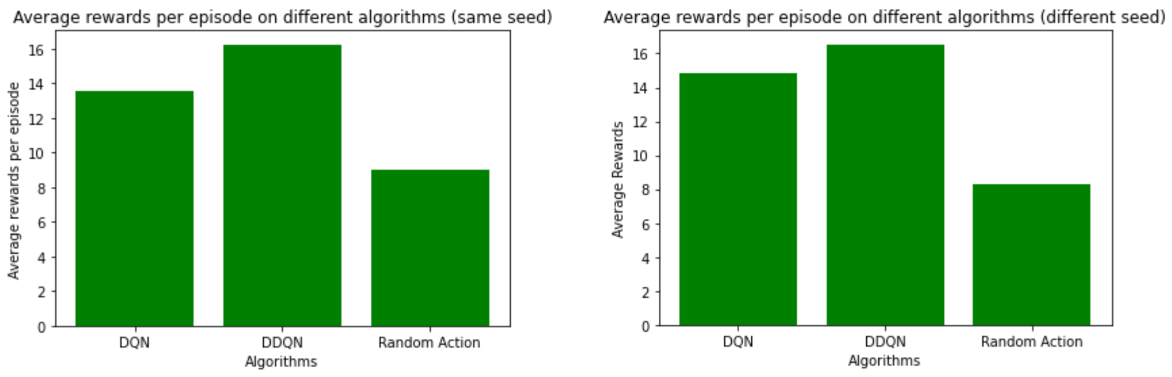


Figure 7: The average reward per episode in the seen and unseen environment for DQN, DDQN and Random Action

Based on the results shown in Figure 7, DDQN performs better when compared to DQN and random action with average rewards of 16.24 for the seen environment and 16.54 for the unseen environment. DQN performs slightly worse than DDQN at 13.57 for the seen environment and 14.84 for the unseen environment. Furthermore, both models performed significantly better when compared to random action.

4.4 Analysing emergent behaviour on DQN and DDQN

Figure 8 shows the cumulative action and average speed by DQN, DDQN and random action. The data were obtained during the testing of the algorithms on the seen and unseen seeds.

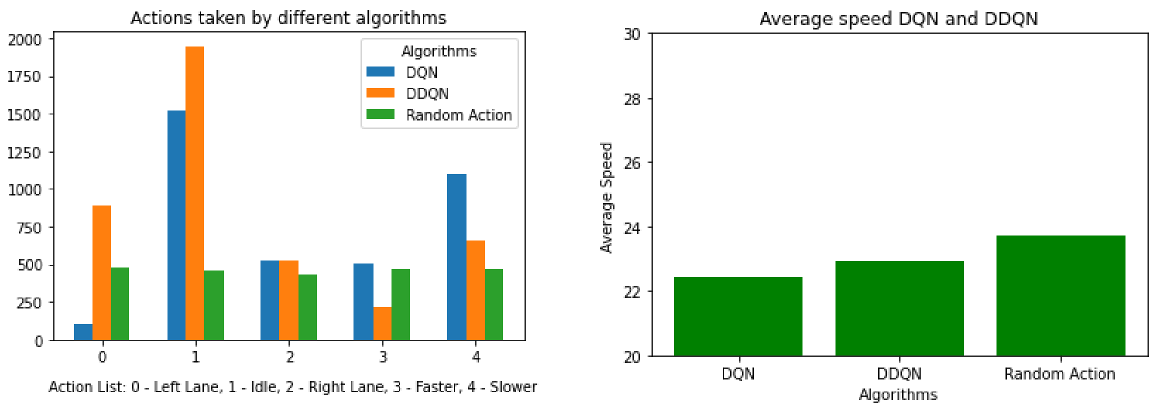


Figure 8: Cumulative action taken, and the average speed of agents trained by DQN, DDQN and agent that performs random action

The results had shown that both DQN and DDQN stayed in an idle state the most and decelerating the vehicle came in the second place. Furthermore, the average speed for both algorithms is lower than random action at 22.42 for DQN and 22.95 for DDQN.

4.5 Performance after retraining on updated rewards system on Deep Q-Network (DQN) and Double Q-Learning (DDQN)

Figure 9 shows the results that are obtained from testing the agents in different algorithms after retraining with an updated rewards policy.

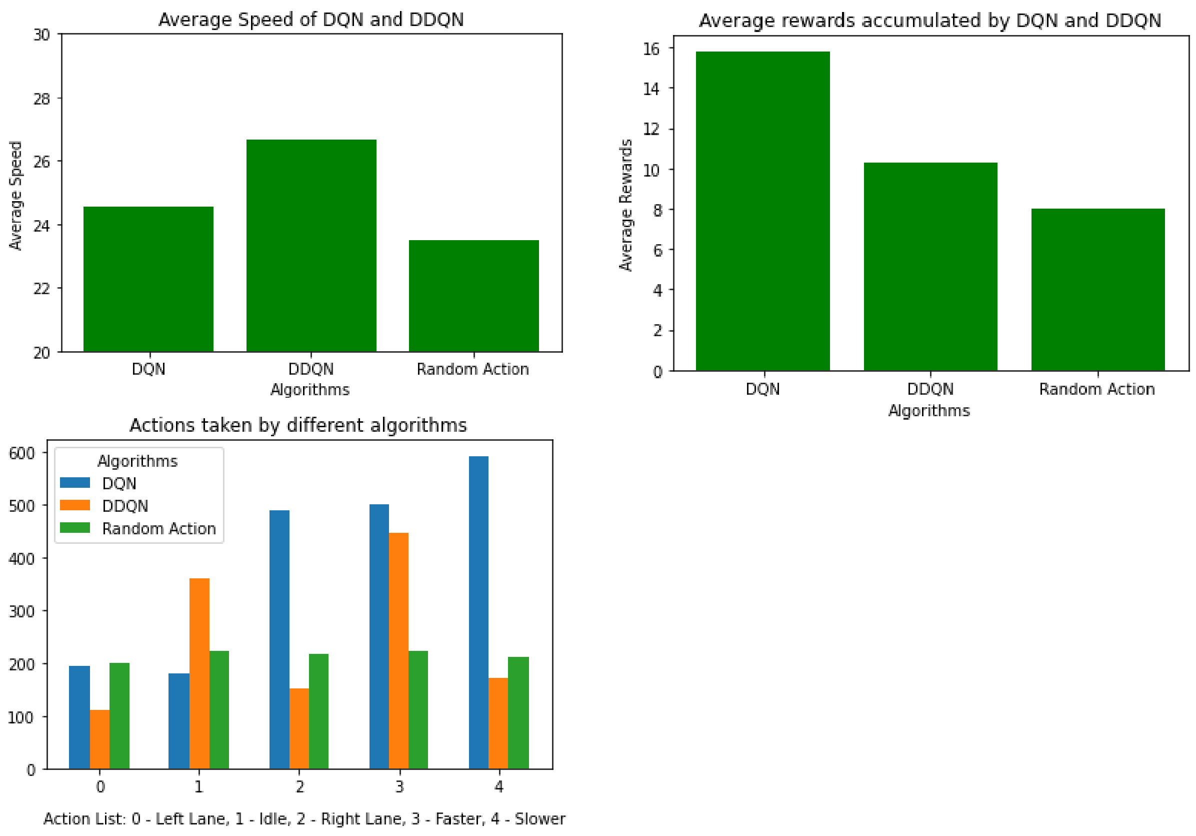


Figure 9: Cumulative action taken, Average Speed and the Average Rewards Cumulated of agents trained by DQN, DDQN and agent that performs random action after retraining with updated rewards policy

The results show that the average speed has increased for both updated DQN and DDQN and the average rewards taken by the updated DQN are similar to the first DQN. Furthermore, the actions for DQN are more distributed with more actions on changing lanes and controlling the speed of the agent instead of remaining idle or only decelerating. However, the updated DDQN slightly underperformed when compared to the previous DDQN on the average rewards although it has the highest average speed.

5 DISCUSSION

5.1 The effects of minimum epsilon rate in Q-Learning

As shown in Figures 4 and 5, the optimal minimum epsilon rate plays a crucial role in balancing the exploration and exploitation of Q-Learning. If the minimum epsilon rate is set too high, the agent may never explore new states and ignored the best policy. Furthermore, if the epsilon rate is set too low, the agent may excessively explore new states and never reach the optimal policy.

5.2 The training and testing performance of DQN and DDQN

The moving average that is shown in Figure 6 had shown how overestimation of the state-action pair in DQN can lead to inconsistency in training and causes a decrease in average reward during training.

This is because the DQN agent might start to prefer the states that it believes to be the most rewarding even though the states are not the most rewarding which leads to the agent taking suboptimal actions and not reaching the optimal policy.

On the other hand, the moving average for DDQN is much more consistent where the agent had not experienced any large drop in moving average and is constantly improving throughout the entire training period.

5.3 Emergent behaviour and retraining agents with an updated reward system

The graph in Figure 8 concluded that the agents are not overtaking other vehicles in the environment and only slow down to avoid crashing into neighbouring vehicles.

By tuning the reward systems, we can observe that the average speed of the agents significantly increased, and the action taken by the agents is more dispersed after retraining. For DQN, the average rewards also remain the same when compared with the previous version of DQN which had meant that the updated DQN had learned to overtake neighbouring vehicles after updating the rewards system.

On the other hand, the average rewards of DDQN don’t perform well although the average speed is slightly higher than DQN. The suspected reason for the behaviour is it might be an underestimation bias which can make the training process more unstable.

5.4 Challenges during implementation

The main challenge when implementing this project is balancing the exploration-exploitation trade-off where it is difficult to tune the learning parameters of our project. After that, it is also difficult to tune the reward system when trying to eradicate the emergent behaviour so that the agent will be provided with the right amount of reward for it to accurately captures the desired behaviour.

.

6 CONCLUSION

In a conclusion, this paper has addressed the 5 questions that are proposed in this research. Most of the results aligned with the underlying working concept of DQN and DDQN. However, there is a drop in average rewards with the agent that is trained with DDQN after tuning with the rewards system.

Firstly, we experimented with the minimum epsilon rate and determined that 0.05 is the most optimal for our tasks. Second, we determined that DDQN performs better than DQN during training and testing on the same seed and unseen seed before tuning the rewards system. Thirdly, we had found emergent behaviour and eradicated it by retuning the rewards system.

6.1 Limitation

Due to time constraints, the drop in average rewards of DDQN is also not being investigated as there is a strict time constraint for this project.

6.2 Future Works

In the future, the drop in average rewards of DDQN can be investigated by further researching and retuning the hyperparameters and rewards system.

7 REFERENCES

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning.

Van Hasselt, H., Guez, A. and Silver, D., 2016, March. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).

Bellman, R., 1957. A Markovian decision process. Journal of mathematics and mechanics, pp.679-684.

Bellman, R., 1954. The theory of dynamic programming. Bulletin of the American Mathematical Society, 60(6), pp.503-515.

Sutton, R., Bach, F. and Barto, A., 2018. Reinforcement Learning. Massachusetts: MIT Press Ltd.

Tesauro, G., 1994. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, *6*(2), pp.215-219.

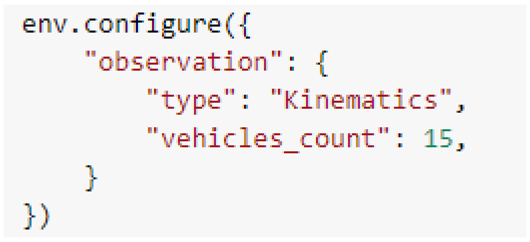
Leurent, E., 2018. An Environment for Autonomous Driving Decision-Making (Version 1.4) [Computer software]. https://github.com/eleurent/highway-env

8 APPENDICES

Appendix A: Hyperparameters

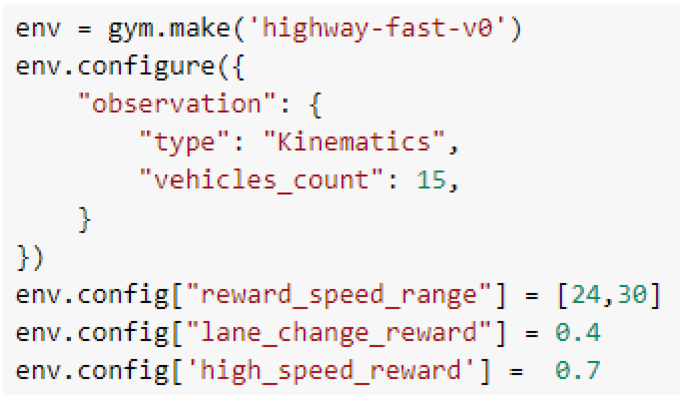
Hyperparameters	Value
Minimum Epsilon	0.05
Epsilon Decay Rate	0.995
Learning Rate	5e-4
Batch Size	64
Discount Rate	0.8

Appendix B: Environment Configuration A



```
env.configure({
    "observation": {
        "type": "Kinematics",
        "vehicles_count": 15,
    }
})
```

Appendix C: Environment Configuration B



```
env = gym.make('highway-fast-v0')
env.configure({
    "observation": {
        "type": "Kinematics",
        "vehicles_count": 15,
    }
})
env.config["reward_speed_range"] = [24,30]
env.config["lane_change_reward"] = 0.4
env.config['high_speed_reward'] =  0.7
```