NOVA IMS Information Management School

Reinforcement Learning Approaches to Autonomous Driving

Reinforcement Learning

Master Degree Program in Data Science and Advanced Analytics

r20201614 – Pedro Barão

20230567 – Adriana Costinha

20230577 – Ana Filipa Silva

20230755 – Beatriz Vasconcelos

1.Introduction

In this project, we aim to train a reinforcement learning (RL) agent to solve the Highway environment. It requires the agent to balance the trade-off between exploration and exploitation to achieve optimal performance. The agent must learn to navigate through dynamic traffic conditions, requiring sophisticated decision-making capabilities to balance speed and safety effectively. The challenge lies in ensuring that the agent can generalise its learned policy to various traffic scenarios, maintaining high performance without compromising safety.

The environment configuration is extremely important since it acts as the set of rules of how everything will work. Although it varies depending on the combination we're working on, its structure is mainly the same. It has the parametrization of our problem, followed by the choice of observation and action type, the reward system and finally the parameters defined for visualisation. For this project we chose to work with two observation types, *TimeToCollision* and *Kinematics*, for action types, we also chose two, *DiscreteMetaAction* and *ContinuousAction*, and finally for algorithms we used *DQN*, *SARSA* and *SAC* explained further below.

The complexity of the states and actions in the *Highway* environment presented several issues when using certain combinations of algorithms and observation spaces. These issues are further explained in other sections of the report, including the need to reduce the number of features observed for the *SARSA* algorithm.

The reward system used depends on the combination used. Our baseline was to define the two rewards that appeared more relevant, collision and speed, as can be seen on Table 1, leaving all others to their default values. Further down, we explain what changes we made for each combination.

Table 1 - Initial reward system.

Reward	Values
collision	-1
Speed range	[20,30] m/s

2.Environment Observation Types

2.1 Time to Collision

The *TimeToCollision* observation type provides information about how soon the car is expected to collide with other vehicles in its environment, helping the agent to take actions to avoid collisions and navigate safely.

It is an array, $V \times L \times H$, that represents the predicted time to collision of observed vehicles on the same road as the car. Where V represents the velocity, L the lanes around the current lane and H represents the discrete time values with one second steps[1].

2.2 Kinematics

The *Kinematics* observation type provides detailed information about nearby vehicles, enabling the vehicle to make informed decisions to navigate safely. It consists of an $V \times F$ array that describes a list of V nearby vehicles by a set of features of size F[2].

3.Agent Action Types

3.1 Continuous Actions

The *ContinuousAction* type allows the agent to directly control the vehicle's throttle and steering angle, providing low-level control over the vehicle's kinematics, we can define the ranges for those two when creating a continuous action space.

3.2 Discrete Meta Actions

On the other hand, the *DiscreteMetaAction* type adds a layer of control over the continuous low-level control, allowing the car to automatically adjust its speed and position on the road. The available meta-actions include lane changes and adjustments to cruising speed. These actions enable the agent to make decisions about lane changes and cruising speeds, facilitating safe and efficient navigation.

4. Algorithms

4.1 DQN

We know Q-learning aims to learn a policy that tells an agent what action to take under certain circumstances. It does this by learning a Q-function, *Q(s,a)*, and estimates the expected cumulative reward of taking action a in state s and following the optimal policy after. This function is updated based on the *Bellman* equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \times maxQ(s',a') - Q(s,a))$$

[1] *highway-env Documentation*. Farama.org.https://highway-env.farama.org/observations/#time-to-collision
[2] *highway-env Documentation*. Farama.org. https://highway-env.farama.org/observations/#kinematics

Overtime, Q-learning converges to the optimal Q-values, allowing the agent to choose actions that maximise cumulative rewards.
Deep Q-Networks algorithm, *DQN*, combines Q-learning with deep neural networks, enabling agents to learn optimal policies in complex environments. Its process can be summarised in seven steps[3]:

1. State Representation: Converts the environment's current state into a numerical format.
2. Q-Network: Uses a deep neural network to output Q-values for each possible action given the state.
3. Experience Replay: Stores experiences - state, action, reward, next state - in a replay buffer.
4. Mini-Batch Updates: Samples experiences from the buffer to update the network using the *Bellman* equation.
5. Exploration vs. Exploitation: Chooses actions using a balance of random exploration and greedy exploitation.
6. Target Network: Maintains a separate target network for stable Q-value updates, periodically syncing it with the main network.
7. Iteration: Continuously repeats the process to refine the policy.

DQN was chosen for its effectiveness in environments with discrete action spaces and its capability to handle high-dimensional state spaces, such as those found in the *Highway* environment. Additionally, we chose *MlpPolicy* as the agent's policy, as used in class.

4.2 SARSA

SARSA is a reinforcement learning algorithm that updates its policy based on the actions taken and their outcomes while following the policy it's learning from. It operates by looking at the current state and action, the reward received, and the next state and action, using these to update its value estimates.

This algorithm was included for its on-policy nature, which can lead to more stable learning and policies that better reflect the actual behaviour of the agent. In the *Highway* environment, where safety is critical, *SARSA*'s more conservative approach may help in developing policies that avoid risky actions, thus minimising crash risks while still learning to navigate efficiently.

[3] Shruti Dhumne. *Deep Q-Network (DQN) - Shruti Dhumne - Medium*. Medium. https://medium.com/@shruti.dhumne/deep-q-network-dqn-90e1a8799871

4.3 SAC

Soft Actor-Critic (*SAC*) is an off-policy actor-critic algorithm that optimises a stochastic policy using the maximum entropy reinforcement learning framework. A central innovation in SAC is the inclusion of an entropy term in the reward function, which encourages the policy to start by acting randomly. The agent gradually reduces randomness as it gains confidence in its learned policy. This promotes a balance between exploration and exploitation. Additionally, SAC utilises a Q-value function, similar to other actor-critic algorithms, to estimate the value of state-action pairs.

SAC was selected for its stability, sample efficiency, and robustness, particularly in continuous control tasks. The entropy regularisation in *SAC* ensures continuous exploration for complex decision-making environments.

5. Combinations

5.1 *DQN* with *TimetoCollision* and *DiscreteMetaActions (DQN1)*

When using *DQN* with *TimetoCollision* and *DiscreteMetaActions* we tested possible reward values, deciding to maintain the original reward system since we were able to obtain great results with it, as shown in *Table 2*.

Table 2 - Rewards for DQN with TimetoCollisionand DiscreteMetaActions.

Reward	Values
Collision	-1
Speed range	[20,30] m/s

For the *DQN*, we also tested different parameters of the algorithm itself. The learning rate was decreased to 5e-4, the batch size to 32, gamma to 0.8, and the number of timesteps to 200. These parameters were tested with different values, although the best combination was achieved with the ones referred.

5.2 *DQN* with *Kinematics* and *DiscreteMetaActions (DQN2)*

DQN was then used again with the same type of actions, but with a different observation space: *Kinematics*. This space revealed itself to be more challenging, there were many available features and configuration settings that could be changed. At first, the default configuration was used, but with no success. This default configuration only looked at the x and y position of the cars and their velocity in the x and y axis. There was also another parameter “presence” which indicated which vehicles were close to the agent and whose actions were being considered. With these settings, the car seemed to be oblivious to its surroundings, not taking into

consideration the other vehicles on the highway. This most commonly resulted in crashes with no apparent effort of the agent to avoid them. Rarely the car would randomly change lanes without any obvious intent behind it.

In an attempt to improve this, more features were added to the list of observed characteristics of the agent and the other cars: “heading”, the orientation of the vehicles in radians; “long_off”, the longitudinal offset of the vehicle which indicates how far it is along the length of the lane; and “cos_d” and sin_d”, which define the trigonometric directions of the vehicle’s destination. The purpose of adding these features was to aid in the prediction of the other objects’ movement and to understand the relative position of vehicles in the same lane, hoping to improve the predictive power and safety of the agent. Besides these added observed features, the number of vehicles observed was increased to 10 and the “observe_intentions” parameter was set to true. The normalisation and clipping of the features was tested, but no differences were seen. These parameters were changed iteratively, in order to test if adding each one would improve the model. After many different tests, the agent’s decision-making barely improved, so the reward system was changed next.

Table 3 - Rewards for DQN with Kinematics and DiscreteMetaActions.

Reward	Values
Collision	-2
Speed range	[25,35] m/s
High Speed	0.15
Right Lane	0.1
Lane Change	-0.05

The changes in the reward system noticeably changed the behaviour of the car. After not normalising the rewards, we noticed that if the rewards of staying in the right lane and speeding were too high, the agent would always turn to the rightmost lane at the start and speed until it crashed. To avoid this, a slightly more negative penalty was given for crashing, -2, and the rewards of staying in the right lane and speeding were changed to 0.1 and 0.15 respectively (*Table 3*). After these changes, the car would default again to the rightmost lane, but then speed down to the lowest speed at which the rewards were still being given, effectively letting every other vehicle pass him so that the agent would have no obstacles in front of him until the end of the episode. To prevent this, the minimum speed at which rewards were given was increased to 25. This made the car behave slightly better, but only rarely would it actually show any learning by avoiding cars.

As these changes were being made, different parameters of the algorithm itself were also tested. The learning rate was increased to 1e-3, the batch size to 64, and gamma to 0.95. Most importantly, the number of timesteps to learn with was increased to 10000, but the agent didn’t get noticeably smarter.

5.3 *SARSA* with *Kinematics* and *DiscreteMetaActions*

Since *SARSA* uses a table to store Q-values for every state-action pair, it requires a finite number of states and actions, requiring the discretization of the state space. When implementing this algorithm with *Kinematics*, we encountered memory issues related to storing the Q-table. This constraint led us to limit our model to only 4 vehicles, focusing on the features deemed most important: presence, x, y, and vx. For these last three parameters, we needed to balance the memory usage associated with the size of the Q-table and the accuracy of the state representation. Consequently, we opted to use bins of size four.

When working with this setup, we noticed that the car consistently crashed into others, either by maintaining or increasing its speed within the same lane or by turning into a car adjacent to it. Our initial approach was to significantly penalise the agent whenever it crashed into another car, changing the *collision_reward* to -100. However, this did not alter its behaviour. When faced with such a high penalty, the agent might learn to prioritise avoiding negative outcomes at all costs, which could lead it to choose less efficient routes simply to prevent collisions. To avoid overwhelming the agent with excessive negative rewards, we adjusted the collision parameter to -20 instead.

In an attempt to counteract the problematic behaviour described, we designed a proximity penalty to discourage the agent from getting too close to other vehicles. This penalty is calculated based on the proximity to the nearest car—the closer the agent is, the larger the penalty. Despite this, the agent continued to behave as before, even though the overall reward was negative.

We speculated that perhaps the penalties were not being directly and immediately linked to specific actions, thus diminishing their effectiveness in modifying behaviour. This issue could also be related to the space discretization in this algorithm, leading to poor state representation. Consequently, the features used to represent the state to the agent, might not include adequate information on the proximity or relative speed to other vehicles.
Therefore, we decided to lower the speed range reward which could increase safety, giving the agent more time to react to sudden changes or obstacles. Unfortunately, this did not seem to have much of an impact as well.

Table 4 - Rewards for SARSA with Kinematics and DiscreteMetaActions.

Reward	Values
Collision	-20
Speed range	[10,20] m/s

After training the agent for 1500 iterations, applying the proximity penalty and changing the reward values as shown in *Table 4*, although the car stayed within the lanes and sometimes

turned correctly, the crash rate was very high, as its actions appeared to be totally random, as can be seen in *Figure 3* and *4*.

5.4 *SAC* with *Kinematics* and *ContinuousActions*

We tested different configurations for the observation to determine the most effective inputs for the RL agent. The configuration that worked well involved observing the kinematics of 10 nearby vehicles, including features such as presence, coordinates (x and y), velocities (*vx* and *vy*), and orientation (*cos_h* and *sin_h*). This comprehensive set of features significantly enhanced the agent's performance. In contrast, a simpler configuration with fewer vehicles and normalised values proved less effective. The richer observation set provided the agent with more comprehensive state information, leading to better decision-making and navigation.

For this algorithm we used the continuous action space, we explored some different parameters such as acceleration range, steering range, and whether to simulate dynamics. Adjusting the steering range to limit movement was crucial; it prevented the agent from spinning but made obstacle avoidance more challenging. On the other hand, increasing the steering range resulted in erratic driving behaviour.

Similarly, the acceleration range required careful tuning. Negative ranges led the car to reverse, which was undesirable. Restricting the range to positive values, from zero upwards, proved to be the most effective. We also experimented with enabling dynamical simulations, which occasionally helped the car to stay on the road but often slowed it down too much, making it difficult to keep up with other vehicles.

To address these issues, we designed a custom reward function. This function penalised the agent for deviating from the road and rewarded it for maintaining a safe distance from other vehicles and moving towards the target. The custom reward combined default rewards (Table 5), distance rewards, on-road rewards, and target distance rewards, proving to be essential in teaching the agent to stay on the road. Initially, we also included a collision penalty, but removed it after finding that it made the overall reward too negative, hindering the learning process.

Table 5 - Rewards for SAC with Kinematics and ContinuousActions.

Reward	Values
Collision	-2
Speed range	[20,30] m/s

Our experiments with the custom reward function revealed significant challenges in training the agent. Despite efforts to balance speed and safety, we couldn’t teach the car to slow down properly, since when we included that option (negative ranges on acceleration) led the car to reverse.

We did try to penalise slow speeds or reversing but it often led to unintended behaviours, and the car started to go off road again. The best-performing model was one that consistently stayed on the road but sometimes made dangerous overtakes due to maintaining high speeds consistently.

We found some trouble in making our model learn consistently, which can be due to the fact that *SAC* starts by encouraging the agent to act randomly. Despite successfully learning to stay on the road, the agent encountered difficulties when trying to avoid other cars. It followed a sinusoidal path, ultimately leading to crashes. This behaviour emerged because the agent faced a larger penalty for staying off-road compared to the penalty for crashing. However, when we increased the penalty for crashes the agent tried to balance penalties by staying close to the road but still outside in order to completely avoid crashing. We did try to increase the timesteps to see if the behaviour changed, but it appeared to only increase the learning process, so we left it at 5000 timesteps.

6. Discussion

For the discussion of the four combinations tested, we decided to plot four different graphics, such as the speed, the travel distance, total rewards and how many times collisions occurred for each combination. These plots were the result of simulations with the same reward system over 100 episodes for each trained model/solution.

Figure 1 shows the average vehicle speed and travelling distance of the four combinations we tried. A higher speed implies that the agent could run through the driving scenario faster and achieve greater cumulative rewards. The results clearly demonstrate that *DQN2* achieved a more stable speed over episodes, compared to *SARSA* and *SAC*. Although *SAC* achieved higher speeds, it travelled almost no distance (Figure 2). This was due to the algorithm failing to teach the car to decelerate. Consequently, the car would accelerate rapidly at the beginning, reach high speeds, and then crash almost immediately after the episode started.

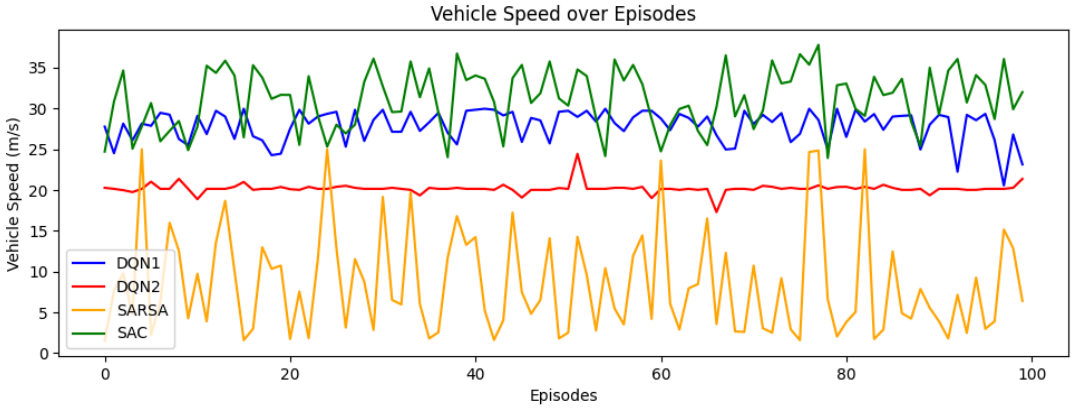

Figure 1 - Average vehicle speed over episodes, for each combination.

Figure 2 shows the travelling distance of the four combinations we tried. A higher travelling distance means the agent could drive longer during a time limit without collision. As we can see, DQN1 has a higher travelling distance, because of its higher vehicle speed as well as its smaller number of collisions, *figure 3*.

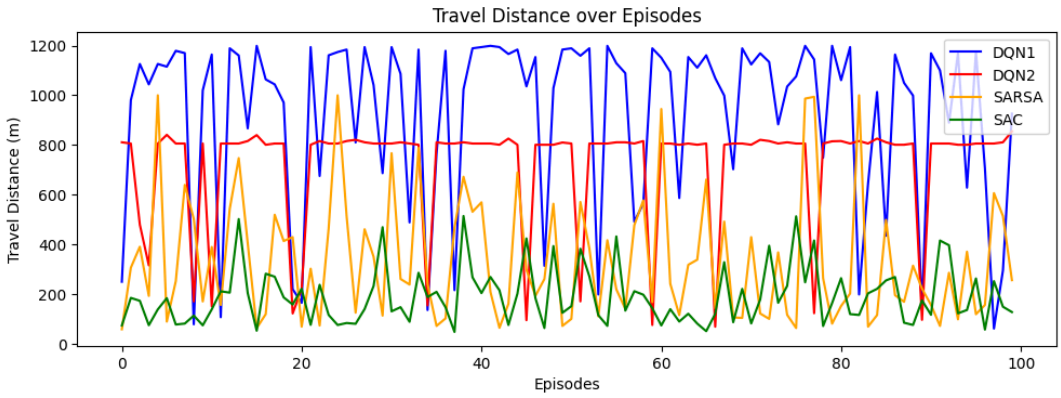

Figure 2 - Travelling distance over episodes, for each combination.

Figure 3 represents the total count of collisions during the 100 episodes we did for each algorithm. We can easily see *SAC* never finished one episode without colliding with another vehicle, as well as *SARSA*, with only three finished episodes without collisions. Between both *DQN*s, we can conclude *DQN2* collided less times than *DQN1*, even though it had a lower reward value, *figure 4*, as well as has a lower speed value, *figure 2*. An interesting observation in *DQN1* was that when the car was “trapped” between multiple vehicles, it would decelerate in order to safely overtake, a behaviour that was not consistently seen in *DQN2*.

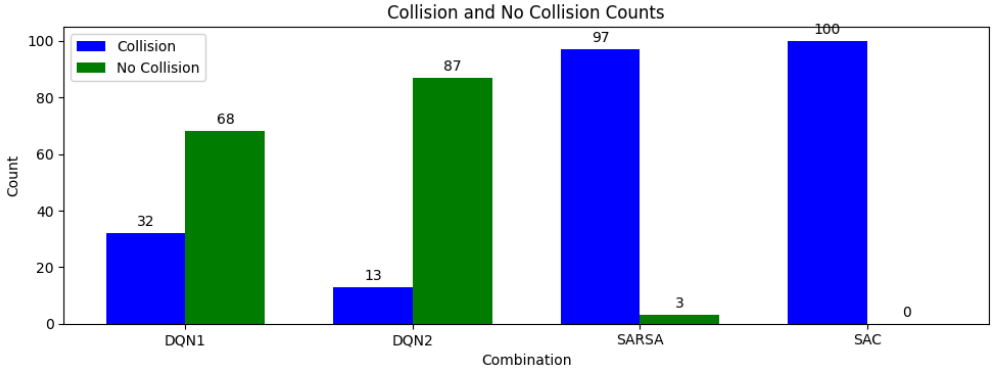

Figure 3 - Total collisions in each combination for 100 episodes.

Figure 4 shows the total rewards accumulated over different episodes for each algorithm. A higher reward indicates more safe driving while staying on the preferred lane and maintaining high speed, reflecting better driving performance. The results clearly demonstrate that both *DQNs* not only achieved higher rewards consistently but also exhibited superior training stability and faster learning speed compared to *SARSA* and *SAC*. The difference between both *DQNs* is

due to the fact that *DQN1* has a higher travelling distance as well as a higher speed, resulting in higher rewards, although they're both consistent for all episodes.

Figure 4 - Total rewards over episodes, for each combination.

As the safety claim is the first concern for the actual application of our project, the best combination we worked with was between *DQN1* and *DQN2*, where we can conclude *DQN* is the superior algorithm.

7. Conclusion

We recommend using *DQN* for the Highway environment due to its superior performance as it uses deep neural networks to approximate the Q-value function, enabling it to generalise well across similar states and adapt more dynamically to the fast-changing conditions in navigating traffic, avoiding crashes, and maintaining high speed.

SARSA, due to the need for discretization of the state space, might not efficiently handle complex, high-dimensional state spaces, such as the one used. Since this algorithm relies on the trajectory taken, including exploratory actions, it can lead to suboptimal performance.

In *SAC*, the agent's inability to decelerate effectively and the challenges in penalising slow speeds without inducing unintended behaviours remain significant limitations. While the best-performing model managed to stay on the road, it sometimes exhibited risky behaviours due to maintaining high speeds constantly. Further research and development are necessary to address these issues and improve the agent's decision-making capabilities on these models.

One challenge we encountered during this project was the use of *Kinematics*. With this observation type, it appeared as though the car lacked perception of other vehicles, which complicated the management of reward functions. When attempting to establish specific rewards, the agent consistently took actions that did not align with the intended rewards.