CSCE 790-001: Deep Reinforcement Learning and Search Motion Planning in Dense Traffic

Huang, Hung-Tien Workman, Timothy Cale

December 5, 2021

1 Abstract

A major challenge in autonomous driving is anticipating the actions of other drivers. In dense traffic scenarios this presents many unknowns. To navigate safely, an autonomous vehicle must reason about the likely actions of other drivers as well as anticipate how those drivers may react to its own actions. We propose to use off-the-shelf Python software [1] to model typical traffic scenarios and apply reinforcement learning strategies to teach an autonomous vehicle to navigate the environment safely. We will investigate the effects of some hyperparameters on the learned policy as well as compare the autonomous vehicle's performance for different traffic levels. Our policy shows an improvement over the random policy, though there is some instability in the learning rate. Visual inspection of the the simulation running our learned model shows that the vehicle still occasionally crashes in a manner that a human driver would consider easily avoidable, such as rear-ending the vehicle in front. Nonetheless, our policy can successfully navigate the environment in the majority of cases.

2 Introduction

Driving in real world dense traffic scenarios remains a challenging task for autonomous vehicles as actions depend on nearby traffic participants [2]. For example, for an autonomous vehicle to merge from the on-ramp to the main lane, the vehicle must first reason whether nearby vehicles are willing to yield or not, then interact with multiple road-users to induce them to yield, and finally maneuver the lane change when a safe margin is available [3].

In the current investigation, a reinforcement learning agent is trained to navigate a multi-lane highway. The agent is trained to make decisions based on the location and the velocity of itself and nearby vehicles. Currently, an action value function is learned from deep Q-learning [4] and the agent acts greedily with respect to the action value function; such network will be used as the baseline model. The discrete soft actor-critic method [5] is planned to be implemented and compared to the baseline model.

3 Related Work

The traditional motion planning methods are too conservative and fail in dense traffic scenarios [6] [7]. Works that utilize probability models to take the dynamics between traffic participants into consideration do not scale due to the curse of dimensionality [8] [9] [10]. Deep Reinforcement learning has the capability to avoid the curse of dimensionality. However, as pointed by [3], the work introduced in [11] does not provide safety guarantees and the one in [12] is behaving too conservatively to provide safety guarantees.

To address the issue, the Interactive Model Predictive Controller (IntMPC) framework is introduced in [3]. An interaction-aware policy is learned and used to provide a velocity reference to a local optimization-based planner. Simulation results demonstrate that IntMPC triggers interactive negotiating behavior to reason about the other drivers' cooperation and exploit their cooperativeness to induce them to yield while remaining safe.

4 Approach

To solve the lane merging problem, a simplified IntMPC framework introduced in [3] was implemented in the open-source highway-env [1] simulator. The ego-vehicle identifies the leader and follower vehicles surrounding it, takes their location and velocity w.r.t its inertial frame, and then chooses to either merge left, merge right, speed

up, slow down, or maintain its velocity (idle). After an action is taken by the ego-vehicle, the Intelligent Driver Model (IDM) [13] simulates the behavior of all other vehicles. At present, deep Q-learning is used to solve the Model Decision Process (MDP) as a baseline.

In the current investigation, the global path planner is assumed to issue a more abstract command, which is telling the ego-vehicle to drive down the highway indefinitely. In other words, the agent has no knowledge of which lane it should be in. The discrete vehicle controller used in the current investigation is a wrapper around the continuous vehicle controller that also solves the MPC problem subject to slightly different constraints. Also, the only difference between P-IDM and IDM is the method of selecting the potential sets of leader and follower vehicles.

4.1 Leader and Follower

The leader is defined as the nearest vehicle in front of the ego-vehicle and the follower as the nearest vehicle behind it. Both leader and follower may be in lanes adjacent to the ego-vehicle. The state of the leader and follower vehicles in lane $i, \forall i \in [0, 2] \cap \mathbb{Z}$, at time step k are denoted as $\mathbf{s}^l_{i,k}$ and $\mathbf{s}^f_{i,k}$, where the lane that the ego-vehicle is currently in is defined to be $i = 0$.

4.2 Interactive Planner

The interactive planner defines the MDP to be solved. The objective of the interactive planner is to allow the ego-vehicle to travel the maximum distance without collision before the episode timeout occurs.

State

Given the state of the leader $\mathbf{s}^l_{i,k}$ and follower $\mathbf{s}^f_{i,k}$ at line i, the neighbor vehicles state $\mathbf{N}_i$ w.r.t the inertial frame of the ego-vehicle is denoted as:

$$\mathbf{N}_{i,k} = \langle \mathbf{s}^l_{i,k} - \mathbf{s}^0_{i,k}, \mathbf{s}^f_{i,k} - \mathbf{s}^0_{i,k}, \rangle, \forall i \in [0, 1, 2] \cap \mathbb{Z} \tag{1}$$

The observation $\mathbf{O}_k$ available to the RL agent at time step k is defined as:

$$\mathbf{O}_k = \begin{bmatrix} \mathbf{s}^0_k & \mathbf{N}_{0,k} & \mathbf{N}_{1,k} & \mathbf{N}_{2,k} \end{bmatrix} \tag{2}$$

Note that the states of the MDP is defined to be the observation $\mathcal{O}$ that only includes the state of the ego-vehicle and 6 other neighbor vehicles instead of the entire state space $\mathcal{S}$ that contains the states of all vehicle.

Reward

The reward of the MDP is as same as the one defined by highway-env [1] and is engineered to encourage the ego-vehicle to make a decision that is fast and collision-free. For any time step k, the reward is defined as:

$$R_k(s, a) = \alpha \frac{v - v_{\min}}{v_{\max} - v_{\min}} - \beta \tag{3}$$

, where α controls the importance of travelling faster and β is the collision penalty.

Policy and Action

The action a is selected from the action space $\mathcal{A}$ that consists of one of the following action: merge left, merge right, speed up, slow down, and idle. The policy π is a stochastic discrete probability distribution of the action $\mathbf{A}$ given the observation $\mathbf{O}$, and such probability distribution is assumed to be a multinomial distribution.

$$\mathbf{A}_k \mid \mathbf{o}_k \sim \text{Multinomial}\,(1,\ \mathbf{p}) \tag{4a}$$

$$\mathbf{p} = \langle p_{\text{merge left}}, p_{\text{merge right}}, p_{\text{faster}}, p_{\text{slower}}, p_{\text{stay put}} \rangle^\intercal \tag{4b}$$

$$\pi(a_k \mid \mathbf{o}_k) = p(a_k \mid \mathbf{o}_k), \forall a_k \in \mathcal{A} \tag{4c}$$

4.3 Evaluation

The RL agents will be evaluated with the following evaluation metrics:

- Distance Traveled: One objective of the RL agent is to generate a sequence of decisions that gets the ego-vehicle to the destination in the shortest amount of time. Therefore, the faster the ego-vehicle is driving, the longer the distance the ego-vehicle could travel. Higher is better.

- Collisions: The percentage episodes terminated due to collision among all the episodes being tested. Lower is better.
- Steps Until Crash: For all the episodes that are terminated due to collision, the number of steps the ego-vehicle took until collision. Higher is better.

4.4 Baseline Comparison

As a comparison to our implementation, we will use the stable-baselines3 [14] Python library to train a model. The library contains implementations of several reinforcement learning algorithms and integrates seamlessly with highway-env [1]. In particular, we will use their DQN implementation, which uses a replay buffer, target network, and gradient clipping. Their networks train much faster than ours even for a higher episode count, which will allow us to experiment with more hyperparameter sweeps. We will compare the distance traveled, collision rate, and the number of environment steps before a crash to our implementation.

5 Experimental Results

In the following section, the performance of random policy is first described. The current performance of the proposed method is then reported. As a comparison, the performance of the DQN from stable baseline is used to compare with the proposed method.

5.1 Random Policy

The random policy is defined as choosing one of the action in the action space randomly with equal probability. The random policy is used to run 100 episodes. Out of 100 episodes, 91 episodes terminated crashing, i.e. around 90% of the episodes terminated crashing. For the episodes that terminated crash, the random policy on average takes 10 decisions to crash the vehicle since the simulation begins with standard deviation 6.91. The average total distance travelled is 308 meters with standard deviation 196.73 meters.

5.2 Simplified-IntMPC Implementation

The experiments are executed by pairing different environment configuration with DQN configuration. The environment configurations are shown in Table 1 and the DQN configurations are shown in Table 2. For every 5 training episodes, the network will be put to evaluation mode and be used to simulate 20 validation episodes. The DQN is trained with Adam optimizer.

In milestone report, it is realized that the DQN is suffering stability issue, i.e. the performance is oscillating as training episodes increase. Upon careful inspection, it is realized that there exists some issues within the DQN implementation. Therefore, the performance previously shown is not reported here. However, fixing the DQN implementation does not resolve the instability.

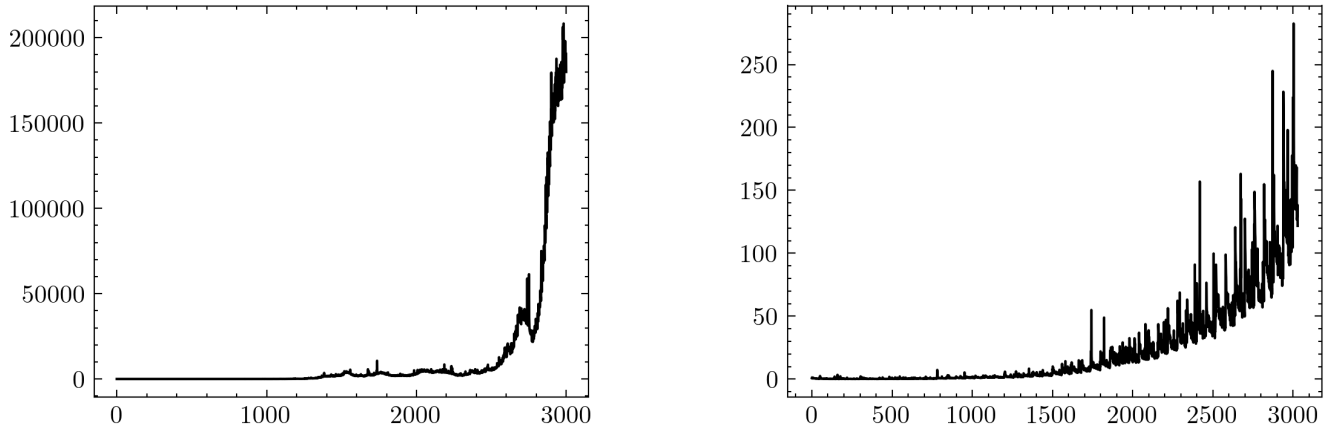
Originally, the reward signal is configured to encourage the agent to travel as far as it could within the given time frame. However, it was suspected that the observed instability might be caused by the reward signal being too complicated, i.e. the agent learned to drive as fast as it could but failed to learn not to collide with other vehicles. As a result, the fast speed reward α in both environment configuration listed in Table 1 is set to 0.0; as the reward for encouraging travelling faster α is set to zero, the reward speed range is shown as not available. Note that the only difference between the two configuration is the collision punishment β.

Name	Number of Lanes	Vehicle Counts	Initial Spacing	Max. Episode Steps	α	β	Reward Speed Range	Default Action
00	4	50	1.0	30	0.0	-1.0	N/A	Slower
01	4	50	1.0	30	0.0	-5.0	N/A	Slower

Table 1: The highway environment configuration.

Name	ϵ	ϵ Decay	ϵ Update	Discount	Training Episodes	Replay Buffer Size	Batch Size	Target Network Update
00	0.5	0.99	10	0.9	5000	1000	500	20
01	0.5	0.99	10	0.9	5000	1000	500	10
02	0.5	0.99	10	0.9	5050	5000	3000	100

Table 2: The deep Q-Network configuration.



(a) env_01_dqn_02

(b) env_01_dqn_03

Figure 1: The training loss trained under different DQN configuration. The vertical axis is the training loss and the horizontal axis is the number of optimization step. Due to the significant difference in magnitude, the training loss are not plotted together in a single plot. Note that increasing target network update frequency reduces the magnitude of blow up.

5.2.1 Training Loss

The training loss is realized to be one of the metrics that could reflect how the DQN is learning towards the end of the investigation. Therefore, only some experiments have the training loss data and is shown in Figure 1. Upon inspection, it is realized the training loss blows up as the training episodes increase. It is suspected that the observed situation is the unbounded divergence phenomena described in [15]. However, whether our DQN is actually suffering unbounded divergence is not scientifically verified. Note that increasing target network update frequency reduces the magnitude of blow up.

5.2.2 Validation Total Reward

The DQN is put into evaluation mode every 5 training episodes and be used to run 20 validation episodes. The validation total reward is the average of the total reward of the 20 finished validation episodes. The validation reward is shown in Figure 2. It can be observed that the validation reward is demonstrating an oscillatory pattern. It is suspected that the pattern is also related to the unbounded divergence phenomena mentioned in [15] as the network might be over-optimistic about certain state. However, such intuition is not yet verified.

Learning rate of the optimizer is later realized high learning rate 10^{-3} could be a contributing factor to the poor learning performance. Therefore, two of the experiments are run with smaller 10^{-5}. However, due to the limited time frame, not all the experiment configurations are tested with different learning rate. As a result, the relationship between learning rate and DQN performance cannot be drawn scientifically. But initial inspection on Figure 2 suggested that smaller learning rate does not improve DQN performance.

5.2.3 Distance Travelled

The distance travelled is collected in the way as same as total reward signal and is presented in Figure 3. Despite the oscillatory pattern shown in the reward signal, the DQN policy still outperforms the random agent in terms of the total distance travelled regardless of training configuration. Note that the only difference between the two

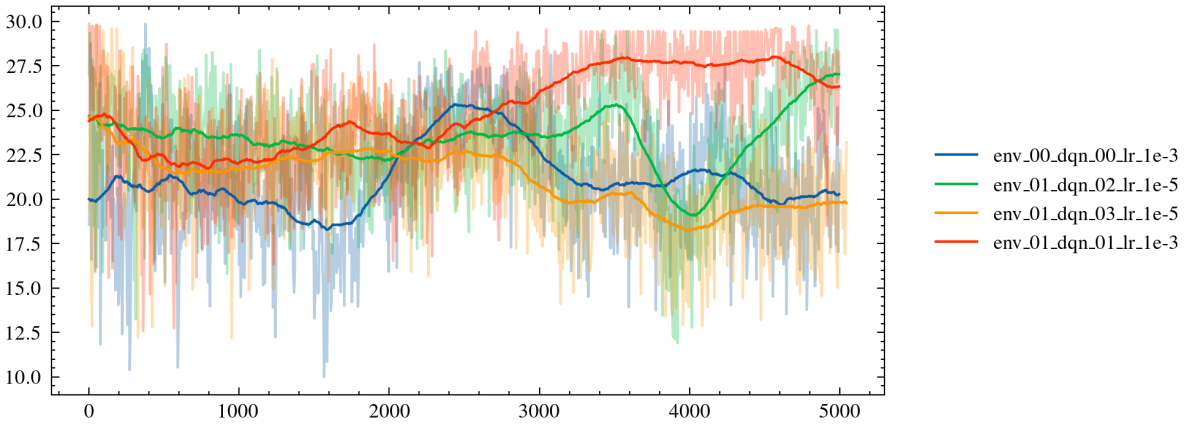


Figure 2: The validation total reward over training episode. The vertical axis is the validation total reward and the horizontal axis is the training episode. The original reward signal is presented with the semi-transparent line; the solid line is the running average of 100 consecutive validation rewards over time (training episodes).

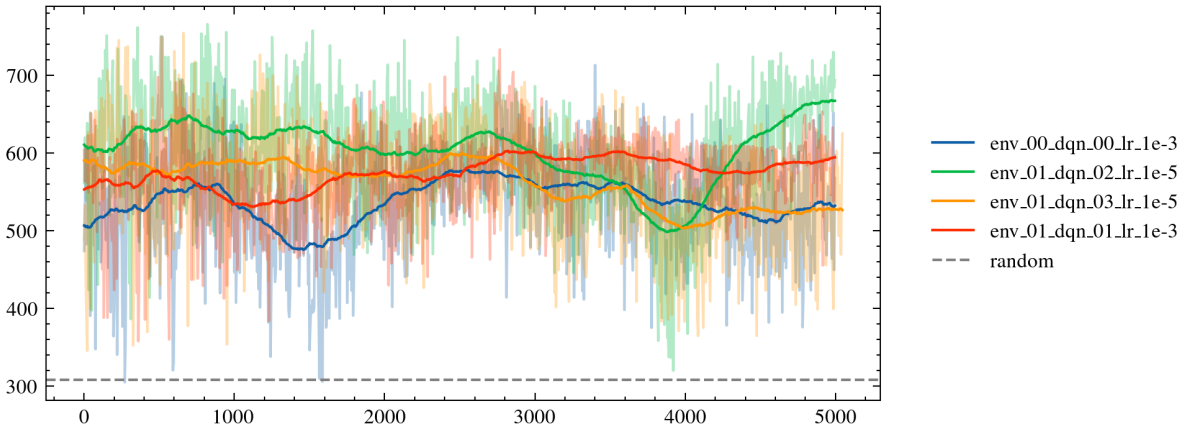


Figure 3: The validation distance travelled over training episode. The vertical axis is the distance travelled in meters and the horizontal axis is the training episode. The original distance travelled is presented with the semi-transparent line; the solid line is the running average of 100 consecutive distance travelled over time (training episodes).

environment configuration in Table 1 is the collision penalty β. However, the random agent is behaving randomly, which is independent of the collision punishment. Therefore, the performance of random policy is comparable across all experiment configurations.

5.2.4 Number of Collision

The number of collision is collected in the way as same as total reward signal and is presented in Figure 4. It can be seen that DQN policies still outperform the random agent in terms of number of collision regardless of training configuration. In previous section, it is shown that random policy crashed 90% of the time when being evaluated with 100 episodes. Therefore, the performance shown in Figure 4 is calculated by multiplying number of validation episodes by ratio of time it crashes, which is $90\% \times 20 = 18$.

5.2.5 Steps Until Crash

The steps until crash is collected in the way as same as total reward signal and is presented in Figure 6. For the episodes that actually terminated crash, the DQN and the random agent takes around the same number of decisions (steps) to crash the vehicle as the random agent. Upon inspection, the number of steps to crash the vehicle does not seem to remain constant throughout the entire training process.

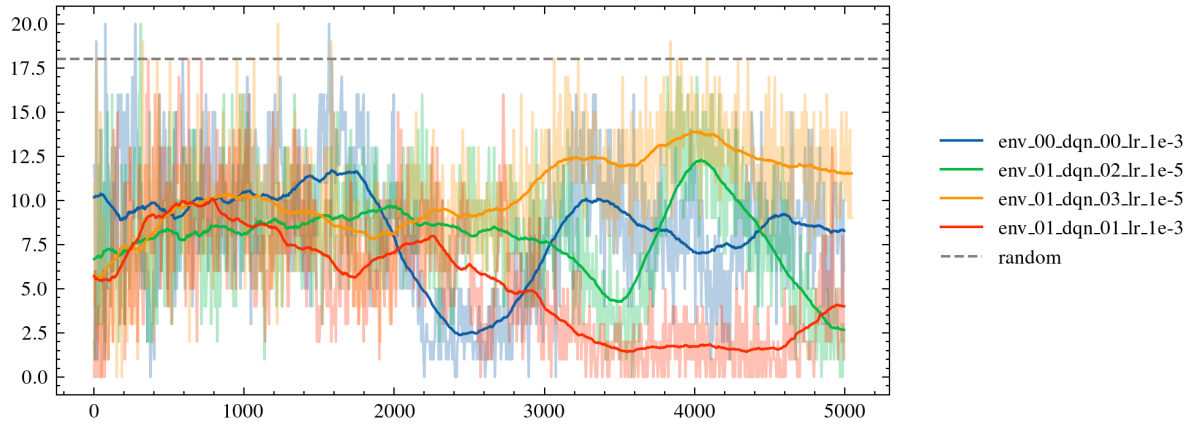
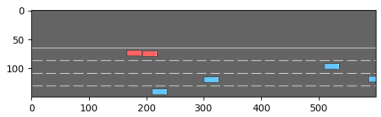
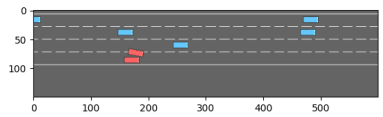


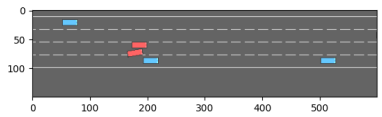
Figure 4: The validation number of collision over training episode. The vertical axis is the number of collision and the horizontal axis is the training episode. The original number of collision is presented with the semi-transparent line; the solid line is the running average of 100 consecutive distance travelled over time (training episodes).



(a) The ego-vehicle rear-ended the vehicle at the front.



(b) The ego-vehicle ignored the vehicle immediately next to it.



(c) Both vehicles were merging at the same time.

Figure 5: The screenshot of how the episodes terminated.

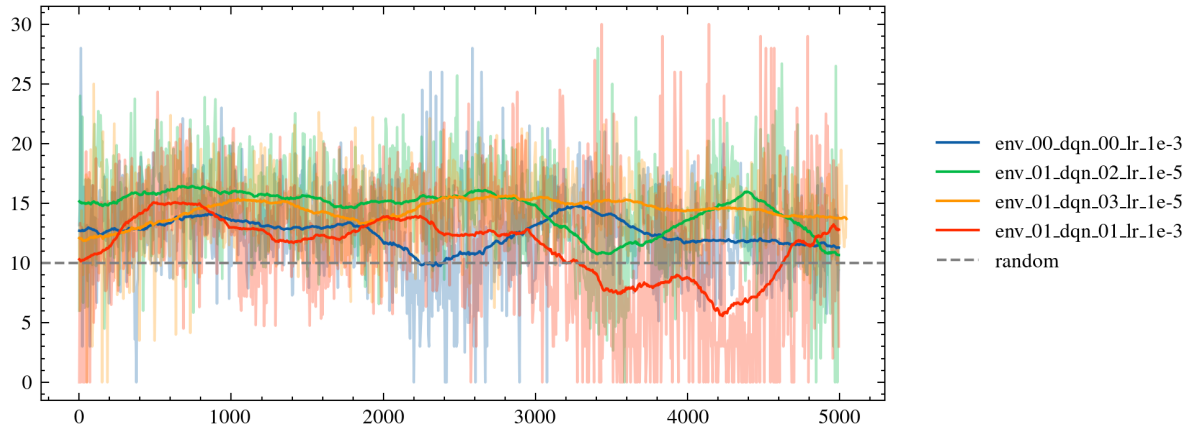


Figure 6: The validation number of collision over training episode. The vertical axis is the number of collision and the horizontal axis is the training episode. The original number of collision is presented with the semi-transparent line; the solid line is the running average of 100 consecutive distance travelled over time (training episodes).

5.3 Stable Baselines

The stable baselines uses the full observation of the entire environment. That is, the DQN receives a $V \times F$ array, where V is the number of vehicles in the environment and F is the number of features. In our case, F is the x- and y-positions and velocities of each vehicle. This means the stable-baselines implementation is not a direct comparison to our implementation, as our implementation uses only the nearby vehicles.

5.3.1 First Implementation

The stable-baselines3 DQN was trained on an environment with 100 vehicles for many permutations of the following hyperparameters: learning rate, replay buffer size, and number of hidden nodes in the network architecture. It was found that a larger replay buffer improved learning stability and that a learning rate on the order of 10^{-4} with 2 hidden layers of 256 nodes performed best. As a result, the following experiments used:

- Learning Rate: 0.0005
- Replay Buffer Size: 15000
- Network Architecture: 256 x 256

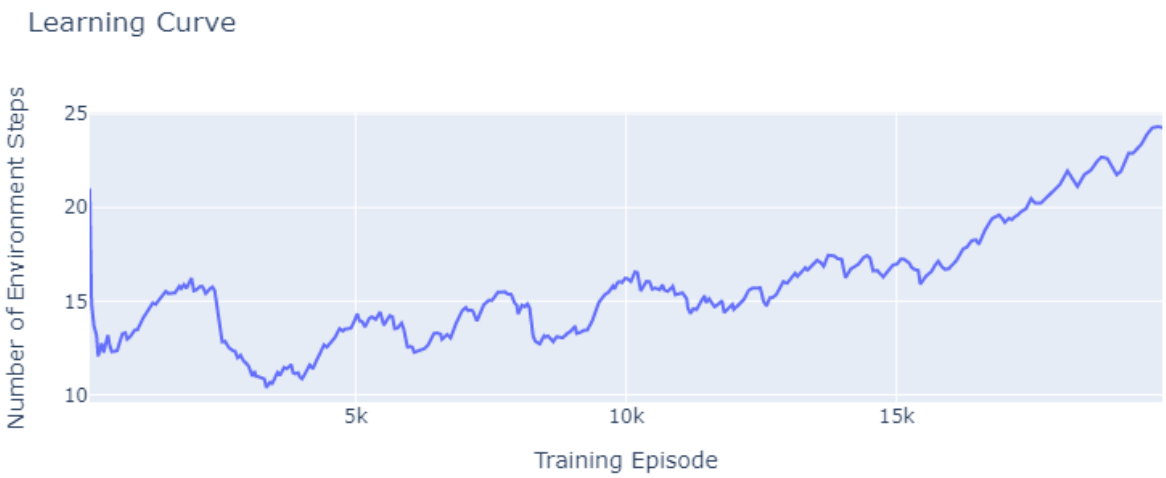


Figure 7: Learning Curve for first stable-baselines DQN

The training process showed some instability until reaching 15000 episodes, which was the size of the replay buffer. This model was simulated for 100 validation episodes and its performance was quantified using the travel distance of the ego vehicle. The mean travel distance was 791m with a standard deviation of 148m. 89% of the simulations completed without the vehicle crashing.

While these results performed better than our implementation, the autonomous vehicle continued to crash in ways that would be obvious to a human driver, such as increasing speed until it rear-ended another vehicle. These are indicated by the outlier points in Figure 8. Example videos are available on the Blackboard assignment.

5.3.2 Investigation of Environment Parameters

There were some shortcomings with the previous experiment setup. Notably,

- The environment used 100 vehicles, which meant that simulation and training was time-consuming even with multiprocessing.
- The episode duration was relatively low at a maximum of 40 steps. It is possible that the training did not see enough states, especially since the observation matrix is not always returned in a consistent manner. That is,

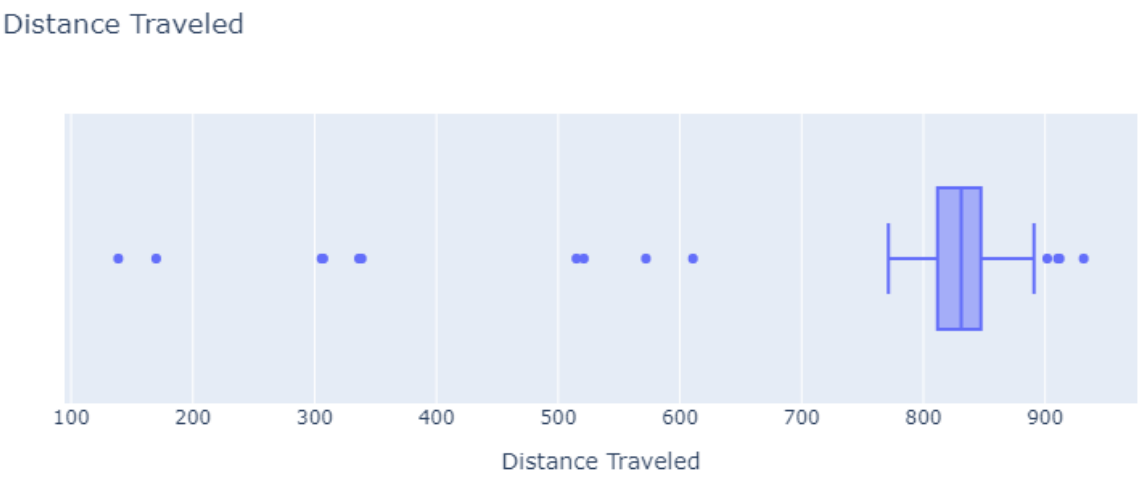


Figure 8: Travel distance for 100 simulations using the first stable-baselines model

the same physical arrangement of vehicles can have many possible matrices since the order of the vehicles is not always the same. This means the same arrangement of vehicles could be interpreted as a new state, even though it had been seen before.

- The initial density of the vehicles did not change, so even though there were more vehicles in the environment, the autonomous vehicle only had to avoid a few at a time.

To address these shortcomings, we aimed to reduce the number of vehicles, increase the density, and increase the duration of each episode. In addition, we decided to try slightly adjusting the learning rate as well. We performed DQN training on each permutation of the following:

- Vehicle Count: [25, 50]
- Vehicle Density: [1, 2]
- Duration: [40, 80]
- Learning Rate: [0.0001, 0.0005]

The most striking observation is how much worse the DQN learned with a higher vehicle density as shown in Figure 9. Increasing vehicle density results in an agent that performs much worse by comparison. It is possible this could be rectified by increasing the number of training episodes as the denser environment is more difficult to navigate, thus needing more time to learn a good policy.

The next most apparent partition of the learning curves is the learning rate. All the best performing models had a learning rate of 0.0005, as seen in the upper group in Figure 10.

Finally, both number of vehicles in the environment and the duration of the simulation seem to have no major effect on the learned models as shown in Figures 11 and 12. We observed that the best performing configuration had 50 vehicles, a learning rate of 0.0005, a vehicle density of 1, and a 40s episode duration. Though the 80s duration of this configuration performed nearly as well. We performed 100 validation runs for both of these models and recorded the same metrics. The model that was trained with a max episode size of 40 was run on an environnment that terminated at 40 steps. Likewise the model trained on a duration of 80 was run on an environment that terminated at 80 steps.

We would expect the distance traveled to be much larger for the model trained on a longer episode size, however this was not exactly the case. We can see in Figure 13 that the 80s duration does have a larger range of travel distances because some of its runs were able to complete the full 80s without crashing. However its variance is much higher and its median travel distance was 200m less than the 40s duration model. Both box plots cluster around the 400m - 900m range, which suggests that the environment maximum duration does not contribute to the

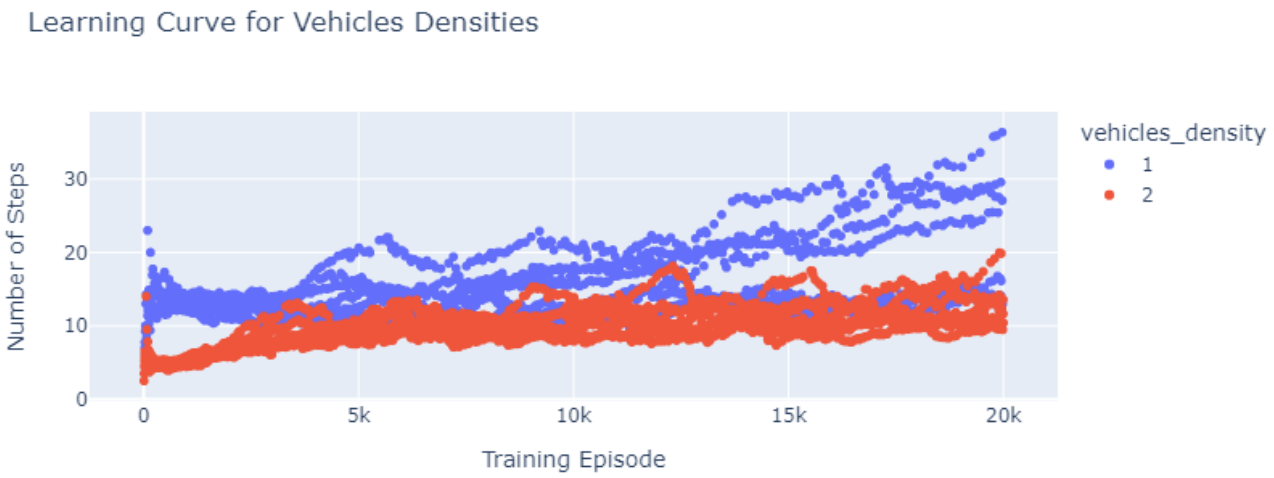


Figure 9: Learning curve for the environment parameter sweeps, colored by vehicle density

Figure 10: Learning curve for the environment parameter sweeps, colored by learning rate

Duration	Mean Distance	Median Distance	Mean Steps	Percent Crashed
40	669	849	28	49%
80	791	642	34	79%

Table 3: Validation results for best stable-baselines sweep cases

performance. It could be that the 80s duration model needs more training episodes due to the inconsistent ordering of the observation matrix as previously stated. In addition, both models performed worse than the original model which was trained on 100 vehicles, compared to the 50 vehicles used in this environment.

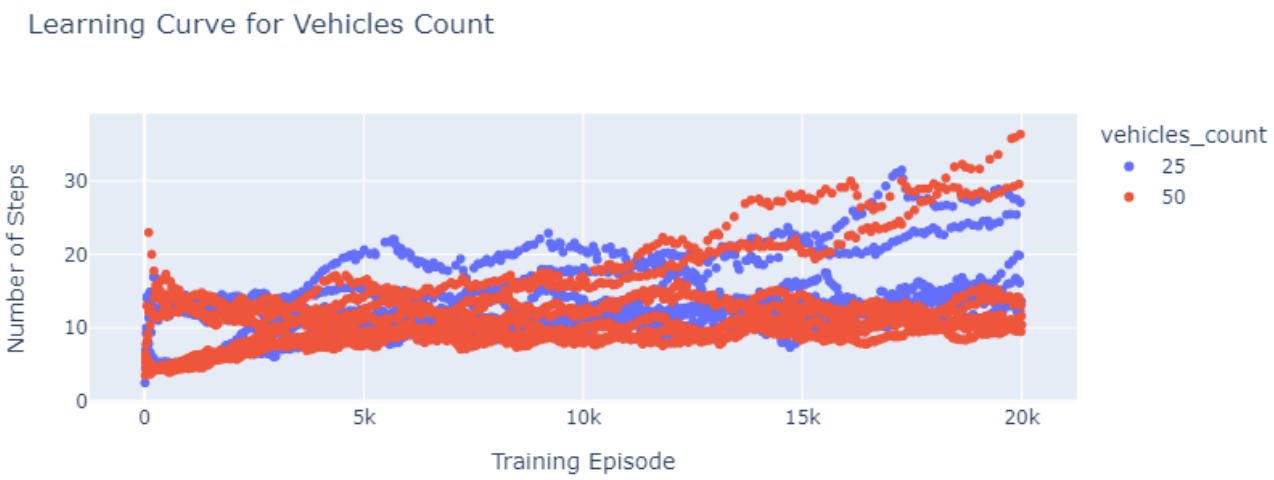


Figure 11: Learning curve for the environment parameter sweeps, colored by number of vehicles

Figure 12: Learning curve for the environment parameter sweeps, colored by episode duration

6 Conclusion

Previously, it was suspected that the DQN agent prefers continuous lane change over slowing down is caused by the α term in the reward signal. However, the DQN does not crash less even when the α term is set to zero, i.e. the agent is only encouraged not to crash the vehicle. Upon inspection, the reward signal is showing an oscillatory behavior throughout the training. Further investigation suggested that the training loss for DQN diverges overtime. However, reducing learning rate does not improve the situation. Therefore, it is currently suspected that the DQN is suffering the unbounded divergence phenomena described in [15], which the DQN is being over-optimistic about the action it is taking given the current observation.

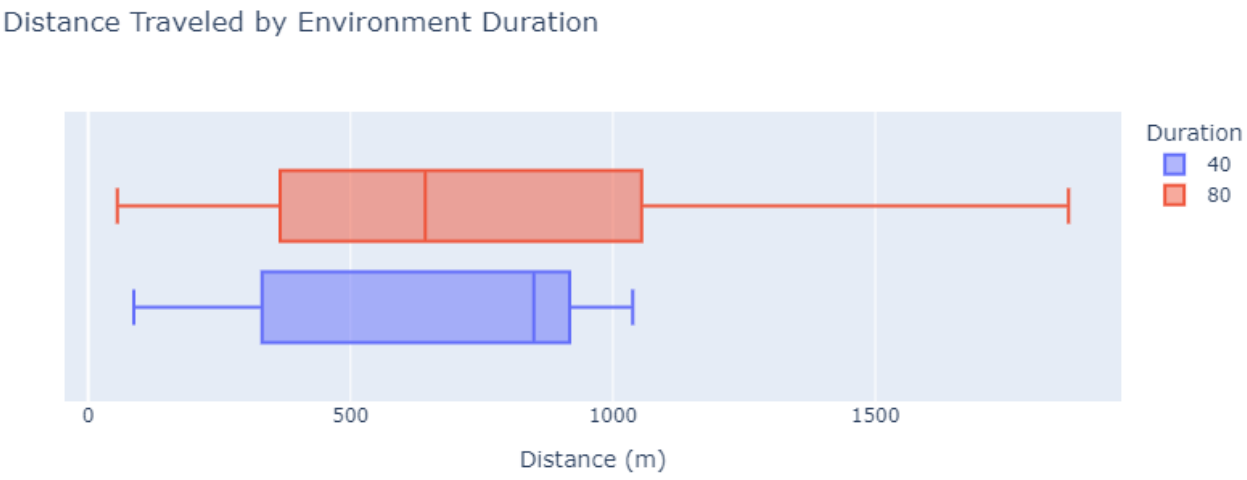


Figure 13: Distance traveled for the two best models, colored by max episode duration

.

References

[1] Edouard Leurent. *An Environment for Autonomous Driving Decision-Making*. `https://github.com/eleurent/highway-env`. 2018.

[2] Simon Ulbrich et al. "Structuring Cooperative Behavior Planning Implementations for Automated Driving". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015, pp. 2159–2165. DOI: `10.1109/ITSC.2015.349`.

[3] Bruno Brito, Achin Agarwal, and Javier Alonso-Mora. "Learning Interaction-aware Guidance Policies for Motion Planning in Dense Traffic Scenarios". In: (July 2021). arXiv: `2107.04538 [cs.RO]`.

[4] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (Dec. 2013). arXiv: `1312.5602 [cs.LG]`.

[5] Petros Christodoulou. "Soft Actor-Critic for Discrete Action Settings". In: (Oct. 2019). arXiv: `1910.07207 [cs.LG]`.

[6] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. "Planning and Decision-Making for Autonomous Vehicles". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (May 2018), pp. 187–210. DOI: `10.1146/annurev-control-060117-105157`.

[7] David González et al. "A Review of Motion Planning Techniques for Automated Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 17.4 (2016), pp. 1135–1145. DOI: `10.1109/TITS.2015.2498841`.

[8] Edward Schmerling et al. "Multimodal Probabilistic Model-Based Planning for Human-Robot Interaction". In: (Oct. 2017). arXiv: `1710.09483 [cs.RO]`.

[9] Simon Le Cleac'h, Mac Schwager, and Zachary Manchester. "LUCIDGames: Online Unscented Inverse Dynamic Games for Adaptive Trajectory Prediction and Planning". In: (Nov. 2020). arXiv: `2011.08152 [cs.RO]`.

[10] Pete Trautman. "Sparse Interacting Gaussian Processes: Efficiency and Optimality Theorems of Autonomous Crowd Navigation". In: (May 2017). arXiv: `1705.03639 [cs.RO]`.

[11] Dhruv Mauria Saxena et al. "Driving in Dense Traffic with Model-Free Reinforcement Learning". In: (Sept. 2019). DOI: `10.1109/ICRA40945.2020.9197132`. arXiv: `1909.06710 [cs.RO]`.

[12] Maxime Bouton et al. "Reinforcement Learning with Probabilistic Guarantees for Autonomous Driving". In: *Workshop on Safety Risk and Uncertainty in Reinforcement Learning, Conference on Uncertainty in Artificial Intelligence (UAI), 2018* (Apr. 2019). arXiv: `1904.07189 [cs.RO]`.

[13] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. "Congested Traffic States in Empirical Observations and Microscopic Simulations". In: *Physical Review E 62, 1805-1824 (2000)* (Feb. 2000). DOI: 10.1103/PhysRevE.62.1805. arXiv: cond-mat/0002177 [cond-mat.stat-mech].

[14] Ashley Hill et al. "Stable Baselines". In: *GitHub repository* (2018).

[15] Hado van Hasselt et al. "Deep Reinforcement Learning and the Deadly Triad". In: (Dec. 2018). arXiv: 1812.02648 [cs.AI].