

---

# Mastering the Highway using Rainbow DQN and PPO

---

**Jia Sheng Liu**

Department of Computer Science  
University of Bath  
Bath, BA2 7AY  
js186@bath.ac.uk

**Le Lyu**

Department of Computer Science  
University of Bath  
Bath, BA2 7AY  
112300@bath.ac.uk

**Zhe Yang**

Department of Computer Science  
University of Bath  
Bath, BA2 7AY  
zy756@bath.ac.uk

**Leonid Zhuravlev**

Department of Computer Science  
University of Bath  
Bath, BA2 7AY  
1z2091@bath.ac.uk

## 1 Problem Definition (1 Page)

A clear, precise and concise description of your chosen problem, including the states, actions, transition dynamics, and the reward function. You will lose marks for an unclear, incorrect, or incomplete problem definition. You should also discuss the difficulty of your chosen problem and justify why it cannot be solved effectively using tabular reinforcement learning methods.

The Problem Definition we are evaluating is of a highway, where a car is driving and has the goal of driving as fast as possible for a set amount of time (10 seconds for now) without crashing to other cars. This is a continuous problem because the car positions on the highway will be random and will follow some rules of the highway, such as giving way. So, it will require non-tabular reinforcement learning methods to efficiently learn from the most optimal method to traverse this highway environment.

### 1.1 States

The states will be different combinations of the 4 lanes of the highway and the random number of cars per lane. Not including the agent car, the amount of cars per lane should be at most 2.

### 1.2 Actions

The actions for any car on the highway environment is to stay in the same lane (idle), move left or right. Furthermore, continuous actions can be applied using kinematics where the speed of going left and right and the distance will also be taken into consideration when training the agent using non-tabular methods.

### 1.3 Transition dynamics

The transition dynamics of the agent will be the probabilities of going to each of the lanes next to the agent, as well as staying in the same lane. Furthermore, there is also the probability of the agent to crash for each lane the agent is on.

### 1.4 Reward function

As the goal of the agent is to be as fast as possible, the reward function will be negative rewards when the car crashes. However, the negative rewards need to be balanced so the agent isn't scared of going

too fast because it is afraid of crashing. Moreover, the agent will be intrinsically motivated to get ahead as possible in the time it is given.

## **1.5 Difficulty and challenges**

# **2 Background (< 1 Page)**

## **2.1 Existing Literature and Results**

Several reinforcement learning (RL) solutions have been implemented for highway driving tasks. In general, they can be divided into two main classifications: policy-based methods, which optimize the reward via gradient ascent (?), and value-based methods, which select the next action based on the cumulative benefit of each action.

### **2.1.1 Policy-Based Methods**

Policy-based methods, particularly those utilizing Proximal Policy Optimization (PPO), have emerged as a dominant solution due to their effectiveness in handling a wide range of challenging tasks and their simplicity in implementation and tuning (?).

A key advantage of PPO-based methods is their ability to integrate high-level decision making. For example, (?) used a customized reward function to penalize risky actions and generate different driving styles, including behaviors such as “keep lane,” “overtake,” and “move to the rightmost lane.”

To simplify decision-making and make solutions more interpretable, these solutions often adopt a hierarchical architecture that separates the Decision Maker (DM) and the Behavior Executor (BE) (?), (?), (?). The Decision Maker focuses on selecting high-level strategic actions, such as “keep lane” or “overtake,” while the Behavior Executor translates these decisions into low-level actions controls. Hence, simplifies decision making and allows modularity.

However, while effective, these PPO-based solutions can be unnecessarily complex for simpler highway environments with discrete action spaces, such as the one in our chosen problem. Their performance often heavily relies on the definition of policies, which may limit adaptability. For example, the stabilizing decisions in (?) may restrict the agent’s ability to respond dynamically to changing traffic conditions. Similarly, fixed driving styles (e.g., “aggressive” or “comfort”) implemented in (?), while realistic, may not align with our goal of developing an adaptive agent.

### **2.1.2 Value-based methods**

Value-based methods, specifically Deep Q-Networks (DQN), are the primary solution for highway driving tasks. DQN estimates the action-value function (Q-values) to determine the optimal action for the agent at each state, making it particularly well-suited for environments with discrete action spaces.

A significant strength of DQN lies in its ability to handle complex decision-making through the use of attention mechanisms. For instance, (?) and (?) introduced techniques that prioritize vehicles most likely to influence the agent’s decisions.

At the same time, DQN maintains a high degree of interpretability, which is crucial for understanding and analyzing the agent’s behavior. (?) employed SHapley Additive exPlanations (SHAP) to quantify the importance of environmental factors (such as vehicle speed, position, and distance) that influence specific decisions. Furthermore, (?) provided multiple visualization views for SHAP values. These tools enable a detailed analysis of the agent’s decision-making process across varying traffic conditions and timeframes.

While existing DQN provides a solid foundation, there remains significant room for improvement, particularly in enhancing learning efficiency and performance stability.

# **3 Method (2 Page)**

A description of the method(s) used to solve your chosen problem, an explanation of how these methods work (in your own words), and an explanation of why you chose these specific methods.

Students should demonstrate a good understanding of their chosen method(s) and give reasonable justifications for their algorithmic choices. This should include demonstrating a reasonable appreciation of the strengths and weaknesses of alternative methods.(60%)

Students may show evidence of creativity in their algorithmic approach and analysis. Students should demonstrate a deep understanding of their chosen method(s), as well as possible alternative approaches. (70%)

### 3.1 Deep Q-Learning Network (DQN)

DQN used to solve your problem and how they work, and why I chose you it.

Students should demonstrate a good understanding of their chosen method(s) and give reasonable justifications for their algorithmic choices. This should include demonstrating a reasonable appreciation of the strengths and weaknesses of alternative methods.(60%)

show evidence of creativity in their algorithmic approach and analysis. Demonstrate deep understanding of DQN and possible alternative approaches.

DQN is a value based deep reinforcement learning algorithm that combines Q-Learning with deep neural networks to solve our environment as the state space contains too many features that make it hard to handle using a tabular method. Therefore a neural network generates an output of the estimated Q-values of all possible actions given the state. For the chosen problem, there are two types of states that exist are Kinematics and Grey Scale image.

Kinematics refers to an array of nearby vehicles and their features. Some features are the  $x, y$  position of either the agent vehicle or other vehicles offset with the agent vehicle,  $x_{velocity}, y_{velocity}$  of the vehicles. A Multilayer perceptron (MLP) neural network will be used to take in this state and get the Q-values of the available actions. A MLP can learn the nonlinear relationships between data as it uses multiple layers of neurons with non-linear activation functions (ReLU). Then using the Q-Values of the network we get the action with the highest Q-Value. However, as the output is vehicle order dependent, it might not be able to generalize the network towards vehicles in different orders.

This is the reason for Grayscale Images, where we process the scenario of the environment with the road and the vehicles. This provides a more realistic and accurate representations of the state. Therefore, a Convolutional Neural Network (CNN) is used to process the grayscale image of the environment instead of getting the features of nearby vehicles. Moreover, the CNN can also take channels of the image normally represented as  $r, g, b$ , however as we are using grayscale we can use a set number of previous frames as the channels for a broader representation of the state. Nonetheless, this will be computationally heavier as it will take more dimensions and more data than an array.

Furthermore, the inputs for the networks are the states. DQN processes the states of the by first storing them into a memory replay circular buffer to be accessed after each time step. This is so that it can have a equal distribution of all the states that the agent has seen, so that it wouldn't forget any previously seen states. Nonetheless, the buffer still has a limit and so old states will be overwritten when new ones come in and the buffer is full. As the agent needs to learn at the start of the training, the agent will traverse through the environment an just store the state relevant information in there.

As the agent learn from the memory replay it uses the Bellman equation on the target network versus the trained network for the best possible action against the action chosen and its discount value. Denote with the equation (not sure if it should be put) Then used the mean square error loss as the Huber Loss (maybe it should be L1 smooth loss) to get how far the best action is from the action chosen and update the q training network's parameters using gradient descent and backpropagation to output an action for similar states. Furthermore, to prevent exploding gradients I have clipped the gradients to  $[-1, 1]$  ? for more stable training.

Epsilon-Greedy Policy: This uses epsilon decay to slowly decrease epsilon which decreases the amount of randomness that the actor performs. Instead of having a constant that goes down slowly regardless of the total amount of episodes defined where the epsilon might be big before ending the training. Linear annealing schedule was used to decrease the epsilon depending on the total amount of episodes. A float (0 to 1) will be assigned to decide if the actor should reach the minimum epsilon during 1 to 100% of the training

Furthermore, the following extensions for Rainbow DQN were implemented based on the results it showed during the evaluation of the model and the metrics. The metrics are stored using tensorboard to keep track of the average rewards, exploration rate (epsilon), episode length and the average loss of the episode. Moreover, due to the limitation of our personal computers, Google Colab was used to train the model. As Google Colab can't render the environment for displaying purposes, saving and loading the parameters and buffers of the neural networks. As the rendering of the model is done locally, loading of the model was done locally and the videos were recorded. I could also import a package to automatically record a video of the render, but due to time and priorities, it wasn't achieved,

## 4 Rainbow DQN

Rainbow DQN refers to various extensions on top of the basic DQN to improve the accuracy, speed, and efficiency of the agent to reach the most optimal policy. The extensions N-Steps, Double, Noisy Network, Prioritized Experience Replay (PER) and Dueling Network were implemented.

N-steps is an extension to the Bellman update from the Replay Memory. Depending on the  $n$ , we retrieve  $n$  states after the chosen one to capture the future information of the states and then aggregated those  $Q$  values into the selected state. This increases the accuracy of the representation of the memory by reducing the variance and learns more about the states with less episodes.

Moreover, the max operator of the actions values tends to prefer overestimated (higher) values instead of the lower ones. Therefore, it would continuously select the highest action value instead of getting the values of the best action, learning to a suboptimal policy. Double DQN then takes the best actions from the trained network and uses it to get the probability ( $Q$  values) of that action in the target network loss to learn the best action in the state. This will stop the target value when overestimating the action values by separating the action selection and evaluation.

The Noisy Network is a Network or a Layer that is used for explorations instead of epsilon-greedy policy which is predetermined decrease of randomly selecting actions which is suitable for simpler environments. This paper, adds noise to the weights ( $\sigma$  and  $\mu$ ) to the weights of a network for the network to slowly adapt to the noisy over a period of time resulting in more adaptive exploration. The intensity of the noise is the epsilon which is initialized in the beginning of the network and sampled using the Factorized Gaussian Distribution. Put the equation here:

Noise was implemented using the two weights ( $\sigma$  and  $\mu$ ) as two vectors being of the size of the input and output respectively. Factorized is used to decrease the sample amount and also ensuring non-negative values to make it more scalable

PER is a new Memory Replay for storing and sampling memories based on their importance, which how much loss they generate. Instead of randomly sampling experiences, PER is used to sample a proportion of the states based on how much it influences the agent. The storage uses a Sum Segment Tree to store the indexes of the memories and their corresponding priorities to be retrieved as the leafs and the parent nodes as the sums. The leafs are stored as  $(2 * \text{capacity}) + \text{index}$  and then propagate upwards to update the sum all stored in an array. Then the average sum of the priorities in the tree is used to get the index with the highest priority. These indexes retrieve the transitions and its equivalent weights to how much should the loss be punished to be updated. This utilizes importance sampling  $(1/N * 1/P(i))^\beta$  to compensate for when  $\beta$  is 1 for non-uniform probabilities. The retrieval of the elements are based on dividing the index by two to get the sum of it.

Prioritized replay introduces bias because it changes this distribution in an uncontrolled fashion, and therefore changes the solution that the estimates will converge to (even if the policy and state distribution are fixed).

Correct bias using importance-sampling (IS) that fully compensates for the non-uniform probabilities  $P(i) i^\beta = 1$ .

Dueling Network consists of 2 additional layers that separate the estimation of the state value and the action advantage function. The action advantage function is the difference between the  $Q$ -values for a state. The state value ( $V$ ) and the advantage value ( $A$ ) are then aggregated layer to learn the  $Q$  value of the state without considering its actions, and weighting how important it truly is. I take the mean of the advantage layer to normalize the output so  $V(s)$  doesn't contain information about the action values. The benefits during training is that because it uses the TD error, it separates updating

the state value function and action advantage function to reordering the best actions. Furthermore, only one output neural (the action chosen) is being updated at a time, so the separation leads to more stable learning and generalizability across different states and actions.

## **5 Proximal Policy Optimization (PPO)**

Building upon these value-based RL methods, we implemented PPO, a policy-based method to explore policy optimization as an alternative to value iteration. PPO's recent successful applications in similar highway driving tasks (Section??) suggest that it is a reasonable candidate for our implementation. By introducing PPO, we sought to leverage its capacity for stable policy updates and explore its potential for achieving more adaptive behavior in dynamic traffic scenarios.

## **6 Results (1 Page)**

To evaluate the quantitative results, the tensorboard's results display how well the agent trains for the environment.

Display the different Extension of the Rainbow as well as MLP and CNN.

A presentation of your results, showing how quickly and how well your agent(s) learn (i.e. improve their policies).

include informative baselines for comparison best agent human agent random agent

## **7 Discussion (< 2 Page)**

An evaluation of how well you solved your chosen problem.

evaluate the performance of their agent(s),and include comparisons with key baselines and some alternative approaches.(60%)

perform an in-depth evaluation of their chosen method(s) and include comparisons to and evaluations of additional baselines and alternative methods.(70%)

Monte Carlo

DQN The implementation of Rainbow DQN showed that MLP wasn't the best policy since it didn't have as much information as with CNN. Therefore, as most of the features were implemented using the CNN policy, it would result in a better outcome.

TODO: talk about the difference between the different stuff

PPO

Overall, it still does seem that this environment is still more suitable for humans to control as the human agent performed the best and got the best results. So, it can be said that self-driving cars would require either as many senses as humans and be trained for tremendous amounts of times to reach a level that humans are at.

## **8 Future Work (0.5 Page)**

A discussion of potential future work you would complete if you had more time.

Potential future work that could be completed with more time is to finish PPO and make it better than DQN as shown in some other results.

## **9 Personal Experience (0.5 Page)**

A discussion of your personal experience with the project, such as difficulties or pleasant surprises you encountered while completing it.

There were various difficulties with the implementation of Rainbow DQN. Firstly, the neural networks used (MLP and CNN) were initially used softmax when getting the probabilities, however this proved that large values for the CNN at the start of training proved that it would converge to a policy that would produce the same results as softmax makes the probabilities the same and would always choose the first one during argmax. Therefore, to debug the neural network, the CNN was simplified and removed the softmax to fix issues regarding the actions chosen from the policy.