Tactical Decision-Making Strategy for Autonomous Driving using Deep Reinforcement Learning Techniques

Final Project Report

By

Team Members

Sivaranjani Kumar
Vigneshkumar Thangarajan

December 12th, 2020

Table of Contents

Chapter 1. Project Overview

1.1 Project Summary:

The high-level decision-making strategies for autonomous driving have seen less progress compared to other areas. It's because only few use cases are addressed and most of them cannot scale to more complex scenes, when decision-making involves interacting with other human drivers, whose behaviors are uncertain and difficult to model explicitly. The cost involved in collecting human driving data is very expensive and not possible to record all the possible driving scenarios. The promising approach to achieve this without much hassle is to train a decision-making policy using reinforcement learning. RL leverages driving agent to automatically learn a complex driving policy to imitate the human driving decisions. The goal of our project is to train an autonomous driving agent to drive efficiently by following the safe maneuvers to overtake other driving vehicles in a simulated highway environment. To achieve this, we have implemented two DRL techniques, DQN and DDQN with Dueling Network Architecture which will increase the efficiency of driving agent in a highway environment.

1.2 Introduction:

Interests in autonomous vehicles have seen great increase in recent years, from automakers to high-tech companies and research institutions. The main purpose of making an autonomous vehicle is to avoid human errors, reduce accidents and traffic congestion. A deep reinforcement learning plays a vital role in achieving this. Decision making strategy in the autonomous vehicles are prime area to focus since it is a critical task in overtaking. The DQN algorithm which combines the Q-learning and neural network is used to find the optimal decision-making policy for providing better driving experience. Deep reinforcement learning is used as a strong tool for dealing with sequential decision-making problems. Many attempts have been made to study the complete autonomous vehicle with error free. To make the system error free, autonomous vehicles should be trained in a way that the penalties earned by an agent should be as low as possible. The highway decision-making problem using the deep reinforcement techniques achieves a safe and efficient driving environment with better performance. We trained and compared the performance of two DRL techniques DQN and DDQN with dueling architecture which improves the driving efficiency of an agent.

Chapter 2. Project Definition

2.1 Problem Statement

For autonomous vehicles to operate efficiently, it needs to have four significant modules, perception, planning, decision-making and control. In this, decision-making policy is important since it requires human level perception to take actions when driving in a highway environment. Highway environments consist of multiple lanes where the ego vehicle and the surrounding vehicles can drive. The ego vehicle exhibits certain driving capabilities and tries to overtake the nearby vehicle. The behaviors of an ego vehicle which is a driving agent include acceleration, braking, lane-changing and lake-keeping. The problem to be addressed is generate the decision-making strategy in a way that it mimics the human behavior and follow safe maneuvers while overtaking other vehicles in a highway environment.

2.2 Motivation

The motivation for making the vehicle to operate autonomously is avoid the reduce accidents and traffic congestions. Moreover, autonomous driving involves human lives, and every life must be protected in terms of providing safe driving. The idea behind creating autonomous vehicle was to avoid loss of lives because of human errors and efficient driving. There are so many possibilities and scenarios that must be learned by the AI system in vehicles before it could be go completely automatic on road, which involves years of training and still, we are far away from achieving it. Collecting all scenarios for driving data is not possible. Recent advances in reinforcement learning have made it possible to train the driving agent which learns automatically by following a decision-making policy and mimics the human driving behavior.

2.3 Contribution

Team Members	**Contributions**
Sivaranjani Kumar	• Setting up the highway environment for autonomous driving. • Creating configuration files for driving environment. • Running simulations for highway driving. • Ablation study • Model training - Deep-Q Network algorithm. • Optimizing and testing the DQN agent to make optimal decisions.
Vigneshkumar Thangarajan	• Requirement and dependency identification for DDQN based RL

	agent. • Model Training - DDQN with Dueling architecture algorithm. • Hyperparameter Tuning. • Creating comparisons for two models. • Optimizing and testing the DDQN agent with dueling architecture to make optimal decisions.

Table 1. Team members and Contributions

Chapter 3. Project Implementation

3.1 Methodology

3.1.1 Highway Driving Environment

The highway environment consists of multiple lanes where the ego vehicle which is a driving agent and other surrounding vehicles can drive. The agent is driving on a straight highway with several lanes, and is rewarded for reaching a high speed, staying on the rightmost lanes, and avoiding collisions. The reward points for the agent are specified in the config file.

```
RIGHT_LANE_REWARD: float= 0.1

HIGH_SPEED_REWARD: float= 0.4

LANE_CHANGE_REWARD: float= 0

COLLISION_REWARD: -1

CONTROLLED_VEHICLE: 1
```

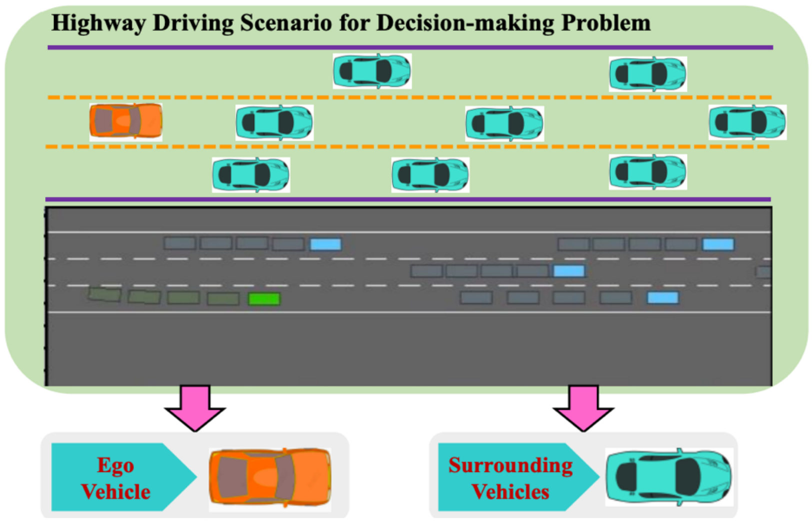


Figure 1. Highway Driving Environment

The action is executed by the ego vehicle and all other surrounding vehicles exhibit the default behavior for several simulation timesteps and they deigned randomly to create uncertainties in the driving environment. The behaviors of an agent involve lane changing, lane-keeping, acceleration, and braking. Accelerating and surpassing other vehicles is a typical driving behavior called overtaking. The movements of all the vehicles in the highway environments are mastered by the hierarchical motion controller.

Hierarchical Motion Controller: To manage the lateral and longitudinal movements of the ego and surrounding vehicles. It consists of two parts. The upper level contains two models, which are Intelligent driver model (IDM) and Minimize overall braking induced by lane changes (MOBIL). The lower level focuses on regulating vehicle velocity and acceleration. The original speed of the ego vehicle is chosen from [23, 25] m/s and the maximum speed of vehicle is 40 m/s. The length and width of all vehicles are 5m and 2m. The initial velocity of the surrounding vehicles is randomly chosen from [20, 23] m/s, and their behaviors are manipulated by IDM and MOBIL. Car-following and collision free is handled by IDM and it is implemented for Adaptive Cruise Controller (ACC). The longitudinal acceleration of the vehicle is determined by IDM as follows:

$$a = a_{\max} \cdot [1 - (\frac{v}{v_{tar}})^{\delta} - (\frac{d_{tar}}{\Delta d})^2]$$

v and a – current speed and acceleration.

amax - Desired speed.

Vtar and dtar – target velocity and distance.

amax and dtar – desired speed is achieved.

3.1.2 RL Concept

Reinforcement learning defines the learning process as agent interacts with the environment and the goal of the agent is to maximize the total cumulative reward overall. The policy, which maximizes the total cumulative reward is called the **optimal policy**. The main idea here is to find the optimal policy by using the Bellman's equation which describes the equation as optimal values of a state S, is equal to an action A which gives the maximum possible expected immediate reward R, plus the discounted long-term reward for the next state S':

$$\begin{aligned} &V_{\pi}(S) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = S] \\ &\pi = policy \\ &\gamma = [0.95 - 0.99] \end{aligned}$$

But the action-value function or Q-function represents the agent's estimate of how good it is to take an action A in state St which will help the agent to determine what actions need to be taken in the respective state to gain maximum reward and equation as follows:

$$Q(S, A) = E[R + \gamma V(S_{t+1})]$$

Where E is the expectation that gives the sum of probability of an element and R is the reward and V(S) is the value function. The state-action function should be updated in the recursive form and the equation is given as follows:

$$Q(S, A) = Q(S, A) + \alpha \left(R + \gamma \underbrace{\max Q(S', A')}_{A' \in \mathrm{A}} - Q(S, A) \right)$$

As the environment goes bigger, the number of potential states could be very large. Even to calculate a rough approximation for every state in an environment, we will need hundreds of billions of transitions evenly distributed over the states which is not practical. For example, in the highway environment, if we use raw pixels as individual states, then we have too many states to track and approximate values for. The other problem with traditional RL is it has limitations to discrete action spaces and cannot solve continuous control problems.

3.1.3 Deep Q Network

The traditional Deep reinforcement learning are the powerful to deal with the long sequential problems and it is independent of the historical data. But DRL was unable to address highway overtaking problems because of the continuous action space and large state space. To overcome this, we use deep neural network which is a non-linear function approximator to map the state and action into a value. When it is used in context of Q-learning, it refers to the Deep Q-Network (DQN). It is the first DRL methods proposed by the DeepMind. Deep neural networks empower RL to directly deal with high dimensional states like images. Motivation for Deep Q Network to solve this problem is – vast number of states causing numerous combinations of possible state-action combinations. Highway environment is ever changing and requires lot of time to explore all the state. The following equation explains the update value of Q-function:

$$\overset{\text{Update Current Q-value}}{\boxed{Q(s_t,a_t)}} \leftarrow \underset{\text{Current Q-value}}{\boxed{Q(s_t,a_t)}} + \overset{\text{Learning Rate}}{\boxed{\alpha}}\Big(\underset{\text{Rewards}}{\boxed{R_{t+1}}} + \underset{\text{Discount factor}}{\boxed{\gamma}}\overset{\text{Estimated reward from next action}}{\boxed{\max_{a_{t+1}} Q(s_{t+1},a_{t+1})}} - \boxed{Q(s_t,a_t)}\Big)$$

3.1.4 Dueling DQN Algorithm

Dueling DQN has two separate network one to estimate the state-value and the other one to estimate the advantages for each action in that state. This solves one of the specific problems to this domain. In environments like Atari single agent games and highway environment like this, at most of the states the agent action is not relevant. For example, when there are no chance of collision in any lane, the agent action is of least important. Dueling architecture addresses the problem of Q value overshooting and prevents model to learn from high Q values.

$$Q^{\pi}(s,a;\theta) = V^{\pi}(s;\theta_1) + A^{\pi}(s,a;\theta_2)$$

$$Q^{\pi}(s,a;\theta) = V^{\pi}(s;\theta_1) + (A^{\pi}(s,a;\theta_2) - \max_{a'} A^{\pi}(s,a';\theta_2))$$

$$a^{*} = \arg\max_{a'} Q(s,a';\theta) = \arg\max_{a'} A(s,a';\theta_2)$$

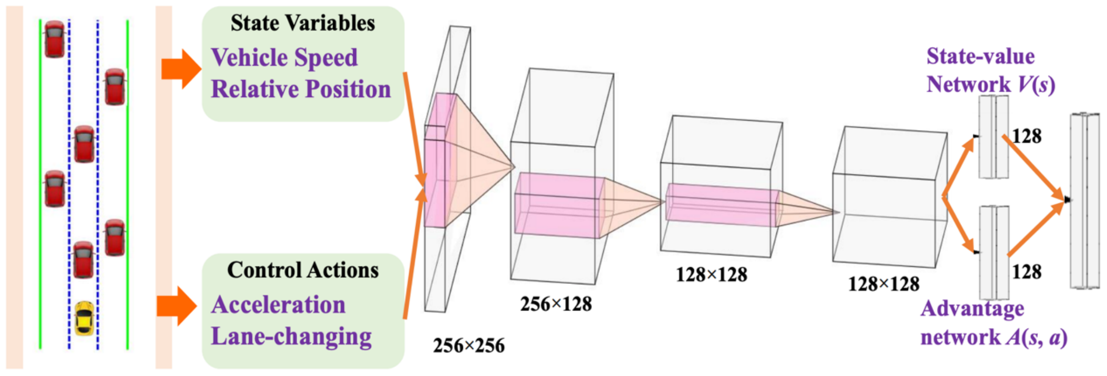


Figure 2. Dueling DQN Architecture

The Dueling DQN architecture is modification is a neural network architecture modification. So, this improvement can be used in any of the Deep Reinforcement learning models like Double DQN, SARSA etc., In figure 2. we can see that the input the CNN is state variables and control actions. The state variables are nothing but the relative position of the ego vehicle and vehicle speed. Vehicle speed is retrieved from the simulation environment by inputting a consequent sequence of images which are snapshot of the agent taking step in environment. The next three layers are convolution neural network followed by a fully connect layer. This fully connected layer is split into two – one layer caters to the computation of state value network and another caters to the computation of Advantage value the actions. Finally, the output of these two layers is aggregated and loss is calculated.

3.2 Implementation and Results

3.2.1 Highway Environment

The ego-vehicle is driving on a multilane highway populated with other vehicles. The agent's main objective is to drive efficiently by following safe maneuvers while avoiding collisions with neighboring vehicles. The observations, actions, rewards, and dynamics_of an environment are parametrized by a configuration, defined as a **config** dictionary. For highway environments, several types of observations can be used. They are defined in the observation module. They are Kinematics, grayscale, occupancy grid and time to collision. In our project we have implemented kinematics observation which creates an array that describes the nearby vehicles by a feature set of size which helps the agent to communicate with the surrounding vehicles.

Terminologies:

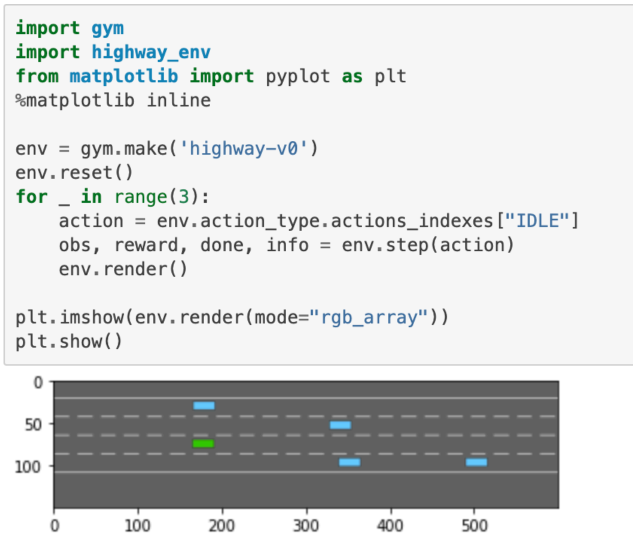
Agent – Ego Vehicle

Environment- Highway Environment

Action – Lane Keeping, Lane Changing, Acceleration and Braking

Reward – Positive value for avoiding collision and negative value for collision.

Below is the sample code which implements the highway environment.



```
import gym
import highway_env
from matplotlib import pyplot as plt
%matplotlib inline

env = gym.make('highway-v0')
env.reset()
for _ in range(3):
    action = env.action_type.actions_indexes["IDLE"]
    obs, reward, done, info = env.step(action)
    env.render()

plt.imshow(env.render(mode="rgb_array"))
plt.show()
```

Figure 3. Sample code and simulation for highway environment

3.2.2 Deep Q-Network

Q-learning: Q-learning algorithm is used get the projected future reward after an action has taken place. It gives the long-term value of taking an action A in state S. These actions maximize the average discounted future reward. Its explorations are completely decoupled form the action, but it has limitation to discrete action space. For continuous state space like highway environment, we use deep Q-learning.

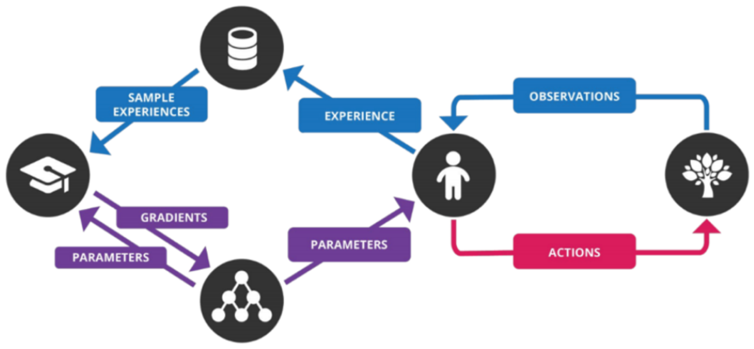


Figure 4. DQN Architecture

Deep Q-learning: Deep neural networks are functional approximators and its takes only state as input and gives Q-value as output for each action. To overcome the non - **i.i.d** problem to RL which is independent and identically distributed of input data and correlation issues in the sampling of input data which will cause the model to overfit, DQN used Experience Replay buffer and target network to predict the Q-value.

Experience Replay: This replay buffer stores the experience which is received from the environment after an action has taken by the agent. It stores transition states, rewards, actions, and states which are necessary data to perform Q learning. These are sampled as random mini batches to update neural networks.

To measure the discrepancy between the approximated and actual Q table in DQN, the loss function is introduced like the following expression:

$$Loss = (r + \gamma max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2$$

DDQN: In DQN**,** since the target Q value continuously changes with each iteration and training becomes unstable as it has one network to calculate the target and prediction value. In Double DQN we have two identical networks – one to learn the Q value and the other to learn the target value.

Target Network and Prediction network - Using separate network to estimate the target and has the same architecture as function approximator. After c iterations, the parameters from the prediction network are passed to target network. This leads to stable training because it keeps the target functions fixed.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[\boxed{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)} - Q(S_t, A_t) \right]$$

Target Network **Prediction Network**

3.2.3 Hyperparameter Tuning:

Several model training experiments are undertaken using Mlflow. All the model artifacts are saved and served by a web application server for visualization and comparison purposes. The best model hyper-parameters are found as below.

1. Gamma: 0.8. This value denotes the weightage given to the future reward. If 0, it means future reward is not taken into consideration and if 1, it means we are weighing high the future to calculate the Q value.

2. n_steps: 1. This shows the number of steps by agent in environment that is taken into consideration to learn the Q function.

3. batch_size: 32. This denotes the size of the sample that is chosen in a single iteration of training. Value 1 shows that we are doing stochastic gradient descent.
4. memory_capacity: 15000. This value shows the size of replay memory buffer.

5. target_update: 50. This value shows the number of episodes after which the weights from the prediction network is copied to the target network.

6. Exploration. This part shows the variables that alters the agent's exploration and exploitation strategy.

   i. "method": "EpsilonGreedy"
   ii. "tau": 6000
   iii. "temperature": 1.0
   iv. "final_temperature": 0.05
   v. "loss_function": "l2"

3.2.4 Dueling DQN Architecture:

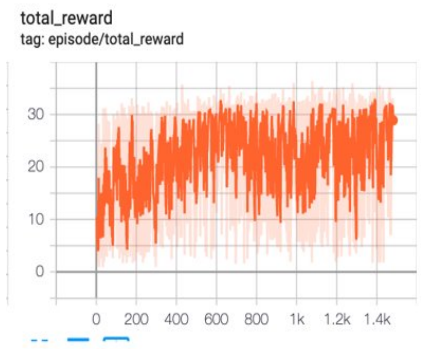
The Dueling DQN architecture is a neural network modification as we discussed before. So, all Dueling DQN specific implementations are done in model definitions. All the implementations are done by following the object-oriented design principles. The high-level architecture parameters are defined in JSON file. The basic building blocks like Multi-layer perceptron are defined in a separate class. These build block classes are used as parent class to construct highly specialize classes like Dueling architecture. It consists of two layers - Value in and Advantage in. Following that is two layers – Value out and Advantage out. Finally, an aggregation layer is used to aggregate the output of Value layer and Advantage layer.

3.2.5 Results

Techniques	**Episodes**	**Training Time**	**Testing Time**	**Reward for 3 episodes in testing**	**Result**
DQN	**2000**	**9 hr**	**28s**	**[35.8, 11.1, 37.4]**	**High variance Takes longer time to train**
Dueling DQN	**2000**	**7 hr**	**26s**	**[30.7, 30.7, 32.7]**	**Variance is handled Agent learns to avoid collision.**

Table 2. Results and Comparisons (DQN and DDQN with Dueling)

3.3 Conclusions



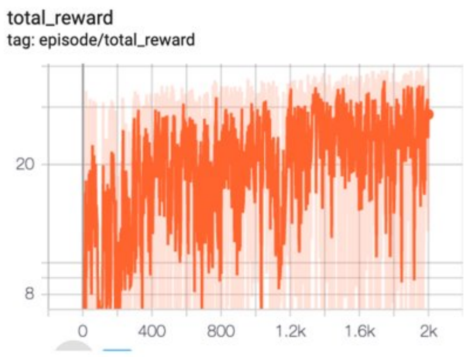


Figure 5. Comparison of episode-reward in DQN and Dueling DQN

After training for 2000 episodes, the simulation results shows that DQN with Dueling network architecture performs better than DQN. From figure 3. in both the models we can observe high variance in the rewards. Comparatively, we can see the high variance error is lesser in Dueling architecture. We also observed the crashing of ego vehicle in DQN and no such crash is recorded in Dueling model after training completion. In future, we can try Noisy DQN model and DQN with Prioritized Experience Replay to see if we can improve the current model performance.

Chapter 4. Appendix

4.1 Code

GitHub Link: https://github.com/RL-AutonomousDriving/RL_algorithm

4.2 References

[1] Liao, Jiangdong., Liu, Teng., Tang, Xiaolin., Mu, Xingyu., Huang, Bing., & Cao, Dongpu, "Decision-making strategy on Highway for Autonomous Vehicles using Deep Reinforcement Learning"

[2] Li, Xin., Xu, Xin., Zuo, Lei, "Reinforcement Learning based overtaking decision-making for highway autonomous driving" in 2015 Sixth International Conference on Intelligent Control and Information Processing (ICICIP)

[3] L. Edouard, "An environment for autonomous driving decisionmaking," https://github.com/ eleurent/highway-env, GitHub, 2018.

[4] M. Zhou, X. Qu, and S. Jin, "On the impact of cooperative autonomous vehicles in improving freeway merging: a modified intelligent driver model-based approach," IEEE Trans. Intell. Transport. Syst., vol. 18, no. 6, pp. 1422-1428, June 2017.

[5] C. J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving," IEEE Transactions on Intelligent Vehicles, vol. 5, no. 2, pp. 294-305, 2019.

[6] C. J. Hoel, K. Wolff, and L. Laine, "Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation," arXiv preprint arXiv:2004.10439, 2020.

[7] P. Hart, and A. Knoll, "Using counterfactual reasoning and reinforcement learning for decision-making in autonomous driving," arXiv preprint arXiv:2003.11919, 2020.

[8] A. Furda, and L. Vlacic, "Enabling safe autonomous driving in real-world city traffic using multiple criteria decisions making," IEEE Intelli. Trans. Sys. Magaz., vol. 3, no. 1, pp. 4-17, 2011.

[9] S. Nageshrao, H. E. Tseng, and D. Filev, "Autonomous highway driving using deep reinforcement learning," In 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 2326-2331, 2019.

[10] S. Han, and F. Miao, "Behavior Planning for Connected Autonomous Vehicles Using Feedback Deep Reinforcement Learning," arXiv preprint arXiv:2003.04371, 2020.