華中科技大學

Enhancing Autonomous Cars Intelligence Using Deep Reinforcement Learning

Student name	**:**	**Abdulkadir Duran Adan 阿丹**
Student Number	**:**	**I202321026**
Major	**:**	**Artificial Intelligence**
School	**:**	**School of Artificial and Automation**
Instructor	**:**	**Zhu Lijun**

Enhancing Autonomous Cars Intelligence Using Deep Reinforcement Learning

Abstract

Autonomous driving is currently one of the hottest and most interesting topics in research and business, and Some of the world's biggest companies, including Tesla, Waymo, and Apple, are working on autonomous vehicles. However, one of the main challenges in autonomous driving today is decision-making and this is where Deep Reinforcement Learning (DRL) comes in to make the right decisions through experimentation and exploration.

This study introduces a method for safely navigating a Self-driving vehicle to avoid obstacles and reach its destination in highway scenarios using a combination of deep Q Networks (DQN) and neural networks (NN). DQN is trained in simulation to serve as a central decision-making unit using Q value functions for actions estimating future rewards and then finding the optimal solution that proposes targets for a trajectory planner. As the self-driving car task involves making decisions based on sensor inputs and navigating through a complex environment, NN allows the Agent (virtual car) to learn a mapping from raw sensor inputs to action outputs facilitating effective decision-making.

After continuous training through exploration and exploitation, the self-driving car was able to reach its destination successfully without hitting any obstacles, demonstrating that the suggested method is capable of safe and efficient driving.

Please make sure to watch the demonstration videos titled "Video_1.mp4" and " Video_1.mp4" located in the "Result_videos" folder. The source code for the model is available in the DRL_SelfDrivingCar Folder.

Keywords: DRL; DQN, NN; Autonomous driving ;

Table of Contents

ENHANCING AUTONOMOUS CARS INTELLIGENCE USING DEEP REINFORCEMENT LEARNING.....

ABSTRACT I

TABLE OF CONTENTS..... II

1. INTRODUCTION 1

1.1 BACKGROUND 2

1.2 OBJECTIVE.....

2. LITERATURE REVIEW 3

2.1 REINFORCEMENT LEARNING 3

2.2 DEEP LEARNING 3

2.3 DEEP Q-LEARNING NETWORKS..... 4

3. METHODOLOGY 5

4. EXPERIMENTAL SETUP..... 7

4.1 ENVIRONMENT..... 7

4.2 REWARDS, ACTIONS, AND STATES 8

4.3 DEEP Q-NETWORK AND EXPERIENCE REPLAY..... 9

5. RESULT AND MODEL EVALUATION 14

6. CONCLUSION 17

REFERENCES 18

Enhancing Autonomous Cars Intelligence Using Deep Reinforcement Learning

1. Introduction

Today, Autonomous driving is a highly researched field both in industry and academia. Autonomous vehicles have been gaining significant attention in recent years due to their potential to revolutionize the transportation industry. This revolution promises enhanced road safety, optimized traffic flow, and improved fuel economy, as recent studies highlighted [1], [2], [3], [4]. However, enabling a car to safely drive and navigate through complex environments is a challenging task. Every second on the road, an autonomous vehicle must make several decisions based on the current road situation. For example, it must decide which lane to take or how to approach other cars. The surrounding environment, which is the basis for these decisions, can be composed of various elements like static obstacles, other road participants, traffic signs, and pedestrians. Processing such a vast amount of information and using it to make appropriate decisions is currently one of the biggest challenges in the field of autonomous driving [5]. Designing rules for all possible scenarios is almost impossible. Due to this, current applications are limited to rural areas with medium complexity, or the function itself, like preceding a car, is rather simple.

This study introduces a combination of DQN and NN and how they can be used to navigate safely in different complex road scenarios, avoiding any obstacles. In the context of self-driving cars, Deep Reinforcement Learning (DRL) has emerged as a powerful method for training autonomous vehicles to learn and adapt to complex driving scenarios.

Reinforcement learning (RL) is a learning-based approach that uses decision-making agents. This approach has the potential to scale to all driving situations. In recent years, many researchers have leveraged RL to tackle critical decision-making problems in autonomous driving. For example, the Deep Q-Network (DQN) algorithm has been used to handle intersections and highway scenarios [6], [7],[8] and a policy gradient approach has been implemented for lane merging and all found that DQN agents can successfully navigate urban environments. [9] extended this work to highway driving, using a dueling DQN algorithm to develop a decision-making strategy for overtaking. [10] also reported high accuracy in lane following and obstacle avoidance using a DQN-based autonomous driving system. These studies collectively highlight the potential of DQN in enhancing the

safety and efficiency of autonomous driving. Some studies have conducted both training and evaluation in simulated environments, while others have trained the agent in simulations and then tested it in the real world [11],[12].

1.1 Background

Autonomous vehicles are responsible for safely driving amidst dynamic traffic conditions. They must follow road markings, maintain a safe distance from other vehicles, react appropriately to emergencies, and navigate to the desired destination. In 2017, Udacity partnered with Didi Chuxing to organize the first self-driving car challenge aimed at developing effective methods for detecting obstacles using camera and LIDAR data. This challenge could help self-driving cars prioritize the safety of road users and pedestrians. Additionally, Udacity organized another challenge to predict the steering angle using deep learning based on the image inputs from a camera mounted on the windshield.

However, most research and designs focus on the basic features of self-driving cars, neglecting the quality of their driving skills and efficiency in getting passengers to their destinations. The driving skill of a self-driving car agent becomes a priority to deliver the best transportation service compared to a manned vehicle. For instance, a self-driving car agent should be capable of switching to the optimal lane that maximizes its speed rather than following slow-moving vehicles in a predefined lane all the time. This is crucial because passengers expect to reach their destination by taking the shortest journey and shortest time with safety.

1.2 Objective

The objective of this is study to design and implement a self-driving car agent that successfully drives itself to-and-from its destination while not hitting (for the most part) any obstacle to maximize its reward and find the optimal solution. NN is used to build a self-driving car-agent to interact with complex conditions. DQN is used to train the model to adapt to the dynamic environment and extract map features. The agent takes action to learn the optimal policy to make a decision and choose the optimal available option that maximizes its reward.

2. Literature Overview

2.1 Reinforcement learning

Reinforcement learning involves an agent generating the optimal policy in an environment to maximize rewards. It is distinct from supervised and unsupervised learning. Recent research has made significant progress in the application of reinforcement learning to autonomous driving [13],[10]. Both explore the use of deep reinforcement learning for control and navigation tasks, with the latter specifically focusing on urban environments. [14] provides a comprehensive review of reinforcement learning techniques and algorithms used by major autonomous vehicle companies, while also proposing alternative solutions. [15] contributes to the field by modeling the interaction between autonomous vehicles and their environment as a stochastic Markov decision process, and using reinforcement learning to achieve desired driving behaviors. The result shows that the desired driving behavior of an autonomous vehicle is obtained using reinforcement learning. These studies collectively highlight the potential of reinforcement learning in enhancing the capabilities of autonomous driving systems.

2.2 Deep Learning

A range of studies have explored the use of deep learning in autonomous driving. [16] developed a real-time control system for self-driving cars using a CNN, with an impressive 85.03% accuracy in predicting steering angles. The CNN model learns from images and steering angles collected during driving in the training phase. The results demonstrate that the CNN is capable of learning diversified tasks related to lanes and roads, including following with and without lane marking, direction planning, and automatic control. [17] used a CNN to process lidar data for obstacle evasion and parking maneuvers, achieving performance comparable to a finite-state machine. In the training, CNN was used to process the lidar data of an autonomous vehicle to obtain the steering angle for obstacle evasion and parking maneuvers. The lidar data and other measurements are introduced into the CNN by mapping the 400 polar vectors (ρi, ϕi) in a 20×20 normalized matrix. The main findings of the paper are the successful use of a CNN to

process lidar data for an autonomous vehicle's steering angle predictions and its comparable performance to a finite state machine in a training simulator.

These studies have shown how Convolutional Neural Networks (CNNs) can be used in autonomous driving to improve its applications

2.3 Deep Q-Learning Networks

Recent studies have demonstrated the effectiveness of Deep Q-learning networks (DQNs) in autonomous driving. [18] shows Deep Q-Networks can safely navigate an autonomous vehicle in highway scenarios by combining deep Q-Networks and insight from control theory. The main findings are the introduction of a combination of machine learning and conventional control theory for safe navigation in highway scenarios, the demonstration of efficient and safe driving behavior, and the effectiveness of the Deep Q-Network for decision-making in solving highway scenarios efficiently and without collisions. [19] developed a self-driving car simulation using DQN, highlighting its potential for real-world applications. These studies collectively suggest that DQN is a viable approach for autonomous driving, particularly in complex and dynamic environments. The main findings of the paper are the proposal of a car game using Deep Q-Learning for simulating a self-driving autonomous car, the explanation of the difference between Q-Learning and Deep Q-Learning, and the description of the reward-based nature of the game. the result shows the Performance of the self-driving car in reaching the destination without hitting any obstacles.

These studies collectively suggest that DQN is a viable approach for autonomous driving, particularly in complex and dynamic environments.

3. Methodology

In this paper, we implement the deep Q-learning algorithm to train a self-driving car using a neural network. The primary objective is to teach the car how to efficiently navigate its surroundings and reach its destination without colliding with any obstacles. To achieve this goal, we leverage Deep Q Networks (DQN) and Neural Networks methodologies.

Deep Q-learning is the combination of Q-learning and Deep Learning. This results in a Deep Q-Network (DQN). In Deep Q-Learning, we provide actions and states as input values and receive Q-Values as outputs, which associates states with Q-Values. Deep Q-Learning can store all past experiences, enabling it to know the Q-Values of all possible actions. The process is as follows:

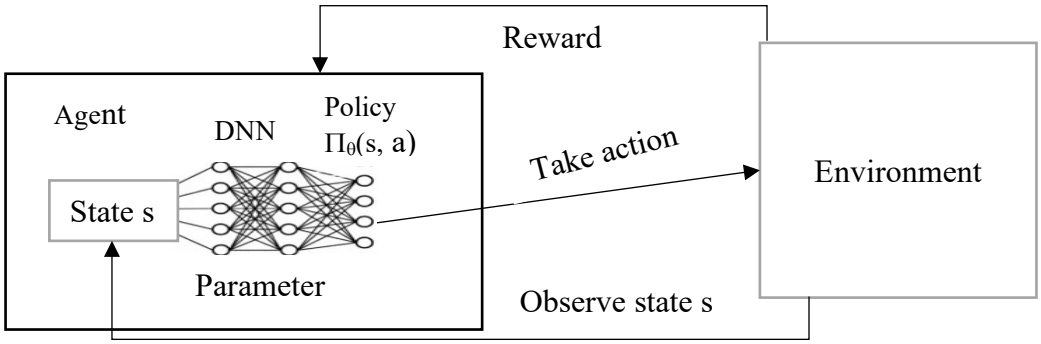

Figure 3-1: Model Architecture

Let's examine the definitions of these important terms in our model:

Table 3-1 RL terms used and their meaning in the model

Term	Description
Agent	self-driving car
Action	The actions the car takes like moving forward, left, or right in degrees
Environment	The virtual environment where the car navigates to perform its

	actions
Reward	The agent's action is assessed through a reward system, which can be either positive or negative, indicating a good or bad outcome.
State	The current situation of the car

Our model utilizes the Deep Q Learning algorithm which makes use of a variant of Deep Q Network (DQN). To predict the Q-values of a specific state, a neural network is used and the loss is computed using the temporal difference (TD) error. Temporal Difference is the method by which our agent learns from the environment to find optimal Q-Values and make a decision. The method involves subtracting the expected Q-value of any state and action from the Q-value calculated afterward. The temporal difference (TD) error is calculated using the equation below:

$$TD_t = r_t + \gamma \cdot max_{a'} Q(s_{t+1,} a'; \theta^-) - Q(s_{t+1,} a_{t,} \theta) \tag{3-1}$$

Where r_tb is the reward at time *t*, γ *is* the discount factor, a_t is the chosen action, s_t is the current state, $Q(s_{t+1,} a'; \theta^-)$ is the Q-value predicted by the neural network assigned to the current state-action pair, $max_{a'} Q(s_{t+1,} a'; \theta^-)$ is the Q-value prediction meant for the next state $s_{t+1,}$ considering the action a' that maximizes the Q-value, using a target neural network with parameters θ^-, θ represents the current neural network parameters.

The TD error is used to calculate the loss and update the parameters of the neural network by backpropagation to minimize the loss. The loss is calculated by taking the average of the squared temporal difference (TD) errors using the equation below.

$$Loss = \frac{1}{2}(TD_t)^2 \tag{3-2}$$

4. Experimental Setup

4.1 Environment

To create a virtual environment for the car to navigate and reach its destination, we use the open-source Python framework, **Kivy**. Kivy was ideal as it allows one to develop mobile apps and other applications and that's where we set up the environment (the virtual map) for the car.

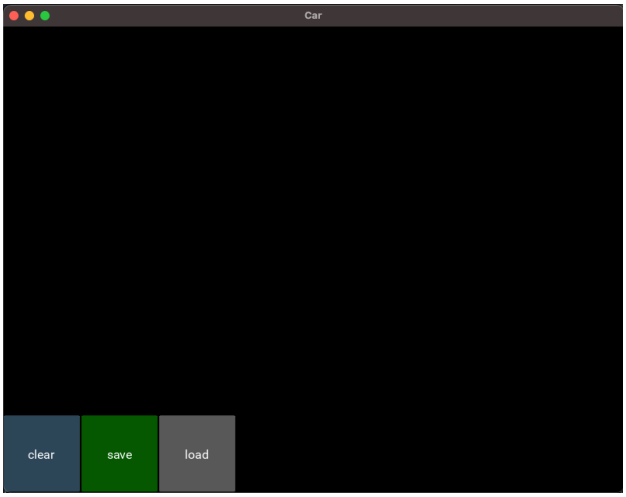

Figure 4-1: Kivy Screen as a virtual environment for the car

The car must travel from the top-left to the bottom-right of the map and follow the path or the road without hitting any obstacles until the program is stopped as shown in the figure below.

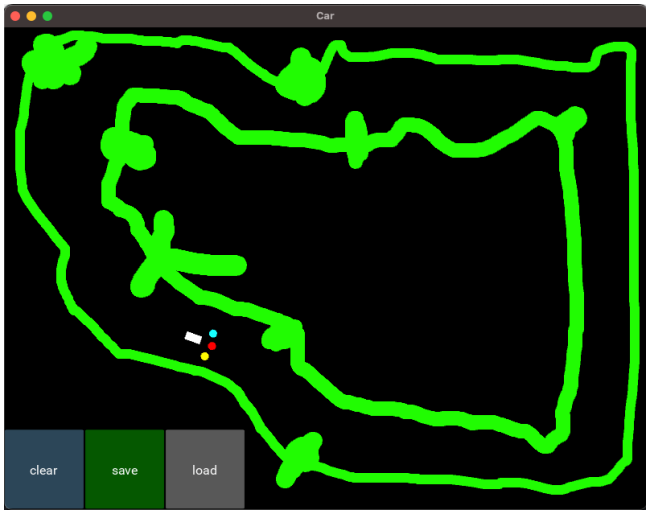

Figure 4-2 Self-driving car navigating through a complex road

4.2 Rewards, Actions, and States

Rewards: if the car (agent) hits any obstacle, a -5 reward (Largest negative reward) is given as a penalty, a +2.5 reward for reaching the destination, +0.1 reward for going towards the destination, a -0.05 reward for moving away from the destination, and finally, a reward of -0.1 is given to the agent (car) if it goes too close to the edges of the map.

Actions and states: As the agent (car) can take multiple actions and end up in multiple states in an environment, we should set specific directions based on the current state. We have four directions to choose from: left, right, up, or down. However, a car cannot simply rotate 180 degrees and move in that direction. Instead, it must take smaller, incremental actions to make a turn as shown in the table below.

Table 4-1 Agent's directions

Angle (degree)	Direction
0°	Going and continue the same direction staying on the previous trajectory.
20°	Incrementally turning to the right.
-20°	Incrementally turning to the left

To ensure that our virtual car can navigate safely and reach its destination, it needs to be able to recognize its current state and determine the appropriate course of action. To achieve this, we place three sensors on the car to detect obstacles in its path (see Figure 4-2). The car's sensors receive environmental signals, including proximity information and orientation. Our agent then uses this information to determine the direction or action that the car should take next to avoid any potential collisions or hazards.

4.3 Deep Q-Network and Experience Replay

To implement the model for the virtual self-driving car, we use PyToch to train the model. PyTorch is a Python library primarily used in deep learning for machine learning purposes.

Deep Q-Network: the brain that enables the virtual self-driving car using the neural network. Initially, data is input into the input layer. The hidden layer consists of 30 nodes, and the output layer has three nodes, which activate one of three actions the agent can take.

In our model, we utilize a Q-network represented by the “network” class as shown in the code snippet below. The architecture consists of two fully connected layers: the first layer (fc1) has a ReLU activation function, and the second layer (fc2) outputs Q-values that correspond to different actions.

```
class Network(nn.Module):
    def __init__(self, input_size, nb_action):
        super(Network, self).__init__()
        self.input_size = input_size
        self.nb_action = nb_action
        self.fc1 = nn.Linear(input_size, 30)
        self.fc2 = nn.Linear(30, nb_action)
    def forward(self, state):
        x = F.relu(self.fc1(state))
        q_values = self.fc2(x)
        return q_values
```

The code snippet above defines the class of our neural network. The first function, called "__init__", initializes the class and sets up the structure of the network. In this case, the function uses a parameter of 30 to represent the number of nodes in the hidden layer. The second function, called "forward", activates the output of the neural network. The first

line of this function is called the rectifier function, which is then used to determine the Q values of the next states. These Q values are then returned from the function.

Experience Replay: Experience Replay is a technique used to improve the stability and efficiency of the learning process. It enables the agent to learn from past experiences by storing and randomly selecting samples from its memory. By doing so, the agent can learn from its mistakes and achieve better performance in the future. We create the class (ReplayMemory) to create the Experience Replay for our agent and it is responsible for storing and sampling experiences, with a capacity of 100,000. The code is shown below.

```
class ReplayMemory(object):
    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
    def push(self, event):
        self.memory.append(event)
        if len(self.memory) > self.capacity:
            del self.memory[0]
    def sample(self, batch_size):
        samples = zip(*random.sample(self.memory, batch_size))
        return map(lambda x: Variable(torch.cat(x, 0)), samples)
```

The (__init__) function initializes the capacity and the memory for storing. The “push” function is responsible for Storing events in memory and the “sample” function Samples batches from memory. During the learning process, our agent saves a 'last_brain.pth' (PyTorch file) file to remember what it has learned so far.

To integrate Q-learning with deep reinforcement learning, we implement a Q-learning algorithm class (Dqn) that facilitates the training of the self-driving car agent. The class orchestrates the training process with functions for action selection, learning from experiences, and updating the Q-network. the code is shown below.

```
class Dqn():
```

```
    def __init__(self, input_size, nb_action, gamma):
        self.gamma = gamma
        self.reward_window = []
        self.model = Network(input_size, nb_action)
        self.memory = ReplayMemory(100000)
        self.optimizer = optim.Adam(self.model.parameters(),
lr = 0.001)
        self.last_state =
torch.Tensor(input_size).unsqueeze(0)
        self.last_action = 0
        self.last_reward = 0
      def select_action(self, state):
        probs = F.softmax(self.model(Variable(state,
volatile = True))*100)
        action = probs.multinomial(2)
        return action.data[0,0]
    def learn(self, batch_state, batch_next_state,
batch_reward, batch_action):
        outputs = self.model(batch_state).gather(1,
batch_action.unsqueeze(1)).squeeze(1)
        next_outputs =
self.model(batch_next_state).detach().max(1)[0]
        target = self.gamma*next_outputs + batch_reward
        td_loss = F.smooth_l1_loss(outputs, target)
        self.optimizer.zero_grad()
        td_loss.backward()
```

```
        self.optimizer.step()
    def update(self, reward, new_signal):
        new_state =
torch.Tensor(new_signal).float().unsqueeze(0)
        self.memory.push((self.last_state, new_state,
torch.LongTensor([int(self.last_action)]),
torch.Tensor([self.last_reward])))
        action = self.select_action(new_state)
        if len(self.memory.memory) > 100:
            batch_state, batch_next_state, batch_action,
batch_reward = self.memory.sample(100)
            self.learn(batch_state, batch_next_state,
batch_reward, batch_action)
        self.last_action = action
        self.last_state = new_state
        self.last_reward = reward
        self.reward_window.append(reward)
        if len(self.reward_window) > 1000:
            del self.reward_window[0]
        return action
    def score(self):
        return
sum(self.reward_window)/(len(self.reward_window)+1.)
    def save(self):
        torch.save({'state_dict': self.model.state_dict(),
```

```
                    'optimizer' :
self.optimizer.state_dict(),
                }, 'last_brain.pth')
    def load(self):
        if os.path.isfile('last_brain.pth'):
            print("=> loading checkpoint... ")
            checkpoint = torch.load('last_brain.pth')
        self.model.load_state_dict(checkpoint['state_dict'])
     self.optimizer.load_state_dict(checkpoint['optimizer'])
            print("done !")
        else:
            print("no checkpoint found...")
```

The function named (__init__) initializes the size of the input state vector, number of possible actions, discount factor for future rewards, window to store recent rewards for tracking the agent's performance, an instance of the (Network class) representing the Q-network, an instance of the (ReplayMemory) class for experience replay, Adam optimizer for updating the neural network parameters. The function (select_action) uses the Q-network to select an action by applying softmax and multinomial sampling to a given state. The function 'learn' performs a Q-learning update using a batch of experiences, including state, next_state, action, and reward. The (update) function updates the agent's behavior based on the observed reward and new signal (state). It also manages the learning from batches of experiences. The (score) function calculates the average score over the reward window, providing a measure of the agent's performance. The save and load functions save and load the Q-network's state and optimizer's state to/from a file ('last_brain.pth').

5. Result and Model Evaluation

Finally, Let's see the virtual self-driving car in action, learning to avoid obstacles and reach its destination.

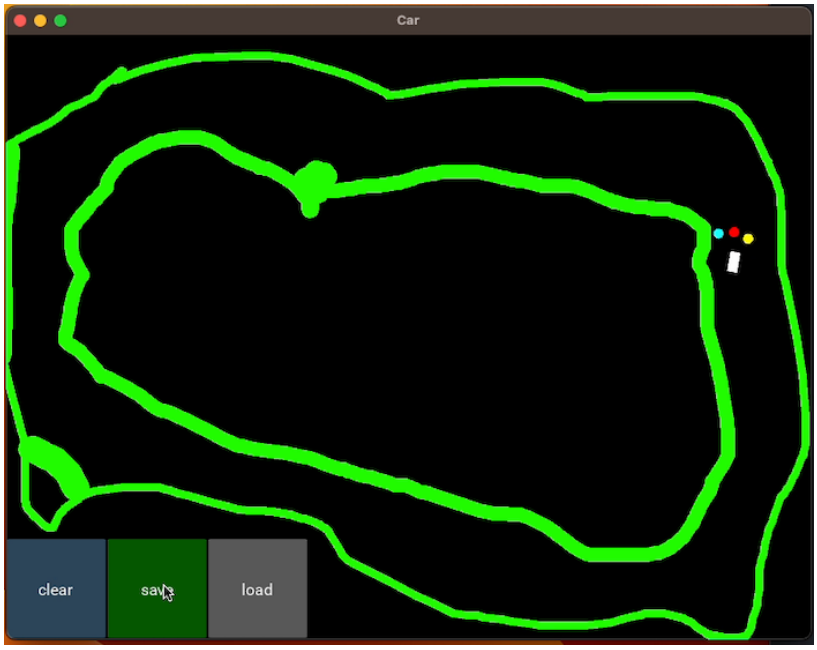

Figure 5-1: Demonstration of the virtual self-driving car.

Now, our virtual self-driving car is capable of navigating through complex circular roads avoiding hitting obstacles most of the time, and reaching its destination as we can from the figure above.

Note:

- The "save" saves the state of the trained model and uses matplotlib graph to generate a plot of the training scores when pressed. The button enables saving the trained model for future use and visualizes the training progress.
- The "load" button checks if a saved model checkpoint file ('last_brain.pth') exists and loads the model and optimizer state from that file when pressed. The button allows the user to resume training or continue the simulation using a previously saved model checkpoint.
- The "clear" button clears the drawing road on the canvas and sets the environment to be a matrix of zeros when pressed. The button allows the user to modify the environment by manually drawing on the map.

As we can see in the graph below, the learning process of the car takes place after every single action taken, after many iterations over time, the car would become an expert at any given route.

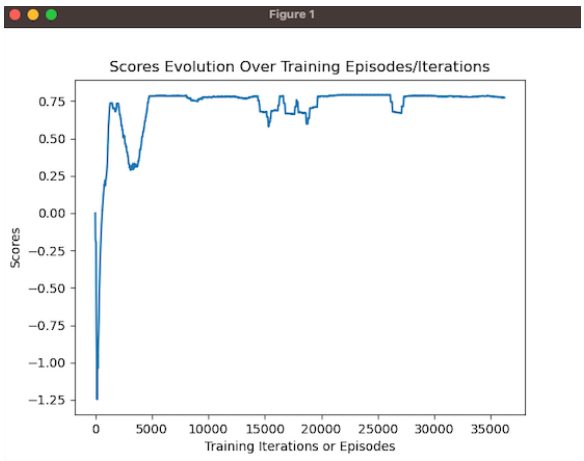

Figure 5-2: Graph showing scores over iterations during training

Based on the Figure above, the score of the agent for each episode increased gradually over the episode for all episodes. The models were able to improve the maximum score over training episodes when the car was running on a complex circular road. As we can see in the graph, the learning process of the car takes place after every single action taken, after many iterations over time, the car would become an expert at any given route

Now, let's see the car running on a straight road as shown in the figure below.

Figure 5-3: self-driving car navigating through a rough straight road.

The car proceeds straight to the destination navigating through the straight path and returns to its starting point after reaching the destination without hitting the obstacle.

From Figure 5-2, it is clear that the agent (car) takes less time to interact with the environment and learns faster to navigate through a straight path to reach the destination than navigating through a complex and circular road (compare Figure 5-2 and Figure 5-4).

Watch “Video_1.mp4” and “Video_2.mp4” in the “Result_videos” folder for demonstration.

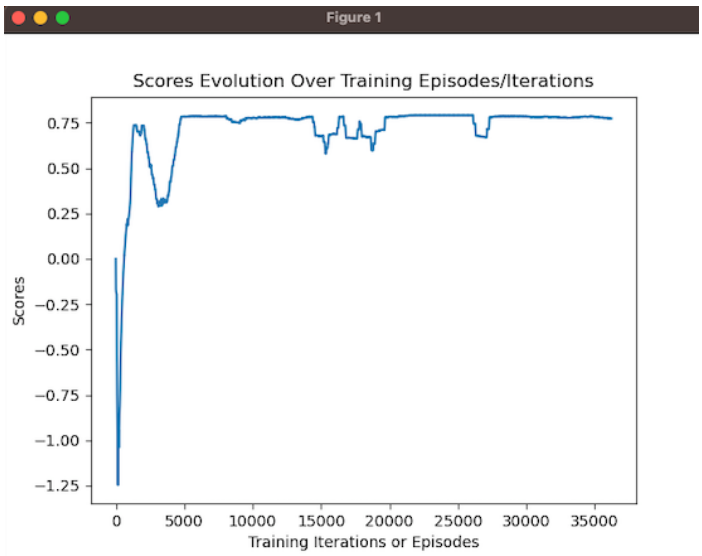

Figure 5-4: car driving itself on a straight road

The agent (car) a steadily increasing trend in scores over iterations indicates that the agent is learning faster to navigate and follow the straight path effectively, while fluctuations or a plateau may indicate challenges or convergence issues.

As we noted, temporal difference is the key to learning used in the learning of our car as it updates the Q-values, enabling the model to better estimate the expected future rewards and helps the model converge towards more accurate estimates over time, improving the quality of the learned policy.

6. Conclusion

In this study, we aimed to train a self-driving car using deep reinforcement learning techniques, specifically employing Deep Q Networks (DQN) and Neural Networks. Our main goal was to create an autonomous agent with the ability to navigate through its environment, avoid obstacles, and successfully reach its destination.

After a continuous training process involving interaction between the self-driving car and its environment through exploration and exploitation, the car learned to navigate the environment, avoid obstacles, and reach its destination without collisions. The use of Matplotlib enabled us to track the agent's progress by visualizing the scores across training episodes.

Our study has successfully demonstrated the effectiveness of applying deep reinforcement learning techniques to the complex problem of self-driving car navigation. The model developed during this study showed the ability to learn a policy that enables the car to make informed decisions in dynamic environments. Through the use of experience replay and temporal difference learning, the model was trained efficiently and stably, proving the effectiveness of these mechanisms. In conclusion, this study highlights the potential of using deep reinforcement learning techniques to develop safe and efficient self-driving car systems.

References

Research articles

[1] S. Feng *et al.*, "Dense reinforcement learning for safety validation of autonomous vehicles," *Nature*, 2023, 615(7953): 620–627.

[2] J. Dong, S. Chen, M. Miralinaghi, T. Chen, and S. Labi, "Development and testing of an image transformer for explainable autonomous driving systems," *J. Intell. Connect. Veh.*, 2022, 5(3): 235–249.

[3] H. Shi, D. Chen, N. Zheng, X. Wang, Y. Zhou, and B. Ran, "A deep reinforcement learning based distributed control strategy for connected automated vehicles in mixed traffic platoon," *Transp. Res. Part C Emerg. Technol.*, 2023, 148(98): 104019.

[4] Y. Han, M. Wang, and L. Leclercq, "Leveraging reinforcement learning for dynamic traffic control: A survey and challenges for field implementation," *Commun. Transp. Res.*, 2023, 3(342): 100104.

[5] I. Barabás, A. Todoruţ, N. Cordoş, and A. Molea, "Current challenges in autonomous driving," *IOP Conf. Ser. Mater. Sci. Eng.*, 2017, 252(1): 012096.

[6] Y. Cheng, X. Hu, K. Chen, X. Yu, and Y. Luo, "Online longitudinal trajectory planning for connected and autonomous vehicles in mixed traffic flow with deep reinforcement learning approach," *J. Intell. Transp. Syst.*, 2023, 27(3): 396–410.

[7] Ó. Pérez-Gil *et al.*, "DQN-Based Deep Reinforcement Learning for Autonomous Driving," in *Advances in Physical Agents II*, vol. 1285, L. M. Bergasa, M. Ocaña, R. Barea, E. López-Guillén, and P. Revenga, Eds., in Advances in Intelligent Systems and Computing, vol. 1285. , Cham: Springer International Publishing, 2021, pp. 60–76.

[8] T. Okuyama, T. Gonsalves, and J. Upadhay, "Autonomous Driving System based on Deep Q Learnig," in *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, Mar. 2018, pp. 201–205.

[9] J. Liao, T. Liu, X. Tang, X. Mu, B. Huang, and D. Cao, "Decision-Making Strategy on Highway for Autonomous Vehicles Using Deep Reinforcement Learning," *IEEE Access*, vol. 8, pp. 177804–177814, 2020

[10] A. R. Fayjie, S. Hossain, D. Oualid, and D.-J. Lee, "Driverless Car: Autonomous Driving Using Deep Reinforcement Learning in Urban Environment," in *2018 15th International Conference on Ubiquitous Robots (UR)*, Jun. 2018, pp. 896–901.

[11] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to Real Reinforcement Learning for Autonomous Driving." arXiv, Sep. 26, 2017.

[12] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst." arXiv, Dec. 07, 2018.

[13] Y. O. Anisimov and D. A. Katcai, “Deep reinforcement learning approach for autonomous driving,” in *Journal of Physics: Conference Series*, IOP Publishing, 2021, p. 012013.

[14] E. Jebessa, K. Olana, K. Getachew, S. Isteefanos, and T. K. Mohd, “Analysis of Reinforcement Learning in Autonomous Vehicles,” in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, 2022, pp. 0087–0091.

[15] C. You, J. Lu, D. Filev, and P. Tsiotras, “Highway traffic modeling and decision making for autonomous vehicle using reinforcement learning,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 1227–1232.

[16] W. Dangskul, K. Phattaravatin, K. Rattanaporn, and Y. Kidjaidure, “Real-Time Control Using Convolution Neural Network for Self-Driving Cars,” in *2021 7th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, IEEE, 2021, pp. 125–128.

[17] O. González-Miranda and J. M. Ibarra-Zannatha, “Use of convolutional neural networks for autonomous driving maneuver,” in *2021 XXIII Robotics Mexican Congress (ComRob)*, IEEE, 2021, pp. 63–67.

[18] M. P. Ronecker and Y. Zhu, “Deep Q-network based decision making for autonomous driving,” in *2019 3rd international conference on robotics and automation sciences (ICRAS)*, IEEE, 2019, pp. 154–160.

[19] H. Soni, R. Vyas, and K. K. Hiran, “Self-Autonomous Car Simulation Using Deep Q-Learning Algorithm,” in *2022 International Conference on Trends in Quantum Computing and Emerging Business Technologies (TQCEBT)*, IEEE, 2022, pp. 1–4.