ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

AUTOMATED LANE CHANGE DECISION MAKING FOR AUTONOMOUS VEHICLES USING MACHINE LEARNING TECHNIQUES

M.Sc. THESIS

Mehdi NASIRI

Department of Control and Automation Engineering

Control and Automation Engineering Programme

JULY 2020

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

AUTOMATED LANE CHANGE DECISION MAKING FOR AUTONOMOUS VEHICLES USING MACHINE LEARNING TECHNIQUES

M.Sc. THESIS

Mehdi NASIRI
(504161143)

Department of Control and Automation Engineering

Control and Automation Engineering Programme

Thesis Advisor: Assoc. Prof. Dr. Gülay ÖKE GÜNEL

JULY 2020

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ TEKNİKLERİ İLE OTONOM ARAÇLARDA ŞERİT DEĞİŞİMİNE KARAR VERME

YÜKSEK LİSANS TEZİ

Mehdi NASIRI
(504161143)

Kontrol ve Otomasyon Mühendisliği Anabilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

Tez Danışmanı: Assoc. Prof. Dr. Gülay ÖKE GÜNEL

TEMMUZ 2020

Mehdi NASIRI, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504161143 successfully defended the thesis entitled "AUTOMATED LANE CHANGE DECISION MAKING FOR AUTONOMOUS VEHICLES USING MACHINE LEARNING TECHNIQUES", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Gülay ÖKE GÜNEL** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Volkan SEZER** Istanbul Technical University	.....
	Asst. Prof. Dr. Figen ÖZEN Haliç University	.....

Date of Submission : 15 June 2020
Date of Defense : 23 July 2020

To my family,

FOREWORD

This thesis is written as a completion of the M.Sc. program at Istanbul Technical University. The subject of this thesis is to study an algorithm that allows a vehicle to change its lanes autonomously in a safe but customized manner without explicit initiation by the driver. This work has been educating me, and I have learned a lot in the course of this part. Many new topics emerged during this work, such as Imitation Learning, deep neural network, reinforcement learning, and decision-making, which has been full of joy and learning. Autonomous lane change with the decision-making topic has always been so interesting to me, and in this work, the optimization part was pleasurable and full of excitement. Besides all, working with the simulated virtual environment of real environment to get the result that we can use in real-world is a feeling like you are doing something real, and the proposed work would apply to the actual industrial projects someday soon.

To begin with, I would like to express my sincere gratitude to my advisor Assoc. Prof. Dr. Gülay Öke Günel, for being an excellent mentor during my work for her patience, motivation, and immense knowledge. Her insights on how to perform and convey research had a significant influence on me. She has always been so supportive, and the manner of her leadership made her students so confident in themselves and decision-makers in the course of the research, which I am not an exception to that. I also would like to thank my wife and my family and friends. I am so grateful for the lessons I took with Professor Zehra Çataltepe that provided me with great insights into machine learning related topics. Finally, I would like to thank my family for their love and support. They always encouraged me to pursue my dreams, which I'm so grateful to them.

July 2020

Mehdi NASIRI
(Control and Automation Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

AI	**:** Artificial Intelligence
ML	**:** Machine Learning
DL	**:** Deep Learning
NN	**:** Neural Network
CNN	**:** Convolutional Neural Network
RL	**:** Reinforcement Learning
LP	**:** Linear Programming
DP	**:** Dynamic Programming
ITU	**:** Istanbul Technical University

LIST OF TABLES

Page

LIST OF FIGURES

Page

AUTOMATED LANE CHANGE DECISION MAKING FOR AUTONOMOUS VEHICLES USING MACHINE LEARNING TECHNIQUES

SUMMARY

Autonomous cars play a significant role in the future of transportation. Due to the progress in Artificial Intelligence, it is anticipated that future smart cars and trucks will be driverless, accident avoiding, and efficient. They eliminate human driving errors and supply safe travel by reducing reaction time lag and safe lane change. To reach these goals, automakers have invested in the research areas regarding the current challenges to reach the expected results. Following the recent advancement in machine-learning algorithms, it is expected that driver-less cars will appear in the market within the next two decades.

Self-driving cars received considerable attention during the past ten years. The concurrency of this attention with the rise of deep learning and deep reinforcement learning algorithms is not a coincidence. Deep learning algorithms found their path into the autonomous vehicle applications first by applying convolutional neural networks (CNN) to image classifications. The obtained promising results have motivated researchers in the area of the autonomous vehicles to utilize deep neural networks in the perception layer of the advanced driver-assistance systems (ADAS). The perception layer in ADAS is responsible for detecting and classifying actors surrounding the ego vehicle, e.g., cars, pedestrians, and cyclists. After creating a bird-eye-view (BEV) mapping of the environment, the sensor fusion layer identifies and tracks the other actors in the scene. Then, a decision-making algorithm produces high-level actions - e.g., ChangeLane, Accelerate, or Stop - to minimize a pre-defined cost function, usually to avoid collisions and achieve a goal location as soon as possible. Trivially, after producing optimal high-level actions, lower-level controllers generate the desired throttle and steering angle to follow the given commands.

This thesis investigates autonomous lane-change by utilizing different methods such as Q-table, reinforcement learning, and neural network. The autonomous lane change is one of the crucial parts of ADAS's decision-making center, especially while driving on highways. Currently, the autonomous vehicles, that actively drive in cities cannot perform safe and reliable lane changes without relying on the human driver. Most of the companies utilize the Finite State Machine approaches to generate heuristic decisions based on the driving situation. However, these techniques may not generalize enough to capture different driving situations affected by the environment, road, and traffic conditions. Recently, Reinforcement Learning algorithms have shown promising results in producing discrete actions by observing discrete and continuous environments. Thus, we have decided to apply these algorithms to tactical decision making in highway driving tasks for self-driving cars. To this end, we have simulated the grid-world of the car on PyGame environment with four number of lanes and three different actions such as turning left, turning right, and stay on the lane. Finally, we

compared the results of different methods and came up with different future work scenarios to see the feasible impact.

MAKİNE ÖĞRENMESİ TEKNİKLERİ İLE OTONOM ARAÇLARDA ŞERİT DEĞİŞİMİNE KARAR VERME

ÖZET

Otonom otomobiller ulaşımın geleceğinde önemli bir rol oynamaktadır. Yapay Zekadaki büyüme nedeniyle, gelecekteki akıllı otomobillerin ve kamyonların sürücüsüz, kazadan kaçınma ve verimli olacağı öngörülmektedir. Tepki süresi gecikmesini ve güvenli şerit değişimini azaltarak insan sürüş hatalarını ortadan kaldırır ve güvenli seyahat sağlarlar. Bu hedeflere ulaşmak için, otomobil üreticileri beklenen sonuçlara ulaşmak için mevcut zorluklarla ilgili araştırma alanlarına yatırım yapmıştır. Makine öğrenimi algoritmalarındaki son ilerlemenin ardından, sürücüsüz araçların önümüzdeki yirmi yıl içinde piyasaya çıkması bekleniyor.

Kendi kendini süren otomobiller son on yıl boyunca büyük ilgi gördü. Derin öğrenmenin ve derin pekiştirici öğrenme algoritmalarının yükselmesiyle bu dikkatin eşzamanlılığı bir tesadüf değildir. Derin öğrenme algoritmaları özerk araç uygulamalarına giden yolu önce görüntü sınıflandırmalarına evrişimli sinir ağları (CNN) uygulayarak buldu. Ortaya çıkan ümit verici sonuçlar, otonom araçlar alanındaki araştırmacıları, gelişmiş sürücü destek sistemlerinin (ADAS) algılama katmanında derin sinir ağlarını kullanmaya teşvik etti.

ADAS'taki algılama katmanı, ego aracını çevreleyen aktörleri, örneğin otomobilleri, yayaları ve bisikletlileri tespit etmekten ve sınıflandırmaktan sorumludur. Ortamın kuşbakışı (BEV) eşlemesi oluşturulduktan sonra, sensör füzyon katmanı sahnedeki diğer aktörleri tanımlar ve izler. Daha sonra, bir karar verme algoritması, genellikle çarpışmalardan kaçınmak ve mümkün olan en kısa sürede bir hedef konuma ulaşmak için önceden tanımlanmış bir maliyet işlevini en aza indirmek için üst düzey eylemler (ör. ChangeLane, Hızlandır veya Durdur) üretir. Önemsiz olarak, optimum yüksek seviye eylemler ürettikten sonra, alt seviye kontrolörleri verilen komutları takip etmek için istenen gaz ve direksiyon açısını üretir.

Bu tez, Q-tablosu, pekiştirici öğrenme ve sinir ağı gibi farklı yöntemler kullanarak otonom şerit değişimini araştırmaktadır. Otonom şerit değişikliği, özellikle otoyollarda sürerken ADAS'ın karar verme merkezinin önemli parçalarından biridir. Şu anda, şehirlerde aktif olarak sürülen otonom araçlar, insan sürücüsüne güvenmeden güvenli ve güvenilir şerit değişiklikleri gerçekleştiremiyor. Şirketlerin çoğu, sürüş durumuna bağlı olarak sezgisel kararlar oluşturmak için Sonlu Durum Makinesi yaklaşımlarını kullanır. Ancak, bu teknikler çevre, yol ve trafik koşullarından etkilenen farklı sürüş durumlarını yakalayacak kadar genelleme yapmayabilir.

Son zamanlarda, Pekiştirmeli öğrenme algoritmaları, ayrık ve sürekli ortamları gözlemleyerek ayrık eylemler üretmede umut verici sonuçlar göstermiştir. Bu nedenle, bu algoritmaları kendi kendini süren otomobiller için otoyol sürüş görevlerinde taktik karar verme sürecine uygulamaya karar verdik. Bu amaçla, otomobilin ızgara

dünyasını dört şeritli ve sola dönme, sağa dönme ve şeritte kalma gibi üç farklı eylemle PyGame ortamında simüle ettik. Son olarak, farklı yöntemlerin sonuçlarını karşılaştırdık ve uygulanabilir etkiyi görmek için gelecekteki farklı çalışma senaryoları bulduk.

Tezimde, bir RL aracısını eğitebileceğim özelleştirilmiş bir stokastik trafik ortamı geliştirmeyi hedefliyorum. Daha sonra şerit değiştirme problemini derin Q ağları ile çözeceğim. İndirgeme faktörü, keşif oranı ve gizli katmanın boyutu, nöronlar, öğrenme hızı gibi sinir ağı gibi Güçlendirme öğrenme tarafı için en uygun parametreleri elde etmek için bir hiper parametre optimizasyon tekniği uyguluyorum. Ayrıca, sezgisel yöntemlere dayalı bir temel algoritma geliştiriyorum ve RL ajanlarını onunla karşılaştırıyorum.

Diğer çalışmaları kontrol, derin öğrenme ve pekiştirmeli öğrenme yaklaşımları olarak sınıflandırabilirim. Diğer yöntemler klasik yöntemlere göre daha iyi çözümler sunsa da, eğitim aldıkları belirli alanlara uygulanabilir ve farklı alanlardan yeni girdiler sağlandığında, genelleme yeteneklerini azaltan optimal çözümler vermede başarısız olurlar. Bu çalışmaya ilgi duyan diğer bir yaklaşım, pekiştirmeli öğrenmedir. RL'ye dayalı yaklaşımlar çevre ile etkileşime girdiği için, önceki yaklaşımlardan daha iyi genelleştirebildiği gibi belirsiz ortamlara karşı da dayanıklıdır. Özellik çıkarıcı olarak kullanılan derin sinir ağı sayesinde uzun vadeli kararlar verebilirler.

İşime en uygun yaklaşım, pekiştirmeli öğrenmeye dayalı yaklaşımlardır. Son zamanlarda bu alanda pek çok çalışma yapıldı. Süpervizör, tahmin edilen gözlem vektörünü ve gerçek değerleri izler. RL'nin belirsiz ortamlarda, ölçüm seslerine dayanıklı daha genel ve muhafazakar politikalar üreterek klasik karar verme yaklaşımlarından daha iyi performans gösterebileceğini gösterirler. En son teknolojiye sahip yaklaşımlar umut verici sonuçlar gösterse de, çalışmalarımın geldiği yerde RL performansı ve ayrıntılı hiperparametre optimizasyonu için uygun bir analiz sağlamakta eksikler.

Markov karar süreci, pekiştirmeli öğrenme probleminin resmileştirilmesidir. MDP'ye dayalı bir problemi resmileştirirsek, bunu RL çözümleri ile çözmeye çalışmak gibi pekiştirmeli öğrenme problemi olarak düşünebiliriz diyebilirim. Temel olarak, bir MDP, 5 parçalı bir durum, eylem, geçiş modeli, ödül ve indirim faktörüdür. Örnek olarak, eğer temsilci harekete geçerse, hiçbir şey olmama olasılığı vardır ve aynı durumda kalırsa, ödülsüz başka bir duruma geçme olasılığı vardır. Çevremiz sezgisel olarak benzer kuralları takip eder.

RL'nin amaç fonksiyonuna bağlı olarak, rho'nun ödül fonksiyonu olduğu ve gama'nın gelecekteki adımları ne kadar önemsediğimizi tanımlayan bir indirim faktörü olduğu kümülatif ödülü maksimize etmek amaçlanmaktadır. Bu sorunun çözümü, durum-eylem çiftlerinin değerini tanımlayan özyinelemeli bellman formülüdür.

Katkılarım, bir RL ajanını eğitebileceğim, karar verme problemine uygun özelleştirilmiş bir trafik ortamı oluşturmaktır. Sinir ağının Q- Derin RL'ye olan ihtiyacı tanımlayan DQN çerçevesindeki değer tahmincisi. Ayrıca, sezgisel sonsuz durum makinesini kullanarak politika üreten bir temel model tasarladım. Nihayetinde, Deep RL'deki farklı tür durum veya gözlem temsillerini analiz ediyor ve yorumluyorum.Kurala dayalı bir temel ile DNN ve CNN tabanlı RL'nin karşılaştırma sonuçlarını görebilirsiniz.

Gelecekte, çalışmalarımı dinamik arabalar, gerçekçi grafikler ve daha iyi senaryo oluşturmayı içeren yüksek kaliteli bir simülasyon kullanacak şekilde genişletmek istiyorum. Ayrıca, düşük seviyeli denetleyicinin yerinde kalacak kadar iyi olduğuna inandığım için, üst düzey karar verme olarak RL'yi kullanmak ve eylemleri gerçekleştirmek için düşük düzey denetleyicileri kullanmak istiyorum.

1. INTRODUCTION

Autonomous cars in recent years have become an interesting subject in engineering and computer science. Most car manufacturers have launched semi self-supported vehicles, including Tesla Motors, which allegedly be completely automated by 2023. This trend has motivated scientists to search for modern techniques to solve the self-autonomy problem in self-driving cars.

This section describes the "Autonomous Vehicles" (AV) concept. It explains what is the meaning of tactical decision-making in autonomous vehicles, and motivates why it is worth exploring as a research field. It proceeds to represent a collection of related academic papers and literature surveys that have been used here. Finally, it describes the goals of this study, the drawbacks, and an outline of the thesis.

The vast majority of the world's vehicles are operating on roads and not on rails or in the air or water. If this class of vehicles could be automated, the societal advantages of automating vehicle operations could also be greater.

Nowadays, autonomous cars bring us numerous advantages to our transportation systems, economic, societal, etc. The new technologies could impact vehicle health, con-management, and travel behavior. Overall, significant social AV impacts in the form of crash savings, decrease in the time of travel, parking benefits, and fuel efficiency. Driving vehicles is one of the long-standing goals of Artificial Intelligence. Before AI, ground vehicle control and orientation were discussed using feedback screening techniques that attempt to stabilize the vehicle using sensory fusion and onboard event camera information gathered. Autonomous vehicles (AV) or driver-less cars can detect the environment using several methods such as Lidar, Radar, Odometry, GPS, Laser Focus, and computer vision. Driving an autonomous car is expected to have a high level of skill, attention, and experience from a human driver [1, 2].

1.1 Motivation

Lane change in traffic [3], adaptive cruise control [4], blind-spot warning [5], and automatic braking systems [6] are a subset of state-of-the-art innovations that exist in cars today. The lane change tactic is one of the most extensively researched autonomous driving operations after trajectory analysis for autonomous vehicles. This development is attributed to several factors, such as issues related to road safety, environmental benefits, and economic considerations. Autonomous cars play a significant role in the future of transportation. With the advent of powerful processing units, artificial intelligence methods such as deep learning are being used extensively, which facilitates the advancement of autonomous vehicles. It is foreseen that autonomous vehicles increase safety on the roads by providing intelligence into the systems. Automated lane change maneuver is a critical feature in autonomous driving. It will provide economic benefits to all actors on the road by increasing the traffic flow. Also, it decreases the fuel or energy consumption of the vehicles by decreasing the need for acceleration and deceleration. A phantom motor car was already reported in The Milwaukee Sentinel newspaper in 1926 [7]. To defraud people, the idea of that was a ghost car, and the 'self-driving' car was managed by radio signals from another vehicle behind him. Technology has come a long way since 1926, and it is not unusual today to see semi-automated vehicles on public roads, depending purely on the knowledge collected from its sensors.

Autonomous vehicle technology refers not only to passenger cars but also to commercial vehicles, such as integrated vehicles. Around a third of the overall travel costs for an articulated vehicle are related to the driver's wages [8]. The laws relating to the driver's working hours are limited to the amount of time that the truck will be in operation, which is lower compared with an autonomous vehicle. Therefore autonomous will yield economic benefits.

In the scope of an autonomous vehicle, there are several concerns, one example being how to guarantee a safe interaction with human-driven cars. Human driver's actions can be unexpected. So, it is critical to be able to predict the surrounding traffic in order to give the opportunity to the automated vehicle to drive safely. In this study,

it is aimed to develop a learning-based decision-making algorithm for a vehicle in dense traffic that will command lane change to the vehicle while avoiding the probable collisions with surrounding vehicles. The aim of the proposed algorithm is to reach the final goal in the traffic environment with safe lane-change maneuvers. Based on the complexity of the environment, the uncertainty assigned to the spawned vehicles in the traffic, and the large size of state-action pairs, a deep reinforcement learning (RL) algorithm is developed, which is an intrinsically model-free method and appropriate for the problem at hand.

There are numerous methods applicable to this problem ranging from purely rule-based to end-to-end deep learning methods. The neural network (or deep neural network) requires access to the whole training data in advance form the environment, which is not applicable in decision-making problems for the traffic environment due to the lack of such a dataset. Recalling the complexity of the environment and the uncertainty assigned to the spawned vehicles in the traffic, a reinforcement learning (RL) algorithm is chosen, which is an intrinsically model-free method and appropriate for the problem at hand. Firstly, the problem is solved by generating a Q-table with value-iteration and policy-iteration methods, and the optimal solution (actions)is retrieved from it. In this case, all possible combinations of state-action pairs should be seen and evaluated, and since the environment in this work is stochastic with many surrounding vehicles, it is not feasible to proceed with the solution based on Q-table. At this step, a function approximation method (neural-network) is developed to substitute for the large Q-table discussed previously, which generates optimal Q-value and its corresponding actions faster and, in some cases addresses the failure of the classical Q-table method that would not be able to cover the whole state-action space.

1.2 Literature Review

Autonomous driving cars [9] are designed to enhance the safety of traffic by reducing the effect of individual errors. An autonomous driving car provides a substantial rise in speed, aims to improve comfort, driving experience, and safety. In fact, they have become an interesting subject in engineering and computer science. Most car manufacturers have launched semi self-supported vehicles, including Tesla Motors, which will allegedly be completely automated by 2023. This trend has motivated

scientists to search for modern techniques to solve the self-autonomy problem in self-driving cars.

The problem of autonomous driving can be expressed as finding the optimal steering angle and throttle values using the sensory perception implemented on the vehicle to drive from point A to point B safely. A common approach is to divide the problem into hierarchical sub-problems and solve them separately.

As summarized in [10], the perception layer calculates the raw sensory measurements, and the fusion layer combines these measurements to generate an occupancy grid summarizing the positions and velocities of the surrounding vehicles. The planning layer receives this information and generates a feasible path to follow.

Finally, feedback control systems [11, 12] are used to stabilize the vehicle while following the generated path. Initially, most of the classical approaches in the literature attempted to solve each of these layers using classical feedback control systems [13, 14]. They have shown that the classical approaches are reliable and safe enough to be utilized, not only on simulations but also in the actual vehicles.

Employing these methodologies helped the car manufacturers to achieve a level of autonomy in which the car drives autonomously, however, the lack of intelligence in the system required the presence of a human supervisor throughout the driving task.

Recently, the Artificial Intelligence (AI) algorithms [15] have shown considerable achievements in multiple scientific problems, i.e.classification [16] and regression [17] problems, and the Reinforcement Learning technique [18] in the problem of autonomous Atari game play [19].

These achievements have motivated the researchers to apply AI techniques to the autonomous driving problems hoping to increase the level of intelligence in the advanced driver-assistance systems (ADAS). e.g., end-to-end object detection algorithms [20, 21] improved the performance in the perception layer.

In addition to the success of learning-based methods, this trend was accelerated considerably by huge investments from companies like Google (DeepMind) and NVIDIA. Various techniques have been applied to the issue of autonomous cars. On the experimental side, the most successful research is the NIVIDIA group [22]. They trained an end-to-end Deep Convolutional Neural Network that maps the input camera

perceptions directly to the value of the steering angle. The goal is to get structured outputs of driving-relevant objects from the pixel inputs instead of directly controlling the car.

This architecture was used to map the recorded onboard camera frames to a controlled human steering angle. The results showed an incredibly small error in controlling the steering angle. However, the generated decisions were biased mostly to the trained tracks on which the vehicle has been driven.

Besides Deep Learning methods [23], there are also competing RL approaches that attempt to address the self-driving car problem. The procedure is to leave the RL agent to interact in the simulation environment to take actions and collect useful information. This trial and error loop preserves until the agent converges to a reliable policy to drive the vehicle successfully. To the best knowledge of the author, most of these approaches are applied in simulation environments, such as TORCS [24–26] and World Rally Championship 6 (WRC6) [27]. The results seem promising, so it may be applied to the actual experimental vehicles.

This thesis focuses on the high-level decision making of autonomous vehicles on the highway driving task. The goal is to generate the optimal lane change actions, a.k.a LaneChangeRight, LaneChangeLeft, and StayOnLane, to avoid making collisions with other vehicles. Many works have approached the lane changing problem using control approaches ranging from vision-based to fuzzy and predictive control algorithms [13, 28, 29]. These methods only consider the local traffic situations affecting the decisions. This ends up achieving a slightly improved performance compared with simple heuristics using finite state machines.

In order to generate optimal high-level decisions, the algorithm should capture the complex and uncertain behaviors of the surrounding vehicles to avoid long-term traffic congestion while maintaining the safety criteria.

High-level decision-making in autonomous driving is mainly approached by either rule-based or machine learning (ML) techniques. Rule-based approaches attempt to tune a feasible heuristics to generate decisions for specific environments. Trivially, these approaches do not consider the complexity of the surrounding traffic and the uncertainty of the sensory measurements, leading to unreliable decisions that require

human supervision during the driving. Machine learning approaches, on the other hand, are capable of capturing the uncertainties originated from measurements while predicting the unforeseen situations. Different learning approaches from end-to-end imitation learning [22,30,31] to Deep Reinforcement Learning (DRL) algorithms [32] have been applied to this problem. Deep Reinforcement Learning has been proved to generate efficient policies to drive through uncertain traffic safely. Wang et al. [33] trained an RL agent for the high-level decision making for lane change policies.

The most important contribution of this work relies on designing a supervising system that overwrites the RL decisions in unsafe scenarios. The supervisor monitors the predicted observation vector and true values. If the error becomes larger than a threshold, the RL lane change actions are replaced with stay-in-the-lane action.

In another venture, Alizadeh et al. [34] apply RL to an environment with noisy measurements. They show that RL can outperform the classical decision-making approaches in uncertain environments by generating more general and conservative policies that are robust to measurement noises. Although the state-of-the-art approaches show promising results, they lack to provide a suitable analysis for the RL performance and detailed hyperparameter optimization.

Motivated from the discussed articles in the literature, this thesis, approaches to the problem of high-level decision-making in the highway driving task by applying a deep reinforcement learning algorithm to a customized simulation environment, a.k.a DeepCars. The thesis attempts to fill out the gaps in the literature to provide a detailed analysis of the RL decision-making procedure while comparing the results with rule-based approaches.

In short, the thesis provides a comprehensive Q-table analysis to justify the use of Neural Networks as the Q-value estimator in Deep Q-Network architecture. The observation vector is tailored to summarize the DeepCars environment perfectly at each time-step.

A rule-based strategy that generates a greedy policy using heuristics infinite state machine theory has been designed. Then, the RL performance is evaluated by comparing it with the rule-based agent. The results suggested the superiority of the RL algorithm in comparison with human-designed heuristics. Finally, a hyperparameter

optimization technique is provided to search for the best parameter setting and the network architecture in the problem.

It's worth noting that educational extensions to RL have been introduced to improve the control performance of various systems, including autonomous cars. These extensions include deep-Q-networks (DQN) [35], continuous action space [36], and double DQN (D-DQN) [37]. The most successful extension, though, could be the actor critical (A3C) method [38] that overcame most of the preceding self-driving car problem approaches. In the thesis, the DQN algorithm has been selected to apply to the problem.

Other works can be classified as control, deep learning, and reinforcement learning approaches. Control and vision-based approaches provide safe and reliable solutions in both simulation and real-world, but their solutions depend only on local information, and there is a lack of intelligence. They impose many assumptions, which makes their solutions less generalizable.

Another approach is end-to-end learning, which is based on deep learning methods. Although they provide better solutions compared to the classical methods, they are applicable to the specific domains where they have been trained on, and when new inputs are provided from different domains, they fail to give optimal solutions that lower their generalization capability.

Another approach that is of interest in this work is reinforcement learning. Since the approaches based on RL interact with the environment, they can generalize better than previous approaches; also, they are robust to the uncertain environment. They can give long-term decisions, thanks to the deep neural network used as the feature extractor.

There are three very relevant approaches which utilize the control approach; first reference [29] uses the classical control method and provides steering angle output required to make lane change maneuver.

The second paper [13] uses a model predictive control approach again to output the steering angle required to make the lane change. Both of them work well, considering the provided local information.

The third paper [14] uses the camera output and processes the features in the camera frames to extract the local features required to generate steering angle commands. In

all approaches, there is a need for a supervisor or monitor and control the output of the methods since they are not intelligent methods.

For a learning-based approach, there are many works in the literature, yet the paper by Nvidia is the most fundamental one where it is possible to observe similar types of intuitions. In, the corresponding steering angles to the camera frames are recorded and given to the supervised deep learning method to learn.

The most relevant approaches to this work are the approaches based on reinforcement learning [33, 34]. Recently many works have been performed in this domain, here a snapshot of two relevant and unique works are provided the approach in uses DQN for high-level decision making for lane change policies. The most important contribution of this work relies on designing a supervising system that overwrites the RL decisions in unsafe scenarios.

The supervisor monitors the predicted observation vector and true values. If the error becomes larger than a threshold, the RL lane change actions are replaced with stay-in-the-lane action. In another venture, RL is applied to an environment with noisy measurements.

These works show that RL can outperform the classical decision-making approaches in uncertain environments by generating more general and conservative policies that are robust to measurement noises. Although the state-of-the-art approaches show promising results, they lack to provide a suitable analysis for the RL performance and the detailed hyperparameter optimization, where this thesis comes through.

1.3 Objective and Contributions

In this thesis, it is aimed to implement a learning-based decision-making algorithm for a vehicle in dense traffic that will command lane change to the vehicle while avoiding the probable collisions with surrounding vehicles and develop a customized stochastic traffic environment where an RL agent can be trained. There are numerous approaches to this problem ranging from purely rule-based to end-to-end Deep Learning methods.

Then the lane change problem will be solved with deep Q-networks. A hyper-parameter optimization technique is performed to achieve the optimal parameters both for the Reinforcement learning side, such as discount factor,

exploration rate, and neural network such as the numbers of the hidden layers, neurons, learning rate, and so on. Also, a baseline algorithm is devised based on heuristics to provide a comparison with the RL agents. Recalling the complexity of the environment and the uncertainty assigned to the spawned vehicles in the traffic, it is preferred to implement a deep RL algorithm, which is an intrinsically model-free method and appropriate for the problem at hand. Firstly, the problem is solved by generating a Q-table with Q-learning method, and the optimal solution (actions) are retrieved from it. In this case, all possible combinations of state-action pairs should be seen and evaluated, and since the environment in this work is stochastic with many surrounding vehicles, it is not feasible to proceed with the solution based on Q-table. At this step, a function approximation method (neural-network) is implemented to substitute the large Q-table discussed previously, which generates optimal Q-value and its corresponding actions faster and in some cases addresses the failure of the classical Q-table method that would not be able to cover the whole state-action space (Figure 1.1).

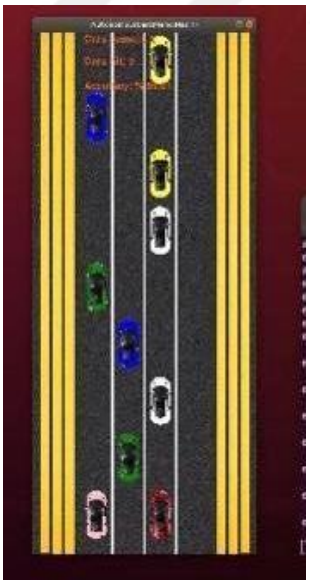


Figure 1.1a : Grid world

L x0 x1 x2	Left	Stay	Right
[0,0, 0, 0, 0]	0.000000	0.000000	0.000000
[0,0, 0, 0, 1]	0.000000	0.000000	0.000000
[0,0, 0, 0, 2]	0.000000	0.000000	0.000000
[0,0, 0, 0, 3]	0.000000	0.000000	0.000000
[0,0, 0, 0, 4]	0.000000	0.000000	0.000000
[0,0, 0, 0, 5]	0.000000	0.000000	0.000000
[0,0, 0, 0, 6]	0.000000	0.000000	0.000000
[0,0, 0, 0, 7]	0.000000	0.000000	0.000000
[0,0, 0, 0, 8]	0.000000	0.000000	0.000000
[0,0, 0, 1, 0]	0.000000	0.000000	0.000000
[0,0, 0, 1, 1]	0.000000	0.000000	0.000000
[0,0, 0, 1, 2]	0.000000	0.000000	0.000000
[0,0, 0, 1, 3]	0.000000	0.000000	0.000000
[0,0, 0, 1, 4]	0.000000	0.000000	0.000000
[0,0, 0, 1, 5]	0.000000	0.000000	0.000000
[0,0, 0, 1, 6]	0.000000	0.000000	0.000000
[0,0, 0, 1, 7]	0.000000	0.000000	0.000000
[0,0, 0, 1, 8]	0.000000	0.000000	0.000000
[0,0, 0, 2, 0]	0.000000	0.000000	0.000000
[0,0, 0, 2, 1]	0.000000	0.000000	0.000000
[0,0, 0, 2, 2]	0.000000	0.000000	0.000000
[0,0, 0, 2, 3]	0.000000	0.000000	0.000000
[0,0, 0, 2, 4]	0.000000	0.000000	0.000000
[0,0, 0, 2, 5]	0.000000	0.000000	0.000000
[0,0, 0, 2, 6]	0.000000	0.000000	0.000000
[0,0, 0, 2, 7]	0.000000	0.000000	0.000000

Figure 1.1b : Q-table

Figure 1.1 : Q-table and grid environment of autonomous car.

The competent to the RL algorithm used in this work is dynamic programming (DP) algorithm, which is known as model-based RL. DP is an offline method that requires the transition model of the environment, either analytically or data-driven. Deriving

the analytic transition model of the environment is not possible for the problem in this work. Also, the data-driven model requires a vast number of trajectories of data that could possibly cover the whole possible maneuvers, which again is not feasible (in terms of time complexity and the number of possible trajectories) in this work due to the stochastic nature of the environment. Therefore, it is decided to use the RL algorithm, which is a model-free algorithm, and mostly applicable for tactical decision making (lane change) algorithms in traffic environments. As it is stated before, raw sensory inputs such as onboard camera video frames or automotive RGB-D sensors were used in most literature studies to train a neural network to predict the action needed to drive the vehicle. The performance of their methods was validated by studies in a number of video games or unreal simulations. The end-to-end learning techniques can suffer from poor reliability in guiding the agent as a consequence of the numerous inconsistencies while interacting in the real world when it comes to implementing these methods in the real world. Towards this purpose, in the thesis, the issue is split into various control level problems, and it is sought to independently overcome each point. In fact, the raw sensory inputs would be used in the challenge of autonomous vehicles to interpret the environment and map it to a more basic two-dimensional space. The AI will focus on higher-level decisions, such as lane change, drive, stop, accelerate, and so on. Decisions at the lower level, such as the steering angle may be made using a classical control technique or AI agent.

The scheme of this method is illustrated in Figure 1.2. The method is more secure than end-to-end techniques. This strategy is just like the methodologies of multi-loop control in control engineering since we do not accomplish a single mapping from the high-level perception with considerable uncertainty to the lowest level actuator control.

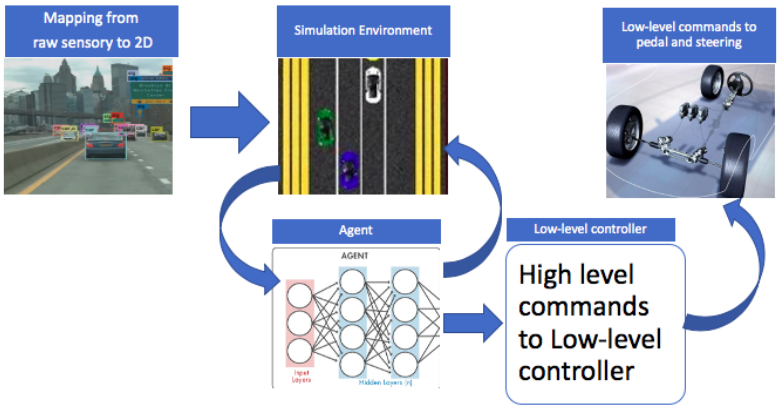


Figure 1.2 : Design of our approach to a self-driving car problem.

In this thesis, the issue of high-level decision-making for an autonomous vehicle is addressed using Q-learning, which is the classical technique of RL. The $\varepsilon-$greedy algorithm is applied to the problem described in the DeepCars simulation environment in which it is designed and implemented. After making comments on the efficiency of the Q-learning algorithm, the same problem is implemented with the deep Reinforcement Learning(RL) method. Finally, the findings are analyzed, and a comparison between the two approaches is carried out.

A comprehensive Q-table analysis is provided to justify the use of Neural Networks as the Q-value estimator in DQN architecture. The observation vector is tailored to summarize the DeepCars environment at each time-step perfectly. A rule-based strategy that generates a greedy policy has been designed using heuristics infinite state machine theory. Then, the RL performance is evaluated by comparing it with the rule-based agent. The results suggested the superiority of the RL algorithm in comparison with human-designed heuristics. Finally, hyperparameter optimization techniques are provided to search for the best parameter setting and the network architecture in the problem.

2. BACKGROUND MATERIAL

2.1 An Introduction on Reinforcement Learning (RL)

Reinforcement learning lies between supervised and unsupervised learning (Figure 2.1). Supervised learning utilizes labels to predict target value. In comparison, unsupervised learning has no labels and predicts the target value by exploring patterns.

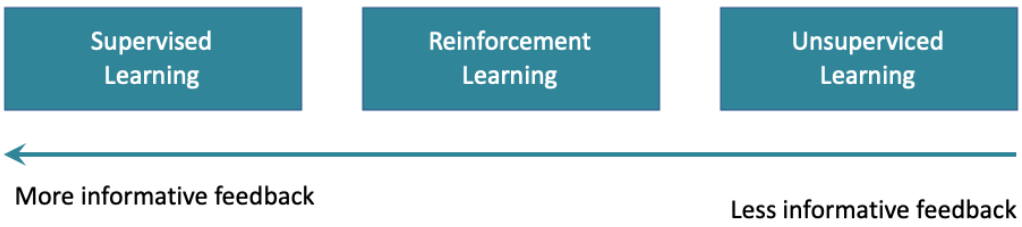


Figure 2.1 : Reinforcement learning on the Machine Learning spectrum.

Reinforcement learning is a machine learning technique that discovers policy through trial and error in an interactive environment. Reinforcement learning uses feedback from its own actions and experiences to take action in the environment. In reinforcement learning, the goal is to find an appropriate action model that would maximize the agent's total cumulative reward. Unsupervised and reinforcement learning uses the mapping between input and output, while supervised learning employs the feedback given to the agent to execute a task. In other words, reinforcement learning utilizes rewards or punishments as signals of positive and negative behavior. Figure 3.3 represents the basic idea and elements involved in a reinforcement learning model.

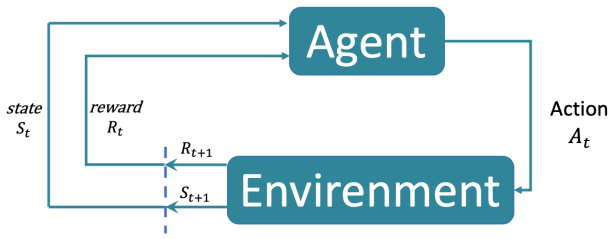


Figure 2.2 : Basic idea and elements involved in a reinforcement learning model.

Reinforcement learning intuitively is inspired by early-stage human and animal learning process. In fact, an agent controller delivers an action to the environment and accordingly receives the next state, reward value of the corresponding state, and action pair as it is illustrated in Figure 2.3.

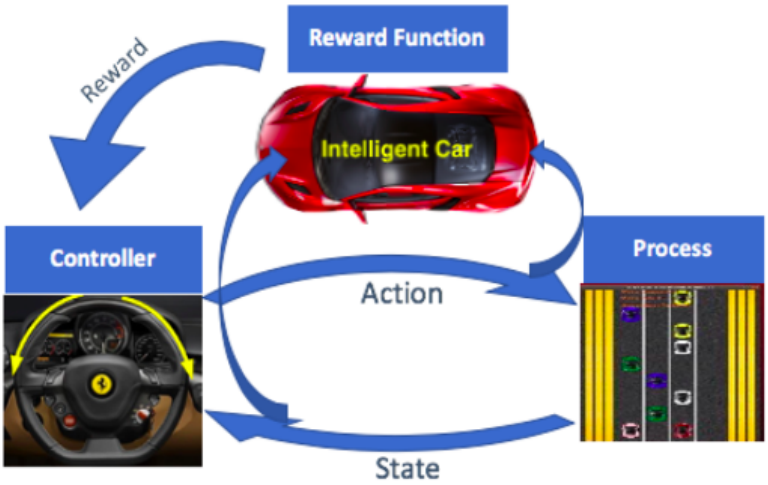


Figure 2.3 : RL-Model.

Key terms that describe the elements of an RL problem are:

Environment: Physical world in which the agent is operating

State: Current status of the agent.

Reward: Environmental feedback.

Policy: Method to map the agent's state to actions.

Value: Upcoming reward that an agent has received by taking action in a particular state.

Let us consider the lane change problem. The goal of the agent is to change lane in the grid world while avoiding accidents and maximize the cumulative reward of changing lane. The grid world is an interactive environment for the agent.

2.1.1 Reinforcement learning

Dynamic programming (DP) and reinforcement learning (RL) intend to control the agent or process in an environment by observing states, performing actions, and receiving rewards [39]. Figure 3.3 summarizes and illustrates the work-flow of the RL.

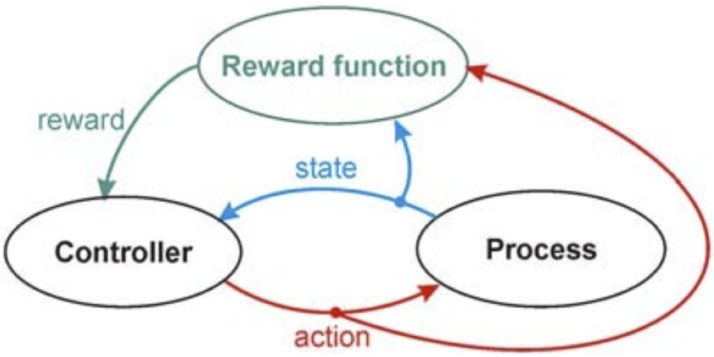


Figure 2.4 : An overview of the RL agent interacting in the environment while observing the states and rewards.

When the agent is in a particular state and takes action, the environment may be either deterministic or stochastic. If the final state is revealed, then the environment is recognized as deterministic; otherwise, the final state values can be described using a probabilistic distribution in which the environment can be named stochastic.

Dynamic programming and reinforcement learning are methods for problem-solving in which actions (decisions) are applied to the system over an extended period to achieve the desired goal. Dynamic programming methods require a system model, while the methods in reinforcement learning do not require a system model. In dynamic programming, the time variable is commonly discrete, and actions are taken at every discrete time step, resulting in a recursive decision-making problem.

Actions are performed in a closed-loop, which means that the result of previous actions is monitored and taken into account in the selection of new actions. Rewards are provided for the assessment of one-step decision-making performance, and the goal is to optimize and maximize long-term performance, calculated by the total award accumulated over the course of interaction (Equation 2.1):

$$Q(x,u) = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) \tag{2.1}$$

Which is the accumulated reward by taking action of a in the state of x. The discount factor $\gamma \in [0,1)$ results in exponential weighting. If we set the discount factor $\gamma = 0$, then this thesis strategy will be short-sighted and rely only on the immediate rewards. If we want to balance between immediate and future rewards, we should set a discount factor to something like $\gamma = 0.9$. If our environment is deterministic and the same actions always result in same rewards, then we can set $\gamma = 1$. The controller considers its rewards. The main purpose is to determine the optimal value of Q for each

state-action pair and choose an action that acquires the maximum value of Q for each state. Now, considering the next state as $x\prime$ and the next action as $a\prime$, it has been proven that [40], if the optimal value $Q^{\star}(x\prime, a\prime)$ of the sequence $x\prime$ was known for all possible actions $a\prime$, then the best strategy is to choose the action $a\prime$ to boost the expected value of $r + \gamma Q^{\star}(x\prime)$ (Equation 2.2):

$$Q^{\star}(x,a) = \mathbf{E}_{x\prime}\{r + \gamma \max_{a\prime} Q(x\prime, a\prime)|x,a\} \tag{2.2}$$

2.1.2 Value iteration (model-based RL)

Q-functions, Q and $Q^{\star}$, are characterized iteratively by Bellman formulas, which are central to iteration value and algorithms of policy iteration. The Bellman Optimality Equation defines $Q^{\star}$ and states that the optimal value of the action a taken in the state x equals the amount of the immediate reward and the discounted optimum value received by the best action in the next state. Therefore, the value of Q can be sequentially calculated using the Equation 2.3:

$$Q_{i+i}(x,a) = \mathbf{E}_{x\prime}\{r + \gamma \max_{a\prime} Q_i(x\prime, a\prime)|x,a\} \tag{2.3}$$

Iterating all state and action pairs outputs in that $Q-$values for all state-action pairs are calculated and converged to the optimum value. These optimum values will be recorded in a table called the Q-table. Consequently, the action that has the highest value is selected for each state. This method is named as value-iteration, which iterates across all state-action pairs and calculates the corresponding Q-values.

These discussions are specific to model-based systems where the system's model and the environment are available. However, the environmental model is not known in our problem, which is the autonomous driving lane change. The decisions that will be taken by the drivers of the other vehicles are unknown. Also, the precise model of roads and highways can never be accurate enough to consider the system as model-based.

2.1.3 Q-learning (model-free RL)

So far, the model-based value iteration was explained. After that, we consider RL, the most widely used algorithm for model-free value iteration, and discuss Q-learning. Q-learning starts with an arbitrary initial Q function and tehn the Q-table is filled with the use of the observed status transitions, i.e., the form data tuples without a model,

$x_k, u_k x_{k+1}, r_{k+1}$ [41, 42]. After each transition, the Q-function is updated using this content tuple as follows (Equation 2.4):

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k \left[r_{k+1} + \gamma \max_{u\prime} Q_k(x_{k+1},u\prime) - Q_k(x_k,u_k) \right] \qquad \textbf{(2.4)}$$

where $\alpha_k \in (0,1]$ is the learning rate. The term between square brackets is the temporal difference, i.e. the difference between the updated estimate of $r_{k+1}+\gamma \max u\prime Q_k(x_{k+1},u\prime)$ of the optimal Q-value of (x_k,u_k), and the current estimate of $Q_k(x_k,u_k)$.

When a Q-table is initialized randomly, its predictions are initially random as well. If we pick an action with the highest Q-value, the action will be random, and the agent performs crude "exploration." As a Q-function converges, it returns more consistent Q-values, and the number of exploration decreases. So one could say that Q-learning incorporates the exploration as part of the algorithm. But this exploration is "greedy," it settles with the first effective strategy it finds. The controller must always make use of its current experience by choosing greedy steps from the current Q function to achieve successful results. This is a typical example of online RL exploration and operation. One traditional way to balance exploitation with Q-learning is to scan ε-greedy [18]. The pseudocode of a greedy ε exploration and exploitation for the Q-learning method is depicted in Figure 2.5, as well.

ALGORITHM Q-learning with ε-greedy exploration.

Input: discount factor γ,
exploration schedule $\{\varepsilon_k\}_{k=0}^{\infty}$, learning rate schedule $\{\alpha_k\}_{k=0}^{\infty}$

1: initialize Q-function, e.g., $Q_0 \leftarrow 0$
2: measure initial state x_0
3: **for** every time step $k = 0,1,2,\ldots$ **do**
4: $u_k \leftarrow \begin{cases} u \in \arg\max_{\bar{u}} Q_k(x_k,\bar{u}) & \text{with probability } 1-\varepsilon_k \text{ (exploit)} \\ \text{a uniformly random action in } U & \text{with probability } \varepsilon_k \text{ (explore)} \end{cases}$
5: apply u_k, measure next state x_{k+1} and reward r_{k+1}
6: $Q_{k+1}(x_k,u_k) \leftarrow Q_k(x_k,u_k) + \alpha_k[r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u') - Q_k(x_k,u_k)]$
7: **end for**

Figure 2.5 : Pseudo code for Q-learning algorithm.

2.1.4 Markov decision processes (MDPs)

Markov Decision Processes are mathematical frameworks to describe an environment in reinforcement learning, and almost all RL problems can be formalized using MDPs. An MDP (Figure 2.6) consists of a set of finite environment states S, a set of possible actions A(s) in each state, a real-valued reward function R(s), and a transition model P(s', s | a). However, real-world environments are more likely to lack any prior knowledge of environment dynamics. Model-free RL methods come handy in such cases.

MDP is appropriate for sequential decision making, where it needs full access to the environment states. If we can define an environment such that the next states only depend on the current states and do not require previous information, that environment or process can be modeled as MDP.

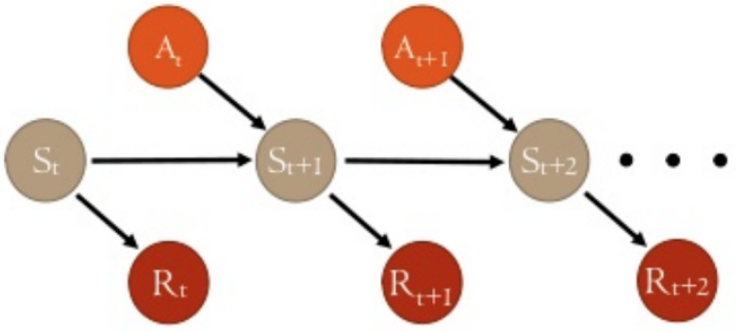


Figure 2.6 : MDP-Model.

Markov's decision process is the formalization of the reinforcement learning problem that solves most problems with discrete actions. We can say that if we formalize a problem based on MDP, we can think of it as the reinforcement learning problem as trying to solve it with RL solutions. Basically, an MDP is a 5-tuple consisting of state, action, transition model, reward, and discount factor. As an example in Figure 2.7, we can see that in S_1 state, if the agent takes action a_0, there is 0.1 probability that nothing will happen and it remains in the same state, there is 0.2 possibility that it will end up in S_2 state with no reward, and with 0.7 probability it will end up in S_0 with +5 reward value. This thesis environment intuitively follows similar rules.

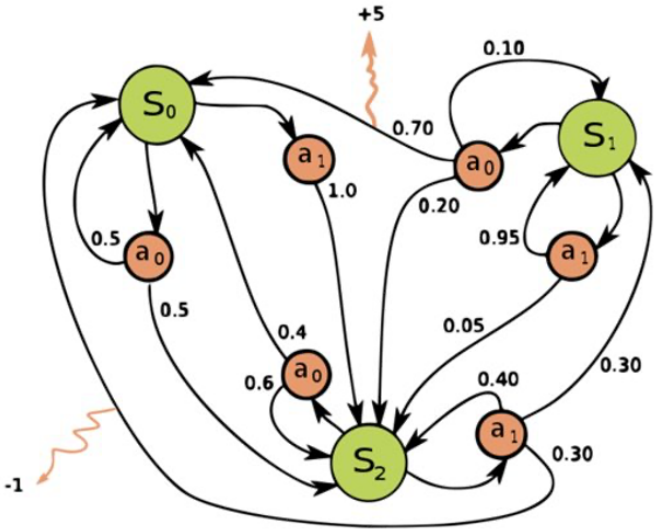


Figure 2.7 : MDP-Model tuple.

2.1.5 Q-learning or value-iteration methods

Q-learning is a generally used model-free approach that can be used to construct a self-playing agent. It learns the action-value function Q(s, a), how good it is to take action at a particular state. Essentially, a scalar value is assigned to a given state over an action. A good representation of the algorithm is provided in the following chart in Figure 2.8.

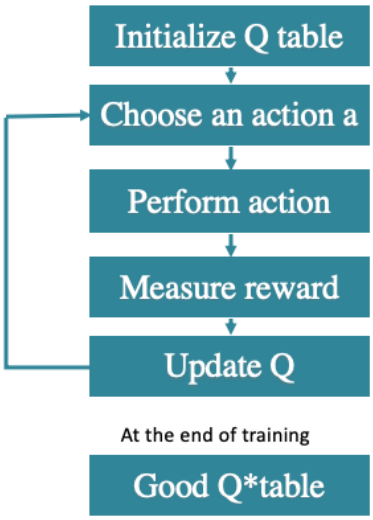


Figure 2.8 : Q-learning steps.

2.1.6 Deep Q neural network (DQN)

Neural Networks can also be used in Q-learning. The motivation behind this is related to large state space environments where defining a Q-table would be a very complex, time-consuming, and challenging task. Instead of constructing a Q-table,

Neural Networks approximate Q-values for each state-based action (Figure 2.9). Naive DQN [43] has three convolutional layers and two fully connected layers to estimate Q values directly from images. On the other hand, the linear model has only one fully connected layer with some learning techniques. Both models learn Q values using Q learning methodology. The experiments show that naive DQN has very poor results, worse than even a linear model, because DNN easily overfits in online reinforcement learning. DQN overcomes unstable learning by using mainly four techniques.

- Experience Replay: The most important trick is experience replay. DNN is easily overfitted current episodes. Once DNN is overfitted, it's hard to produce various experiences. To solve this problem, experience replay store experiences, including state transitions, rewards, and actions, which are the necessary information to perform Q learning, and makes mini-batches to update neural networks. This breaks the similarity of subsequent training samples, which otherwise might drive the network into a local minimum. Also, experience replay makes the training task more similar to usual supervised learning, which simplifies debugging and testing the algorithm.

- Target Network: Unstable target function makes training difficult. So the target network technique fixes parameters of the target function and replaces them with the latest network every thousand steps.

- Clipping Rewards: Each RL problem has different score scales, which can make the training unstable. Thus Clipping Rewards technique clips scores so that all positive rewards are set as +1, and all negative rewards are set as -1.

- Skipping Frames: Skipping frames technique is that DQN calculates Q values every four frames and uses the past four frames as inputs instead of every 30 frames. This reduces the computational cost and gathers more experiences.

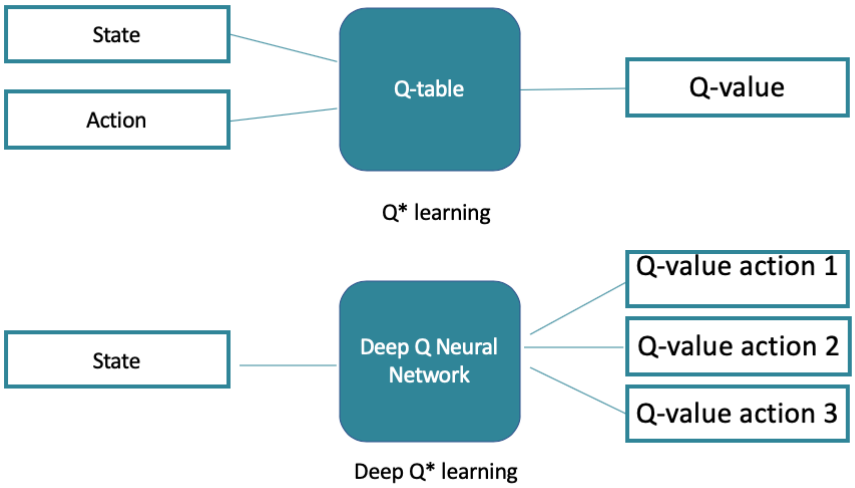


Figure 2.9 : Deep Q neural networks (DQN) architecture.

In order to elaborate Deep Q Neural Network, some fundamental information about deep neural networks must be presented first. Some introduction will be discussed in the next section.

2.2 Neural Networks

Artificial neural networks generally consist of the input layer, hidden layers, and output layers. The input layer consists of a list of input features. For example for a single image, the input layer consists of a single array of three dimensions, the input features are the width and height of the image (measured as the number of pixels) and the number of color channels (RGB), i.e., one channel for black and white and three or more channels for color images. Hidden layer(s) allow the designer to model nonlinear and complex functions. They are "hidden" because the true values of their nodes are unknown in the training dataset. In fact, we only know the input and output. Each neural network has at least one hidden layer, and each deep neural network has multiple hidden layers. The most common type of hidden layers is convolution and fully-connected layers. In between these layers, different activation functions are used, such as ReLU, Leaky ReLU, Tanh, etc. The activity of neurons can be changed using the activation functions. Neurons are the processing units of the network. Each neuron weighs and sums the different inputs and passes them through an activation function. The output layer is the final layer with neurons. This is where the data comes out of the model. So the number of neurons needs to be exactly the number of

outputs, i.e., it depends on the problem that is solved using the network. For example in CIFAR-10 [44], which is a classification dataset, there are ten outputs since the problem is to classify the input images to one of the ten categories in this data set. One of the advantages of deep neural networks over those kinds of datasets is that no manual feature is required, such as specific shapes, colors, edges, etc. A three-layer, fully connected network is shown in Figure 2.10.

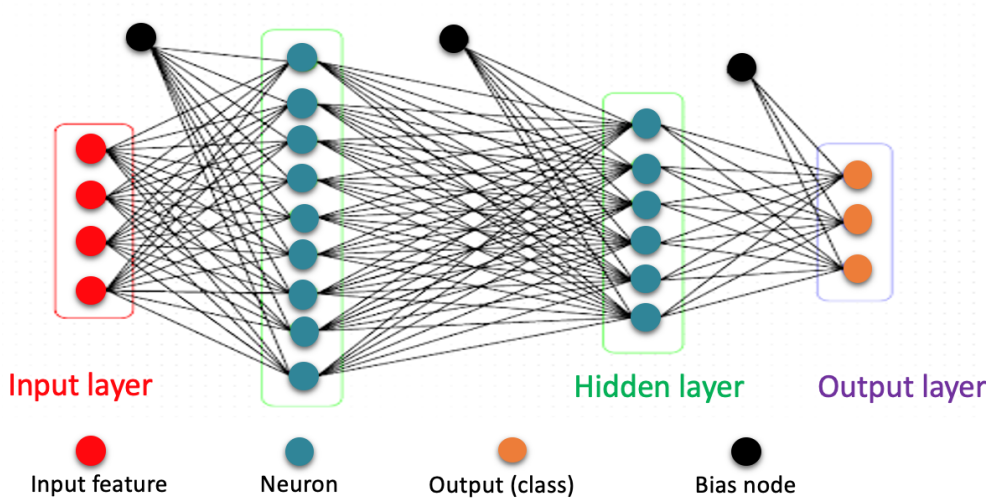


Figure 2.10 : A three-layer fully connected neural networks (DNN) .

Depending on the problem and data set, the network can be trained in two ways, either supervised learning or unsupervised learning. If we take a supervised approach, our data set needs to be labeled in advance. Using unsupervised learning, the data set will be unlabeled, and the network will learn by itself, or some feedback must be provided to the network. In both cases, we need to define a loss function, which is then minimized during the learning process. With supervised learning, we need to give the algorithm examples of when and in which situation must do the action. For unsupervised learning, the algorithm needs to figure out what the most characteristic features. Basically, we would give the algorithm the above Figure 2.10 and it would group them by characteristics.Let's overview convolutional neural networks that are mostly consist of convolutional, pooling, fully connected layers with some activation functions in between.

2.2.1 Deep neural networks (DNNs)

Researchers were inspired by mother nature, and they have been trying to imitate the inner workings of a biological brain for years. The resulting mathematical

representation is the artificial neural network. The artificial neural network is a system of nodes, called neurons that receive inputs and send outputs to each other. The perceptron is a mathematical model of a biological neuron. While in actual neurons, the dendrite receives electrical signals from the axons of other neurons, in the perceptron, these electrical signals are represented as numerical values. At the synapses between the dendrites and axons, electrical signals are modulated in various amounts. This is also modeled in the perceptron by multiplying each input value by a value called the weight. An actual neuron fires an output signal only when the total strength of the input signals exceeds a certain threshold. We model this phenomenon in a perceptron by calculating the weighted sum of the inputs to represent the total strength of the input signals and applying a step function on the sum to determine its output. As in biological neural networks, this output is fed to other perceptrons.

2.2.2 Convolutional neural networks (CNNs / ConvNets)

Convolutional Neural Networks are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product, and optionally follows it with a non-linearity (activation function). The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they have a loss function on the last (fully-connected) layer. A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. We will stack these layers to form a full ConvNet architecture. It is common to periodically insert a Pooling layer in-between successive Convolution layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting. Pooling layers buy you a translation invariance; hence the network becomes insensitive to the location of an object in the image. That makes perfect sense for a classification task like ImageNet, but in autonomous driving, the location of objects in the environment is crucial in determining the potential reward, and we wouldn't want to discard this information! Neurons in a fully connected layer have full connections to all activations in the previous layer. Their activations can hence be computed with a matrix multiplication

followed by a bias offset. We have seen that Convolutional Networks are commonly made up of only three layer types: CONV, POOL (we assume Max pool unless stated otherwise), and FC (short for fully-connected). We will also explicitly write the RELU activation function as a layer, which applies elementwise non-linearity. For instance, AlexNet [45] is the first ConvNet proposed in 2012 for image classification on ImageNet data set and is illustrated in Figure 2.11. AlexNet has achieved an incredible score in ILSVRC 2012, image classification competition by using DNN.

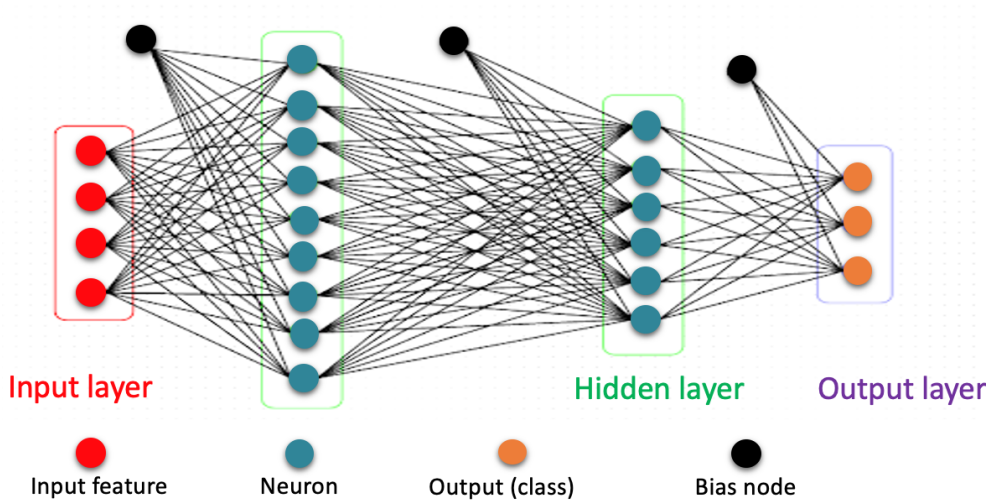


Figure 2.11 : AlexNet deep convolutional neural networks.

3. LANE CHANGE DECISION MAKING USING REINFORCEMENT LEARNING

In this thesis, we aim to utilize the highway driving task's decision-making problem for self-driving cars. We will show the proposed methods in the literature using control strategies and classical optimization. Then, utilizing a rule-based approach, we will generate high-level actions to navigate through the traffic. We will investigate these methodologies and their correlation with the accuracy of the measurements and the robustness of the mathematical descriptions. This section describes the various approaches to the autonomous driving problem using classical and state-of-the-art methods. We need to break down a complex issue like ADAS into smaller issues that are hierarchically connected and can be addressed independently. The details of the stack of different layers are discussed in section 3.1.1. Also, the simulation environment, devised RL-based solution with different experimental setups are explained.

3.1 Advanced Driver-Assistance Systems (ADAS)

Advanced driver-assistance systems (ADAS) are electronic systems that help drivers in parking functions and driving. ADAS systems use automated equipment for detecting nearby obstacles and performing lane change safely. Many of the road accidents are caused by human error, which is almost avoidable by ADAS. ADAS increases safety through early warning and automated systems. Some of these systems are built into certain vehicles. Technological innovation and the explosion of automation initiatives have greatly increased the popularity of safety systems in vehicles. A small sample of the available systems include the following:

- Adaptive cruise control
- Traction control
- Anti-lock brakes

- Traffic signals recognition
- Forward collision warning
- Lane departure warning
- High beam safety system

Future-generation ADAS will implement wireless network connectivity to enable vehicle to vehicle and vehicle to infrastructure interaction. Basically, cars will be able to communicate with each other and through a great mainframe to provide a more safe, automated driving experience.

3.1.1 Classical approaches

Self-driving cars are equipped with a stack of sensors that perceive the environment information and provide raw measurements. Producing safe and reliable driving policies require complex processing of the sensory measurements and derivations involving comprehensive mathematical expressions that summarize the surrounding actors' positions and behaviors. It is not feasible to generate a single mathematical expression that describes the environment and produces optimal actions. It is more favorable to divide the entire problem into smaller problems that are hierarchically connected and can be addressed independently. Advanced driver-assistance systems (ADAS) can be represented in a multi-layer hierarchical architecture, as illustrated in Figure 3.1.

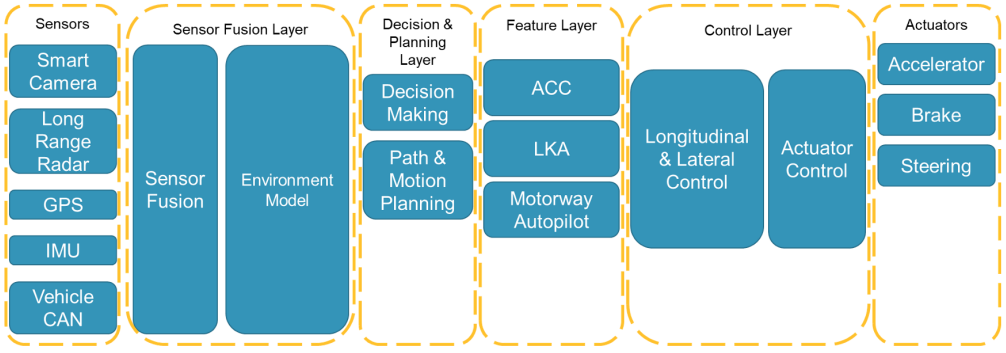


Figure 3.1 : Schematic architecture of ADAS.

The first layer of ADAS consists of various sensors' configurations, including but not limited to, smart camera, radar, LiDAR, GPS, and IMU. The information generated by these measurements is grouped and processed in the sensor fusion layer. These

processes may include the 3D and 2D object detections, state estimation, and object tracking using Kalman Filter. The sensor fusion layer generates a birds-eye-view representation of the ego car's environment, which incorporates the actors' position, velocity, and acceleration.

Decision-making and planning layer receives the occupancy grid. It processes the information to generate the desired path to achieve the destination as fast as possible and avoid collisions simultaneously. Typically, this layer consists of behavior planning and trajectory generation. The behavior planning formulates the problem as a finite state machine that defines the environment's various configurations as different state values and produces high-level commands such as, LaneChange, Stop, and Accelerate. The local planner receives this information and commands and generates a dynamically feasible and optimal trajectory which captures the optimal driving criteria such as comfort and safety. The generated paths usually consist of time-stamped waypoints in the Cartesian coordinate system.

The feature layer is optionally deployed to program several complicated driving scenarios such as lane-keeping and cruise control. This layer increases the level of intelligence of the ADAS, contributing to safety and reliability.

The generated optimal path from the other layer is fed into the lower-level control layer. The speed is controlled in a separate longitudinal control loop by producing the desired throttle percentage. The lateral controller generates the required steering angle to follow the generated waypoints.

3.1.2 Supervised end-to-end techniques

Here in Figure 3.2, the architecture of an end-to-end learning scheme can be seen where there is a direct mapping between sensor inputs and the model outputs, which are actuators commands. With end to end learning, cars can fully learn to drive with only steering angle and video as the training inputs. No human help is needed. The inputs are usually from a high-dimensional feature space (e.g., images or point clouds). This is opposed to traditional processing pipelines, where at first objects are detected in the input image, after which a path is planned, and finally, the computed control values are executed. End to end learning can also be formulated as a back-propagation algorithm scaled up to complex models. The back-propagation algorithm for gradient estimation

in deep networks is now efficiently implemented on parallel Graphics Processing Units (GPUs). This kind of processing allows the training of large and complex network architectures, which in turn require huge amounts of training samples. Using this approach [22] a CNN can be trained, which maps raw pixels from a single front-facing camera directly to steering commands. The training data is composed of images and steering commands collected in driving scenarios performed in a diverse set of lighting and weather conditions and on different road types. Prior to training, the data is enriched using augmentation, adding artificial shifts, and rotations to the original data.

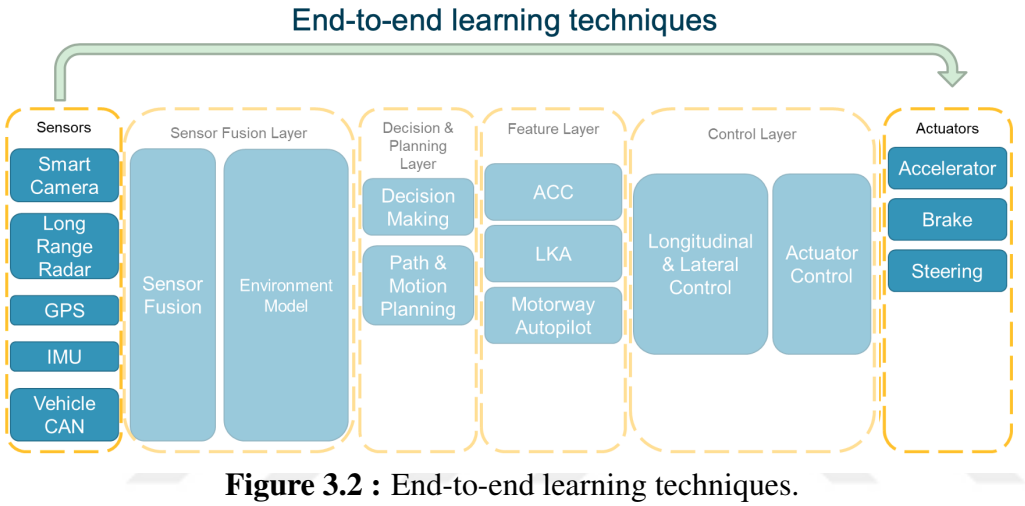


Figure 3.2 : End-to-end learning techniques.

3.1.3 Reinforcement learning approaches

Another method is to use reinforcement learning, as shown here in Figure 3.3, by getting states of the environment as the output of the fusion layer and directly learning to perform optimal commands to the actuators while trying to achieve a specific goal. This is mainly accomplished in simulation, where an autonomous agent can safely explore different driving strategies. The agent will receive the current state of the application, deciding on the next control commands, and then getting a reward on the next iteration. Reinforcement learning-based control seems to be the inclusion of classical model-based control techniques. The classical controller provides a stable and deterministic model on top of which the policy of the neural network is estimated. In this way, the hard constraints of the modeled system are transferred into the neural network policy. A policy is trained on real-world image data and for the task of aggressive driving. In this case, a CNN referred to as the learner is trained with optimal trajectory examples provided at training time by a predictive model controller.

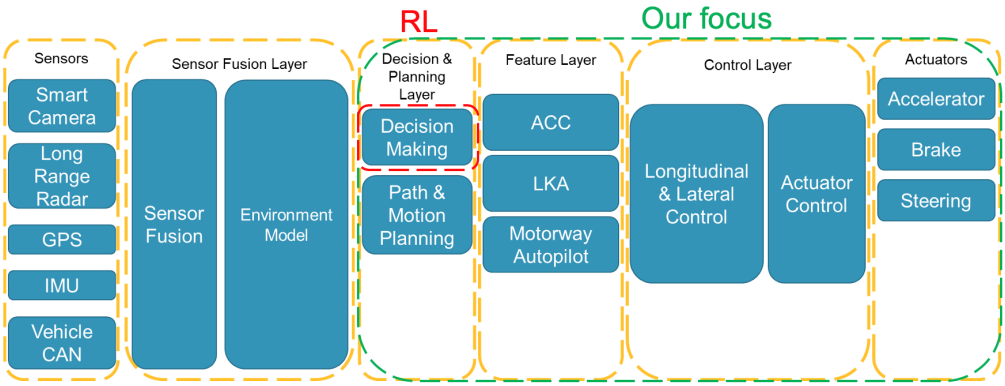


Figure 3.3 : Our focus on end-to-end learning techniques.

3.2 Lane Change Maneuver

Lane change maneuver is defined as a driving maneuver that moves a vehicle from the current lane to the adjacent lane, either left or right, in the same direction. It involves both lateral and longitudinal dynamics, making it a critical maneuver compared to the maneuvers solely based on longitudinal dynamics. Making a lane change maneuver automated is of great importance in higher levels of autonomy [46], precisely level 2 and above.

Different approaches are taken for automated lane change problems, ranging from purely rule-based to fully AI-based end-to-end methods. As described in section 1.2, most of the machine learning-based approaches consider the lane change maneuver as a decision-making problem, and once the algorithm decides on the lane-change maneuver, classical methods are utilized to accomplish the lane change maneuver through generating required trajectory and control commands for the actuators.

This thesis uses reinforcement learning for automated lane change decision making in a customized traffic environment. Since RL is an interactive learning method, the simulation environment is a principal component and plays a critical role in the whole RL setup. Accordingly, this thesis devises a customized stochastic traffic simulation environment with uncertainty in the spawning of upcoming vehicles. From now on, the aforementioned environment is called the DeepCars simulation environment.

3.3 DeepCars Simulation Environment

This thesis assumes a perfect mapping from the 3D world to the 2D simulation environment with no information loss. The sensor fusion layer performs the required mapping, as shown in Figure 3.1. Pygame [47], which is a set of open-source and free Python modules designed for writing video games, is used to construct the DeepCars simulation environment. Figure 3.4 shows a layout of the simulation environment used in this thesis.

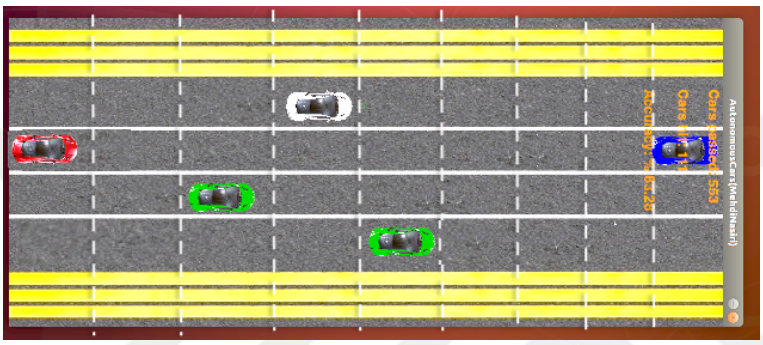


Figure 3.4 : Layout of DeepCars simulation environment.

In our experiment, input of the game are high-level control commands which provide the game reward and state as the output. The states and the actions in our RL algorithm are discrete to formalize the challenge in Markov Decision Process (MDP). Moreover, the task area is comprised of three main actions:

- Right: Move to the right lane
- Left: Move to the left lane
- Stay: Do not change the lane

The Markov Equation 3.1 is stated as:

$$X = \begin{bmatrix} lane\# & x_0 & x_1 & \dots & x_n \end{bmatrix} \tag{3.1}$$

in which the lane number of the agent car is the first item and the distance to the nearest car in lane i is shown with x_i. It is noteworthy to mention that the grid numbers and lane numbers start from 0. For example, the state vector for the Figure 3.5 is as $X = \begin{bmatrix} 1 & 3 & 8 & 2 & 4 \end{bmatrix}$

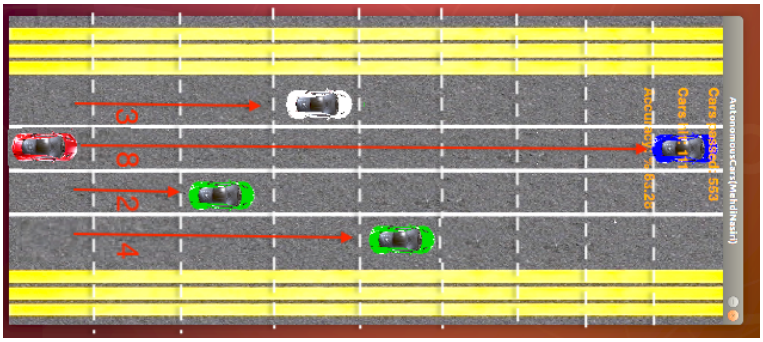


Figure 3.5 : An example for the illustration of game state.

Also, the sight of the car is 9 (from 0 to 8) which means a vehicle agent can see the farthest car that is nine rectangles (grids) far away. In fact, these restrictions represent the limit of the range of sensors used outside of the car.

3.4 Reinforcement Learning for DeepCars

The Q-learning algorithm must be used as a starting point (Figure 2.5) for high-level agent control in the DeepCars simulation. The most important goal is to train an agent to avoid an accident with other cars in the simulation environment. The four-lane highway is shown in Figure 3.6.

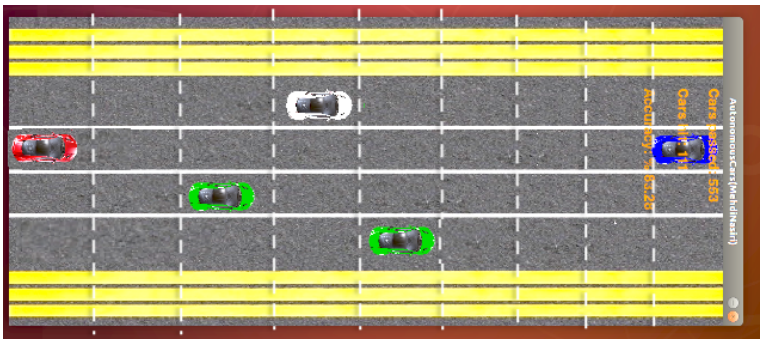


Figure 3.6 : Four-lane DeepCars environment.

By utilizing the Equation 3.1 as the Markov state, we would have $4 \times 9^4 = 26244$ combinations. This means that the size of the Q-table is $4 \times 9^3 \times 3$ which means 78732 number of elements in Q table. For instance to understand environment we will illustrate a cropped part of the initial Q-table in Figure 3.7.

L x0 x1 x2 x3	Left	Stay	Right
[0, 0, 0, 0, 0]	0.000000	0.000000	0.000000
[0, 0, 0, 0, 1]	0.000000	0.000000	0.000000
[0, 0, 0, 0, 2]	0.000000	0.000000	0.000000
[0, 0, 0, 0, 3]	0.000000	0.000000	0.000000
[0, 0, 0, 0, 4]	0.000000	0.000000	0.000000
[0, 0, 0, 0, 5]	0.000000	0.000000	0.000000
[0, 0, 0, 0, 6]	0.000000	0.000000	0.000000
[0, 0, 0, 0, 7]	0.000000	0.000000	0.000000
[0, 0, 0, 0, 8]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 0]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 1]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 2]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 3]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 4]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 5]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 6]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 7]	0.000000	0.000000	0.000000
[0, 0, 0, 1, 8]	0.000000	0.000000	0.000000
[0, 0, 0, 2, 0]	0.000000	0.000000	0.000000
[0, 0, 0, 2, 1]	0.000000	0.000000	0.000000
[0, 0, 0, 2, 2]	0.000000	0.000000	0.000000
[0, 0, 0, 2, 3]	0.000000	0.000000	0.000000
[0, 0, 0, 2, 4]	0.000000	0.000000	0.000000
[0, 0, 0, 2, 5]	0.000000	0.000000	0.000000
[0, 0, 0, 2, 6]	0.000000	0.000000	0.000000
[0, 0, 0, 2, 7]	0.000000	0.000000	0.000000

Figure 3.7 : A crop of the initial Q-table for the 4-lane problem.

The next step is to fill the Q table and train the agent using the Q-learning algorithm. We have to define other parameters to do this. The algorithm parameters are chosen accordingly:

- value of discount factor: $\gamma = 0.9$
- value of learning rate: $\alpha = 0.01$
- value of exploration: $\varepsilon = 0.2$
- value of exploitation: $1 - \varepsilon = 0.8$
- reward: $r = \begin{cases} +1 \text{ every frame (no collision happened)} \\ -1 \qquad\qquad \text{if collision happened} \end{cases}$

We trained the agent for 10000 frames, which took almost ten minutes for less than three layers. In the test set, the resulting Q-table was saved. Our agent performance has been tested and recorded for frames of $1,000$. We calculated the accuracy in the following terms in Equation 3.2 to comment on the agent 's performance:

$$\text{Accuracy} = \frac{\text{\# of passed cars}}{\text{\# of all cars (passed + collided)}} \times 100 \qquad \textbf{(3.2)}$$

We got a promising result for the training and testing data sets, the results are shown as follows:

- Training: Accuracy = 99.04%
- Test: Accuracy = 97.28%

This situation occurs because the training set did not experience all of the combinations of state-action. And since the agent does not use any approximations of functions, similar situations can not be determined, and accidents may happen. For instance, cropped Q-table is shown in Figure 3.8 for the situation in Figure 3.9 ($X = \begin{bmatrix}1 & 2 & 5 & 4 & 0\end{bmatrix}$) Changing the lane to the right is the optimal action. This example shows how decisions are made using the Q-table, which has already been filled. Although, it is obvious that the optimal action for similar situations (e.g., $X' = \begin{bmatrix}1 & 2 & 5 & 4 & 0\end{bmatrix}$) is also the same, while in Q-table the corresponding row is still zero since the agent has never experienced X'. So if the agent faces the X', the action is chosen will not be optimal. This example illustrates the importance of the approximator function by which the agent can approximate similar situations never experienced in the training set and select the optimal action accordingly. In the next section, this issue is investigated further.

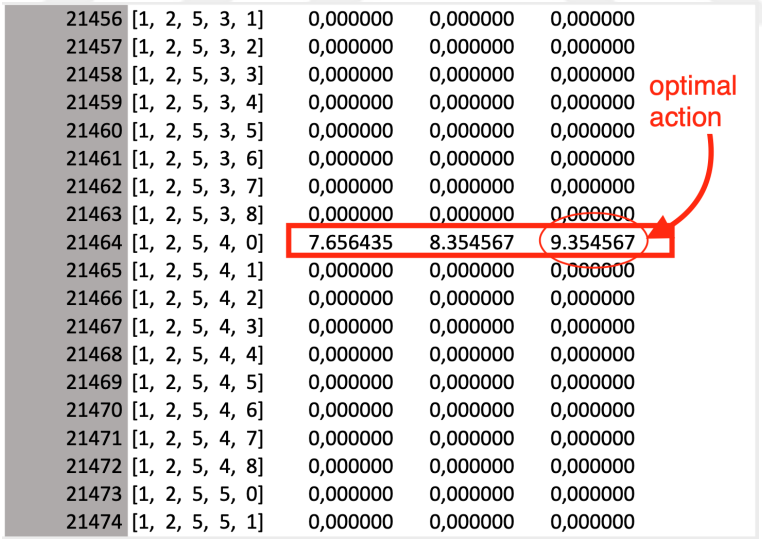


Figure 3.8 : A crop of the trained Q-table for the 4-lane problem.

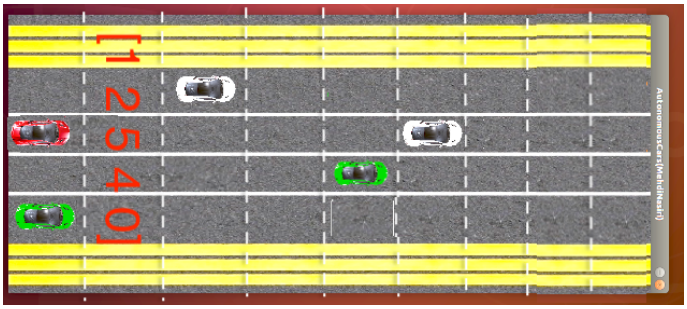


Figure 3.9 : A screen-shot of the game in test level for the state shown in Figure 3.8

3.5 Deep Reinforcement Learning for DeepCars

As mentioned in the past section, the Q-learning lacks the generalization of the ideal actions to similar cases. Furthermore, the curse of dimensionality is another problem when we use the Q-learning method. In our case, as the quantity of lanes increases, the number of Markov states rises up massively. Besides other states similar to the speeds and velocity of the cars makes the issue even worse. This causes us to use the approximator function. Clearly, the equations of agent and environment movements and dynamics are highly nonlinear in overall, so using linear function approximators may not lead us to the favorable results.

Recently, the deep neural networks [48] have proven to be a successful option as a nonlinear function approximator for highly coupling nonlinear systems. As explained in chapter 1, there are many scientists who have studied and implemented the deep reinforcement learning approaches to the problem of high-level autonomous cars decision making. In this thesis, the proposed approach in [35] is implemented. The primary purpose here is to construct a single algorithm that would be able to learn a wide range of skills on a variety of demanding tasks. The principal concept is explained in the following.

The position of the player car and the location and direction of movement of the other cars will determine the state of the environment in the simulation environment, although the intuitive representation is specific to the game. A question that can be asked is whether it is possible to come up with something more universal that would suit all the conditions of the games.

To design a suitable Markov state representation that meets all of the expectations discussed above, we have studied the literature and came up with the following options.

- Environment matrix
- Screen pixels

In the first approach, i.e., environment matrix, we omitted the line-of-sight limitation of the ego and summarized the vehicles' positions in a simple matrix. The matrix size is equivalent to the number of grids on the screen. For instance, the corresponding matrix 3.3 for the game state demonstrated in Figure 3.5 is as follows,

$$X = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \tag{3.3}$$

Trivially, the matrix dimension matches the grid size of the environment. In addition, zeros indicate empty grids; ones show that the grid is occupied by an actor, and 2 represents the ego vehicle's location. Flattening the matrix before feeding into the neural network transforms the state matrix to a vector. In addition, instead of using the number two, one may append the ego's position as a binary vector to the flattened vector. This normalizes the values of the state to be zeros and ones, which stabilizes the training of the neural network.

The second option for the Markov state vector is screen pixels, and they indirectly carry all relevant details about the game situation including the cars' location in the scene and the player's vehicle.

If the same pre-processing is applied to the game screens as the Q learning algorithm in Figure 3.9, the last frame is taken, reduced to 60×60 and converted to gray with 256 gray levels, we would have a possible player status of $256^{80\times80}$ and the same number of rows in our imaginary Q table. One would argue that several pixel combinations (and thus states) never happen. It would also be possible to present it as a sparse table containing only the visited states. Even then, most states are rarely visited, so it would take a very long time for the Q-table to converge. Ideally, we would also like a good

Q-value, the estimate for states that we have not yet seen. Instead of giving neural network states and actions to estimate the associated Q-value, we could only use the game screens to input and output the Q-value by each possible action. This technique has the benefit that if we want to change the Q-value or choose the action with the highest Q-value, we have to go one way forward through the system and have all the Q values available instantly for all actions.

Q-values can be any real values which make it a task of regression, that can be optimized with the following simple squared error loss (Equation 3.4):

$$L = \frac{1}{2}[r + \max_{a'} Q(s', a') - Q(s, a)] \quad \textbf{(3.4)}$$

Given a transition (s, a, r, s'), the Q-table update rule in the prior algorithm must be replaced with the following:

I: Do a feedforward pass to the current state s to get the predicted Q-values for all actions.

II: Do a feedforward pass for the next state s' and calculate the maximum total network output, $\max_{a'} Q(s', a')$

III: Set Q-value aim for action to $r + \max_{a'} Q(s', a')$ (use the max calculated in step 2). For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those outputs.

IV: Update the weights using back-propagation.

The suggested network is called Deep Q-Network (DQN) [35]. During the game, all the (s, a, r, s') events are saved in a replay memory. When training on a network, random mini-batches from replay memory are used instead of the other recent transitions. Otherwise, it could lead the network to a local minimum. Experience replay also makes the task of training more similar to the usual one. Supervised learning simplifies the algorithm's debugging and testing. In fact, we gather human-beings perception, and then utilize their perception to train the network. The overall structure of the training and testing algorithms is shown in Figure 3.10 and Figure 3.11, respectively.

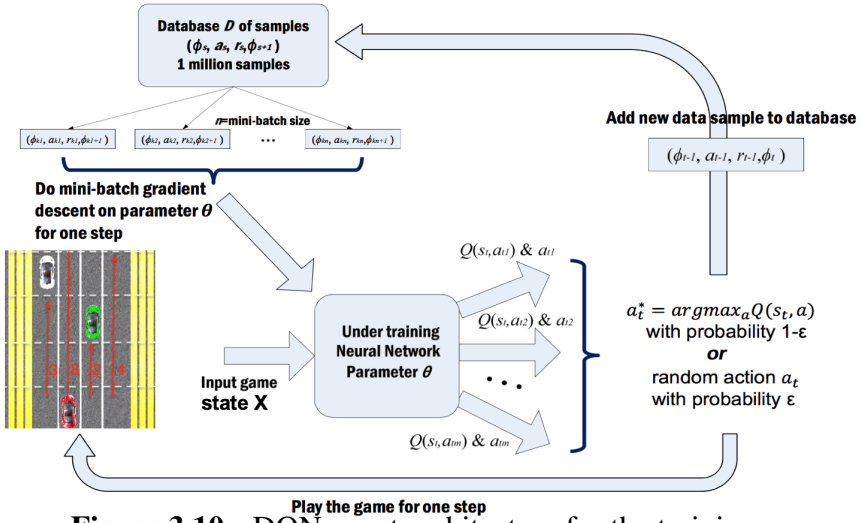


Figure 3.10 : DQN agent architecture for the training.

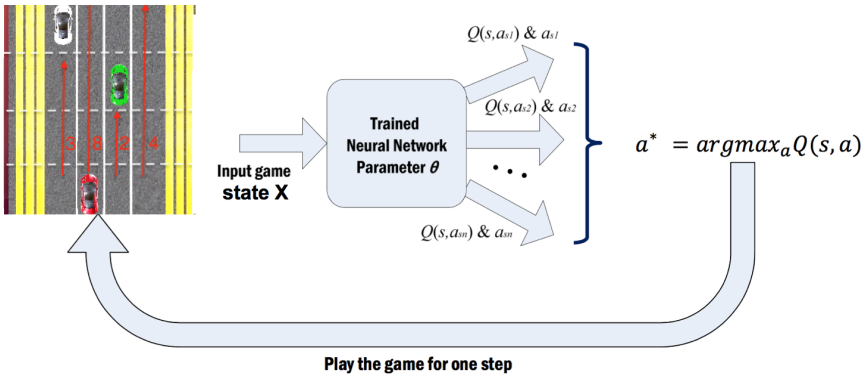


Figure 3.11 : DQN agent architecture for the test.

Following the separate options we have defined for the Markov state vector - environment matrix and screen pixels - we have utilized separate neural network architectures to estimate the Q-values.

3.5.1 Dense model

In this section, we try to train the Deep Q-learning model summarized in Figure 3.10. As we are planning to utilize the flattened environment matrix (Equation 3.3) as the observation space for RL, we expect densely connected layers would give us a promising result. Our expectation is to improve the results that we achieved in section 3.4. Designing a suitable network architecture for the neural network to be used in DQN is not a trivial process.

A clear understanding of the state and action spaces, environmental uncertainty, and problem definition is required to come up with an efficient and precise architecture. A rule of thumb indicates that the number of neurons used in the hidden layer should be a number between the state and action sizes. In our case, the state vector consists of 36

values, and three action values define the optimal decision. This gave us a clue to start with shallow architectures consisting of only three layers and 2 to 4 neurons on each. However, as we increased the density of the network by using more neurons, the results got improved. Thus, we decided to perform a simple hyper-parameter optimization to tune the hyper-parameters of the model.

4. HYPERPARAMETER OPTIMIZATION

4.1 Deep Neural Networks

A consensus among the computer science researchers states that there is not a universal optimal hyperparameter setting that fits for all problems. In neural networks, a hyperparameter is a parameter that does not change during the training. The learning rate, discount factor, number of hidden layers, and the number of neurons in each layer are good examples of such parameters. The various sets of these variables result in a different performance for the model. An important problem is how to set these values to get the best results.

A well-known approach is to perform hyperparameter optimization. Separating the parameter to be optimized as a different variable in the parameter space and defining the test accuracy as a negative cost value transforms the problem into an optimization setup. Multiple approaches are available to perform the optimization, most of which require multiple training and validation phases to converge to a (sub)-optimal setting. Linear search, Grid Search, Bayesian Optimization, and Genetic Algorithms are examples of the available methods. Each approaches the problem from different perspectives.

As we expect to have partially shallow network architecture, we decided to choose the Grid Search. In this approach, we discretize each parameter into a set of candidate values. Then, we create a multi-dimensional space consisting of all of the combinations for parameters. Obviously, this will create a complex hyper-space that would be infeasible to search through all of the candidate points in the space. Thus, we deliberately fixed some of the parameters - e.g., activation functions to ReLu, learning rate to 0.01, and loss function to categorical cross-entropy - to shrink the space dimensions and facilitate the optimization.

Fixing some of the parameters enabled us to perform the grid search optimization on the density of the layers. We decided to optimize the number of neurons in the hidden

layers as they play the most crucial role in neural networks training. Exploiting various network architectures resulted in various training performance, as seen in Figure 4.1.

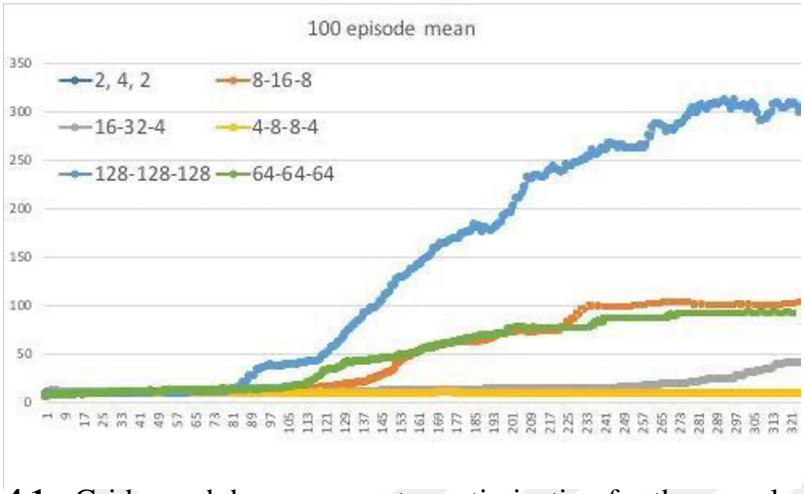


Figure 4.1 : Grid search hyperparameter optimization for the neural networks architecture. 128-128-128 indicates a three-layer network with 128 neurons in each layer.

In contrary to our expectations, the model performance has improved as we moved deeper. It seems using three layers, each consisting of 128 neurons, is performing better than other architectures as it receives the highest mean reward during the training. One may suggest comparing the network performances in terms of accuracy (Equation 3.2) instead of the mean episodic rewards. Figure 4.2 suggests that the same architecture that resulted in the highest mean reward also performed better in terms of avoiding collisions with other cars.

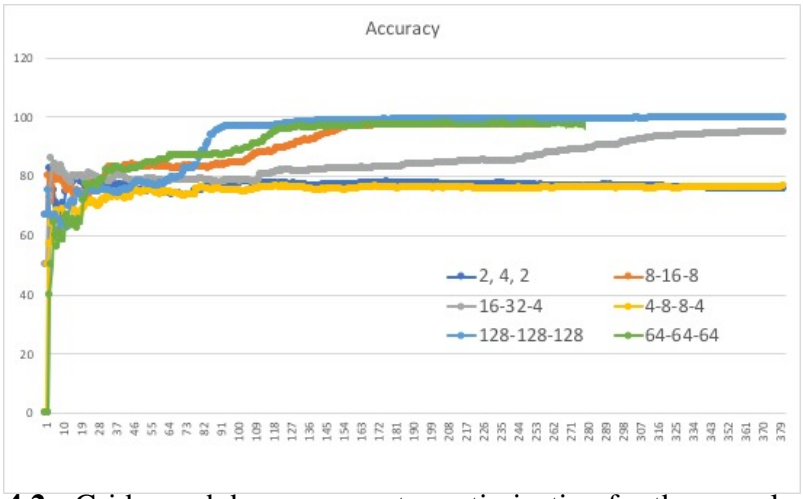


Figure 4.2 : Grid search hyperparameter optimization for the neural networks architecture. 128-128-128 indicates a three-layer network with 128 neurons in each layer.

Although it looks like the model received 100% accuracy, this is not the case. After training the model for 100000 frames and testing on 10000 frames, we have received the following performances:

- Training: Accuracy = 97.03%
- Test: Accuracy = 96.05%

This is a considerable improvement comparing with the previous section where we used the classical Q-learning with Q-tables.

4.2 Deep Convolutional Neural Networks

This improvement motivated us to look for more complex architectures and state representations, hoping to improve the performance even further to 100%, where no collision happens as the agent drives in a conservative manner to comply with the safety criteria. As discussed earlier, raw game pixels and deep convolutional neural networks (CNN) may contribute to this. The network architecture we used is illustrated in Table 4.1.

Table 4.1 : The CNN architecture: Input: 60×60 image, Output: three Q-values.

Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	$84 \times 84 \times 3$	8×8	4	32	ReLU	$20 \times 20 \times 32$
conv2	$20 \times 20 \times 32$	4×4	2	64	ReLU	$9 \times 9 \times 64$
conv3	$9 \times 9 \times 64$	3×3	1	64	ReLU	$7 \times 7 \times 64$
fc4	$7 \times 7 \times 64$				ReLU	512
fc5	512				Linear	3

We run our algorithm on a PC equipped with CPU: Intel Core i7 3.2GHz and GPU: GeForce GTX 1080 Ti. The model trained with CNN, along with its weights, has been saved for every 1000 training steps that gave us the opportunities to select the ideal model just before the model overfitting. After training for 100000 frames, which took almost 23 hours, the accuracy of the system (Equation 3.2) is plotted in Figure 4.3.

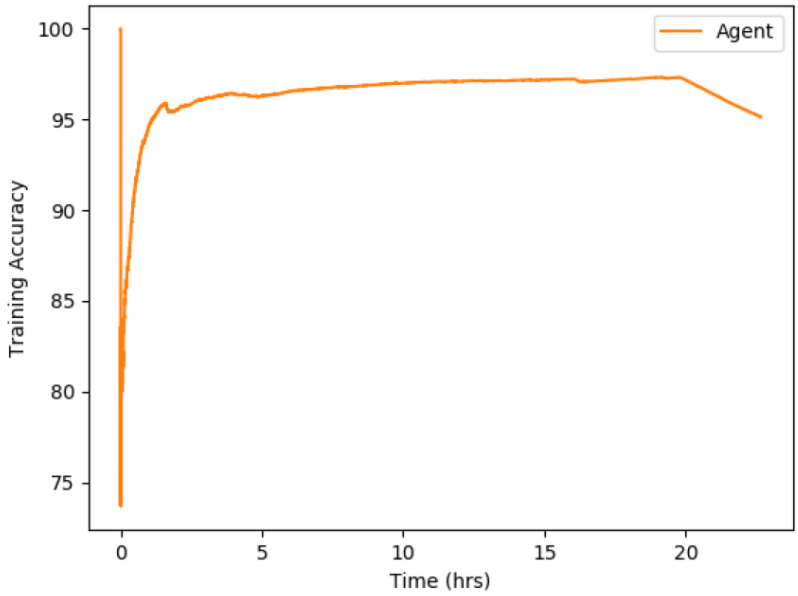


Figure 4.3 : The accuracy (Equation 3.2) of DQN agent was trained for 100'000 frames.

Obviously, after $t = 18hrs$, the overfitting happened. This has reduced the precision of the agent. Therefore, we select the model that has been saved before that time. By testing for 10000 frames, the DQN agent drove the car autonomously and achieved 100% accuracy. This means that after 10000 frames, no accident has happened. This is a great success comparing with the Q-learning technique in which the maximum accuracy used to be approximately 98%.

This is not the end of this problem because there are a bunch of application parameters that can be manipulated and make the application harder to play. For instance, it is possible to challenge the DQN agent in the DeepCars environment with more than three lanes and with more severe traffic congestion. It is obvious that the performance of the agent will decrease, and we need to seek new solutions.

The most popular extensions for DQN that can increase the capability of this agent can be the Deep Deterministic Policy Gradient (DDPG) technique [36], in which, instead of discretizing the action space, continuous actions are considered. Another successful variation of DQN is Double Q-learning (DDQN) [37], in which two CNN networks are used in the training section. One is being updated every single step, and the other one gets updated periodically (e.g., once 100 frames). The second network is used

to calculate the action value of the previous state, which removes the maximization function of the DQN. This has shown better results in the stability point of view of the problem.

5. CONCLUSION & FUTURE WORK

5.1 Conclusion

In this thesis, we approached the decision-making problem on the highway driving task for self-driving cars. We have shown that the proposed methods in the literature approached the problem using classical optimization and control strategies, where a mathematical description of the ego vehicle's surrounding environment is provided. Then, using a rule-based strategy, high-level actions have been generated to navigate through the traffic. We have shown that these methodologies rely on the accuracy of the measurements and the robustness of the mathematical descriptions.

Following the recent successful applications of the Reinforcement Learning (RL) methods, we have decided to approach the high-level decision-making problem in autonomous driving using this subfamily of artificial intelligence algorithms. Initially, we designed a simulation environment for the decision-making on the highway-driving task called DeepCars. We approached the problem using the classical model-free Q-table-based Q-learning approach. At each time-step, the agent receives the distance to the closest vehicle at each lane along with the lane number of the ego vehicle and generates high-level actions, i.e., ChangeLaneRight, ChangeLaneLeft, and StayOnLane. At each time-step, the algorithm updates the Q value for the experienced state-action pair on the Q-table using Bellman's recursive equation. Obviously, action values (Q values) stand for the expected future reward for each state-action pair. After running the simulation for a finite number of steps, the Q-table values converge to steady states. During the test phase, the algorithm leverages the recorded Q-table to search for the observed state's row and selects the action with the maximum Q value. We have shown that using this approach; the agent learns a greedy policy to avoid collisions with other vehicles; however, after running for 20000 frames, we have experienced a considerable amount of failures.

We noticed that to improve the performance of the reinforcement learning algorithm on generating reliable decisions, a reliable function approximator for the Q value estimation is required. This improves the agent's performance for the unseen scenarios. Recently, deep neural networks have shown promising results in classification and regression tasks, as well as function approximation. Incorporating the neural network into the RL to approximate the Q values is called deep reinforcement learning. We have implemented the Deep Q-Network (DQN) algorithm with the experience replay memory to apply to our problem in the DeepCars. Initially, we used a simple dense multi-layer perceptron architecture as the function approximator in DQN and trained the agent. During the test time, we disabled the network's weights' update using backpropagation and evaluated the trained model. We have shown a huge improvement in comparison with the rule-based approach, and the classical Q-table based Q-learning method. This was to be expected since DQN not only improved the Q value estimation of the agent for the unseen scenarios, but it also enabled the agent to take random actions using the epsilon-greedy method to explore the environment as well as exploiting the learned policies.

To improve the agent's performance further, we have performed a greedy-based hyperparameter optimization for the neural network architecture and parameters. We have shown the impressive effect of the hyperparameter values on the agent's performance. However, the agent was still making a few collisions after running for a considerable number of steps. Finally, to boost the deep reinforcement learning performance on generating the high-level actions on DeepCars, we have implemented the convolutional neural network (CNN) as the feature extractor, the backbone of the function approximator. This boosted the agent's performance since the trained agent did not make a single collision during the test phase making the algorithm achieve 100% accuracy for the test scenario.

In summary, we have shown that for high-level decision making on autonomous driving, the classical methods perform well in terms of safety; however, they lack the intelligence to generate optimal decisions. Using a deep reinforcement learning algorithm with a CNN architecture in the feature extraction, we trained an agent

that learned the optimal policy to drive in a safe and intelligent way by avoiding the collision in dense traffic and achieve the goal point within the smallest number of steps.

5.2 Future Work

In the future, we would like to extend this work to use a high-fidelity simulation that includes dynamic cars, realistic graphics, and better scenario generation. Also, we would like to use RL as high-level decision making and use low-level controllers to perform the actions since the low-level controller is good enough to remain in place.

REFERENCES

[1] **Vishnukumar, H.J.**, **Butting, B.**, **Müller, C. and Sax, E.** (2017). Machine learning and deep neural network—Artificial intelligence core for lab and real-world test and validation for ADAS and autonomous vehicles: AI for efficient and quality test and validation, *2017 Intelligent Systems Conference (IntelliSys)*, IEEE, pp.714–721.

[2] **Fagnant, D.J. and Kockelman, K.** (2015). Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations, *Transportation Research Part A: Policy and Practice*, *77*, 167–181.

[3] **Schofield, K.**, (2008), Automotive lane change aid, uS Patent 7,355,524.

[4] **Milanés, V.**, **Shladover, S.E.**, **Spring, J.**, **Nowakowski, C.**, **Kawazoe, H. and Nakamura, M.** (2013). Cooperative adaptive cruise control in real traffic situations, *IEEE Transactions on intelligent transportation systems*, *15*(1), 296–305.

[5] **Miller, R.H. and Tascillo, A.L.**, (2005), Blind spot warning system for an automotive vehicle, uS Patent 6,859,148.

[6] **Eneh, I. and Okafor, P.** (2014). Design of an automatic brake control system using artificial neural network, *International Journal of Scientific and Engineering Research*, *5*(4), 1239–1245.

[7] **Sentinel, M.** (1926). Phantom Auto'will tour city, *Milwaukee Sentin*, *4*.

[8] **Barradas, S.**, (2013), The Real Cost of Trucking—Per Mile Operating Cost of a Commercial Truck. The Trucker's Report, 2013.

[9] **Chen, C.**, **Seff, A.**, **Kornhauser, A. and Xiao, J.** (2015). Deepdriving: Learning affordance for direct perception in autonomous driving, *Proceedings of the IEEE International Conference on Computer Vision*, pp.2722–2730.

[10] **Moghadam, M. and Elkaim, G.H.** (2019). A hierarchical architecture for sequential decision-making in autonomous driving using deep reinforcement learning, *arXiv preprint arXiv:1906.08464*.

[11] **Phillips, C.L. and Habor, R.D.** (1995). *Feedback control systems*, Simon & Schuster, Inc.

[12] **Stefani, R.T.** (1993). *Design of feedback control systems*, Oxford University Press, Inc.

[13] **Falcone, P.**, **Borrelli, F.**, **Asgari, J.**, **Tseng, H.E. and Hrovat, D.** (2007). Predictive active steering control for autonomous vehicle systems, *IEEE Transactions on control systems technology*, *15*(3), 566–580.

[14] **Broggi, A.**, **Bertozzi, M.**, **Fascioli, A.**, **Bianco, C.G.L. and Piazzi, A.** (1999). The ARGO autonomous vehicle's vision and control systems, *International Journal of Intelligent Control and Systems*, *3*(4), 409–441.

[15] **Russell, S. and Norvig, P.** (2002). Artificial intelligence: a modern approach.

[16] **Krizhevsky, A.**, **Sutskever, I. and Hinton, G.E.** (2012). Imagenet classification with deep convolutional neural networks, *Advances in neural information processing systems*, pp.1097–1105.

[17] **Specht, D.F.** *et al.* (1991). A general regression neural network, *IEEE transactions on neural networks*, *2*(6), 568–576.

[18] **Sutton, R.S.**, **Barto, A.G.** *et al.* (1998). *Introduction to reinforcement learning*, volume135, MIT press Cambridge.

[19] **Mnih, V.**, **Kavukcuoglu, K.**, **Silver, D.**, **Graves, A.**, **Antonoglou, I.**, **Wierstra, D. and Riedmiller, M.** (2013). Playing atari with deep reinforcement learning, *arXiv preprint arXiv:1312.5602*.

[20] **Lang, A.H.**, **Vora, S.**, **Caesar, H.**, **Zhou, L.**, **Yang, J. and Beijbom, O.** (2019). Pointpillars: Fast encoders for object detection from point clouds, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.12697–12705.

[21] **Yan, Y.**, **Mao, Y. and Li, B.** (2018). Second: Sparsely embedded convolutional detection, *Sensors*, *18*(10), 3337.

[22] **Bojarski, M.**, **Del Testa, D.**, **Dworakowski, D.**, **Firner, B.**, **Flepp, B.**, **Goyal, P.**, **Jackel, L.D.**, **Monfort, M.**, **Muller, U.**, **Zhang, J.** *et al.* (2016). End to end learning for self-driving cars, *arXiv preprint arXiv:1604.07316*.

[23] **Goodfellow, I.**, **Bengio, Y. and Courville, A.** (2016). *Deep learning*, MIT press.

[24] **Koutník, J.**, **Cuccu, G.**, **Schmidhuber, J. and Gomez, F.** (2013). Evolving large-scale neural networks for vision-based reinforcement learning, *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp.1061–1068.

[25] **Sallab, A.E.**, **Abdou, M.**, **Perot, E. and Yogamani, S.** (2016). End-to-end deep reinforcement learning for lane keeping assist, *arXiv preprint arXiv:1612.04340*.

[26] **Aly, M.** (2008). Real time detection of lane markers in urban streets, *2008 IEEE Intelligent Vehicles Symposium*, IEEE, pp.7–12.

[27] **Perot, E.**, **Jaritz, M.**, **Toromanoff, M. and De Charette, R.** (2017). End-to-end driving in a realistic racing game with deep reinforcement learning, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp.3–4.

[28] **Taylor, C.J.**, **Košecká, J.**, **Blasi, R. and Malik, J.** (1999). A comparative study of vision-based lateral control strategies for autonomous highway driving, *The International Journal of Robotics Research*, *18*(5), 442–453.

[29] **Hatipoglu, C.**, **Ozguner, U. and Redmill, K.A.** (2003). Automated lane change controller design, *IEEE transactions on intelligent transportation systems*, *4*(1), 13–22.

[30] **Bicer, Y.**, **Moghadam, M.**, **Sahin, C.**, **Eroglu, B. and Üre, N.K.** (2019). Vision-based uav guidance for autonomous landing with deep neural networks, *AIAA Scitech 2019 Forum*, p.0140.

[31] **Bicer, Y.**, **Alizadeh, A.**, **Ure, N.K.**, **Erdogan, A. and Kizilirmak, O.** (2019). Sample efficient interactive end-to-end deep learning for self-driving cars with selective multi-class safe dataset aggregation, *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp.2629–2634.

[32] **Sallab, A.E.**, **Abdou, M.**, **Perot, E. and Yogamani, S.** (2017). Deep reinforcement learning framework for autonomous driving, *Electronic Imaging*, *2017*(19), 70–76.

[33] **Wang, J.**, **Zhang, Q.**, **Zhao, D. and Chen, Y.** (2019). Lane change decision-making through deep reinforcement learning with rule-based constraints, *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp.1–6.

[34] **Alizadeh, A.**, **Moghadam, M.**, **Bicer, Y.**, **Ure, N.K.**, **Yavas, U. and Kurtulus, C.** (2019). Automated Lane Change Decision Making using Deep Reinforcement Learning in Dynamic and Uncertain Highway Environment, *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, pp.1399–1404.

[35] **Mnih, V.**, **Kavukcuoglu, K.**, **Silver, D.**, **Rusu, A.A.**, **Veness, J.**, **Bellemare, M.G.**, **Graves, A.**, **Riedmiller, M.**, **Fidjeland, A.K.**, **Ostrovski, G.** *et al.* (2015). Human-level control through deep reinforcement learning, *Nature*, *518*(7540), 529–533.

[36] **Lillicrap, T.P.**, **Hunt, J.J.**, **Pritzel, A.**, **Heess, N.**, **Erez, T.**, **Tassa, Y.**, **Silver, D. and Wierstra, D.** (2015). Continuous control with deep reinforcement learning, *arXiv preprint arXiv:1509.02971*.

[37] **Van Hasselt, H.**, **Guez, A. and Silver, D.** (2016). Deep reinforcement learning with double q-learning, *Thirtieth AAAI conference on artificial intelligence*.

[38] **Mnih, V.**, **Badia, A.P.**, **Mirza, M.**, **Graves, A.**, **Lillicrap, T.**, **Harley, T.**, **Silver, D. and Kavukcuoglu, K.** (2016). Asynchronous methods for deep reinforcement learning, *International conference on machine learning*, pp.1928–1937.

[39] **Busoniu, L.**, **Babuska, R.**, **De Schutter, B. and Ernst, D.**, Reinforcement Learning and Dynamic Programming Using Function Approximators. 2010.

[40] **Busoniu, L.**, **Babuska, R.**, **De Schutter, B. and Ernst, D.** (2010). *Reinforcement learning and dynamic programming using function approximators*, volume 39, CRC press.

[41] **Watkins, C.J.C.H.** (1989). Learning from delayed rewards.

[42] **Watkins, C.J. and Dayan, P.** (1992). Q-learning, *Machine learning*, *8*(3-4), 279–292.

[43] **Logothetis, N.K.** (2007). The ins and outs of fMRI signals, *Nature neuroscience*, *10*(10), 1230–1232.

[44] **Krizhevsky, A. and Hinton, G.** (2010). Convolutional deep belief networks on cifar-10, *Unpublished manuscript*, *40*(7), 1–9.

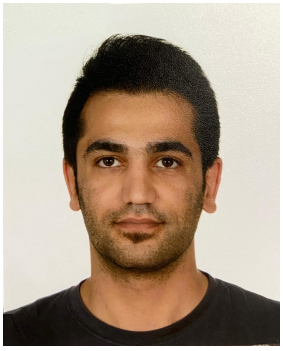
[45] **Krizhevsky, A.**, **Sutskever, I. and Hinton, G.E.**, (2012). Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems, pp.1097–1105.

[46] **Driving, A.** (2014). Levels of driving automation are defined in new SAE international standard J3016: 2014, *SAE International: Warrendale, PA, USA*.

[47] **Url-1**, `<http://pyglet.org>,date2019`.

[48] **LeCun, Y.**, **Bengio, Y. and Hinton, G.** (2015). Deep learning. nature 521.

CURRICULUM VITAE



Name Surname : Mehdi Nasiri

Place and Date of Birth : Iran, Khoy - August 01, 1989

E-Mail : nasiri17@itu.edu.tr

EDUCATION :

- **B.Sc.** : 2012, Azarbayjan Shahid Madani University Electrical Engineering
- **M.Sc.** : 2020, Istanbul Technical University, Faculty of Electrical and Electronics

PROFESSIONAL EXPERIENCE AND REWARDS:

- Won 2nd prize among more than 25 groups of Robotic team, 2006
- Car robot with combination of Augmented Reality to communicate and teach children