الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

BADJI MOKHTAR ANNABA-UNIVERSITY
Faculty of Technology
Department of Computer Science

جامعة باجي مختار – عنابة
كلية التكنولوجيا
قسم الاعلام الآلي

Domain: Mathematics and Computer Science
Branch: Computer Science
Specialty: Embedded Computing

Decision Making In Autonomous Systems

A Dissertation Submitted to the Department of Computer Science in Partial Fulfillment of the Requirements for the LMD Doctorate Degree in Computer Science.

Submitted by
Rais Med Saber

Supervised by
Pr. Boudour Rachid

Co-Supervised by
Dr. Bougueroua Lamine

Academic Year: 2022/2023

Abstract

The appearance of Machine Learning and Deep Learning techniques under the Artificial Intelligence field provided new potentials in various fields of automation. One of the most anticipated technologies the world is waiting for its full appearance which relies on these techniques is Autonomous Vehicles. This technology has known multiple transformations over the last 100 years, till reaching high levels of autonomy nowadays. Though, this technology is very complicated and has a lot of limitations that researchers need to overcome before its full lunch.

Machine Learning combined with deep learning provided an acceptable solution when it comes to decision making, planning, and learning. Where many algorithms and approaches have been developed to upgrade these aspects.

Deep Reinforcement Learning with its variety of methods is probably the most promising family that can provide machines with a human-level ability to learn or an even superior one.

Two of the highly used Deep Reinforcement Learning methods are Deep Q Learning and Deep SARSA. In this thesis, we investigated the effectiveness of these two methods and proposed a novel extension for each algorithm that surpassed the original's performance.

In the first method, we integrated Deep SARSA with the music-based metaheuristic optimization algorithm dubbed "Harmony Search Algorithm" for a parametrable extension that can be extended to various situations, while integrating a novel policy that takes decisions based on pre-registered consecutive actions.

In the second method, we extended Deep Q Learning algorithm and proposed "Alternative Bidirectional Q Learning", a method that has a novel policy for both actions selection and artificial neural models fitting process. Both policies are based on three consecutive data related to each real-time state the agents are in.

The validation of these two extensions was done using various simulation environments dedicated for testing and developing Deep Reinforcement Learning methods, and the results showed the effectiveness of the proposed methods in handling decision making for autonomous vehicles.

Keywords: Decision Making, Autonomous Systems, Autonomous Vehicles, Machine Learning, Deep Learning, Reinforcement Learning.

ملخص

قدم ظهور التعلم الآلي وتقنيات التعلم العميق في مجال الذكاء الاصطناعي إمكانات جديدة في مجالات التحكم الآلي. واحدة من أكثر التقنيات المنتظرة في العالم هي المركبات الذاتية. وقد عرفت هذه التكنولوجيا تحولات متعددة على مدى السنوات المئة الماضية ، حتى الوصول إلى مستوى عال من التحكم الذاتي الآن. على الرغم من أن هذه التكنولوجيا معقدة للغاية ولديها الكثير من القيود التي يحتاج الباحثون إلى التغلب عليها. يوفر التعلم الآلي جنبا إلى جنب مع التعلم العميق حلا مقبولا عندما يتعلق الأمر بالتخطيط والتعلم الذاتي. حيث تم تطوير العديد من الخوارزميات لرفع مستوى هذه الجوانب. التعلم المعزز العميق هو على الارجح الأسرة الواعدة التي يمكن أن توفر للآلات قدرة على التعلم مساوية أو تتعدى قدرة البشر. في هذه الأطروحة ، بحثنا في فعالية طريقتين من أكثر أساليب التعلم المعزز العميق المستخدمة واقترحنا امتدادا جديدا لكل خوارزمية لتجاوز الأداء الأصلي. في الطريقة الأولى ، قمنا بدمج ديب سارسا مع خوارزمية التحسين المستندة على الموسيقى المسماة هارموني سيرش من أجل اعطاء الخوارزمية الجديدة طبيعة قابلة للتغير حسب المحيط، مع دمج سياسة جديدة تتخذ قرارات بناء على إجراءات متتالية مسجلة مسبقا. في الطريقة الثانية ، قمنا بتوسيع خوارزمية ديب ك ليرنين واقترحنا امتدادا لديه سياسة جديدة لكل من اختيار الإجراءات وتدريب النماذج العصبية الاصطناعية. تستند كلتا السياستين إلى ثلاث بيانات متتالية تتعلق بكل حالة للوكلاء في الوقت الفعلي. تم التحقق من قدرة هذين الامتدادين باستخدام بيئات محاكاة مختلفة مخصصة لاختبار وتطوير طرق التعلم المعزز العميق ، وأظهرت النتائج فعالية الطرق المقترحة في التعامل مع صنع القرار للمركبات ذاتية القيادة.

الكلمات المفتاحية: صنع القرار ، الأنظمة المستقلة ، المركبات الذاتية ، التعلم الآلي ، التعلم العميق ، التعلم المعزز.

Résumé

L'apparition des techniques d'apprentissage automatique et d'apprentissage profond dans le domaine de l'intelligence artificielle a ouvert de nouvelles perspectives dans divers domaines de l'automatisation. L'une des technologies les plus attendues au monde, qui est basée sur ces techniques, est celle des véhicules autonomes. Cette technologie a connu de multiples transformations au cours des 100 dernières années, pour atteindre aujourd'hui un haut niveau d'autonomie. Cependant, cette technologie est très compliquée et présente de nombreuses limitations que les chercheurs doivent surmonter avant son épanouissement. L'apprentissage automatique combiné à l'apprentissage profond a fourni une solution en matière de prise de décision, de planification et d'apprentissage. De nombreux algorithmes et approches ont été développés pour améliorer ces aspects. L'apprentissage par renforcement profond, avec sa variété de méthodes, est probablement la famille la plus prometteuse qui peut fournir aux machines une capacité d'apprentissage de niveau humain, voire les dépasser. Deux des méthodes d'apprentissage par renforcement profond les plus utilisées sont Deep Q Learning et Deep SARSA. Dans cette thèse, nous avons étudié l'efficacité de ces deux méthodes et proposé une nouvelle extension pour chaque algorithme qui a dépassé les performances originales. Dans la première méthode, nous avons intégré Deep SARSA à l'algorithme d'optimisation méta-heuristique basé sur la musique appelé "Harmony Search Algorithm" pour une extension paramétrable qui peut être étendue à diverses situations, tout en intégrant une nouvelle politique qui prend des décisions basées sur des actions consécutives préenregistrées. Dans la deuxième méthode, nous avons étendu l'algorithme Deep Q Learning et proposé "Alternative Bidirectional Q Learning", une méthode qui a une nouvelle politique pour la sélection des actions et le processus d'ajustement des réseaux de neurones artificiels. Les deux politiques sont basées sur trois données consécutives liées à chaque état en temps réel dans lequel se trouvent les agents. La validation de ces deux extensions a été effectuée à l'aide de divers environnements de simulation dédiés au test et au développement des méthodes d'apprentissage par renforcement profond, et les résultats ont montré l'efficacité des méthodes proposées dans la prise de décision pour les véhicules autonomes.

Mots-clés: Prise de décision, systèmes autonomes, véhicules autonomes, apprentissage automatique, apprentissage profond, apprentissage par renforcement.

Acknowledgements

Praise be to Allah, for all his blesses, for his guidance, and for letting this work be done in the best possible way and reach the end of a road to start a new one with more successes inshallah.

To mom and dad, for all their support and prayers to see me reach the top places and get the highest degrees, may god protect them.

To my brothers, for all their help, for the laughs, for the joy, I am grateful to have them, may god grant them all great things in life.

Many thanks to my supervisor Pr Boudour Rachid, with his planning, his thoughts and ideas, and his strategies this work was done in a smooth way.

I would like to thank as well my co-supervisor Dr Bougueroua Lamine for granting me the opportunity to have an internship at EFREI school in paris, which was a great new experience for me filled with new discoveries.

I appreciate as well the efforts made by the reviewers and the time dedicated to read and review this thesis.

And finally, to my beloved partner, without her nothing from this work could ever be done and no light could ever be seen from the end of the tunnel that leads me to take the PhD degree. She fought hardly with me in many ways just to see my success, and I'm so grateful for this bless that god gave me and I cannot wait for us to reach higher and higher places together. I'm blessed with you my dear.

Contents

List of Figures

List of Tables

Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
AV	**A**utonomous **V**ehicle
CPU	**C**entral **P**rocessing **U**nit
DARPA	**D**efense **A**dvanced **R**esearch **P**rojects **A**gency
DL	**D**eep **L**earning
DQN	**D**eep **Q** **L**earning
DRL	**D**eep **R**einforcement **L**earning
DS	**D**eep **S**ARSA
GDP	**G**ross **D**omestic **P**roduct
GM	**G**eneral **M**otors
HM	**H**armony **M**emory
HMC	**H**armony **M**emory **C**onsidering
HMCR	**H**armony **M**emory **C**onsidering **R**ate
IEEE	**I**nstitute **E**lectrical **E**lectronics **E**ngineers
MCTS	**M**onte **C**arlo **T**ree **S**earch
MDP	**M**arkov **D**ecision **P**rocess
MSE	**M**ean **S**quare **E**rror
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
QN	**Q** **L**earning
RL	**R**einforcement **L**earning
SARSA	**S**tate **A**ction **R**eward **S**tate **A**ction
TD	**T**emporal **D**ifference

Dedication

For my dear partner ...

For my precious parents ...

For my cherished brothers ...

For my friends ...

For all the family ...

And For myself ...

Chapter 1

Introduction

"Once you trust a self-driving car
with your life,
you pretty much will trust
Artificial Intelligence with anything."

Dave Waters

Artificial Intelligence and mainly Machine Learning grew up as an interesting research subject in last decades with the goal of having machines mimic human beings in multiple tasks, in order to reach, at the final stage of development, what we call Autonomous Systems that can provide more safety, accuracy, productivity by replacing humans in such tasks in numerous fields of application. [1].

But Machine learning as a standalone concept had its limits [2], therefore, researchers attempted to integrate multiple approaches for enhancement purposes due to many difficulties encountered, including complexity, stability, cost effectiveness, etc [3].

The main promising approaches that were used in the hybridisation process with machine learning methods included neural networks [4], which were inspired by the human brain's reverse engineering process of gathering information, classifying it, and then finding the most important parts of these information to use it in future tasks[5].

The neural networks got largely known under the deep learning concept [6].

This concept revolves in general around training a number of hidden layers to increase the performance quality in time [7].

Till nowadays, deep learning is growing dramatically in both usability and popularity, with thousands of researchers studying it, upgrading its concepts, and using it in tackling various challenges in multiple fields.

Latest researches proved the efficiency of novel and hybrid machine learning techniques in handling the autonomy part of machines and robots [8], whether it is partial, nearly full or full control.

While the subjects of studies included the most popular field in the autonomy's industry which is Autonomous Vehicles field.

This one requires highly sophisticated methods to handle its hardest section known as the decision making sub-system, and the most promising machine learning family according to literature to handle decision making for autonomous vehicles and most of the autonomous systems is Reinforcement Learning [9, 10].

The main objectives of this thesis are: investigating the capacity of reinforcement learning in handling decision making in autonomous cars, how deep learning influenced this family of methods, and how it is possible to upgrade these methods to have better performance, stability, learning capability, and autonomy.

1.1 Motivations:

In this thesis, we focused on exploring the ability of deep learning combined with reinforcement learning methods to handle the control of autonomous agents, and their applications in the decision making field for self-driving vehicles, these methods are known today as Deep Reinforcement Learning models (DRL) [11].

Deep reinforcement learning methods are of high importance in artificial intelligence field in general, and specifically in enhancing the learning capability of machines, where many researchers stated that in order to solve the real artificial intelligence problems, it is needed to adopt these methods because of many factors that affect the learning process, related mostly to the nature of the surroundings and how can machines intercept it, represent it, and use it in getting new knowledge continuously [12].

At first, studies being made on simple tasks, where the tabular nature of reinforcement learning alone without combining it with deep learning models as mentioned before was enough to represent the treated environments [13].

But, due to the problem of dimensions and the increase in the treated environments' complexity, deep learning was integrated to reinforcement learning and this allowed to address more challenges and more fields of research including autonomous vehicles systems that represent one of the biggest study fields of AI in this decade [14].

Autonomous vehicles major advantages is what made it an interesting topic to tackle in many studies including this thesis, and also the fact that the world is still far from reaching a truly autonomous car whether we talk about the hardware part or the software part with all its sub-systems.

Still, any upgrade in any side of self-driving cars will be counted as an another step towards reaching the world of autonomy in roads (Figure 1.1) [15].

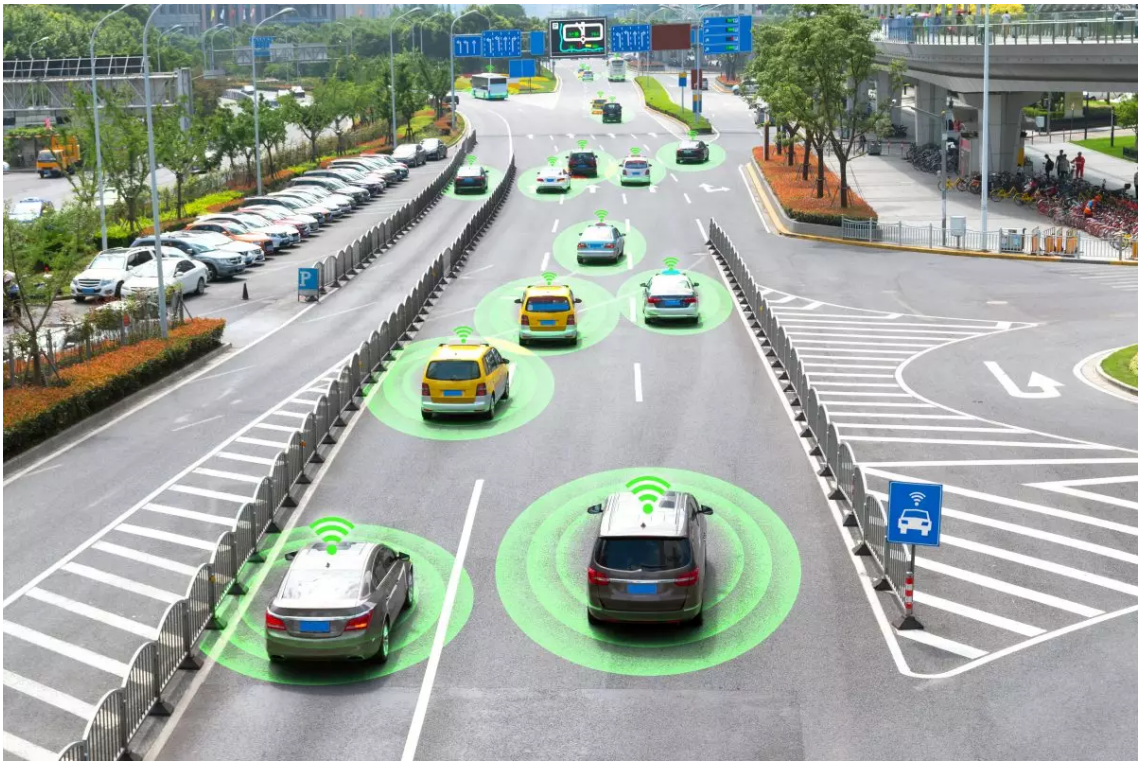


FIGURE 1.1: Driver-less Roads

Therefore, we focused mainly as mentioned before in upgrading the decision making sub-system which demands highly adapted methods to multiple aspects, and the most promising methods are the deep reinforcement learning algorithms.

DRL field of research is based on simulation, because it is not possible to keep testing over and over again even the simplest autonomous systems in real environments, hence, simulators are used to replicate real world scenarios and this is a real advantage for this topic because it facilitates the production and test procedures.

A popular study case that shows the advantages of DRL and the use of simulation to reach and surpass human level is Google deep mind group's paper, where they used a group of Atari games as the test environments, and this shows that even games represent a good environment to develop high level methods (Figure 1.2) [11].

"Play is the highest form of learning".Albert Einstein.

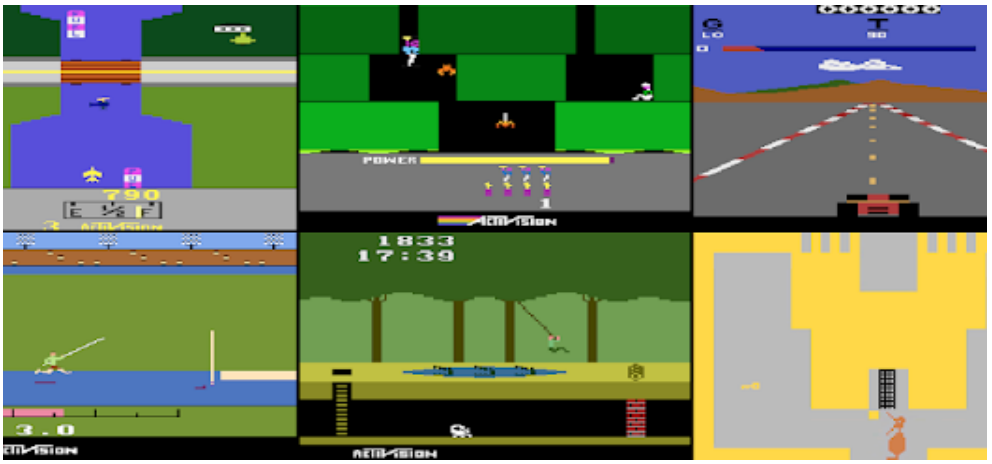


FIGURE 1.2: Examples of Atari Games Environments

1.2 Autonomous Vehicles:

Till recently, vehicles were solely made for humans to drive, and the idea of adding even a small portion of autonomy to these systems was just words on papers or scenes in movies or stories.

But with the advancements in computing power, sensors, data processing, and software, we started seeing levels of autonomy added to various kinds of vehicles and it is no longer just an imagination to reach the level of transportation where humans' control is not required anymore, but at the current moment we are still far from achieving it due to many limitations.

Although, by reaching the highest level of this technology, massive effects are expected to show up on the current transportation system and many benefits to humans and society in general will appear.

The main benefit and objective of reaching a high level of autonomy is to save humans lives.
According to the United States' National Safety Council with a 10.5% increase in the year of 2021 from 2020, approximately 43 000 people died on the roads of the U.S..
This number alone represents the bad situation of today's transportation systems without counting the heavily injured people and the damage made to nature and machines (Figure 1.3)[16].

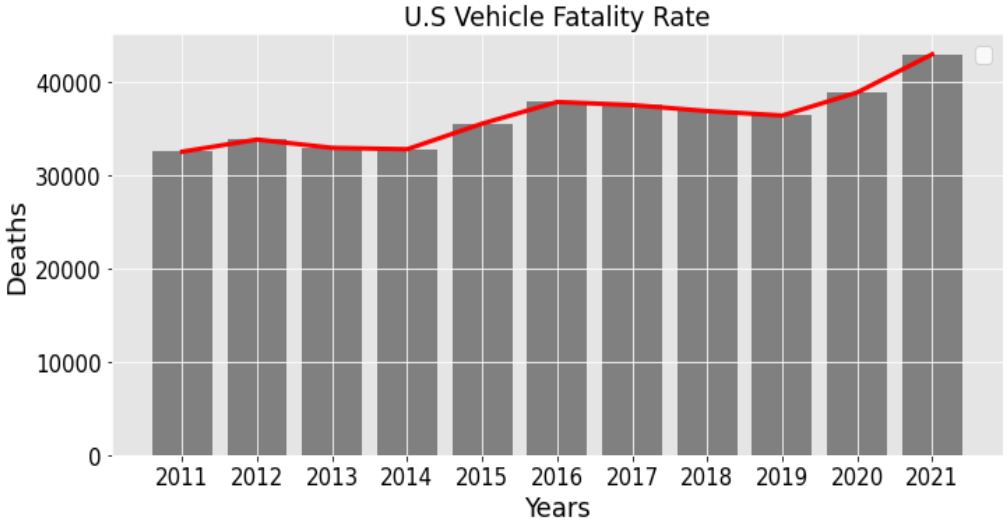


FIGURE 1.3: Roads Death Records of U.S

This is not limited to just vehicle casualties but adding a level of autonomy in other dangerous work fields can help us reduce the danger that the workers face each day.

During last decade, noticeable progress was made towards reaching the fully autonomous cars, whether in perception systems that are responsible of gathering data or in low-level control systems that do not require heavily complex theory methods to handle, although high-level decision making is still yet a field that is not as much as explored as previous fields and requires more advancements due to the complexity of road situations.

1.2.1 From History To Present:

An introduction to the history of autonomous vehicles is presented in this section to give the reader an overall look on how and when the idea of autonomy started, and why it represented an interesting idea to the world since long.

1.2.1.1 A Moment In History:

The idea of vehicles moving autonomously started even before vehicles became a real invention, where history takes us back to 1478 when Leonardo da Vinci

once lived and created his self-propelled cart that can move alone in an already programmed road (Figure 1.4).

Even when it doesn't include lot of mechanisms to be called a high technology invention, but around that time it was one of the biggest things that the humans set their eyes on, opened their minds to new possibilities, and raised thoughts to a different level [17].

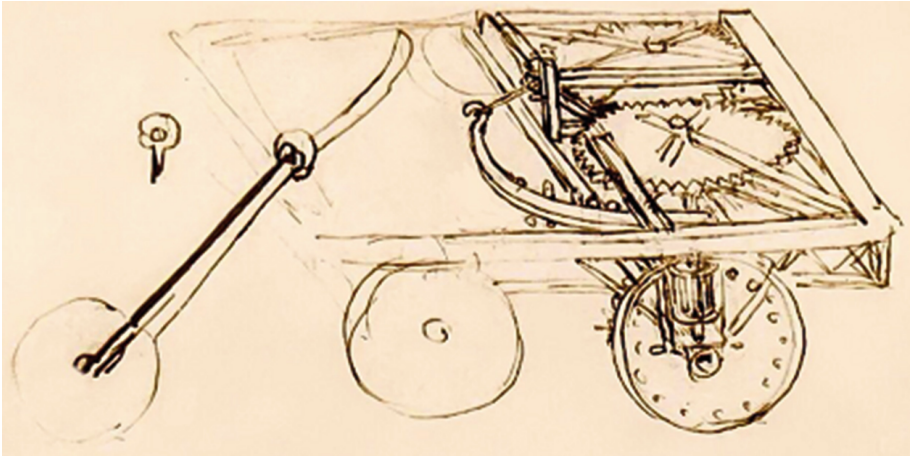


FIGURE 1.4: Da Vinci Self-Propelled-Vehicle.

In the 18th century, Kant defined autonomy in a way that inventors and researchers can take as the base for their work to be achieved in a correct and uniform manner and it goes like this: "a self-determination within a super-ordinate (moral) law".

According to this definition, the vehicle must behave like a human being in a way that it can replicate some of the humans emotions but more rationally.

In other words, all society's moral laws must be predefined in the vehicle's system, and all decisions made by the vehicle must be based on these laws.

This was a goal that is still yet to be achieved, even when for the last 100 years researchers were always convinced that we are just 20 years away from achieving full autonomy in our cars.

Around 1935, an educational movie was launched, in which the US science fiction author David H. Keller tells the story of "the living machine", a car driven by voice, this movie was commissioned by general motors(GM) and produced by Jan Handy (1886-1983).

In 1938, the magazine "Popular Science" reported an auto transport system for the 1st time that uses a guide wire vision technology, a technology that remained in the light till the 70s.

And one year later, the first real demonstration of autonomous vehicles was at the New York's World Fair named Futurama by General Motors. The vision was that it wont take long till autonomous vehicles will navigate U.S. roads.

Although this vision was related to a lot of obstacles, where the main challenge was in order to apply this technology, large changes to the infrastructure must be applied.

But since the society needs to see autonomous vehicles in action to trust this new technology, and a sufficient market penetration for the technology needs to be achieved, large-scale infrastructure investments cannot be applied, therefore real-world applications were limited to few systems like Personal Rapid Transport(autonomous airport shuttles) and Automated People Mover (Metros).

A lesson researchers learned from those years is that in order to let autonomous vehicles reach the market, it must be designed to drive in the existing environment and handle its uncertainties [18].

1.2.1.2 A Wide Jump:

The idea of autonomous vehicles returned in 1977 precisely in Tsukuba Japan, when Sadayuki Tsugawa's team from the mechanical engineering lab was able to create the first visually guided AV which used two cameras to detect its surroundings. This was a huge step towards reaching the desired fully autonomous car.

Following the major development in computer and network technologies, driverless vehicles were the focus of the world when the U.S defense advanced research projects agency (same agency behind the start of the internet) launched DARPA challenges, a set of competitions for designing and building self-driving vehicles [17].
Three editions of DARPA were held, in 2004, 2005, and 2007.

In the first edition, the challenge consisted of a one hundred and fifty miles off-road course, the best vehicle was able to finish just 7 miles from the full road.

Although in the next edition, a major advancement was already achieved when five vehicles completed the full road, the fastest vehicle to finish the road was able to reach an average speed of 30 km/h.

Two years later in the last challenge called DARPA Urban Challenge (Figure 1.5), the road consisted of an urban area including traffic where cars must follow the traffic laws. Six cars were able to finish the full road, and it was the start of major advancements and a lot of researches all around the world [19].

FIGURE 1.5: Darpa Urban Challenge Start Line.

1.2.2 Categorization:

Multiple classifications for autonomous vehicles were made by different associations based on the level of humans' interference in driving vehicles.

The Federal Highway Research Institute BaSt defined automation in cars as a five level structure where level 1 represents driver only situation, level 2 is assisted driving, level 3 is partial automated, level 4 named highly automated, and finally level 5 which represents the fully automated vehicle [20].

The National Highway Traffic Safety Administration NHTSA had their own categorization that is close to BaSt's where they named five levels as follows: level 0 no automation, level 1 function specific automation, level 2 combined function, level 3 limited self-driving automation,and level 4 full self-driving automation [21].

But The most well-known and used till these days is the united states Society of Automotive Engineers (SAE International) classification. In which, six levels of automation were defined: level 0 no automation, level 1 driver assistance, level 2 partial automation, level 3 conditional automation, level 4 high automation, level 5 full automation.(with high order automation representing the last 3 levels) [22].

For a better clarification of these levels, Figure 1.6 below explains in more details the SAE categorization.

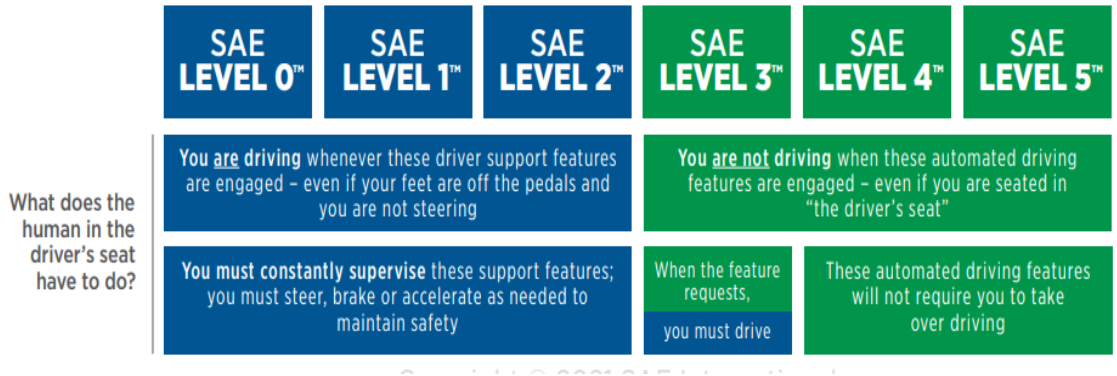


FIGURE 1.6: SAE Automation Levels in Self-Driving Cars.

1.2.3 Benefits of Automation in Vehicles:

Great potentials can be brought to the world by the self-driving vehicles technology [23].

Researchers aimed with this technology to solve multiple problems our roads hold [24], where the main benefit they want to achieve is the increased safety [17].

globally, every year over a million people are killed in road accidents, and according to studies in the united states 90% of these crashes are caused by human errors.

Therefore, since machines aren't affected by the deficiencies the humans have like tiredness, carelessness, hastiness, they can theoretically lower the number of roads casualties and deaths dramatically.

Another advantage is traffic flow optimization in highways or in cities.

since the 80s, developers gave this a major priority and created specific programs for this matter, like the European program of vehicle automation named " Program for an European traffic with highest efficiency and unprecedented safety (Prometheus, 1987 – 1994).

By optimizing the traffic flow we can save time, fuel, and reduce the major pressure of long trips and/or urgent trips when we need to reach a certain destination in the shortest time possible.

Autonomous vehicles can increase also the collective efficiency, where nations with average or low income suffer from 3% of Gross domestic product (GDP) in average because of vehicles accidents.

Moreover, replacing the current individually-owned vehicles that are powered with gas with autonomous cars can lead to creating efficient travel systems like a mobility-on-demand city architecture composed only of electrical cars.

Vehicles will travel to charging stations automatically, and the system will adjust vehicles locations according to the demands of clients in different areas and this will reduce the number of vehicles needed in the roads.

Also, removing the need for humans to drive will let people invest travelling time in other things even for just getting some rest, and will reduce the stress of interacting with other drivers in the roads.

Autonomy will grant more efficiency to heavy vehicles used for traveling goods by various companies, where trips scheduling will get easier to avoid rush hours and even traveling at night will be available at the same pace of the morning trips.

Finally, self-driving cars will grant disabled people, old people, and whoever suffers from a temporary or permanent injury the ability to travel wherever they want safely without needing a caretaker in most cases. A capable driver won't be needed as well and this will add flexibility to our trips.

1.2.4 Difficulties That Arise:

With every new revolution and evolution, new challenges appear [17], and in this section we will mention multiple walls the autonomous vehicles technology needs to break in order for it to be a trusted world wide used technology [25].

Interacting with cars with various levels of automation is one of the biggest if not probably the largest challenge autonomous vehicle will encounter, because it is not possible to go from no self-driving car running in streets to complete autonomy in roads.

Therefore, autonomous vehicles decision making systems must take into consideration the situations that can be caused by other autonomous vehicles and normal drivers.

This causes another dilemma, where autonomous vehicles must follow road rules flawlessly but in case of interacting with human drivers sometimes those rules must be ignored to prevent accidents.

The variety of roads and the uncertainties of each type must be considered during development.

Also, forecasting weather events precisely represents another challenge for researchers, where a reliable forecasting autonomous system must be developed and integrated with the decision making systems for self-driving cars.

Moreover, self-driving vehicles will still need to interact with humans like passengers, where a certain level of understanding people's emotions and state must be available to clearly understand the environment and prevent accidents.

Another thing related to this problem is ethics. Researchers are divided to whom who agree and who don't when it comes to adding the concept of ethics and emotions to vehicles.

Where ethics is defined according to IEEE as "treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national, region, sexual orientation, gender identity, gender expression".

For example, in case of a sure accident with one of two people, when applying the concept of ethics as defined by IEEE, the autonomous car shouldn't pick for example the older person to collide with just because the younger probably has more years to live.

This example alone shows that ethics are a big challenge not just to developers but to country-leaders to decide the rules that cover them where no 100% correct answer is available now and probably in the future as well.

From just the challenges mentioned above, we can see that many obstacles must be put into consideration while developing autonomous vehicles.

1.3 Decision-making for autonomous driving:

Three main levels of decision making for autonomous vehicles compose the full decision making sub-system, these levels are: strategic, tactical and operational level (Figure 1.7) [26, 27].

First level mainly for planning routes according to the desired destinations of the travelers, while the tactical level is responsible of updating the route decided by the strategic level depending on the perceived real-time state of traffic and environment.

Lastly the operational level converts the final route modified by the tactical level into a detailed trajectory that will be used to handle the vehicle's actuators.

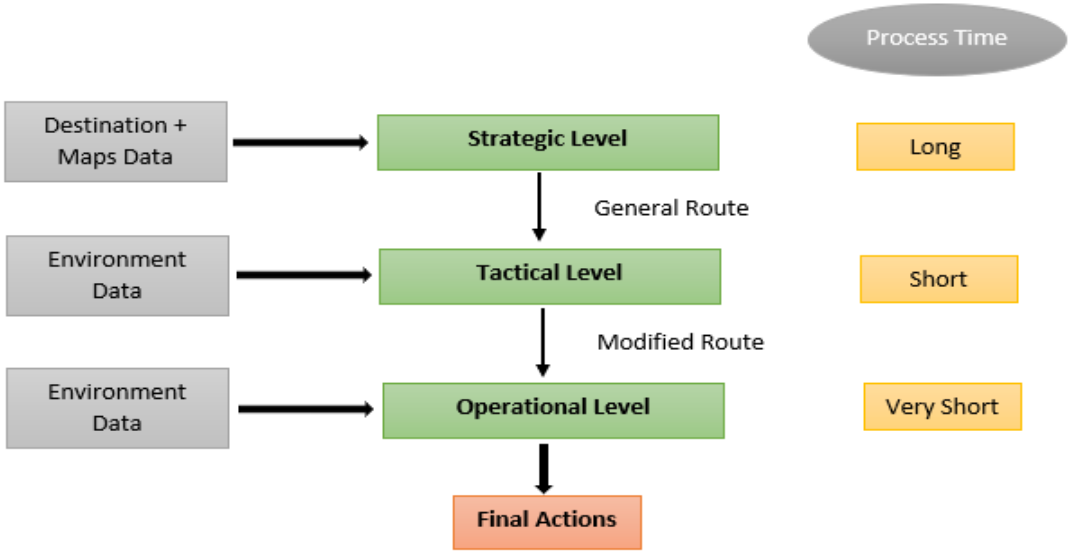


FIGURE 1.7: Decision Making Levels.

In this study, we mainly focused on the middle level tactical decision making or known as behavioral decision making.

This system must handle a wide range of environments from highways to urban areas, and be able to grant efficient decisions for any situation expected or unexpected while considering the interactions with other vehicles in roads.

Also, High level of uncertainty can appear in certain scenarios, therefore developing a perfectly reliable system is not an easy task if it is possible in the first place.

1.3.1 Approach:

Numerous approaches for behavioural decision making exist already, for a deep overview about these methods see next chapters.

Although, majority of these methods aren't suitable for the real world's complexity and just designed for limited scenarios.

Hybrid methods were one of the solutions that handle more complex scenarios but the cost and complexity of these methods can be really high.

Therefore, researchers must put into consideration while developing the new approaches that handle decision making for autonomous vehicles or any other autonomous system the complexity of both environments and models.

In this thesis, approaches that are based on learning are adopted.

These approaches have the potential to reach a generalized system that can handle multiple scenarios effectively.

The base of these methods is Reinforcement Learning approach which showed the best potential between all machine learning archetypes in handling decision making.

Therefore, the goal of this thesis is to investigate reinforcement learning methods and propose modifications to these methods in a way that can overcome their limitations and upgrade the decision making capability of autonomous agents.

1.3.2 Limitations:

Following points were not considered when we updated Reinforcement learning methods to tackle the decision making problem in this thesis:

• To grant safety using a learning based approach is not discussed in this thesis, in general another safety layer must be added to handle this side effectively.

• Experiments and all tests in this thesis were made using simulators. How an approach can be used in real world is out of this thesis' scope.

• Multi-agents control is not considered in this thesis as well.

• Straightforward environments' states formulations were used to better analyze the results obtained, however in real world scenarios more complex representations are needed.

1.4 Contributions:

The main contributions of this thesis are:

• Two novel deep reinforcement learning algorithms were proposed for tactical decision making, that provided enhanced performance.

• Enhanced stability during learning and achieving a better balance between exploration and exploitation processes that will be explained in upcoming chapters.

• An extension of the popular DQN and Deep SARSA algorithms that both allow for a better handling of various scenarios, and dealing with uncertain and unknown situations.

• Robust and cost-effective new approaches that don't require a complex neural network architecture.

1.5 Thesis outline:

Chapter two presents a background on related works to this thesis, followed by chapter three that includes a detailed introduction of Reinforcement Learning which is the main concept used in the proposed works. The fourth chapter presents the experiments being made and the novel propositions. And we finish afterwards with a conclusion to this work and some perspectives.

1.6 Chapter Conclusion:

In this chapter, we briefly introduced the concept of machine learning which represents the peak of today's Artificial Intelligence fields alongside Deep Learning and Neural Networks Concepts.

These concepts were the base of our contributions that are mainly focusing on the decision making field in autonomous vehicles.

After listing our motivations that resulted in focusing on this field, we gave an overall look on the history of autonomy and how it has evolved over the years.

In the last years, high level of autonomy was achieved in our vehicles and many categorizations for this matter were introduced that defined what point we reached in this field.

In this chapter as well, we mentioned many benefits that this technology (autonomous vehicles) can provide to the humans and to societies in general. But with every new technology many challenges arise that researchers must face in order for it to be fully successful.

Lastly, we introduced the approach that was the base of our contributions which is reinforcement learning that represents the most promising field of machine learning, while listing the different limitations that we faced as well.

Chapter 2

Background and Literature Review

"A computer would deserve
to be called intelligent
if it could deceive a human into believing
that it was a human."

Alan Turing

In this chapter, several concepts are introduced that are necessary for the reader to better understand the experiments introduced in later chapters.

However, in reality, autonomous agents are not systems that are easy to be deployed.

2.1 Autonomous Agents:

In this section we briefly explain the concept of agents which is important for readers to understand the fundamentals of this thesis.

According to [28], agents are defined as systems running in specific environments, these agents are capable of taking decisions autonomously in order to reach the desired goal of each environment.

These decisions are chosen depending on the state of the environments at each time-step t, where agents' success rate needs to be maximized in time by selecting the best possible action in each state after calculating the possibility of each action while depending on previous experiences and actions applied in similar previous states.

Autonomous agents are the most suitable in such situations.

Many approaches were considered and applied in the literature to create autonomous agents that can handle decision making for self-driving vehicles.

Below we mention the main methods used that are the base of our propositions.

2.2 Rule-based and planning-based methods:

Rule based methods was the first approach that was adopted to handle behavioral decision making in autonomous vehicles, and it represents a handcrafted set of states and actions that should cover all the possible scenarios of the environment treated [29].

In DARPA Urban challenge, the Carnegie Mellon University team won in this edition by adopting a rule-based approach, in which multiple modules were defined

for each driving scenario that can be encountered. Other contestants like Virginia and Stanford Tech adopted similar approaches too [30].

Even when this approach proved to be successful for events like the urban challenge where the environment was controlled and limited, the complexity of real-world driving scenarios won't allow for this approach to be used in such situations.

Another section of methods considers the autonomous decision making problem as a motion planning task, thus they use a model for predicting the motion of agents in the surrounding area, and after that the necessary action in every moment for the ego vehicle [31].

Because all the predictions and the self-driving car's plan are not related, this leads to a responsive behavior.

Hence, we don't consider the autonomous vehicle's interaction with similar agents explicitly, although through re-planning it can occur implicitly.

By creating a discrete state space, the search based planners select an algorithm from A* like Dijkstra's algorithm to be applied [32].

This type of methods was frequently used during DARPA Urban Challenge.

But the graph-search algorithms could not cover the real-time necessities, therefore, to handle motion planning sampling-based algorithms were used, for example rapidly-exploring random trees [33].

Another approach that was used to handle motion planning was optimization-based techniques.

In [34], authors addressed the optimal control for highway scenarios by proposing a semi-reactive trajectory generation method, that can be integrated into the behavioral layer.

In some approaches, a probabilistic prediction is used as input to the motion planner because the behavior of humans in most cases is complex and varies from one human to another.

In a study made by [35], the previous concept is shown where they tried to increase the safety while driving in intersections.

More of the motion planning approaches developed for driving autonomously are included in [36], and [37].

Usually, decision making problems are defined as either Markov Decision Processes (MDPs) [38], or Partially Observable Markov Decision Processes (POMDPs) depending on the nature of the environment that specifies the state spaces complexity and form [39].

By using MDP or POMDP we can model several uncertainties, for example: in current time-step t, in future changes of traffic and surroundings, in the interaction with other entities of the roads, etc.

Nonetheless, Finding an optimal policy for a problem illustrated as a POMDP is difficult and uncontrollable, hence numerous approximate methods have been developed to tackle this problem. The approximate methods were divided into online and offline methods (Figure 2.1).
The difference between both is the moment when agents perceive their data.
In offline algorithms, all data are known and introduced to the agent before execution, at the opposite of online algorithms where either part or all of the data needed are perceived during execution time [40].

Several effective offline methods exist in the literature that address POMDPs planning in complex scenarios.

In [41], authors handled the measurement uncertainty and occlusions in intersections scenarios by proposing an offline planner, in this planner the policy is pre-computed through a representation of states that is learned for a specific scenario.

Kamkarian P. et al. proposed in [42] a new offline path planner too designed for single point robots, where the multi-layer solution algorithm works as a unit on workspace components like obstacles to determine the best trajectory in cluttered environments.

in [43], Xianyuan Zhan et al. tackled the problems of computational restriction during agent training and the flexibility control by proposing a light-weighted model-based offline planning framework that grants a high performance planning.

Nonetheless, pre-calculating a policy that needs to be valid for multiple real world scenarios that largely differ from one another was not achievable in an effective way and held back these offline methods.

As for online methods, as we mentioned before, all data will be received in real time and the policy will be calculated during the execution phase, this makes the online methods more versatile compared to the offline methods.

But, because of the limitations in the computational resources the problem treated must be carefully formulated, and even with that the quality of the solution can hardly reach the desired level.

In [44], Ulbrich et al. addressed the problem of changing lanes during driving in highway scenarios using a POMDP framework.
They created a high level state space that is problem specific in order to rise the possibility of solving the addressed problem with an exhaustive search. The state space consisted of states that specify if changing lanes is possible or not in the current situation, and if it is beneficial to change lanes or not as well. The method was effective but it is not possible to generalize it because the state space is very specialized.

Similar work was done as well by Bai et al.[45], in which an Intention-aware online POMDP planner was proposed for autonomous driving in a crowd.

The Monte Carlo Tree Search algorithms were used also to solve POMDP as an online approach, where in [46] Karimi et al. made advantage of this family of algorithms to handle changing lanes for vehicles in highway scenarios.

The Tree Search method was extended even more by using the concept of progressive widening [47], for the purpose of handling continuous state descriptions. Although to estimate the intentions of other drivers a particle filter was used.

Other researchers tried to gather the advantages of both offline and online planning by combining these approaches. Where in [48], Sonu et al. proposed a hierarchical decision making structure, in which the problem of taking decisions was handled through two levels that were modeled as MDPs (authors assumed that each state is fully observable). To solve the high-level MDP a value iteration was used offline, as for the low-level MDP it was solved online using MCTS.

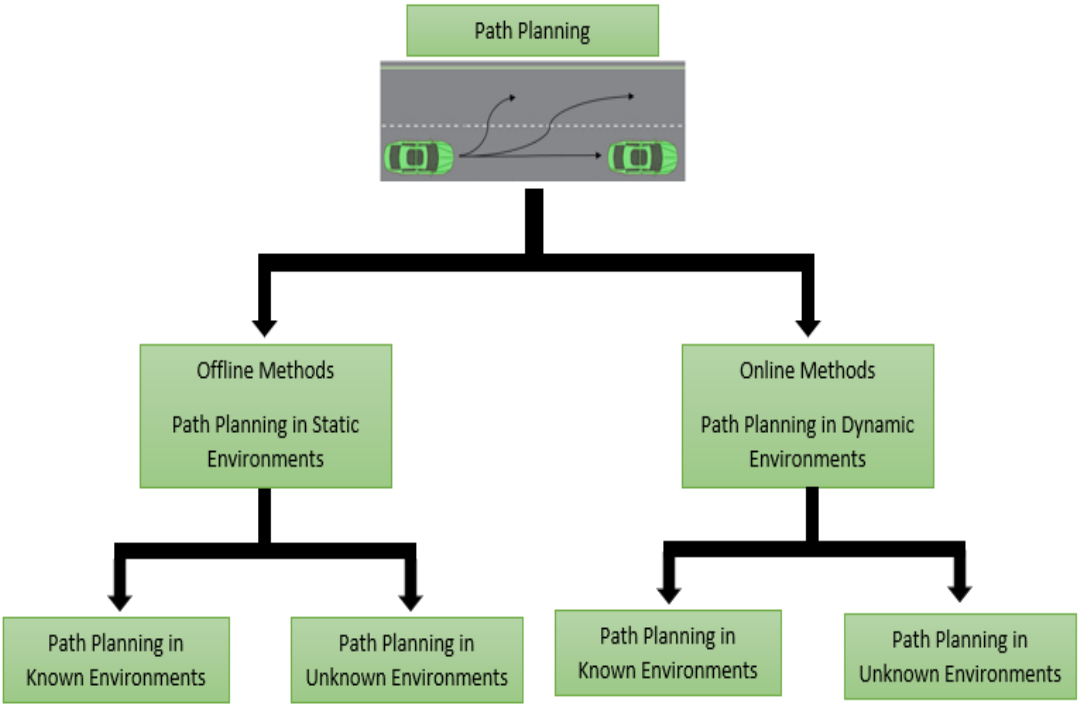


FIGURE 2.1: Path Planning.

2.3 Deep Learning:

The Deep Learning concept became known to the majority of researchers in recent years with having artificial neural networks as its backbone [49].

Even when in this thesis and in our contributions we focused on using Deep Learning concept and integrating neural networks with other methods that we will discuss in the upcoming sections.

Other statistics and machine learning approaches exist in the literature that are powerful and popular, for example Linear Regression, which is an efficient and simple computation method. This method applies least-squares fitting estimation for the approximation of a linear function to a dataset [50].

Researchers consider deep learning as a sub-field of machine learning, where it is based on the concept of self learning and self improving with the examination of certain algorithms.

Although, when simpler concepts are used in machine learning in general, as mentioned above, deep learning uses the notion of artificial neural networks that is

mainly designed for the purpose of copying the humans learning and thinking abilities [51].

Previously, the computation power limited the potentials of the neural networks and their complexity, but recently the big data analytics advancements allowed researchers to use larger and more complex and advanced neural networks, this led to an increase in the quality of learning, observing, and reacting to various scenarios while surpassing human beings.

In sections below, multiple concepts related to Deep Learning are introduced to better grasp the fundamentals of this field that we used in our researches.

2.3.1 Neuron Model:

Many efforts were made that were inspired biologically to create a mathematical algorithmic version of the human brain.

The main ideas were associated with the knowledge acquired about the nerves' connections and their biological activities, the biological neuron models (Figure 2.2), and the experiences been made. Although, these efforts were resolved under some restrictions and assumptions.

The main assumptions focused on how important the neuronal activity is. And because the full human brain is not always active at each single moment, the neurons supposed to have an 'all or none' signal pattern.

Where a certain voltage must be reached above so the neuron can be activated, e.g, from the sum of input connections we can reach an output signal, thus a cause relation effect is created [52].

In an artificial neural model, delays should be considered as well but only respectively to synaptic delay. Another matter is the static network nature because the human brain does not change drastically after each computation.

Also, when an artificial neuron model is in the development phase we can consider dendrites as input connections that lead to a nucleus [53].

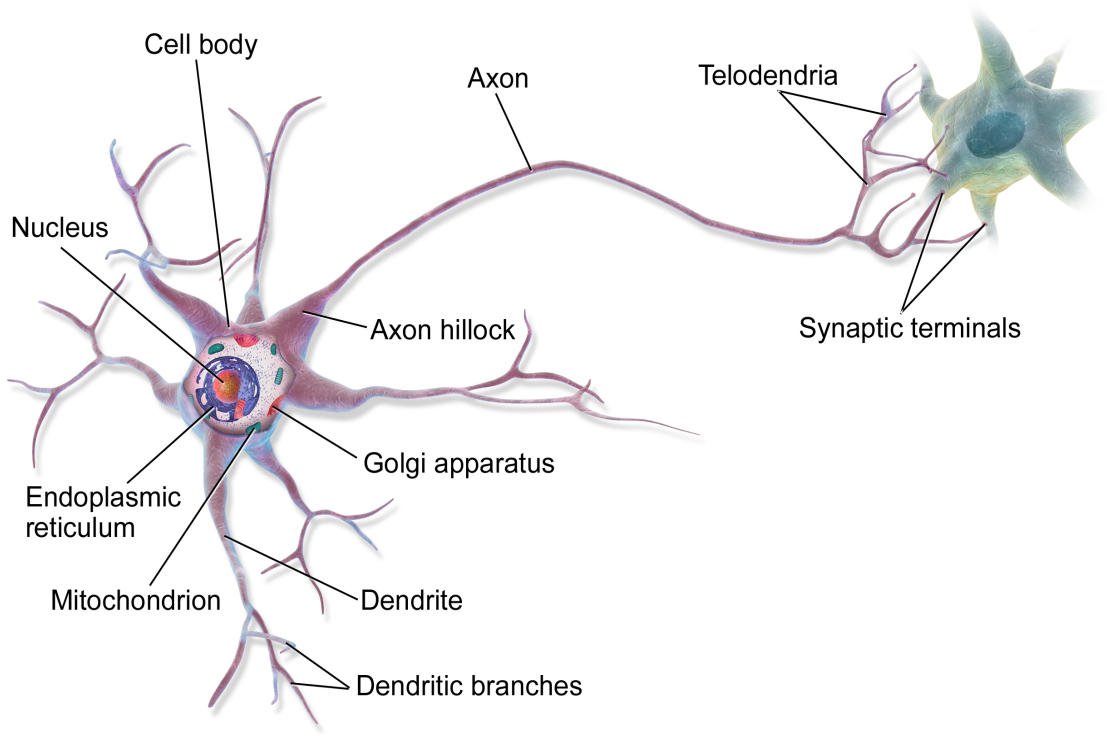


FIGURE 2.2: Neuron Model

2.3.2 Neural Network

We call a group of neuron units that are interconnected, an Artificial Neural Network (ANN) system [54].

In general, the artificial neural network consists of different multiple layers, these layers are composed of various types of neurons, where the neurons of a specific layer are tied to the previous layer [55].

The three types of layers a neural network can be composed of are:

• The input layer: it consists of various types of input neurons, it provides the neural network with initial data in the form of attributes that are based on the desired output that will be reached by the network.

• The Hidden layer: it is generally composed of multiple neurons and it is responsible of converting the high-dimension data to low-dimension.

• The Output layer: which is the final layer of a neural network, and it is responsible of producing the final predictions, before outputting the final results it applies its specific weights and biases.

If we consider a simple one hidden layer neural network, we can represent the input layer as a matrix or a vector [56].

The first step will be to multiply this matrix with randomly initialized weights.

Next, we add a bias term to this multiplication data that will be passed through a non linear function in order to reach the output layer.

The weights size is initialized to:

$$n_h * n_x \tag{2.1}$$

n_h represents the hidden layer's number of neurons.
While n_x is the input layer's neurons number.

The first hidden layer's equation is as follows:

$$h = ActivF(W \cdot x + b_1) \tag{2.2}$$

In this equation, ActivF is the activation function and the dot multiplication is an element-wise multiplication.

Through the hidden layer value we will produce the output result after multiplying the hidden value that represents the input value in this case with the weights before adding the bias term.

After that we pass the results through a non-linear function that will produce the final output function.

$$y = ActivF(W_1 \cdot x + b_2) \tag{2.3}$$

y presents the neural network's output.
The weights size coming from the hidden layer to the output neuron is of size:

$$n_h * n_y \tag{2.4}$$

n_y represents the output neuron's size.
In figure below a simple neural network with all its layers is represented [57].

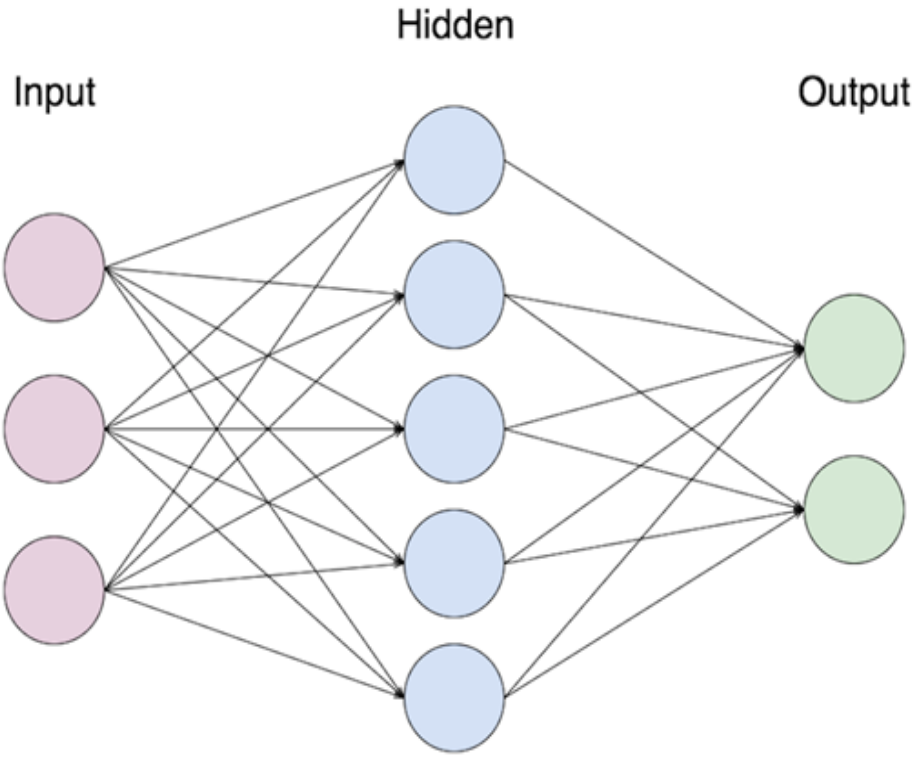


FIGURE 2.3: Simple Neural Network

A run of a neural network is called an epoch, and the first run of a neural network will have weights that were initialized randomly [58].

In a run there are two main steps which are:

• Feedforward: In this step the flow of data in the neural network will be from the left to the right, in other words from the input layer to the hidden layers and lastly to the output layer.

• Backpropagation: After producing the output, its compared with the gold standard output and then the loss value will be calculated. After the calculation of the loss, the standardized weights will be achieved by distributing the weights error from the output layer to the input layer.

Multiple iterations or epochs can be executed until reaching a low stable loss value.

In order to reach a minimum loss and the best weights, gradient descent method is used alongside the backpropagation algorithm.

The gradient descent represents the partial derivative of the errors based on each layer's activation function.
Through these errors calculation, the weights and biases will be adjusted.

In a simple example using the below formula we can represent how the biases and weights will be adjusted in each iteration, while having $I_s(f)$ as the output layer's calculated error.

$$W = W - \alpha\delta_w I_s(f) \tag{2.5}$$

Where: W represents the weights.
α represents the learning rate that is multiplied by error's change according to the weights.

To adjust each layer's biases same calculation will be made.

$$b = b - \alpha\delta_b I_s(f) \tag{2.6}$$

Although, in the first layer the derivation will not happen because it represents the input layer, and this layer does not need an adjustment.

Another property of an artificial neural network that can be adjusted is the number of hidden layers, alongside the number of neurons of each layer [59].

These properties are known as hyper parameters, and they must be tuned in order to adjust and upgrade the performance of the neural network.

By adding more than one hidden layer, a neural network is transformed into a deep neural network. As far as the complexity of tasks goes, researchers tend to use more complex neural networks with multiple layers that can reach even a hundred or more.

In the two next sub-sections we explain more in depth the loss function responsible for calculating the loss of each epoch that tells if the performance of a neural network is upgraded or not, and we mention as well multiple activation functions that can be used while constructing a neural network.

2.3.3 Loss Function:

Loss Function notion is known widely in many fields (machine learning, information theory, statistics, etc.), and it is referred to sometimes as performance measurement, objective function, or even energy function [60].

Generally it is an output layer function and it is used to measure the performance of a deep neural network, where we reach the best output prediction of our model when we have the lowest loss value.

Below is a figure representing an example of a loss function [61].

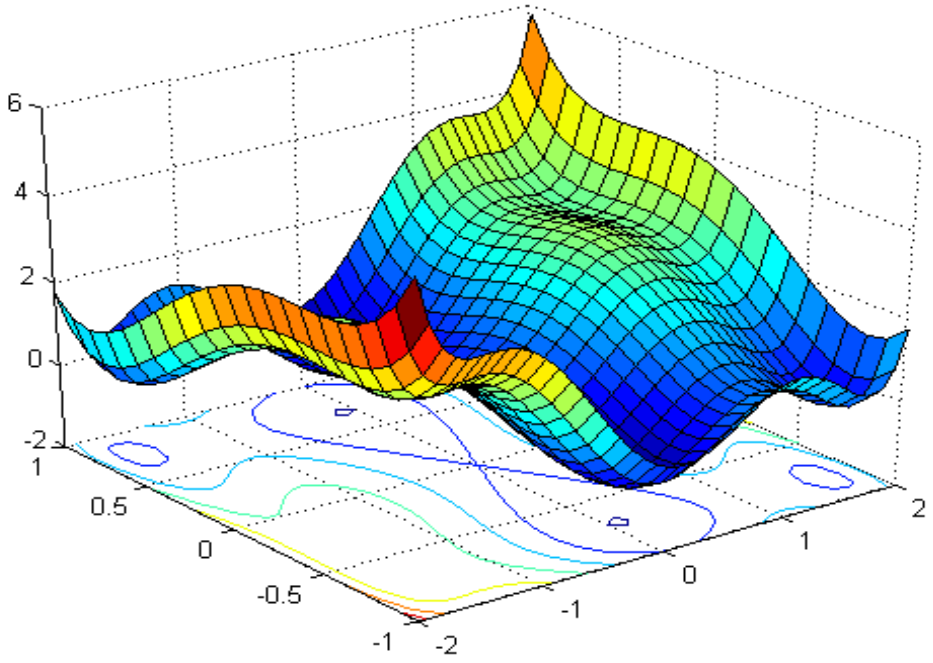


FIGURE 2.4: Loss Function Example

Two main categories separate the loss functions depending on the problems type we address in the real world, and they are the classification category and the regression category.

Next we name some of the most commonly used loss functions in the literature.

2.3.3.1 Binary Cross-Entropy Loss / LOG Loss:

This function represents the main function used for classification problems.

This function has two forms and two formulas to represent these forms depending on the number of classes.

In case of classes number equal to 2, then it is a binary classification and the loss function is represented as follows:

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)) \tag{2.7}$$

In case of classes number bigger than 2, it is a multi-class classification and the loss function is represented as follows:

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot log(\hat{y}_i) \tag{2.8}$$

2.3.3.2 Hinge Loss:

A replacement for binary cross-entropy loss function for classification problems is the hinge loss function, although it was mainly created to evaluate support vector machine models.

$$L = max(0, 1 - y * f(x)) \tag{2.9}$$

2.3.3.3 Mean Square Error Loss:

It represents the most commonly used function in regression problems, and it averages the squared differences between real and predicted values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{2.10}$$

2.3.3.4 Mean Absolute Error Loss:

The second well known function for regression problems, where it averages the absolute differences between real and predicted values.

$$L_\gamma(y, y^p) = \sum_{i=y_i<y_i^p} (\gamma - 1).|y_i - y_i^p| + \sum_{i=y_i \geq y_i^p} (\gamma).|y_i - y_i^p| \tag{2.11}$$

2.3.3.5 Quantile Loss:

From the name of the function, it is applied to predict quantiles, where for a predictions set, the loss value is its average.

$$L_\gamma(y, y^p) = \sum_{i=y_i<y_i^p} (\gamma - 1).|y_i - y_i^p| + \sum_{i=y_i \geq y_i^p} (\gamma).|y_i - y_i^p| \tag{2.12}$$

2.3.4 Activation Functions:

The name of activation functions comes from its role where it is responsible for activating a neuron or not after the calculation of the weighted sum and the addition of bias to it. For the purpose of introducing non-linearity into a neuron output, in order to make it possible to update the weights and biases of neurons by providing gradients with the error of the output, and to allow learning tasks with higher complexity.
Without an activation function a neural network is just a linear regression model.

Many activation functions exist in the literature, we mention below some of the popular ones used by researchers [62].

2.3.4.1 Sigmoid Function:

This function was the first to be mainly used before more sophisticated functions were developed like the Rectified Linear Unit function. The main advantage of the sigmoid function is the suitability to modeling probabilities because its output is limited between 0 and 1. Although, it is not good for learning based on gradient, because its regions are largely saturated.

Below is the equation representing the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

2.3.4.2 hyperbolic tangent Function (tanh):

One of the first functions that had been used alongside the sigmoid function. It became less utilized because of the same problem as sigmoid function which is the saturated regions.

The formula representing the tanh function is as follows:

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.14}$$

2.3.4.3 Rectified Linear Unit Function (ReLU):

ReLU activation function is the most well-known and used function in the literature, due to its suitability for gradient based learning, since it doesn't suffer from the regions problem that sigmoid and tanh functions suffer from, because it's active in half its domain.

Below is the equation defining the ReLU function:

$$ReLU(x) = max\{0, x\} \tag{2.15}$$

2.3.4.4 Exponential Linear Unit Function (ELU):

ELU is also one of the highly used functions, because it extends the ReLU function while not saturating the negative values. Thus, the mean activation is pushed to zero, and because of that normalizing the input is not needed.

We describe the Exponential Linear Unit function through the equation below:

$$ELU(x) = \begin{cases} x, & if x > 0 \\ \alpha(e^x - 1), & if x \leq 0 \end{cases} \tag{2.16}$$

α controls the ELU negative saturation limit (most used value for α is 1).

2.4 Machine Learning:

Machine Learning is probably the most promising field of artificial intelligence.

According to Arthur Samuel, an American pioneer in AI and computer gaming fields, machine learning is defined as " the field of study that gives computers the ability to learn without being explicitly programmed" [63].

In other words this field provides machine with the most important capability of humans which is learning from their environments without being taught or receiving instructions from other humans.

This field grew widely and is used now by most of the big companies in the world as part of their main operations due to its large capability to handle complex tasks.

Machine learning field is composed of three main paradigms as follows:

• Supervised Learning: The objective of supervised learning is to learn a function through some labelled data. Based on input-output pairs set, this function maps input data to output data [64].

• Unsupervised Learning: at the opposite of supervised learning, unsupervised learning works on large amounts of unlabelled data in order to draw inferences from it [65].

• Reinforcement Learning: This area according to researchers around the globe is the most promising area of machine learning and it is the main focus of our contributions. It resolves on having agents act in a world (environment) and get rewards for actions selected. Agents do not know what are the best actions to take at first so they discover that by maximizing the rewards given [9, 10].

Machine Learning field can be divided into two sub-fields according to the required output [66].

These two sub-fields are :

• Classification: It is generally defined as a supervised learning problem where agents need to learn how to classify objects based on some input data.

• Regression: In short, regression resolves on finding interconnections between variables. Where in machine learning, based on these interconnections of variables of a certain dataset an event outcome is predicted.

Many types of algorithms are used to solve this kind of problems, e.g., Decision Tree, Support Vector Machine, Neural Networks and Deep Learning.

To better understand the concept of machine learning, more details about its main paradigms are in the next two sections and in the third chapter named Reinforcement Learning.

2.4.1 Supervised Learning:

As we mentioned before, the objective of supervised learning is to find a function we name it f : X Y , where X represents the input and Y the output of the function:

$$Y = f(X) \tag{2.17}$$

As we see in the formula above, we predict the output desired according to the input that has been given.

The type of the input provided can have various format based on the algorithm used.

The loss function will be calculated as the difference between the gold standard output and the predicted output Y, and the process will proceed towards minimizing the loss in order to give the algorithm the ability to process unseen data.

$$L(Y_i; f(X_i)) \tag{2.18}$$

In the equation above, L is the loss function that is calculated through the model's predicted class and the gold standard output class.

Datasets used in learning process will be divided into three:

• The Training Dataset: These data will be used in the learning phase and will be passed as input data (labelled) in order to reach the output which is a class.

• Validation Dataset: Through these data we will validate the model based on the accuracy metric.

• Test Dataset: These data represent the unseen data that will be passed to the model which was previously trained as input to confirm its accuracy in predicting the output class.

Nonetheless, dividing the dataset as mentioned before means we will reduce the amount of training data and with this we may lose some important trends in the dataset, this will rise the error induced by bias.

Therefore, an alternative is needed which is the cross validation method [67].

In the cross validation method, we divide the dataset into k sets, and then we repeat the learning process k times.

In each iteration, we select one of the k sets as the test/validation set, and the rest of the sets are combined and used as the training data.

To obtain the model's total effectiveness we average the error estimation over all k steps.

From this we can notice that each point in our initial dataset will be used as a validation set once exactly, while it will be used k-1 times as training data, this will reduce the bias because the big part of the data is used always in fitting, while

at the same time we will reduce the variance because most of the data is used as validation data.

2.4.2 Unsupervised Learning:

In unsupervised learning, the learning and training processes are done based on unlabelled data that are usually hard to explore.

This learning type provides the most effective solution when we have data with unknown dimensions, also when a model's outcome is not known to the user.

And this type of learning can be taken as a regression problem.

Numerous methods exist under the unsupervised learning field:

• Clustering: In this technique, datasets are divided into various sets, each set members must be similar in a way. However, this method is not very effective because it can overestimate the similarities between sets' members.

• Anomaly detection: this method is used in general to detect human errors while entering data by detecting the unusual data points in a dataset.

• Various models can be used in the unsupervised learning when the outcome is not known to the user like the encoder-decoder models. e.g., is the autoencoder, in which based on dimension reduction the learning process will happen.

The input data that has been given to the encoder will be encoded in a certain format then decoded to reach the desired output.

2.5 Chapter Conclusion:

In this chapter, a literature overview about methods and approaches that are used in autonomous agents field was introduced.

Where these approaches were divided into two sections : Rule Based methods and learning methods.

The rule based methods was the first approach that researchers used to handle decision making in self-driving vehicles and autonomous machines in general, it was sufficient for the limited environments it was applied for.

But with the increase in the complexity of the tasks being treated a more sophisticated approach was needed, hence the learning methods were founded, and mainly machine learning and deep learning.

Therefore we provided an overall look on concepts related to deep learning, while introducing machine learning with its three main paradigms as well: Supervised Learning, Unsupervised Learning, and Reinforcement learning.

We gave a brief introduction to both supervised and unsupervised learning to better understand the differences between these two classes and the class that our contributions were based on which is the reinforcement learning paradigm.

Chapter 3

Reinforcement Learning

"If we wish to predict the future of
machine learning
all we need to do is identify
ways in which people learn but computers
don't,... yet"

Tom Mitchell

Nowadays, reinforcement learning as one of the main machine learning archetypes represents the peak of mimicking human's capability of learning, and the most promising method to handle decision making problems.

In this chapter we go deep into explaining this paradigm with its famous methods.

3.1 Reinforcement Learning Introduction

According to [10], Reinforcement learning is a framework in which an agent interacts with its environment autonomously based on trial and error concept. This concept was inspired mainly from the early age humans and their curiosity to try unknown new things and learn what is good and what is bad [9].

In this same way, and at the opposite of other machine learning approaches, the reinforcement learning agent explores its surroundings while not knowing which action is the best to take at each situation.

The agent selects at each time step an action while aiming to maximize the sum of rewards it receives after each selected action, in order to reach the final desired goal illustrated in general by a large reward.

Since the reinforcement learning agent is selecting actions, trying new scenarios, and learning at the same time, more advanced methods are needed than what is used in supervised and unsupervised learning.

On top of that, without any knowledge about the environment the agent will be thrown into, an effective exploration and exploitation strategy is needed (more details about exploration and exploitation in upcoming sections).

By using the appropriate method and strategy, the agent will progressively enhance its performance and reach the best behavior for the environment treated.

In general, every learning problem description is constituted of an agent (entity that learns), a reward function (helps agents in learning), performance and accuracy measurement entity (variables or functions that describe how well the agent is performing at the moment), and a goal to achieve (purpose of learning).

3.2 Environment:

Environments are the worlds that the reinforcement learning agents discover, and learn to behave in.

In real world problems, e.g. robot moving, balancing a pole, throwing a ball, the environment can be represented as a 3D space or in the form of pictures in other words 2D.

It can cover as well a full virtual universe, discussion platforms, video games (ATARI, Gym envs), or other simulations (airplane control, conflicts training, etc).

Many formats exist that are used to describe the states of environments, the simplest yet effective one is to describe a state as a vector.

We can represent a state with a space of N dimension included in an environment state space D as follows:

$$S_t \subset D \qquad\qquad S_t \in R^N \tag{3.1}$$

where S_t represents a state at a specific time step t that is contained in the state space of our environment named D.

The dimension of the state space depends on the problem defined.

An appropriate selection of the environment and a correct definition of the state space can affect largely the training process of reinforcement learning agents.

But it is not an easy task to know exactly the best representation possible. The key features of the world treated must be identified in order to make this task easier.

a simple example is robots navigation, where a multidimensional representation of its surroundings is not necessarily needed, the dimension of the state space can be reduced by perceiving the environment as 2D frames or as a simplified vector.

3.3 Rewards and Goals:

Reinforcement is defined in psychology as applying a consequence that enhances the behavior of an organism in the future every time a certain antecedent stimulus precede that behavior [68].

In this same way a reinforcement learning agent learns by receiving a reward value that leads it to upgrade its policy.

That is why agents aim to maximize the sum of rewards accumulated:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{3.2}$$

R_t represents the return in the current time step t, where the goal of the agent is maximizing this return value in time for the current task. Below is another representation of the same equation:

$$R_t = \sum_{k=0}^{T} r_{t+k+1} \tag{3.3}$$

Although, an agent can give more importance to the nearest future reward than to distant future rewards or grant equality to all rewards, this is applied using a discount factor.

A discount factor represents a learning factor, where $0 \leq \gamma \leq 1$.

All future rewards will be discounted by γ^{k-1}. By reducing the discount factor γ to near 0, the reinforcement learning agent prioritizes the closer future rewards than the distant future.

On the other hand, when we increase the discount factor, the agent tends to have a greedy behavior and gives the same importance to the distant future rewards as the close ones.

The previous equation is updated then after adding the discount factor as follows:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{3.4}$$

3.4 Markovian Decision Processes:

For a better and a clearer comparison between decision making problems whether it is in theory or about real world tasks, a common world wide framework is essential to represent the challenges tackled in the field of decision making.

In order to simplify the task of creating this framework, Markov Property was adopted that leads to memory-less processes [69].

Furthermore, the combination of inheriting the Markov chains, producing possible new outcomes through adding actions, and generating rewards that identify the good environments states from the bad ones, is what defines a Markov Decision Process.

In our contributions' experiments, the environments that have been treated were formulated as Markov Decision Processes.

The standardized reinforcement learning formulation through MDP consists of:

• A states space D, that includes the possible states of the environment being treated

• An actions space A, which consists of all actions the reinforcement learning agent can apply in the environment picked for training in any state.

• A reward function r, it represents the quality of an action selected in a certain state, in other words the level of effectiveness of an action selected in a specific state, where $r = \psi(s_t, a_t, s_{t+1})$.

• A transition model T, it represents the probability of reaching s_{t+1} from s_t through applying an action a_t, $T(s_t, a_t, s_{t+1}) = p(s_{t+1}|a_t, s_t)$.

3.5 Partially Observable Markov Decision Process:

Partially Observable Markov Decision Process or POMDP can be defined as an extension to Markov Decision Process, in which not all states of the environment being treated are known to the autonomous agent [39].

As an example, we can consider an agent discovering a maze. The agent travels the maze in a first person view, thus the information provided won't be enough to select the best decision and this violates the Markov property.

Hence, in a decision making task represented as a Partially Observable Markov Decision Process, the agent needs to reach the best solution by constructing its specific state representation, and this can be possible for example by appending the full history of previous runs to the state.

3.6 Functions to Improve Behaviour:

In the next sub-sections the agent's behaviour will be specified mathematically.

Where a reinforcement learning agent is supposed to gather consecutive rewards from multiple episodes for it to be truly learning by reinforcement.

Being in a specific state of an environment will be followed by selecting an action from the action space, applying this action will lead to generating a new state, and a reward related to this transition will be given to the agent.

This behaviour turns into a routine for the agent that can be described as "what is the best action to take in each state of this environment".

3.6.1 Policy:

The continuous behaviour of an agent transforms into a control policy, which represents a map that specifies at each state of the environment which action the agent takes.

An agent's policy has two main types being deterministic and stochastic [70].

If an agent's policy is deterministic, it indicates that executing an action at any state will happen no matter what, in other words its granted for the action to be executed.

At the opposite of a stochastic policy, in which the execution of actions is related to a probability.

The following mathematical formula represents a deterministic policy :

$$\pi(s_t) = a_t, \qquad s_t \subset D \quad a_t \subset A \tag{3.5}$$

In the case of a stochastic policy we can represent it through the following formula, in which the policy is a probability distribution of an action a at a time step t in a state s:

$$\pi(a_t|s_t) = p_i, \qquad s_t \subset D \qquad a_t \subset A \tag{3.6}$$

In theory, an optimal policy is the policy which for each state s at time step t, the reward value for this state is maximized.
In other words it is the policy that, for a particular environment, maximizes the rewards amount collected, and we refer to it by π^*.

We can consider as well a policy optimal or at least sub-optimal if the learning algorithm was given enough episodes of training while it has good exploration and exploitation strategies that lead it to converge.

We refer to a sub-optimal policy with the Bellman's Principe of Optimality [69].

3.6.2 Value Function:

The purpose of a value function is to evaluate a policy in a state $s_t \in D$ at the condition of using the same policy afterwards [71].

Where the policy effectiveness through the value function consists of the collected discounted sum of rewards.

This is what differentiates it from having just rewards values to measure how good the actions selected are, in a way where rewards are immediate (measure the current state of the agent), while the value function estimates how a decision at time step t can affect the rewards in the future.

$$V : V^{\pi} \rightarrow R, \quad V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s\} \tag{3.7}$$

Also, we can say that the trial and error method can approximate the value function through the experiences gathered in the problem being treated.

3.6.3 Quality Function:

Multiple methods exist in the literature with the purpose of acquiring a nearly optimal or an optimal policy through experiments.

One of the most used approaches for this matter is the quality function (interchangeably Q-value) [12].

This function is used as well to measure the effectiveness of executing a specific action in a state at a time step t through the consecutively gathered rewards by applying a policy.

The Quality function and the value function are close in their definitions, but the quality function considers as well the action a at a time step t, where it consists of the long term rewards of executing it at a state through a policy π :

$$Q : S \times A \rightarrow R \;\; Q(s,a)^{\pi} = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s, a_t = a\} \tag{3.8}$$

The value of a state V^{π^*} and the Q value $Q^{\pi^*}(s_t, a_t)$ are equal if we consider the policy used optimal π^*.

$$s_t \subset D \quad a_t \subset A \quad V^{\pi^*}(s_t) = Q^{\pi^*}(s_t, a_t) = argmax\, Q^{\pi}(s_t, a_t) \tag{3.9}$$

3.7 Temporal Difference Learning:

In the upcoming section we will discuss a simple yet effective class of reinforcement learning methods, and probably it represents the most famous and highly used class in the literature, which is the Temporal Difference Learning (TD) [72].

TD learning extends concepts from both Monte Carlo methods [73] (the fact that they sample from the environment), and from Dynamic Programming (The updates are made through the current estimation) [74].

Temporal Difference algorithms can learn from the raw experiences gathered in time without the need of an environment model.

Another advantage is that in TD algorithms after each experience the agent gets, the value function will be updated at the opposite of Monte Carlo methods where a full episode must be completed before the update happens.

Although because of that, updating estimations will rely on previously learnt estimations.

Furthermore, In TD learning methods it is not necessary to discount every single episode like in the Monte Carlo algorithms, because the learning process happens based on every transition from state s to state s' by executing an action a at a time step t, and getting the correspondent reward r at that time step.

Before we discuss the two most well-known methods of Temporal Difference Learning to better explain the fundamentals of this class that was the first base of our contributions in this thesis, some concepts are defined in the next few sections that are related directly to both reinforcement learning algorithms that come after.

Starting with one of the main dilemmas of not just reinforcement learning algorithms, but artificial intelligence in general, which is exploration and exploitation dilemma.

3.7.1 Exploration and Exploitation:

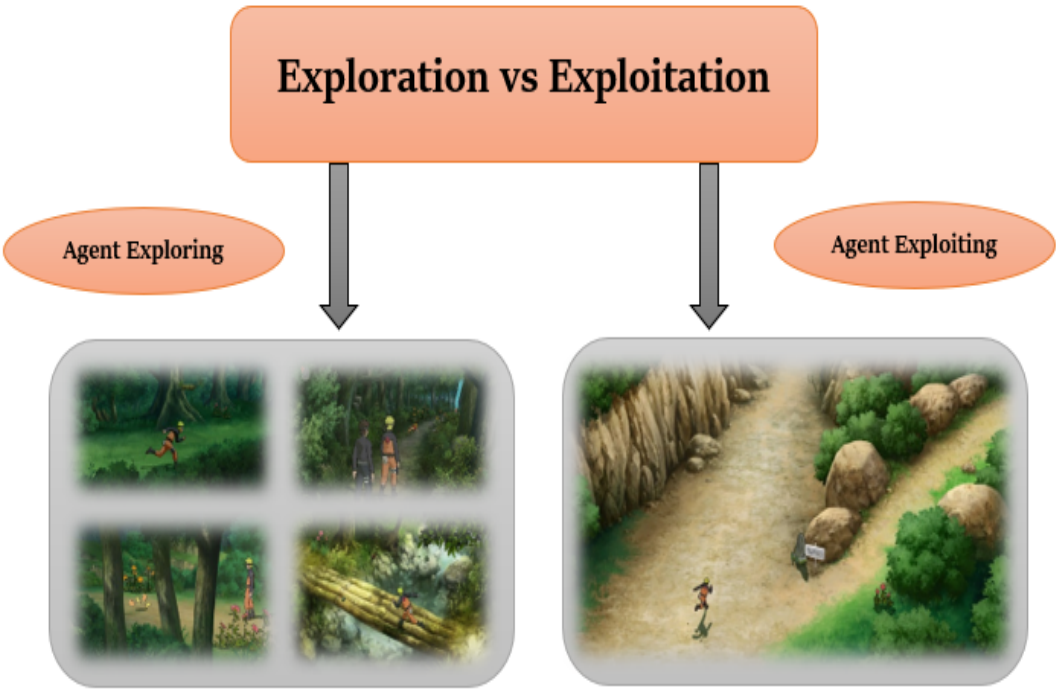


FIGURE 3.1: Exploration Vs Exploitation

Exploration and Exploitation (Figure 3.1 [75]), as mentioned before are highly important features for autonomous agents, in a way where the quality of the exploration and exploitation policy will define the quality of the autonomous agent's policy used to navigate the world being treated.

By exploitation we refer to the act of selecting the most suitable action to the current state of the environment that the agent is navigating based on the available data, although this does not grant the optimally of the action selected.

In order for the agent to be able to select optimal or nearly optimal decisions in all scenarios, it needs to apply exploration. And by exploration we mean the act of trying various actions from the action space in different situations to collect enough information that are used to optimize the agent's policy.

Although, if we have a policy with a high exploration rate the upgrade in the policy will be so slow, and the behaviour of the agent will be random in most parts. While on the other side if we rise the exploitation rate it will be hard to

reach an optimal behaviour because without enough data on the environment the agent's policy will not converge.

Hence, the question arises, is it better to prioritize exploration of the environment in order to gather more information about the unknown states to enhance the policy ? or should we give exploitation more importance and just keep following the higher rewards road ?.

Acquiring a good balance between both can lead to generating a highly effective policy.

One of the possible solutions to this problem is to let the agent explore its surroundings and try various actions in different states, for it to gather a large amount of data on the environment, and test situations that are unreachable if we'll have a low exploration rate.

After reaching the point where the agent thinks it went deep enough in the world being treated, it should then exploit the knowledge collected about the environment in order to improve its decisions gradually and reach an optimal policy.

But, in the case of a dynamic environment, it will be hard to decide if the agent explored the world enough or hasn't yet.

Various methods exist to handle actions selection and to grant an acceptable level of balance between agents exploitation and exploration processes.

3.7.2 Epsilon-Greedy Strategy Selection

One of the well-known actions selection strategies is the epsilon greedy policy [76].

It is a very effective policy that grants a good level of handling both exploring the environment and exploitation of the gathered knowledge.

In this policy, through a quality function Q that receives a state s and an action a as parameters, the action that will be selected is the action with the highest Q value in the current state s_t with a probability of $1 - \epsilon$.

Knowing that epsilon values are included in between zero and one interval ($0 \leq \epsilon \leq 1$), since ϵ indicates a probability.

We select as well a random action to be executed with a probability of ϵ.

Below is the pseudo code of the policy explained above:

Algorithm 1: Epsilon Greedy Policy.

```
n ← uniform random number between 0 and 1 ;
if n < ε then
  |  select random action a_t
else
  |  a ← maxQ(s_t)
end
```

If the Q values function outputs a vector and not a scalar, then we should use a simple max operator instead of a linear loop of actions.

We can notice from the pseudo code above that ϵ value will stay constant in all training phase, this will reduce the rate of exploiting the collected knowledge through the training episodes.

Therefore, a small but effective change in the previous method is to gradually decrease the epsilon value in time.

This will allow for a high rate exploration at the start of training when the agent can try lot of (action,state) combinations, then the agent will stop exploring slowly, and starts depending on the experiences gathered to select the best action in each state.

This is known as the ϵ-greedy decreasing strategy.

the decrease in epsilon value can be carried out through many methods.

The main idea is to initialize ϵ_0 to a value close to 1 in order to increase the exploration rate to the max at the start.

After obtaining the epsilon discount factor df_ϵ, the epsilon value will be updated after each call of the actions selection method as follows:

$$\epsilon_t = \frac{\epsilon_0}{1 + tdf_\epsilon} \tag{3.10}$$

3.7.3 TD Learning Methods Categories:

Temporal Difference Learning methods are usually divided into two sections: On-Policy methods and Off-Policy methods [77].

Before we introduce what is the difference between these two families, two notions must be re-explained in a simpler way, and they are the target policy and the behaviour policy.

By the behavior policy, we refer to the policy used to select actions at each state in order to navigate the environment.

As for the target policy, we mean the policy which the agent tries to learn for the treated environment (for the target policy the agent learns value function).

Going back now to the TD learning methods families.

For the On-Policy methods they represent the approaches that evaluate and enhance the exact same policy the agents use for actions selection.
in other words the behavior policy is same as the target policy in this family of methods.

At the opposite of Off-Policy methods in which the behavior policy differs from the target policy.

On and Off policy methods differ in other things apart from the main property of each that we mentioned already, so below we list few more of their ups and downs:

For On-policy Approaches:

• Always focusing on exploration to reach the optimal policy.

• Can fall in local minima problem.

For Off-policy Approaches:

• Both policies used can be different a lot.

• Approaches can be a bit slow but they stay flexible in new parts of the world they navigate.

3.7.4 Q-Learning (Off-Policy):

The most well-known reinforcement learning algorithm is Q learning, which is a model-free off-policy algorithm [11].

In other words it does not need an environment model, and can solve tasks through rewards and stochastic transitions with no need for adaptation.

This algorithm is used for problems that can be described through a finite Markov decision process.

Hence the states of the world being treated are finite and not continuous, because this approach is based on saving Q values for each (action,state) pair in a table and updating them while the Q learning agent explores its environment.

Thus, the Q learning algorithm reaches an optimal policy by maximizing the total reward expected value, while giving it enough exploration period.

The Q values in the Q table will be updated as follows:

$$Q : S \times A \rightarrow R \tag{3.11}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{3.12}$$

Where:

α represents the learning rate that controls the rate of updating Q values with the new data gathered and it is limited between 0 and 1.

γ represents the discount factor used to decrease the estimated Q values of upcoming states and it's limited between 0 and 1 too.

The full pseudo code of the Q learning algorithm is given below:

Algorithm 2: Q Learning (Off-Policy).

Initialize Q(s,a) = 0 for all a ∈ A and s ∈ S;
while *Episodes Not Over* **do**
  Initialize state s_t;
  while *Current Episode Not Over* **do**
    select a_t, based on an exploration strategy from s_t;
    apply action a_t;
    observe $r_{t+1} and s_{t+1}$;
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
    update state s_t with new state s_{t+1};
    if *s_t is terminal* **then**
      $Q(s_{t+1}, at + 1) = 0$
    end
  end
end

3.7.5 State-Action-Reward-State-Action(SARSA) (On-Policy)

SARSA algorithm is probably the second most used reinforcement learning algorithm and it represents a Q learning variation [78].

At the opposite of the Q learning algorithm which is an Off-Policy algorithm that learns the Q values through a greedy approach, the SARSA algorithm is an On-Policy algorithm that learns the Q values through the actions executed.

The difference between both algorithms is shown in the formula below used to update Q values:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{3.13}$$

We can notice that the formula is based on several information being: the current state, the current action, the reward obtained, the next state, and the next action, and from this the SARSA algorithm got its name (state,action,reward,state,action).

Same as Q learning algorithm, the SARSA policy will continue improving, while the update of the (state,action) pairs stays active.

Below a full pseudo code of the SARSA algorithm is presented to better understand this approach, and highlight the differences between it and the Q learning algorithm:

Algorithm 3: SARSA (On-Policy).

Initialize Q(s,a) = 0 for all a ∈ A and s ∈ S;
while *Episodes Not Over* **do**
  Initialize state s_t;
  select a_t, based on an exploration strategy from s_t;
  while *Current Episode Not Over* **do**
    apply action a_t;
    observe $r_{t+1} and s_{t+1}$;
    select a_{t+1}, based on an exploration strategy from s_{t+1};
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
    update state s_t with new state s_{t+1};
    update action a_t with action a_{t+1};
    if s_t *is terminal* **then**
      $Q(s_{t+1}, at + 1) = 0$
    end
  end
end

3.8 Function Approximation:

Various challenges that are mostly related to continuous domains appeared when researchers attempted to solve decision making and control tasks using reinforcement learning algorithms even with their major potential in this field.

At first reinforcement learning problems depended on tables-based solutions, where for every (state,action) pair a tabular form was defined to represent it.

The collected data in the table are used to update the algorithm policy and enhance its performance, although this approach in most cases is not feasible because of the time and memory limitations when having environments with high-dimensional states. This problem is called the curse of dimensionality [14].

These algorithms face other optimization problems as well, for example: over-training, over-fitting, picking the initial parameters, etc.

Therefore, to be able to use reinforcement learning algorithms to solve real-world problems effectively, multiple information must be put into consideration :

• The fact that the real world is not discrete, specially in autonomous machines field where researchers must handle continuous states in most cases.

• Even when we consider that the environments states are continuous, it is not possible to track all their dynamic components with the same accuracy.

• A rule based approach was used before to try to handle dynamicity of states but it remains very limited.

From the limitations we mentioned above we can conclude that the tabular form of the reinforcement learning algorithms is not the final perfect solution for decision making problems.

Many approaches were applied to handle mostly the dimensions problem and the most promising one is what we saw in chapter 2 which is Deep Learning.

In this thesis we aimed to explore the approaches that combined reinforcement learning with deep learning, where they showed great improvements in many fields of applications.

Deep learning as a standalone approach was applied in many domains, whether its defined as classification or regression:

• Classification class is related to discrete output.

• Regression class is related to continuous output.

3.9 Integrating Deep Learning with Reinforcement Learning:

We assume at first that through a simple reinforcement learning framework, with the purpose of maximizing collected rewards in the process of applying actions to update the environment state at each step, all the intelligence attributes that consist of: planning, perception, imagination, knowledge, control, memory, etc.. are yielded.

From the above assumption, a question arises!. How can we really induce all those intelligence attributes through a simple framework with the goal of optimizing its policy that is related to just one goal?.

We take as an example of a simple reward maximization procedure, an autonomous floor cleaning machine that maximizes the house cleanliness.

To reach the desired goal in a complex environment such as a house, the intelligence attributes must not be neglected.

Therefore, we can consider all those attributes as sub-goals that assist the agent in reaching the best policy that maximizes the rewards of the treated environment.

Hence, all what we need is a single goal to reach everything desirable in any environment.

For this same matter, deep learning as a universal framework offered many solutions and provided many insights in AI topics.

Any task we want to finish or problem we want to solve can be called an objective.

In deep learning for the desired objective, a function will be learned and optimized in order to reach that goal with the best sequence of actions possible.

A worldwide known function approximator is used by deep learning. It consists of various neural networks, and through this class all functions can be represented with more or less acceptable accuracy [79].

Therefore, to fully make use of the advantages of deep learning it was integrated with reinforcement learning in what is known as the deep reinforcement learning methods.

3.9.1 Brief Introduction to Deep Q Learning:

In 2015, One of the Google research groups called DeepMind was able to successfully integrate Deep Neural Networks concept with the classic Reinforcement learning algorithm "Q learning" by adding a new notion named Experience Replay [11].

The new method called Deep Q Learning was tested on various Atari games and provided high performance and even surpassed the human-level performance on some of the games.

In most tasks, representing the Q-function with a table that contains the value of each state-action pair is not practical.

Therefore, in deep Q learning a function approximator is trained that consists of a neural network with θ parameters.

The training procedure will be held through a loss function that should be minimized at each time step i as followed:

$$L_i(\theta_i) = E_{s,a,r,s_{t+1}\sim\rho(.)}[(y_i - Q(s,a;\theta_i))^2] \tag{3.14}$$

where:

- $y_i = r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{i-1})$, and it is known as the temporal difference target (TD target).

- $yi - Q$ represents the temporal difference error (TD error).

- ρ is the distribution over transition s, a, r, s_{t+1} gathered from the world treated, and it is known as the behaviour distribution.

Same as Q learning algorithm, the deep Q learning version is an off-policy method which learns a target policy that is different from the behaviour policy.

In general, an $\epsilon - greedy$ policy is used to select actions, with an ϵ probability to select a random action in order to explore the environment and cover at least most of the action-state space. While the deep Q learning agent learns with a greedy policy as follows: $a = max_a Q(s, a : \theta)$.

As for the experience replay concept mentioned above, it was added to enhance the stability of the neural network updates.

At each time step, information about the current state, action picked, reward received and the new state reached are stored in a buffer named the replay buffer.

Then at each episode of training, a mini-batch will be extracted from the replay buffer and will be used to calculate the loss and its gradient instead of only latest information available.

This will add more credibility to the data gathered because it will be used in multiple updates, and will increase the stability of learning.

Below is a chart-flow of the Deep Q learning algorithm that explains the different steps of this method.

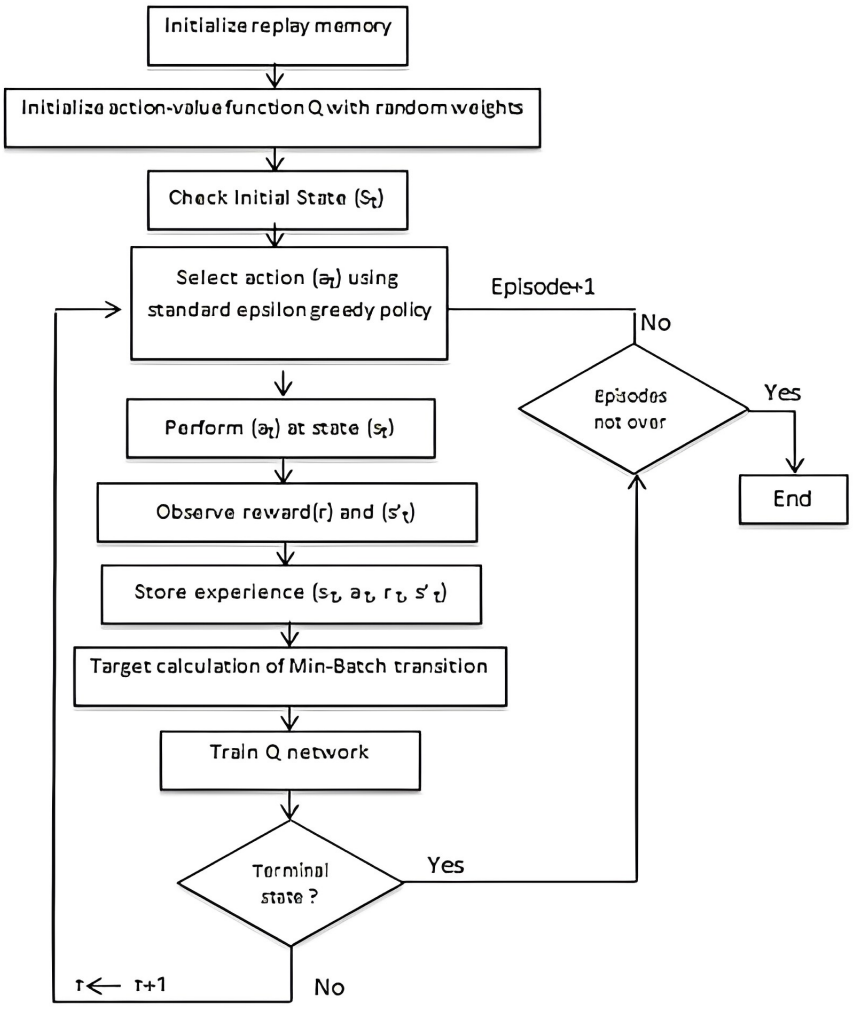


FIGURE 3.2: Deep Q Learning Chart-Flow

3.9.2 Brief Introduction to Deep SARSA:

Deep SARSA is one of the most popular deep reinforcement learning algorithms and it represents an extension of the previously mentioned Deep Q Learning algorithm.

Although in deep SARSA same as the base reinforcement learning algorithm SARSA, the agent learns the optimal policy and behaves with the same policy, and this is what mainly differentiates it from Deep Q Learning method.

In deep SARSA similar as in deep Q learning method the Q value function will be approximated using an artificial neural network instead of the standard static Q table.

We explain with the chart-flow below the various parts of Deep SARSA algorithm to better discriminate it from Deep Q Learning before giving more details about it in the upcoming chapter.

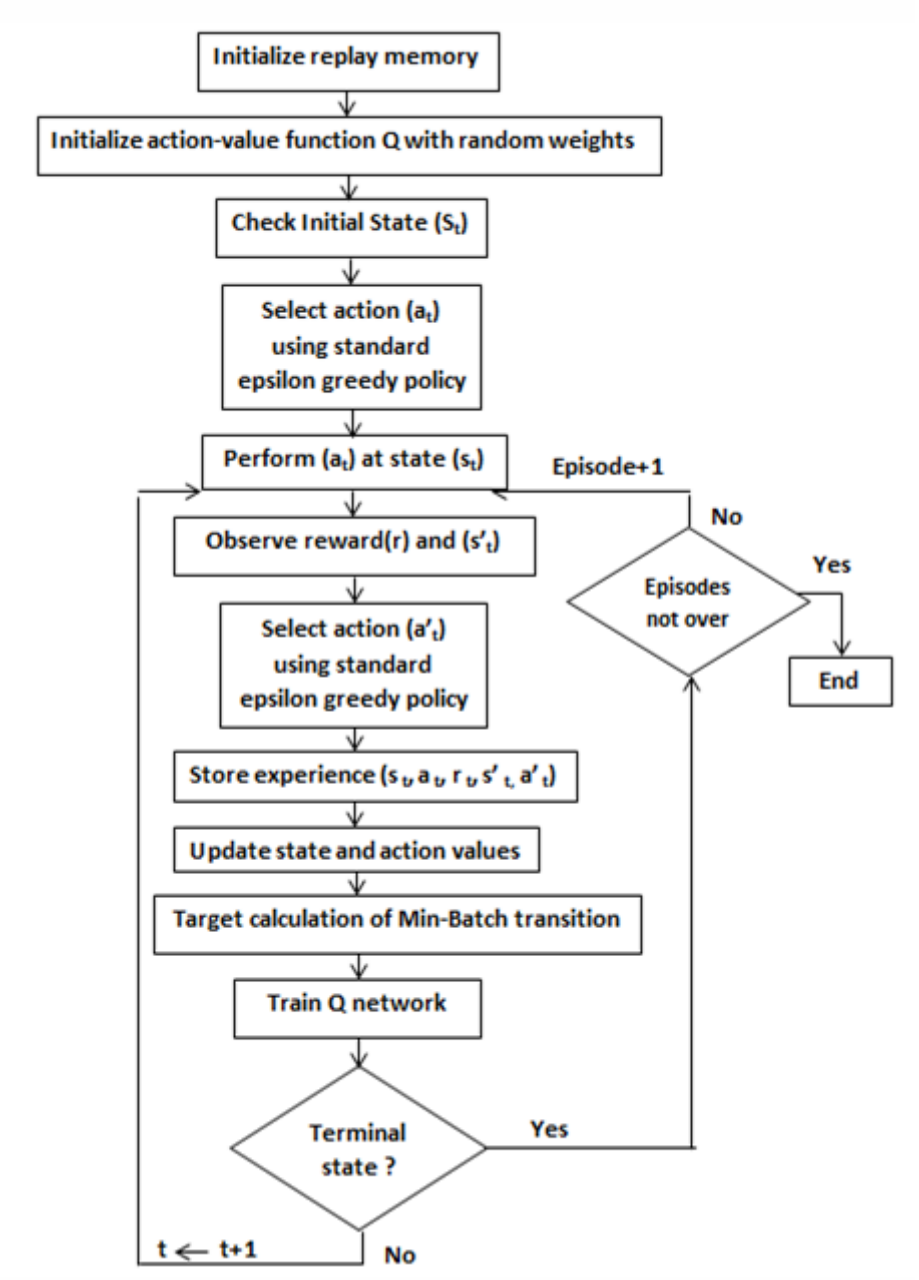


FIGURE 3.3: Deep SARSA Chart-Flow

3.10 Chapter Conclusion:

This chapter held the main notions, definitions, and explanations the reader needs to understand the contributions we provided.

We started with a general introduction to reinforcement learning that represents the most promising approach in machine learning according to many researchers.

Lots of concepts related to this archetype were provided in the chapter, including: Environments, Rewards, Policies, Problems Definition, Exploration and exploitation strategies.

Where the agents based on this approach are the closest to how humans behave and try to learn new concepts, finish tasks, and tackle encountered problems.

We introduced afterwards the most well-known class of algorithms in this paradigm which is Temporal Difference Learning. Two of the main algorithms of this class were the base of our contributions and they are Q learning and SARSA algorithms.

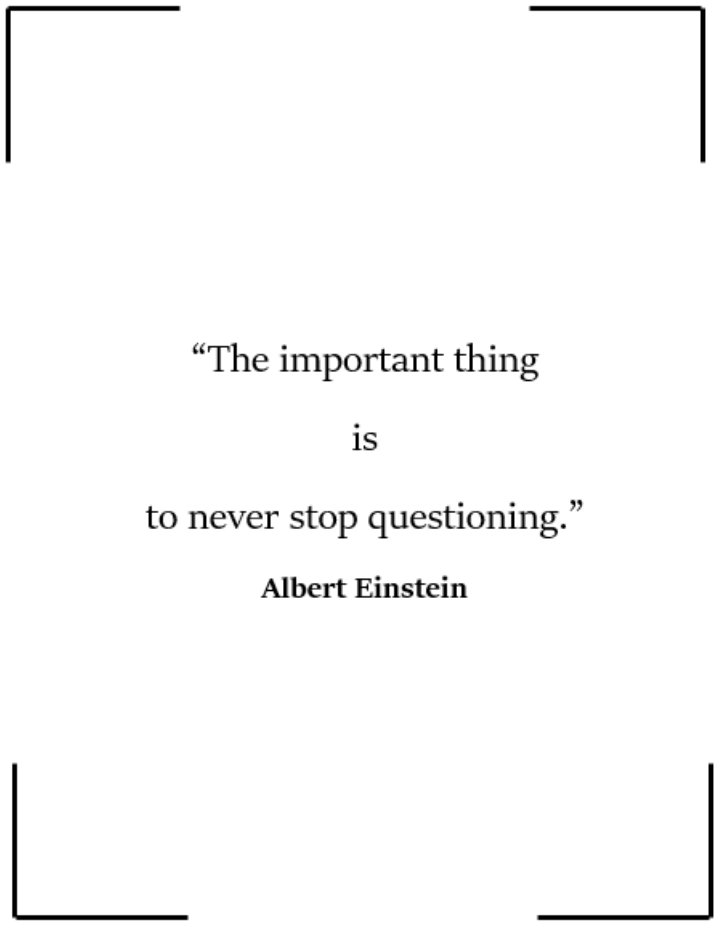
An explanation of these two algorithms was provided while showing the main differences between them.

Although, because of some limitations, these algorithms have (mainly their tabular form), researchers integrated Deep Learning with this class and provided what are known as Deep Reinforcement Learning algorithms. This new class of algorithms provided better performance and enlarged the space of tasks that can be treated.

Therefore, we provided lastly a brief introduction to Deep Q learning and Deep SARSA, two of the main deep reinforcement learning algorithms.

Chapter 4

Contributions



Science is based on contributions, based on the curiosity of researchers in investigating new dimensions and tackle new problems, because the road of contributions has no clear end in the majority of subjects specially with the appearance of artificial intelligence and the integration of this concept in numerous fields in various ways.

In this chapter, our contributions to the decision making field are introduced, and through them more concepts related to deep reinforcement learning are further explained.

4.1 Harmonic SK Deep SARSA

The intricacy of making judgments for an autonomous vehicle (AV) to prevent fatal traffic accidents, provide safety and comfort, and lessen traffic, calls for advancements in the decision-making discipline.

Many algorithms and strategies were used to address these problems, with reinforcement learning (RL) algorithms and deep learning techniques being the most popular ones.

This section will introduce the first paper that proposed the novel extension of the popular Deep Reinforcement learning algorithm Deep SARSA (State-Action-Reward-State-Action), named "Harmonic SK Deep SARSA" [80]. It takes advantage of the stability that the SARSA algorithm offers and employs the idea of comparable and cumulative states stored in a different memory to boost the method's stability and attain exceptional performance that SARSA was unable to achieve because of its nature (on-policy algorithm).

The algorithm's flexibility to unanticipated scenarios during learning and to unexpected changes in the environment was demonstrated through the investigation of our novel extension. Also, the convergence rate was upgraded and the computational load was reduced, since they are important in enhancing decision making that necessities real-time consecutive decisions.

A gym environment simulator named "Highway-env" was used for test and comparison purposes, that contains various highways scenarios with multiple dynamic obstacles.

The test results demonstrated that the proposed algorithm surpassed the comparison algorithms in terms of learning stability and performance, as measured by the following metrics: average loss, average accuracy, maximum speed, average speed, and total reward per episode.

In the next sections more details about the proposed algorithm and the results obtained will be given.

4.1.1 Approach:

Before we explain the proposed Harmonic SK Deep SARSA, we set below the pseudo-code of the Deep SARSA algorithm that was the base of the novel method to better understand the parts of the new extension.

Algorithm 4: Deep SARSA

while *Episodes Not Over* **do**
  Initialize replay memory m;
  Initialize action-value function Q with random weights;
  Observe initial state;
  if *randomvalue* $\leq$ *epsilon* **then**
    select a random action a;
  else
    $actiona \leftarrow argmaxQ(s, a')$
  end
  while *Current Episode Not Over* **do**
    Apply the action a ;
    Observe reward r and the new state s';
    if *randomvalue* $\leq$ *epsilon* **then**
      select a random action a';
    else
      $actiona' \leftarrow argmaxQ(s, a'')$
    end
    Store experience (s , a , r , s',a') in replay memory m;
    update s,a with s',a';
    Sample random transitions (ss ,aa , rr , ss', aa') from reply memory m;
    while *minibatch tansitions not over* **do**
      Calculate target for current minibatch transition;
      if *ss' is terminal state* **then**
        $target \leftarrow rr$
      else
        $target \leftarrow rr + \gamma * Q(ss', aa')$–$Q(ss, aa)$
      end
    end
    Train the Q network;
  end
end

4.1.1.1 SK Deep SARSA

We used a cumulative memory in this extension to store data about the state of the environment at each time step.

We utilize these information in the selection of actions that automatically affect the q values update and the neural network model fitting.

Saving data and using it to select actions in this memory are based on three rules that represent the main idea of the proposed algorithm.

The three rules are : K-Consecutive data, S-similar states, and Largest number of best rewards rule.

We provide below the definition of each rule:

- K-Consecutive data: Each one of the lists that make up the components of the cumulative memory includes K consecutive "state, action, reward" tuples.
Only when K activities have been completed does the saving process in this memory begin, after that a new element is added to the memory at each subsequent time step t.

- S-similar states: A minimum of S cumulative memory components should be accessible for the action selection process.
With the first tuple containing a state that is comparable to the current state of the agent's perception of the environment to be used in choosing the action to take in the subsequent phase.

-After selecting in phase 2 the group of elements, one will be chosen based on a "highest number of best rewards in a K time-steps time interval " rule.

The action of the first tuple of this element will be selected as the current action, remembering that every element is composed of (state,action,reward) tuples.

The full steps and details about the new strategy used in the novel extension are provided next:

- For each action selected, We iterate over each component of the cumulative memory.

If:
$cum\_mem[i][0][state] = current\_state$
Then cum_mem[i] is saved.

- We check next if $Count(saved\_elements) < S$.
If not enough elements are available, a random action is chosen from the action space.

- If $Count(saved\_elements) > S$, the current action will be chosen from saved elements, each of which contains K successive tuples of action, observation, and reward.

- Each saved element's weight is set to zero.

- Then, using a variable named index that has a range of 0 to K-1 and stands for the time-steps, we loop through each element.

- The reward values of the tuples at the current index of each saved element are now compared.

The element whose reward value is highest receives a weight increase of 1.

- Until we reach the last tuple we keep processing the rewards comparison.

- The first tuple action of the element with the highest weight will therefore be the action chosen.
To put it another way, the element with the greatest number of the best actions in the following K successive states.

- If two or more elements share the same weight, we choose a random action from these elements first tuple.

The new strategy will have an impact on both updating Q values and fitting the model because they both directly depend on the mechanism used for choosing actions.

Bellow is the pseudo code of the proposed SK Deep SARSA algorithm.

Algorithm 5: SK Deep SARSA

```
while Episodes Not Over do
    Initialize replay memory m;
    Initialize cumulative memory ;
    Initialize action-value function Q with random weights;
    Observe initial state;
    if randomvalue ≤ epsilon then
        select a random action a;
    else
        actiona ← argmaxQ(s, :)
    end
    while Current Episode Not Over do
        Apply the action a ;
        Observe reward r and the new state s';
        if K actions applied then
            save last K (action,state,reward) in the cumulative memory;
        end
        if randomvalue ≤ epsilon then
            if S elements in the cumulative memory with first tuple's state same as the current state
              exist then
                select element with more better decisions per K states saved;
                if more than 1 element with same number of good decisions then
                    random selection between elements first tuple's actions;
                else
                    select first tuple's action of element;
                end
            else
            end
            select a random action a';
        else
            actiona' ← argmaxQ(s, :)
        end
        Store experience ( s , a , r , s',a' ) in replay memory m;
        update s,a with s',a';
        Sample random transitions( ss ,aa , rr , ss', aa' ) from reply memory m;
        while minibatch tansitions not over do
            Calculate target for current minibatch transition;
            if ss' is terminal state then
                target ← rr
            else
                target ← rr + γ ∗ Q(ss', aa')
            end
        end
        Train the Q network;
    end
end
```

4.1.1.2 Selection of Parameters using Harmony Search algorithm:

As was already noted, the two primary factors that determine how well the SK deep SARSA performs are S (the number of comparable states kept in the cumulative memory) and K (the amount of (state, action, reward) tuples).

The deep learning model's accuracy and loss values, as well as the agent's flexibility in changing lanes and top speed during training, all have a significant impact on how well the suggested method performs.

As a result, we made the decision to employ the harmony search method to choose the ideal pairing of S and K parameters while taking metrics such as loss, accuracy, and maximum speed into account [81].

4.1.1.2.1 Harmony Search Algorithm Adaptation: The HS algorithm for the SK-DS algorithm is modified as follows:

- The first population of the S and K parameters combinations is used to initialize the harmony memory:

$$S_i, K_i = S_{il}, K_{il} + Rand * (S_h, K_h - S_l, K_l) \tag{4.1}$$

Where S_l, S_h, K_l, K_h are the lowest S value, the biggest S value, the lowest K value, the biggest K value respectively.

While we represent every combination with a vector. Therefore, the HM represents a list of vectors.

- Next, we begin creating and assessing additional combinations up until the terminal criteria is met.

- Creating a new candidate (combination of S,K) is done through j steps (j represents a candidate length, $j = 2$).

The following directives will be carried out for each component of the new candidate:

Verify whether a random value is below the HMCR boundary (the HS parameter determines whether or not we choose a candidate element at random).

The new candidate's current element is randomly chosen depending on the HMCR test.

Then we proceed to the next element or continue with the next instruction.

If the element is not chosen at random, element j of the new candidate will be the jth element of a randomly chosen HM member.

If we find that the current j-th element of the new S,K combination equals either of the current parameter interval edges, then a pitch adjustement is applied.

A predetermined fitness function will assess the new S and K combination after it has been built.

The worst HM member will be replaced if the new S,K combo candidate performs better than it, otherwise it will be ignored, and we will start creating and assessing new elements again.

Below is a pseudo code that shows how we combined the proposed SK deep SARSA with HS algorithm.

Algorithm 6: Harmony SK deep SARSA

```
Initialize harmony memory with first population;
Initialize HMC, HMCR, and number of parameters;
while stop criteria is not reached do
    for j ← 0; i < numberofparameters; j ← j + 1 do
        if rand(0,11) < HMCR then
            Xj ← jth dimension of a randomly selected HM member.
              Apply pitch adjustment distance( + 1 or -1 or keep same value( no adjustment) or random
              value if Xj = max or min possible value of current parameter);
        else
            Let Xjin X be a random value;
        end
    end
    Execute SK deep SARSA with new parameters(fitness);
    if X's fitness is better than the fitness of the worst HM member then
        Replace the worst HM member with X;
    else
        Disregard X;
    end
end
```

The full framework of the new extension of Deep SARSA algorithm is presented below as in the paper.

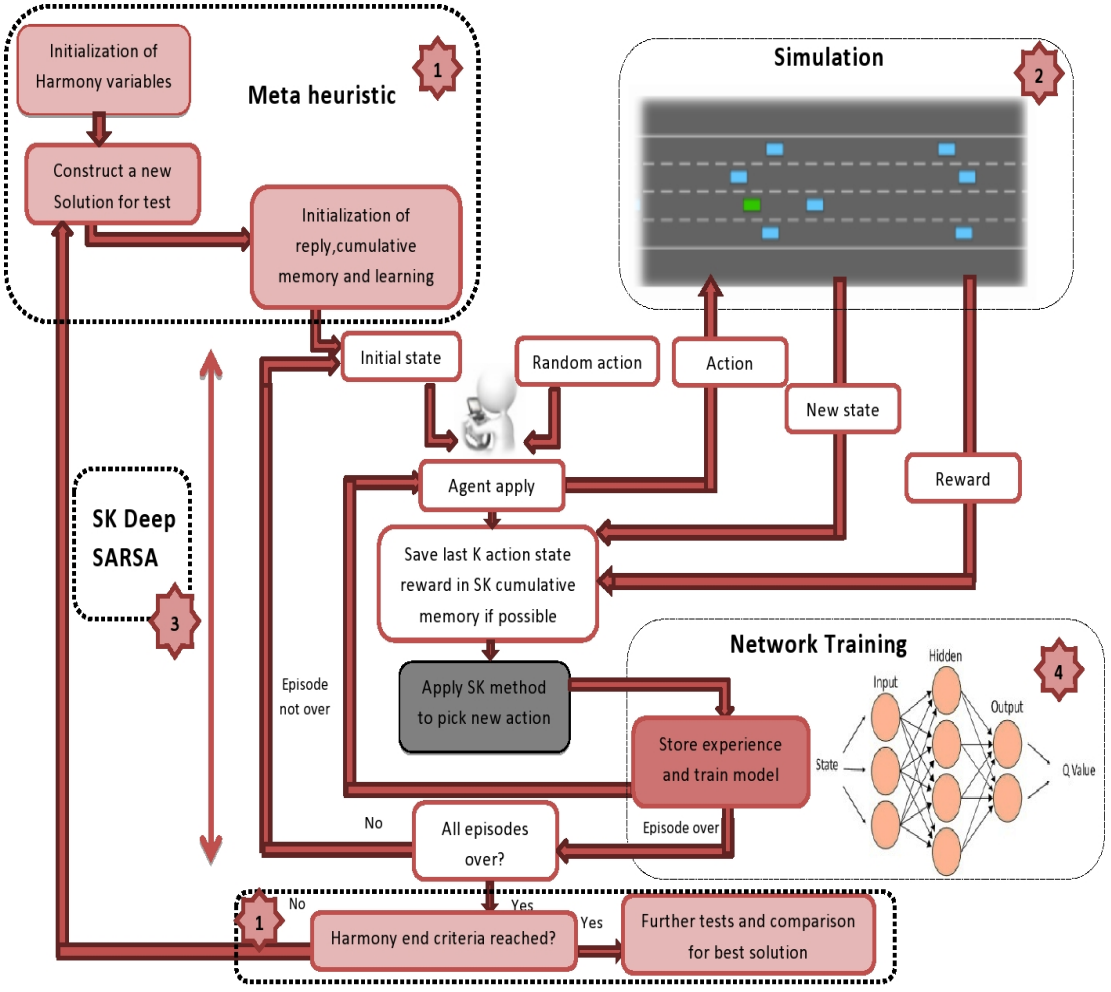


FIGURE 4.1: Harmonic SK Deep SARSA Framework.

4.1.2 Problem Formulation

The issue raised in this paper can be described as a Markov Decision Process (MDP) [69] in which the agent is aware of all possible states.

4.1.2.1 Environment (Simulation):

In order to demonstrate the proposed algorithm's adaptability to environmental changes, we utilized various N-lane highway simulation setups as the primary testing setups, along with several additional scenarios, including a highway that

gets updated from 4 lanes to 2 lanes, another highway setup that changes from 4 lanes to 6 lanes, and a merge scenario.

The Gym toolkit was used to create a simulator called "highway-env" [82], that is used to develop and contrast reinforcement learning methods.

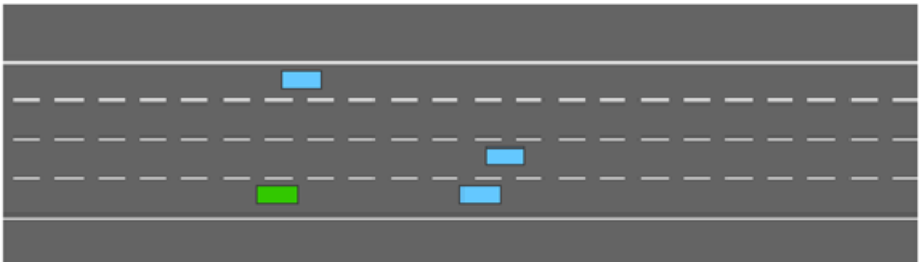


FIGURE 4.2: Highway environment

The major case utilized to test the proposed algorithm is illustrated in Figure 4.2 above; where the self-driving car is represented by a green car, which is our agent that is run by the suggested algorithm.

The starting automobiles count that the agent strives to avoid collision with is 50, and a 30 steps length was picked for each episode.

Our self-driving vehicle has a top speed of 30(m/s) and a bottom speed of 20(m/s) on the highway.

The simulator informs the agent on the velocity and placement of the self-driving car and each surrounding vehicle at each time step.

Our self-driving car wants to travel as fast as it can on the highway without colliding with any other vehicle, and it better stays in the right lane to maximize its benefits.

4.1.2.2 States:

The simulator, as previously noted, offers all necessary information.

When utilizing "highway-env," a variety of observations can be used, including the kinematics observation (the one chosen in this study), a grayscale picture, and an occupancy grid.

The first row of the kinematics observation, which contains information about the self-driving car, is a VxF array, as described in the "highway-env" documentation.

The data given are referred to as features, and they stand for the parameters where:

Column 01: represents an id of each car.

Column 02-03: have x and y values that correspond to the vehicle's position on the road.

Column 04-05: hold Vx and Vy values, which stand for the vehicle's velocity.

TABLE 4.1: State observation representation

Vehicle	x	y	Vx	Vy
AV	2.0	3.0	11.0	9.0
V_1	-9.0	5.0	13.0	0
V_2	4.0	15.0	9.0	1.5
....	....	....	....	....
V_n	19.4	9.6	20.0	2.2

Every time the number of vehicles on the highway changes, the observation's size, which is proportionate to the number of obstructions, is updated.

While coding, the observation is represented as a numpy array that has a (5,5) shape, and it is sent to the agent alongside other data like the reward, that is generated after executing an action at a time step t through a step function that takes this action as its parameter, and affects accordingly the environment. Then, to train the model we reshape the observation to a (1,25) array.

4.1.2.3 Actions:

We employed a discrete actions space, which is made up of actions that alter lanes and vehicle speed, with the agent following the road at the specified velocity.

There may be circumstances where certain actions are not available. For instance, the vehicle cannot choose action 3 "accelerate" after exceeding the highway's maximum speed limit.

TABLE 4.2: Discrete actions

Action	Description
zero	deviate to the left lane
one	Stay in lane
two	deviate to the right lane
three	increase speed
four	decrease speed

4.1.2.4 Rewards:

In this environment, the main goal was to travel at the fastest possible speed while staying on the wanted lane of the highway and attempting to avoid crashes with the other vehicles; thus, the reward function includes a velocity term and a collision term:

$$R(s,a) = a \times \frac{V - Vmin}{Vmax - Vmin} - b \times colission \tag{4.2}$$

Where V is the vehicle's current speed, Vmin is the lowest speed, and Vmax is the highest speed it is capable of traveling.

4.1.3 Experiments:

4.1.3.1 Searching for the best parameters:

The first part of the experiments was to find the best combination of S and K parameters.

The proposed method "Harmonic SK Deep SARSA" was executed for a couple of iterations in order to try multiple combinations and reach the best possible one, based on the adapted harmony search algorithm integrated with the new extension.

Loss, accuracy, and maximum speed values were the three metrics that determine the best values, where we calculate the average speed, the average loss, and the average accuracy value throughout the last 30 episodes.

At this stage, we concentrated on the most recent 30 episodes while calculating comparison values for various parameters combinations to ensure that the agent successfully completes the entire length of the path in every episode.

The upcoming Table shows the iterations data that resulted in the best combination of S and K parameters:

TABLE 4.3: Selecting parameters process

Iteration	**New Comb.**	**Average Loss** **Average Accuracy** **Average Maximum Speed**	**Update Harmony Memory ?**	**First Solution**	**Second Solution**	**Third Solution**
First	S : 8 K : 4	0.5439933960424971 0.7624972473995913 26.439721514925285	Yes	S : 8 K : 4	-	-
Second	S : 8 K : 6	0.4651962781774948 0.796222172381712 23.120705154665416	Yes	S : 8 K : 4	S : 8 K : 6	-
Third	S : 10 K : 4	0.41615157792702173 0.8166777146464647 28.56668064621099	Yes	S : 8 K : 4	S : 8 K : 6	S : 10 K : 4
Fourth	S : 9 K : 4	0.4387629679259327 0.7480425347222224 26.57660624907089	Yes	S : 9 K : 4	S : 8 K : 6	S : 10 K : 4
Fifth	S : 9 K : 5	0.41080831041463833 0.882155417856245 28.926913303093592	Yes	S : 9 K : 4	S : 9 K : 5	S : 10 K : 4
Sixth	S : 6 K : 5	0.4511613169079374 0.8914071637426902 26.473138641632612	No	S : 9 K : 4	S : 9 K : 5	S : 10 K : 4
Seventh	S : 7 K : 4	0.5673312256519313 0.8056850830610022 26.09416854206156	No	S : 9 K : 4	S : 9 K : 5	S : 10 K : 4
Eighth	S : 10 K : 7	0.4573308635056107 0.8294926803520555 25.14396366881703	No	S : 9 K : 4	S : 9 K : 5	S : 10 K : 4
Ninth	S : 10 K : 5	0.46381271409077773 0.7839236111111111 25.706139385822237	No	S : 9 K : 4	S : 9 K : 5	S : 10 K : 4
Tenth	S : 8 K : 7	0.4167807707583739 0.8194791666666668 24.49103907330196	Yes	S : 8 K : 7	S : 9 K : 5	S : 10 K : 4
Eleventh	S : 7 K : 5	0.46793439650149266 0.7821701388888889 27.981323946550013	No	S : 9 K : 4	S : 9 K : 5	S : 10 K : 4
Twelfth	S : 7 K : 6	0.48524234256258153 0.8941513338411317 23.922167283103736	No	S : 9 K : 4	S : 9 K : 5	S : 10 K : 4

4.1.3.2 Comparison Metrics:

Adopting proper comparison measures is important in order to completely study and analyze the effectiveness of the suggested algorithm.

Five widely known metrics were adopted in the paper, being: the average loss value, the average accuracy, the maximum speed value reached, the average speed, and finally the total reward.

Various algorithms were used for comparison purpose to investigate the robustness of the new extension, where the main two comparison algorithms are Deep Q Network and Deep SARSA that were previously explained.

We employed the kinematics representation for our environment states along side with the discrete action space, and we trained for 120 episodes each algorithm, where each episode is composed of 30 samples, the agent can either finish all the road or crash with another vehicle.

From this comparison the proposed approach proved its stability in the learning process compared to the deep SARSA algorithm while keeping up with avoiding collisions and achieving the highest performance in each episode.

Metrics used for comparison were :

- Mean Square Error.

- Accuracy.

- Reward.

- Speed.

- Average Speed.

The table below shows the calculation formula for each metric:

TABLE 4.4: Comparison Metrics definitions and equations

Metric	Definition	Equation
MSE	**Mean Square Error:** a frequently used measure of the differences between values predicted by a model or an estimator and the values observed	MSE= $\frac{1}{N}\sum_{n=1}^{\infty}(\hat{y}_i - y_i)^2$
Accuracy	**Accuracy :** the fraction of action selections our model got right	Accuracy= $\frac{NumberOfCorrectActionSelection}{TotalNumberOfSelections}$
R	**Reward function:** composed of a velocity term and a collision term where **v,vmin,vmaxv,vmin,vmax** are the current, minimum and maximum speed of the ego-vehicle respectively , and a,b are two coefficients.	**R(s,a)=** $a\times \frac{(V-Vmin)}{(Vmax-Vmin)} - b \times colission$
S	**Speed (Rate):** scalar quantity that measures the distance traveled (d) over the change in time (Δt)	S = $\frac{d}{\Delta t}$
Average speed	**Average speed:** The sum of speed valued per step at each episode divided by 80 (max steps per episode)	Average speed= $\frac{\sum Sperstep}{MaxStepsCountPerEpisode}$

As noted in the simulation portion, experiments were carried out in the highway and merge environments.

Each algorithm was trained for a total of 120 episodes with 30 samples in each training session.

Where we employed kinematics representation for the environment states coupled with discrete action space.

Below are the plots that shows the results of the comparison between the proposed algorithm and Deep Q Network and Deep SARSA to show the major improvement.

4.1.3.3 Highway Comparison Results:

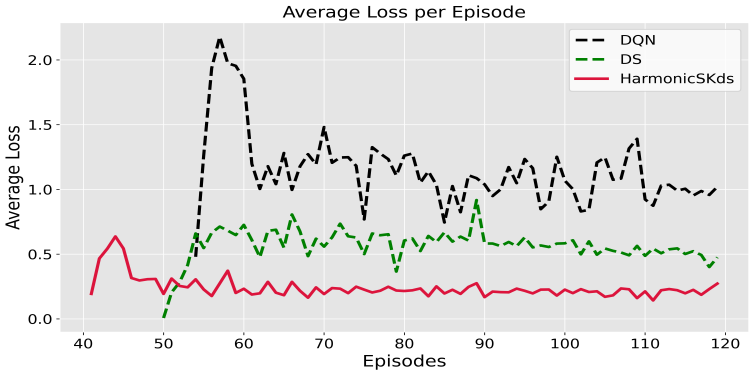


FIGURE 4.3: Avg Loss Comparison

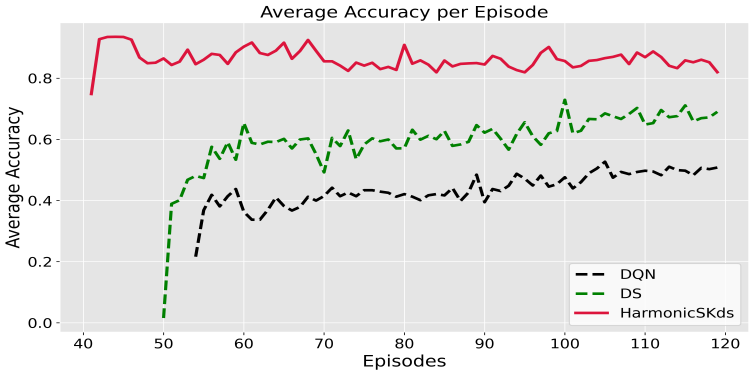


FIGURE 4.4: Avg Accuracy Comparison

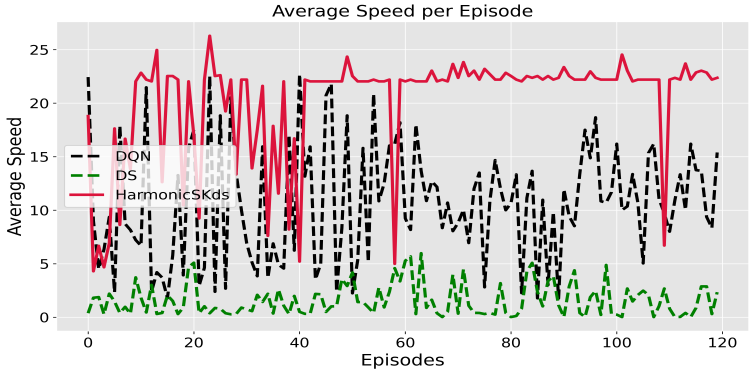


FIGURE 4.5: Average Speed Comparison

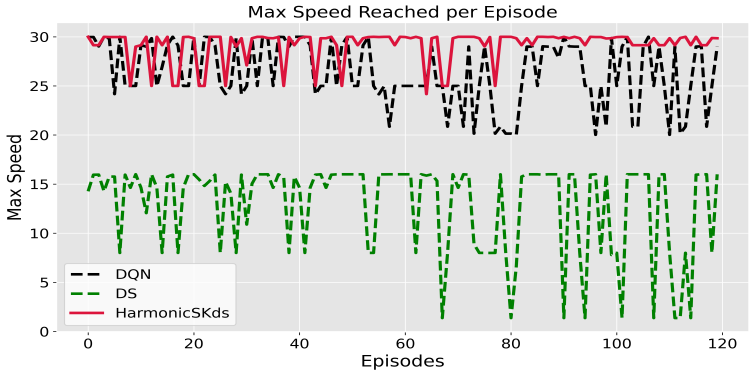


FIGURE 4.6: Max Speed Comparison

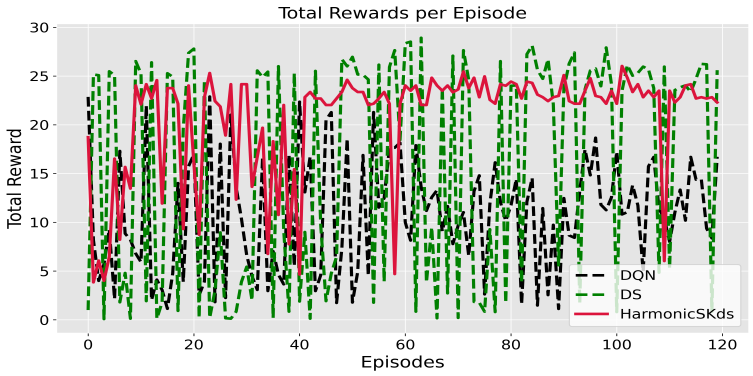


FIGURE 4.7: Total Reward Comparison

Four new calculable metrics were used in extended tests and comparisons to examine more thoroughly the effectiveness of the suggested algorithm. - Average loss reached in last 30 episodes. - Average accuracy reached in last 30 episodes. - Average speed in training during last 30 episodes. - Average total reward during training. In table 4.5 below the results of the extended tests are shown.

TABLE 4.5: Extended Tests Results in Highway Scenario

Algorithms	**DQN**	**DS**	**Harmonic SK DS**
Average Loss	1.048	0.540	**0.208**
Average Accuracy	0.475	0.653	**0.856**
Average Speed(m/s)	12.150	1.426	**21.998**
Average total reward	12.519	21.357	**22.696**

4.1.3.4 Merge Results:

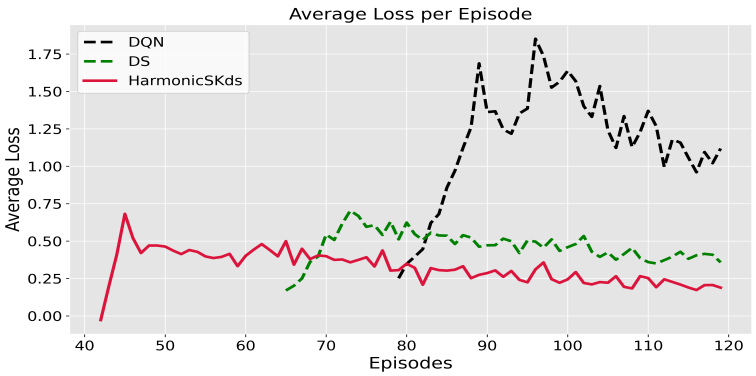


FIGURE 4.8: Avg Loss Comparison

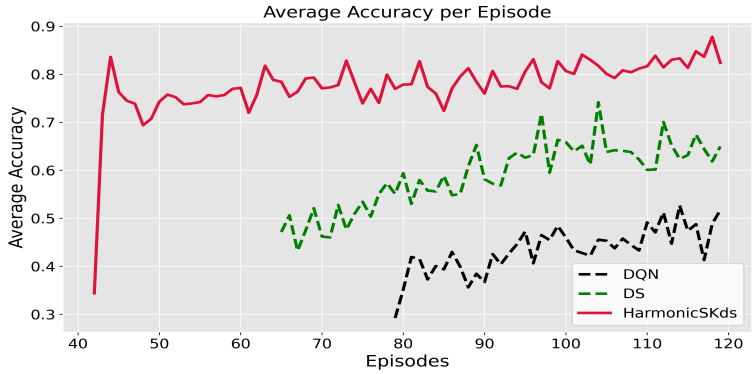


FIGURE 4.9: Avg Accuracy Comparison

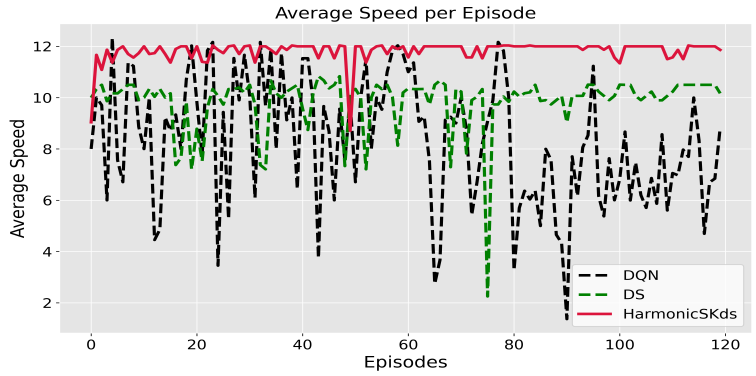


FIGURE 4.10: Avg Speed Comparison

Table 4.6 provides detailed results for this setup using the same comparison measures.

TABLE 4.6: Merge Scenario Results

Algorithms	Average Loss	Average Accuracy	Average Speed(m/s)
DQN	1.312	0.453	7.038
DS	0.433	0.636	10.218
Harmonic SKDS	**0.239**	**0.811**	**11.898**

4.1.3.5 Complexity of proposed model

In order to explore the improvements of the suggested approach in relation to the benchmark models, predefined libraries were used to calculate the execution time and memory load.

This allowed us to check thoroughly the complexity of the suggested technique and the results are shown in table 4.7 below.

TABLE 4.7: Computational Load Results

Metrics	CPU User Time	CPU System Time	Peak Memory	Memory Increment
DQN	31 min 12 s	40 s	569.62 MiB	4.63 MiB
DS	12 min 30 s	13.2 s	529.71 MiB	5.00 MiB
Harmonic SK DS	12 min 56 s	14.5 s	524.22 MiB	4.45 MiB

4.1.3.6 Efficiency in unexpected situations:

More tests were conducted to see how well the suggested algorithm would respond to unexpected changes in the dynamic highway scenario.

During the training phase, the environment's design underwent some alterations, going from a four-lane highway with 50 nearby cars to:

- 50 neighbouring cars in a highway with 2 lanes.

- 80 neighbouring cars in a highway with 6 lanes.

And the results are shown in table 4.8 below.

TABLE 4.8: Unexpected Scenarios Results

Metrics	6 Lane Highway 80 Cars	2 Lane Highway 50 Cars
Best Loss	0.2395131684239475	0.24134124492107856
Best Accuracy	0.9200014823	0.8968236
Average Max Speed	28.92148529852313	27.51813964564123
Average Total Reward	19.1421354651231	18.1684139

4.1.3.7 Sensitivity Analysis For Parameters Selection:

Additional tests were conducted on the highway environment using various combinations of S and K parameters, where we averaged the outcome of 10 runs per combination. The results are in Figure 4.11 and Table 4.9 below.

These tests were done to further investigate the impact of the parameters chosen using the modified harmony search algorithm on the performance of the SK DS algorithm.

TABLE 4.9: Sensitivity of Parameters selection

K S	2	4	6	8	10	12	14
2	14.9183562	14.709177748	15.84667637	14.25773947	18.60701226	18.954506945	8.474210231
4	9.783876251	15.4198698313	11.173716564	9.84052078321	17.4657537285	16.0743809221	14.5322326389
6	18.0562704631	20.8671068531	11.9435696612	16.8085881406	11.6369586359	15.3987877111	14.1842735139
8	18.6920481062	20.7193855641	21.4299991798	16.6093624790	20.4965438903	14.7182077786	18.0113237159
10	19.4647929620	18.5279412724	15.4827309852	16.3505541428	18.8451688480	15.8042715313	19.0434907871
12	12.82209428647	20.7368740465	19.42206507166	18.76752543871	18.72865971937	16.57166270115	16.9523728737
14	13.3007118543	18.2239447302	18.3084892232	14.95382692	21.0496402672	16.8105644796	20.8829670035

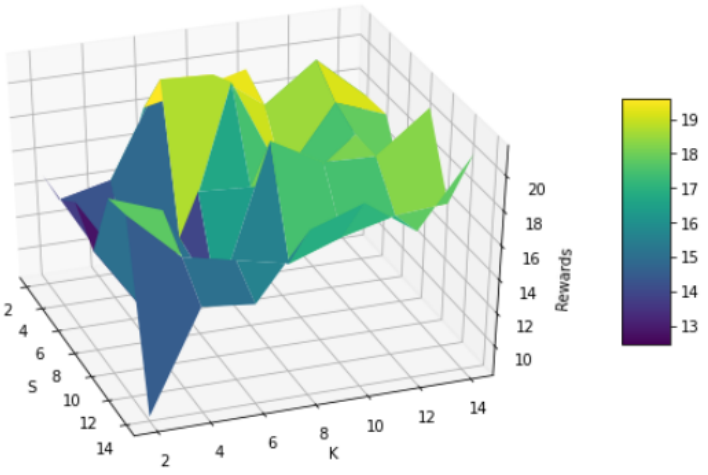


FIGURE 4.11: Sensitivity of Parameters selection

4.1.3.8 Robustness Test

In another part of this research, we attempted to evaluate the robustness of our model and its efficacy when changing inputs in various ways through many experiments on the suggested algorithm.

After training the model on a four-lane highway, we tested it in a number of additional configurations that weren't the same as the setup that was initially used for training.

In order to create a new testing environment, the number of highway lanes was adjusted, and the volume of traffic was continuously raised to make it harder to avoid obstructions.

Between the various settings, comparison was made using the total reward measure.

Table 4.10 and the plot 4.12 below show the results of the robustness experiment.

TABLE 4.10: Total Rewards Values per configuration

Vehicles Lanes	50	60	70	80	90	100	150
2 lanes	22.7560348	22.600000008	23.134052299	18.957609248	20.414088708	19.970050177	18.550725089
3 lanes	26.78638265	23.468123001	24.470702259	25.517626304	21.082444002	21.482042818	19.03779571
4 lanes	23.2907959	25.12413015	22.7667730	24.138233	22.2147589	23.93461645	22.0648355743
5 lanes	23.53830128	22.116225517	22.341224987	22.277158878	23.150205728	20.116222514	22.582312744
6 lanes	21.9817591542	23.7335397538	21.2349471828	23.500020224	22.36891265	23.964562135	20.4407230471
7 lanes	22.73202124	23.612215457	24.89888895	23.09120751	20.779874571	24.526209172	22.9578886678
8 lanes	21.29672971	20.65752908	22.527421	20.85468956	25.04451337	22.44836487	22.133261573

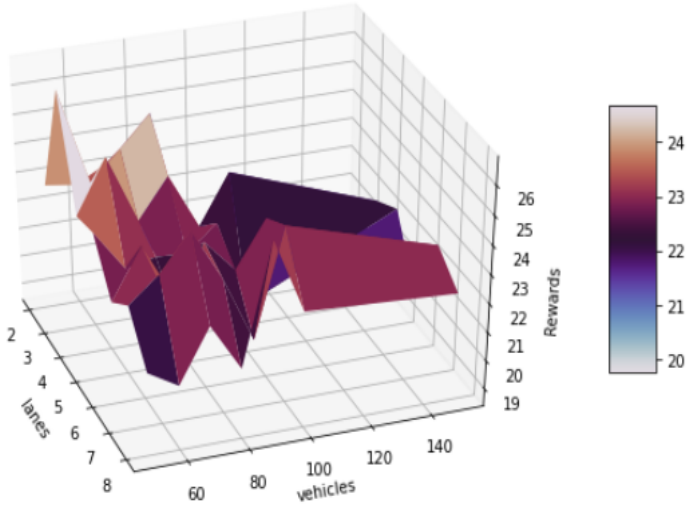


FIGURE 4.12: Robustness Results

4.1.3.9 Multi-Configuration learning performance:

The performance of pre-trained RL agents is affected differently by changing learning scenarios, and the quantity of data acquired from simulators mid-learning.

Multiple training sessions and testing were conducted while changing the highway scenario's starting configuration in order to increase or decrease the amount of data used for training (lanes and cars count in the highway) in the section of this paper entitled "Multi-Configuration learning performance."

On shared highway configurations, the various pre-trained agents were utilized to monitor the effects of altering the size of the training set throughout the test phase.

Table 4.11 displays the average outcomes of 10 tests with 10 runs for each setup for the different agents.

These experiments were done on three pre-trained RL agents that had been trained on three different data scales: a two-lane highway, a five-lane highway, and an eight-lane highway.

Three new setups were used for test after learning and they are: a three-lane, a six-lane, and a nine-lane highway scenario.

TABLE 4.11: Results of Multi-Configuration

Agents	Metrics	3-Lane Config	6-Lane Config	9-Lane Config
Small-Data Scale 2-Lane 50 cars	Accuracy	0.8461	0.85001	0.752
	Reward	22.58423	20.13359	18.8462135
	Comput-Time	10 min 23 s	10 min 35 s	10 min 30 s
	Speed	20.7757417	19.60579	17.96442
Medium-Data Scale 5-Lane 100 cars	Accuracy	0.87123	0.84982	0.8592
	Reward	22.49523	22.82169	22.911364
	Comput-Time	10 min 31 s	10 min 30 s	10 min 42 s
	Speed	20.02576750	20.9502346	22.43142
Large-Data Scale 8-Lane 150 cars	Accuracy	0.825	0.94641	0.982241
	Reward	22.12315	25.7469522	23.131259
	Comput-Time	10 min 26 s	10 min 38 s	10 min 47 s
	Speed	18.3919384	25.1274192	23.587719

4.1.3.10 scalability of the proposed model:

Scalability of a model in machine learning refers to how expanding the training set affects the resources needed for machine learning algorithms during training, particularly memory burden along with the accuracy supplied.

The section of this paper labeled "scalability of the proposed model" contains further in-depth analyses that capture the primary metric related to the scalability of machine learning models as well as the average accuracy obtained of ten runs per configuration; to show how changing in the configuration- of the training process with short intervals can affect the proposed model's memory load.

TABLE 4.12: Scalability Results

Configuration		Peak Memory	Memory Increment	Accuracy
Two-Lane	**50 Cars**	523.97 MiB	4.20 MiB	0.876719
	100 Cars	534.54 MiB	4.51 MiB	0.797845
Four-Lane	**50 Cars**	521.22 MiB	4.45 MiB	0.84321
	100 Cars	515.36 MiB	4.26 MiB	0.863028
Six-Lane	**100 Cars**	529.90 MiB	5.42 MiB	0.81706
	150 Cars	511.37 MiB	4.88 MiB	0.8232748
Eight-Lane	**100 Cars**	508.97 MiB	4.03 MiB	0.835634
	150 Cars	513.34 MiB	4.18 MiB	0.800451
Ten-Lane	**150 Cars**	515.39 MiB	4.72 MiB	0.8162137
	200 Cars	525.39 MiB	5.48 MiB	0.817845

4.1.4 Discussion and Conclusion:

Using an enhanced Deep SARSA model and an adjusted Harmony Search method, a brand-new algorithm dubbed Harmonic SK Deep SARSA was proposed in this research.

A Markov Decision Process was used to represent the issue. It exhibits different roadway scenarios with a merge scenario.

In this task, the autonomous vehicle must prevent collisions with obstacles on the road, travel at the fastest possible speed, switch lanes without swerving, and adapt to the path's unanticipated changes.

The key contribution of this study was the invention of an innovative algorithm that can tolerate unforeseen modifications during the learning process while gaining the best policy, achieving both high performance and increased stability in decision-making in real time.

The novel extension was built around precise performance-improving parameters.

When compared to other learning models, the parametric aspect of the suggested model, which requires selecting the S and K parameters using the customized harmony optimizer, gave a larger policy space.

This allowed us to improve the model's robustness by generalizing across unfamiliar situations in the middle of the learning phase.

To prove that the suggested strategy is effective, a gym environment was employed to mimic the various instances.

Additional tests were run to demonstrate the proposed model's flexibility to unforeseen events throughout the learning phase and the testing phase. The results revealed that Harmonic SK Deep SARSA surpassed the comparison algorithms in both the learning phase and the testing phase.

The model's complexity was examined and it was determined that it is unaffected by the implementation of the new policies, thus the duration of learning was kept to a minimum.

Although, a parameters-based method can lead to an increase in the controller's complexity, a greater count of parameters, the necessity for a more complicated parameters selection process, and an increase in the execution cost, it can increase the generality of the proposed model.

The proposed model proved that it can be a good choice that exhibits significant potentials for solving decision-making problems in a variety of contexts, such as autonomous vehicles, gaming, aerial navigation, and industrial robots, which call for several real-time consecutive judgments.

4.2 Alternative Bidirectional Q Network

As mentioned before, learning strategies, particularly reinforcement learning, were applied to further improve decision making in the field of self-driving vehicles to give more safety, luxury, minimize traffic, and fatalities.

Although, there is still a chance that these algorithms could be improved because of numerous issues, such as convergence rate, stability, handling numerous dynamic environments, raw performance, resilience, and algorithms complexity.

To address these issues, mainly handling multiple environments, we proposed in a second paper a new extension of Deep Q Network algorithm called "Alternative Bidirectional Q Network" [83].

This extension's major goal is to improve both Q values update and exploration strategies' stability and performance in order to fill a vacuum in the literature that often sets focus on just one side to deal with decision-making across many setups (avoiding obstacles, goal oriented environments, etc).

Data about previous, present, and upcoming states are used to update the Q values in "Alternative bidirectional Q Network," where the actions are chosen based on the relationship between these data to handle numerous scenarios: highways, roundabouts, merges, and parking.

This idea increased the stability during learning and gave reinforcement learning agents an improved equilibrium between exploitation and exploration.

The proposed algorithm was trained and tested using the same gym simulator as in the first paper, but with additional situations.

4.2.1 Approach

The fundamental idea behind reinforcement learning methods is to store samples of data from episodes at training time t, and then utilize those samples to fit the model later.

DQN, the widely used deep reinforcement learning approach, is an off-policy approach used in this research that estimates the Q values of each (state, action) pair by approximating the Q values function using neural networks.

Each data sample used for updating Q values is recorded in a replay memory and contains knowledge about a time-step t.

Using batches of this data, a greedy policy is employed to learn new information.

The following DQN pseudo code is offered to further clarify the DQN ideas used for the construction of our suggested extension:

Algorithm 7: Deep Q Network.

```
while Episodes Not Over do
    Initialize replay memory m;
    Initialize action-value function Q with random weights;
    Observe initial state;
    while Current Episode Not Over do
        if randomvalue ≤ epsilon then
            select a random action ;
        else
            actiona ← argmaxQ(s, a′)
        end
        Apply the action a ;
        Observe reward r and the new state s';
        Store experience ( s , a , r , s' ) in replay memory m;
        update state s with new state s';
        Sample random transitions ( ss ,aa , rr , ss' ) from replay memory m;
        while minibatch tansitions not over do
            Calculate target for current minibatch transition;
            if ss' is terminal state then
                target ← rr
            else
                target ← rr + γ ∗ maxQ(ss′, aa′)
            end
        end
        Train the Q network;
    end
end
```

The replay memory used to train the proposed technique contains elements that carry data from three successive time-steps: t-1, t, and t+1.

The information saved at each point in time includes the state, the action chosen, and the reward received after completing the action.

Thus, the replay memory won't begin to save data until the first two actions have been successfully completed, regardless of how well they worked or what rewards were given for each one of them.

As a result, the replay memory stores a combination of nine pieces of data for each time-step t: the previous state, the previous action, the previous reward, the current state, the current action, the current reward, and the upcoming state, the forthcoming action, and the upcoming reward.

The action selection strategy (selecting actions every time-step) and the updating of Q values to fit the model in the novel approach are both based on the replay memory.

4.2.1.1 Action Selection Policy:

In this section, we go into more depth about how to choose actions based on data stored in the adaptive replay memory.

- The current state of the environment being treated is passed as an argument to the action selection function.

- Next, an epsilon greedy strategy is used to either choose a random action (which is often chosen at the initial phase of training), or move on to the next steps:

$$Action\,at\,time\,t \begin{cases} random\,(a), & \text{with probability } \epsilon \\ new\,policy, & \text{with probability } 1-\epsilon \end{cases} \tag{4.3}$$

- As was previously indicated, each replay memory element has the following information: Previous state, Previous action, Previous reward, Current state, Current action, Current reward, and Next state, Next action, Next reward. The memory is looped through, and:

if *replayM [i][current_state] = current_state* **then**
 ∟ save this element

- If no elements matching the requirement are identified, we choose a random action.

- We compute diff1 and diff2 for each element I discovered, where:
diff1 = elmnt_i[current_reward] - elmnt_i[previous_reward]
diff2 = elmnt_i[upcoming_reward] - elmnt_i[current_reward]

$$diff2 > diff1 \begin{cases} \text{true, save elmnt\_i[current\_action], diff2.} \\ \text{false, discard element i.} \end{cases} \tag{4.4}$$

- We choose the action with the greatest difference related to it from the list of chosen actions, and this action will be carried out while the simulator is in its current state.

The pseudo code of the action selection method is detailed in the following Algorithm 8.

Algorithm 8: Actions Selection Policy Function.

```
if random value ≤ epsilon then
    return random action;
else
    for element in replay memory do
        if current state = element's saved current state then
            save element;
        end
    end
    if no saved element then
        return random action;
    else
        for element in saved elements do
            calculate saved current reward - saved previous reward;
            calculate saved upcoming reward - saved current reward;
            if second difference > first difference then
                save second difference and element's current action;
            end
        end
        search for the saved current action with best difference and return it ;
    end
end
```

4.2.1.2 Q Values Update and Model Fitting:

The replay function's instructions for fitting the neural model and changing Q values are listed below.

- Pick a batch at random from the replay memory.

-In order to save time, we batch predict all previous, present, and upcoming states and save the outcomes in the corresponding previous, current, and upcoming targets.

- For each batch element:

1- We calculate:

- The difference between the most recent and preceding reward.
- The difference between the next reward and the most recent reward.

2- When the first difference is smaller than the second difference:

$$previous\_target \leftarrow current\_reward + \gamma * Q(ss, aa) \tag{4.5}$$

$$current\_target \leftarrow current\_reward + \gamma * Q(ss', aa') \tag{4.6}$$

$$upcoming\_target \leftarrow current\_reward + \gamma * Q(ss', aa') \tag{4.7}$$

When the opposite: the current, previous, and upcoming target elements will get affected the most recent reward only.

3- Using the previous,current, and upcoming target to fit the model.

Algorithm 9: Replay Function for updating Q values and Fitting Neural Network.

sample mini-batch from replay memory;
mini-batch prediction ;
while *mini-batch transitions not over* **do**
  if *upcoming reward - current reward current reward - previous reward* **then**
    $previous\_targetelement'sQvalue \leftarrow currentreward + \gamma * maxQ(currentstate, currentaction)$
    $current\_targetelement'sQvalue \leftarrow currentreward + \gamma * maxQ(nextstate, nextaction)$
    $upcoming\_targetelement'sQvalue \leftarrow currentreward + \gamma * maxQ(nextstate, nextaction)$
  else
    previous_target element's Q value ← current reward ;
    current_target element's Q value ← current reward ;
    upcoming_target element's Q value ← current reward;
  end
end
fit model with previous_target;
fit model with current_target;
fit model with upcoming_target;

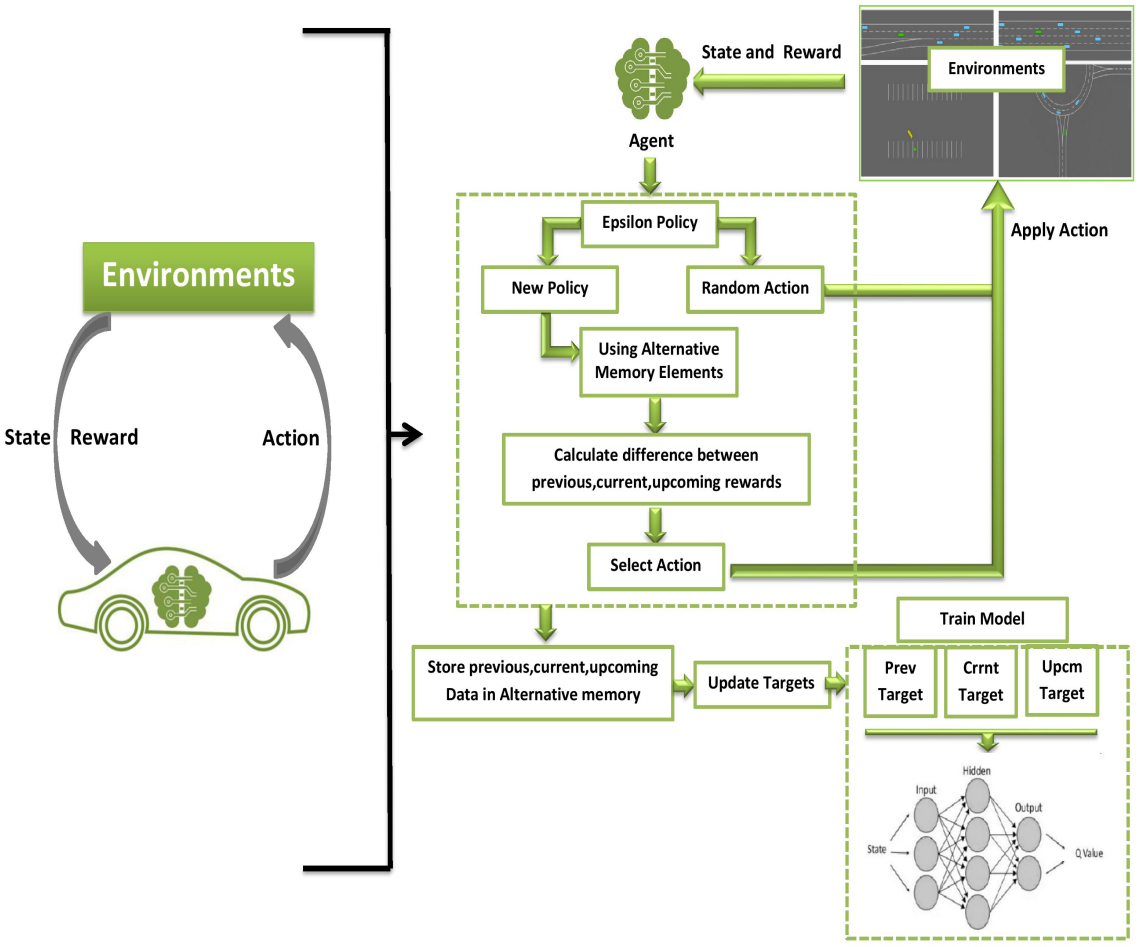


FIGURE 4.13: Architecture of Alternative Bidirectional Q Network.

4.2.2 Problem Formulation:

In this study, we formulated the issue as a Markov Decision Process, where the agent is aware of the conditions in each environment at every time step.

4.2.2.1 Simulation:

The same gym-toolkit simulator from the prior research was used because it offers a variety of learning contexts, and multiple scenarios are treated to show the superior performance of the proposed algorithm against different Deep reinforcement learning techniques.

The environments that were treated in this paper are:

4.2.2.1.1 Highway: In a model of a four-lane highway (Figure 4.14), the self-driving car's objective is to prevent crashes with other autos while traveling at a high speed.

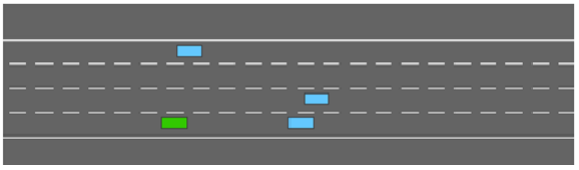


FIGURE 4.14: Highway Scenario.

4.2.2.1.2 Merge: Figure 4.15 illustrates the merge scenario in which the autonomous vehicle must maintain a high speed while allowing vehicles to merge into highway lanes and avoiding collisions with them as well as the vehicles that are on the highway previously.

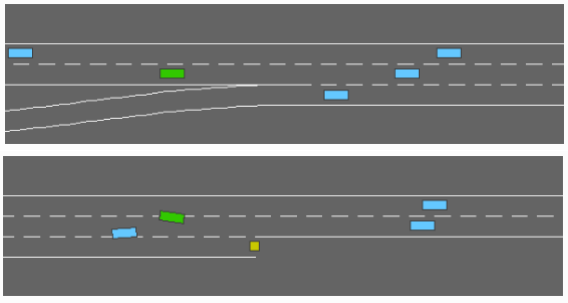


FIGURE 4.15: Merge Scenario.

4.2.2.1.3 Roundabout Environment: In order to navigate a roundabout safely in this setting, the self-driving car needs to learn how to follow the flow of traffic, handle lane changes, and prevent crashes all at once. The treated environment is depicted below (Figure 4.16).

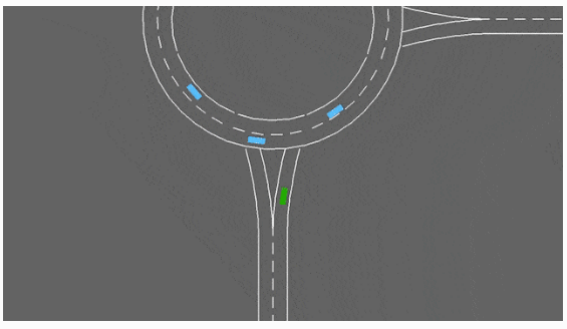


FIGURE 4.16: Roundabout Scenario.

4.2.2.1.4 Parking Environment: This setting serves as an example of a goal-based situation in which agents must learn how to accomplish the set objective with the fewest and most effective actions. The autonomous car ought to understand how to position itself in a certain area of the parking lot, as seen in Figure 4.17 below.

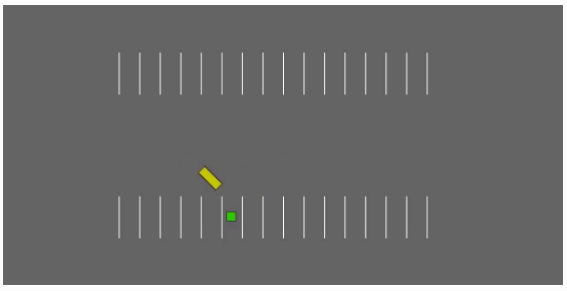


FIGURE 4.17: Parking Scenario.

4.2.2.2 States Representations:

In this paper two types of representations were used to illustrate the environment and they are : Kinematics observation and Time to collision observation.

4.2.2.2.1 Kinematics Observation: According to the gym-env records, this kind is expressed by a vehicle X feature table, where agents are provided with details on the location and speed of the autonomous vehicle as well as any surrounding cars, such as the position(x,y) and the velocity(Vx,Vy).
Both the highway and the merge (on-ramp) environments were subject to this kind of observation.
An example of the kinematics observation is presented in Table 4.13 below.

TABLE 4.13: Kinematics observation representation.

Vehicle	x	y	Vx	Vy
Self-driving car	2.0	3.5	16.0	0
Vehicle 1	10.0	5.0	12.0	0.2
Vehicle 2	-13.0	1.0	10.5	0
....	....	....	....	....
Vehicle V	20.3	11.0	19.0	0.5

An expanded version of this observation, known as "KinematicsGoal", was utilized for the parking environment. Instead of the positions of the vehicles the car ought to evade, as in the highway scenario, this version includes the location of the destination the car should attain.

4.2.2.2.2 Time To Collision Observation: This kind of observation is employed in roundabout environments and is represented by a VxLxH list which contains information on the anticipated time to collide with other vehicles.

Where V is the self-driving car speed values, L is the number of lines, and H is the number of discounted times values with a step of one second. The prediction is represented over as one hot encoding over the VxLxH array.

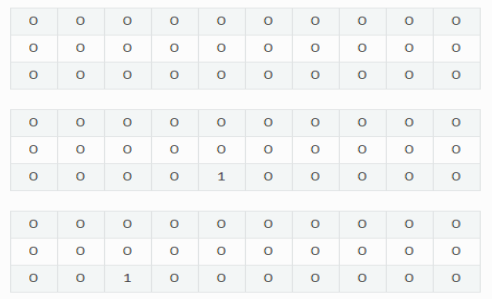


FIGURE 4.18: Time to Collision Observation.

As an example of this observation from the manual (Figure 4.18), we use the following three AV speed values: 15 m/s, 20 m/s, and 25 m/s, L = 3, and H = 10 to assume a vehicle traveling at a speed of 30 m/s and being 15 meters away from the autonomous vehicle.
The projected time to collision for each of the specified speeds is, in turn, $infty$, 5 seconds, and 2.5 seconds.

4.2.2.3 Actions Space:

The self-driving car selects an action for moving through the simulator's varied environments at each time step from one of the five given actions listed in the table below.

TABLE 4.14: Actions Space.

Action	Description
0	Move to left lane
1	Stay in lane
2	Move to right lane
3	Accelerate
4	Decelerate

4.2.3 Rewards:

The agent will receive a reward using the following formula for every chosen action in a particular state of an environment:

$$\frac{R(s,a) = a \times (V - Vmin)}{(Vmax - Vmin) - b \times collision} \tag{4.8}$$

Where V stands for the vehicle's current speed, Vmin and Vmax stand for the minimum and maximum speed, respectively.

Despite the fact that in a parking environment the desired target destination must be provided and the velocity is unimportant, it was replaced by the following weighted p-norm between the goal state and agent state:

$$R(s,a) = -||s - s_g||^p_{W,p} - b \times collision \tag{4.9}$$

where:

$$s = [x, y, v_x, v_y, cos\phi, sin\phi] \tag{4.10}$$

$$s_g = [x_g, y_g, 0, 0, cos\phi_g, sin\phi_g] \tag{4.11}$$

$$||x||_w, p = (\sum_i |W_i x_i|^p)^{1/p} \tag{4.12}$$

Instead of using a Euclidean norm, a p-norm is employed to create a goal with a smaller reward spike.

4.2.4 Experimentation:

For this paper, in order to demonstrate the viability and effectiveness of the suggested Alternative bidirectional Q network method, five simulation scenarios were used for experimental objectives as indicated above, while adopting four metrics from the ones previously defined in the first article and they are : MSE, Accuracy, Reward, and speed.

Next, the plots and detailed tables that contain the results of comparison between the proposed method and Deep Q Learning algorithm alongside Deep SARSA will be provided to show how effective the new extension was in all environments.

4.2.4.1 Highway Results:

The main environment treated "Highway" results are provided below.

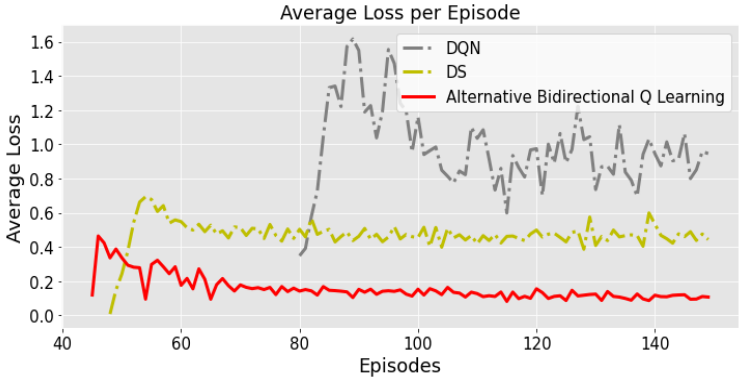


FIGURE 4.19: Avg Loss Comparison

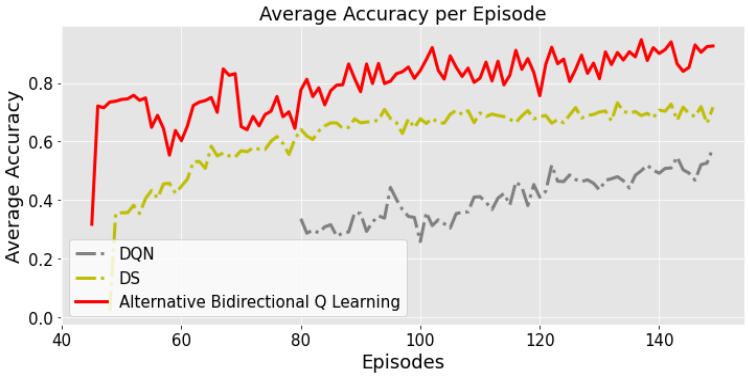


FIGURE 4.20: Avg Accuracy Comparison

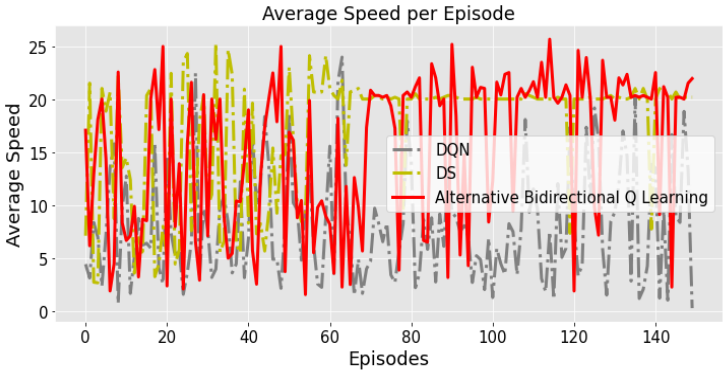


FIGURE 4.21: Avg Speed Comparison

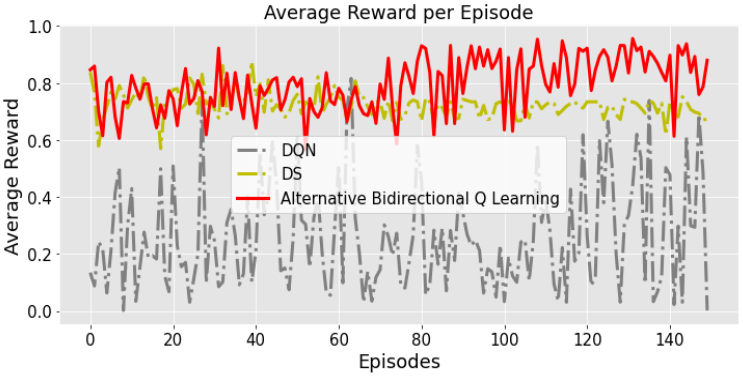


FIGURE 4.22: Total Reward Comparison

TABLE 4.15: Detailed Results of Highway Scenario.

Algorithms	Average Loss	Average Accuracy	Average Speed(m/s)	Average Total Reward
DQN	0.9144	0.4663	8.9955	0.30921
DS	0.4645	0.6918	8.995	0.7122
Alternative Bidirectional QL	**0.18758**	**0.87246**	**19.6718**	**0.8010**

4.2.4.2 Merge:

Below are the plots showing the results in the merge environment treated, followed by Table 4.16 holding the detailed results of this scenario.

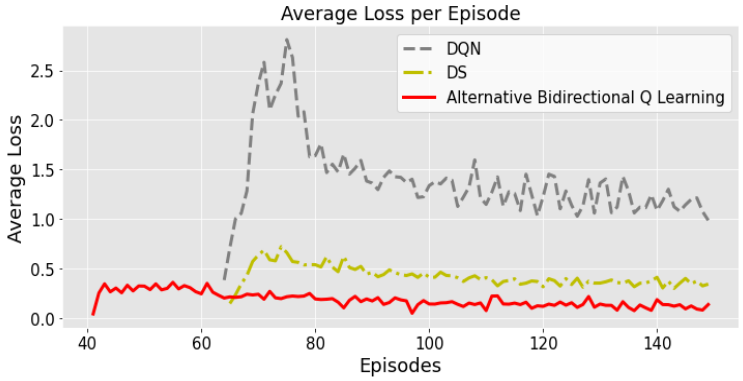


FIGURE 4.23: Comparison of Average Loss Values per episode

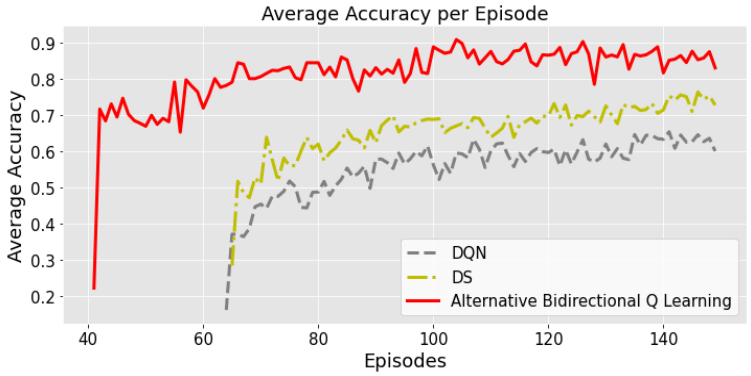


FIGURE 4.24: Comparison of Average Accuracy Values per episode

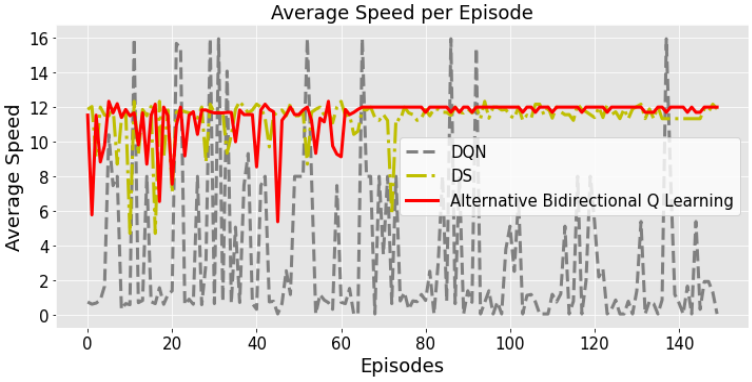


FIGURE 4.25: Comparison of Average Speed Values per episode

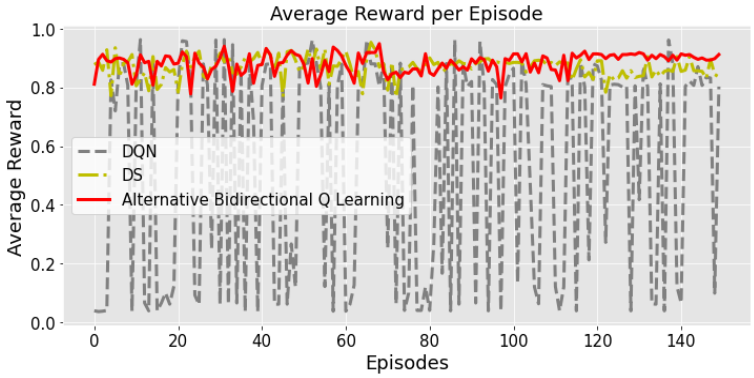


FIGURE 4.26: Comparison of Total Rewards per episode

TABLE 4.16: Detailed Results of Merge Scenario.

Algorithms	Average Loss	Average Accuracy	Average Speed(m/s)	Average Total Reward
DQN	1.203	0.6047	2.1790	0.5847
DS	0.3595	0.7061	11.6337	0.8561
Alternative Bidirectional QL	**0.13051**	**0.80616**	**11.9252**	**0.9232**

4.2.4.3 Roundabout:

Next, we show how the new extension handled the roundabout environment as well with same efficiency as previous environments.

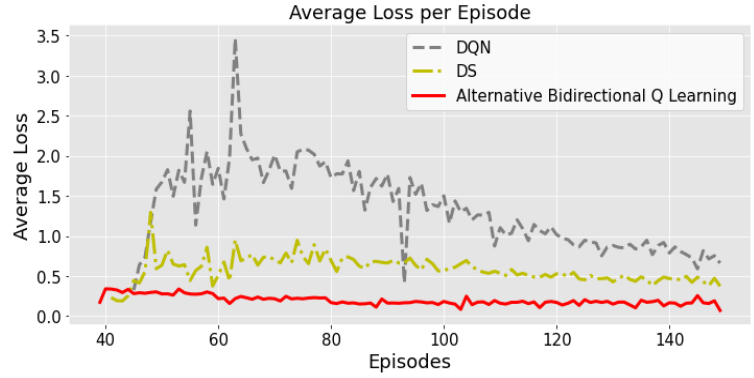


FIGURE 4.27: Avg Loss Comparison

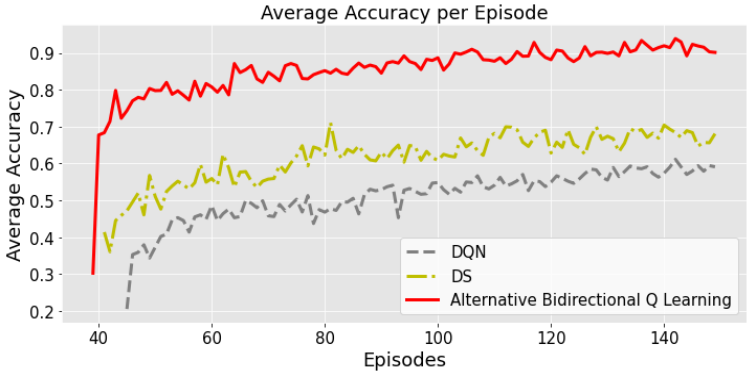


FIGURE 4.28: Avg Accuracy Comparison

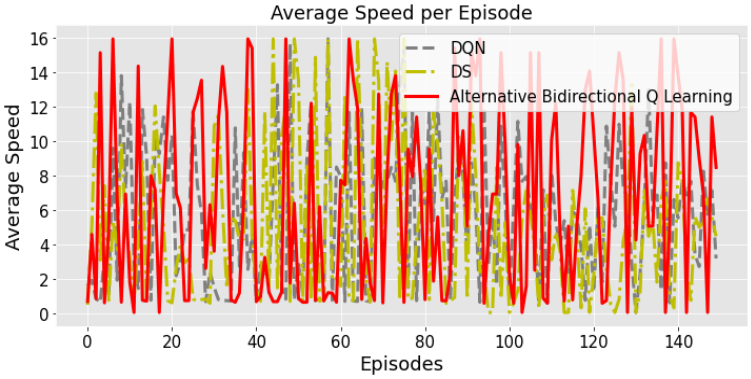


FIGURE 4.29: Avg Speed Comparison

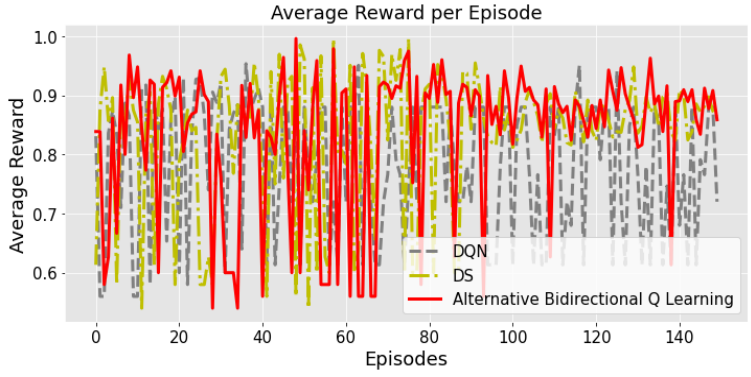


FIGURE 4.30: Total Reward Comparison

TABLE 4.17: Detailed Results of Roundabout Scenario.

Algorithms	Average Loss	Average Accuracy	Average Speed(m/s)	Average Total Reward
DQN	0.9011	0.5692	5.4336	0.7732
DS	0.4779	0.6714	3.9531	0.8652
Alternative Bidirectional QL	**0.16534**	**0.82842**	**7.9770**	**0.8746**

4.2.4.4 Parking:

The final environment represents a parking scenario that differs a lot from the previous ones and the results are shown below.

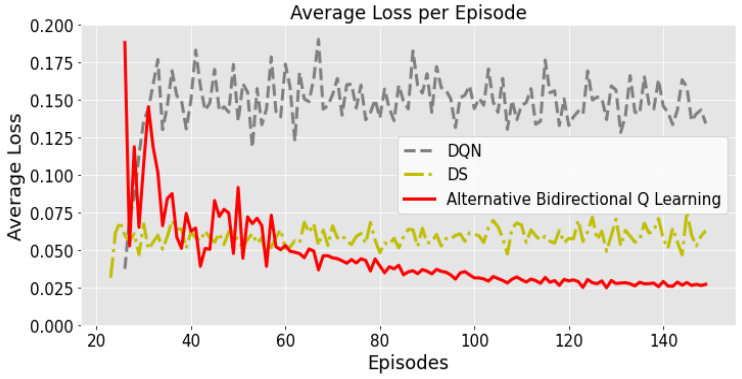


FIGURE 4.31: Avg Loss Comparison

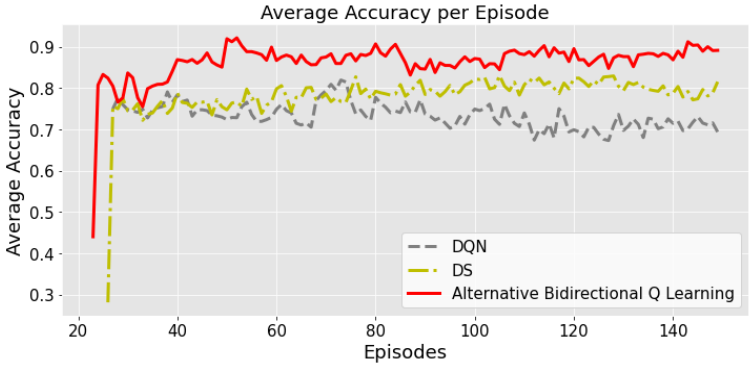


FIGURE 4.32: Avg Accuracy Comparison

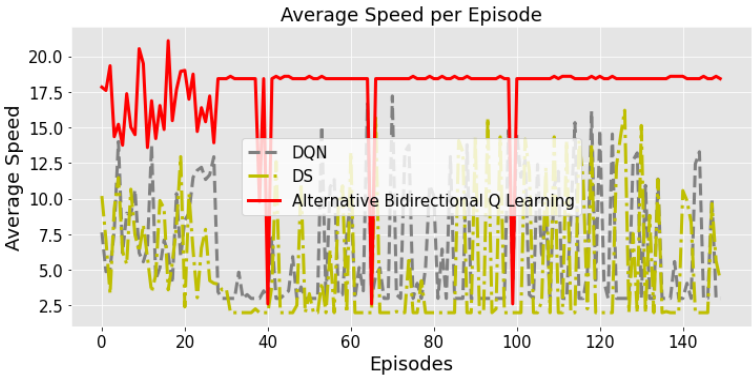


FIGURE 4.33: Avg Speed Values Comparison

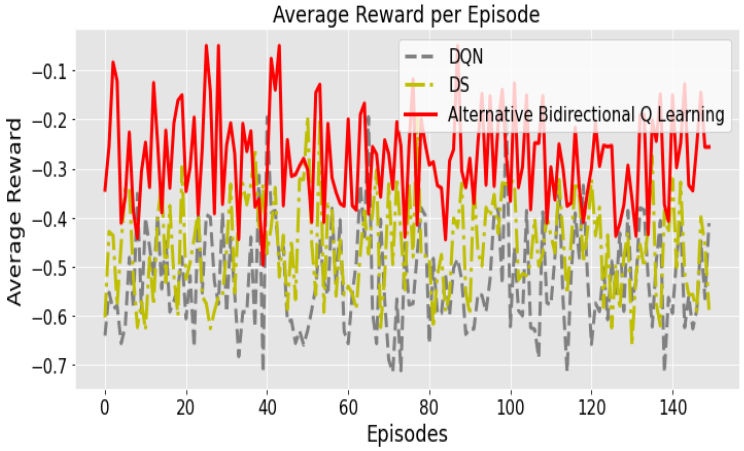


FIGURE 4.34: Total Rewards Comparison

TABLE 4.18: Detailed Results of Parking Scenario.

Algorithms	Average Loss	Average Accuracy	Average Speed(m/s)	Average Total Reward
DQN	0.1475	0.7067	8.7531	-0.5191
DS	0.0599	0.8038	10.8434	-0.4671
Alternative Bidirectional QL	**0.02805**	**0.8812**	**16.4770**	**-0.2937**

4.2.4.5 Complexity Analysis:

Experiments related to the complexity of the new extension presented was made as well in this paper, where two predefined libraries were used in the calculation of the CPU and memory load across the various environments.

Results are presented in Tables 4.19 and 4.20 below:

TABLE 4.19: CPU User Time Results.

Environments	Highway	Merge	Roundabout	Parking
DQN	30 min 42 s	11 min 21 s	23 min 05 s	18 min 52 s
DS	11 min 22 s	8 min 14 s	11 min 35 s	10 min 12 s
The proposed Algorithm	11 min 49 s	8 min 25 s	11 min 22 s	9 min 38 s

TABLE 4.20: Peak Memory Results.

Environments	Highway	Merge	Roundabout	Parking
DQN	556.84 MiB	542.14 MiB	551.28 MiB	554.69 MiB
DS	532.65 MiB	526.46 MiB	529.95 MiB	530.41 MiB
The proposed Algorithm	546.22 MiB	542.62 MiB	540.37 MiB	539.18 MiB

4.2.4.6 Robustness Test:

The ability of models to learn rules that enable them to achieve an acceptable level of adaptation to unanticipated changes in environments beyond the training phase is a key characteristic of reinforcement learning algorithms.

Due to the many variations this environment offers, the highway scenario was considered the main research environment in this matter's numerous tests, to showcase the durability of the suggested method.

We evaluated the suggested algorithm agent to see how well it could adapt to different alterations associated with the lanes count and obstructions (cars) on the highway following training on a highway with four lanes.

We compared numerous possible highway designs using the accuracy measure as the comparison parameter.

TABLE 4.21: Average Accuracy Values per configuration.

Vehicles / **Lanes**	**50**	**60**	**70**	**80**	**90**	**100**
2 lanes	0.85146	0.86462	0.8596	0.8262	0.8281	0.7913
4 lanes	0.8821	0.89013	0.86442	0.8361	0.8421	0.82016
6 lanes	0.86165	0.84213	0.84131	0.8521	0.83244	0.82882
8 lanes	0.88413	0.88663	0.87963	0.82613	0.8413	0.8236
10 lanes	0.89135	0.88331	0.86994	0.87236	0.87119	0.8689
12 lanes	0.89019	0.892	0.87662	0.88333	0.84087	0.85009

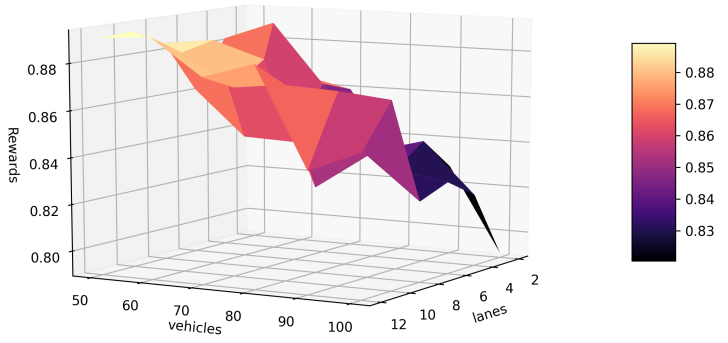


FIGURE 4.35: Robustness Analysis.

4.2.5 Discussion and Conclusion:

Decision Making is of the utmost importance and utility for autonomous vehicles.

A good navigation algorithm needs to be capable of making accurate decisions under a variety of circumstances and be highly adaptable to unforeseen events.

The novel policy of choosing actions based on reward values of three sequential time-steps (t-1, t, t+1) was able to increase the harmony between exploration and exploitation using the previously outlined method. As a result, convergence speed increased, and performance peaked after only a short period of training. Unlike DRL techniques, which frequently call for extensive training sessions to achieve acceptable decision-making level of precision and prevent errors.

The four scenarios—highway, merging, roundabout, and parking—that are utilized to show the viability of the suggested schemes all showed stability in performance and learning, which supported the vital significance of the updated Q value and neural model fitting policy.

Also, we confirmed the capability of the proposed method in handling unexpected changes in its surroundings following the test conducted on a prior trained agent in various configurations of the highway environment, and we proved this way that the new policies are robust.

A cost-effective improvement is desired because updating DRL methods typically leads to increased complexity, which can increase memory and CPU power consumption.
Therefore, We put these notions to the test, and the results revealed that employing the suggested strategy compared to previous models resulted in a significant improvement in CPU user time. While not ideal, memory utilization was tolerable given the performance levels reached.

Overall, the suggested technique performed well in numerous test scenarios, and it is potentially applicable to other situations with similar properties (discrete action spaces, continuous or discrete state spaces).

4.3 Chapter Conclusion:

In this chapter we introduced two novel deep reinforcement learning extensions being: "Harmonic SK Deep SARSA" and "Alternative Bidirectional Q Learning", derived from the Deep SARSA and Deep Q Learning algorithm respectively.

Various tests and experiments were made related to the performance, stability, complexity, robustness, and adaptability of these two extensions were conducted in both papers and the results were shown in both algorithms sections.

The tests were made in a simulation environment developed for the purpose of constructing new reinforcement learning methods, while comparing both of the new extensions to other benchmark models.

Good results were achieved with both our new algorithms in most test environments and scenarios, and this was confirmed through several state of the art metrics.

Even when we enhanced several sides like convergence rate, stability, adaptability to unexpected changes in the agents surroundings; various limitations are still available to address in future works.

Chapter 5

Conclusion And Perspectives

"You never have
an idea of what you might accomplish.
All that you do is you pursue
a question
and see where it can lead."

Jonas Salk

5.1 Conclusion:

Autonomous systems in general, and Autonomous vehicles field precisely, saw a major leap over the last decades.

Many advancements were made, and others are still in progress in all sides related to the development of this technology, whether in hardware or software, in perception or planning, etc.

Although, in order to fully take advantage of the autonomous vehicles technology, many limitations must be faced and more upgrades must be done.

In this thesis we focused on the software side of things, and mainly the decision making system in autonomous vehicles which is an important part that is responsible of deciding the actions that will be taken by the self-driving vehicle in all times.

The decision making field itself has known lot of improvements over the years, where it started from defining simple rules till reaching the highly effective field of Artificial Intelligence named Machine Learning, and mainly its most promising paradigm known as Reinforcement Learning integrated with Deep Learning which represents another famous and effective concept in a lot of AI fields.

We investigated the efficiency of RL and DL through this work and proposed new extensions of two of the most used and well-known deep reinforcement learning algorithms being: Deep Q Learning, and Deep SARSA.

Through our novel methods we managed to enhance performance, stability and robustness of the base models and confirmed the effectiveness of deep reinforcement learning in handling autonomous decision making of agents in various simulated environments.

5.2 Perspectives:

Through this thesis we were able to upgrade multiple aspects related to taking decisions for autonomous vehicles, and this was confirmed by numerous experiments that we made over two different papers.

Nonetheless, multiple directions were not considered in this work that can be the base for our future works.

• Extend the proposed extensions to handle continuous action spaces which represent a major challenge in this field, where lot of environments and situations require more specific and detailed actions to handle all the possibilities.

• Connection between vehicles and controlling multiple agents through the proposed methods were not tested, and these are interesting topics since this technology will result in roads filled with self-driving vehicles.

• Ethics and safety of passengers are another direction we can go to in our works, because it represents a needed matter to discuss and handle before we can take full advantage of this new technology.

• Experiencing the real world through the novel methods and upgrading them for this matter is desirable.

Publications List

Journal Papers:

Rais, M. S., Boudour, R., Zouaidia, K., and Bougueroua, L. (2022). Decision making for autonomous vehicles in highway scenarios using Harmonic SK Deep SARSA. Applied Intelligence, 1-18.

Rais, M. S., Zouaidia, K., and Boudour, R. (2022). Enhanced decision making in multi-scenarios for autonomous vehicles using alternative bidirectional Q network. Neural Computing and Applications, 1-16.

Zouaidia, K., Rais, M. S., Ghanemi, S. (2023). Weather forecasting based on hybrid decomposition methods and adaptive deep learning strategy. Neural Computing and Applications, 1-16.

Zouaidia, K., Ghanemi, S., Rais, M. S., Bougueroua, L., and Katarzyna, W. W. (2021). Hybrid intelligent framework for one-day ahead wind speed forecasting. Neural Computing and Applications, 33(23), 16591-16608.

Zouaidia, K., Ghanemi, S., Rais, M. S. (2021). Hourly Wind Speed Forecasting Using FFT-Encoder-Decoder-LSTM in South West of Algeria (Adrar) (Extended conference paper). International Journal of Informatics and Applied Mathematics, 4(1), 72-83.

Conference Papers:

Rais M.S., Boudour, R., and Zouaidia, K. (2020).Avoiding obstacles in a road with a maze structure using reinforcement learning methods in Third conference on informatics and applied mathematics.

Zouaidia, K., Ghanemi, S., Rais, M. S. (2020) Hourly Wind Speed Forecasting using FFT-Encoder-Decoder-LSTM in South West of Algeria (Adrar) in Third Conference on Informatics and Applied Mathematics, IAM's 2020.

Zouaidia, K., Ghanemi, S., Rais, M. S. (2020, December). Wind speed forecasting based on discrete wavelet transform, moving average method and gated recurrent Unit. In International Conference in Artificial Intelligence in Renewable Energetic Systems (pp. 71-78). Springer, Cham.

Khouloud, Z., Saber, R. M. (2021, September). Multi-Step Wind Speed Forecasting Based on Hybrid Deep Learning Model and Trailing Moving Average Denoising Technique. In 2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI) (pp. 1-5). IEEE.

Bibliography

[1] Watson D. et al. Autonomous systems. *Johns Hopkins APL technical digest*, 26(4):368–376, 2005.

[2] Wang L. C. et al. An autonomous system view to apply machine learning. *IEEE International Test Conference*, pages 1–10, 2018.

[3] Reich Y. et al. Evaluating machine learning models for engineering problems. *Artificial Intelligence in Engineering*, 13(3):257–272, 1999.

[4] Zouaidia K. et al. Hybrid intelligent framework for one-day ahead wind speed forecasting. *Neural Computing and Applications*, 33(23):16591–16608, 2021.

[5] Basso K. et al. Reverse engineering of regulatory networks in human b cells. *IET Intelligent Transport Systems*, 37(4):382–390, 2005.

[6] Anderson J.A. et al. An introduction to neural networks. *MIT press*, 1995.

[7] Miller W. et al. Neural networks for control. 1995.

[8] Chen J. et al. Integrations between autonomous systems and modern computing techniques: a mini review. *Sensors.*, 19(18):3897, 2019.

[9] Sutton R.S. et al. Reinforcement learning: An introduction. *MIT press.*, 2018.

[10] Kaelbling L.P. et al. Reinforcement learning: A survey. *Journal of artificial intelligence research.*, 4:237–285, 1996.

[11] Mnih V. et al. Human-level control through deep reinforcement learning. *nature.*, 518(7540):529–533, 2015.

[12] Arulkumaran K. et al. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine.*, 34(6):26–38, 2017.

[13] Shteingart H. et al. Reinforcement learning and human behavior. current opinion. *Neurobiology.*, 25:93–98, 2014.

[14] Köppen M. et al. The curse of dimensionality. *5th online world conference on soft computing in industrial applications.*, 1:4–8, 2000.

[15] Mindy Support. 5 main reasons why fully autonomous cars are not on the roads yet?, 2020. URL https://mindy-support.com/news-post/5-main-reasons-why-fully-autonomous-cars-are-not-on-the-roads-yet/.

[16] National safety council injury facts. URL https://injuryfacts.nsc.org/.

[17] Maurer M. et al. *Autonomous driving: technical, legal and social aspects.* Springer Nature, 2016.

[18] Russell P. How autonomous vehicles will profoundly change the world. *Epub ahead of print.*, 2015.

[19] Urmson C. et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of field Robotics.*, 25(8):425–466, 2009.

[20] Gold C. *Modeling of Take-Over Performance in Highly Automated Vehicle Guidance*. PhD thesis, 01 2017.

[21] United States Department For Safety. Automated vechiles for safety. URL https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

[22] SAE International. Sae levels of driving automation™ refined for clarity and international audience. URL https://www.sae.org/blog/sae-j3016-update.

[23] Rajasekhar M.V. et al. Autonomous vehicles: The future of automobiles. *IEEE International Transportation Electrification Conference.*, pages 1–6, 2015.

[24] Yang J. et al. In-vehicle technology for self-driving cars: Advantages and challenges for aging drivers. *International Journal of Automotive Technology.*, 15(2):333–340, 2014.

[25] Martinez-Diaz M. et al. Autonomous vehicles: theoretical and practical challenges. *Transportation Research Procedia.*, 33:275–282, 2018.

[26] Li L. et al. Humanlike driving: Empirical decision-making system for autonomous vehicles. *IEEE Transactions on Vehicular Technology.*, 67(8): 6814–6823, 2018.

[27] HatenaBlog. The challenges of making decisions in autonomous driving., 2021. URL https://tech.tier4.jp/entry/2021/03/24/160000#:~:text=For%20autonomous%20vehicles%2C%20decision%20making,be%20executed%20by%20the%20controller.

[28] Maes P. Modeling adaptive autonomous agents. *Artificial life.*, $1(1_2)$: $135 - -162, 1993$.

[29] Jadbabaie A. et al. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *Transactions on automatic control.*, 48(6):988–1001, 2003.

[30] Montemerlo M. et al. Winning the darpa grand challenge with an ai robot. *AAAI.*, pages 982–987, 2006.

[31] Institute of Measurement and Control Systems. Decision-making and motion planning. URL https://www.mrt.kit.edu/english/Decision-Making_and_Motion_Planning.php.

[32] Zhu D. et al. A new algorithm based on dijkstra for vehicle path planning considering intersection attribute. *IEEE Access.*, 9:19761–19775, 2021.

[33] LaValle S.M. et al. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics*, pages 303–307, 2001.

[34] Werling M. et al. Optimal trajectory generation for dynamic street scenarios in a frenet frame. *IEEE International Conference on Robotics and Automation*, pages 987–993, 2010.

[35] Damerow F. et al. Risk-based driver assistance for approaching intersections of limited visibility. *IEEE International Conference on Vehicular Electronics and Safety*, pages 178–184, 2017.

[36] Gonzalez H. et al. Motion planning with visibility constraints: Building autonomous observers. *Robotics Research*, pages 95–101, 1998.

[37] Hwang Y.K. et al. Gross motion planning—a survey. *ACM Computing Surveys (CSUR)*, 24(3):219–291, 1992.

[38] Otterlo M.V. et al. Reinforcement learning and markov decision processes. *Reinforcement learning*, pages 3–42, 2012.

[39] Spaan M.T. et al. Partially observable markov decision processes. *Reinforcement learning*, pages 387–414, 2012.

[40] Ben-David S. et al. Online learning versus offline learning. *Machine Learning*, 29:45–63, 1997.

[41] Brechtel S. et al. Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps. *17th international IEEE conference on intelligent transportation systems (ITSC)*, pages 392–399, 2014.

[42] Kamkarian P. et al. Robotic offline path planning. *Southern Illinois University at Carbondale*, 2015.

[43] Zhan X. et al. Model-based offline planning with trajectory pruning. *arXiv preprint arXiv:2105.07351*, 2021.

[44] Ulbrich S. et al. Probabilistic online pomdp decision making for lane changes in fully automated driving. *16th International IEEE Conference on Intelligent Transportation Systems*, pages 2063–2067, 2013.

[45] Bai et al. Intention-aware online pomdp planning for autonomous driving in a crowd. *IEEE International Conference on Robotics and Automation*, pages 454–460, 2015.

[46] Karimi S. et al. Monte carlo tree search and cognitive hierarchy theory for interactive-behavior prediction in fast trajectory planning and automated lane change. *International Conference on Computers and Games*, 1(1), 2021.

[47] Mern J. et al. Bayesian optimized monte carlo planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):11880–11887, 2021.

[48] Sonu E. et al. Exploiting hierarchy for scalable decision making in autonomous driving. *IEEE Intelligent Vehicles Symposium*, pages 2203–2208, 2018.

[49] LeCun Y. et al. Deep learning. *nature*, 521(7553):436–444, 2015.

[50] Ij H. et al. Statistics versus machine learning. *Nat Methods*, 15(4):233, 2018.

[51] Janiesch C. et al. Machine learning and deep learning. *Electron Markets*, 31: 685–695, 2021.

[52] Kriegeskorte N. et al. Neural network models and deep learning. *Current Biology*, 29(7):R231–R236, 2019.

[53] wikipedia. Neuron. URL https://en.wikipedia.org/wiki/Neuron.

[54] Dayhoff J.E. eural network architectures: an introduction. *Van Nostrand Reinhold Co*, 1990.

[55] Priddy K.L. Artificial neural networks: an introduction. *SPIE press*, 68, 2005.

[56] Rukshan P. One hidden layer (shallow) neural network architecture, 2021. URL https://rukshanpramoditha.medium.com/one-hidden-layer-shallow-neural-network-architecture-d45097f649e6.

[57] cperales. Simple-neural-netwok, 2019. URL https://github.com/cperales/Simple-Neural-Netwok.

[58] Sagar S. Epoch vs batch size vs iterations, 2017. URL https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

[59] Thomas A.J et al. Two hidden layers are usually better than one. *International conference on engineering applications of neural networks*, pages 279–290, 2017.

[60] Sparsh G. The 7 most common machine learning loss functions, 2022. URL https://builtin.com/machine-learning/common-loss-functions.

[61] Data_Science_Today. Loss function, 2018. URL https://www.datasciencetoday.net/images/lossfunc.png.

[62] Szandała T. Review and comparison of commonly used activation functions for deep neural networks. *Bio-inspired neurocomputing*, pages 203–224, 2021.

[63] Samuel A.L. Machine learning. *The Technology Review*, 62(1):42–45, 1959.

[64] Cunningham P. et al. Supervised learning. *Machine learning techniques for multimedia*, pages 21–49, 2008.

[65] Hastie T. et al. Unsupervised learning. *The elements of statistical learning*, pages 485–585, 2003.

[66] Kumar A. et al. Machine learning: classification and regression. 1970.

[67] Berrar D. Cross-validation. 2019.

[68] Wikipedia. Reinforcement. URL https://en.wikipedia.org/wiki/Reinforcement.

[69] Bellman R. A markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957.

[70] Silver D. et al. Deterministic policy gradient algorithms. *International conference on machine learning*, pages 387–395, 2014.

[71] Littman M.L. Value-function reinforcement learning in markov games. *Cognitive systems research*, 2(1):55–66, 2001.

[72] Barto A.G. Temporal difference learning. *Scholarpedia*, 2(11):1604, 2007.

[73] Metropolis N. et al. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[74] Bellman R. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[75] BandaiNamco. Storm series video games, 2022. URL https://en.bandainamcoent.eu/.

[76] Sami S. Epsilon-greedy algorithm in reinforcement learning, 2022. URL https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/.

[77] Hristo H. Off-policy vs. on-policy reinforcement learning, 2022. URL https://www.baeldung.com/cs/off-policy-vs-on-policy.

[78] AlindGupta. Sarsa reinforcement learning, 2021. URL https://www.geeksforgeeks.org/sarsa-reinforcement-learning/?ref=gcse.

[79] Sandro S. Introduction to deep learning - from logical calculus to artificial intelligence. *Undergraduate Topics in Computer Science*, pages 1–16, 2018.

[80] Rais M.S. et al. Decision making for autonomous vehicles in highway scenarios using harmonic sk deep sarsa. *Applied Intelligence*, pages 1–18, 2022.

[81] Zong W.G. et al. A new heuristic optimization algorithm: Harmony search. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 78:60–68, 2001.

[82] Leurent E. 'highway-env' an environment for autonomous driving decision-making, 2018. URL https://github.com/eleurent/highway-env.

[83] Rais M.S. et al. Enhanced decision making in multi-scenarios for autonomous vehicles using alternative bidirectional q network. *Neural Computing And Applications*, pages 1–16, 2022.