KADİR HAS UNIVERSITY SCHOOL OF GRADUATE STUDIES
PROGRAM OF ELECTRONICS ENGINEERING

AUTONOMOUS VEHICLE CONTROL USING REINFORCEMENT LEARNING

HÜMA BOZKURT

MASTER'S THESIS

ISTANBUL, DECEMBER, 2020

Hüma Bozkurt M. Sc. Thesis 2020

AUTONOMOUS VEHICLE CONTROL USING REINFORCEMENT LEARNING

HÜMA BOZKURT

MASTER’S THESIS

Submitted to the School of Graduate Studies of Kadir Has University in partial fulfillment of the requirements for the degree of Master’s in the Program of Electronics Engineering

ISTANBUL, DECEMBER, 2020

DECLARATION OF RESEARCH ETHICS / METHODS OF DISSEMINATION

I, HÜMA BOZKURT, hereby declare that;

- this master's thesis is my own original work and that due references have been appropriately provided on all supporting literature and resources;
- this master's thesis contains no material that has been submitted or accepted for a degree or diploma in any other educational institution;
- I have followed *Kadir Has University Academic Ethics Principles prepared in accordance with the "The Council of Higher Education's Ethical Conduct Principles"*

In addition, I understand that any false claim in respect of this work will result in disciplinary action in accordance with University regulations.

Furthermore, both printed and electronic copies of my work will be kept in Kadir Has Information Center under the following condition as indicated below :
The full content of my thesis/project will be accessible from everywhere by all means.

HÜMA BOZKURT
03/12/2020

KADİR HAS UNIVERSITY SCHOOL OF GRADUATE STUDIES

ACCEPTANCE AND APPROVAL

This work entitled **AUTONOMOUS VEHICLE CONTROL USING REINFORCEMENT LEARNING** prepared by **HÜMA BOZKURT** has been judged to be succesful at the defense exam held on 03/12/2020 and accepted by our jury as master's thesis.

APPROVED BY:

Assoc. Prof. Dr. Atilla ÖZMEN (Advisor) …………………………

Prof. Dr. Tülay YILDIRIM (Co-Advisor) …………………………

Asst. Prof. Dr. Baran TANDER …………………………

Asst. Prof. Dr. Ertuğrul SAATÇİ …………………………

I certify that the above signatures belong to the faculty members named above.

…………………………

Dean of School of Graduate Studies

DATE OF APPROVAL: 03/12/2020

TABLE OF CONTENTS

AUTONOMOUS VEHICLE CONTROL USING REINFORCEMENT LEARNING

ABSTRACT

Autonomous vehicles have become an important research topic where artificial intelligence is applied. As the research increases, by means of the applications of artificial intelligence algorithms in different areas, enable the working mechanisms of the systems to become more optimal due to the change of factors such as human power, time, energy and control. It has been observed that deep learning and machine learning algorithms have advantages and disadvantages in different situations and conditions. Since deep learning algorithms require large amounts of data, studies on the reinforcement learning model based on the experience from the environment and based on the reward-punishment system have recently concentrated and some striking results have been obtained. Reinforcement learning is considered a powerful AI paradigm that can be used to teach machines through interaction with the environment and learning from their mistakes.

In this thesis, an environment was created based on a two-dimensional vehicle scenario created using a pyglet simulation tool. A comparative simulation study of different reinforcement learning algorithms such as Q-Learning, SARSA and Deep Q-Network (DQN) is presented on this environment. While making this comparison, a certain learning criterion was added, and also, parameters such as epsilon value, step number were changed, and changes in training and test stages were analyzed. For this study, the actors (agent, sensor, obstacles etc.) provided by the simulator program were supported. Through the feedback provided by the sensors, the reinforcement learning agent trains himself on the basis of these algorithms and determines a movement strategy to explore the environment limited to a specific area.

Keywords: Autonomous vehicle, Reinforcement learning, SARSA, Q-learning, Deep

Q-learning

OTONOM BİR ARACIN PEKİŞTİRMELİ ÖĞRENME İLE KONTROLÜ

ÖZET

Otonom araçlar, yapay zekanın uygulandığı önemli bir araştırma konusu haline geldi. Araştırmalar yoğunlaştıkça yapay zeka algoritmalarının farklı alanlardaki uygulamaları ile insan müdahalesi, zaman, enerji, kontrol gibi faktörlerin değişimine bağlı olarak sistemlerin çalışma mekanizmalarının ve hedeflerine varış doğrultularının daha optimal bir sonuca ulaştırdığı görülmektedir. Derin öğrenme ve makine öğrenme algoritmalarının farklı durum ve koşullarda avantaj ve dezavantajları olduğu görülmüştür. Derin öğrenme algoritmaları yüksek miktarda veriye ihtiyaç duyduğundan, çevreden gelen deneyimlerle hareket eden ve ödül-ceza sistemine dayanan takviye öğrenme modeli üzerine çalışmalar yakın zamanda yoğunlaşmıştır ve bazı çarpıcı sonuçlar elde edilmiştir. Takviye öğrenimi, çevre ile etkileşim ve hatalarından öğrenme yoluyla makineleri öğretmek için kullanılabilecek güçlü bir AI paradigması olarak kabul edilir.

Bu tez çalışmasında pyglet simülasyon aracı kullanılarak oluşturulmuş iki boyutlu bir araç senaryosu baz alınarak bir ortam oluşturulmuştur. Bu ortam üzerinde Q-Learning, SARSA ve Deep Q-Network (DQN) gibi farklı pekiştirici öğrenme algoritmalarının karşılaştırmalı bir simülasyon çalışması sunulmaktadır. Bu karşılaştırmayı yaparken belirli bir öğrenme kriteri eklenmiştir ve ayrıca epsilon değeri, step sayısı gibi parametreler değiştirilerek training ve test aşamalarındaki değişiklikler analiz edilmiştir. Bu çalışma için simülatör programı tarafından sağlanan aktörler (ajan, sensör, engeller vb.) ile desteklenmiştir. Sensörler tarafından sağlanan geri bildirim yoluyla, pekiştirmeli öğrenme ajanı, kendini bu algoritmaları baz alarak eğiterek belirli bir alanla sınırlı ortamı keşfetmek için bir hareket stratejisi belirler.

Anahtar Sözcükler: Otonom araç, Pekiştirmeli öğrenme, Q-öğrenme algoritması, SARSA, Derin Q-öğrenme

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Associate Professor Atila Özmen for his contribution in the research and writing process while preparing my thesis.

I would like to thank to my thesis co-supervisor Professor Tülay Yıldırım for her feedback on my research work. Her vision motivated me a lot through my research.

I would also like to thank the valuable scientists in different parts of the world who have worked on this subject and shared their ideas with me during my research process.

Last but not the least; I would like to express my thankfulness to my dear family who supported me all my life and empowered me with spiritual support throughout my studies.

To my dear family

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABBREVIATION

AI	: Artificial Intelligence
ANN	: Artificial Neural Network
DDAC	: Deep Deterministic Actor Critic
DL	: Deep Learning
DNN	: Deep Neural Network
DQL	: Deep Q-Learning
DQN	: Deep Q-Network
DP	: Dynamical Programming
GPI	: Generalized Policy Iteration
GPS	: Global Positioning System
LIDAR	: Light Detection and Ranging
MC	: Monte Carlo
MDP	: Markov Decision Processes
MSE	: Minimum Square Error
RL	: Reinforcement Learning
PCA	: Principal Component Analysis
PI	: Policy Iteration
POMDP	: Partially Observable Markov Decision Processes
TDM	: Temporal Difference Methods

1. INTRODUCTION

Recently autonomous vehicles have become a scientific field of study, which is increasingly concentrated. A vision about self-driving cars was first propounded in 1918. Later on, in order to minimize dependence on the driver in autonomous vehicles, the initiatives started with communication intensity, an inventor, Francis Houdina introduced the first radio-controlled driverless vehicle (1925). Then the first car with this invention was Chrysler Imperial. This cruise control happened with that rotational speed was calculated by speedometer. The actuator was electric motor adjusted by throttle position (Imperial, 1958). Afterwards, Carnegie Mellon's NAVLAB vehicle was being demonstrated to perform lane-following using camera images then the focus is on algorithms that can perform complex movements that human beings can make today (Thorpe et al., 1988).

With autonomous vehicles whose speed of development has increased more since the 2000s, it is aimed to prevent possible collisions with the attention disturbance of the drivers, to perform multiple tasks at the same time, to strengthen the control mechanism, to reduce the repetitive boring tasks and the dependence on similar manpower in various tasks. Besides, it is aimed to maximize efficiency and optimization in daily life, transportation. In line with these requirements, driver monitoring system, cruise control, automated parking, rear collision warning, blind spot indicator, traffic sign recognition and more applications have emerged.

Autonomous vehicle projects evaluate sensor data, analyses and provide vehicle movements with machine learning. Data can be received from any possible source through sensors such as LIDAR, cameras, radars, sonar, GPS. In order for autonomous driving to take place, some methods are needed to create a specific software framework as well as specific hardware components are required.

It basically can be divided into five competences;

- Localization: The first basic element of the vehicle's interaction with its surroundings is to know its position this is the basis for the autonomous vehicle to realize its purpose.
- Perception: This is how the autonomous vehicles understand their environment. These approaches include a lot of components for recognizing driving-relevant objects. For example, cars, lanes, pedestrians, traffic signs, traffic lights etc.
- Prediction: Autonomous vehicles predict the behaviour of people and vehicles in their environment. It must also be able to build internal models that predict the future states of the environment.
- Path Planning: It is called the route that the autonomous vehicle will follow.
- Control: In the control part, the steering direction, speed and braking condition of the vehicle are set.

According to SAE (Society of Automotive Engineers) International, at present time autonomous vehicles divided by 6 levels to Level 5 from Level 0 as J3 016. (SAE International, 2014)

- Level 0 (No Automation) - All major systems are controlled by humans.
- Level 1 (Driver Assistance) - Certain systems, such as cruise control or automatic braking, may be controlled by the car, one at a time.
- Level 2 (Partial Automation) - The car offers at least two simultaneous automated functions, like acceleration and steering, but requires humans for safe operation.
- Level 3 (Conditional Automation) - The car can manage all safety-critical functions under certain conditions, but the driver is expected to take over when alerted.
- Level 4 (High Automation) - The car is fully-autonomous in some driving scenarios, though not all.
- Level 5 (Full Automation) - The car is completely capable of self-driving in every situation.

The system is complex and requires a machine learning algorithm in order for a vehicle to learn a self-driving ride. Because of the dynamic and complex structure of the system, it is not possible to design a model in which the agent can learn all possible situations, and the classical coding method cannot achieve the purpose of the system here. While driving, good attention, experience and ability are expected from the driver. This situation represents an important challenge for understanding of the machine. The following section will give an overview for reinforcement learning and motivation for using RL algorithms. Then, contribution of the thesis will be given followed by the organization of the dissertation.

1.1 Literature Review

Artificial Intelligence algorithms have been involved in autonomous vehicles and their interactions in the recent times. Machine learning is seen as a subset of AI. There are three algorithms in ML that can be defined as learning problems. The learning sorts of machine learning application can be identified as:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Supervised Learning	Unsupervised Learning	Reinforcement Learning
• Makes machine learn demonstrably • Stands data with clearly defined output • There is a direct feedback • Predicts outcome/future	• Machine comprehends the data. (identifies patterns/structure) • Computation is not direct or qualitative. • No feedback required. • Does not predict.	• It is an approach AI. • Learning structure is reward based. • The model/agent learns how to act in a certain environment. • The goal is to maximize the award.

Table 1.1 Comparison of fundamental machine learning concepts

Supervised learning is machine learning technique that produces a function based on training data. In other words, in this learning technique, it generates a function that matches inputs (labelled data) and desired outputs. Training data consists of both inputs and outputs. The function can be determined by curve-fitting (regression) or classification algorithms. Linear regression is a technique often used to determine whether there is a linear relationship between inputs and outputs.

It can often be used to solve predictive and forecasting problems and many other data mining problems. Classification techniques focus on predicting a qualitative response by recognizing patterns and examining data. There are some widely used classification techniques. These techniques include logistic regression, linear discriminant analysis, K-nearest neighbours, trees, neural networks, support vector machines (Talabis et al., 2015).

In unsupervised learning, it is not clear which class the input data belongs to. This ML algorithm uses a function to predict an unknown structure on unlabeled data. It makes inferences about data according to the distances, neighbourhood relations and density of the data samples. In general, this is used in areas such as recommendation systems, marketing systems, customer segmentation and size reduction. Most used unsupervised learning algorithms are clustering, association rules, principal component analysis (PCA).

Reinforcement learning which is kind of the learning that has been studied more recently than other types of the machine learning are the concept where the most appropriate behaviour or action is reinforced with a positive reward. An RL agent/model learns by interacting with its environment and observing the results of these interactions in the absence of training dataset. The agent uses RL algorithms to occur this learning.

RL is used in systems with real-time decision making, recommendations, healthcare, artificial intelligence for games, robotic, autonomous driving, computer vision(recognition, detection, perception) and skill acquisition systems such as the possibility of later learning skill acquisition systems.

RL can be used for different tasks in autonomous vehicles. RL algorithms have been used in different application areas of autonomous vehicles. Some examples of this are examined below, together with their goals and conclusions. El Sallab et al. introduced a DRL system for lane keeping assist using Deep Q Network and Deep Deterministic Actor Critic algorithms. They compared Q-learning whose are separate actions and DDAC, whose actions are continuous. They also concluded that the more they set termination conditions for the same algorithm, the slower convergence time of learning (2016). Unlike a low-dimensional discrete state-space agent, the DQN agent was created to perform the autonomous car driving task from raw sensory inputs, and it was shown that the vehicle can be successfully controlled in the simulation environment even though it did not achieve a similar success (Vitelli and Nayabi, 2016). Zheng et al. established a 14-DOF (14 degrees of freedom) model adapted to the highway environment and implemented RL to the decision-making process and confirmed this in the simulation model. Simulation results was demonstrated that the decision-making system is effective and provided an important foundation to the real decision-making problem (2013). Desjardins and Chaib-draa investigated CACC by proposing a RL approach for the design of autonomous longitudinal vehicle controller using a policygradient algorithm (2011).

Xia et al. proposed a learning algorithm with deep Q-learning by repeating filtered experiences for self-driving vehicles. A model based on combinations of DL and RL is presented. Compared with the existing neurally placed Q-iteration algorithm, their model reduced time consumption by 71.2% in 300 trials. In addition, their algorithm has increased stability by 50% in 50 tests (2016). Wang et al. proposed an RL-based approach on finding an optimal driving policy to train the agent to learn an automatic lane change behavior. They designed the Q-function estimator, which contributed to the computational efficiency of the deep Q-learning algorithm (2018). El Sallab et al. provided a short overview of deep RL and described their proposed framework. After testing this framework in a simulator, the results show autonomous maneuvering learning in the scenario of complex road curves and simple interaction of other vehicles (2017). Chae et al. introduced the new autonomous braking system based on deep RL. The system learns a smart brake control method using the DQN method from the experiences obtained in the simulated environment, and as a result, the autonomous braking system is designed (2017). In study of Kardell and Kuosku, two reinforcement models called Deterministic Policy Slope (DDPG) and Experience Repetitive ActorCritical (ACER) were investigated using only image data and the vehicle's internal states as input. The models could get rid of a series of errors that put the car on the wrong driving lane (2017). In a simple environment of lane marks and static obstacles, a simulation study was conducted to train the agent using DQN. This is an investigation towards true driving (Okuyama, Gonsalves Upadhay, 2018). Fayjie et al. presented a DRL model for autonomous navigation and avoidance of obstacles with autonomous cars applied in an urban environment to a vehicle simulated with DQN (2018). Zhang et al. designed and trained the DRL based vehicle speed control system with real driving data. Dual Q-learning and DNN combined to form DQN. It is aimed to create a network capable of learning and providing the best control decisions in continuous and environmental action situations (2018). Chen et al. presented a framework for model free DRL that can be applied in autonomous driving scenarios. Then they applied three modern model free deep reinforcement learning algorithms (DDQN, TD3, SAC) to improve performance. The results have shown that their methods have the ability to solve tasks well (2019). Gao et al. studied the PCC (predictive cruise control) problem for a platoon of CAV(connected and autonomous vehicle). The RL strategy was used to develop a distributed optimal state feedback controller. The simulation results indicate that the resulting controller shows that each vehicle can adjust the starting time, speed and acceleration while reducing the opening time of each vehicle while following the desired paths (2019).

1.2 Motivation for Reinforcement Learning on Autonomous Vehicles

Reinforcement Learning is a form of ML algorithms which is a branch of AI. This learning model is one of three sub-branches of the ML structure. RL is a principled mathematical framework for experience-driven, goal-directed learning and decision making (Sutton & Barto, 2018). The RL framework was formulated as a basic structure, the agent shows an action according to the environment, it is called policy and expects a response from the environment (Sutton, 1988). The resulting reactions are subject to a predefined reward system.

Unlike other machine learning methodologies, reinforcement learning models are trained by exploring the environment. For example, supervised learning which is a function learning to obtain the desired output set from the given input set, receives a label in each decision since it is independent of each of the decisions, however RL decision is dependent, so we give tags to the series of dependent decisions. RL models operate in the goal-directed motion logic. It has a working principle with the awardpunishment method, which is a method of learning from its mistakes by interacting with the environment.

Reinforcement learning is also being used in many applications such as robotics, production, data processing and machine learning. Begin with these reinforcement learning models have been deployed in games. We can give earlier examples of Go game (Silver et al., 2016) and Atari game (Mnih et al., 2013). In recent years, RL models on the autonomous vehicle have increased due to its advantages. Since these models are caused by its interaction with the environment, the main subject in autonomous vehicle applications is to create a traffic simulation in a good simulation environment. The reinforcement learning (RL) framework for the controlling have long been used. Chris Watkins introduced Q-Learning in his thesis (1989). It is based on Markov Decision Process (MDP).

It is very difficult to make the driving process autonomous, which is mostly complex and dynamic. Dynamic obstacles, such as the reactions of drivers, vehicles interacting among themselves, pedestrians and other moving objects, can be constantly changing. It is difficult and time consuming to design a scenario that covers all of these through the supervised learning data set.

This thesis is about the simulation results of a simulated vehicle that will successfully complete the section without hitting a middle obstacle and hitting frame boundaries designed like a static obstacle at the same time. The algorithm and simulation results are demonstrated in the following sections.

1.3. Contribution

Autonomous driving has a complex system. The classical coding method is not feasible, as it is laborious to design all possible situations to teach the agent. As deep learning algorithms require a lot of data and also no sample data is available, studies in this area are becoming difficult. Due to this situation, deep learning algorithms have been replaced by more reinforcement learning applications.

By applying several reinforcement learning algorithms, the efficiency of autonomous driving control according to these algorithms has been compared. In order to properly create the simulation environment and control the autonomous vehicle with RL-based control algorithms, all the details of the system requirements must be taken into account. In the literature research, this thesis is seen as the example of a comparison study made by changing certain parameters of Q operator-based algorithms for selfdetermination application of an agent's motion strategy.

1.4 Dissertation Organization

This dissertation consists of 4 chapters. The literature review for reinforcement learning in autonomous vehicles is presented and the motivation of the thesis is given in Chapter 1. In Chapter 2, the algorithms to be used in autonomous driving are explained. The basic algorithms and mathematical theorems that form the basis of these Q learning based RL algorithms are explained. In Chapter 3, performance of algorithms is analyzed through simulation results. Simulation results are compared and discussed. Finally, Chapter 4 concludes the thesis and gives directions.

2. THEORY

2.1. Introduction to Reinforcement Learning

RL is a learning model that there exists an environment and agent. Reward-punishment logic has been established in this learning model. After the agent interacts with the environment, it is expected to receive an award with every move. This whole process repeats itself in every action step until the goal is achieved. Operating logic of RL is shown in the Figure 2.1. That is, the agent needs to act by trial and error to discover the most appropriate policy to maximize the cumulative reward.

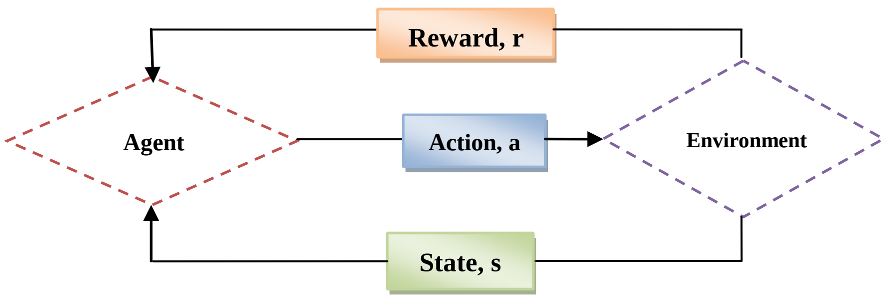


Figure 2.1: RL typical mechanism

Reinforcement learning: it is not limited to the terms mentioned above. RL model becomes functional together with other elements in its terminology. These are transition possibilities, immediate rewards, policy and performance metric.

Transition probability: It is an estimate of how the environment will react to the actions of the agent, specifying the possibilities of the consequences corresponding to each action. The model has the reward and state of the next stage due to the different possibilities of the next possible actions. Consider that action a is selected in state s. While the next state be j, it denotes p(s, a, j) the possibility of shifting from state s to state j under the move of action 'a' for one step. This is called transition probability.

Policy: It is the term that defines the action to be chosen for each situation to be visited. In some cases, no action is selected. For example, consider that a policy named π is to be applied. Then π(j) will denote the action selected by this policy for state j.

Performance metric: There is a performance metric for any selected policy that measures how well the policy's performance is. The aim is to implement a policy that has the best performance metric. Gosavi discussed two different metrics (2019). The first is the average reward of a policy, and the second is the discount reward.

Average reward: In immediate rewards, the system usually gets a value that we call a reward that is positive or negative as it passes from one state to another. The average reward, calculated over a very long period of time, is basically indicated by dividing the total instant rewards by the number of passes. The purpose of the average reward for MDP is to find a movement policy that maximizes policy performance measurement.

Discount reward: The discounted reward actually measures present value of the total of the rewards obtained in the future on an infinite time horizon. The objective of the discounted-reward MDP is to find the policy that maximizes the performance metric (discounted reward) of the policy starting from every state.

There are 3 basic approaches to implement the RL algorithm.

- Value Based: An RL method in which the value function is tried to be maximized.

- Policy Based: It is tried to find a policy in which the action taken in any case will be optimal in order to gain maximum profit in the future. If these are examined under two conditions;
    - Deterministic: In any case, the same action is generated by the policy.
    - Stochastic: Every action has a certain probability.
- Model Based: In this method, a virtual model is created for each environment and the agent tries to learn in this special environment. Since the model is different in every environment, there is no specific solution or algorithm for this type.

RL is based on the solution of Markov decision problems. RL uses Markov decision processes to define the interaction between a learning agent and its environment in terms of state action and rewards (Sutton and Barto, 2018).

2.1.1 Markov decision process

MDP provides us with a mathematical framework to decide related to a model. If the transition from one state to another is random, MDP is applied. The environment in which an agent perform and is fully observable can be defined as the Markov Decision Process. So this is known as a Markov characteristic, where the future is independent of the past. The important difference between MDP and RL is that the transition probability vector is not known and needs to be learned during interaction with the environment or to work with algorithms that do not depend on this vector. In the solution of MDPs, the last state time is always considered finite. This makes MDPs easier than reinforcement learning problems. (Kardell and Kuosku, 2017).

$$P(s_t+1|s_t) = P(s_t+1s|_1, s_2, ..., s_t) \qquad (2.1)$$

Formally MDP occur from tuple M = (S, A, P, R, γ) that has finite set of states, actions, transitional probabilities, rewards and a discount factor.
S: it is a finite set of states
A: A finite set of actions
P: The state transition probability matrix γ: A discount factor. $0 \leq \gamma < 1$ Controls how much future rewards will have an impact on optimal decisions (Littman, 1994).

Figure 2.2 shows a simple case MDP with five different states.

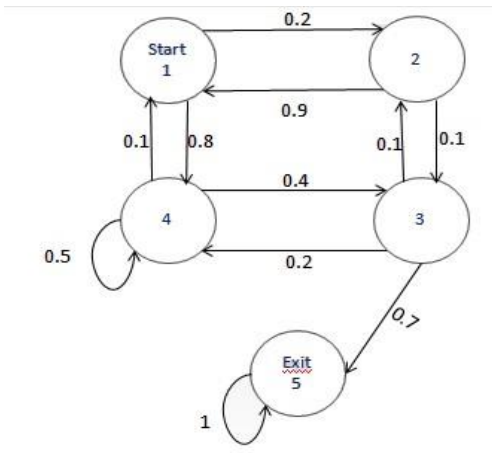


Figure 2.2: MDP with five different states

The MDP is based on the Bellman equation. Bellman Equation, known as dynamic programming, makes the calculation of the value function easier. Its purpose is to find the most appropriate solution by dividing a complex problem into simple and repetitive sub-problems, unlike the method of adding multiple time steps. A deterministic Belmann equation;

$$V(s_t) = \max_{a_t}(R(a_t, s_t) + \gamma V(s_{t+1})) \tag{2.2}$$

In MDP, the equation is established through the states, which may be the next state. MDP’s value of being state equation is described as (2.3) (Puterman, 1994).

$$V(s_t) = \max_{a_t}(R(s_t, a_t) + \gamma \sum P(s_t, a, s_{t+1}) V(s_{t+1})) \tag{2.3}$$

2.1.2 Partially observable markov decision process

The partially observable Markov decision process is a special form of MDP. If the agent cannot fully observe the environment it is in, the POMDP model is applied. In other words, the decision maker in MDP has predetermined the next state, but POMDP is an uncertainty model. It only knows the probabilities of all possible situations in which it will act. For example, in a 4-states model, the decision maker cannot know precisely what state it was in after the action of a_1 it chose while in the s_2 state, only the probability of the current situation is s_1, s_2, s_3 and s_4. With these possibilities, the concept of "belief state" arises. Belief state in a 4-states model is seen in Figure 2.3 (Patrick & Teichteil-Königsbuch, 2010).

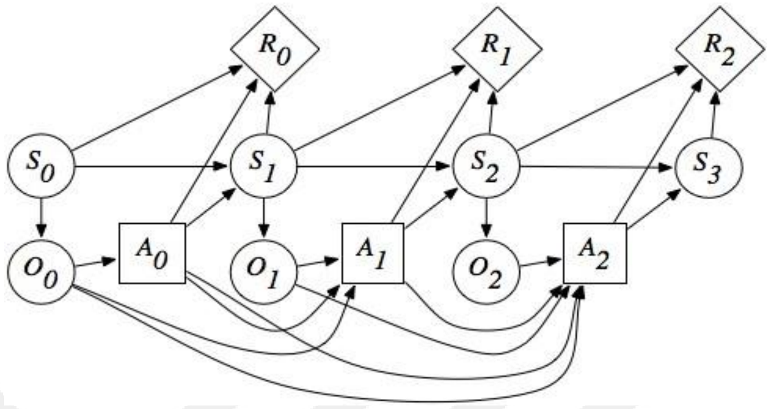


Figure 2.3: Belief state in a 4-states model

For situations that are not fully observable, Partially Observable Markov Decision Processes (POMDPs) was introduced (Smallwood & Sondik, 1973). A discrete-time POMDP has an agent and a model around its environment. Formally, a POMDP is 7tuple (S, A, T, R, Ω, O, γ) (Wikipedia Contributors, 2020).

S: A set of states

A: A set of actions

T: A set of conditional transition probabilities between states

R: The reward function

Ω: a set of observations (perceptions) O: set

of conditional observation probabilities γ:

The discount factor 0≤ γ≤1

Belief States: POMDP is always getting a new observation when it selects an action and goes into another belief state. Observations obtained in each state will affect the probability of occurrence of other states (belief states) in the next state. These observations give data about the state that the decision maker will go through in the next step. The agent’s belief related to this prediction is described as following equation:

$$b'(s') = aO(s',o)\Sigma T(s,a,s')b(s) \tag{2.4}$$

The decision loop in here is based on the concept of belief state, the agent decides the action a =π (b), it takes in the way that calculating the new belief state b' after moving to the next s' state. Russel and Norvig offered a real approach to POMDP problem and turn into MDP problem on a corresponding belief state space by using τ(b, a ,b′) and ρ(b,a) instead of T(s, a, s′) and R(s,a) (1994).

$$\begin{cases} \tau(b, a, b') = \sum_{o \in O} T(b'b, a, o) T(o|a, b) \\ r(b, a) = \sum_{x \in S} b(s) R(s, a) \end{cases} \tag{2.5}$$

τ : belief state transition function

When$T(b'b, a, o)$ = 1, after applying the POMDP decision cycle, b' becomes the next belief state of the agent, otherwise it is the opposite. Decision Network of POMDP is given in Figure 2.4 (Wikipedia Contributors, 2020).

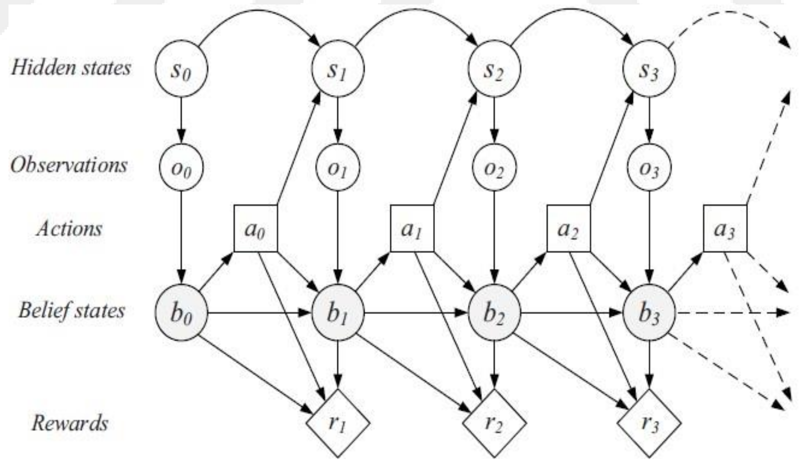
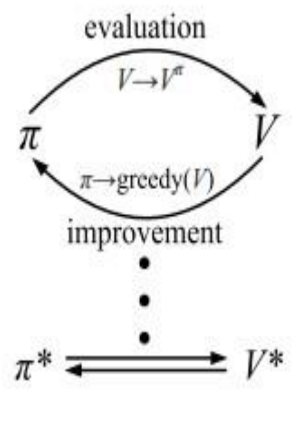


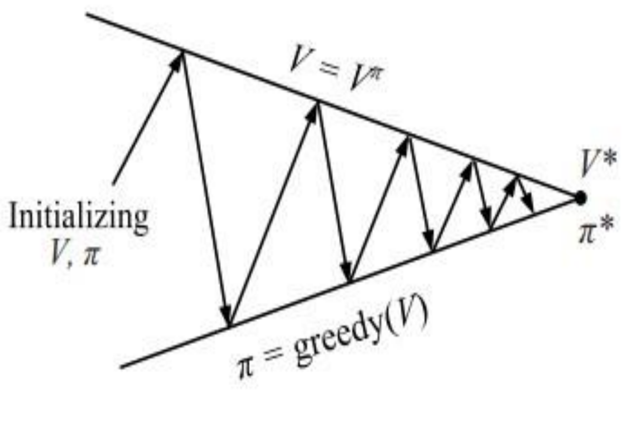
Figure 2.4: Decision network of POMDP

2.1.3 Model based algorithms: Dynamic programming

Basically Dynamic programming is a mathematical optimization problem. It can be defined DP as algorithms that calculate the most appropriate policy to solve the problem given as an MDP. DP separates a complex problem into simple sub-problems and provides the optimal infrastructure by finding the most suitable solutions for subproblems. A classic DP needs a good environmental model and large calculations. In this context, DP is considered as a model based algorithm. Model-based algorithms need the dynamics of the environment and reward function (Bertsekas, 1996). In DP, the environment is considered to be a limited MDP and the situation, action and reward sets are assumed to be finite. The DP's main theme is to use value functions to find out how to search for best policies. DP uses updated Bellman equivalents to correct the approach of value functions.

Dynamic Programming algorithms classify as policy iteration (PI) and value iteration (VI) (Sutton and Barto, 2018). These two algorithms act on the (generalized policy iteration) GPI principle (Sutton and Barto, 2018). GPI refer to a general concept that enables the interaction of policy evaluation and policy improvement processes. Nearly all of reinforcement learning algorithms are expressed as a well-defined GPI. They have describable policies and value functions. The policy is continually developed according to the value function and the value function is always directed to the value function of the policy, as seen in the form (a). The arrows in scheme (b) represent the behaviour of policy iteration (Sutton and Barto, 2018). As long as both processes continue to update all states, the ultimate goal is to approach the best value function and the best policy for the system model.





(a) **(b)**

Figure 2.5: Generalized Policy Iteration

There are the specified ways to find the best form for the policy found in the logic of DP algorithms. At first, policy evaluation is performed by calculating the value function. Then, the policy is tried to be optimized by using this value function. Best policy finding method with combined two steps is described as policy iteration. Each policy evaluation, which is an iterative calculation, begins with the value function of the previous policy. Steps for policy iteration are given in Algorithm 1.

Algorithm 1: Policy Iteration based DP (Sutton and Barto, 2018)

```
Initialization ∀ s ∈ S: V(s) ∈ ℝ, π(s) ∈ A(s)
İnitialize π(s) with an arbitrary action and V(s) with an arbitrary value;
Repeat
Policy Evaluation Repeat:
    Δ ← 0
    Repeat for each s ∈ S:
        V ← V(s)
        V(s) ← $\sum_{,r} p(s,r \mid s,\pi(s))\,[r + \gamma V(s)]$  s
        Δ ← max(Δ, |v “ V(s)|)
        Until Δ < 0
Policy Improvement
Policy-stable â˜ ← true
For each s S:
    old-action ← π(s)
    π(s) ← $argmax_a \sum_{s,r} p(s,r \mid s,a)[r + \gamma V(s)]$
    If old-action ≠ π(s), then policy-stable ← false
  If policy-stable, then stop and return;
  Else go to policy evaluation
Output: An optimal policy
```

Value Iteration (VI) involves overlap of evaluation and improvement processes. Rather than completely separating the evaluation and improvement processes, the value iteration approach interrupts evaluation after a single iteration. The Pseudo code of VI is shown in Algorithm 2 (Sutton and Barto, 2018).

```
Algorithm 2: Value Iteration based DP
Algorithm parameter: θ > 0 (accuracy of estimation) ∀s ∈ $S^+$
        initialize V(s) with an arbitrary value
Repeat
  ∀s   : $V_{m+1}(s) = max_{a \in A}[r(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s, \pi_t(s), s') V_m(s')]$;∈ S
                                                          ← m + 1
 until ∀s ∈ S: $|V_m(s) - V_{m+1}(s)| < \varepsilon$
 π                $_a \sum_{s',r}[r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_m(s')$
   m

(s) = $argmax$                                       ];
Output = An optimal policy
```

2.1.4 Model-free algorithms

Unlike dynamic programming, which is a learning method based on a particular model structure, there are RL methods which are model-free. RL focuses on MDPs to approximation and incomplete information and the need for sampling and discovery to collect statistical information about this unknown model (Xia, 2015). Such RL algorithms are concerned with how to achieve the most appropriate policy to maximize the expected cumulative reward when there is no an environmental model. In model based learning methods, the model is expected to have an acceptable structure. Efforts to create a model fit can lead that model-free algorithms are more advantageous on solving problems. An RL agent and environment can be created in case of s ∈ S. Actions that are discrete or continuous a ∈ A can be performed. It contains all information contained in the current state to predict future states. Each step receives the scalar reward value, which is assumed to be a function of agent state and observation.

Model-free algorithms are introduced as shown in Figure 2.6 (Li et al., 2018).

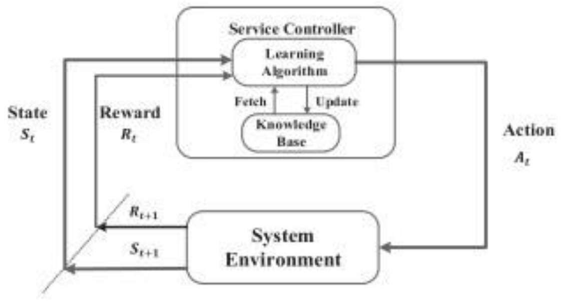


Figure 2.6 Schematic of model-free RL

A possible reward can be designed based on the energy costs of the actions taken on the road to reach the target and rewards of achieving the goals. It is designed to find a π policy that goes to states and takes the expected reward to the highest level based on the RL algorithm model. π policy can be deterministic or stochastic. The RL model is expected to discover the relationship between state action and reward to conclude this policy. A selection is required that determines whether the algorithm will depend on known actions with high rewards, or whether to randomly choose new actions to discover new strategies with a higher reward. This is known as exploration and exploitation.

The action selection style of the RL agent is off-policy and on-policy. If RL algorithms are training their agents only with experience from existing policy, this is called policybased algorithms. Policy based algorithms are often simpler and take notice of first (Sutton and Barto, 2018). However in on-policy algorithms, when the policy and behaviour of the agent are changed, it becomes more inefficient according to off-policy algorithms because of previous experiences cannot be utilized. For this reason, offpolicy algorithms are more preferable. The off policy learning model has a behaviour policy and a estimation policy. Behaviour policy is used for decision making. In order to discover all possibilities, all actions must have the probability of being selected. Estimation policy which is evaluated and developed is completely greedy because it does not affect decisions.

2.1.4.1 Monte carlo methods

Monte Carlo methods are model-free methods that do not have full knowledge of the environment. The MC method is a non-bootstrapping solution method for model-free algorithms use sampling to estimate the value function and discover the most appropriate policy. Pseudo code for estimating value is shown in Algorithm 3 (Sutton and Barto, 2018).

Algorithm 3: Value Estimation for Monte Carlo Method

```
First-visit MC Prediction, for estimating V ≈ $v_\pi$
Input: a policy $\pi$ to be evaluated Initialize:
∀  s ∈ S: V(s) ∈ ℝ arbitrarily ∀
   s ∈ S: Returns(s) ← an empty
list
Loop forever (for each episode):
        Generate an episode following $\pi$:
        G ← 0
        Loop for each episode, t = T-1, T-2, ...., 0:
                G ← γG + $R_{t+1}$
                For $S_t$ appearing in $S_0$, $S_1$ , $S_{t-1}$ ...,  :
                        Append G to Returns($S_t$ )
                        V($S_t$) ← $S_t$) average(Returns( )
```

MC methods do not need the transition function since no requiring whole information of environment, the estimation is updated through experience rather than the next state. The steps to be followed in the on-policy method can be seen in Algorithm 4 (Sutton and Barto, 2018). This experience represents exemplary sequences of situations and movements, awards simulated by the environment or from real interaction. It is noteworthy to learn from actual experiences, because it does not require prior knowledge about the dynamics of the environment, but can still achieve optimal behaviour. The model should produce only sample transitions, not the exact probability distributions of all possible transitions required for DP. In many cases, it is easy to create sampled experiences based on desired probability distributions, but it is not possible to obtain distributions clearly.

```
Algorithm 4: On-Policy Monte Carlo Method
Initialization ∀s ∈ S, ∀a ∈ A, Q(s,a)
∀ ∈ ∀ ∈ ∈ ℝ: initialize Q(s,a) with an arbitrariry actions
Π     S, a A: Returns(s,a) ← empty list
  an arbitrary $\varepsilon$-soft policy
Repeat (for each episode)
1. Generate an episode following π;
2. Loop for each step of episode, t = T-1, T-2,…, 0;
            For each pair Q($S_t$,$a_t$) appearing in the episode;
a. G ← γ  G + $R_{t+1}$
b. Append G Returns($S_t$ ,  $a_t$)
c. Q(s,a) ←  average (Returns($S_t$,$a_t$ ))
        A* ←  $argmax_a(Q(s_t, a_t))$;
        For  ∀a ∈ A($s_t$):
            Π($s_t$,a) ← $\begin{cases} 1-\varepsilon+\frac{\varepsilon}{|A(s_t)|} & if\ a = A^* \\ \frac{\varepsilon}{|A(s_t)|} & if\ a \neq A^* \end{cases}$
```

Monte Carlo methods are to solve RL problem according to average sample returns. Monte Carlo methods are defined for episodic tasks to ensure that well-defined returns exist (Sutton and Barto, 2018). It is assumed that the experience is divided into episodes and ends at the end of all episodes, regardless of which actions are selected. Value estimates and policies are changed shortly after an episode is completed. Therefore, Monte Carlo methods may be meaningful according to evaluation at the end of each episode, but it does not make sense in step-by-step (online). Pseudo code of offpolicy method is given in Algorithm 5 (Sutton and Barto, 2018).

Algorithm 5: Off-Policy Monte Carlo Method

```
$\forall s \in S, \forall a \in A$:
İnitialize                                                        Q(s, a) (arbitrarily)
$\pi(s) \leftarrow argmax_a Q(sia)$
Repeat
For each episode
Select a using any policy p: $S_0 , A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$           H
$\leftarrow 0$
W $\leftarrow$ 1
Repeat for each step of episode, t =T$-1$, T$-2$ ,…,0:
        H $\leftarrow \gamma$H + $R_{t+1}$
                                                  C($S_t$ , $A_t$ )$\leftarrow$ C($S_t$ , $A_t$ )
  $S_t, A_t) \leftarrow$ Q($S_t, A_t$) + $\frac{W}{C(S_t,A_t)}$[H $-$ Q($S_t, A_t$) ]  + W
 Q( For $\forall$s
$\in$ S:
        $\pi_{(S_t)} \leftarrow argmax_a Q(S_t$, a)
W â † W$\frac{1}{p(S,A)}$
```

2.1.4.2 Temporal difference methods

The TD method combines the specific aspects of Monte Carlo and dynamic programming approaches. At here as in the DP method, the expected value of the next state is used to strengthen the prediction, during the optimization of the value function for an initial state. This process is called bootstrapping. TD model does not have to have environmental dynamics like in MC method, it is model-free and learning takes place from raw experiences. Rather than rely on actual value and exact returns like in MC methods TD methods do not have wait until the end of the episode to update the expected reward prediction in the future, it just waits until the next time step to update the value estimations. In fact, in the case of TD (0) or single-step TD, learning takes place at every step. It updates the value functions online after each step. It takes notice of tasks which do not have a clear terminal state, learning and approximation value functions (non-episodic, non-deterministic or time-varying value functions.

While the target is G_t in MC method, the target which is value of $R_{t+1}$and S_{t+1} are determined in TD learning. MC, TD and DP methods use Bellman equation for updates; In MC;

$$\mathrm{V}(S_t) \leftarrow \mathrm{V}(S_t) + \alpha[G_t - V(S_t)] \tag{2.6}$$

As is known, MC method reaches the solution according to the average sample turns. G_t is the actual return after time t. While MC has to wait until the end of the episode to determine the increase in Vs and to know G_t, TD has to wait until the next time step.

In DP;

$$\mathrm{V}(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})] \tag{2.7}$$

In TD;

$$\mathrm{V}(S_t) \leftarrow \mathrm{V}(S_t) + \alpha\, [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.8}$$

α = learning rate. It takes values from 0 to 1. Learning takes place quickly if the values are close to 1. If 0, the learning value has not been changed. γ = discount factor, $0 < \gamma < 1$. This factor decides the value of future rewards based on current rewards. When it gets close to 0, the algorithm provides convergence.

The simplest TD method performs the update immediately after receiving the value of $R_{t+1}$and transition to the state S_{t+1}. This method is called as TD (0). TD prediction is showed in the form of Pseudo code in Algorithm 6 (Sutton and Barto, 2018).

Algorithm 6: Estimating $V_\pi\ TD(0)$

```
Input: the policy $\pi$ to be evaluated
Initialize V(s) arbitrarily (e.g., V(s) = 0, $\forall s$
$\in S$ ) Repeat (for each episode):      Initialize S
    Repeat (for each step of episode):
A $\leftarrow$    action given by $\pi$ for S
        Take action A, observe R, S'
        V(S) $\leftarrow$ V(S)+$\alpha$[R+$\gamma$V(S')-V(S)]
        S $\leftarrow$ S'
Until S is terminal
```

TD error is the difference between the estimated value and the better estimate value as seen in the equation 2.9.

$$\varepsilon_t = (R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{2.9}$$

There are two learning methods discussed in TD control. The first is SARSA, and the second is Q-Learning. While there is a exploration-exploitation trade off in MC methods, the approach here is form of on-policy and off-policy. SARSA algorithm has on-policy method while Q-learning algorithm has off-policy method.

2.2 Reinforcement Learning Methods and System Models

2.2.1 The general overview

The table 2.1 below shows the basic points where the RL algorithms are separated and share similarity. It can be considered a general overview of the specific approaches that algorithms have while providing learning.

Table 2.1 Comparison of RL algorithms

Algorithm	Description	Estimation Update	Model	Policy	Action Space / State Space	Operator
Dynamic Programming	Recursive	Step-by-Step	Modelbased	Off-Policy	Discrete/Discrete	Value function
Monte Carlo	Every visit to MC	Episode-byEpisode	Modelfree	Off-Policy	Discrete/Discrete	Sampling
Q-Learning	State-Action-Reward-State	Step-by-Step	Modelfree	Off-Policy	Discrete/Discrete	Q-Value
SARSA	State-ActionRewardState-Action	Step-by-Step	Modelfree	On-Policy	Discrete/Discrete	Q-Value

DQN	Deep Q Network	Step-by-Step	Modelfree	Off-Policy	Discrete/Continuous	Q-Value

The purpose of RL is to learn a good strategy for the agent from experimental trials and relatively simple feedback received. An agent in an unknown environment interacts with the environment to maximize cumulative rewards. The agent in a certain place makes its first move, as a result of which it gets a reward value, reaches a new state. The cycle continues until the environment sends a terminal state that ends with the episode, thereby achieving the target by following the most appropriate policy with observations from the environment. The structures called as the agent is the RL algorithms, and most of these algorithms follow the model described above.

2.2.2. Q-learning

The Q-learning method of the RL algorithm class, which is the sub-branch of ML methods, is one of the TD learning types with off-policy and model-free features. The basic parameters for Q-learning consist of environment, agent, state, action and reward. It aims to find the maximum value under deterministic conditions within the action sets in the motion set. The aim is to find the optimal path and reach the maximum reward. It operates with logic of state - action - reward and state again. According to this process;

 Each new situation depends on many parameters.

- The agent uses the experiences gained in each iteration to multiply the places it can go on its way to the award.
- These experiences are kept in the Q-table.
- The agent behaves randomly, since the Q-table initially has zero values.
- This random structure will continue until the agent finds the first reward.
- When the agent finds the reward, it updates the Q-table and thus keeps it in memory.
- Each time, the agent guesses and moves to the next step according to this algorithm and tries to reach the reward.
- After reaching the reward, the agent starts to act randomly again and tries to find the reward again.

- As this process continues, the agent learns the environment thoroughly and decides where to go in which state.

The operation steps of the Q learning algorithm are as shown in Figure 2.7.

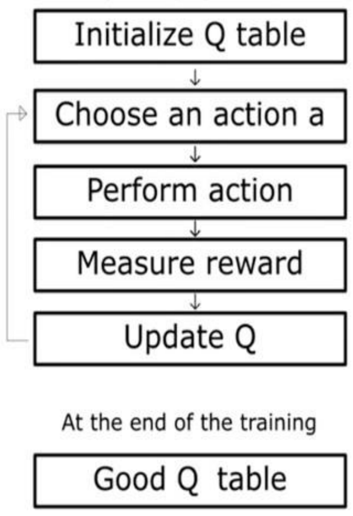


Figure 2.7. Q-Learning Algorithm Process

The basic structure of the Q-learning formula is Bellman equation. Q-learning can be considered as an improved version of Bellman Equation. The Q-learning algorithm is demonstrated by developing equations step by step.

- Bellman Equation: The agent receives random actions until it finds the maximum reward, thereby creating a path in the environment. Actions here are deterministic. The actions to be taken are deterministic.

$$V(s) = \max_a (R(a,s) + \gamma V(s')) \qquad (2.10)$$

V(s) = value of being state
R(a, s) = value of reward

V(s’) = value of being next state

- Markov Decision Process: It is a mathematical framework that occurs when Bellman Equation takes a stochastic approach. If outcome (i.e. transition from one state to another state) will occur within a probability, this decision making state is called MDP. So V (s') is not certain. According to this;

$$V(s) = \max_a (R(a,s) + \gamma \sum_{s'} P(s,a,s')V(s')) \tag{2.11}$$

- Consist of Q-learning function;

$$Q(s,a) = R(a,s) + \gamma \sum P(s,a,s)V(s) \tag{2.12}$$

$$V(s') = \max_a (Q(s,a)) \tag{2.13}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} (P(s,a,s') \max_a (Q(s',a')) \tag{2.14}$$

- TD located in square bracket is the difference between the Q value formed in the next time step and the current estimate of the optimal Q value. In this last step, where the feature of TD method in Q* learning will be seen, it is seen in the algorithm equation (2.17).

$$TD = Q(s,a)_{t+1} - Q(s,a)_t \tag{2.15}$$

$$Q(s,a)_{t+1} = Q(s,a)_t + \alpha TD_{t+1} \tag{2.16}$$

$$NewQ(s,a) = Q(s,a) + \alpha\, [R(s,a) + \gamma \max_a Q'(s',a') - Q(s,a)] \tag{2.17}$$

New Q(s,a): new Q value for that state and the action
Q(s,a): current Q value
Max Q(s’, a’): Maximum expected future reward given the new state (s’) and all possible actions at that new state.

Learning rate (α): It is a parameter set between 0 and 1. If it is 0, it means that no learning has taken place and the Q value has not been updated. Value of learning rate close to 1 means that learning takes place quickly.

Discount Rate (γ): $\gamma \in (0,1)$. If this factor approaches 1, it strives for a high long-term reward. If it is slightly lower than 1, learning the Q function causes errors and instability to increase when the value function approaches an ANN (Wikipedia Contributors, 2020). In this case, starting with a lower discount factor is to accelerate learning towards the most appropriate value.

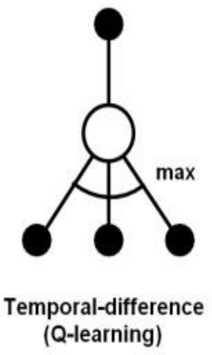


Figure 2.8: The backup diagram of TD (Q-Learning)

In addition, the ε-Greedy method is applied in the exploration - exploitation selection in Q-learning. Accordingly, if a value less then ε is produced randomly, it is suitable for exploration, even if a large value appears, exploitation decision is made and the current policy is followed.

$$a = \begin{cases} a & 1-\varepsilon \\ random\ action & \varepsilon \end{cases}$$

(2.18) ε: Probability of Exploration, $\varepsilon \in (0,1)$

There is also a living penalty statement in Q-learning. If the living penalty parameter is high, this will prevent the value from maximizing the total reward, and the agent produces the quickest solution and can choose the risky way. Pseudo code of Q-learning algorithm is given in Algorithm 7 (D. Pandey & P. Pandey, 2010).

Algorithm 7: Q-learning algorithm

```
Initialize Q(s,a) arbitrarily
Repeat (for each episode)
Initialize s
Repeat (for each step of episode):
Choose a from s using policy derived from Q
Take action a, observe r, s’
Q(s,a) = Q(s,a) + α [R(s,a) + $\gamma^{max}_{a}$Q’(s’, a’) – Q(s, a)
          ] s←    s’:
Until s is terminal
```

2.2.3 SARSA algorithm

One of the algorithms in TD control is SARSA. SARSA is also known as state action reward next state next action. It was first technically proposed more descriptive by name "Modified Connectionist Q-Learning" (Rummery & Niranjan, 1994). After a SARSA agent interacts with the environment, it updates its policy based on the actions it takes, so it is known as a policy-based algorithm. SARSA uses the action value function Q and follows the π policy. GPI (Generalized Policy Iteration as described in section 2.1.3) is used to take action based on policy π. (The -greedy approach takes over for developing policy and also selection of exploration - exploitation dilemma).

$$NewQ(s,a) = Q(s,a) + \alpha\, [R(s,a) + \gamma Q(s',a') - Q(s,a)] \qquad (2.19)$$

This update is repeated each time from the last non-terminal s state. This formula uses (s, a, r, s', a') parameters that create the transition from one state-action pair to another. The word SARSA was created as a representation of these parameters. The hyperparameters of α (learning rate) and γ (discount rate) are used on the same logic as in Q-learning. The backup diagram is given for SARSA in Figure 2.9.

- TD target is R(s,a) + γQ(s’,a’).
- TD error is R(s,a) + γQ(s’,a’) – Q(s,a).

Figure 2.9: The backup diagram of TD (SARSA)

It is simple to create an on-policy control algorithm according to the SARSA estimation method. As with all policy-based methods, q_π is continuously estimated for the π behaviour policy. It also changes π against greed in relation to q_π. The steps followed to reach the optimal result in the SARSA learning algorithm are shown in Algorithm 8 (Xu et al., 2018).

Algorithm 8: SARSA Learning Algorithm

```
Initialize Q(s,a) arbitrary **Repeat**
(for each episode):
    Initialize S
Choose a from S using policy derived from Q
      (e.g., – greedy)
Take action a, observe r, s'
        Choose a' from s' using policy derived from Q
        (e.g., $\varepsilon$ – greedy)
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
      S ←       ← s'; a   a'
  **Until** S is terminal
**Until** all Q(s,a) is convergent
```

When looking at the Pseudo code of both algorithms, Q-learning first updates the Q function, the action to be selected in the next iteration derived from the updated Qfunction is not required to be equal to the next action selected to update the Q. But SARSA first selects a' and s', then updates the Q-function. The convergence properties of the Sarsa algorithm may vary according to the feature of the policy's dependence on Q. Sarsa's convergence to an optimal policy and action-value function with the probability of 1 occurs when all state-action couples are visited infinitely many times and the policy converges the limit of the greedy principle. (for example; with ε-greedy policies by setting ε=1/t)

2.2.4 Artificial neural network with Q-learning: DQN

Q-learning method which is one of the RL algorithms is a good learning model, but when the number of Q-values is high, it is difficult to create a learning model using the Q-table in the Q-learning method. To overcome this situation, the Q-learning model and neural networks are combined (Wu et al., 2017). This is called as Deep Q-Network (DQN).

Increasing the number of Q-values causes the model to become complex and can only be applied using artificial neural networks. Q-learning has quality of action that is called as Q-function. The goal is to increase the quality of action. In other words, there should be such an action that as a result of this action, the Q function has a high value and in the end, a good reward is obtained.

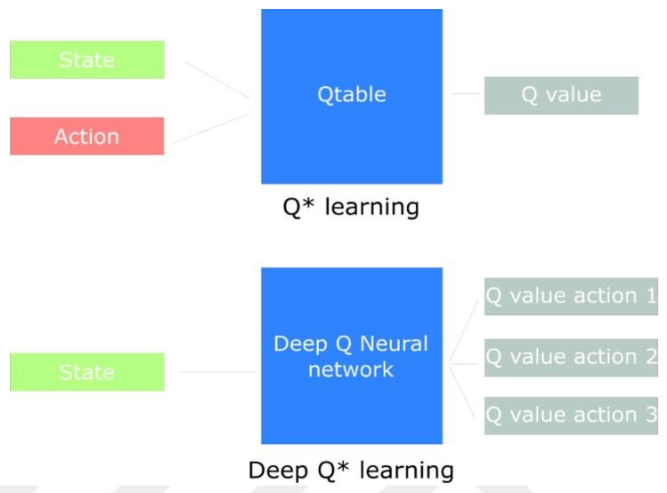


Figure 2.10: Q learning vs deep Q learning

There are several reasons why the Q-learning method is not sufficient alone. The first one is set of state. There are states in which the agent moves in environment. Assuming that the number of these states are one thousand; this can be a suitable number for Qlearning. However, when this number reaches tens of thousands and even millions, the Q-learning method becomes difficult to implement since the Q-table cannot be created. Getting Q values by the Q-table may become impossible in terms of hardware. We can define this as a resource problem.

Another reason is number of state. Suppose that the agent is trained using the Qlearning algorithm under a specified number of states. When this agent faces a state case that is not found in the Q-table for another environment, the Q-learning algorithm fails because it cannot produce a solution in this case. Considering from another point of view, a simple Q-learning method does not work in complex environments. Once a picture has been processed, each pixel is defined as state, which means there are thousands of states. Since this size will be challenging for the Q-table capacity, it causes a decrease in performance and thus, the scenario to be realized cannot be overcome.

In the deep Q-learning method, function approximation is used to calculate Q-value. Instead of directly calculating Q-value, a function that approximates to Q-value arises. ANN are used to perform this process with function approximation. DNN is used to estimate Q values. Finally DQN will approximate optimal Q-function. Structure and logic of ANN are explained in detail in Appendix.

The below pseudo code is followed on training of the Algorithm 9 (Mnih et al., 2015).

Algorithm 9: Deep Q-learning with experience replay

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights θ
For episode = 1, M do
    Initialize sequence $s_1$ = {$x_1$ } and preprocessed sequence $\phi_1$
        = ϕ($s_1$ )    For t = 1, T do
        With probability $\varepsilon$ select a random action $a_t$
        Otherwise select $a_t = argmax_a Q(\phi(s_t), a;$ θ)
        Execute action $a_t$ $in\ emulator\ and\ observe\ reward\ r_t\ and\ image\ x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}\ and\ preprocess\ \phi_{t+1}$= ϕ($s_{t+1}$)
        Store transition ($\phi_t, a_t, r_t, \phi_{t+1}) \ in\ D$
        Sample random mini-batch of transitions ($\phi_j, a_j, r_j, \phi_{j+1}$) from D

$Set\ y_j = \begin{cases} r_j & if\ episode\ , terminates\ at\ step\ j+1 \\ rj + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j, \theta))^2$ with respect to the network
        parameters Î¸
        Every C steps reset $\hat{Q} = Q$
    End For
End For
```

In deep Q-learning, training takes place in the neural network. In this algorithm, the loss function statement that measures the error rate of the model is mentioned with the activation of the artificial neural network. Loss function must be calculated for the training to take place. Loss function is also expressed as MSE;

$$\text{Loss Function: } MSE = \sum(\hat{Q} - (Q_{target} + R))^2 \qquad (2.20)$$

$\widehat{Q}$: The value resulting from neural network (trained Q-value)

Q_{target}: Actual expected value

R : Reward value

The Q_{target}value seen in this function is one of the important elements that improve learning performance.

State information (observation) from the environment enters the neural network as an input layer and passes through the artificial neural network with forward prediction, and as a result, the trained Q-values of the neural network are output. Here, the largest Qvalue determines the action to be performed and the cycle continues in this way.

$$L_i(\theta_i) = \mathbb{E}(s,a,r,s')\sim U(D)\left[\left(r + \gamma maxQ(s',a';\, \theta_i^-) - Q(s,a;\, \theta_i)\right)^2\right] \qquad (2.21)$$

The process of the DQL algorithm and neural network model is showed in Figure 2.10. Afterwards the basic structure and properties of the neural network are specified. Initially, the agent waiting in the environment gets an action, then returns the environment observation and reward value. The information received from the environment goes to the neural network, training in the neural network starts this point. Parameters such as hidden layer and number of neurons are determined according to need. (For example, the complexity of the environment and the scenario) State information from the environment enters the neural network with forward prediction. As a result, output neurons called as trained Q value as the number of actions are seen. The largest trained Q value value acts in the environment as action. The loss function shown in Figure 2.10 is required for function training.

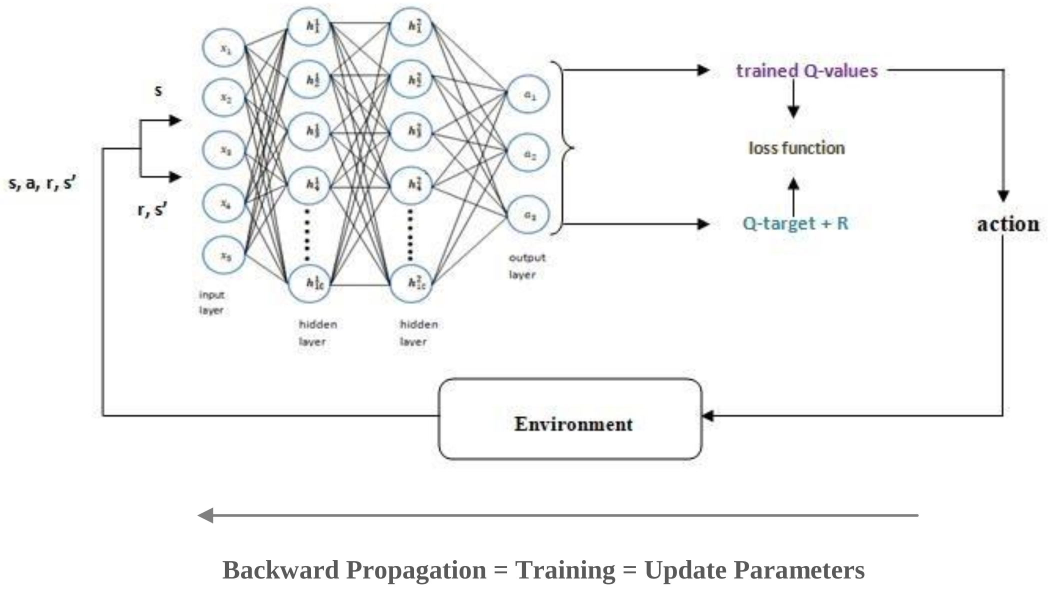


Figure 2.11: Operation sequence of DQL model with the neural network

Considering the temporal difference method in the structure of Q-learning based algorithms, according to equation (2.8);

$$\mathrm{TD} = \underbrace{R(s,a) + \gamma \max_a Q'(s',a')}_{\textbf{Q-target + R}} - \underbrace{Q(s,a)}_{\textbf{trained Q-values}}$$

The ANN model in Figure 2.10 is done with the sequential method since keras is used. Sequential method to be created for the neural network is the basic structure to be built on the neural network. Two hidden layers are added on the model. There are 10 neurons in the hidden layer added first. The states taken as input are determined as the values that five sensors take in the continous state.

The activation function needed by the layer is determined as the ReLu (rectifier linear unit) function. Glorot uniform, also called Xavier uniform initiator, is used as initializer. This draws samples from a uniform distribution in the form of [-limit, limit] that determines the processed state of input and output weights as the boundary. (Glorot et al., 2010).

$$\text{limit} = \text{sqrt}(6 / (\text{fan\_in} + \text{fan\_out})) \tag{2.22}$$

fan_in = the number of input units in the weight tensor fan_out = the number of output units in the weight tensor

When it comes to the output layer, the linear function is used as the activation function. In the car environment library, it is conditioned that if the discrete action is true, the size of the action will equal the length of the sequence of discrete actions.There are three discrete actions denoted as [-1, 0, 1] which are indicating backward, stop and forward actions respectively.

3. SIMULATION RESULTS

A representative vehicle will move on a road with an aim and with a course of action without hitting any obstacle. It can be thought of as the vehicle's target successfully completing the appropriate roundabout turns. The simulation environment was created with the Pyglet tool which is a multimedia library for Python and which provides a programming interface for the concept of objects. Finding the goal in the shortest way, with less time and more accuracy is the main motivation.

In this study, necessary commands for the applicability of the algorithms are added to the existing simulation environment (Zhou, 2018). The main objects in the environment consist of a vehicle, 5 sensors and an obstacle. A frame consisting of rows and columns selected as 500x500, an obstacle around the vehicle without hitting its surface and the starting point of the vehicle are given in the Figure 3.1.

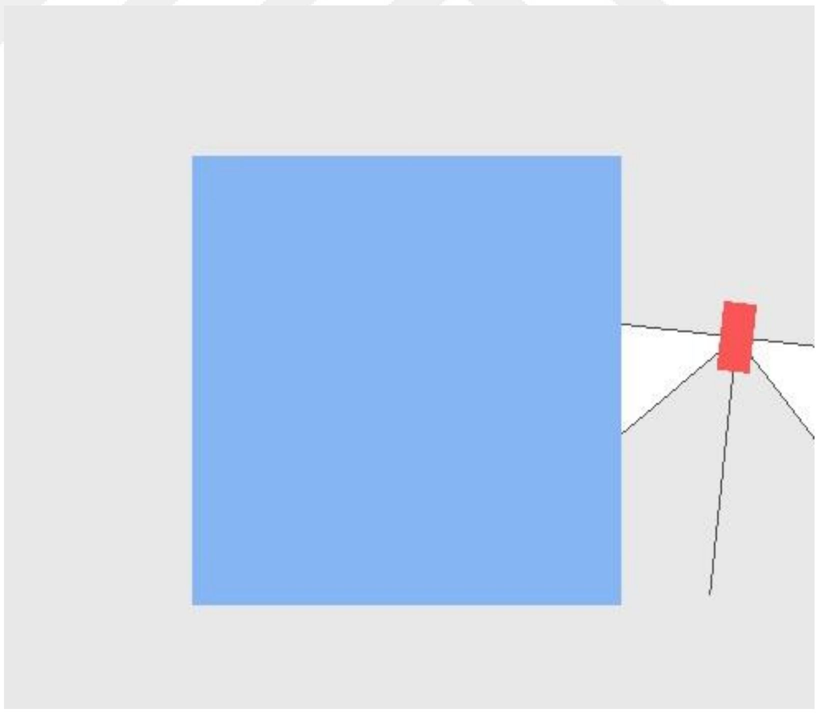


Figure 3.1: The simulation environment and agent’s starting point

The reinforcement learning algorithms that have been introduced in Chapter 2, applied on the same vehicle model in the same environment with the same initial conditions. Qlearning, SARSA and DQN algorithms are used as decision-makers in this study. Training and tests were divided into sections in this study.

The maximum number of steps in a section was determined as 1000. After 1000 steps, the environment will be reset and a new episode will start. If the vehicle did not collide with any windows or obstacles, the environment's reward function returns +1 as a reward for each step. If the car collides with the window or obstacle, the reward returned by the environment will be -100 and the episode will end as the collision is a terminal situation.Based on a maximum number of 1000 steps, a 900 winner reward has been determined for the auto agency.

The total of rewards accumulated during each episode is checked and compared to the winning reward and the number of wins is increased by 1. If the number of gains for consecutive segments is equal to the number of wins, the learning of the algorithm is stopped. Also, the test epsilon was determined in the test function of each algorithm to add some uncertainty to the testing of the Q-table and neural network model. The test epsilon value is 0.05, and it brings 5% uncertainty in the behavior of the agent when choosing an action. The applied SARSA and Q-learning algorithm makes a Q-table for Q values. The Q table is made for the problem of the grid world in which states and actions are both finite and discrete. Also, when testing Q-tables from SARSA and Qlearning algorithms, by changing the test_epsilon value from 0.05 to 0; any uncertainty in testing the Q-table can be eliminated.

In this study, the steady state of each sensor in a 2D car environment is divided into five separate states between 0 and 1 (0.0, 0.25, 0.50, 0.75 and 1.0). This separation was made to make it possible to the discrete from the continuous state of the environment and create a Q table.

3.1. Simulation with Q-Learning Algorithm

The Q learning algorithm, which is the basic operator for most of the RL algorithms, refers to the iterative training of the Q value. The simulation results of the decision making algorithm is given in Figure 3.2 and Figure 3.3.

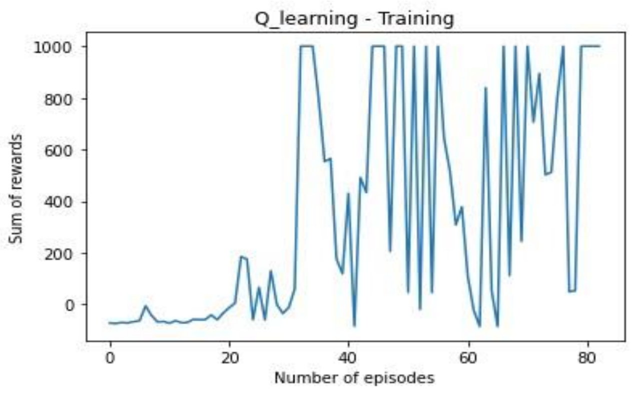


Figure 3.2. Training results of Q-Learning algorithm

The target of the Q-learning agent is 32 for the first time. It is seen that the optimal policy has been reached after episode 80. The winning ratio for a reward higher than winning reward is %82. Avarege reward for 100 episodes is 915,23. This means that the training of the Q-learning agent is not too fast or too slow. In order to solve the situation of being able to move without hitting the obstacle, the speed of reaching the target can be said to be equivalent to the average performance of other agents. It may seem that the training performance is not the best, but it has gradually gotten better in each episode. According to the policy learned during the training phase, it is seen that the cumulative rewards reach a more stable and higher value after the 60th episode.

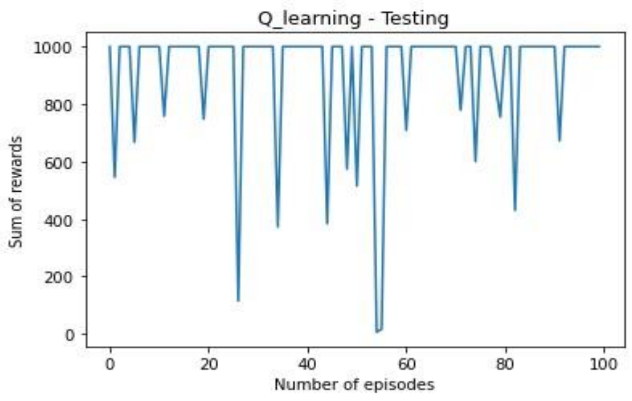


Figure3.3 Testing results of Q-Learning algorithm

3.2. Simulation with SARSA Algorithm

It is stated in the literature that SARSA is insufficient among these algorithms.. The simulation results of the SARSA algorithm using the decision making algorithm are given in Figure 3.3. The section where it reaches the maximum reward can be seen as episode 25, and after episode 30 it reaches the optimal policy. Looking at these results; the winning ratio for a reward higher than winning reward is %81. Avarege reward for 100 episodes is 839,57.

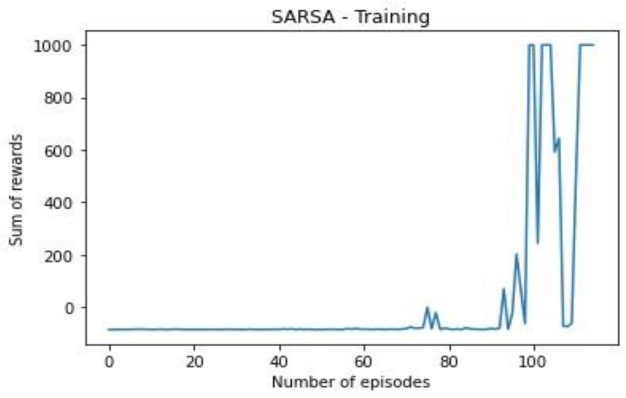


Figure 3.4. Training results of SARSA algorithm

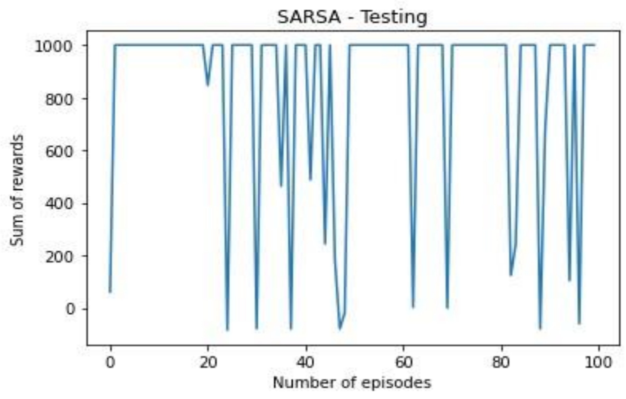


Figure 3.5 Testing results of SARSA algorithm

In addition, there are some methods to improve the results of Q learning and SARSA algorithms with some changes. Changing some parameters (such as number of wins which is needed for learning criteria of algorithm, learning rate, and number of episode) set 0

When 5, 10 and 20 values are given for the num_wins variable parameter determined for the learning criterion on the Q-learning algorithm; the changes that occur are observed as in Figures 3.6.

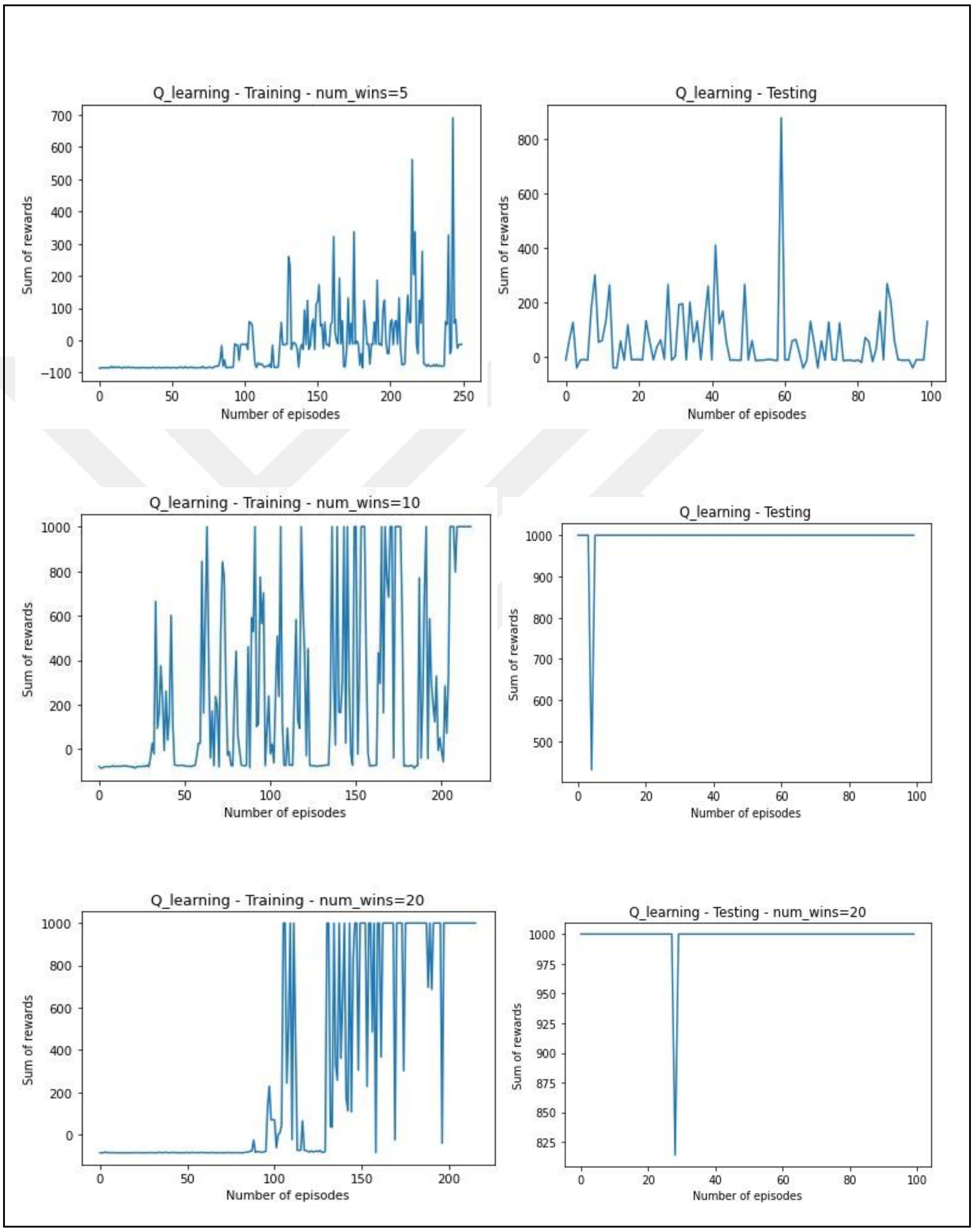


Figure 3.6 The variations in the output of the Q-learning to the win count

For the SARSA algorithm, Figure 3.7 can be observed for the results where the num_wins parameter gets 5, 10, and 20 values respectively.

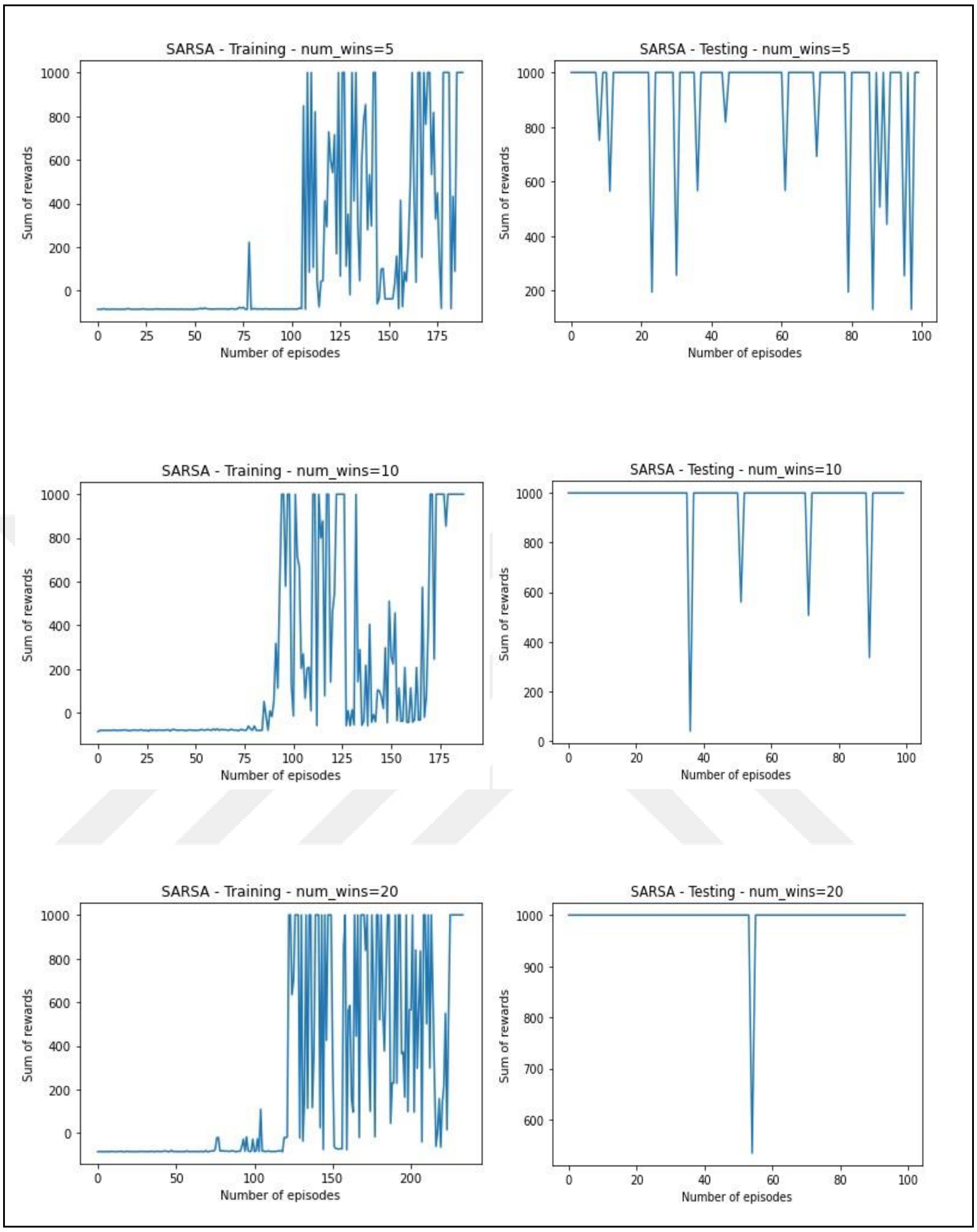


Figure 3.7 The variations in the output of the SARSA to the win count

When the Figure 3.6 and Figure 3.7 are examined, it is seen that has reached the optimum policy in general more decisively. Considering the test results, it can be said that the cumulative rewards reach a higher level.

3.2. Simulation with Deep Q-learning Algorithm

Simulation results of Deep Q-learning algorithm as the decision-making algorithm is given by Figure 3.8. It can be seen here that the optimal policy is found in episode 70th. The reason why the agent had difficulty in reaching the target in the first episodes is that the neural network has not been experienced yet. Winning ratio for a reward higher than winning reward is 100%. Average reward for 100 episodes is 1000. The exact accuracy here is due to the fact that the neural network is quite adequate for this scenario.

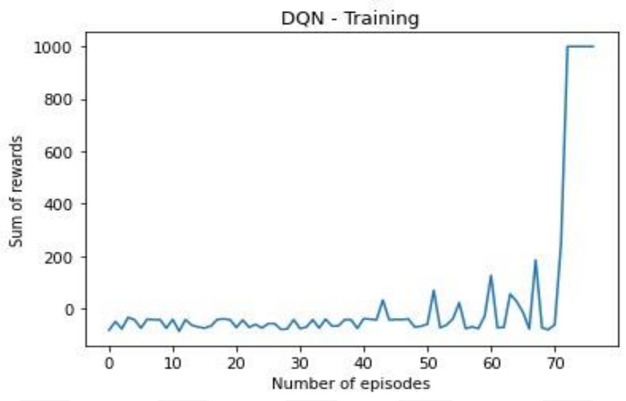


Figure 3.8 Training results of Deep Q-Learning algorithm

For many cases with infinite possible states, the linear approach and the Q table is not the right choice, as seen here DQL has been trained on continuous states and discrete actions, concluding that the nonlinear function approach using neural network is a very powerful approach.This environment, which is simple for the neural network to learn quickly, ensures that the test output has 100% accuracy.

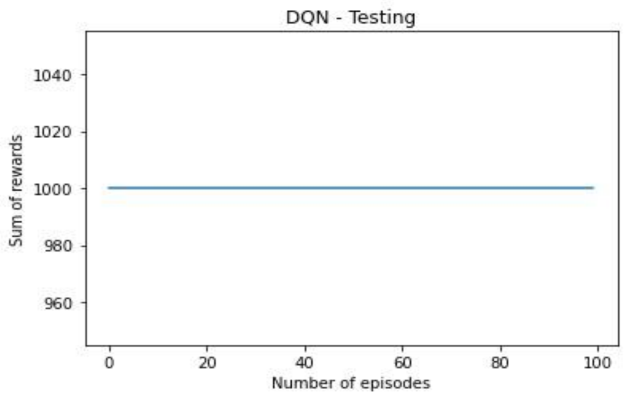


Figure 3.9 Testing results of Deep Q-Learning algorithm

With the parameters determined for the DQL algorithm, it is seen that the result can reach 100% success in the test phase (Figure 3.9). The quality of the policy learned by the algorithm emerges with the values of the parameters such as learning speed, winning_reward change, which affect the training criteria of the DQL algorithm. The changes according to the different winning rewards is shown in Figure 3.10.

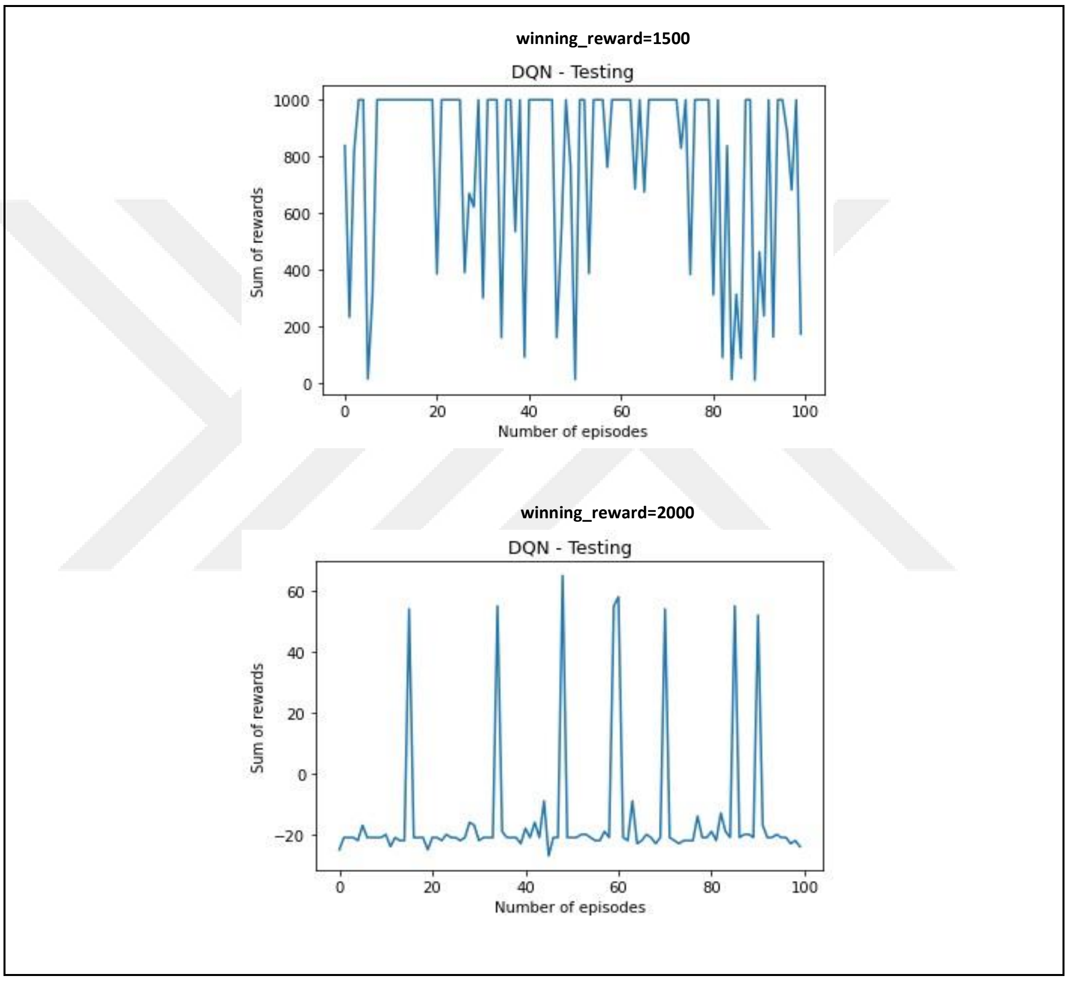


Figure 3.10 Testing results of DQL algorithm under different winning rewards

The more indecision occurs in the testing phase as the frequency of the agent to reach the optimal policy decreases. For 100 episodes, it is seen that when the winning reward is 2000, the average reward value is -14.50, and when the winning reward is 1500, the average value can increase to 787.83. It has been analyzed that the further away from the goal of winning reward, the less successful the test results are.

The learning rate, which is considered as one of the basic parameters in RL algorithms, is one of the parameters that most affects the cumulative reward (status of success) of the agent. With the decrease in the learning rate, the cumulative reward decreases and the test was unsuccessful. It was observed that the agent reached the optimal state as a result of the values taken by the other parameters along with the 0.001 learning rate value. In the graphics shown in Figure 3.11, the testing results are shown when the learning rate value is 0.01 and 0.0004, respectively.

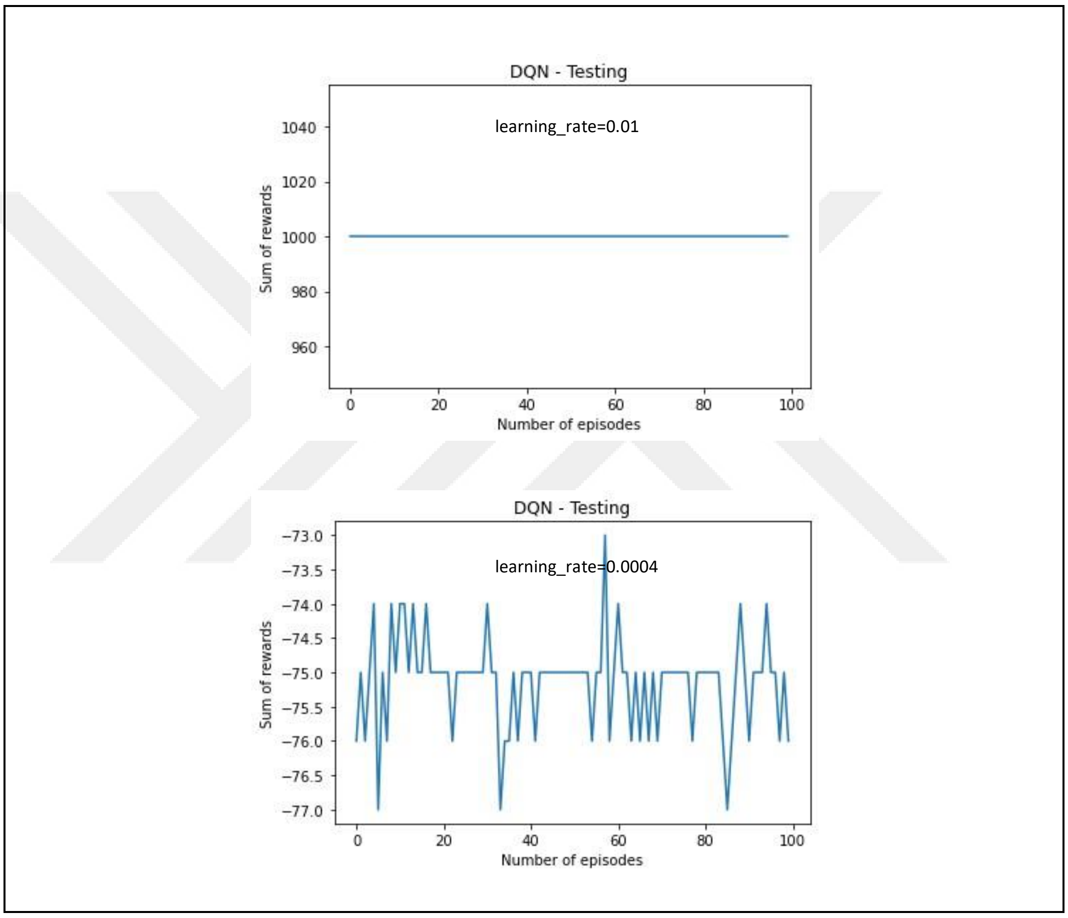


Figure 3.11 Testing results of DQL algorithm under different learning rates

When the learning rate was 0.01, the representative reached the highest cumulative reward in the 70th episode with the 100% success of the test result. When the learning rate is as low as 0.0004, the avarage reward value for 100 episodes becomes -75.14. Considering the evaluations, an optimal and more stable training and test result is seen when the learning rate parameter and the winning reward value are respectively 0.001 and 900 for this scenario.

4. CONCLUSION

It is aimed to determine the most efficient RL algorithms and to compare their performances. The most important point in this study is the change in algorithms that occurs with the activation of neural networks. The inclusion of the neural network makes the RL algorithms more effective. Firstly, the applied SARSA and Q-learning algorithm creates a Q-table for Q values. The table Q is made for the problem of the grid world where states and actions are both finite and discrete. While Deep Q learning algorithm is applied in continuous state due to function convergence feature, SARSA and Q learning algorithm are applied under finite and discrete state action. Related to this, the sensor states in the given environment were converted into a discrete state and the state number was brought to a finite state.

First of all, training and tests were divided into sections. The maximum number of steps in a section is set as 1000, after 1000 steps, the environment will be reset and a new section will start. If the vehicle did not collide with any windows or obstacles, the environment's reward function returns +1 as a reward for each step. If the vehicle collides with a window or obstacle, the reward value returned by the environment will be -100 and the episode will be terminated as the collision is a terminal condition. Based on the maximum number of steps of 1000, a reward of 900 value was determined for the vehicle. The total of reward accumulated during each episode is checked, comparing the winner with the reward, increasing the number of wins by 1. If the number of wins for consecutive divisions is equal to the number of wins, the learning of the algorithm will be stopped. A solution method was followed by dividing the steady state of each sensor in the 2D vehicle environment into five separate states between 0 and 1 (0, 0, 0.25, 0.50, 0.75 and 1.0). It is possible that these 5 separate situations for each sensor are not sufficient to obtain a very good optimal solution using Q-learning and SARSA algorithm. There are continuously infinite values between 0 and 1 for the value of each sensor, and if these are converted to certain discrete values and the same result is desired with DQL, the sensor states must be divided into 10 or 20 separate states for each sensor. It will turn the Q-table into a huge table with hundreds of thousands or millions of pairs of situation actions.

The linear approach to making a Q-table for Q-values using Q-learning and SARSA only gives better results for small grid world problems. However, for many situations and situations with infinite possible states, the linear approach and the Q-table are not the right choice. By assigning different values to some parameters for all algorithms in this study, the results for the training and testing phase were observed. Considering this study, the structure of the system should be considered in the selection of the algorithm. The algorithm should be selected according to system requirements, taking into account many parameters such as how complex the system is, the presence of static and dynamic obstacles, and the number of probabilistic situations. In more complex scenarios, for example; For a traffic scenario with pedestrians, traffic lights and other vehicles, an algorithm in which a more layered deep neural network is activated will give more effective results.

DQL has been trained on continuous states and discrete actions, and the nonlinear function approach using neural networks can be seen to be a very powerful approach. It has been concluded that the nonlinear function approach using neural network is a very powerful approach.

REFERENCES

Wikipedia contributors. "Houdina Radio Control." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 1 Nov. 2019. Web. 4 Apr. 2020.

Wikipedia contributors. "Cruise control." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 1 Apr. 2020. Web. 4 Apr. 2020.

C. Thorpe, M. H. Hebert, T. Kanade and S. A. Shafer, "Vision and navigation for the Carnegie-Mellon Navlab," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 362-373, May 1988.

SAE International (2014), Society of Automotive Engineers International Standard, J3016.

Mark Ryan M.,Talabis, Robert McPherson, I.Miyamoto, Jason L.Martin, D.Kaye, Finding Security Insights, Patterns and Anomalies in Big Data 2015, Pages 1-12

Sallab, Ahmad El, Mohammed Abdou, Etienne Perot and Senthil Kumar Yogamani. “End-to- End Deep Reinforcement Learning for Lane Keeping

Assist.” *ArXiv* abs/1612.04340 (2016): n. pag.

A. N. Matt Vitelli, “Carma: A deep reinforcement learning approach to autonomous driving,” 2016, https://web.stanford.edu/~anayebi/projects/CS_239_Final_Project_Writeup.pdf.

Rui Zheng, Chunming Liu and Qi Guo, "A decision-making method for autonomous vehicles based on simulation and reinforcement learning," 2013 International Conference on Machine Learning and Cybernetics, Tianjin, 2013, pp. 362-369, doi: 10.1109/ICMLC.2013.6890495.

C. Desjardins and B. Chaib-draa, "Cooperative Adaptive Cruise Control: A Reinforcement Learning Approach," in IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 4, pp. 1248-1260, Dec. 2011, doi: 10.1109/TITS.2011.2157145.

W. Xia, H. Li and B. Li, "A Control Strategy of Autonomous Vehicles Based on Deep Reinforcement Learning," 2016 9th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, 2016, pp. 198-201, doi: 10.1109/ISCID.2016.2054.

P. Wang, C. Chan and A. de La Fortelle, "A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers," 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, 2018, pp. 1379-1384, doi: 10.1109/IVS.2018.8500556.

Sallab, Ahmad & Abdou, Mohammed & Perot, Etienne & Yogamani, Senthil. (2017). Deep Reinforcement Learning framework for Autonomous Driving. Electronic Imaging. 2017. 70-76. 10.2352/ISSN.2470-1173.2017.19.AVM-023.

H. Chae, C. M. Kang, B. Kim, J. Kim, C. C. Chung and J. W. Choi, "Autonomous braking system via deep reinforcement learning," 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, 2017, pp. 1-6, doi: 10.1109/ITSC.2017.8317839.

S. Kardell and M. Kuosku, “Autonomous vehicle control via deep reinforcement learning,” Master’s thesis, 2017.

T. Okuyama, T. Gonsalves and J. Upadhay, "Autonomous Driving System based on Deep Q Learnig," 2018 International Conference on Intelligent Autonomous Systems (ICoIAS), Singapore, 2018, pp. 201-205, doi: 10.1109/ICoIAS.2018.8494053.

A. R. Fayjie, S. Hossain, D. Oualid and D. Lee, "Driverless Car: Autonomous Driving Using Deep Reinforcement Learning in Urban Environment," 2018 15th International Conference on Ubiquitous Robots (UR), Honolulu, HI, 2018, pp. 896-901, doi: 10.1109/URAI.2018.8441797.

Zhang, Yi & Sun, Ping & Yin, Yuhan & Lin, Lin & Wang, Xuesong. (2018). Human like Autonomous Vehicle Speed Control by Deep Reinforcement Learning with Double Q-Learning. 1251-1256. 10.1109/IVS.2018.8500630.

J. Chen, B. Yuan and M. Tomizuka, "Model-free Deep Reinforcement Learning for Urban Autonomous Driving," *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Auckland, New Zealand, 2019, pp. 2765-2771.

W. Gao, A. Odekunle, Y. Chen and Z. Jiang, "Predictive cruise control of connected and autonomous vehicles via reinforcement learning," in *IET Control Theory & Applications*, vol. 13, no. 17, pp. 2849-2855, 26 11 2019.

Richard S. Sutton and Andrew G. Barto (2018). Reinforcement learning: An introduction. Second Edition. MIT press.

Sutton, Richard. (1988). Learning to Predict by the Method of Temporal Differences. Machine Learning. 3. 9-44. 10.1007/BF00115009.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," CoRR, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

Watkins, C. J. (1989). Learning from delayed rewards. Ph.D. dissertation, University of Cambridge England.

A. Gosavi. A Tutorial for Reinforcement Learning, Department of Engineering Management and Systems Engineering, Missouri University of Science and Technology, Rolla, 2019

Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In *11th International Conference on Machine Learning*, 157–163 (1994)

Puterman, M.L. (1994). Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA, 1st Edn**.**

Fabiani, Patrick & Teichteil-Königsbuch, Florent. (2010). Markov Decision Processes in Artificial Intelligence. 10.1002/9781118557426.ch13.

Smallwood, Sondik (1973). The optimal control of partially observable markov processes over a finite horizon. Operations Research, 21, 1071–1088.

Wikipedia contributors. "Partially observable Markov decision process." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 5 Apr. 2020. Web. 14 Apr. 2020.

Chen Xia. Intelligent Mobile Robot Learning In Autonomous Navigation. Automatic Control Engineering. Ecole Centrale De Lille, 2015.

Bertsekas, D.P. (1996). Dynamic programming and optimal control, vol. 1. Athena Scientific Belmont, Massachusetts.

Li, Qizhen & Zhao, Lianwen & Gao, Jie & Liang, Hongbin & Zhao, Lian & Tang, Xiaohu. (2018). SMDP-Based Coordinated Virtual Machine Allocations in CloudFog Computing Systems. IEEE Internet of Things Journal. PP. 1-1. 10.1109/JIOT.2018.2818680.

Wikipedia contributors. "Q-learning." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 19 Apr. 2020. Web.

D. Pandey and P. Pandey, "Approximate Q-Learning: An Introduction," 2010 Second International Conference on Machine Learning and Computing, Bangalore, 2010, pp. 317-320.

Rummery, G. & Niranjan, Mahesan. (1994). On-Line Q-Learning Using Connectionist Systems. Technical Report CUED/F-INFENG/TR 166.

XU, Zhi-xiong & CAO, Lei & Xiliang, Chen & LI, Chen-xi & ZHANG, Yong-liang & LAI,Jun. (2018). Deep Reinforcement Learning with Sarsa and Q-Learning: A Hybrid Approach. IEICE Transactions on Information and Systems. E101.D. 2315-2322.10.1587/transinf.2017EDP7278

M. Zhou, Renforcement Learning with Tensorflow, https //github.com/ MorvanZho /Reinforcement-learning-with-tensorflow,[Online; accessed 04-09-2020], 2018.

J. Wu, S. Shin, C. Kim and S. Kim, "Effective lazy training method for deep q-network in obstacle avoidance and path planning," *2017 IEEE International Conference on*

Systems, Man, and Cybernetics (SMC), Banff, AB, 2017, pp. 1799-1804, doi: 10.1109/SMC.2017.8122877.

Mnih, Volodymyr & Kavukcuoglu, Koray & Silver, David & Rusu, Andrei & Veness, Joel & Bellemare, Marc & Graves, Alex & Riedmiller, Martin & Fidjeland, Andreas & Ostrovski, Georg & Petersen, Stig & Beattie, Charles & Sadik, Amir & Antonoglou, Ioannis & King, Helen & Kumaran, Dharshan & Wierstra, Daan & Legg, Shane & Hassabis, Demis. (2015). Human-level control through deep reinforcement learning. Nature. 518. 529-33. 10.1038/nature14236.

Xavier Glorot, Yoshua Bengio. (2010) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings 9:249-256.

APPENDIX

A.1 Artificial Neural Network

ANN is an information processing system taken from biological neural networks analogy. This neural network is also called deep learning. Every artificial neural network can be called deep learning, not every deep learning is ANN. Deep learning models include Convolution Neural Network, (CNN) Recurrent Neural Network (RNN) and Generative Adversarial Network (GAN). Logistic regression, which is a machine learning algorithm and used to model the probability of a class or event, is the basis of the artificial neural network. it can be mentioned as the simplest neural network. The difference from logistic regression is that it contains one or more hidden layers. The concept of deep learning is said to change according to the hardware features of the computer. While it can be considered twenty layers deep years ago, it has reached hundreds and thousands of layers of neural networks in recent years. The factor that determines the number of layers in the neural network is the number of hidden and output layers.

If a two-layer neural network is explained on the following figure;

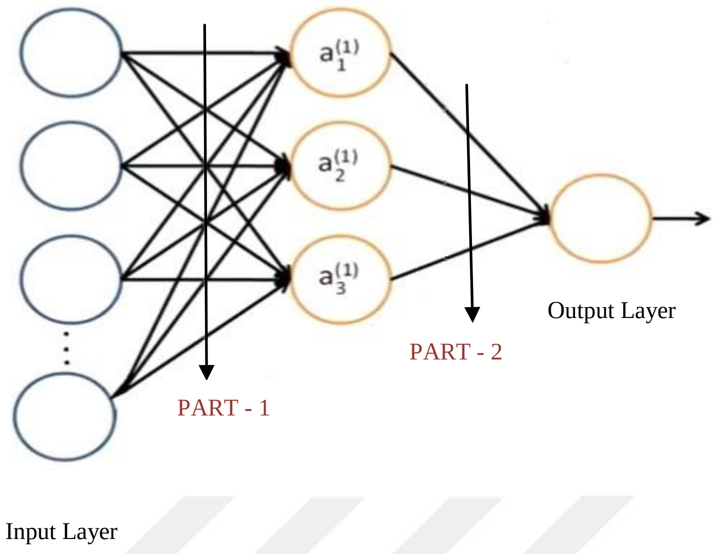


Figure A.1. Two-layer neural network structure

The number of nodes in the hidden layer is optional. Number of nodes, the number of hidden layer and learning rate are hyper parameters. Unlike the logistic function, ANN also uses the tan-h function as an activation function. The output of the tan-h function takes values between -1 and 1. This function is preferred for hidden layer. In binary classification, the sigmoid function is used only in the output layer. Since the average of the tan-h function is closer to 0, it can be said to be better than the sigmoid function. This indicates that it holds the data better at the center and it means that it is not biased to one side. PART – 1:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \tanh(Z^{[1]})$$

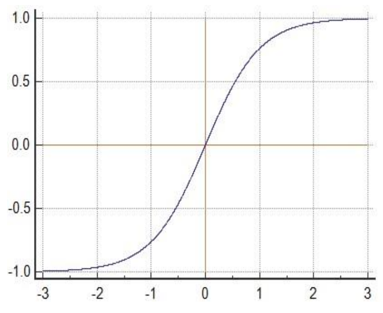


Figure A.2. tanh function

PART – 2:

$$Z^{[2]} \quad W^{[2]}A^{[2]} + \quad = \quad b^{[2]}$$

$$A^{\ 2]} = \sigma(Z^{[2]}) \qquad [$$

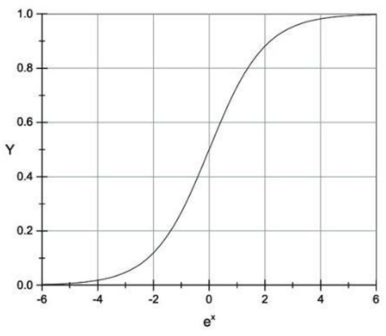


Figure A.3. sigmoid(σ) function

$A^{[1]}$: a value obtained in hidden layer , Z : a variable resulting from calculations
X : input parameter $W^{[1]}, W^{[2]}$:weight, $b^{[1]}, b^{[2]}$:bias
The reason for using the activation function is to increase the non-linearity in the model learned from the data. Non-linearity is considered to be a complex feature, so the data is learned so well.

When the 2-layer neural network is sampled, the following steps are applied to create a model for a neural network.

1. Size of layers and initializing parameters (weights and bias): Weight parameter is defined as 0.01, while bias is defined as zero. The reason why zero is not given to weight is that after the certain iteration, the gradient descent does not make any difference. It always starts to calculate the same thing and is no different from linear regression, so that diversity is not achieved. Weights are randomly identified with small values to contribute to diversity and learning different things. If the value given to weight is high, the result of the tanh function will be close to 1 or 1. And since the derivative of the curve close to 1 is 0, the update is very slow.

2. Forward propagation: As the name implies, input data is fed forward through the network. Each hidden layer accepts the input data, processes it according to the activation function and switches to the consecutive layer. Input data should only be fed forward to produce some output. The data should not flow in the opposite direction during output

production, otherwise it creates a loop and the output can never be produced. Such network configurations are known as feed-forward networks. Feed forward network helps propagate forward.

3. Loss function and cost function: A loss function is used to optimize parameter values in a neural network model. The loss functions map a set of parameter values for the network to a scalar value that indicates how well this parameter accomplished the task the network intended to perform. A cost function is a measure of how well a neural network is compared to the training example and expected output. It may also depend on variables such as weight and bias. Cost function consists of a single value, not a vector, because it rates how well the neural network is as a whole.

4. Backward propagation: Backward propagation is performed to update weight 1, bias 1, weight 2 and bias 2 values. Derivatives of the cost function are obtained by weight and then weights are updated in a certain rule in accordance with these derivatives. Backward propagation is a way to propagate the total loss back to the neural network to know how responsible for the loss of each node, and then updates the weights to minimize the loss by giving the nodes higher error and vice versa.

5. Update parameters: After changing the parameters in backward propagation, these parameters are updated. Learning rate is usually a parameter to be achieved by trying, but generally learning rate value is selected as 0.01 value of default.
After the parameters are updated, the learning part is completed and the model is ready. Structures that make up the model are weight and bias parameters. After the parameters are updated and the learning process occurs the learning part in deep learning is completed and this means that there is a prediction model available for test.

6. Prediction with learnt parameters weight and bias: At this learning stage, the network is trained by adjusting the weights to estimate the correct class label of the input samples. The advantages of neural networks include their high tolerance to noisy data, as well as their ability to classify patterns they are not trained for.

7. Formation of the model: As a result of the steps mentioned above, the model is formed. The x_train, y_train which are the input parameters in the model are the parameters required to test the model and to update the weight, bias (for the realization

of learning). Testing the learned parameters means the prediction. The number of iterations indicates how long the learning process will take.

A.2. The Generated Codes in Python

```
import tensorflow as tf
import pandas as pd import
numpy as np import os
import time import
matplotlib.pyplot as plt class
Sarsa():
   def __init__(self, env, gamma=0.99, learning_rate=0.1,
learning_rate_decay=0.000001, epsilon=0.1,
epsilon_decay=0.000001):
        self.env = env         self.gamma = gamma
self.learning_rate = learning_rate
self.learning_rate_decay = learning_rate_decay
        self.epsilon = epsilon
self.epsilon_decay = epsilon_decay
self.num_actions = env.action_dim
self.num_states = env.state_dim
        self.Q = np.zeros((self.num_states, self.num_actions))     def
train(self, num_episodes, num_steps, winning_reward, num_wins,
q_table_csv_filepath, visibility=False):         print("\nTraining using
SARSA algorithm\n")         list_of_rewards = []         win_count = 0
for i in range(num_episodes):
            s_t = self.env.reset()
sum_reward = 0             self.epsilon -=
self.epsilon_decay             if
np.random.uniform() > self.epsilon:
a_t = np.argmax(self.Q[s_t, :])             else:
                a_t = self.env.sample_action()
for j in range(num_steps):                  if
visibility:
                    self.env.render()                 s_t1,
r_t, done = self.env.step(a_t)
sum_reward += r_t                if
np.random.uniform() > self.epsilon:
a_t1 = np.argmax(self.Q[s_t1, :])
else:
                    a_t1 = self.env.sample_action()              self.learning_rate
-= self.learning_rate_decay              self.Q[s_t, a_t] = self.Q[s_t, a_t] +
self.learning_rate * (                  (r_t + self.gamma * self.Q[s_t1, a_t1]) -
self.Q[s_t, a_t])              s_t = s_t1              a_t = a_t1              if
done:                  break            print(" Episode {:5d}/{:5d} Step
{:4d}/{:4d} Training Reward {:4d}.".              format(i + 1,
num_episodes, j + 1, num_steps, sum_reward))            if sum_reward
>= winning_reward:
```

```
                win_count += 1
if win_count >= num_wins:
break             else:
                win_count = 0
list_of_rewards.append(sum_reward)          if
visibility:
            self.env.close()
        plt.plot(range(len(list_of_rewards)), list_of_rewards)
        plt.title('SARSA - Training')
plt.xlabel('Number of episodes')
plt.ylabel('Sum of rewards')        plt.show()
df = pd.DataFrame(self.Q)
df.to_csv(q_table_csv_filepath, index=False)
print("\n")
        print("-----------------------------------------------------------------")
print("Training completed and values of Q-table have been stored on." +
q_table_csv_filepath + " file.")
        print("-----------------------------------------------------------------")


    def test(self, num_test_episodes, num_steps, winning_reward,
q_table_csv_filepath, test_epsilon=0.05, delay_per_episode=1.0,
visibility=False):        if(os.path.exists(q_table_csv_filepath)):
            print("\nTesting of Q-table from SARSA algorithm\n")
with open(q_table_csv_filepath) as q_table:
                df_q = pd.read_csv(q_table)
q_arr = df_q.to_numpy()
win_count = 0             total_reward = 0
list_of_rewards = [ ]             for i in
range(num_test_episodes):
                s_t = self.env.reset()
accum_reward = 0                 for j
in range(num_steps):                      if
visibility:
                        self.env.render()                    if
np.random.uniform() > test_epsilon:
                        a_t = np.argmax(q_arr[s_t, :])
else:
                        a_t = self.env.sample_action()
s_t1, r_t, done = self.env.step(a_t)
accum_reward += r_t
                    s_t = s_t1                    if done:                     break
print(" Episode {:4d}/{:4d} Step {:4d}/{:4d} Testing Reward {:4d}.".
                      format(i + 1, num_test_episodes, j + 1, num_steps,
accum_reward))
                total_reward += accum_reward
list_of_rewards.append(accum_reward)                 if
accum_reward >= winning_reward:
                    win_count        +=         1
time.sleep(delay_per_episode)             if
visibility:
```

```
                self.env.close()
plt.plot(range(len(list_of_rewards)), list_of_rewards)
            plt.title('SARSA - Testing')
plt.xlabel('Number of episodes')
plt.ylabel('Sum of rewards')
            plt.show()
print("\n")
            print("-------------------------------------------------------")
print("Winning ratio for a reward higher than winning reward is:" +
"{:3.2f} %".format((win_count / num_test_episodes) * 100))
print("Average reward for {:4d} episodes is {:3.2f}".format(
num_test_episodes, total_reward / num_test_episodes))            print("----
---------------------------------------------------")        else:            print("CSV
file not found for Q-table. Check the path of file.")


class Q_learning():
    def __init__(self, env, gamma=0.99, learning_rate=0.1,
learning_rate_decay=0.000001, epsilon=0.1,
epsilon_decay=0.000001):        self.env = env
self.gamma = gamma        self.learning_rate =
learning_rate        self.learning_rate_decay =
learning_rate_decay
        self.epsilon = epsilon        self.epsilon_decay =
epsilon_decay        self.num_actions = env.action_dim
self.num_states = env.state_dim        self.Q =
np.zeros((self.num_states, self.num_actions))

    def train(self, num_episodes, num_steps, winning_reward, num_wins,
q_table_csv_filepath, visibility=False):
        print("\nTraining using Q-learning algorithm\n")
win_count = 0        list_of_rewards = []        for i
in range(num_episodes):
            s_t = self.env.reset()
sum_reward = 0            self.epsilon -=
self.epsilon_decay            for j in
range(num_steps):                if visibility:
                    self.env.render()                    if
np.random.uniform()          >          self.epsilon:
a_t = np.argmax(self.Q[s_t, :])                else:
                    a_t = self.env.sample_action()
s_t1, r_t, done = self.env.step(a_t)
                sum_reward += r_t
                self.learning_rate -= self.learning_rate_decay
self.Q[s_t, a_t] = self.Q[s_t, a_t] + self.learning_rate * (                    (r_t
+ self.gamma * np.max(self.Q[s_t1, :])) -                    self.Q[s_t, a_t])
s_t = s_t1                if done:                    break            print(" Episode
{:4d}/{:4d} Step {:4d}/{:4d} Training Reward {:4d}.".                format(i +
1, num_episodes, j + 1, num_steps, sum_reward))            if sum_reward
>= winning_reward:
```

```
                win_count += 1
if win_count >= num_wins:
break              else:
                win_count = 0
list_of_rewards.append(sum_reward)          if
visibility:
            self.env.close()
plt.plot(range(len(list_of_rewards)), list_of_rewards)
        plt.title('Q_learning - Training')
plt.xlabel('Number of episodes')
plt.ylabel('Sum of rewards')         plt.show()
df = pd.DataFrame(self.Q)
df.to_csv(q_table_csv_filepath, index=False)
print("\n")
        print("------------------------------------------------------------")
print("Training completed and values of Q-table have been stored on." +
q_table_csv_filepath + " file.")          print("--------------------------------------------
-----------------")

    def test(self, num_test_episodes, num_steps, winning_reward,
q_table_csv_filepath, test_epsilon=0.05, delay_per_episode=1.0,
visibility=False):          if(os.path.exists(q_table_csv_filepath)):
            print("\nTesting of Q-table from Q-learning algorithm\n")
with open(q_table_csv_filepath) as q_table:
                df_q = pd.read_csv(q_table)
q_arr = df_q.to_numpy()
win_count = 0             total_reward = 0
list_of_rewards = []             for i in
range(num_test_episodes):
                s_t = self.env.reset()
accum_reward = 0                  for j
in range(num_steps):                      if
visibility:
                        self.env.render()                    if
np.random.uniform() > test_epsilon:
a_t = np.argmax(q_arr[s_t, :])                   else:
                        a_t = self.env.sample_action()
s_t1, r_t, done = self.env.step(a_t)
accum_reward += r_t
                    s_t = s_t1                    if done:                         break
print(" Episode {:4d}/{:4d} Step {:4d}/{:4d} Testing Reward {:4d}.".
                      format(i + 1, num_test_episodes, j + 1, num_steps,
                             accum_reward))
total_reward += accum_reward
list_of_rewards.append(accum_reward)
if accum_reward >= winning_reward:
                    win_count          +=          1
time.sleep(delay_per_episode)               if
visibility:
```

```
                self.env.close()
plt.plot(range(len(list_of_rewards)), list_of_rewards)
            plt.title('Q_learning - Testing')
plt.xlabel('Number of episodes')             plt.ylabel('Sum
of rewards')
            plt.show()
print("\n")
            print("-------------------------------------------------------")
print("Winning ratio for a reward higher than winning reward is:" +
"{:3.2f} %".format((win_count / num_test_episodes) * 100))
print("Average reward for {:4d} episodes is {:3.2f}".format(
num_test_episodes, total_reward / num_test_episodes))             print("----
---------------------------------------------------")         else:            print("CSV
file not found for Q-table. Check the path of file.")

class Memory():     def __init__(self,
memory_size, data_dim=5):
        self.memory_size = memory_size
self.data_dim = data_dim
        self.experience = []

    def add_memory(self, instance):
if len(instance) == self.data_dim:
            if len(self.experience) < self.memory_size:
                self.experience.append(instance)
else:
                del self.experience[0]
self.experience.append(instance)

    def retrieve_randomly(self, batch_size):         samples_index
= np.random.randint(0, len(self.experience),
size=batch_size)
        random_samples = [ ]
for i in samples_index:
random_samples.append(self.e
xperience[i])         return
random_samples     def
retrieve_all(self):         return
self.experience     def
occupied_memory(self):
return len(self.experience)

class E_greedy_policy():     def __init__(self, decay_steps,
start_value=1.0, end_value=0.1):
        self.start_value = start_value         self.end_value = end_value
self.decay_steps = decay_steps         self.decay_rate =
(self.start_value-self.end_value)/self.decay_steps         self.e_value =
start_value
```

```
class DQN():
   def __init__(self, env, train_model, target_model, optimizer,
gamma=0.99):
        self.env = env
self.train_model = train_model
self.target_model = target_model
self.optimizer = optimizer
        self.gamma = gamma

   def train(self, num_episodes, num_steps, memory_replay_storage,
epsilon_decay_steps, min_update_steps, batch_size,
target_model_update_steps, winning_reward, num_wins,
saved_model_filepath, visibility=False):

        print("\nTraining using DQN algorithm\n")
exp_replay = Memory(memory_replay_storage)
        e_policy = E_greedy_policy(epsilon_decay_steps)

        train_reward_per_eps = []
total_steps = 0         win_count = 0
for i in range(num_episodes):
obs = self.env.reset()             obs =
obs.reshape(-1, obs.shape[0])
sum_reward = 0              for j in
range(num_steps):                 if visibility:
                    self.env.render()
total_steps += 1
                prev_obs = obs
                 if
e_policy.decision():
                    action = np.argmax(self.train_model(prev_obs))
else:
                    action = self.env.sample_action()
                obs, reward, done = self.env.step(action)
obs = obs.reshape(-1, obs.shape[0])

                exp_replay.add_memory([prev_obs, action, reward, done, obs])
sum_reward += reward                 if exp_replay.occupied_memory() >
min_update_steps:                    samples =
exp_replay.retrieve_randomly(batch_size)                     states = [ ]
targets = [ ]                  for p_state, act, rew, dn, n_state in samples:
                        q_val = rew + ((1 - dn) * self.gamma * np.max(
                            self.target_model(n_state)))
y_target = self.train_model(p_state)
y_temp = np.zeros(y_target.shape)
for k in range(y_temp.shape[1]):
y_temp[0][k] = y_target[0][k]
y_temp[0][act] = q_val
states.append(p_state[0])
targets.append(y_temp[0])                   all_states =
np.stack(states, axis=0)                    all_targets =
np.stack(targets, axis=0)                    with
tf.GradientTape() as tape:
                        all_actuals = self.train_model(all_states)
loss = tf.keras.losses.MSE(all_targets, all_actuals)
grads = tape.gradient(loss,
self.train_model.trainable_variables)
self.optimizer.apply_gradients(                         zip(grads,
self.train_model.trainable_variables))                  if total_steps %
target_model_update_steps == 0:
                    self.target_model.set_weights(
self.train_model.get_weights())                 if
done:                       break
            train_reward_per_eps.append(sum_reward)
            print("Episode: " +
                  "{:4d}/{:4d} Steps: {:4d}/{:4d} Training Reward: {:4d}".
                  format(i+1, num_episodes, j+1, num_steps, sum_reward))
if sum_reward > winning_reward:
                win_count += 1
if win_count >= num_wins:
break               else:
                win_count = 0

        self.train_model.save(saved_model_filepath)
print("\n")
        print("-------------------------------------------------------------")
print("Training completed and values of Q-table have been stored on." +
saved_model_filepath + " file.")         print("-------------------------------------------
------------------")         if visibility:
            self.env.close()         plt.plot(range(len(train_reward_per_eps)),
train_reward_per_eps)         plt.title('DQN - Training')
plt.xlabel('Number of episodes')         plt.ylabel('Sum of rewards')
plt.show()

    def test(self, num_test_episodes, num_steps, winning_reward,
             saved_model_filepath, test_epsilon=0.05,
delay_per_episode=1.0, visibility=False):
if(os.path.exists(saved_model_filepath)):
            print("\nTesting model from DQN algorithm\n")
model = tf.keras.models.load_model(saved_model_filepath)
reward_per_eps = []             total_sum = 0             win_counter =
0             for i in range(num_test_episodes):
                obs = self.env.reset()
sum_reward = 0                  for j in
range(num_steps):                     if
visibility:
                        self.env.render()                   if
np.random.uniform() > test_epsilon:
                        y_pred = model(obs.reshape(-1, obs.shape[0]))
action = np.argmax(y_pred)                    else:
```

```
                        action = self.env.sample_action()
obs, reward, done = self.env.step(action)
sum_reward += reward                         if done:
break                  print("Episode: " +
                      "{:4d}/{:4d} Steps: {:4d}/{:4d} Testing Reward: {:4d}".
format(i+1, num_test_episodes, j+1, num_steps,
                             sum_reward))                  if
sum_reward >= winning_reward:
win_counter +=1
time.sleep(delay_per_episode)
reward_per_eps.append(sum_reward)
total_sum += sum_reward

            average_reward = total_sum / num_test_episodes
win_ratio = (win_counter / num_test_episodes) * 100              if
visibility:
                self.env.close()
plt.plot(range(len(reward_per_eps)), reward_per_eps)
            plt.title('DQN - Testing')
plt.xlabel('Number of episodes')
plt.ylabel('Sum of rewards')
            plt.show()
print("\n")
```

CURRICULUM VITAE

She received her B.Sc. degree in Electrical & Electronics Engineering in 2015 from Fırat University. She was an exchange student in third year of the B.Sc. in Lillebaelt Academy of Professional Higher Education, Odense, Denmark. After working as an electronics engineer in the private sector for 2 years, she continued her M.Sc. degree in Electronics Engineering at Kadir Has University. She continues her profession as an electronic engineer in the Ministry of Defense. Her research interests include deep learning, machine learning and autonomous vehicles.