MEF UNIVERSITY

QPICAR – DEEP Q-LEARNING

Capstone Project

Özge Beğde

İSTANBUL, 2021

MEF UNIVERSITY

QPICAR – DEEP Q-LEARNING

Capstone Project

Özge Beğde

Advisor: Asst. Prof. Tuna Çakar

İSTANBUL, 2021

MEF UNIVERSITY

Name of the project: QPICAR – Deep Q-Learning
Name/Last Name of the Student: Özge Beğde
Date of Thesis Defense: 25/01/2021

I hereby state that the graduation project prepared by Özge Beğde has been completed under my supervision. I accept this work as a "Graduation Project".

25/01/2021
Asst. Prof. Tuna Çakar

I hereby state that I have examined this graduation project by Özge Beğde which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

25/01/2021

Prof. Dr. Özgür Özlük
Director
of
Big Data Analytics Program

We hereby state that we have held the graduation examination of Özge Beğde and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. Asst. Prof. Tuna Çakar	……..
2. Prof. Dr. Özgür Özlük	……..

Academic Honesty Pledge

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Özge Beğde 25/12/2021

EXECUTIVE SUMMARY

QPICAR – DEEP Q-LEARNING

Özge Beğde

Advisor: Asst. Prof. Tuna Çakar

JANUARY, 2021, 25 pages

The aim of the project is to train a smart tool kit named "Sunfounder Raspberry Pi Robot Car" to move without hitting the walls in a closed area. The goal is to maximize the driving time without crashing by reducing the number of hits. Ultrasonic sensor data collected from the vehicle are processed with reinforcement learning and deep reinforcement learning algorithms and the results are compared. In this study, Python programming language is used.

In this study, firstly, the Q-Learning method, which is a reinforcement learning algorithm based on Markov decision processes, is used. The method basically relies on a memory table, Q-Table, in which the Q-values of the agent moving from one state to another are kept. This table is updated according to the results of the Bellman equation in every action of the agent, and as a result of this iterative process, it is optimized to provide that the agent moves to maximize its rewards.

Deep Q-Learning (DQN) is used as a deep reinforcement learning algorithm. This algorithm was developed by the DeepMind Technologies team in 2013. Basically, it is based on the use of the Bellman equation, which is an element of the Q-Learning method, in combination with neural networks. This method is often used for training agents in complex and multidimensional environments such as video games. Due to the different type of the data used on the algorithm, minor changes were made to adapt it to the study. RElu and Softplus are used as activation functions.

The results of the training process show that the DQN algorithm has an important advantage in terms of training the agent in a short time. At this point, the results are in accordance with other academic studies demonstrating the success of the DQN algorithm for complex environments.

For future work, by differentiating the equipment that collects data on the vehicle, different data types such as image, temperature value, oxygen value can be collected and processed. At the same time, with changes to the reward setup in the algorithm, the agent can be trained to move to a specific target or to take actions to avoid a specific target.

Key Words: Machine learning, Reinforcement learning, Q-learning, Deep reinforcement learning.

ÖZET

QPICAR – DEEP Q-LEARNING

Özge Beğde

Proje Danışmanı: Asst. Prof. Tuna Çakar

OCAK, 2021, 25 sayfa

Projenin amacını, "Sunfounder Raspberry Pi Robot Car" ismindeki akıllı bir araç kitinin kapalı bir alan içerisinde çeperlere çarpmaksızın hareket etmesini sağlayacak biçimde eğitilmesi oluşturmaktadır. Hedef çarpma sayılarını azaltarak çarpmaksızın geçen sürüş süresini maksimize etmektir. Araçtan toplanan ultrasonik sensör verileri pekiştirmeli öğrenme ve derin pekiştirmeli öğrenme algoritmaları ile işlenmiş ve sonuçlar karşılaştırılmıştır. Çalışma kapsamında Python programlama dili kullanılmıştır.

Bu çalışmada ilk olarak bir pekiştirmeli öğrenme algoritması olan ve Markov karar süreçlerine dayanan Q-Learning yöntemi kullanılmıştır. Metodun temelinde bir konumdan diğerine geçişin ödüllerinin tutulduğu bir hafıza tablosu yer almaktadır. Bu tablo, aracın her aksiyonunda Bellman denklemi ile gelen sonuçlara göre güncellenmekte ve bu yinelemeli sürecin sonucunda aracın ödüllerini maksimize edecek biçimde hareket etmesini sağlayacak şekilde optimize olmaktadır.

Derin pekiştirmeli öğrenme algoritması olarak ise Deep Q-Learning (DQN) kullanılmıştır. Bu algoritma 2013 yılında DeepMind Technologies ekibi tarafından geliştirilmiştir. Temelde, Q-Learning metodunun bir unsuru olan Bellman denkleminin nöral ağlar ile kombine edilerek kullanılmasına dayanmaktadır. Bu yöntem genellikle ajanların video oyunları gibi karmaşık ve çok boyutlu ortamlarda eğitilmesi için kullanılmaktadır. Algoritma üzerinde, kullanılan verilerin farklı nitelikte olması sebebiyle, çalışmaya uyarlamak adına minör değişiklikler gerçekleştirilmiştir. Aktivasyon fonksiyonu olarak RElu ve Softplus kullanılmıştır.

Eğitim sürecinin sonucunda, DQN algoritmasının aracın kısa sürede eğitilebilmesi adına önemli avantaja sahip olduğu görülmektedir. Bu noktada edinilen sonuçlar DQN algoritmasının karmaşık ortamlar için başarısını ortaya koyan diğer akademik çalışmalar ile uyum içerisindedir.

Aracın üzerindeki data toplayan donanımın farklılaştırılması ile görüntü, sıcaklık değeri, oksijen değeri gibi farklı veri türlerinin toplanması ve işlenmesi sağlanabilir. Aynı zamanda, algoritmada yer alan ödül kurgusunda yapılacak değişiklikler ile ajan belirli bir hedefe hareket etme ya da belirli bir hedeften kaçınma aksiyonlarını alacak şekilde eğitilebilir.

Anahtar Kelimeler: Makine öğrenmesi, Pekiştirmeli öğrenme, Q-öğrenme, Derin pekiştirmeli öğrenme.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Self-learning systems are one of the newest and most popular artificial intelligence technologies. The characteristic of a self-learning system is to learn based on experience, without explicitly programmed to perform a task. By using data, the system recognizes patterns and optimizes efficiency and accuracy. The system starts with a limited information and increases it along the way by using past experience. And also, it does not depend on a single environment, it is able to adapt to different situations.

In machine learning field, there are different kind of self-learning methods. In this study, for a simple self-learning case, one of the most recent methods of deep reinforcement learning algorithms is used and compared with a reinforcement learning algorithm.

The aim of the project is to train a smart robot car kit, which has Raspberry Pi and ultrasonic sensors on it, to move in a closed area without crashing. The goal is to collect the data from ultrasonic sensors on the car to run algorithms. The qualifications of agent and the environment in which the robot moves, and the methods are explained in report elaborately. In this project, self-learning methods outside the field of machine learning are excluded.

2. THEORY

2.1. Machine Learning

Machine learning is a branch of artificial intelligence. Machine learning systems has the ability to learn automatically by using past experience or data, and there is no need to be programmed explicitly. The models created by machine learning algorithms can be used for making future predictions, classifications or gaining knowledge from data. (Alpaydın, 2004)

Machine learning algorithms are divided into four fields such as supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning algorithms. The purpose of supervised learning is to map an input to a correct output. Labels provide the correct values of output data as a supervisor. On the other hand, in the unsupervised learning, there is only input data and it is aimed to find patterns in it. In semi-supervised learning both labeled and unlabeled data are used combinedly. Sometimes outputs are not just data points. For applications whose output is an action, it is important to obtain the sequence of actions that form the strategy to achieve the goals rather than the correct value of a single input. Reinforcement learning algorithms are used for these cases and aim to generate a strategy by learning from past good action sequence. (Alpaydın, 2004) Reinforcement learning algorithms, which are the main issue of this study, will be discussed in detail in the next chapter.

2.2. Reinforcement Learning

2.2.1. Mathematical background of reinforcement learning

Reinforcement learning is an area of machine learning dealing with sequential decision problems. A sequential decision problem emerges in a stochastic environment which the utility of the system is affected by a sequence of decisions. (Russell&Norvig, 2010) It is concerned with how an agent should take actions in an environment in order to generate a policy which provides for reaching to the ultimate goal. Its main argument is learning from interaction. Secondly, at every step the agent only knows the current state and immediate reward, but it has no information whether it is the best action for long-term objectives or not. It differentiates from other machine learning algorithms in these respects. (Kaelbling, 1996)

Reinforcement learning method is based on Markov Decision Process (MDP), a discrete time stochastic control process, in which the probable next state depends only on the current state. (Mohit, 2019) As it is stated by Marco Weiring and Martijn Van Otterlo (2012) "MDPs have become the de facto standard formalism for learning sequential decision making." (p. 4)

Basic concepts of MDP are the state transition probability function, the reward function and the discounting factor. The state transition probability function, $P_a(s, s')$ where a is an action, s is a state and s′ is the next possible state, gives the probabilities of moving from the current state to any of the next possible states by taking an action. The reward function, $R_a(s, s')$ gives the reward of transitioning from the current state to next state conditioned on the action taken. The discounting factor, γ, which is a real number between 0 and 1, discounts future rewards to the present step. (Mohit, 2019) The discount factor balances agent's will between taking immediate and future rewards. System tries to create a long-term strategy by reviewing not just the immediate reward but also the rewards and penalties of next states and actions. (Sutton&Barto, 2014)

The policy, $\pi(a|s)$**,** is the probability that agent takes an action a in the state s. The policy is the distribution over all actions for each state. Finding a policy to maximize the total sum of rewards, discounted by discounting factor, is the main objective of the MDP or a Reinforcement Learning agent under MDP. These cumulative discounted rewards are referred to as the objective function. Thus, solving an MDP problem is to find the policy which maximizes the objective function, which is given in Figure 1. (Mohik, 2019)

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$$

Figure 1: Objective Function.

The function, which gives the expected value of both the present and discounted future rewards, is called the value function. If the action is added to this equation the action-value function will be obtained. The action-value function is also known as Q-function. The Bellman equation, given in Figure 2, gives mathematical solution of both the value function and Q-function. (Mohit, 2019) Q-function is the main concept of Q-Learning method.

$$Q_\pi(s,a) = \sum_{s'} \pi_{(s,s')} \left[R_{sas'} + \gamma \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i r_{t+k+2} |_{s_{t+1}=s'} \right] \right]$$

Figure 2: Action - value function or Q-function.

As it is mentioned by Mohit Sewak (2019), "This equation simply says that the action–value/Q-value of a given action when the agent is in a particular state is the sum of the expectancy of discounted rewards of each state that could be reached by taking this particular action when the agent is in the given current state, weighed by the probability of reaching that new state from the given current state by taking the specified action in consideration." (p. 23)

To sum up, the agent's objective is to maximize expected return. To achieve this, the agent has to have a method to measure whether a state or action is good or not. At this point, the value function comes into play. It estimates how good it is for an agent to be in a given state or to perform a given action in a given state. The value function is defined with respect to the expected return, which is related to the action of agent. The agent acts under the policy. Thus, action-value function can be taken as the values of an action under policy π.

The aim of the reinforcement learning algorithms is to state the best action sequence for the agent. For this purpose, action-value function is estimated by using Bellman's equation, (Mnih et al., 2015) In the following chapter, the basic process of the reinforcement learning algorithms will be explained.

2.2.2. Components of reinforcement learning

Basic mechanism of reinforcement learning method is given in the Figure 3.

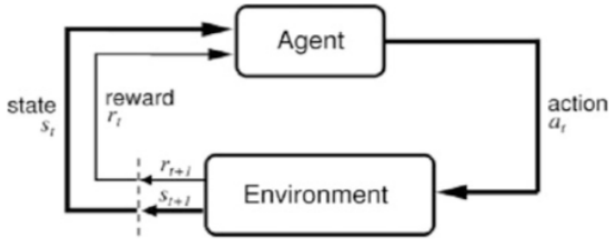

Figure 3: Basic mechanism of reinforcement learning methods.

Essential components of the reinforcement learning process, as can be seen from Figure 3, are listed as agent, environment, action, state, rewards and policy. (Venkatachalam,

2020) The agent is the main component. It aims to learn to take the best possible action under a given state by interacting with the environment. The environment presents different possible states that agent can take. State is presented by environment to the agent in order to take an action. Action refers the transition from a current state to the possible next state. Reward is generated depending on both state and action. (Mohit, 2019) The reward function can differ related to the objectives to be achieved. It can be defined by using imitation learning as well as by providing human feedback. (François-Lavet et al., 2018)

The Figure 3 can be simply explained as follows. At time t, agent is in a given state. According to the policy, the agent takes an action and moves to the next state. It takes the reward or penalty with respect to the action. The action-value function is calculated according to Bellman's equation.

There are different reinforcement learning methods. Some of these methods can be listed as Temporal Difference Learning, Q-Learning, State-Action-Reward-State-Action (SARSA) and Watkin's Q. (Sutton&Barto, 2014) In the scope of the study, only Q-Learning, which is an off-policy temporal difference control algorithm, will be mentioned in detail in the following chapter.

2.2.3. Q-Learning

Q-learning is a value-based and iterative reinforcement learning algorithm. The aim of value-based reinforcement learning algorithms is to learn the expected consequences of a certain action in a given state from experience. (Sutton&Barto, 2014)

The main concept of the algorithm is Q-Table. Q-Table can be taken as the memory of the agent and it consists all possible states and actions of agent. It is mostly initialized as empty and it is updated iteratively after every action. Q-Table is updated to the Q-function which is defined in "2.2.1 Mathematical background of reinforcement learning". Every state-action pair in the Q-Table has a Q-value which is the result of Q-function. For every action, agent checks Q-Table and chooses the highest Q-value according to the current state. With this iterative process, these Q-values converges to their optimal value at the end of the training session. Therefore, it can be said that, the goal of Q-Learning is to find a policy that is better than all other policies by learning the optimal Q-values of each state-action pair.

Important parameters for Q–Learning are the learning rate, the discount factor and exploration factor. These parameters determine the probability of the algorithm whether

relies on optimal future reward or immediate reward or choose a random action. (Venkatachalam, 2020) The discount factor is explained in "2.2.1 Mathematical background of reinforcement learning".

$$Q_{(s,a)} = (1-\alpha)Q_{(s,a)} + \alpha(r + \gamma \max_{a'} Q_{(s',a')})$$

Figure 4: Q-value of a given state s and action a.

While the first part of the equation in Figure 4 is mentioned as old value, the second part is called as learned value. The learning rate, α, determines how effective the learned value will be for Q-value. If the learning rate is so small, for example 0, then the learned value totally discarded and nothing new will be learned by the agent.

In order to understand exploration factor, denoted by ε, first "Explore vs Exploit dilemma" has to be explained. If the agent chooses actions only regarding the Q-values in the table, after a while, it may tend to act in certain patterns rather than realizing learning process. In order to avoid this problem, the agent will be allowed to act randomly from time to time. In this case, this random move is named as exploration and the action which is recommended by Q-Table is called exploitation.

One of the most known algorithms to provide balance between exploration and exploitation is epsilon greedy strategy. (Mohit, 2019) Epsilon is a real number between 0 and 1, and it denotes the probability of agent's taking an exploration action.

Q-Learning algorithm is very useful and efficient for simple environments with small state space. However, when the environment gets more complex, other algorithms should be used such as Deep Q-Learning (DQN). In following chapters, deep neural networks and DQN will be explained in detail.

2.2.4. Deep neural networks

Artificial Neural Networks (ANN) which are used for image processing, speech recognition, recommendation systems and so on., are the core idea of deep learning. An artificial neural network occurs from artificial neurons. When an ANN has two or more hidden layers, it is called a Deep Neural Network (DNN). (Geron, 2017) Compelling developments have been achieved in robotics also with deep learning algorithms. (Finn et al., 2015)

Every layer in the network has a set of nodes. The connection between each neural network layer has weights and an activation function that transforms the output of each node in a layer. ("Neural Networks: Structure", 2020) The main aim is to minimize cost or error function. Gradient descent and backpropagation algorithm are core concepts for neural networks. For each training instance the backpropagation algorithm measures the error and by passing all the network reversely measures the error contribution from each connection. And in the gradient descent step it changes the weights of connections in order to reduce error. (Geron, 2017)

2.2.5. Deep Q-Learning

The concept of Deep Q-Learning or Deep Q-Networks (DQN), which is a combination of Q-Learning algorithm and Deep Learning, is developed by Google Deep Mind research team in 2013. (Mnih et al., 2013) In 2015, the research team succeed to create a single DQN which can learn to play 49 different Atari games. Moreover, in 29 of 49 games, it gained scores that exceeded human performance. (Mnih et al., 2015)

The DQN method is mostly used for more sophisticated and complex environments with bigger state-spaces, in which Q-learning algorithm could be inefficient and infeasible. The input of the network is mostly high-dimensional sensory data like vision and speech, and output is value function estimating future rewards. (Mnih et al., 2015)

DQN, as it is mentioned earlier, is a combination of Q-Learning and Deep Learning. In this case, Deep Learning refers Convolutional Neural Networks (CNN), which is a widely used architecture for image processing. As can be seen in Figure 5, the input of neural network is sensor or visual data, and the output is the action of agent. (Mnih et al., 2015)

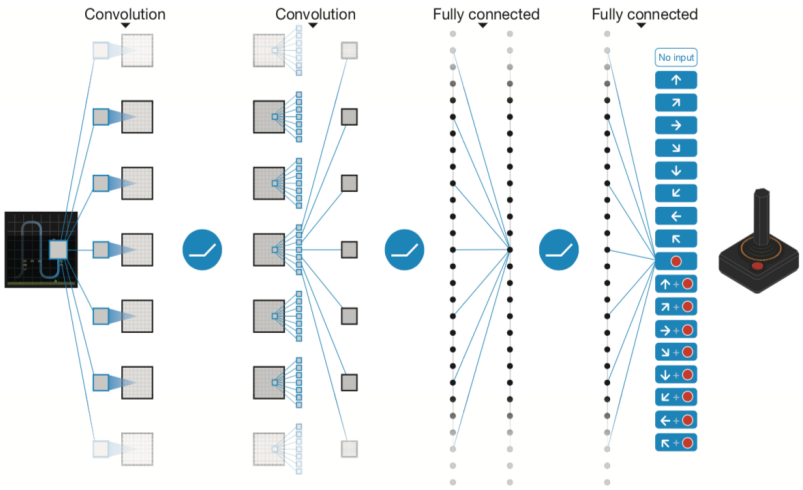

Figure 5: Convolutional Neural Networks schema.

In this study, the combination of ANN and Q-Learning is used. However, it is possible to train the agent by combining Q-Learning with a Recurrent Neural Network (RNN). (El Sallab et al., 2017)

One of the most important concepts of DQN is experience replay. For Q-Learning method, every (state, action, reward, next state) vector, which is used to train Q-function, is taken as an experience. Applying this concept directly to DQN can be problematic in some respects. First of all, while the agent learns the new behaviors, the distribution of data changes from classical reinforcement learning methods. This is a problem for deep learning algorithms that work with a fixed underlying distribution assumption. Secondly, in reinforcement learning successive experiences generates highly correlated sequences, that can cause imbalances for most of deep learning algorithms since they assume the data samples are independent. Experience replay is the method to deal with these issues. (Mnih et al., 2013) Replay memory size is determined at the beginning of training process.

Different from the Q-Learning algorithm, in DQN, experience tuples, generated from source, are collected in a memory-buffer rather than used directly. To train the agent, experience tuples are picked randomly from this pool. Thus, correlation of data is reduced, and distribution becomes smoother. (Mnih et al., 2013) Obviously, there are other techniques to choose experience tuples from memory-buffer, such as prioritized experience play. (Mohit, 2019) However, this subject is beyond the scope of the study.

The other important concepts of DQN are target network and policy network. In the earlier chapter it is stated that the first part of the Q-function refers the present state-action

value, while second part of it refers the target Q-value for the next state-action. While in classical Q-learning method these two parts are same, in DQN they are treated as different networks. In this case, there are two networks, Q-Network (or policy network) and target network. At every fixed number of steps, target network is updated with the actively updated Q-Network, and they become synced. (Mohit, 2019)

While Q-Learning algorithm is a value-based iterative process which compute Q-values directly to find the optimal Q-function, DQN is a non-linear function approximator used to estimate the Q-function.

3. METHODOLOGY

3.1. Agent and Environment

In this study, the agent that is trained is the Sunfounder Raspberry Pi Robot Car. It works with all Raspberry models; our agent has Raspberry Pi 3. It has 4 wheels and 1 ultrasonic obstacle avoidance module. 5 ultrasonic sensors are added on the car, thus it gathers sonic data from 6 ultrasonic sensors. It has ability to move forward, backward and to turn left and right. The robot is programmed by Python language.

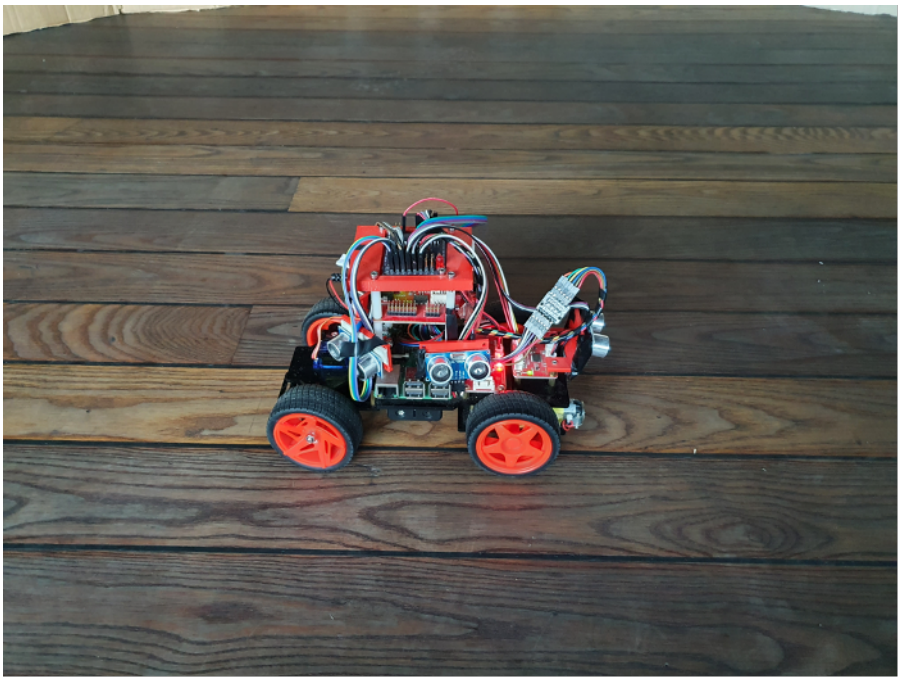

Figure 6: Sunfounder Raspberry Pi Robot Car.

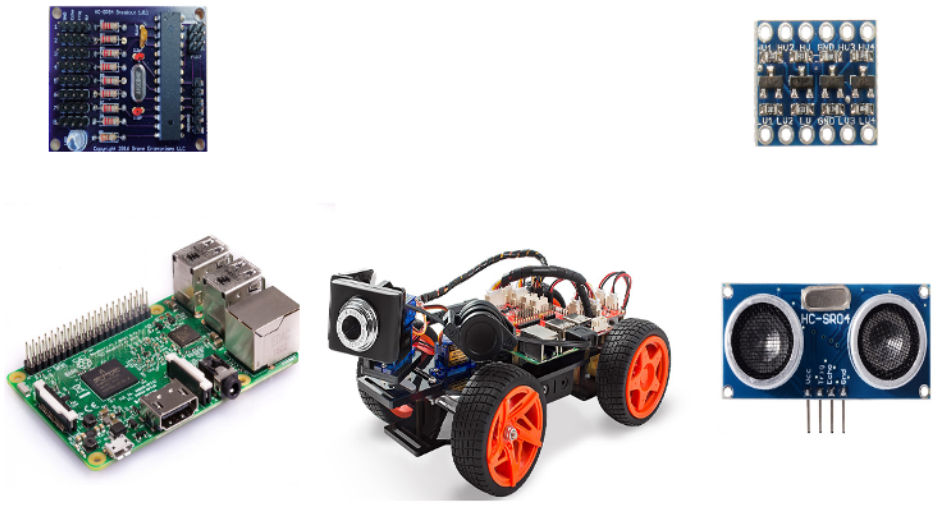

Figure 7: The hardware of Sunfounder Raspberry Pi Robot Car.

The environment in which the car moves is a closed circle with 145 cm diameter. Our aim is that the car moves in this circle without crashing walls. For our agent, environment is stochastic. It cannot observe the states, it just knows the distance from the sensor data and converts them to state. And it also knows the reward based on its action. The objective is to minimize crash number and maximize the time passing without crashing. In the following chapter the training process is explained in detail.

3.2. Training Process

Before the training process, the sensors of car are initialized and checked by using Octasonic module and actions are defined by using Picar and Picar.Sunfounder_PCA9685 modules.

First of all, crash distance, which is the minimum distance between sensor and walls that car is able to continue moving, is given as 12 cm and a random backward movement with 3 steps is defined for crash state. It means if the distance between car and walls drops below 12 cm, the car stops moving and makes a random movement to the backward. Actually, this random move rule is not necessary for learning methods. On the other hand, without this restriction, the agent would stop at every time it comes into crash position and it has to be carried to starting position by hand. And also, randomness contributes the exploration process of car. This requirement is preserved in the same way in both algorithms. Every crash situation is logged to review algorithm's performance.

3.2.1. Q-Learning algorithm

For Q-Learning training process, firstly, Q-Table is created with the number of total possible states and the number of actions. Every state is converted to index in order to make Q-Table simpler. Q-Table is initialized with zeros. At the end of every epoch, the system asks whether to save recent Q-Table or not. If the Q-Table is saved, it is an option to use old Q-Table without initializing a new one with zeros. Discount rate and learning rate is given as 0.1 and 0.01. The coefficient for randomness of chosen action, ε, is given as 0.7. In this study, decaying epsilon-greedy strategy is used. It means the ε value is decreasing 0.005 at every epoch. Thus, while in the earlier part of training process the agent moves randomly, after a while it has more willingness to exploitation.

Reward function is defined before training process. Since the goal of agent is to move forward without crashing walls, in other words running in circle, the reward of moving forward is the greatest one, while turning right and left have the same reward. When the distance between the car and the boundaries of the circle gets closer to crash distance, the agent takes penalty.

For every movement, first of all, the car gathers distance data via ultrasonic sensors. It checks the crash distance from the data gathered by front sensor. If it is in crash distance, it makes the random backward movement as mentioned earlier and the process starts over. If it is not, the sensor data is taken and converted to state, according to the conditions below;

If Distance <= Crash Distance, Then State=-1
If Crash Distance < Distance <= 20 cm, Then State=0
If 21 cm < Distance <= 32 cm, Then State=1
If 32 cm < Distance, Then State=2

After determining the state, according to ε of this epoch the car takes the best action or random action. The best action is found by checking the Q-values in the Q-Table. The agent picks the action with the highest Q-value related to its state-action pair. Therefore, the agent moves and takes a new state. While it is making the move, the reward is calculated according to the new state. Q-Table is updated using Bellman's equation. Thus, the first cycle is completed.

In each cycle the Q-values in the Q-table are updated. And in this way after a while, the Q-values of the expected state-action pairs in the table increase. Thus, Q-table is optimized in the way the agent avoids crashing, and the learning process is completed.

Workflow of Q-Learning is given in Figure 8.

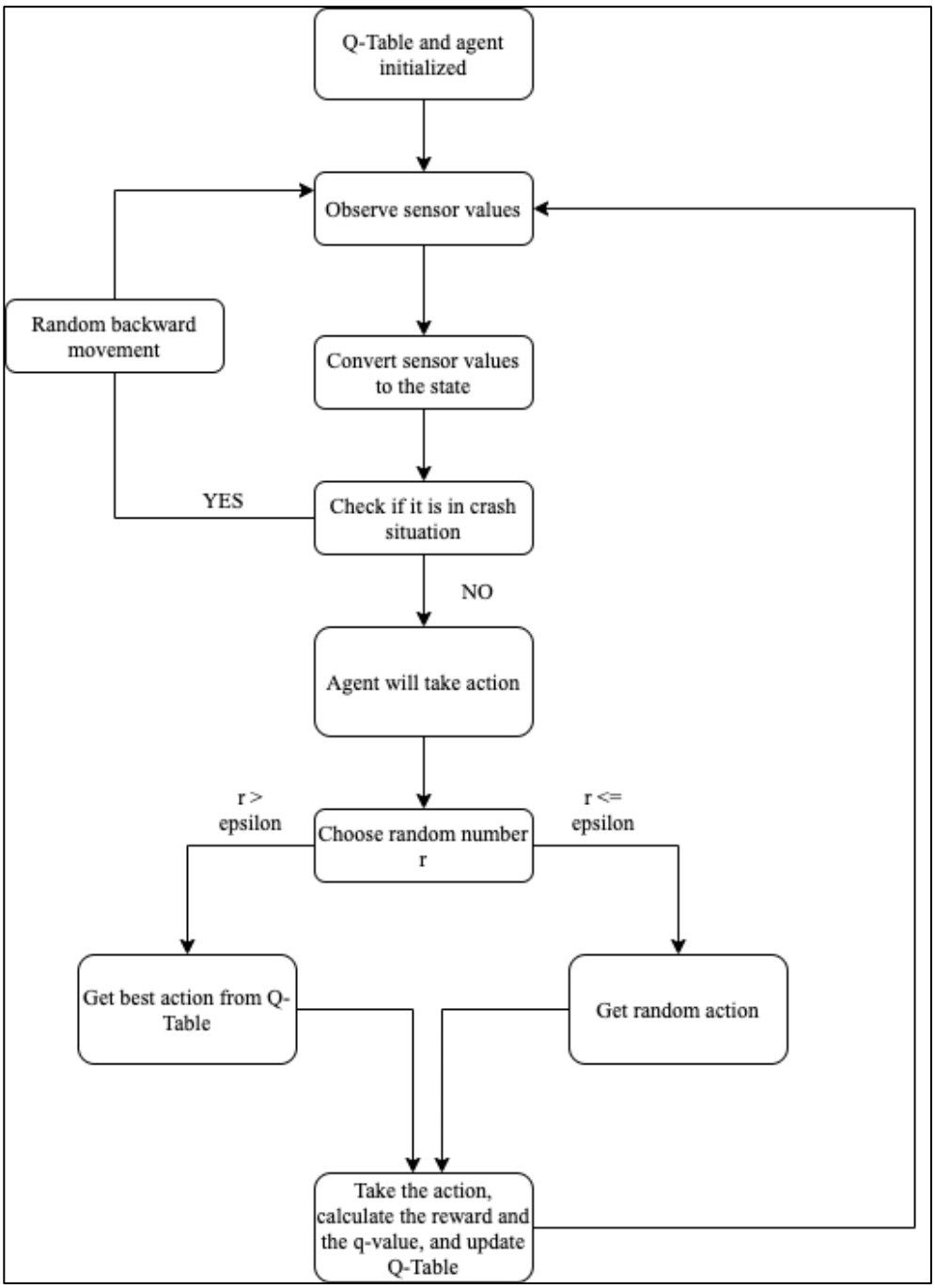

Figure 8: Workflow of Q-Learning.

3.2.2. Deep Q-Learning algorithm

Normally, training process for DQN algorithm starts with initializing replay memory capacity and, the policy network and target network with random weights between 0 and 1, but not zero. Replay memory batch size, γ, ε, and crash distance parameters are given. On the other hand, since the ultrasonic sensor data is not as sequential as image data, the replay memory method is not performed for this study. Crash distance again is chosen 12 cm for both methods. It is also decaying 0.005 for every epoch in DQN. ε, self-learning rate and discount rate is given 0.07, 0.01 and 0.3, respectively. Decaying epsilon-greedy strategy is also used.

Policy network and target network are created, actually they are the clones of each other. Both of them has 5 nodes in the input layer and 3 nodes in the output layer, since the agent has 5 states and 3 actions. The hidden layer has 15 modes. RELU and Tanh are used as activation functions. To train the networks, backpropagation algorithm and gradient descent methods are used.

The training process of DQN is similar to the Q-Learning. For every movement, the agent takes the ultrasonic sensor data and checks the crash situation. If it is in crash distance, it makes the random backward movement as mentioned earlier and the process starts over. However, different from the Q-Learning in DQN method there is not a table in which the consequences of actions are stored. Thus, to get best action agent cannot refer to the table. Instead of that, the results are kept in a neural network system.

In this algorithm, the state vector is derived from sensor output. While every node in input layer refers to the elements of state vector, every node in output layer corresponds to a Q-function. In other words, the number of neurons in the output layer is equal to the number of actions of the agent, and each neuron predicts a Q-value according to an action in the agent's action list. The purpose of this method is to find optimal Q-functions by updating the weight matrices after each experiment. Cost function is the difference between target network and policy network, and the goal is to minimize it.

Workflow of Deep Q-Learning is given Figure 9.

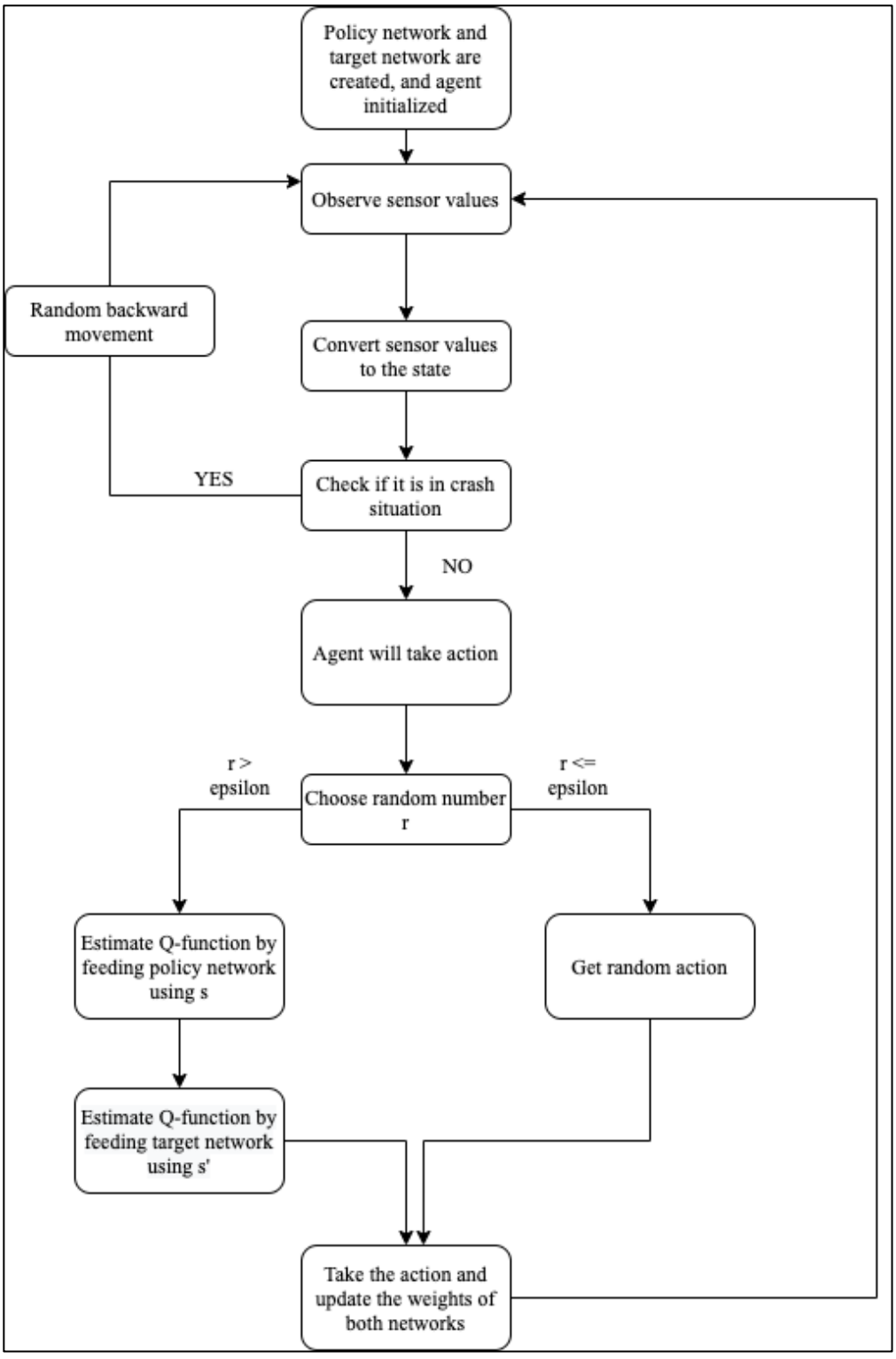

Figure 9: Workflow of Deep Q-Learning.

4. RESULTS

The agent is trained for 120 minutes with Q-Learning algorithm and 75 minutes for DQN algorithm with two different activation functions, such as RELU and Softplus. The training time is determined by the performance of the agent. The performance of algorithms is reviewed in two manners, the average duration of action without crash and the total number of crashes, in a ten-minute and five-minute period. Both metrics are calculated from crash logs.

Table 1 demonstrates the average duration of action without crash and the total number of crashes for Q-Learning, DQN with RELU function, and DQN with Softplus function.

Table 1: The average duration of action without crash and total number of crashes.

Metric	Q-Table	DQN (RELU)	DQN (Softplus)
Avr. Duration of Action Without Crash	9.95	24.22	26.81
Total Number of Crash	466	158	146

It can be seen that from Table 1, as a self-learning method DQN algorithms have a great advantage over the Q-Learning method for a complex environment. Even though, the training time of Q-Table method is longer than the others, unfortunately it does not catch the performance of them. In the end of the training, the agent which uses the Q-Learning algorithm still enters the crash position more than the ones trained by other algorithms.

The Figure 10 and the Figure 11 demonstrate the crash numbers in a ten and five-minute period for Q-Learning, DQN with RELU function and DQN with Softplus function, respectively.

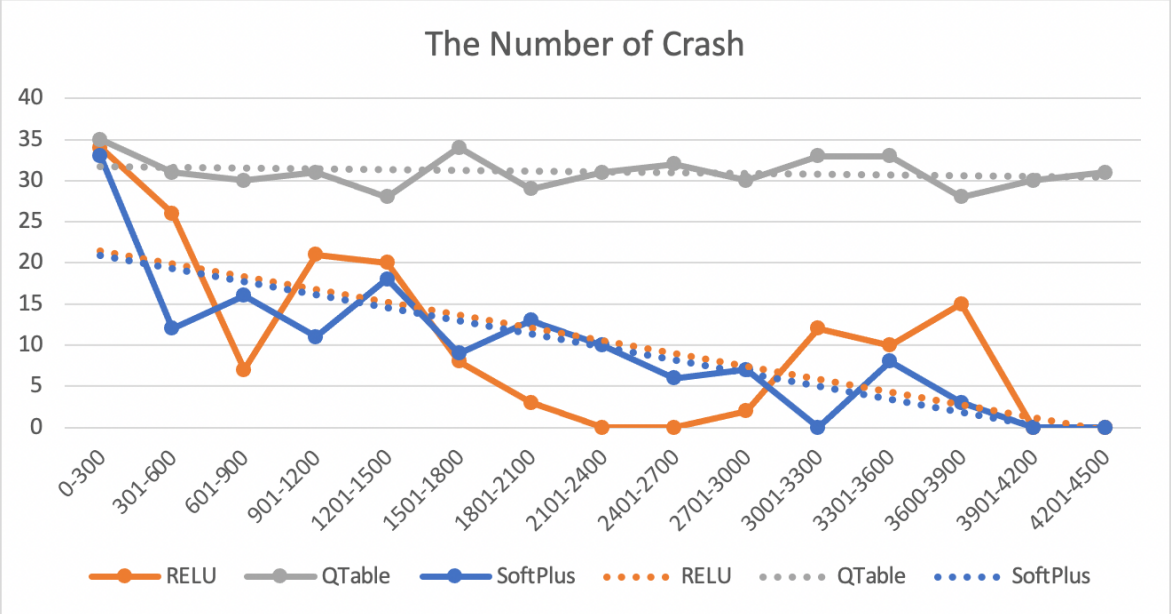

Figure 10: The crash number in a ten-minute period.

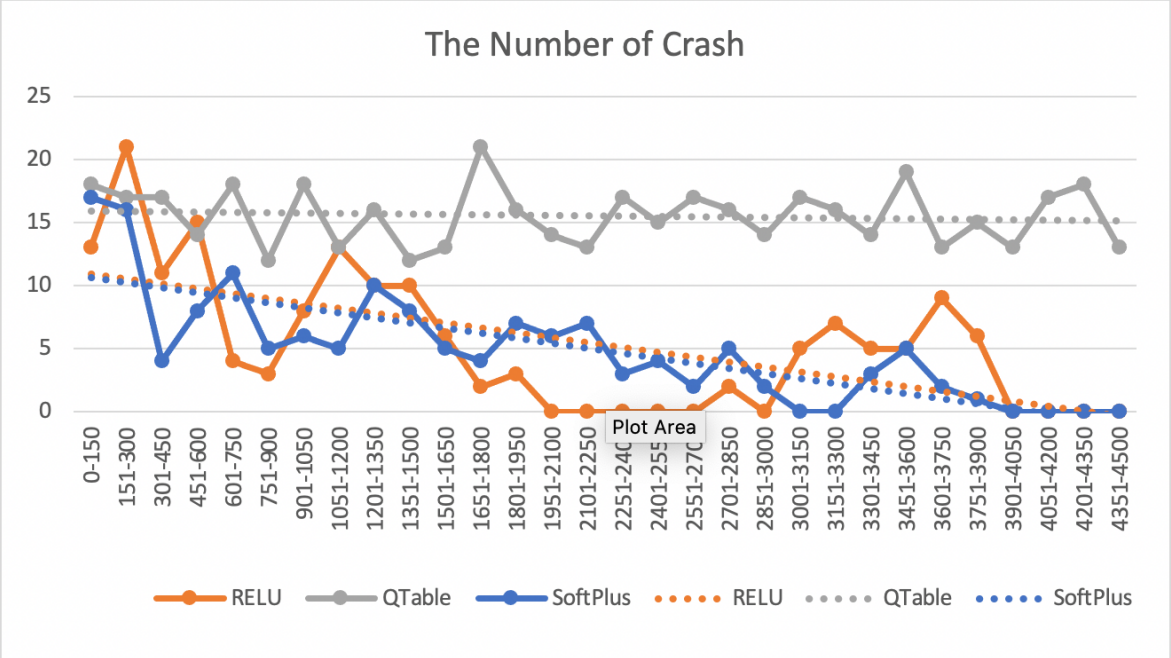

Figure 11: The crash number in a five-minute period.

As it can be seen from the graphs, the number of crashes is decreasing dramatically for both DQN algorithms, especially in first 10 minutes, with respect to Q-Learning method. It can be said that the learning curve of DQN algorithms has a much sharper slope.

In DQN with RElu function between 3000 and 4000 seconds, the mean of crash number of the agent increases to nearly 10 from 0. This situation arises from the random actions the agent chooses at the point where it approaches the periphery of the environment. Until it survives from this situation, the agent has experienced more crashes. However, in the observations made during the experiment, it is understood that when the agent chooses the best action in spite of random action, it immediately exits from the crash distance. For DQN with Softplus function a similar pattern can be seen between 3300 and 3900 seconds.

5. DISCUSSION

DQN is a recent algorithm which is developed by the DeepMind Technologies team in 2013. This method has been used mostly for image data in order to train video game agents. In this study, it is tried to adapt this algorithm in order to train an agent with ultrasonic sensor data. In this context, it can be said that this study offers an adapted version of the combination of deep learning and reinforcement learning methods.

During the training process carried out with the Q-Learning method, the average driving time without a crash is 9.95 seconds, while the same criterion is 24.22 and 26.81 seconds for the RElu and Softplus functions, respectively, in the DQN algorithm. At the end of the training, the total crash number of the agent trained with the Q-Learning method is 466, while the agent trained with the DQN algorithm had 158 for the RElu function and 146 for the Softplus function.

Observations made during the experiment also support the results seen in the table and graph in the “Result” section. Agents working with DQN algorithms learns that drawing circles in the environment is the most advantageous way in the very beginning of the training process and shows a tendency to move in circles again even after random movements. There is no noticeable difference between the results of the two activation functions, RElu and Softplus.

On the other hand, it is seen that the agent trained with the Q-Table has a tendency towards circling behavior in a longer time. Although, it draws circles without hitting the walls during training process, it is observed that the slope of the learning curve is low, and optimization of the Q-Table takes more time than the optimization of neural networks. As a result, it can be said that DQN algorithms give more efficient results in the self-learning experiment presented in this study.

6. FUTURE WORK

For future work, by differentiating the equipment that collects data on the agent, different data types such as image, temperature, oxygen value can be collected and processed. As an example, it is a good idea to train SunFounder Smart Video Car Kit V2.0 PiCar, which has wide-angle USB webcam to real-time image/video transmission. Thus, instead of sonic sensor data, image data can be taken and processed. At the same time, with changes to the reward setup in the algorithm, the agent can be trained to move to a specific target or to take actions to avoid a specific target. Moreover, the agent can be trained by Double DQN algorithm, which is a recent technique has better performance for some video games. (Hasselt et al., 2016)

In this way, it is possible to design the agent to act in a way that it can develop functional solutions to the problems encountered in daily life. At the same time, it can be contributed to the literature by training DQN algorithms, which are fed mainly with visual data, with using other data types.

APPENDIX A

The code and logs for Q-Learning and DQN algorithms can be found at the link below.

https://github.com/OzgeBegde/Deep-Q-Learning

REFERENCES

Alpaydin, E. (2004). *Introduction to machine learning*. MIT press.

Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., & Abbeel, P. (2015). Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. *arXiv preprint arXiv:1509.06113*, *25*.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*.

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, 237-285.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529-533.

Neural Networks: Structure. (2020, April 1). Retrieved from https://developers.google.com/.

Russell, S., & Norvig, P. (2010). Artificial intelligence: a modern approach.

Sallab, A. E., Abdou, M., Perot, E., & Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, *2017*(19), 70-76.

Sewak, M. (2019). *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*. Springer.

Sutton, R. S., & Barto, A. G. (2014). *Reinforcement learning: An introduction*. MIT press.

Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).

Venkatachalam, M. (2020, April 1). Q-Learning. Retrieved from https://towardsdatascience.com/.

Wiering, M., & Otterlo, M., V. (2012). *Reinforcement Learning : State of the Art*. Springer.