BENCHMARKING HINDSIGHT EXPERIENCE REPLAY REINFORCEMENT LEARNING METHODS ON VEHICLE PARKING ENVIRONMENT

DENEYİM TEKRARLI PEKİŞTİRMELİ ÖĞRENME YÖNTEMLERİNİN OTONOM PARK PROBLEMİNDE KIYASLANMASI

MEHMET ERTEKİN

PROF. DR MEHMET ÖNDER EFE
Supervisor

Submitted to
Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Engineering

2021

To my loved ones...

ABSTRACT

BENCHMARKING HINDSIGHT EXPERIENCE REPLAY REINFORCEMENT LEARNING METHODS ON VEHICLE PARKING ENVIRONMENT

Mehmet ERTEKİN

Master of Science, Department of Computer Engineering

Supervisor: Prof. Dr. Mehmet Önder EFE

December 2021, 56 pages

In the age we live in, both passenger transportation and freight transportation are of great importance. Parking the vehicles when they reach the target point is challenging for both humans and automatic parking systems. Artificial intelligence-based methods are used for this task where traditional control methods are insufficient. A common strategy for solving this kind of problem is planning a trajectory using heuristic search algorithms and following that trajectory using traditional control methods. On the other hand, reinforcement learning algorithms are developing algorithms that can be used in solving this kind of problem. HER (Hindsight Experience Replay) method is a wrapper algorithm that increases unsuccessful attempts when used with reinforcement learning algorithms. In this thesis, Twin Delayed Policy Gradient (TD3), Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC) reinforcement learning algorithms are studied. The comparison of these algorithms, which have been compared with their raw form on different problems in the literature, with the HER algorithm in the autonomous parking problem has contributed to the literature. In the designed working environment, an artificial intelligence control system was designed with HER supported reinforcement learning methods on a vehicle model whose throttle and steering commands are constantly controlled in space. The designed control system controls the vehicle and enables it to park at the target point. It has been shown by the studies that the studied reinforcement learning methods can solve the autonomous parking problem, and the algorithm performances are compared. Experiments have shown that the TD3 algorithm, which was launched as an improved version of the DDPG algorithm, could not perform better than the DDPG algorithm when used in the autonomous parking problem with HER. The most successful of the algorithms used in this study was the SAC algorithm.

Keywords: reinforcement learning, HER, DDPG, TD3, SAC, dynamic control, actor-critic

ÖZET

DENEYİM TEKRARLI PEKİŞTIRMELİ ÖĞRENME YÖNTEMLERİNİN OTONOM PARK PROBLEMİNDE KIYASLANMASI

Mehmet ERTEKİN

Yüksek Lisans, Bilgisayar Mühendisliği

Tez Danışmanı: Prof. Dr. Mehmet Önder EFE

Aralık 2021, 56 sayfa

İçinde bulunduğumuz çağda gerek yolcu taşımacılığı gerekse yük taşımacılığı büyük önem taşımaktadır. Araçların hedef noktaya ulaştıklarında park etmeleri hem insanlar için hem otomatik park sistemleri için oldukça zorlayıcı bir görevdir. Geleneksel kontrol yöntemlerinin yetersiz kaldığı bu görev için yapay zekâ tabanlı yöntemlerden yardım alınmaktadır. Bu tarz problemlerin çözümü için yaygın olarak kullanılan yöntem sezgisel arama algoritmaları ile yol planlayıp geleneksel kontrol yöntemleri ile planlanan yolu takip etmektir. Diğer taraftan pekiştirmeli öğrenme algoritmaları gelişmekte olan ve bu tarz problemlerin çözümünde kullanılabilecek yapay zekâ algoritmalarıdır. HER (İng. Hindsight Experience Replay) yöntemi ise pekiştirmeli öğrenme algoritmaları ile kullanıldığında başarısız denemelerindeki performansı arttıran sarmal (İng. wrapper) algoritmadır. Bu tezde TD3 (İng. Twin Delayed Policy Gradient), DDPG (İng. Deep Deterministic Policy Gradient), SAC (İng. Soft Actor Critic) pekiştirmeli öğrenme algoritmaları çalışılmıştır. Literatürde ham halleri ile farklı problemler üzerinde kıyaslaması yapılmış bu algoritmaların HER algoritması ile otonom park probleminde kıyaslanması literatüre katkı sağlamıştır. Tasarlanan çalışma ortamında sürekli uzayda

(İng. continuous space) gaz ve direksiyon komutları kontrol edilen bir araç modeli üzerinde HER destekli pekiştirmeli öğrenme yöntemleri ile yapay zekâ kontrol sistemi tasarlanmıştır. Tasarlanan kontrol sistemi aracı kontrol ederek hedef noktaya park etmesini sağlamaktadır. Yapılan çalışmalarla kullanılan pekiştirmeli öğrenme yöntemlerinin otonom park problemini çözebileceği gösterilmiş ve algoritma performansları kıyaslanmıştır. Yapılan deneyler göstermiştir ki, DDPG algoritmasının geliştirilmiş versiyonu olarak lanse edilen TD3 algoritması HER ile otonom park probleminde kullanıldığında DDPG algoritmasından iyi bir performans sergileyememiştir. Bu çalışmada kullanılan algoritmaların en başarılısı SAC algoritması çıkmıştır.

Anahtar Kelimeler: pekiştirmeli öğrenme, HER, DDPG, TD3, SAC, dinamik kontrol, aktör-kritik

ACKNOWLEDGMENTS

I would like to thank Prof. Dr. Mehmet Önder EFE, who has always guided me with a valuable contribution.

Mehmet ERTEKİN

TABLE OF CONTENTS

FIGURES

TABLES

SYMBOLS AND ABBREVIATIONS

Symbols

η	Learning rate
σ	Activation function
$\odot$	Element wise product, Hadamard product
$\doteq$	Defined as
γ	Discount rate
ϕ	Actor network parameters
θ	Critic network parameters

Abbreviations

SAC	Soft Actor Critic
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
PID	Proportional Integral Derivative
LQ	Linear Quadratic
MDP	Markov Decision Process
DQN	Deep Q-Networks
RRT	Rapidly-exploring Random Tree

1. INTRODUCTION

1.1. Overview

Transportation vehicles are one of the essential parts of our daily lives. They make our lives better but at some cost. In the 2019 year of Turkey, 418,488 traffic accidents happened, and 2,524 people died in these accidents. The root cause of these accidents is driver fault by 87% percentage[1]. Autonomous vehicles could improve the safety and comfort of transportation. Using safe autonomous vehicles could help us save thousands of lives in traffic accidents every year. Moreover, parking a vehicle is a complicated and stressful task for both humans and classical control systems. Safe autonomous parking vehicles will save people's time and effort.

Researchers have been studying driving assistance systems and autonomous self-driving systems for decades. Defense Advanced Research Projects Agency announced several autonomous vehicle development competitions in 2003. These competitions led a couple of successful autonomous vehicle technologies to appear. For example, Stanford's Junior team in 2007 DARPA urban challenge came up with a hybrid A* algorithm [2]. It was a significant improvement for vehicle navigation and control technology.

On the other hand, individual research groups and companies like Tesla make progress upon neural network-based approaches. DeepPicar is an excellent example of CNN-based autonomous car controller design. They feed deep CNNs with images for controlling vehicle[3].

1.2. Motivation

Reinforcement learning is an essential concept of today's world. However, some control problems are not solvable with traditional methods. Especially ones that require strategy and planning like parking a vehicle or playing a chess game.

Benchmarks show that different learning problems have different best reinforcement solutions. No algorithm shows the best performance in all the control problems [4]. In this work, we wondered the current best performing method on parking car controller problem.

1.3. Aim of The Thesis

The development of machine learning systems let various reinforcement learning algorithms and methods to be developed. It is seen that almost every dynamic control problem has a different best performance reinforcement learning solution. In the study performed within the scope of this thesis, a reinforcement learning based autonomous control system is designed, and a performance comparison of SAC, DDPG, and TD3 methods based on the HER algorithm is made.

In this direction, the aims and objectives of the thesis study are as follows.

- Giving general information about machine learning methods categorization and their improvement over the years.
- Giving general information about reinforcement learning and dynamic control systems.
- Designing a reinforcement learning system as a dynamic control system.
- Giving detailed information about TD3, DDPG, SAC, HER algorithms.
- Describing parking simulation environment model.
- Observing the relative performance of TD3, DDP, SAC algorithms with each other on the parking task.
- Comparing parking task performances of TD3, DDPG, and SAC algorithms wrapped with HER.
- Supporting autonomous control systems development by making decision-making steps of researchers easier.

1.4. Thesis Structure

This study consists of five chapters. The first chapter is the introduction, which aims to introduce the research field to the reader. The chapter includes a brief overview, motivation of the study, the aim of this thesis, and thesis structure.

The second chapter is the background. This chapter explains the theoretical background of machine learning, reinforcement learning, and control theory. This background information is fundamental for understanding state of the art actor-critic algorithms, which will be given in the next chapter.

The third chapter is the used reinforcement algorithms. In this chapter, DDPG, TD3, SAC, and HER algorithms are inspected.

The environment used in this study is given in chapter four. Core equations that are needed to calculate the next state and reward are given in this chapter.

The fifth chapter is the related works chapter. In this chapter related literature works are studied.

The sixth chapter is the simulation results chapter. The chapter includes details and configurations of the experimental environment.

Chapter six is the work done chapter. This chapter consists of experiment setup configuration and experiment flow.

The last chapter is the conclusion and future work. Experiment results are discussed, and the conclusion of this work is given in this chapter. This chapter also includes possible future studies in this research area.

2. BACKGROUND

Autonomous driving systems could be simplified into two parts: perceptron and planning. Perceptron part is about taking data from sensors and figuring out the environment's state and vehicle state like line detection and localization [5]. Planning, on the other hand, is about deciding maneuvers making plans, and controlling the vehicle. So instead of perceptron, this thesis work is about planning.

2.1. Machine Learning

When the exact solution to a problem is known, an algorithm is written that takes an input, processes it with a precisely defined way, and outputs the results. However, some tasks have no exact solution, so that it is not possible to write a precise algorithm. For example, a spam detection system has input as email and output as if it is spam mail or not. We do not know the process from input to output. Machine learning methods come to the rescue here. Machine learning is developing systems that try to find the best possible method that converts inputs to output using example data or past experiences. Machine learning methods highly rely on the theory of statistics and mathematical model.[6]

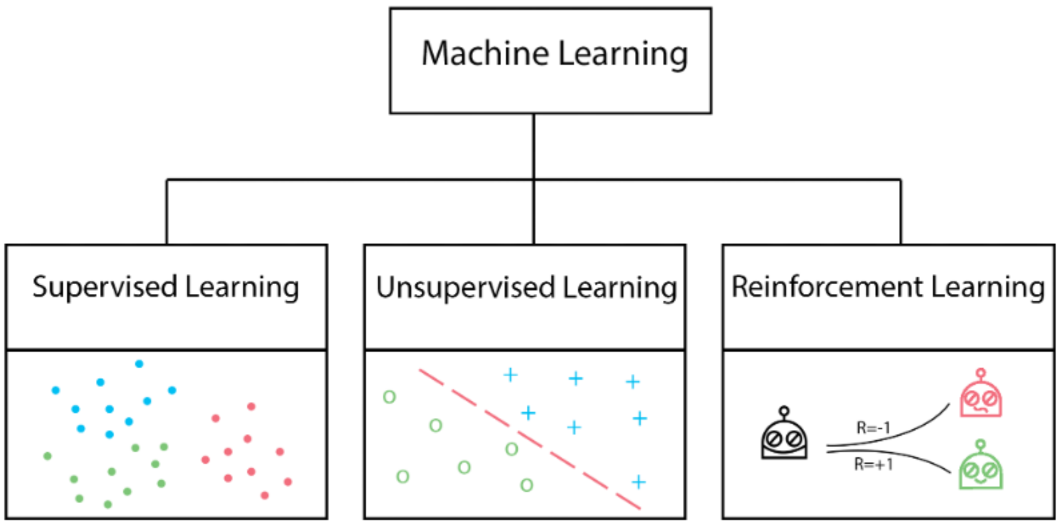


Figure 1. Classification of machine learning systems

There are three main categories of machine learning systems: reinforcement learning, unsupervised learning, supervised learning as shown in Figure 1 [7]. Supervised learning systems take training data that consists of some input and output matches. Upon that training set, supervised learning systems build mathematical models to predict outputs of future inputs. Supervised learning systems are a good fit for regression and classification problems. Unsupervised learning systems on the other hand, take training data consisting of only inputs and try to find structures, patterns in data to detect commonalities are absent or not with future inputs. Finally, reinforcement learning algorithms make decisions that they made and improve their models upon that reward. This thesis work is about reinforcement learning systems.

2.2. Artificial Neural Networks

Artificial neural networks are one of the most used machine learning system design methods today. ANNs simulate biological brains. The human biological brain has cells named neurons. These neurons are connected via axons, dendrites, and synapse mechanisms, as illustrated in Figure 2.[8]

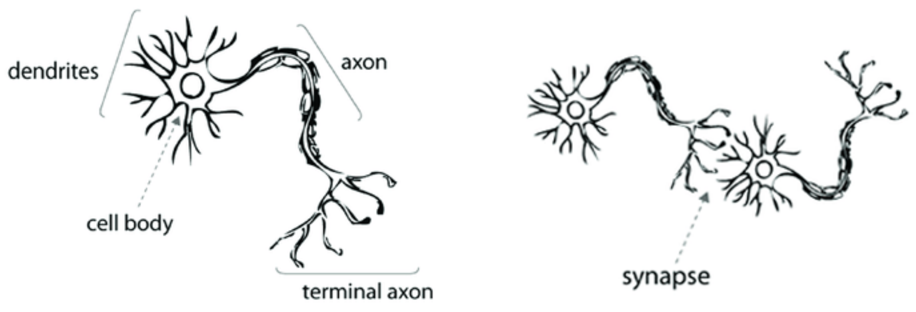


Figure 2. Biological neurons

Artificial neural networks on the other hand, have computation units named neurons. Artificial neurons take inputs scaled with some weights, process them via activation function, and send results to output as shown in Figure 3. Learning tasks are accomplished by optimizing weights for the best output required for the given inputs.

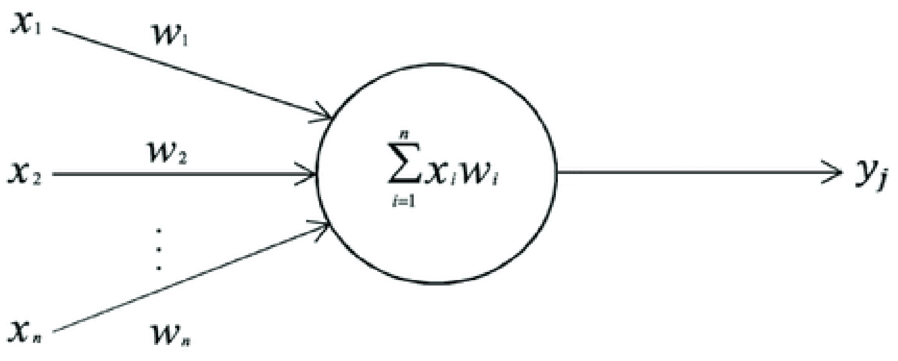


Figure 3. Artificial neuron

Artificial neurons come together and create layers. Layers come together and create an ANN. Three common types of neural network layers are the input layer, hidden layer, and output layer. The input layer is the layer that is the first stop of neural network data. It can be environment observations, pixels, etc., anything that is the input of a machine learning system. Hidden layers are layers that are between the input layers and output layer. A neural network could have many hidden layers or none. The output layer is the last layer before the output of the ANN, the output layer shows the current result of the learning system. As sampled, three-three-three-two feed-forward neural network architecture is given in Figure 4.

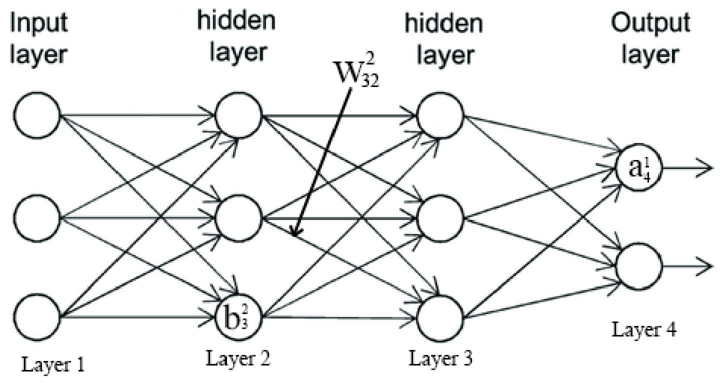


Figure 4. Neural network architecture

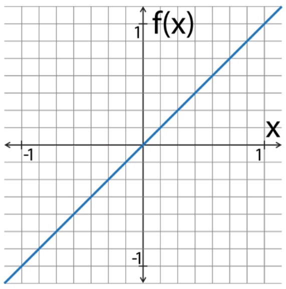
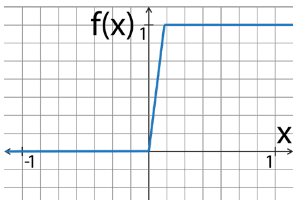
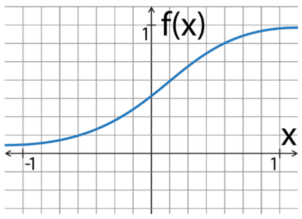
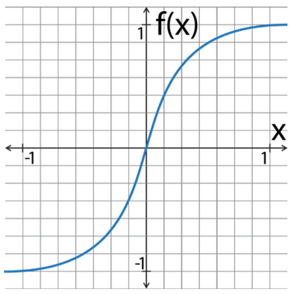
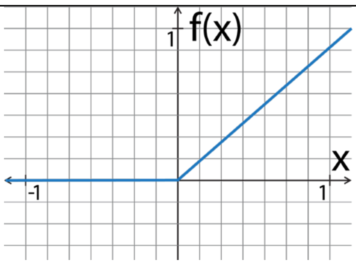
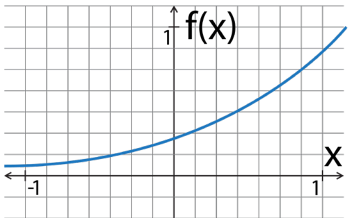
Neurons take the sum of weighted inputs w^l_{jk} where l is next layer j is next neuron k is current neuron count and passes into activation function a result of activation function determines the output of a neuron. ANNs could have biases that work as extra neurons without activation function just a constant. Equation (1) shows the output of a neuron.

$$a^l_j = \sigma\left(\sum_k w^l_{jk}\, a^{l-1}_k + b^l_j\right) \tag{1}$$

There exist different types of activation functions. Table 1 shows a couple of different activation functions used in ANNs. Feeding ANN with information into the input layer and then propagating through the output layer to get ANN output is called forward propagation. After doing forward propagation, the cost function gives the quality of a result. Sometimes referred as loss or objective function [9]. For inputs of x and target of $y(x)$ quadratic cost function is given in equation (2).

$$C = \frac{1}{2n}\sum_{x} \|y(x) - a^{L}(x)\|^{2} \tag{2}$$

Table 1. Some activation functions used in neural networks

Name	**Plot**	**Function**
Identity		x
Binary Step, Threshold		$\begin{cases} 0 & if\ x < 0 \\ 1 & if\ x \geq 0 \end{cases}$
Sigmoid		$\sigma(x) = \dfrac{1}{1 + e^{-x}}$
Tanh		$\tanh(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
ReLU		$\begin{cases} 0 & if\ x \leq 0 \\ x & if\ x > 0 \end{cases}$
Softplus		$\ln(1 + e^{x})$

There exist different types of learning algorithms. One of the most famous and fundamental one is the backpropagation algorithm. David E. et al. showed the power of the backpropagation algorithm in 1986 [10]. Since then, many improvements have been made and many more on the way.

Backpropagation algorithm tries to minimize cost function by changing weights and biases in ANN using partial derivatives $\partial C/\partial w_{jk}^{l}$ and $\partial C/\partial b_{j}^{l}$. Error in the j^{th} neuron of i^{th} layer is given in equation (3) where z_j^l is the output of j^{th} neuron in l^{th} layer.

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l} \tag{3}$$

The error of the output layer is given in equation (4). Where the first term measures the change rate of the cost function, the second term on the rights measures the change rate of the activation function. Because forward propagation gives z^L equation (4) could be easily computed. Equation (4) is component wise representation; it is written as matrix form in equation (5) where $\odot$ donates element wise product sometimes called as Hadamard product or Schur product [9]. $\nabla_a C$ on the other hand, means the change of C with respect to a, which means $(a^L - y)$ thus gives us a fully matrix-based representation equation (6).

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \tag{4}$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{5}$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L) \tag{6}$$

The calculation formula of errors for hidden layers is given in equation (7). Where the first part of element wise product is the multiplication of weight matrix transform by error. The second part is the derivative of the transform function. Equations (6) and (7) are enough to compute the error in any layer in ANN.

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{7}$$

The rate of change in cost equations for bias and weights are given in equations (8) and (9). These equations show how to take partial derivatives. When the derivative of the activation function is small ($\sigma'(z) \approx 0$) ANN will learn slowly. Whenever this situation occurs, it is sad that neurons saturated, or weights stopped learning.

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \ or \ simply \ \frac{\partial C}{\partial b} = \delta \tag{8}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \ or \ simply \ \frac{\partial C}{\partial w} = a_{in} \delta_{out} \tag{9}$$

Backpropagation equations make it possible to compute the gradient of cost function efficiently. It computes errors starting from the last layer. After calculating errors, it is common to apply gradient-based learning algorithms like gradient descent or stochastic gradient descent for completing a learning task. Following pseudo code applies gradient descent with backpropagation given m training examples.

Algorithm 1: Backpropagation

```
Step 1   Initialize weights
Step 2   For each training data x, do the following steps:
Step 3       Feed-forward
Step 4       Compute output error with equation (6)
Step 5       Back propagate error with equation (7)
Step 6   Update weights with gradient descent which means for weights
         apply rule $w^l \rightarrow w^l - \frac{\eta}{m}\sum_x \delta^{x,l}(a^{x,l-1})^T$ and for biases apply
         $b^l \rightarrow b^l - \frac{\eta}{m}\sum_x \delta^{x,l}$
```

2.3. Classic Control Methods

Wolfram Math World dictionary defines control theory as “the mathematical study of how to manipulate the parameters affecting the behavior of a system to produce the desired or optimal outcome”[11]. Design and analysis of automatic control systems are a fundamental part of today's engineering and science fields. Automatic control system applications include space-vehicle systems, temperature control, path follower systems, pressure control systems, etc.

Control systems have two significant variables; controlled variable and manipulated variable or control signal. Control systems create control signals or manipulated variables to control target properties of the environment which is called a controlled variable. The control environment in which the control system is built is called plant. A plant could be an air conditioner, autonomous vehicle, cleaning robot, etc. A combination of different modules comes together and creates a system. Systems could be in biological, physical, electronic, software environments. Sometimes defects could occur on signals between components of a system. This situation is called disturbance or noise. Open-loop and closed-loop control systems are the two main categories of control systems.

Control systems that take plant output and a reference signal to generate control signals are called feedback control systems or closed-loop control systems. As shown in Figure 5 closed-loop control systems take the difference between input and feedback signal difference defined as actuating error and feed controller with that actuating error.

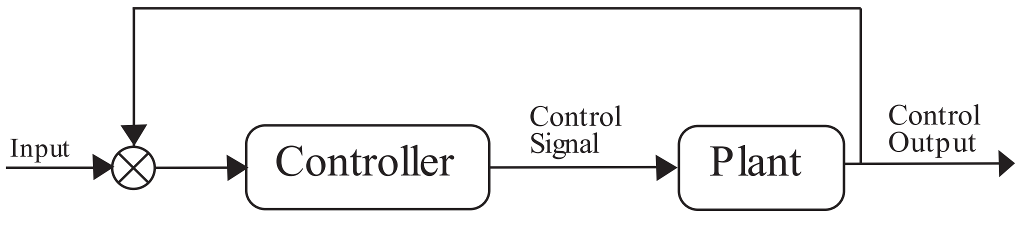


Figure 5. Closed-Loop Control System Architecture

On the other hand, the output of the plant has no effect on controller action in open-loop control systems, as shown in Figure 6. The performance of open-loop control systems highly depends on controller configuration.

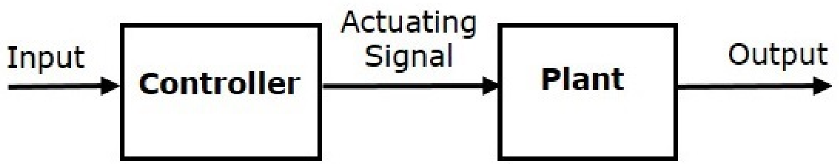


Figure 6. Open-Loop Control System Architecture

Traditional controllers like LQ controllers and PID controllers are not enough when the control problem includes strategic planning. The autonomous vehicle control task is one of the control tasks that need strategic planning. To solve this kind of strategic control problem, researchers have used a couple of different path planning algorithms and methods, mostly together with the path-following controllers. Recent developments on machine learning systems, especially in neural networks and reinforcement learning lead researchers and control system designers to use ANN-based methods.

2.4. Reinforcement Learning

Learning systems that take rewards after their actions and shape following action decisions according to rewards is called reinforcement learning. Reinforcement learning systems that are affected by reward immediately on the following action are called trial and error search otherwise, it is called delayed reward [12]. Learning modules of reinforcement learning systems are called agents. Agents work like controllers in classic control systems. Moreover, plants in the classic control system are called environments in reinforcement learning literature. Learning agents sense the environment and take actions that affect the environment. Learning agents also have some target or goal to complete. Markov decision process builds up with these three-stage sensing the environment, acting, having a target.

Reinforcement learning differs from both supervised learning and unsupervised learning. Supervised learning systems learn from prelabeled data where each of the data includes detailed situation information and decision to be made in that situation. After the learning phases, supervised learning systems try to generalize training situations so that it can label data that are not present in the training set. Supervised learning is an essential kind of machine learning method, but it is not suitable for interactive learning problems. Unsupervised learning methods, on the other hand, try to find some hidden groups in given unlabeled data. Reinforcement learning systems try to maximize reward instead of finding hidden structures.

Reinforcement learning accommodates some challenges. The trade between exploitation and exploration is one of the fundamental challenges of reinforcement learning systems. Reinforcement agents must choose between exploiting an action that has been experienced before which has a relatively high reward and exploring an action that has not been explored yet, or it is known that it has relatively low reward but has potential for better action possibilities in the future.

Reinforcement learning has highly related application fields in different disciplines of science and engineering. Integration of machine learning and artificial intelligence methods with fields like optimization theory, control theory, and statistics is a trend of the last couple of years which led researchers to create and improve reinforcement learning methods. One of the good examples of reinforcement learning application is solving a course of dimensionality problem in control theory with parameterized approximators. Moreover, reinforcement learning has a strong relation with neuroscience and psychology. Although most of the reinforcement learning algorithms are inspired by biological learning systems researchers in neuroscience, have also been taking advantage of reinforcement learning systems in their research [12].

Agent and environment are the two main elements of reinforcement learning. Besides these main elements, reinforcement learning systems also have four sub-elements: value function, policy, environment model, and reward signal.

The policy function of reinforcement learning takes states as input and outputs an action idea to take for agents. Policies could be simple as lookup tables and could be complex as using ANNs.

On the other hand, reward signals or reward functions define how close the reinforcement learning system is to the goal of the problem. Environment sends reward signal to learning agents, and agents try to maximize the total reward obtained. Reward signals show the immediate result of the last action.

Thirdly value functions show the quality of actions like reward signals, but unlike reward signals, value functions show long-run qualities of actions. Value functions try to estimate the total amount of rewards that an agent could possibly get if that specific action is taken.

Some reinforcement learning systems have a model of the environment, which is a simulation of the environment. Environment models give insights to agents about how the environment will behave in given state and action pairs. Models help agents to create plans before making action decisions. Reinforcement learning methods that use environment modes and plans are called model-based methods. Model-based reinforcement algorithms have access to or learn the environment model. The main advantage of model-based algorithms is that agents can make plans of time. The main downside of the model is that usually, a good environment model is not available. Methods that do not use environments model are called model-free methods or trial and error learners.

2.5. Markov Decision Process

MDPs formalize classical decision-making steps where rewards affect not only immediate actions but also future actions for better rewards in the long term. MDPs made it possible to make precise theoretical statements because MDPs are the mathematically idealized representation of reinforcement learning problems. MDPs have two main components; agents and environments which are continuously interacting with themselves. Agents make decisions and are responsible for the learning process. Everything other than the agent in the system is the environment. Agents send actions to the environment, and the environment returns state and reward to the agent as represented in Figure 7.

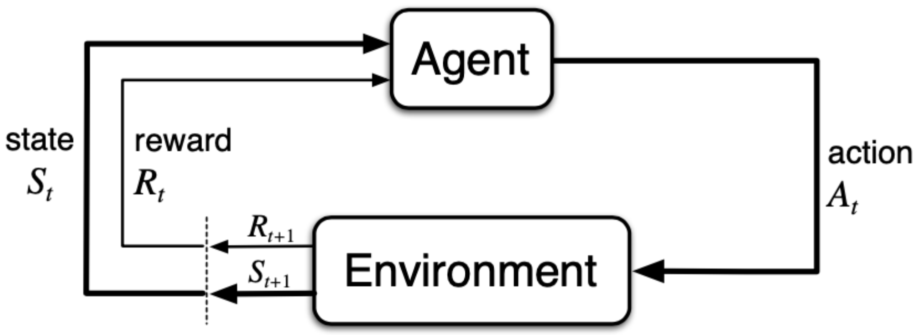


Figure 7. Markov decision process interactions

If the state, action, and rewards are in finite sets, it is called finite MDP. Finite MDPs have discrete S_t and R_t probability distributions depend upon immediately preceding state S_{t-1} and action A_{t-1}. The dynamics of MDPs are shown in equation (10).

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{10}$$

$$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1, \qquad \forall \ s \in S, a \in A(s) \tag{11}$$

States in the Markov decision process should include all past relative agent interaction information. If it does, it is called Markov property. From the four argument p function given in equation (10) three argument state transition probabilities (12), two arguments expected rewards for state–action probabilities (13), three arguments expected rewards for state action next state (14) can be computed [12].

$$p(s' \mid s, a) \doteq Pr\{S_t = s' \mid S_{t-1} = s,\ A_{t-1} = a\} \tag{12}$$

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = \text{s}, \quad A_{t-1} = a] = \sum_{r \epsilon R} r \sum_{s' \epsilon S} p(s', r \mid s, a) \tag{13}$$

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = \text{s}, \quad A_{t-1} = a, \quad S_t = \text{s}'] = \sum_{r \epsilon R} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)} \tag{14}$$

As stated, the reinforcement learning agent tries to maximize total rewards. Generally, there exists an expected reward function which is denoted as $G_t$which is the sum of expected rewards R. Sometimes, reinforcement learning problems logically break into parts. These parts are called episodes, and these types of reinforcement learning tasks are called episodic tasks. Episodes have one or more starting states and one or more ending states. After the reinforcement learning agent gets into a terminal state next episode begins with one of the starting states. Some other reinforcement learning tasks do not logically break into parts. These types of tasks are called continuing reinforcement learning tasks.

One of the popular approaches to calculating expected reward G_t is to multiply rewards by some exponential constant γ which is called the discount rate. The discount rate determines the weights of future renature rewards between 0 and 1.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{15}$$

When the discount rate gets closer to zero, immediate rewards become more important. On the opposite side, when the discount rate gets closer to one, future rewards become more important.

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{16}$$

Most of the reinforcement learning algorithms have functions called value function v. Value functions determine the quality of the state that the agent is currently in. Future rewards of reinforcement learning agents depend on agents' decisions. Agent decisions are determined by the function called policy function. Policy functions can be deterministic or stochastic. Deterministic policies are denoted by μ. Stochastic policies are denoted by π. Value of a state s under policy π is given in equation (17) where $\mathbb{E}_\pi$ is the expected value of the random variable when agents follow policy π at time t.

$$v_\pi(s) \doteq \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|S_t = s\right), \qquad \forall\, s \in \mathcal{S} \tag{17}$$

Reinforcement learning algorithms usually need another function that will determine the quality of an action a given state s and policy π. This function is called the action-value function for policy π and is represented in equation (18).

$$q_\pi(s, a) \doteq \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|S_t = s, A_t = a\right) \tag{18}$$

The state-action pair sequence that the machine learning algorithm produces for a given scenario is called a trajectory. Trajectories are denoted by τ.

$$\tau = (s_0, a_0, s_1, a_1, \dots) \tag{19}$$

There exist one or more optimal ways to solve reinforcement learning tasks. The optimal policy function is represented as π_* optimal state value function is represented as v_* and optimal action-value function is denoted as q_*.

$$v_*(s) \doteq \max v_\pi(s) \tag{20}$$

$$q_*(s, a) \doteq \max q_\pi(s, a) \tag{21}$$

Value functions in reinforcement learning satisfy a self-consistent recursive relationship. This recursive relationship shown in equations (22), (23), and (24) is called Bellman Equations founded by American mathematician Richard Bellman. Bellman equations are a mathematical representation of the idea that the value of a state is the expected reward for being there plus expected rewards for future actions from that state.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi(G_t|S_t = s) \\
&= \mathbb{E}_\pi(R_{t+1} + \gamma G_{t+1}|S_t = s) \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma \mathbb{E}_\pi(G_{t+1}|S_{t+1} = s')] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]
\end{aligned} \tag{22}
$$

$$
\begin{aligned}
v_*(s) &= \max q_{\pi*}(s, a) \\
&= \max_a \mathbb{E}_{\pi*}(G_t|S_t = s,\ A_t = a) \\
&= \max_a \mathbb{E}_{\pi*}(R_{t+1} + \gamma G_{t+1}|S_t = s,\ A_t = a) \\
&= \max_a \mathbb{E}_{\pi*}(R_{t+1} + \gamma v_*(S_{t+1})|S_t = s,\ A_t = a) \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]
\end{aligned} \tag{23}
$$

$$
\begin{aligned}
q_*(s, a) &= \mathbb{E}\left(R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \middle| S_t = s,\ A_t = a\right) \\
&= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')]
\end{aligned} \tag{24}
$$

2.6. Deep Q Networks

Chris Watkins introduced Q-learning in 1989 [13], which was later proved by Watkins and Dayan in 1992 [14]. The idea behind Q learning is that if we store what to do in every state-action pair in a table called Q table and update that table according to Bellman Equation and temporal difference learning.[15]We would have a model-free reinforcement learning algorithm.

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \bigg)}^{\text{temporal difference}} \tag{25}$$

Q learning stores expected reward at given state and action pairs which mean Q values. Q values are arbitrarily initialized by the programmer. At every time step Q table is updated by the rule given in equation (25). The α parameter in this expression is the step size or the learning rate, which determines the importance of recently acquired experience. When α gets closer to zero, algorithms learn slowly, stick with old experiences, or when α goes up to 1, the algorithm does not care about old information, and it tries to maximize the compatibility of new information. Similarly, the discount factor γ determines the importance of expected future rewards. If the discount factor γ gets closer to 0, the agent starts to consider only current rewards, and this will make the agent "myopic". Else if the discount factor γ goes up to 1, the agent starts to optimize long term expected reward.

Algorithm 2: Q-Learning

```
Step 1  Initialize Q table for all action-state pairs
Step 2  For each episode, do the following steps:
Step 3      do
Step 4          Choose action A using policy derived from Q
Step 5          Apply action A, observe reward S, and next state S'
Step 6          Update Q table according to equation (25)
Step 7          Assign S' to S
Step 8      until the end of the episode
Step 9  end for
```

One of the prime problems with Q learning was the curse of dimensionality problem. When there exists high dimension, the state-space of Q learning becomes unable to scale. The computing power required increases exponentially. Main et al. find a method to solve the curse of dimensionality problem in Q-learning. The idea behind their algorithm was instead of Q tables to store expected values. They use ANN to guess future rewards. They took raw pixel outputs of Atari games and successfully applied their algorithm to couple of games in 2013. This method is called DQN [16]. They solved the curse of dimensionality problem by applying convolutional neural networks at the first layers of ANN structure. They also demonstrated experience replay buffer mechanism.

Figure 8. Screenshots of some Atari games that the DQN algorithm succeeded

DQN imitates human behavior by allowing an agent to explore the environment and gather information. The agent collects data and puts it into a replay buffer in the training process. DQN architecture concept is illustrated in Figure 9. [17]

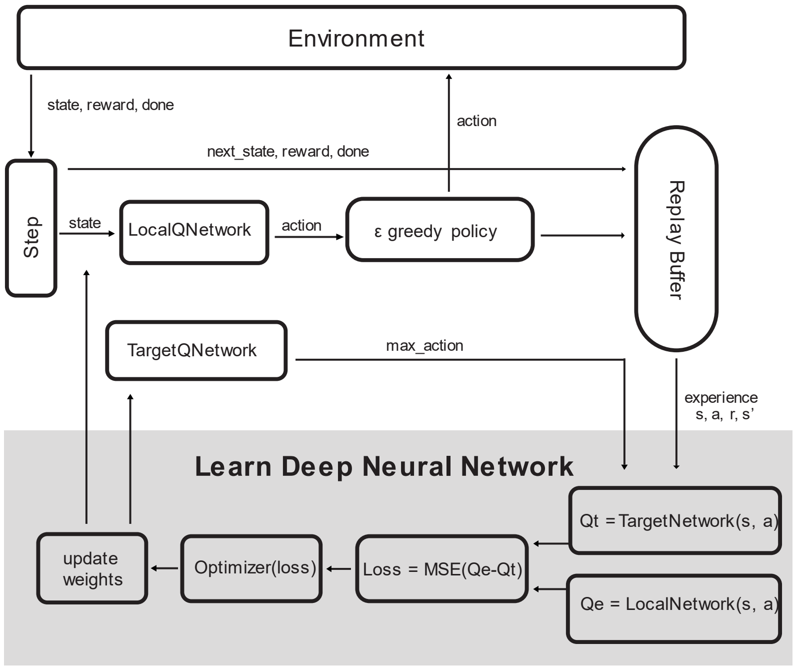


Figure 9. DQN algorithm architecture

During the exploration of the environment, the agent collects data to be used in improving Q-network. The agent acts randomly at the beginning of the training process to build a complete general picture of Q value relations. Randomly acting agents get insufficient overtime. To solve this inefficiency agent looks at Q-network to decide how to act. This approach is called the epsilon-greedy method, which just means changing the decision mechanism of the agent between random and Q policy using a probability donated by ϵ (epsilon).

$$Loss = (r + \gamma \max_{a'} Q(s', a';\ \theta) - Q(s, a;\ \theta))^2 \tag{26}$$

DQN algorithm has inspirations of supervised learning in its core. The goal of DQN is to approximate complex, nonlinear Q function with an ANN. The loss function is the squared difference between target and prediction. The loss function value is minimized by updating the weights of ANN. Loss function can be defined in DQN just like supervised learning methods, as shown in equation (26).

DQN uses two different Q networks: local and target Q networks. These two different Q networks are used to find the prediction value (θ) and the target value (θ'). During the training process, periodically target network weights are updated by copying weights of the actual Q network. Pausing the target network for a while and then updating target networks weights with actual Q network weights makes the learning process more stable. Applying a replay buffer mechanism on top of this makes even more stable learning process.

In conclusion, agent's experiences $e_t = (s_t,\ a_t,\ r_t,\ s_{t+1}\ )$ are stored in a dataset $D = e_1, e_2, \ldots, e_N$, collected over episodes into experience replay buffer. Experience samples $e \sim D$ are collected from the experience replay buffer, and mini-batch updates are applied. After mini-batch updates by the experience replay buffer, the agent selects action with ϵ greedy policy. Agent histories produced with ϕ function from experience replay buffer. These histories are going to be used as ANN input. The complete algorithm is presented below as a pseudo code.

Algorithm 3: DQN with Experience Replay Buffer

```
1   Create experience replay buffer
2   Create action-value network Q with random weights
3   for every episode, do
4       Create sequence$s_1 = \{x_1\}$ and $\phi_1 = \phi(s_1)$
5       do
6           if $\epsilon$ probability is satisfied
7               select random action $a_t$
8           else
9               select max valuable action $a_t = \max_a Q^*(\phi(s_t), a;\ \theta)$
10          Execute action $a_t$ , observe reward $r_t$ and image $x_{t+1}$
11          Assign $s_{t+1} = s_t, a_t, r_t, x_{t+1}$ and calculate $\phi_{t+1} = \phi(s_{t+1})$
12          Save transition to replay buffer as <S, A, R, S'>  $(\phi_t, a_t, r_t, \phi_{t+1})$
13          Generate random minibatch of  $(\phi_t, a_t, r_t, \phi_{t+1})$ from replay buffer
14          Set $y_j = \begin{cases} r_j, \ for\ terminal\ \phi_{j+1} \\ r_j + \gamma \max_{a'} Q\ (\phi_{j+1}, a';\ \theta),\ for\ non-terminal\ \phi_{j+1} \end{cases}$
15          Perform gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$
16      until the end of the episode
17  end for
```

2.6. Reinforce Algorithm

As previously discussed, Q learning or deep Q learning algorithm tries to find the optimal policy, which is the selection of best Q valued action in every state, which could be formulated as equation (27) . These Q value-based methods estimate the optimal value function before working on optimal policy. It is also possible to work directly on the policy function without Q value estimations.

$$\pi(s) = arg\max_{a} Q(s,a) \tag{27}$$

The policy approach is more sufficient than the Q value estimate approach for two reasons. Firstly, total reward is more important than the highest Q value at every step. In an MDP model agent takes observation from the environment and needs a decision to make. The agent needs to find the best action to take in the next step. The highest value estimated action might not be an optimal one. Secondly, environments could have a high number of actions or includes continuous actions. Continuous action like steering angle makes optimization problems become hard. It is more feasible to apply a policy approach and avoid value function-based approaches. This is where policy gradient algorithms come to help [18]

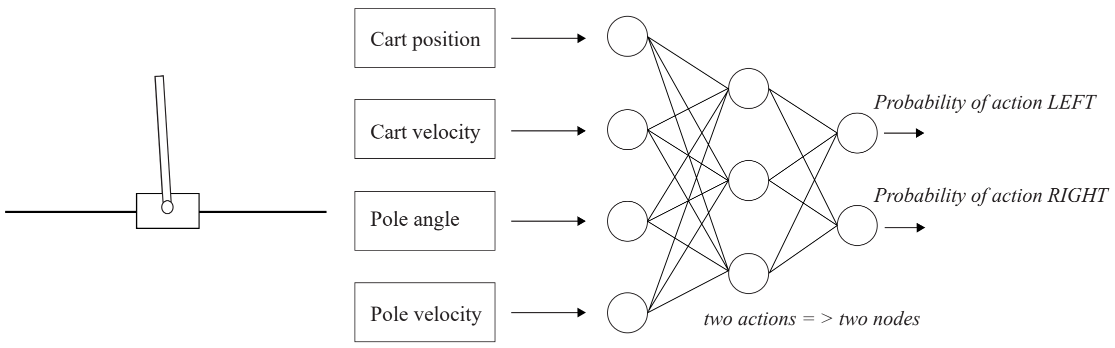

Cart position
Cart velocity
Pole angle
Pole velocity
Probability of action LEFT
Probability of action RIGHT
two actions = > two nodes

Figure 10. Cart pole policy approximation with neural network

Cart pole environment is investigated as an example. Cart pole environment outputs are velocity and position feature of the car and, angle and speed feature of the pole. The agent takes these and decides whether go left or go right. Just like DQN, a neural network is constructed with inputs of agents as input. But unlike DQN, the neural network returns probabilities of actions. The agent makes decisions according to these probabilities. Another difference with DQN with Reinforce Algorithm is DQN takes argmax of output layer but reinforce algorithm considers probability distribution.

The training objective of reinforce algorithm is to find the best possible values for the ANN so that for each state-input, it returns probabilities where it is most likely to select an action that leads to maximum reward. Reinforce learning agent updates ANN weighs by interacting with the environment to maximize expected return G. Expected return G can be expressed as Equation (28) where p is the probability of possible trajectories where $R(\tau)$ is the total reward of a specific trajectory.

$$G(\theta) = \sum_{\tau} p(\tau;\theta)R(\tau) \tag{28}$$

Calculating real expected reward value requires evaluation of all possible trajectories. It can be computationally very expensive to calculate all possible trajectories. One way to reduce this complexity is to use a sample of some number of trajectories (m) generated by the policy.[18] The gradient is given in equation (29). This gradient estimation can be used to update the weights of the policy.

$$\nabla_{\Theta} G(\theta) \approx \hat{g} := \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H} \nabla_{\Theta} \log \pi_{\theta}(a_t^{(i)}|s_t^{(i)})R(\tau^{(i)}) \tag{29}$$

Algorithm 4: Reinforce

```
Step 1   Initialize policy parameters $\theta$
Step 2   for every episode, do
Step 3       Collect M trajectory using policy $\pi_\theta$ with horizon H
                 $\tau = (s_0, a_0, \dots, s_H, a_H, s_{H+1})$
Step 4       Estimate gradient $\hat{g}$ using trajectories.
             Shown in Equation (29)
Step 5       Update ANN weights using gradient $\hat{g}$.
                 $\theta \leftarrow \theta + \alpha\hat{g}$
Step 6   end for
```

2.7 Actor-Critic Methods

Actor-critic methods combine the policy-based algorithms like reinforce and value-based algorithms like DQN. Actor-critic methods represent policy and value functions independent from each other due to their separate memory structure. Actor-critic methods select actions with their actor mechanism and criticize those decisions with critic mechanism. It can be said that actor is the policy function of actor-critic methods, and critic is the value function of actor-critic methods.

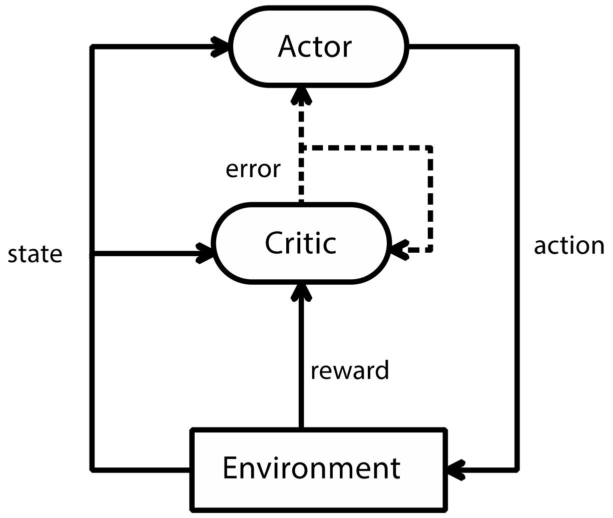


Figure 11. Actor-critic method architecture

Critic mechanism maps state to values. At the end of each iteration, the critic calculates errors to find out learning is better or worse than expected. Error calculation is given in Equation (30, where V is the value function. If the error is positive, it is more likely for that action to be selected in the future. Else if the error is negative, then it is less likely for that action to be selected in the future [19].

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{30}$$

3. USED REINFORCEMENT LEARNING ALGORITHMS

3.1. Deep Deterministic Policy Gradient Algorithm

As discussed before, DQN can solve complex reinforcement learning tasks with high dimensional observation spaces but only the ones with limited discrete action spaces. When it comes to solving continuous actions, it fails due to the curse of dimensionality problem. Lilicrap et al. [20] combined DQN and DPG algorithms to use reinforcement learning in continuous control tasks.

DDPG utilizes two separate actor and critic networks. An actor is responsible for determining best actions from probabilities by configuring the weight parameters θ. A critic is responsible for the evaluation of actions generated by the actor-network. The actor deterministically approximates the optimal policy, which means the actor generates the best possible actions for any given state. The actor-network of DDPG uses a policy-based learning model and tries to directly estimate optimal policy by maximizing rewards through gradient ascent. Critic network, on the other hand, uses a value-based learning model to estimate the quality of state-action pairs.

Algorithm 5: DDPG

```
1  Initialize actor $Q(s, a \mid \theta^Q)$ and critic $\mu(s|\theta^\mu)$ network
   with random $\theta$ parameters
2  Assign initial weights to target networks $Q'$ and $\mu'$
   with weights $\theta^{Q'} \longleftarrow \theta^Q,\ \theta^{\mu'} \leftarrow \theta^\mu$
3  Create experience replay buffer R
4  for every episode, do
5      Create task P for exploration of actions
6      Set initial state $s_1$
7      for t in range (0, T) do
8          Choose action $a_t = \mu(s_t|\theta^\mu) + P_t$ using
           exploration noise and policy at time t
9          Apply action $a_t$ and take reward $r_t$ , new state $s_{t+1}$
10         Save transition $(s_1, a_i, r_i, s_{i+1})$ in replay buffer R
11         Create minibatch of N transitions $(s_1, a_i, r_i, s_{i+1})$ from R
12         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
13         Optimize critic by loss minimization: L=$\frac{1}{N}\sum_i(y_i - Q(s_i, a_i|\theta^Q))^2$
14         Optimize the actor policy using the sampled policy gradient:
               $\nabla_{\theta^u} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
15         One step optimization of the target networks
           by making them closer to current networks
               $\theta^{Q'} \longleftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
               $\theta^{\mu'} \longleftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
16     end for
17 end for
```

3.2. Twin Delayed Temporal Difference Algorithm

Fujimoto et al. [21] claim that the Q function of DDPG is commonly made overestimations, which causes poor policy updates and significant biases. TD3 algorithm is proposed to solve this problem and increase the performance of DDPG.

Firstly, the single Q function of DDPG is changed to double Q function, which represents the twin keyword in naming. Minimum of Q value is used to form targets. This clipped double Q-learning mechanism helps to avoid any additional overestimation over the standard mechanism. Also, minimization of the Q function provides low-variance value estimations with stable learning targets that help policy updates to be safer.

Secondly, TD3 delays policy and Its target network updates relative to the Q function. That is where the delayed keyword in the algorithm name came from. The authors of TD3 recommend one policy update per every two Q-function updates.

Finally, TD3 adds noise to the target action. This noise helps to avoid deterministic policy overfitting to narrow peaks in the value estimates and smooths out Q.

Algorithm 6: TD3

```
1   Randomly initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$ and actor-network $\pi_\phi$
2   Create target networks $\theta'_1 \longleftarrow \theta_1, \theta'_2 \longleftarrow \theta_2, \phi' \longleftarrow \phi$
3   Create replay buffer R
4   $\boldsymbol{for}\ t\ in\ range(0, T)$
5       Choose action $a \sim \pi_\phi(s) + \epsilon$ with exploration noise $\epsilon \sim \mathcal{N}(0, \sigma)$
6       Run $a$ and take reward $r$ new state $s'$
7       Save transition $(s, a, r, s')$ in R
8       Create minibatch of N transitions $(s, a, r, s')$ randomly from R
9       $\tilde{a} \longleftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
10      $y \longleftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
11      Update critics $\theta_i \leftarrow argmin_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
12      **if** $t$ mod $d$ **then**
13          Update actor parameters $\phi$ with deterministic policy gradient:
            $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
14          Optimize the target networks:
            $\theta'_i \longleftarrow \tau\theta_i + (1 - \tau)\theta'_i$
            $\emptyset' \longleftarrow \tau\phi + (1 - \tau)\phi'$
15      **end if**
16  **end for**
```

3.3. Soft Actor-Critic Algorithm

Haarnoja et al. [22] announced the SAC algorithm with the contributions of UC Berkley and Google in 2018. They developed SAC to be a sample efficient, not sensitive to hyperparameters, off-policy model-free reinforcement learning algorithm.

The maximum entropy reinforcement learning framework is used in the SAC algorithm, which uses the objective function given in equation (31). Here the expectation is constructed from the policy on the left side and the actual dynamics of the system on the right. The optimal policy of the SAC algorithm maximizes expected return together with expected entropy. Temperature parameter α controls the tradeoff between policy and dynamics of the system. Authors showed in a technical report [23] that α temperature parameter could be learned automatically instead of treating it as a hyperparameter.

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t} r(s_t, a_t) - \alpha \log(\pi(s_t|a_t))\right] \tag{31}$$

SAC algorithm has a couple of different versions. In this work, open ai implementation is studied [24].

Algorithm 7: SAC

```
1  Initialize parameters $\theta$, Q function parameters $\phi_1, \phi_2$
2  Set target params to main params $\phi_{targ1} \leftarrow \phi_1,\ \phi_{targ2} \leftarrow \phi_2$
3  Create replay buffer R
4  while not converged
5      Observe state s and select action $a \sim \pi_\theta(.\,|s)$
6      Run action $a$ and save reward $r$, new state $s'$ and done signal d
7      Store transition $(s, a, r, s')$ in R
8      If its terminal reset environment state, then
9      If it is time to update, then
10         for j in range (however many updates), do
11             Sample random minibatch of B transitions $(s, a, r, s')$ from R
12             Compute targets for Q functions:
               $y(r, s', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{targ,i}(s', \tilde{a}') - \alpha \log \pi_\theta(s' |\, \tilde{a}')\right),$
               $\tilde{a}' \sim \pi_\theta(.\,|s')$
13             One step optimize Q functions by gradient descent
               $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)} \left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \quad for\ i = 1,2$
14             One step optimize policy by gradient ascent
               $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s) - \alpha log \pi_\theta(\tilde{a}_\theta(s)|s)\right),$
               Where$\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(.\,|s)$
15             Optimize target networks
               $\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1-\rho)\phi_i \quad for\ i = 1,2$
16         end for
17     end if
18 end while
```

3.4. Hindsight Experience Replay Algorithm

Sparse rewards in learning environments are one of the biggest challenges for reinforcement learning systems. Which is agent gets remarkable reward when only it reaches the goal state. Andrychowicz et al. [25] presented the HER algorithm to improve performance on sparse reward and multi-goal reinforcement learning tasks. HER algorithm can be combined with off-policy reinforcement learning algorithms to improve sample efficiency.

The idea behind HER is to store a set of experienced episodes in a replay buffer not only with the original goal but also with a subset of other goals. HER is independent of the initial distribution of environment states. Algorithm 8 shows HER algorithm in detail.

Algorithm 8: HER

```
1  Given:
2      • an off-policy RL solution $\mathcal{F}$ like TD3, SAC, DDPG
3      • a strategy $\mathbb{S}$ for sampling goals for replay buffer.
4      • a reward function $r: S\ x\ A\ x\ G \longrightarrow \mathbb{R}$
5  Initialize algorithm $\mathcal{F}$
6  Create replay buffer R
7  for every episode, do
8      Select a goal $g$ and initial state $s_0$
9      for t in range (0, T) do
10         Select action $a_t$ using the behavioral policy from $\mathcal{F}$
                   $a_t \longleftarrow \pi_b(s_t \| g)$
11         Execute action $a_t$ and observe the new state $s_{t+1}$
12     end for
13     for t in range (0, T) do
14                 $r_t := r(s_t, a_t, g)$
15         Store transition $(s_t \| g, a_t, r_t, s_{t+1} \| g)$ in R
16         Take a set of additional goals $G := \mathbb{S}$
17         for $g'$ in G do
18                 $r' := r(s_t, a_t, g')$
19             Store transition $(s_t \| g', a_t, {r'}_t, s_{t+1} \| g')$
20         end for
21     end for
22     for t in range (0, N) do
23         Take mini batch $\mathcal{B}$ from replay buffer R
24         Do one training loop using $\mathcal{F}$ and $\mathcal{B}$
25     end for
26 end for
```

4. USED ENVIRONMENT SIMULATION MODEL

Creating and using environment simulation models is one of the most important tasks for reinforcement learning researchers. Leurent et al. [26] open-sourced their environments on GitHub. In this work, their parking environment is used.

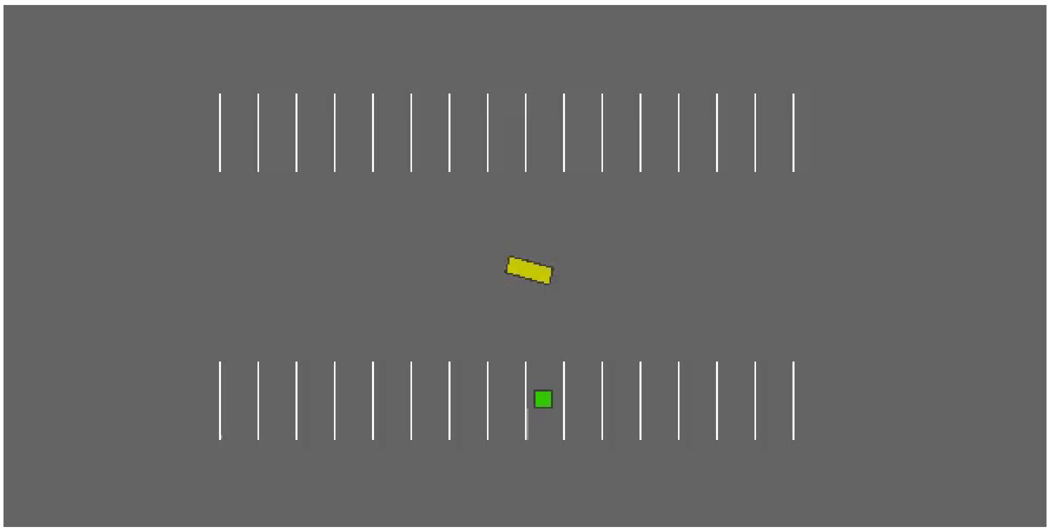


Figure 12. Parking environment

The parking environment uses bicycle kinematic model [27], which is given at equation (36) where (x, y) is vehicle position, v is speed, α is heading angle, a is acceleration command, θ is slip angle at the center of gravity, δ is steering angle.

$$\begin{gathered}\dot{x} = v\cos(\alpha + \theta) \\ \dot{y} = v\sin(\alpha + \theta) \\ \dot{v} = a \\ \dot{\alpha} = \frac{v}{l}\sin\theta \\ \theta = \tan^{-1}(1/2\tan\delta)\end{gathered} \tag{32}$$

The parking environment implements a low-level controller on top of the vehicle to track the given target speed and trajectory. The longitudinal controller is a proportional controller to control speed given in equation (33). The lateral controller, on the other hand, is a proportional-derivative controller to control heading, combined with some nonlinearities inverting the kinematic model. Equations (34) lateral position controller and Equations (35) show lateral heading controller. Where a is acceleration, v is velocity, v_r is reference velocity, K_p is controller gain, Δ_{lat} is the lateral position of the vehicle, $v_{lat,r}$ is lateral velocity command, $\Delta\psi_r$ is heading variation, v_L is lane heading, ψ_r is target heading, δ is front-wheel angle control.

$$a = K_p(v_r - v) \tag{33}$$

$$\begin{aligned} v_{lat,r} &= -K_{p,lat}\Delta_{lat} \\ \Delta\psi_r &= \sin^{-1}\left(\frac{v_{lat,r}}{v}\right) \end{aligned} \tag{34}$$

$$\begin{aligned} \psi_r &= \psi_L + \Delta\psi_r \\ \dot{\psi}_r &= K_{p,\psi}(\psi_r - \psi) \\ \delta &= \sin^{-1}\left(\frac{l}{2v}\dot{\psi}_r\right) \end{aligned} \tag{35}$$

Reward function of parking environment is given at equation (36)(36) where $s = \left[x, y, v_x, v_y, \cos\psi, \sin\psi\right]$, $s_g = \left[x_g, y_g, 0{,}0, \cos\psi_g, \sin\psi_g\right]$, $\|z\|_{W,p} = \left(\sum_i |W_i x_i|^p\right)^{1/p}$ .In order to have a narrower spike of rewards at goal p-norm is preferred instead of Euclidian norm.

$$R(s,a) = -\left\|s - s_g\right\|_{W,p}^{p} \tag{36}$$

5. RELATED WORKS

Autonomous control researchers have been studying self-driving systems for decades. DARPA announced several autonomous vehicle development competitions in 2003. These competitions led a couple of successful autonomous vehicle technologies to appear. For example, Stanford's Junior team in 2007 DARPA urban challenge came up with a hybrid A* algorithm. Sandford's team used A* algorithm to plan a trajectory and then use classical controllers to follow that trajectory [2]. MIT team on the other side used the RRT algorithm to create a trajectory and then follow that trajectory using classical control systems [28]. Evestedt et al. [29, 30] studied both path tracking and motion planning of an RRT-based approach on a trailered truck. There are studies on heuristic algorithms to make them more suitable for path planning [31, 32].

While plan trajectory with heuristic algorithms and follow using classic controller approach improving, recent studies on reinforcement learning made it possible to use reinforcement learning approaches in this kind of continuous control problems. Kendal et al. [33] were the first team that was able to use deep reinforcement learning on real-world autonomous vehicle in 2018. Their model was learning from raw pixel input [34, 35]. After training in simulation environments autonomous vehicle was able to drive successfully. There are also studies on controlling different types of robots like unmanned air vehicles. Kazim is one of the researchers that studied deep reinforcement learning algorithms to create autonomous unmanned air vehicles in 2021 [36].

The increasing diversity of reinforcement algorithms and application areas raises the need for benchmarking different algorithms with each other. Duan et al. [4] created a diversity of algorithms from simple ones like reinforce to more complex ones like the DDPG algorithm in 2016. They used commonly used old environments like card pole balancing [37–40] and walker [41–44].

6. WORK DONE

In this work, we compare TD3, DDPG, and SAC algorithms with and without HER wrapper on parking environment. Algorithms are used from the open-source stable baselines library [45]. Table 2 shows configurations that are used in experiments. Configuration parameters are chosen as default parameters that are published with the algorithm if it's possible else commonly used open-source algorithm parameters are used.

Table 2. Experiment configuration table

Parameter	**Value**
Maximum Episode Length	100
Size of Buffer	1 000 000
Batch Size	256
Time Steps Total	200 000
Learning Rate (η)	0,001
Gamma (γ)	0,95
Network Architecture	[256, 256, 256]
Tau (τ)	0,005
Policy Delay for TD3	2
Target Policy Noise for TD3	0,2
Target Noise Clip for TD3	0,5
Target Update Interval for SAC	1

For every reinforcement learning algorithm training episode starts with vehicle creation at a random location and a random goal. After that reinforcement learning algorithm (actor network) makes steering and throttle decisions. This decision is sent into the environment and the environment calculates the next state $[x, y, v_x, v_y, \cos\psi, \sin\psi]$ and reward signals using equations (32-36). The transition pair (s, a, r, s') is added to replay buffer and the algorithm trains one step. We trained 200 000 steps and observed training performance of the algorithms.

MacBook pro with core i7 hardware is used for simulations. Python programming language is chosen as the implementation language. For each of the actor, critic, and target networks same ANN architecture is chosen which has 3 hidden layers with 256 neurons for each layer. The reason for choosing this size is it’s enough for convergence and it takes a reasonable amount of time to train in simulation hardware.

7. SIMULATION RESULTS

Visualization of performance metrics is done by the open-source tensor board module of the TensorFlow Python library [46]. Figures that are given in this chapter are smoothed by Equation (37). Here smoothing weight w is chosen as 0,9 for Figure 17 and 0,6 for all the other figures.

$$\begin{cases} f(0) = & data[0] \\ f(t) = & f(t-1) * w + (1-w) * data[t] \end{cases} \tag{37}$$

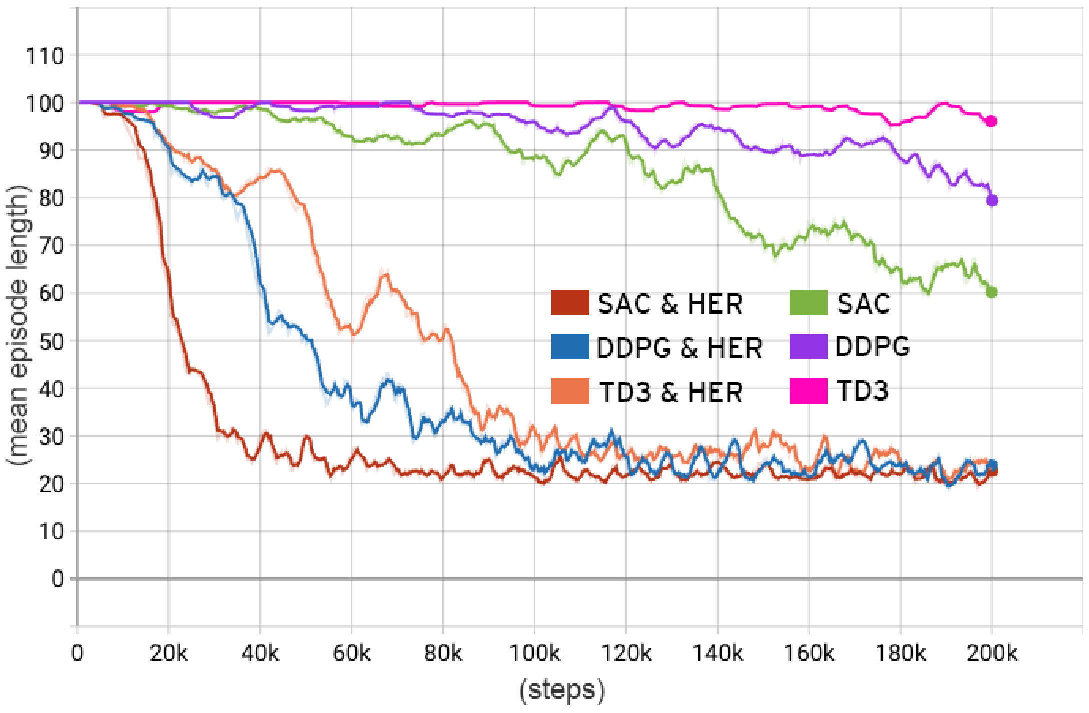


Figure 13. Mean episode length per algorithm steps shows parking speed at training

Figure 13 shows the mean length of episodes for agents. Here step represents the one action of the reinforcement learning system upon a given situation. An episode is completed when a task is successfully completed or failed. A failure can be caused by hitting a boundary or exceeding the maximum episode limit, which is 100. So, episode length can be shortened by failure or success and can have a maximum value of 100. Episode length is expected to be shortened over time, as seen in Figure 13.

The mean of episode rewards is shown in Figure 14 where higher rewards indicate better performance. Reward function indicates a negative distance between the current state and target state, as shown in equation (36).

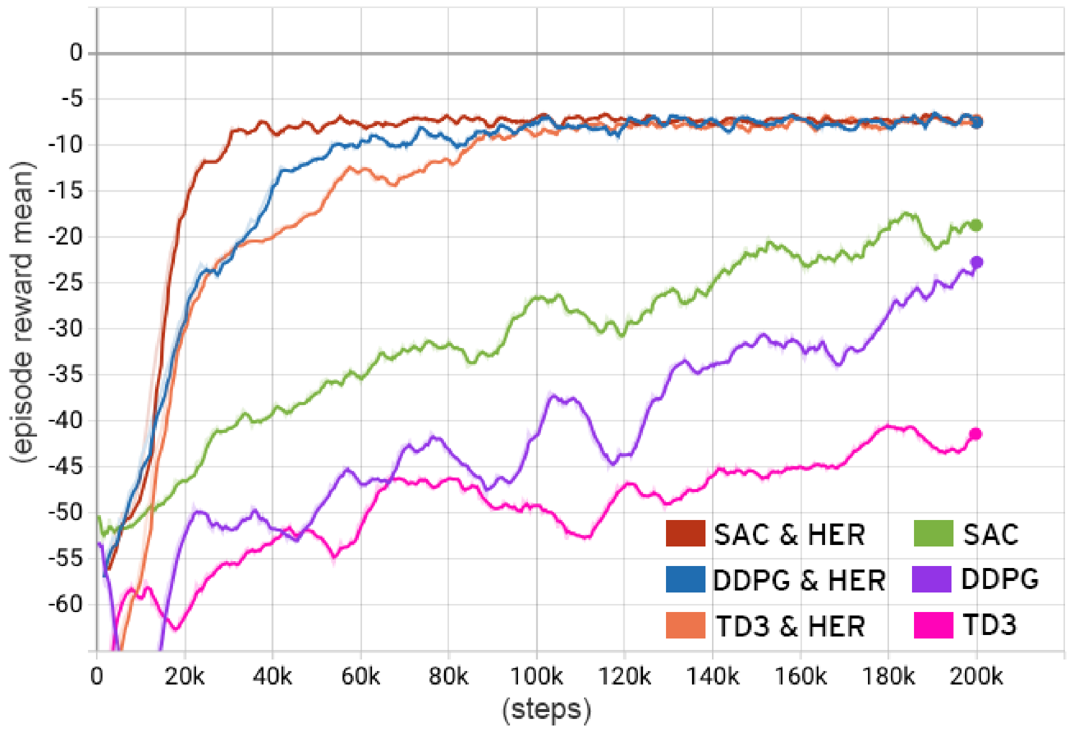


Figure 14. Mean episode reward values per algorithm training steps

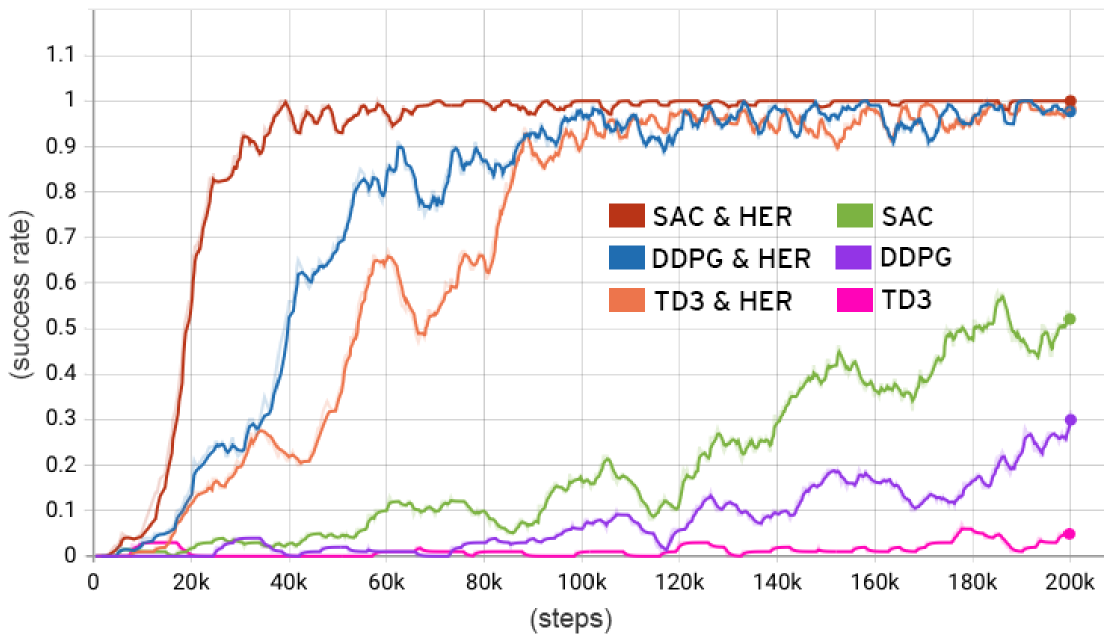


Figure 15. Average episode success rate per training steps

The success rate is shown in Figure 15 and Figure 16. The success rate is obtained as the successful episode count in the last 100 episodes at that time divided by 100. The success rate of 1 means that, last 100 parking attempt was successful in the simulation environment. Every algorithm executed 200 thousand steps. When it comes to absolute total execution time, Figure 16 shows it took 1hour 25 minutes to execute TD3, 1 hour 22 minutes to execute DDPG, 1 hour 59 minutes to execute SAC algorithms for two hundred thousand simulation steps.

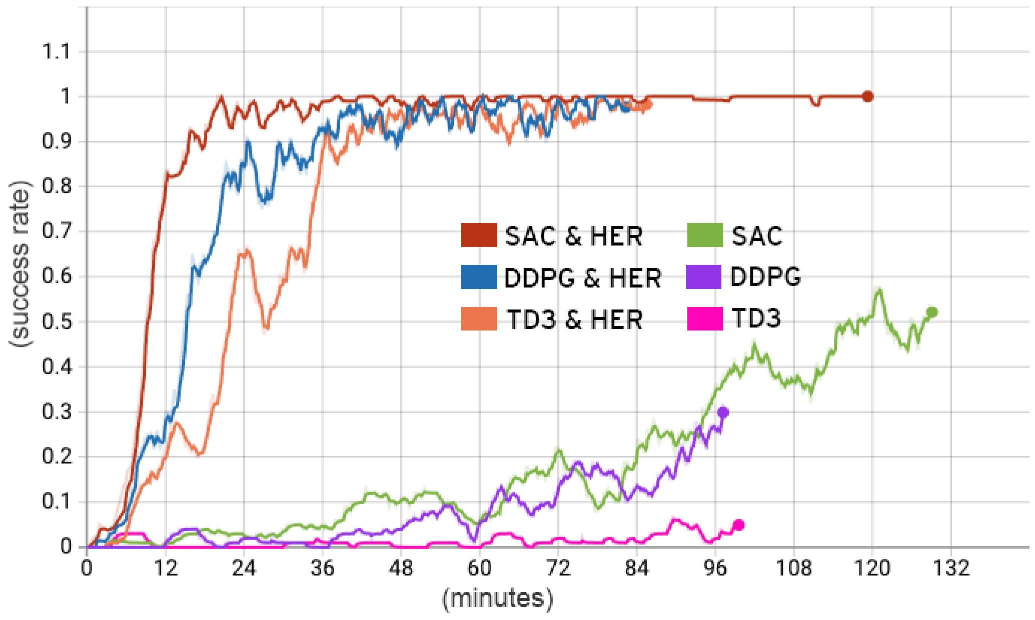


Figure 16. Average episode success rate per relative time

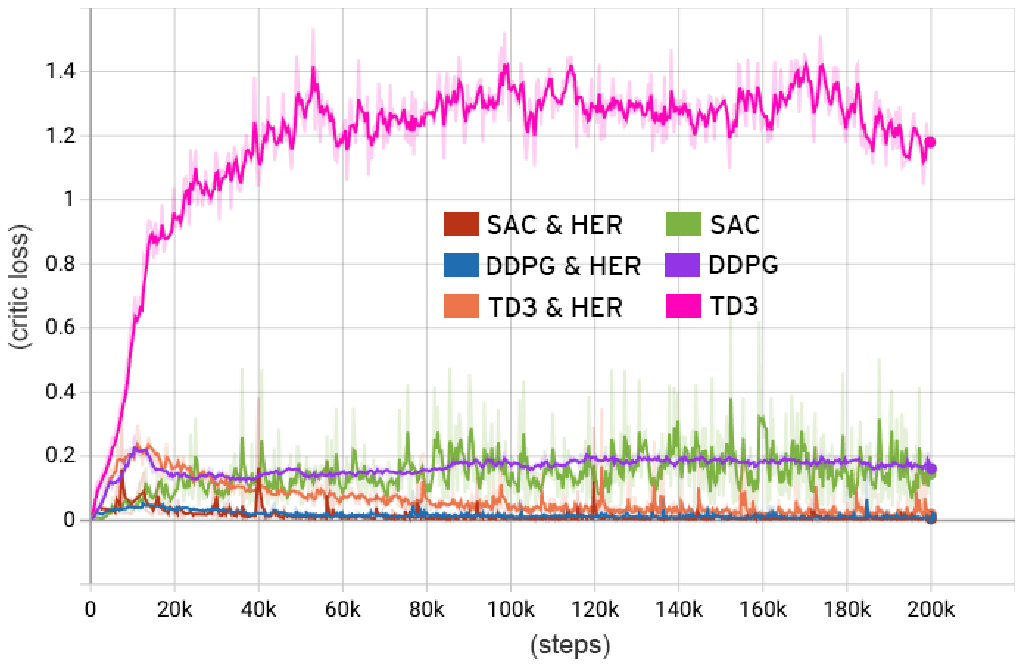


Figure 17. Average critic network loss value per algorithm training steps

Critic loss given in Figure 17 represents the training performance of critic networks in inspected algorithms. Critic loss value is simply the distance between intermediate reward value y and value of critic network. Critic loss functions are given for each algorithm in pseudo-codes (Algorithm 5, 6, 7). Here we see that the TD3 algorithm has a noticeable higher error at calculating the quality of actions that the actor mechanism of TD3 made. It seems like taking a minimum of two critic networks mechanism of TD3 doesn't help as much here as it is supposed to do.

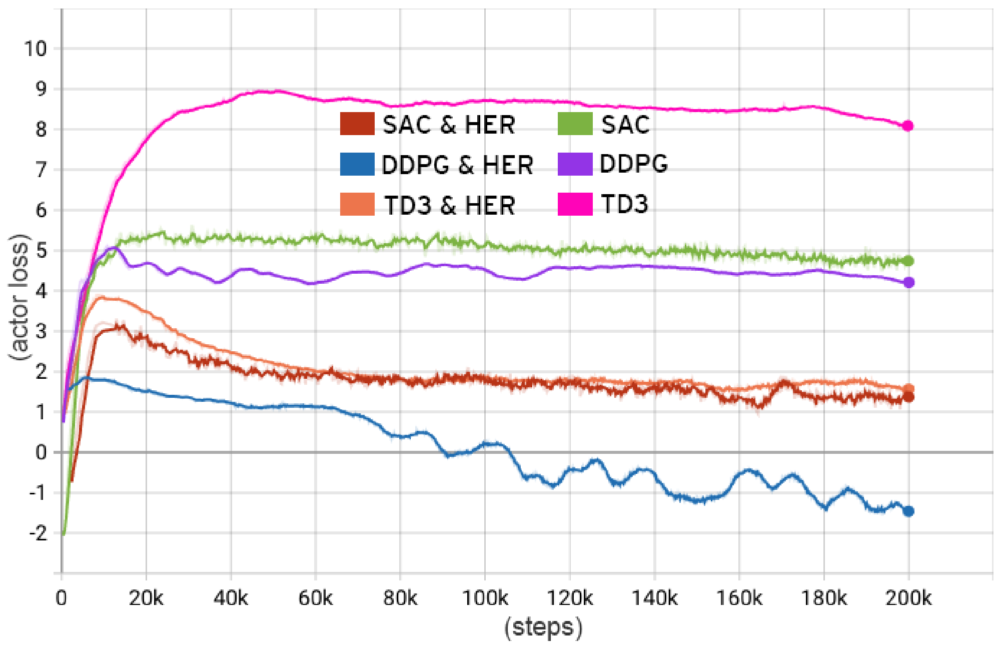


Figure 18. Average actor network loss value per algorithm training steps

Actor loss given in Figure 18 represents the training performance of actor networks in inspected algorithms. Actor loss is simply the value of the critic network in given situation pair. This chart shows us that, critic network of DDPG is not stable compared to SAC and TD3 algorithms wrapped with HER algorithm.

8. CONCLUSION AND FUTURE WORK

In this work, the autonomous control of a simplified car is considered for parking operation. Actor-critic-based reinforcement learning methods are examined to design intelligent controllers. Three types of reinforcement learning algorithms are combined with the HER wrapper algorithm and compared in a simulation environment. These methods are SAC, TD3, and DDPG.

Laurent's parking simulation environment is used for training reinforcement learning agents. All actor-critic algorithms learned how to park the car autonomously, using reward feedback and state information (location, velocity, position) from the environment.

Experiments show that with actor-critic reinforcement learning, it is possible to perform autonomous parking tasks. Experiment results show that the SAC algorithm is better than TD3 and DDPG in overall performance when it comes to the parking car continuous control problem. From Figure 15 we can conclude that the SAC algorithm needs fewer steps for convergence. From Figure 16 we can conclude that the SAC algorithm needs more time to complete the same number of steps with other algorithms, but because of its efficiency, it converges in the shortest time.

TD3 algorithm writers claim that TD3 has improved the learning speed and performance of DDPG a lot. From Figure 13, Figure 14, and Figure 15 we can conclude that in our experiment setup, TD3 is performing worse than DDPG, and TD3 is not faster than DDPG. One of the important claims of TD3 is that it reduces the overestimation bias of the critic network in DDPG. So, the critic network of TD3 is expected to have a lower loss value than the critic network of DDPG at the training stage. But from Figure 17 we can conclude TD3 has no critic network training improvements over DDPG.

Future work can be done with more realistic simulation environments that use realistic sensor inputs to localize vehicles by making dynamic mapping. Static obstacles like trees and dynamic obstacles like pedestrians can be added to have obstacle avoidance. A popular approach for solving autonomous parking problems is to plan a trajectory using heuristic search algorithms and follow that trajectory with classical control methods. Trajectory planning can be done with reinforcement learning algorithms in future work. Planning trajectory will make the control system more reliable. Lastly, it can be applied to real-world vehicles.

REFERENCES

[1] Trafik Eğitim ve Araştırma Dairesi Başkanlığı, http://trafik.gov.tr/kurumlar/trafik.gov.tr/04-Istatistik/Aylik/aralik19.pdf Emniyet Genel Müdürlüğü ÜLKE GENELİ, **(accessed March 28, 2021)**.

[2] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, S. Thrun, Junior: The Stanford Entry in the Urban Challenge, **2008**.

[3] M.G. Bechtel, E. McEllhiney, M. Kim, H. Yun, https://doi.org/10.1109/RTCSA.2018.00011DeepPicar: A low-cost deep neural network-based autonomous car, in: Proceedings - 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018, Institute of Electrical and Electronics Engineers Inc.: pp. 11–21, **2019.**

[4] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, http://arxiv.org/abs/1604.06778Benchmarking Deep Reinforcement Learning for Continuous Control, **2016**.

[5] Z. Zhao, Q. Wang, X. Li, https://doi.org/10.1016/j.neucom.2020.06.094Deep reinforcement learning based lane detection and localization, Neurocomputing. 413 328–338 **2020**.

[6] T. Dietterich, C. Bishop, D. Heckerman, M. Jordan, M. Kearns, Introduction to Machine Learning Second Edition Adaptive Computation and Machine Learning, **2010**.

[7] https://www.mathworks.com/content/dam/mathworks/ebook/gated/reinforcement-learning-ebook-all-chapters.pdf, Reinforcement Learning with MATLAB, **(accessed April 4, 2021)**.

[8] C.C. Aggarwal, Neural Networks and Deep Learning, **2018**.

[9] Micheal Nilsen, http://neuralnetworksanddeeplearning.com Neural Networks and Deep Learning, Determination Press, **(accessed April 17, 2021)**.

[10] D.E. Rumelhart, G.E. Hinton, R.J. Williams, https://doi.org/10.1038/323533a0 Learning representations by back-propagating errors, Nature. 323 533–536 (**1986**).

[11] Eric W. Weisstein, https://mathworld.wolfram.com/ControlTheory.html Control Theory -- from Wolfram MathWorld, **(accessed April 23, 2021)**.

[12] Richard S. Sutto, Andrew G. Barto, Reinforcement learning: an introduction, **2018**.

[13] C.J.C.H. Watkins, Learning From Delayed Rewards, **1989**.

[14] C.J.C.H. Watkins, P. Dayan, https://doi.org/10.1007/BF00992698Q-learning, Machine Learning. 8 279–292 (**1992**).

[15] R.S. Sutton, Learning to Predict by the Methods of Temporal Differences, **1988**.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, http://arxiv.org/abs/1312.5602, Playing Atari with Deep Reinforcement Learning, (**2013**).

[17] Markus Buchholz, https://medium.com/@markus.x.buchholz/deep-reinforcement-learning-introduction-deep-q-network-dqn-algorithm-fb74bf4d6862, Deep Reinforcement Learning. Introduction. Deep Q Network (DQN) algorithm., **(accessed May 16, 2021)**.

[18] Markus Buchholz, https://medium.com/@markus.x.buchholz/deep-reinforcement-learning-deep-deterministic-policy-gradient-ddpg-algoritm-5a823da91b43, Deep Reinforcement Learning. Deep Deterministic Policy Gradient (DDPG) algorithm., **(accessed May 17, 2021)**.

[19] R.S. Sutton, A.G. Barto, Reinforcement learning: an introduction, ser. Adaptive computation and machine learning, Cambridge, Massachusetts: MIT Press. 6 15–17 (**1998**).

[20] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, https://goo.gl/J4PIAz Continuous control with deep reinforcement learning, in: 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings, International Conference on Learning Representations, ICLR, **2016**.

[21] S. Fujimoto, H. van Hoof, D. Meger, Addressing Function Approximation Error in Actor-Critic Methods, **2018**.

[22] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, **2019**.

[23] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, S. Levine, Soft Actor-Critic Algorithms and Applications, (**2019**).

[24] Josh Achiam, . https://spinningup.openai.com/en/latest/algorithms/sac.html Soft Actor-Critic — Spinning Up documentation, **(accessed August 1, 2021)**.

[25] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba, . http://arxiv.org/abs/1707.01495 (accessed March 28, 2021) Hindsight Experience Replay, Advances in Neural Information Processing Systems. 2017-December 5049–5059 (**2017**).

[26] Leurent, Edouard, . https://github.com/eleurent/highway-env An Environment for Autonomous Driving Decision-Making, GitHub. **(accessed March 28, 2021)**.

[27] P. Polack, F. Altché, B. d'Andréa-Novel, A. de La Fortelle, The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?, in: 2017 IEEE Intelligent Vehicles Symposium (IV), IEEE, **2017**: pp. 812–818.

[28] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, N. Dormann, Stable Baselines3, GitHub Repository. (**2019**).

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, G. Research, . www.tensorflow.org.TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, **2015**.

[30] N. Evestedt, O. Ljungqvist, D. Axehill, https://doi.org/10.1109/IROS.2016.7759544Motion planning for a reversing general 2-trailer configuration using Closed-Loop RRT, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), **2016**: pp. 3690–3697.

[31] M. Hemmat, Z. Liu, Y. Zhang, https://doi.org/10.1109/ICAMechS.2017.8316535Real-time path planning and following for nonholonomic unmanned ground vehicles, in: **2017**: pp. 202–207.

[32] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli, S. Teller, https://doi.org/10.1109/ICRA.2011.5980479Anytime Motion Planning using the RRT*, in: 2011 IEEE International Conference on Robotics and Automation, **2011**: pp. 1478–1483.

[33] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, A. Shah http://arxiv.org/abs/1807.00412 (accessed January 1, 2022) Learning to Drive in a Day, (**2018**).

[34] M. Watter, J.T. Springenberg, J. Boedecker, M. Riedmiller, https://arxiv.org/abs/1506.07365 (accessed January 2, 2022) Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images, (**2015**).

[35] N. Wahlström, T.B. Schön, M.P. Deisenroth, https://doi.org/10.1016/j.ifacol.2015.12.271Peer review under responsibility of International Federation of Automatic Control, (**2015**).

[36] Kazım Burak Ünal, A Comparative Study Of Deep Reinforcement Learning Methods And Conventional Controllers For Aerial Manipulation, **2021**.

[37] D. Michie, R.A. Chambers, BOXES: An experiment in adaptive control, Machine Intelligence. 2 137–152 (**1968**).

[38] B. Widrow, Pattern recognition and adaptive control, IEEE Transactions on Applications and Industry. 83 269–277 (**1964**).

[39] P.E.K. Donaldson, Error decorrelation: a technique for matching a class of functions, in: Proceedings of the Third International Conference on Medical Electronics, **1960**: pp. 173–178.

[40] A. Stephenson, XX. On induced stability, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. 15 233–236 (**1908**).

[41] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine Learning, **2015**: pp. 1889–1897.

[42] S. Levine, V. Koltun, Guided policy search, in: International Conference on Machine Learning, **2013**: pp. 1–9.

[43] T. Erez, Y. Tassa, E. Todorov, Infinite horizon model predictive control for nonlinear periodic tasks, Manuscript under Review. 4 (**2011**).

[44] M.H. Raibert, J.K. Hodgins, Animation of dynamic legged locomotion, in: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, **1991**: pp. 349–358.

[45] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, N. Dormann, Stable Baselines3, GitHub Repository. (**2019**).

[46] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, G. Research, . www.tensorflow.org.TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, **2015**.