DEEP REINFORCEMENT LEARNING APPLIED TO POWER CONTROL IN D2D COMMUNICATIONS

LUCAS BAIÃO PIRES

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DEEP REINFORCEMENT LEARNING APPLIED TO POWER CONTROL IN D2D COMMUNICATIONS

APRENDIZAGEM POR REFORÇO PROFUNDA APLICADA AO CONTROLE DE POTÊNCIA NAS COMUNICAÇÕES D2D

LUCAS BAIÃO PIRES

ORIENTADOR: MENEZES, LEONARDO R. A. X.

DISSERTAÇÃO DE MESTRADO EM
ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPGEE.DM-769/21

BRASÍLIA/DF: JUNHO - 2021

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DEEP REINFORCEMENT LEARNING APPLIED TO POWER CONTROL IN D2D COMMUNICATIONS

LUCAS BAIÃO PIRES

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

_____

Prof. Dr. Leonardo R. A. X. Menezes – ENE/Universidade de Brasília
Orientador

_____

Prof. Dr. Leonardo Aguayo – ENE/Universidade de Brasília
Membro Interno

_____

Prof. Dr. Leonardo da Cunha Brito – EMC/Universidade Federal de Goiás
Membro Externo

BRASÍLIA, 28 DE JUNHO DE 2021.

FICHA CATALOGRÁFICA

PIRES, LUCAS B.
Deep Reinforcement Learning Applied to Power Control in D2D Communications [Distrito Federal] 2021.
xv, 101p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2021).
Dissertação de mestrado – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. aprendizagem por reforço	2. comunicação D2D
3. aprendizagem profunda	4. 5G
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

PIRES, L. B. (2021). Deep Reinforcement Learning Applied to Power Control in D2D Communications . Dissertação de mestrado em Engenharia Elétrica, Publicação PPGEE.DM-769/21, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 101p.

CESSÃO DE DIREITOS

AUTOR: Lucas Baião Pires
TÍTULO: Deep Reinforcement Learning Applied to Power Control in D2D Communications .
GRAU: Mestre ANO: 2021

Lucas Baião Pires

Departamento de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Página em branco.

ACKNOWLEDGMENTS

Pela orientação, agradeço aos professores Paulo Henrique Portela de Carvalho e Leonardo Rodrigues Araújo Xavier de Menezes.
Pela composição da banca, agradeço aos professores Leonardo Aguayo e Leonardo da Cunha Brito.
Pelo apoio financeiro, agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).
Pela presteza no atendimento, agradeço aos funcionários e professores do Departamento de Engenharia Elétrica (ENE).
Por fim, agradeço a todos que me apoiaram durante a realização deste trabalho.

RESUMO

Título: Aprendizagem por Reforço Profunda Aplicada ao Controle de Potência nas Comunicações D2D
Autor: Lucas Baião Pires
Orientador: Menezes, Leonardo R. A. X.
Programa de Pós-Graduação em Engenharia Elétrica
Brasília, 28 de junho de 2021

O rápido avanço da digitalização do mundo, juntamente com as novas aplicações e serviços suportados pelo 5G, trazem a necessidade de se aumentar a eficiência espectral dos sistemas de comunicações móveis. Uma das propostas para atingir este objetivo, e também habilitar diversas aplicações da Indústria 4.0, são as comunicações D2D. Neste trabalho, são apresentados estudos sobre a viabilidade e desempenho de técnicas de aprendizagem por reforço no controle de potência de dispositivos nas comunicações D2D.

A fim de obter soluções mais adaptativas, e capazes de aprender com o ambiente em que atuam, optou-se por explorar técnicas de aprendizagem por reforço, que são parte da família de algoritmos de aprendizagem de máquina, e que são capazes de encontrar soluções para diversos problemas utilizando experiências adquiridas, sem a necessidade de uma base de dados anterior ao processo de treinamento.

Neste trabalho, exploraram-se duas técnicas de ações discretas, que seguem diferentes paradigmas, e uma técnica de ações contínuas. Para cada uma delas, são discorridas as teorias que as compõem, bem como os processos de decisão markovianos desenvolvidos para a implementação destas. Os desempenhos das soluções são avaliados e comparados entre si, bem como a capacidade de adaptação de cada uma das soluções. Os resultados foram obtidos por meio de um simulador computacional de interferências, desenvolvido ao longo do trabalho.

As principais contribuições deste trabalho são as proposições de algoritmos baseados em aprendizagem profunda por reforço para o aumento da eficiência espectral do sistema, juntamente com a satisfação dos requisitos de qualidade de serviço do usuário primário, por meio do controle de potência de transmissão na comunicação D2D no modo *inband underlay*. Além disso, são feitas diversas análises dos algoritmos propostos, para entendimento do comportamento dessas soluções.

Palavras-chave: aprendizagem por reforço, comunicação D2D, aprendizagem profunda, 5G.

ABSTRACT

Title: Deep Reinforcement Learning Applied to Power Control in D2D Communications
Author: Lucas Baião Pires
Supervisor: Menezes, Leonardo R. A. X.
Graduate Program in Electrical Engineering
Brasília, June 28th, 2021

The fast digitalization advancement in the world, along with the new applications and services supported by 5G, brings the need to increase the spectral efficiency for the mobile communications systems. One of the propositions for helping achieve this goal, and enabling diverse applications in Industry 4.0, is D2D communications. This work presents studies on the performance and the viability of reinforcement learning techniques applied to device power control in D2D communication.

In order to achieve more adaptable solutions, and capable of learning with the environment they act upon, it was opted on exploring reinforcement learning techniques, which are members of the machine learning algorithms family, and are able to find solutions, for diverse problems, using acquired experiences, without requiring a database, prior to the training.

In this work, two discrete-action techniques were explored, which are based upon different paradigms, and one continuous-action technique. For each one of them, it was presented their fundamental theories, as well as the developed markovian processes for their implementation. The solutions' performances are evaluated, along with their adaptation capacities. The results were obtained through a computational interferences simulator, developed throughout this work.

The main contributions from this work are the proposed deep reinforcement learning-based algorithms for enhancing the system spectral efficiency, while satisfying the primary user QoS requirements, through the transmit power control in inband underlay D2D communication. Furthermore, this work provides in-depth analysis on the proposed algorithms, for a better understanding on the solutions' behaviours.

Keywords: reinforcement learning, D2D communication, deep learning, 5G.

SUMMARY

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

3GPP 3rd Generation Partnership Project. 1, 2, 40

5G Fifth Generation of Mobile Communications. 1, 2, 90

A2C Advantage Actor-Critic. xi–xiii, 32, 39, 47, 48, 50, 62, 64–66, 71, 72, 76, 77, 80, 84–86, 90, 93

A3C Asynchronous Advantage Actor-Critic. 32, 33, 39

AI Artificial Inteligence. 3

AR Augmented Reality. 1, 2

BAN Body Area Network. xiii, 57, 58, 93

BS Base Station. xii, 4, 41–48, 50, 58, 61, 69–71, 80, 81, 88, 89

CLCE Centralized-Learning Distributed-Execution. 50, 51, 65, 92

CLDE Centralized-Learning Distributed-Execution. 47–50, 60, 65, 66, 81, 92

CPU Central Processing Unit. 32

D2D Device-to-Device. xi, xii, 2–5, 14, 40–48, 50, 51, 56, 60, 61, 63, 65, 66, 68–72, 74–81, 83, 85, 87, 90, 92–94

DDPG Deep Deterministic Policy Gradient. xi–xiii, 4, 5, 35–39, 50, 51, 53, 62, 64–66, 71, 78, 79, 81, 86–88, 90–94

DPG Deterministic Policy Gradient. 35

DQL Deep Q-Learning. xi–xiii, 4, 25–27, 32, 35, 39, 47–50, 61, 63, 65, 66, 70–72, 74, 75, 80, 82–84, 90, 93

DQN Deep Q-Network. 25

DRL Deep Reinforcement Learning. 3–5, 14, 34, 35, 37–40, 45–47, 66, 90, 92–94

eMTC enhanced Machine Type Communications. 1

GAE General Advantage Estimate. 34, 39, 62

GPU Graphics Processing Unit. 32

IoT Internet of Things. 1, 2, 57, 93

ITU-R International Telecommunication Union - Radio Sector. 1, 57, 93

LTE Long Term Evolution. 1, 71

M2M Machine-to-Machine. 1
MDP Markovian Decision Process. 4, 7, 8, 14, 46
MLP Multilayer Perceptron. xi, 19–21, 23, 49–51, 53
MUE Mobile User Equipment. xi, xii, 2–4, 41–46, 48–51, 61, 65–67, 69–82, 84, 86, 88, 90, 92–94

NB-IoT Narrow Band IoT. 1
NLOS non-line-of-sight. xiii, 58
NLP Natural Language Processing. 15, 21
NN Neural Network. xi, 3, 5, 15, 19–21, 23, 31, 36, 37, 47–51, 53, 61, 62
NR New Radio. 2

p.d.f. probability density function. 7, 8, 30
ProSe Proximity Services. 2, 40

QoS Quality of Service. 3–5, 41, 43–45, 49, 65, 66, 72, 80, 81, 90, 92–95

RB Resource Block. 44–46, 94
ReLU Rectified Linear Unit. xi, 18, 61, 62
RL Reinforcement Learning. 3–6, 14, 23, 25, 39, 47, 93, 94

SGD Stochastic Gradient Descent. 22, 23
SINR Signal-to-Noise-Ratio. xi, xii, 4, 42, 44–46, 48–51, 60, 61, 65–72, 74, 76, 78, 80–88, 90, 92, 93

TD Temporal-Difference. 11–13, 34

V2V Vehicle-to-Vehicle. 4, 5
V2X Vehicle-to-Everything. 40
VR Virtual Reality. 1, 2

WINNER II Wide Area Wireless World Initiative New Radio II. xiii, 57, 58, 93

1 INTRODUCTION

The year of 2020 was marked by the COVID-19 pandemic [1, 2]. The social distancing, lockdowns and movement constraints caused by the pandemic, lead society to a great advance into digitalization. Just between Q3 2019 and Q3 2020, the mobile traffic grew 50 percent [3].

By the end of 2020, the number of 5G subscriptions, with 5G-capable devices, grew to 220 million. LTE subscriptions also increased to above 4.5 billion. In 2026, 8.8 billion mobile subscriptions are expected. 5G adoption is predicted to be significantly faster than that of Long Term Evolution (LTE), accounting for over 40 percent of all mobile subscriptions in 2026, and carrying more than half of the world's mobile data traffic [3].

The expected growth on the number of connected devices and bandwidth-hungry applications presents a challenge for the Fifth Generation of Mobile Communications (5G) [4]. The International Telecommunication Union - Radio Sector (ITU-R) has envisioned 5G to deliver higher performance and support a broader range of applications, when in comparison to its predecessor, the LTE system. 5G is expected to offer users 20 Gbps peak data rates. The spectral efficiency should be improved to 3 times higher than what LTE achieves. The system must also be able to deliver 1 ms over-the-air latency, for real-time response applications. Additionally, 5G should also support connection densities of 10^6 users/km^2, e.g., massive Machine-to-Machine (M2M) communications [5].

Besides the growth on the number of users, the increase in mobile traffic will also be boosted by new services, using new technologies such as Virtual Reality (VR) and Augmented Reality (AR), which generate large amounts of traffic [3].

In Release 13, the 3rd Generation Partnership Project (3GPP) standardized the access for Internet of Things (IoT) and M2M applications in LTE, with the proposed Narrow Band IoT (NB-IoT) and enhanced Machine Type Communications (eMTC), also called Cat-M, technologies [6]. By 2023, 14.7 billion M2M connections are expected, corresponding to half of the global connected devices. Nearly half of these connections will be from connected home applications, e.g. home automation, security, tracking applications. Connected car applications will grow the fastest from 2018 to 2023, e.g., fleet management, in-vehicle entertainment systems, emergency calling, Internet, vehicle diagnostics, navigation [7]. Six billion IoT connections are expected by 2026 [3].

IoT has been divided into three primary use cases: Massive IoT, Broadband IoT and Critical IoT.

Massive IoT is intended for wide-area cases, where large numbers of low-cost, low-

complexity devices, with long battery power and low throughput, are connected, consuming vast amount of services. Some examples of devices are sensors, wearables, trackers and meters. Broadband IoT is similar to Massive IoT, but requires higher throughput, lower latency and larger data volumes [3].

Critical IoT refers to wide- and local-area use cases, where low latency and high reliability are required. These requirements will be made possible by the advanced time-critical communication capabilities of 5G New Radio (NR). Some example cases are advanced cloud gaming, cloud-based AR/VR, cloud robotics, autonomous vehicles, and real time coordination and control of machines and processes [3].

5G is designed to also enable the implementation of the Industry 4.0, which will depend on advanced industrial automation and artificial intelligence, applied not only to the production plants, but to the whole supply chain [3].

Two proposed solutions for improving the system's spectral efficiency are Device-to-Device (D2D) communication and network densification [8].

D2D communication consists of devices talking to each other, directly, without having their signals travel the system's infrastructure [8].

Network densification refers to reducing the network cell size, and increasing the number of cells. This leads to an increased number of resources, higher data rates, lower power consumption and lower latency [8].

In this work, we focus on D2D communication. This type of communication comes into play as part of the solution for improving spectral efficiency and helping enable the numerous upcoming IoT applications [8].

1.1 D2D COMMUNICATION

D2D communication makes it possible for devices to communicate directly, mostly when in proximity, which usually enables high quality transmissions [9]. Additionally, D2D is supposed to enable a wide range of services, such as the 3GPP Proximity Services (ProSe) and a variety of IoT applications.

This type of communication has a number of modes. When focusing in spectral efficiency enhancement, D2D communication in underlay mode is the most interesting of modes, for this purpose [8, 10]. In underlay mode, the network Mobile User Equipment (MUE) shares its allocated resources with the devices performing D2D communication. Therefore, multiple devices are able to transmit using the same resources.

However, this spectrum reusage has its expenses. In order to be viable, it must happen in

such a way the MUE Quality of Service (QoS) is not violated, since the MUE is the primary customer, who owns the resource during the communication. Hence, power control and interference coordination techniques must be employed, in order to guarantee high quality for the high priority users, while enabling D2D communication, when possible.

1.2 REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a family of machine learning algorithms, that are able to learn through experience and provide solutions to problems where finding analytical solutions would prove to be very difficult or impractical [11].

RL algorithms learn how to act through trial and error processes, by interacting with the environment. It is different from supervised learning, since it does not require previously acquired data for the training process. It is also different from unsupervised learning, because it requires some knowledge from the engineer. The engineer's knowledge is passed to the algorithm in the form of a reward function, which is a real-valued signal, that measures the quality of the RL algorithm actions. In the training process, the algorithm is supposed to try and achieve high rewards. Thus, the reward is fundamental for the learning process and it shapes the solution's behaviour [11].

When interference mitigation and power control techniques are discussed, the expectations orbit around solutions that are able to deal with a great variety of dynamic situations. Analytical models for such complex conditions may be impractical to obtain. In such cases, machine learning algorithms, such as reinforcement learning, may thrive and assist in achieving satisfactory solutions.

1.3 MOTIVATION AND GOALS

During the past decades, the machine learning and Artificial Inteligence (AI) fields have been largely driven by the progress achieved by deep learning [12, 13, 14, 15]. Deep learning refers to Neural Network (NN)s with multiple hidden layers [16]. By increasing the number of hidden layers, the NN is able to represent functions with increasing complexity. It was not long ago, RL methods that were able to leverage the power of deep learning were invented [17, 18], giving birth to what is called Deep Reinforcement Learning (DRL).

Given the present opportunities for D2D communications and the benefits it may bring, this work proposes and studies DRL-based power control techniques, designed to increase the system spectral efficiency, while providing high QoS for the MUE. The main goal is to evaluate these techniques' performances, and practicability, in the inband D2D underlay

communication.

In this work, three DRL algorithms are explored. They serve as the core for the proposed power allocation frameworks. It is desired to compare different RL paradigms, as well as different application strategies, and see how they translate to the power allocation problem. This work seeks to evaluate not only the algorithms' general behaviour, but also their actions in specific controlled scenarios. Additionally, this work aims to provide a condensed base of knowledge on the topics of D2D communication, reinforcement learning, neural networks and deep reinforcement learning.

1.4 RELATED WORKS

The idea of exploring RL in underlay D2D communication has already been applied by other works. In [19, 20], Q-Learning multi-agent schemes are proposed for power control. The works present interesting concepts, but the proposed Markovian Decision Process (MDP)s are limited, since they must be compatible with tabular Q-Learning, a RL method that is not able to handle a continuous-states space. Furthermore, they do not present the MUE availability, therefore not verifying if the proposed solutions do satisfy the QoS requirements.

Deep Q-Learning (DQL) for D2D underlay communication is explored in [21], where an algorithm for both resource and power allocation is proposed. The work presents interesting results, but it also fails in verifying the primary user SINR requirements validity, by only presenting the increase in the system spectral efficiency.

Zhang et al, in [22], proposes an interesting DQL-based solution for joint resource and power allocation. It takes a different approach from what is discussed here. It suggests a scheme that optimizes only the current situation, instead of trying to generalize for a wide range of scenarios. The work takes care of verifying the effects on the interference suffered by the BS.

Besides RL, other optimization methods are also applied to the problem of power allocation in underlay D2D communication. In [23], an analytical algorithm is developed. Chen et al, in [24], applies particle swarm optimization to the problem.

DRL is also applied to other D2D communication scenarios. Nguyen et al, in [25], applied DQL to power control in D2D overlay communication. The approach focuses on maximizing energy efficiency. Xu et al, in [26], applies Deep Deterministic Policy Gradient (DDPG) to power control in Vehicle-to-Vehicle (V2V) communications, in a scenario where the vehicles share the same resources, and no primary user is considered. Nguyen et al, in [27], proposes a DDPG-based scheme for power allocation in V2V that tries to maximize en-

ergy efficiency while satisfying the V2V pairs QoS, with no primary user considered. Zhang et al, in [28], applies DDPG to joint mode selection and power control in D2D communication.

To the best of the author's knowledge, this is the first work to compare three different DRL techniques, applied to D2D underlay communication. It is also the first to propose actor-critic and continuous-actions space algorithms to solve this problem, while approaching it with solutions that generalize for a wide range of scenarios. Furthermore, this is one of the few works that focuses on spectral efficiency improvement while rigorously satisfying the primary user QoS requirements.

1.5 OUTLINE

Chapter 2 lays down the reinforcement learning concepts. It brings the base theories, problems formalizations, different algorithms and paradigms.

Chapter 3 presents the theoretical background for neural networks. It goes from the basic processing unit, the neuron, to more advanced architectures and the optimization methods applied in the training process.

Chapter 4 speaks about deep reinforcement learning and how the deep NNs fit into the RL context. It presents three different algorithms and discusses the main benefits of combining deep learning and reinforcement learning together.

Chapter 5 goes deeper into D2D communication concepts and context. It presents all the formulations made on the D2D underlay use case and the simulated system model. At last, the proposed DRL-based algorithms are presented, along with implementations.

Chapter 6 presents the computational simulations that were performed during this work, along with their methodology. The simulations results are also presented and discussed in the chapter.

Chapter 7 brings the final thoughts on the work, along with the last comments on the proposed solutions and their results. A section with future works suggestions ends the chapter.

2 REINFORCEMENT LEARNING

RL is a machine learning field where the algorithm is not told how to act, but it must find out by itself, in order to maximize a numeric reward signal [11]. In the most challenging cases, the method must find solutions to maximize the long term reward, which may lead to planning on taking smaller short term rewards. In order to find these near-optimal solutions, the algorithm performs trial-and-error search, exploring the problem and discovering the rewards.

In this chapter, the fundamental theory for understanding the reinforcement learning concepts discussed in this text are laid down. We begin by discussing how reinforcement learning problems are formalized as markovian decision processes. Next, the basic concepts of rewards, returns and value functions are described. On the sequence, we talk about the very useful Belmann equations and optimal policies. At last, we describe some classic reinforcement learning methods to obtain near-optimal policies.

2.1 MODEL-FREE AND MODEL-BASED RL

RL algorithms may be designed so the agent learns, or has access to, the environment's model. This model is a function which predicts the state transitions and rewards [29].

When the agent learns a model of the environment, it is able to plan ahead, seeing the outcomes of a range of possible choices, and making its decisions based upon them. When this approach is possible, it has proved to achieve remarkable results [30].

However, it is often not possible to have a ground-truth model of the environment. In this case, the agent would have to learn the model by experience, which creates challenges. The biggest challenge is that bias in the model might be exploited by the agent, making it so the agent performs well with respect to the learned model, but behaves sub-optimally in the real environment [29].

Algorithms that rely on knowing/learning a model of the environment are called *model-based*. The algorithms that do not depend on an environment model are called *model-free*.

Model-free algorithms are usually easier to implement and tune, besides having an inferior sample efficiency in comparison to model-based methods. They are also more popular and more extensively developed [29]. In this work, we focus on the model-free family of algorithms.

2.2 MARKOVIAN DECISION PROCESS

Reinforcement Learning is a goal-directed approach to machine learning [11]. The reinforcement learning problem is formalized as a Markovian Decision Process (MDP) [11]. This formalization was created so precise theoretical statements can be made.

The learner interacts with the environment, trying to discover the actions that maximize a numerical reward signal.

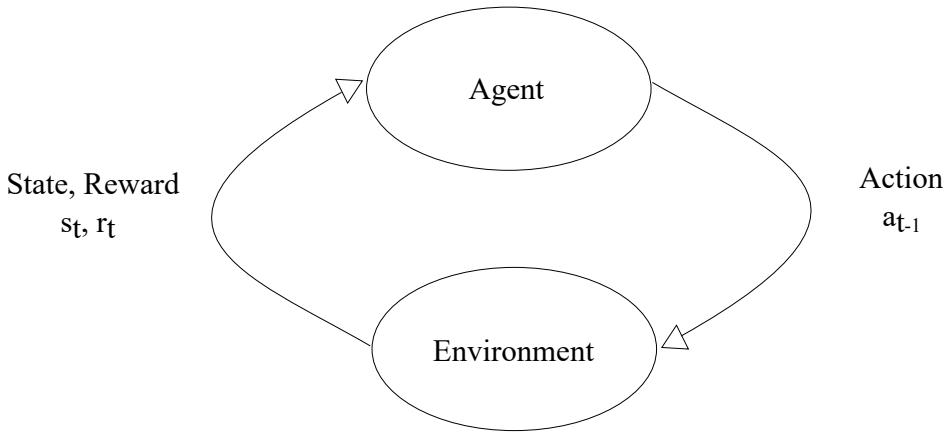

Figure 2.1 – Reinforcement Learning flow.

The learner is called agent. According to its observation $o \in \boldsymbol{O}$, the agent takes an action $a \in \boldsymbol{A}$, which changes the environment state. The environment returns a reward $r \in \boldsymbol{R}$, which measures the quality of the agent's action. $\boldsymbol{O}$ and $\boldsymbol{A}$ are called *observation*- and *action*- spaces, respectively.

The environment may be partially or fully observable to the agent. In the first case, the agent's observation o will be different from the real environment state $s \in \boldsymbol{S}$, whereas in the latter case, $o = s$. $\boldsymbol{S}$ is the *state*-space. Across the literature, s is used to denote both the agent's observation as well as the environment state. So, from now on, we will treat these terms interchangeably, and use only the s notation.

The agent maps the perceived state to an action by using a *policy*. The policy is defined by its parameters θ. The policy might be given by a deterministic function:

$$a = \mu_\theta(s) \tag{2.1}$$

It may also be stochastic:

$$a \sim \pi_\theta(.|s) \tag{2.2}$$

where π is a probability density function (p.d.f.).

The state transitions are ruled by a transition law, which may also be deterministic:

$$s_{t+1} = f(s_t, a_t), \tag{2.3}$$

or stochastic:

$$s_{t+1} \sim P(\cdot|s_t, a_t), \tag{2.4}$$

where P is a p.d.f..

The MDP and the agent, by interacting, produce a sequence of states, actions and rewards. This sequence is called a trajectory τ:

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \cdots), \tag{2.5}$$

where s_t, a_t, r_t represent the state, action and reward on step t, respectively.

The MDP dynamics are defined by the *state-transition probabilities* p, as $p : \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S} \rightarrow [0, 1]$,

$$p(s'|s, a) = P\{s_t = s'|s_{t-1} = s, a_{t-1} = a\}. \tag{2.6}$$

The probability of occurrence of a state s' depends only on the preceding state s, and the preceding action a. Therefore, the MDP is said to have the *Markov Property* [11].

2.3 REWARDS AND RETURNS

At each step, the environment passes a *reward* signal to the agent. This signal is very important, since it translates, numerically, the optimization objective. The agent seeks to maximize the reward's expectation, which means it will not necessarily look forward for maximizing short-term rewards, but to maximize the cumulative long-term reward series.

A sequence of interactions between agent and environment is called *episode*. If the environment has a terminal state, T, the episodes have finite duration. On the contrary, if there are no terminal states, the episode will go on indefinitely.

Episodes are not related to one another. So, when one episode ends, another episode begins in an state that is independent from the previous episode.

The episodes end on the same terminal states, but with different trajectories [11]. Therefore, it is interesting to measure how different the episodes were and which trajectory is better. The *return*, G, is a measurement based on the episode rewards.

The simplest return is given by the summation of the episode rewards:

$$G_t = \sum_{k=0}^{\infty} r_{t+k+1} \tag{2.7}$$

If the environment has no terminal state, the return in (2.7) might explode. Therefore, a more well-behaved expression is more popular, which is the *discounted return*:

$$\begin{aligned} G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \\ 0 \leq \gamma \leq 1 \end{aligned} \tag{2.8}$$

where γ is called *discount factor*. Note that G_t may be rewritten, in a recursive way. This will prove to be useful in the coming results.

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \\ G_t &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \cdots) \\ G_t &= r_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.9}$$

2.4 VALUE FUNCTIONS

In Reinforcement Learning, in order to evaluate policies, we use the so-called *value functions*. The main value functions are presented next. From here on, we will use π to refer to policies, in general.

2.4.1 State-Value Function

The state-value function, also called V-function, measures how good it is for the agent to be in a state s, while following a policy π. Formally, it is the expected return, when starting in s and following π thereafter. Note that the value-function of a terminal state is 0.

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right], \forall s \in \boldsymbol{S}. \tag{2.10}$$

2.4.2 Action-Value Function

When we wish to measure how good it is for the agent to take an action a, given the environment is in state s, we use the *action-value* function, also called Q-function. Formally, this function is the expected return from when action a is taken, while the environment is on

state s, and policy π is followed thereafter.

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | s_t = s, a_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right]. \tag{2.11}$$

2.5 BELLMAN EQUATIONS

Value functions satisfy a recursive relationship, similar to (2.9). Looking at the state-value function, (2.10), for any policy π, and any state s, the following condition holds:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1} | s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1}) | s_t = s], \end{aligned} \tag{2.12}$$

which is called the *Bellman equation for* v_π [11]. This equation expresses how the value of a state and its successor states relate to each other. It averages over all the possible actions, states and rewards, given initial state s, and states that $v_\pi(s)$ must be equal to the discounted value of the next state, $v_\pi(s')$, plus the expected reward for the trajectory.

Similarly, we have the Bellman equation for $q_\pi(s,a)$,

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1} | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]. \end{aligned} \tag{2.13}$$

2.6 OPTIMAL POLICIES AND OPTIMAL VALUE FUNCTIONS

In reinforcement learning, we wish to find the policy which will achieve the best return in our environment. A policy π is said to be superior to another policy π' if $v_\pi(s) > v_{\pi'}(s), \forall s \in \boldsymbol{S}$. Therefore, the *optimal policy* π^* is defined by

$$v_{\pi^*}(s) = v^*(s) \geq v_{\pi'}, \forall \pi', \forall s \in \boldsymbol{S}. \tag{2.14}$$

(2.14) states that there may be more than one optimal policy. However, all of them will have the same value function, $v^*(s)$, which is called the *optimal state-value function* [11]. The optimal policy may also be defined by

$$v^*(s) = \max_\pi v_\pi(s), \forall s \in \boldsymbol{S}. \tag{2.15}$$

The optimal policies also have the same *optimal action-value function*, q^* [11], defined as:

$$q^*(s,a) = \max_{\pi} q_\pi(s,a), \forall s, a \in \boldsymbol{S}, \boldsymbol{A}. \tag{2.16}$$

This function calculates the expected return for taking action a, in state s, and following the optimal policy afterwards. We can write q^* as function of v^*, as follows:

$$q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma v^*(s_{t+1}) | s_t = s, a_t = a]. \tag{2.17}$$

Since v^* still is a function value, it may also be written using the Bellman equation. Additionally, knowing that multiple optimal policies share the same value function, v^* is not bound to any specific policy. The Bellman equation for v^* is called the *Bellman optimality equation*, and is defined as follows

$$\begin{aligned} v^*(s) &= \max_{a \in \boldsymbol{A}} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | s_t = s, a_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[r_{t+1} + \gamma G_{t+1} | s_t = s, a_t = a] \\ &= \max_a \mathbb{E}[r_{t+1} + \gamma v^*(s_{t+1}) | s_t = s, a_t = a]. \end{aligned} \tag{2.18}$$

In a similar fashion, we obtain the Belmann optimality equation for q^*,

$$q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} q^*(s_{t+1}, a') | s_t = s, a_t = a] \tag{2.19}$$

2.7 TEMPORAL-DIFFERENCE LEARNING

Temporal-Difference (TD) learning is a family of methods to find optimal policies. It combines ideas from Monte Carlo methods and dynamic programming [11]. TD learning methods are able to learn from the experience, without having a model of the environment's dynamics. It updates its estimates based, partially, on other estimates it already has, without waiting for the final outcome. This property is called *bootstraping*.

The simplest way to estimate v_π, using TD, would be to make the update

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \tag{2.20}$$

where $V(s)$ is the estimate of $v_\pi(s)$ and α is the step size parameter. The α parameter affects how fast the algorithm converges. High values of α may cause divergence, while small values may cause the algorithm to take too long to converge [11]. In this approach,

the method updates its estimates looking only one step ahead, since the target for its update is $r_{t+1} + \gamma V(s_{t+1})$, giving it the name of *one-step TD*. This quantity, used on the update, is called *TD error*, and is denoted by:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t). \tag{2.21}$$

The reader may notice that δ_t denotes the error between $V(s)$ and its approximation $r_t + V(s_{t+1})$.

2.8 EXPLORATION AND EXPLOITATION

Reinforcement Learning control methods, in order to explore the action space, they must act in a non-optimal way, so they may find the optimal actions. This is called *exploring*. When the method decides to act by taking the optimal actions, they are said to be *exploiting*. How to choose between exploring and exploiting is a dilemma. In reinforcement learning, there are two categories of methods, *on-policy* and *off-policy* methods, that will approach this dilemma differently [11].

On-policy methods try to improve the policy that is used to make decisions. They learn action values not for the optimal policy, but for a near-optimal policy that still explores. During learning, this near-optimal policy is improved and approaches the optimal policy.

Off-policy methods attempt to improve a different policy from the one used to generate the training data. These methods use one exploratory policy, to generate behavior, and optimizes another policy, that becomes the optimal policy. The policy being learned is called the *target policy*, and the other one is called the *behaviour policy*.

A commonly used exploratory policy is the ϵ-greedy policy. It consists of choosing a random action, a_{random}, for exploration, with probability ϵ, and choosing the action that is known to return the largest action value, $q(s, a)$, with probability $1 - \epsilon$. The policy is defined as follows:

$$\pi(s) = \begin{cases} a_{\text{random}} & \text{with probability } \epsilon \\ \arg\max\limits_{a} q(s, a) & \text{with probability } 1 - \epsilon \end{cases}, \tag{2.22}$$

where $0 \leq \epsilon \leq 1$. When ϵ is close to one, the policy has a high probability of returning random actions, encouraging exploration. As ϵ decreases and approaches 0, the policy decreases the probability of returning random actions, and increases the probability of optimal actions, that maximize the action value function $q(s, a)$. Employing these optimal actions is the exploitation process.

2.9 SARSA

Sarsa is an on-policy TD control method. It consists of learning an action-value function. In an on-policy method, we must estimate $q_\pi(s, a)$ for the policy π for all states s and actions a. In order to do this, we can apply the one-step TD method to obtain $Q(s, a)$, which is the estimate of the value function $q_\pi(s, a)$. The estimation is performed by applying the following recursive update:

$$Q(s_t, a_t) \leftarrow Q(s, a) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]. \tag{2.23}$$

In this case, the TD error is $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$.

The update is done after every transition from a nonterminal state. If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1})$ is zero. The update is a function of the tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, hence the name Sarsa.

2.10 Q-LEARNING

Q-Learning is an off-policy control method, that approximates the optimal value function, q_*, regardless of the followed policy. The update is defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]. \tag{2.24}$$

As long as all state-action pairs are visited and updated, the policy being followed will not affect the algorithm convergence. This simplifies the analysis and enabled early convergence proofs [11].

Algorithm 1: Q-Learning for estimating $\pi \approx \pi_*$ [11].

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$, number of episodes M
Initialize $Q(s, a)\ \forall s, a \in \boldsymbol{S}, \boldsymbol{A}$, arbitrarily except that $Q(terminal, \cdot) = 0$
for *episode = 1, M* **do**
  Initialize s
  for *step = 1, terminal step* **do**
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
    $s \leftarrow s'$
  end
end

The classical approach for estimating the optimal action value function, using Q-Learning,

is to initialize a table with arbitrary action values for every state-action pair. As the agent visits the state-action pairs, the corresponding table values are updated. Figure 2.2 illustrates such table, where $s^i, a^i \in \boldsymbol{S}, \boldsymbol{A}$ are all the possible states and actions, respectively. This approach makes it necessary for the use of discrete environment states and discrete actions. In order to apply Q-Learning to continuous states, additional techniques are needed [17, 31].

s \ a	a^0	a^1	...
s^0	$Q(s^0, a^0)$	$Q(s^0, a^1)$	
s^1	$Q(s^1, a^0)$	$Q(s^1, a^1)$	
...			

Figure 2.2 – Illustration of a Q-values table.

Discrete states and actions may prove to be impractical for many control applications, incurring in information loss and less precise control. Depending on the discretization degree, the number of states and actions may become too large, increasing the size of the table.

A very large table would be expensive to store and it would cause the state-action visiting process to take too long, slowing the learning process.

2.11 CONCLUSION

In this chapter, we discussed the fundamental concepts of reinforcement learning. We introduced the MDP formulation, as well as the basic concepts of returns and value functions. We described the Bellman equations, and how they are used to formulate the presented reinforcement learning algorithms, that are used to obtain near-optimal policies.

Although RL algorithms are powerful, there are still some steps to take before we apply them to complex situations, such as power control in D2D communications. Nowadays, RL has been combined with neural networks, resulting into Deep Reinforcement Learning (DRL). A family of algorithms that are capable of solving diverse complex problems with high problem abstraction and easy-to-implement continuous state space solutions [32, 33, 34].

The next chapter presents neural networks. It is the missing piece before heading to DRL.

3 NEURAL NETWORKS

As defined by Haykin, in [35], "a NN is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use."

NNs are machines, that seek to model the brain's working mechanism, in order to perform tasks or functions of interest. It is usually implemented by software, in digital computers. Similar to the brain, artificial NNs have a basic processing unit, called neuron. The network is able to learn with information, and store the acquired knowledge into interneuron connection strengths, called synaptic weights [35].

In the past decades, artificial NNs have achieved many breakthroughs in diverse fields. Some examples of these are Natural Language Processing (NLP) [12], image classification [13], autonomous vehicles [14], health care [15], and many others.

In this chapter, we will cover the basics of this topic, so the reader may follow this work's coming subjects. We begin by introducing the NN's primary processing unit, the neuron, and explaining the computation it performs. Next, we present a classic neural network architecture and describe its features. At the last part of the chapter, we discuss classical optimization techniques, and how they are applied in order to make the networks learn with data.

3.1 NEURONS

Similar to the neuron of a human brain, the neuron of artificial neural network is the fundamental processing unit. The neurons and its connections constitute the neural network. A neuron is pictured in Figure 3.1. There is also a simpler, more practical neuron representation, seen in Figure 3.2. It is called the *architectural graph*, and it is a partial definition of the neuron, used to describe the layout of neural networks.

The connections between neurons, also called *synapses*, are characterized by numeric weights. A signal $x_{kj} \in \mathbb{R}$ at the input of synapse j connected to neuron k is multiplied by weight $w_{kj} \in \mathbb{R}$. The synaptic signals are multiplied by the weights and added. This constitute a *linear combiner*. The resulting signal is passed to an *activation function*. The activation function may introduce non-linearity to the neuron, making it suitable for dealing with non-linear problems. The activation function may also limit the neuron's output, so it assumes only finite values. This process is depicted by Figure 3.1.

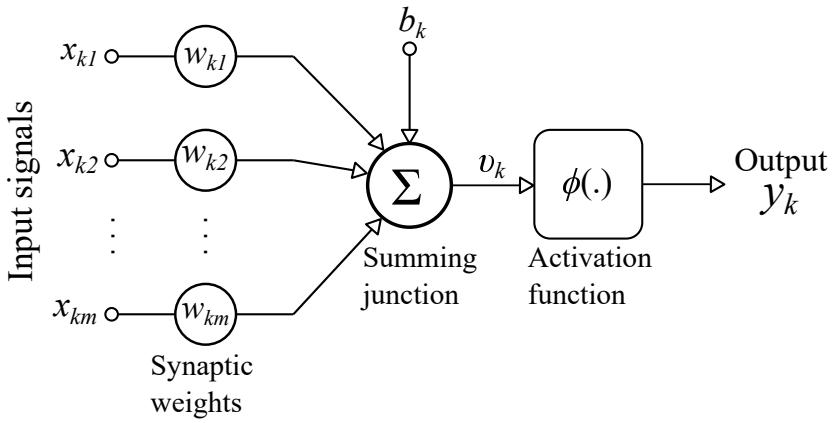

Figure 3.1 – Neuron illustration.

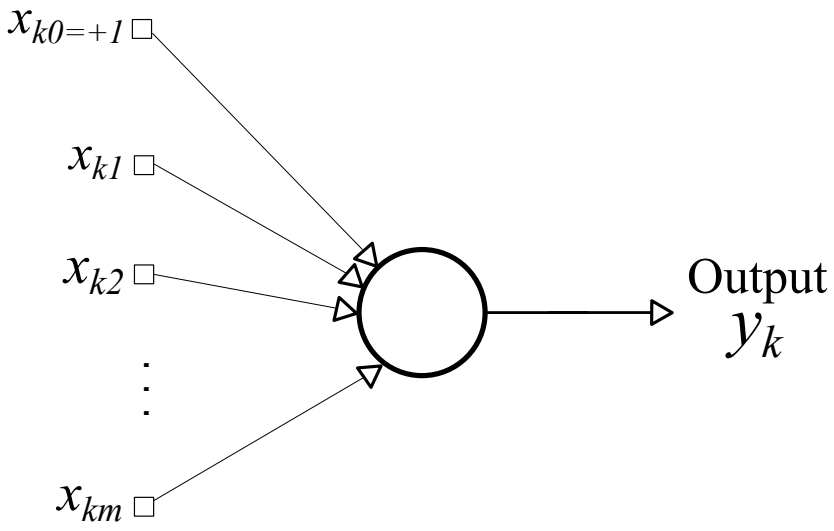

Figure 3.2 – Neuron architectural graph.

Mathematically, the linear combiner output is given by

$$v_k = \sum_{j=1}^{m} w_{kj} x_{kj}. \tag{3.1}$$

The neuron's output is defined as

$$y_k = \phi(v_k + b_k), \tag{3.2}$$

where ϕ is the activation function, and b_k is an arbitrary bias term.

3.1.1 Activation Functions

We will define some of the most common activation functions.

At first, we have the *sigmoid* activation function [35]. It is defined by

$$\phi(\nu) = \frac{1}{1 + \exp(-a\nu)}. \tag{3.3}$$

The sigmoid function is a "S"-shaped function. The a parameter defines the function's slope. Its values are limited by $0 \leq \phi(\nu) \leq 1$. The function plot can be seen in Figure 3.3.

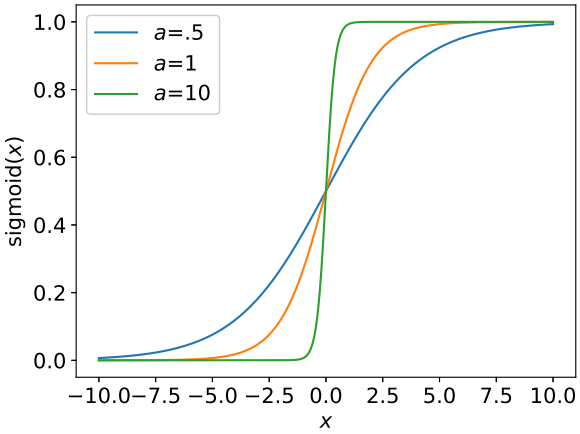

Figure 3.3 – Sigmoid function for different values of a.

Next, there is the *hyperbolic tangent*, or *tanh*, activation function [35]. It is described by

$$\phi(\nu) = \tanh(\nu) = \frac{\sinh(\nu)}{\cosh(\nu)} = \frac{e^{\nu} - e^{-\nu}}{e^{\nu} + e^{-\nu}}. \tag{3.4}$$

The tanh function's output is limited by $-1 \leq \phi(\nu) \leq 1$. The biggest difference between

the sigmoid and tanh functions is their output ranges. A drawback present in these functions is that their saturations may cause gradient-based learning to be difficult [36]. The tanh function plot is displayed on Figure 3.4

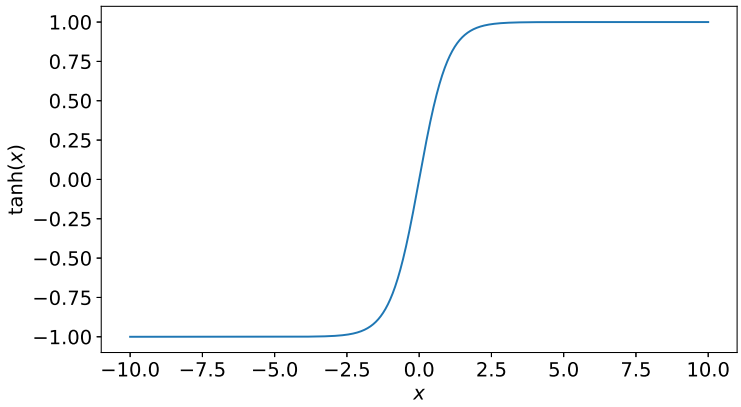

Figure 3.4 – Hyperbolic tangent function plot.

Another activation function is the Rectified Linear Unit (ReLU) function [36].The mathematical definition is

$$\phi(\nu) = \nu^{+} = \max(0, \nu). \tag{3.5}$$

Its output is characterized by assuming the same value as the input, if the input is positive, or zero otherwise. A good characteristic of this function is its low computational cost, and the simplicity in differentiating it. One problem with this function is that the networks will not learn, via gradient-methods, on examples where their activation is zero. The ReLU function plot is seen on Figure 3.5.

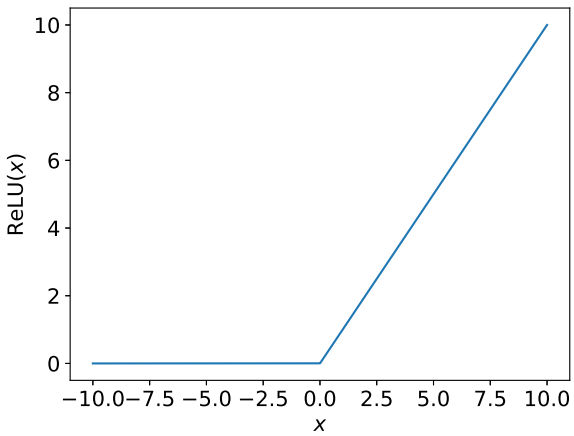

Figure 3.5 – ReLU function plot.

The last activation function to be presented is the *softmax* function [16]. It is defined as

follows:

$$\phi(\nu_i) = \frac{\exp(\nu_i)}{\sum_j \exp(\nu_j)}. \tag{3.6}$$

This function takes an input of real numbers, and generates a probability distribution based on it. Therefore, we have that

$$\sum_j \phi(\nu_j) = 1. \tag{3.7}$$

Most generally, the softmax function is used on the output of classifiers, in order to represent the probability distribution over different classes. It is rarely used inside the model itself.

The choice of which activation function to use depends on the purpose the neural network is serving.

3.2 FEEDFORWARD NEURAL NETWORKS

The neurons of a network, and signal flows inside the network, may be disposed according to different *architectures*. In this section, we will describe the most common architecture, the *feedforward neural network*.

In a feedforward neural network, the neurons are disposed in layers [35]. We have an input layer of source nodes, that projects onto another layer, called the hidden layer. The hidden layer nodes may project onto other hidden layers, or onto the output layer. The network may have zero, one, or more than one hidden layers. Therefore, this kind of NN is also called Multilayer Perceptron (MLP). Adding more hidden layers may enable the network to extract higher-order statistics from its input [35].

Computing the NN's output is performed only in one direction, from the input layer to the output layer, and it is called a *forward pass*. All the neurons in one layer connect to all the neurons in consecutive layers. This means there are weighted links among these neurons. A MLP illustration is given by Figure 3.6.

The goal of a MLP is to approximate some function f^*. The network defines a mapping $y = f(x;\theta)$, where θ is the set of parameters, or weights, of the NN. The NN learns the θ that results on the best approximation of f^*.

The MLP may be seen as series of composed functions [36]. For example, $f(x) = f^{(n)}(f^{(n-1)}(\cdots f^{(2)}(f^{(1)}(x))))$, where $f^{(1)}$ is the output of the first layer of the network, $f^{(2)}$ is the output of the second layer of network, and so forth. The number of layers, n, defines the *depth* of the model. That explains the terminology *deep learning*, when referring to NNs.

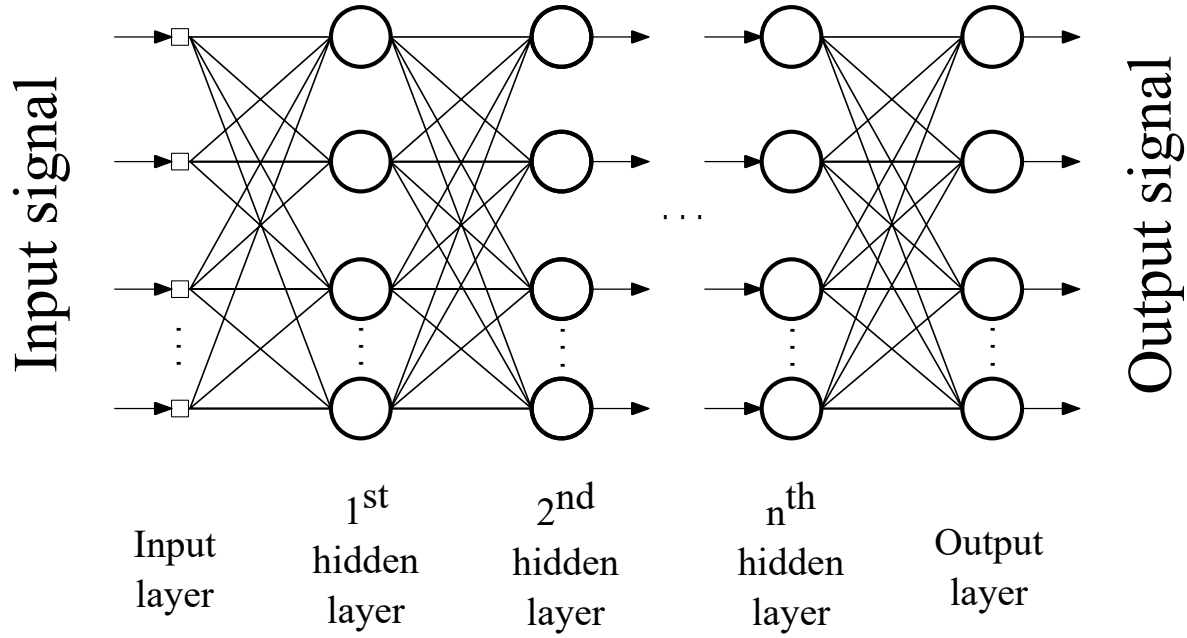

Figure 3.6 – MLP illustration.

The final layer of the NN is called the *output layer*, while the first layer is the *input layer*.

During training, the NN parameters are updated in order to approximate $f(x)$ to $f^*(x)$. The inputs x are presented to the network along with the corresponding output, or *label*, $y \approx f^*(x)$. The training specifies how the output layer should behave for each input. The outputs of the other layers are not specified by the training data. In the training process, the learning algorithm decides by itself how the other layers should behave. Since the desired input for these layers is not known, or is not defined by the training data, these layers are called *hidden layers*.

The MLP operations may be represented as matrix operations. The output of layer n is given by

$$\boldsymbol{y}^{(n)}_{1\times M^{(n)}} = \phi\left(\boldsymbol{x}^{(n)}_{1\times N^{(n)}} \cdot \boldsymbol{w}^{(n)}_{N^{(n)}\times M^{(n)}}\right). \tag{3.8}$$

In (3.8), $\boldsymbol{x}_{1\times N^{(n)}}$ is the n-th layer input vector. For hidden layers and the output layer, $\boldsymbol{x}^{(n)} = \boldsymbol{y}^{(n-1)}$. For the input layer, the input vector will be the same as the NN's.

The n-th layer weights matrix is given by $\boldsymbol{w}^{(n)}_{N^{(n)}\times M^{(n)}}$, which represents the connections between the nodes on layer n and $n-1$. When there are no connections between certain nodes, the weight for the link between them is simply zero valued.

The n-th layer output vector is denoted by $\boldsymbol{y}^{(n)}_{1\times M^{(n)}}$. It is the outcome of the activation function, ϕ, over the result of the linear combination of the layer inputs and weights. We assume that all neurons in a layer have the same activation function.

$N^{(n)}$ denotes the n-th layer input vector size, and $M^{(n)}$ denotes number of neurons in the layer.

The MLP's output is obtained by performing the operation in (3.8) from the input layer to the output layer, consecutively, in the forward direction.

There are other very important, and promising, architectures. Convolutional neural network is an architecture widely used in image classification [13, 37]. Long-short term memory is a successful architecture on time-series forecasting [38] and NLP [39]. Transformer is another architecture, that has recently taken over the NLP field by storm, due to its great capabilities [12, 40].

These architectures will not be elaborated upon, because they are out of this work's context. However, the reader is encouraged to read about them. In this work, we mainly use the MLP architecture.

3.3 GRADIENT-BASED LEARNING

3.3.1 Steepest descent method

Most machine learning algorithms make use of gradient-based optimization [36]. It consists of minimizing, or maximizing, an *objective function*. In our case, in deep learning, we wish to minimize an objective function, which will be called *loss function*, $L(x;\theta)$. In deep learning, we wish to solve the problem

$$\begin{aligned} &\min_{\theta} L(x;\theta) \\ &L(x;\theta^*) \leq L(x;\theta), \forall\theta \end{aligned} \tag{3.9}$$

which translates to finding a set of NN optimal parameters θ^* that minimizes the loss function.

Given a function with multiple inputs, $f(\boldsymbol{x})$, the partial derivative of this function, $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$, measures the change on f caused by variable x_i at point $\boldsymbol{x}$. The gradient of f, denoted by $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$, is a vector containing all the partial derivatives of f.

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \cdots, \frac{\partial f(\boldsymbol{x})}{\partial x_i} \right) \tag{3.10}$$

In order to minimize f, we may use the *directional derivative*, so we may find the direction in which f decreases the fastest [36]:

$$\begin{aligned} &\min_{\boldsymbol{u}, \boldsymbol{u}^T\boldsymbol{u}=1} \boldsymbol{u}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \\ &= \min_{\boldsymbol{u}, \boldsymbol{u}^T\boldsymbol{u}=1} ||\boldsymbol{u}|| \cdot ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})|| \cos\theta, \end{aligned} \tag{3.11}$$

where $\boldsymbol{u}$ is a unit vector, or the direction, and θ is the angle between $\boldsymbol{u}$ and the gradient vector. Since $\boldsymbol{u}$ is a unit vector, $||\boldsymbol{u}|| = 1$. Ignoring factors that do not depend on $\boldsymbol{u}$, the problem in (3.11) reduces to $\min_{\boldsymbol{u}} \cos\theta$. Therefore, we wish to find $\boldsymbol{u}$ so that $\cos\theta = -1$. This means the direction opposite to the gradient's direction is the direction in which f decreases. Moving in this direction, in order to find the minimum value of the function, is known as the *method of steepest descent*, and is described as follows:

$$\boldsymbol{x}' = \boldsymbol{x} - \alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x}), \tag{3.12}$$

where α is the *learning rate*, which determines the step size on the minimizing direction. Applying the same method to minimize the loss function $L(x, \theta)$, we arrive at the optimization step:

$$\theta' = \theta - \alpha \nabla_{\theta} L(x, \theta). \tag{3.13}$$

3.3.2 Stochastic Gradient Descent

Usually, the loss function may be decomposed in a sum over training examples, as follows [36]:

$$J(\theta) = \mathbb{E}[L(\boldsymbol{x}, \boldsymbol{y}, \theta)] = \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}, \theta), \tag{3.14}$$

where $J(\theta)$ is the resulting loss function. We approximate the loss expectation by a summation over the losses for the m inputs and outputs in the dataset.

When taking the gradient of $J(\theta)$, we obtain

$$\nabla_{\theta} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\theta} L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}, \theta). \tag{3.15}$$

This operation has a computational cost of $O(m)$. When the dataset grows to the size of billions of examples, it may become impractical to compute (3.15).

The advantage of Stochastic Gradient Descent (SGD) relies on it approximating the gradient of an expectation. The approximation may be computed using small sets of data, called *minibatches*, of size m'. Therefore, the estimate of the gradient is

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\theta} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}, \theta). \tag{3.16}$$

The method update is given by:

$$\theta \leftarrow \theta - \alpha \boldsymbol{g}, \tag{3.17}$$

where α is the learning rate.

SGD is not guaranteed to find the global minimum, but is capable of finding low values of the cost function in reasonable time. It has proven to be quite useful on deep learning, since it provides a scalable way of training nonlinear models on large datasets [36].

There are many more optimizers applied to deep learning. Some of the most famous are Adam [41], AdamW [42] and RMSprop [43, 44], which are more complex than SGD. We will not elaborate further on these optimizers, since this would be out of work's scope. Throughout this work, we used Adam and AdamW optimizers for our deep learning models. This choice was made after empirical experiments that indicated the results obtained with them were good.

3.4 BACK-PROPAGATION

Back-propagation is the technique for implementing gradient descent in weight space for a NN [35]. The idea is to compute the partial derivatives of a function $F(\theta, \boldsymbol{x})$, with respect to all network parameters θ, for an input vector $\boldsymbol{x}$. In our case, we wish to obtain the gradient of the total loss function, $\nabla_\theta J(\theta)$, so we may update θ and minimize the loss function [36].

We will not present the demonstration for the back-propagation algorithm, since it is extensive. This method is very famous, and widely used in deep learning. Therefore, the reader may easily find references about it in the literature [35, 36].

3.5 CONCLUSION

In this chapter, we covered the basic concepts of artificial neural networks. We presented the neuron, the network basic processing unit, and a more complex structure, the MLP, which is the result of a multitude of connected neurons. At last, we discussed optimization methods for updating the NN's parameters, and making the network learn.

Neural networks are very powerful and versatile. We saw that, in deep learning, by just adding nodes and layers to a network, it is possible to increase it's non linearity. Furthermore, these structures are able to learn with sampled data approximate functions, making them good candidates for function approximators for the policies and value functions in RL. Neural networks also work with continuous values, a great advantage over the discrete tabular representations we saw in Chapter 2, enabling us to work with continuous states and action spaces.

Deep Reinforcement Learning comes from combining Deep Learning and Reinforcement

Learning. With the knowledge of reinforcement learning, seen in Chapter 2, and neural networks, we are ready to proceed to this new topic, in the next chapter.

4 DEEP REINFORCEMENT LEARNING

In the past decade, deep learning has proven to be capable of extracting high-level features from raw data, achieving breakthroughs in computer vision [13, 37], and speech recognition [45, 46]. However, employing deep learning along with RL presents several challenges [17].

The first challenge is that most deep learning applications require large amounts of manually labeled data, for supervised learning. On the other hand, RL algorithms must learn from a scalar reward signal that is usually sparse, noisy and delayed [17]. The delay between action and reward may be long, which may be more complex than the direct input-output mapping found in supervised learning. There is also the issue of deep learning algorithms assuming the data samples to be independent, while in RL we often find sequences of highly correlated states. Additionally, in RL, the data distribution often changes according to the algorithm's learning, which may be problematic since deep learning methods usually assume a fixed data distribution. Fortunately, there are techniques that are able to handle these problems, and still leverage the benefits of deep learning on reinforcement learning.

4.1 DEEP Q-LEARNING

The DQL algorithm revolves around the idea of implementing the Q-Learning algorithm, leveraging the power of deep learning, in order to approximate the optimal value function, q^*, with the Bellman equation given in (2.19), by a parametrized function approximator $Q(s, a; \theta) \approx q^*(s, a)$.

In our case, the function approximator will be a neural network with parameters θ, and this network will be called a Deep Q-Network (DQN). The DQN will be trained by minimizing the loss functions $L_i(\theta_i)$, that changes on every i-th iteration,

$$\begin{aligned} L_i(\theta_i) &= \mathbb{E}_{s,a\sim p(\cdot)}[(y_i - Q(s, a; \theta_i))^2] \\ y_i &= \mathbb{E}_{s'\in \boldsymbol{S}}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a], \end{aligned} \tag{4.1}$$

where y_i is the target for the i-th iteration. We can see in (4.1) that the targets depend on the parameters θ_{i-1}, which contrasts with supervised learning, where the targets are fixed before the beginning of the training.

Differentiating (4.1), we have

$$\nabla_{\theta_i}(\theta_i) = \mathbb{E}_{s,a\sim p(\cdot);s\in \boldsymbol{S}}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)]. \tag{4.2}$$

Instead of calculating the expectation on (4.2), it is common practice to optimize the loss function by stochastic gradient descent. If θ is updated after every time-step, and the expectations are replaced by single samples from the transitions probabilities distribution p, then we obtain the Q-learning algorithm [17].

Seeking to mitigate the problems of correlated data and non-stationary distributions, DQL uses an *experience replay* mechanism, which randomly samples previous transitions, smoothing the training distribution over many past behaviours.

The experience replay stores the agent's experience at each time step $e_t = (s_t, a_t, r_t, s_{t+1})$, in a finite memory buffer $\mathcal{D} = e_1, e_2, \cdots, e_N$, pooled over many episodes, called *replay memory*. During training, Q-learning updates are applied to mini-batches of experiences, sampled from $\mathcal{D}$. After experience replay, the agent selects an action according to an ϵ-greedy policy.

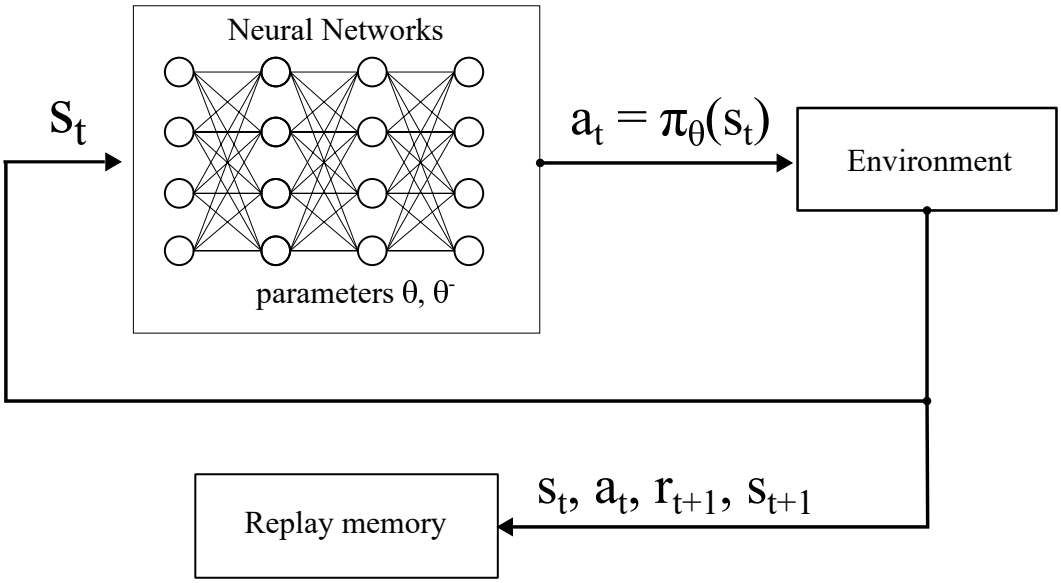

Figure 4.1 – Deep Q-Learning illustration.

An additional improvement to DQL is to use a separate network for generating the targets y_i in the Q-Learning update [47]. After every E updates, the Q network is cloned to a $\hat{Q}$ target network, that generates the update targets y_i. This feature contributes to the algorithm's stability, in comparison to traditional Q-learning, where an update that increases $Q(s_t, a_t)$ frequently increases $Q(s_{t+1}, a_t)$ as well, for all a, also increasing the target y_i, which may lead to oscillations or divergence of the policy. Using an older set of parameters, to generate the targets, causes a delay between the moment of the update and the moment on which the update affects y_i, making divergences more unlikely. We denote the target network parameters by θ^-. Figure 4.1 illustrates the DQL algorithm.

It is also beneficial to clip the error term $r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta_i)$ between -1 and 1. This feature also contributed for the algorithm's stability [47].

The DQL algorithm is presented in Algorithm 2.

Algorithm 2: Deep Q-Learning with experience replay.

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights θ^-
for *episode*=1, M **do**
  Initialize s
  for $t = 1$, *terminal step* **do**
    Observe s and choose a using an ϵ-greedy policy
    Take action a; observe r and s'
    Store transition $e_t = (s, a, r, s')$ in $\mathcal{D}$
    Sample random minibatch of transitions $e_i = (s_i, a_i, r_i, s_{j+1})$ from $\mathcal{D}$
    if *episode terminates at step* $i + 1$ **then**
      $y_i = r_i$
    end
    else
      $y_i = r_i + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-)$
    end
    Perform a gradient descent step on $(y_i - Q(s_i, a_i; \theta))^2$ with respect to θ
    Every E steps reset $\hat{Q} = Q$
  end
end

4.2 POLICY OPTIMIZATION AND VALUE OPTIMIZATION

Up until now, we focused on algorithms that learn by performing *value optimization*. Value optimization algorithms focus on learning approximators of the optimal value functions, such as the optimal action-value function, q^*, or the optimal state-value function, v^*. This optimization often is off-policy, since the training data is collected at any point during training, regardless of the policy used to obtain it [29]. The resulting policy is equivalent to choosing the actions that maximize the approximated value function, for each state s.

The other optimization family is called *policy optimization*. These methods represent the policy explicitly as a function $\pi_\theta(a|s)$. The policy parameters, θ, are optimized directly by gradient optimization performed on the objective function, or an approximation, $J(\pi_\theta)$. Usually, these are on-policy algorithms, since they perform the policy update only with the data acquired by that policy [29]. Policy optimization algorithms may also make use of value function approximations, leveraging them to enhance the quality of its updates.

In comparison to value optimization, policy optimization algorithms tend to be more stable, since they optimize the policy directly based on the objective function. In value optimization, the policy performance is indirectly optimized, since the algorithm trains a

function approximator to satisfy a self-consistency equation [29]. Function approximation, bootstrapping and off-policy training, all contribute to a divergence in value optimization methods [11]. However, value optimization methods are substantially more sample efficient, and may benefit of data acquired by different policies, which is quite useful when dealing with the exploration-exploitation dilemma [32, 48].

Currently, some of the most acknowledged, high-performance, state-of-the art, algorithms are DeepMind's Agent 57 [48], which is a value optimization algorithm, and OpenAI's PPO [33, 34, 49], which is a policy optimization algorithm. In the next sections, we will go through two popular policy optimization algorithms: REINFORCE [50] and A2C/A3C [18].

4.3 POLICY GRADIENT

We consider a stochastic, parameterized policy, π_θ. In contrast with value optimization algorithms, now we wish to maximize our objective function

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G_t(\tau)], \tag{4.3}$$

where $G_t(\tau)$ is the finite-horizon undiscounted return, given by

$$\begin{aligned} G(\tau) &= \sum_{t=0}^{T-1} r_{t+1} \\ \tau &= (s_0, a_0, r_1, s_1, a_1, \cdots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T), \end{aligned} \tag{4.4}$$

where τ is the trajectory.

We intend to optimize the policy by gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)|_{\theta_k}, \tag{4.5}$$

where $\nabla_\theta J(\pi_\theta)$ is called the *policy gradient*, which is key for optimizing *policy gradient algorithms*.

In order to numerically compute the policy gradient, we need to derive the gradient analytical expression and use a Monte Carlo estimate of the gradient's expected value, from a finite number of interactions between agent and environment.

In order to derive the analytical expression, we begin by defining the probability of a trajectory.

$$P(\tau|\theta) = p_0(s_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t), \tag{4.6}$$

where $p_0(s_0)$ is the initial state s_0 probability of occurrence.

We also make use of the *log-derivative trick*, which is based on the derivative of $\log f(x)$,

$$\frac{d}{dx}\log f(x) = \frac{1}{f(x)}\frac{d}{dx}f(x). \tag{4.7}$$

Applying the logarithm function to (4.6), we arrive at the *log-probability of a trajectory*,

$$\log P(\tau|\theta) = \log p_0(s_0) + \sum_{t=0}^{T}\left(\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)\right). \tag{4.8}$$

For the next steps, we use the fact that the gradients of $p_0(s_0)$, $p(s_{t+1}|s_t, a_t)$ and $G(\tau)$, with respect to θ, are zero. By differentiating (4.8), we obtain the *grad-log-prob of a trajectory*,

$$\begin{aligned}\nabla_\theta \log P(\tau|\theta) &= \nabla_\theta \log p_0(s_0) + \sum_{t=0}^{T-1}\left(\nabla_\theta \log p(s_{t+1}|s_t, a_t) + \nabla_\theta \log \pi_\theta(a_t|s_t)\right) \\ &= \sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t).\end{aligned} \tag{4.9}$$

Now, by making use of (4.3), (4.7) and (4.9), we may derive the expression for the policy gradient:

$$\begin{aligned}\nabla_\theta J(\pi\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau|\theta)G(\tau)d\tau \\ &= \int_\tau \nabla_\theta P(\tau|\theta)G(\tau)d\tau \\ &= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)G(\tau)d\tau \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta)G(\tau)] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)G(\tau)].\end{aligned} \tag{4.10}$$

Since (4.10) is an expectation, we may approximate its value by

$$\hat{g} = \frac{1}{|\boldsymbol{\tau}|}\sum_{\tau\in\boldsymbol{\tau}}\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)G(\tau), \tag{4.11}$$

where $\boldsymbol{\tau} = \tau_0, \tau_1, \cdots, \tau_{N-1}$, is the set of trajectories experienced by the agent while using the same policy π_θ.

The expression in (4.11) is the simplest version of the policy gradient. If we manage to represent the policy π_θ as a p.d.f., it is possible to estimate $\nabla_\theta \log \pi_\theta(a|s)$, using Monte-Carlo methods.

According to (4.10), the log-probabilities of all actions are scaled according to $G(\tau)$, which is the return with respect to all the rewards in trajectory τ. However, it is desirable that the actions log-probabilities are scaled according only to the actions consequences. This means we should compute the returns based on the rewards that come after the action is taken. Therefore, we will now express the return as follows:

$$G_t(\tau) = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}. \tag{4.12}$$

Applying (4.12), the expression for the policy gradient, which is called *reward-to-go* policy gradient [29], becomes

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t(\tau)\right]. \tag{4.13}$$

By removing the past rewards influence on the returns calculation, the number of trajectories needed for approximating the policy gradient is reduced, because the past rewards increases the amount of noise and variance added to the returns [29].

If we calculate (4.11) for only one trajectory at a time, over many episodes, we obtain the REINFORCE algorithm [11, 50], which is a Monte-Carlo policy optimization method. The method's procedural description is presented in Algorithm 3.

Algorithm 3: REINFORCE [50].

Initiate a differentiable policy parameterization $\pi(a|s,\theta)$
Initiate step size $\alpha > 0$
Initiate policy parameters θ
for *episode=1, M* **do**
  Generate a trajectory $\tau = s_0, a_0, r_1, \cdots, s_{T-1}, a_{T-1}, r_T, s_T$, following $\pi(a|s,\theta)$
  Set $\nabla_\theta J(\pi_\theta) = 0$
  for *step=1, T* **do**
    $G_t(\tau) = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$
    $\nabla_\theta J(\pi_\theta) = \nabla_\theta J(\pi_\theta) + G_t(\tau)\nabla_\theta \log(\pi_\theta(a_t|s_t))$
  end
  $\theta = \theta + \alpha \nabla_\theta J(\pi_\theta)$
end

4.4 BASELINES IN POLICY GRADIENTS

It is possible to expand the result in (4.13), to include a comparison of the return value to an arbitrary *baseline function* $b(s)$, which depends only on the state s_t, as follows [11, 29]:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(G_t(\tau) - b(s_t)\right)\right]. \tag{4.14}$$

A common choice of baseline function is the approximation of the *on-policy value function*, $V^\pi(s_t)$. This choice reduces the variance in the sample estimate for the policy gradient. It contributes for a faster and more stable policy learning [29].

$V^\pi(s_t)$ might be approximated by a NN, with parameters ψ by minimizing the following mean-squared-error objective function:

$$\mathbb{E}_{s_t, G(\tau)\sim\pi_k}\left[\left(V_\psi^\pi(s_t) - G(\tau)^2\right)\right], \tag{4.15}$$

where π_k is the policy at epoch k.

4.5 ACTOR-CRITIC ALGORITHMS

Methods that learn approximations to both policy and value functions are often called *actor-critic* methods. The actor refers to the learned policy and the critic refers to the learned value function [11].

Up until now, the policy gradients we have seen have taken upon the form

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right], \tag{4.16}$$

where Φ_t could have been

$$\begin{aligned} &\Phi_t = G(\tau), \\ &\Phi_t = G_t(\tau), \\ &\Phi_t = G_t(\tau) - b(s_t). \end{aligned} \tag{4.17}$$

Another two forms that Φ_t may assume are the approximation of the *on-policy action-value function*, $Q^\pi(s_t, a_t)$, and the so-called *advantage function*, $A^\pi(s_t, a_t)$, given by

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t). \tag{4.18}$$

The advantage function compares the performance of action a_t with the other actions, on average, for the current policy, for state s_t [29]. A popular algorithm that makes use of the advantage function is the Asynchronous Advantage Actor-Critic (A3C) algorithm [18].

A3C is an asynchronous algorithm, which means it executes multiple agents in parallel, on multiple instances of the environment, making the agent's trajectories decorrelated. The parallelism's effect is similar to what is achieved by the experience replay mechanism, in DQL. In terms of realization, A3C may be run in a computer's multiple processing threads.

Besides employing the concepts found on the REINFORCE algorithm, and the advantage function, A3C also employs an entropy-based technique for improving exploration and avoid premature convergence to suboptimal policies. From information theory, the entropy H of a random variable X, with possible outcomes $x_1, \cdots, x_n$, is given by

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i). \tag{4.19}$$

The technique consists of adding the entropy to the objective function [18, 51]. The gradient of the objective function at step t, including the entropy regularization term, becomes

$$\nabla_\theta \left(\log \pi_\theta(a_t|s_t) A(a_t, a_t) + \beta H(\pi_\theta(s_t)) \right). \tag{4.20}$$

The A3C procedural algorithm is given in Algorithm 4 [18].

The A3C algorithm inspired its synchronous version, Advantage Actor-Critic (A2C). In this algorithm, the updates are performed after all the actors have finished their experiencing process, and averaging over all the actors observations. In comparison to A3C, A2C is more effective on Graphics Processing Unit (GPU), and is faster than a Central Processing Unit (CPU) -based A3C implementation when using larger policies [52].

Algorithm 4: A3C - pseudocode for each actor-learner thread [18].

Assume global shared parameters vectors θ and ψ and global shared counter $T = 0$
Assume thread-specific parameter vectors θ' and ψ'
Initialize maximum step counter t_{max}
Initialize thread step counter $t = 1$
while *t* < *T* **do**
  Set $\nabla_\theta J_{actor}(\pi_\theta) = 0$
  Set $\nabla_\psi J_{critic}(V_\psi) = 0$
  Set $\theta' = \theta$
  Set $\psi' = \psi$
  Set $t_{start} = t$
  while s_t *is not terminal or* $t - t_{start} < t_{max}$ **do**
    Perform $a_t \sim \pi_{\theta'}(a_t|s_t)$
    Observe reward r_t and new state s_{t+1}
    $t = t + 1$
    $T = T + 1$
  end
  if s_t *is terminal* **then**
    $R = 0$
  end
  else
    $R = V_\psi(s_t)$
  end
  for $i \in \{t-1, \cdots, t_{start}\}$ **do**
    $R = r_i + \gamma R$
    $\nabla_\theta J_{actor}(\pi_\theta) = \nabla_\theta J_{actor}(\pi_\theta) + \nabla_{\theta'}(\log \pi_{\theta'}(a_i|s_i) A^{\pi_{\theta'}}(a_i, s_i) + \beta H(\pi_{\theta'}(s_i)))$
    $\nabla_\psi J_{critic}(V_\psi) = \nabla_\psi J_{critic}(V_\psi) + \nabla_{\psi'}(R - V_{\psi'}(s_i))$
  end
  Perform asynchronous update of actor's parameters θ with $\nabla_\theta J_{actor}(\pi_\theta)$
  Perform asynchronous update of critic's parameters ψ with $\nabla_\psi J_{critic}(V_\psi)$
end

4.6 GENERALIZED ADVANTAGE ESTIMATION

General Advantage Estimate (GAE) is a more sophisticated and promising method for estimating the advantage function. It significantly reduces the variance, while maintaining a tolerable level of bias on the policy gradient estimation [53].

Let us consider the TD residual of V with discount γ,

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{4.21}$$

where $0 \leq \gamma \leq 1$.

The sum of k of these δ terms is denoted by

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma \delta_{t+l}^V. \tag{4.22}$$

GAE(γ, λ) is defined as the exponentially-weighted average of these k-step estimators:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V, \tag{4.23}$$

where $0 \leq \lambda \leq 1$.

The λ parameter controls the tradeoff between bias and variance on the advantage estimator [53]. Although λ and γ affect the bias-variance compromise, they have different roles and function well with different values ranges. The γ parameter scales the V function, which is independent from the λ parameter. By taking $\gamma < 1$, bias is introduced into the policy gradient estimate, despite of the V function's accuracy. Taking $\lambda < 1$ will introduce bias to the policy gradient estimate only when the V function has a low accuracy. Therefore, it has been found that the best values for λ are much lower than the best values for γ, since the γ parameter is more likely to introduce bias to the estimator than the λ parameter, for a value function with considerable accuracy [53].

4.7 DEEP DETERMINISTIC POLICY GRADIENT

So far, the discussed DRL methods have been dealing only with discrete action spaces. However, interesting tasks, such as physical control tasks, have continuous real valued action spaces. A possible approach to continuous action spaces would be the discretization of the action space. This method is limited by the curse of dimensionality: the number of degrees of freedom causes the number of actions to increase exponentially [54]. Furthermore, the

discretization method has great impact on the algorithm's performance.

Deep Deterministic Policy Gradient (DDPG) is a DRL algorithm designed to deal with problems that demand high dimensional continuous action spaces. It combines the actor-critic concept with the off-policy learning applied by DQL. DDPG is model-free, easy to implement and scale to difficult problems and larger networks [54].

Up until now, when discussing policy gradient methods, only stochastic policies have been considered. However, it is also possible to work with deterministic policies, as mentioned in Equations 2.3 and 2.4, respectively. DDPG is based on Deterministic Policy Gradient (DPG) [55].

The DDPG algorithm has an actor, represented by the deterministic policy function $\mu(s, \theta)$, where θ is the policy network's parameters set, and a critic, which learns the action value function $Q(s, a)$ using the Bellman equation (2.13),

$$Q^*(s, a) = \mathbb{E}_{s' \sim P}\left[r(s, a) + \gamma \max_{a'} Q^*(s', a')\right]. \tag{4.24}$$

For the critic, similar to DQL, the loss function is the Bellman equation error (4.1). Therefore, DDPG is an off-policy algorithm, since the Bellman equation does not depend on a particular policy. This fact allows DDPG to also use an experience replay buffer, in order to learn from decorrelated samples.

In order to improve stability and convergence, the algorithm also makes use of target networks, Q^- and μ^-, for both critic and actor, respectively, for calculating the target values. However, the target networks updates are done differently from DQL. They are performed once per main networks update, by applying soft updates:

$$\begin{aligned} \theta^- &\leftarrow (1 - \xi)\theta^- + \xi\theta \\ \psi^- &\leftarrow (1 - \xi)\psi^- + \xi\psi \end{aligned} \tag{4.25}$$

where $0 < \xi \leq 1$, ψ is the actor network's parameters and ψ^- is the target network parameters. ξ is usually close to zero.

Since it is desired to find a policy $\mu_\theta(s)$ that maximizes the action-value function $Q_\psi(s, a)$, assuming Q is differentiable with respect to action, the actor's objective function becomes

$$\nabla J_\theta(\mu_\theta) = \mathbb{E}\left[Q_\psi(s, \mu_\theta(s))\right], \tag{4.26}$$

which is proved to be the off-policy deterministic policy gradient [29, 54, 55]. Gradient ascent is applied to maximize (4.26).

The exploration is performed by adding stochastic noise to the actor network's outputs, e.g., Ornstein-Uhlenbeck [54], Gaussian Noise [29]. Therefore, in exploration, the actions is

given by

$$a(s) = \mu_\theta(s) + \mathcal{N}, \tag{4.27}$$

where $\mathcal{N}$ is the stochastic noise. The noise may be chosen to suit the environment.

Algorithm 5: Vanilla DDPG [54].

Randomly initialize critic Q_ψ and actor μ_θ networks, with weights ψ and θ, respectively
Initialize target networks Q^- and μ^- with weights $\theta^- \leftarrow \theta, \mu^- \leftarrow \mu$
Initialize replay buffer $\mathcal{D}$
for *episode=1,M* **do**
  Receive initial observation state s_1
  for *t=1,T* **do**
    Select action $a_t = \mu_\theta(s_t) + \mathcal{N}$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$
    Set target $y_i = r_i + \gamma Q^-(s_{i+1}, \mu^-(s_{i+1}))$
    Update the critic by minimizing the loss $L = \frac{1}{N}\sum_i (y_i - Q_\psi(s_i, a_i))^2$
    Update the actor policy using the sampled policy gradient

$$\nabla J_\theta(\mu_\theta) = \frac{1}{N}\sum_i \nabla_\theta Q_\psi(s_i, \mu_\theta(s_i))$$

    Update target networks with (4.25)

$$\theta^- \leftarrow (1-\xi)\theta^- + \xi\theta$$
$$\psi^- \leftarrow (1-\xi)\psi^- + \xi\psi$$

  end
end

4.8 PARAMETER SPACE NOISE FOR EXPLORATION

In order to increase exploration, besides employing noise on the action space, it is also possible to input noise on the actor's parameters space, which corresponds to the policy NN's parameters [56, 57]. This is formulated as

$$\tilde{\theta} = \theta + \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}), \tag{4.28}$$

where $\tilde{\theta}$ represents the perturbed parameters, $\mathcal{N}(\mathbf{0}, \sigma^2\boldsymbol{I})$ is a stochastic noise with zero mean and σ^2 variance.

Different layers within a NN may have different sensitivities to the added noise. Addi-

tionally, choosing the value for the scaling factor σ is not trivial. In order to solve both of these problems, [57] proposes Adaptive Scaling.

It is not possible to perfectly understand the effects the noise addition may cause on a neural network. Furthermore, as training progresses, the NN moves from a random initialization around zero towards a complex non-linear mapping from input space to output space. Therefore, it is easier to approach this problem by moving it to the action space.

This trick makes it possible to measure the perturbation effect in the more comprehensive action space. By using this approach, we may adapt σ according to:

$$\sigma_{k+1} = \begin{cases} \alpha\sigma_k & \text{if } d(\mu(s), \tilde{\mu}(s)) \leq \delta, \\ \frac{1}{\alpha}\sigma_k & \text{otherwise,} \end{cases} \tag{4.29}$$

where $d(\mu(s), \tilde{\mu}(s))$ defines a distance measure between $\mu(s)$ and $\tilde{\mu}(s)$, $\delta \in \mathbb{R}^+$ is a distance threshold value, and $\alpha \in \mathbb{R}^+$ defines the change rate on the current scaling value σ_k. In other words, (4.29) means that σ_k increases if the distance between policy and perturbed policy is smaller than δ, and it decreases otherwise.

There is no practical way to compute the distance between two policies[57]. Therefore, the estimated expected distance is used, by randomly sampling states mini-batches $s_1, \cdots, s_M$:

$$\mathbb{E}\left[d(\mu, \tilde{\mu})\right] \approx \frac{1}{M} \sum_{i=1}^{M} d\left(\mu(s_i), \tilde{\mu}(s_i)\right) \tag{4.30}$$

At last, $d(\cdot, \cdot)$ must still be defined. There are multiple choices for the distance measurement, and it depends on the DRL algorithm, as well as the properties one wishes to achieve with it. It ends up being a design choice.

For DDPG, the proposed distance measurement is [57]:

$$d(\mu, \tilde{\mu}) = \sqrt{\frac{1}{|\mathcal{A}|} \sum_{i=1}^{|\mathcal{A}|} \mathbb{E}\left[(\mu(s)_i - \tilde{\mu}(s)_i)^2\right]}, \tag{4.31}$$

where $\mathbb{E}[\cdot]$ is estimated using a mini-batch of states sampled from the replay buffer $\mathcal{D}$, $|\mathcal{A}|$ is the number of continuous actions, and $\mu(s)_i$ denotes the $i-th$ action selected by μ.

Algorithm 6: Parameter Space Noise Exploration with Adaptive Scaling [57]

Define DRL algorithm $\mathbb{A}$, e.g., DDPG
Initiate $\sigma_0, \delta, \alpha \in \mathbb{R}^+$
Define distance measure $d(\cdot, \cdot)$
Initiate intervals $T_{train}, T_{adapt} \in \mathbb{R}^+$
Initiate $\mathbb{A}$ and $\mu = \mu_\theta$
Initiate $\tilde{\mu}$ for exploration
Initiate $\bar{\mu}$ for noise adaptation
for *episode*=$1, \cdots, M$ **do**
  Perturb $\tilde{\theta} \leftarrow \theta + \mathcal{N}(0, \sigma_k^2)$ and obtain $\tilde{\mu}$
  for $t = 1, \cdots, T$ **do**
    Sample an action $a_t = \tilde{\mu}(s_t)$
    Execute action a_t and observe the new state s_{t+1}
    if $t \mod T_{train} = 0$ **then**
      Execute training step of $\mathbb{A}$
    end
    if $t \mod T_{adapt} = 0$ **then**
      Perturb $\bar{\theta} \leftarrow \theta + \mathcal{N}(0, \sigma_k^2)$ and obtain $\bar{\mu}$
      Estimate $d_k \leftarrow \mathbb{E}[d(\mu(s), \tilde{\mu}(s)]$
      if $d_k \leq \delta$ **then**
        $\sigma_{k+1} \leftarrow \alpha \sigma_k$
      end
      else
        $\sigma_{k+1} \leftarrow \frac{1}{\alpha} \sigma_k$
      end
      $k \leftarrow k + 1$
    end
  end
end

4.9 CONCLUSION

In this chapter, we covered some key DRL algorithms. We presented DQL, which leverages the computing power of neural networks in order to improve upon the traditional Q-Learning algorithm. DQL is able to deal with high-dimensional continuous state spaces with simple and direct implementations, which is a great advantage on previous classic RL methods.

Next, we explore policy optimization algorithms, another paradigm in reinforcement learning, different from the value optimization techniques that have been presented so far, which leads us to the REINFORCE algorithm. On the sequence, the actor-critic algorithms are presented, which are a family of methods that makes use of both value- and policy-optimization concepts. We talk about GAE, which is a method for estimating the advantage, a scalar quantity that is key on the A3C/A2C algorithms.

Policy optimization algorithms have a stronger analytical convergence, when compared to value-optimization algorithms. Both paradigms have their own pros and cons, and comparisons between them are still debated by the community. That motivates this work to explore both of them, in the upcoming chapters.

Both DQL and the presented actor-critic algorithms are suited for discrete action-spaces. DDPG moves into the realm of continuous action spaces. By employing concepts from the previous methods, DDPG is able to deal with high-dimension continuous action space problems.

At last, a new exploration paradigm is presented. So far, the exploration concerned only the action space, mainly by introducing stochastic elements to it. Parameter space noise comes into play as a new tool for improving exploration for DDPG.

The DRL concepts in this chapter constitute the foundation for the power allocation frameworks that will be presented in the next chapter.

5 SYSTEM MODEL AND PROBLEM FORMULATION

In this chapter, we explain D2D communication and its concepts, and go over different use cases. Afterwards, we present our system model and the optimization problem we wish to solve, concerning the inband-underlay use case. Next, the DRL-based optimization frameworks are presented in-depth. We explain all the algorithms, their implementations, and designs.

5.1 D2D COMMUNICATION

D2D communication allows close devices to communicate directly among themselves, often with high quality, due to the proximity between them [9]. Figure 5.1 presents an example of D2D communication.

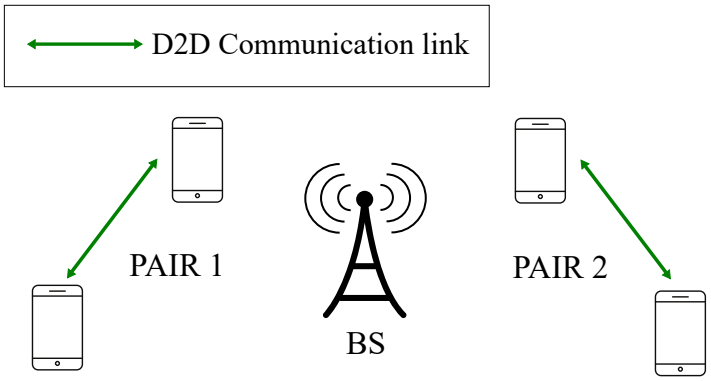

Figure 5.1 – Two device pairs performing D2D communication.

In the last decade, D2D communication has been available through technologies that use the unlicensed spectrum, e.g., Bluetooth, RFID, Zigbee, and others [8]. However, in Releases 12 [58] and 13 [6], the 3GPP introduced ProSe, a wide range of applications that will be using licensed spectrum, and provided via D2D communication.

ProSe is thought to enable applications for commercial/social use, network offloading, public safety and consistency of user reachability and mobility [58]. Release 13 included Vehicle-to-Everything (V2X) communication under the umbrella of ProSe [6].

5.1.1 D2D classifications

D2D is classified into two categories: *inband* and *outband* D2D [10, 8].

- **Inband**: refers to D2D communication under licensed spectrum. This category is further divided into two categories: *overlay* and *underlay* [10, 8].
  - **Underlay**: D2D communication with shared spectrum resources. This means the devices performing D2D communication will be sharing the resource with a Mobile User Equipment (MUE), which is the cellular user. This D2D mode may provide enhanced spectral efficiency, due to the resources re-use. It brings challenges on interference management and resource allocation. Interference management methods may be applied, but at the expense of additional overhead at the base station [8, 10].
  - **Overlay**: D2D communication with reserved spectrum resources. This scheme presents the advantages of reduced interference, since the D2D devices now have their own separate spectral bands to communicate. It also allows better scheduling and power control. However, the portion of the spectrum allocated for D2D communication may be inefficiently used, leading to poor resource utilization and system throughput [8, 10].
- **Outband**: D2D communication in unlicensed spectrum. This type of communication requires device compatibility. Since the communication happens in unlicensed spectrum, the interference problem becomes more complex [10]. Additionally, coordinating the communication over different bands is also challenging [8]. Outband communication may be divided into two categories: *controlled* and *autonomous* communication.
  - **Controlled**: the interface between radio interfaces, e.g, Bluetooth, ZigBee, Wi-Fi Direct, is controlled by the network. The BS may prioritize particular transmissions, in order to enhance QoS. Consequently, this mode increases the system's spectral efficiency, at the expense of a larger signaling overhead [8].
  - **Autonomous**: the D2D communication is controlled by the devices involved in it. This approach lessens the workload of the network and does not need major BS changes during deployment. The resource allocation is performed by the D2D devices, reducing the signaling overhead [8].

5.1.2 Challenges in D2D communication

The main challenges in D2D communications are *network discovery*, *network security*, *interference management*, *mobility* and *mode selection* [8, 10].

5.1.2.1 Network discovery

Network discovery refers to devices discovering other devices that are able to establish direct connection [8]. The devices share their location/distance, channel state, device ID, and other informations, with each other. This information is used to determine the feasibility of the communication [8].

The discovery process may be performed with the assistance of a centralized network entity, such as the BS. The devices inform the network about their intent to communicate, and the BS initiates the exchange of essential information to begin the communication [8]. This type of discovery process has been called *centralized* [8] or *network centric* discovery [10].

There is also *distributed* [8], or *device centric* [10], discovery. In this process, the devices locate each other without the involvement of the BS. They send control signals in order to locate other devices. This scheme presents difficulties of synchronization and interference. We can find studies, comparing discovery schemes, in [59, 60, 61].

5.1.2.2 Network security

Network security refers to the techniques employed to protect the communication from attacks. D2D communication must deal with threats, faced both by both cellular and ad-hoc wireless networks, that affect authentication, confidentiality, integrity and availability. Therefore, D2D communications must be safe when the devices exchange information with the cellular network, and must offer security, in direct proximity communications, that is independent from the cellular network [8, 10, 62]. We can find studies about D2D communications security in [63, 64].

5.1.2.3 Interference management

Interference is one of the major impairments affecting D2D communication. It can compromise the devices Signal-to-Noise-Ratio (SINR)a, along with the quality of the transmissions [10].

In outband communication, the D2D links suffer interference from each other, as well from other devices, using the same unlicensed spectrum, that are not part of the cellular network [62].

In inband communication, the D2D links suffer interference from each other, as well as the MUE [62].

In order to deal with these problems, interference-aware resource management is often

proposed. The resource management is usually treated as an optimization problem, with transmit power and QoS constraints. Power allocation, transmission scheduling and modulation and coding schemes are all part of resource management, and may be optimized in order to mitigate interference [62]. The reader may find interference management studies in [65].

5.1.2.4 Mobility

Most of the research related to D2D communication has focused on static users. Therefore, it is needed for more research concerning dynamic situations, with mobile devices [62].

When mobility is concerned, topics such as interference handling, handover and multihop communication become important investigation subjects [62]. Evaluating users mobility patterns and the impact caused by them on communication reliability is also a key challenge [10]. The reader may find more about mobility in D2D communication in [66, 67, 68].

5.1.2.5 Mode selection

Mode selection refers to the choice the devices have between using the cellular network or communicating directly, using D2D communication [62]. It is also extended for choosing between outband, inband-underlay and inband-overlay modes [8].

The decision on mode selection may be motivated by objectives such as low latency, low transmit power, high QoS, and high spectral efficiency [62].

Mode selection studies are found in [8, 69, 70].

In this work, we focus on power allocation schemes for enhancing spectral efficiency and mitigating interference, in the inband-underlay scenario. These schemes will be presented on the following sections.

5.2 SYSTEM MODEL

In our model, we consider a single cell, in which D2D communication users and MUEs coexist, sharing the same resource (inband-underlay mode). The users are pedestrians, that may walk or stay still. The D2D devices are grouped in transmitter-receiver pairs. We denote the set of MUEs by $\boldsymbol{M} = \{1, \cdots, M\}$ and the set of D2D pairs by $\boldsymbol{N} = \{1, \cdots, N\}$. All the devices are distributed randomly inside the BS coverage area. The coverage area is assumed to be circular.

We study D2D communication over the uplink transmission, where MUEs and D2D users

share the same Resource Block (RB). This means the D2D pairs transmit at the same time the MUE is transmitting to the BS. The resource is allocated to the MUE, and, consequently, there is only one MUE per RB. The set of available RBs is denoted by $\boldsymbol{K} = \{1, \cdots, K\}$. In this situation, there are two types of interference we must deal with. Interference 1 is the interference suffered by the BS, caused by the D2D transmitters, impacting the MUE's QoS. Interference 2 is the interference on the D2D receivers. It comes from the MUE, as well as the other D2D transmitters. It impacts the D2D pairs' QoS. Figure 5.2 illustrates the studied scenario.

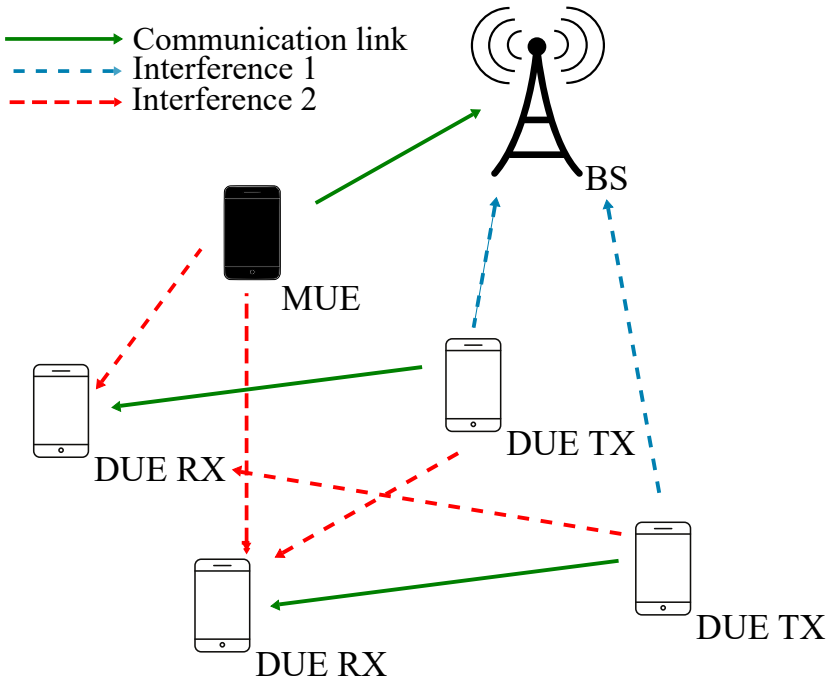

Figure 5.2 – D2D inband-underlay communication scenario.

We assume different RBs are accessed by orthogonal signals, allowing us to treat each RB independently, similarly to what is done in [19].

We measure the devices SINR, in order to calculate the system spectral efficiency and QoS. The SINR obtained by the i-th D2D user, on the k-th RB, is given by

$$\gamma_k^{d_i} = \frac{p_k^{d_i} \cdot g_k^{d_{ii}}}{\sigma^2 + p_k^m \cdot g_k^{mi} + \sum\limits_{j \in \boldsymbol{R}_k}^{j \neq i} p_k^{d_j} \cdot g_k^{d_{ji}}}, i = 1, 2, \cdots, N \tag{5.1}$$

where $p_k^{d_i}$ and p_k^m denote the i-th D2D pair's transmitter transmission power and the MUE uplink transmission power, respectively, both sharing the k-th RB. Both transmission powers, per RB, are superiorly bounded by p_{max}, which means $p_k^{d_i}, p_k^m \leq p_{max}, \forall i, m \in \boldsymbol{N}, \boldsymbol{M}$. $\boldsymbol{R}_k$ is the set of D2D pairs sharing the $k-$th RB. The channel gain between the i-th D2D pair devices, on the k-th RB, is given by $g_k^{d_{ii}}$. The channel gain between the MUE and the i-th

D2D pair receiver, on the k-th RB, is given by g_k^{mi}. At last, the channel gain between the j-th D2D pair transmitter and the i-th D2D pair receiver, on the k-th RB, is denoted by $g_k^{d_{ji}}$. The noise power is depicted as σ^2. We also need to know the MUE SINR, on the k-th RB, which is

$$\gamma_k^m = \frac{p_k^m \cdot g_k^{m0}}{\sigma^2 + \sum\limits_{j \in \boldsymbol{R}_k} p_k^{d_j} \cdot g_k^{j0}} \tag{5.2}$$

Following the established convention, the channel gains between the MUE and the BS, and between the j-th D2D transmitter and the BS, both on the k-th RB, are given by g_k^{m0} and g_k^{j0}, respectively.

5.3 PROBLEM FORMULATION

It is desired to maximize the D2D pairs' spectral efficiencies while maintaining the MUE QoS at a minimum desired level. For the sake of simplicity, RB allocation is given and fixed. Therefore, the optimization problem can be written as:

$$\begin{aligned} &\max_{\boldsymbol{p}} \sum_{k=1}^{K} \left\{ \log_2\left(1+\gamma_k^m\right) + \sum_{i \in \boldsymbol{R}_k} \log_2\left(1+\gamma_k^{d_i}\right) \right\} \\ &\gamma_k^m \geq \tau_0 \\ &0 \leq p_k^{d_i} \leq p_{max}, \forall i, k \end{aligned} \tag{5.3}$$

where $\boldsymbol{p} = (p_k^{d_1}, \cdots, p_k^{d_i}, \cdots, p_k^{d_N}), \forall k, i \in \boldsymbol{K}, \boldsymbol{N}$, and τ_0 is the minimum MUE SINR level requirement. This means that, in order for the communication process to be considered successful, the MUE SINR must satisfy $\gamma_k^m \geq \tau_0$.

The optimization problem in (5.3) translates to finding the set of transmission powers, for each D2D pair on the k-th RB, which will maximize the capacity for the MUE and the D2D pairs, while satisfying two restrictions. The power allocator is supposed to solve the optimization problem at the beginning of every time slot. The restrictions are the minimum MUE SINR, τ_0, and the maximum transmission power level, p_{max}.

The MUE is assumed to have perfect channel knowledge and to always transmit with the necessary amount of power, limited to p_{max}, in order to obtain a SINR level of $\tau_0 + \tau_1$, where τ_1 is a safety margin. It has been show that the problem in (5.3) is NP-hard and, in general, non-convex [71, 72]. In order to solve this problem, we resort to DRL-based frameworks.

The transmission delay on D2D- and device-to-BS- communications is considered to be much smaller than the time slot duration. Hence, we neglect prediction error due to this delay and assume that, once the nodes have completed channel measurements, this information will

also be available to the other nodes in the network.

At the beginning of time slot t, the D2D transmitter i_0 learns the channel gains to its receiver i_1, $g_k^{d_{ii}}$. The BS informs the channel gain between itself and i_1 and it also gives information on the MUE: the channel gain g_k^{m0} between MUE and itself, the MUE current SINR γ_k^m, and the MUE position coordinates c_m.

Transmitter i_0 also exchanges information with the other D2D transmitters $j_0 \in \boldsymbol{N}, j \neq i$. It updates the other devices on the channel gain g_k^{i0} between itself and the BS, along with its own SINR $\gamma_k^{d_i}$ and the position coordinates c_i for itself and its receiver. The other D2D transmitters j_0 also provide the equivalent information, $\gamma_k^{d_j}, g_k^{j0}, c_j$, about themselves. All the channel gains are considered to be perfect estimations. Figure 5.3 illustrates the information exchange between devices.

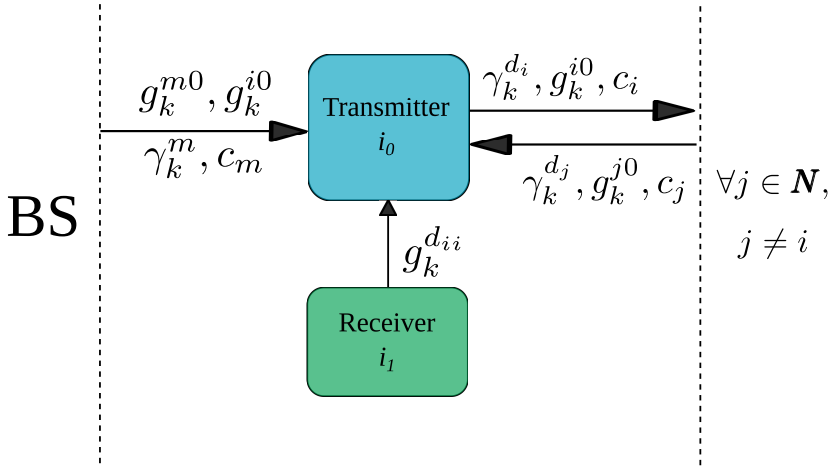

Figure 5.3 – Information exchange between devices at the beginning of time slot t.

5.4 DRL-BASED POWER ALLOCATION SCHEMES FOR D2D COMMUNICATION

The problem in 5.3 is simplified by decomposing it into K-parallel sub-optimal problems, each problem being solved for each RB. The RBs are considered to be independent from each other, which means there is no interference between them. We consider that, at the power allocation stage, the RBs allocation is already defined. In order to determine the power allocation for all RBs, we must run the DRL power allocation schemes for each RB.

Inspired by [19, 57, 72], we propose two different DRL schemes for D2D power allocation: *centralized learning-centralized execution* and *centralized learning-distributed execution*, for solving the problem in (5.3). The schemes modeling, along with the designed MDPs, illustrations, network architectures and pseudocode algorithms are presented in the following sections.

5.4.1 Centralized Learning-Distributed Execution

This scheme was thought for the discrete action spaces algorithms, i.e., DQL and A2C. These algorithms suffer the curse of dimensionality, i.e., the number of discrete actions increases exponentially with the number of degrees of freedom. If a centralized agent was to control N D2D transmitters, the actions space would have a size of $|\boldsymbol{A}| = L^N$, where L is the number of transmit power levels. Besides requiring ever larger NNs to represent such problem, exploring such a large action space becomes impractical.

In order to mitigate this problem, this scheme was developed. This concept has already been tested in [73]. Centralized-Learning Distributed-Execution (CLDE) is a multi-agent DRL scheme where, as the name states, the learning/training process is centralized, while execution is performed in a distributed way. It is similar to what is presented in [72]. The agents are the D2D transmitters. By having one agent per D2D transmitter, they action space size now becomes $|\boldsymbol{A}| = L$, which is a practical size.

In multi-agent RL, the state transitions depend on all agents' joint actions. Multi-agent learning still needs more research and improvements on theoretical guarantees. There is an extensive effort on further developing this theory [74, 75, 76, 77].

In this approach, the agents are assumed to be *partially homogeneous*. According to [77], homogeneous agents have common reward functions, that aligns all agents' interests. Additionally, in the case of large populations, the agents play an interchangeable role, and can hardly be distinguished from each other. In our case, we call the agents partially homogeneous, because they share only the same penalty. The positive rewards are given individually, according to each agents' spectral efficiency, as we will see up ahead. However, the agents are similar, with the same set of states and actions, and indistinguishable from each other, i.e., if their places are switched, they should act in similar, if not in equal, manner. With this in mind, we conclude all agents may benefit from each others' accumulated experience, and train them all based on the same set of joint accumulated experiences.

In order to leverage the agents' homogeneity, a centralized learning scheme is proposed. As illustrated by Figure 5.4, the agents interact with the environment and store their experiences in a *common experience storage*. During training, the central NNs receive a batch of experiences, from the common storage, which they are trained upon. Notice that this batch contains experiences from all agents. After training, identical copies from the trained NN are transmitted to the agents. Despite having copies from the same NN, the agents still take different actions, because their local states are different. The centralized training, as well as the storage of the central NNs, may be executed at a network entity with enough computational power, such as the BS or even at the cloud [78].

Similar to [72], we define the state of agent i, on time slot t, as $s_t^i \in \boldsymbol{S}$, which is composed of the information agent i has. Agent i's action on time slot t is defined as $a_t^i \in \boldsymbol{A}$.

When using DQL, the *common experience storage* is a central experience replay memory, where all agents store their experiences. The experiences are the tuples $(s_{t-1}^i, a_{t-1}^i, r_t^i, s_t^i) \forall i \in \boldsymbol{N}$. These experiences are used for training the central NNs.

When using A2C, the *experience*, denoted in Figure 5.4, becomes the actions log-probabilities along with the estimated value functions, which are estimated by the critic, the obtained distributions entropies and rewards.

The *common experience storage* is a simple buffer, where the all agents' experiences are stored. Following every episode, the experiences are used for training, and discarded after the policy update, due to the on-policy nature of A2C.

The A2C-based scheme is fully detailed in Algorithm 8.

5.4.1.1 States

The CLDE solutions build their states based on the devices information exchange depicted in Figure 5.3. The states are composed by:

- **Devices coordinates**: All D2D devices and MUE 2-D position coordinates at the moment the transmission happens. Given the number of D2D devices N, this information demands $4 * N + 2$ input nodes on the NNs.
- **Devices SINRs**: All D2D devices and MUE SINR levels in the previous transmission, which is the previous time slot. This information requires $N + 1$ input nodes.
- **Channel gains**: The channel gain for: the channel between D2D transmitter and receiver; all the channels from D2D transmitters to the BS; the channel between MUE and BS. These are perfect estimation of the channel gains at the moment the transmission happens. This information requires $2 + N$ network input nodes.

The total required network input nodes is $4N + 2 + N + 1 + 2 + N = 6N + 5$.

5.4.1.2 Actions

For DQL and A2C, the actions set $\boldsymbol{A}$ is composed by discrete power levels. Considering $|\boldsymbol{A}| > 1$, the set $\boldsymbol{A}$ is given by

$$\boldsymbol{A} = \left\{P_1, P_2, \cdots, P_{|\boldsymbol{A}|}\right\}, \tag{5.4}$$

where P_i is a discrete power level. It is important to notice the choice of discrete power levels has great impact on the algorithms performance.

5.4.1.3 Reward

The proposed reward function, for the CLDE algorithms, is

$$R_k^i = \begin{cases} \alpha_1([\gamma_k^m]_{dB} - [\tau_0]_{dB}), & \text{if } \gamma_k^m < \tau_0 \\ \gamma_k^{d_i}, & \text{if } \gamma_k^m \geq \tau_0 \text{ and } \gamma_k^{d_i} \leq 10 \\ 10 + \alpha_2[\gamma_k^{d_i}]_{dB}, & \text{if } \gamma_k^m \geq \tau_0 \text{ and } \gamma_k^{d_i} > 10, \end{cases} \tag{5.5}$$

where τ_0 is the minimum MUE SINR threshold, $[\cdot]_{dB} = 10\log_{10}(\cdot)$, and $\alpha_1, \alpha_2 \in \mathbb{R}^+$ are arbitrary constants for changing the function slopes.

The first term in (5.5) is the penalty for violating the MUE QoS requirement, translated as a SINR threshold. Note that this penalty depends solely on the MUE SINR. Therefore, when the requirement is not fulfilled, all agents receive the same penalty. This is for encouraging the agents to cooperate in order to deliver a good QoS to the primary user.

The other reward terms are valid when $\gamma_k^m > \tau_0$. They are the actual bonuses from the reward function and they are given individually. In this way, some competition among the agents is promoted for when the MUE QoS is already guaranteed.

Since a SINR follows $\gamma \geq 0$, the second term of the reward connects the other two terms, keeping the function's continuity. The log function is applied in order to keep the reward function absolute values from becoming very large. Overall, (5.5) was designed to be continuous, monotonically non-decreasing and to not have flat/constant spots. Constant value reward functions, such as the one seen in [19], performed poorly in this work first attempts.

5.4.1.4 Neural Networks

For DQL, the central NNs are the main and the target networks. The i-th agents' NNs are copies from the central main network. The target NN has the same architecture as the main NN, it serves as reference for the main NN, and it is updated with the main NN's weights θ every E time steps. After training, only the main NN remains, for making the new decisions, while the target NN is no longer used. The training is performed online, i.e., the policy is updated after every time step.

Both the main and target NNs are MLPs. Figure 5.6 illustrates such architecture. The input layer length, for this network, is $|\boldsymbol{x}| = 6N + 5$. The output layer has a length of $|\boldsymbol{y}| = |\boldsymbol{A}|$, where $\boldsymbol{y}$ is an array representing the NN's output. The NNs output $\boldsymbol{y}$ is the approximation of the state-value function, $Q(s, a)$. The centralized training-distributed execution scheme's DQL version is fully detailed in Algorithm 7.

For A2C, the central NNs are the central actor and critic networks. The i-th agent's NNs are copies of the central actor and critic networks. The actor outputs the agent's action, and the critic outputs the value function, which serves as a scaling factor for the calculated returns, during training. The training is not online. This means the networks are updated at the end of episodes, based only on the experiences acquired during the episode.

Both networks are MLPs. The actor follows the same architecture as the DQL networks. The critic differs only in its output layer, that has a length of $|\boldsymbol{y}| = 1$, which means only one output node. The critic returns a continuous real value, which is an approximation of the state value function, $V(s)$.

5.4.2 Centralized Learning-Centralized Execution

Centralized-Learning Distributed-Execution (CLCE) is a fully centralized, mono-agent scheme. A single central agent controls the transmit powers of all D2D transmitters. DDPG is able to handle such scheme. Since it is designed for dealing with high-dimensional continuous actions spaces, DDPG does not present the curse of dimensionality problem. Thanks to its continuous outputs, a central DDPG agent, controling N D2D transmitters, would have an actions space of size $\boldsymbol{A} = N$, which is a practical value.

Figure 5.5 illustrates the CLCE scheme. The *Central Experience Storage* is the experience replay memory, where transitions $(s_{t-1}, a_{t-1}, r_t, s_t)$ are stored. The *Central NNs* are the actor and critic networks.

5.4.2.1 States

The CLCE scheme states are quite similar to the CLDE states. However, they are all passed to a single agent. They are composed by:

- **Devices coordinates**: All D2D devices and MUE 2-D position coordinates, at the moment of the transmission. Given the number of D2D devices N, this information demands $4N + 2$ input nodes on the NNs.
- **Devices SINRs**: All D2D devices MUE SINR levels in the past transmission. This information requires $N + 1$ input nodes.
- **Channel gains**: The channel gain for: the channel between D2D transmitters and receivers; all the channels from D2D transmitters to the BS; the channel between MUE and BS. These are perfect estimates of the channel gains at the moment of the transmission. This information requires $2N + 1$ network input nodes.

The total required input nodes is $4N + 2 + N + 1 + 2N + 1 = 7N + 4$.

5.4.2.2 Actions

Given N agents, the actions space will be given by

$$\boldsymbol{A} = P_1, P_2, \cdots, P_N, \tag{5.6}$$

where P_i is the $i-th$ D2D transmitter continuous real transmit power, and $|\boldsymbol{A}| = N$.

5.4.2.3 Reward

The proposed reward, for CLCE, is actually quite simpler than (5.5):

$$R_k^i = \alpha_1 \min([\tau_0]_{dB}, [\gamma_k^m]_{dB}) + \alpha_2 \frac{1}{N} \sum_{i=1}^{N} [\gamma_k^{d_i}]_{dB}, \tag{5.7}$$

where $\min(\cdot, \cdot)$ is the minimum function,

$$\min(a, b) = \begin{cases} a, & \text{if } a \leq b \\ b, & \text{otherwise.} \end{cases} \tag{5.8}$$

The first term is for encouraging the algorithm to provide only the required MUE SINR level, while also serving as penalty for when the MUE SINR is low. The second term is the average of the D2D transmitter SINRs and it encourages the agent to increase the transmitters' power levels. The log operations maps SINRs that satisfy $0 \leq \gamma \leq 1$ to negative values, serving as penalty, while also keeping high SINR values into a smaller range of values.

The reward in (5.7) was chosen over (5.5), for CLCE, because it provided better performances for this scheme.

5.4.2.4 Neural Networks

For CLCE DDPG, the actor NN is a MLP, as illustrated by Figure 5.6. Its input size is $|\boldsymbol{x}| = 7N + 4$ and its output size $|\boldsymbol{y}| = N$. The critic network is also a MLP, but with a slight difference. This NN inputs are the state s_t and the action a_t. However, a_t is input directly into the NN's first hidden layer [54]. The input layer size is $|\boldsymbol{x}| = 7N + 4$. The first hidden layer size is $N_1 + |\boldsymbol{A}|$, as depicted by Figure 5.7. The output size is $|\boldsymbol{y}| = 1$

5.5 SCHEMES ILLUSTRATIONS

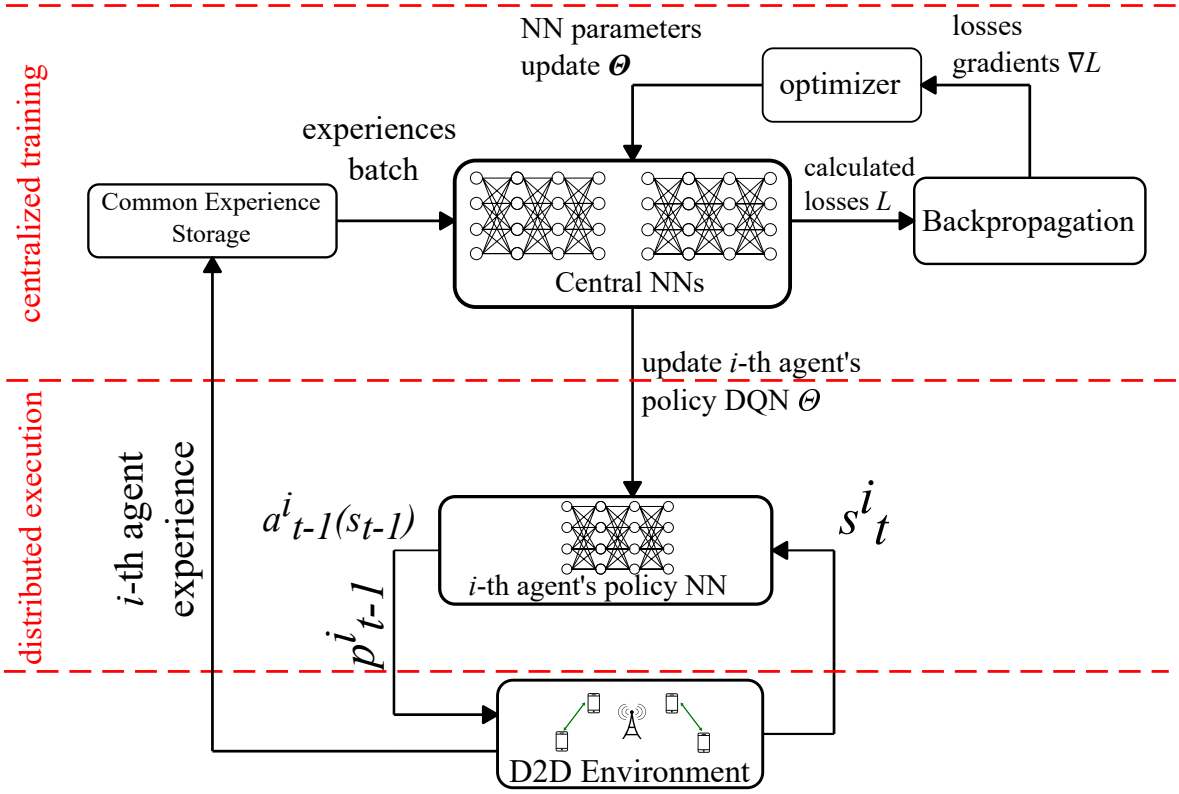

Figure 5.4 – Centralized learning-distributed execution scheme illustration.

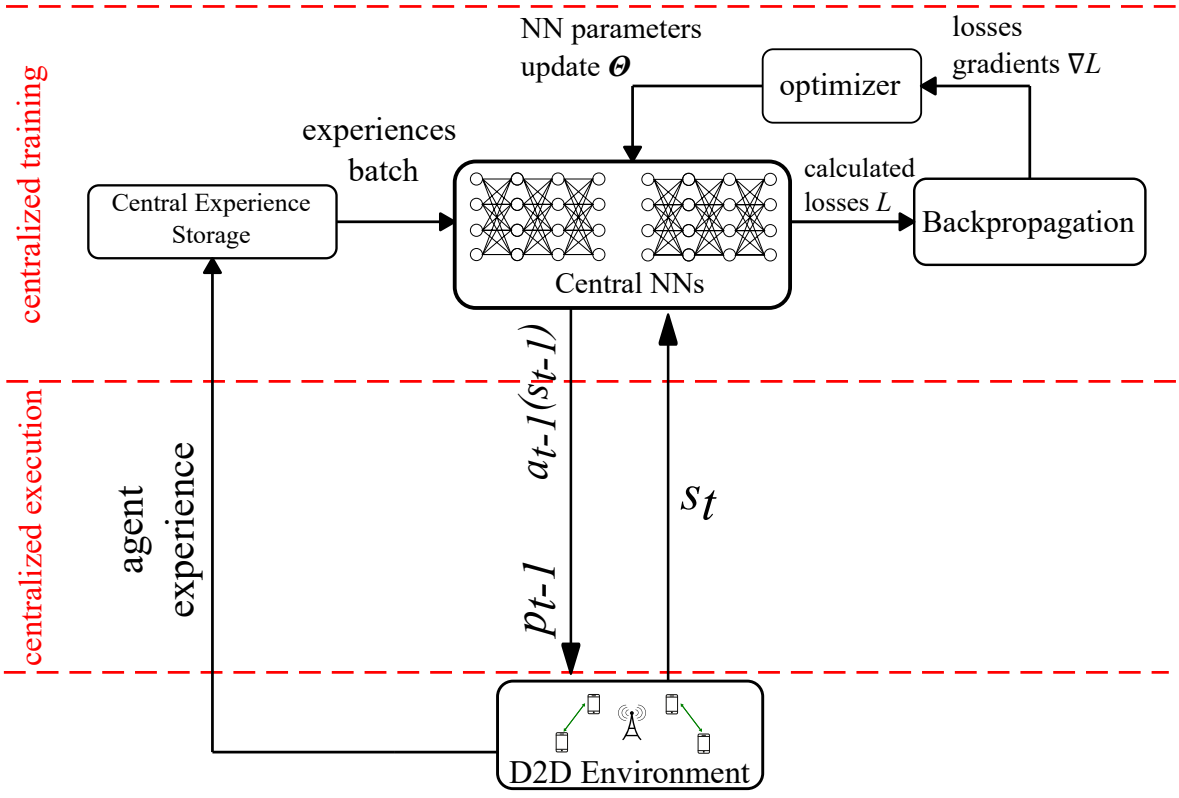

Figure 5.5 – Centralized learning-centralized execution scheme illustration.

5.6 NETWORK ARCHITECTURE ILLUSTRATIONS

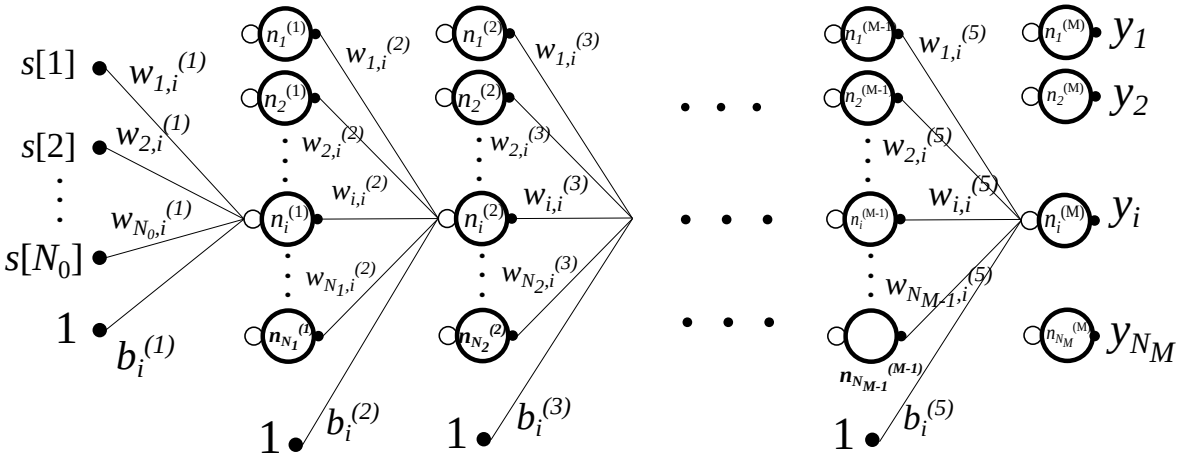

Figure 5.6 – Illustration of an MLP with M layers, and a variable amount of nodes per layer.

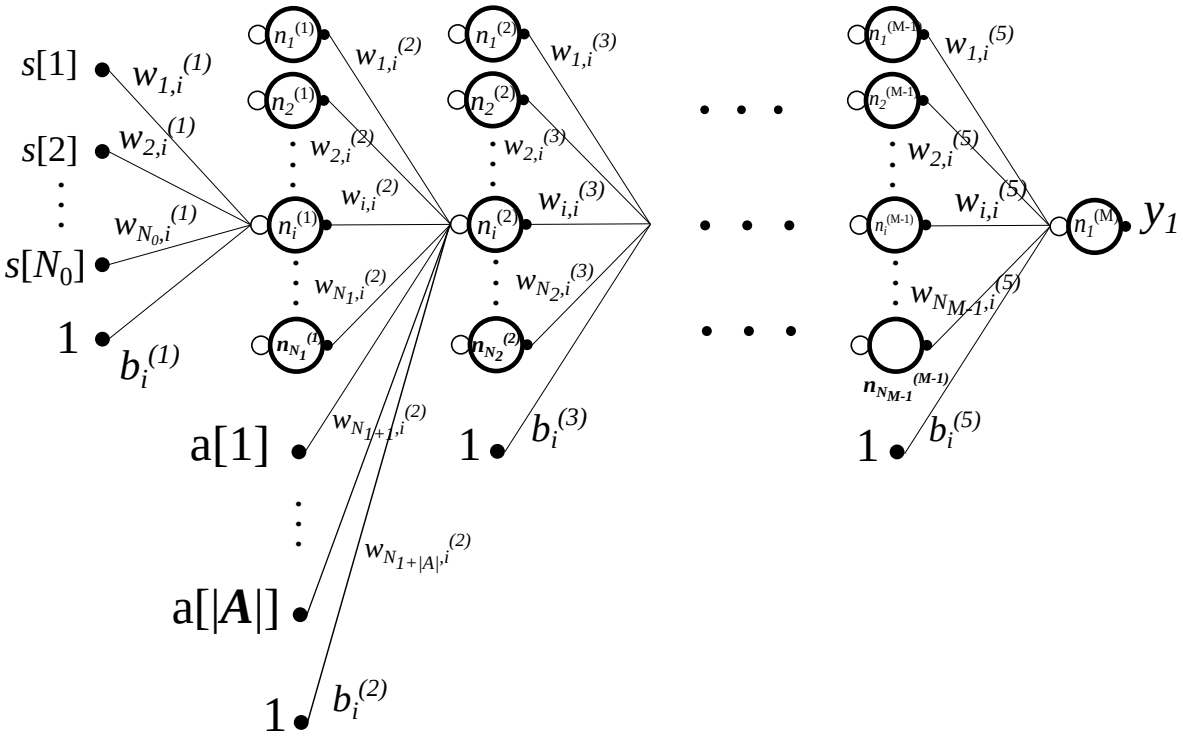

Figure 5.7 – DDPG critic NN illustration. The action input is inserted directly at the first hidden layer.

5.7 ALGORITHMS

Algorithm 7: DQL-based centralized learning-distributed execution

set number of agents N
set number of episodes M
set number of steps in an episode T
initialize experience replay memory $\mathcal{D}$
initialize $\epsilon, \epsilon_{\text{min}}, \delta, B$
for *episode = 1, M* **do**
  for *t=1,T* **do**
    initialize states sequence $\boldsymbol{s}_t := (s_t^1, \cdots, s_t^N)$
    for $i = 1, N$ **do**
      generate a random number $x \in (0, 1]$
      if $x < \epsilon$ **then**
        select random action a_t^i
      else
        select $a_t^i := \arg\max_a Q(s_t^i, a_t^i; \theta)$
      end
      execute a_t^i
    end
    if $\epsilon > \epsilon_{\text{min}}$ **then**
      $\epsilon := \epsilon - \delta$
    end
    $\boldsymbol{R}_t := (R_t^1, \cdots, R_t^N)$
    $\boldsymbol{a}_t := (a_t^1, \cdots, a_t^N)$
    $\boldsymbol{s}_{t+1} := (s_{t+1}^1, \cdots, s_{t+1}^N)$
    store all transitions $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{r}_t, \boldsymbol{s}_{t+1})$ in $\boldsymbol{D}$
    sample random mini-batches of transitions $(s_j^n, a_j^n, R_j^n, s_{j+1}^n)$ from $\boldsymbol{D}$
    if *episode terminates at step* $j + 1$ **then**
      $y_j := R_j^n$
    else
      $y_j := R_j^n + \gamma \max_{a'} Q'(\phi_{j+1}, a'; \theta^-)$
    end
    perform a gradient step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to θ
    if $t \bmod B = 0$ **then**
      $Q' = Q$
    end
  end
end

Algorithm 8: A2C-based centralized learning-distributed execution

set number of environments E
set number of agents N
set number of episodes M
set number of steps in an episode T
set actor learning rate $\alpha_A \geq 0$
set critic learning rate $\alpha_C \geq 0$
randomly initialize actor and critic parameters θ_A, θ_C
for $episode = 0, M-1$ **do**
  for $environment = 0, E-1$ **do**
    for $t = 0, T-1$ **do**
      for $n = 0, N-1$ **do**
        gather and store data $(s_{n_t}, a_{n_t}, r_{n_t}, s'_{n_t})$ by acting
      end
    end
    for $n = 0, N-1$ **do**
      for $t = 0, T-1$ **do**
        calculate predicted $\hat{V}^{\pi}(s_{n_t})$ using the critic
        calculate the advantage $\hat{A}^{\pi}_{n}(s_{n_t}, a_{n_t})$ using GAE
        calculate $V^{\pi}_{tar}(s_{n_t})$ using trajectory data
        calculate entropies $H(\pi_{\theta}(s_{n_t}))$
      end
    end
  end
  calculate critic loss:
  $L_{crit}(\theta_C) = \frac{1}{NT} \sum_{n=0}^{N-1} \left(\sum_{t=0}^{T-1} \left(\hat{V}^{\pi}(s_{n_t}) - V^{\pi}_{tar}(s_{n_t}) \right)^2 \right)$
  calculate actor loss for each agent:
  for $n = 0, N-1$ **do**
    $L^{n}_{pol}(\theta_A) = \sum_{t=0}^{T-1} \left(\hat{A}^{\pi}_{n}(s_{n_t}, a_{n_t}) \log \pi_{\theta_A}(a_{n_t}|s_{n_t}) + \beta H(\pi_{\theta}(s_{n_t})) \right)$
  end
  calculate actor loss:
  $L_{pol}(\theta_A) = \frac{1}{N} \sum_{n=0}^{N-1} L^{n}_{pol}(\theta_A)$
  update critic parameters θ_c with respect to $L_{crit}(\theta_c)$
  update actor parameters θ_a with respect to $L_{pol}(\theta_a)$
end

Algorithm 9: DDPG-base centralized-learning centralized-execution

set number of D2D pairs N
set number of episodes M
set number of steps in an episode T
initialize experience replay memory $\mathcal{D}$
initialize parameters noise process $\mathcal{N}_p$
initialize actions noise process $\mathcal{N}_a$
initialize critic Q_ψ and actor μ_θ networks, with parameters ψ and θ, respectively
initialize target networks Q^- and μ^- with parameters $\theta^- \leftarrow \theta$ and $\psi^- \leftarrow \psi$
initialize $\tilde{\mu}$ for exploration, and noise adaptation, with parameters $\tilde{\theta} \leftarrow \theta$
initialize $\sigma_k = \sigma_0$ and $k = 0$
for *episode*=$1, \cdots, M$ **do**
  perturb $\tilde{\theta} \leftarrow \theta + \mathcal{N}_p(0, \sigma_k^2)$ and obtain $\tilde{\mu}$
  for $t = 1, \cdots, T$ **do**
    sample an action $a_t = \tilde{\mu}(s_t) + \mathcal{N}_a$
  end
  execute action a_t and observe new state s_{t+1}
  store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
  if $t \mod T_{train} = 0$ **then**
    sample a random minibatch of E transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$
    set target $y_i = r_i + \gamma Q^-(s_{i+1}, \mu^-(s_{i+1}))$
    update critic by minimizing the loss $L = \frac{1}{E}\sum_i (y_i - Q_\psi(s_i, a_i))^2$
    update actor using the sampled policy gradient

$$\nabla J_\theta(\mu_\theta) = \frac{1}{N}\sum_i \nabla_\theta Q_\psi(s_i, \mu_\theta(s_i))$$

    update target networks with (4.25)

$$\theta^- \leftarrow (1-\xi)\theta^- + \xi\theta$$
$$\psi^- \leftarrow (1-\xi)\psi^- + \xi\psi$$

  end
  if $t \mod T_{adapt} = 0$ **then**
    Estimate $d_k \leftarrow \mathbb{E}[d(\mu(s), \tilde{\mu}(s)]$
    if $d_k \leq \delta$ **then**
      $\sigma_{k+1} \leftarrow \alpha\sigma_k$
    end
    else
      $\sigma_{k+1} \leftarrow \frac{1}{\alpha}\sigma_k$
    end
    $k \leftarrow k + 1$
  end
  update $\mathcal{N}_a$
end

6 SIMULATIONS AND RESULTS

The results were obtained through a computational simulator, developed during this work. The software was developed using the Python programming language [79]. The deep learning was implemented with the PyTorch library [80]. The software was developed by the author and its code is available on GitHub [81].

In this chapter, we explain the performed simulations, the parameters choices and their implementations. Afterwards, we go over the obtained results presentations and discussions.

6.1 CHANNEL MODELS

In our simulations, we considered two channel models: the Body Area Network (BAN) [82] and the Wide Area Wireless World Initiative New Radio II (WINNER II) [83] C2 channel models. These models were chosen in order to bring the simulations closer to a realistic urban scenario.

The BAN model is proposed in the ITU-R journal. It describes scenarios where the devices are on the vicinities of the human body. These scenarios are described with more depth in [83]. We chose to apply the off-body, office scenario, in order to simulate IoT devices close to the human body. The channel loss, for the 2.4 GHz frequency, is given by

$$L = L_0(d_0) + 10n_{pl}\log(d/d_0) + \Delta L_{ls} + \Delta L_{ss}, \tag{6.1}$$

where $L_0(d_0)$ is the mean path loss at the reference distance d_0, n_{pl} is the path loss exponent, ΔL_{ls} and ΔL_{ss} are the large-scale and small-scale fading components, respectively.

The model's large-scale fading is modeled as a log-normal distribution [84], with log-mean μ_L and log-standard deviation σ_L.

The small-scale fading follows a Nakagami distribution [84] with a shape parameter m_L and scale parameter Ω_L.

The channel parameters are given in Table 6.1. The mean path loss is displayed in Figure 6.1. The large-scale and small-scale distributions may be seen in Figures 6.2 and 6.3, respectively.

The second channel model we adopted was the WINNER II C2 [82]. This model describes an urban scenario, where the base station is clearly above the surrounding building heights, and the buildings height and density are mostly homogeneous. We consider the

Table 6.1 – BAN channel parameters.

Parameter	Value
Environment	Office
d_0	1 m
$L_0(d_0)$	32 dB
n_{pl}	1.71
μ_L	0 dB
σ_L	1.2 dB
m_L	0.9
Ω_L	1

Table 6.2 – WINNER II C2 NLOS channel parameters.

Parameter	Value
A_1	$(45.9 - 6.55\log_{10}(h_{BS}))\log_{10}(d)$
A_2	$34.46 + 5.83\log_{10}(h_{BS})$
A_3	23
μ_{ls}	0 dB
σ_{ls}	8 dB
μ_{ss}	0 dB
σ_{ss}	4 dB

non-line-of-sight (NLOS) scenario.

The general WINNER II channel path loss expression is given on the format

$$L = A_1 \log_{10}(d) + A_2 + A_3 \log_{10}(f_c/5) + \Delta L_{ls} + \Delta L_{ss}, \tag{6.2}$$

where A_1 is a fitting parameter that included the path-loss exponent, A_2 is the intercept and A_3 describes the path loss frequency dependency. The carrier frequency is denoted by f_c, and may vary from 2 to 6 GHz. ΔL_{ls} and ΔL_{ss} are, respectively, the large- and small-scale fading components.

The large-scale fading component is defined by a log-normal distribution with zero log-mean μ_{ls} and log-standard deviation σ_{ls}. The small-fading component is not defined by the model. Since it is a NLOS scenario, we decided to model the small-fading component following a Rayleigh distribution [84], with mean μ_{ss} and standard deviation σ_{ss}.

For the C2 NLOS scenario, the channel loss is given in Table 6.2, where h_{BS} is the BS height, and d is the distance between transmitter and the loss measurement spot.

The model path loss is presented in Figure 6.1. Figures 6.2 and 6.3 present the large- and small- scale fading components, respectively.

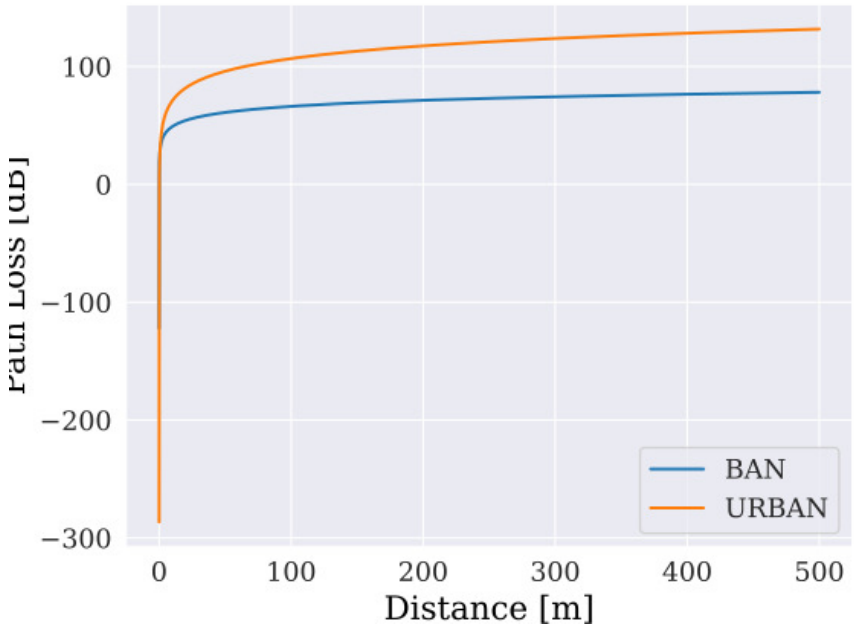

Figure 6.1 – Path losses.

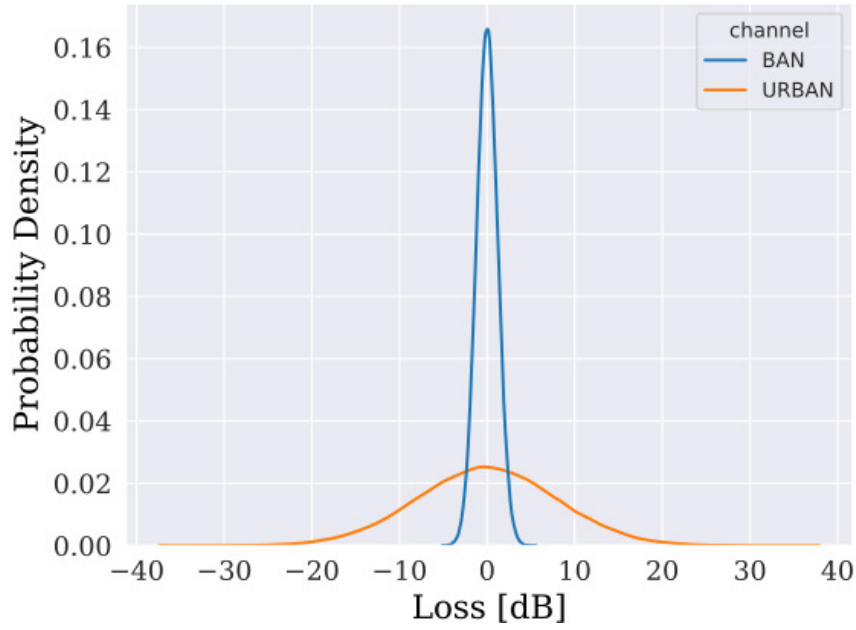

Figure 6.2 – Large scale fadings.

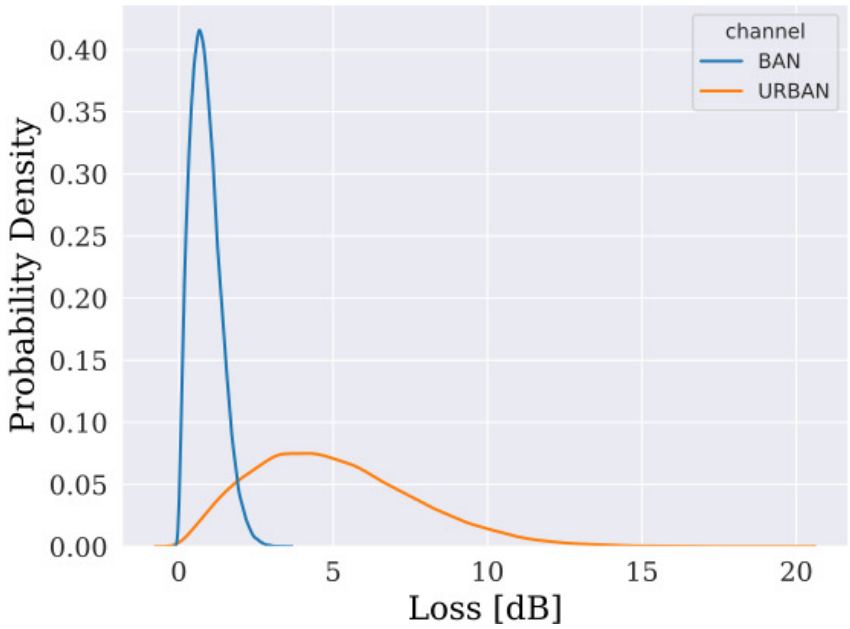

Figure 6.3 – Small scale fadings.

6.2 SIMULATIONS METHODOLOGY

All results were produced with a computational interferences simulator, as mentioned before. The software is responsible for randomly positioning the devices, generating the channels, and calculating the devices SINRs, as well as other metrics. The calculations results are used for obtaining the states and rewards values. The states are passed to the algorithms, that use this information to define the D2D transmission power levels. The rewards are used for the loss functions calculations, that are used in the algorithms training stages. In fewer words, the simulator truly acts as the environment in Figure 2.1.

Throughout the results, presented in this work, the environment parameters remained unchanged. Table 6.3 presents such parameters. The training parameters, for all algorithms, are displayed in Tables 6.4, 6.5 and 6.6.

After testing different power level sets, the chosen power values for the CLDE algorithms were

$$\boldsymbol{A} = [-90, -40, -30, -20, -10, 0],$$

where the values are in dBW.

Parameter	Value
Number of MUEs	1
Carrier frequency f_c	2.4 GHz
BS radius	1000 m
BS antenna height	25 m
BS gain	17 dBi
Minimum D2D initial pair distance	1.5 m
Maximum D2D initial pair distance	15 m
Devices height	1.5 m
Devices gain	4 dBi
Maximum devices transmit power	10 dBW
Noise power	-146 dBW
MUE SINR threshold τ_0	6 dB
MUE SINR margin τ_1	6 dB

Table 6.3 – Environment parameters.

Parameter	Value
Training steps	$4 \cdot 10^4$
Replay buffer size $\lvert\mathcal{D}\rvert$	10^4
Batch size	128
Return discount factor γ	0.5
NN hidden layers size	64
NN number of hidden layers	2
Activation function	ReLU
Optimizer	Adam [41]
Optimizer learning rate	$2 \cdot 10^{-4}$
Target NN update interval	50
ϵ-greedy initial value	1
ϵ-greedy minimum value	0.01
ϵ-greedy decay	$5 \cdot 10^{-5}$
Reward parameter α_1	2.0
Reward parameter α_2	0.1
Devices position distribution	Uniform

Table 6.4 – DQL training parameters.

Parameter	Value
Number of environments	4
Return discount factor γ	0.5
Entropy discount β	0.01
Training steps	$2 \cdot 10^4$
NN hidden layers size	64
NN number of hidden layers	2
Activation function	ReLU
Optimizers	Adam [41]
Actor optimizer learning rate	$2 \cdot 10^{-4}$
Critic optimizer learning rate	$2 \cdot 10^{-3}$
GAE λ	0.95
Reward parameter α_1	2.0
Reward parameter α_2	0.1
Devices positions distribution	Uniform

Table 6.5 – A2C training parameters.

Parameter	Value
Replay buffer size $\lvert\mathcal{D}\rvert$	$2 \cdot 10^4$
Batch size	128
Return discount factor γ	0.9
Training steps	$3 \cdot 10^4$
Soft update parameter ξ	0.05
Initial parameters space noise σ_0	0.1
Parameters space noise adaptation coefficient α	1.01
Parameters space noise adaptation interval T_{adapt}	5
Target NNs update interval T_{train}	1
Actions space noise mean $\mu_{\mathcal{N}_a}$	0
Actions space noise initial scale $\sigma_{\mathcal{N}_a}$	4
Actions space noise minimum scale $\sigma_{\mathcal{N}_a}$	10^-3
NN hidden layers size	64
NN number of hidden layers	2
Activation function	ReLU
Optimizers	AdamW [42]
Actor optimizer learning rate	$2 \cdot 10^{-4}$
Critic optimizer learning rate	$2 \cdot 10^{-3}$
GAE λ	0.95
Reward parameter α_1	10.0
Reward parameter α_2	1.0
Devices positions distribution	Normal $\mathcal{N}(0, 450)$

Table 6.6 – DDPG training parameters.

6.3 MODELS CONVERGENCE

In this section, the rewards curves obtained in training are displayed, in order to show the algorithms convergence. Figures 6.4, 6.5 and 6.6 exhibit the curves. All the curves were obtained for the two D2D pairs case.

The plots show that all proposed algorithms have well behaved convergences. Throughout the training process, the rewards functions converge and remain at defined regions.

In all figures, the bands around the plotted curves stand for the 95% confidence interval, assuming the average estimated values are distributed following a normal distribution. For each calculated average, 100 simulations were run, during the training process, in order to obtain the results.

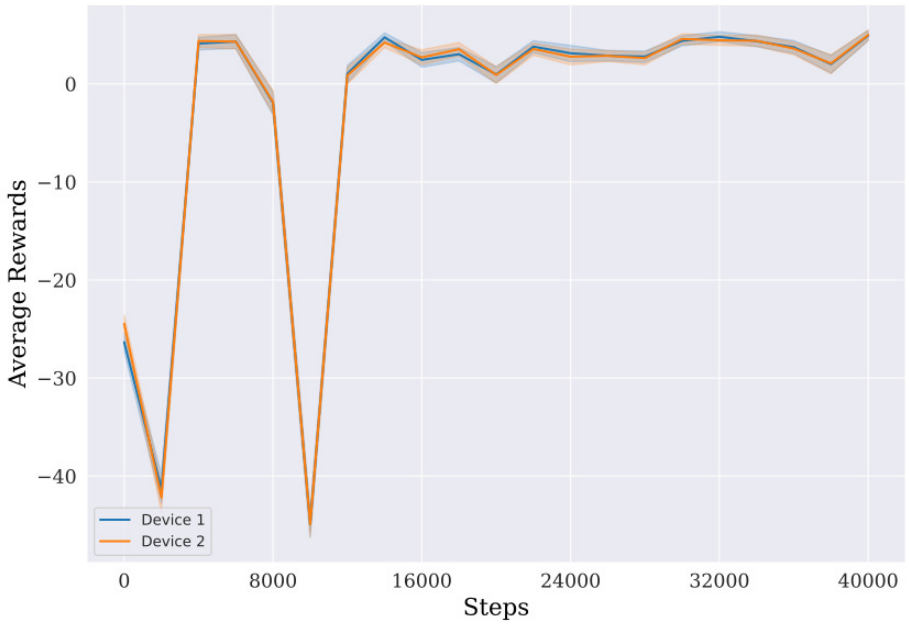

Figure 6.4 – DQL training rewards.

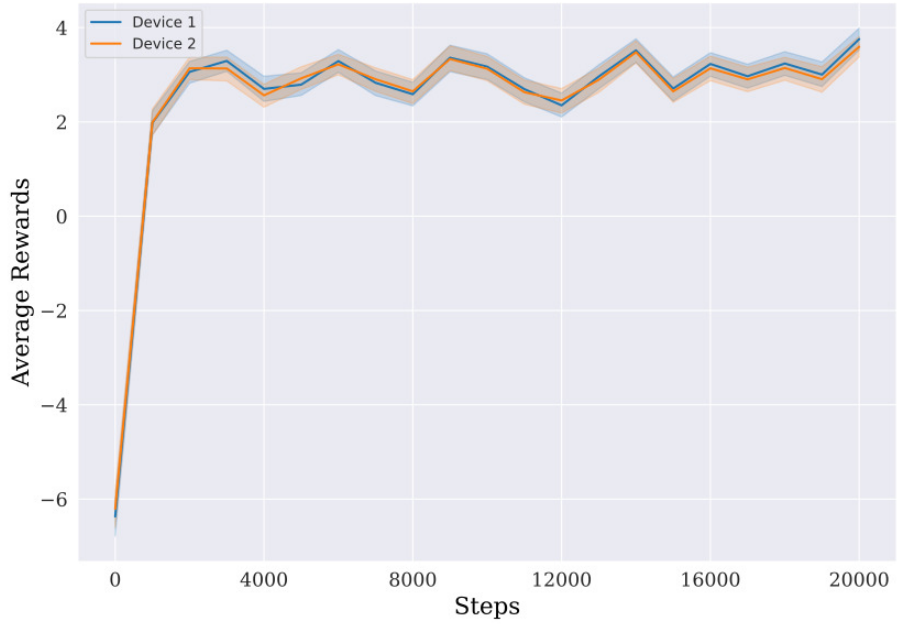

Figure 6.5 – A2C training rewards.

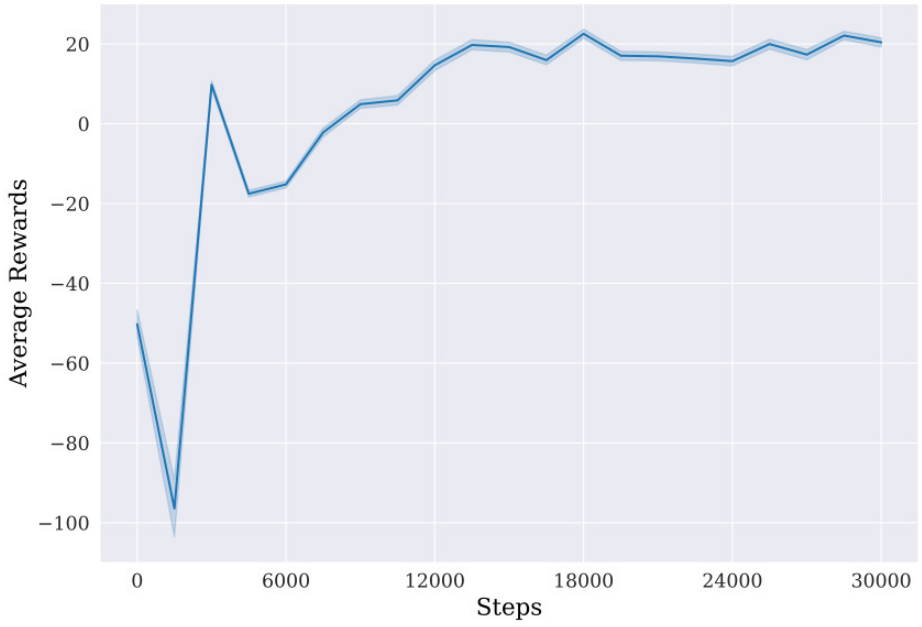

Figure 6.6 – DDPG training rewards.

6.4 NUMBER OF D2D PAIRS

In this section, the results show the influence of the amount of D2D pairs on the algorithms' average performance, so we may have a better overview on the solutions' behaviour. The algorithms are also compared to two random-based power control algorithms, in order to guarantee the proposed solutions learning capabilities and prove they are capable of better performances than the ones resulted from random power allocation. For each number of D2D pairs, for each algorithm, 10000 simulations were run in order to obtain the plotted averages.

The two random algorithms are called *Discrete*- and *Continuous*- random. In discrete-random, the transmitters choose their transmission powers from a discrete set of transmission power values, following a discrete uniform distribution. In a similar way, in the continuous-random strategy, the transmitters select a transmission power value from a range of continuous values, following a continuous uniform distribution.

Figure 6.7 exhibits the average MUE SINR according to the number of D2D pairs. It is noticeable the CLDE solutions, DQL and A2C, are not able to maintain a high MUE QoS level, as the average MUE SINR decreases as the number of D2D pairs increases. The CLCE solution, DDPG, was able to maintain a high and constant average MUE QoS, despite the increase on the number of D2D pairs.

The DQL algorithm failed to provide good MUE SINR levels when the number of D2D pairs was greater than two. This happened because the algorithm failed to converge for these conditions and was unable to find reasonable power allocations.

The MUE SINR directly affects the MUE availability, which is shown in Figure 6.8. The MUE availability stands for the percentage of times the MUE SINR stood above the minimum requirement.

The CLDE solutions were able to maintain the availability around 80% up till two D2D pairs, having a noticeable performance drop from this point on, specially the DQL algorithm. DDPG was able to keep high and stable MUE availability levels, around 100%. Both random algorithms performed poorly. The discrete-random algorithm was able to outperform the continuous one due to the discrete power values set, which has a more controllable and restricted set of options. The continuous-random algorithm picks its power values from a continuous range of values, which provides much more freedom, and bad choices, when in comparison with the discrete values set.

Figure 6.9 displays the average D2D SINR versus the number of D2D pairs. Both CLDE solutions provide similar results up until two D2D pairs. From there on, the DQL algorithm provides around 10 dB more D2D SINR than the A2C one. This difference shows the A2C algorithm acts in a more controlled way, transmitting with smaller power levels than the

DQL one.

By comparing Figures 6.9 and 6.8, it is perceived that the D2D SINR increase is accompanied by a decrease in MUE availability. This is explained by the agents transmitting at power levels that are not optimal and higher than what they should be. Therefore, this behaviour is not desired, since it impairs the MUE QoS.

The same critic applies to the random-algorithms. Both solutions act by choosing high and inappropriate transmission power levels, providing high D2D SINRs at the expense of the MUE SINR.

The DDPG goes in the opposite way, by providing low average D2D SINRs and guaranteeing high QoS for the MUE. This was caused by the low transmission power values employed by the agents.

Figure 6.11 shows the average rewards versus the number of D2D pairs. At first, it is important to notice the distinguished behaviour by the DDPG and *continuous random* solutions when in comparison to the other curves. This happened because their reward function is different from the other algorithms'.

The A2C algorithm presents an almost constant reward curve, and the DQL presents a decreasing one. Both random algorithms present low reward values due to their disregard to the MUE SINR. All the CLDE solutions provide higher average rewards than the ones obtained by the random algorithms. The DDPG solution rewards are explained by the high praise its reward function gives to high MUE SINR levels.

In conclusion, all the DRL-based power control solutions were able to surpass random schemes, which goes to show their learning capabilities. The CLDE solutions presented more aggressive behaviours, by transmitting with higher power levels, culminating into higher D2D SINRs and lower MUE SINRs. In contrast, the DDPG had a more conservative behaviour, by transmitting with smaller power levels, providing MUE QoS at the expense of lower D2D SINRs.

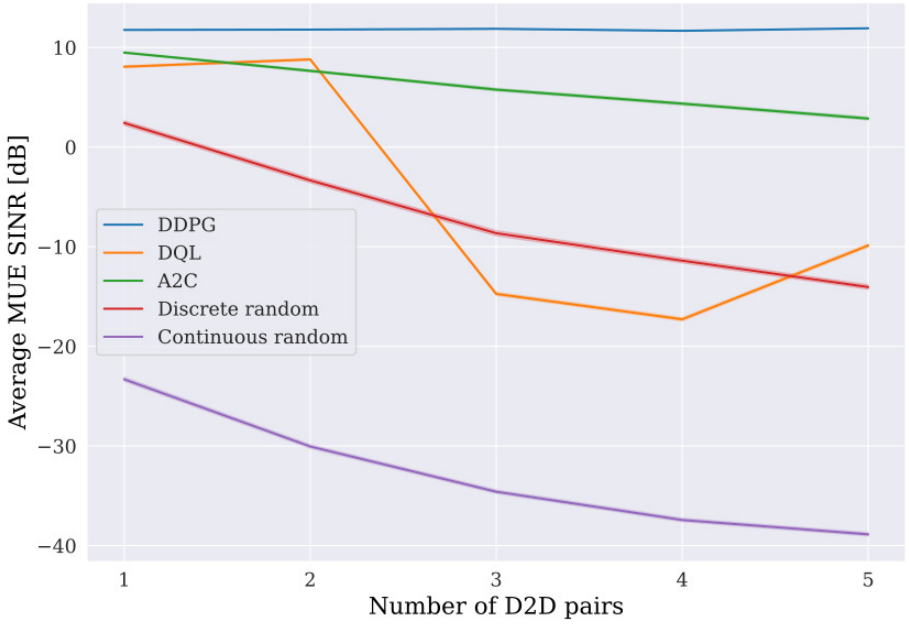

Figure 6.7 – MUE SINR comparison.

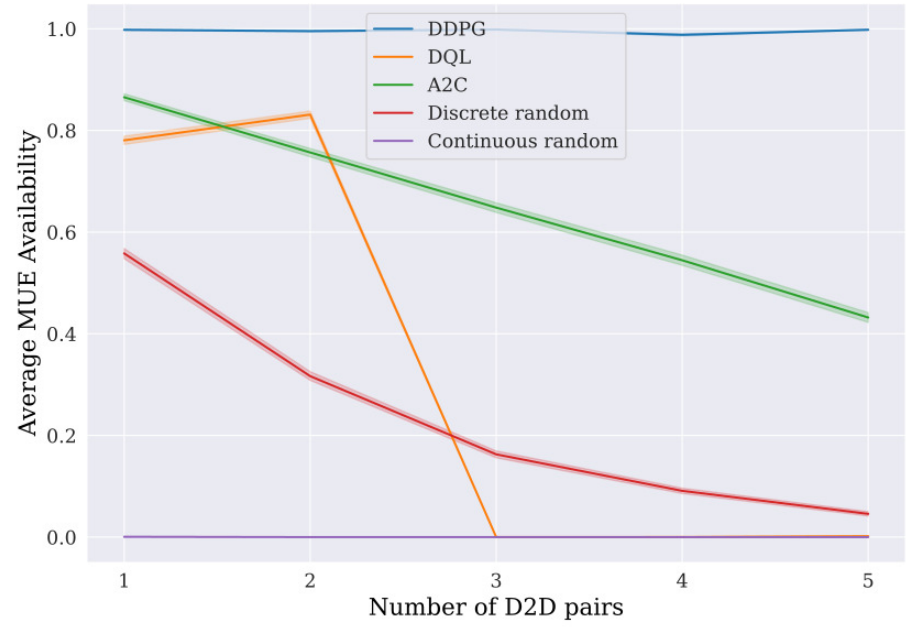

Figure 6.8 – MUE availability comparison.

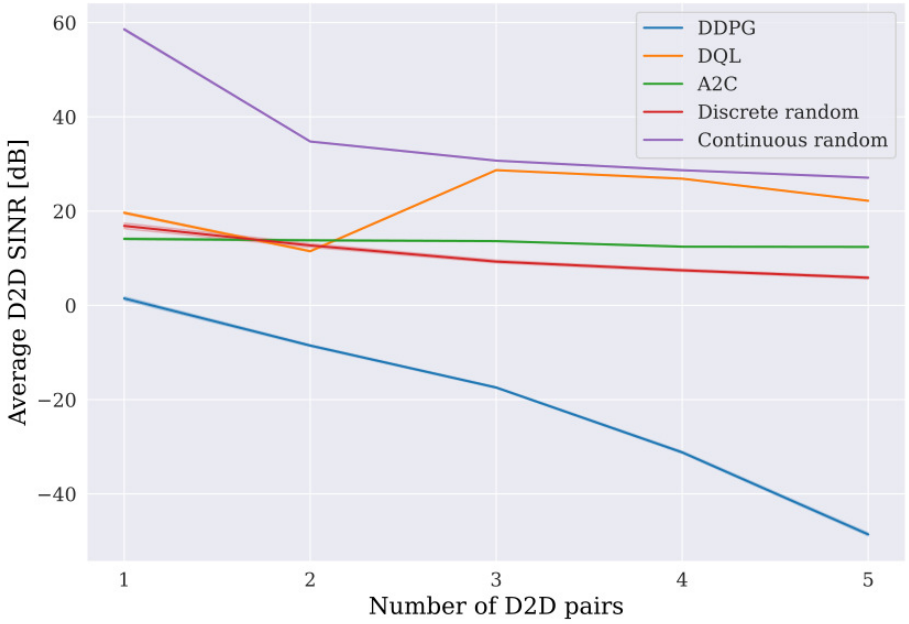

Figure 6.9 – D2D SINR comparison.

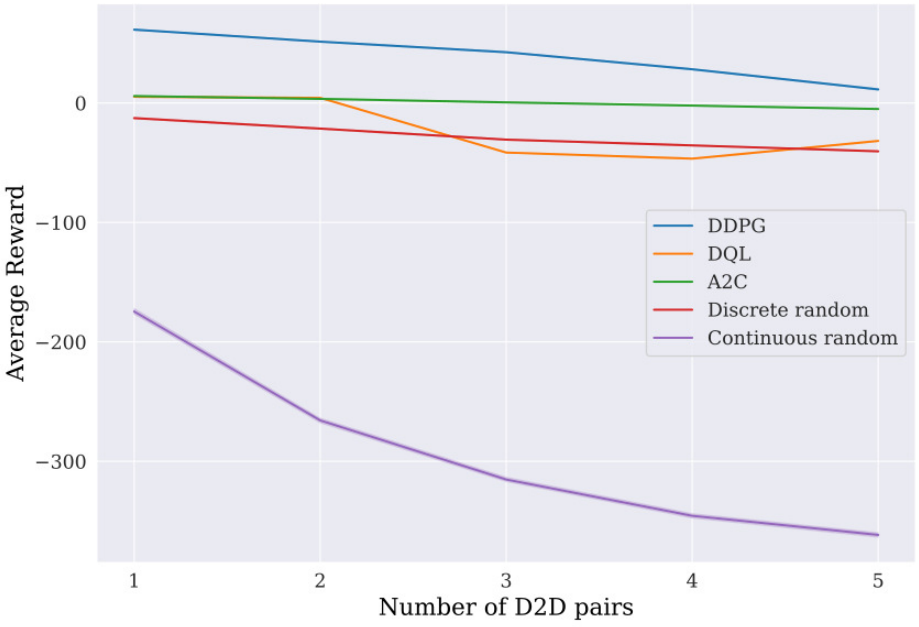

Figure 6.10 – Rewards comparison.

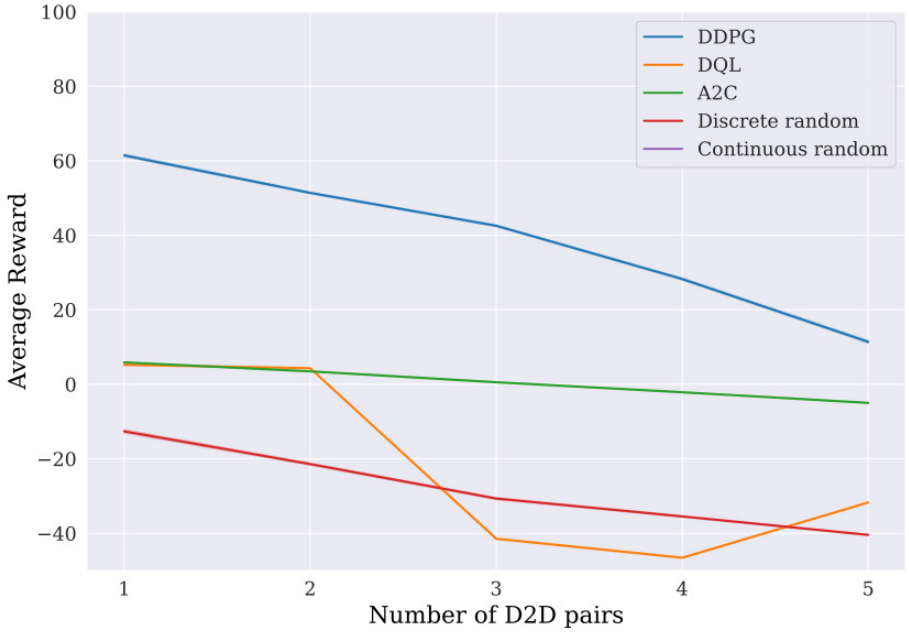

Figure 6.11 – Rewards comparison. The *continuous random* curve was removed for better resolution.

6.5 CONTROLLED EXPERIMENTS

In the last section, the results provided knowledge on the algorithms' behaviour. However, these results were obtained by averaging the measurements over a number of episodes. Therefore, these results provide *macro*, general informations, across many situations.

In order to obtain more in-depth insights on the algorithms' behaviour, controlled experiments are presented here, along with their results. These experiments are scenarios with expected outcomes. By comparing the algorithms actions with the expected outcomes, it is possible to evaluate the solutions' performance and adaptability in these scenarios.

In all experiments, there are two D2D pairs, one MUE and the BS. For Experiment 1 and Experiment 2, the obtained measurements were the MUE and D2D SINRs, transmit powers, and the MUE availability. In this case, the availability corresponds to the binary interference indicator I_k.

6.5.1 Experiment 1

In this experiment, the D2D pairs begin close to the BS and move to the cell's edge during the episode. In contrast, the MUE starts at the cell's edge and moves to the cell's center,

close to the BS. Figure 6.12 displays the devices initial positions and Figure 6.13 shows the devices trajectories. The distance between the D2D transmitters and their respective receivers stays constant at 5 meters, throughout the whole experiment. Figure 6.14 displays the MUE transmission power in Experiments 1 and 2.

The expected outcome for this experiment is for the D2D transmitters to remain turned off in the initial moments. As the pairs move away from the BS and the MUE approaches the BS, the D2D agents are expected to similarly increase their transmission power levels, since the further distance mitigates the interference on the BS and both pairs are equidistant to the BS. The power levels chosen by the agents might still differ due to different fadings experienced by the channels. The average availability, across the episodes, is expected to be close to one.

6.5.1.1 DQL

This section contains the discussions about the results obtained with the DQL algorithm in Experiment 1.

Figure 6.15 presents the MUE SINR throughout Experiment 1. As time passes, it increases due to the MUE getting closer to the BS. Around 225 s, the SINR drop matches the instant one of D2D transmitters raises its transmission power level, as can be seen in Figure 6.16.

Figure 6.16 shows the D2D pairs SINR during Experiment 1 and Figure 6.17 displays the transmission powers employed by the D2D transmitters. The results show the devices begin the experiment by transmitting at -40 dBW. Near 225 s, one of the devices raises its transmit power to -20 dBW. In this moment, the D2D pairs are further away from the BS and the MUE is closer to the BS, when in comparison to the beginning of the experiment.

Figure 6.18 exhibits the MUE availability in the course of Experiment 1. At the beginning, the availability is zero and it increases as the time passes and the devices change positions. Around 250 s the availability quickly drops, in an instant device 0 has already increased its transmit power. This happened because of the fast fading channels. Throughout the episode, the average availability was 76.28%.

By looking at the results obtained with the DQL algorithm, it is possible to say the solution acted partially as expected. The agents wait until they are further away from the BS, and the MUE is closer to the BS, before raising their transmit power. However, their initial transmit power should be lower. While close to the BS, at the experiment's beginning, they chose the -40 dBW power level, instead of the lower -90 dBW, which is considered turned off. This would have avoided the initial low MUE availability. Only one of the devices raised its transmit power level, when it was expected that both devices would do it in a similar way.

6.5.1.2 A2C

In this section, the results obtained with the A2C algorithm, for Experiment 1, are discussed.

Figure 6.19 shows the MUE SINR, throughout the experiment. It presents a similar behaviour to what has been seen previously, for the DQL solution.

The D2D SINRs, displayed in Figure 6.20, reflect the employed transmission powers that Figure 6.21 presents. The devices kept alternating between the -40 dBW and -30 dBW power levels. It was not possible to define a clear power allocation pattern for the algorithm.

Figure 6.22 exhibits the MUE availability during Experiment 1. It begins at zero and increases as the D2D pairs get away from the BS and the MUE approaches the BS. The average availability was 74.57%.

The A2C solution was not able to provide the expected power allocation. Instead of turning off at the experiment's initial moments, and increasing the transmit powers as the experiment progressed, the agents kept alternating between two power levels, without a clear action pattern. However, they were able to achieve an average MUE availability that is close to the one achieved by the DQL solution.

6.5.1.3 DDPG

This section deliberates on the results obtained by the DDPG solution in Experiment 1. This solution clearly acts differently when in comparison to the previous ones

Figure 6.23 displays the MUE SINR. Right from the start, the MUE SINR already is above the minimum threshold of 6 dB, and it increases according to the devices motion.

Figure 6.24 presents D2D SINRs, during the experiment. They begin at lower levels and increases as time passes, matching the devices power control behaviour, which is seen in Figure 6.25. Both devices begin transmitting at low power levels, around -90 dBW, which is considered turned off, and increase their power levels similarly. A possible explanation for the difference in power levels is the different channel fadings. Although the final transmit powers are very low, around -50 dBW and -70 dBW, they are still close to the LTE minimum transmit power specification [85].

Despite employing such low transmit powers, the D2D devices were still able to achieve decent SINR levels, surpassing 10 dB. The algorithms' policy resulted on a average MUE availability of 100%, as depicted by Figure 6.26.

The DDPG solution partially fulfilled the expectations. Both the D2D transmitters begun with really low power levels and, similarly, increased the transmit powers as the D2D pairs got further away from the BS and the MUE got closer to the BS. However, the algorithm

could have employed higher power levels towards the end of the experiment, since the other solutions, A2C and DQL, were able to transmit at higher power levels, and achieve higher D2D SINRs without compromising the MUE QoS.

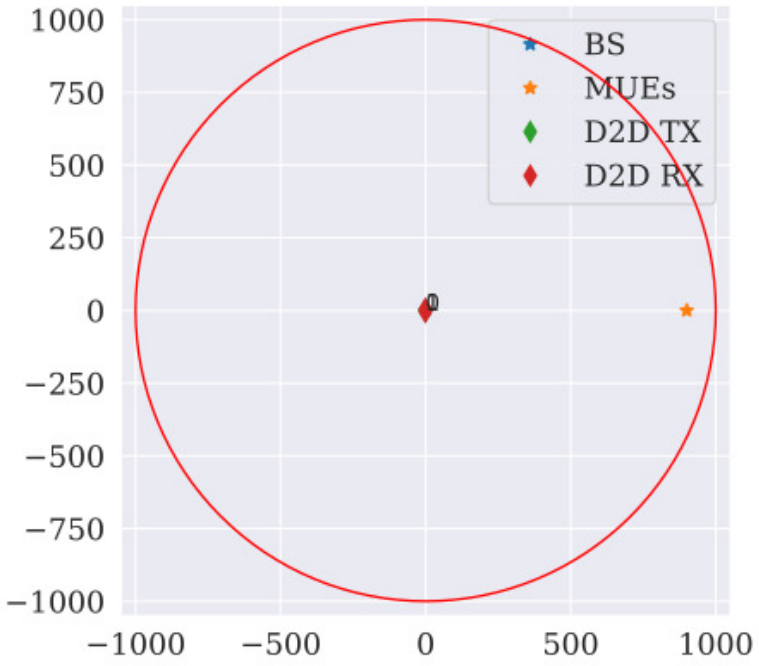

Figure 6.12 – Devices original positions in Experiments 1 and 2.

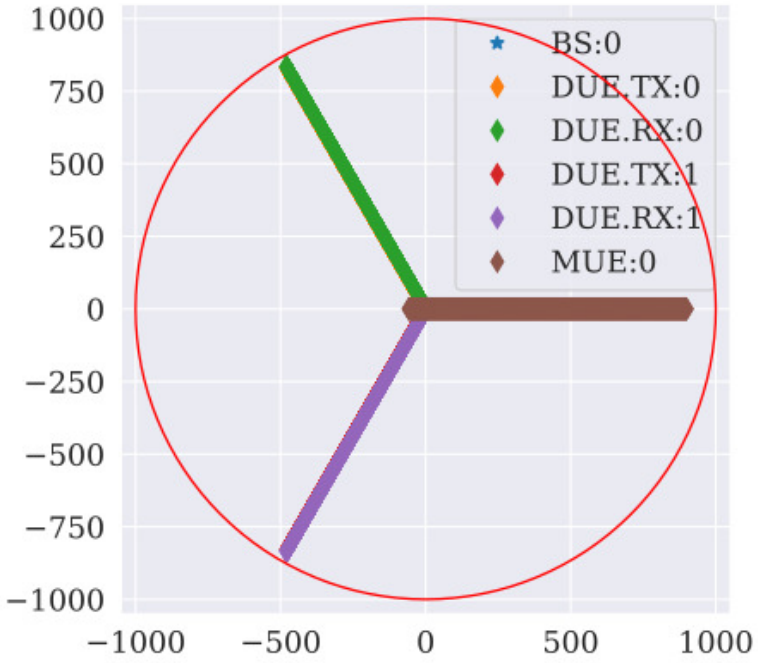

Figure 6.13 – Devices trajectories in Experiment 1.

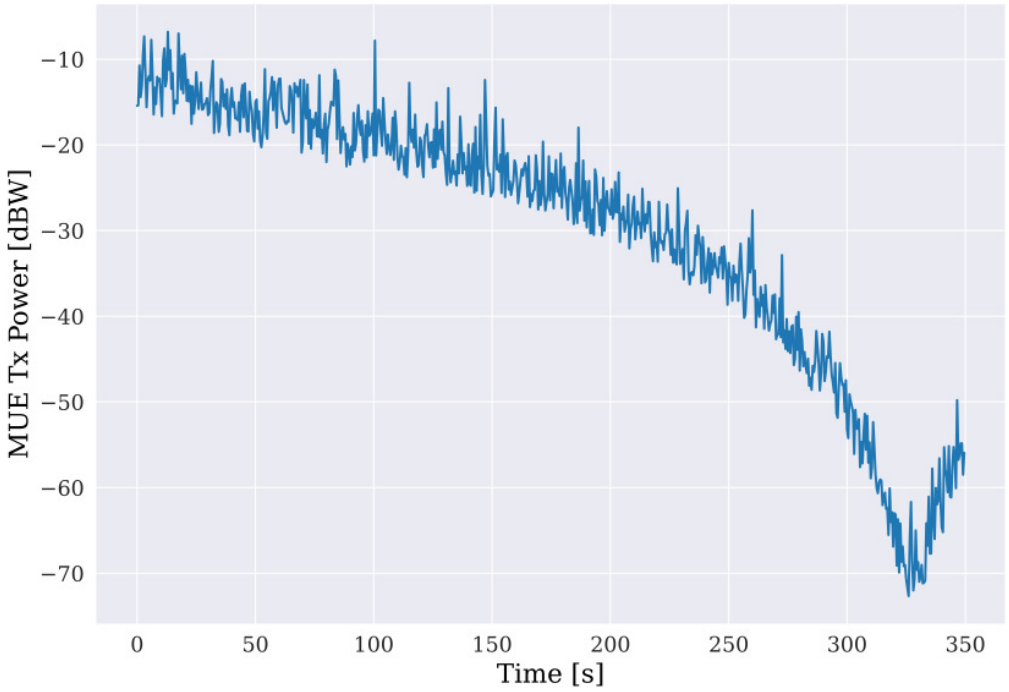

Figure 6.14 – MUE transmission powers in Experiments 1 and 2.

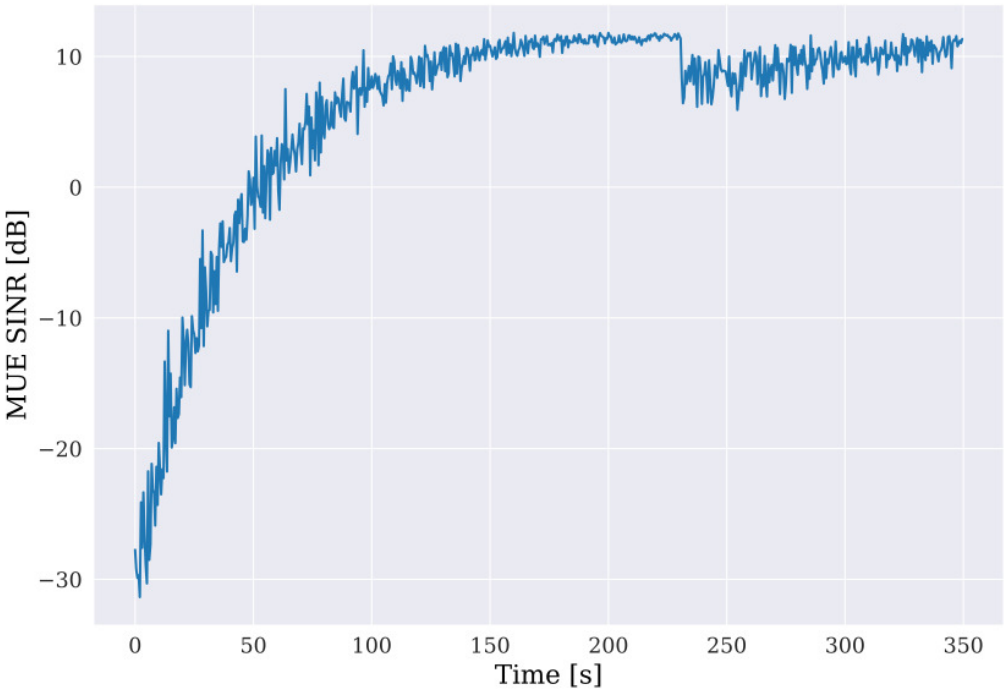

Figure 6.15 – MUE SINR for DQL in experiment 1.

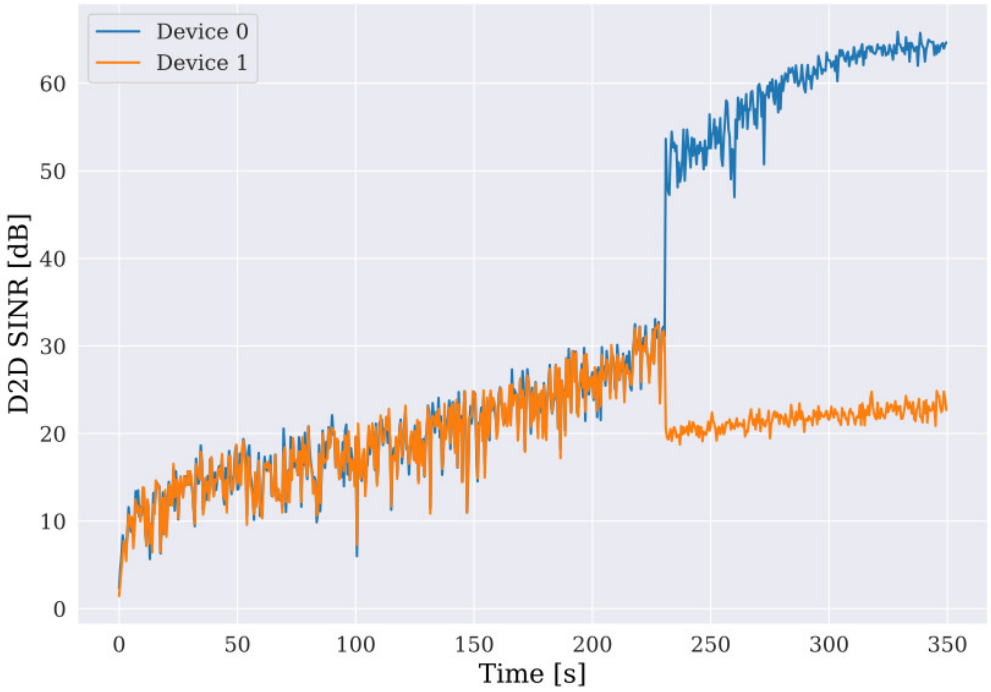

Figure 6.16 – D2D SINR for DQL in experiment 1.

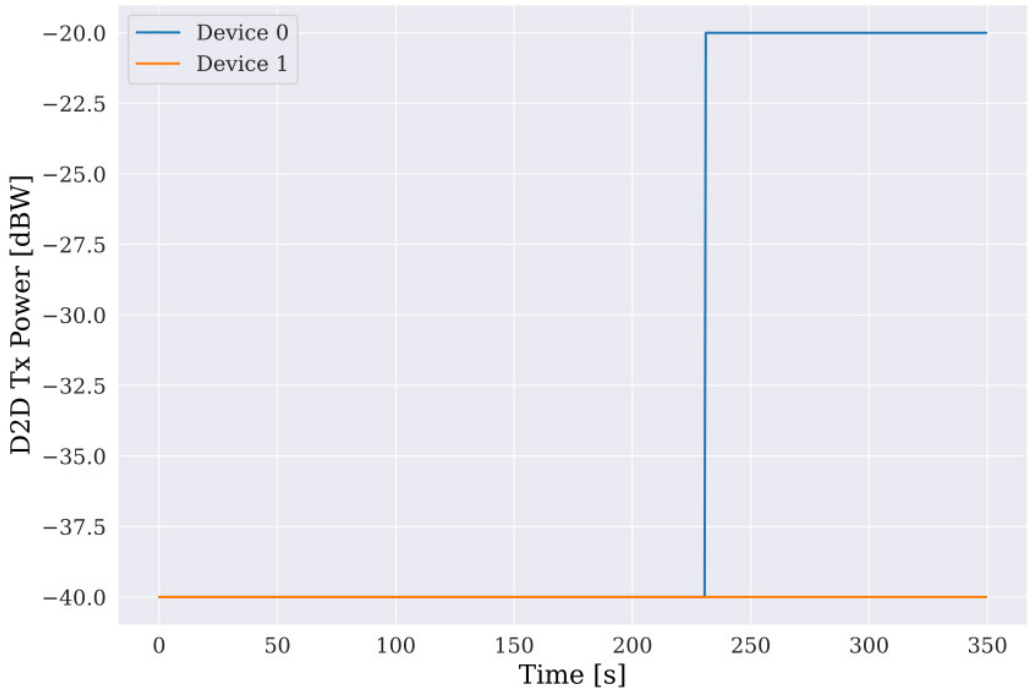

Figure 6.17 – D2D transmission powers for DQL in experiment 1.

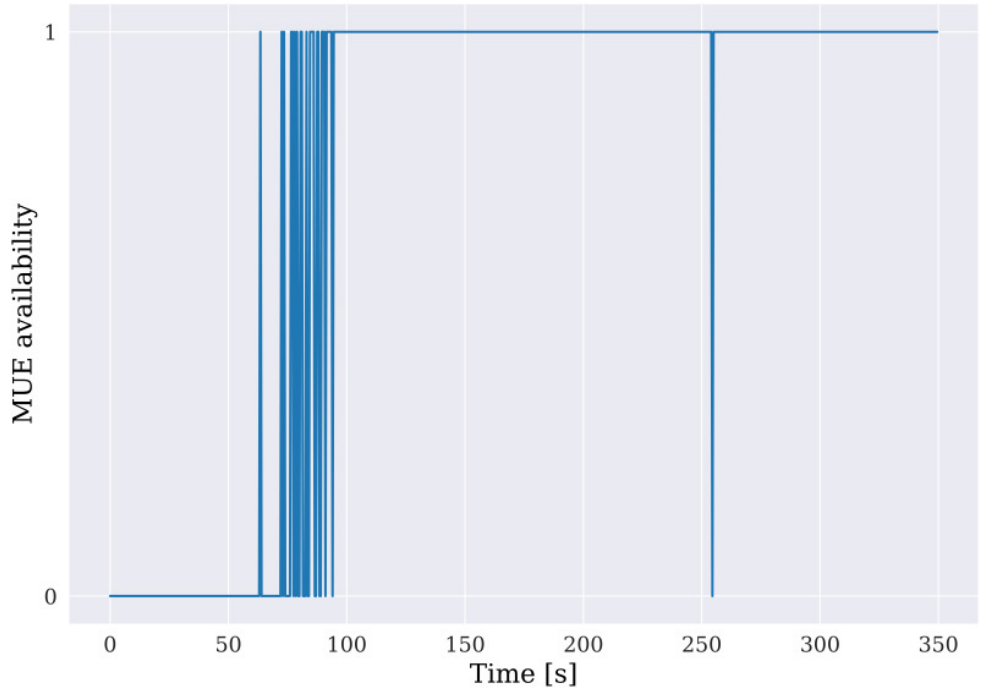

Figure 6.18 – MUE availability for DQL in experiment 1. Average availability of 0.7628.

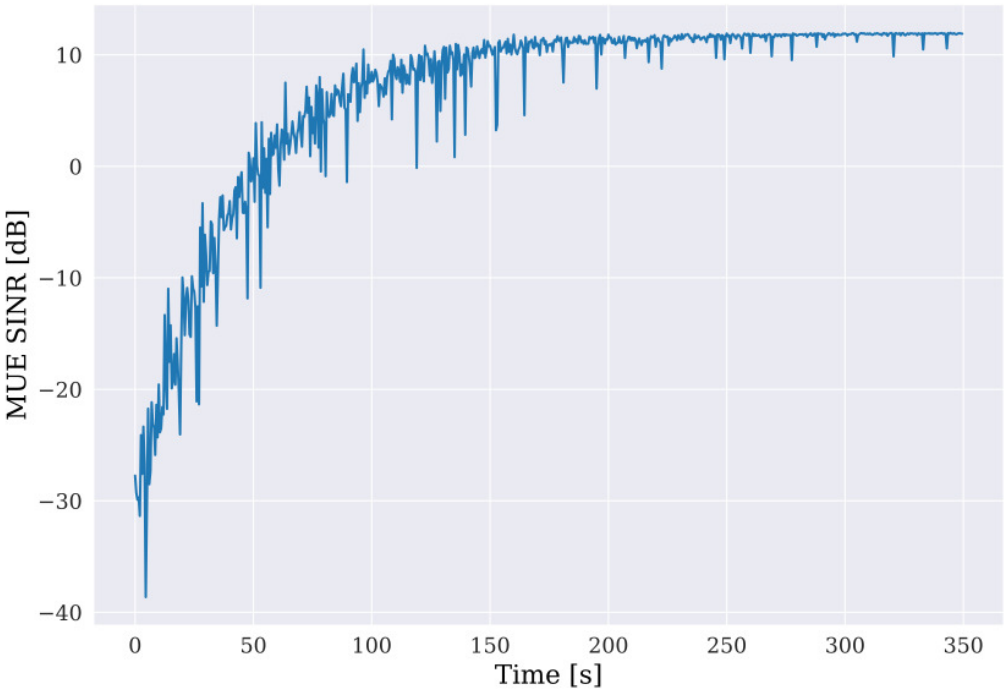

Figure 6.19 – MUE SINR for A2C in experiment 1.

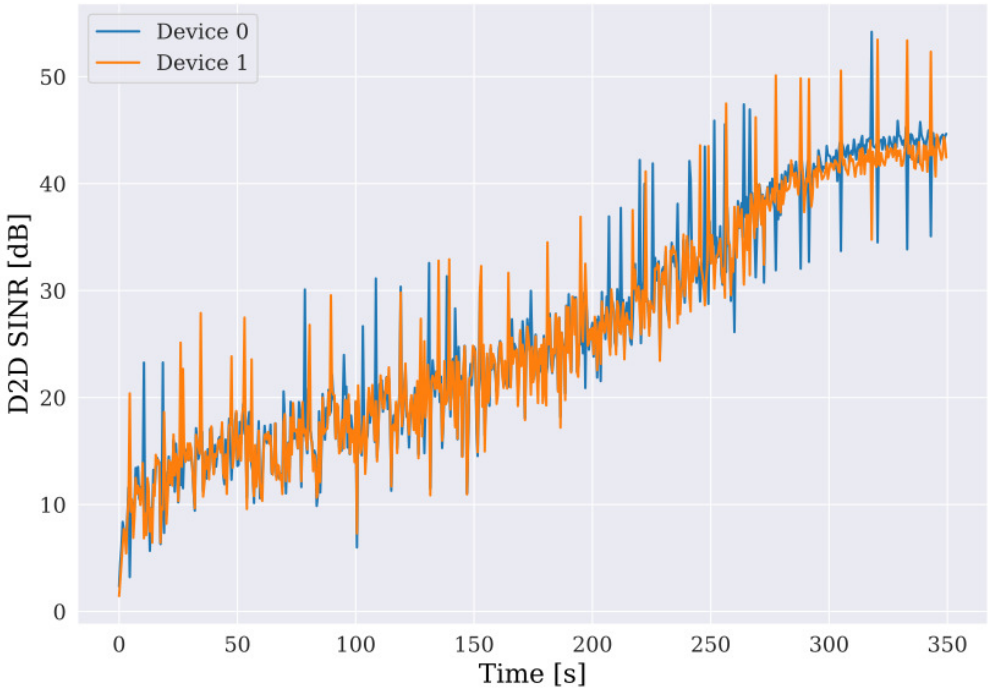

Figure 6.20 – D2D SINR for A2C in experiment 1.

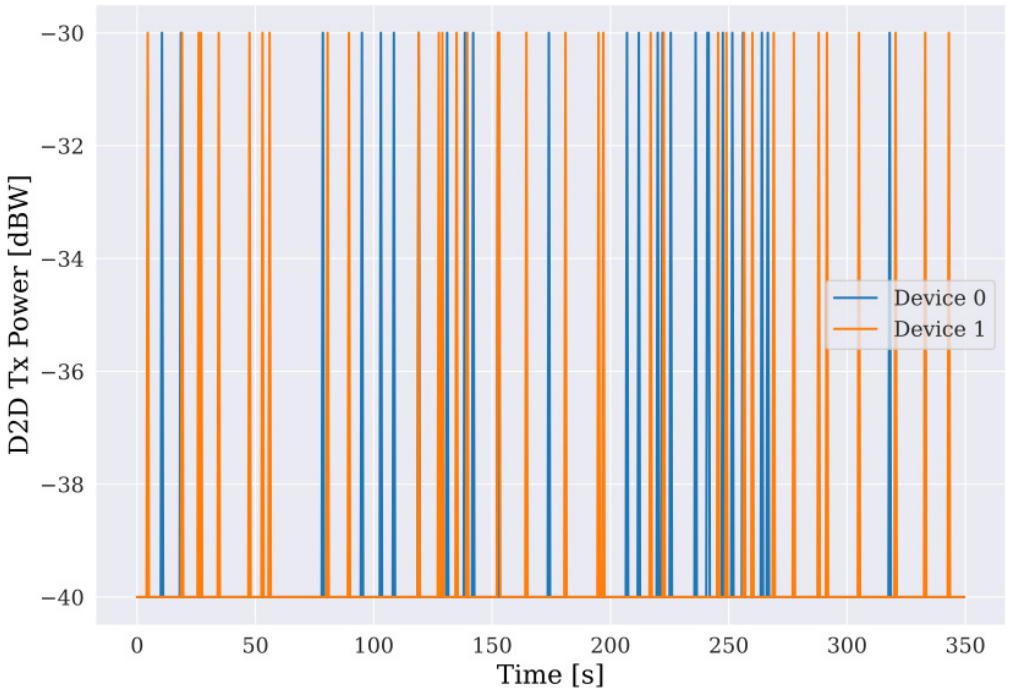

Figure 6.21 – D2D transmission powers for A2C in experiment 1.

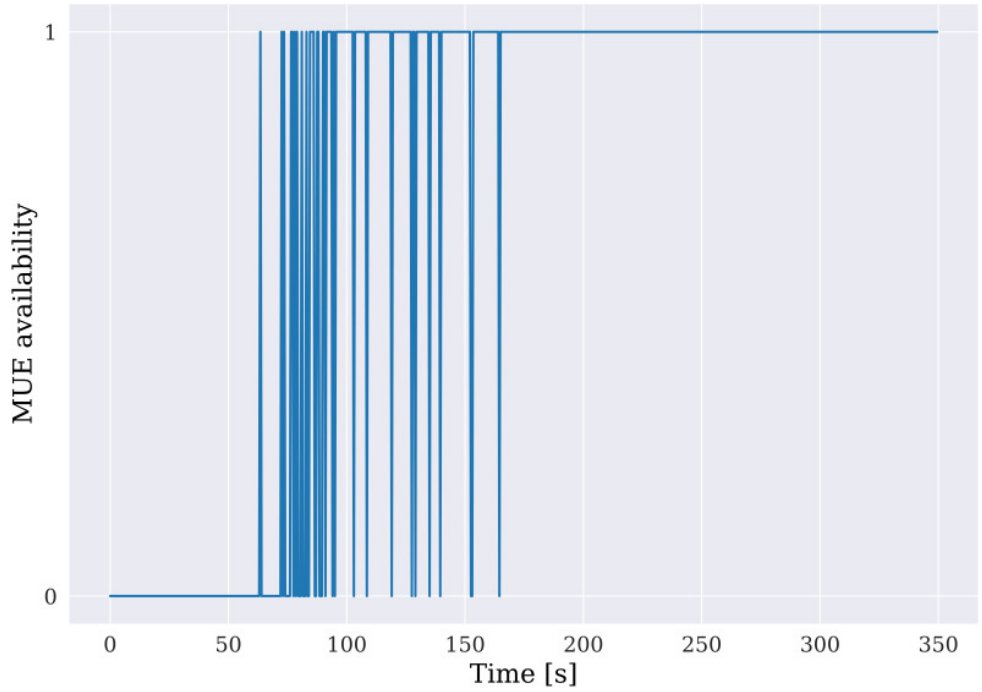

Figure 6.22 – MUE availability for A2C in experiment 1. Average availability of 0.7457.

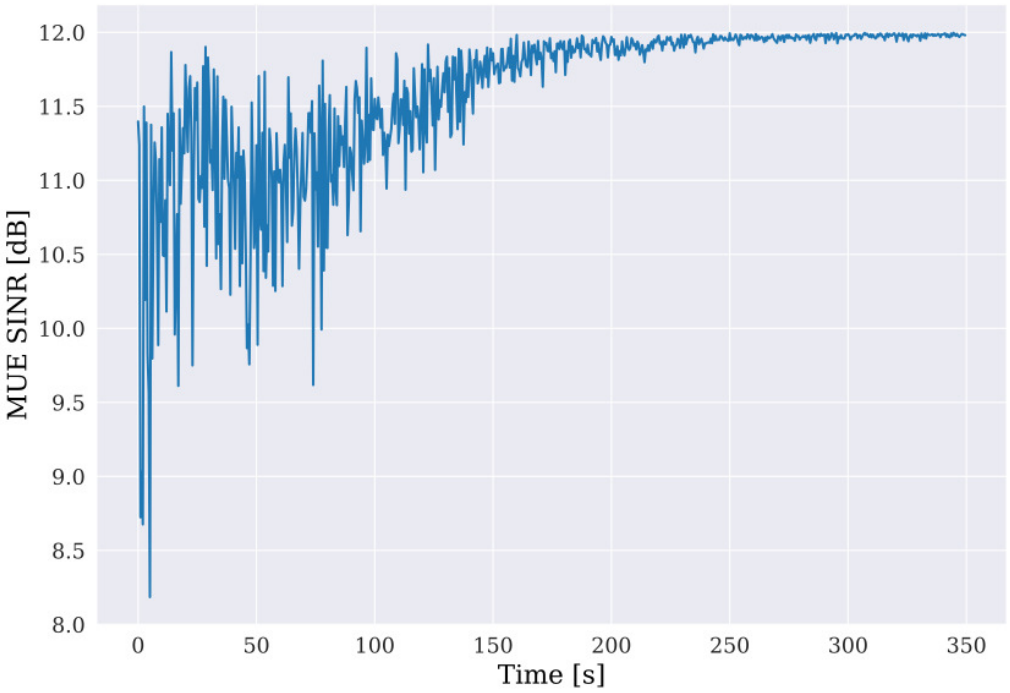

Figure 6.23 – MUE SINR for DDPG in experiment 1.

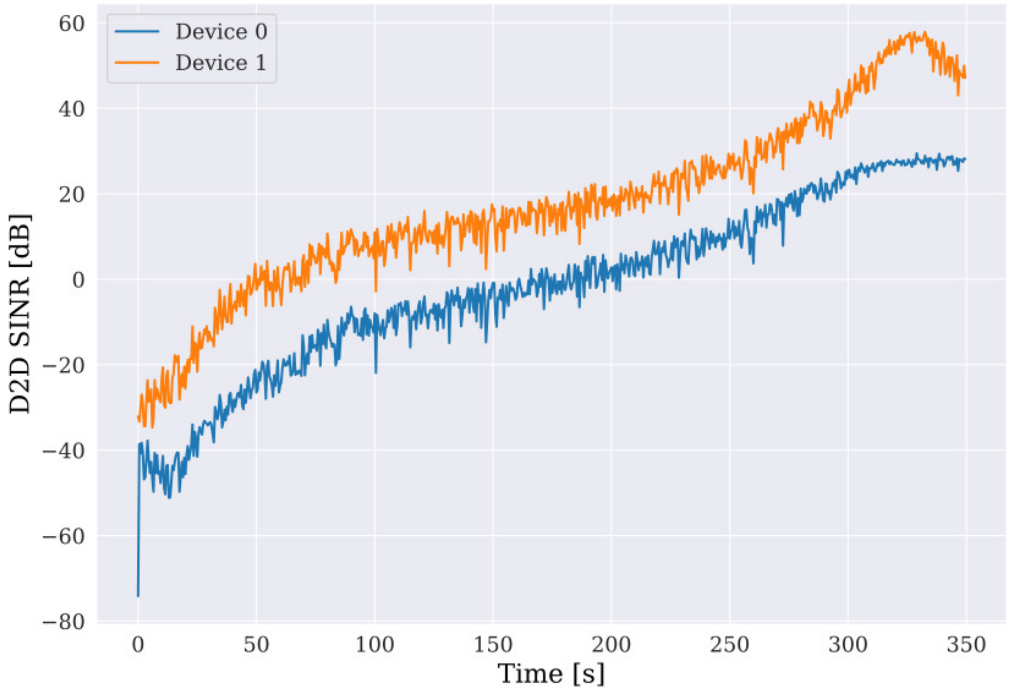

Figure 6.24 – D2D SINR for DDPG in experiment 1.

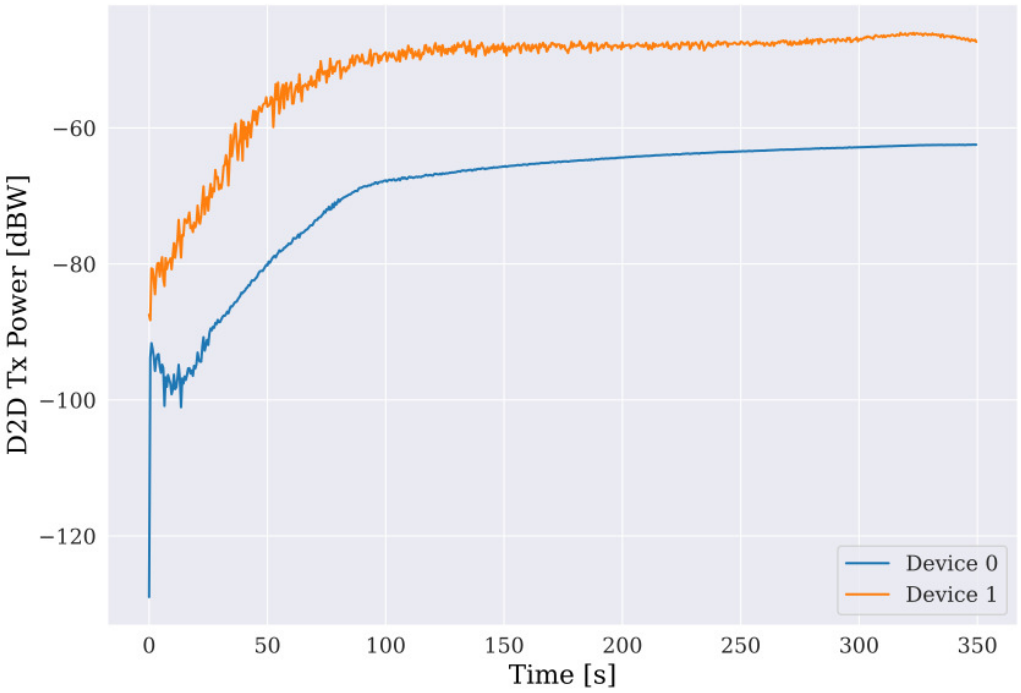

Figure 6.25 – D2D transmission powers for DDPG in experiment 1.

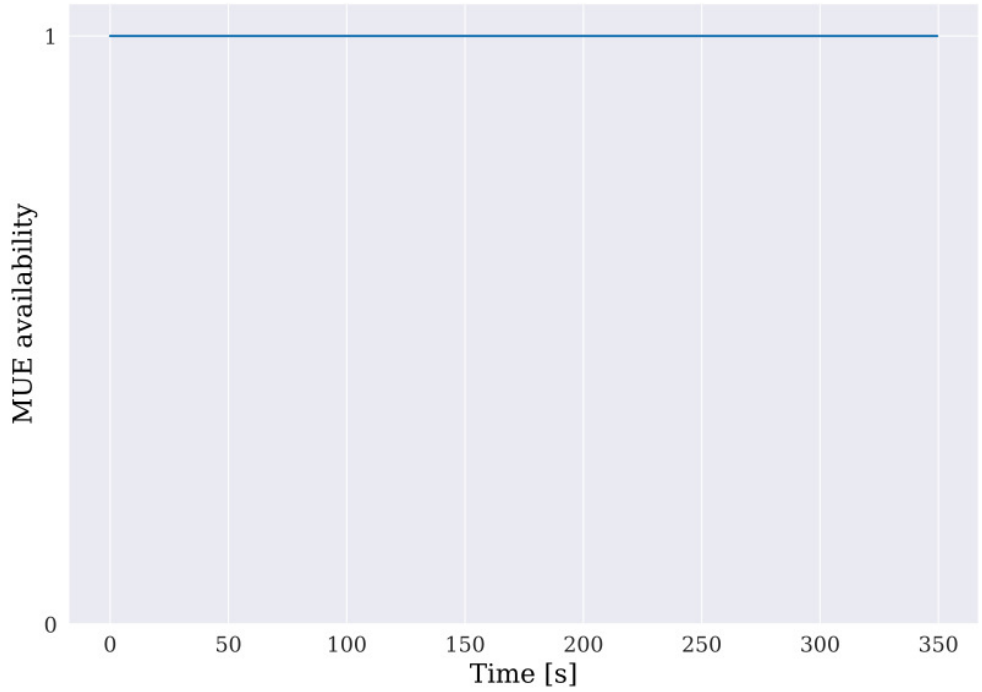

Figure 6.26 – MUE availability for DDPG in experiment 1. Average availability of 1.0.

6.5.2 Experiment 2

In this experiment, the devices begin at the same start positions as in Experiment 1, which is depicted by Figure 6.12. However, the trajectories are now slightly different. The MUE and the D2D pair 1 have the same motion as in Experiment 1, while D2D pair 0 stands still the whole time. The idea is to check if the agents are able to acknowledge their distinguished cases and act accordingly. Figure 6.27 presents the new devices trajectories.

The expected outcome for this experiment is for D2D transmitter 0 to apply low transmit power levels, or even to remain completely shut down for the whole experiment, while D2D transmitter 1 increases its transmitter power as it distances itself from the BS and the MUE approaches the BS. The average availability, across the experiments, is expected to be close to 100%.

6.5.2.1 DQL

In Experiment 2, the DQL solution had a bad performance. Throughout the whole experiment, the MUE SINR remained below the minimum threshold, as depicted by Figures 6.28 and 6.31.

Both D2D pairs had high SINRs, at the expense of the MUE QoS. Similar to what happened on Experiment 1, both devices begun transmitting with -40 dBW and, towards the end of the experiment, device 0 increased its transmit power to -20 dBW, around 300 s. This effect is felt on all devices' SINRs, as depicted by Figures 6.28 and 6.29.

The solution did not behave as expected. Both D2D transmitters should have chosen the -90 dBW power level from the start. As time passes, device 0 should have increased its transmit power, as it moved further away from the BS. Device 1 should have kept at -90 dBW power level, as it remained close to the BS.

6.5.2.2 A2C

Despite acting differently from the DQL solution, A2C also had a poor performance in Experiment 2. By looking at Figures 6.32 and 6.35, it is possible to see the that the solution fails to provide the minimum MUE SINR requirement.

The agents keep interchanging between the -40 dBW and -30 dBW power levels, without a clear pattern. Therefore, the solution did not act as expected.

6.5.2.3 DDPG

The DDPG algorithm was able to satisfy the MUE QoS requirement throughout the whole experiment, as Figures 6.36 and 6.39 show.

The power allocation strategy is depicted by Figure 6.38. In the experiment's beginning, both D2D agents transmit at -90 dBW, which is a very low power level. As Device 1 gets further away from the BS, it increases its transmit power to -40 dBW, a much higher power level, while Device 0 keeps its transmit power at really low levels. This is reflected by Figure 6.37, which shows the D2D SINRs.

In the end, the DDPG algorithm was able to achieve a high average MUE availability of 99.857%, as presented by Figure 6.39, while providing a 60 dB SINR to Device 1.

The DDPG solution performed according to the expectations. Device 1 raised its power level as its distance to the BS increased, while Device 0 kept its transmit power at really low values, since it stood next to the BS during the whole experiment. Therefore, it was expected for Device 1 to have high SINR levels and for Device 0 to have a low SINR. The MUE achieved a good QoS for almost the whole experiment.

6.5.3 Conclusions on Experiments 1 and 2

Experiments 1 and 2 provided interesting results in order to evaluate the algorithms' performances in controlled scenarios. It was possible to confirm the CLDE solutions performed poorly in these scenarios, despite achieving interesting *macro* results, as presented in Section 6.4, such as high D2D SINR and MUE availability, up to two D2D pairs.

On the other hand, DDPG presented low D2D SINR in Section 6.4, but proved to be much more adaptable and coherent in Experiments 1 and 2, providing very high MUE availability and good D2D SINR levels. Therefore, it is decided the DDPG solution, among the presented algorithms, had the best performance in the experiments.

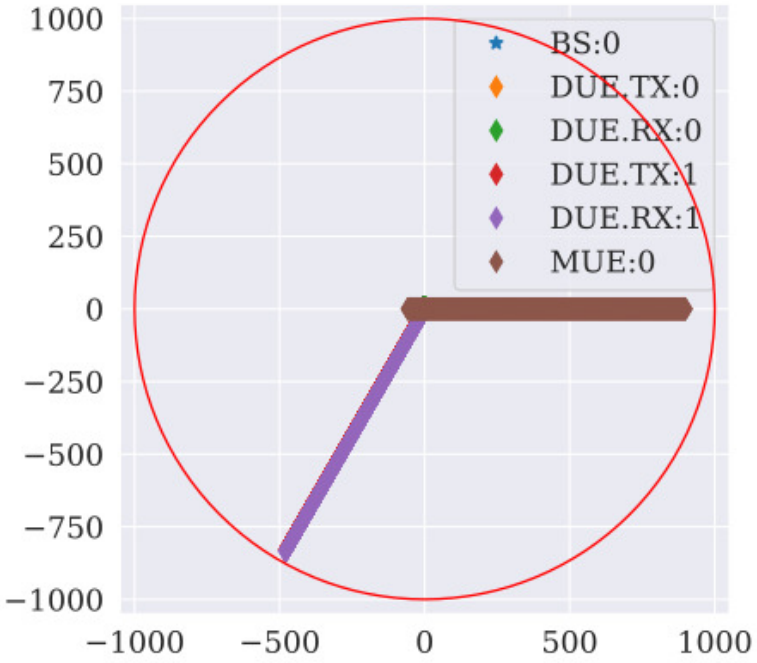

Figure 6.27 – Devices trajectories in Experiment 2.

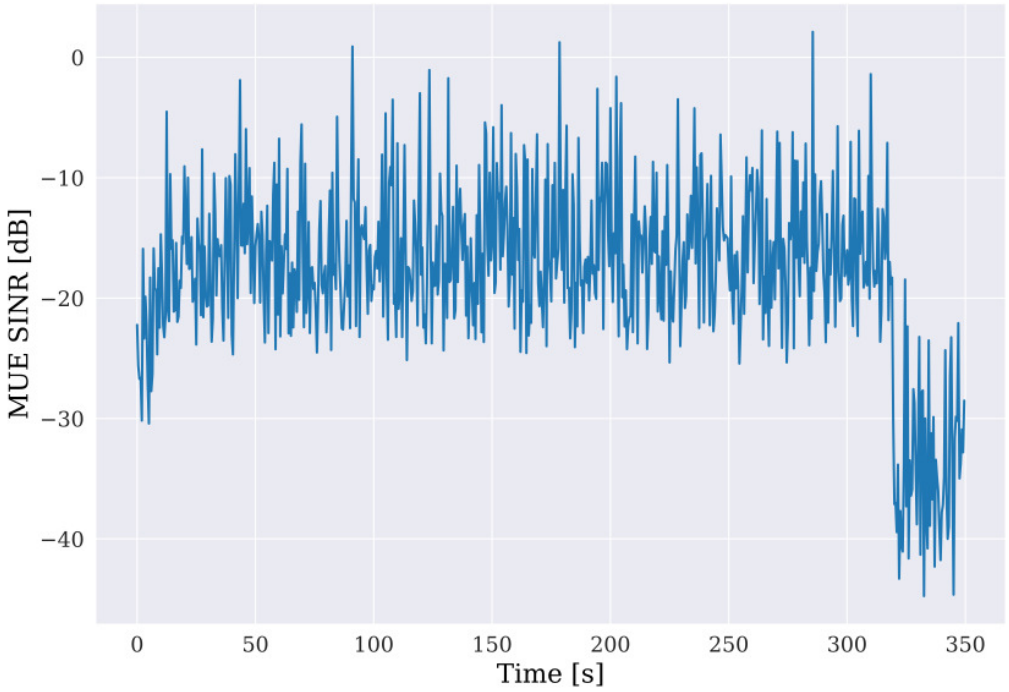

Figure 6.28 – MUE SINR for DQL in Experiment 2.

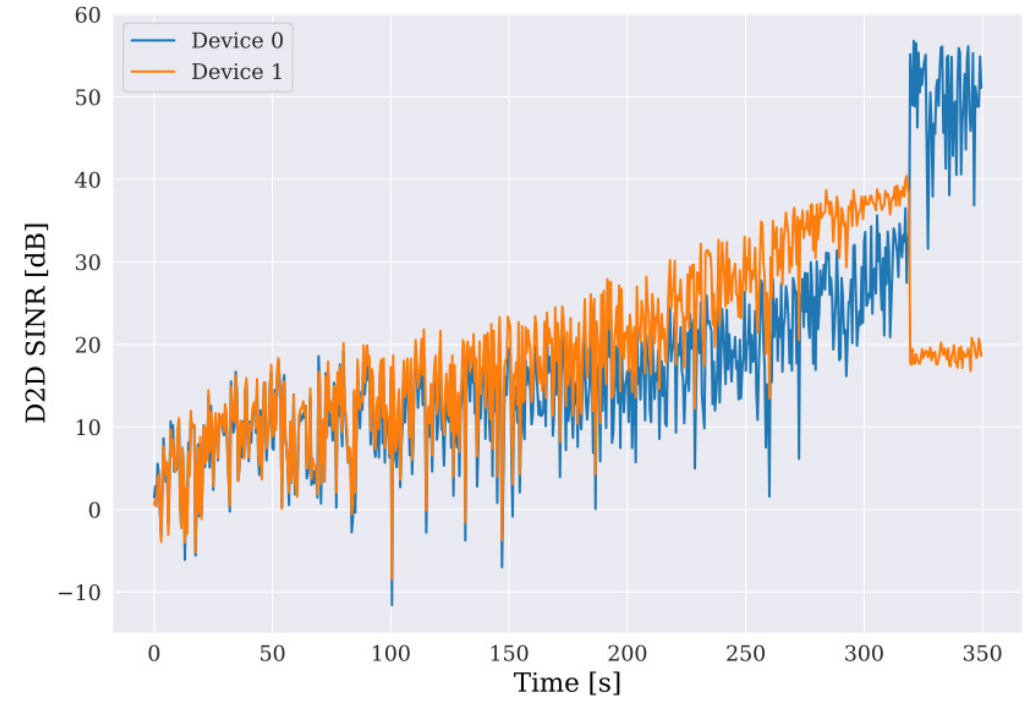

Figure 6.29 – D2D SINR for DQL in Experiment 2.

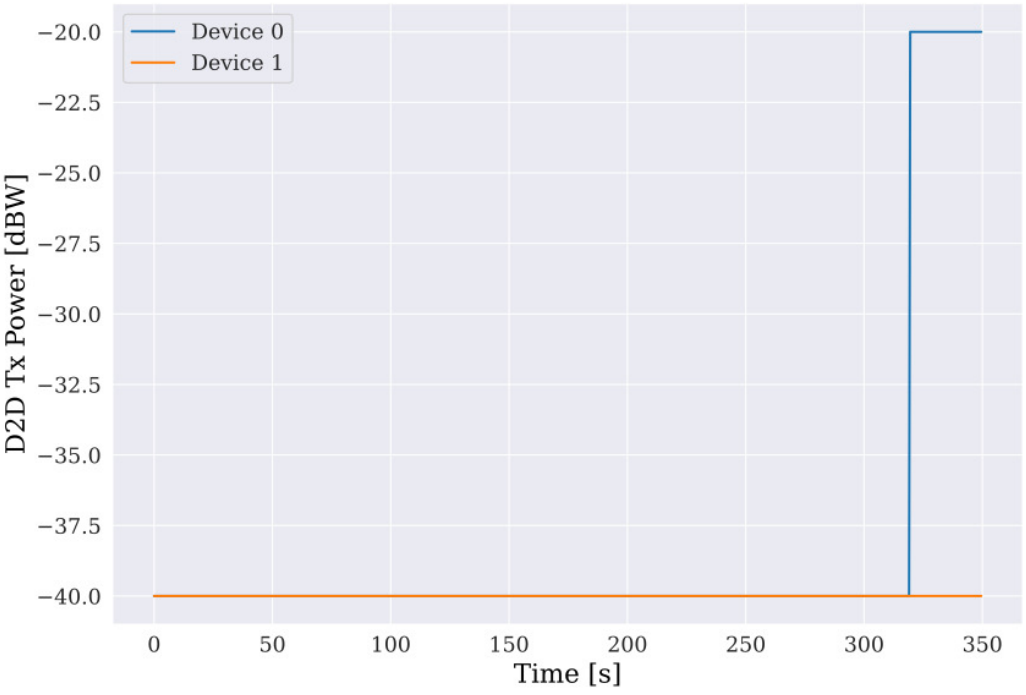

Figure 6.30 – D2D transmission powers for DQL in Experiment 2.

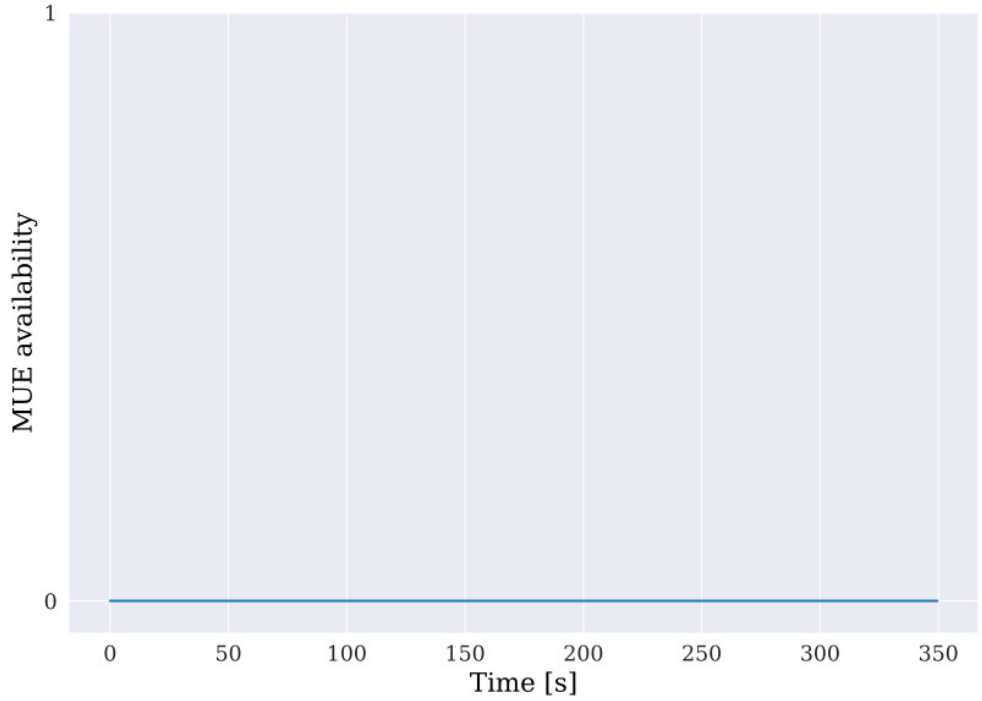

Figure 6.31 – MUE availability for DQL in Experiment 2. Average availability of 0%.

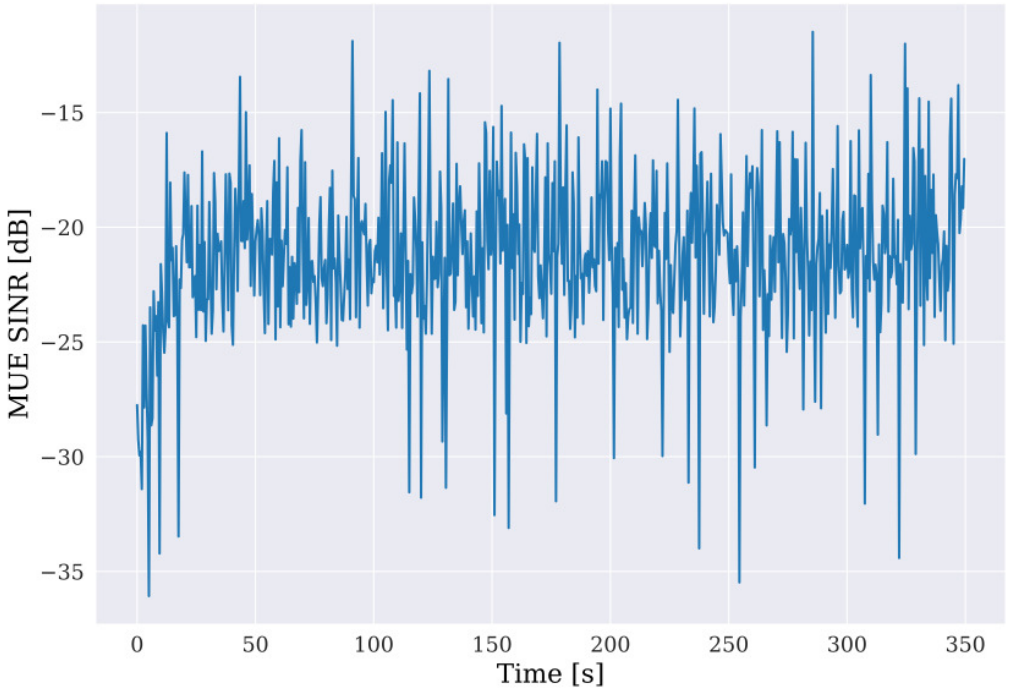

Figure 6.32 – MUE SINR for A2C in Experiment 2.

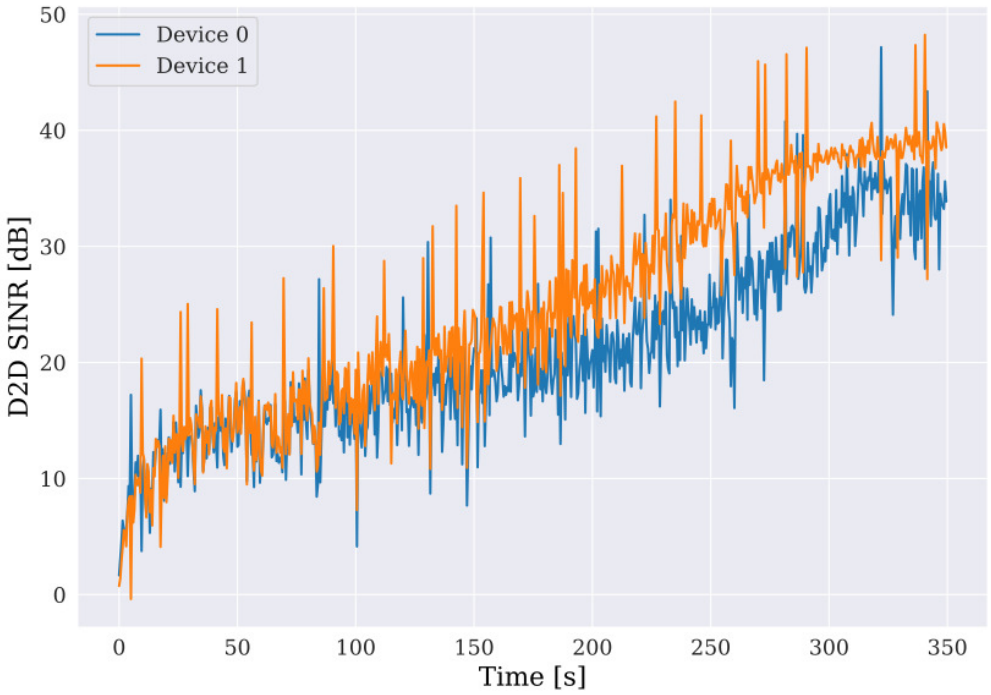

Figure 6.33 – D2D SINR for A2C in Experiment 2.

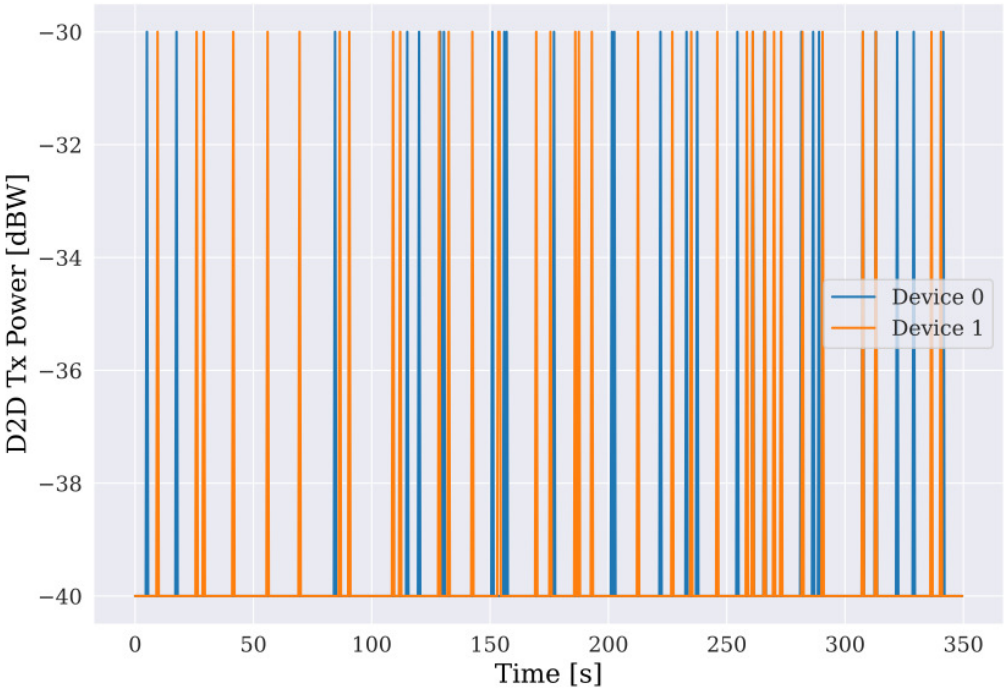

Figure 6.34 – D2D transmission powers for A2C in Experiment 2.

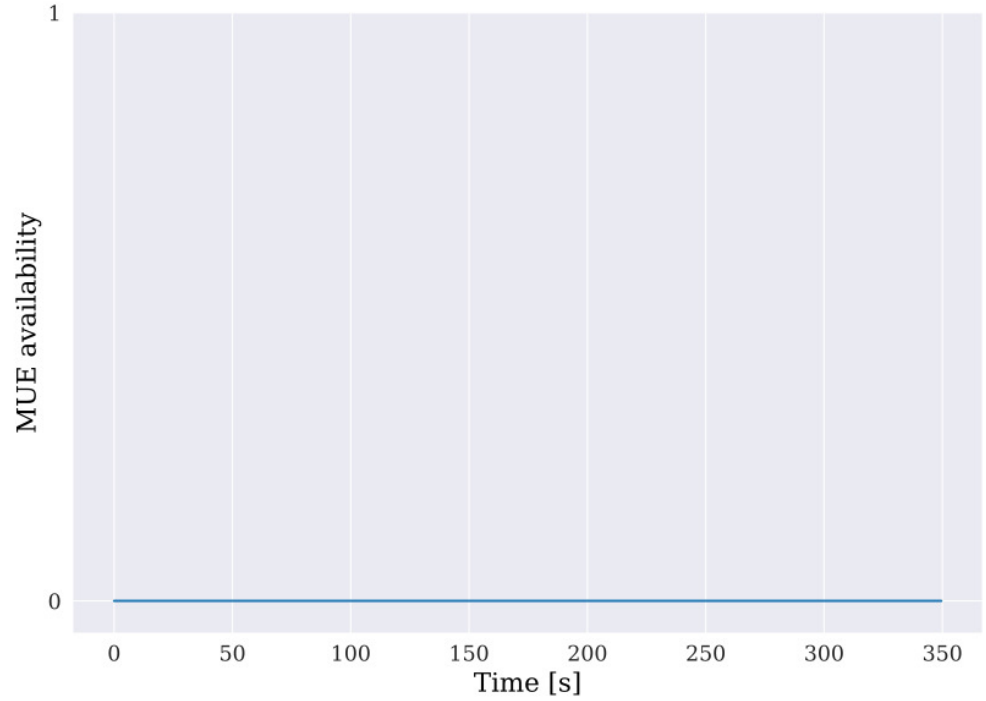

Figure 6.35 – MUE availability for A2C in Experiment 2. Average availability of 0%.

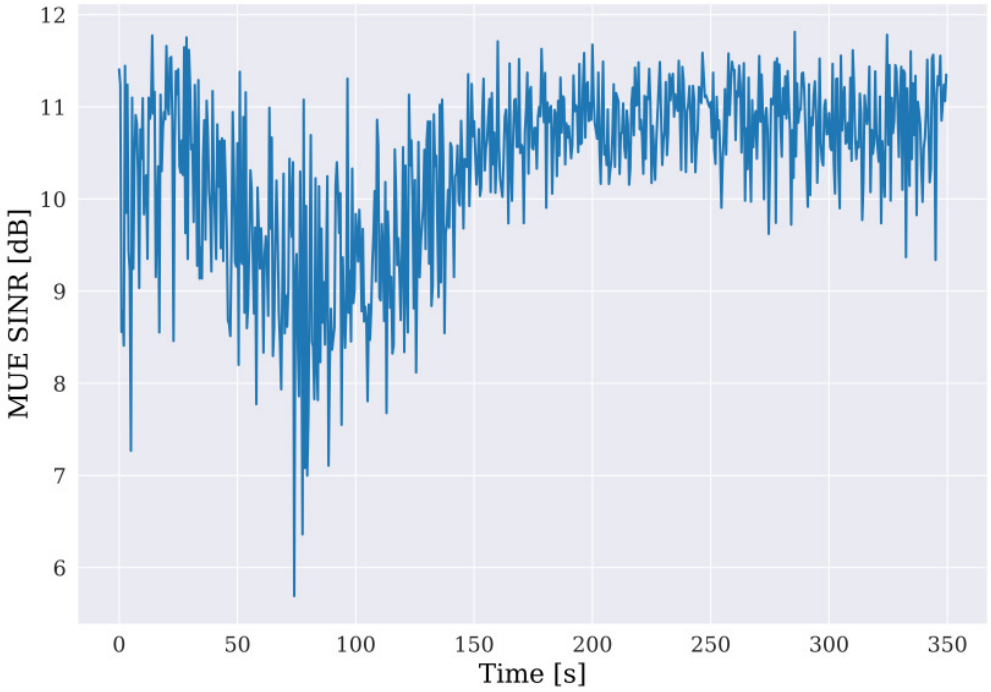

Figure 6.36 – MUE SINR for DDPG in Experiment 2.

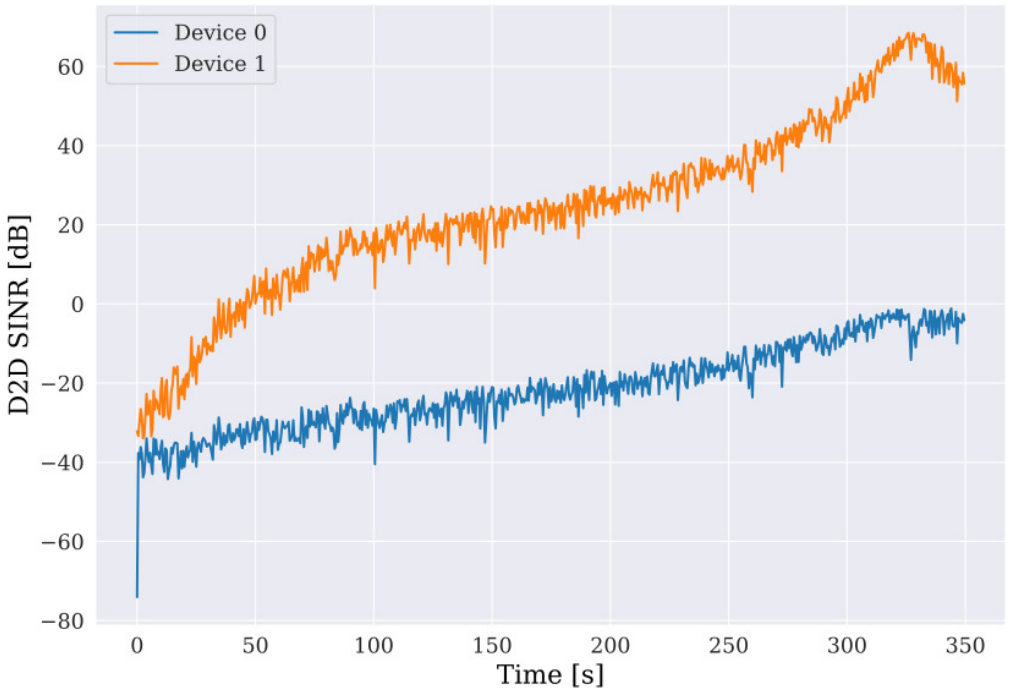

Figure 6.37 – D2D SINR for DDPG in Experiment 2.

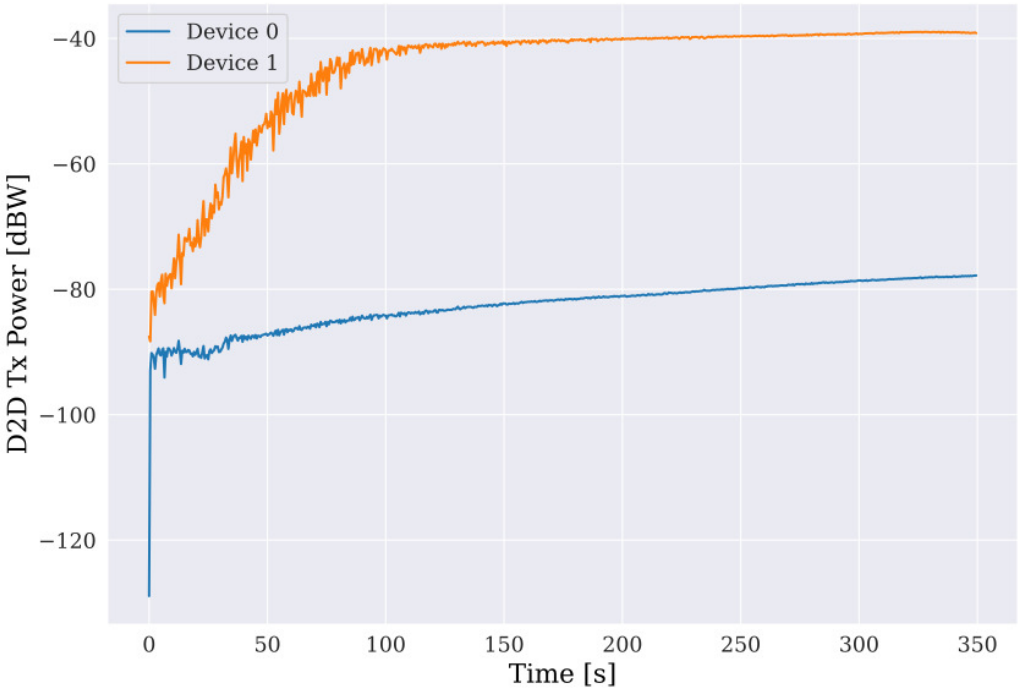

Figure 6.38 – D2D transmission powers for DDPG in Experiment 2.

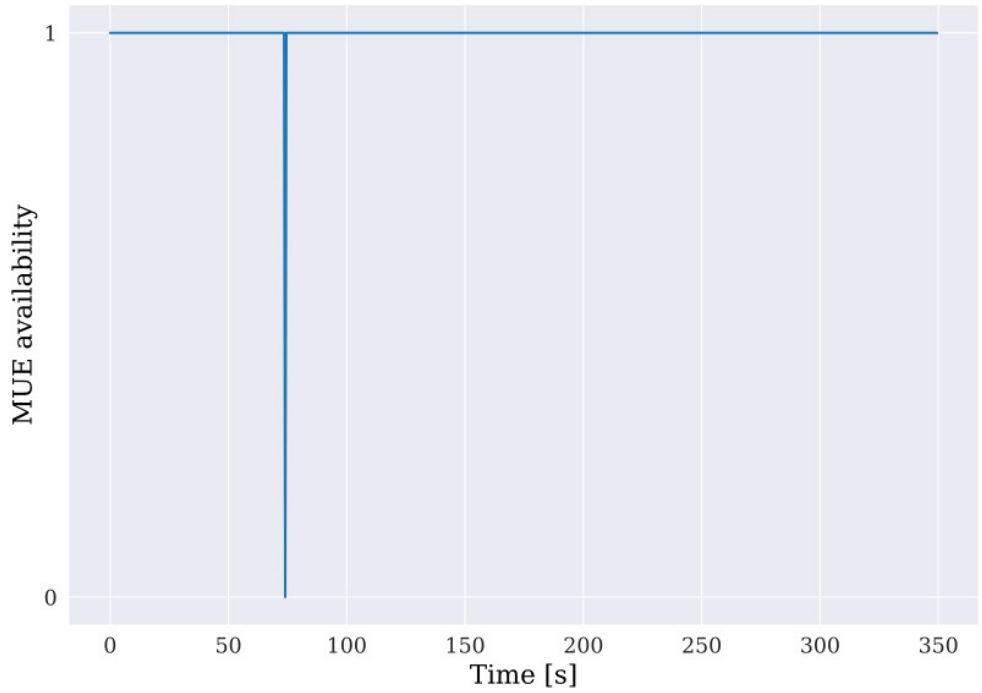

Figure 6.39 – MUE availability for DDPG in Experiment 2. Average availability of 0.99857.

6.6 DDPG ON DETERMINISTIC CHANNELS

In this experiment, DDPG was tested on similar conditions to Experiment 1, but in a environment where the channels' gain are composed only by the deterministic pathlosses. Figure 6.40 depicts the channel losses from both devices to the BS. It is possible to see that both devices experience similar channel conditions.

The fadings were removed in order to observe if the algorithm would allocate the same transmission power levels to both transmitters. Figure 6.41 shows the allocated transmission powers. The results show the devices transmission power levels differ in about 20 dBW. This goes to show that the DDPG algorithm does not provide fairness.

The lack of fairness may be explained by the reward function. The proposed reward function for DDPG, which is defined in (5.7), focuses only on maximizing the sum of the devices SINR, and it is not concerned about fairness.

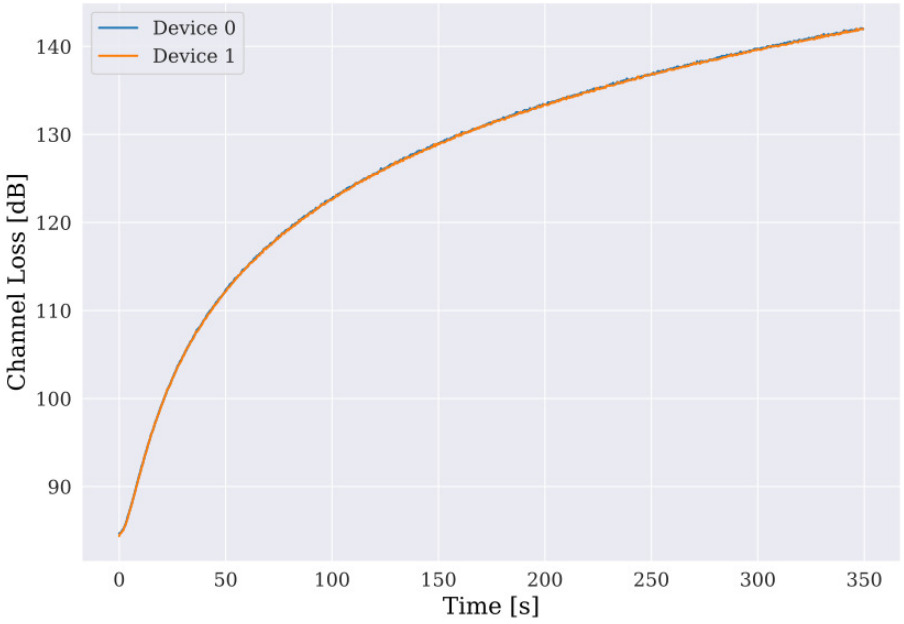

Figure 6.40 – Channel to BS.

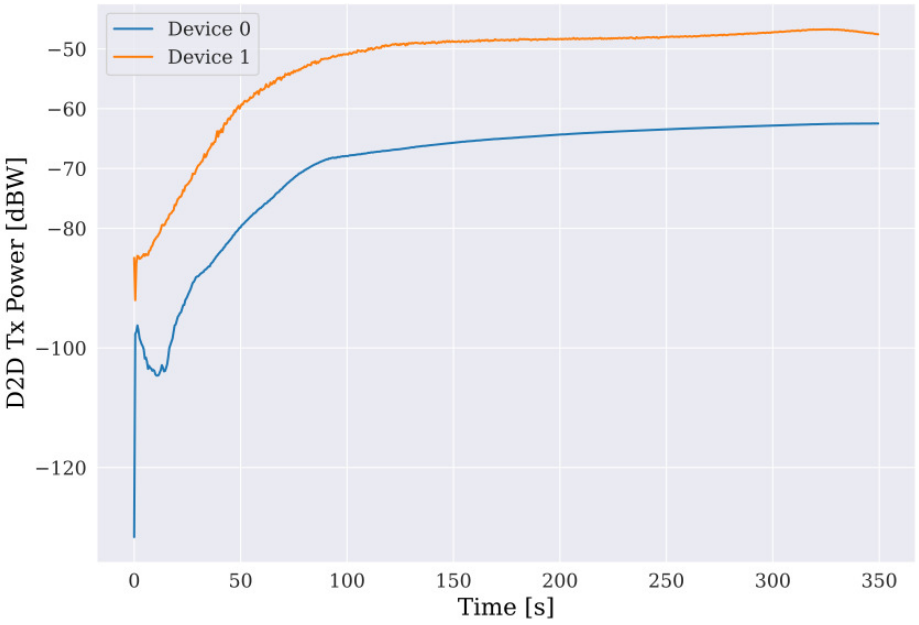

Figure 6.41 – Devices transmission powers.

6.7 SPECTRAL EFFICIENCY

The final results show the spectral efficiency provided by the algorithms. The main objective of such algorithms is, indeed, to improve this metric and help the mobile communication systems achieve the expectations that were drawn for 5G, while guaranteeing acceptable levels of QoS to the primary user.

Figure 6.42 shows the average system spectral efficiencies, provided by the algorithms, according to the number of D2D pairs. The system spectral efficiency is the sum of all devices spectral efficiencies.

Despite the high spectral efficiencies obtained with DQL, A2C and the random algorithms, these solutions provide poor MUE QoS, as already presented by Figure 6.8. Therefore, these algorithms are not suited for the task at hand.

On the other hand, DDPG provides significant gains on spectral efficiency, while guaranteeing high SINR levels to the MUE. For only one D2D pair, the algorithm is able to increase the average system spectral efficiency from 4.07 bps/Hz to 7.01 bps/Hz, which is an improvement of 72%. As the number of D2D pairs increase, the improvement on the spectral efficiency decreases, since the algorithm lowers the D2D transmit powers accordingly, in order to keep satisfying the MUE QoS requirement. This is verified as the DDPG curve approaches the "No D2D communication" curve.

Figure 6.43 portrays the spectral efficiency obtained in Experiment 1. This is a very favorable scenario for D2D communication and it goes to highlight the possible spectral efficiency gains this type of communication may provide, along with the proposed DDPG-based power control algorithm.

During the system spectral efficiency peak, the proposed algorithm was able to improve this metric from 7.01 bps/Hz to 32.90 bps/Hz, which is a 369% increase. This improvement really shows the potential of combining D2D communications with DRL-based techniques.

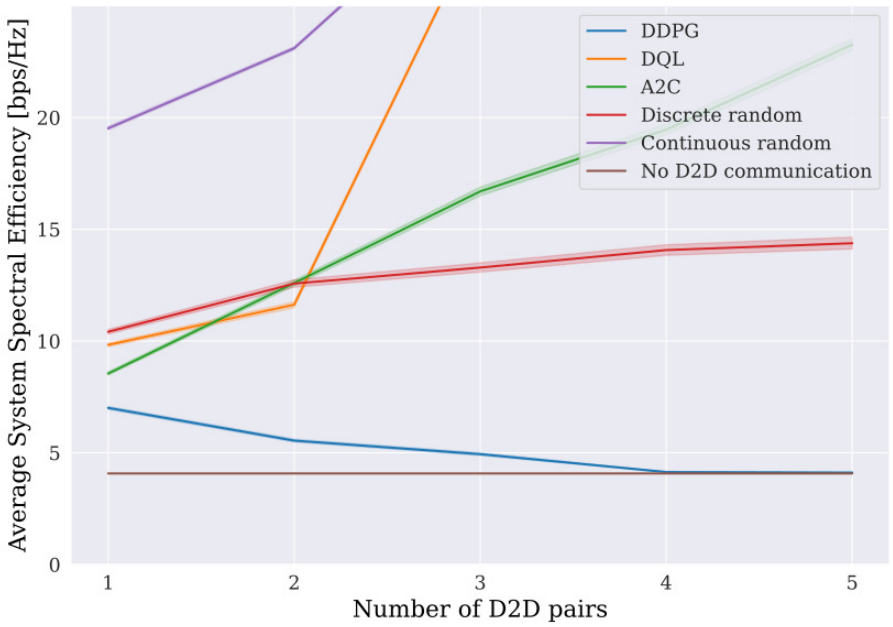

Figure 6.42 – Average System Spectral efficiencies.

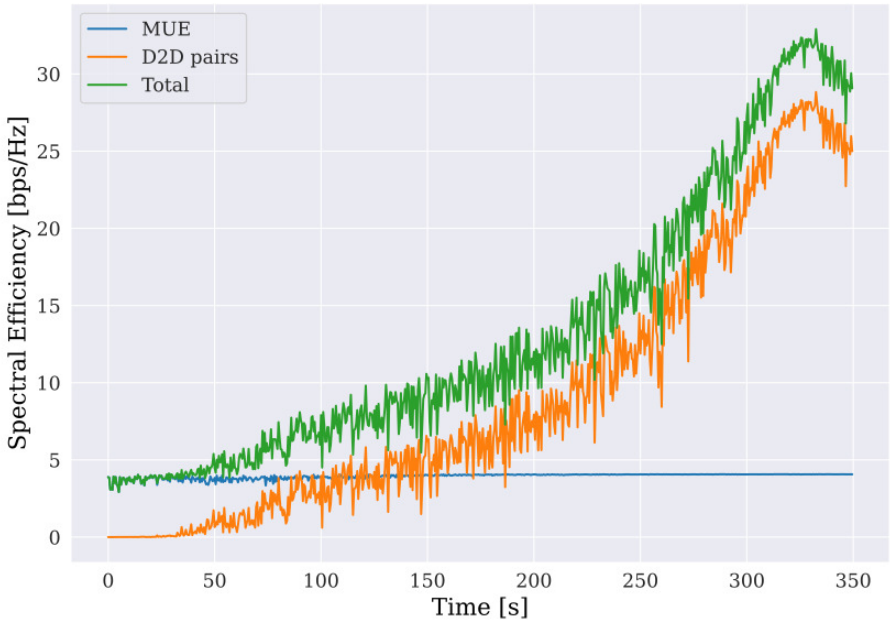

Figure 6.43 – Spectral efficiencies for DDPG in Experiment 1.

6.8 CONCLUSIONS

This chapter presented the simulations methodology along with the obtained results. The simulations' goal was to evaluate the proposed DRL-based algorithms and obtain not only *macro*-results, but also in-depth knowledge on the solutions' behaviours.

The proposed DRL-based solutions had better performances than random allocation algorithms, which goes to show all algorithms were able to achieve degrees of knowledge on the task at hand. The CLDE solutions provided aggressive solutions, that provided high D2D SINR and poor MUE availability. The CLCE solution went on the opposite way, being conservative with the power allocation and providing high QoS to the primary user.

Experiments 1 and 2 evaluated how the algorithms would perform in controlled scenarios. Despite showing ability to adapt to the experiments, the CLDE solutions were note able to able to achieve good performances and failed to deliver acceptable levels of MUE availability. On the other hand, the CLCE algorithm provided high MUE and D2D SINR levels, acting in adaptable and coherent ways, achieving the best performance among the proposed solutions.

Although DDPG had satisfying performances, it is still very conservative and allocates very low power levels to D2D transmitters. There is still room for improvement in this area, in order to have the algorithm make more optimal decisions.

By evaluating all the obtained results, the DDPG-based algorithm is considered to be the best among the proposed solutions. It is able to provide high MUE availability, along with high system spectral efficiency gains.

7 CONCLUSIONS

This work investigated the application of DRL-based techniques to the power allocation problem in inband underlay D2D communication.

At first, fundamental concepts were introduced, in order to ease the understanding on the presented topics. The base theory concerning reinforcement learning was presented. The problem formalization as a markovian decision process, diverse RL algorithms and paradigms were elucidated.

Another important discussed topic was neural networks. This work presented the base theory, as well as more advanced architectures, along with optimization methods for training. With these topics clarified, it was possible to explore the field of this work's main tool, deep reinforcement learning.

D2D communication, in inband underlay mode, was studied in a scenario envisioned to approximate reality, by making use of the WINNER II and ITU-R BAN channel models, simulating IoT devices, performing D2D communication, in an urban environment. During transmission, the devices could move in pedestrian speeds, or stay still. The transmission happened at uplink times.

The three proposed power control solutions were based on: DQL, which is an off-policy value-optimization algorithm, A2C, an on-policy actor-critic-optimization algorithm, and DDPG, an off-policy actor-critic-algorithm. The first two solutions, which make use of DQL and A2C, respectively, output discrete-value actions, and were employed under a multi-agent scheme, in order to avoid the curse of dimensionality. DDPG could be applied in a single-agent centralized scheme, since its continuous-value actions nature enables this approach. All these techniques were evaluated in the proposed scenario, and compared among themselves.

The obtained results show that the multi-agent solutions tend to behave more aggressively, easily violating the MUE QoS. These solutions also tend to choose actions that have good average returns. This behaviour has a negative impact on the solutions' adaptability, making them act poorly on some of the experiments.

On the other hand, the single-agent solution, implemented using DDPG, was inclined to satisfy the MUE SINR requirements, at the expense of the quality of D2D communication, which is a desired outcome. Therefore, on average, this solution presents smaller SINR values for the D2D pairs. Additionally, the central agent solution was able to react accordingly in the majority of the experiments, which goes to show the adaptability of this solution.

It is important to highlight that discrete-value actions algorithms have access to a much

smaller array of action options, when in comparison to continuous-value actions algorithms. The choice of the action options set has a great influence in the algorithms performance, and the process of defining such set is a complex problem by itself. It is also important to notice that, although continuous-value algorithms are better suited to this works' subject, there are problems where discrete-value algorithms are better equipped to deal with, such as situations where the actions are inherently discrete, e.g., most of video-games [47].

Based on this work's results, it has been concluded that the DDPG solution has proven to be the most reliable, consistent and adaptable, among the tested algorithms. It is able to enhance the system's spectral efficiency, while satisfying the primary user's QoS requirement. The solution displays promising possibilities for improvements.

FUTURE WORKS

According to this work's results, continuous-value DRL-based techniques are a promising field for the power control problem in D2D underlay communication. Therefore, opportunities for improvement, along with new experiments, were identified. It is suggested:

- experimenting with other states and rewards functions, different from what is done in this work. These design choices have a great impact on the algorithms' behaviour and may be the easiest feature to improve upon;
- testing other acclaimed continuous-value actions methods, such as the ones proposed in [49, 86, 87]. Different algorithms may perform better, or worse, than the solutions presented in this work;
- experimenting with other RL paradigms, such as model-based algorithms. DeepMind has achieved great success with this approach [30, 88, 89].
- testing for multiple RBs. In this work, the simulations were performed considering only one RB. When multiple resource blocks are concerned, the transmission power constraints become more complex, since the agents must decide upon how much power to allocate on each RB, without violating the maximum transmit power restriction. This is an interesting problem and may be implemented by expanding upon this work's contributions.
- trying out these solution in more complex simulators, such as full fledged system simulators or network simulators.
- experimenting with multiple applications requirements. In this work, the MUE QoS requirement remained the same across all experiments. It would be interesting to test

with dynamic QoS requirements, in order to test the solutions adaptability. This variable could be incorporated to the design by altering the rewards and the states definitions.

BIBLIOGRAPHY

1 HU, B. et al. Characteristics of SARS-CoV-2 and COVID-19. *Nature Reviews Microbiology*, Nature Publishing Group, p. 1–14, 2020.

2 ORGANIZATION, W. H. et al. *WHO COVID-19 Preparedness and Response Progress Report-1 February to 30 June 2020*. [S.l.]: World Health Organization, 2020.

3 CERWALL, P. et al. Ericsson mobility report. *Hg. v. Ericsson*, 2020.

4 CHETTRI, L.; BERA, R. A comprehensive survey on internet of things (IoT) towards 5G wireless systems. *IEEE Internet of Things Journal*, IEEE, 2019.

5 ITU-R. *IMT Vision–Framework and overall objectives of the future development of IMT for 2020 and beyond*. [S.l.]: International Telecommunication Union - Radio Sector, 2015.

6 3GPP. *TR 45.820 V13.1.0; Technical Specification Group GSM/EDGE Radio Access Network; Cellular system support for ultra-low complexity and low throughput Internet of Things (CIoT)(Release 13)*. [S.l.]: 3rd Generation Partnership Project, 2015.

7 AL., C. et. *Cisco Annual Internet Report (2018–2023)*. [S.l.]: Cisco, 2020.

8 JAMEEL, F. et al. A survey of device-to-device communications: Research issues and challenges. *IEEE Communications Surveys & Tutorials*, IEEE, v. 20, n. 3, p. 2133–2168, 2018.

9 KAI, Y. et al. Resource allocation and performance analysis of cellular-assisted ofdma device-to-device communications. *IEEE Transactions on Wireless Communications*, IEEE, v. 18, n. 1, p. 416–431, 2018.

10 ANSARI, R. I. et al. 5G D2D networks: Techniques, challenges, and future prospects. *IEEE Systems Journal*, IEEE, v. 12, n. 4, p. 3970–3984, 2017.

11 SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

12 RADFORD, A. et al. Language models are unsupervised multitask learners. *OpenAI blog*, v. 1, n. 8, p. 9, 2019.

13 HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

14 INC., T. *Autopilot AI*. Tesla Inc., 2020. Disponível em: <https://www.tesla.com/autopilotAI>.

15 ESTEVA, A. et al. A guide to deep learning in healthcare. *Nature medicine*, Nature Publishing Group, v. 25, n. 1, p. 24–29, 2019.

16 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.

17 MNIH, V. et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

18 MNIH, V. et al. Asynchronous methods for deep reinforcement learning. In: *International conference on machine learning*. [S.l.: s.n.], 2016. p. 1928–1937.

19 NIE, S. et al. Q-learning based power control algorithm for D2D communication. In: IEEE. *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. [S.l.], 2016. p. 1–6.

20 TOUMI, S.; HAMDI, M.; ZAIED, M. An adaptive Q-learning approach to power control for D2D communications. In: IEEE. *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. [S.l.], 2018. p. 206–209.

21 YU, S.; JEONG, Y. J.; LEE, J. W. Resource allocation scheme based on deep reinforcement learning for device-to-device communications. In: IEEE. *2021 International Conference on Information Networking (ICOIN)*. [S.l.], 2021. p. 712–714.

22 ZHANG, X. et al. Deep multi-agent reinforcement learning for resource allocation in D2D communication underlaying cellular networks. In: IEEE. *2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*. [S.l.], 2020. p. 55–60.

23 SUN, M.; DING, Y.; GOUSSETIS, G. Adaptive mode selection and power allocation for D2D underlay cellular networks with dynamic fading channel. In: IEEE. *2020 International Conference on UK-China Emerging Technologies (UCET)*. [S.l.], 2020. p. 1–4.

24 CHEN, R.; XU, J. Particle swarm optimization based power allocation for D2D underlaying cellular networks. In: IEEE. *2017 IEEE 17th International Conference on Communication Technology (ICCT)*. [S.l.], 2017. p. 503–507.

25 NGUYEN, K. K. et al. Non-cooperative energy efficient power allocation game in D2D communication: A multi-agent deep reinforcement learning approach. *IEEE Access*, IEEE, v. 7, p. 100480–100490, 2019.

26 XU, Y.-H. et al. Deep deterministic policy gradient (DDPG)-based resource allocation scheme for noma vehicular communications. *IEEE Access*, IEEE, v. 8, p. 18797–18807, 2020.

27 NGUYEN, K. K. et al. Distributed deep deterministic policy gradient for power allocation control in D2D-based v2v communications. *IEEE Access*, IEEE, v. 7, p. 164533–164543, 2019.

28 ZHANG, T.; ZHU, K.; WANG, J. Energy-efficient mode selection and resource allocation for D2D-enabled heterogeneous networks: A deep reinforcement learning approach. *IEEE Transactions on Wireless Communications*, IEEE, 2020.

29 ACHIAM, J.; ABBEEL, P. *Spinning Up in Deep RL*. OpenAI, 2018. Disponível em: <https://spinningup.openai.com/en/latest>.

30 SILVER, D. et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

31 LEITE, J. P. *Aplicação de Técnicas de Aprendizado por Reforço à Alocação de Recursos e ao Escalonamento de Usuários em Sistemas de Telecomunicações*. [S.l.]: University of Brasilia, Department of Electrical Engineering, 2014.

32 BADIA, A. P. et al. *Agent57: Outperforming the human Atari benchmark*. DeepMind, 2020. Disponível em: <https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark>.

33 BAKER, B. et al. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

34 BERNER, C. et al. DOTA 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

35 HAYKIN, S. S. et al. *Neural networks and learning machines/Simon Haykin.* [S.l.]: New York: Prentice Hall,, 2009.

36 LEARNING, D. et al. *Adaptive Computation and Machine Learning Series*. [S.l.]: MIT Press, 2016.

37 KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, ACM New York, NY, USA, v. 60, n. 6, p. 84–90, 2017.

38 KARIM, F. et al. LSTM fully convolutional networks for time series classification. *IEEE access*, IEEE, v. 6, p. 1662–1669, 2017.

39 WANG, Y. et al. Attention-based LSTM for aspect-level sentiment classification. In: *Proceedings of the 2016 conference on empirical methods in natural language processing*. [S.l.: s.n.], 2016. p. 606–615.

40 DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

41 KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

42 LOSHCHILOV, I.; HUTTER, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

43 HINTON, G.; SRIVASTAVA, N.; SWERSKY, K. Overview of mini-batch gradient descent. *Neural Networks for Machine Learning*, v. 575, 2012.

44 GRAVES, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

45 AMODEI, D. et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In: *International conference on machine learning*. [S.l.: s.n.], 2016. p. 173–182.

46 DENG, L.; HINTON, G.; KINGSBURY, B. New types of deep neural network learning for speech recognition and related applications: An overview. In: IEEE. *2013 IEEE international conference on acoustics, speech and signal processing*. [S.l.], 2013. p. 8599–8603.

47 MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

48 BADIA, A. P. et al. Agent57: Outperforming the atari human benchmark. *arXiv preprint arXiv:2003.13350*, 2020.

49 SCHULMAN, J. et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

50 SUTTON, R. S. et al. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, v. 12, p. 1057–1063, 1999.

51 WILLIAMS, R. J.; PENG, J. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, Taylor & Francis, v. 3, n. 3, p. 241–268, 1991.

52 WU, Y. et al. *OpenAI Baselines: ACKTR & A2C*. OpenAI, 2017. Disponível em: <https://openai.com/blog/baselines-acktr-a2c>.

53 SCHULMAN, J. et al. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

54 LILLICRAP, T. P. et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

55 SILVER, D. et al. Deterministic policy gradient algorithms. In: PMLR. *International conference on machine learning*. [S.l.], 2014. p. 387–395.

56 PLAPPERT, M. et al. Parameter space noise for exploration in deep reinforcement learning. *arXiv preprint arXiv:1706.01905*, 2017.

57 PLAPPERT, M. *Parameter space noise for exploration in deep reinforcement learning*. [S.l.]: Karlsruhe Institue of Technology, 2017.

58 3GPP. *Overview of 3GPP Release 12 V0.2.0 (2015-09)*. [S.l.]: 3rd Generation Partnership Project, 2015.

59 XENAKIS, D. et al. Performance analysis of network-assisted D2D discovery in random spatial networks. *IEEE Transactions on Wireless Communications*, IEEE, v. 15, n. 8, p. 5695–5707, 2016.

60 HU, L. Resource allocation for network-assisted device-to-device discovery. In: IEEE. *2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE)*. [S.l.], 2014. p. 1–5.

61 AHISHAKIYE, F.; LI, F. Y. Service discovery protocols in D2D-enabled cellular networks: Reactive versus proactive. In: IEEE. *2014 IEEE Globecom Workshops (GC Wkshps)*. [S.l.], 2014. p. 833–838.

62 KAR, U. N.; SANYAL, D. K. An overview of device-to-device communication in cellular networks. *ICT express*, Elsevier, v. 4, n. 4, p. 203–208, 2018.

63 HAUS, M. et al. Security and privacy in device-to-device (D2D) communication: A review. *IEEE Communications Surveys & Tutorials*, IEEE, v. 19, n. 2, p. 1054–1079, 2017.

64 GANDOTRA, P.; JHA, R. K.; JAIN, S. A survey on device-to-device (D2D) communication: Architecture and security issues. *Journal of Network and Computer Applications*, Elsevier, v. 78, p. 9–29, 2017.

65 ALI, S.; AHMAD, A. Resource allocation, interference management, and mode selection in device-to-device communication: a survey. *Transactions on Emerging Telecommunications Technologies*, Wiley Online Library, v. 28, n. 7, p. e3148, 2017.

66 ORSINO, A. et al. Effects of heterogeneous mobility on D2D-and drone-assisted mission-critical MTC in 5G. *IEEE Communications Magazine*, IEEE, v. 55, n. 2, p. 79–87, 2017.

67 WANG, Z. et al. Propagation-and mobility-aware D2D social content replication. *IEEE Transactions on Mobile Computing*, IEEE, v. 16, n. 4, p. 1107–1120, 2016.

68 CHEN, H.-Y.; SHIH, M.-J.; WEI, H.-Y. Handover mechanism for device-to-device communication. In: IEEE. *2015 IEEE conference on standards for communications and networking (CSCN)*. [S.l.], 2015. p. 72–77.

69 LIU, Z. et al. Mode selection for device-to-device (D2D) communication under lte-advanced networks. In: IEEE. *2012 IEEE International Conference on Communications (ICC)*. [S.l.], 2012. p. 5563–5567.

70 YU, G. et al. Joint mode selection and resource allocation for device-to-device communications. *IEEE transactions on communications*, IEEE, v. 62, n. 11, p. 3814–3824, 2014.

71 LUO, Z.-Q.; ZHANG, S. Dynamic spectrum management: Complexity and duality. *IEEE journal of selected topics in signal processing*, IEEE, v. 2, n. 1, p. 57–73, 2008.

72 NASIR, Y. S.; GUO, D. Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 37, n. 10, p. 2239–2250, 2019.

73 PIRES, L. B.; CARVALHO, P. H. P. de. Deep q-learning framework for improving spectral efficiency in D2D communication. *XXXVIII SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES E PROCESSAMENTO DE SINAIS*, SBrT, 2020.

74 LANCTOT, M. et al. A unified game-theoretic approach to multiagent reinforcement learning. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 4190–4203.

75 HERNANDEZ-LEAL, P. et al. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

76 PEROLAT, J.; PIOT, B.; PIETQUIN, O. Actor-critic fictitious play in simultaneous move multistage games. In: . [S.l.: s.n.], 2018.

77 ZHANG, K.; YANG, Z.; BAŞAR, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.

78 AL., H. et. *Huawei 5G Wireless Network Planning Solution White Paper*. [S.l.]: Huawei Technologies Co., Ltd., 2018.

79 FOUNDATION, P. S. *Welcome to Python.org*. Python Software Foundation, 2020. Disponível em: <https://www.python.org/>.

80 PYTORCH. *PyTorch*. Pytorch, 2020. Disponível em: <https://www.pytorch.org/>.

81 PIRES, L. B. *lbaiao/sys-simulator-2*. GitHub, 2021. Disponível em: <https://github.com/lbaiao/sys-simulator-2>.

82 SALOUS, S. et al. Iracon propagation measurements and channel models for 5G and beyond. *ITU Journal: ICT Discoveries*, v. 2, n. 1, 2019.

83 MEINILÄ, J. et al. WINNER II channel models. *Radio Technologies and Concepts for IMT-Advanced*, Wiley Online Library, p. 39–92, 2009.

84 SAUNDERS, S. R.; ARAGÓN-ZAVALA, A. *Antennas and propagation for wireless communication systems*. [S.l.]: John Wiley & Sons, 2007.

85 ETSI, T. 136 101 v10. 3.0 (2011-06) lte. *Evolved universal terresterial radio access (E-UTRA)*.

86 FUJIMOTO, S.; HOOF, H.; MEGER, D. Addressing function approximation error in actor-critic methods. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2018. p. 1587–1596.

87 HAARNOJA, T. et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2018. p. 1861–1870.

88 SILVER, D. et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, American Association for the Advancement of Science, v. 362, n. 6419, p. 1140–1144, 2018.

89 SCHRITTWIESER, J. et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, Nature Publishing Group, v. 588, n. 7839, p. 604–609, 2020.